My PhD Notes

Nicholas Rotella

August 12, 2012

Forward

Congratulations! You've (somehow) managed to stumble across my vast repository of knowledge on all sorts of fun and exciting topics somehow related to the work of this robotics PhD student. This document contains all of my notes since I started my PhD in 2012. Many portions have come from my study of various internet sources including others' personal notes; I found these to be extremely useful in my studies and so decided to put my notes online as well. Other portions have been transcribed by hand into LaTeX from texts and interspersed with notes and addendums which (I hope) provide additional insight or a different interpretation of the material (I'm working on adding references where applicable). In all cases, I do not profit off of the material found in this document; it is merely an informal place for me to collect interesting notes and thoughts for later use. Most of my notes are the product of years of taking courses, reading texts, scouring the web and spending time working out intuitive explanations for concepts which once stumped me and putting them into my own words. This document is constantly evolving and likely contains tons of errors at any given time; in any case, I hope it will be useful to you, dear reader, in some way.

Contents

Chapter 1

Calculus

1.1 Fundamental Theorem of Calculus

Part 1: The first part of the theorem (sometimes called the first fundamental theorem) deals with the derivative of a definite integral. Let f be a continuous real-valued function defined on the interval $[a, b]$. Let F be the function defined for all x in this interval

$$F(x) = \int_a^x f(t)dt$$

Then F is continuous on $[a, b]$ and differentiable on (a, b); for all x in (a, b) we have

$$\frac{dF}{dt} = f(x)$$

ie the derivative of the integral is equal to the integrand evaluated at the upper limit.

Part 2: The second part of the theorem (sometimes called the second fundamental theorem) deals with the evaluation of a definite integral. Let f and F be real-valued functions defined on $[a, b]$ such that the derivative of F is f. If f is continuous over the closed interval (or at least Riemann integrable) then

$$\int_a^b f(x)dx = F(b) - F(a)$$

1.2 Mean Value Theorem

If a function f is continuous on the (closed) interval $[a, b]$ where $a \leq b$ and differentiable on the (open) interval (a, b) then there exists a point c such that

$$f'(c) = \frac{f(b) - f(a)}{b - a}$$

In other words, there is a point c between a and b such that the tangent at c is equal to the secant through the endpoints a and b of the interval. [**?**]

1.3 Taylor Series Expansion

Consider the value of an infinitely differentiable function $f(x)$ in the vicinity of a point x_0 (usually taken to be an equilibrium point of the function, ie a point where the derivatives vanish). For a small deviation $(x - x_0)$ about this point, the value of the function can be expressed as

$$f(x) = \sum_{n=0}^{\infty} (x - x_0)^n \frac{f^{(n)}(x_0)}{n!}$$

This infinite series the is *Taylor Series expansion* about the point x_0. Defining $h = (x - x_0)$ this expansion is

$$f(x) = f(x_0) + hf(x) + h^2 \frac{f'(x)}{2!} + h^3 \frac{f''(x)}{3!} + \cdots$$

For a function of multiple variables $f(x_1, x_2, \cdots x_d)$ this expansion can also be used. Consider here the simple case in which the function $f(x, y)$ is expanded about the point (a, b). Then to first order

$$f(x, y) \approx f(a, b) + (x - a)\frac{\partial f(a, b)}{\partial x} + (y - b)\frac{\partial f(a, b)}{\partial y}$$

In general, the multidimensional Taylor Series expansion of $f(\mathbf{x})$ about a point $\mathbf{a}$ (where $\mathbf{x}$ is the vector of variables on which the function depends and $\mathbf{a}$ is a vector of the expansion point coordinates) can be expressed compactly as

$$f(\mathbf{x}) = f(\mathbf{a}) + Df(\mathbf{a})^T(\mathbf{x} - \mathbf{a}) + (\mathbf{x} - \mathbf{a})^T \frac{D^2 f(\mathbf{a})}{2!}(\mathbf{x} - \mathbf{a}) + \cdots$$

where $Df(\mathbf{a})$ and $D^2 f(\mathbf{a})$ are the gradient vector and Hessian matrix of the expansion point vector, respectively.

1.4 Rules of Integration

1) When the variable of integration in a definite integral is changed via a *u-substitution*, the limits of the integral must also be changed. Consider for example the integral

$$\int_{x_1=0}^{x_2=3} 2x(x^2 + 1)^3 dx$$

Letting $u = x^2 + 1$ we have $du = 2xdx$ and thus $dx = du/2x$. The limits of integration become $u_1 = 0^2 + 1 = 1$ and $u_2 = 3^2 + 1 = 10$ and thus the integral becomes

$$\int_{u_1}^{u_2=10} u^3 du$$

2) When the limits of a definite integral are exchanged, the integral is negated. This is easily seen since

$$\int_a^b f(x)dx = F(b) - F(a) = -(F(a) - F(b)) = -\int_b^a f(x)dx$$

Integration by Parts

Given an integral involving the product of two functions $u(x)$ and $v(x)$ of the form

$$\int_a^b u(x)v(x)dx$$

we may compute the result using integration by parts.

Lemma:

Let $u(x), v(x)$ be continously differentiable (that is, their derivatives are continuous) functions of x. Then we can compute

$$\int_a^b \frac{d}{dx}\left[u(x)v(x)\right]dx = \int_a^b \left[u'(x)v(x) + u(x)v'(x)\right]dx$$

where the product rule has been used. Using the fundamental theorem of calculus to simplify the left side and distributing the integral on the right side yields

$$[u(x)v(x)]_a^b = \int_a^b u'(x)v(x)dx + \int_a^b u(x)v'(x)dx$$

Noting that $du(x) = u'(x)dx$ and $dv(x) = v'(x)dx$, we may write after rearranging

$$\int u(x)dv = u(x)v(x) - \int v(x)du$$

Note that there should probably be some magic calculus to make the definite integral turn into an indefinite integral, but let's ignore that for now. We illustrate the use of the above result in the following example.

Example:

Use integration by parts to compute the indefinite integral

$$\int x\cos x dx$$

First, let $u(x) = x$ and $dv(x) = \cos x dx$ so that $du(x) = dx$ and $v(x) = \int \cos x dx = \sin x$. Using the result from the lemma, we have

$$\begin{aligned}\int u(x)dv &= u(x)v(x) - \int v(x)du \\ \int x\cos x dx &= x\sin x - \int \sin x dx \\ &= x\cos x + \cos x\end{aligned}$$

The idea behind this method is that u and dv can be chosen such that the integral $\int v(x)du$ is easier to compute than the original integral.

1.5 Gradients, Jacobians and Hessians

1.5.1 The Gradient

Let $f : R^n \to R$ be a scalar-valued function differentiable at a point p. For some vector v, the gradient is DEFINED AS the vector whose dot product with v yields the directional derivative in that direction at the point p, ie $D_v f = \nabla f(p) \cdot v$. More generally, let f be the same scalar-valued function and let $\sigma(t) : I \to R^n$ be a curve parameterized by $t \in I$ where $I \subset R$. If $\sigma(p) = a$ and $\sigma^{'}(p) = v$ then $\frac{d}{dt} f(\sigma(t))|_{t=p} = \frac{df}{d\sigma}\frac{d\sigma}{dt} = \nabla f \cdot v$ is the directional derivative of f in the direction v.

Here we define the gradient ∇f in a coordinate-free manner such that it applies to differentiable manifolds in the general case. However, in cartesian coordinates we define it explicitly as follows. Let $f = f(x_1(t), x_2(t), \cdots, x_n(t))$ so that

$$D_v f = \frac{df}{dt} = \frac{\partial \sigma}{\partial x_1}\frac{dx_1}{dt} + \frac{\partial \sigma}{\partial x_2}\frac{dx_2}{dt} + \cdots + \frac{\partial \sigma}{\partial x_n}\frac{dx_n}{dt} = \nabla f \cdot v$$

where $\nabla = \frac{\partial}{\partial x_1} + \cdots \frac{\partial}{\partial x_n}$ is the gradient operator and $v = \frac{dx_1}{dt} + \cdots + \frac{dx_2}{dt}$ is the direction of interest.

1.5.2 The Jacobian

For a vector-valued function $f : R^n \to R^m$, we can break f into m scalar-valued functions $\{f_1, \cdots, f_m\}$ and used the above results to define

$$D_v f = J(f)v$$

where

$$J(f) = \begin{pmatrix} \nabla f_1^T \\ \nabla f_2^T \\ \vdots \\ \nabla f_m^T \end{pmatrix}$$

is the Jacobian of f. In cartesian coordinates, it is the matrix

$$J(f) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_m} \end{pmatrix} \in R^{n \times m}$$

1.5.3 The Hessian

What about second partial derivatives? Again, let $f : R^n \to R$ be a scalar valued function with directional derivative $D_v f = \nabla f \cdot v = g(v)$. Note that g, like f, is a scalar-valued function which we will assume to be differentiable. As above, we define $\gamma(t) : I \to R^n$ to be a parameterized curve with $\gamma'(p) = v$ so that the directional derivative of g at p in the direction of v is given by $D_u g = \frac{\partial g}{\partial \gamma}\frac{d\gamma}{dt} = \frac{\partial g}{\partial \gamma} \cdot v$. However, $g = \nabla f \cdot u$ so by the product rule we have

$$\frac{\partial g}{\partial \gamma} = \frac{\partial \nabla f}{\partial \gamma} u + \nabla f \frac{\partial u}{\partial \gamma} = \nabla \cdot \nabla f \cdot u = \nabla^2 f \cdot u$$

In cartesian coordinates, since $H = \nabla^2 f$ is a matrix (it is the Jacobian of the gradient) we can finally write

$$D_{uv} f = D_u g = (\nabla^2 f \cdot u) \cdot v = u^T H v$$

We call $H \in R^{n \times n}$ the Hessian of f; it is the symmetric (as long as all second derivatives of f are continuous) matrix through which we can compute second-order directional derivatives of f. For example, the quadratic form $D_{uu} f = u^T H u$ provides the second derivative of f in the direction u. If $u^T H u < 0$ for all u then H is negative definite and hence all second derivatives of f are decreasing, making f a convex function. The Hessian has the form

$$H = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{pmatrix}$$

1.6 Data Interpolation

TODO: Come back and flesh out this section. For now, just some notes gleaned from the Ascher Numerical Methods text to jog my memory when I return here.

The basic idea is that we want to find some function which exactly passes through a set of points. For now, we consider only polynomials (though there are other choices of basis functions, of course). For two points, we find a line; for three points, we find a quadratic function; in general, we can fit N points with a polynomial of order $N - 1$. This results in a linear system having N equations and unknowns defined by a so-called Vandermonde matrix. As long as the x-coordinates of the points are unique, we can find a UNIQUE interpolating polynomial (this is important). However, this matrix has a particularly ill-conditioned structure. We can solve a different problem for the same exact polynomial (remember, it's unique) by using a different set of polynomials as a basis - one example is the Lagrange polynomials. These are polynomials of degree N, all

of which but one vanish at each data point and thus satisfy the interpolation condition. These are just two ways to solve the same problem.

We can do "better" (among other problems, high order polynomials can oscillate a lot) by using piecewise interpolants and imposing continuity conditions between successive approximating functions. For example, the simplest piecewise polynomial interpolation uses linear functions (piecewise linear or "broken line"). There are $2N$ coefficients to choose and N interpolation constraints plus N continuity conditions (on the zeroth derivative) so this can be solved. The problem with piecewise linear is obviously that the first derivatives are not continuous at the interpolated points. We can use a higher-order polynomial, for example a cubic, to solve this problem. We then have $4N$ coefficients and only $2N$ constraints. If the first derivative is also specified at each data point, we get $2N$ more constraints (because the functions preceding and following the point both need to have the specified derivative at the point and the problem can be solved. The piecewise interpolant can be made even smoother by using a fifth-order polynomial (total of $6N$ variables) and adding $2N$ more constraints using known accelerations at the points (note that specifying accelerations using a fourth-order polynomial would result in an overconstrained system with no exact solution). Since it can be shown from calculus of variations that a function which has sixth derivative equal to zero (for example, a fifth order polynomial) minimizes jerk along the trajectory (when starting and ending at rest?). Thus, using minimum jerk trajectories as interpolants and imposing position, velocity and acceleration constraints at the data points leads to a fully-constrained problem resulting in an interpolating solution which, in some sense, minimizes jerk.

Let's say you don't have information about the derivatives of the function at the data points - then you can instead use "splines," for example cublic splines. Rather than forcing derivatives of successive functions at the points to have a pre-specified derivative, this approach enforces continuity of derivatives between successive pieces. Cubic splines thus have $4N$ variables and $4N - 2$ constraints (because continuity is enforced only at interior points) so we additionally need 2 constraints at the endpoints (which can be specified in a number of ways). Higher-order splines are underdetermined and thus allow flexibility of the solution/allow enforcing smoothness at higher derivatives.

1.6.1 Interpolating MPC Solutions

Often in robotics we encounter the need to interpolate a discrete-time solution to a model predictive control (MPC) problem. For example, discrete-time LIPM-based approaches for center of mass planning involve solving a linear system (or in the case of added constraints, a QP) for COM jerk which is integrated to determine the center of mass position, velocity and acceleration. The advantage here is that we can make the timestep between control points large - on the order of hundreds of milliseconds - so that we can solve for the optimal motion over a horizon of several steps (usually 1-2 seconds in length).

In this case, we are solving directly for COM jerk which is piecewise constant at say 0.1s, however we want to control a robot at a timestep of 0.001s. The COM position, velocity and acceleration output from the MPC are simply the optimal jerk integrated

at the coarser timestep of 0.1s, so it doesn't actually make sense to try to track these values.

On the other hand, if we were to integrate the piecewise constant jerk at 0.001s we'd get something which diverges from the coarse plan and also doesn't achieve what we want. What's the best compromise here? Using cubic splines may work but leaves us with piecewise linear accelerations. Ideally we'd use quintic splines but then they may have too much freedom and oscillate (saw this with min jerk trajectories).

Chapter 2

Linear Algebra

2.1 Basic Theory

The theory of this section comes from my informal notes on Gilbert Strang's excellent online course[1] and associated textbook[2]; these notes are incomplete but touch on many elementary concepts in linear algebra and include insights both from the aforementioned course and from my own experience.

2.1.1 Matrix Multiplication

We can view the matrix-vector product of an $m \times n$ matrix A and an $n \times 1$ vector x to produce an $m \times 1$ vector b ($Ax = b$) in two ways. The first says that b is a linear combination of the columns of A; this gives rise to the column view in which we look for the combination of the columns which produces b. The second says that the components of b are equal to the dot product of each row of A with x; this gives rise to the row view in which we look for the intersection of lines (or planes or hyperplanes, depending on the dimension n).

The matrix-matrix product AB (where A is $m \times n$ and B is $n \times p$) can be computed in a number of ways:

- Each entry $ab_{ij} = \sum_k a_{ik} b_{kj}$
- Each column of AB is the product of A with each column of B individually
- AB is the sum of n $m \times p$ outer products (rank one matrices) formed from the product of the i^{th} column of A with the i^{th} row of B.

Multiplication can also be done in blocks instead of single entries and the same methods above apply.

[1] https://ocw.mit.edu/courses/mathematics/18-06-linear-algebra-spring-2010/
[2] http://math.mit.edu/ gs/linearalgebra/

2.1.2 Schwarz Inequality

The dot product (inner product) of two vectors is $u^T v = ||u||||v||$ mod $cos(\theta)$. Since the absolute value of cosine is at most 1, we know that $|u^T v| <= ||u||||v||$.

2.1.3 Triangle Inequality

As for the sides of a triangle, $||u + v|| \leq ||u|| + ||v||$. This comes from the Schwarz Inequality.

2.1.4 Gaussian Elimination

Gaussian Elimination solves $Ax = b$ directly for x by reducing the system to $Ux = c$ via elementary row operations. The idea is to clear out entries of A under the diagonal by adding multiples of higher rows to lower rows, producing (for an invertible matrix) an upper triangular matrix U. The entries on the diagonal of U are called the pivots.

By augmenting the right hand side b to A as $[Ab]$ and performing elimination on the augmented matrix, A is reduced to U and b is reduced to c. At this point, back substitution is used to solve for the components of x (which is the same solution as for the original system) from the bottom up. Elimination -¿ back substitution.

When A is $m \times n$, U will not be upper triangular but instead will be an Echelon (staircase) matrix. In these cases A is not invertible and there is not necessarily a solution to $Ax = b$ (or $Ux = c$). Note: Elimination works (preserves the solution x to $Ax = b$) because these row operations DO NOT ALTER THE ROW SPACE! The rows of U are linear combinations of the rows of A; x remains in the row space.

2.1.5 LU Decomposition

A can be factored into the product LU which is the result of Guassian Elimination. L is a lower triangular matrix which records the steps taken in elimination; it has 1s on its diagonal and its off-diagonal entries are the multipliers used in elimination. U is an upper triangular matrix which has the pivots on its diagonal.

A is reduced to U through a series of elementary row operations or, equivalently, by multiplication with a number of elementary matrices which perform these operations. In the LU decomposition, L is the product of the inverses of these E matrices, in the reverse order of which they were applied to A. The entries of L are then the multipliers used in elimination. NOTE: if row exchanges are necessary in elimination then the factorization is instead $PA = LU$.

LU Decomposition can also be written in the more symmetric LDU decomposition form. This is accomplished by dividing the rows of U by the pivots to produce 1s on the diagonal (like in L) and inserting the matrix D which has the pivots on its diagonal.

For symmetric matrices, the LDU factorization simplifies to $A = LDL^T$. This is advantageous because then only L and D need to be stored (and can be stored as one matrix with the 1's on the diagonal of L replaced with the elements of D).

LU Decomposition is used to solve $Ax = b$ as follows. First, factor A into L and U by elimination. Second, solve the intermediate system $Lc = b$ for c (the new right hand side resulting from elimination) and then finally solve $Ux = c$ for x. The advantage here is that A only needs to be factored once, after which L and U can be used to solve for x given any right hand side b. This is much more efficient than performing elimination on an augmented matrix $[Ab]$ for every new b.

2.1.6 Gauss-Jordan Elimination

We can also use elimination to compute the inverse of A rather than going directly to x. Just as elimination solved $Ax = b$ for x, it can solve $AA^{-1} = I$ for A^{-1}. The key is to solve for one column of the inverse at a time, ie (for a 3×3 matrix) solve $Ax_1 = e_1$, $Ax_2 = e_2$, $Ax_3 = e_3$ where the xs are the columns of A^{-1} and the es are the columns of the identity. This can be done all at once by augmenting the whole identity onto A and performing elimination, ie eliminate on $[AI]$. While Gaussian Elimination would stop once the matrix is in the upper triangular form U, Gauss-Jordan continues to the reduced echelon form by then eliminating entries above the pivots by adding lower rows to those above them and then finally by dividing each row by its pivot. This reduces $[AI]$ to $[IA^{-1}]$.

2.1.7 Matrix Inverse

A (square eg $n \times n$) matrix is invertible if there exists a matrix A^{-1} such that $A^{-1}A = AA^{-1} = I$. The inverse only exists if the matrix has full column/row rank, meaning that elimination results in n pivot columns/rows. Fundamentally, this means that A^{-1} only exists when the columns/rows of A are independent and span R^m and R^n respectively. In this case, it is always possible to solve $Ax = b$ for a particular solution x. In addition, $Ax = 0$ has only one solution the zero vector. When A is invertible, $Ax = b$ always has exactly one solution.

Some quick properties of the inverse:

- The inverse can be calculated using Gauss-Jordan Elimination or Cramers Rule.
- The inverse of a product ABC is $C^{-1}B^{-1}A^{-1}$.
- The inverse of A^T is the transpose of A^{-1}.
- The inverse of a diagonal matrix A (with all nonzero diagonal entries) has 1 divided by its diagonal entries.
- The inverse of a symmetric matrix is symmetric.

2.1.8 Determinant

A single number which encodes a lot of information about an $n \times n$ (square) matrix A. The determinant of A is 0 when the matrix is singular (has no inverse). This is because

the determinant is 0 when the matrix has dependent rows/columns which is the case when the inverse does not exist. There are 3 properties which define the determinant for any matrix A:

- The determinant of the identity (of any size) is 1.
- The determinant changes sign when two rows are exchanged.
- The determinant is a linear function of each row separately, meaning that multiplying a single row by constant t multiplies the determinant by t and adding row 1 of A to row 1 of A (which is the same as A except for a single row) is the sum of $det(A)$ and $det(A)$.

The above properties, taken together, lead to many others:

- Again, if two rows (or columns, since this always happens together) are equal then the determinant is 0.
- Elementary row operations which take A to U do NOT change the determinant of A, provided no row exchanges are needed to get to U. $det(A) = +/-det(U) = +/-$ product of the pivots.
- A matrix with a row of zeros has 0 determinant.
- If A is triangular then the determinant is the product of its diagonal entries (also because these are the pivots!)
- The determinant of AB is the product of the determinants of A and B. By this rule, the determinant of the inverse of A is always 1 divided by the determinant of A.
- A and A^T have the same determinant.

There are 3 ways to compute the determinant:

- The determinant is the product of the pivots of U since $det(A) = det(L)det(U)$ and $det(L)$ is always 1 (has 1s on its diagonal). If row exchanges were involved in elimination, then $det(P)det(A) = det(L)det(U)$ and $det(A) = +/-$ the product of the pivots. Note that the determinant of the upper-left corner submatrix A_k is the product of the first k pivots. Thus, the k^{th} pivot $d_k = det(A_k)/det(A_{k-1})$.
- The determinant is a sum of $n!$ terms, each of which comes from a column permutation. $det(A) = \sum det(P)a_{1i}a_{2j}a_{3k}\dots$ In this formula, sum over all column permutations and assign each term a sign according to the number of column exchanges required to create that permutation. For example, $a_{12}a_{21}a_{33}$ is multiplied by -1 since it requires 1 column exchange to restore the column order to $1, 2, 3$. $a_{12}a_{23}a_{31}$ requires 2 exchanges and is thus multiplied by $(-1)^2$ which is just 1.

- The determinant is the dot product of any row i of A with its cofactors using other rows, ie $det(A) = a_{i1}C_{i1} + a_{i2}C_{i2} + \ldots + a_{in}C_{in}$ where each of these cofactors is $C_{ij} = (-1)^{i+j}det(M_{ij})$ and the submatrix M_{ij} throws out row i and column j. This leads to a recursive definition for computing the determinant of an $n \times n$ matrix. A determinant of order n is a combination of determinants of order $n-1$, multiplied by the as. The determinants of order $n-1$ are combinations of determinants of order $n-2$ and so on.

Method 1 is how computers actually find $det(A)$. Method 2 is nice because it gives an explicit formula. Method 3 is handy when a matrix has many zeros, because then the as multiplying the cofactors cancel out whole terms in the sum of aCs.

Some quick determinant facts:

- The determinant of A is the area of the parallelogram constructed by the rows of A. Half of this determinant is the area of the triangle.
- The cross product comes from a special determinant with unit vectors i, j, k in the first row and two vectors u and v in the second and third rows. It is unique to vectors of three dimensions! The cross product produces a third vector orthogonal to u and v. $u \times v = ||u||||v|| \mod sin(\theta)$ whereas the dot product has $\mod cos(\theta)$. $(u \times v) = (v \times u)$ because of a row exchange.
- The scalar triple product $(u \times v)\dot{w}$ is the volume of a box with sides u, v, w. it is zero when the three vectors lie in the same plane, for a number of reasons.

2.1.9 Cramers Rule

An explicit formula for each component of x (rather than using elimination) can be found by using Cramers Rule as follows. Take the product of A and a matrix X_1 which is the identity with its first column replaced by x. Then $AX_1 = B_1$ where B_1 is just A with its first column replaced by b. $det(A)det(X_1) = det(B_1)$ but $det(X_1)$ is just x_1. Therefore, x_1 is $det(B_1)/det(A)$. Each component x_i of x can be solved for in this manner by computing $x_i = det(B_i)/det(A)$. This process can also (obviously) be used to solve for each element of A^{-1}. It turns out that $A^{-1} = C^T/det(A)$ where C is the cofactor matrix containing each cofactor C_{ij} of A. Any one element of A^{-1} is computed to be $(A^{-1})_{ij} = C_{ji}/det(A)$ ¡- note that this has C_{ji}, not C_{ij}!

2.1.10 Permutation Matrix

An $m \times m$ permutation matrix P multiplies an $m \times n$ matrix A on the left to perform a row exchange. Multiplying A by P on the right exchanges columns of A. There are $m!$ of the $m \times m$ permutation matrices.

The inverse of P is equal to its transpose. Pre/post multiplying by $P^{-1} = P^T$ brings the rows/columns of A back into their original order.

2.1.11 Reduced Row Echelon Form

A matrix R is in reduced row echelon form if its pivots are all 1s and the pivot columns contain only 0s above and below the pivots. Taken together, the pivot columns contain the $m \times m$ identity (they will be in order though not necessarily adjacent). Just as reduction takes $Ax = b$ to $Ux = c$, reduction all the way to R produces the system $Rx = d$.

The rref of an invertible matrix is the identity matrix of the same size. The inverse of A can be found by augmenting $[AI]$ and reducing A to R (which will simultaneously reduce I to A^{-1}).

2.1.12 Vector Space

A vector space is a set of vectors whose linear combinations ALWAYS remain within the space. This necessitates the inclusion of the zero vector into any vector space. Vector spaces include R^n, C (complex), M (space of all $n \times n$ matrices), F (space of all real functions), Z (zero vector only), etc. A vector space is spanned by a basis; the dimension of a vector space is the number of vectors in every basis.

Note: the solution to $Ax = b$ does NOT constitute a vector space because it does not include the zero vector! The solutions to $Ax = 0$ always form a vector space, but when $Ax = b$ has a solution then this particular solution shifts the space spanned by the nullspace vectors away from the origin, leaving out the zero vector. (think $x = x_p + c_1 x_{n1} + c_2 x_{n2} + \ldots$ the constants can be chosen zero but x_p will always be nonzero).

2.1.13 Basis

The basis for a vector space is a set of independent vectors which span the space, ie every vector in the space is a unique linear combination of the basis vectors. There is only one way to represent each vector in a space using its basis but the basis itself is not unique. For example, while the standard basis for R^n is comprised of the columns of the $n \times n$ identity matrix, the columns of any $n \times n$ invertible matrix (independent columns!) also span R^n.

The pivot columns of A are a basis of for its column space; the pivot rows of A are a basis for its row space.

2.1.14 Span

A set of vectors spans a space if their linear combinations fill the space.

2.1.15 Subspace

A subspace is a space within a vector space. The smallest subspace is Z (contains only the zero vector). Again, all linear combinations of vectors in the subspace remain in the subspace. If we have a vector space V then the set of vectors S in V spans the subspace SS which is all linear combinations of the vectors in S.

2.1.16 Orthogonal Subspaces

Two spaces V and W are orthogonal subspaces if every vector in V is orthogonal to every vector in W.

On a related note, vector space W is the orthogonal complement of a subspace V if it contains EVERY vector which is orthogonal to V, ie $W + V = R^n$.

2.1.17 Rank

The rank r of a matrix A is the number of pivot rows/columns in the reduced U or reduced echelon form R. This results from the fact that the rank gives the true dimension of the row/column spaces of A it is equal to the number of independent rows/columns.

The rank tells us a lot about the solutions $x = x_p + x_n$ to $Ax = b$:

- If $r = m = n$ then A is square and invertible (independent columns). $C(A)$ spans R^m and thus every right hand side b is in the column space there is a solution x_p for every b. $N(A)$ has dimension 0 and contains only the zero vector. $Ax = b$ has exactly one solution, the particular solution $x = x_p$.

- If $r = m < n$ (full row rank, A is short and wide) then the column rank is also $r = m$ and thus $C(A)$ spans R^m (every b is in the column space, so there is always a particular solution). $N(A)$ has dimension $r - m$ and the same number of special solutions. $Ax = b$ has infinite solutions (of the form $x = x_p + c_1 x_{n_1} + c_2 x_{n_2} + \ldots$).

- If $r = n < m$ (full column rank, A is tall and thin) then $C(A)$ only spans a subspace of R^m and thus there is not necessarily a solution for the given b (it must lie in the subspace spanned by the columns for their to be a particular solution). $N(A)$ has dimension 0 since the columns are independent and thus there are no special solutions. $Ax = b$ has either one solution ($x = x_p$) or 0 solutions. In this case we can solve for a least-squares (approximate) solution.

- If $r < m$ and $r < n$ then there are always solutions to $Ax = 0$ but only sometimes for $Ax = b$. By augmenting a generic vector b to matrix A and performing reduction, we can find the conditions which the components of b must satisfy in order for there to be a solution. $Ax = b$ has either zero solutions (we must resort to least-squares) or infinite solutions (of the form $x = x_p + c_1 x_{n_1} + c_2 x_{n_2} + \ldots$).

Note that a column vector a times a row vector a^T always produces a rank one matrix $A = aa^T$.

2.1.18 Column Space

The column space $C(A)$ of A is the subspace of R^m spanned by the columns of A. It contains all linear combinations $Ax = b$. If b is not in $C(A)$ then there cannot be a solution x to $Ax = b$. For example, the column space of a 3×2 matrix with independent columns is a plane in R^3 (careful: it is NOT the space R^2) The column space of an $n \times n$

matrix with independent columns is all of R^n. The column space has dimension rank r, the same as the dimension of the row space.

2.1.19 Nullspace

The nullspace $N(A)$ of A is the subspace of R^n spanned by all vectors x for which $Ax = 0$. $N(A)$ is spanned by $n - r$ independent vectors (where r is the rank of the matrix) and thus $dim(N(A)) = n - r$. This reflects the fact that the nullspace and the row space are orthogonal complements (their dimensions must add to n).

In order to $Ax = 0$ to have nonzero solutions ($x \neq 0$), the columns of A must be dependent. A (square) invertible matrix always has independent columns and thus its nullspace is only the vector $x = 0$. A noninvertible square matrix will have nonzero x in its nullspace, however. An $m \times n$ matrix with $n > m$ will always have nonzero solutions to $Ax = 0$ because its rank is at most m (and thus $n - m$ special solutions); this is because there are more vectors than needed to span R^m so they cannot possibly all be independent. On the other hand, an $m \times n$ matrix with $m > n$ has rank at most $r = n$; $Ax = 0$ has either zero or infinite solutions (the columns of A are not dependent by default as in the other case).

The solutions to this equation are called the special solutions and can be found as follows. Reduce A to U (or even further to R since elimination does not change the basis for $N(A)$) and choose the components of x multiplying each column of A as follows. For each of the $n - r$ special solutions, choose the component of x which multiplies one of the free (pivotless) columns to be 1 and the components which multiply the other free columns to be zero. Solve for the components of x multiplying the pivot columns. The special solutions go into the columns of the $n \times n - r$ nullspace matrix N for which AN equals the $m \times n - r$ matrix of zeros. In practice, the nullspace is usually found through a decomposition method (for example QR or SVD) for which there are efficient numerical implementations.

2.1.20 Row Space

The row space $C(A^T)$ is same as the column space of A^T it is the subspace of R^n spanned by the rows of A. The row space and the nullspace are orthogonal complements (the dot product of each row of A with any x in $N(A)$ yields 0). The pivot rows of A and U and R constitute a basis for the row space (elimination does NOT change the row space or the nullspace!)

Particular solutions x_p to $Ax = b$ are in the row space; this is evident because the dot product of x_p with each row of A produces a nonzero component of b.

2.1.21 Left Nullspace

The left nullspace $N(A^T)$ is the nullspace of A^T, ie all vectors x for which $A^T x = 0$ or equivalently $x^T A = 0^T$. $N(A^T)$ is a subspace of R^m and has dimension $m - r$. This reflects the fact that the column space and the left nullspace are orthogonal complements.

The last $m-r$ rows/columns of the identity constitute a basis for $N(R^T)$ since the last $m-r$ rows of R are always all zero. The last $m-r$ rows/columns of E (the elimination matrix which reduces A to R) constitute a basis for $N(A^T)$.

The left nullspace is important for projections and especially for least squares approximations (we require the projection error to be in the left nullspace of A, which is the matrix whose columns are the vectors which span the lower-dimensional space).

2.1.22 Fundamental Theorem of Linear Algebra/The Four Subspaces

The following theory summarizes the previous sections and comprises the Fundamental Theorem of Linear Algebra by relating the four subspaces - column space, row space, nullspace and left nullspace:

- $C(A)$ and $C(A^T)$ both have dimension r. The nullspaces $N(A)$ and $N(A^T)$ have dimensions $n-r$ and $m-r$ respectively.
- $N(A)$ is the orthogonal complement of the row space $C(A^T)$ in R^{n-r} and $N(A^T)$ is the orthogonal complement of the column space $C(A)$ in R^m.

Since $C(A^T)$ and $N(A)$ are complements, every solution x can be split into a row space component x_r (x_p, particular solution) and a nullspace component x_n. When A multiplies $x = x_r + x_n$, the nullspace component goes to zero since $Ax_n = 0$ and the row space component goes to the column space since $Ax_r = Ax = b$.

2.1.23 Projections

The projection p of a vector b onto a subspace spanned by the columns of a matrix A is the portion of b which lies along the vectors which span that subspace. The matrix which achieves this projection is P such that $p = Pb$.

For projection onto a line in the direction of a vector a, we seek the closest point on the line to the original vector b. This is $p = ta$ where t is some scalar. The key is that the error $b-p$ should be orthogonal to the vector a. We thus require $a^T(b-p) = 0$ and with $p = ta$ this leads to $t = a^Tb/a^Ta$. The projection is then $p = ta$ and the projection matrix which satisfies $p = Pb$ is $P = aa^T/a^Ta$. If a is a unit vector ie $||a|| = ||a||^2 = a^Ta = 1$ then the denominator, which is a normalization term, goes to one and $P = aa^T$.

2.1.24 Least Squares

Least squares is the process of projecting a vector b onto a subspace spanned by the columns of a matrix A. We wish to find a vector x which combines the columns of A such that $Ax = p$ approximates b in the lower-dimensional subspace with error $e = b - p = b - Ax$. Each component of x gives the projection onto each column of A; we choose these vectors such that the error in the approximation is orthogonal to each of the vectors which span the subspace (the columns of A). These conditions are called

the Normal Equations (we choose the projection such that the error is normal to the subspace). In this way, we do the best job we can of approximating b; the error represents information about b which we cannot capture using the given subspace. Combining the Normal Equations into a single matrix equation yields $A^T(b - Ax) = 0$. This of course leads to the solution $x = (A^TA)^{-1}A^Tb$. The projection p is then equal to Ax and the projection matrix P is $P = (A^TA)^{-1}A^T$. In practice we compute the so-called pseudoinverse $(A^TA)^{-1}A^T$ via other methods because solving the Normal Equations directly can have numerical stability issues.

Another way to view these conditions is that we require the error to be in the left nullspace of A. Since this subspace of R^m is the orthogonal complement of the column space of A, we are choosing the combination of the columns which produces the best approximation of b all the error is in the orthogonal space, so we have represented b as best as possible using the given information! The picture of the spaces shows that b is outside the column space, so no vector x in the row space can produce it. This is because $b = p + e$, where p is the projection (which is in the column space) and e is the error (which is in the left nullspace). The nullspace is the zero vector here because the columns of A are independent.

When A is an orthogonal matrix Q, the solution drastically simplifies to $x = Q^Tb$, $p = Qx$ and $P = QQ^T$. This is due to the fact that A^TA (again a normalization term) turns into Q^TQ which is just I (since the columns are orthonormal, or orthogonal and normalized). This coupling or correlation matrix is the identity, meaning that there is no interdependence among different directions! The projection p is then the sum of one-dimensional projections onto the columns of Q. No inverses to worry about just a bunch of dot products. Further, when Q is square then the subspace is the whole space and b is projected into itself, but because Q is orthogonal this actually breaks b into orthogonal pieces! This is the foundation of Fourier and other transforms. We can throw away pieces as we wish to get coarser and coarser approximations to b. We can do something similar with other decompositions like the SVD.

2.1.25 Orthogonal Matrix

A square matrix Q is orthogonal when $Q^T = Q^{-1}$, which results from the fact that the columns of Q are orthonormal (each is normal and orthogonal to all others). All matrices Q with orthonormal columns satisfy $Q^TQ = I$ (square or rectangular) but we only give the name orthogonal to square Q.

2.1.26 Gram-Schmidt

We can produce an orthogonal (or, even better, orthonormal) basis for a subspace from any set of vectors which span the space. The Gram-Schmidt Process is as follows:

- Begin with the first vector v_1 in the set (order doesnt actually matter). This vector, unchanged, will be the first vector in the new basis.

- Take the second vector v_2 and subtract from it its projection along the first vector in the new basis. Whats left is the component of v_2 which is orthogonal to v_1 (the information about v_2 which v_1 couldnt capture). This is the second vector in the new basis.
- Take the third vector v_3 and subtract from it its projections along the first two vectors in the new basis (not the original first two vectors). The remaining component is the third vector in the new basis.
- Continue this process for all n vectors.
- Finally, to make the basis orthonormal, divide each new basis vector by its length.

2.1.27 QR Decomposition

A matrix A whose columns are the basis for a subspace can be factored into $A = QR$, where Q is an orthonormal basis and R is a triangular matrix which produces this basis. This is the matrix form of Gram-Schmidt. This is easily seen by writing out the equations for the orthonormal basis vectors and solving them for the original basis vectors of A.

Assuming $A \in R^{m \times n}$ with $m > n$ and full column rank n, the so-called "full" QR decomposition results in $Q \in R^{m \times m}$ and $R \in R^{m \times n}$.

Here, the first n columns of Q form an orthonormal basis Q_1 for the column space $R(A)$ and the remaining $m - n$ columns form an orthonormal basis Q_2 for the left nullspace $N(A^T)$. Q is orthogonal and thus $Q^T Q = QQ^T = I$.

The first n rows and columns of the matrix R form an upper-triangular matrix which describes the steps taken to transform the basis given by the original columns of A into the orthogonal basis given by the columns of Q_1. The remaining $m - n$ rows of R are zero.

The factorization can thus be written in block form as

$$A = \begin{pmatrix} Q_1 & Q_2 \end{pmatrix} \begin{pmatrix} R_1 \\ 0 \end{pmatrix}$$

We thus have that $A = Q_1 R_1$; this is the "economy" factorization. Note that in this case Q_1 is not orthogonal but $Q_1^T Q_1 = I$. The fact that the basis for $N(A^T)$ gets multiplied by a matrix of zeros underlies the fact that the matrix A does not specify a basis for the nullspace - we can choose any basis, even an orthonormal one if we'd like!

We can solve the Least Squares problem using $A = QR$; it has the solution $x = R^{-1} Q^T b$. In practice, $Rx = Q^T b$ is solved for x by back substitution which is very fast.

2.1.28 Eigenvalues/Eigenvectors

Eigenvalues λ and eigenvectors x for a particular square matrix A solve the equation $Ax = \lambda x$. That is, when certain vectors called eigenvectors are multiplied by A, they remain in the same direction and are simply scaled by their corresponding eigenvalue.

These eigenvalue/vector pairs can be solved for as follows. First, move the right side over to produce $Ax - \lambda x = 0$. Factoring out x yields $(A - \lambda I)x = 0$. We are interested in finding nonzero vectors x which satisfy this equation; thus, we look for values of λ where the matrix in parentheses becomes singular ie vectors in the nullspace of $(A - \lambda I)$ (otherwise it is invertible and its nullspace is just the zero vector!) We thus require $det(A - \lambda I) = 0$ which results in a polynomial in λ (called the characteristic equation or characteristic polynomial) whose roots are the eigenvalues of A. We then substitute each λ (one at a time) into $(A - \lambda I)x = 0$ and solve for the corresponding eigenvector x.

Some important properties of the eigenvalues:

- The product of the eigenvalues is equal to the determinant of the matrix (and thus to the product of the pivots), ie $\prod_i \lambda_i = det(A)$
- The sum of the eigenvalues is equal to the sum of the diagonal entries of the matrix (the trace of A), ie $sum_i \lambda_i = tr(A)$
- The number of positive eigenvalues is equal to the number of positive pivots - this is important in determining stability in controls, for example.
- The eigenvalues of $A + B$ and AB are only $\alpha + \beta$ and $\alpha\beta$ (respectively) if A and B share the corresponding eigenvector x. Matrices which commute ($AB = BA$) share the same eigenvector matrix.
- The $n \times n$ identity matrix has all its n eigenvalues equal to 1 and has all vectors as its eigenvectors. Thus we have that the eigenvalues of $A + kI$ are each the eigenvalues of A plus k.

To solve for eigenvalues, we are essentially shifting the original matrix in order to make it singular; if A was singular to start then we know that (at least) one of its eigenvalues is zero. In fact, a singular matrix has $n - r$ of its eigenvalues equal to zero; these are the same special solutions which span the nullspace. In general (not just for $\lambda = 0$) a matrix can have repeated eigenvalues; the number of times an eigenvalue is repeated is called its algebraic multiplicity (AM). The number of independent eigenvectors corresponding to that eigenvalue is called its geometric multiplicity (GM). $GM \leq AM$ always!

The importance of eigenvalues/vectors is that they lead to a factorization which turns A into a diagonal matrix. This eigendecomposition results in $\Lambda = S^{-1}AS$ where S is the eigenvector matrix whose columns are the eigenvectors of A and Λ is the diagonal matrix of eigenvalues. Written as a decomposition of A, we have $A = S\Lambda S^{-1}$. This means that A can be written as the sum of n rank-one matrices, the i^{th} of which is $\lambda_i x_i x_i^T$ the eigenvalues/vectors essentially allow us to decouple different dimensions of A. Note that A can only be diagonalized to Λ when there are n independent eigenvectors ($GM = 1$ for every eigenvalue). Otherwise, S has no inverse because it has dependent columns (dependent eigenvectors).

Note that a matrix can have repeated eigenvalues with independent eigenvectors this is fine, we can still diagonalize. We can multiply eigenvectors by any nonzero constants

without changing the solutions to $Ax = \lambda x$, meaning that we can fill S with unit eigenvectors just as well.

The eigendecomposition allows for computation of the powers of a matrix easily the k^{th} power of A is simply $A^k = S\Lambda^k S^{-1}$. This is important in the solution of difference equations of the form $u_{k+1} = Au_k$. The factorization also leads to the matrix exponential $e^{At} = Se^{\Lambda t}S^{-1}$ which is important in the solution of systems of coupled linear, first order, ordinary, constant coefficient differential equations of the form $du/du = Au$. Note that any second (or higher) order equation can be broken into a coupled system of first order equations.

Solving Systems of Equations using the Eigendecomposition

The solution to the difference equation $u_{k+1} = Au_k$ involves powers of A. This equation starts from an initial vector u_0 and evolves with iterations. The solution using the power of matrix A is $u_k = A^k u_0 = S\Lambda^k S^{-1}u_0$. However, its insightful to also express the solution as $u_k = c_1\lambda_1^k x_1 + \ldots + c_n\lambda^k x_n$ as follows. First, find the eigenvalues λ and the eigenvectors x of A. Next, find $c_1, \ldots, c_n$ which express the initial vector as a linear combination of the eigenvectors $u_0 = c_1x_1 + \ldots + c_nx_n$. Finally, multiply each term by λ^k corresponding to each eigenvector to find u_k. The idea here is that we can break the solution down into a combination of the eigenvectors since these vectors do not change for powers of A (they are simply multiplied by the corresponding eigenvalue each time A is applied). We thus follow each component individually and take their combination to form u_k.

Differential equations of the form $du/dt = \lambda u$ have solutions of the form $u(t) = Ce^{\lambda}t$. Since we are working with a system of differential equations of this form combined into $du/dt = Au$, we have n pure exponential solutions of the form $u = xe^{\lambda t}$. When substituted into $du/du = Au$, this reduces to $Ax = \lambda x$ so were working with eigenvectors/values again. Solution works in the same way as for difference equations: find the constants that produce $u(0)$ from the eigenvectors and solve for $u(t) = c_1x_1e^{\lambda_1 t} + \ldots + c_nx_ne^{\lambda t}$. We also have the solution $u(t) = u(0)e^{At}$ where the matrix exponential is defined using the infinite series definition of e^x and is seen to factor into $e^{At} = Se^{\Lambda t}S^{-1}$. e^{At} always has its inverse e^{-At}, its eigenvalues are always $e^{\lambda t}$ and when A is skew-symmetric e^{At} is orthogonal so its inverse=transpose.

2.1.29 Similarity Transformations

The eigendecomposition of a matrix A resulting in $\Lambda = S^{-1}AS$ is one of infinite possible *similarity transformations* (albeit a very special one, since it diagonalizes A). Formally, two matrices A and B are *similar* if $B = P^{-1}AP$ for some invertible matrix P. This is nothing more than a representation of the matrix A in a new basis defined by P (just as the eigendecomposition represents a matrix in the basis of its eigenvectors, where its action is solely scaling in independent dimensions).

Since similar matrices represent the same linear transformation in different bases, they share fundamental properties including characteristic polynomial, rank, determi-

nant, trace, eigenvalues (along with their algebraic multiplicies) and geometric multiplicities (but not eigenvectors themselves, which are also transformed according to P).

2.1.30 Singular Value Decomposition

Recall that square matrices with independent eigenvectors can be diagonalized as $A = S\Lambda S^{-1}$ or, for symmetric matrices, $A = Q\Lambda Q^T$. The concept of eigenvectors and eigenvalues is restricted to such square, invertible matrices. However, the Singular Value Decomposition allows us to diagonalize any matrix $A \in R^{m \times n}$ having rank r using not one basis S or Q but two bases U and V. This factorization is then

$$A = U\Sigma V^T$$

where $U \in R^{nxr}$ and $V \in R^{mxr}$ each have orthonormal columns and $\Sigma \in R^{r \times r}$ is the diagonal matrix of singular values of A.

Forming $A^T A$ using the SVD reveals that

$$A^T A = V\Sigma^T U^T U\Sigma V^T = V\Sigma^T \Sigma V^T$$

which, since $A^T A$ is symmetric, means the right singular vectors (columns of V) are the eigenvectors of $A^T A$ and the singular values are the square roots of the eigenvlaues of $A^T A$.

One can thus compute the SVD by forming $A^T A$ and AA^T and computing the corresponding eigenvectors/eigenvalues. Alternatively, since the vectors in U and V are related through the singular values, one can find the vectors of one matrix directly from those of the other. Note that U and V are conventionally constructed such that the singular values down the diagonal of Σ are ordered from greatest to least (this specifies how to arrange the vectors in these matrices since in theory there is no "correct" way to order the eigenvectors - this convention is just most useful).

Forming AA^T (also symmetric) similarly results in

$$AA^T = U\Sigma\Sigma^T U^T$$

so the left singular vectors (columns of U) are the eigenvectors of AA^T and the singular values are also the square roots of the eigenvalues of AA^T.

The above form of the SVD is often referred to as the "economy-size" SVD. Just as we had the "full" QR decomposition which added to the matrix Q an orthonormal basis for the left nullspace, we have the "full" SVD which adds to U an orthonormal basis for the left nullspace $N(A^T)$ and adds to V an orthonormal basis for the nullspace $N(A)$. We thus have the same factorization

$$A = U\Sigma V^T$$

where $U \in R^{nxn}$ and $V \in R^{mxm}$ together specify orthonormal bases for all four fundamental subspaces and $\Sigma \in R^{m \times n}$ has gained an additional $m - r$ rows and $n - r$ columns of zeros. The full SVD is still computed from the eigendecompositions of $A^T A$

and AA^T but one additionally needs to find bases for the nullspaces in order to make U and V square. The way the SVD represents the four subspaces can be seen as follows. Since $V^TV = I$ we can write

$$AV = U\Sigma$$

which makes it clear that the vectors in V belong to the row space $R(A^T)$ and nullspace $N(A)$ of A (since AV is composed from inner products between the rows of A and the columns of V, and since the last $n - r$ colums of $U\Sigma$ are zero). Likewise, the vectors in U belong to the column space $R(A)$ and left nullspace $N(A^T)$ of A as can been seen in the same way by writing

$$U^TA = \Sigma V^T \rightarrow A^TU = V\Sigma$$

We have r of each of these vectors and thus (since the rank of the matrix is equal to the dimensionality of both row and column space) the matrices U and V must specify orthonormal bases for these spaces!

The SVD is a product of investigating the eigenproblem for A^TA (or AA^T). We have

$$A^TAv_i = \sigma_i^2 v_i$$

where we define v_i to already be normalized such that the eigenvectors are unit vectors. Multiplying both sides by v_i^T yields

$$\begin{aligned} v_i^TA^TAv_i &= \sigma_i^2 v_i^Tv_i \\ (Av_i)^T(Av_i) &= \sigma_i^2 \\ ||Av_i||_2^2 &= \sigma_i^2 \end{aligned}$$

which implies that $||Av_i||_2 = \sigma_i$.

Multiplying both sides of the eigenvalue equation for A^TA instead by A yields

$$\begin{aligned} A(A^TAv_i) &= \sigma_i^2 Av_i \\ (AA^T)Av_i &= \sigma_i^2 Av_i \\ AA^T(Av_i) &= \sigma_i^2(Av_i) \end{aligned}$$

and thus $u_i = \frac{Av_i}{||Av_i||_2} = \frac{Av_i}{\sigma_i}$ is a unit eigenvector of AA^T.

Note that A can also be written in terms of r rank one projections as

$$A = u_1\sigma_1v_1^T + u_2\sigma_2v_2^T + \cdots u_r\sigma_rv_r^T$$

using the SVD. Note that the sum of r rank one matrices is a matrix with rank r. An application such as image compression thus simply expresses the image as a matrix and approximates it using the n rank one matrices corresponding to the n largest singular values. This is then the best rank n approximation to the original image.

2.2 Complex Linear Algebra

The following sections introduce a number of concepts in linear algebra in complex space, denoted C (as opposed to linear algebra on the space of real numbers, R). This information (and many other concepts in this chapter) mostly comes from the text *Matrix Analysis* by Horn and Johnson as well as Jason Fulman's Applied Matrix Analysis course at USC.

2.2.1 Complex Inner Product Spaces

Let V be a complex inner product space. For $u, v \in V$ we define an inner product (u, v) on V such that

1. $(u, v) = (v, \bar{u})$
2. $(u, u) \geq 0$ with $(u, u) = 0 \leftrightarrow u = 0$
3. $(\alpha u + \beta v, w) = \alpha(u, w) + \beta(v, w)$ for $w \in V$ and $\alpha, \beta \in C$

With $V = C^n$ the inner product becomes

$$(u, v) = \sum_{i=1}^{n} u_i \bar{v}_i$$

Theorem: Cauchy-Schwartz Inequality:
If $u, v \in V$ then

$$|(u, v)| \leq ||u||||v||$$

When $V = R^n$ this is clear since $(u, v) = ||u||||v|| \cos\theta$.

Theorem: If V is a finite-dimensional inner product space and W is a subspace of V, then $V = W + W^{\perp}$.

V is thus the direct sum of W and $W^{\perp}$, ie there is a unique way to write any $v \in V$ as a linear combination of vectors in W and $W^{\perp}$.

Remarks: We work over the field $F = C$ because C is algebraically closed, ie all polynomials having coefficients in C have their roots in C.

2.2.2 Unitary Transformations

Definition: A linear map $T : V \to V$ over the complex field is *unitary* if $(Tu, Tv) = (u, v)$ for all u, v.

A unitary transformation preserves the complex inner products and hence preserves length. In addition, the converse is true: if $(Tv, Tv) = (v, v)$ for all v then T is unitary.

Theorem: A linear map is unitary iff it takes an orthonormal bases of V to another orthonormal basis of V; this is proven as follows.

Proof. $\rightarrow$ First, suppose that T is unitary and let $\{v_1, \ldots, v_n\}$ be an orthonormal basis of V. Since T is unitary, we have $(Tv_i, Tv_j) = (v_i, v_j) = \delta_{ij}$ for all i, j which is also an orthonormal basis of V.

$\leftarrow$ Now, suppose that both $\{v_1, \ldots, v_n\}$ and $\{Tv_1, \ldots, Tv_n\}$ are orthonormal bases of V and $u, w \in V$. Then we can use this basis to write $u = \sum_{i=1}^{n} \alpha_i v_i$ and $w = \sum_{i=1}^{n} \beta_i v_i$ where $\alpha_i, \beta_i \in C$ for all i. Since the v_i's are orthonormal, we have

$$(u, w) = \sum_{i=1}^{n} \alpha_i \bar{\beta}_i$$

Also, by linearity of T, we have

$$Tu = \sum_{i=1}^{n} \alpha_i T v_i, \quad Tw = \sum_{i=1}^{n} \beta_i T v_i$$

Since the Tv_i's are orthonormal, we have

$$(Tu, Tw) = \sum_{i=1}^{n} \alpha_i \bar{\beta}_i$$

so $(Tu, Tw) = (u, w)$ for all $u, w \in V$ and hence T is unitary.

□

Definition: if $T : V \to V$ is linear, define the *hermitian adjoint* of T written T^* by $(Tu, v) = (u, T^* v)$.

Lemma: If $T : V \to V$ is linear, so is T^* and the hermitian adjoint has the following properties.

1. $(T^*)^* = T$
2. $(S + T)^* = S^* + T^*$
3. $(\lambda S)^* = \bar{\lambda} S^*$
4. $(ST)^* = T^* S^*$

Proof. To show that T^* is linear if T is linear, we must show that $T^*(v+w) = T^*v + T^*w$ and $T^*(\lambda v) = \lambda T^* v$.

If $u, v, w \in V$ then

$$\begin{aligned}(u, T^*(v + w)) &= (Tu, v + w) \\ &= (Tu, v) + (Tu, w) \\ &= (u, T^* v) + (u, T^* w) \\ &= (u, T^* v + T^* w)\end{aligned}$$

Since this hold for all u we must have $T^*(v+w) = T^*v + T^*w$.

Similarly, for $\lambda \in C$ we have $(u, T^*(\lambda v)) = (Tu, \lambda v) = \bar{\lambda}(Tu, v) = \bar{\lambda}(u, T^*v) = (u, \lambda T^*v)$. Since this holds for all u we must have $T^*(\lambda v) = \lambda T^*v$. Thus, T^* is linear if T is linear.

Also, to show that $(T^*)^* = T$ we write $(u, (T^*)^*) = (T^*u, v) = (v, \bar{T^*}u) = (T\bar{v}, u) = (u, Tv)$. Since this holds for all u, we must have $(T^*)^* = T$.

□

Lemma: T is unitary iff $T^*T = 1$ (the identity map, since we're working in a coordinate-free way thus far).

Proof. $\rightarrow$ First, assume T is unitary and compute $(u, T^*Tv) = (Tu, Tv) = (u, v)$. Since this holds for all u, $T^*T = 1$.

$\leftarrow$ Now, assume that $T^*T = 1$. Then $(u, v) = (u, (T^*T)v) = (Tu, Tv)$ by the definition of the hermitian adjoint. Since this holds for all u, v we have that T must be unitary. □

We now seek to assign a basis to T and determine how T^* is represented in this basis.

Theorem: If $\{v_1, \ldots, v_n\}$ is an orthonormal basis of V and linear map T in this basis is represented by the matrix $[\alpha_{ij}]$, then the matrix of T^* in this basis is $[\beta_{ij}]$ where $\beta_{ij} = \bar{\alpha}_{ji}$. That is, the matrix representing T^* is the *conjugate transpose* of the matrix representing T in the basis.

Proof. Since the matrices of T and T^* in the given basis are $[\alpha_{ij}]$ and $[\beta_{ij}]$, we have

$$Tv_i = \sum_{k=1}^{n} \alpha_{ki} v_k, \quad T^*v_i = \sum_{k=1}^{n} \beta_{ki} v_k$$

Since the v_i's are orthonormal, we have

$$\beta_{ji} = (T^*v_i, v_j) = (v_i, Tv_j) = (v_i, \sum_{k=1}^{n} \alpha_{ki} v_k) = \bar{\alpha}_{ij}$$

Thus the matrix representing T^* in orthonormal basis V is the conjugate transpose of the matrix representing T in that basis. □

Proposition: If $U, V \in M_n$ are unitary, then UV is also unitary.

Proof. We have $(UV)(UV)^* = UVV^*U^* = I$ since U and V are unitary; thus, UV must be unitary. □

The set of unitary matrices forms a group together with the binary operation of matrix multiplication. Recall that a group is a set rogether with a binary operation which satisfies closure, associativity, identity and inverse. The product of two unitary matrices is unitary from the above proposition, guaranteeing closure; matrix multiplication is associative; the identity matrix is unitary; the inverse of any unitary matrix exists and is its conjugate transpose.

Since $U^*U = UU^* = I$, every column of U has Euclidean norm one and hence no entry of U may have absolute value greater than one. In addition, we know that the eigenvalues of U must have absolute value one. The unitary group is thus said to be *bounded*, and it can be shown that it is also *closed* since the limit matrix of an infinite sequence of unitary matrices is unitary. Since the group is both closed and bounded, it is *compact*.

2.2.3 Unitary Similarity

Let $U \in M_n$ be a unitary matrix and $A \in M_n$ by any square complex matrix. Since $U^* = U^{-1}$, $B = U^*AU$ is a similarity transformation; we say that A is *unitarily similar* to B.

Theorem: Let $U, V \in M_n$ be unitary, let $A, B \in M_{m,n}$ and suppose that $A = UBV$. Then we have $\sum_{i,j=1}^n |b_{ij}|^2 = \sum_{i,j}^n |a_{ij}|^2$. In particular, this is satisfied if $m = n$ and $V = U^*$ - that is, if A is unitarily similar to B.

Proof. Since $\mathrm{tr}A^*A = \sum_{i,j=1}^n |a_{ij}|^2$, we check that $\mathrm{tr}A^*A = \mathrm{tr}B^*B$. Compute $\mathrm{tr}A^*A = \mathrm{tr}(UBV)^*(UBV) = \mathrm{tr}(V^*B^*U^*UBV) = \mathrm{tr}V^*B^*BV = \mathrm{tr}B^*BVV^* = \mathrm{tr}B^*B$ □

Theorem (Schur Factorization): Let $A \in M_n$ have eigenvalues $\lambda_1, \ldots, \lambda_n$ and let $v \in C^n$ be the unit eigenvector corresponding to λ_1. There exists a unitary $U = [v, u_2, \cdots, u_n] \in M_n$ such that $U^*AU = T = [t_{ij}]$ is upper triangular with diagonal entries $t_{ii} = \lambda_i, i = 1, \ldots, n$.

Proof. Let $U_1 = [v, u_2, \cdots, u_n]$ be any unitary matrix whose first column is the unit eigenvector v. Then

$$\begin{aligned} U_1^*AU_1 = U_1^*[Av, Au_2, \cdots, Au_n] &= U_1^*[\lambda_1 v, Au_2, \cdots, Au_n] \\ &= \begin{bmatrix} v^* \\ u_2^* \\ \vdots u_n^* \end{bmatrix} \begin{bmatrix} \lambda_1 v & Au_2 & \cdots & Au_n \end{bmatrix} \\ &= \begin{bmatrix} \lambda_1 v^*v & v^*Au_2 & \cdots & v^*Au_n \\ \lambda_1 u_2^*v & & & \\ \vdots & & A_1 & \\ \lambda_1 u_n^*v & & & \end{bmatrix} = \begin{bmatrix} \lambda_1 & * \\ 0 & A_1 \end{bmatrix} \end{aligned}$$

because $v^*v = 1$ and the columns of U_1 are orthonormal. We have $A_1 = [u_i^*Au_j]_{i,j=2}^n \in M_{n-1}$ which has eigenvalues $\lambda_2, \ldots, \lambda_n$ (why?). If $n = 2$ then A has been triangularized. If not, let $w \in C^{n-1}$ be an eigenvector of A_1 associated with λ_2 and perform the preceding reduction on A_1 by choosing $U_2 \in M_{n-1}$ to be any unitary matrix whose first column is w. Thus,

$$U_2^*A_1U_2 = \begin{bmatrix} \lambda_2 & * \\ 0 & A_2 \end{bmatrix}$$

Thus, we let $V_2 = 1 \oplus U_2$ and compute the unitary similarity

$$(U_1V_2)^*AU_1V_2 = V_2^*U_1^*AU_1V_2 = \begin{bmatrix} \lambda_1 & * & * \\ 0 & \lambda_2 & * \\ 0 & 0 & A_2 \end{bmatrix}$$

where $A_2 \in M_{n-2}$ has eigenvalues $\lambda_3, \ldots, \lambda_n$. Continue this reduction to produce unitary matrices $U_i \in M_{n-i+1}, i = 1, \ldots, n-1$ and $V_i \in M_n, i = 2, \ldots, n-2$. The matrix $U = U_1V_2V_3 \cdots V_{n-2}$ is unitary and thus U^*AU is upper triangular from the process above. □

Remarks: Note that if $A \in M_n(R)$ has only real eigenvalues, the above triangularization can be performed using orthogonal matrices.

Definition: A *family* of matrices $F \subseteq M_n$ is a nonempty finite or infinite set of matrices. A *commuting family* is a family of matrices in which every pair of matrices commutes.

Definition: For a given $A \in M_n$, a subspace $W \subseteq C^n$ is *A-invariant* if $Aw \in W$ for all $w \in W$. For a given family $F \subseteq M_n$, a subspace $W \subseteq C^n$ is *F-invariant* if W is A-invariant for all $A \in F$.

Definition: A subspace $W \subseteq C^n$ is *reducible* if some nontrivial $(W \neq \{0\}, W \neq C^n)$ subspace of C^n is F-invariant; otherwise, it is *irreducible*.

Observation: Let $A \in M_n$ with $n \geq 2$ and suppose that $W \subseteq C^n$ is a k-dimensional subspace with $1 < k < n$. If W is a nonzero A-invariant subspace, then some vector in W is an eigenvector of A.

Proof. Choose a basis $s_1, \ldots, s_k$ for W and let $S_1 = [s_1 \cdots s_k]$. Choose any $s_{k+1}, \ldots, s_n$ such that $s_1, \ldots, s_n$ is a basis for C^n and let $S_2 = [s_{k+1} \cdots s_n]$. Let $S = [S_1S_2]$ and note that this matrix has linearly independent columns and thus is nonsingular.

If the subspace W is A-invariant, then $As_i \in W$ for each $i = 1, \ldots, k$ which means that (since S_1 is a basis for W) each As_i is a linear combination of $s_1, \ldots, s_k$. We may thus write $AS_1 = S_1B$ for some $B \in M_k$. Now, let λ be an eigenvalue of B and let $\xi \in C^k$ be the corresponding eigenvector. Then $S_1\xi \in W$ (and is nonzero because S_1 has rank k) since S_1 is a basis for the subspace W. But $A(S_1\xi) = (AS_1)\xi = S_1B\xi = S_1(\lambda\xi) = \lambda(S_1x)$ and thus A must have an eigenvector in W.

□

Lemma: Let $F \subset M_n$ be a commuting family. Then some nonzero vector in C^n is an eigenvector of every $A \in F$.

Proof. First, note that while it is trivial, there is always a nonzero F-invariant subspace of C^n - that is, C^n itself. Let m be the minimum dimension of all nonzero F-invariant subspaces of C^n.

□

Theorem: Let $F \subseteq M_n$ be a nonempty commuting family. Then there exists a unitary $U \in M_n$ such that U^*AU is upper triangular for all $A \in F$.

Proof. We know that for a commuting family F there is some nonzero vector in C^n which is an eigenvector of every $A \in F$. Thus, we perform the above triangularization by choosing at each step and eigenvector which is common to the commuting matrices. An outline of the process is as follows.

First, choose some v which is a (unit) eigenvector of every $A \in F$ and construct the unitary matrix U_1. Compute $U_1^* A U_1$ as above to obtain the matrix $A_2 \in M_{n-1}$; since $U_1^* A U_1$ is block triangular and since similarity preserves commutativity, A_2 commutes with all matrices of this form produced by reducing some $A \in F$ as described above. Thus, there is a unit eigenvector common to all A_2 which can be chosen to produce U_2 and so on. Continuing in this manner produces a U which simultaneously triangularizes all $A \in F$. □

2.2.4 Hermitian Transformations

Definition: A linear transformation T is *hermitian* or *self-adjoint* if $T^* = T$. Similarly, a transformation is *skew-hermitian* if $T^* = -T$.

Theorem: Any linear map $S : V \to V$ can be written in the form $S = A + iB$ where $A = \left(\frac{S+S^*}{2}\right)$ is hermitian and $B = \left(\frac{S-S^*}{2}\right)$ is skew hermitian. This is called the *Toeplitz Decomposition.*

Theorem: If T is hermitian, all its eigenvalues are real.

Proof. Let λ be an eigenvalue of T. Then there exists a vector $v \neq 0$ such that $Tv = \lambda v$. Compute:

$$\lambda(v, v) = (\lambda v, v) = (Tv, v) = (v, T^* v) = (v, Tv) = (v, \lambda v) = \bar{\lambda}(v, v)$$

Since $(v, v) \neq 0$, we have $\lambda = \bar{\lambda}$ which implies that λ must be real. □

Theorem: If T is unitary and λ is an eigenvalue of T, then $|\lambda| = 1$.

Proof. Let v be an eigenvector of T associated with the eigenvalue λ. Then

$$(v, v) = (Tv, Tv) = (\lambda v, \lambda v) = \lambda\bar{\lambda}(v, v)$$

Since $(v, v) \neq 0$, we have $\lambda\bar{\lambda} = |\lambda|^2 = 1$ and thus $|\lambda| = 1$. □

2.2.5 Normal Linear Transformations

Definition: A linear map T is *normal* if $TT^* = T^*T$. That is, the matrix representing a normal transformation commutes with its conjugate transpose. Unitary, hermitian and skew-hermitian transformations are normal.

Theorem: Let $N : V \to V$ be a normal linear map. Then there exists an orthonormal basis consisting of the eigenvectors of N in which N is diagonal. Equivalently, there exists a unitary matrix U such that $U^* N U$ is diagonal, that is, N is unitarily similar to a diagonal matrix.

Proposition: A normal matrix N is:

1. Hermitian iff all its eigenvalues are real.

2. Unitary iff all its eigenvalues have absolute value one.

Proof. 1. $\rightarrow$ If N is hermitian, all its eigenvalues must be real as shown earlier.

$\leftarrow$ Assume that N is normal and all its eigenvalues are real. Thus, there exists a unitary U such that $D = U^*NU$ is diagonal with real entries. it follows that

$$U^*NU = D = D^* = U^*N^*U$$

Thus, $N = N^*$ and N must be hermitian.

2. $\rightarrow$ If N is unitary, all its eigenvalues have absolute value one as shown earlier.

$\leftarrow$ Assume that N is normal and all its eigenvalues have absolute value one. Thus, there exists a unitary U such that $D = U^*NU$ is diagonal. Since $D^* = U^*N^*U$, we have

$$D^*D = (U^*N^*U)(U^*NU) = U^*N^*NU = I$$

since D is the diagonal matrix of eigenvalues of N. This implies that $N^*N = NN^* = I$ and thus N must be unitary. □

Definition: If $T : V \rightarrow V$ is hermitian and $(Tv, v) \geq 0$ for all $v \in V$, T is called *nonnegative* and it is written $T \succeq 0$. If T is hermitian and $(Tv, v) > 0$ for all $v \in V$, T is called *positive* and it is written $T \succ 0$.

Proposition: A hermitian linear map is:

1. Nonnegative iff all its eigenvalues are nonnegative.

2. Positive iff all its eigenvalues are strictly positive.

Proof. $\rightarrow$ Suppose that $T \succeq 0$ and let λ be the eigenvalue of T corresponding to eigenvector $v \neq 0$. Thus,

$$(Tv, v) = (\lambda v, v) = \lambda(v, v) \geq 0$$

and since $(v, v) > 0$, we must have $\lambda \geq 0$.

$\leftarrow$ Now, assume that the hermitian map has nonnegative eigenvalues. Since T is hermitian, it can be written in terms of an orthonormal basis of eigenvectors $\{v_1, \ldots, v_n\}$. Given any $v \in V$, we may thus write v in this basis as

$$v = \sum_{i=1}^{n} \alpha_i v_i$$

Thus, $Tv = \sum_{i=1}^{n} \alpha_i Tv_i = \sum_{i=1}^{n} \alpha_i \lambda_i v_i$ and we compute

$$(Tv, v) = \left(\sum_{i=1}^{n} \alpha_i \lambda_i v_i, \sum_{i=1}^{n} \alpha_i v_i \right) = \sum_{i=1}^{n} \lambda_i \alpha_i \bar{\alpha}_i = \sum_{i=1}^{n} \lambda_i |\alpha_i|^2$$

Since $\lambda_i \geq 0$ and $|\alpha_i|^2 \geq 0$ for all i, we conclude that $(Tv, v) \geq 0$ for any $v \in V$ and thus $T \succeq 0$. □

Proposition: A linear map $T : V \to V$ is nonnegative iff there exists some matrix A such that $T = AA^*$.

Proof. $\leftarrow$ Assume that there exists some A such that $T = AA^*$. Clearly, T is hermitian since $T^* = (AA^*)^* = AA^* = T$. Thus, for any $v \in V$ we have $(AA^*v, v) = (A^*v, A^*v) = (Av, Av) = ||Av||^2 \geq 0$ and thus T must be nonnegative.

$\rightarrow$ Now, assume that T is nonnegative. Then there exists a unitary U such that $D = U^*TU$ is diagonal with nonzero values equal to the eigenvalues of T. Since the eigenvalues of T are nonnegative, we may define $D^{1/2} = \text{diag}\{\sqrt{\lambda_1}, \ldots, \sqrt{\lambda_n}\}$ and $A = UD^{1/2}U^*$. We thus have $T = AA^* = (UD^{1/2}U^*)(U(D^{1/2})^*U^*) = U(D^{1/2})^2U^* = UDU^*$. □

Remarks: Over the field of real numbers, unitary matrices are called *orthogonal* and hermitian matrices are called *symmetric*. Many of the above properties apply to the real cases, as orthogonal matrices are unitary and symmetric matrices are hermitian.

2.3 Nearest Orthogonal Matrix

Let $R \in R^{3\times 3}$ and let $Q \in O(3)$ be an orthogonal matrix. We wish to find the orthogonal matrix Q closest to R in terms of the Frobenius norm, ie

$$Q = \arg \min_{Q \in O(3)} ||Q - R||_F^2 = \text{tr}[(Q - R)^T(Q - R)]$$

This reduces to

$$\begin{aligned} \text{tr}[(Q - R)^T(Q - R)] &= \text{tr}[Q^TQ - Q^TR - R^TQ + R^TR] \\ &= \text{tr}[Q^TQ] - 2\text{tr}[Q^TR] + \text{tr}[R^TR] \\ &= 3 - 2\text{tr}[Q^TR] + \text{tr}[R^TR] \end{aligned}$$

since $\text{tr}[A] = \text{tr}[A^T]$ and $Q^TQ = I$. It follows that in order to minimize the given function, we must minimize the second term in the final expression above. The problem thus becomes

$$Q = \arg \max_{Q \in O(3)} \text{tr}[Q^TR]$$

We can use the SVD to write $R = U\Sigma V^T$ so that $\text{tr}[Q^TR] = \text{tr}[Q^TU\Sigma V^T] = \text{tr}[V^T(Q^T\Sigma V^T)V] = \text{tr}[V^TQ^TU\Sigma]$ since similarity transformations preserve the trace. Defining the orthogonal matrix $Z = V^TQ^TU$, we may write

$$\text{tr}[Q^TR] = \text{tr}[Z\Sigma] = \sum_{i=1}^{3} z_{ii}\sigma_i \leq \sum_{i=1}^{3} \sigma_i$$

where the inequality follows from the fact that $z_{ii} \leq 1 \forall i = 1, 2, 3$ since the rows of Z are orthonormal. It follows that the new cost function is maximized when $Z = I$

and hence $Q = UV^T$. Note that this analysis holds for the nearest unitary matrix to produce, analogously, $Q = UV^*$

2.4 Damped Least Squares

Recall that the damped least squares solution to the linear system $Ax = b$ is

$$x_{DLS} = (A^T A + \lambda^2 I)^{-1} A^T b = A^T (AA^T + \lambda^2 I)^{-1} b$$

This is known to be a reliable solution for underdetermined problems with poor condition (ie, singular values close to zero) - but why? The SVD of the matrix $A^T(AA^T + \lambda^2 I)^{-1}$ is

$$\begin{aligned} A^T(AA^T + \lambda^2 I) &= (UDV^T)^T((UDV^T)(UDV^T)^T + \lambda^2 I)^{-1} \\ &= VD^T U^T (UDD^T U^T + \lambda^2 UU^T)^{-1} \\ &= VD^T U^T (U(DD^T + \lambda^2 I)U^T)^{-1} \\ &= VD^T U^T U(DD^T + \lambda^2 I)^{-1} U^T \\ &= VD^T (DD^T + \lambda^2 I)^{-1} U^T \\ &= VEU^T \end{aligned}$$

where $E = D^T(DD^T + \lambda^2 I)^{-1}$ is the diagonal matrix whose entries are

$$e_{i,i} = \frac{\sigma_i}{\sigma_i^2 + \lambda^2}$$

Thus, when $\sigma_i >> \lambda$ the damped least squares solution is approximately equal to the pseudoinverse (SVD) solution; on the other hand, when σ_i is close to zero then the damped least squares solution prevents the pseudoinverse from "blowing up" in these directions. Effectively, this is like a "smooth" cutoff of the rank-deficient directions (rather than a hard threshold, which would normally be used).

2.5 Principal Component Analysis (PCA)

Let $\tilde{x}_i \in R^m$ be a training sample and let μ_i be its corresponding mean. The covariance of n training samples is defined to be

$$\Sigma = \sum_{i=1}^{n} (\tilde{x}_i - \mu_i)(\tilde{x}_i - \mu_i)^T$$

Let $x_i = \tilde{x} - \mu_i$ be the i^{th} *centered* training sample and let

$$X = [x_1 \cdots x_n]$$

be the *design matrix* of training data points. Then we may write $\Sigma = XX^T$ where the normalization term $\frac{1}{n}$ has been dropped for simplicity.

Consider computing the covariance matrix of the projection of this dataset onto a k-dimensional subspace spanned by $\{u_1, u_2, \dots u_k\}$. For an arbitrary subspace we know that the projection of x_i onto $U = [u_1 \cdots u_k]$ is given by $\hat{x}_i = (U^TU)^{-1}U^Tx_i$ and so the design matrix of the data represented in the new basis is $\hat{X} = [\hat{x}_1 \cdots \hat{x}_n] = (U^TU)^{-1}U^T[x_1 \cdots x_n] = (U^TU)^{-1}U^TX$. It follows that the covariance matrix is

$$\begin{aligned}\Sigma_U &= \hat{X}\hat{X}^T \\ &= ((U^TU)^{-1}U^TX)((U^TU)^{-1}U^TX)^T \\ &= (U^TU)^{-1}U^TXX^TU(U^TU)^{-T}\end{aligned}$$

In the case that the u_i are an orthonormal basis, we have simply $\Sigma_U = U^TXX^TU$.

Consider computing the variance of the projection of the data onto a single dimension specified by the unit vector $\frac{u}{||u||}$. In this case we have

$$\sigma_u = \frac{u^TXX^Tu}{u^Tu}$$

The quadratic form involving $\Sigma = XX^T$ thus gives the variance of the data along any dimension! Further, we recognize this as a Rayleigh form; it can be shown that the maximum variance is given by the largest eigenvalue of Σ and that this occurs when u is equal to the corresponding eigenvector (since the eigenvectors of a symmetric matrix are orthogonal, we can simply choose these as the principal component directions and be done).

We thus choose the first principal component v_0 to be the eigenvector corresponding to the largest eigenvalue. We proceed to choose the remaining principal directions recursively as

$$v_k = \arg\max_{||u||=1} u^TX_kX_k^Tu$$

where

$$X_k = (X_{k-1} - \sum_{j=1}^{k-1} v_jv_j^TX_{k-1})$$

is the matrix of data points minus their projection onto the components which were already selected, ie the matrix of residuals after the $k-1$ projection step. Intuitively, we wish to choose the next principal direction to maximize the variance of what's left after the previous projections. Consider the case $k = 1$; we see that

$$\begin{aligned}
\Sigma_1 &= X_1 X_1^T \\
&= (X - v_0 v_0^T X)(X - v_0 v_0^T X)^T \\
&= (X - v_0 v_0^T X)(X^T - X^T v_0 v_0^T) \\
&= XX^T - XX^T v_0 v_0^T + v_0 v_0^T XX^T - v_0 v_0^T XX^T v_0 v_0^T \\
&= XX^T - 2XX^T v_0 v_0^T + v_0 v_0^T XX^T v_0 v_0^T \\
&= XX^T - 2\lambda_1 v_0 v_0^T + \lambda_1 v_0 v_0^T v_0 v_0^T \\
&= XX^T - \lambda_1 v_0 v_0^T \\
&= \sum_{i=1}^{m} \lambda_i q_i q_i^T - \lambda_1 q_1 q_1^T \\
&= \sum_{i=2}^{m} \lambda_i q_i q_i^T
\end{aligned}$$

where λ_i is the eigenvalue corresponding to the i^{th} largest eigenvector q_i of Σ_1. We know that maximizing $u^T \Sigma_1 u$ amounts to choosing the largest eigenvector of Σ_1 - which is simply the second largest eigenvector of Σ according to the above. We can see that, in general, v_k should be chosen as the eigenvector corresponding to the $k+1$ largest eigenvalue of Σ.

This implies that PCA reduces to nothing more than the eigendecomposition of the data covariance matrix XX^T or, equivalently, the Singular Value Decomposition (SVD) of X. Of course, PCA (and these equivalent decompositions) are applicable only to data points which exist in Euclidean space; it is not obvious how to generalize the notions of statistics and projections to the manifold setting.

2.6 Cholesky Decomposition

Let $A \in R^{n \times n}$ be a symmetric (in general, Hermitian), positive definite or semi-definite matrix. Since A is square, it can be decomposed (under certain conditions and ignoring permutations) as $A = LU$; however, since $A = A^T$ we have

$$U^T L^T = A^T = A = LU$$

which implies that $U = L$ and hence $A = LL^T$. This is known as the Cholesky decomposition. It is implied that this factorization is only valid for positive-definite and positive semi-definite A since $x^T Ax = x^T LL^T x = ||L^T x||^2 \geq 0$. When A is semi-definite, however, the factorization may not exist.

Note that we can also write this as $A = LDL^T$ where L is again lower triangular but with diagonal entries equal to one and D is the diagonal matrix of the eigenvalues of A. While the Cholesky decomposition is valid only for positive semi-definite matrices (since it involves square roots of the eigenvalues of A which would cause the result to otherwise

be complex), the so-called symmetric indefinite factorization $A = LDL^T$ avoids taking square roots and is thus valid for any square matrix.

The matrix L in the Cholesky factorization is lower triangular and a matrix square root $A^{1/2}$ of A. The square root of a matrix is not unique; for example, for some positive-definite A we have the spectral factorization (in this case the SED or symmetric eigenvalue decomposition) $A = Q\Lambda Q^T$ and thus can define $A^{1/2} = Q\Lambda^{1/2}Q^T$.

2.6.1 Generating samples from a multivariate Gaussian

Consider a matrix $X \in R^{m\times n}$ composed of n mean-centered data samples each of dimension m (each column is a sample). We know that the covariance matrix of these samples is given by $\Sigma = XX^T$. Now, suppose that the points are independent and identically distributed according to a standard normal distribution so that $\Sigma = I$. Now, let P be an arbitrary covariance matrix and L its matrix square root. Then we can correlate the sample according to P by transforming $\tilde{X} = LX$ so that $\tilde{\Sigma} = \tilde{X}\tilde{X}^T = (LX)(LX)^T = LXX^TL^T = LL^T = P$. Similarly, if X has covariance P then $\tilde{X} = L^{-1}X$ has covariance I - in other words, L^{-1} uncorrelates the data samples. The matrix L is any square root, but it is often computationally advantageous to choose the Cholesky factor. This is used for generating normally distributed data with a particular covariance matrix in MATLAB. It's also the basis for generating a set of "sigma points" which capture the mean and covariance of a distribution in an Unscented Kalman Filter - the $2n$ sigma points are chosen as the UKF mean plus/minus the columns of the matrix $\sqrt{nP}$ which guarantees that their mean is the UKF mean and their covariance matrix is $\Sigma = \frac{1}{n}(\sqrt{nP})(\sqrt{nP})^T = \sqrt{P}\sqrt{P}^T = P$ as desired. This is not the only way to choose the sigma points so that they capture the mean and covariance of the filter (the matrix square root is not unique) but it has been shown to be the "best."

2.7 Statistical Significance

2.7.1 Mahalanobis Distance

The *Mahalanobis distance* measures the distance between a point x and a distribution parameterized by mean μ and covariance matrix Σ. This can be used, for example, to determine whether or not x belongs to the set described by the distribution.

In a single dimension, we would intuitively measure the distance of x from the distribution by computing how many standard deviations x lies from the mean μ. That is,

$$d(x) = \frac{x - \mu}{\sigma}$$

The Mahalanobis distance generalizes this concept to higher dimensions; it is computed as

$$d(x) = \sqrt{(x - \mu)^T\Sigma^{-1}(x - \mu)}$$

Suppose we have a multivariate Gaussian distribution D with zero mean and the covariance matrix $\Sigma = diag(2, 5)$. This implies that the data in the y-direction is spread out more than the data in the x-direction. Accordingly, the distance of a point from the distribution should take this into account. By computing distance with respect to Σ^{-1}, we weight distance according to the spread of the data; the more spread in a direction, the less the distance. The Mahalanobis distance thus accounts for this effect.

Note that we can define a general distance metric between any two points $x, y \sim D$ as

$$d(x, y) = \sqrt{(x - y)^T \Sigma^{-1} (x - y)}$$

2.7.2 Chi-squared Distribution

The *Chi-squared distribution* (abbreviated χ^2 or χ^2_k to denote k degrees of freedom) is the distribution of a sum of the squares of k independent standard normal random variables.

That is, given $Z_1, \ldots, Z_k$ independent standard normal random variables (random variables each drawn from an independent 1-D Gaussian distribution having zero mean and unit variance) then

$$Q = \sum_{i=1}^{k} Z_i^2 \sim \chi^2_k$$

or in other words Q is Chi-squared distributed.

Relatonship between Chi-squared distribution and Mahalanobis distance

In the case of a multivariate normal distribution, the covariance matrix Σ is simply the identity. Then the squared distance of a point to the mean (which is zero) is χ^2-distributed.

For a general multivariate Gaussian disribution $D \sim \{\mu, \Sigma\}$, we know that we can uncorrelate a point x drawn from D by transforming it as $\bar{x} = L^{-1}x$ where L is computed from the Cholesky decomposition $\Sigma = LL^T$. It follows that $z = L^{-1}(x - \mu)$ is a vector of independent standard normal random variables and thus

$$z^T z = (x - \mu)^T L^{-T} L^{-1} (x - \mu) = (x - \mu)^T \Sigma^{-1} (x - \mu)$$

is χ^2-distributed. This is also the squared Mahalanobis distance of x from D, implying that for a general multivariate Gaussian distribution the χ^2 distribution is nothing more than the distribution over squared Mahalanobis distances.

The Chi-squared distribution allows us to determine the likelihood of data by using its cumulative distribution function (CDF). This is related to the *p-value* used in statistics.

Why not simply use the CDF of the multivariate Gaussian distribution itself in order to compute likelihood? The problem is that this CDF has no closed-form expression.

However, we know that the PDF is monotonically-decreasing with respect to the Mahalanobis distance, allowing us to instead compute the probability that a point falls inside the ellipsoid determined by the Mahalanobis distance. This is χ^2-distributed and hence has a closed-form CDF.

Kalman Filter Validation Gates

A *validation gate* is a method for determining whether a measurement in a Kalman Filter (here an EKF) agrees with the assumed measurement model (and thus whether it contains useful information and should be used for an update).

The idea is simple: compute the Mahalanobis distance of the innovations v_{k+1} of the current measurement with respect to the covariance of the innovations S_{k+1}. That is,

$$e^2 = v_{k+1}^T S_{k+1}^{-1} v_{k+1}$$

where $v_{k+1} = z_{k+1} - \hat{z}_{k+1}$ and $S_{k+1} = H_{k+1} P_{k+1} H_{k+1}^T + R_{k+1}$. Since we know that this distance e^2 is a squared Malahanobis distance and is thus χ^2-distributed, we can set a confidence level and reject any measurement for which $e^2 \geq g^2$ where g^2 is looked up using the χ^2 CDF and chosen confidence level.

Alternatively, if our measurements come from sensors prone to modes of failure which we know how to model, then we can use a similar method both to detect failure and switch measurement models. This is done by computing the squared Mahalanobis distance as above for all measurement models corresponding to all possible modes of failure and choosing the model with the smallest distance. In this way, we can continue to use the measurements rather than rejecting them. This could be useful, for example, if a GPS receiver fails in such a way that it continues to report correct positions but with much more noise; the measurements remain useful but their statistical properties have changed.

2.8 Skew-Symmetric Matrices

The quadratic form of a skew-symmetric matrix is

$$x^T A x = -x^T A^T x = -(x^T A x)^T = -x^T A x$$

since the transpose of a number is just itself. In order for

$$x^T A x = -x^T A x$$

to hold, we must have $x^T A x = 0$.

2.9 Positive Definiteness

Theorem: For any positive-definite matrix M and invertible Q, the forms $Q^T M Q$ and $Q^{-1} M Q$ are also positive-definite.

Proof. Since M is positive-definite, it admits a Cholesky factorization of the form $M = R^T R$ where R is upper-triangular. We thus have $Q^T MQ = Q^T R^T RQ$. For any x, we can thus write $x^T Q^T MQx = x^T Q^T R^T QRx = (QRx)^T (QRx) = ||QRx||^2 \geq 0$ with equality iff $x = 0$.

□

2.10 Cayley-Hamilton Theorem

Let A be an $n \times n$ matrix and let $p(\lambda) = \det(\lambda I - A)$ denote the *characteristic polynomial* of A. Then $p(A) = 0$.

This can be proven easily for the specific case in which A is diagonalizable, though the general case holds for all such matrices.

Using this assumption, A can be diagonalized as

$$A = SDS^{-1}$$

where S is an invertible matrix and D is the diagonal matrix

$$D = \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \ddots & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & \lambda_n \end{pmatrix}$$

The k^{th} power of D is then given by

$$D^k = \begin{pmatrix} \lambda_1^k & 0 & \cdots & 0 \\ 0 & \ddots & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & \lambda_n^k \end{pmatrix}$$

which implies that

$$p(D) = \begin{pmatrix} p(\lambda_1) & 0 & \cdots & 0 \\ 0 & \ddots & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & p(\lambda_n) \end{pmatrix}$$

Each of these numbers λ_j is an eigenvalue of A which implies that $p(\lambda_j) = 0$ for $j = 1, \cdots, n$; thus, $p(D) = 0$.

Since $A = SDS^{-1}$ implies $A^k = SD^k S^{-1}$ for all k we therefore have $p(A)A = Sp(D)S^{-1}$. Since $p(D) = 0$, we must have $p(A) = 0$.

2.10.1 Cayley-Hamilton Theorem (General Proof)

Recall that a vector x is an eigenvector of A corresponding to the eigenvalue s if

$$Ax = sx$$

This leads to the equation

$$(sI - A)x = 0$$

where $\Phi(s) = (sI - A)^{-1}$ is the *resolvent*.

Obviously this holds for any value of s with the vector $x = 0$ but this is uninteresting; an eigenvector is defined as a vector which, when acted upon by A is only scaled (does not change direction). We therefore first solve for eigenvalues by requiring $x \neq 0$; this implies that the resolvent does not exist - in other words, $det(sI - A)$ must be zero (otherwise only the zero vector would lie in its nullspace).

Expanding this determinant yields the *characteristic polynomial*

$$D(s) = |sI - A| = s^n + a_1 s^{n-1} + \cdots + a_{n-1}s + a_n$$

which is an n^{th} degree polynomial in s having coefficients $[1, a_2, \ldots, a_n]$ which are each functions of the elements of A. To verify that $D(s)$ must have degree n, note that one term is the product of the diagonal elements of $(sI - A)$, ie $(s - a_{11})(s - a_{22}) \cdots (s - a_{nn})$; this is clearly a polynomial of degree n with the coefficient of s^n being unity. Further, every other term will involve at most $n - 1$ diagonal elements and will thus have, at most, degree $n - 1$.

The *characteristic equation* of A is then defined to be

$$D(s) = |sI - A| = s^n + a_1 s^{n-1} + \cdots + a_{n-1}s + a_n$$

This is an equation with n roots which correspond to the eigenvalues of the system.

Note that from Cramer's rule the inverse of any matrix - here $(sI - A)$ - can be written as

$$\Phi(s) = (sI - A)^{-1} = \frac{\text{adj}(sI - A)}{|sI - A}$$

where $\text{adj}(sI - A)$ denotes the *adjoint matrix* of A. Each element of this matrix corresponds to the determinant of a submatrix of A having one row and one column removed; thus, the adjoint can be written as a matrix polynomial of the form

$$\text{adj}(sI - A) = E_1 s^{n-1} + E_2 s^{n-2} + \cdots + E_n$$

since each of its elements is a characteristic polynomial of order $n - 1$ corresponding to a submatrix of A. Note that since the adjoint is $n \times n$ each matrix E_i is also $n \times n$. We can then write

$$\text{adj}(sI - A) = (sI - A)^{-1}|sI - A| = E_1 s^{n-1} + E_2 s^{n-2} + \cdots + E_n$$

from Cramer's rule. multiplying both sides by $(sI - A)$ yields

$$|sI - A| = (sI - A)(E_1 s^{n-1} + E_2 s^{n-2} + \cdots + E_n)$$

Substituting the characteristic polynomial on the left hand side and expanding the multiplication on the right hand side produces

$$s^n I + a_1 s^{n-1} I + \cdots + a_{n-1} sI + a_n I = E_1 s^n + (E_2 - AE_1)s^{n-1} + \cdots + (E_n - AE_{n-1})s - AE_n$$

Equating powers of s on either side yields the following set of equations which can be solved recursively for the coefficient matrices composing the adjoint given the coefficients a_i of the characteristic polynomial.

$$\begin{aligned} E_1 &= I \\ E_2 - AE_1 &= a_1 I \\ E_3 - AE_2 &= a_2 I \\ &\vdots \\ E_n - AE_{n-1} &= a_{n-1} I \\ -AE_n &= a_n I \end{aligned}$$

Thi is not our purpose here, though. Instead we write in order

$$\begin{aligned} E_2 &= AE_1 + a_1 I = AI + a_1 I = A + a_1 I \\ E_3 &= AE_2 + a_2 I = A^2 + a_1 A + a_2 I \\ E_4 &= AE_3 + a_3 I = A^3 + a_1 A^2 + a_2 A + a_3 I \\ &\vdots \\ E_n &= A^{n-1} + a_1 A^{n-2} + \cdots + a_{n-1} I \end{aligned}$$

Multiplying the last equation by A yields

$$AE_n = A^n + a_1 A^{n-1} + \cdots + a_{n-1} A$$

but from before we have $-AE_n = a_n I$. Thus,

$$-a_n I = A^n + a_1 A^{n-1} + \cdots + a_{n-1} A$$

and hence

$$A^n + a_1 A^{n-1} + \cdots + a_{n-1} A + a_n I = 0$$

The result is the characteristic equation of A with powers of A substituted for powers of s; this is the *Cayley-Hamilton theorem* which essentially says that *every matrix satisfies its own characteristic equation.*

2.11 Quadratic Forms, Norms, and Singular Values

2.11.1 Vector Norms

The *p-norm* of a vector $x \in R^n$ is defined to be

$$||x||_p = (\sum_{i=1}^{n} |x_i|^p)^{1/p}$$

For $p = 2$, this is the familiar Euclidean norm $||x||_2 = \sqrt{x^T x}$. We may visualize these norms by considering the shapes of regions of constant norm. For $p = 1$, this region is a diamond; for $p = 2$ it is a circle. For $p > 2$ the shape approaches a square as $p \to \infty$. Thus, the $p = \infty$ norm is defined to be

$$||x||_\infty = \max_i |x_i|$$

2.11.2 Matrix Norms

How can we measure the "size" of a matrix? Since matrices represent linear transformations which act on vectors, we can define the norm of a matrix by its action on a vector. For any $A \in C^{m \times n}$ we thus define the *induced p-norm* to be

$$||A||_p = \max_{x \neq 0} \frac{||Ax||_p}{||x||_p} = \max_{||x||_p = 1} ||Ax_p||$$

where the two definitions above are equivalent. It can be shown that:
For $p = 1$, the norm is equal to the maximum of the column sums, ie

$$||A||_1 = \max_{1 \leq j \leq n} \sum_{i=1}^{m} |a_{ij}|$$

For $p = 2$, the norm is equal to the maximum singular value of A (maximum eigenvalue of $A^T A$), ie

$$||A||_2 = \lambda_{max}(A^T A)$$

For $p = \infty$, the norm is equal to the maximum of the row sums, ie

$$||A||_1 = \max_{1 \leq i \leq m} \sum_{i=1}^{n} |a_{ij}|$$

2.11.3 Quadratic Forms

The following expression

$$q = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} x_i x_j$$

is known as a *quadratic form*; in matrix notation this is

$$q = x^T Ax$$

It is important to note that the matrix A corresponding to q is not unique since the coefficient of the product $x_i x_j$ is $(a_{ij} + a_{ji})$; any matrix $\bar{A}$ with $(\bar{a_{ij}} + \bar{a_{ji}}) = (a_{ij} + a_{ji})$ will produce the same quadratic form q. The quadratic form is therefore specific to symmetric matrices as this guarantees uniqueness.

Consider the transformation of variables

$$x = Ty$$

The quadratic form can be expressed in terms of this transformation as

$$q = x^T Ax = y^T T^T ATy = y^T By$$

where $B = T^T AT$; this is known as a *congruence* transformation (A and B are congruent matrices). Such a transformation can always be found for a real, symmetric matrix A which diagonalizes A such that $B = \Lambda = T^T AT$ is diagonal. In this case T is the orthogonal matrix or the eigenvectors of A and Λ has the (all real) eigenvalues of A on its diagonal. The quadratic form thus becomes

$$q = y^T \Lambda y = \begin{pmatrix} y_1 & \cdots & y_n \end{pmatrix} \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{pmatrix} \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$$

Thus the quadratic form is

$$q = \lambda_1 y_1^2 + \lambda_2 y_2^2 + \cdots + \lambda_n y_n^2$$

2.12 Condition Number

First, the *sub-ordinate* matrix norm (in general) is defined as

$$||A|| = \max_{||x|| \neq 0} \frac{||Ax||}{||x||} = \max_{||x||=1} ||Ax||$$

The condition number of a matrix A is given by

$$\kappa(A) = ||A||||A^{-1}||$$

and is derived is as follows. Consider the linear system

$$Ax = b$$

The condition number describes how much the solution x changes due to a perturbation in the values in b. Thus, also consider the perturbed system

$$A\tilde{x} = b + \delta b$$

The magnitude of the absolute error in x is therefore given by

$$||\tilde{x} - x|| = ||A^{-1}\delta b|| \leq ||A^{-1}||||\delta b||$$

Also, since $||b|| = ||Ax|| \leq ||A||||x||$ for any x (by definition of the matrix norm) then

$$\frac{1}{||x||} \leq \frac{||A||}{||b||}$$

Therefore, the *relative* change in the solution is

$$\frac{||\tilde{x} - x||}{||x||} \leq ||A||||A^{-1}||\frac{||\delta b||}{||b||}$$

We thus find that the relative change in x is at most $\kappa(A) = ||A||||A^{-1}||$ times the relative change in b.

If the chosen norm $||A||$ denotes the 2-norm of the matrix A, then $||A||_2$ is computed as follows.

$$||A||_2 = \max_{||x||=1} \left[(Ax)^T(Ax)\right]^{\frac{1}{2}} = \max_{||x||=1} \left[x^T A^T Ax\right]^{\frac{1}{2}} = \max_{||x||=1} \left[\lambda_{A^TA} x^T x\right]^{\frac{1}{2}} = \max_i \sqrt{\lambda_{A^TA,i}}$$

2.13 Least Squares and Related Decompositions

2.13.1 Normal Equations

The first solution comes directly from forming the normal equations in their matrix-vector form

$$A^T Ax = A^T b$$

and solving for x using Gaussian Elimination via the MATLAB command $x\_NE = (A' * A)\backslash(A' * b)$. Since the columns of A are constructed to be linearly independent, this solution is unique.

If the matrix A is rank-deficient (has dependent columns) or if $n > m$ then A^TA is singular and the normal equations break down. In this case the system ius underdetermined - there exist an infinite number of solutions which minimize the squared 2-norm of the residual.

In this case, one must further constrain the problem. This is commonly done by seeking the solution which both minimizes the residual and has the smallest 2-norm $||x||_2$. This solution is computed using the (right??) pseudoinverse to be

$$x = A^T(AA^T)^{-1}b$$

2.13.2 QR Decomposition

Consider the same linear system written in terms of the "full" QR factorization:

$$QRx = b$$

Using the fact that $Q^TQ = I$ we have the system $Rx = Q^Tb$. We note that the the first n components of the vector Q^Tb are the projection of b onto the orthonormal basis for the column space; the last $m - n$ components are the projection of b onto the orthonormal basis of the left nullspace. It thus follows that

$$Rx = Q^Tb = \begin{pmatrix} c \\ d \end{pmatrix}$$

where c is the right hand side corresponding to the least-squares system $R_1x = c$ and d is the residual! Note that d is the residual expressed in terms of the orthonormal basis Q_2; it is not the same residual as for the original system (which is $r = b - Ax$) but it does have the same magnitude.

Recall the formulation of the least squares solution for an overdetermined system. We sought the solution which minimized the L_2 norm of the residual $r = b - Ax$ or, in terms of geometry, sent the residual to the left nullspace $N(A^T)$ to achieve an orthogonal projection into the column space $R(A)$. This is precisely what multiplying b by Q^T does! The vector c corresponds to the portion of b which lies in the *transformed* column space of A; we need only use R_1 to bring the solution back into the original (non-orthonormal) basis of A.

Since R_1 is upper-triangular, $R_1x = c$ is solved easily via backsubstitution. The solution x can thus be solved for using the command $x\_QR = R\_1\backslash(Q\_1' * b)$.

Note that, in general, the QR decomposition is not unique; there are an infinite number of orthonormal bases which can be chosen to compose Q. If A is square and invertible then enforcing $r_{ii} > 0$ in R guarantees uniqueness and if A is rectangular with full rank then enforcing $r_{ii} > 0$ in R_1 guarantees uniqueness.

The full QR decomposition is performed in MATLAB using Householder transformations (reflections) of the form

$$H = I - 2\frac{uu^T}{u^Tu}$$

Applying H to any vector x reflects the vector about the plane defined by the chosen normal u.

Note that H is both symmetric and orthonormal so $H^{-1} = H^T = H$ and thus $H^TH = H^2 = I$.

The goal of using such reflection matrices to perform QR decomposition is to reduce, one column at a time, the matrix A to the upper-triangular matrix R.

Note that, since H is orthogonal, $||Hx||_2^2 = x^TH^THx = x^Tx = ||x||_2^2$. Thus we can only use such matrices to transform a vector into another vector of the same length!

MATLAB Solution

The MATLAB solution to any linear system $Ax = b$ is computed using the command $x\_MAT = A\backslash b$. If A is invertible, the solution is directly computed using Gaussian Elimination. If A is rectangular with $m > n$, however, then MATLAB first performs QR factorization and then uses this to compute the least-squares solution as was done to compute x_QR.

2.13.3 Singular Value Decomposition

The pseudoinverse can be formed from the SVD and used to solve least-squares problems as

$$x = A^{\dagger}b = V\Sigma^{-1}U^{T}b$$

This solution x_SVD is defined for any matrix A; when $m > n$ and $rank(A) = n$, it identical (from a theoretical standpoint) to the other least-squares solutions described above. From a numerical standpoint, however, the SVD solution is the most stable of all solutions.

2.14 Conjugate Gradient Method

Again we wish to solve $Ax = b$ where $A \in R^{n\times n}$ is symmetric and positive-definite; the conjugate gradient method achieves this by expressing the solution in terms of a basis of conjugate directions. Two vectors u and v are said to be *conjugate (with respect to A)* if

$$u^{T}Av = 0$$

This defines an inner product with respect to A which can be written as $\langle u, v\rangle_A$. It's clear that if u is conjugate to v then v is conjugate to u since

$$\langle u, v\rangle_A = u^{T}Av = v^{T}A^{T}u = v^{T}Au = \langle v, u\rangle_A$$

Suppose we (somehow) find a set of *mutually conjugate* vectors $\{p_1, \ldots, p_n\}$; these form a basis for R^n so we can express the solution to $Ax = b$ as

$$x^{*} = \sum_{i=1}^{n} \alpha_i p_i$$

We can thus write

$$Ax^* = \sum_{i=1}^{n} \alpha_i A p_i$$
$$p_k^T A x^* = \sum_{i=1}^{n} \alpha_i p_k^T A p_i$$
$$p_k^T b = \alpha_k p_k^T A p_k$$

and thus we can solve for α_k for any $k = 1, \ldots, n$ as

$$\alpha_k = \frac{p_k^T b}{p_k^T A p_k}$$

The use of the conjugate gradient method as a *direct method* for solving $Ax = b$ as shown above is not particularly advantageous over other methods; its real use is as an *iterative method*. Because of the way in which the basis is chosen (formed from conjugate directions), we can actually get away with using only a few of these directions to approximate the solution. This is especially useful for solving large linear systems approximately.

The iterative method proceeds in a fashion similar to Gram-Schmidt orthogonalization; we can consider the conjugate gradient method as being similar to using an orthogonal basis to solve the problem (for example QR decomposition-based methods) however the use of conjugacy with respect to A rather than plain orthogonality means the chosen basis is "better" and leads to much faster convergence.

This method is called the conjugate gradient method because the iterative solution chooses the first basis vector as the gradient and chooses the remainder to be conjugate with respect to A.

The method can be applied to nonlinear systems as well for which the gradient is available (numerically or analytically).

2.15 Underdetermined Systems

Consider the linear system

$$Ax = b$$

where $A \in R^{m \times n}$ has $n > m$ and is full rank; that is, there are more unknowns than equations. In this case the columns of A span all of R^n; there is no left nullspace $N(A^T)$ and thus there must exist a solution to the system.

There is, however, a nullspace $N(A)$ with dimension $n - m$ and thus infinite solutions of the form $x = x_p + \sum_i^{n-m} c_i x_{ni}$ exist where the c_i's are arbitrary constants (since solutions in the nullspace do not affect the solution to the system). Any of these solutions will do, but for a particular application it may be desirable to shape the solution in some way be exploiting this redundancy in the solution.

In many applications, the most suitable solution is that with the minimum norm $||x||_2$ of all solutions. This solution must be $x = x_p$, otherwise we could simply add solutions from the nullspace to grow the norm without affecting the product Ax. We thus desire the unique solution x which lies entirely in the row space (has no component in the nullspace).

One method of obtaining this solution while simultaneously guaranteeing good numerical properties is through the use of QR decomposition.

We know that QR decomposition for an overdetermined system with full column rank ($A \in R^{m \times n}, r = n$) provides us with orthonormal bases for the column space $R(A)$ and left nullspace $N(A^T)$ of A. This means that the QR decomposition of the *transpose* of an underdetermined system with full row rank ($A \in R^{m \times n}, r = m$) provides us with orthonormal bases for the other two subspaces - the row space $R(A^T)$ and the nullspace $N(A)$. Thus,

$$A^T = QR = \begin{pmatrix} Q_1 & Q_2 \end{pmatrix} \begin{pmatrix} R_1 \\ 0 \end{pmatrix}$$

where $Q_1 \in R^{n \times n}$ is the basis for $R(A^T)$, $Q_2 \in R^{n \times n-m}$ is the basis for $N(A)$ and $R_1 \in R^{m \times m}$ is the upper triangular matrix used to orthogonalize A^T. We then have $A = R^T Q^T$ and thus the system $Ax = b$ becomes

$$\begin{pmatrix} R_1^T & 0 \end{pmatrix} \begin{pmatrix} Q_1^T x \\ Q_2^T x \end{pmatrix} = b$$

It's clear from the above expression that the solution vector can be broken into two components - one which is a projection onto the row space and the other which is a projection onto the nullspace. As discussed above, we want the solution to lie entirely in the row space - this means that the projection $Q_2^T x$ onto the nullspace should be zero. We therefore have the system

$$R_1^T Q_1^T x = b$$

for which the solution is the desired minimum norm solution. We define $z = Q_1^T x$ and solve first $R_1^T z = b$ by forward substitution (R_1^T is lower triangular) and then solve for x as $x = Q_1 z$.

Note that the pseudoinverse from the SVD also produces the minimum norm solution in such a case.

2.16 Projections Onto Subspaces

Formally, we define a (linear, orthogonal) projection on a finite-dimensional vector space W to be the operator $P : W \to U$ where U and V are the range and kernel of P, respectively.

Properties of this projection are as follows:

- First, P is idempotent ($P^k = P$ for any $k \geq 1$) - in other words, projecting more than once does not change the result.
- Second, P is the identity operator on U, ie $Px = x \forall x \in U$ - this follows from idempotence.
- Additionally, we have that $W = U \oplus V$ where $\oplus$ denotes the direct sum - this means that any vector $x \in W$ can be decomposed as $x = u + v$ where $u = Px$ and $v = x - Px = (I - P)x$.

From the last point, we have that P projects onto the subspace U and $Q = (I - P)$ projects onto the orthogonal complement of U, here V. This will come in handy later.

Now, let A be a matrix onto the columns of which we wish to project a vector b. This is precisely the problem least-squares solves. Recall that the general solution is $\hat{b} = Pb$ where $P = A(A^TA)^{-1}A^T$; we can easily verify that $P^2 = A(A^TA)^{-1}A^TA(A^TA)^{-1}A^T = A(A^TA)^{-1}A^T$ so P is idempotent. Keep in mind that this formula is derived with respect to the standard 2-norm - if we have chosen a weighted norm such as $||x||_W = \sqrt{x^TWx}$ then we would recover the weighted least-squares solution.

Anyway, the point is that the least-squares solution is just a projection. Now, let's assume A is a matrix composed of orthonormal basis vectors (such that $A^TA = I$). Then clearly the projection reduces to $P = AA^T$. The term $(A^TA)^{-1}$ in the general least-squares solution is a sort of "normalization factor."

Let's return to the fact that $(I - P)$ projects onto the orthogonal complement of the subspace spanned by projection P. This shows up in solving underdetermined systems, ie

$$Ax = b$$

where $A \in R^{m \times n}$ has $n > m$ and is full rank; that is, there are more unknowns than equations, leading to infinite solutions. We can decompose the general solution to the problem as

$$x = Pb + (I - P)b$$

where P is an orthonormal basis for the row space and $(I - P)$ an orthonormal basis for the nullspace of A. We can find these in infinite ways, but once choice is to use the SVD. Recall from $A = U\Sigma V^T$ that V holds an orthonormal basis for the row space and its orthogonal complement, the nullspace. We have that

$$x = A^\dagger b + (I - A^\dagger A)b_0$$

where $A^\dagger = V\Sigma^\dagger U^T$ and b_0 is an arbitrary vector. It's clear that $Ax = AA^\dagger b + A(I - A^\dagger A)b_0 = b$ since $AA^\dagger = I$, but let's take a closer look.

$$\begin{aligned} x &= A^\dagger b + (I - A^\dagger A)b_0 \\ &= (V\Sigma^\dagger U^T)b + (I - (U\Sigma V^T)(V\Sigma^\dagger U^T)b_0 \\ &= (V\Sigma^\dagger U^T)b + (VV^T - (V\Sigma^\dagger U^T)(U\Sigma V^T))b_0 \\ &= (V\Sigma^\dagger U^T)b + (V(I - \Sigma^\dagger\Sigma)V^T))b_0 \end{aligned}$$

The second term in this last line has $\Sigma^\dagger\Sigma$ which is a diagonal matrix with ones in the first r diagonal entries and zeros on the remaining $n - r$ - so $(I - \Sigma^\dagger\Sigma)$ instead has ones in the *last* $n - r$ spots. This is a "selector" matrix which picks out the orthonormal basis for the nullspace contained in the last columns of V. Hence, $(I - A^\dagger A)$ nothing more than a projector into the nullspace of A, allowing for an arbitrary choice of b_0.

Chapter 3

Differential Geometry

3.1 Curves, Surfaces and Manifolds

A *(topological) manifold* is, roughly speaking, a space which is locally Euclidean. A manifold is said to be of *dimension n* if there exists at every point a *homeomorphism* (continuous map having a continuous inverse) between a neighbourhood of that point and R^n. In this case, it is called an *n-manifold*; in general it is assumed that manifolds have a fixed dimension (they are *pure*). For example, a one-dimensional manifold is a curve; a two-dimensional manifold is a surface.

A *chart* is an invertible, structure-preserving map (homeomorphism) ϕ between an open subset of the manifold U and n-dimensional Euclidean space and is denoted by (U, ϕ). In general, it is not possible to cover the entire manifold with a single chart; the collection of multiple, overlapping charts which describes a manifold is called an *atlas*. Recall that we described curves in terms of a single parameter t along with a map $\alpha(t)$; similarly, we described surfaces in terms of the parameter space defined by u and v along with a map $x(u, v)$. An atlas of charts can thus be thought of as the extension of these parameterizations to general n-manifolds.

3.1.1 Smoothness

A *differentiable* or *smooth manifold* is a manifold which has a global *differential structure*, allowing differential calculus to be done on the manifold. Any manifold can be given local differential structure at a point by constructing the linear *tangent space* from its chart; however, in order to induce a global differential structure, we must impose smoothness constraints on the intersections of different charts in the manifold's atlas.

Consider an arbitrary n-manifold M and let ϕ_α and ϕ_β be maps from the open subsets $U_\alpha \subset M$ and $U_\beta \subset M$ to the subsets W_α and W_β of R^n; that is, we have $\phi_\alpha : U_\alpha \to W_\alpha$ and $\phi_\beta : U_\beta \to W_\beta$. Consider the intersection $U_{\alpha\beta} = U_\alpha \cap U_\beta \subset M$ which is the overlap of the domains of the two charts. The images of these sets according to each of the charts are $W_{\alpha\beta} = \phi_\alpha(U_{\alpha\beta}) \subset R^n$ and $W_{\beta\alpha} = \phi_\beta(U_{\alpha\beta}) \subset R^n$. We define the *transition map* between the charts as the mapping $\phi_{\alpha\beta} : W_{\alpha\beta} \to W_{\beta\alpha}$ between subsets in R^n given

by the chart composition $\phi_{\alpha\beta} = \phi_\beta \circ \phi_\alpha^{-1}$. Two charts are denoted C^k-*compatible* if $W_{\alpha\beta}$ and $W_{\beta\alpha}$ are open and the transition maps $\phi_{\alpha\beta}$ and $\phi_{\beta\alpha}$ have continuous derivatives of order k. If all charts in an atlas are C^k-compatible then they form a C^k-*atlas* which defines a C^k *differential manifold*.

For $k = 0$ we simply have a topological manifold, ie one whose transition maps are *continuous*. This definition of smoothness extends to lower-dimensional manifolds as well. For example, consider a piecewise curve described by the function

$$y = \begin{cases} f_1(x) = x & x \leq 0 \\ f_2(x) = x^2 & x \geq 0 \end{cases}$$

where $\phi_1 = f_1^{-1} = y$ and $\phi_2 = f_2^{-1} = \sqrt{y}$ are charts mapping y to x (from the curve to the interval) with domains which overlap at the point $x = 0$. The transition maps are $\phi_{12} = \phi_2 \circ \phi_1^{-1} == 1/x$ has derivative $-1/x^2$ which is clearly not continuous at the point $x = 0$, making it a C^0 differentiable manifold in R (in other words, a continuous curve). FIX THIS

3.1.2 Functions on Manifolds

Let $f : M \to R$ be a real-valued function defined on an n-dimensional differentiable manifold. The function is *differentiable* at a point $p \in U \subset M$ if and only if $\tilde{f} = f \circ \phi^{-1} : \phi(U) \subset R^n \to R$ is differentiable at $\phi(p)$. That is, f is differentiable iff the function defined on the manifold is differentiable with respect to the Euclidean space. Differentiability thus depends on the choice of chart ϕ at point $p \in U$ which is not unique since there may be many overlapping charts which contain p. However, if f is differentiable with respect to one chart at p then it is differentiable with respect to any other since the manifold is smooth (has smooth transition charts).

3.2 Quaternions and Rotations

The rotation group (or special orthogonal group in three dimensions) is a manifold denoted by $SO(3)$. The orthogonal group $O(n)$ is the group of all length-preserving linear transformations under the operation of composition; the special orthogonal group $SO(n)$ is the restriction of the orthogonal group to those transformations which additionally preserve orientation (handedness of the space).

From Euler's rotation theorem, we know that any rotation in three dimensions can be represented by a rotation around an axis by an angle. This naturally leads to visualizing the rotation group as the sphere in four-dimensional space or S^3. Recall that the n-dimensional real projective space RP^n is the topological space of all lines passing through the origin in R^{n+1}; this space is diffeomorphic to S^3 with antipodal points identified (or "glued together"). Thus, charts on $SO(3)$ which try to model the manifold using R^3 will inevitably run into problems.

The first such parameterization is that of Euler angles, which represent the rotation group by a sequence of three rotation angles. It can be shown that Euler angles lie on

a torus and that the map from angles to rotations (from the torus to the real projective space of dimension three) is not a covering map ie is not a homeomorphism (continuous bijection with a continuous inverse) and therefore is not a diffeomorphism (a differentiable homeomorphism). To see that the space of Euler angles is a torus, consider the following. Consider representing rotations in two dimensions with two Euler angles. These angles each vary from 0 to 2π and can thus be represented by a plane (subset of R^2). However, rotations are cyclic and so we must roll the plane up into a tube and then bend it around into a torus in order to satisfy this constraint. The Euler angles in 3D live on a higher dimensional torus.

Recall that we defined a regular parameterization of a curve to be one whose tangent vector is nonzero; likewise, we defined a regular parameterized surface to be one whose Jacobian map is nonsingular. Considering the first-order Taylor series expansions of each of these parameterizations, it's clear that regularity of parametrization implies that these maps are one-to-one. Analogously, we define a diffeomorphic chart to be one whose topological rank is equal to the dimension of the manifold. If $f : M \to N$ be a differentiable map between differentiable manifolds then we define the rank of f at a point $p \in M$ to be the rank of the derivative of f at p. Since the map from the torus to RP^3 is not diffeomorphic, this implies that there are points at which the rank of this map is less than three and thus changes in the tangent space of rotations don't show up as changes in Euler angles. We can think of this as the number of degrees of freedom of this system being reduced at these points; this is known as gimbal lock because it describes the phenomenon of gimbals aligning such that they no longer describe independent rotations.

Quaternions, on the other hand, provide a covering map from S^3 to RP^3 and hence are free from such singularities.

3.2.1 Interpolating Rotations (SLERP)

SLERP, or Spherical Linear Interpolation, is used to interpolate between elements of SO(3). If one simply attempts to perform linear interpolation between two rotations, the result will not be a valid rotation (it will not lie on the manifold). While it is possible to "fix" this issue by various means (for example, by performing linear interpolation and then projecting the result back onto the manifold), the proper way to interpolate elements of a manifold is to use its geodesics (curves of shortest length between points). For rotations $R_1, R_2 \in SO(3)$ we interpolate rotations between R_1 and R_2 as

$$R_t = e^{t \log (R_2^{-1} R_1)} R_1$$

where $t \in [0, 1]$. We thus perform linear interpolation on the manifold, and since SO(3) is isomorphic to the unit sphere S(3) we call this spherical linear interpolation. Note that R_1, R_2 denote either rotation

3.3 Lie Groups

A group is a set together with a binary operation which satisfies the properties of closure, associativity, identity and inverse. A Lie group - denoted G - is a differentiable manifold which is also a group (the group operation and the inverse operation are both differentiable). Similar to a Riemannian manifold, the local structure of a Lie group is described by its tangent space.

The tangent space at the identity is known as the infinitesimal group or, more commonly, the Lie algebra. The Lie algebra g of a Lie group G is a vector space together with an operation called the Lie bracket denoted $[x, y]$ for $x, y \in g$ which is

- Bilinear, ie satisfies $[x_1 + x_2, y] = [x_1, y] + [x_2, y]$, $[x, y_1 + y_2] = [x, y_1] + [x, y_2]$, $[\alpha x, y] = \alpha[x, y]$ and $[x, \alpha y] = \alpha[x, y]$.
- Alternating, ie satisfies $[x, x] = 0$.
- Satisfies the Jacobi identity $[x, [y, z]] + [z, [x, y]] + [y, [z, x]] = 0$.

Note that the first two properties imply anticommutativity of the Lie bracket since

$$[x + y, x + y] = [x, x] + [y, x] + [x, y] + [y, y] = [y, x] + [x, y] = 0$$

implies that $[y, x] = -[x, y]$. Also note that the Jacobi identity can be rewritten as $[x, [y, z]] = [[x, y], z] + [[z, x], y]$ which implies that the Lie algebra is associative iff $[[z, x], y] = 0$ for any $x, y, z \in g$. Finally, the Lie algebra is commutative or Abelian if $[x, y] = [y, x]$ which, from the anticommutativity property, implies that we must have $[x, y] = 0$. For example, if $g = M(n)$ then $[x, y] = xy - yx$; if $g = R^3$ then $[x, y] = x \times y$ (the cross product).

3.4 Principal Geodesic Analysis

Principal Geodesic Analysis (PGA) is a method for generalizing the concept of PCA to data which lie on a manifold M rather than in R^n. This section presents some of the basic tools used in the development of PGA.

First, we limit our approach to smooth manifolds equipped with an inner product $\langle \cdot, \cdot \rangle$ defined on the tangent space T_pM at each point $p \in M$ such that this inner product varies smoothly across the manifold. This makes M into a Riemannian manifold with corresponding Riemannian metric. The existence of such a metric allows us to perform calculus on the manifold - for example, given a smooth curve $\gamma(t) : R \to M$ on the manifold, we compute its length as $L_\gamma(a, b) = \int_a^b ||\gamma(t)'||dt$ where $||\gamma'(t)|| = \sqrt{\langle \gamma', \gamma' \rangle}$. Distance on a Riemannian manifold $d(x, y)$ is defined as the length of the shortest curve among all curves in M (called a geodesic curve) between two points $x, y \in M$.

Likewise, the existence of such a metric allows us to treat M as being locally Euclidean and use techniques from linear algebra within the tangent space at each point. In order to relate quantities in the tangent space and the manifold in the neighbourhood

of a point $p \in M$, we make use of the exponential map $\exp_p : T_pM \to M$. This is the generalization of the exponential function to Riemannian manifolds; roughly speaking, it relates a direction in the tangent space of M at a point $p \in M$ to a nearby point on the manifold. More precisely: given a vector $v \in T_pM$, there exists a geodesic curve $\gamma \in M$ with $\gamma(0) = p$ and $\gamma'(0) = v$ such that $\exp_p(v) = \gamma(1)$. Since all geodesics are defined as having constant speed, we know that $||\gamma'(t)|| = ||v||$; note again that the norm is with respect to the Riemannian metric at each point. We thus have that

$$d(p, \exp_p(v)) = L_\gamma(\gamma^{-1}(p), \gamma^{-1}(\exp_p(v))) = L_\gamma(0, 1) = \int_0^1 ||\gamma'(t)||dt = ||v|| \int_0^1 dt = ||v||$$

which implies that the exponential map preserves distances from p. Note that a manifold is geodesically complete if all its geodesics have domain R (rather than some subset of R, ie they extend indefinitely); this is important because, by the Hopf-Rinow theorem, it implies that there exists at least one geodesic between any two points on the manifold. Equivalently, it implies that the for each $p \in M$, the exponential map is defined on the entire tangent space T_pM.

However, $\exp_p$ is a diffeomorphism (isomorphism of a smooth manifold aka one-to-one function) only in the neighbourhood of $0 \in T_pM$, where its inverse is $\log_p : M \to T_pM$. It follows that, for $x \in M$ and $v \in T_pM$ such that $\exp_x(v) = y \in M$, we have

$$d(x, y) = d(x, \exp_x(v)) = ||v|| = ||\log_x(\exp_x(v))|| = ||\log_x(y)||$$

where the norm of a vector in the tangent space is simply the appropriate Euclidean norm. The map $\log_x(y)$ thus measures the geodesic distance from $x \in M$ to $y \in M$. We can also write this as

$$d(x, y) = ||\log(x^{-1}y)||$$

where the logarithm above is defined at the identity.

We begin the derivation with a brief review of PCA.

3.4.1 Principal Component Analysis (Review)

Given a dataset, PCA is a method for determining the directions in which the data has the most variability. The data can then be projected onto a basis of these principal components in such a way that the variability of the original dataset is preserved as explained below.

Let $x_i = \tilde{x} - \mu_i$ be the i^{th} centered data point and let

$$X = [x_1 \cdots x_n]$$

be the matrix of data points. Then the covariance matrix of this dataset is $\Sigma = XX^T$ where the normalization term $\frac{1}{n}$ has been dropped for simplicity. Now, consider computing the covariance matrix of the projection of this dataset onto a k-dimensional

subspace spanned by the vectors $\{u_1, u_2, \dots u_k\}$. For an arbitrary subspace we know that the least-squares projection of x_i onto $U = [u_1 \cdots u_k]$ is given by $\hat{x}_i = (U^T U)^{-1} U^T x_i$ or $\hat{x}_i = U^T x_i$ if the basis is orthonormal. The matrix of data points represented in the new basis is then $\hat{X} = [\hat{x}_1 \cdots \hat{x}_n] = U^T X$. It follows that the covariance matrix is

$$\begin{aligned}\Sigma_U &= \hat{X}\hat{X}^T \\ &= (U^T X)(U^T X)^T \\ &= U^T X X^T U\end{aligned}$$

Now, consider computing the variance of the projection of the data onto a single dimension specified by the unit vector $\frac{u}{||u||}$. In this case we have

$$\sigma_u = \frac{u^T X X^T u}{u^T u}$$

The quadratic form involving $\Sigma = XX^T$ thus gives the variance of the data along any dimension! Further, we recognize this as a Rayleigh form; it can be shown that the maximum variance is given by the largest eigenvalue of Σ and that this occurs when u is equal to the corresponding eigenvector (since the eigenvectors of a symmetric matrix are orthogonal, we can simply choose these as the principal component directions and be done).

We thus choose the first principal component v_0 to be the eigenvector corresponding to the largest eigenvalue. We proceed to choose the remaining principal directions recursively as

$$v_k = \arg\max_{||u||=1} u^T X_k X_k^T u$$

where

$$X_k = (X_{k-1} - \sum_{j=1}^{k-1} v_j v_j^T X_{k-1})$$

is the matrix of data points minus their projection onto the components which were already selected, ie the matrix of residuals after the $k-1$ projection step. Intuitively, we wish to choose the next principal direction to maximize the variance of what's left after the previous projections. Consider the case $k = 1$; we see that

$$\begin{aligned}
\Sigma_1 &= X_1 X_1^T \\
&= (X - v_0 v_0^T X)(X - v_0 v_0^T X)^T \\
&= (X - v_0 v_0^T X)(X^T - X^T v_0 v_0^T) \\
&= XX^T - XX^T v_0 v_0^T + v_0 v_0^T XX^T - v_0 v_0^T XX^T v_0 v_0^T \\
&= XX^T - 2XX^T v_0 v_0^T + v_0 v_0^T XX^T v_0 v_0^T \\
&= XX^T - 2\lambda_1 v_0 v_0^T + \lambda_1 v_0 v_0^T v_0 v_0^T \\
&= XX^T - \lambda_1 v_0 v_0^T \\
&= \sum_{i=1}^{m} \lambda_i q_i q_i^T - \lambda_1 q_1 q_1^T \\
&= \sum_{i=2}^{m} \lambda_i q_i q_i^T
\end{aligned}$$

where λ_i is the eigenvalue corresponding to the i^{th} largest eigenvector q_i of Σ_1. We know that maximizing $u^T \Sigma_1 u$ amounts to choosing the largest eigenvector of Σ_1 - which is simply the second largest eigenvector of Σ according to the above. We can see that, in general, v_k should be chosen as the eigenvector corresponding to the $k+1$ largest eigenvalue of Σ.

This implies that PCA reduces to nothing more than the eigendecomposition of the data covariance matrix XX^T or, equivalently, the Singular Value Decomposition (SVD) of X. Of course, PCA (and these equivalent decompositions) are applicable only to data points which exist in Euclidean space; it is not obvious how to generalize the notions of statistics and projections to the manifold setting.

3.4.2 Extension of PCA to manifolds

In order to ease the transition to manifolds, we define PCA in an alternative manner as follows. The first principal component is the one-dimensional subspace satisfying

$$u^{(1)} = \arg \min_{||v||=1} \sum_{i=1}^{N} ||x_i - vv^T x_i||^2$$

We then define the remaining components recursively as

$$u^{(k)} = \arg \min_{||v||=1} \sum_{i=1}^{N} ||x_i - \tilde{x}_i||^2$$

where

$$\tilde{x}_i = \sum_{j=1}^{k-1} v_j v_j^T x_i + vv^T x_i$$

where $\tilde{x}_i$ is the projection of x_i onto the first k principal components. Intuitively, at each step we seek to choose $u^{(k)}$ in order to minimize the norm of the residual $x_i - \tilde{x}_i$ given the $k-1$ principal components which were already chosen. Since the projection and the residual are orthogonal, we can consider minimizing the residual to be equivalent to maximizing the projection; the two approaches are thus equivalent in the linear case.

We have thus shown that dimensionality reduction via PCA is performed by projecting the high-dimensional data onto a linear subspace of lower dimension spanned by the principal components of the data. Likewise, PGA projects data onto a series of geodesic submanifolds which best capture the variability in the data. In order to describe the method, we must first define the concepts of mean, variance and projection in the manifold setting.

3.4.3 Statistics of Manifold Data

The arithmetic mean of a set of points $\{x_i, \ldots, x_N\} \in R^n$ is defined as

$$\mu = \frac{1}{N} \sum_{i=1}^{N} x_i$$

This definition results from the optimization problem

$$\mu = \arg\min_y \sum_{i=1}^{N} ||y - x_i||^2$$

How do we extend this mean to manifolds? We might first try embedding manifold-valued data in R^n and computing the above mean. However, the result is not guaranteed to lie in M so we then project it to the nearest point on the manifold (in terms of Euclidean distance). We call this the *extrinsic mean* of the data; it does not make use of the Riemannian structure of M and it depends entirely on the chosen embedding.

On the other hand, we may define the mean of manifold-valued data more methodically as the solution to the problem

$$\mu = \arg\min_{y \in M} \sum_{i=1}^{N} d(x_i, y)^2$$

where the distance metric is the Riemannian metric. For a general metric space and associated distance metric, this is known as the Frechet mean or *intrinsic mean* of the set. The result is automatically constrained to lie on the manifold and depends only on the chosen metric (which is a property intrinsic to the manifold).

Recall that the distance between two points $x, y \in M$ can be written as $d(x, y) = ||\log(x^{-1}y)||$. For $x, y \in R$, we know that $z = \log(e^x e^y) = \log(e^{x+y}) = x + y$; however, this is not true in general. For a Lie group G with Lie algebra g, we have the Campbell-Baker-Hausdorff (CBH) forumla which states that for $x, y \in g$

$$z = \log(e^x e^y) = x + y + \frac{1}{2}[x, y] + \frac{1}{12}[x, [x, y]] + \cdots$$

where $[x, y]$ denotes the Lie bracket of x and y. To first order, we may write $z \approx x+y$; we can thus write

$$d(x, y) = ||\log(\exp(\log(x^{-1}))\exp(y))|| \approx ||\log(x^{-1}) + \log(y)|| = ||\log(y) - \log(x)||$$

The problem of determining the intrinsic mean thus becomes

$$\mu = \arg\min_{y \in M} \sum_{i=1}^{N} ||\log(y) - \log(x)||^2$$

which has the solution

$$\log(\mu) = \frac{1}{N} \sum_{i=1}^{N} \log(x_i)$$

it follows that the first-order estimate of the intrinsic mean is then

$$\hat{\mu} = \exp\left(\frac{1}{N} \sum_{i=1}^{N} \log(x_i)\right)$$

The true mean can be found through a process of iterative refinement. Recall that the CBH formula is defined for elements in the Lie algebra, ie in the tangent space at the identity; by shifting our data to the identity, we limit the approximation error. We thus proceed by

- Choosing one of the data points (or any other known point on the manifold) to be the initial mean.
- Left translating all data points by the inverse of the initial mean such that they lie in the neighborhood of the identity (with the initial mean becoming the identity).
- Computing the mean of the translated points.
- Updating the previous mean using the mean computed at the identity.

This process is repeated until the update of the mean falls below a predetermined threshold. More formally,

It can be shown that this algorithm is equivalent to gradient descent on the original cost function and that it converges if the data is well-localized and the initial mean is chosen carefully.

The variance of data $\{x_i, \ldots, x_N\} \in R$ can be defined as

$$\sigma^2 = \frac{1}{N} \sum_{i=1}^{N} (x_i - \mu)^2$$

Analogously, for manifold data we define the variance as

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N} d(\mu, x_i)^2 = \frac{1}{N}\sum_{i=1}^{N} ||\log(\mu^{-1}x_i)||^2$$

which is clearly a generalization of the real-valued case.

3.4.4 Geodesic Subspaces and Projection

In R^n, we define the *orthogonal* projection of a vector x onto the direction specified by a unit vector v as $v^T x$. This is the coordinate of the projection along the direction given by v; in the original space, the projection is simply $\tilde{x} = vv^T x$. This definition results from the fact that we choose $\tilde{x}$ to be a scalar multiple of v such that the projection error (or residual) is orthogonal to v.

In the manifold case, we don't have the notion of orthogonality which we used in the linear case. Instead, we define the projection of a point x on a set H more generally as

$$\tilde{x} = \arg\min_{y \in H} d(x, y)^2$$

In the linear case, we considered projections onto linear subspaces ie lines. The natural generalization to manifolds is projection onto geodesics, which can be viewed as straight lines on the manifold. We refer to this as a geodesic subspace or geodesic submanifold H of the manifold M and use the Riemannian metric as the distance function in the definition of the projection.

Chapter 4

Dynamics

4.1 Intro to Analytical Dynamics

Consider a system of n point masses (particles) whose motion is described by the coordinates of each particle with respect to some inertial Cartesian frame of reference. Each of these particles has three positional components; the components of all n particles can thus be stacked into the $3n$ dimensional vector $x(t)$. Assume that the initial position and velocity of each particle, denoted $x(t_0)$ and $\dot{x}(t_0)$, are known. Also assume that the impressed force on each particle is known, ie the vector $F(t) \in R^{3n}$ is given. The motion of the unconstrained system is then described by Netwon's law:

$$M\ddot{x}(t) = F(x(t), \dot{x}(t), t)$$

where $M \in R^{3n \times 3n}$ is the diagonal matrix of particle masses (which appear in threes) and the impressed force can clearly be a function of the system state and time. We can solve for the motion of the particles simply as

$$\ddot{x} = M^{-1}F = a(t)$$

Of course, this problem is uninteresting and has few practical applications because we have assumed that the particles move independently of one another and freely in space. We thus enforce constraints on the motion of the particles, expressed as the set of m consistent constraint equations of the form

$$D(x(t), t)\dot{x} = g(x(t), t)$$

where $D \in R^{m \times 3n}$ and $g \in R^{m \times 1}$ specify the set of constraints.

The central problem we're concerned with in analytical dynamics is then the following: given the position $x(t)$, velocity $\dot{x}(t)$ and impressed force $F(t)$ acting on a system of particles at time t, determine the instantaneous acceleration $\ddot{x}(t)$ of the system in the presence of a set of constraints on the system's motion.

We can think of the constraints as creating forces which alter the free motion of the particles, ie

$$M\ddot{x}(t) = F(x(t), \dot{x}(t), t) + F^c$$

where the constraint force $F^c \in R^{3n\times 1}$ is what we wish to determine. This constraint force must exist such that it both forces the unconstrained motion of the system to obey the constraints and satisfies accepted principles of analytical dynamics (for example, Gauss' principle of least action).

4.2 Constraints

4.2.1 Holonomic Constraints

Any constraint of the form

$$f(x, t) = 0$$

is called a *holonomic* constraint. If the constraint does not depend on time, then it is called *scleronomic*; otherwise, it is called *rheonomic* (which is more general). The scleronomic constraint

$$f(x) = 0$$

essentially constraints this configuration point $x \in R^{3n\times 1}$ to lie on a $(3n-1)$-dimensional surface (manifold) in the full space R^{3n}. A time-varying constraint then constraints the configuration to lie on a surface which deforms over time. Assuming the general rheonomic constraint has partial derivatives, its total differential (which describes how the constraint changes with infinitesimal changes in all its dependent variables) is

$$\sum_{i=1}^{n} \frac{\partial f}{\partial x_i} dx_i + \sum_{i=1}^{n} \frac{\partial f}{\partial y_i} dy_i + \sum_{i=1}^{n} \frac{\partial f}{\partial z_i} dz_i + \frac{\partial f}{\partial t} dt = 0$$

This equation describes the constraints on the infinitesimal displacements dx_i, dy_i, dz_i, dt and will be useful later. This equation is clearly integrable (it integrates to the constraint) and is said to be in *Pfaffian* form.

Alternatively, differentiating the rheonomic constraint with respect to time alone yields

$$\sum_{i=1}^{n} \frac{\partial f}{\partial x_i} \dot{x}_i + \sum_{i=1}^{n} \frac{\partial f}{\partial y_i} \dot{y}_i + \sum_{i=1}^{n} \frac{\partial f}{\partial z_i} \dot{z}_i + \frac{\partial f}{\partial t} = 0$$

Any set of velocities $\{\dot{x}, \dot{y}, \dot{z}\}$ of all particles which satisfy the above equation also satisfy the constraint; for this reason, the set is called a *possible* set of velocities.

If a system of particles is subjected to multiple constraints, the particles are then constrained to lie on the surface defined by the intersection of all constraints. If there exist $3n$ independent constraints, then the system of particles is constrained to a single

configuration point (which moves in time in the case of rheonomic constraints) and thus its motion is independent of the impressed forces!

In general, let there be h holonomic constraints on the system written as

$$f(x,t) = 0 \quad \forall i = 1, 2, \ldots, h$$

These constraints are written in Pfaffian form using the total derivative and chain rule as

$$\sum_{j=1}^{3n} d_{ij}(x,t)dx_j + g_i(x,t)dt \quad \forall i = 1, 2, \ldots, h$$

in terms of infinitesimal displacements where

$$d_{ij}(x,t) = \frac{\partial f_i(x,t)}{\partial x_j}, \quad g_i(x,t) = \frac{\partial f_i(x,t)}{\partial t}$$

Alternatively, they can be written in terms of possible velocities as

$$\sum_{j=1}^{3n} d_{ij}(x,t)\dot{x}_j + g_i(x,t) \quad \forall i = 1, 2, \ldots, h$$

These h constrains reduce the dimensionality of the space of possible system configurations from $3n$ to $(3n-h)$. We will be concerned with whether or not these constraints are *consistent* with one-another rather than independent; fulfillment of any one constraint should not make it impossible to fulfill any other.

4.2.2 Nonholonomic Constraints

Any constraint which cannot be put into the form

$$f(x,t) = 0$$

is defined to be *nonholonomic*. These include, for example, inequality constraints. More precisely, any Pfaffian form

$$\sum_{j=1}^{3n} d_{ij}(x,t)dx_j + g_i(x,t)dt \quad \forall i = 1, 2, \ldots, r$$

which is NOT integrable leads to a nonholonomic constraint. While we can specify constraints on the infinitesimal displacements of the system subject to a nonholonomic constraint in exactly the same way as for holonomic constraints, we cannot find the corresponding restrictions on finite displacements because we cannot integrate their Pfaffian forms.

However, both the possible velocity constraints for the h holonomic and r nonholonomic constraints can be differentiated (with respect to time) to produce a set

of $m = h + r$ constraints on possible accelerations of the system. This is useful since Netwon's law works on the level of accelerations. Assuming the functions $d_{ij}(x,t)$ and $g_i(x,t)$ are sufficiently smooth, we obtain

$$\sum_{j=1}^{3n} d_{ij}(x,t)\ddot{x}_j + \sum_{j=1}^{3n}\left[\sum_{k=1}^{3n}\frac{d_{ij}(x,t)}{\partial x_k}\dot{x}_k\right]\dot{x}_j + \sum_{j=1}^{3n}\frac{d_{ij}(x,t)}{\partial t}\dot{x}_j + \sum_{k=1}^{3n}\frac{\partial g_i(x,t)}{\partial x_k}\dot{x}_k + \frac{\partial g_i(x,t)}{\partial t} = 0 \quad \forall i = 1,2,\ldots,m$$

after multiple applications of the chain rule (remembering that each d_{ij} is a function of both t and x which is itself a function of t). We can write these acceleration constraints in matrix form as

$$A(x,t)\ddot{x} = b(x,\dot{x},t)$$

where $\ddot{x}$ is the vector of stacked acceleration components of all particles in the system. More generally, we will allow the elements of A to depend on velocity as well, resulting in acceleration constraints of the form

$$A(x,\dot{x},t)\ddot{x} = b(x,\dot{x},t)$$

which can be though of as arising from differentiating constraints of the form $\phi(x,\dot{x},t) = 0$ twice.

Once we put our constraints in this form, we don't really care whether they are holonomic or not. If k of these m total constraints are linearly independent, the rank of the matrix A is k and we can arbitrarily prescribe $3n-k$ of the components of $\ddot{x}$ since they lie in the nullspace of A and thus do not affect satisfaction of the constraints.

We call $d = 3n - k$ the number of *degrees of freedom* of the system. Of course, we assume all constraints are consistent.

4.3 Gauss' Principle

The problem of the unconstrained motion of a system of particles is described by the equation

$$Ma = F(x(t),\dot{x}(t),t)$$

where M is the diagonal matrix of particle masses (in sets of threes), a is the vector of stacked unconstrained accelerations and F is the vector of stacked impressed forces, assumed to be known given the state of the system.

However, the motion of the particles changes when subject to constraints; we denote the constrained acceleration of the system by the vector $\ddot{x}(t)$. There are many *possible* accelerations at any time t which satisfy the acceleration-level constraints; Gauss'

principle states that among all possible accelerations, the true constrained accelerations minimize the form

$$G(\ddot{x}) = (\ddot{x} - a)^T M(\ddot{x} - a) = (M^{1/2}\ddot{x} - M^{1/2}a)^T(M^{1/2}\ddot{x} - M^{1/2}a)$$

where the scalar $G(\ddot{x})$ is called the *Gaussian*. In other words, among all possible constrained accelerations, the ones which materialize are those which are closest to the unconstrained accelerations in terms of the $L2$ norm defined by the matrix M.

4.4 The Fundamental Equation

The constrained acceleration which minimizes the Gaussian is given by

$$\ddot{x} = a + M^{-1/2}(AM^{-1/2})^{\dagger}(b - Aa)$$

where $(AM^{-1/2})^{\dagger}$ is the unique MP-inverse of the *Constraint Matrix* matrix $AM^{-1/2}$. We can verify this be proving that $\ddot{x}$ both satisfies the constraints and results in the minimum Gaussian among all constraint-consistent accelerations. Alternatively, this can be obtained directly by minimizing the Gaussian with respect to the constraints since this is simply a least-squares problem with linear equality constraints (and is thus solvable via the method of Lagrange multipliers, for example).

From the fact that $Ma = F(t)$ where F are the impressed forces acting on the system, we can multiply the fundamental equation given above on the left by M and write the result as

$$M\ddot{x} = Ma + M^{1/2}(AM^{-1/2})^{\dagger}(b - Aa) = F(t) + F^c(t)$$

where $F^c(t)$ denotes the additional so-called *constraint force* acting on the system.

Chapter 5

Systems and Control Theory

5.1 System Representations

The objective of *control theory* is to manipulate the input to a system such that the outputs behave according to some specifications. We thus focus on the input-output behavior of systems.

We define the *state* $x(t)$ of a system to be the minimum set of parameters that, if provided at some initial time $t = t_0$ together with the input $u(t)$ for $t \geq t_0$, allow for unique determination of the output $y(t)$. The definition of the system state is not unique, nor does it have to be finite.

The state representation of a general nonlinear, time-varying system with n states, m inputs and k outputs may be written as

$$\begin{aligned} \dot{x}_i(t) &= f_i(t, x_1(t), \ldots, x_n(t), u_1(t), \ldots, u_m(t)), \quad i = 1, \ldots, n \\ y_i(t) &= g(t, x_1(t), \ldots, x_n(t), u_1(t), \ldots, u_n(t)), \quad i = 1, \ldots, k \end{aligned}$$

or, more compactly, as

$$\dot{x}(t) = f(t, x(t), u(t)), \quad y(t) = g(t, x(t), u(t))$$

where $x = [x_1, \ldots, x_n]^T \in R^n$, $u = [u_1, \ldots, u_m]^T \in R^m$, $y = [y_1, \ldots, y_k]^T \in R^k$ and

$$f(t, x(t), u(t)) = \begin{bmatrix} f_1(t, x(t), u(t)) \\ \vdots \\ f_n(t, x(t), u(t)) \end{bmatrix} \quad g(t, x(t), u(t)) = \begin{bmatrix} g_1(t, x(t), u(t)) \\ \vdots \\ g_n(t, x(t), u(t)) \end{bmatrix}$$

If the system is linear then it has the form

$$\begin{aligned}
\dot{x}_1(t) &= a_{11}x_1 + \cdots + a_{1n}x_n + b_{11}u_1 + \cdots + b_{1m}u_m \\
&\vdots \\
\dot{x}_n(t) &= a_{n1}x_1 + \cdots + a_{nn}x_n + b_{n1}u_1 + \cdots + b_{nm}u_m \\
\\
\dot{y}_1(t) &= c_{11}x_1 + \cdots + c_{1n}x_n + d_{11}u_1 + \cdots + d_{1m}u_m \\
&\vdots \\
\dot{y}_k(t) &= c_{k1}x_1 + \cdots + c_{kn}x_n + d_{k1}u_1 + \cdots + d_{km}u_m
\end{aligned}$$

where the coefficients may be time varying (for an LTV system). We can write the above system compactly in *state-space form* as

$$\dot{x}(t) = A(t)x(t) + B(t)u(t), \quad y(t) = C(t)x(t) + D(t)u(t)$$

where

$$A(t) = \begin{bmatrix} a_{11}(t) & \cdots & a_{1n}(t) \\ \vdots & \ddots & \vdots \\ a_{n1}(t) & \cdots & a_{nn}(t) \end{bmatrix}, \quad B(t) = \begin{bmatrix} b_{11}(t) & \cdots & b_{1m}(t) \\ \vdots & \ddots & \vdots \\ b_{n1}(t) & \cdots & a_{nm}(t) \end{bmatrix}$$

$$C(t) = \begin{bmatrix} c_{11}(t) & \cdots & c_{1n}(t) \\ \vdots & \ddots & \vdots \\ c_{k1}(t) & \cdots & c_{kn}(t) \end{bmatrix}, \quad D(t) = \begin{bmatrix} d_{11}(t) & \cdots & a_{1m}(t) \\ \vdots & \ddots & \vdots \\ d_{k1}(t) & \cdots & a_{km}(t) \end{bmatrix}$$

where $A(t) \in R^{n\times n}$ is the *dynamics matrix*, $B(t) \in R^{n\times m}$ is the *input matrix*, $C(t) \in R^{k\times n}$ is the *output matrix* and $D(t) \in R^{k\times m}$ is the *(direct) feedthrough matrix.*

5.1.1 Properties of Linear Systems:

By definition, a linear dynamic system is one in which, for any $\alpha, \beta \in R$, we have

$$G(\alpha u_1(t) + \beta u_2(t)) = \alpha G(u_1(t)) + \beta G(u_2(t)) = \alpha y_1(t) + \beta y_2(t)$$

where G is an operator representing the system.

For linear systems, the condition for existence and uniqueness of a solution for some input $u(t)$ is that the elements of the dynamics matrix $a_{ij}(t)$ are piecewise continuous functions of time. LTI systems always have unique solutions.

5.1.2 Linearization of State Equations

The state $x_e \in R^n$ is an *equilibrium* state of the general dynamic system

$$\dot{x}(t) = f(t, x(t), u(t))$$
$$y(t) = g(t, x(t), u(t))$$

at time $t = t_0$ for a given constant input $u(t) = u_e$ if $f(t, x_e, u_e) = 0$. Note that for a time-invariant system, equilibrium at $t = t_0$ implies equilibrium for any time $t \geq t_0$.

We may linearize a system about an equilibrium point by considering the system dynamics in response to a small perturbation about that point. The result is that a nonlinear, time-invariant system can be linearized about a point $\{x_e, u_e\}$ as

$$\dot{x}(t) = Ax(t) + Bu(t), \quad y(t) = Cx(t) + Du(t)$$

where

$$A = \frac{\partial f}{\partial x}, \quad B = \frac{\partial f}{\partial u}$$
$$C = \frac{\partial g}{\partial x}, \quad B = \frac{\partial g}{\partial u}$$

5.1.3 Transfer Functions of a State-Space Realization

Given the LTI system

$$\dot{x}(t) = Ax(t) + Bu(t), \quad y(t) = Cx(t) + Du(t)$$

we take the Laplace transform of both sides to yield

$$sx(s) - x(0) = Ax(s) + Bu(s), \quad y(s) = Cx(s) + Du(s)$$

Solving the first equation for $x(s)$ in terms of $u(s)$ and $x(0)$ and substituting the result into the second equation, we find that

$$y(s) = C(sI - A)^{-1}x(0) + G(s)u(s)$$

where

$$G(s) = C(sI - A)^{-1}B + D$$

is the $k \times m$ *transfer function matrix* composed of km scalar transfer functions; the entry G_{ij} is the SISO transfer function relating input j to output i. The first term in the expression for $y(s)$ is the homogeneous solution corresponding to initial condition $x(0)$.

Note that while the state-space realization $\{A, B, C, D\}$ is not unique, the transfer function $G(s)$ is unique.

Given a transfer function $G(s)$, we say that the state realization $S : \{A, B, C, D\}$ is a state space realization of $G(s)$ if

$$G(s) = C(sI - A)^{-1}B + D$$

From the formula for the matrix inverse we know that

$$G(s) = \frac{1}{|sI - A|}C(\text{adj}(sI - A)^T)B + D$$

Since adj$A$ contains contains continuous determinants of $(n-1)\times(n-1)$ submatrices of A, then the elements of $G(s)$ must be *proper* rational functions, ie they are ratios of polynomials of equal orders. If $D = 0$ then the elements of $G(s)$ are ratios of polynomials of order $(n-1)$ and polynomials of order n; these are *strictly proper* rational functions.

5.1.4 Realization of SISO Transfer Functions

Controllable Canonical Form:

Given the SISO transfer function

$$g(s) = \frac{y(s)}{u(s)} = \frac{b_1 s^{n-1} + b_2 s^{n-2} + \cdots + b_n}{s^n + a_1 s^{n-1} + a_2 s^{n-2} + \cdots + a_n}$$

we seek to find a state-space realization of $g(s)$ (recall that such a realization is not unique).

We define

$$\frac{\phi(s)}{u(s)} = \frac{1}{s^n + a_1 s^{n-1} + a_2 s^{n-2} + \cdots + a_n} = \frac{1}{a(s)}$$

so that

$$\phi^{(n)} + a_1\phi^{(n-1)} + a_2\phi^{(n-2)} + \cdots + a_n\phi = u$$

Now, we define our states to be $x_1 = \phi$, $x_2 = \phi^{(2)}$, and so on until $x_{n-1} = \phi^{(n-1)}$. Using these states, the dynamics of our system are

$$\begin{aligned}
\dot{x}_1 &= x_2 \\
\dot{x}_2 &= x_3 \\
\dot{x}_3 &= x_4 \\
&\vdots \\
\dot{x}_{n-1} &= -a_1 x_{n-1} - a_2 x_{n-2} - \cdots - a_n x_1 + u
\end{aligned}$$

and the output equation becomes

$$y(s) = b_1 s^{n-1}\frac{u(s)}{a(s)} + b_2 s^{n-2}\frac{u(s)}{a(s)} + \cdots + b_n\frac{u(s)}{a(s)}$$

Recall however that $\phi(s) = \frac{u(s)}{a(s)}$; thus,

$$y(s) = b_n x_1 + b_{n-1}x_2 + \cdots b_1 x_n$$

The state-space realization of the system is then

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\\ \vdots\\ \dot{x}_n\end{bmatrix} = \begin{bmatrix}0 & 1 & \cdots & 0\\ \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & \cdots & 1\\ -a_n & \cdots & -a_2 & -a_1\end{bmatrix} + \begin{bmatrix}0\\0\\\vdots\\1\end{bmatrix} u$$

$$y = \begin{bmatrix}b_n & b_{n-1} & \cdots & b_1\end{bmatrix}\begin{bmatrix}x_1\\x_2\\\vdots\\x_n\end{bmatrix}$$

Observable Canonical Form:

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\\ \vdots\\ \dot{x}_n\end{bmatrix} = \begin{bmatrix}0 & \cdots & 0 & -a_n\\ 1 & \cdots & 0 & -a_{n-1}\\ \vdots & \ddots & \vdots & \vdots\\ 0 & \cdots & 1 & -a_1\end{bmatrix} + \begin{bmatrix}b_n\\b_{n-1}\\\vdots\\b_1\end{bmatrix} u$$

$$y = \begin{bmatrix}0 & 0 & \cdots & 1\end{bmatrix}\begin{bmatrix}x_1\\x_2\\\vdots\\x_n\end{bmatrix}$$

Note that $A_o = A_c^T$, $B_o = C_c^T$, $C_o = B_c^T$ and $D_o = D_c$. It is thus said that the controllable and observable canonical forms are *dual*.

Systems with a direct feedthrough term:

Assume that the order of the numerator and denominator are equal (the transfer function is a *proper* rational function).

$$g(s) = \frac{y(s)}{u(s)} = \frac{b_0 s^n + b_1 s^{n-1} + b_2 s^{n-2} + \cdots + b_n}{s^n + a_1 s^{n-1} + a_2 s^{n-2} + \cdots + a_n}$$

Then $g(s)$ can be rewritten as

$$\begin{aligned} g(s) &= \frac{b_0 s^n + b_1 s^{n-1} + b_2 s^{n-2} + \cdots + b_n}{s^n + a_1 s^{n-1} + a_2 s^{n-2} + \cdots + a_n} \\ &= \frac{b_0[s^n + a_1 s^{n-1} + a_2 s^{n-2} + \cdots + a_n] + [(b_1 - b_0 a_1)s^{n-1} + (b_2 - b_0 a_2)s^{n-2} + \cdots + (b_n - b_0 a_n)]}{s^n + a_1 s^{n-1} + a_2 s^{n-2} + \cdots + a_n} \\ &= b_0 + \frac{(b_1 - b_0 a_1)s^{n-1} + (b_2 - b_0 a_2)s^{n-2} + \cdots + (b_n - b_0 a_n)}{s^n + a_1^{n-1} + a_2 s^{n-2} + \cdots + a_n} \end{aligned}$$

which makes it clear that b_0 represents the feedthrough term from input to output. Note that we can write this system in the above canonical forms (a direct feedthrough term will be introduced in the output equation).

Parallel Realizations:

Given the strictly proper SISO transfer function

$$g(s) = \frac{y(s)}{u(s)} = \frac{b_1 s^{n-1} + b_2 s^{n-2} + \cdots + b_n}{s^n + a_1 s^{n-1} + a_2 s^{n-2} + \cdots + a_n}$$

Assume that all the roots $\lambda_i, i = 1, \ldots, n$ of the characteristic polynomial are distinct. We can thus factor $g(s)$ as

$$g(s) = \frac{y(s)}{u(s)} = \frac{g_1}{s - \lambda_1} + \frac{g_2}{s - \lambda_2} + \cdots + \frac{g_n}{s - \lambda_n}$$

Now, define the states

$$\begin{aligned} x_1(s) &= \frac{g_1}{s - \lambda_1} u(s) \\ &\vdots \\ x_n(s) &= \frac{g_n}{s - \lambda_n} u(s) \end{aligned}$$

Hence, we have the system described by

$$\begin{aligned} \dot{x}_i &= \lambda_i x_i + g_i u \quad i = 1, \ldots, n \\ y &= x_1 + \cdots + x_n \end{aligned}$$

The SISO system thus decouples into n one-dimensional SISO systems! The corresponding state space realization is thus parameterized by

$$A = \begin{bmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_n \end{bmatrix}, \quad B = \begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ g_n \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix}$$

Note that if some $g_i = 0$, there was a pole-zero cancellation. From inspection of the state-space system, it's clear that the corresponding state x_i thus cannot be affected by input u (since all the states are decoupled). We thus say that x_i is an *uncontrollable* state.

Alternatively, we may define the states to be

$$x_i(s) = \frac{1}{s - \lambda_1} u(s), \quad i = 1, \ldots, n$$

which leads to the realization

$$A = \begin{bmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_n \end{bmatrix}, \quad B = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

$$C = \begin{bmatrix} g_1 & g_2 & \cdots & g_n \end{bmatrix}$$

If some $g_i = 0$ in this case due to a pole-zero cancellation, inspection of the above realization shows that the corresponding state x_i does not show up in the output y and hence cannot be measured! We thus say that x_i is an *unobservable* state. By a simple change of state variables, we have shown that controllability and observability are two sides of the same coin - you can always make an uncontrollable state controllable, but at the cost of observability (and vice-versa).

Finally, define the states as

$$x_i(s) = \frac{b_i}{s - \lambda_1} u(s), \quad i = 1, \ldots, n$$

where $c_i b_i = g_i$. This leads to the realization

$$A = \begin{bmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_n \end{bmatrix}, \quad B = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

$$C = \begin{bmatrix} c_1 & c_2 & \cdots & c_n \end{bmatrix}$$

In this case, if some $g_i = 0$ then either $b_i = 0$ or $c_i = 0$ and the corresponding state x_i is either uncontrollable or unobservable (you can't tell which it is!)

Note that if the transfer function is proper instead of strictly proper, we simply factor it as above and go through the same steps to find the corresponding parallel realization (a direct feedthrough term will appear).

5.1.5 Equivalent Realizations

Given the linear, time-invariant, state-space system

$$\dot{x} = Ax + Bu, \quad y = Cx + Du$$

we seek to find the representation of the system when the state vector is defined in a new basis corresponding to some nonsingular matrix T. The representation $\bar{x}$ of x in the basis T is written $x = T\bar{x}$; solving for $\bar{x}$ yields $\bar{x} = T^{-1}x$. Writing the system in terms of this new state yields

$$T\dot{\bar{x}} = AT\bar{x} + Bu, \quad y = CT\bar{x} + Du$$

Multiplying both sides of the dynamics equation by T^{-1}, we have

$$\dot{\bar{x}} = \bar{A}\bar{x} + \bar{B}u, \quad y = \bar{C}\bar{x} + \bar{D}u$$

where $\bar{A} = T^{-1}AT$, $\bar{B} = T^{-1}B$, $\bar{C} = CT$ and $\bar{D} = D$. These two systems are said to be *algebraically equivalent* and the corresponding state transformation is called a *similarity transformation.*

Note that the transfer functions of algebraically equivalent systems are identical. This is proven below.

$$\begin{aligned}
\bar{G}(s) &= \bar{C}(sI - \bar{A})^{-1}\bar{B} + \bar{D} \\
&= CT(sI - T^{-1}AT)^{-1}T^{-1}B + D \\
&= CT(sI - T^{-1}A^{-1}T)T^{-1}B + D \\
&= CT(sI)T^{-1}B - CT(T^{-1}A^{-1}T)T^{-1}B + D \\
&= C(sI)B - CA^{-1}B + D \\
&= C(sI - A)^{-1}B + D \\
&= G(s)
\end{aligned}$$

5.2 Solutions of Linear Systems

5.2.1 LTV Systems and The Peano-Baker Series

Given the LTV system

$$\dot{x}(t) = A(t)x(t) + B(t)u(t), \quad y(t) = C(t)x(t) + D(t)u(t)$$

we seek to find the solution to the homogeneous system

$$\dot{x}(t) = A(t)x(t), \quad x(t_0) = x_0$$

The unique solution to the above unforced system is given by

$$x(t) = \Phi(t, t_0)x_0$$

where

$$\Phi(t,t_0) = I + \int_{t_0}^{t} A(s_1)ds_1 + \int_{t_0}^{t} A(s_1)\left[\int_{t_0}^{s_1} A(s_2)ds_2\right]ds_1 + \int_{t_0}^{t} A(s_1)\left[\int_{t_0}^{s_1} A(s_2)\left[\int_{t_0}^{s_2} A(s_3)ds_3\right]ds_2\right]ds_1 + \cdots$$

is the *state transition matrix* of the system.

Of course, one never actual calculates the solution in this form as it's intractable for all but LTI systems. To prove that this solution is consistent with the ODE, we must show that it satisfies the initial conditions and the ODE itself.

First, note that $\Phi(t_0, t_0) = I$ and $x(t_0) = \Phi(t_0, t_0)x_0 = x_0$. Thus, the initial conditions are satisfied.

Next, note that the derivative of the state transition matrix is

$$\begin{aligned}
\dot{\Phi}(t, t_0) &= A(t) + A(t)\int_{t_0}^{s_1} A(s_2)ds_2 + A(t)\int_{t_0}^{s_1} A(s_2)\int_{t_0}^{s_2} A(s_3)ds_3ds_2 + \cdots \\
&= A(t)\Phi(t, t_0)
\end{aligned}$$

and hence $x(t) = \dot{\Phi}(t, t_0)x_0$ satisfies the homogeneous system $\dot{x} = A(t)x$.

5.2.2 Properties of the State Transition Matrix

It was shown that the solution to the unforced (homogeneous) system $\dot{x}(t) = A(t)x(t)$ is $x(t) = \Phi(t, t_0)x_0$ where the state transition matrix $\Phi(t, t_0)$ has the properties (from inspection of the Peano-Baker series)

$$\begin{aligned}\dot{\Phi}(t, t_0) &= A(t)\Phi(t, t_0)\\ \Phi(t_0, t_0) &= I\end{aligned}$$

The state transition matrix additionally has the following properties:

(i) $\Phi(t_2, t_0) = \Phi(t_2, t_1)\Phi(t_1, t_0)$ (this is known as the *semi-group property.* Note that it is not necessary that $t_1 < t_2$.)

(ii) $\Phi(t, t_0)$ is nonsingular for all $t \geq t_0$ and $\Phi^{-1}(t, t_0) = \Phi(t_0, t)$.

(iii) $\dot{\Phi}(t_0, t) = -\Phi(t_0, t)A(t)$ and also $\Phi(t_0, t)^T = -A^T(t)\Phi^T(t_0, t)$.

Proof. (i) We know that $x(t_1) = \Phi(t_1, t_0)x(t_0)$ and $x(t_2) = \Phi(t_2, t_0)x(t_0)$. We also know that $x(t_2) = \Phi(t_2, t_1)x(t_1)$. Thus,

$$\begin{aligned}x(t_2) &= \Phi(t_2, t_0)x(t_0)\\ &= \Phi(t_2, t_1)x(t_1)\\ &= \Phi(t_2, t_1)\Phi(t_1, t_0)x(t_0)\end{aligned}$$

and thus $\Phi(t_2, t_0) = \Phi(t_2, t_1)\Phi(t_1, t_0)$.

(ii) We know from the previous property that $\Phi(t, t) = \Phi(t, t_0)\Phi(t_0, t)$. But from the series form of $\Phi(t, t_0)$ we also know that $\Phi(t, t) = I$. Thus, $\text{rank}(\Phi(t, t_0)) = \text{rank}(\Phi(t_0, t)) = n$ and the inverse of $\Phi(t, t_0)$ always exists and is equal to $\Phi(t_0, t)$.

(iii) We know that $I = \Phi(t, t) = \Phi(t, t_0)\Phi(t_0, t)$ from the first property. Differentiating on both sides yields

$$\Phi(t, t_0)\dot{\Phi}(t_0, t) + \dot{\Phi}(t, t_0)\Phi(t_0, t) = 0$$

However, we already know that $\dot{\Phi}(t, t_0) = A(t)\Phi(t, t_0)$ and thus

$$\begin{aligned}\Phi(t, t_0)\dot{\Phi}(t_0, t) + A(t)\Phi(t, t_0)\Phi(t_0, t) &= 0\\ \Phi(t, t_0)\dot{\Phi}(t_0, t) + A(t) &= 0\end{aligned}$$

Multiplying both sides by $\Phi^{-1}(t, t_0) = \Phi(t_0, t)$ and solving for $\dot{\Phi}(t_0, t)$ yields

$$\dot{\Phi}(t_0, t) = -\Phi(t_0, t)A(t)$$

as desired. □

5.2.3 Solution of the Forced System

It is a property of linear systems that the solution in response to an input can be written as the sum of the unforced (homogeneous) and forces responses.

We now seek to solve the linear, first-order system of ODEs

$$\dot{x}(t) = A(t)x(t) + B(t)u(t)$$

The solution will be shown to be

$$x(t) = \Phi(t, t_0)x(t_0) + \int_{t_0}^{t} \Phi(t, \tau)B(\tau)u(\tau)d\tau$$

Proof. We assume a solution to the total system in the form of the unforced solution but with unknown parameters, ie

$$x(t) = \Phi(t, t_0)k(t)$$

where $k(t) \in R^n$ is to be determined. This method of solution is known as *variation of parameters*.

Differentiating, we have that

$$\dot{x}(t) = \Phi(t, t_0)\dot{k}(t) + \dot{\Phi}(t, t_0)k(t)$$

Substituting this into the forced system, we have

$$\Phi(t, t_0)\dot{k}(t) + \dot{\Phi}(t, t_0)k(t) = A(t)\Phi(t, t_0)k(t) + B(t)u(t)$$

But we know that $\dot{\Phi}(t, t_0) = A(t)\Phi(t, t_0)$. Thus,

$$\Phi(t, t_0)\dot{k}(t)+A(t)\Phi(t, t_0)k(t) = A(t)\Phi(t, t_0)k(t)+B(t)u(t) \quad \rightarrow \quad \Phi(t, t_0)\dot{k}(t) = B(t)u(t)$$

Solving for $\dot{k}(t)$ and integrating from t_0 to t, we have

$$k(t) = k(t_0) + \int_{t_0}^{t} \Phi(\tau, t_0)^{-1}B(\tau)u(\tau)d\tau$$

where τ is just a dummy variable for integration. To determine the integration constant $k(t_0)$, recall that $x(t_0) = x_0$. Since $x(t_0) = \Phi(t_0, t_0)k(t_0) = x_0$, we have $k(t_0) = x_0$.

The solution to the system is then

$$x(t) = \Phi(t, t_0)x_0 + \Phi(t, t_0) \int_{t_0}^{t} \Phi(\tau, t_0)^{-1}B(\tau)u(\tau)d\tau$$

Since $\Phi(t, t_0)$ is constant with respect to the integral, it can be pulled inside. Additionally, recall that $\Phi^{-1}(\tau, t_0) = \Phi(t_0, \tau)$ and thus we have $\Phi(t, t_0)\Phi(t_0, \tau) = \Phi(t, \tau)$ by the semigroup property.

The solution then becomes

$$x(t) = \Phi(t, t_0)x_0 + \int_{t_0}^{t} \Phi(t, \tau)B(\tau)u(\tau)d\tau$$

as desired. The first term is the homogeneous (unforced) solution and the second is the forced solution. Note that in order to determine how a system responds to any input, one only needs to know the state transition matrix (solution to the unforced system).

□

5.3 Solution of LTI Systems

Recall that the state transition matrix is defined in general by the infinite Peano-Baker series. Computation of this matrix is intractable; for LTI systems, however, the state transition matrix can be readily computed.

Consider the unforced LTI system

$$\dot{x}(t) = Ax(t), \quad x(t_0) = x_0$$

We seek to find the solution $x(t)$ independently of the definition of the state transition matrix (which defines the general solution for linear systems). Assume a solution in the form of an infinite series as shown below (where $\tilde{t} = t - t_0$).

$$x(\tilde{t}) = \sum_{i=0}^{\infty} a_i \tilde{t}^i$$

Substituting into the unforced system, we have

$$\sum_{i=0}^{\infty} i a_i \tilde{t}^{(i-1)} = A \sum_{i=0}^{\infty} a_i \tilde{t}^i$$

Since we seek to compute the coefficients a_i in the assumed solution, we set the coefficients of equal powers of $\tilde{t}$ equal, yielding

$$a_i = \frac{1}{i!} A^i a_0, \quad i = 1, \ldots, n$$

But $x(t = t_0) = x(\tilde{t} = 0) = x_0 = a_0$ so

$$x(t) = \sum_{i=0}^{\infty} \frac{A^i (t - t_0)^i}{i!} x_0 = e^{A(t-t_0)} x_0$$

Note that the matrix exponential satisfies all the properties of the state transition matrix.

Arriving at the above form of solution using Laplace transforms is much simpler. Consider again the homogeneous system

$$\dot{x} = Ax, \quad x(0) = x_0$$

Taking the Laplace transform yields $sx(s) - x_0 = Ax(s)$. Solving for $x(s)$ yields $x(s) = (sI - A)^{-1}x_0$ and the inverse Laplace transform of $(sI - A)^{-1}$ is known to be e^{At}. Thus, $x(t) = e^{At}x_0$ (and since the system is LTI, we can shift time to account for $t_0 \neq 0$).

5.3.1 The Matrix Exponential

The infinite series definition of the exponential e^A of a matrix A is not the most convenient for computations. We can obtain a closed-form expression for the matrix exponential by writing A using its eigendecomposition as $A = S\Lambda S^{-1}$ where S is the *modal* matrix of eigenvectors of A and Λ is the diagonal matrix of eigenvalues of A.

From linear algebra, we know that a matrix A can be diagonalized if and only if it has n linearly independent eigenvectors. Also, we know that a matrix has n linearly independent eigenvectors if and only if it has n distinct eigenvalues. Thus, for a matrix to be diagonalizable (or *similar* to a diagonal matrix Λ) then it must have n distinct eigenvalues.

Using $A = S\Lambda S^{-1}$ to compute e^A, we have

$$\begin{aligned}
e^A &= I + A + \frac{A^2}{2!} + \cdots \\
&= I + S\Lambda S^{-1} + \frac{1}{2!}(S\Lambda S^{-1})(S\Lambda S^{-1}) + \cdots \\
&= SIS^{-1} + S\Lambda S^{-1} + \frac{S\Lambda^2 S^{-1}}{2!} + \cdots \\
&= S\left[I + \Lambda + \frac{\Lambda^2}{2} + \cdots\right]S^{-1} \\
&= Se^{\Lambda}S^{-1}
\end{aligned}$$

where e^{Λ} is simply $\text{diag}\{e^{\lambda_1}, \ldots, e^{\lambda_n}\}$.

An alternative way to arrive at the same answer is to consider the state transformation $x = Sz$ where S is again the matrix of eigenvectors of A. Recall that our system is

$$\dot{x} = Ax, \quad x(t_0) = x_0$$

We thus have the transformed system

$$\dot{z} = S^{-1}ASz = \Lambda z, \quad z(t_0) = S^{-1}x_0$$

Since the system written in this basis has a diagonal dynamics matrix, the solution of each $z_i(t)$ is decoupled from the rest. Rewriting the system, we have

$$\begin{aligned}
\dot{z}_1 &= \lambda_1 z_1 \quad &\rightarrow z_1(t) &= e^{\lambda_1(t-t_0)} z_1(t_0) \\
&\vdots \\
\dot{z}_n &= \lambda_1 z_n \quad &\rightarrow z_n(t) &= e^{\lambda_n(t-t_0)} z_n(t_0)
\end{aligned}$$

Written compactly, we have

$$z(t) = e^{\Lambda(t-t_0)} z(t_0)$$

Transforming the state back yields

$$x(t) = S e^{\Lambda(t-t_0)} S^{-1} x_0 = e^{A(t-t_0)} x_0$$

which is the same solution as obtained previously.

Regardless of which way it's computed, the matrix exponential arises because the system can be decoupled into n first order systems by a suitable change of basis. The question then is: what can be done in the case in which A cannot be diagonalized?

Jordan Canonical Form

If A cannot be diagonalized, then there must be eigenvalues with multiplicities greater than one (and hence linearly dependent eigenvectors). In the case that A has k distinct eigenvalues, we may write

$$J = S^{-1} A S$$

where $J \in M_n$ is a "nearly-diagonal" matrix called the *Jordan canonical form*. This matrix has the form

$$J = \begin{bmatrix} J_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & J_k \end{bmatrix}$$

where each $J_i \in M_{m_i}$ is a *Jordan block* corresponding to eigenvalue λ_i (which has multiplicity m_i) of the form

$$J_i = \begin{bmatrix} \lambda_i & 1 & 0 & 0 \\ 0 & \lambda_i & \ddots & 0 \\ 0 & \ddots & \ddots & 1 \\ 0 & 0 & 0 & \lambda_i \end{bmatrix}$$

Note that $\sum_i m_i = n$. To find the columns of the matrix S, we write $AS = SJ$ out as

$$A[S_1, S_2, \cdots, S_k] = [S_1, S_2, \cdots, S_k] \begin{bmatrix} J_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & J_k \end{bmatrix}$$

where each $S_i \in C^{n \times m_i}$ is a submatrix of S. Thus, we may write

$$AS_i = S_i J_i, \quad i = 1, \ldots, k$$

Consider now a particular Jordan block J_i and write

$$A[s_{i1}, s_{i2}, \cdots, s_{im_i}] = [s_{i1}, s_{i2}, \cdots, s_{im_i}] \begin{bmatrix} \lambda_i & 1 & 0 & 0 \\ 0 & \lambda_i & \ddots & 0 \\ 0 & \ddots & \ddots & 1 \\ 0 & 0 & 0 & \lambda_i \end{bmatrix}$$

Multiplying this out, we have

$$\begin{aligned} As_{i1} &= \lambda_i s_{i1} \quad \rightarrow (A - \lambda_i I)s_{i1} = 0 \\ As_{i2} &= s_{i1} + \lambda_i s_{i2} \quad \rightarrow (A - \lambda_i)s_{i2} = s_{i1} \\ &\vdots \\ As_{im_i} &= s_{i(m_i-1)} + \lambda_i s_{im_i} \quad \rightarrow (A - \lambda_i)s_{im_i} = s_{i(m_i-1)} \end{aligned}$$

The vectors s_{ij} are called the *generalized eigenvectors* of A. These vectors are thus solved for recursively starting with s_{i1}.

Now, consider application of the Jordan normal form to solving a homogeneous linear system whose dynamics matrix has repeated eigenvalues. We first find the transition matrix corresponding to a single Jordan block by making the state transformation $x = Sz$. Thus, we have $\dot{z} = S^{-1}ASz = Jz$ and $z(t_0) = S^{-1}x(t_0)$ or

$$\begin{bmatrix} \dot{z}_1 \\ \dot{z}_2 \\ \vdots \\ \dot{z}_n \end{bmatrix} = \begin{bmatrix} \lambda & 1 & 0 & 0 \\ 0 & \lambda & \ddots & 0 \\ 0 & \ddots & \ddots & 1 \\ 0 & 0 & 0 & \lambda \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix}$$

We may rewrite the above system as

$$\begin{aligned} \dot{z}_1 &= \lambda z_1 + z_2 \\ \dot{z}_2 &= \lambda z_2 + z_3 \\ &\vdots \\ \dot{z}_n &= \lambda z_n \end{aligned}$$

Thus, we solve these first-order systems starting from the last equation and back-substitute into the rest as follows.

The solution to the last equation is

$$z_n(t) = e^{\lambda(t-t_0)} z_n(t_0)$$

Using this, the solution to equation $n-1$ is then

$$\dot{z}_{n-1} = \lambda z_{n-1} + e^{\lambda(t-t_0)} z_n(t_0) \quad \rightarrow z_{n-1}(t) = e^{\lambda(t-t_0)} z_{n-1}(t_0) + (t-t_0) e^{\lambda(t-t_0)} z_n(t_0)$$

Continuing this process results in the general solution

$$z_{n-j}(t) = e^{\lambda(t-t_0)} z_{n-j}(t_0) + (t-t_0) e^{\lambda(t-t_0)} z_{n-j-1}(t_0) + \cdots + \frac{(t-t_0)^{n-1}}{(n-1)!} e^{\lambda(t-t_0)} z_n(t_0)$$

for all $j = 0, 1, \cdots, n$. Thus, we can write the solution to the whole system as

$$z(t) = e^{J(t-t_0)} z(t_0) = \begin{bmatrix} 1 & (t-t_0) & \frac{(t-t_0)^2}{2!} & \cdots & \frac{(t-t_0)^{n-2}}{(n-2)!} & \frac{(t-t_0)^{n-1}}{(n-1)!} \\ 0 & 1 & (t-t_0) & \ddots & \ddots & \frac{(t-t_0)^{n-2}}{(n-2)!} \\ 0 & 0 & 1 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \frac{(t-t_0)^2}{2!} \\ 0 & \cdots & 0 & 0 & 1 & (t-t_0) \\ 0 & \cdots & 0 & 0 & 0 & 1 \end{bmatrix} e^{\lambda(t-t_0)} z(t_0)$$

This is the solution for a single Jordan block - how do we solve the whole system? Recall from linear algebra that if the function of a matrix $f(A)$ is represented by a convergent series in A, and if $A = A_1 \oplus \cdots \oplus A_k$ is block-diagonal, then $f(A) = f(A_1) \oplus \cdots \oplus f(A_k)$.

Thus, if $J = J_1 \oplus \cdots \oplus J_k$, then $e^{J(t-t_0)} = e^{J_1(t-t_0)} \oplus \cdots \oplus e^{J_k(t-t_0)}$ where each $e^{J_i(t-t_0)}$ is computed as detailed above.

5.3.2 Discretization of LTI Systems

Given a continuous LTI system in state space form

$$\dot{x} = Ax + Bu, \quad y = Cx + D$$

we may find the corresponding discrete-time system as follows. Assuming a zero-order hold such that the input $u(t)$ remains constant over a time interval T, the solution to the continuous system over the sampling interval kT to $(k+1)T$ may be written as

$$x((k+1)T) = e^{A((k+1)T-kT)} x(kT) + \int_{kT}^{(k+1)T} e^{A((k+1)T-\lambda)} Bu(kT) d\lambda$$

Defining $\tau = (k+1)T - \lambda$, we change variables in the integral using $d\tau = -d\lambda$; the limits of integration must thus run from $(k+1)T - kT = T$ to $(k+1)T - (k+1)T = 0$.

$$x((k+1)T) = e^{AT}x(kT) - u(kT)\int_T^0 e^{A\tau}Bd\tau$$

Switching the limits of integration and writing $x(kT) = x_k$ and $u(kT) = u_k$ yields the equivalent discrete-time system

$$x_{k+1} = Fx_k + Gu_k$$

where

$$F = e^{AT}, \quad G = \int_0^T e^{A\tau}Bd\tau$$

5.4 Stability

5.4.1 Internal Stability of Linear Systems

Given the unforced LTV state space system

$$\dot{x}(t) = A(t)x(t), \quad x(t_0) = x_0$$

it's clear that the equilibrium point is at $x = 0$.
For further discussion on stability, we must first recall

Vector Norms

The *p-norm* of a vector $x \in R^n$ is defined to be

$$||x||_p = (\sum_{i=1}^{n} |x_i|^p)^{1/p}$$

For $p = 2$, this is the familiar Euclidean norm $||x||_2 = \sqrt{x^T x}$. We may visualize these norms by considering the shapes of regions of constant norm. For $p = 1$, this region is a diamond; for $p = 2$ it is a circle. For $p > 2$ the shape approaches a square as $p \to \infty$. Thus, the $p = \infty$ norm is defined to be

$$||x||_\infty = \max_i |x_i|$$

Matrix Norms

How can we measure the "size" of a matrix? Since matrices represent linear transformations which act on vectors, we can define the norm of a matrix by its action on a vector. For any $A \in C^{m\times n}$ we thus define the *induced p-norm* to be

$$||A||_p = \max_{x \neq 0} \frac{||Ax||_p}{||x||_p} = \max_{||x||_p=1} ||Ax_p||$$

where the two definitions above are equivalent. It can be shown that:
For $p = 1$, the norm is equal to the maximum of the column sums, ie

$$||A||_1 = \max_{1 \leq j \leq n} \sum_{i=1}^{m} |a_{ij}|$$

For $p = 2$, the norm is equal to the maximum singular value of A (maximum eigenvalue of $A^T A$), ie

$$||A||_2 = \lambda_{max}(A^T A)$$

For $p = \infty$, the norm is equal to the maximum of the row sums, ie

$$||A||_1 = \max_{1 \leq i \leq m} \sum_{i=1}^{n} |a_{ij}|$$

5.4.2 Types of Stability

Note that in all of the following definitions, stability is not dependent on the norm chosen!

Uniform Stability

An LTV system is *uniformly stable* if there exists a finite, positive constant γ such that for any x_0, t_0 the corresponding solution satisfies

$$||x(t)|| \leq \gamma ||x_0||$$

Theorem: The LTV system is uniformly stable if and only if there exists a finite, positive constant γ such that

$$||\Phi(t, \tau)|| < \gamma$$

for $t \geq \tau$. The constant γ determines the upper bound on the norm of the state.

Proof. $\leftarrow$ Assume that such a constant exists. Then the solution $x(t) = \Phi(t, t_0)x_0$ implies

$$||x(t)|| = ||\Phi(t, t_0)x_0|| \leq ||\Phi(t, t_0)||||x_0|| \leq \gamma ||x_0||$$

by Cauchy-Schwarz and thus the system is uniformly stable by the definition above.
$\rightarrow$ Now, assume that the system is uniformly stable and show that such a constant exists such that the norm of the state transition matrix is bounded (not proven in class). $\square$

Uniform Exponential Stability

The LTV system is *uniformly exponentially stable* if and only if there exist finite, positive constants γ, λ such that for all x_0, t_0 the solution satisfies

$$||x(t)|| \leq \gamma e^{-\lambda(t-t_0)}||x_0||$$

Note that exponential stability implies that $x(t) \to 0$ as $t \to \infty$ while uniform stability does not. The constant λ determines the rate of convergence of the norm of the state.

Theorem: The LTV system is uniformly exponentially stable if and only if there exist γ, λ such that

$$||\Phi(t, \tau)|| \leq \gamma e^{-\lambda(t-\tau)}$$

for all $t \geq \tau$. The proof (in the easy direction) is similar to that for uniform stability.

Exponential Stability of LTI Systems

Theorem: The LTI system $\dot{x} = Ax, x(t_0) = x_0$ is uniformly exponentially stable if and only if all the eigenvalues of A have negative real parts.

Proof. $\leftarrow$ Recall that the solution to the above system is $x(t) = e^{A(t-t0)}x_0$. If A is diagonalizable then the theorem is clear from the fact that $e^{A(t-t_0)} = Se^{\Lambda(t-t_0)}S^{-1}$ where Λ is diagonal.

We know that for a general LTI system

$$\Phi(t, t_0) = e^{A(t-t_0)} = Se^{J(t-t_0)}S^{-1}$$

since A is similar via S to the "near-diagonal" Jordan canonical form $J = J_1 \oplus \cdots \oplus J_n$. Also note that $e^J = e^{J_1} \oplus \cdots \oplus e^{J_n}$ (see notes from the previous lecture).

We thus have

$$||\Phi(t, t_0)|| \leq ||S||||e^{J(t-t_0)}||||S^{-1}|| \leq \gamma_1||e^{J(t-t_0)}||\gamma_2$$

Now, how to bound the norm of the exponential of the Jordan form? Recall that for all i, the matrix $e^{J_i} \in C^{n \times n}$ has the form $Te^{\lambda_i(t-t_0)}$ where $T \in R^{\times n}$ is a matrix with entries which are polynomials in $(t - t_0)$ of order at most $n - 1$ where n is the algebraic multiplicity of eigenvalue λ_i.

If $\mathrm{Re}(\lambda_i) < 0$ then since exponential decay dominates polynomial growth for large values of $t - t0$, we have that each $e^{J_i(t-t_0)}$ is bounded by γ_3 such that

$$||e^{J_i(t-t_0)}|| \leq \gamma_3 e^{\lambda_i(t-t_0)}$$

Since this holds for all the Jordan blocks composing J, we have

$$||\Phi(t, t_0)|| \leq \gamma_1\gamma_2\gamma_3||e^{J(t-t_0)}||$$

and thus the LTI system (diagonalizable or not) is exponentially stable as long as $\text{Re}(\lambda_i) < 0$ for all i.

□

Asymptotic Stability

An LTV system is *uniformly asymptotically stable* if (1) it is uniformly stable and (2) Given any positive constant δ there exists a positive T such that for any x_0, t_0 the solution to the system satisfies

$$||x(t)|| \leq \delta ||x_0||, \quad t \geq t_0 + T$$

In other words, the system becomes uniformly stable after finite time T has elapsed.

5.4.3 The Routh Stability Criterion

Given the characteristic polynomial corresponding to the dynamics matrix A of an LTI system,

$$p_A(\lambda) = \lambda^n + a_1\lambda^{n-1} + \cdots + a_n$$

the *Routh Stability Criterion* allows for determination of the number of unstable (positive real part) roots based on an algebraic test.

First, note that a necessary (but not sufficient) condition for stability (all eigenvalues have negative real parts) is that the coefficients of the characteristic polynomial are all positive.

To check sufficiency, construct the *Routh array* and ensure that the first column is positive.

5.4.4 Lyapunov Stability Theory

It has been motivated in class that the "energy" of a linear system is of a quadratic form. Intuitively, the energy of a system must go to zero as time goes to infinity if the system is stable. *Lyapunov stability theory* extends this idea to linear systems for which energy is not readily defined.

Let the energy associated with a system having state $x(t)$ at time t be

$$V(x,t) = x^T(t)P(t)x(t)$$

where $P(t) \in R^{n\times n}$ is symmetric and positive semidefinite (so that the energy in the system is always ≥ 0). We wish to study the rate of change of energy in the system and thus compute

$$\frac{dV}{dt} = \dot{x}^T Px + x^T\dot{P}x + x^TP\dot{x}$$

but we know that $\dot{x} = Ax$, so

$$\begin{aligned}\dot{V} &= \dot{x}^T P x + x^T \dot{P} x + x^T P \dot{x} \\ &= x^T A^T P x + x^T \dot{P} x + x^T P A x \\ &= x^T \left[A^T P + P A + \dot{P} \right] x\end{aligned}$$

In order for the system to be stable, we require that the rate of change of its energy must be negative. We state this formally (with additional conditions) as follows.

Uniform Stability

The linear system is *uniformly stable* if there exists a $P(t) \in R^n$ which is continuously differentiable, symmetric for all t and satisfies

1. $\eta I \preceq P(t) \preceq \rho I, \quad \eta, \rho > 0$
2. $A^T(t)P(t) + P(t)A(t) + \dot{P}(t) \preceq 0 \forall t$

Proof. $\leftarrow$ Consider integrating the derivative of energy V of the system from time t_0 to time t to determine the change in energy between these times. This yields

$$\int_{t_0}^{t} \frac{dV}{d\tau} d\tau = V(t) - V(t_0) = x^T(t)P(t)x(t) - x^T(t_0)P(t_0)x(t_0)$$

where the definition of $V(x,t)$ has been used. However, from above, we have the expression

$$\frac{dV}{dt} = x^T(t) \left[A^T(t)P(t) + P(t)A(t) + \dot{P}(t) \right] x(t)$$

And thus the same integral yields

$$\int_{t_0}^{t} \frac{dV}{d\tau} d\tau = \int_{t_0}^{t} x^T(\tau) \left[A^T(\tau)P(\tau) + P(\tau)A(\tau) + \dot{P}(\tau) \right] x(\tau) d\tau$$

From the second condition above, the integrand is the quadratic form of a negative semidefinite matrix and thus is nonpositive for all time; the integral must therefore be nonpositive. Thus,

$$x^T(t)P(t)x(t) - x^T(t_0)P(t_0)x(t_0) \leq 0$$

and hence

$$x^T(t)P(t)x(t) \leq x^T(t_0)P(t_0)x(t_0)$$

for all $t \geq t_0$. From the first condition above, we then have

$$x^T(t)P(t)x(t) \leq x^T(t_0)P(t_0)x(t_0)$$
$$\eta x(t)^T x(t) \leq \rho x(t_0)^T x(t_0)$$
$$||x(t)|| \leq \sqrt{\frac{\rho}{\eta}}||x(t_0)||$$

which implies uniform stability.

□

Note that this is a sufficient condition for uniform stability of an LTV system, but not a necessary one.

Exponential Stability

The linear system is *exponentially stable* if there exists a $P(t) \in R^n$ which is continuously differentiable, symmetric for all t and satisfies

1. $\eta I \preceq P(t) \preceq \rho I, \quad \eta, \rho > 0$
2. $A^T(t)P(t) + P(t)A(t) + \dot{P}(t) \preceq -Q \forall t$

where Q is symmetric and positive definite. (Note that for a positive definite matrix Q we have $\lambda_{min}(Q)||x||_2^2 \leq x^T Q x \leq \lambda_{max}(Q)||x_2^2)$.

The above theorem applies to all linear system and provides a sufficient condition for exponential stability. However, for an LTI system we have a stronger theorem as follows.

Exponential Stability of LTI Systems

An LTI system is exponentially stable if and only if the
Lyapunov equation

$$A^T P + PA = -Q$$

has a unique solution P for some positive definite matrix Q.
Notes:

1. The above theorem provides a recipe for checking for exponential stability: start with some $Q \succ 0$ and solve for P. If any P isn't symmetric positive semidefinite then the system can immediately be declared unstable.
2. $Q \succ 0$ is arbitrary, so $Q = I$ is often chosen.
3. The Lyapunov equation can be written as a *Linear Matrix Inequality* as follows: find P such that $A^T P + PA \prec 0$.
4. The solution to the Lyaponuv equation for any Q is given by $P = \int_0^\infty e^{A^T t} Q e^{At} dt$.

5.4.5 BIBO Stability

Consider the LTV system with zero initial conditions given by

$$\begin{aligned} \dot{x}(t) &= A(t)x(t) + B(t)u(t) \\ y &= C(t)x(t) + D(t)u(t) \end{aligned}$$

Recall that the output of this system can be written as

$$y(t) = \int_0^t C(t)\Phi(t,\tau)B(\tau)u(\tau)d\tau + D(t)u(t)$$

The above system is *uniformly BIBO stable* if there exists a finite, positive constant γ such that for every input $u(t)$ the forced response $y(t)$ satisfies

$$\sup_{t\in[0,\infty]}||y_f(t)|| = \gamma \sup_{t\in[0,\infty]}||u(t)||$$

where sup differs from max in that it denotes the *minimal* upper bound.

Theorem: The LTV system is BIBO stable if and only if every entry of $D(t)$ is uniformly bounded and

$$\sup_{t\geq 0}\int_0^t |g_{ij}(t,\tau)|d\tau < \infty$$

for every entry $g_{ij}(t,\tau)$ of the matrix $G(t,\tau) = C(t)\Phi(t,\tau)B(\tau)$ (this is the *impulse response function*, the matrix which gets convolved with $u(t)$ to produce the output $y(t)$).

Proof. $\leftarrow$ Compute the norm of the output as follows.

$$\begin{aligned} ||y(t)|| &= ||C(t)\int_0^t \Phi(t,\tau)B(\tau)u(\tau)d\tau + D(t)u(t)|| \\ &\leq \int_0^t ||C(t)\Phi(t,\tau)B(\tau)||||u(\tau)||d\tau + ||D(t)||||u(t)|| \end{aligned}$$

which results from the triangle and Cauchy-Schwarz norm inequalities. Now, define $\mu = \sup_{t\in[0,\infty]}||u(t)||$ and $\delta = \sup_{t\in[0,\infty]}$ so that

$$||y(t)|| \leq \left(\int_0^t ||G(t,\tau)||d\tau + \delta\right)\mu$$

Note that

$$||G(t,\tau)|| \leq \sum_{i,j=1}^{n} |g_{ij}(t,\tau)|$$

from the definition of the matrix p-norms (consider $p = 1$ or $p = \infty$ which sum only in one column or row, since the type of norm used does not affect the stability analysis). Likewise, we may write

$$\int_0^t ||G(t,\tau)||d\tau \leq \int_0^t \sum_{i,j=1}^{n} |g_{ij}(t,\tau)|d\tau$$

and thus

$$||y(t)|| \leq \sup \int_0^t ||G(t,\tau)||d\tau + \delta < \infty$$

from the second condition above.

□

For LTI systems, we have $G(t,\tau) = Ce^{A(t-\tau)}B$ so the second condition above becomes

$$\sup_{t\geq 0} \int_0^t |g_{ij}(t-\tau)|d\tau < \infty$$

Further, for LTI systems we have the following theorem.

Theorem: An LTI system is BIBO stable if and only if every pole of every entry of the transfer function $G(s) = C(sI - A)^{-1}B$ has strictly negative real part.

The important thing to note here is: *all poles of the transfer functions are eigenvalues of the system but not all eigenvalues of the system show up as poles of the transfer functions!*

Due to cancellation, it is possible for a system to be BIBO stable but not internally (exponentially) stable. We conclude with the following theorem.

Theorem: An LTI system is BIBO stable if it is internally (exponentially) stable (this implies that all the eigenvalues of the system have negative real part). The converse is not true in general!

5.5 Controllability and Observability

All linear controllers involve state feedback and thus two important questions arise: (i) Is it always possible to apply an input such that the state of the system is forced to some desired state? (ii) Is it always possible to determine the state of the system (needed to apply feedback) from the measured output?

A linear system (time-varying, in general) is said to be *controllable* if there exists a finite time $t_f > t_0$ such that for any initial initial state $x(t_0)$ and any desired state $x(t_f)$ there exists a finite finite/bounded control input $u(t)$ defined on $[t_0, t_f]$ which transfers the system state from $x(t_0)$ to $x(t_f)$.

A linear system is said to be *observable* if there exists a finite $t_f > t_0$ such that for any initial state $x(t_0)$, knowledge of the input $u(t)$ and the measured output $y(t)$ over the interval $[t_0, t_f]$ is sufficient to determine $x(t)$ on the interval.

5.5.1 Controllability

A linear system is *controllable* over the interval $[t_0, t_f]$ if and only if the *controllability Grammian*

$$W_c(t_0, t_f) = \int_{t_0}^{t} \Phi(t_0, \tau) B(\tau) B^T(\tau) \Phi^T(t_0, \tau) d\tau$$

is nonsingular.

Proof. $\rightarrow$ Given the LTV system and and the desired final state $x(t_f)$, we have for some control input $u(t)$ the solution

$$x(t_f) = \Phi(t_f, t_0) x(t_0) + \int_{t_0}^{t_f} \Phi(t_f, \tau) B(\tau) u(\tau) d\tau$$

We desire to solve the above equation for $u(t)$ given $x(t_0) = x_0$. Defining $F(\tau) = \Phi(t_f, \tau) B(\tau)$ and rearranging, we have

$$x_f - \Phi(t_f, t_0) x_0 = \int_{t_0}^{t_f} F(\tau) u(\tau) d\tau$$

In order for the above system to have a solution $u(t)$, we know that $F(t)$ must have linearly independent rows. The condition for linear independence of the rows of $F(t)$ is that the matrix

$$\int_{t_0}^{t_f} F(\tau) F^T(\tau) d\tau = \int_{t_0}^{t} \Phi(t_0, \tau) B(\tau) B^T(\tau) \Phi^T(t_0, \tau) d\tau = W_c(t_0, t_f)$$

is nonsingular. □

For an LTI system, we have that

$$F(\tau) = e^{A(t_0 - \tau)} B$$

In order for the rows of $F(t)$ to be linearly independent in general, we must show that for some $t' \in [t_0, t_f]$

$$\operatorname{rank} \begin{bmatrix} F(t') & \dot{F}(t') & F^{(2)}(t') & \cdots & F^{(n)}(t') & \cdots \end{bmatrix} = n$$

or, in the LTI case,

$$\operatorname{rank} \begin{bmatrix} e^{A(t_0 - t')} B & -e^{A(t_0 - t')} AB & e^{A(t_0 - t')} A^2 B & \cdots & (-1)^{n-1} e^{A(t_0 - t')} A^{n-1} B \cdots \end{bmatrix} = n$$

Since we are free to choose any $t' \in [t_0, t_f]$, choose $t' = t_0$. Then controllability is determined by the rank of the *controllability matrix* as

$$\operatorname{rank} \begin{bmatrix} B & AB & A^2 B & \cdots & A^{n-1} B & \cdots \end{bmatrix} = n$$

Recall that from Cayley-Hamilton, we have that $p_\lambda(A) = A^n + a_1 A^{n-1} + a_2 A^{n-1} + \cdots + a_n I = 0$ and thus

$$\begin{aligned} A^n &= -a_1 A^{n-1} - a_2 A^{n-2} - \cdots - a_n I \\ A^{n+1} &= -a_1 A^n - a_2 A^{n-1} - \cdots - a_n A \\ &\vdots \\ A^{n+j} &= -a_1 A^{n+j-1} - a_2 A^{n+j-2} - \cdots - a_n A^j \end{aligned}$$

and thus every power of A after A^{n-1} is a linear combination of lower powers of A. Terms in the controllability matrix involving powers of A past A^{n-1} thus do not affect the rank of the controllability matrix! The condition for controllability of the LTI system thus becomes

$$\text{rank } C(A,B) = \text{rank } \begin{bmatrix} B & AB & A^2B & \cdots & A^{n-1}B \end{bmatrix} = n$$

where $C(A,B) \in R^{n \times nm}$ is the controllability matrix. For a SISO system, $C(A,b) \in R^{n \times n}$ must have nonzero determinant.

5.5.2 Observability

A linear (in general, time-varying) system is *observable* on the interval $[t_0, t_f]$ if and only if the *observability Grammian*

$$W_o(t_0, t_f) = \int_{t_0}^{t_f} \Phi^T(\tau, t_0) C^T(\tau) C(\tau) \Phi(\tau, t_0) d\tau$$

is nonsingular.

Following a similar method as given for controllability, it can be shown that the condition for observability of an LTI system is

$$\text{rank } O(A,C) = \text{rank } \begin{bmatrix} C^T & A^T C^T & (A^T)^2 C^T & \cdots & (A^T)^{n-1} C^T \end{bmatrix} = n$$

where $O(A,C)$ is the *observability matrix.* Also note that we may write the condition as

$$\text{rank } O(A,C) = \text{rank } O^T(A,C) \text{rank} \begin{bmatrix} C \\ CA \\ CA^2 \\ \cdots \\ C^{n-1}A \end{bmatrix} = n$$

5.5.3 Duality

Note that setting $C = B^T$ and $A = A^T$ in $O(A, C)$ yields $C(A, B)$; it is thus said that observability and controllability are *dual*.

Let

$$\dot{x} = Ax + Bu, \quad y = Cx$$

denote the *primal system* and let

$$\dot{z} = A^T z + C^T u, \quad y = B^T z$$

denote the *dual system*. Duality says that the primal system is controllable if the dual system is observable (and vice-versa).

The following statements are equivalent for LTI systems:

- The pair (A, B) is controllable.
- The matrix W_c is nonsingular.
- rank$C(A, B) = n$
- The matrix $[A - \lambda I, B] \in R^{n \times 2n}$ has rank n for every eigenvalue λ of A (this is the Popov-Bellevitch-Hautus or PBH test). Equivalently, the system is uncontrollable iff there exists a left eigenvector x of A which lies in the nullspace of B, ie $x^T A = \lambda x^T$ and $x^T B = 0$.
- If all eigenvalues of A have negative real part, then the unique solution to the Lyapunov equation $AW_c + W_c A^T = -BB^T$ is positive-definite and given by $W_c = \int_0^\infty e^{A\tau} BB^T e^{A^T \tau} d\tau$. Thus, for a stable system we can calculate the controllability Grammian using this Lyapunov equation (for which there are methods of solution). Note that this comes from taking the derivative of the matrix W_c and solving for the steady-state (ie $\dot{W}_c = 0$) solution of the resulting Lyapunov equation.

5.5.4 Lyapunov Stability (updated)

Given an LTI system, the following conditions are equivalent:

- The system is asymptotically stable.
- The system is exponentially stable.
- The real parts of all eigenvalues of A are negative.
- The Lyapunov equation $PA + A^T P = -Q$ has a unique, positive-definite solution P for any positive-definite matrix Q. The solution is given by $P = \int_0^\infty a^{A^T t} Q e^{At} dt$ (it's clear that if the real parts of $\lambda(A)$ are nonnegative then this solution will be unbounded!)

- There exists a matrix $P \succ 0$ for which the inequality $A^TP + PA \prec 0$ (there are multiple solutions P and if none are positive definite then the system is unstable!)
- For every pair (A, B) that is controllable, there is a unique solution $P \succ 0$ to the Lyapunov equation $AP + PA^T = -BB^T$ (Note that BB^T is always positive semi-definite since $x^TBB^Tx = (B^Tx)^T(B^Tx) = ||B^Tx||_2^2 \geq 0$). The solution to this equation is the controllability grammian, which is guaranteed to be positive definite for a controllable system.
- For every pair (A, C) that is observable, there is a unique solution $P \succ 0$ to the Lyapunov equation $A^TP + PA = -C^TC$ (Note that C^TC is always positive semi-definite since $x^TC^TCx = (Cx)^T(Cx) = ||Cx||_2^2 \geq 0$). The solution to this equation is the observability grammian, which is guaranteed to be positive definite for an observable system.

Note that if A is not symmetric, its quadratic form can still be defined by writing $A = \frac{1}{2}(A+A^T)+\frac{1}{2}(A-A^T)$ where $(A+A^T)$ is symmetric and $(A-A^T)$ is antisymmetric (or skew-symmetric). It can be shown that $x^T(A-A^T)x = 0$ and thus we have $x^TAx = x^T(A+A^T)x$ (to see this, note that for some antisymmetric matrix R, we have $(x^TRx) = (x^TRx)^T = x^TR^Tx = -x^TRx$ and thus $x^TRx = 0$).

5.5.5 Equivalent System Representations

Recall that the linear system

$$\dot{x} = Ax + Bu, \quad y = Cx$$

is said to be similar to the system

$$\dot{\bar{x}} = \bar{A}\bar{x} + \bar{B}u, \quad y = \bar{C}\bar{x}$$

where $\bar{T} = T^{-1}AT$, $\bar{B} = T^{-1}B$, and $\bar{C} = CT$ via the state transformation $x = T\bar{x}$ where T is some nonsingular matrix. For such systems, we have $G(s) = C(sI-A)^{-1}B = \bar{C}(sI - \bar{A})^{-1}\bar{B}$, that is, the transfer function of the system is the same.

Theorem: Observability and controllability properties are invariant under equivalence tranformations.

Proof. Consider the controllability matrix $C(A, B)$. For the original and transformed systems, this is

$$C(A, B) = \begin{bmatrix} B & AB & A^2B & \cdots & A^{n-1}B \end{bmatrix}$$
$$C(\bar{A}, \bar{B}) = \begin{bmatrix} \bar{B} & \bar{A}\bar{B} & \bar{A}^2\bar{B} & \cdots & \bar{A}^{n-1}\bar{B} \end{bmatrix}$$

Note that since $\bar{A} = T^{-1}AT$, we have $\bar{A}^k = (T^{-1}AT)(T^{-1}A)\cdots(T^{-1}AT) = T^{-1}A^kT$ and thus $\bar{A}^k\bar{B} = (T^{-1}A^kT)(T^{-1}B) = T^{-1}A^kB$ and so

$$\begin{aligned} C(\bar{A}, \bar{B}) &= \begin{bmatrix} \bar{B} & \bar{A}\bar{B} & \bar{A}^2\bar{B} & \cdots & \bar{A}^{n-1}\bar{B} \end{bmatrix} \\ &= \begin{bmatrix} T^{-1}B & T^{-1}AB & T^{-1}A^2B & \cdots & T^{-1}A^{n-1}B \end{bmatrix} \\ &= T^{-1}C(A, B) \end{aligned}$$

Assume that the original system is controllable. Recall that for two matrices A, B with A invertible, we know that rank (AB) = rank B, that is, multiplication by an invertible matrix does not change the rank. Since rank $C(\bar{A}, \bar{B})$ = rank $T^{-1}C(A, B)$ = rank $C(A, B) = n$, we have that the transformed system must also be controllable.

□

Theorem: Suppose that the LTI system $\{A, B, C\}$ is such that rank $C(A, B) = r < n$, that is, the system is uncontrollable. Then there exists an equivalence transformation T such that

$$\bar{A} = \begin{bmatrix} \bar{A}_c & \bar{A}_{c\bar{c}} \\ 0 & \bar{A}_{\bar{c}} \end{bmatrix}, \quad \bar{B} = \begin{bmatrix} \bar{B}_c \\ 0 \end{bmatrix}, \quad \bar{C} = \begin{bmatrix} \bar{C}_c & \bar{C}_{\bar{c}} \end{bmatrix}$$

where for a system with n states, m inputs and k outputs,

$$\begin{aligned} A_c &\in R^{r \times r} \\ \bar{A}_{c\bar{c}} &\in R^{r \times n-r} \\ \bar{A}_{\bar{c}} &\in R^{n-r \times n-r} \\ \bar{B}_c &\in R^{r \times m} \\ \bar{C}_c &\in R^{k \times r} \\ \bar{C}_{\bar{c}} &\in R^{k \times n-r} \end{aligned}$$

The following statements about this realization hold.

- The pair $(\bar{A}_c, \bar{B}_c)$ is controllable, that is, rank $C(\bar{A}_c, \bar{B}_c) = r$.
- The states of the transformed system $\{\bar{A}, \bar{B}, \bar{C}\}$ are separated into controllable and uncontrollable with $\bar{x} = [\bar{x}_c^T, \bar{x}_{\bar{c}}^T]^T$ where $\bar{x}_c \in R^r$ and $\bar{x}_{\bar{c}} \in R^{n-r}$.
- The transfer function satisfies $G(s) = C(sI - A)^{-1}B = \bar{C}_c(sI - \bar{A}_c)^{-1}\bar{B}_c$. Note that since $(sI - A)^{-1}$ involves an n^{th} order polynomial and $(sI - \bar{A}_c)$ involves an r^{th} order polynomial (where $r < n$) there must be $n - r$ cancellations.

Writing out the equations for the transformed system using this realization, we have

$$\begin{aligned} \dot{\bar{x}}_c &= \bar{A}_c\bar{x}_c + \bar{A}_{c\bar{c}}\bar{x}_{\bar{c}} + \bar{B}_c u \\ \dot{\bar{x}}_{\bar{c}} &= \bar{A}_{\bar{c}}\bar{x}_{\bar{c}} \end{aligned}$$

The solution to the second system of equations is clearly $\bar{x}_c(t) = e^{\bar{A}_{\bar{c}}t}\bar{x}_c(0)$ and thus the control input has no effect on these states!

Since the original and transformed state vectors are related by the transformation $\bar{x} = T^{-1}x$, we may write $T = \begin{bmatrix} R_1 \\ R_2 \end{bmatrix}$ and thus $\bar{x} = \begin{bmatrix} \bar{x}_c \\ \bar{x}_{\bar{c}} \end{bmatrix} = \begin{bmatrix} R_1 x \\ R_2 x \end{bmatrix}$ which makes it clear that the controllable and uncontrollable states are linear combinations of the original states.

Theorem: Suppose that the LTI system $\{A, B, C\}$ is such that rank $O(A, C) = r < n$, that is, the system is unobservable. Then there exists an equivalence transformation T such that

$$\bar{A} = \begin{bmatrix} \bar{A}_o & 0 \\ \bar{A}_{o\bar{o}} & \bar{A}_{\bar{o}} \end{bmatrix}, \quad \bar{B} = \begin{bmatrix} \bar{B}_o \\ \bar{B}_{\bar{o}} \end{bmatrix}, \quad \bar{C} = \begin{bmatrix} \bar{C}_o & 0 \end{bmatrix}$$

where for a system with n states, m inputs and k outputs,

$$\begin{aligned} A_o &\in R^{r\times r} \\ \bar{A}_{o\bar{o}} &\in R^{n-r\times r} \\ \bar{A}_{\bar{o}} &\in R^{n-r\times n-r} \\ \bar{B}_o &\in R^{r\times m} \\ \bar{B}_{\bar{o}} &\in R^{n-r\times m} \\ \bar{C}_o &\in R^{k\times r} \end{aligned}$$

The following statements about this realization hold.

- The pair $(\bar{A}_o, \bar{C}_o)$ is observable, that is, rank $O(\bar{A}_c, \bar{C}_o) = r$.
- The states of the transformed system $\{\bar{A}, \bar{B}, \bar{C}\}$ are separated into observable and unobservable with $\bar{x} = [\bar{x}_o^T, \bar{x}_{\bar{o}}^T]^T$ where $\bar{x}_o \in R^r$ and $\bar{x}_{\bar{o}} \in R^{n-r}$.
- The transfer function satisfies $G(s) = C(sI - A)^{-1}B = \bar{C}_c(sI - \bar{A}_c)^{-1}\bar{B}_c$ (again, cancellation must occur).

Writing out the equations for the transformed system using this realization, we have

$$\begin{aligned} \dot{\bar{x}}_o &= \bar{A}_o\bar{x}_o + \bar{B}_o u \\ \dot{\bar{x}}_{\bar{o}} &= \bar{A}_{o\bar{o}}\bar{x}_o + \bar{A}_{\bar{o}}\bar{x}_{\bar{o}} + \bar{B}_{\bar{o}}u \\ y &= \bar{C}_o\bar{x}_o \end{aligned}$$

It's thus clear from the output equation that $\bar{x}_{\bar{o}}$ contributes nothing to the output and thus cannot be observed! Again, we can show that the observable and unobservable states are linear combinations of the states of the original system.

There are four types of states: those which are controllable, those which are observable, those which are controllable and observable and those which are neither.

Minimal Realizations

A realization is called *minimal* if it has the smallest matrix A among all triples $\{A, B, C\}$ which satisfy $G(s) = C(sI-A)^{-1}B$ (of course, there are infinite such minimal realizations via similarity so this representation is not unique).

Theorem: A realization is *minimal* if and only if it is both controllable and observable.

If $\{A_1, B_1, C_1\}$ and $\{A_2, B_2, C_2\}$ are two minimal realizations of a transfer function $G(s)$, there exists a unique, invertible matrix T such that

$$A_2 = T^{-1}A_1T, \quad B_2 = T^{-1}B_1, \quad C_2 = C_1T$$

Furthermore, T is given by either

$$T = C_1C_2^T(C_2C_2^T)^{-1}$$
$$T = (O_2^TO_2)^{-1}O_2^TO_1$$

where we note that the first equation involves the right pseudoinverse $C_2^\dagger = C_2^T(C_2C_2^T)^{-1}$ and the second equation involves the left pseudoinverse $O_2^\dagger = (O_2^TO_2)^{-1}O_2^T$. For a SISO system, these matrices are square and so the expressions reduce to

$$T = C_1C_2^{-1}$$
$$T = O_2^{-1}O_1$$

5.6 State Feedback Control

Assuming that the state of the system

$$x = Ax + Bu, \quad y = Cx$$

is known exactly, we design a feedback controller $u_{fb} = -Kx$; letting $u_{ext} = \nu$ be a reference or disturbance input, the total input is then $u = -Kx + \nu$ and the closed-loop system becomes

$$x = (A - BK)x + B\nu, \quad y = Cx$$

The dynamics of the system are thus governed by the matrix $A_{cl} = A - BK$, referred to as the closed-loop dynamics matrix. Our choice of the gain matrix K thus determines how the system behaves.

5.6.1 Popov-Bellevitch-Hautus Controllability Test

The Popov-Bellevitch-Hautus (PBH) test states that the pair (A, B) is controllable if every eigenvector of A^T (left eigenvector of A) is not in the nullspace of B^T (the left nullspace of B).

Theorem: If the pair (A, B) is controllable, then the pair $(A+\mu I, B)$, $\mu \in R$ is also controllable.

Proof. The matrix $A + \mu I$ has the same eigenvectors as A and eigenvalues each shifted by μ. If (A, B) is controllable (passes the above PBH test) then since the eigenvectors do not change, $(A + \mu I, B)$ must also be controllable. □

Since $(A+\mu I, B)$ is controllable, we know that the solution to the Lyapunov equation

$$(\mu I + A)W + W(\mu I + A)^T = -BB^T$$

exists and is positive definite (specifically, it is the controllability grammian). Expanding this expression and noting that W is nonsingular, we find the following.

$$\begin{aligned}(\mu I + A)W + W(\mu I + A)^T &= -BB^T \\ AW + WA^T + BB^T &= -2\mu W \\ W^{-1}AW + A^T + W^{-1}BB^T &= -2\mu I \\ W^{-1}A + A^T W^{-1} + W^{-1}BB^T W^{-1} &= -2\mu W^{-1}\end{aligned}$$

Letting $P = W^{-1}$ (which is also positive definite since the eigenvalues of P are the reciprocals of the eigenvalues of W), this becomes

$$PA + A^T P + PBB^T P = -2\mu P$$

We can write this equation as (???)

$$P(A - BK) + (A - BK)^T P = -2\mu P$$

Since $P \succ 0$, then $2\mu P \succ 0$; since we know that there exists a positive definite solution $P = W^{-1}$ to this Lyapunov equation, it follows that $A - BK$ is stable.

Theorem: If the LTI system is controllable, then it is possible to find a feedback controller $u = -Kx$ which places all eigenvalues of the closed-loop system in the complex semiplane $Re(s) \leq \mu$. If the system is not controllable, then there exists a state representation for which the state vector is partitioned into the controllable states $\bar{x}_c$ and the uncontrollable states $\bar{x}_{\bar{c}}$ with dynamics

$$\begin{aligned}\dot{\bar{x}}_c &= \bar{A}_c \bar{x}_c + \bar{A}_{c\bar{c}} \bar{x}_{\bar{c}} + \bar{B}_c u \\ \dot{\bar{x}}_{\bar{c}} &= \bar{A}_{\bar{c}} \bar{x}_{\bar{c}}\end{aligned}$$

Since the input u cannot affect the uncontrollable states $\bar{x}_{\bar{c}}$, we can never choose a gain matrix which stabilizes these states. The best we can do is to design a state feedback controller for the controllable part $barx_c$ by treating the term $d = \bar{A}_{c\bar{c}}\bar{x}_{\bar{c}}$ as a disturbance. Since $(\bar{A}_c, \bar{B}_c)$ is controllable, we can design a controller $u = -K_c\bar{x}_c$ which places the place the poles of matrix $(\bar{A}_c - \bar{B}_c K_c)$ and thus determines the response of the controllable part.

If $\bar{A}_{c\bar{c}}$ is a stable matrix then the uncontrollable system is stable; in this case we say that the total system is *stabilizable* rather than controllable.

Theorem: If the LTI system is stabilizable (that is, the uncontrollable states are stable) then it is always possible to design a feedback controller $u = -Kx$ such that the closed-loop dynamics matrix $(A - BK)$ is stable.

5.6.2 State Feedback for SISO Systems

Consider the SISO system

$$x = Ax + bu, \quad y = c^T x$$

having the transfer function (assumed here to be strictly proper) given by

$$g(s) = \frac{y(s)}{u(s)} = \frac{b_1 s^{n-1} + b_2 s^{n-2} + \ + b_n}{s^n + a_1 s^{n-1} + a_2 s^{n-2} + \ + a_n}$$

We can always put such a system into controllable form as follows.

$$\dot{x}_c = A_c x_c + b_c u, \quad y = c_c^T x_c$$

where

$$A_c = \begin{bmatrix} -a_1 & \cdots & -a_{n-1} & -a_n \\ 1 & \cdots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 1 & 0 \end{bmatrix}, \quad b_c = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad c_c = \begin{bmatrix} b_1 & b_2 & \cdots & b_n \end{bmatrix}$$

Assume a control law $u = -k_c^T x_c + \nu$ so that the closed-loop system becomes $\dot{x} = A_{cl}x_c + b_c\nu$ where $A_{cl} = (A_c - b_c k_c^T)$. Its clear that

$$b_c k_c^T = \begin{bmatrix} 1 \\ \vdots \\ 0 \end{bmatrix} \begin{bmatrix} k_{c1} & \cdots & k_{cn} \end{bmatrix} = \begin{bmatrix} k_{c1} & \cdots & k_{cn} \\ 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{bmatrix}$$

and thus

$$A - b_c k_c^T = \begin{bmatrix} -a_1 - k_{c1} & \cdots & -a_{n-1} - k_{cn-1} & -a_n - k_{cn} \\ 0 & \cdots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 1 & 0 \end{bmatrix}$$

We see that A_{cl} is still in controllable form; thus, the characteristic polynomial of the closed-loop system is

$$s^n + (a_1 + k_{c1})s^{n-1} + (a_2 + k_{c2})s^{n-2} + \cdots + (a_n + k_{cn})$$

Since the coefficients of the characteristic polynomial of the closed-loop transfer function are functions of the eigenvalues of A_{cl}, we can choose the coefficients of k_c to place the eigenvalues of A_{cl}.

Assume that we desire to place the closed-loop eigenvalues at the locations $\hat{\lambda}_1, \cdots, \hat{\lambda}_n$; the desired characteristic polynomial is thus

$$(s - \hat{\lambda}_1)(s - \hat{\lambda}_2) \cdots (s - \hat{\lambda}_n) = s^n + \alpha_1 s^{n-1} + \cdots + \alpha_n$$

To place the eigenvalues as such, it follows that

$$\begin{gathered} \alpha_1 = a_1 + k_{c1} \rightarrow k_{c1} = \alpha_1 - a_1 \\ \vdots \\ \alpha_n = a_n + k_{cn} \rightarrow k_{cn} = \alpha_n - a_n \end{gathered}$$

and thus $k_c = (\alpha - a)$ achieves the desired pole placement. Its very important to realize that state feedback does not alter the zeros of the system.

Of course, this only works for systems in controllable form, whereas most systems are not given in controllable form; we thus seek a means of designing a feedback controller for any controllable system. Recall that we can always find a transformation T which puts a given system into controllable form via the state transformation $x = Tx_c$. If we design a feedback controller $u_c = -k_c x_c$ for the transformed system then the feedback controller for the original system is simply $u = -k_c T^{-1} x$. We thus must determine the transformation T which puts the system into controllable form.

Recall that any minimal realization can be transformed to another minimal realization via the transformation matrix $T = C_1 C_2^\dagger = C_1^\dagger C_2$ or in the SISO case $T = C_1 C_2^{-1} = C_1^{-1} C_2$. For the purposes of controller design, we assumed that the state of the system could be measured precisely (ie, that the system is observable) and it is required that the system be controllable. We have thus assumed that the system is minimal (both controllable and observable). It follows that we may transform the original system into controllable form using the transformation $T = C C_c^{-1}$ where C and C_c are the controllability matrices of the original and controllable realizations, respectively.

It can be shown that

$$C_c^{-1} = \Delta^T = \begin{bmatrix} 1 & a_1 & a_2 & \cdots & a_{n-1} \\ & \ddots & \ddots & \ddots & \vdots \\ & & \ddots & \ddots & a_2 \\ & & & \ddots & a_1 \\ & & & & 1 \end{bmatrix}$$

and thus $T = C\Delta^T$ and finally $k^T = (\alpha - a)^T \Delta^{-T} C^{-1}$. Obviously, if the system is not controllable then C will not be invertible.

5.6.3 SISO Pole Placement using Lyapunov Equation

The method given above for placing the eigenvalues of a closed-loop SISO system using state feedback is not the only way of choosing the gain vector k. Alternatively, assume that the closed-loop dynamics matrix is similar to a second matrix F which has the desired eigenvalues via some nonsingular matrix T, ie

$$(A - bk^T) = TFT^{-1}$$

Since A_{cl} and F are similar, they have the same eigenvalues. We thus seek to determine k and T such that this similarity holds. Expanding out this equation yields

$$AT - TF = bk^T T = b\bar{k}^T$$

where we have defined $\bar{k}^T = k^T T$. This is thus a Lyapunov equation of the form $AX + XB = -C$ where $A = A$, $B = F$ and $C = -b\bar{k}^T$. If we are given $\bar{k}$ then we can solve the above equation for T subject to the conditions of the following theorem.

Theorem: If A and F have no eigenvalues in common then the above Lyapunov equation can be solved for a unique, nonsingular T if (A, b) is controllable and $(F, \bar{k}^T)$ is observable.

The theorem suggests the following procedure for eigenvalue placement using the above formula:

- First, select $F \in R^{n \times n}$ which has the desired eigenvalues. Note that F can share no eigenvalues with A (all the poles must be moved).
- Second, select $k \in R^n$ such that the pair $(F, \bar{k}^T)$ is observable.
- Finally, solve $AT - TF = b\bar{k}^T$ for T and conclude that $k^T = \bar{k}^T T^{-1}$.

The simplest way to choose F and $\bar{k}$ such that the pair $(F, \bar{k}^T T)$ is observable is to choose F to be the diagonal matrix of desired eigenvalues and $\bar{k}$ to be the vector of all ones. Note that if any of the desired eigenvalues are complex, choosing F in this way will result in a complex gain vector. Instead, choose F to be real block diagonal where a complex eigenvalue $\lambda = \alpha + \beta i$ shows up in the 2×2 block

$$\begin{bmatrix} \alpha & \beta \\ -\beta & \alpha \end{bmatrix}$$

5.7 Controller Design for MIMO Systems

Consider the MIMO system

$$x = Ax + Bu, \quad y = Cx$$

where $B \in R^{n \times m}$ and $u \in R^m$. Recall that pole placement for SISO systems was a matter of determining the gain vector $k \in R^n$ which placed the eigenvalues of the closed-loop dynamics matrix at the desired locations. Since there were n scalar gains and the characteristic polynomial was or order n, the gain vector could be chosen uniquely.

In MIMO systems (specifically multi-input systems, since the output does not affect controller design) the gain matrix $K \in R^{m \times n}$ cannot be uniquely specified given the desired eigenvalue locations. However, the advantage to there being infinite solutions to this problem is that one can both place the poles and alter the response in other ways.

Note that if a system is controllable using only one of its inputs, ie $\text{rank}\{C(A, b_i)\} = n$, then it is controllable using all of them. This is evident from the fact that when additional inputs are used, the controllability matrix grows in column dimension; even when the full matrix $C(A, B) \in R^{n \times mn}$ is used, it still has full row rank n. We have the following theorem which facilitates controller design for MIMO systems.

Theorem: All eigenvalues of $A_{cl} = (A - BK)$ can be placed arbitrarily with some $K \in R^{m \times n}$ iff the pair (A, B) is controllable.

Cyclic Design for MIMO Systems

The simplest way to choose a feedback law which accomplishes eigenvalue placement is to use the *cyclic* design method.

Theorem: Let A be cyclic (having Jordan form with one block associated with each distinct eigenvalue). If the pair (A, B) is controllable then for (almost) any $v \in R^m$ the pair (A, Bv) is controllable.

Using the preceding theorem, one can place the poles of the system $\dot{x} = Ax + \bar{b}w$ where $\bar{b} = Bv$ and $w = -k^T x$. The closed-loop system is thus $\dot{x} = (A - BK)x$ where $K = vk^T$. The effect of v is to distribute the required input among multiple actuators. For example, choosing $v = e_i$ uses only the i^{th} actuator while choosing v to be the vector of all ones evenly distributes control (???).

Lyapunov Method for MIMO Systems

We can also choose K using the Lyapunov method previously introduced for SISO systems.

- First, select $F \in R^{n \times n}$ which has the desired eigenvalues. Note that F can share no eigenvalues with A (all the poles must be moved).
- Second, select $\bar{K} \in R^{m \times n}$ such that the pair $(F, \bar{K})$ is observable.
- Finally, solve $AT - TF = b\bar{K}$ for the unique T and conclude that $K = \bar{K}T^{-1}$.
- If T is singular, choose a different $\bar{K}$ and start over.

(Questions: 1) If there are infinite solutions to the general problem of MIMO controller design, how is T here unique? 2) Why is it possible for T to be singular in this case but not in the SISO case?)

Note again that state feedback cannot affect the zeros of the system, which also play a large role in the reponse. For example, a system with a positive zero is called non-minimal phase; the result is that such a system typically moves in the opposite direction at first in the case of a step input.

5.7.1 Disturbance Rejection

Given the linear system

$$\dot{x} = Ax + Bu, \quad y = Cx$$

assume that the input is a constant (step) disturbance of unknown magnitude which we want to reject. Thus, we desire to drive the output to zero in the presence of this input. This can be accomplished by defining the additional state $x_{n+1} = \int_0^t y d\tau$ so that this state has dynamics $\dot{x}_{n+1} = y$. The augmented system is then

$$\begin{bmatrix} \dot{x} \\ \dot{x}_{n+1} \end{bmatrix} = \begin{bmatrix} A & 0 \\ C & 0 \end{bmatrix} \begin{bmatrix} x \\ x_{n+1} \end{bmatrix} + \begin{bmatrix} B \\ 0 \end{bmatrix} u$$

We then design a feedback controller for the augmented system so that the system is stable. At equilibrium, we have $\dot{x}_{n+1} = y = 0$ and thus the system will reject the disturbance as desired.

5.8 Optimal Control

Given the LTI system

$$\dot{x} = Ax + Bu, \quad y = Cx$$

we wish to find the feedback gain matrix K for which the control $u = -Kx$ minimizes the quadratic cost function

$$J = \int_0^T (x^T Q x + u^T R u) dt$$

called the *performance criterion* where $Q \succeq 0$ and $R \succ 0$ are weighting matrices and T is the terminal time. The solution to the optimal control problem is given by

$$K(t) = R^{-1}B^T P(t)$$

where $P(t)$ is the solution to the Ricatti equation

$$-\dot{P}(t) = A^T P(t) + P(t)A + P(t)BR^{-1}B^T P(t) + Q$$

Using the fact that $P(T) = 0$, the above equation is solved starting with $t = T$ and going back in time to find $P(t)$ for all $t < T$. However, in the case where $T \to \infty$ it can be shown that $P(t)$ becomes a constant matrix and thus $P = 0$. The Ricatti equation thus becomes an algebraic (instead of differential) equation given by

$$PA + A^T P - PBR^{-1}B^T P + Q = 0$$

5.9 Observers

In practice, we can never measure the state of a system directly. Even in the rare case in which C is invertible, solving directly for the state as $x = C^{-1}y$ is problematic because C is often incorrectly modeled.

Instead, we seek to design a structure called an observer which produces and estimate of the state of the system given the control input and measured output.

One possibility for an observer is to simply mimic the structure of the system itself. In general, this would give

$$\begin{aligned}\dot{x}(t) &= A(t)x(t) + B(t)u(t)\\ \dot{\hat{x}}(t) &= A(t)\hat{x}(t) + B(t)u(t)\end{aligned}$$

Defining the estimation error to be $e(t) = x(t) - \hat{x}(t)$, we have

$$\dot{e}(t) = A(t)e(t)$$

Clearly, this observer has some issues. First, the error only converges to zero if the system has stable open-loop dynamics. Second, the rate of convergence of the error to zero depends entirely on the open-loop eigenvalues and thus we have no control over this.

In order to solve these problems, we need to make use of the measured output which provides information about the state. We thus feed back the difference in the measured and expected outputs multiplied by a gain matrix L to yield the observer structure

$$\begin{aligned}\dot{\hat{x}} &= A\hat{x} + Bu + L(y - \hat{y})\\ &= A\hat{x} + Bu + L(y - C\hat{x})\\ &= (A - LC)\hat{x} + Bu + Ly\end{aligned}$$

In this case, the estimation error dynamics become

$$\dot{e} = (A - LC)e$$

and thus we may influence the convergence of the error to zero by choosing the gain matrix L to place the eigenvalues of the closed-loop observer matrix $A_o = A - LC$ as desired. How do we choose L to accomplish this, though?

Note that $\det(sI - A_o) = \det(sI - A^T) = \det(sI - (A^T - C^T L^T))$ and thus the task of choosing an observer gain L is identical to that of choosing a gain matrix K in the state feedback problem with the substitutions $A = A^T$, $B = C^T$ and $K = L^T$. In the SISO case, we thus have

$$l^T = (\alpha - a)^T C_c C^{-1}(A^T, c^T) = (\alpha - a)^T O_o O^{-1}(A, c)$$

by duality of controllability and observability. Of course, we can also use the Lyapunov method to accomplish SISO pole placement. For MIMO systems we may also use any of the methods discussed for controller designs (cyclic method, Lyapunov method, optimal control) but with the above substitutions from duality.

Recall that the structure of the SISO observer is

$$\dot{\hat{x}} = A_o\hat{x} + bu + ly, \quad \hat{y} = C\hat{x}$$

where $\hat{y}$ is the output from the observer. Taking the Laplace transform of both sides of the observer dynamics, we obtain $s\hat{X}(s) = A_o\hat{X}(s) + lY(s) + bU(s)$; solving for $\hat{X}(s)$ yields $\hat{X}(s) = (sI - A_o)^{-1}bU(s) + C(sI - A_o)^{-1}lY(s)$ and thus

$$\hat{Y}(s) = C(sI - A_o)^{-1}bU(s) + C(sI - A_o)^{-1}lY(s)$$

Since this is a linear system, we know that the output is a linear combination of the transformed inputs; thus, we may set $Y(s) = 0$ or $U(s) = 0$ in order to solve for the transfer functions relating the output to the measurement or the control input, respectively. These are then simply

$$G_y(s) = \frac{\hat{Y}(s)}{Y(s)} = C(sI - A_o)^{-1}l$$
$$G_u(s) = \frac{\hat{Y}(s)}{U(s)} = C(sI - A_o)^{-1}b$$

Note that both the measured output and input are, in practice, commonly afflicted by high-frequency noise. By choosing the eigenvalues of A_o to be very negative, we make the system have a high bandwith; as a result, high frequency noise may be amplified if the observer gain is too big! The gain is thus limited in practice by the noise in these inputs to the observer.

5.9.1 Reduced Order Observers

Often, one or more outputs in a MIMO system directly measure states; that is, C is of the form $C = [Ip0]$. Rather than using an n^{th} order observer, we can construct an observer of reduced order $n - p$ as follows.

Partition the state vector as $x = [x_a^T, x_b^T]^T$ where $x_a \in R^p$ corresponds to those states which are measured directly. We thus only need to estimate $x_b \in R^{(n-p)}$. The state equations can be written as

$$\begin{bmatrix} \dot{x}_a \\ \dot{x}_b \end{bmatrix} = \begin{bmatrix} A_{aa} & A_{ab} \\ A_{ba} & A_{bb} \end{bmatrix} \begin{bmatrix} x_a \\ x_b \end{bmatrix} + \begin{bmatrix} B_a \\ B_b \end{bmatrix} u$$

The states we wish to estimate thus have the dynamics

$$\dot{x}_b = A_{bb}x_b + B_b u + A_{ba}x_a$$

Since x_a is known at all times, we may treat it like a second input. The dynamics of the measured states are

$$\dot{x}_a = A_{aa}x_a + A_{ab}x_b + B_a u$$

We wish to express xb as a function of the known signals x_a and u. This yields

$$y_b = A_{ab}x_b = \dot{x}_a - A_{aa}x_a - B_a u$$

Making the substitutions $A = A_{bb}$, $Bu = (A_{ba}x_a + B_b u)$, $C = A_{ab}$ and $y = y_b$, the observer for x_b has the structure

$$\begin{aligned} \dot{x}_b &= (A_{bb} - LA_{ab})\hat{x}_b + (A_{ba}x_a + B_b u) + Ly_b \\ &= (A_{bb} - LA_{ab})\hat{x}_b + (A_{ba}x_a + B_b u) + L(\hat{x}_a - A_{aa}x_a - B_a u) \end{aligned}$$

The fact that the derivative of the measurement $y = x_a$ shows up in the above dynamics is an implementation problem, though. We can solve this problem by defining a new state $z = \hat{x}_b - Ly$ such that the observer can be written as

$$\begin{aligned} \dot{z} &= (A_{bb} - LA_{ab})z + [(A_{bb} - LA_{ab})L + (A_{ba} - LA_{aa})]y + (B_b - LB_a)u \\ &= A_o z + L_o y + B_o u \end{aligned}$$

where $y = x_a$ and u are known. The estimate of the state x_b is then computed as

$$\hat{x}_b = z + Ly$$

Additionally, it can be shown that the estimation error of the reduced order observer is described by the equation

$$\dot{e} = (A_{bb} - LA_{ab})e$$

Finally, note that

$$Z(s) = (sI - A_o)^{-1}L_o y + (sI - A_o)^{-1}B_o u$$

and thus

$$\hat{X}_b(s) = \left[(sI - A_o)^{-1}L_o + L\right] y + (sI - A_o)^{-1}B_o u$$

Note that L shows up as a direct feedthrough term in this representation.

It was previously shown how to design a reduced order observer for a system having $C = [I_p 0]$ describing its measured output. However, what if C is not given in this form? We seek a state transformation which will accomplish this so that we can design a reduced order observer in the same manner.

Define the state vector $\bar{x}$ to be

$$\bar{x} = \begin{bmatrix} y \\ \hat{C}x \end{bmatrix} = \begin{bmatrix} C \\ \hat{C} \end{bmatrix} x = Ux$$

where $C \in R^{(n-p)timesn}$ is any matrix chosen such that the transformation U is nonsin- gular. In this representation, the first p states are thus the outputs of the original system. The transformed system is then

$$\dot{\bar{x}} = UAU^{-1}\bar{x} + UBu, \quad y = CU^{-1}\bar{x} = [I_p 0]\bar{x}$$

Now, we can design an observer for the last $(n - p)$ states of $\bar{x}$ as before.

5.9.2 Observer Sensitivity

Recall that a system and its corresponding full-order observer have the dynamics

$$\dot{x} = Ax + Bu$$
$$\dot{\hat{x}} = (\hat{A} - L\hat{C})\hat{x} + \hat{B}u + Ly$$

where $\hat{A}$, $\hat{B}$ and $\hat{C}$ denote the models of A, B and C used in the observer. If these models coincide with the actual system then we have that the estimation error is governed by the equation

$$\dot{e} = (A - LC)e$$

If the model is imperfect, though, then we dont get the right cancellation to produce this equation and so the observer wont work well. Further, it can be shown that a reduced order observer is actually more sensitive to such modeling imperfections than is a full order observer.

5.9.3 Disturbance Estimation

Suppose that the state space system subject to disturbances is given as

$$\dot{x} = Ax + Bu + Fd$$

where d is the disturbance and F is a matrix which relates how the disturbances affect the state (it can be equal to B). Further, assume that we know a model of the disturbance dynamics given by

$$\dot{d} = A_d d$$

Defining the augmented state vector $z = [x^T, d^T]^T$ we may write

$$\begin{bmatrix} \dot{x} \\ \dot{d} \end{bmatrix} = \begin{bmatrix} A & F \\ 0 & A_d \end{bmatrix} \begin{bmatrix} x \\ d \end{bmatrix} + \begin{bmatrix} B \\ 0 \end{bmatrix} u, \quad y = \begin{bmatrix} C & 0 \end{bmatrix} \begin{bmatrix} x \\ d \end{bmatrix}$$

where we let $\bar{A}$, $\bar{B}$ and $\bar{C}$ denote the new system matrices. Assuming $\bar{A}$, $\bar{C}$ is observable we may write an observer for z in the usual manner, ie

$$\dot{z} = (\bar{A} - L\bar{C})z + \bar{B}u + Ly$$

The only question is how to model the disturbance dynamics. Some examples are given below.

- Assume d is a step (constant) disturbance of unknown magnitude, ie $d = C$. Define $z_1 = d = C$ and thus $\dot{z}_1 = 0$.
- Assume d is a ramp disturbance $d = d_0 + d_1 t$ with d_0, d_1 unknown. Define $z_1 = d$ and $z_2 = \dot{d} = d_1$ so that $\dot{z}_1 = \dot{d} = z_2$ and $\dot{z}_2 = \ddot{d} = 0$.
- Assume d is a harmonic of frequency ω but unknown amplitude given by $d = A\sin(\omega t + \phi)$. Define $z_1 = d$ and $z_2 = \dot{d}$ so that $\dot{z}_1 = z_2$ and $\dot{z}_2 = \ddot{d} = -\omega^2 z_1$.

Note that we can (in theory, but not usually in practice) reject disturbances caused by measurement noise n fed through matrix R in a similar way if the dynamics $\dot{n} = A_n n$ are known, resulting in the augmented system

$$\begin{bmatrix} \dot{x} \\ \dot{n} \end{bmatrix} = \begin{bmatrix} A & 0 \\ 0 & A_n \end{bmatrix} \begin{bmatrix} x \\ n \end{bmatrix} + \begin{bmatrix} B \\ 0 \end{bmatrix} u, \quad y = \begin{bmatrix} C & R \end{bmatrix} \begin{bmatrix} x \\ n \end{bmatrix}$$

5.10 Compensators and The Separation Principle

Thus far, we have designed a state feedback controller assuming the state of the system was known, resulting in the closed-loop dynamics

$$\dot{x} = (A - BK)x + B\nu, \quad y = Cx$$

However, we have also designed an observer to estimate the state of the system; the state feedback input is thus $u = -K\hat{x}$ and thus the closed-loop system using state feedback based on the observer estimate is

$$\dot{x} = Ax - BK\hat{x} + B\nu\dot{\hat{x}} = (A - LC - BK)\hat{x} + B\nu + Ly$$

Using the fact that $e = x - \hat{x}$, we can write these closed-loop equations in terms of the state and the estimation error as

$$\dot{x} = Ax - BK(x - e) + B\nu = (A - BK)x + BKe + B\nu$$
$$\dot{x} - \dot{e} = (A - LC - BK)(x - e) + B\nu + Ly \rightarrow \dot{e} = (A - LC)e$$

It follows that

$$\begin{bmatrix}\dot{x}\\ \dot{e}\end{bmatrix} = \begin{bmatrix}(A - BK) & BK\\ 0 & (A - LC)\end{bmatrix}\begin{bmatrix}x\\ e\end{bmatrix} + \begin{bmatrix}B\\ 0\end{bmatrix}\nu, \quad y = \begin{bmatrix}C & 0\end{bmatrix}\begin{bmatrix}x\\ e\end{bmatrix}$$

We know that the determinant of a block triangular matrix is the product of the determinants of its diagonal blocks. Denoting the above dynamics matrix by A_{cl}, we thus have

$$\det(sI - A_{cl}) = \det(sI - (A - BK))\det(sI - (A - LC))$$

which proves that the eigenvalues of the closed-loop system are the eigenvalues of the controller plus the eigenvalues of the observer. This is known as the *separation principle*. We may thus design a controller and an observer separately and put them together without them affecting one another.

We denote the combination of a controller and an observer a *compensator*; it transforms the measured output of the system into a feedback control input. Its transfer function is given by

$$G_{comp}(s) = \frac{U(s)}{Y(s)} = -K[sI - (A - LC - BK)]^{-1}L$$

Finally, note that the closed-loop transfer function of a system with a compensator is

$$G_{cl}(s) = \frac{Y(s)}{\nu(s)} = C(sI - (A - BK))^{-1}B$$

This corresponds to a system of order n despite the fact that the compensator is of order $2n$ - the n observer poles were thus cancelled! Recall that a transfer function describes steady-state (non-transient, ie zero initial conditions) behavior. During the transient phase there is some estimation error, but after convergence the observer poles cancel and the system looks like an n^{th} order system!

5.11 Frequency Domain Control Theory

5.11.1 Impulse Response

For a linear SISO system, an input signal u can be written using unit-area pulse signals (dirac deltas) as

$$u_\Delta(t) = \sum_{k=0}^{\infty} \Delta u(k\Delta)\delta_\Delta(t - k\Delta), \quad \forall t \geq 0$$

since the dirac-delta has unit area, width Δ and height $\frac{1}{\Delta}$.

For each $\tau \geq 0$, let $g_\Delta(t-\tau)$ denote an output corresponding to the input $\delta_\Delta(t-\tau)$ such that $\delta_\Delta(t-\tau) \Rightarrow g_\Delta(t-\tau)$.

Due to linearity, the output $y_\Delta(t)$ from input $u_\Delta(t)$ above can be written as

$$y_\Delta(t) = \sum_{k=0}^{\infty} \Delta u(k\Delta) g_\Delta(t - k\Delta), \quad \forall t \geq 0$$

In the limit as $\Delta \to 0$, $u_\Delta(t) \to u(t)$) and thus

$$lim_{\Delta \to 0} y_\Delta(t) = \int_0^{\infty} u(\tau) g(t-\tau), \quad \forall t \geq 0$$

where $\tau = k\Delta$ and

$$g(t-\tau) = lim_{\Delta \to 0} g_\Delta(t-\tau)$$

This function g maps an input Dirac pulse (of zero length but unit area) at time τ to an output at time t.

If the system is *causal* (the output at time t depends only on inputs before time t) and *time-invariant* (an input shifted in time produces the same output shifted in time) then the output is the *convolution* of $u(t)$ and $g(t)$

$$u * g = \int_0^{t} u(\tau) g(t-\tau), \quad \forall t \geq 0$$

Note that the upper limit changes due to the causality of the system.

For a MIMO system with k inputs and m outputs, the vector-valued *output* signal $y(t) \in \Re^m$ is

$$y(t) = \int_0^{\infty} G(t,\tau) u(\tau) d\tau, \quad \forall t \geq 0$$

where $G(t,\tau) \in \Re^{m \times k}$ is the matrix-valued *impulse response* signal and $u(\tau) \in \Re^k$ is the vector-valued *input* signal. The entry $g_{ij}(t,\tau)$ is the i^{th} entry of an output at time t corresponding to a Dirac pulse applied at the j^{th} input at time τ.

5.11.2 Laplace Transform

If the system is causal and time-invariant then the system response can be computed by moving from the *time domain* (where the output is the convolution of the input and impulse response) to the *frequency domain* (where the transformed output is the product of the transformed input and impulse responses).

Given a continuous-time signal $x(t)$, $t \geq 0$ its *unilateral Laplace transform* is given by

$$\mathscr{L}(x(t)) = X(s) = \int_0^\infty e^{-st} x(t) dt, \quad s \in \mathscr{C}$$

Importantly, the Laplace transform of the convolution of two signals $x(t)$ and $y(t)$ is

$$\mathscr{L}(x * y) = \mathscr{L}\left[\int_0^t x(\tau) y(t-\tau) d\tau\right] = X(s)Y(s)$$

The Laplace transform of the derivative $\dot{x}(t)$ is

$$\mathscr{L}(\dot{x}(t)) = sX(s) - x(0)$$

Transfer Function

The Laplace transform of the output $y(t)$ of a continuous-time linear system is

$$\mathscr{L}\left[\int_0^\infty G(t-\tau)u(\tau)d\tau\right] = \int_0^\infty \int_0^\infty e^{-st} G(t-\tau)u(\tau)d\tau dt$$

Changing the order of integration and rearranging yields

$$Y(s) = \int_0^\infty \left(\int_0^\infty e^{-s(t-\tau)} G(t-\tau)dt\right) e^{-s\tau} u(\tau)d\tau$$

Substituting $\bar{t} = t - \tau$ into the integral in parentheses, we have $d\bar{t} = dt$ and the lower limit changes to $0 - \tau = -\tau$. Thus,

$$\int_0^\infty e^{-s(t-\tau)} G(t-\tau)dt = \int_{-\tau}^\infty e^{-s(\bar{t})} G(\bar{t})dt$$

However, since the system is assumed to be causal, the integral cannot depend on time before 0 and thus

$$\int_{-\tau}^\infty e^{-s(\bar{t})} G(\bar{t})dt = \int_{-\tau}^0 e^{-s(\bar{t})} G(\bar{t})dt + \int_0^\infty e^{-s(\bar{t})} G(\bar{t})dt = \int_0^\infty e^{-s(\bar{t})} G(\bar{t})dt = G(s)$$

Returning to the expression for $Y(s)$, we now have

$$Y(s) = G(s) \int_0^\infty e^{-s\tau} u(\tau)d\tau = G(s)U(s)$$

Therefore, we see that $G(s)$ relates the transform of the input $u(t)$ and the transform of the output $y(t)$. We call this the *transfer function* of the continuous-time, causal, LTI system (called the *transfer matrix* for a MIMO system).

Transfer function $G(s)$ is simply the Laplace transform of the impulse response

$$G(s) = \int_0^\infty e^{-st} G(t) dt, \quad s \in \mathscr{C}$$

5.11.3 Z-Transform

The (unilateral) Z-transform of a discrete-time signal $y[n]$ is

$$Y(z) = (Z)(y[n]) = \sum_{n=0}^{\infty} z^{-n} y[n]$$

The transfer function $G(z)$ of a discrete-time system can be derived in the same manner as was done above for a continuous-time system.

$$G(z) = \sum_{n=0}^{\infty} z^{-n} G[n]$$

5.11.4 Frequency Domain Analysis

The closed loop transfer function of an open loop system $G(s)$ with feedback $H(s)$ and gain K is

$$H(s) = \frac{KG(s)}{1 + KG(s)H(s)}$$

The denominator $1 + KG(s)H(s)$ is the characteristic polynomial which determines the poles of the closed loop transfer function (the eigenvalues of the closed loop A matrix). Assuming unity feedback ($H(s) = 1$) we have the requirement that

$$1 + KG(s)H(s) > 0$$

for stability. A *pole* is called as such because the magnitude of the transfer function goes to infinity at the these values of s (since they cause the denominator of the tranfer function to go to zero); plotting the magnitude as a function of s results in "poles" sticking out from a surface plot. A *zero* is called as such because at these points the numerator of $G(s)$ (and thus $G(s)$ itself) goes to zero magnitude.

Root Locus

The root locus is a graphical method for determining how the location of the closed loop poles in the complex frequency plane move as the gain K is increased (recalling that $s = \sigma + j\omega$, the root locus plots $\Im(s) = j\omega$ versus $\Re(s) = \sigma$).

As $K \to 0$, the closed loop poles approach the open loop poles (poles of $G(s)$); as $K \to \infty$, the closed loop poles approach the open loop zeros (zeros of $G(s)$). The open loop transfer function is assumed to be a rational function of s of the form

$$G(s) = C\frac{\prod_{i=1}^{m}(s - z_i)}{\prod_{i=1}^{n}(s - p_i)}$$

where m is the number of open loop zeros, n is the number of open loop poles, and constant C can be absorbed into the gain by defining $\bar{K} = KC$.

When $n > m$ ie the order of the numerator is greater than the order of the denominator then there is an excess of poles and thus $(n - m)$ go to infinity rather than going to corresponding open loop zeros.

Nyquist Diagram

The Nyquist diagram is a graphical method for determining the stability of a system. Recalling that $1 + KG(s) = 0$ is the condition for instability, this is equivalent to

$$G(s) = -\frac{1}{K}$$

Thus, a value of s in the right half of the s plane for which $G(s) = -\frac{1}{K}$ indicates that the system is unstable.

The Nyquist diagram is a plot of the $z = G(s)$ plane; every point in the right half of the s plane is mapped to the z plane via the open loop transfer function $G(s)$. If this map encompasses the point $z = -\frac{1}{K}$ then the corresponding value of s must be in the right half of the s plane and the system is unstable.

It is enough to just map the boundary of the right half of the s plane into the z plane; this boundary is the imaginary axis. Thus, the contour in the Nyquist diagram is the imaginary axis.

If the imaginary axis ($z = G(s)$ contour) encircles the point $z = -\frac{1}{K}$ in the clockwise direction, then the system is unstable. If the system has pole(s) on the imaginary axis, then the procedure for constructing the diagram varies.

The Nyquist diagram is a polar plot of $z = G(s)$ for the imaginary axis, ie where $s = j\omega$. The imaginary part of $G(s)$ is plotted against the real part of $G(s)$ with frequency ω as the parameter.

Bode Plot

While the Nyquist diagram plots $\Im(G(s))$ versus $\Re(G(s))$ (parameterized by ω) it is often more useful to plot these two parts versus frequency on separate graphs. The Bode plot then consists of an amplitude plot and a phase plot where

$$G(s = j\omega) = |G(j\omega)|e^{j\theta(\omega)}$$

The amplitude is plotted in decibels (dB) as

$$D(\omega) = 20 \log_{10} |G(j\omega)|$$

and the phase is plotted in degrees.

When $G(s)$ has only real poles and zeros, it can be written in the form

$$G(s) = G_0 \frac{(1 + \frac{s}{z_1}) \cdots (1 + \frac{s}{z_m})}{(1 + \frac{s}{p_1}) \cdots (1 + \frac{s}{p_n})}$$

where $G(s)$ has m open loop zeros and n open loop poles. When $s = 0$ the magnitude equals the DC gain, ie $D(j\omega = 0) = 20 \log_{10} (G_0)$; when $s = j\omega$ it can be shown that $D(\omega)$ factors so that

$$D(\omega) = 20 \log (G_0) + \sum_{i=1}^{m} 10 \log \left[1 + \left(\frac{\omega^2}{z_i} \right) \right] - \sum_{i=1}^{n} 10 \log \left[1 + \left(\frac{\omega^2}{p_i} \right) \right]$$

Similarly for the phase,

$$\theta(\omega) = \sum_{i=1}^{m} \tan^{-1} \left(\frac{\omega}{z_i} \right) - \sum_{i=1}^{n} \tan^{-1} \left(\frac{\omega}{p_i} \right)$$

Thus, the log-magnitude plot for any ω is the sum of the log-magnitude plots of each of the contributing factors; with increasing frequency, a zero "pushes up" both amplitude and phase while a pole "pulls down" both amplitude and phase.

5.11.5 System Type

Consider a simple error-driven feedback control system with $e = y_r - y$ where y_r is the reference input and thus the input to the plant is $u = Ke$.

The transfer function from the reference input to the error is thus

$$H_e(s) = \frac{1}{1 + KG(s)} = \frac{e(s)}{y_r(s)}$$

Note that the *return difference* $1 + KG(s)$ in the denominator should be large in order for the error to be small. The way to do this is obviously by increaing the gain K; of course, since we can't make the gain arbitrarily large, we cannot expect to design a system which tracks a reference with arbitrarily small error. We might not even want to track a reference with very small error because the reference often contains noise.

Consider when the reference is a polynomial function in time

$$y_r = C_1 + C_2 t + \cdots + \frac{C_{m+1}}{m!} t^m$$

We say that the system is of *type* m if it can track such an m^{th} degree polynomial reference input with finite (but nonzero) steady-state error. Such a system can track a

polynomial reference of degree $m-1$ or less with zero error but the error in tracking a polynomial reference of degree $m+1$ or higher becomes infinite.

The steady-state error of the system can be determined using the final value theorem for the Laplace transform:

$$e_{ss} = \lim_{t\to\infty} e(t) = \lim_{s\to 0} se(s)$$

Here we have

$$e(s) = \frac{1}{1+KG(s)} y_r(s)$$

and the Laplace transform of the reference is

$$y_r(s) = \frac{C_1}{s} + \frac{C_2}{s^2} + \cdots + \frac{C_{m+1}}{s^{m+1}} = \frac{C_1 s^m + C_2 s^{m-1} + \cdots + C_{m+1}}{s^{m+1}}$$

Thus, we have

$$se(s) = \frac{1}{1+KG(s)} \frac{C_1 s^m + C_2 s^{m-1} + \cdots + C_{m+1}}{s^{m+1}}$$

The limit as $s \to 0$ of $se(s)$ will be infinite if $G(0)$ is finite because the factor s^m in the denominator of the transformed reference goes to zero. We thus need $G(s) \to \infty$ as $s \to 0$ to cancel this effect, iomplying that we need $G(s)$ to have a pole of *at least* order m at $s = 0$.

If

$$G(s) = \frac{N(s)}{s^p D(s)}$$

where neither $N(s)$ or $D(s)$ have roots at the origin, then

$$se(s) = \frac{1}{1+K\frac{N(s)}{s^p D(s)}} \frac{C_1 s^m + C_2 s^{m-1} + \cdots + C_{m+1}}{s^m} \tag{5.1}$$

$$= \frac{s^{p-m} D(s)}{s^p D(s) + KN(s)} (C_1 s^m + C_2 s^{m-1} + \cdots + C_{m+1}) \tag{5.2}$$

This implies that in the limit as $s \to 0$ we have three possibilities. If $p > m$, the error will be zero. If $p = m$, the error will be finite but nonzero. If $p < m$, the error will be infinite.

The system type is thus determined by the order of the pole at $s = 0$ in the open-loop process (plant); a system of type m or greater is required to track a reference of order m with finite (or zero, for type $m+1$ or greater) steady-state error.

The exact error for reference inputs of different orders given the system type can de derived from the above relations.

5.12 Older Controls Notes

This section contains different notes on many topics already introduced earlier in this chapter. These notes come primarily from the excellent older textbook *Control System Design: An Introduction to State-Space Methods* by Bernard Friedland[1] and will eventually be consolidated with the other notes on the same topics.

5.12.1 Controllability and Observability

Controllability is a measure of how well the internal states of a system can be manipulated.

A system is *controllable* if it is possible to apply a control sequence which takes the state from the initial state $\mathbf{x}(0)$ to any desired final state $\mathbf{x}(t_f)$ in a finite amount of time ($t_f < \infty$).

Observability is a measure of how well the internal states of a system can be inferred from the knowledge of the inputs to the system and its outputs over time.

A state $x_i(t)$ is *observable* if for any time $t_f > 0$ the initial state of the system $x_i(0)$ can be determined from the time history of the input $\mathbf{u}(t)$ and the output $\mathbf{y}(t)$ in the interval $[0, t_f]$. The system is observable if all its internal states are observable.

The definition results from the fact that if the matrices $\mathbf{A}$ and $\mathbf{C}$ of the state space model of the system are know, then to obtain the value of any state $x_i(t)$ at any time t requires only the determination of the corresponding initial state.

Consider the state space model for the linear, time-invariant, discrete-time system given below. The results are analogous for continuous-time systems.

$$\mathbf{x}[k+1] = \mathbf{A}\mathbf{x}[k] \quad \mathbf{y}[k] = \mathbf{C}\mathbf{x}[k]$$

Assuming matrices $\mathbf{A}$ and $\mathbf{C}$ are known, we have

$$\begin{aligned}
\mathbf{y}[0] &= \mathbf{C}\mathbf{x}[0] \\
\mathbf{y}[1] &= \mathbf{C}\mathbf{x}[1] = \mathbf{C}\left(\mathbf{A}\mathbf{x}[0]\right) \\
\mathbf{y}[2] &= \mathbf{C}\mathbf{x}[2] = \mathbf{C}\left(\mathbf{A}^2\mathbf{x}[0]\right) \\
&\vdots \\
\mathbf{y}[n-1] &= \mathbf{C}\mathbf{x}[n-1] = \mathbf{C}\left(\mathbf{A}^{n-1}\mathbf{x}[0]\right)
\end{aligned}$$

We therefore have

$$\mathbf{O}\mathbf{x}[0] = \mathbf{y}$$

where $\mathbf{y}$ is the vector of outputs at times $k = 0$ through $k = n-1$, $\mathbf{x}[0]$ is the vector of initial conditions for all states, and

[1] https://books.google.com/books/about/Control_system_design.html?id=2M1SAAAAMAAJ

$$\mathbf{O}(\mathbf{A},\mathbf{C}) = \begin{pmatrix} \mathbf{C} \\ \mathbf{CA} \\ \mathbf{CA}^2 \\ \vdots \\ \mathbf{CA}^{n-1} \end{pmatrix}$$

is the *observability* matrix. If $\mathbf{O}$ has full rank ($r = n$) then there exists a unique solution for the initial condition vector $\mathbf{x}[0]$ and thus the system is observable.

5.12.2 Design of a Regulator for a Single Input System

In general (for systems with multiple inputs) the design of a regulator entails determining a gain matrix G such that the control input becomes $u = -Gx$, ie linear feedback.

For a single input system described by $\dot{x} = Ax + Bu$, the gain matrix becomes a gain vector $g = [g_1, g_2, \cdots g_k]$ where each gain is associated with one of that states of the k^{th} order system.

Substituting the in the control law yields the closed-loop system

$$\dot{x} = (A - bg)x$$

Thus, the closed-loop dynamics matrix is $A_c = (A - bg)$ and we wish to choose the gains such that this matrix has its eigenvalues equal to the desired set $\{\hat{a_1}, \hat{a_2}, \cdots \hat{a_k}\}$. We could simply expand the characteristic polynomial of A_c - in this case, the coefficients of powers of s would be functions of the gains, allowing us to solve k simultaneous equations for the gains required to produce the desired coefficients.

However, this process is computationally intense. Instead, if the system is given in companion (controller canonical) form

$$\begin{aligned} A &= \begin{pmatrix} -a_1 & -a_1 & \cdots & -a_k \\ 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix} \\ b &= \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \end{pmatrix}^T \end{aligned}$$

then the closed loop dynamics matrix with $g = \begin{pmatrix} g_1 & g_2 & \cdots & g_k \end{pmatrix}$ becomes

$$A_c = (A - bg) = \begin{pmatrix} -a_1 - g_1 & -a_1 - g_1 & \cdots & -a_k - g_k \\ 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix}$$

That is, the gains are simply added to the coefficients of the open-loop matrix A to form the closed-loop matrix A_c. We can therefore simply choose

$$g_i = \hat{a}_i - a$$

For a system which does not have its dynamics represented in this companion form (usually this is the case), we can transform the system using

$$\bar{x} = Tx$$

where T is a linear transformation. Such transformations preserve the dynamics of the original system, ie the characteristic equation (and thus the locations of poles or eigenvalues of A) are unchanged. Then

$$\dot{\bar{x}} = \bar{A}\bar{x} + \bar{b}u$$

where $\bar{A} = TAT^{-1}$ and $\bar{b} = Tb$. Then the gain matrix (vector in this case) of the transformed system is $\bar{g} = \hat{a} - \bar{a} = \hat{a} - a$ since $\bar{a} = a$ due to the fact that the characteristic equation is unchanged by the linear transformation T.

Thus the control law for the original system is

$$u = -gx = -g(T^{-1}\bar{x}) = -\bar{g}\bar{x}$$

and therefore the gain vector for the transformed system is $\bar{g} = gT^{-1}$. Thus, the gain for the original system is $g = T\bar{g} = T(\hat{a} - a)$. To easily choose the gains, we transform the system to canonical form using T, find the gains for the new system, and transform the gains back using T. If T is known then this process is trivial.

The transformation T can be expressed as the product $T = UV$, where U is equal to the inverse of the controllability matrix

$$Q = [b, Ab, \cdots, A^{k-1}b]$$

and V is the inverse of the triangular matrix

$$W = \begin{pmatrix} 1 & a_1 & \cdots & a_{k-1} \\ 0 & 1 & \cdots & a_{k-2} \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}$$

The matrix U brings A into observer canonical form and the matrix V transforms from the observer form into the controller form. The implication is that the system must be controllable in order for Q to have an inverse (otherwise Q is rank-deficient).

The desired gain matrix g can thus be written as

$$g = [(QW)^{-1}](\hat{a} - a)$$

For a system with multiple inputs, $u = -Gx$ where G is the full gain matrix. This means that there are km gains to be specified where m is the number of inputs (dimensionality of u). There are thus more gains than are needed to place all closed-loop poles, allowing the designer the flexibility to place poles and meet other design requirements as well.

Given that there are practical limits to how large the control input can be in a system (either due to additional cost/weight or due to saturation above a certain gain value), there are several things to note about the above formula (called the *Bass-Gura* formula). First, the each gain is proportional to the distance which the corresponding pole is to be moved; moving the pole a shorter distance requires a lower gain and thus a smaller control input. Second, note that the gains are inversely proportional to the controllability matrix - hence, a system which is less controllable requires a higher gain to move a pole the same distance (qualitatively).

In general, one should not try to move the poles further left than are required. Since poles further in the left-hand plane will decay out faster than those which are closer, control inputs will be governed by the former but system response speed will be slowed by the latter (poles further from the origin decay faster). This suggested that one might optimize pole placement by moving all poles to roughly the same distance (real parts) from the origin to make the control effort more efficient. The Butterworth configuration spaces poles evenly about the complex plane at a constant distance from the origin. Another concern is bandwidth - one wants to keep the bandwidth high enough to achieve the desired speed of responce but low enough to avoid exciting high-frequency modes or responding strongly to noise.

NOTE: Residues are the coefficients (in the numerator) in partial fraction expansion of a transfer function.

5.12.3 Linear Observers

For a dynamic system represented in state space form

$$\dot{x} = Ax + Bu$$

we now assume that we cannot directly measure the state of the system; instead, we can only measure the observation vector

$$y = Cx$$

where y is of lower dimension than x. In truth, we say that the matrix C is square but has less than full rank - this means that even though we have the same number of observations as state variables, the outputes we measure are not all independent. We thus cannot invert C to solve for x directly.

It is theoretically possible to use past observation data to find the state known at a past time (using an integral from Ch 5) and integrate, using the system dynamics given above, to find the state at the current time. However, this method of extrapolation

requires computation of a difficult integral and inevitably introduces errors present in the measurement of y.

Instead, we seek to obtain an estimate $\hat{x}$ as follows. Let the estimate be the output of the dynamic system

$$\dot{\hat{x}} = \hat{A}\hat{x} + \hat{B}u + Ky$$

excited by measurement y and input u. We wish to selec $\hat{A}$ and $\hat{B}$ such that the error

$$e = x - \hat{x}$$

is small. This is called *Luenberger's method*. The differential equation governing the time evolution of the error is

$$\begin{aligned} \dot{e} &= \dot{x} - \dot{\hat{x}} = Ax + Bu - \hat{A}(x - e) - \hat{B}u - KCx \\ &= \hat{A}e + (A - KC - \hat{A})x + (B - \hat{B})u \end{aligned}$$

which comes from the previous equations in this section. We desire for the error to go asymptotically to zero, ie we want the system

$$\dot{e} = \hat{A}e$$

with $\hat{A}$ being a stable dynamics matrix, ie a matrix with all negative eigenvalues (poles in the LHP). This requires that

$$\begin{aligned} \hat{A} &= A - KC \\ \hat{B} &= B \end{aligned}$$

This imples that we cannot choose $\hat{A}$, $\hat{B}$ and K arbitrarily; $\hat{B}$ must be the control matrix of the system and the choice of K determines $\hat{A}$. We thus are tasked with choosing K. These restrictions can be written into the dynamic system for the state estimate as

$$\begin{aligned} \dot{\hat{x}} &= (A - KC)\hat{x} + Bu + Ky \\ &= A\hat{x} + Bu + K(y - C\hat{x}) \end{aligned}$$

Thus, the differential equation governing the state estimate is the same as above but with an additional input $K(y - C\hat{x}) = KC(x - \hat{x})$ where $r = y - C\hat{x}$ is called the *residual*. The residual is the difference between the true and estimated outputs and tends to zero when the error is forced to zero.

The observer is thus in the form of a feeback system where the residual takes the role of the error. Determining a matrix K which makes closed-loop dynamics matrix $\hat{A} = (A - KC)$ have only negative eigenvalues is analagous to the task of determining

the gain matrix G which shaped the dynamic response in the previous section. As long as the observability matrix

$$N = [C^T, A^T C^T, \cdots (A^T)^{k-1} C^T]$$

has full rank ($r = k$) then the eigenvalues of $\hat{A}$ can be placed at any locations desired.

For a system with a single output, we have $\hat{A} = A - kc^T$ where $c^T = \begin{pmatrix} c_1 & c_2 & \cdots & c_k \end{pmatrix}$. If we take the transpose of this matrix we get

$$\hat{A}^T = A^T - ck^T$$

which has the same form as the closed-loop matrix $A_c = A - bg^T$ from the single-input full-state (all states known) feedback problem of the previous section. Determining the gain vector k is the same problem as determining the gain vector g from the feedback problem.

Using the Bass-Gura formula again, we find that

$$k = [(NW)^T]^{-1}(\hat{a} - a)$$

where N is the observability matrix (given above) and W is again the triangular matrix

$$W = \begin{pmatrix} 1 & a_1 & \cdots & a_{k-1} \\ 0 & 1 & \cdots & a_{k-2} \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}$$

Note again that the presence of multiple outputs just permits flexibility in design (place the eigenvalues and shape system response attributes).

Reduced Order Observers

When there is one output for every state variable (the observation equation is $y = Cx$) then the state vector can be computed simply as $x = C^{-1}y$. In this case an observer is not needed. But is a k^{th} order observer needed (for a k^{th} order system) when only some fraction of the state vector is not measureable? It makes sense that an observer of lower order is sufficient in this case.

Consider a state vector x composed of two smaller state vectors - x_1 which can be measured directly and x_2 which cannot. That is,

$$x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

The two systems corresponding to these state vectors are

$$\dot{x_1} = A_{11}x_1 + A_{12}x_2 + B_1u$$
$$\dot{x_2} = A_{21}x_1 + A_{22}x_2 + B_2u$$

and the observation is simply given by $y = C_1x_1$ since only x_1 can be measured. The "standard observer" for each of these variables is then

$$\dot{\hat{x_1}} = A_{11}\hat{x_1} + A_{12}\hat{x_2} + B_1u + K_1(y - C_1\hat{x_1})$$
$$\dot{\hat{x_2}} = A_{21}\hat{x_1} + A_{22}\hat{x_2} + B_2u + K_2(y - C_1\hat{x_1})$$

Of course, there's no reason to keep the first observer since $x_1 = \hat{x_1} = C_1^{-1}y$ can be measured directly. Using this fact, the second observer becomes

$$\dot{\hat{x_2}} = A_{21}C_1^{-1}y + A_{22}\hat{x_2} + B_2u$$

The dynamic behavior of this observer is then governed by the eigenvalues of the open-loop dynamics matrix A_{22} which the control system design cannot influence (it is a submatrix of A and is part of the plant model). If the eigenvalues of this matrix happen to be negative then this observer could work fine, but if they are not then the observer must be formulated in a more general form.

Consider forming the system

$$\hat{x_2} = Ly + z$$
$$\dot{z} = Fz + Py + Hu$$

where z is the state of a $(k-l)^{th}$ order system. Note that the letters chosen for these matrices have no special meaning. *NOTE TO SELF: why use this form of system?? Where does this come from?*

As for the full-order observer, the error is $e = x - \hat{x}$ which is here

$$e = \begin{pmatrix} e_1 \\ e_2 \end{pmatrix} = \begin{pmatrix} x_1 - \hat{x_1} \\ x_2 - \hat{x_2} \end{pmatrix}$$

but of course $e_1 = 0$ because the states comprising x_1 can be measured directly. We therefore consider only e_2, the differential equation for which is

$$\dot{e_2} = \dot{x_2} - \dot{\hat{x_2}} = A_{21}x_1 + A_{22}x_2 + B_2u - L\dot{y} - \dot{z}$$

Using the formula for $\dot{z}$ and the fact that $\dot{y} = C_1x_1 = C_1[A_{11}x_1 + A_{12}x_2 + B_1u]$ yields

$$\dot{e_2} = A_{21}x_1 + A_{22}x_2 + B_2u - LC_1[A_{11}x_1 + A_{12}x_2 + B_1u] - Fz - Py - Hu$$

To eliminate the remaining z (which has no significance in the original system) and y we recognize that $z = \hat{x_2} - Ly = (x_2 - e_2) - LC_1x_1$ and $y = C_1x_1$. The above equation then becomes

$$\dot{e_2} = A_{21}x_1 + A_{22}x_2 + B_2u - LC_1[A_{11}x_1 + A_{12}x_2 + B_1u] - F[x_2 - e_2 - LC_1x_1] - PC_1x_1 - Hu$$

Finally, simplifying the above expression be grouping terms multiplying e_2, x_1, x_2, and u yields

$$\dot{e_2} = Fe_2 + (A_{21} - LC_1A_{11} - PC_1 + FLC_1)x_1 + (A_{22} - LC_1A_{12} - F)x_2 + (B_2 - LC_1B_1 - H)u$$

In order for the observation error to go asymptotically to zero for all x_1, x_2 and u we must require that the matrices multiplying these vectors be zero (we want the time evolution of the error to be independent of the value of the states and the control input). This requires

$$\begin{aligned} F &= A_{22} - LC_1A_{12} \\ H &= B_2 - LC_1B_1 \\ PC_1 &= A_{21} - LC_1A_{11} + FLC_1 \end{aligned}$$

What remains is the differential equation $\dot{e_2} = Fe_2$ and thus the eigenvalues of F must be negative in order for the error differential equation to be asymptotically stable.

We therefore must choose L such that the poles of $F = A_{22} - LC_1A_{12}$ are in the LHP. This is the same problem as choosing the matrix K such that the poles of $\hat{A} = A - KC$ are stable for the full-order linear observer; here we have $A = A_{22}$ and $C = C_1A_{12}$. Again, it is only possible to choose such a matrix L if the observability matrix

$$N = [A_{12}^T C_1^T, A_{22}^T A_{12}^T C_1^T, \cdots (A_{22}^T)^{k-l-1} A_{12}^T C_1^T]$$

is of (full) rank $k - l$. Once L has been selected, the matrices H and P can easily be solved for from the equations given above.

Note that if the state vector cannot be defined as the concatenation of two subvectors x_1 and x_2 as was done above then a more general reduced-order observer is needed. In practice, the state vector can typically be broken into subvectors in this manner.

5.12.4 Disturbances and Tracking (Exogeneous Inputs)

Assume that a state-space system is in the form

$$\dot{x} = Ax + Bu + Fx_d$$

where x_d is a disturbance vector (which we may or may not be able to measure directly).

We are also going to require the system to track a reference state x_r. These two vectors are assumed to be governed by the known differential equations

$$\begin{aligned} \dot{x_d} &= A_d x_d \\ \dot{x_r} &= A_r x_r \end{aligned}$$

These are clearly unforced differential equations ie are not subject to control by the designer (the full system is necessarily uncontrollable).

We are concerned with the error in tracking a reference state $e = x - x_r$ for which the governing differential equation (from the equations given above) is

$$\begin{aligned} \dot{e} &= \dot{x} - \dot{x_r} = A(e + x_r) + Fx_d + Bu - A_r x_r \quad (5.3) \\ &= Ae + (A - A_r)x_r + Fx_d + Bu \quad (5.4) \end{aligned}$$

Defining x_0 to be the vector composed of the exogeneous inputs

$$x_0 = \begin{pmatrix} x_r \\ x_d \end{pmatrix}$$

and E to be the matrix composed as

$$E = \begin{pmatrix} A - A_r & F \end{pmatrix}$$

we can write

$$\dot{e} = Ae + Ex_0 + Bu$$

Further, if we compose the *metastate*

$$x = \begin{pmatrix} x_o \\ e \end{pmatrix}$$

then this satisfies the *metastate equation* $\dot{x} = Ax + Bu$ where

$$A = \begin{pmatrix} A & E \\ 0 & A_0 \end{pmatrix} \qquad A_0 = \begin{pmatrix} A_r & 0 \\ 0 & A_d \end{pmatrix} \qquad B = \begin{pmatrix} B \\ 0 \end{pmatrix}$$

The output (observation) equation is, in general,

$$y = Cx = C_e e + C_r x_r + C_d x_d$$

However, it should be noted that sometimes only the error can be measured; in this case $C = \begin{pmatrix} C & 0 & 0 \end{pmatrix}$. Note that the reference input x_r appears in the error differential equation through the term $(A - A_r)x_r$. This implies that if the reference can be produced from the unforced (homogeneous) solutions to the open-loop system (in this case $A = A_r$)

then it does not contribute to the error. Unless the dynamics matrix A has repeated eigenvalues at the origin (and thus non-exponential solutions) then it is impossible to track anything else (steps, ramps, etc). We already expect this as we know that the system must have a pole at the origin of order m to track polynomial inputs of order $m-1$ or less with zero error.

Of course, since the metastate is inherently uncontrollable (the disturbance and reference systems are unforced) we cannot use the pole placement technique of previous sections.

Instead, we work directly with the error differential equation $\dot{e} = Ae + Bu + Ex_0$. We essentially consider the exogeneous vector x_0 to be an input just like u; the idea is to design u such that it counteracts the effects of the exogeneous inputs.

Consider a linear control law

$$u = -Ge - G_0 x_0 = -Ge - G_r x_r - G_d x_d$$

Here we will assume in our design that the exogeneous vector and the system error are both accessible for measurement during operation, which may not always be the case. The gain matrices involved in our control law can be chosen independent of the availability of these measurements.

The error diffiential equation given above becomes

$$\dot{e} = Ae + Ex_0 - B(Ge + G_0 x_0)$$

which is essentially a linear system excited by the homogeneous input x_0.

It is impossible to choose the gain matrices G and G_0 to keep the system error zero for all x_0 and e. Instead, we require tha:

1) The closed-loop system should be asymptotically-stable. 2) A linear combination of the error state variables should be zero in the steady state (rather than the whole state vector going to zero).

In order to satisfy the first condition, we need $A_c = A - BG$ (which si the closed-loop dynamics matrix for the error differential equation) to have negative eigenvalues (poles in the LHP). This can be accomplished by choosing G to shift the roots in the same manner as was done in previous sections.

The second condition is tougher to satisfy. Since steady state implies $\dot{e} = 0$ we have

$$(A - BG)e = (BG_0 - E)x_0$$

The closed-loop dynamics matrix is necessarily stable (from the first condition) and so its inverse must exist (no roots at the origin). Thus, the steady state error is given by

$$e = (A - BG)^{-1}(BG_0 - E)x_0$$

This error, as was noted, will not be zero. Instead, we required that $y = Ce = 0$ where C is a singular matrix of proper dimension (C must be singular or this equation would force $e = 0$ which is not possible). Thus, we require that

$$C(A - BG)^{-1}(BG_0 - E)x_0 = 0$$

However, since we want this to hold for every x_0, we require that

$$C(A - BG)^{-1}(BG_0 - E) = 0$$

ie this term must be the matrix of zeros in order for every x_0 to lie in its nullspace. We can write the above equation as

$$C(A - BG)^{-1}BG_0 = C(A - BG)^{-1}E$$

Consider solving the left hand side of this equation for G_0. If y has dimension j then C is a $j \times k$ matrix; assume $(A - BG)^{-1})$ is a $k \times k$ matrix and B is a $k \times m$ matrix (where m is the number of control variables). Then the matrix multiplying G_0 has size $j \times m$ and we have three cases.

If $j > m$ then the columns of the matrix span only a subspace of dimension j and thus there is no solution (unless the right hand side happens to lie in this subspace which is a very special case).

If $j < m$ then the columns of the matrix must be dependent and there are thus infinite solutions to the underdetermined system. This is not a problem, as it allows for a choice of G_0 which may satisfy additional design criteria.

If $j = m$ and the matrix is nonsingular then there is exactly one solution for G_0. In this case we have

$$G_0 = [C(A - BG)^{-1}B]^{-1}C(A - BG)^{-1}E$$

It can be shown that $C(A - BG)^{-1}B$ is invertible if and only if

$$\lim_{s \to 0} H_0(s) = |C(sI - A)^{-1}B| \neq 0$$

Note that the invertibility of this matrix does not depend on the gain matrix G; indeed, we can choose any G which makes the closed-loop dynamics matrix of the error differential equation asymptotically stable without affecting the invertibility of this matrix. Of course, the choice of G affects the computed G_0.

Note that x_r does not need to have all its components specified in most cases - we may wish to track references for only some of the state variables. In such a case, the component of x_0 corresponding to the reference will be of lower dimension than k and the corresponding submatrix of E will shrink.

Measuring the Error and Exogeneous Inputs

As mentioned above, there may very well be cases in which we cannot directly measure the error and/or exogeneous inputs; such cases require the addition of observers which

extimate these variables. Without this information we obviously cannot implement the control law developed above.

We now assume that the entire metastate x (composed of the error and exogeneous inputs) cannot be directly measured. If only part of the metastate cannot be measured then we can implement a *reduced-order* observer, as detailed later in this section. We thus assume that the observation vector depends on both the error and the exogeneous inputs:

$$y = Ce + Dx_0 = \tilde{C}x$$

where

$$\tilde{C} = \begin{pmatrix} C & D \end{pmatrix}$$

Applying the general observer equation (seen earlier) to the metasystem yields

$$\dot{\hat{x}} = A\hat{x} + Bu + K(y - \tilde{C}\hat{x})$$

which can be separated into two coupled equations - one governing the error and the other governing the exogeneous input.

$$\begin{aligned} \dot{\hat{e}} &= A\hat{e} + Bu + E\hat{x_0} + K_e(y - C\hat{e} - D\hat{x_0}) \\ \dot{\hat{x_0}} &= A_0\hat{x_0} + K_0(y - C\hat{e} - D\hat{x_0}) \end{aligned}$$

Note that the closed-loop dynamics matrix of the metasystem is

$$\hat{A} = A - K\tilde{C} = \begin{pmatrix} A - K_cC & E - K_eD \\ -K_0C & A_0 - K_0D \end{pmatrix}$$

If the metasystem is observable then the poles can be moved to arbitrary locations via the choice of K.

5.12.5 Compensator Design

Consider yet again the dynamic process

$$\dot{x} = Ax + Bu$$

with observation vector $y = Cx$ and suppose that we have already designed a full-state feedback control law of the form

$$u = -Gx$$

as detailed in a previous section. Also suppose that we have designed a full-state linear observer

$$\dot{\hat{x}} = A\hat{x} + Bu + K(y - C\hat{x})$$

How do we combine the controller and observer into a compensator? The *separation principle* says that we should use the control law

$$u = -G\hat{x}$$

That is, the control law should simply use the previously-designed G with the estimate $\hat{x}$ from the previously-designed observer. The compensator is therefore a combination of a controller and an observer; the full system is the combination of this compensator and the original plant. We are thus dealing with a system of order $2k$ (k state variables in the plant and another k from the state estimates in the compensator).

Applying the control law dictated by the separation principle, we have

$$\dot{x} = Ax - BG\hat{x}$$

The equation for the observer becomes

$$\dot{\hat{x}} = A\hat{x} - BG\hat{x} + KC(x - \hat{x})$$

Again, defining the observer error to be $e = x - \hat{x}$ we have

$$\begin{aligned}\dot{e} &= \dot{x} - \dot{\hat{x}} \\ &= Ax - BG\hat{x} - [A\hat{x} - BG\hat{x} + KC(x - \hat{x})] \\ &= (A - KC)e\end{aligned}$$

and since $\hat{x} = x - e$,

$$\begin{aligned}\dot{x} &= Ax - BG\hat{x} \\ &= Ax - BG(x - e) \\ &= (A - BG)x + BGe\end{aligned}$$

The system of order $2k$ may therefore be defined by a state composed of e and x and governed by the differential equations

$$\begin{aligned}\dot{e} &= (A - KC)e \\ \dot{x} &= (A - BG)x + BGe\end{aligned}$$

The first equation generates the estimation error e and the evolution of the state is then driven by this error. In order for this combined system to be stable, the eigenvalues of both $\hat{A} = (A - KC)$ and $A_c = (A - BG)$ must be negative. These are the closed-loop matrices of the full-state observer and the full-state controller, respectively; designing each of these individually ensures that the overall system will be stable.

Similarly, a compensator can be designed using a reduced-order observer following the results of the previous section.

5.12.6 Optimal Control

Consider the dynamic process characterized by, as usual,

$$\dot{x} = Ax + Bu$$

where we seek a linear control law

$$u(t) = -Gx(t)$$

Instead of choosing the matrix G such that the closed loop poles are moved to desired locations, we determine the gain matrix which minimizes the cost function

$$V = \int_t^T \left[x^T(\tau)Q(\tau)x(\tau) + u^T(\tau)Ru(\tau)\right] d\tau$$

where Q and R are symmetric matrices, the lower integrand t is the *present* time and the upper integrand T is the *terminal* time (the time different $T - t$ is called the control interval or "time-to-go"). This cost function (or *performance criterion*) is thus the integral of a quadratic form in the state x plus a quadratic form in the control u.

The minimization of such an integral is rarely the "true" goal of the control system; however, the true design objectives often cannot be expressed mathematically or cannot be solved easily even if they can be expressed precisely. On the other hand, if the problem is simplified so that it becomes easy to express and solve then the design may not even meet the desired characteristics. In such a case, optimal control thus offers a practical compromise between a difficult problem and an artificially simplified problem.

The first of the two quadratic forms represents a penalty on the deviation of the state from the origin and the second represents the "cost of control" (a penalty on using control). Note that this means the desired state is the origin! Consideration of optimal control for a system in which the desired state is nonzero (ie a reference to be tracked) will be considered later.

The matrix Q essentially specifies the importance of the components of the state vector *relative* to one another. The matrix R specifies the cost of the various components of the control vector. The reason for including this term is to limit the amount of control which is used since, in reality, actuators providing the control cannot provide arbitrarily large control signals. In theory one might think that as much control as possible should be used - this will push the closed loop poles furthest from the imaginary axis and thus yield the fastest response time. Of course, one must consider the cost of control (ie size, weight, energy) as well as the fact that saturation may occur. if the control signal saturates, the system will exhibit unexpected behavior because the designer placed the poles assuming no saturation. Matrix R is thus typically selected large enough that saturation is avoided.

When the control law above is used to control the standard dynamic process given above, we of course have

$$\dot{x} = Ax - BGx = A_c(t)x$$

where $A_c = A - BG$ is the closed loop dynamics matrix which we will here allow to vary with time (thus A,B and G are not restricted to be constant matrices). Since the dynamics matrix is non-constant, we cannot write the solution in terms of the matrix exponential; instead, the solution to this differential equation can be written

$$x(\tau) = \Phi_c(\tau, t)x(t)$$

where Φ_c is the state-transition matric corresponding to A_c. This equation simply says that the state at any time τ depends linearly on the state at any other - past or future (or even present, in which case Φ_c is defined to equal the identity) - time t. Thankfully, since no simple expression is available for the state-transition matrix (in general), it will be needed in the following derivation.

Substituting the linear feedback control law into the performace criterion integral yields

$$\begin{aligned} V &= \int_t^T \left[x^T(\tau)Q(\tau)x(\tau) + x^T(\tau)G^T(\tau)RG(\tau)x(\tau)\right] d\tau \\ &= \int_t^T \left[x^T(t)\Phi_c(\tau, t)\left\{Q + G^T RG\right\}\Phi_c(\tau, t)x(t)\right] d\tau \end{aligned}$$

Since the initial state $x(t)$ does not depend on time, we can move it outside the integral to yield

$$V = x^T(t)M(t, T)x(t)$$

where M is a symmetric matrix defined to be

$$M(t, T) = \int_t^T \Phi_c(\tau, t)\{Q + G^T RG\}\Phi_c(\tau, t)d\tau$$

The optimum gain matrix G thus minimizes this integral. We wish to obtain a differential equation to which M is the solution. Noting that V is a function of the initial time t we can write

$$V(t) = \int_t^T x^T(\tau)L(\tau)x(\tau)d\tau$$

where $L = Q + G^T RG$. We now wish to find the derivative $\frac{dV}{dt}$ of the integral $V(t)$, recalling that the first part of the fundamental theorem of calculus states that for all x in (a, b)

$$\frac{d}{dt}\int_a^x f(t)dt = f(x)$$

By first switching the upper and lower limits (which negates the integral) and then applying this theorem to $V(t)$ we find that

$$\frac{dV}{dt} = -x^T(t)L(t)x(t)$$

However, we can obtain another expression for the derivative of V from $V = x^T(t)M(t,T)x(t)$. This yields

$$\frac{dV}{dt} = \dot{x}^T(t)M(t,T)x(t) + x^T(t)M(\dot{t},T)x(t) + x^T(t)M(t,T)\dot{x}(t)$$

Since $\dot{x}(t) = A_c x(t)$ this simplifies to

$$\frac{dV}{dt} = x^T(t)\left[A_c^T(t)(t)M(t,T) + M(\dot{t},T) + M(t,T)A_c(t)\right]\dot{x}(t)$$

Comparing expressions for $\frac{dV}{dt}$ leads directly to the fact that

$$-L = A_c^T M + \dot{M} + MA_c$$

where it should be noted that these matrices are functions of time (the notation has been omitted here for simplicity). The differential equation for which M is the solution is

$$-\dot{M} = MA_c + A_c^T M + L$$

where again $L = Q + G^T RG$. We need one initial condition in order to specify the solution to the above differential equation completely; since we already know that the solution is

$$M(t,T) = \int_t^T \Phi_c(\tau,t)L(t)\Phi_c(\tau,t)d\tau$$

then clearly $M(T,T) = 0$ is the required condition.

We are now tasked with finding the gain matrix G which minimizes the cost function. For *any* choice of G (optimal or no) the closed-loop performance is given by

$$V(t) = x^T(t)M(t,T)x(t)$$

where $M(t,T)$ is the solution to

$$\begin{aligned} -\dot{M} &= M(A-BG) + (A-BG)^T M + (Q + G^t RG) \\ &= M(A-BG) + (A^T - G^T B^T)M + Q + G^t RG \end{aligned}$$

We now wish to find the G which minimizes M. By this we mean that we wish to find the optimal soluion $\hat{M}$ for which the quadratic form

$$\hat{V} = x^T \hat{M} x < x^T M x$$

for arbitrary an initial state $x(t)$ and for all $M \neq \hat{M}$.

This optimal solution $\hat{M}$ and corresponding optimal gain $\hat{G}$ must satisfy

$$-\dot{\hat{M}} = \hat{M}(A - B\hat{G}) + (A^T - \hat{G}^T B^T)\hat{M} + Q + \hat{G}^T R\hat{G}$$

Any nonoptimal solution and gain can be expressed in terms of the optimum solution and gain as

$$M = \hat{M} + N$$
$$G = \hat{G} + Z$$

The differential equation satisfied by the nonoptimal variables is thus

$$-(\dot{\hat{M}}+\dot{N})+ = (\hat{M}+N)[A-B(\hat{G}+Z)]+[A^T-(\hat{G}^T+Z^T)B^T](\hat{M}+N)+Q+(\hat{G}^T+Z^T)R(\hat{G}+Z)$$

Subtracting the differential equation corresponding to the optimal solution from that corresponding to the non-optimal solution yields the followsing differential equation for N.

$$-\dot{N} = NA_c + A_c^T N + (\hat{G}^T R - \hat{M}B)Z + Z^T(R\hat{G} - B^T\hat{M}) + Z^T RZ$$

where $A_c = A - BG = A - B(\hat{G} - Z)$. This equation has the same form as

$$-\dot{M} = MA_c + A_c^T M + L$$

where in this case

$$L = (\hat{G}^T R - \hat{M}B)Z + Z^T(R\hat{G} - B^T\hat{M}) + Z^T RZ$$

The solution for N is thus of the form

$$N(t,T) = \int_t^T \Phi_c^T(\tau,t)L\Phi_c(\tau,t)d\tau$$

If $\hat{V}$ is minimized then we must have

$$x^T\hat{M}x \leq x^T(M+N)x = x^T\hat{M}x + x^T Nx$$

This implies that the N must be positive (semi) definite. Based on the integral for N above, in order for the quadratic form $x^T Nx$ to be positive L must be positive (semi) definite since for any positive-semidefinite M and invertible Q we know that $Q^T MQ$ is also positive-semidefinite. Looking at the expression for L in differential equation for N, we see that for sufficiently-small Z the linear terms dominate the quadratic term and it's theoretically possible to choose Z such that L is negative definite. The only way to avoid this is for the linear terms to dissappear and thus

$$R\hat{G} - B^T\hat{M} = 0$$

which leads to the optimum gain

$$\hat{G} = R^{-1}B^T\hat{M}$$

Thus the differential equation for M becomes

$$-\dot{\hat{M}} = \hat{M}A + A^T\hat{M} - \hat{M}BR^{-1}B^T\hat{M} + Q$$

This matrix differential equation is known as the *Riccati equation.* It is the sum of linear terms and a quadratic term; because of the quadratic term, there is no general formula for the solution. In most cases, $\hat{M}(t,T)$ is solved for analytically. This is often done by numerically integrating the differential equation backward in time (since the condition to be satisfied $\hat{M}(T,T) = 0$ is at the terminal time). There are $k(k+1)/2$ coupled, scalar equations which must be integrated because $\hat{M}$ is symmetric.

5.13 Nonlinear Systems

The dynamic system

$$\dot{x} = f(x,u)y = g(x,u)$$

is said to be nonlinear when the functions f and g are nonlinear in the state x and control input u. State space representation and corresponding control theory is no longer valid for such a system; however, it is possible to analyze and control such a system by using a linear approximation.

5.13.1 Linearization around an Equilibrium Point

Consider the linearization of the above nonlinear dynamic system about an equilibrium point (x_0, u_0). Such an equilibrium point is defined as a point where $f(x_0, u_0) = 0$ or in other words where the derivative $\dot{x}$ vanishes.

Such an equilibrium point is either an attractor or repeller - that is, small deviations from this point either cause the system to fall back to equilibrium or diverge from this point. One can thus determine whether such a point is considered to be stable or unstable; this is done by investigating the derivatives of x.

Assuming a stable equilibrium point is found, we can safely linearize the system around it because small deviations will pull it back to this point. Consider then the case in which we apply an input $u(t) = u_0 + \delta u(t)$ which is perturbed from the equilibrium point. Additionally, the initial condition is here $x(0) = x_0 + \delta x_0$. The corresponding output of the system will then be close to $y = g(x_0, u_0)$ but not quite equal to this value. How much do the state $x(t)$ and the ouput $y(t)$ then differ due to these perturbations?

We define $\delta x(t) = x(t) - x_0$ and $\delta y(t) = y(t) - y_0$. Thus, we have

$$\delta y = g(x,u) - y_0 = g(x_0 + \delta x, u_0 + \delta u) - g(x_0, u_0)$$

The Taylor Series expansion of g about the equilibrium point (x_0, u_0) is, to first order,

$$g(x,u) \approx g(x_0,u_0) + \frac{\partial g(x_0,u_0)}{\partial x}\delta x + \frac{\partial g(x_0,u_0)}{\partial u}\delta u$$

Thus, we have

$$\delta y = \frac{\partial g(x_0,u_0)}{\partial x}\delta x + \frac{\partial g(x_0,u_0)}{\partial u}\delta u$$

The time evolution of δx is determined by taking its time derivative. Thus,

$$\delta \dot{x} = \frac{d}{dt}(x(t) - x_0) = \dot{x} = f(x,u) = f(x_0 + \delta x, u_0 + \delta u)$$

Similarly, we can expand f about (x_0, u_0) to yield

$$f(x,u) \approx f(x_0,u_0) + \frac{\partial f(x_0,u_0)}{\partial x}\delta x + \frac{\partial f(x_0,u_0)}{\partial u}\delta u$$

We thus have an linear system defined by

$$\delta \dot{x} = A\delta x + B\delta u \delta y = C\delta x + D\delta u$$

where the usual system matrices A, B, C, and D are the Jacobian matrices defined by

$$A = \frac{\partial f(x_0,u_0)}{\partial x} B = \frac{\partial f(x_0,u_0)}{\partial u} \qquad C = \frac{\partial g(x_0,u_0)}{\partial x} D = \frac{\partial g(x_0,u_0)}{\partial u}$$

This completes our local linearization around the stable equilibrium point (x_0, u_0).

We can also linearize about a desired trajectory $(x_{des}(t), u_{des}(t))$ in the same manner; in this case, the system generally becomes LTV with $\{A(t), B(t), C(t), D(t)\}$ because the partial derivatives given above must be evaluated at each timestep. There are however certain systems for which linearization around certain trajectories results in an LTI system.

5.13.2 Nonlinear Observability

Recall that a linear system is said to be *observable* if there exists a finite $t_f > t_0$ such that for any initial state $x(t_0)$, knowledge of the input $u(t)$ and the measured output $y(t)$ over the interval $[t_0, t_f]$ is sufficient to determine $x(t)$ on the interval.

A linear (in general, time-varying) system is *observable* on the interval $[t_0, t_f]$ if and only if the *observability Grammian*

$$W_o(t_0,t_f) = \int_{t_0}^{t_f} \Phi^T(\tau,t_0)C^T(\tau)C(\tau)\Phi(\tau,t_0)d\tau$$

is nonsingular. However, for a linear, time-invariant (LTI) system it can be shown that the condition for observability is

$$\text{rank } O(A,C) = \text{rank} \begin{bmatrix} C \\ CA \\ CA^2 \\ \cdots \\ C^{n-1}A \end{bmatrix} = n$$

where $O(A,C)$ is called the *observability matrix*. If this condition does not hold, then the unobservable subspace is given by the nullspace of $O(A,C)$.

For nonlinear systems, observability isn't as well-defined. Essentially, we want to know how much the output $y = h(x)$ changes with a change in the state $\dot{x} = f(x)$. Equivalently, we want to know the derivative of vector field $h(x)$ along the flow of vector field $f(x)$. We can write:

$$\begin{aligned} y &= h(x) \\ \dot{y} &= \frac{dh(x)}{dx}\frac{dx}{dt} = \mathscr{L}_1(x) \\ \ddot{y} &= \frac{d\mathscr{L}_1(x)}{dx}\frac{dx}{dt} = \mathscr{L}_2(x) \end{aligned}$$

and so on where $\mathscr{L}_i$ is called the i^{th} Lie derivative of $h(x)$. We can then write:

$$\begin{aligned} \frac{\partial}{\partial t}y &= \frac{\partial h}{\partial x}\frac{\partial x}{\partial t} = \nabla h \frac{\partial x}{\partial t} \\ \frac{\partial}{\partial t}\dot{y} &= \frac{\partial \mathscr{L}_1}{\partial x}\frac{\partial x}{\partial t} = \nabla \mathscr{L}_1 \frac{\partial x}{\partial t} \\ \frac{\partial}{\partial t}\ddot{y} &= \frac{\partial \mathscr{L}_2}{\partial x}\frac{\partial x}{\partial t} = \nabla \mathscr{L}_2 \frac{\partial x}{\partial t} \end{aligned}$$

and so on. Now, "multiply" both sides by ∂t to get

$$\begin{pmatrix} \partial y \\ \partial \dot{y} \\ \partial \ddot{y} \\ \vdots \end{pmatrix} = \begin{pmatrix} \nabla h \\ \nabla \mathscr{L}_1 \\ \nabla \mathscr{L}_2 \\ \vdots \end{pmatrix} \partial x$$

where we define the nonlinear observability matrix to be

$$\mathscr{O} = \begin{pmatrix} \nabla h \\ \nabla \mathscr{L}_1 \\ \nabla \mathscr{L}_2 \\ \vdots \end{pmatrix}$$

So $\mathscr{O}$ relates a perturbation in the state to changes in the output and its derivatives.

We investigate the observability of an estimator having nonlinear prediction and/or measurement models by forming the nonlinear observability matrix[2]. As in the linear case, the state is observable if O has full rank; unobservable state combinations are parameterized by the nullspace of O. Given a system with nonlinear process model $\dot{x} = f(x)$ and measurement model $y = h(x)$,

$$O = \begin{bmatrix} \nabla h(x) \\ \nabla(\nabla h \bullet f(x)) \\ \nabla(\nabla(\nabla h \bullet f(x)) \bullet f(x)) \\ \vdots \end{bmatrix}$$

where ∇ denotes the Jacobian with respect to x. In the linear case, $\nabla h = C$ and $f(x) = Ax$ so $\nabla(\nabla h \bullet f(x)) = \nabla(CAx) = CA$, $\nabla(\nabla(\nabla h \bullet f(x)) \bullet f(x)) = CA^2$ and so on. Unlike in the linear case, there is no condition limiting the size of O; however, only a finite number of derivatives usually need be taken before successive rows become zero (and thus no longer affect the rank). Essentially, O illustrates that states are observable because they are measured directly or because they appear in the dynamics (of some order) of a state which is measured directly. Note that since O is state-dependent in general, this procedure investigates *local* observability.

5.14 System Modeling

System modeling results in differential *equations of motion* and can be done either by using Newton's second law (summing forces/torques to directly produce differential equations) or by using Lagrange's equation (which instead requires a description of the potential and kinetic energy of the system).

Example: Inverted Pendulum on a Cart

The cart is modeled as a point mass with mass M; the pendulum has length $2l$ and is fixed to the center of mass of the cart, which is the location of the origin.

The kinetic energy of the cart is simply

$$T_c = \frac{1}{2} M \dot{x}^2$$

The kinetic energy of the pendulum as represented in the chosen cartesian coordinates is a function of the angle θ. We know that the center of mass of the pendulum is located at a distance l from the center of mass of the cart (and thus the origin) so its position in cartesian coordinates is given by

$$\begin{aligned} x_p &= x + l \sin\theta \\ y_p &= l \cos\theta \end{aligned}$$

The kinetic energy of the pendulum is then

[2] https://www.math.ucdavis.edu/~krener/1-25/10.IEEETAC77.pdf

$$T_p = \frac{1}{2}m\left[\dot{x_p}^2 + \dot{y_p}^2\right]$$

where

$$\begin{aligned}\dot{x_p}^2 &= (\dot{x} + l\dot{\theta}\cos\theta)^2 = \dot{x}^2 + 2l\dot{x}\dot{\theta}\cos\theta + l^2\dot{\theta}^2\cos^2\theta \\ \dot{y_p}^2 &= (l\dot{\theta}\sin\theta)^2 = l^2\dot{\theta}^2\sin^2\theta\end{aligned}$$

Thus,

$$T_p = \frac{1}{2}m\left[\dot{x}^2 + 2l\dot{x}\dot{\theta}\cos\theta + l^2\dot{\theta}^2(\sin^2\theta + \cos^2\theta)\right]$$

Finally, using the fact that $\sin^2\theta + \cos^2\theta = 1$, the total kinetic energy of the system is

$$T = T_c + T_p = \frac{1}{2}(M+m)\dot{x}^2 + ml\dot{x}\dot{\theta}\cos\theta + \frac{1}{2}ml^2\dot{\theta}^2$$

The only potential energy in the system is the gravitational potential energy of the pendulum's center of mass (taking $y = 0$ to be the reference ie where $U = 0$). Thus,

$$U = mgl\cos\theta$$

The Lagrangian can then be formed as

$$L = T - U = \frac{1}{2}(M+m)\dot{x}^2 + ml\dot{x}\dot{\theta}\cos\theta + \frac{1}{2}ml^2\dot{\theta}^2 - mgl\cos\theta$$

Lagrange's equations for this system are then

$$\begin{aligned}\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{x}}\right) - \frac{\partial L}{\partial x} &= F - b\dot{x} \\ \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}}\right) - \frac{\partial L}{\partial \theta} &= 0\end{aligned}$$

Where the partial derivatives of L as as follows.

$$\begin{aligned}\frac{\partial L}{\partial \dot{x}} &= (M+m)\dot{x} + ml\dot{\theta}\cos\theta \\ \frac{\partial L}{\partial x} &= 0 \\ \frac{\partial L}{\partial \dot{\theta}} &= ml\dot{x}\cos\theta + ml^2\dot{\theta} \\ \frac{\partial L}{\partial \theta} &= ml\dot{x}\dot{\theta}\sin\theta + mgl\sin\theta\end{aligned}$$

The first of Lagrange's equations is thus

$$\begin{aligned}
\frac{d}{dt}\Big((M+m)\dot{x} + ml\dot{\theta}\cos\theta\Big) - 0 &= F - b\dot{x} \\
(M+m)\ddot{x} + \Big[-ml\dot{\theta}^2\sin\theta + ml\ddot{\theta}\cos\theta\Big] + b\dot{x} &= F \\
(M+m)\ddot{x} + ml\ddot{\theta}\cos\theta - ml\dot{\theta}^2\sin\theta + b\dot{x} &= F
\end{aligned}$$

The second of Lagrange's equations is thus

$$\begin{aligned}
\frac{d}{dt}\Big(ml\dot{x}\cos\theta + ml^2\dot{\theta}\Big) - \Big[ml\dot{x}\dot{\theta}\sin\theta + mgl\sin\theta\Big] &= 0 \\
\Big[ml\dot{x}\dot{\theta}\sin\theta + ml\ddot{x}\cos\theta\Big] + ml^2\ddot{\theta} - ml\dot{x}\dot{\theta}\sin\theta - mgl\sin\theta &= 0 \\
ml\ddot{x}\cos\theta + ml^2\ddot{\theta} - mgl\sin\theta &= 0 \\
\ddot{x}\cos\theta + l\ddot{\theta} - g\sin\theta &= 0
\end{aligned}$$

The differential equations of motion describing the system are thus

$$\begin{aligned}
(M+m)\ddot{x} + ml\ddot{\theta}\cos\theta - ml\dot{\theta}^2\sin\theta + b\dot{x} &= F \\
\ddot{x}\cos\theta + l\ddot{\theta} - g\sin\theta &= 0
\end{aligned}$$

We will now use these equations to solve for $\ddot{x}$ and $\ddot{\theta}$ as functions of $\{\dot{x}, x, \dot{\theta}, \theta\}$ in preparation for reducing this system of two second-order equations into a system of four first-order equations. Solving the second equation for $\ddot{\theta}$ yields

$$\ddot{\theta} = \frac{1}{l}[-\ddot{x}\cos\theta + g\sin\theta]$$

Substituting this result into the first equation of motion and simplifying yields

$$\ddot{x} = \frac{ml\dot{\theta}^2\sin\theta - mg\sin\theta\cos\theta - b\dot{x} + F}{(M + m(1+\cos^2\theta))}$$

Solving the second equation for $\ddot{x}$ yields

$$\ddot{x} = \frac{1}{\cos\theta}[-l\ddot{\theta} + g\sin\theta]$$

Substituting this result into the first equation of motion and simplifying yields

$$\ddot{\theta} = \frac{-g(M+m)\tan\theta + ml\dot{\theta}^2\sin\theta - b\dot{x} + F}{ml\cos\theta - \frac{l}{\cos\theta}(M+m)}$$

We now choose state variables $\{x_1, x_2, x_3, x_4\} = \{x, \dot{x}, \theta, \dot{\theta}\}$ such that our system of equations becomes

$$\begin{aligned}
\dot{x}_1 &= x_2 \\
\dot{x}_2 &= \frac{-mg\sin x_3\cos x_3 + mlx_4^2\sin x_3 - bx_2 + F}{(M + m(1+\cos^2 x_3))} \\
\dot{x}_3 &= x_4 \\
\dot{x}_4 &= \frac{-g(M+m)\tan x_3 + mlx_4^2\sin x_3 - bx_2 + F}{ml\cos x_3 - \frac{l}{\cos x_3}(M+m)}
\end{aligned}$$

with the outputs

$$\begin{aligned}
y_1 &= x_1 \\
y_2 &= x_3
\end{aligned}$$

After linearization, the state space formulation of this system is

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & \frac{-b}{M} & \frac{mg}{M} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{-b}{Ml} & \frac{g(M+m)}{Ml} & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} + \begin{pmatrix} 0 \\ \frac{1}{M} \\ 0 \\ \frac{1}{Ml} \end{pmatrix} F$$

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \end{pmatrix} F$$

Chapter 6

Robotics

6.1 Kinematics

The position and orientation (together, pose) of a rigid body in space can be described with respect to a coordinate frame in terms of a translation from the frame's origin and (a series of) rotations about the axes of the frame.

Let the fixed coordinate frame with origin O be specified by the unit axes $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$; let the body coordinate frame with origin O' be specified by unit axes $\mathbf{x}'$, $\mathbf{y}'$ and $\mathbf{z}'$.

Then the position of O' with respect to O is

$$\mathbf{o}' = o'_x\mathbf{x} + o'_y\mathbf{y} + o'_z\mathbf{z}$$

which can be written, in terms of the axes of the fixed frame, as the vector $\mathbf{o}' = [o'_x, o'_y, o'_z]^T$.

The orientation of the body frame with respect to the fixed frame is specified by expressing the axes of the body frame in terms of the axes of the fixed frame. This yields

$$\begin{aligned}\mathbf{x}' &= x'_x\mathbf{x} + x'_y\mathbf{y} + x'_z\mathbf{z}\\ \mathbf{y}' &= y'_x\mathbf{x} + y'_y\mathbf{y} + y'_z\mathbf{z}\\ \mathbf{z}' &= z'_x\mathbf{x} + z'_y\mathbf{y} + z'_z\mathbf{z}\end{aligned}$$

These relations can be expressed compactly in the rotation matrix

$$R = \begin{pmatrix}\mathbf{x}' & \mathbf{y}' & \mathbf{z}'\end{pmatrix} = \begin{pmatrix}x'_x & y'_x & z'_x\\ x'_y & y'_y & z'_y\\ x'_z & y'_z & z'_z\end{pmatrix} = \begin{pmatrix}\mathbf{x}'^T\mathbf{x} & \mathbf{y}'^T\mathbf{x} & \mathbf{z}'^T\mathbf{x}\\ \mathbf{x}'^T\mathbf{y} & \mathbf{y}'^T\mathbf{y} & \mathbf{z}'^T\mathbf{y}\\ \mathbf{x}'^T\mathbf{z} & \mathbf{y}'^T\mathbf{z} & \mathbf{z}'^T\mathbf{z}\end{pmatrix}$$

This matrix simply describes the body frame's axes in terms of the fixed frame. Note that since the axes of both frames are unit vectors these inner products are equal

to the cosines of the angles between the corresponding axes. For this reason, the rotation matrix is also referred to as the direction cosine matrix (where each element is referred to as a direction cosine).

Note that this matrix is orthogonal so its columns are mutually orthogonal. In fact, R is orthonormal - each column also has unit norm. As for all such orthogonal matrices, $R^{-1} = R^T$ and thus $R^T R = RR^T = I$.

Note that since the inverse of R (which defines the reverse mapping) is its transpose, the rows of this matrix are the axes of the fixed frame in terms of the axes of the body frame.

6.1.1 Elementary Rotations:

Consider a coordinate frame with origin O defined by the axes $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$. Consider rotating this frame by an angle θ about one of its axes (where a counter-clockwise rotation is taken to be positive). The rotation matrices describing the orientation of the axes $\mathbf{x}'$, $\mathbf{y}'$ and $\mathbf{z}'$ of the rotated frame relative to the original frame after a single such *elementary rotation* are

$$R_x(\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix}$$

$$R_y(\theta) = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix}$$

$$R_z(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Note that, for $k = x, y, z$ we have $R_k(-\theta) = R_k^T(\theta)$.

6.1.2 Vector Representation:

Consider again two frames with the same origin O but different sets of unit axes. A vector $\mathbf{p}$ can be described in terms of the first frame as $\mathbf{p} = [p_x, p_y, p_z]^T$ and in terms of the second frame as $\mathbf{p}' = [p'_x, p'_y, p'_z]^T$. Since these vectors correspond to the same point in space, we must have $\mathbf{p} = \mathbf{p}'$ and thus

$$\mathbf{p} = p'_x\mathbf{x}' + p'_y\mathbf{y}' + p'_z\mathbf{z}' = \begin{pmatrix} \mathbf{x}' & \mathbf{y}' & \mathbf{z}' \\ & & \end{pmatrix} \mathbf{p}'$$

This is, of course, $\mathbf{p} = R\mathbf{p}'$; R is the matrix which transforms the vector from the second frame to the first. Since R is orthogonal, the inverse transformation is $\mathbf{p}' = R\mathbf{p}$.

Note that since R is an orthogonal transformation we have

$$||R\mathbf{p}'|| = (R\mathbf{p}')^T(R\mathbf{p}') = \mathbf{p}'^T R^T R\mathbf{p}' = \mathbf{p}'^T\mathbf{p}' = ||\mathbf{p}'||$$

The matrix R therefore does not change the norm of the vector but only its direction.

6.1.3 Composition of Rotations

Consider a point $\mathbf{p}$ in space and its representations in the frames O^0, O^1 and O^2 having a common origin O. Let R_i^j denote the rotation matrix describing the orientation of frame i with respect to frame j; then we have

$$\mathbf{p^1} = R_2^1\mathbf{p^2}$$

We also have, using the equation above,

$$\mathbf{p^0} = R_1^0\mathbf{p^1} = R_1^0R_2^1\mathbf{p^2}$$

Finally, we can write

$$\mathbf{p^0} = R_2^0\mathbf{p^2} = R_1^0R_2^1\mathbf{p^2}$$

From the above equation it is obvious that we must have

$$R_2^0 = R_1^0R_2^1$$

Here it is interpreted that a frame aligned with O^0 is first rotated into alignment with O^1 with R_1^0 and then this frame is rotated into alignment with frame O^2 using R_2^1. That is, successive rotations can be applied to a vector via post-multiplication of the corresponding rotation matrices following the order of rotations. The overall rotation is therefore expressed as the composition of partial rotations; each rotation is defined with respect to the axes of the *current frame*.

When each rotation is made instead with respect to the axes of a *fixed frame*, the elementary rotation matrices are premultiplied in the order in which the rotations are applied.

6.2 Orientation Representations

6.2.1 Euler Angles

Rotation matrices, while intuitive, are inherently redundant as they use nine parameters to describe a rotation with only three degrees of freedom. Instead, a *minimal representation* can be obtained by using a set of three angles $\phi = [\alpha, \beta, \gamma]^T$ each of which corresponds to an elementary rotation about one of the coordinate axes. A generic rotation matrix can then be formed by composing a sequence of the three corresponding elementary rotation matrices.

There are thus $3^3 = 27$ possible combinations of three successive rotations; however, there are 15 such combinations which produce successive rotations about parallel axes so there are only $3 * 2 * 2 = 12$ valid sets/sequences.

Rotation Order and Naming

The naming of rotation angle sets/sequences can be extremely confusing and often ambiguous. Here, we refer to a sequence by (for example) *x-y-z* which means that a rotation is first applied about the x-axis, then the y-axis, and finally about the z-axis (see the next section for clarification about *which* x,y, and z axes we refer to with such a sequence).

Rotation matrices corresponding to a composite rotation through angles α, β and γ are formed by multiplying elemental rotations $R_X(\alpha)$, $R_Y(\beta)$ and $R_Z(\gamma)$ as, for example:

$$R = R_X(\alpha)R_Y(\beta)R_Z(\gamma)$$

This is often referred to as the XYZ rotation sequence because this is the order in which the elemental rotations appear. However, since we apply successive rotations via *left-multiplication*, this rotation actually corresponds to the sequence *z-y-x* and NOT *x-y-z*.

Intrinsic versus Extrinsic Rotations

Rotation sets/sequences are specified as either *intrinsic* or *extrinsic* depending on what frames the specified angles are referring to.

- *Intrinsic* rotations compose three elemental rotations about *mobile* axes which are attached to the rotation object and are specified by sequencies such as *x-y'-z"* (note the usage of primes which denote successive new frames).
- *Extrinsic* rotations compose three elemental rotations about *fixed* axes which coincide with the starting frame of the rotation object, for example the sequence *x-y-z*.

Euler Angles versus Tait-Bryan Angles

It's important to note that there are two formalisms for specifying rotations in terms of a minimal number of coordinates, both of which are often called Euler angles. Six of the 12 possible valid rotation sequences below to each formalism, as explained below:

- *"Proper" Euler angles* denote elemental rotation sequences which use the same axis for the first and third rotations, such as *z-x'-z"* (intrinsic) or *z-x-z* (extrinsic). The six possible sequences are:
  - *z-x'-z"* (intrinsic) or *z-x-z* (extrinsic)
  - *x-y'-x"* (intrinsic) or *x-y-x* (extrinsic)
  - *y-z'-y"* (intrinsic) or *y-z-y* (extrinsic)
  - *z-y'-z"* (intrinsic) or *z-y-z* (extrinsic)
  - *x-z'-x"* (intrinsic) or *x-z-x* (extrinsic)
  - *y-x'-y"* (intrinsic) or *y-x-y* (extrinsic)

- *Tait-Bryan angles* always compose elemental rotations about three distinct axes, for example *x-y'-z"* (intrinsic) or *x-y-z* (extrinsic). This convention is arguably easier to visualize in terms of elemental rotations and is normally used in the aerospace industry. The six possible sequences are:
  - *x-y'-z"* (intrinsic) or *x-y-z* (extrinsic)
  - *z-y'-x"* (intrinsic) or *z-y-x* (extrinsic)
  - *z-x'-y"* (intrinsic) or *z-x-y* (extrinsic)
  - *x-z'-y"* (intrinsic) or *x-z-y* (extrinsic)
  - *z-y'-x"* (intrinsic) or *z-y-x* (extrinsic)
  - *y-x'-z"* (intrinsic) or *y-x-z* (extrinsic)

  Note that the *x-y'-z"* sequence is known as *Roll-Pitch-Yaw* (or *Roll-Pitch-Yaw XYZ*), whereas the *z-y'-x"* (intrinsic) sequence is commonly known as *Yaw-Pitch-Roll* (or *Roll-Pitch-Yaw ZYX*).

In practice, the difference between proper Euler and Tait-Bryan angles is simply a matter of nomenclature and all sequences are referred to as Euler angles, as we will do in the remainder of this section.

Frame Handedness

The ambiguity of *handedness* refers to the chosen convention for a positive rotation about an axis; we only consider right-handed frames which means that a positive angle is a rotation to the right when looking down the axis in the positive direction.

Multiplication Order

Since a point can be represented equivalently as a column vector v or row vector w, rotation matrices can either pre-multiply column vectors as Rv or post-multiply row vectors as wR. However, Rv produces an opposite rotation to wR since $(wR)^T = R^T w^T = R^T v$. Thus, we must post-multiply by R^T to obtain the same result. We only consider representive points as column vectors in this text, therefore rotations are applied via pre-multiplication as Rv.

Active (alibi) versus Passive (alias) Rotations

There are two ways of interpreting the action of a rotation on a vector:

- The rotation changes the vector itself (*active* or *alibi* transformation) within the same coordinate frame.
- The rotation changes the coordinate frame in which the vector is described (*active* or *alibi* transformation), however the vector always describes the same object.

When composing rotations from elemental single-axis transformations, we must be careful to specify whether the elemental rotations use active or passive convention. The inverse of an active transformation is a passive transformation (their rotation matrices are simply related via a transpose).

Example: Roll-Pitch-Yaw Angles (x-y'-z" intrinsic)

The *Roll-Pitch-Yaw XYZ* or simple *Roll-Pitch-Yaw* angles are often used in robotics and aeronautics; they are equivalent to what is sometimes referred to as the *Euler ZYX* angles. The elemental rotations here are made with respect to a frame attached to the rotating object. The total rotation is composed as follows:

- Rotate the *initial* frame by α about the x-axis (roll); this corresponds to the matrix $R_x(\alpha)$.
- Rotate the *current* frame by β about the its y'-axis (pitch); this this corresponds to the matrix $R_y(\beta)$.
- Rotate the *current* frame by γ about its z"-axis (yaw); this corresponds to the matrix $R_z(\gamma)$.

The resulting orientation is composed via pre-multiplication of the matrices corresponding to the elemental rotations. This yields

$$R(\phi) = R_z(\gamma)R_y(\beta)R_x(\alpha) = \begin{pmatrix} c_\gamma c_\beta & c_\gamma s_\beta s_\alpha - s_\gamma c_\alpha & c_\gamma s_\beta c_\alpha + s_\gamma s_\alpha \\ s_\gamma c_\beta & s_\gamma s_\beta s_\alpha + c_\gamma c_\alpha & s_\gamma s_\beta c_\alpha - c_\gamma s_\alpha \\ -s_\beta & c_\beta s_\alpha & c_\beta c_\alpha \end{pmatrix}$$

The inverse solution is given by

$$\begin{aligned} \gamma &= Atan2(R_{21}, R_{11}) \\ \beta &= Atan2(-R_{31}, \sqrt{R_{32}^2 + R_{33}^2}) \\ \alpha &= Atan2(R_{32}, R_{33}) \end{aligned}$$

and is valid for $\beta \in (-\pi/2, \pi/2)$. This solution degenerates when $c_\beta = 0$ ie when $\beta = -\pi/2, \pi/2$; in this case only the sum or difference of γ and α can be determined.

6.2.2 Angle-Axis

A nonminimal representation of the orientation of a rigid body can be obtained by expressing the orientation as a rotation by θ about a unit vector $\mathbf{r} = [r_x, r_y, r_z]^T$ defined with respect to the frame O. This requires four parameters rather than just three.

In order to arrive at the corresponding rotation matrix we need to use the fixed frame in which $\mathbf{r}$ is defined; we do this by first aligning the axis with the frame via a series of rotations and then realigning with the direction $\mathbf{r}$ as follows:

- Align $\mathbf{r}$ with the z-axis, which is done by first rotating by $-\alpha$ about the z-axis and then by $-\beta$ about the y-axis. We are not ready to make rotations with respect to the fixed frame, which we can formalize.
- Next, rotate by θ about the z-axis.
- Finally, realign with the initial direction of $\mathbf{r}$ via a rotation by β about the y-axis and then a rotation by α about the z-axis.

In total, these steps lead to the rotation matrix

$$R(\theta, \mathbf{r}) = R_z(\alpha)R_y(\beta)R_z(\theta)R_y(-\beta)R_z(-\alpha)$$

where we note that, since these rotations are made with respect to a *fixed* frame, we use premultiplication.

The resulting rotation matrix is then

$$R(\theta, \mathbf{r}) = \begin{pmatrix} r_x^2(1-c_\theta)+c_\theta & r_x r_y(1-c_\theta)-r_z s_\theta & r_x r_z(1-c_\theta)+r_y s_\theta \\ r_x r_y(1-c_\theta)+r_z s_\theta & r_y^2(1-c_\theta)+c_\theta & r_y r_z(1-c_\theta)-r_x s_\theta \\ r_x r_z(1-c_\theta)-r_y s_\theta & r_y r_z(1-c_\theta)+r_x s_\theta & r_x^2(1-c_\theta)+c_\theta \end{pmatrix}$$

For this matrix we have $R(-\theta, \mathbf{r}) = R(\theta, \mathbf{r})$ which means that a rotation of $-\theta$ about $-\mathbf{r}$ is indistinguishable from a rotation of θ about $\mathbf{r}$. Therefore, the angle-axis representation is NOT unique.

This can be seen more clearly by considering thje inverse problem. From the above rotation matrix, we have

$$\theta = \cos^{-1}\left(\frac{r_{11}+r_{22}+r_{33}-1}{2}\right)$$
$$\mathbf{r} = \frac{1}{2\sin(\theta)}\begin{pmatrix} r_{32}-r_{23} \\ r_{13}-r_{31} \\ r_{21}-r_{12} \end{pmatrix}$$

for $s_\theta \neq 0$. Note that the three components of $\mathbf{r}$ are actually not independent but are rather constrained by the condition

$$r_x^2 + r_y^2 + r_z^2 = 1$$

as would be expected considering that this is a nonminimal representation.

Additionally, if $s_\theta = 0$ then the above expressions become meaningless. For null rotation ($\theta = 0$) the unit vector $\mathbf{r}$ becomes arbitrary; this is a singularity.

6.2.3 Unit Quaternion

The drawbacks of the axis-angle formulation can be overcome by the unit quaternion, defined by

$$\eta = \cos\frac{\theta}{2}$$
$$\epsilon = \sin\frac{\theta}{2}\mathbf{r}$$

where η is the scalar part of the quaternion and $\epsilon = [\epsilon_x, \epsilon_y, \epsilon_z]^T$ is the vector part. They are constrained by the condition

$$\eta^2 + \epsilon_x^2 + \epsilon_y^2 + \epsilon_z^2 = 1$$

which is why this is called the *unit* quaternion. Note that a rotation by $-\theta$ about $-\mathbf{r}$ gives the same quaternion as that corresponding to a rotation by θ about $\mathbf{r}$, solving the non-uniqueness problem of axis-angle representation.

Using the above definitions and the results from the axis-angle formulation, the rotation matrix corresponding to the unit quaternion is

$$R(\eta, \mathbf{r}) = \begin{pmatrix} 2(\eta^2 + \epsilon_x^2) - 1 & 2(\epsilon_x\epsilon_y - \eta\epsilon_z) & 2(\epsilon_x\epsilon_z + \eta\epsilon_y) \\ 2(\epsilon_x\epsilon_y + \eta\epsilon_z) & 2(\eta^2 + \epsilon_y^2) - 1 & 2(\epsilon_y\epsilon_z - \eta\epsilon_x) \\ 2(\epsilon_x\epsilon_z - \eta\epsilon_y) & 2(\epsilon_y\epsilon_z + \eta\epsilon_x) & 2(\eta^2 + \epsilon_z^2) - 1 \end{pmatrix}$$

The solution to the inverse problem for the unit quaternion is

$$\eta = \frac{1}{2}\sqrt{R_{11} + R_{22} + R_{33} + 1}$$
$$\epsilon = \frac{1}{2}\begin{pmatrix} sgn(R_{32} - R_{23})\sqrt{R_{11} - R_{22} - R_{33} + 1} \\ sgn(R_{13} - R_{31})\sqrt{R_{22} - R_{33} - R_{11} + 1} \\ sgn(R_{21} - R_{12})\sqrt{R_{33} - R_{11} - R_{22} + 1} \end{pmatrix}$$

Here it has been assumed that $\eta \geq 0$ which corresponds to an angle $\theta \in [-\pi, \pi]$; thus, any rotation can be descibed. Note that, in contrast to the axis-angle formulation, no singularities occur for the unit quaternion.

The quaternion corresponding to $R^{-1} = R^T$ is denoted as Q^{-1} and is computed as $Q^{-1} = \{\eta, -\epsilon\}$.

The quaternion product of Q_1 and Q_2 corresponding to the product R_1R_2 is given by

$$Q_1 * Q_2 = \{\eta_1\eta_2 - \epsilon_\mathbf{1}{}^T\epsilon_\mathbf{2}, \eta_1\epsilon_\mathbf{2} + \eta_2\epsilon_\mathbf{1} + \epsilon_\mathbf{1} \times \epsilon_\mathbf{2}\}$$

Note that the product $Q_1 * Q_1^{-1} = \{1, \mathbf{0}\}$.

Quaternion Conventions

6.3 Quaternion Review and Conventions

It is important to note that there are two different conventions for using quaternions, each of which is self-consistent; unfortunately, the quaternion algebra used in these conventions is often mixed up in the literature, resulting in inconsistent implementations [1]. We use that proposed as a standard by JPL in a white paper by Breckenridge[2] A review of quaternions using this convention is given in this section.

Briefly, quaternions are one of several choices for representing $SO(3)$, the Lie group of rotations. The advantage of quaternions over other parameterizations is their numerical properties, efficiency and lack of singularities. We denote a quaternion by

$$q = \begin{pmatrix} q_x \\ q_y \\ q_z \\ q_w \end{pmatrix}$$

where q_w and $q_v = [q_x, q_y, q_z]^T$ are the scalar and vector components, respectively. Note that the ordering of these components does not depend on any conventions and is purely a matter of preference. We choose this ordering here to match the literature but actually implement quaternions in the reverse order to match the SL simulation environment.

All rotations are active, meaning that they act to rotate vectors; the quaternion representing the base orientation is written as $q = q_W^B$ which specifies a rotation from the world frame W to the base frame B. This quaternion corresponds to the rotation matrix $C = C[q]$ which rotates vectors defined in the world frame into the base frame.

Successive rotations about local axes are composed via left-multiplication, ie $R_A^C = R_B^C R_A^B$ represents a rotation from frame A to frame B (given in terms of the frame A basis) followed by a rotation from frame B to frame C (given in terms of the frame B basis). Analogously, we have $q_A^C = q_B^C \otimes q_A^B$ for quaternions where $\otimes$ denotes quaternion multiplication.

A vector in frame A is rotated into frame C as $v^C = R_A^C v^A$, with the reverse transformation given by $v^A = (R_A^C)^{-1} v^C$ where $(R_A^C)^{-1} = R_C^A = (R_A^C)^T$ since this is a rotation matrix (orthogonal matrix with det $= +1$). Vector rotations using quaternions are achieved through conjugation as

$$v^C = q_A^C \otimes \begin{pmatrix} v^A \\ 0 \end{pmatrix} \otimes (q_A^C)^{-1}$$

[1] `http://www.ladispe.polito.it/corsi/meccatronica/02JHCOR/2011-12/Slides/Shuster_Pub_1993h_J_Repsurv_scan.pdf`

[2] This technical report seems to have mysteriously vanished from the internet as investigated by `http://fzheng.me/2017/11/12/quaternion_conventions_en/`, who also suggests that the other convention (as originally proposed by Hamilton) may be a better choice as it's used by popular libraries. We use the JPL convenion solely because it was used in a previous work on which our base state estimation paper was based. In any case, one must remain consistent within the same work.

where $[v^A, 0]^T$ is called a pure quaternion and $(q_A^C)^{-1} = q_C^A = [-q_x, -q_y, -q_z, q_w]$ is the inverse or conjugate quaternion satisfying $q_A^C \otimes (q_A^C)^{-1} = (q_A^C)^{-1} \otimes q_A^C = q_I$ where $q_I = [0, 0, 0, 1]$ is the identity quaternion.

Quaternion multiplication is defined by

$$q \otimes p = \begin{pmatrix} q_w p_x + q_z p_y - q_y p_z + q_x p_w \\ -q_z p_x + q_w p_y + q_x p_z + qy p_w \\ q_y p_x - q_x p_y + q_w p_z + q_z p_w \\ -q_x p_x - q_y p_y - q_z p_z + q_w p_w \end{pmatrix}$$

This can be written more concisely as the matrix vector multiplication

$$\begin{aligned} q \otimes p = L(q)p &= \begin{pmatrix} q_w I - q_v^\times & q_v \\ -q_v^T & q_w \end{pmatrix} \begin{pmatrix} p_v \\ p_w \end{pmatrix} \\ &= R(p)q = \begin{pmatrix} p_w I + p_v^\times & p_v \\ -p_v^T & p_w \end{pmatrix} \begin{pmatrix} q_v \\ q_w \end{pmatrix} \end{aligned}$$

where $q_v = [q_x, q_y, q_z]^T$ is the vector part of q and

$$q_v^\times = \begin{pmatrix} 0 & -q_z & q_y \\ q_z & 0 & -q_x \\ -q_y & q_x & 0 \end{pmatrix}$$

is the skew-symmetric matrix corresponding to the vector q_v.

The rotation matrix corresponding the quaternion q is given by

$$\begin{aligned} C[q] &= (2q_w^2 - 1)I - 2q_w q_v^\times + 2q_v q_v^T \\ &= \begin{pmatrix} 2q_x^2 + 2q_w^2 - 1 & 2(q_x q_y + q_z q_w) & 2(q_x q_z - q_y q_w) \\ 2(q_x q_y - q_z q_w) & 2q_y^2 + 2q_w^2 - 1 & 2(q_y q_z + q_x q_w) \\ 2(q_x q_z + q_y q_w) & 2(q_y q_z - q_x q_w) & 2q_z^2 + 2q_w^2 - 1 \end{pmatrix} \end{aligned}$$

where the first equation can be seen as equivalent to Rodrigues' identity using the quaternion exponential map

$$\exp(\omega) = \begin{pmatrix} \sin(\frac{||\omega||}{2}) \frac{\omega}{||\omega||} \\ \cos(\frac{||\omega||}{2}) \end{pmatrix}$$

which represents a rotation of $||\omega||$ about an axis $\omega/||\omega||$ as a quaternion. Letting $\delta\phi$ be an infinitesimal rotation, we see that

$$\delta q = \exp(\delta\phi) \approx \begin{pmatrix} \frac{1}{2}\delta\phi \\ 1 \end{pmatrix}$$

is the first-order approximation of an incremental quaternion. It follows from the definition of $C[q]$ that we have the first-order approximation

$$C[\delta q] \approx I - \delta\phi^\times$$

This can also be seen as the first-order approximation of the exponential map for rotation matrices

$$\exp(\delta\phi^{\times}) = \sum_{i=0}^{\infty} \frac{(-\delta\phi^{\times})^i}{i!} \approx I - \delta\phi^{\times}$$

It follows that the first-order expansion of q about a nominal quaternion $\bar{q}$ can be written in matrix form as

$$C[\delta q \otimes \bar{q}] = C[\delta q]C[\bar{q}] = (I - \delta\phi^{\times})\bar{C}$$

where $\bar{C} = C[\bar{q}]$. This approximation will be used in the derivations of the linearized filter dynamics in the next section.

The derivative of a quaternion is related to the angular velocity ω by the equation

$$\dot{q} = \frac{1}{2}\begin{pmatrix}\omega \\ 0\end{pmatrix} \otimes q$$

and the first-order approximation of $\dot{\delta q}$ is given by

$$\dot{\delta q} \approx \begin{pmatrix}\frac{1}{2}\delta\dot{\phi} \\ 0\end{pmatrix}$$

6.3.1 Homogeneous Transformations

The *pose* (position and orientation) of a rigid body in space (a point P with body-fixed frame O_1) is fully specified with respect to a reference frame O_0 by a vector $\mathbf{o}_1^0$ describing the origin of Frame 1 with respect to Frame 0 and a rotation matrix R_1^0 describing the orientation of Frame 1 with respect to Frame 0.

The position of point P in Frame 0 is then

$$\mathbf{p}^0 = \mathbf{o}_1^0 + R_1^0\mathbf{p}^1$$

This specifies a *coordinate transformation* of a vector between the two frames. The inverse transformation comes from premultiplying both sides of the above equation by ${R_1^0}^T = R_0^1$ and solving for p^1; this leads to

$$p^1 = -R_0^1 o_1^0 + R_0^1 p^0$$

These transformations can be expressed in a more compact form by converting to *homogeneous representations* given by

$$\tilde{p}^0 = A_1^0\tilde{p}^1$$
$$\tilde{(p)}^1 = A_0^1\tilde{p}_0 = (A_1^0)^{-1}\tilde{p}^0$$

where $\tilde{p} = [p, 1]^T$ and the homogeneous transformation matrix A_1^0 is defined to be

$$A_1^0 = \begin{pmatrix}R_1^0 & o_1^0 \\ 0^T & 1\end{pmatrix}$$

and, from above,

$$A_0^1 = \begin{pmatrix} R_0^1 & -R_0^1 o_1^0 \\ 0^T & 1 \end{pmatrix}$$

It is important to note that, in general, homogeneous transformation matrices are NOT orthogonal.

When the frames have the same origin, the homogeneous matrix reduces to the orientation matrix; when the frames have different origins, the total rotation is composed as

$$\tilde{p}^0 = A_1^0 A_2^1 \cdots A_n^{n-1} \tilde{p}^n$$

as was previously done for orientation matrices. Note here that postmultiplication is used because these matrices are all defined with respect to the preceding frame (these are partial rotations; this will be useful in defining a systematic way to transform between links of a robot manipulator).

6.3.2 Tracking Orientation

Here we will consider tracking the orientation of a body - represented by a rotation matrix - in terms of the instantaneous angular velocity of that body[3].

The goal is to derive a differential equation which describes the dynamics of the orientation matrix R in terms of the angular velocity vector $\omega = [\omega_x, \omega_y, \omega z]^T$. If the attitude of the body at time t is given by $R(t)$ then the rate of change is given by

$$\dot{R}(t) = \lim_{\delta_t \to 0} \frac{R(t + \delta t) - R(t)}{\delta t}$$

Here the orientation at time $t + \delta t$ can be written as the product of the orientation at time t and a matrix $A(t)$ which is the rotation matrix of the body frame from time t to time $t + \delta t$. Thus,

$$R(t + \delta t) = R(t)A(t)$$

This matrix $A(t)$ is the product of three elementary rotations about the body frame axes; when δt is sufficiently small, the order of these rotations doesn't matter and the update matrix becomes

$$A(t) = I + \delta\Psi = \begin{pmatrix} 1 & -\psi & \theta \\ \psi & 1 & -\phi \\ -\theta & \phi & 1 \end{pmatrix}$$

where

[3]See the following report by Oliver Woodman for more details: `https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-696.pdf`

$$\delta\Psi = \begin{pmatrix} 0 & -\psi & \theta \\ \psi & 0 & -\phi \\ -\theta & \phi & 0 \end{pmatrix}$$

Thus,

$$\begin{aligned} \dot{R}(t) &= \lim_{\delta_t \to 0} \frac{R(t+\delta t) - R(t)}{\delta t} \\ &= \lim_{\delta_t \to 0} \frac{R(t)A(t) - R(t)}{\delta t} \\ &= \lim_{\delta_t \to 0} \frac{R(t)(I + \delta\Phi) - R(t)}{\delta t} \\ &= R(t) \lim_{\delta t \to 0} \delta\Phi\delta t \\ &= C(t)\Omega(t) \end{aligned}$$

where, in the limit, we have

$$\lim_{\delta t \to 0} \frac{\delta\Phi}{\delta t} = \Omega(t) = \begin{pmatrix} 0 & -\omega_z(t) & \omega_y(t) \\ \omega_z(t) & 0 & -\omega_x(t) \\ -\omega_y(t) & \omega_x(t) & 0 \end{pmatrix}$$

which is the skew-symmetric form of the angular velocity vector $\omega(t)$ representing the angular velocity of the body with respect to the body frame at time t.

Thus, the differential equation governing the time evolution of the orientation matrix $\dot{R}(t) = R(t)\Omega(t)$ has the solution

$$R(t) = R(0) \exp\left(\int_0^t \Omega(t)dt\right)$$

where $R(0)$ is the attitude at time $t = 0$.

For a single period $[t, t + \delta t]$ the solution can be written as

$$R(t + \delta t) = R(t) \exp\left(\int_t^{t+\delta t} \Omega(t)dt\right)$$

where, assuming a first-order integration scheme (rectangular rule) is used to integrate the sampled angular velocity, we have

$$B = \int_t^{t+\delta t} \Omega(t)dt = \sigma \begin{pmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{pmatrix}$$

where $\sigma = ||\omega||_2\delta t$. Here the components of the vector ω have been normalized such that $(\omega_x^2 + \omega_y^2 + \omega_z^2) = 1$ (this normalization simplifies the characteristic polynomial).

The characteristic polynomial of B is then (for this particular form of B)

$$p(\lambda) = det(B - \lambda I) = -\lambda^3 - \sigma^2\lambda$$

by the Cayley-Hamilton theorem we have that $p(B) = 0$ and thus

$$-B^3 - \sigma^2 B = 0 \rightarrow B^2 = -\sigma^2$$

The power series expansion of the matrix exponential of B, using the above fact, thus yields

$$\begin{aligned}
\exp(B) &= \sum_{i=0}^{\infty} \frac{B^i}{i!} \\
&= \left(I + B + \frac{B^2}{2!} + \frac{B^3}{3!} + \frac{B^4}{4!} + \cdots\right) \\
&= \left(I + B + \frac{B^2}{2!} - \frac{\sigma^2 B}{3!} - \frac{\sigma^2 B^2}{4!} + \cdots\right) \\
&= \left(I + \left(1 - \frac{\sigma^2}{3!} + \frac{\sigma^4}{5!} + \cdots\right) B + \left(\frac{1}{2!} - \frac{\sigma^2}{4!} + \frac{\sigma^4}{6!} + \cdots\right) B^2\right) \\
&= \left(I + \frac{\sin\sigma}{\sigma} B + \frac{1 - \cos\sigma}{\sigma^2} B^2\right)
\end{aligned}$$

Therefore, the update equation for the orientation matrix becomes

$$R(t + \delta t) = R(t)\left(I + \frac{\sin\sigma}{\sigma} B + \frac{1 - \cos\sigma}{\sigma^2} B^2\right)$$

6.3.3 Rotation Matrices

A rotation matrix is an *orthogonal matrix* with unit determinant. In fact, it is an *orthonormal matrix* because the 2-norm of each of its rows and columns is 1. It expresses a transformation which rotates a vector but does not change its magnitude (since, for any orthogonal matrix Q in general, $||Qx||_2 = (Qx)^T(Qx) = x^T Q^T Qx = x^T x = ||x||_2$ since $Q^T Q = I$).

In 3-dimensional space the rotation matrices corresponding to rotations about the x, y, and z axes are, respectively:

$$R_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{pmatrix}$$

$$R_y = \begin{pmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{pmatrix}$$

$$R_z = \begin{pmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Any rotation can be expressed as the product $R = R_x R_y R_z$, noting that in general these matrices do not commute.

In the context of numerically integrating the angular velocity about the three axes, if the timestep δt used for integration is sufficiently small then the small angle approximations for sine and cosine hold. This results in

$$R = R_x R_y R_z \approx \begin{pmatrix} 1 & -\psi & \theta \\ \psi & 1 & -\phi \\ -\theta & \phi & 1 \end{pmatrix}$$

where the order in which rotations are applied about the individual axes no longer matters.

6.4 Direct Kinematics

A robotics manipulator consists of a series of rigid bodies or *links* connected by means of *joints*. Joints can be of two types: *revolute* or rotational and *prismatic* or translational. The entire structure of the manipulator is formed by the links and joints and is known as a *kinematic chain*. One end of the chain is constrained to a *base* while the other end is usually connected to an *end-effector* for manipulation of objects in space.

There are two types of chains: *open* and *closed* chains. Here we will consider only open chains, which are defined as chains in which there is only one sequence of links which connects the base to the end-effector. Closed chains, on the other hand, are more complicated because their links form loops.

The term *posture* is used to describe the pose of all the rigid bodies composing the chain; the posture is described by the number of degrees of freedom (DOFs) of the manipulator structure. Each DOF typically corresponds to the ability to articulate a joint and is thus termed a *joint variable*. The goal of *direct kinematics* is to find a mapping - via a sequence of homogeneous transformations - which descibes the pose of the end-effector with respect to the base in terms of the joint variables.

The direct kinematics function is thus the transformation matrix

$$T_e^b(\mathbf{q}) = \begin{pmatrix} \mathbf{n}_e^b(\mathbf{q}) & \mathbf{s}_e^b(\mathbf{q}) & \mathbf{a}_e^b(\mathbf{q}) & \mathbf{p}_e^b(\mathbf{q}) \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

where $\mathbf{q}$ is the vector of joint variables, $R_e^b(\mathbf{q}) = [\mathbf{n}_e^b \mathbf{s}_e^b \mathbf{a}_e^b]$ is the rotation matrix describing the orientation of the end-effector frame with respect to the base frame and $\mathbf{p}_e^b$ is the position of the origin of the end-effector frame with respect to the origin of the base frame. Note that all parts of this transformation are functions of the joint variables, ie T_e^b is configuration-dependent!

Consider an open chain of $n+1$ links connected by n joints. It is assumed that Link 0 is fixed and each joint provides the structure with a single degree of freedom (complex joints such as a spherical joint are treated as combinations of single-DOF joints). Attached to each link from 0 to n is a coordinate frame; the coordinate transformation from frame n to frame 0 is then

$$T_n^0(\mathbf{q}) = A_1^0(q_1)A_2^1(q_2)\cdots A_n^{n-1}(q_n)$$

where each homogeneous transformation is incremental (defined relative to the preceding link) and is thus a function of only one joint variable. The direct kinematics function from the end-effector to the base also requires transformations from link 0 to the base and from link n to the end-effector; these are typically constant transformations. The direct kinematics function is thus written as

$$T_e^b(\mathbf{q}) = T_0^b T_n^0(\mathbf{q}) T_e^n$$

6.4.1 Denavit-Hartenberg Convention

The Denavit-Hartenberg Convention is a method for choosing the link frames such that computation of the direct kinematics function is accomplished in a general, systematic fashion. The frame corresponding to the i^{th} link is located at joint $i-1$ and is defined as follows:

- Choose the z-axis z_i of each frame through the axis of the joint.
- Locate the origin of the frame at the intersection of z_i and the common normal to z_i and z_{i-1}. If z_i and z_{i-1} intersect, this is the location of the origin.
- Choose the x-axis x_i along the common normal to z_i and z_{i-1}. If these z-axes intersect, choose x_i to be $z_{i-1} \times z_i$.
- Finally, choose y_i to complete a right-handed coordinate frame.

Thus, in total, there are n links and $n+1$ joints/frames. For each link the DH convention specifies four parameters:

- θ is the angle between x_i and x_{i-1} about z_{i-1} (only variable for a revolute joint).
- d_i is the distance between the origins of frames i and $i-1$ along the direction of axis z_{i-1} (only variable for a prismatic joint).
- α_i is the angle between axes z_{i-1} and z_i about x_i.
- a_i is the distance between the axes z_{i-1} and z_i along the direction of axis x_i (common normal between z-axes).

Note that only one of θ_i and d_i can be variable for each link. In this setup, Link i is located between Frames i and $i-1$; the homogeneous transformation between these frames is given by

$$A_i^{i-1}(q_i) = \begin{pmatrix} c_{\theta_i} & -s_{\theta_i}c_{\alpha_i} & s_{\theta_i}s_{\alpha_i} & a_i c_{\theta_i} \\ s_{\theta_i} & c_{\theta_i}c_{\alpha_i} & -c_{\theta_i}s_{\alpha_i} & a_i s_{\theta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Note that the transformation from Frame i to Frame $i-1$ is a funbction only of joint variable q_i which is either θ_i (revolute joint) or d_i (prismatic joint); all other parameters in this matrix are fixed.

6.4.2 Joint and Operational Space

If a task is to be specified in terms of trajectories for the end-effector, then one must work in the *operational space* defined by the $m \times 1$ vector

$$\mathbf{x}_e = \begin{pmatrix} \mathbf{p}_e \\ mathbfphi_e \end{pmatrix}$$

where $\mathbf{p}_e$ is the position of the manipulator and $\mathbf{phi}_e$ is its orientation; together these specify the end-effector's pose.

On the other hand, the *joint space* is defined by the $n \times 1$ vector of joint variables

$$\mathbf{q} = \begin{pmatrix} q_1 \\ q_2 \\ \vdots \\ q_3 \end{pmatrix}$$

Of course, the operational space variables can be extracted from the homogeneous transformation matrix $A_e^b(\mathbf{q})$ discussed above. We can also write the mapping directly as

$$\mathbf{x}_e = k(\mathbf{q})$$

where $k()$ is a (typically nonlinear) vector function which performs the mapping from joint space to operational space.

The most general operational space has $m = 6$ and thus uses a minimal representation for the orientation of the end-effector; in this case ϕ_e cannot be computed directly from the joint variables (in closed form) and thus this computation must go through the rotation matrix (since the homogeneous transformation gives us the rotation matrix R_e^b and we know how to compute ϕ_e from R_e^b as was discussed in a previous section).

6.4.3 The Workspace

When dealing with the operational space (pose of the end-effector), it is essential to know the possible spatial configurations which the manipulator can attain. The *reachable workspace* is the region in space which the origin of the end-effector frame can reach with at least one orientation; the *dexterous workspace* is the region which the origin of the end-effector can reach while attaining multiple orientations. The dexterous workspace is thus a subspace of the reachable workspace.

The reachable workspace is the geometric locus of points that can be reached by considering the direct kinematics equation

$$\mathbf{p}_e = p_e(\mathbf{q}) \quad \mathbf{q}_{min} \leq \mathbf{q} \leq \mathbf{q}_{max}$$

where $p_e()$ is a continuous vector function (it is just the position part of the function $k()$ introduced earlier) and $\mathbf{q}_{min}$ and $\mathbf{q}_{max}$ are the joint limits for all the joint variables.

6.4.4 Kinematic Redundancy

A manipulator is *kinematically redundant* when it has a number of DOFs greater than the number of variables needed to describe a given task ie when $n > m$. Note, however, than a manipulator can still be *functionally redundant* if $m = n$ and only $r < n$ variables are needed to specify a given task. Thus, redundancy is truly a function of the task being specified. For example, the three-DOF planar arm has $n = 3$ and $m = 3$ (since the pose of the end-effector in the plane is fully specified by its position and angle). When only the position is specified for a given task, however, $r = 2$ and the manipulator becomes *functionally redundant.*

6.4.5 Inverse Kinematics

Most tasks are specified in terms of trajectories in operational space, yet the robot must be controlled in the joint space. For this reason we need an inverse mapping from $\mathbf{x}_e$ to $\mathbf{q}$.

The issue is that the direct kinematics mapping $\mathbf{x}_e$ (which comes from T_e^b) is usually nonlinear; if this were not the case, we could simply write $\mathbf{x}_e = K\mathbf{q}$ and solve the linear system for the vector of joint variables.

Aside from this issue of nonlinearity, there are other problems inherent to the mechanical nature of the manipulator. There may be multiple - or even *infinite* - solutions to the inverse problem. Note that multiple solutions for the same end-effector pose depend not only on the number of DOFs but also the number of non-null DH parameters. On the other hand, if the operational space trajectory moves outside the dexterous workspace of the robot then there will be no solutions.

Closed-form solutions to the inverse kinematics problems require either algebraic/-geometric intuition. On the other hand, numerical solution techniques offer alternatives which can be applied to any manipulator but in general do not allow for the computation of all possible solutions.

6.5 Differential Kinematics

We have already discussed the problems of *direct and inverse kinematics* which deal with the relationship between the joint space and operational space variables. Now we discuss the problems of *direct and inverse differential kinematics* which deal with the relationship between the joint space and operational space velocities. This mapping is described by a matrix called the *Jacobian*.

There are two ways to arrive at such a mapping - the first is called the *geometric* Jacobian and the second is called the *analytical* Jacobian.

The geometric Jacobian is derived in a manner similar to that of the direct kinematics function in which one sums up the contributions of each individual joint velocity to the total end-effector velocity; this mapping is configuration-dependent.

The analytical Jacobian results from differentiating the direct kinematics function (when this function describes the pose with reference to a minimal representation in operational space) with respect to the joint variables.

6.5.1 Geometric Jacobian

For an n-DOF manipulator we have the direct kinematics equation

$$T_e(\mathbf{q}) = \begin{pmatrix} R_e(\mathbf{q}) & \mathbf{p}_e(\mathbf{q}) \\ \\ \mathbf{0}^T & 1 \end{pmatrix}$$

It is desired to express the end-effector linear velocity $\dot{\mathbf{p}}_e$ and the end-effector angular velocity ω_e in terms of the joint velocities $\dot{\mathbf{q}}$. It will be shown that these relations are both linear in the joint velocities and are given by

$$\dot{\mathbf{p}}_e = J_P(\mathbf{q})\dot{\mathbf{q}}$$
$$\omega_e = J_O(\mathbf{q})\dot{\mathbf{q}}$$

where $J_P \in R^{3\times n}$ and $J_O \in R^{3\times n}$ are the Jacobian matrices relating the contributions of the joint velocities to the end-effector linear and angular velocities, respectively. We can write this in compact form as the *differential kinematics equation*

$$\mathbf{v}_e = \begin{pmatrix} \dot{\mathbf{p}}_e \\ \omega_e \end{pmatrix} = J(\mathbf{q})\dot{\mathbf{q}}$$

where

$$J = \begin{pmatrix} J_P \\ J_O \end{pmatrix}$$

is the manipulator geometric Jacobian which is, in general, a function of the manipulator configuration. This matrix is derived as follows.

First, consider the time derivative of a rotation matrix $R = R(t)$. Since such a matrix is orthogonal,

$$R(t)R(t)^T = I$$

Differentiating this expression with respect to time yields

$$R(t)\dot{R}^T(t) + \dot{R}(t)R^T(t) = 0$$

Defining $S(t) = \dot{R}(t)R(t)^T$ we have

$$S(t) + S^T(t) = 0$$

which implies that the matrix $S(t)$ must be *skew-symmetric* since the sum of it and its transpose equals the zero matrix.

Since $R^{-1}(t) = R(t)$ we can solve our expression for $S(t)$ to yield

$$\dot{R}(t) = S(t)R(t)$$

which is the differential equation relating the rotation matrix to its derivative via the skew-symmetric operator S.

Consider a constant vector $\mathbf{p}'$ in a rotating reference frame described by $R(t)$ and its image $\mathbf{p(t)} = R(t)\mathbf{p}'$ in the fixed frame in which $R(t)$ is defined. Taking the derivative of $\mathbf{p(t)}$ yields

$$\dot{\mathbf{p}}(t) = \dot{R}(t)\mathbf{p}'$$

which, using the definition of the derivative of a rotation matrix, can be written as

$$\dot{\mathbf{p}}(t) = S(t)R(t)\mathbf{p}'$$

From mechanics, however, we know that this is simply given by $\dot{\mathbf{p}}(t) = \omega(t) \times \mathbf{p}(t) = \omega(t) \times R(t)\mathbf{p}'$ where $\omega(t) = [\omega_x, \omega_y, \omega z]^T$ is the angular velocity of the rotating frame with respect to the reference frame at time t.

This means that we must have

$$S(\omega(t)) = \begin{pmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{pmatrix}$$

We can thus write $\dot{R} = S(\omega)R$. Further, for a rotation matrix we have the property

$$RS(\omega)R^T = S(R\omega)$$

Now consider the coordinate transformation of a point P from Frame 1 to Frame 0 given by

$$\mathbf{p}^0 = \mathbf{o}_1^0 + R_1^0\mathbf{p}^1$$

Differentiating this expression with respect to time yields

$$\begin{aligned}\dot{\mathbf{p}}^0 &= \dot{\mathbf{o}}_1^0 + R_1^0\dot{\mathbf{p}}^1 + \dot{R}_1^0\mathbf{p}^1 \\ &= \dot{\mathbf{o}}_1^0 + R_1^0\dot{\mathbf{p}}^1 + S(\omega_1^0)R_1^0\mathbf{p}^1 \\ &= \dot{\mathbf{o}}_1^0 + R_1^0\dot{\mathbf{p}}^1 + \omega_1^0 \times \mathbf{r}_1^0\end{aligned}$$

where $\mathbf{r}_1^0 = R_1^0\mathbf{p}^1$ represents the point P after it has been rotated into Frame 0 but not translated, ie $r_1^0 = p^0 - o_1^0$. This is known as the *velocity composition rule*. If the point P is fixed in Frame 1 then this reduces to

$$\dot{\mathbf{p}}^0 = \dot{\mathbf{o}}_1^0 + \omega_1^0 \times \mathbf{r}_1^0$$

Now consider deriving the relationships between the linear and angular velocities of successive frames. Using the same DH convention for choosing link frames, it can be shown that

$$\dot{\mathbf{p}}_i = \dot{\mathbf{p}}_{i-1} + \dot{\mathbf{v}}_{i-1,i} + \omega_{i-1} \times \mathbf{r}_{i-1,i}$$

gives the linear velocity of Link i as a function of the translational and rotational velocities of Link $i-1$. Note that all vectors are expressed with respect to a fixed Frame 0 and that $\mathbf{v}_{i-1,i}$ denotes the velocity of the origin of Frame i with respect to the origin of Frame $i-1$ as expressed in terms of Frame 0. In addition,

$$\omega_i = \omega_{i-1} + \omega_{i-1,i}$$

gives the angular velocity of Link i as a function of the angular velocities of Link $i-1$ and of Link i with respect to Link $i-1$ ($\omega_{i-1,i}$).

Using these general results, we have that for a *prismatic joint*

$$\omega_i = \omega_{i-1}$$

since the orientation of Frame i with respect to $i-1$ does not change when Joint i is moved and thus $\omega_{i-1,i} = 0$. For the linear velocity we have

$$\dot{\mathbf{p}}_i = \dot{\mathbf{p}}_{i-1} + \dot{d}_i\mathbf{z}_{i-1} + \omega_i \times \mathbf{r}_{i-1,i}$$

since this joint is articulated in the direction of axis $\mathbf{z}_i$.

For a *revolute joint* we have

$$\omega_i = \omega_{i-1} + \dot{\theta}_i\mathbf{z}_{i-1}$$

and

$$\dot{\mathbf{p}}_i = \dot{\mathbf{p}}_{i-1} + \omega_i \times \mathbf{r}_{i-1,i}$$

6.5.2 Jacobian Computation

Consider the expression $\dot{\mathbf{p}}_e(\mathbf{q})$ relating the end-effector position to the joint variables. Differentiating this yields

$$\dot{\mathbf{p}}_e = \sum_{i=1}^{n} \frac{\partial \mathbf{p}_e}{\partial q_i} \dot{q}_i = \sum_{i=1}^{n} J_{Pi} \dot{q}_i$$

due to the chain rule. Thus, the linear velocity of the end-effector can be obtained as the sum of n terms, each of which represents the contribution of a single joint to the end-effector linear velocity when all other joints are still.

Thus, we have $J_{Pi} = \mathbf{z}_{i-1}$ for a prismatic joint and $J_{Pi} = \mathbf{z}_{i-1} \times (\mathbf{p}_e - \mathbf{p}_{i-1})$ for a revolute joint.

The angular velocity of the end-effector is given by

$$\omega_e = \omega_n = \sum_{i=1}^{n} \omega_{i-1,i} = \sum_{i=1}^{n} J_{Oi} \dot{q}_i$$

and thus for a prismatic joint $J_{Oi} = 0$ and for a revolute joint $J_{Oi} = \mathbf{z}_{i-1}$.

In summary, the full Jacobian is formed from 3×1 vectors J_{Pi} and J_{Oi} as

$$J = \begin{pmatrix} J_{P1} & J_{P2} & \cdots & J_{Pn} \\ & & & \\ J_{O1} & J_{O2} & \cdots & J_{On} \end{pmatrix}$$

where we have

$$\begin{pmatrix} J_{Pi} \\ J_{Oi} \end{pmatrix} = \begin{cases} \begin{pmatrix} \mathbf{z}_{i-1} \\ \mathbf{0} \end{pmatrix} & \text{for a } \textit{prismatic} \text{ joint} \\ \begin{pmatrix} \mathbf{z}_{i-1} \times (\mathbf{p}_e - \mathbf{p}_{i-1}) \\ \mathbf{z}_{i-1} \end{pmatrix} & \text{for a } \textit{revolute} \text{ joint} \end{cases}$$

The vectors on which the Jacobian depends are functions of the joint variables and can be computed from the direct kinematics relations as follows.

- $\mathbf{z}_{i-1}$ is given by the third column of the rotation matrix R_{i-1}^0; if $z_0 = [0, 0, 1]^T$ then $\mathbf{z}_{i-1} = R_1^0(q_1) \cdots R_{i-1}^{i-2}(q_{i-1})\mathbf{z}_0$.
- $\mathbf{p}_e$ is given by the first three elements of the fourth coumn of the homogeneous transformation matrix $T_e^0 = A_1^0(q_1) \cdots A_n^{n-1}(q_n)$.
- $\mathbf{p}_{i-1}$ is given by the first three elements of the fourth column of the homogeneous transformation matrix $T_{i-1}^0 = A_1^0(q_1) \cdots A_{i-1}^{i-2}(q_{i-1})$

Finally, note that the Jacobian has been developed here to describe the end-effector velocities with respect to the base frame. If it is desired to represent the Jacobian with respect to a different Frame u then the relation is

$$J^u = \begin{pmatrix} R^u & 0 \\ 0 & R^u \end{pmatrix} J$$

Kinematic Singularities

To summarize the preceding section, the Jacobian (geometric or analytical) defines the linear mapping

$$\mathbf{v}_e = J(\mathbf{q})\dot{\mathbf{q}}$$

between the joint velocities $\dot{\mathbf{q}}$ and end-effector velocities $\mathbf{v}_e = [\dot{\mathbf{p}}_e^T, \omega_e^T]^T$. It is, in general, a function of the configuration of the manipulator - that is, $J = J(\mathbf{q})$; as a result, there may be certain configurations in which the Jacobian becomes rank-deficient. These are called *kinematic singularities*.

Such configurations may represent points at which

- The mobility of the manipulator is reduced.
- Infinite solutions to the inverse problem may exist (there is redundancy present).
- Small operational-space velocities may cause large joint space velocities.

As a result, it is desired to be aware of singularities for a given manipulator so as to avoid them if possible. There are two types of singularities: *boundary* singularities which occur when the manipulator is at the boundary of its workspace and *internal* singularities which occur inside the workspace for particular end-effector configurations. Boundary singularities can be avoided if one knows the limits of the workspace but internal singularities can occur anywhere in the workspace.

Since a matrix becomes non-invertible when rank-deficient, it is theoretically possible to determine a manipulator's internal singularities by analyzing when the determinant of the Jacobian goes to zero. This, however, is only useful in practice for simple manipulators.

Redundancy

As introduced previously, the redundancy of a manipulator is a function of the number n of DOFs of the structure, the number m of operational space variables and the number r of operational space variables necessary to specify a given task. In this case it is more useful to consider

$$\mathbf{v}_e = J(\mathbf{q})\dot{\mathbf{q}}$$

where $\mathbf{v}_e \in R^{r\times 1}$ is the vector of operational space variables needed for the given task, $J(\mathbf{q}) \in r \times n$ is the relevant portion of the geometric Jacobian and $\dot{\mathbf{q}} \in R^{n\times 1}$ is the vector of joint velocities (this vector remains the same).

Thus, if $r < n$ then the system is *underdetermined* or redundant; there are more variables than are needed to specify the task. Assuming full (row) rank, the dimensions of the column space $R(J)$ and row space $R(J^T)$ are only r and thus since

$$dim(R(J)) + dim(N(J^T)) = r$$
$$dim(R(J^T)) + dim(N(J)) = n$$

we have that the dimension of the nullspace is $n-r$. There are thus $n-r$ dependent columns in J and therefore infinite solutions for the joint velocities given desired end-effector velocities in the inverse problem. The solution is thus of the form

$$\dot{\mathbf{q}} = \dot{\mathbf{q}} * + R\dot{\mathbf{q}}_0$$

where $P \in R^{n \times n}$ is a matrix specifying a basis for $N(A)$ and $\dot{\mathbf{q}}_0 \in R^{n \times 1}$ is an arbitrary vector of joint velocities. This vector maps to the nullspace through the Jacobian and thus does not affect the end-effector velocity; we can exploit this by choosing $\dot{\mathbf{q}}_0$ to generate desireable *internal motions* which do not change $\mathbf{v}_e$.

NOTE: I don't understand why P would be $n \times n$ when the dimension of $N(A)$ is only $n - r$. Why not $P \in R^{n \times n-1}$? I understand this would make $\dot{\mathbf{q}}_0$ a different size than $\dot{\mathbf{q}}$ but the P presented itself is redundant which seems strange.

Note that the concepts of singularities, intrinsic redundancy and functional redundancy are all closely related. A manipulator which has more DOFs than are needed to fully specify the end-effector pose ($n > m$) is intrinsically redundant; the corresponding system is underdetermined. A manipulator which has $m = n$ can become functionally redundant when only $r < m$ operational space variables are specified for a task; this system is also underdetermined. Finally, when an intrinsically-redundant manipulator ($n > m$) reaches a singularity, it becomes even more redundant.

In all of this discussion we have not mentioned the case in which $m > n$ - that is, the *overdetermined* case. In this situation there can only be a solution to the inverse problem if the desired operational space velocity lies within the column space $R(J)$; otherwise, there is no solution. We can consider a least-squares solution by projecting the desired velocity onto $R(A)$ and solving instead for the joint velocities corresponding to this approximation but this may not be useful. Assuming that the manipulator is designed to have $n \geq m$ DOFs, the corresponding system may become overdetermined only when $n = m$ and the manipulator reaches a singularity.

6.5.3 Inverse Differential Kinematics

In a previous section, we derived the solution to the direct kinematics problem and saw that, in general, it is a nonlinear function of the joint positions $\mathbf{q}$; the relation was

$$\mathbf{x}_e = k(\mathbf{q})$$

Given that this system is nonlinear, one can solve the corresponding inverse solution in closed form only for simple manipulators.

Instead, it would be much preferred to work with a system which is *linear* in the joint variables. As we have seen, the differential kinematics problem is described by

$$\mathbf{v}_e = J(\mathbf{q})\dot{\mathbf{v}}_e$$

and is thus linear in the joint velocities $\dot{\mathbf{q}}$. The fact that the Jacobian itself is a (likely nonlinear) function of the joint variables $\mathbf{q}$ is not an issue. We can thus solve the inverse problem through the differential kinematics formulation instead, solving directly for $\dot{\mathbf{q}}$ and (numerically) integrating the solution to obtain $\mathbf{q}$ (assuming the initial joint state $\mathbf{q}(0)$ is known).

Of course, if the dimension of the operational space relevant to the given task is $r = n$ then the Jacobian is square and full rank and thus

$$\dot{\mathbf{q}} = J^{-1}(\mathbf{q})\mathbf{v}_e$$

is the solution obtained through inversion of the Jacobian.

Inverse Kinematics and Redundancy

The above solution is only valid when the Jacobian is full rank; if the manipulator is redundant then $r < n$ and the system is underdetermined. As is known for such systems, there are infinite solutions for $\dot{\mathbf{q}}$; one logical choice is to choose the solution which satisfies the differential kinematics relation and minimizes the quadratic cost function

$$g(\dot{\mathbf{q}}) = \frac{1}{2}\dot{\mathbf{q}}^T W \dot{\mathbf{q}}$$

where $W \in R^{n\times n}$ is a positive definite weighting matrix. This can be expressed as the minimization of the Lagrangian (modified cost function)

$$g(\dot{\mathbf{q}}, \lambda) = \frac{1}{2}\dot{\mathbf{q}}^T W \dot{\mathbf{q}} + \lambda^T(\mathbf{v}_e - J\dot{\mathbf{q}})$$

with respect to the joint velocity. Note that the differential kinematics relation is expressed as an equality constraint; this ensures it will be satisfied. In this way, the minimization of the cost function becomes a secondary objective in the optimization.

The resulting solution is

$$\dot{\mathbf{q}} = W^{-1}J^T(JW^{-1}J^T)^{-1}\mathbf{v}_e$$

If the weighting matrix is simply chosen to be the identity matrix in R^n (diagonal W with equal weights for all joint velocities) then the above solution becomes the *minimum norm* solution

$$\dot{\mathbf{q}} = J^{\dagger}\mathbf{v}_e$$

where

$$J^\dagger = J^T(JJ^T)^{-1}$$

is the *right pseudoinverse* of J. The obtained solution thus satisfies the differential kinematics relation and has the minimum norm among all possible solutions (see the section on underdetermined systems for more information).

As mentioned previously, since there are $n-r$ dependent columns in J we are free to choose a solution of the form $\dot{\mathbf{q}} = \dot{\mathbf{q}}* + P\dot{\mathbf{q}}_0$ where the columns of P are in the nullspace of J...

We must note that the above solutions are only valid when J has full row rank r; in this case, the dimension of the left nullspace $N(J^T)$ of J is zero because the rows of J are independent. Thus $dim(R(J)) = r$ and the columns of J span all of R^r (the fact that they are redundant doesn't change this).

If J is rank-deficient then the left nullspace becomes non-null; the column space no longer spans R^r and therefore there are no solutions to the inverse problem unless $\mathbf{v}_e$ happens to lie within the subspace of R^r defined by $R(J)$ (and then there are infinite solutions due to redundancy).

Note that this differs from the case in which $r < n$ which can (theoretically) be solved by least-squares because here the rank is less than both r and n (least squares requires $r > n$ and full column rank). In short, there are infinite solutions to the least-squares approximation (the problem in which $\mathbf{v}_e$ is projected onto $R(J)$). *This occurs when a redundant manipulator is at a configuration singularity.*

Even if singularities are avoided, there are still issues with the inverse problem when the manipulator is in the *neighborhood* of a singularity. When the Jacobian is nearly singular its condition number $K(J)$ becomes very large; the right-pseudoinverse involves the term $(JJ^T)^{-1}$ which has a condition number of approximately $K(J)^2$, making the solution prone to error. In this case one might use a QR-based psuedoinverse or the *damped (regularized) least-squares inverse*

$$J^* = J^T(JJ^T + k^2I)^{-1}$$

which results from minimizing the sum of the norm of the residual $\mathbf{v}_e - J(\dot{\mathbf{q}})\dot{\mathbf{q}}$ (as is done in the overdetermined case) plus the norm of the joint velocities (as is done in the underdetermined case). The parameter k is used to establish the relative weight between these two optimization objectives. Note that this is both the unique solution in the rank-deficient case and a "better-conditioned" solution in the nearly-singular case. Whereas the original problem would result in large joint velocities in the neighborhood of a singularity, the damped (regularized) problem keeps results in smaller velocities by essentially putting a prior on the solution which says that the joint velocities should be small (close to zero).

Note that the damped left-pseudoinverse

$$J^* = (J^TJ + k^2I)^{-1}J^T$$

is equal to the damped right-pseudoinverse. The reason we choose the right inverse is because when $J \in R^{m \times n}$ with $n > m$ then $(JJ^T + k^2 I) \in R^{m \times m}$ is more efficient to invert than $(J^T J + k^2 I) \in R^{n \times n}$.

Jacobian Transpose Method for Inverse Kinematics:

Consider the inverse kinematics problem

$$\dot{x} = J\dot{q}$$

which entails solving for the joint trajectory $\dot{q}$ corresponding to a desired end-effector trajectory $\dot{x}$. Suppose that instead of a desired trajectory, we wish to move from a point x to a point g representing the goal. Define the error to be $e = g - x$ and then choose

$$\Delta\theta = \alpha J^T e$$

(where α is a positive constant and J is the manipulator Jacobian) in order to update θ at each timestep. Recall that the relationship between forces on the end effector and joint torques is given by

$$\tau = J^T F$$

The Jacobian Transpose method thus simulates the effect of a sprint with constant α pulling the end-effector towards the goal! Assuming that a small change Δs in end-effector position corresponds to a small change $\Delta\theta$ in joint positions through the differential relationship $\Delta s = J\Delta\theta$, we choose α at each timestep in order to make $\Delta s = \alpha JJ^T e$ as close as possible to e. In other words, we choose α according to

$$\underset{\alpha}{\operatorname{argmin}} \, ||e - \alpha JJ^T e||^2$$

from which we find

$$\begin{aligned} ||e - \alpha JJ^T e||^2 &= (e - \alpha JJ^T e)^T (e - \alpha JJ^T e) \\ &= (e^T - \alpha e^T JJ^T)(e - \alpha JJ^T e) \\ &= e^T e - 2\alpha e^T JJ^T e + \alpha^2 e^T JJ^T JJ^T e \end{aligned}$$

We can't do anything to reduce the first term, so we solve for α such that the other terms cancel. We have

$$2\alpha ||J^T e||^2 = \alpha^2 ||JJ^T e||^2$$

from which it follows that (neglecting the factor of two)

$$\alpha = \frac{||J^T e||^2}{||JJ^T e||^2}$$

6.5.4 Analytical Jacobian

Whereas the geometric Jacobian was computed in a systematic manner for a manipulator by considering the contribution of each joint velocity to the end-effector velocity, the analytical Jacobian is computed directly from differentiation of the direct kinematics equation

$$\mathbf{x}_e = \begin{pmatrix} \mathbf{p}_e \\ \phi_e \end{pmatrix}$$

If this end-effector pose is known in terms of a minimal number of operational space variables (we have $m = 6$ for a manipulator in R^3) then

$$\dot{\mathbf{p}}_e = \frac{\partial \mathbf{p}_e}{\partial \mathbf{q}} \dot{\mathbf{q}} = J_p \dot{\mathbf{q}}$$

specifies the translational velocity of the end-effector frame with respect to the base frame and

$$\dot{\phi}_e = \frac{\partial \phi_e}{\partial \mathbf{q}} \dot{\mathbf{q}} = J_\phi \dot{\mathbf{q}}$$

specifies the time derivative of the orientation of the end-effector frame in the chosen minimal representation. Note that, in general, $\dot{\phi}_e \neq \omega$ and that computation of $J_\phi(\dot{\mathbf{q}})$ usually goes through the computation of the rotation matrix since $\phi_e(\mathbf{q})$ is not usually available in direct form.

The differential kinematics equation obtained from differentiating the direct kinematics equation is thus

$$\dot{\mathbf{x}}_e = \begin{pmatrix} \dot{\mathbf{p}}_e \\ \dot{\phi}_e \end{pmatrix} = \begin{pmatrix} J_P(\mathbf{q}) \\ J_\phi(\mathbf{q}) \end{pmatrix} = J_A(\mathbf{q}) \dot{\mathbf{q}}$$

where

$$J_A(\mathbf{q}) = \frac{\partial k(\mathbf{q})}{\partial \mathbf{q}}$$

is the *analytical Jacobian* which is different from the geometric Jacobian J because $\dot{\phi}_e \neq \omega_e$.

Note, however, that we can transform between the time derivative of the minimal representation of orientation and the angular velocity as

$$\omega_e = T(\phi_e) \dot{\phi}_e$$

where the matrix $T(\phi_e)$ is a transformation matrix specific to the minimal representation used.

While the time derivative of end-effector orientation is represented differently in the two Jacobians, the linear velocity of the end-effector is the same for both. Thus, we can transform between the two representations using

$$\mathbf{v}_e = T_A(\phi)\dot{\mathbf{x}}_e$$

where

$$T_A(\phi) = \begin{pmatrix} I & 0 \\ 0 & T(\phi_e) \end{pmatrix}$$

transforms between end-effector pose representations. Thus, we have that the transformation between Jacobians is

$$J = T_A(\phi)J_A$$

6.5.5 Inverse Kinematics Algorithms

6.6 Principle of Virtual Work

The main idea behind the principle of virtual work is that nature in some way optimizes the trajectory resulting from the differential equations which describe the motion of a physical system. It is derived from the *principle of least action*.

Consider the motion of a particle between two points in space. If a force acts on this particle as it moves then one can compute the work done by the force along any possible path between the two endpoints. The principle of virtual work states that he path which the particle actually follows is the one for which the difference between the work done on this path and the work done on nearby paths is zero. This implies that the work done is minimized by this path; this analysis thus involves the computation of the difference in a function evaluated on nearby paths which is a generalization of the derivative and is formalized by the branch termed the *calculus of variations*.

Consider a particle which moves along a trajectory $r(t)$ from point A to point B due to an applied force F. The total work done by the force along this path is then

$$\int_A^B F \cdot dr$$

where dr is a differential element along the curve $r(t)$. Written in terms of the velocity $v(t)$ along the path, the expression for the work done is

$$\int_{t_0}^{t_1} F \cdot v dt$$

Now consider the work done in moving between the same endpoints but along a slightly different path given by

$$\bar{r}(t) = r(t) + \delta r(t)$$

where $\delta r(t) = \epsilon h(t)$ Here ϵ is a positive scaling constant and $h(t)$ is an arbitrary function with $h(t_0) = h(t_1) = 0$ so that $\bar{r}(t) = r(t)$ at the endpoints. The work done along this path is then

$$\bar{W} = \int_A^B F \cdot d(r + \epsilon h) = \int_{t_0}^{t_1} F \cdot (v + \epsilon \dot{h}) dt$$

The difference in the work between the two paths is then

$$\delta W = W - \bar{W} = \int_{t_0}^{t_1} (F \cdot \epsilon \dot{h}) dt$$

Assuming that $r(t)$ and $h(t)$ depend on the generalized coordinates $q_i, i = 1, \cdots, n$ then the derivative of the variation $\delta r(t) = \epsilon h(t)$ is given by

$$\frac{d}{dt}\delta r = \epsilon \dot{h} = \epsilon \left(\frac{\partial h}{\partial q_1} \dot{q_1} + \cdots + \frac{\partial h}{\partial q_n} \dot{q_n} \right)$$

where the chain rule has been used. We thus have

$$\delta W = \int_{t_0}^{t_1} \left(F \cdot \frac{\partial h}{\partial q_1} \epsilon \dot{q_1} + \cdots + F \cdot \frac{\partial h}{\partial q_n} \epsilon \dot{q_n} \right) dt = \int_{t_0}^{t_1} \left(F \cdot \frac{\partial h}{\partial q_1} \right) \epsilon \dot{q_1} dt + \cdots + \int_{t_0}^{t_1} \left(F \cdot \frac{\partial h}{\partial q_n} \right) \epsilon \dot{q_n} dt$$

Therefore, in order for the virtual work δW to be zero for an arbitrary variation we require that

$$F_i = F \cdot \frac{\partial h}{\partial q_i} = 0, \quad i = 1, \ldots, n$$

where the terms F_i are called the *generalized forces* associated with the virtual displacement.

6.7 D'Alembert's Principle of Virtual Work

The virtual work of the N particle system is computed as the sum of the dot product of each force with the virtual displacement of its point of application

$$\delta W = \sum_{i=1}^{N} F_i \cdot \delta r_i$$

WHY THE SUM?? WHAT HAPPENED TO INTEGRATION??

If the trajectory of the rigid body is defined by the M generalized coordinates q_j then

$$\delta r_i = \sum_{j=1}^{M} \frac{\partial r_i}{\partial q_j} \delta q_j = \sum_{j=1}^{M} \frac{\partial v_i}{\partial \dot{q}_j} \delta q_j$$

where, as before, we have from kinematics that

$$v_i = v + \omega \times (r_i - 1)$$

The virtual work of the system is then

$$\delta W = F_1 \cdot \sum_{j=1}^{M} \frac{\partial v_1}{\partial \dot{q}_j} \delta q_j + \cdots + F_N \cdot \sum_{j=1}^{M} \frac{\partial v_N}{\partial \dot{q}_j} \delta q_j$$

or, when written in terms of coefficients of δq_j,

$$\delta W = \left(\sum_{i=1}^{N} F_1 \cdot \frac{\partial v_i}{\partial \dot{q}_1} \right) \delta q_1 + \cdots + \left(\sum_{i=1}^{N} F_N \cdot \frac{\partial v_i}{\partial \dot{q}_N} \right) \delta q_N$$

Consider the trajectory of a rigid body which is specified by a single generalized coordinate q. The virtual work is then

$$\begin{aligned}
\delta W &= \left(\sum_{i=1}^{N} F_i \cdot \frac{\partial v_i}{\partial \dot{q}} \right) \delta q \\
&= \left(\sum_{i=1}^{N} \left[F_i \cdot \frac{\partial (v + \omega \times (r_i - r))}{\partial \dot{q}} \right] \right) \delta q \\
&= \left(\sum_{i=1}^{N} \left[F_i \cdot \frac{\partial v}{\partial \dot{q}} + (r_i - r) \times F_i \cdot \frac{\partial \omega}{\partial \dot{q}} \right] \right) \delta q \\
&= \left(\left[\sum_{i=1}^{N} F_i \right] \cdot \frac{\partial v}{\partial \dot{q}} + \left[\sum_{i=1}^{N} (r_i - r) \times F_i \right] \cdot \frac{\partial \omega}{\partial \dot{q}} \right) \delta q
\end{aligned}$$

Using the definitions of the resultant force and torque this reduces to

$$\delta W = Q \delta q$$

where

$$Q = \left(F \cdot \frac{\partial v}{\partial \dot{q}} + T \cdot \frac{\partial \omega}{\partial \dot{q}} \right)$$

is the *generalized force* associated with the virtual displacement. If an applied force is conservative (work is path-independent) then the corresponding generalized force can be described from the potential function $V(q)$ (potential energy) to be

$$Q = -\frac{\partial V}{\partial q}$$

since a conservative force is equal to the negative of the gradient of its corresponding potential function. If the trajectory is defined by more than one generalized coordinate we have

$$\delta W = \sum_{i=1}^{M} Q_j \delta q_j$$

where

$$Q_j = F \cdot \frac{\partial v}{\partial \dot{q}_j} + T \cdot \frac{\partial \omega}{\partial \dot{q}_j}$$

For a mechanical system of B rigid bodies - assuming for the present that the dynamics of each rigid body are described by only one generalized coordinate q - the total virtual work is

$$\delta W = \sum_{i=1}^{B} F_i \cdot \frac{\partial v_i}{\partial \dot{q}} + T_i \cdot \frac{\partial \omega_i}{\partial \dot{q}} = Q \delta q$$

where Q is the generalized force acting on the system. If instead the system is defined by M generalized coordinates (the system has M degrees of freedom) then the virtual work is given by

$$\delta W = \sum_{j=1}^{M} Q_j \delta q_j$$

where

$$Q_j = \sum_{i=1}^{B} \left(F_i \cdot \frac{\partial v_i}{\partial \dot{q}_j} + T_i \cdot \frac{\partial \omega_i}{\partial \dot{q}_j} \right)$$

is the generalized force associated with the j^{th} generalized coordinate q_j.

The principle of virtual work states that *static equilibrium* occurs when these generalized forces acting on the system are zero, ie

$$Q_j = 0, \quad j = 1, \ldots, M$$

However, one can extend this principle to apply to a system of rigid bodies in motion by considering the *generalized inertia forces* acting on the system at *dynamic equilibrium.*

Consider a single rigid body translating under the influence of a resultant force F and rotating under the influence of a resultant torque T. Again, assume that this body has dynamics described by a single generalized coordinate q. The *generalized inertia force* associated with this coordinate is then

$$\hat{Q} = -(Ma) \cdot \frac{\partial v}{\partial \dot{q}} - ([I_R]\alpha + \omega \times [I_R]\omega) \cdot \frac{\partial \omega}{\partial \dot{q}}$$

where the expressions for F and T have been used and the negative sign comes from "moving the inertia terms over to the other side". The kinetic energy of a rigid body is

$$T = \frac{1}{2} M v \cdot v + \frac{1}{2} \omega \cdot [I_R]\omega$$

and thus the generalized inertia force can be computed from the kinetic energy as

$$\hat{Q} = -\left(\frac{d}{dt}\frac{\partial T}{\partial \dot{q}} - \frac{\partial T}{\partial q}\right)$$

In the case of a system of B rigid bodies with M generalized coordinates the kinetic energy is

$$T = \sum_{i=1}^{B}\left[\frac{1}{2}Mv_i \cdot v_i + \frac{1}{2}\omega_i \cdot [I_R]\omega_i\right]$$

and the M generalized inertia forces can be computed as

$$\hat{Q}_j = -\left(\frac{d}{dt}\frac{\partial T}{\partial \dot{q}_j} - \frac{\partial T}{\partial q_j}\right)$$

This system of rigid bodies is said to be in *dynamic equilibrium* when the virtual work of the sum of the generalized applied forces Q and the generalized inertia forces $\hat{Q}$ is zero for any virtual displacement δq. Thus, for a system of B bodies described by M generalized coordinates

$$\delta W = \sum_{j=1}^{M}(Q_j + \hat{Q}_j) = 0$$

yields the conditions for dynamic equilibrium

$$Q_j + \hat{Q}_j = 0, \quad j = 1, \ldots, M$$

Using the definition of the generalized inertia forces given above, this can be written as

$$\frac{d}{dt}\frac{\partial T}{\partial \dot{q}_j} - \frac{\partial T}{\partial q_j} = Q_j, \quad j = 1, \ldots, M$$

This is thus a set of M equations which describe the dynamics of the rigid body system. If the applied generalized forces are derivable from potential functions then the above becomes

$$\frac{d}{dt}\frac{\partial T}{\partial \dot{q}_j} - \frac{\partial T}{\partial q_j} = \frac{\partial V}{\partial q_j}, \quad j = 1, \ldots, M$$

If we define the *Lagrangian* to be

$$L = T - V$$

Then the above equations can be written

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}_j} - \frac{\partial L}{\partial q_j} = 0, \quad j = 1, \ldots, M$$

These are known as *Lagrange's Equations of Motion*. Non-conservative generalized forces applied to the system can be included using the form

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}_j} - \frac{\partial L}{\partial q_j} = Q_j, \quad j = 1, \ldots, M$$

where from before

$$Q_j = F \cdot \frac{\partial v}{\partial \dot{q}_j} + T \cdot \frac{\partial \omega}{\partial \dot{q}_j}$$

is the general expression for an applied generalized force.

6.8 Kinematics

Consider a point P whose position in a fixed Frame 0 is defined to be p^0 and in a non-fixed (translating and rotating) Frame 1 to be p^1. Let R_1^0 denote the rotation matrix representing the orientation of Frame 1 with respect to Frame 0 and let o_1^0 denote the position of the origin of Frame 1 with respect to Frame 0. The expression for the point P in Frame 0 can then be written as

$$\mathbf{p}^0 = \mathbf{o}_1^0 + R_1^0 \mathbf{p}^1$$

Differentiating this expression with respect to time yields

$$\begin{aligned}\dot{\mathbf{p}}^0 &= \dot{\mathbf{o}}_1^0 + R_1^0 \dot{\mathbf{p}}^1 + \dot{R}_1^0 \mathbf{p}^1 \\ &= \dot{\mathbf{o}}_1^0 + R_1^0 \dot{\mathbf{p}}^1 + S(\omega_1^0) R_1^0 \mathbf{p}^1 \\ &= \dot{\mathbf{o}}_1^0 + R_1^0 \dot{\mathbf{p}}^1 + \omega_1^0 \times \mathbf{r}_1^0\end{aligned}$$

where the definition of the derivative of a rotation matrix has been used to yield $\dot{R}_1^0 = S(\omega_1^0) R_1^0$.

Here the vector $\mathbf{r}_1^0 = R_1^0 \mathbf{p}^1$ represents the point P after it has been rotated into Frame 0 but not translated, ie $r_1^0 = p^0 - o_1^0$. This is known as the *velocity composition rule*.

If the point P is fixed in Frame 1 then this reduces to

$$\dot{\mathbf{p}}^0 = \dot{\mathbf{o}}_1^0 + \omega_1^0 \times \mathbf{r}_1^0$$

In order to find the acceleration of P with respect to Frame 0 we differentiate again, yielding

$$\begin{aligned}\ddot{\mathbf{p}}^0 &= \ddot{\mathbf{o}}_1^0 + \dot{\omega}_1^0 \times \mathbf{r}_1^0 + \omega_1^0 \times \dot{\mathbf{r}}_1^0 \\ &= \ddot{\mathbf{o}}_1^0 + \alpha_1^0 \times \mathbf{r}_1^0 + \omega_1^0 \times (R_1^0 \dot{\mathbf{p}}^1 + \dot{R}_1^0 \mathbf{p}^1)\end{aligned}$$

Again, since it is assumed that $\mathbf{p}^1$ is fixed and using the expression for the derivative of a rotation matrix we have

$$\ddot{\mathbf{p}}^0 = \ddot{\mathbf{o}}_1^0 + \alpha_1^0 \times \mathbf{r}_1^0 + \omega_1^0 \times (\omega_1^0 \times \mathbf{r}_1^0)$$

6.9 Rigid Body Dynamics

6.9.1 Manipulator Dynamic Model (Lagrange Formulation)

Consider a manipulator composed of N rigid links, each of which is a continuum of particles of infinitesimal mass. The goal is to derive a *dynamic model* which relates the generalized forces applied to the joints of the manipulator and their corresponding accelerations.

The *Lagrange Formulation* states that such a system is governed by N ordinary differential equations, the n^{th} of which is given by

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_n}\right) - \frac{\partial L}{\partial q_n} = \xi_n \quad \forall n \in \{1, \ldots, N\}$$

where the *Lagrangian* of the system is given by

$$L = T - U$$

where T is the total *kinetic* energy of the system and U is the total *potential* energy of the system.

For an N-link manipulator, the total kinetic energy is given by

$$T = \sum_{n=1}^{N} T_{ln}$$

where T_{ln} denotes the kinetic energy of the n^{th} link. Note that this derivation will neglect the effects of the motor (or other actuator). If p_i denotes the position of the i^{th} particle belonging to the n^{th} link then the kinetic energy of this link can be expressed as

$$T_{ln} = \frac{1}{2}\int_{V_n} \dot{p}_i^T \dot{p}_i \rho dV$$

where the link is assumed to be of uniform density ρ. This is thus a triple integral over the link volume.

Consider the position vector p_{Cn} of the center of mass of the link; the position of the i^{th} particle can thus be expressed as

$$p_i = p_{Cn} + r_i$$

where r_i is the vector from the center of mass to the particle. Note here that the center of mass is computed as

$$p_{Cn} = \frac{1}{m_n} \int_{V_n} p_i \rho dV$$

where m_n is the total mass of the link as given by

$$m_n = \int_{V_n} \rho dV$$

From kinematics, the velocity of the i^{th} particle can be expressed as

$$\begin{aligned} \dot{p}_i &= \dot{p}_{C_n} + \omega_n \times r_i \\ &= \dot{p}_{C_n} + S(\omega_n) r_i \end{aligned}$$

where $S(\bullet)$ denotes the skew-symmetric operator and ω_n is the angular velocity of the link with respect to the base frame. Note that the angular velocity is a *free vector* and is the same for all points on the rigid body.

Using this expression for the particle velocity in the expression for the total link kinetic energy yields

$$T_n = \frac{1}{2} \int_{V_n} \left(\dot{p}_{C_n} + S(\omega_n) r_i\right)^T \left(\dot{p}_{C_n} + S(\omega_n) r_i\right) dV$$

Expanding the above expression out yields

$$T_n = \frac{1}{2} \int_{V_n} \left(\dot{p}_{C_n}^T \dot{p}_{C_n} + \dot{p}_{C_n}^T S(\omega_n) r_i + (S(\omega_n) r_i)^T \dot{p}_{C_n} + (S(\omega_n) r_i)^T S(\omega_n) r_i\right) \rho dV$$

The kinetic energy integral can thus be split into four integrals each of which represent a different type of kinetic energy.

Translational:

The translational contribution to the total kinetic energy

$$T_n = \frac{1}{2} \int_{V_n} \dot{p}_{C_n}^T \dot{p}_{C_n} \rho dV$$

is due to the translation of the center of mass of the link. Since the location of the link center of mass does not depend on position within the body it can be pulled outside the integral to yield

$$T_n = \dot{p}_{C_n}^T \dot{p}_{C_n} \int_{V_n} \rho dV = \frac{1}{2} m_n \dot{p}_{C_n}^T \dot{p}_{C_n}$$

If the body were only translating, this would be its total kinetic energy.

Rotational:

The kinetic energy contribution due to the rotation of the rigid body is

$$T_n = \frac{1}{2}\int_{V_n} r_i^T S(\omega_n)^T S(\omega_n) r_i \rho dV$$

This contribution is due to the motion of the particle relative to the translation of the center of mass (in a frame fixed to the COM the relative position of the particle would not remain constant due to the body's rotation).

Exploiting the fact that

$$S(\omega_n)r_i = \omega_n \times r_i = -r_i \times \omega_n = -S(r_i)\omega_n$$

and the fact that the angular velocity can be pulled outside the integral, the rotational kinetic energy can be expressed as

$$\begin{aligned} T_n &= \frac{1}{2}\omega_n^T \left(\int_{V_n} S(r_i)^T S(r_i)\rho dV\right)\omega_n \\ &= \frac{1}{2}\omega_n^T \bar{I}_n \omega_n \end{aligned}$$

The quantity inside the parentheses is defined to be the *inertia tensor* relative the center of mass of the link. That is,

$$\begin{aligned} \bar{I}_n &= \int_{V_n} S(r_i)^T S(r_i)\rho dV \\ &= \begin{pmatrix} \int (r_{iy}^2 + r_{iz}^2)dV & -\int r_{ix}r_{iy}dV & -\int r_{ix}r_{iz}dV \\ * & \int (r_{ix}^2 + r_{iz}^2)dV & -\int r_{iy}r_{iz}dV \\ * & * & \int (r_{ix}^2 + r_{iy}^2)dV \end{pmatrix} \\ &= \begin{pmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ * & I_{yy} & -I_{yz} \\ * & * & I_{zz} \end{pmatrix} \end{aligned}$$

where elements have been omitted for notational simplicity since the inertia matrix is symmetric (also positive-definite).

Body-Frame Inertia Matrix:

One complication with this quantity is that it is configuration-dependent since the vector r_i of the position of the i^{th} particle relative the the center of mass of the link is a function of the link orientation. We therefore seek to write $\bar{I}_n$ as a function of a constant *body-frame* inertia matrix $\bar{I}_n^n$ and the link orientation R_n (which is defined w/r/t the base frame).

The angular velocity of the link with respect to the base as expressed in the base frame is $\omega_{0,n}^0 = \omega_n$. This angular velocity can instead be expressed in the link frame as

$$\omega_n^n = R_i^T \omega_n$$

We can thus write

$$S(\omega_n) r_i = \omega_n \times r_i = R\omega_n^n \times R r_i^n = R(-r_i^n \times \omega_n^n) = -RS(r_i^n)\omega_n^n$$

where r_i^n denotes the position of the particle relative to the link center of mass *in the body frame*. Since the body is rigid, this vector is constant!

The expression for the link kinetic energy can then be written as

$$\begin{aligned} T_n &= \frac{1}{2} \int_{V_n} \omega_n^{nT} S(r_i^n)^T R^T R S(r_i^n) \omega_n^n \rho dV \\ &= \frac{1}{2} \omega_n^{nT} \left(\int_{V_n} S(r_i^n)^T S(r_i^n) \rho dV \right) \omega_n^n \\ &= \frac{1}{2} \omega_n^{nT} \bar{I}_n^n \omega_n^n \\ &= \frac{1}{2} \omega_n^T R_n \bar{I}_n^n R_n^T \omega_n \end{aligned}$$

We thus have the relation

$$\bar{I}_n = R_n \bar{I}_n^n R_n^T$$

where the *body-frame inertia matrix* is defined as

$$\bar{I}_n^n = \int_{V_n} S(r_i^n)^T S(r_i^n) \rho dV$$

Since r_i^n is a constant vector, the resulting matrix is constant. Note also that for planar problems (such as the two-link planar arm, for example) the angular velocity vector is orthogonal to the plane and thus the inertia matrices are the same (and constant).

Mutual:

The final contribution to the total kinetic energy of the link comes from a so-called *mutual* term which involves both translation of the COM and rotation relative to the COM. It is

$$T_n = \frac{1}{2} \int_{V_n} \left(\dot{p}_{C_n}^T S(\omega_n) r_i + (S(\omega_n) r_i)^T \dot{p}_{C_n} \right) \rho dV = 2 \left(\frac{1}{2} \int_{V_n} \dot{p}_{C_n}^T S(\omega_n) r_i \rho dV \right)$$

Note that the velocity of the center of mass and the angular velocity of the link can be pulled out of the integral to yield

$$\begin{aligned} T_n &= \dot{p}_{C_n}^T S(\omega_n) \int_{V_n} r_i \rho dV \\ &= \dot{p}_{C_n}^T S(\omega_n) \int_{V_n} r_i \rho dV \\ &= \dot{p}_{C_n}^T S(\omega_n) \int_{V_n} (p_i - p_{Cn}) \rho dV \\ &= 0 \end{aligned}$$

since

$$\begin{aligned} \int_{V_n} p_i \rho dV - \int_{V_n} p_{Cn} \rho dV &= m_n \left(\frac{1}{m_n} \int_{V_n} p_i \rho dV \right) - p_{Cn} \int_{V_n} \rho dV \\ &= m_n p_{Cn} - m_n p_{Cn} \\ &= 0 \end{aligned}$$

Thus, by choosing the center of mass as the reference point with respect to which the velocities of the individual particles are defined, the contribution of this mutual term to the total kinetic energy disappears!

The total kinetic energy of the link is thus

$$T_n = \frac{1}{2} m_n \dot{p}_{C_n}^T \dot{p}_{C_n} + \frac{1}{2} \omega_n^T R_i \bar{I}_n^n R_i^T \omega_n$$

In order to derive the equations governing the dynamics of the manipulator, we must express the above kinetic energy in terms of the generalized coordinates (joint variables) in order to take the required partial derivatives for Lagrange's formulation.

Using the geometry of the manipulator we can compute the matrix $J^{(l_n)}(q)$ which is the configuration-dependent Jacobian relating the velocity of the center of mass of the n^{th} link to the joint variables. Recall that the Jacobian has the structure

$$J^{(l_n)} = \begin{pmatrix} J_P^{(l_n)} \\ \\ J_O^{(l_n)} \end{pmatrix}$$

where

$$\begin{aligned} \dot{p}_{C_n} &= J_{P_1}^{(l_n)} \dot{q}_1 + J_{P_2}^{(l_n)} \dot{q}_2 + \cdots + J_{P_n}^{(l_n)} \dot{q}_n \\ \omega_n &= J_{O_1}^{(l_n)} \dot{q}_1 + J_{O_2}^{(l_n)} \dot{q}_2 + \cdots + J_{O_n}^{(l_n)} \dot{q}_n \end{aligned}$$

6.9.2 Parallel Axis Theorem (Steiner's Theorem)

Consider a rigid body B whose inertia tensor (matrix) about its center of mass p_C is known; this is given to be

$$\bar{I} =$$

Consider a rigid body B which is composed of N particles P_i each of mass m_i; let a Frame 1 be fixed to the body and a Frame 0 be fixed in the world. Let $\mathbf{r}_i$ denote the position of the particle in the world frame. Newton's second law applied to any such particle in the body yields

$$F_i + \sum_{j=1}^{N} F_{ij} = m_i a_i$$

where F_i is the force applied to the particle and F_{ij} is the internal force which particle P_j exerts on the particle P_i.

In order to simplify the analysis of the motion of the body, we can instead consider the resultant force and resultant torque which produce the same overall translational and rotational motion of the body as the system of particles. These are given by

$$F = \sum_{i=1}^{N} F_i$$
$$T = \sum_{i=1}^{N} (r_i - r) \times F_i$$

with respect to a reference point R where the Frame 1 is fixed. Note here that the resultant force F is the sum of *all* forces acting on the particles - internal and external. Since each of these internal forces has $F_{ij} = -F_{ji}$ by Newton's third law, they vanish from the above expression and the resultant is thus the sum of the external forces only.

We are free to choose *any* point R with respect to which the resultant torque is defined; however, consideration of these equations will lead to a natural choice of reference.

Newton's seconds law for a particle yields

$$F_i = m_i a_i - \sum_{i=1}^{N} F_{ij}$$

and thus the resultant force acting on the body is

$$F = \sum_{i=1}^{N} \left(m_i a_i - \sum_{i=1}^{N} F_{ij} \right)$$

Again, since internal forces cancel, we have

$$F = \sum_{i=1}^{N} m_i a_i$$

The expression for the acceleration a_i of a point P_i as expressed in the world-fixed Frame 0 is, from kinematics,

$$\mathbf{a}_i = \ddot{\mathbf{r}} + \alpha \times (\mathbf{r}_i - \mathbf{r}) + \omega \times (\omega \times (\mathbf{r}_i - \mathbf{r}))$$

where $\ddot{\mathbf{r}}$ denotes the linear acceleration of the origin of the body-fixed reference frame (point R), ω denotes the angular velocity of the body-fixed frame and α denotes the angular acceleration of the body-fixed frame - all with respect to the world frame.

The resultant force is then

$$\begin{aligned} F =& \sum_i m_i a_i \\ =& \sum_i m_i \left[\ddot{\mathbf{r}} + \alpha \times (\mathbf{r}_i - \mathbf{r}) + \omega \times (\omega \times (\mathbf{r}_i - \mathbf{r}))\right] \\ =& \ddot{r} \sum_i m_i + \alpha \times \sum_i m_i(\mathbf{r}_i - \mathbf{r}) + \omega \times (\omega \times \sum_i m_i(\mathbf{r}_i - \mathbf{r})) \end{aligned}$$

where the sum has been moved in all three terms since $\ddot{\mathbf{r}}$, ω and α are describe the motion of the body as a whole and not of individual particles. Clearly, by choosing R such that

$$\sum_i m_i(\mathbf{r}_i - \mathbf{r}) = 0$$

the second and third terms will vanish and we can thus reduce the above expression to

$$F = \ddot{r} \sum_i m_i = M\ddot{\mathbf{r}}$$

where $M = \sum_i m_i$ is the total mass of the body. This choice of R yields

$$\mathbf{r} = \frac{1}{M} \sum_i m_i \mathbf{r}_i$$

which is known as the *center of mass*. Using this expression, the resultant torque becomes

$$\begin{aligned}T &= \sum_i [(\mathbf{r}_i - \mathbf{r}) \times (m_i a_i)] \\ &= \sum_i m_i(\mathbf{r}_i - \mathbf{r}) \times (\ddot{\mathbf{r}} + \alpha \times (\mathbf{r}_i - \mathbf{r}) + \omega \times (\omega \times (\mathbf{r}_i - \mathbf{r}))) \\ &= \left[\sum_i m_i(\mathbf{r}_i - \mathbf{r}) \times \ddot{\mathbf{r}}\right] + \left[\sum_i m_i(\mathbf{r}_i - \mathbf{r}) \times (\alpha \times (\mathbf{r}_i - \mathbf{r}))\right] + \left[\sum_i m_i(\mathbf{r}_i - \mathbf{r}) \times (\omega \times (\omega \times (\mathbf{r}_i - \mathbf{r})))\right]\end{aligned}$$

Swapping around cross products (noting that the cross product does NOT associate) yields

$$T = [I_R]\alpha + \omega \times [I_R]\omega$$

where the *inertia matrix* of the body relative to the reference R (here the center of mass) is

$$[I_R] = -\sum_{i=1}^{N} m_i[\mathbf{r}_i - \mathbf{r}][\mathbf{r}_i - \mathbf{r}]$$

where brackets here indicate a skew-symmetric matrix.

6.10 Feedback Linearization (Inverse Dynamics Control):

Consider the following second-order, nonlinear system governing the dynamics of a manipulator.

$$B(q)\ddot{q} + C(q, \dot{q}) + g(q) = \tau$$

We can rewrite the above system as

$$B(q)\ddot{q} + n(q, \dot{q}) = \tau$$

where $n(q, \dot{q}) = C(q, \dot{q})\dot{q} + g(q)$ for convenience. We wish to choose an input τ such that the system becomes linearized. Assuming we know the model *exactly*, we can accomplish this by cancellation of the nonlinear dynamics. Choosing

$$\tau = B(q)y + n(q, \dot{q})$$

results in the second-order system

$$\ddot{q} = y$$

where y is the input to this linear, *decoupled* system. The above choice of τ is denoted *inverse dynamics control* since it requires the computation of the manipulator's inverse dynamics (the mapping from joint varibles to torques).

Now, assume we wish for the manipulator to follow a desired trajectory specified by $[\ddot{q}_d, \dot{q}_d, q_d]$. The simple choice of $y = \ddot{q}_d$ would lead to the system

$$\ddot{q} = \ddot{q}_d$$

Since the actual trajectory and desired trajectory are related directly via (double) integration, the manipulator would *precisely* track the desired trajectory if and only if the initial position and velocity match those of the desired trajectory. This can be seen by defining the tracking error $e = q - q_d$ and rewriting the above system as

$$\ddot{q} - \ddot{q}_d = \ddot{e} = 0$$

Even though the second derivative of the error is zero, the tracking error may still be constant (due to mismatched initial positions) or growing linearly (due to mismatched initial velocities)!

What choice of input y, then, guarantees convergence of the tracking error to zero? it's clear that choosing

$$y = \ddot{q}_d - K_d(\dot{q} - \dot{q}_d) - K_p(q - q_d)$$

leads to the system

$$(\ddot{q} - \ddot{q}_d) + K_d(\dot{q} - \dot{q}_d) + K_p(q - q_d) = 0$$

or in terms of the tracking error,

$$\ddot{e} + K_d\dot{e} + K_p e = 0$$

If the initial position and velocity match those of the desired trajectory, then the system will follow this trajectory with no error. If they do not match, though, the error will decay to zero according to the gain matrices K_p and K_p which are chosen to ensure stable error dynamics. Thus, for the original system, the input choice

$$\begin{aligned}\tau &= B(q)y + n(q, \dot{q}) \\ y &= \ddot{q}_d - K_d(\dot{q} - \dot{q}_d) - K_p(q - q_d)\end{aligned}$$

ensures accurate tracking. The first equation has the effect of linearizing the system and decoupling the input/output relationship of the joint variables. The second equation ensures stability of the system and of the tracking error dynamics. The issues with inverse dynamics control is that 1) the dynamic model may not be accurately known and 2) computation of the inverse dynamics online (in a control loop running at 1kHz, for example) is demanding and may require approximations to "lighten the load."

6.10.1 Humanoid Robot Control

Inverse dynamics control for humanoid robots (legged robots having so-called *floating base* dynamics in general) is significantly more complicated. For a summary of methods for humanoid robot control, please refer to the Background chapter of my PhD thesis[4].

[4] http://www-clmc.usc.edu/~nrotella/publications/NickRotellaThesis.pdf

Chapter 7

Signals and Filtering

7.1 Probability

The relative frequency definition of probability says that the probability of an event A is

$$p(A) = \frac{Numberoftimes Aoccurs}{Totalnumberofoutcomes}$$

The total number of ways to select k objects from n objects (assuming order doesn't matter) is *n-choose-k* or

$$\binom{n}{k} = \frac{n!}{(n-k)!k!}$$

The *conditional probability* of event A given event B, that is, the probability that event A occurs given that event B has occurred, is

$$P(A|B) = \frac{P(A,B)}{P(B)}$$

where $P(A,B)$ is the *joint probability* of A and B, that is the probability that both events A and B occur. Since

$$P(A,B) = P(A|B)P(B)$$

this implies that the probability of A occuring has some dependence on whether or not B has occurred. If the occurrence of B has no effect on the occurrence of A then

$$P(A,B) = P(A)P(B)$$

because $P(A)$ no longer depends on B, is the two events are *independent.*
Since we can also write $P(A,B) = P(B|A)P(A)$ we can write

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

which is *Bayes' Rule.*

A *random variable* is defined to be a functional mapping between a set of experimental outcomes (the domain) to a set of real numbers (the range).

For example, a random variable X is defined to be the RV which represents the roll of a die. The probability that X is 4 is a *realization* of the RV. The RV exists independently of any of its realizations and must not be confused with them.

The most fundamental property of an RV X is its *probability distribution function (PDF)*

$$F_X(x) = P(X \leq x)$$

where x is a nonrandom independent variable or constant. The PDF has the following properties:

$$\begin{aligned} F_X(x) &\in [0,1] \\ F_X(-\infty) &= 0 \\ F_X(\infty) &= 1 \\ F_X(a) &\leq F_X(b) \quad \text{if } a \leq b \\ P(a < X < b) &= F_X(b) - F_X(a) \end{aligned}$$

The *probability density function (pdf)* of X is defined to be

$$f_X(x) = \frac{dF_X(x)}{dx}$$

That is, the pdf is the derivative of the PDF with respect to x. It has the following properties:

$$\begin{aligned} F_X(x) &\int_{-\infty}^{x} f_X(z)dz \\ f_X(x) &\geq 0 \\ \int_{-\infty}^{\infty} f_X(x)dx &= 1 \\ P(a < X < b) &= \int_a^b f_X(x)dx \end{aligned}$$

The pdf is thus the density of the probability of X with respect to x; summing (integrating) this function adds up the probabilities associated with x. Note that when the RV X represents an experiment with discrete outcomes then the integrals become sums and the pdf is often called the *probability mass function (pmf).*

The *Q-function* of an RV is defined to be one minus the PDF, ie

$$Q(x) = 1 - F_X(x) = P(X > x)$$

In other words, $Q(x)$ is the probability that x does not occur (since the sum or integral of the PDF must be unity).

The conditional distribution and density of X given that an event A occurred are given by

$$F_X(x|A) = P(X \leq x|A) = \frac{P(X \leq x, A)}{P(A)}$$
$$f_X(x|A) = \frac{dF_X(x|A)}{dx}$$

In addition, Bayes' Rule generalizes to these conditional densities.

Now, consider the random variables X_1 and X_2; the conditional pdf of the RV X_1 given that RV X_2 is equal to the realization x_2 is defined as

$$\begin{aligned} f_{X_1|X_2}(x_1|x_2) &= P[(X_1 \leq x_1)|(X_2 = x_2)] \\ &= \frac{f_{X_1,X_2}(x_1, x_2)}{f_{X_2}(x_2)} \end{aligned}$$

Also consider the following product of two conditional pdf's; the result can be extended to any number of RVs.

$$\begin{aligned} f[x_1|(x_2, x_3, x_4)]f[(x_2, x_3)|x_4] &= \frac{f(x_1, x_2, x_3, x_4)}{f(x_2, x_3, x_4)} \frac{f(x_2, x_3, x_4)}{f(x_4)} \\ &= \frac{f(x_1, x_2, x_3, x_4)}{f(x_4)} \\ &= f[(x_1, x_2, x_3)|x_4] \end{aligned}$$

The expected value of a RV X is defined to be its average value over a large number of experiments - this is called the *expectation* of the RV. Consider a function $g(X)$ of the RV X; this function is also a random variable. The expected value of of the function is then

$$E[g(X)] = \int_{-\infty}^{\infty} g(x) f_X(x) dx$$

where $f_X(x)$ is the pdf of X. If the function is $g(X) = X$ then the expected value of X is

$$\bar{x} = E[X] = \int_{-\infty}^{\infty} x f_X(x) dx$$

This is essentially the sum (integral) of all possible outcomes of the experiment (realizations of the RV X) weighted by their probability densities; the expectation is a weighted average.

The *variance* of an RV is a measure of how much we expect the RV to vary from its expected value; it is defined as

$$\sigma_X^2 = E[(X - \bar{x})^2] = \int_{-\infty}^{\infty} (x - \bar{x})^2 f_X(x) dx$$

The square root of the variance is the *standard deviation* of the RV. Note that the variance can also be written as

$$\begin{aligned} \sigma^2 &= E[X^2 - 2X\bar{x} + \bar{x}^2] \\ &= E[X^2] - 2\bar{x}E[X] + \bar{x}^2 \\ &= E[X^2] - 2\bar{x}^2 + \bar{x}^2 \\ &= e[X^2] - \bar{x}^2 \end{aligned}$$

where it has been used that the expected value $\bar{x}$ can be pulled outside expectations since it itself is an expected value (it is known with certainty).

In general, the i^{th} *moment* and *central moment* of X are defined to be

$$\begin{aligned} i^{th} moment of X &= E[X^i] \\ i^{th} central moment of X &= E[(X - \bar{x})^i] \end{aligned}$$

An RV is called *uniform* if its pdf is a constant value between the limits a and b; within these limits the RV has an equally likely probability of attaining any value between a and b but a zero probability of obtaining a value outside these limits. Thus,

$$f_X(x) \begin{cases} \frac{1}{b-a} & x \in [a, b] \\ 0 & \text{otherwise} \end{cases}$$

An RV is called *Gaussian (normal)* if its pdf is given by

$$f_X(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[\frac{-(x - \bar{x})^2}{2\sigma^2}\right]$$

where $\bar{x}$ and σ are the mean and standard deviation of the RV, respectively. This is known as a Gaussian (normal) RV and is denoted by

$$X \sim N(\bar{x}, \sigma^2)$$

7.2 Stochastic Processes

A signal or process is said to be stationary or *strictly* stationary if its joint probability distribution does not change when the signal is shifted in time. Thus, the mean and variance of the process do not change over time. White noise, for example, is strict-sense stationary since its mean is always zero and its autocorrelation is zero for all time lags other than zero where it is σ^2.

A slightly weaker form of stationarity is wide-sense stationarity. In this case, the mean of the process is constant in time but the autocorrelation is a function only of the time lag.

The autocorrelation of a discrete-time signal is given by

$$R_{xx}[k] = E(x[n]x[n-k])$$

where it is assumed that the signal is WSS. It follows from the above definitions that the autocorrelation of a white noise process is

$$R_{xx}[k] = \sigma^2\delta[k]$$

The Power Spectral Density (PSD) function is defined as the Fourier transform of the autocorrelation function; in the discrete domain, this is

$$S_{xx}(\omega) = \sum_{k=-\infty}^{\infty} R_{xx}[k]e^{-j\omega k}$$

This function gives the power (squared signal amplitude) per unit frequency. It follows that the integral of the PSD is the expected power if the signal. For white noise, we have

$$S_{xx}(\omega) = \sum_{k=-\infty}^{\infty} \sigma^2\delta[k]e^{-j\omega k} = \sigma^2$$

ie its PSD is constant (flat across all frequencies). It follows that white noise contains, in theory, infinite power. In real systems, noise is not white at all frequencies and thus its power is finite.

7.3 Signals

$$M_{dB} = -10log_{10}(\frac{P_{out}}{P_{in}}) = -20log_{10}(\frac{V_{out}}{V_{in}})$$

The *cutoff* frequency of a filter is defined to be the frequency at which the input signal is attenuated to half power, ie $P_{out} = 0.5P_{in}$. The gain at this point is then

$$M_{dB} = -10log_{10}(\frac{1}{2}) \approx -3dB$$

The cutoff frequency is therefore known as the *3dB-down* point (also the *corner* or *break* frequency).

The *bandwidth* of a filter is usually defined to be the difference between the upper and lower cutoff frequencies for a bandpass filter or the cutoff frequency itself for a low-pass filter.

7.3.1 Moving Average Filter

The output of an M-tap FIR moving average filter is simply the average of the current sample and the previous $M-1$ samples

$$y[n] = \frac{1}{M}\sum_{k=0}^{M-1} x[n-k]$$

The transfer function of this filter is described by the Fourier transform of the rectangualr pulse, ie

$$H(f) = \frac{\sin(M\pi f)}{M\sin(\pi f)}$$

where here f is the *digital frequency* of the signal defined to be

$$f_{dig} = \frac{f}{f_s}$$

7.3.2 Aliasing

Since digitial frequency is periodic, a sampled signal shows up or *aliases* in the frequency domain (spectrum) at integer multiples of the sampling frequency. If the *bandlimit* of the signal is B then the Nyquist-Shannon sampling theorem says that one should choose the sampling rate to be

$$f_s > 2f$$

in order to prevent aliasing. This is often called the *Nyquist frequency*.

If the sampling rate is fixed then the signal is often low-pass filtered to eliminate any frequency components above the Nyquist frequency prior to A/D conversion.

7.4 Recursive Parameter Estimation

The sample mean $\bar{x}$ and sample covariance P_k of a Gaussian random vector x at time k are defined to be

$$\bar{x}_k = \frac{1}{k}\sum_{i=1}^{k} x_i$$
$$P_k = \frac{1}{k}\sum_{i=1}^{k}(x_i - \bar{x}_i)(x_i - \bar{x}_i)^T$$

respectively. If these parameters are to be computed at each time, then it is useful to formulate their computations recursively in order to minimize computations.

To this end, we write the mean at time k in terms of the mean at time $k-1$ by factoring out the leading term from the sum as follows.

$$\begin{aligned}\bar{x}_k &= \frac{1}{k}\sum_{i=1}^{k} x_i \\ &= \frac{1}{k}x_k + \frac{k-1}{k}\left[\frac{1}{k-1}\sum_{i=1}^{k-1} x_i\right] \\ &= \frac{1}{k}x_k + \frac{k-1}{k}\bar{x}_{k-1}\end{aligned}$$

The same can be done for the covariance, as shown below.

$$\begin{aligned}P_k &= \frac{1}{k}\sum_{i=1}^{k}(x_i - \bar{x}_i)(x_i - \bar{x}_i)^T \\ &= \frac{1}{k}(x_k - \bar{x}_k)(x_k - \bar{x}_k)^T + \frac{k-1}{k}\left[\frac{1}{k-1}\sum_{i=1}^{k-1}(x_i - \bar{x}_i)(x_i - \bar{x}_i)^T\right] \\ &= \frac{1}{k}(x_k - \bar{x}_k)(x_k - \bar{x}_k)^T + \frac{k-1}{k}P_{k-1}\end{aligned}$$

In this way, the sample mean and covariance can be updated online as new samples come in.

7.5 Kalman Filtering

Discrete-Time Model

Consider the following discrete time model in state-space form.

$$x_{k+1} = F_k x_k + G_k w_k$$
$$z_k = y_k + v_k = H_k x_k + v_k$$

where w_k denotes process noise and v_k denotes measurement noise. The goal is to solve the "filtering" problem; that is, to produce an estimate at time k of the state x_k given measurements $z_0, z_1, \ldots, z_k$ leading up to time k.

Noise assumptions

We assume that the processes $\{w_k\}$ and $\{v_k\}$ are

- individually white, meaning that for any $k \neq l$, v_k and v_l are independent of one another and w_k and w_l are independent of one another.
- individually zero mean, gaussian processes with known covariances Q_k and R_k, respectively.
- independent processes.

In summary, it is assumed that the processes $\{w_k\}$ and $\{v_k\}$ are zero mean, independent gaussian processes with covariances denoted by Q_k and R_k, respectively.

Initial state assumptions

The initial state of the system x_0 is assumed to be a gaussian random variable with known mean and covariance given by

$$E[x_0] = \bar{x}_0$$
$$E\{[x_0 - \bar{x}_0][x_0 - \bar{x}_0]^T\} = P_0$$

It is also assumed that x_0 is independent of w_k and v_k for all k.

Propagation of Means and Covariances:

The solution to the difference equation given above is

$$x_k = \Phi_{k,0} x_0 + \sum_{l=0}^{k-1} \Phi_{k,l+1} G_l w_l$$

where

$$\Phi_{k,l} = F_k F_{k-1} \cdots F_l \quad (k > l)$$

is the state-transition matrix which, in the (homogeneous) case of no input (noise), transfers the state from x_l to x_k. Note that $\Phi_{k,l} = \Phi_{k,m}\Phi_{m,l}$ and $\Phi_{k,k} = I$.

Since $x_0, w_0, w_1, \ldots, w_{k-1}$ are jointly gaussian random vectors, it follows that x_k - which is a linear combination of these vectors - is also gaussian. This is due to the fact that linear transformations of gaussian random variables are gaussian. In addition, note that $\{x_k\}$ is a Markov process.

Taking the expected value of both sides of the solution to the original difference equation yields

$$E[x_k] = \Phi_{k,0}\bar{x}_0$$

and therefore, from the definition of the state-transition matrix,

$$E[x_{k+1}] = F_k E[x_k]$$

Additionally, from the measurement model, we have

$$E[z_k] = H_k E[x_k]$$

The covariance of x_k for $k \geq l$ is

$$\begin{aligned}
P_{k,l} &= E\left\{[x_k - \bar{x}_k][x_l - \bar{x}_l]^T\right\} \\
&= E\left\{\left[\left(\Phi_{k,0}x_0 + \sum_{m=0}^{k-1}\Phi_{k,m+1}G_m w_m\right) - \Phi_{k,0}\bar{x}_0\right]\left[\left(\Phi_{l,0}x_0 + \sum_{n=0}^{l-1}\Phi_{l,n+1}G_n w_n\right) - \Phi_{l,0}\bar{x}_0\right]^T\right\} \\
&= E\left\{\left[\Phi_{k,0}(x_0 - \bar{x}_0) + \sum_{m=0}^{k-1}\Phi_{k,m+1}G_m w_m\right]\left[\Phi_{l,0}(x_0 - \bar{x}_0) + \sum_{n=0}^{l-1}\Phi_{l,n+1}G_n w_n\right]^T\right\}
\end{aligned}$$

Now, since $x_0 - \bar{x}_0, w_0, \ldots, w_{k-1}$ are all independent, the cross terms of the expression inside the expectation disappear, leaving

$$P_{k,l} = \Phi_{k,0}E\{[x_0 - \bar{x}_0][x_0 - \bar{x}_0]^T\}\Phi_{l,0}^T + \sum_{m=0}^{l-1}\Phi_{k,m+1}G_m E[w_m w_m^T]G_m^T\Phi_{l,m+1}^T$$

The second term in the above expression is the result of the whiteness of $\{w_k\}$; in the product of the two sums in the original expression, only those terms for which $m = n$ have nonzero expectations.

Substituting the definitions for P_0 and Q_m into the preceding expression and using properties of the state transition matrix yields

$$P_{k,l} = \Phi_{k,l}\left\{\Phi_{l,0}P_0\Phi_{l,0}^T + \sum_{m=0}^{l-1}\Phi_{l,m+1}G_m Q_m G_m^T\Phi_{l,m+1}^T\right\}$$

This is the general expression for the covariance of the state at different times. When $k = l$ this simplifies to

$$P_k = P_{k,k} = \Phi_{k,0} P_0 \Phi_{k,0}^T + \sum_{m=0}^{k-1} \Phi_{k,m+1} G_m Q_m G_m^T \Phi_{k,m+1}^T$$

which implies that

$$P_{k,l} = \Phi_{k,l} P_l, \quad k \geq l$$

Note also that $P_{k,l} = P_{l,k}^T$ and thus

$$P_{k,l} = P_{k,k} \Phi_{l,k}^T, \quad k \leq l$$

We now seek to derive a recursive equation for $P_k = P_{k,k}$. Using the definition of the state-transition matrix, we can write the above as

$$P_k = F_{k-1} \left[\Phi_{k-1,0} P_0 \Phi_{k,0}^T \right] F_{k-1}^T + \left[\sum_{m=0}^{k-2} \Phi_{k-1,m+1} G_m Q_m G_m^T \Phi_{k-1,m+1}^T \right] + \Phi_{k,k} G_k Q_k G_k^T \Phi_{k,k}^T$$

where we have factored out the final term from the sum. Again using the definition of Φ and the property that $\Phi_{k,k} = I$ we have

$$P_k = F_{k-1} \left[\Phi_{k-1,0} P_0 \Phi_{k,0}^T + \sum_{m=0}^{k-2} \Phi_{k-2,m+1} G_m Q_m G_m^T \Phi_{k-2,m+1}^T \right] F_{k-1}^T + G_k Q_k G_k^T$$

The expression inside the parentheses is the definition of P_{k-1}, leading to the difference equation

$$P_k = F_{k-1} P_{k-1} F_{k-1}^T + G_k Q_k G_k^T$$

which recursively describes the time evolution of the covariance of the state.

The covariance of the measurement is computed similarly as follows.

$$\begin{aligned} S_{k,l} &= E\left\{ [z_k - \bar{z}_k][z_l - \bar{z}_l]^T \right\} \\ &= E\left\{ [(H_k x_k + v_k) - H_k \bar{x}_k][(H_l x_l + v_l) - H_l \bar{x}_l]^T \right\} \\ &= E\left\{ [(H_k (x_k - \bar{x}_k) + v_k][H_l (x_l - \bar{x}_l) + v_l]^T \right\} \end{aligned}$$

Note that $\{v_k\}$ is independent of $\{x_k - \bar{x}_k\}$ because the latter process is determined entirely by X_0 and $\{w_k\}$, both of which were assumed to be independent of $\{v_k\}$. Thus the cross terms in the above expression disappear. This yields

$$S_{k,l} = H_k E\left\{[x_k - \bar{x}_k][x_l - \bar{x}_l]^T\right\} H_l^T + E[v_k v_l^T]$$

and thus

$$S_{k,l} = H_k P_{k,l} H_l^T + R_k \delta_{kl}$$

In addition, we can compute the covariance between x_k and z_k as

$$\begin{aligned} cov(x_k, z_k) &= E\{[x_k - \bar{x}_k][z_k - \bar{z}_k]^T\} \\ &= E\{[x_k - \bar{x}_k][H_k(x_k - \bar{x}_k)]^T\} \\ &= E\{[x_k - \bar{x}_k][(x_k - \bar{x}_k)]^T\} H_k^T \\ &= P_k H_k^T \end{aligned}$$

Likewise, we find that

$$cov(z_k, x_k) = H_k P_k$$

Probabilistic Properties

We have already stated that x_k is gaussian since it is a linear combination of x_0 and $w_0, \ldots, w_{k-1}$ (all of which are gaussian); we also know that z_k is gaussian since x_k and v_k are gaussian.

Our goal in filtering, however, is to produce an estimate of x_k by conditioning on z_k. That is, we seek to use the information provided by process $\{z_k\}$ to estimate the state. What form does x_k conditioned on z_k have?

Let the pair of vectors X and Y be *jointly gaussian*, ie the vector composed as $Z = [X^T Y^T]^T$ is gauassian with mean and covariance

$$\bar{z} = \begin{pmatrix} \bar{x} \\ \bar{y} \end{pmatrix} \quad \Sigma = \begin{pmatrix} \Sigma_{xx} & \Sigma_{xy} \\ \Sigma_{yx} & \Sigma_{yy} \end{pmatrix}$$

It can be shown that X conditioned on Y is also gaussian with *conditional mean* and *conditional covariance* given by

$$\begin{aligned} \tilde{x} &= \bar{x} + \Sigma_{xy}\Sigma_{yy}^{-1}(y - \bar{y}) \\ \tilde{\Sigma} &= \Sigma_{xx} - \Sigma_{xy}\Sigma_{yy}^{-1}\Sigma_{yx} \end{aligned}$$

These parameters describe the distribution $p(x|y)$ which, of course, is a function; given a value of y, x is regarded as a variable. Since we seek to estimate x from y, we might logically wish to determine the value of x which maximizes this probability given $Y = y$. This is the *maximum a posteriori (MAP)* estimate of x. This is not the only (nor necessarily the best) way to choose x, however!

Minimum Variance Estimate

Instead of seeking the estimate of x which maximizes the conditional probability density given above, we wish to find the estimate $\hat{x}$ which is closest to the true value of x on average. That is, we seek the *minimum variance estimate* $\hat{x}$ for which

$$E\{||X - \hat{x}||^2 | Y = y\} \leq E\{||X - z||^2 | Y = y\}$$

for all other vectors z which are determined in a different way from y than was x (but are not dependent on x). For any such z we can write

$$\begin{aligned} E\{||X - z||^2 | Y = y\} &= \int_{-\infty}^{\infty} (x - z)^T (x - z) p(x|y) dx \\ &= \int_{-\infty}^{\infty} x^T x p(x|y) dx - 2z^T \int_{-\infty}^{\infty} x p(x|y) dx + z^T z \int_{-\infty}^{\infty} p(x|y) dx \\ &= \left[z^T - \int_{-\infty}^{\infty} x^T p(x|y) dx \right] \left[z - \int_{-\infty}^{\infty} x p(x|y) dx \right] + \cdots \\ &+ \int_{-\infty}^{\infty} x^T x p(x|y) dx + \left|\left| \int_{-\infty}^{\infty} x p(x|y) dx \right|\right|^2 \end{aligned}$$

Clearly, the above expression is minimized when the first term disappears; this follows from the choice

$$z = \int_{-\infty}^{\infty} x p(x|y) dx$$

Hence, the minimum variance estimate of x is the conditional mean estimate

$$\hat{x} = E\{X | Y = y\}$$

With this choice, the variance becomes

$$\begin{aligned} E\{||X - z||^2 | Y = y\} &= \int_{-\infty}^{\infty} x^T x p(x|y) dx + \left|\left| \int_{-\infty}^{\infty} x p(x|y) dx \right|\right|^2 \\ &= E\{||X||^2 | Y = y\} - ||\hat{x}||^2 \end{aligned}$$

The variance is also given by the trace of the corresponding error covariance matrix which, since the estimate is simply the conditional mean, is the covariance matrix

$$\hat{\Sigma} = \Sigma_{xx} - \Sigma_{xx} \Sigma_{yy}^{-1} \Sigma_{yx}$$

associated with the conditional density.

Note that an estimate is termed *unbiased* when the expected value of the the estimation error $e = x - \hat{x}$ given y is zero. Since the minimum variance estimate is equal to the conditional mean, this estimate must be unbiased.

$$E\{X - \hat{x} | Y = y\} = E\{X | Y = y\} - \hat{x} = 0$$

Derivation of the Kalman Filter

The first-principles derivation of the Kalman Filter follows directly from the results of previous sections. We seek to form an estimate $\hat{x}_k$ of the state of the system at time k from the measurements $z_0, \ldots, z_k$. We begin with the initial state and use the results of preceding sections to develop general formulas.

The initial state x_0 is assumed to have known mean $\bar{x}_0 = \hat{x}_0^-$ and covariance P_0^-. As noted previously, x_k and z_k are jointly gaussian for all k with mean and covariance

$$mean = \begin{pmatrix} \bar{x}_k \\ H_k \bar{x}_k \end{pmatrix} \quad cov = \begin{pmatrix} P_k & P_k H_k^T \\ H_k P_k & H_k P_k H_k^T + R_k \end{pmatrix}$$

Therefore for $k = 0$, x_0 conditioned on z_0 is also gaussian with mean

$$\hat{x}_0^+ = \hat{x}_0^- + P_0 H_0^T \left(H_0 P_0^- H_0^T + R_0 \right)^{-1} (z_0 - H_0 \hat{x}_0^-)$$

and covariance

$$P_0^+ = P_0^- - P_0^- H_0^T \left(H_0 P_0^- H_0^T + R_0 \right)^{-1} H_0 P_0^-$$

Based on how how the mean and covariance of the state propagate through the system, it follows that x_1 conditioned on z_0 is gaussian with mean and covariance

$$\hat{x}_1^- = F_0 \hat{x}_0^+ \quad \text{and} \quad P_1^- = F_0 P_0^+ F_0^T + G_0 Q_0 G_0^T$$

Now, the random variable $[x_1^T z_1^T]$ conditioned on z_0 is gaussian with mean and covariance

$$\begin{pmatrix} \hat{x}_1^- \\ H_1 \hat{x}_1^- \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} P_1^- & P_1^- H_1^T \\ H_1 P_1^- & H_1 P_1^- H_1^T + R_1 \end{pmatrix}$$

It follows that x_1 conditioned on z_0, z_1 then has mean

$$\hat{x}_1^+ = \hat{x}_1^- + P_1 H_1^T \left(H_1 P_1^- H_1^T + R_1 \right)^{-1} (z_1 - H_1 \hat{x}_1^-)$$

and covariance

$$P_1^+ = P_1^- - P_1^- H_1^T \left(H_1 P_1^- H_1^T + R_1 \right)^{-1} H_1 P_1^-$$

In general, then, we can write in the same way that

$$\begin{aligned} \hat{x}_k^- &= F_{k-1} \hat{x}_{k-1}^+ \\ P_k^- &= F_{k-1} P_{k-1}^+ F_{k-1}^T + G_{k-1} Q_{k-1} G_{k-1}^T \\ K_k &= P_k H_k^T \left(H_k P_k^- H_k^T + R_k \right)^{-1} \\ \hat{x}_k^+ &= \hat{x}_k^- + K_k (z_k - H_k \hat{x}_k^-) \\ P_k^+ &= P_k^- - K_k H_k P_k^- \end{aligned}$$

Kalman Filtering

Consider the time-invariant state-space system

$$\dot{x} = Ax + Lw$$
$$z = Hx + v$$

where w is the *process noise* vector and v is the *measurement noise* vector. Each is assumed to be white and Gaussian with zero mean and the *continuous* process covariance matrices Q_c and R_c, respectively.

Since the matrices A and L do not depend on time, the solution to this system can be written as

$$x(t_k) = e^{A(t_k - t_{k-1})}x(t_{k-1}) + \int_{t_{k-1}}^{t_k} e^{A(t_k-\tau)}Lw(\tau)d\tau$$

where $\Phi(t_k, t_{k-1}) = e^{A(t_k-t_{k-1})}$ is the *state-transition matrix* governing the system's unforced response between these times. Defining a timestep $\Delta t = t_k - t_{k-1}$ and assuming a zero-order hold such that the noise vector does not change over this timestep, we can write the solution as

$$x_k = F_{k-1}x_{k-1} + L_{k-1}w_{k-1}$$

where

$$F_k = e^{A\Delta t}$$
$$L_k = \int_{t_k}^{t_{k+1}} e^{A(t_{k+1}-\tau)}d\tau L$$

These equations specify how to discretize the dynamics of a continuous system and are found in many texts.

The expected value of the state vector at time k is then

$$\bar{x} = E[x_k] = F_{k-1}\bar{x}_{k-1}$$

since $E[w_k] = 0$ for all k because the process noise is zero-mean.

Defining the estimation error at time k to be $e_k = x_k - \bar{x}_k$, we find that the covariance of the state prediction at time k as a function of the covariance at time $k-1$ is

$$\begin{aligned} P_k &= E[e_k e_k^T] \\ &= E[(x_k - \bar{x}_k)(x_k - \bar{x}_k)^T] \\ &\vdots \\ &= F_{k-1} P_{k-1} F_{k-1}^T + Q_{k-1} \end{aligned}$$

where

$$Q_{k-1} = \int_{t_{k-1}}^{t_k} e^{A(t_k - \tau)} L Q_c L^T e^{A^T (t_k - \tau)} d\tau$$

is the *discrete* process noise covariance matrix. In practice, the discrete dynamics matrix and state covariance matrix are often truncated at first order to yield

$$\begin{aligned} F_{k-1} &\approx I + A\Delta t \\ Q_{k-1} &\approx Q_c \Delta t \end{aligned}$$

Additionally, the continuous measurement covariance matrix can be discretized as

$$R_{k-1} \approx \frac{R_c}{\Delta t}$$

A standard Kalman Filter implementation maintains its estimate of the state vector as the expected value of the state given all prior measurements. This is done by 1) propagating the expected value of the state and estimation error covariance using the linear dynamics (*prediction* step) and 2) updating the expected value of the state and estimation error covariance using a measurement related to the state (*update step*).

The expected value of x will henceforth be denoted by $\hat{x}$; note that $\hat{x}_k^-$ denotes the EKF estimate of x at time k *prior* to the update step (*a priori* estimate) while $\hat{x}_k^+$ denotes the estimate *after* the update step (*a posteriori* estimate). Written another way, this means

$$\begin{aligned} \hat{x}_k^- &= E[x_k | z_1, z_2, \cdots, z_{k-1}] \\ \hat{x}_k^+ &= E[x_k | z_1, z_2, \cdots, z_k] \end{aligned}$$

Likewise, P_k^- denotes the estimation error covariance prior to the update step and P_k^+ denotes the same quantity after the measurement at time k is taken; that is,

$$\begin{aligned} \hat{P}_k^- &= E[(x_k - \hat{x}_k^-)(x_k - \hat{x}_k^-)^T] \\ \hat{P}_k^+ &= E[(x_k - \hat{x}_k^+)(x_k - \hat{x}_k^+)^T] \end{aligned}$$

Thus, the Kalman filter steps and corresponding equations are

Predict:

$$\begin{aligned}\hat{x}_k^- &= E[x_k] = F_k\hat{x}_{k-1} \\ \hat{z}_k &= H_k\hat{x}_k^- \\ P_k^- &= E[(x_k - \hat{x}_k^-)(x_k - \hat{x}_k^-)^T] = F_kP_{k-1}^+F_k^T + Q_k\end{aligned}$$

Update:

$$\begin{aligned}y_k &= z_k - \hat{z}_k \\ S_k &= E[y_ky_k^T] = H_kP_k^-H_k^T + R_k \\ K_k &= P_k^-H_k^TS_k^{-1} \\ \hat{x}_k^+ &= \hat{x}_k^- + K_ky_k \\ P_k^+ &= (I - K_kH_k)P_k^-\end{aligned}$$

Here the quantity $y_k = (z_k - \hat{z}_k)$ is called the *innovations* because it is the part of the measurement z_k which contains new information which can be used to update the state. The covariance of the innovations is given by S_k.

The "size" of K_k (with reference to the scalar case) determines to what extent the measurement at time k is used to correct the prediction. A "larger" state covariance means that the prediction is uncertain, making the Kalman gain larger and thus weighing the effect of the measurement more. A "larger" innovations covariance means the measurement is uncertain, making the gain smaller and thus trusting the prediction more.

Discrete Extended Kalman Filter

The issue is that the Kalman Filter as described above can only be implemented for *linear* systems. A simple way to get around this limitation is to linearize a nonlinear system around the Kalman estimate at each timestep and then use the linearized system matrices to perform the update.

Consider the discrete-time nonlinear system (assuming no control input for simplicity)

$$\begin{aligned}x_k &= f(x_{k-1}, w_k) \\ z_k &= h(x_k, v_k)\end{aligned}$$

where $f(\bullet)$ and $h(\bullet)$ are nonlinear functions of the model parameters. The EKF steps are as follows.

Predict state and measurement:

$$\begin{aligned} \hat{x}_k^- &= f(\hat{x}_{k-1}^+, u_{k-1}) && : \quad \text{a priori state estimate} \\ \hat{z}_k &= h(\hat{x}_k^-) && : \quad \text{Predicted measurement} \end{aligned}$$

Linearize and predict covariance:

$$\begin{aligned} F_k &= \frac{\partial f}{\partial x}|_{\hat{x}_k^-} && : \quad \text{Linearized dynamics matrix} \\ H_k &= \frac{\partial h}{\partial x}|_{\hat{x}_k^-} && : \quad \text{Linearized measurement matrix} \\ P_k^- &= F_k P_{k-1}^+ F_k^T + Q_k && : \quad \text{a priori state covariance} \end{aligned}$$

Update predictions using measurement:

$$\begin{aligned} y_k &= z_k - \hat{z}_k && : \quad \text{Innovations vector} \\ S_k &= H_k P_k^- H_k^T + R_k && : \quad \text{Innovations covariance} \\ K_k &= P_k^- H_k^T S_k^{-1} && : \quad \text{Kalman gain} \\ \hat{x}_k^+ &= \hat{x}_k^- + K_k y_k && : \quad \text{a posteriori state estimate} \\ P_k^+ &= (I - K_k H_k) P_k^- && : \quad \text{a posteriori state covariance} \end{aligned}$$

Prediction entails propagating the state and its covariance as well as computing the expected measurement. However, prediction of the covariance requires dynamics which are linearized around the current state estimate, so things get a bit out of order. First the state (and measurement) are predicted using the nonlinear dynamics, then linearization of is performed first because the matrix F_k is needed in order to propagate the state covariance. The *a priori* covariance is then predicted. Finally, the update step entails obtaining the measurement and using it (along with the linearized dynamics) to update the state and covariance to their *a posteriori* values at time k.

Consider the time-invariant state-space system

$$\dot{x} = Ax + Bu + Lw$$

where w is the *process noise* vector having zero mean and the *continuous* process noise covariance matrix Q_c.

Since the matrices A, B and L do not depend on time, the solution to this system can be written as

$$x(t_k) = e^{A(t_k - t_{k-1})} x(t_{k-1}) + \int_{t_{k-1}}^{t_k} e^{A(t_k - \tau)} \left[Bu(\tau) + Lw(\tau) \right] d\tau$$

where $\Phi(t_{k-1}, t_k) = e^{A(t_k - t_{k-1})}$ is the *state-transition matrix* governing the system's unforced response between these times. Defining a timestep $\Delta t = t_k - t_{k-1}$ and assuming

a zero-order hold such that the input and noise vectors (since the matrices are time-invariant) do not change over this timestep, we can write the solution as

$$x_k = F_{k-1}x_{k-1} + G_{k-1}u_{k-1} + L_{k-1}w_{k-1}$$

where

$$\begin{aligned} F_k &= e^{A\Delta t} \\ G_k &= \int_{t_k}^{t_{k+1}} e^{A(t_{k+1}-\tau)}d\tau B \\ L_k &= \int_{t_k}^{t_{k+1}} e^{A(t_{k+1}-\tau)}d\tau L \end{aligned}$$

The expected value of the state vector at time k is then

$$\bar{x} = E[x_k] = F_{k-1}\bar{x}_{k-1} + G_{k-1}u_{k-1}$$

since the control input is assumed to be known and since $E[w_k] = 0$ for all k.

Defining the estimation error at time k to be $e_k = x_k - \bar{x}_k$, we find that the covariance of the estimation error at time k as a function of the covariance at time $k-1$ is

$$\begin{aligned} P_k &= E[e_k e_k^T] \\ &= E[(x_k - \bar{x}_k)(x_k - \bar{x}_k)^T] \\ &\vdots \\ &= F_{k-1}P_{k-1}F_{k-1}^T + Q_{k-1} \end{aligned}$$

where

$$Q_{k-1} = \int_{t_{k-1}}^{t_k} e^{A(t_k-\tau)}LQ_cL^T e^{A(t_k-\tau)^T)}d\tau$$

is the *discrete* process noise covariance matrix.

A standard discrete-time Kalman Filter implementation maintains its estimate of the state vector as the expected value of the state. This is done by 1) propagating the expected value of the state using the linear dynamics (*predicition* step) and 2) updating the expected value of the state using a measurement related to the state (*update step*). The expected value of x will henceforth be denoted by $\hat{x}$; note that $\hat{x}_k^-$ denotes the EKF estimate of x at time k *prior* to the update step (*a priori* estimate) while $\hat{x}_k^+$ denotes the estimate *after* the update step (*a posteriori* estimate).

Thus,

Predict:

$$\hat{x}_k^- = F_k \hat{x}_{k-1} + G_{k-1} u_{k-1}$$
$$P_k^- = F_k P_{k-1}^+ F_k^T + Q_k$$

Update:

$$y_k = z_k - H_k \hat{x}_k^-$$
$$S_k = H_k P_k^- H_k^T + R_k$$
$$K_k = P_k^- H_k^T S_k^{-1}$$
$$\hat{x}_k^+ = \hat{x}_k^- + K_k y_k$$
$$P_k^+ = (I - K_k H_k) P_k^-$$

7.5.1 Discrete Extended Kalman Filter

The Kalman Filter as described above can only be implemented for *linear* systems. However, a simple way to get around this limitation is to linearize a nonlinear system around the Kalman estimate at each timestep and then use the same equations to perform the update.

Consider the discrete-time nonlinear system

$$x_k = f(x_{k-1}, u_{k-1}) +, w_k$$
$$z_k = h(x_k) + v_k$$

where $f()$ and $h()$ are nonlinear functions of the model parameters. The EKF steps are as follows.

Predict:

$$\hat{x}_k^- = f(\hat{x}_{k-1}^+, u_{k-1})$$
$$P_k^- = F_k P_{k-1}^+ F_k^T + Q_k$$

Linearize:

$$F_k = \frac{\partial f}{\partial x}|_{\hat{x}_k^-, u_k}$$
$$G_k = \frac{\partial f}{\partial u}|_{\hat{x}_k^-, u_k}$$
$$H_k = \frac{\partial h}{\partial x}|_{\hat{x}_k^-}$$

Update:

$$
\begin{aligned}
y_k &= z_k - H_k \hat{x}_k^- \\
S_k &= H_k P_k^- H_k^T + R_k \\
K_k &= P_k^- H_k^T S_k^{-1} \\
\hat{x}_k^+ &= \hat{x}_k^- + K_k y_k \\
P_k^+ &= (I - K_k H_k) P_k^-
\end{aligned}
$$

Chapter 8

Programming

8.1 Coding Style Guidelines

This ‘‘`http://www.rudbek.com/Code_gui.htm#Guideline_No_Space_After_Asterisk_Before_Identifier`’’article has some good coding style guidelines:

For example, in C it's apparently preferred to using `int *myint` rather than `int* myint`, however in C++ it's preferred to use int `&myint` rather than `int& myint`.

8.2 Virtual Functions

An abstract class is one which contains at least one *pure virtual* function, defined as follows:

```
virtual void myfun() = 0;
```

An abstract class cannot be instantiated; it can only be derived from, or used to create a base class pointer to a derived class object.

In contrast, a virtual function (not PURE) is a function which can be overridden by a derived class. Actually, every base class function can be overridden... but virtual functions called through pointers/references to the base class are resolved dynamically (at run-time), whereas a non-virtual function which is overridden by a derived class will have its base class function used when called through a base class pointer. This is what we do to make functionality SL-independent!

8.3 Rule of Three

In pre-c++11, the “rule of three” says that if any of 1) destructor 2) copy constructor or 3) copy assignment operator has been defined for a class, then the others should probably be defined as well. The reasoning is because if any of these functions is used without being declared then it will be implicitly defined by the compiler with some default semantics. If the compiler's default semantics were insufficient enough to cause the programmer to

implement one of these functions, it means that they are also *probably* insufficient for the others.

8.4 Overloading and Overriding

Overloading is when a function is declared with the same name as an existing function but with different parameters. Functions overloaded in a base class are inherited by a derived class.

Overriding, on the other hand, is when a function with the same name AND parameters as a base class function is declared in a derived class. Calls to the function in the derived class will use the derived class implementation, however the base class version can still be called from the derived class using `BaseClass::myFunction()`. Note that if you want the derived class implementation to be called from a pointer to the base class, the base class function must be declared `virtual`.

In c++11, there is an `override` keyword to be used when overriding a virtual method which makes it explicit that your function is overriding the base class function and, more importantly, causes the compiler to check that the overriding is valid.

8.5 Move Semantics (C++11)

You may be familiar already with lvalues and rvalues, but C++11 introduced the concept of rvalue *references* which are denoted with `&&` (rather than the usual single `&` for lvalue references). This makes it possible to differentiate between lvalues and rvalues, which was not possible previously.

This means that, for example, you can overload a function which accepts a `const` reference (which will normally take anything, lvalue or rvalue, as an input) with a version explicitly taking an rvalue reference. This means you can define a version of the function for the particular case in which a temporary (rvalue) is passed in, which can potentially be useful.

This also allows the ability to "steal" the data from an rvalue and set its pointer to null; we call this *move* semantics because the data is not copied from rvalue to lvalue but rather *taken ownership of by the lvalue*. This means that we "steal" the data pointer from the temporary object and then set it to `NULL` (so that when the temporary goes out of scope and its memory is cleaned up, it doesn't deallocate the memory pointed to by the pointer we just stole!) The reason for this feature is to prevent making unnecessary additional copies, moving rather than copying resources.

We could `move` lvalues into other lvalues but this is dangerous because we may, later on, try to use the lvalue which was moved - and won't be able to. However we can safely `move` rvalues into lvalues via rvalue references because they will die when they go out of scope. Actually, this is where `std::move` comes in; it *DOES NOT actually move anything* - it just casts lvalues to rvalues so that the move constructor can be invoked. Technically, `std::move` has nothing more to do with move semantics than that.

The reason we pass `unique_ptr` to classes (which we initialize at the highest, SL-dependent level of the task using `new`) is because we can use polymorphism to pass base class pointers and move them into derived class pointers. This keeps the SL dependency at the highest level. HOWEVER I'm not sure why we need to use `std::move` when we are passing rvalue references to `unique_ptrs` in? why do we need `my_ptr_(std::move(my_ptr))` instead of just `my_ptr_(my_ptr)` which will call the move constructor? And do we need to have explicitly defined move constructors for our classes (because we do NOT)? I *think* we have to use `std::move` because `unique_ptr` cannot be copied, they must be explicitly moved.

The default move constructor (generated by the compiler in the absence of others) is to perform a member-wise move.

8.6 Inheritance

Within the base class, anything public can be seen by anyone, protected can be seen by only the base and derived class, and private only within the base class.

Inheritance is something different: if the derived class inherits from the base class publically, then a base class pointer can be used for a derived class object. This is not true for other types of inheritance, in which only the derived class knows who it inherits from. Thus, for polymorphism, we need public inheritance! Public inheritance basically means all members' access specifiers from the base class are preserved. Protected inheritance means public and protected members of the base class become protected in the derived class, and private inheritance means everything inherited becomes private. Note that the base class' private members are NEVER inherited.

Multiple inheritance uses the following syntax:

```
class MyClass : public BaseClassA, protected BaseClassB, private BaseClass C
{
}
```

and members from each base class are inherited accordingly.

8.7 C++11 Features

What is the difference between `r=&T` and `r=std::ref(T)`? The latter returns type `reference_wrapper` so that `r` isn't actually a reference but a *wrapper* around a reference; use `r.get()` to obtain the reference, ie the same thing as `&T`. One nice feature of `std::ref()` is that it's possible to create an array of `reference_wrapper<type>` whereas it's impossible to create an array of references normally.

Some features which previously existed in Boost were brought to C++11. One is `std::function` which provides a wrapper around a function (or functor or lambda expression, etc) similar to how `std::ref` provides a reference wrapper. Similarly also, this allows the creation of arrays of functions. Another is `std::bind`, which takes a

function pointer and arguments and returns a `std::function` object which can then be eg pushed back in a vector. This is how to add `std::function` objects which have arguments. Additionally, bind allows using *placeholders* for function arguments, so that arguments for which you don't pass a value during binding can be set later, when the function is called using fewer arguments (`https://oopscenities.net/2012/02/24/c11-stdfunction-and-stdbind/`more details here).

`std::thread` replaces `boost::thread` for multithreading applications. You can launch a thread which executes a function by passing a reference to the function and the `this` pointer for the class which the function belongs to, followed by the arguments of the function (it seems you don't need `bind` as in the old boost method??) Here's an example:

```
std::thread(&WalkingStateMachine::rosbagThreadLoop, this, bag_filename)
```

where `bag_filename` is a parameter for the function `rosbagThreadLoop` which is defined elsewhere.

Ranged-based `for` loops allow for iterating over containers like we do in python (eg `for name in names`). This can be combined with the `auto` keyword to make for extremely easy loops:

```
for(const auto &myiter : mycontainer) {}
```

Here, `myiter` does not need to be dereference as its already a reference to each of the elements in `mycontainer`. The `auto` keyword tells the compiler to deduce the proper data type as needed from function/variable definitions. For example,

```
MyDataType myfunction(void){}
auto a = myfunction(); // the compiler figures out that 'a' needs to have type
    'MyDataType'
```

This is particularly useful for iterators, which typically have very lengthy names. Using `auto` saves typing and space but should not be abused because it can make code less readable. Declaring a variable to have type `auto`, eg:

```
auto a;
```

will NOT compile which forces initialization, which can be desirable. Note that `auto` iterators will always be non-const, otherwise a `const_iterator` has to be explicitly used.

8.7.1 `typedef` versus `using`

In C++11, we are able to use `using` to define type aliases in the same way `typedef` was used, for example:

```
typedef std::vector<int> int_vec;
using int_vec = std::vector<int>; // C++11
```

There is no difference between these commands, but `using` may be more natural. The reason this new keyword was introduced was to handle `typedef` cases involving templates like

```
template<typename T> using my_vec = std::vector<T>;
```

8.8 STL (Standard Template Library) Vectors

In C++11 we can brace-initialize `std::vector` as, for example,

```
std::vector<std::vector<int>> myvec = {{1, 2, 3}, {4, 5, 6}};
```

8.9 Iterators

Iterators allow for iterating through a container of elements of any data type, like pointers do with C-style arrays. We define and use an iterator like:

```
std::vector<int>::iterator it;
for(it = myvec.begin(); it != myvec.end(); ++it)
{
  std::cout << *it << std::endl;
}
```

Note that we use `begin()` and `end()` to get the bounds of the container and that we use `!=` rather than `<` because it has less undefined behavior for some iterators. Also note that if `myvec` were of type `const std::vector<int>`, we would need a `const_iterator` to ensure that we don't change any elements of `myvec`. Also note that `*` dereferences the iterator to the corresponding element.

As detailed in a different section, in C++11 the `auto` keyword allows us to replace the bulky iterator definition, and go even further using range-based for loops such as

```
for(auto it : myvec)
{
  std::cout << it << std::endl;
}
```

We can instead use `const auto &it`. Abide by the following rules:

- Choose `auto x` when you want to work with copies.
- Choose `auto &x` when you want to work with original items and may modify them.
- Choose `auto const &x` when you want to work with original items and will not modify them.

Note that we don't need to dereference `x` in any of these cases, as we would have to have done with a normal old iterator.

8.10 Dependency Injection

This is the term for what we do in order to make our code SL-independent (write abstract base classes with pure virtual functions, make the derived class SL-dependent and then use base class pointers to at top level to *inject* the SL dependencies).

8.11 The `static` keyword

The `static` keyword on a class member means that that member is shared by all instances of the class and is accessible using the class scope. A static member variable thus has no `this` pointer because it doesn't belong to a specific class instance.

Static member functions similarly are shared among all class instances and can only access other static member functions and variables. Note that static member variables must be declared in the class `cpp` file or you will get undefined reference linker errors.

Static is useful because it lets all instances of a class share state and gives a way to use functions in a class without having to create an instance, but they should be used carefully because they are essentially global variables.

8.12 Singleton Design Pattern

A *singleton* is a class which is ensures only one instance can be instantiated - this could be useful for example in creating a logger class or a game loop class. This design pattern is achieved using the `static` keyword as follows[1]:

```
class S
{
    public:
        static S& getInstance()
        {
            static S   instance; // Guaranteed to be destroyed.
                                 // Instantiated on first use.
            return instance;
        }
    private:
        S() {}                   // Constructor? (the {} brackets) are needed
            here.

        // C++ 03
        // ========
        // Dont forget to declare these two. You want to make sure they
```

[1] https://stackoverflow.com/questions/1008019/c-singleton-design-pattern

```
        // are unacceptable otherwise you may accidentally get copies of
        // your singleton appearing.
        S(S const&);            // Don't Implement
        void operator=(S const&); // Don't implement

        // C++ 11
        // =======
        // We can use the better technique of deleting the methods
        // we don't want.
    public:
        S(S const&)             = delete;
        void operator=(S const&) = delete;

        // Note: Scott Meyers mentions in his Effective Modern
        //       C++ book, that deleted functions should generally
        //       be public as it results in better error messages
        //       due to the compilers behavior to check accessibility
        //       before deleted status
};
```

Note that an object of this class cannot be created as normal because its constructor is private; instead, the `instance()` function must be called which returns a `static` object and thus remains in scope like a global variable (further calls to `instance()` will return the same object).

8.13 The `friend` Keyword

The `friend` keyword allows a class to declare a function as being accessible from another class. The function can then be implemented outside the class which declares it, but an instance of the class which declared it must be passed because, like static members, there is no `this` pointer.

Similarly, an entire class can be declared as a friend of another class to give the second class access to all of the first class' members. Use of `friend` sometimes requires a forward declaration of a class which it depends on.

Friend is useful to give access to class variables to another class without needing public getter functions, but violates encapsulation and could be dangerous.

8.14 The `const` Keyword and `const` "Correctness"

The `const` keyword does different things depending on where it's placed in eg a function definition. For example:

```
f(const int &myint)
```

is the same as

```
f(int const &myint)
```

in that they both accept a reference to an int which is immutable. It's easy to understand the meaning of either of these by reading *from right to left*. Here, we have "`myint` is a reference to a `const int`."

So-called `const` "correctness" simply refers to the use of the const keyword to protect certain data from being modified. To protect an input to function, you have a few options:

- Pass a `const` reference, eg `void myfunction(const Object& myobj);` which is equivalent to `void myfunction(Object const& myobj);`
- Pass a `const` pointer, eg `void myfunction(const Object* myobj);` which is equivalent to `void myfunction(Object const* myobj);`. This is always confusing because the pointer is not `const`, the *object* it points to is `const`!
- Pass by value (makes a copy), eg `void myFunction(Object myobj);`

Again, reading the pointer declarations *right-to-left* helps clear up ambiguity about what it means:

- `const X* p` means `p` points to an `X` that is `const`: the `X` object cant be changed via `p` but the pointer *can*.
- `X* const p` means `p` is a `const` pointer to an `X` that is non-const: you cant change the pointer `p` itself, but you *can* change the `X` object via `p`.
- `const X* const p` means `p` is a `const` pointer to an `X` that is `const`: you cant change the pointer `p` itself, *nor* can you change the `X` object via `p`.

So if you insist on passing a `const` pointer, you should use `const X* const p` so that neither the object NOR POINTER can be changed.

A `const` *member function* in a class has the form:

```
void myfunction(int a) const;
```

The input is irrelevant here, but the trailing `const` says that this function will not change `*this` in any way; we call it an *inspector* function.

Const overloading allows overloading (redefining) a member function with a trailing `const`. So you can define:

```
void myfunction(int a);
void myfunction(int a) const;
```

And the latter one will get called for a `const` object, while the former will be called on a non-const object. I'm not sure when this is useful.

8.15 C-style `struct`

Structures in C take the form:

```
struct DataStruct {
  int a;
  char* str;
  float b;
  } mystruct;
```

where `mystruct` is a variable of type `struct DataStruct` which gets initialized at the same time as `DataStruct` is defined. To declare another struct of this type:

```
struct DataStruct othermystruct;
```

To make the syntax less cumbersome, we can add a `typedef` after the struct definition:

```
typedef struct DataStruct DataStruct;
```

Or, to be most concise, we can make the struct definition happen INSIDE the typedef:

```
typedef struct {
  int a;
  char* str;
  float b;
  } DataStruct;
```

where `DataStruct` is *not* a variable anymore but the actual type which finishes the typedef statement.

In C, you must initialize structs after declaring them, eg:

```
mystruct.a = 1;
mystruct.b = 2.66;
mystruct.str = malloc(10);
mystruct.str = ``teststring'';
```

Note that we have to allocate memory for the `char*` struct member using `malloc`.

Now, what if we want to dynamically allocate a struct? We would again use `malloc` as

```
DataStruct* mystruct = malloc(sizeof(DataStruct));
```

In C++ we could use `new`, but in C we use `malloc`; both return pointers to the dynamically-allocated data. We again then have to separately allocate space for the string so be careful. Note that `sizeof(DataStruct)` will not necessarily return `sizeof(int)+sizeof(char*` because the compiler will (probably) use struct *padding* to ensure that variables are *self-aligned* in memory, meaning that they start at address spaces which are multiples of

the system word size (32 bit or 64 bit). You can force structures to not use padding by *packing* them, however this could lead to slower access time. This is done as:

```
typedef struct __attribute__((__packed__)) {
  int a;
  char* str;
  float b;
  } DataStruct;
```

8.16 The `inline` Keyword

The keyword `inline` makes a function into a *macro*, ie its usage in code simply pastes its definition in place as C-style macros did. This speeds up program execution by removing the overhead of function calling, however it means that the function is resolved at compile-time and thus requires a full recompilation if the function has changed. It also increases the executable size if code is excessively copied for this purpose.

8.17 Semaphores, Mutexes, and the `atomic` Keyword

A *semaphore* (or "counting" semaphore) is a flow control device used to control access to resources, like shared memory, in a parallelized program. Semaphores are basically numbers which can be incremented/decremented with "p" and "v" operations.

A *mutex* (short for mutual exclustion) is a binary *semaphore* (is a special case) which can be used to synchronize operations between threads in a multithreaded program. Unlike semaphores, only the thread which has "locked" the mutex can "unlock" it, so they are used more often to strictly control access to data.

As an alternative to mutexes, one can use `atomic` variables from C++ available in `<atomic>`. These are generally faster than using mutexes for simple data types. For example, if an `int count` is shared between threads which all may try to increment it at once, making it a `std::atomic<int>` instead will make the operations on it threadsafe without needing to use a mutex.

8.18 Compilation and Linking

Building code involves two steps: compilation and linking. First, the compiler performs preprocessing to deal with things like includes, defines and macros (preprocessing directives are specified with the `#` symbol), then it compiles the code into binary *object files*. These object files (which have a `.o` suffix) cannot actually be used because they do not include any definitions. The object files are then *linked* together and against external libraries and will catch definition (and multiple definition) errors, whereas it's the compiler than catches syntax errors.

There are two types of libraries in C/C++ - static and dynamic. *Static* libraries (usually ending in `.a`) are linked by essentially copying them into the resulting binary as many times as there are applications which link against them - this makes executables larger but reduces overhead by removing function calls.

Instead, we can create a *dynamically-linked library* (usually ending in `.so`) which are "Shared Object libraries." There exists one copy of the library which all applications linked against it will refer to. This reduces executable size and the amount of disk space and memory consumed.

Libraries in general are nice because they can be packaged up and provided to a user without the user having to know how functions are implemented; a header file and library file are sufficient for it to be used.

8.19 The `extern` Keyword

In C, the `extern` keyword is used to declare global variables. When declared as for example `extern int i`, the memory for `i` is not allocated and it thus cannot be used because it was never defined.

However, say we first `#include someheader.h` which defines `int i` - then `i` is defined and can be used. However, `extern int i = 1;` will be defined immediately because there is an initial value supplied.

For functions in C, `extern` tells the compiler that the function exists *somewhere else* and resolving it is deferred to the linker. However, this is actually done by default; all functions are actually effectively preceded by `extern` implicitly, which is what allows us to declare functions in a header and define them in a source file.

A special and important use of `extern` is to expose C++ code to be used in C as follows;

```
extern ''C'' {
  // code here
}
```

This exposes the code inside the brackets to C. Since the C++ compiler normally *mangles* the name of the function (to make it unique, by adding something about parameters, since overloading is possible), we need to explicitly tell it not to using this structure.

8.20 STL `map`

In C++, `std::map<key_type, value_type>` implements a map from keys to values for any data types. This is implemented as a binary search tree rather than using a hash function.

When we iterate through a map, we can use the `first` and `second` attributes of the iterator to access the key and value, respectively. For example:

```
#include <map>
#include <string>
#include <iostream>

std::map<std::string, int> mymap = {{``Apple'', 1}, {``Orange'', 2},
    {``Pear'', 3}};
for(const auto &it : mymap)
{
  std::cout << ``Key: `` << it.first << `` Value: `` << it.second << std::endl;
}
```

8.21 Error Handling

In C++ there are two common ways to handle errors in a program: exceptions and assertions. Generally, it is standard practice to use exceptions for catching errors in public functions (for example, someone passed in an invalid filename or there isn't sufficient memory to allocate resources) while asserations should be used within private functions to catch issues with your own code (used for internal testing to find things which should never happen in production code).

8.22 RAII

RAII or *Resource Allocation is Initialization* is a programming technique in object-oriented languages which binds the lifespan of a resource which must be acquired (for example, dynamically-allocated memory) to the lifespan of an object through which it is accessed.

A class can encapsulate such a resource according to RAII by allocating memory during initialization of an instance (in its constructor) and releasing the resource when it leaves scope or itself is deleted (in its destructor). A program then follows RAII if the resource is only ever used through an instance of the class.