Applied Math Reference

Robert Bridson

October 20, 2004

1 Introduction

This is a quick reference of most of the basic results in applied math that get used over and over in advanced computer graphics. I assume the reader has seen a lot of this already and just needs their memory jogging—or a few holes filled in—thus it is quite terse and doesn't provide much in the way of motivation or proof.

2 Linear Algebra

2.1 Vectors

A vector in $\mathbf{R}^n$ is a list of n real numbers. There's a variety of notation for vectors like this; I'll try to stick to $(x_1, x_2, \ldots, x_n)$, or simply $\vec{x}$ or just x if the context makes it clear. In the special cases of 2D and 3D, we usually use (x, y) or (x, y, z). The i'th component of a vector $\vec{x}$ is denoted x_i.

Vectors can be added (and subtracted) element-by-element

$$\vec{x} + \vec{y} = (x_1 + y_1, \ldots, x_n + y_n)$$

and multiplied by scalars (i.e. regular numbers, not vectors) in the same way

$$a\vec{x} = (ax_1, \ldots, ax_n)$$

The dot-product of two vectors is a scalar value defined by

$$\vec{x} \cdot \vec{y} = x_1 y_1 + x_2 y_2 + \ldots + x_n y_n$$

It is commutative which means it doesn't matter which order you write the vectors: $\vec{x} \cdot \vec{y} = \vec{y} \cdot \vec{x}$. The Euclidean length, or magnitude, of a vector is just

$$\begin{aligned} |\vec{x}| &= \sqrt{\vec{x} \cdot \vec{x}} \\ &= \sqrt{x_1^2 + x_2^2 + \ldots + x_n^2} \end{aligned}$$

This is also called the norm of the vector, or more specifically 2-norm or l_2-norm. The reason for the 2 is that the exponent applied to all the components is two. Other norms, or ways of measuring how big a vector is, exist. The two other common ones are the l_1-norm:

$$|\vec{x}|_1 = |x_1| + |x_2| + \ldots + |x_n|$$

and the l_∞-norm (also called max norm and sup norm – sup is pronounced like soup, and stands for supremum):

$$|\vec{x}|_\infty = \max\left(|x_1|, |x_2|, \ldots, |x_n|\right)$$

Unless specifically noted that something else is being used, always assume the 2-norm is being used.

A unit length vector is one that has magnitude equal to one: $|\vec{x}| = 1$. They are often written with a hat, instead of an arrow: $\hat{x}$. Any nonzero vector can be normalized to unit length (preserving the direction) by dividing by the magnitude: $\hat{x} := \vec{x}/|\vec{x}|$

The dot-product of two vectors has a geometric meaning: $\vec{x} \cdot \vec{y} = |x||y| \cos\theta$, where θ is the angle between the vectors. In particular, orthogonal (or perpendicular) vectors are ones where this angle is $90°$, or equivalently with a zero dot-product: $\vec{x} \cdot \vec{y} = 0$. When two vectors are parallel, the angle is 0, so $\vec{x} \cdot \vec{y} = |x||y|$, or for unit-length vectors $\hat{x} \cdot \hat{y} = 1$. Similarly if the vectors point exactly opposite to each other, the angle is $180°$, then $\vec{x} \cdot \vec{y} = -|x||y|$ or for unit-length vectors $\hat{x} \cdot \hat{y} = -1$.

A common operation is to project a vector onto another. Say we have a unit length vector $\hat{x}$. We can write any other vector $\vec{y}$ as the sum of a part that's parallel to $\hat{x}$ and a part that is orthogonal: $\vec{y} = (\hat{x} \cdot \vec{y})\hat{x} + (\vec{y} - (\hat{x} \cdot \vec{y})\hat{x})$. The first part of the sum, $(\hat{x} \cdot \vec{y})\hat{x}$, is the projection of $\vec{y}$ onto $\hat{x}$. The second part is sometimes called the orthogonalization of $\vec{y}$ with respect to $\hat{x}$. To orthonormalize $\vec{y}$, we additionally normalize the result of the operation to get a perpendicular unit length vector.

In 2D there's a special operator called *perp* (or sometimes *rot*) which rotates a vector $90°$ counter-clockwise to make it perpendicular:

$$(x, y)^\perp = (-y, x)$$

It's easy to verify that $|\vec{x}| = |\vec{x}^\perp|$ and that $\vec{x} \cdot \vec{x}^\perp = 0$.

In 3D there's also a special operator called the cross-product, defined by:

$$(u, v, w) \times (x, y, z) = (wy - vz, uz - wx, vx - uy)$$

Note the result is another 3D vector. This is not a commutative operator – if you change the order of the arguments, you flip the sign of the result: $\vec{u} \times \vec{x} = -\vec{x} \times \vec{u}$. The cross-product vector is automatically perpendicular to both of the arguments: $\vec{u} \cdot (\vec{u} \times \vec{x}) = \vec{x} \cdot (\vec{u} \times \vec{x}) = 0$.

Geometrically, the magnitude of the cross-product is $|\vec{u} \times \vec{x} = |u||x| \sin\theta$, where θ is again the angle between the vectors. So if the vectors are parallel or opposite, the cross-product is zero. If they are orthogonal ($\vec{u} \cdot \vec{x} = 0$) then $\vec{u} \times \vec{x} = |u||x|$. We cna also think of the direction of the cross-product geometrically—it is perpendicular (or normal) to the plane that goes through the two vectors. But, there are two directions that are perpendicular to this plane, opposite to each other—so which way is it? To answer this, we need to use our hands.

We'll always assume a right-handed coordinate system, so take your right hand. Line it up with the first vector, $\vec{u}$. Then curl your fingers towards the second vector $\vec{x}$. The direction that your thumb is pointing in is the direction of the cross-product $\vec{u} \times \vec{x}$. If you were using a left hand your thumb would point the other way—giving a left-handed system. Note that this choice, left vs. right, dictates how you draw the coordinate axes since they can be defined by cross-products, e.g. $(0, 0, 1) = (1, 0, 0) \times (0, 1, 0)$. Note that while right-handed coordinate systems are most common, left-handed systems are not unusual at all—most renderers use left-handed coordinates at least at the end of the graphics pipeline, and several engineering and physics fields have standardized on left-handed conventions.

Somewhat confusingly, people also define a cross-product for 2D vectors. This takes two 2D vectors and returns a scalar: $(u, v) \times (x, y) = vx - uy$. This is actually the third (and only nonzero) component of the 3D cross-product $(u, v, 0) \times (x, y, 0)$. Another way of defining it is $\vec{u} \times \vec{x} = \vec{u}^\perp \cdot \vec{x}$. One useful thing about this is that it lets us compute $\sin\theta$ easily.

A linear combination of vectors $\vec{x}_1$, …, $\vec{x}_k$ is just the sum of them, each multiplied by some scalar: $\alpha_1\vec{x}_1 + \ldots + \alpha_n\vec{x}_k$. A set of vectors is linearly independent if no nontrivial linear combination of them is zero (i.e. no combination other than $0\vec{x}_1 + 0\vec{x}_2 + \ldots + 0\vec{x}_k$). The span of a set of vectors is the vector space of all linear combinations; this is a subspace of $\mathbf{R}^n$. A basis of a subspace is a set of linearly independent vectors that span it, and the dimension of the subspace is the number of vectors in a basis.

2.2 Matrices

An $m \times n$ matrix A is an array of numbers written as:

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & & a_{2n} \\ \vdots & & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}$$

A square matrix is one where $m = n$, and a rectangular matrix is one where $m \neq n$. Just like vectors they can be added and subtracted element-by-element, and multiplied by scalars element-by-element.

A matrix is diagonal if only its diagonal entries are nonzero (and then can be written $A = diag(a_{11}, \dots, a_{nn}$. A matrix is lower triangular if all the entries above the diagonal are zero, and is upper triangular if all the entries below are zero.

An n-dimensional vector can be viewed as a skinny matrix, called a column vector if it's $n \times 1$ or a row vector if it's $1 \times n$. We'll generally always assume vectors are column vectors.

One way to think of an $m \times n$ matrix is as a a list of n column vectors, each of dimension m. We can write this as $A = (\vec{a}_1 | \vec{a}_2 | \dots | \vec{a}_n)$. You can similarly cut up a matrix into a list of row vectors.

You can multiply a matrix times a column vector, $y = Ax$, with the i'th component of the ($m \times 1$ column vector) result $y_i = \sum_{j=1}^{n} a_{ij} x_j$. Similarly you can multiply a row vector times a matrix by $y = xA$ with the j'th component of the ($1 \times n$ row-vector) result $y_j = \sum_{i=1}^{m} x_i a_{ij}$. Note that these have to come in that order—you cannot multiply a column vector times a matrix, or a matrix times a row vector.

More generally, you can multiply two matrices together if the dimensions match. That is, you can multiply a $m \times p$ matrix A by a $p \times n$ matrix B, to get an $m \times n$ result matrix C, with the formula $c_{ij} = \sum_{k=1}^{p} a_{ik} b_{kj}$. This multiplication is not commutative: in general, $AB \neq BA$, though it can happen in special circumstances.

The diagonal of a matrix is the list of elements $(a_{11}, a_{22}, \dots, a_{nn})$ (assuming $m \geq n$). The subdiagonal is the list of elements just below the diagonal, $(a_{21}, a_{32}, \dots, a_{nn-1})$. The superdiagonal is the list of elements just above the diagonal, $(a_{12}, a23, \dots, a_{n-1n})$.

The identity matrix I is a square matrix with all ones on the diagonal and zeros everywhere else: $I = diag(1, \dots, 1)$. It has the nice property that $Ix = x$ for all vectors x.

The inverse of a matrix A, if it exists, is a matrix A^{-1} so that the product is the identity: $AA^{-1} = I$. It doesn't matter which order you multiply them: $A^{-1}A = I$ too. Note that rectangular matrices never have inverses, and even some square matrices do not—they are called singular. The inverse of a product of square matrices is easy, $(AB)^{-1} = B^{-1}A^{-1}$, but remember to reverse the order of the matrices.

If you know the inverse of a matrix A, it's easy to solve a system of linear equations $Ax = b$, where b is given and you have to find x. Just multiply both sides by A^{-1} and simplify $A^{-1}Ax = Ix = x$ to get $x = A^{-1}b$.

A basis of the vector space $\mathbf{R}^n$ is a set of n vectors so that the matrix A with those as column vectors is invertible. For example, the standard basis has vectors e_1 up to e_n, with $e_i = (0, 0, \dots, 1, \dots, 0)$ the vector of all zeros except for a one in the i'th position. The matrix whose columns are the e_i vectors is just the identity, which isn't just invertible but is in fact its own inverse.

The column rank of a matrix is the dimension of the subspace spanned by its column vectors, or alternatively the maximum number of linearly independent columns. The row rank of a matrix is the same but for the rows. For a square matrix, the column rank and the row rank must be equal, and are simply called the rank. An $n \times n$ matrix is invertible if and only if its rank is n, i.e. all of its columns (or rows) are linearly independent. If the rank is less, the matrix is called rank-deficient.

The null-space of a matrix A is the vector space composed of all vectors $\vec{x}$ such that $A\vec{x} = 0$. If a matrix has any nonzero vectors in its null-space, it must be singular.

The range of a matrix A is the vector space composed of all vectors $\vec{y}$ that are equal to $A\vec{x}$ for some $\vec{x}$. It is equal to the span of the column vectors of A.

The transpose of a matrix is a reflection across its diagonal. We write it A^T, and define $(A^T)_{ij} = A_{ji}$. If A is $m \times n$, then A^T is $n \times m$. We can switch a column-vector to a row-vector or vice versa by transposition. The transpose of a product of matrices is easy, $(AB)^T = B^T A^T$, but remember to reverse the order of the matrices (just like inverses).

A square matrix is called symmetric if it's equal to its transpose, $A = A^T$, and skew-symmetric if it is equal to the negative, $A = -A^T$. Note all skew symmetric matrices must have an all zero diagonal.

With the transpose, we can now write the dot-product of two column vectors as $\vec{x} \cdot \vec{y} = x^T y$, and similarly the magnitude squared is $|\vec{x}|^2 = x^T x$.

The outer-product of two vectors is $\vec{x} \otimes \vec{y} = xy^T$. This is a matrix, who's (i, j)'th entry is $x_i y_j$. One common use of the outer-product is in rewriting the operation of projecting one vector onto another:

$$\begin{aligned} \vec{x}(\vec{x} \cdot \vec{y}) &= x(x^T y) \\ &= (xx^T)y = (x \otimes x)y \end{aligned}$$

The outer product matrix has to have rank one, and in fact all rank-one matrices are of this form.

We can also write the perp operator in 2D using a matrix-vector multiply:

$$\vec{x}^{\perp} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} x$$

The cross-product also is a matrix-vector multiply in disguise:

$$(u, v, w) \times (x, y, z) = \begin{pmatrix} 0 & -w & v \\ w & 0 & -u \\ -v & u & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

This matrix is the "star matrix" of the vector $\vec{u}$, written $u^\star$. Notice it is skew-symmetric. In fact, any 3×3 skew-symmetric matrix S is the star matrix of the vector (S_{32}, S_{13}, S_{23}).

An orthogonal matrix Q is one whose transpose is its inverse: $Q^{-1} = Q^T$, so that $QQ^T = Q^T Q = I$. Orthogonal matrices have the nice property that they preserve the length of a vector when they multiply, i.e. for orthogonal Q, $|Qx| = |x|$. The column vectors of Q are all orthonormal: unit length and perpendicular to each other. The same holds true for the row vectors.

If you multiply all the points in an object by an orthogonal matrix, then since no lengths between points can change, you must have rotated and/or reflected the object. In fact, any rotation and/or reflection can be represented by an orthogonal matrix.

The Gram-Schmidt algorithm turns a matrix A into an orthogonal matrix Q by orthonormalizing each column vector with respect to the ones before it. That is, the first column is $q_1 = a_1/|a_1|$, the second column q_2 results from orthonormalizing a_2 with respect to q_1, the third column q_3 from orthogonalizing a_3 with respect to q_1 and then the result of that with respect to q_2 and then normalizing, etc. It can be easily shown that this algorithm is equivalent to factoring A into the product QR where Q is the orthogonal matrix result, and R is an upper triangular matrix storing the coefficients used in orthonormalization. There are numerically better ways to compute the QR decomposition.

A rectangular orthogonal matrix is one whose columns or rows (whichever dimension is longer) are orthonormal. For example, if Q is taller than it is wide, it is orthogonal if the columnn vectors are orthonormal. This is equivalent to saying $Q^T Q = I$. Note that $QQ^T \neq I$ in this case.

A projection P is a matrix where if you apply it twice it doesn't change the result: $P^2 = P$. An orthogonal projection is of the form QQ^T where Q is a rectangular orthogonal matrix (usually taller than it is wide). In fact, the simplest case of this is the operation of projecting a vector onto another that we saw before, where Q was just a unit-length column vector.

An eigenvector u and associated eigenvalue λ of a matrix A satisfy $Au = \lambda u$. That is, when you transform (with A) vectors that happen to be parallel to u, all you're doing is scaling their length by λ without changing their direction. This is a very useful concept. So called normal matrices have a complete basis of eigenvalues; not all matrices have this property, but in practice it is unusual. A symmetric (real-valued) matrix always has a complete basis of real eigenvalues, and in fact a real orthonormal basis. Most other matrices do have a complete basis of eigenvalues, but they may be complex and have complex eigenvectors.

Note that there may be more than one linearly independent eigenvector for a single eigenvalue. (In the extreme case, the $n \times n$ identity matrix has just one eigenvalue, 1, but a set of n linearly independent eigenvectors for it.) The number of linearly independent eigenvectors, or equivalently the dimension of the so-called eigenspace spanned by them, is called the multiplicity of the eigenvalue. Often when we talk about the set of eigenvalues of a matrix, we will count each one as many times as its multiplicity, so an $n \times n$ normal matrix always has n eigenvalues even if some of them are repeated.

If U is a matrix with a complete basis of eigenvectors for A, then $U^{-1}AU$ is a diagonal matrix Λ, whose diagonal entries are the corresponding eigenvectors (the eigenvalues are repeated according to their multiplicity). Again, for a symmetric real A, U is orthogonal, so $U^{-1} = U^T$.

The trace of a matrix is the sum of the diagonal entries: $\text{tr}(A) = a_{11} + \ldots + a_{nn}$. This is automatically the same as the sum of the eigenvalues (counting eigenvalues according to their multiplicity).

The determinant of a matrix, written $\det(A)$, is the product of all its eigenvalues (counting eigenvalues according to their multiplicity). The determinant of a 2×2 matrix turns out to be $a_{11}a_{22} - a_{12}a21$, and the formula for a 3×3 matrix is:

$$\begin{aligned} \det(A) &= a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a13a_{21}a_{32} \\ &\quad -a_{13}a_{22}a_{31} - a_{12}a_{21}a_{33} - a11a_{23}a_{32} \end{aligned}$$

There are methodical ways to compute determinants for larger matrices, that avoid having to solve for the eigenvalues, but it's rarely necessary to do this.

A matrix is singular if and only if it has a zero eigenvalue, which also means if and only if its determinant is zero. The eigenspace of the eigenvalue 0 is just the null-space of the matrix.

A symmetric positive definite (SPD) matrix A is one that is symmetric and has the property that $x^TAx > 0$ for all vectors x. One common way that they arise is if A is the product B^TB of some other matrix B (which could even be rectangular). Another characterization is that all the eigenvalues are positive. Related concepts are negative definite matrices ($x^TAx < 0$ for all x, eigenvalues all negative), positive semi-definite matrices ($x^TAx \geq 0$, eigenvalues are ≥ 0), negative semi-definite matrices ($x^TAx \leq 0$, eigenvalues are ≤ 0), and indefinite matrices (all that's left—mixed positive and negative values).

A common way that symmetric positive (semi-)definite matrices arise is as the product of a possibly rectangular matrix and its transpose: $A = B^TB$. Note then $x^TAx = x^TB^TBx = (Bx)^T(Bx) = |Bx|^2 \geq 0$.

For example, consider solving the linear least squares problem: find a vector x such that $|Ax-b|$ is minimized. If A is square an invertible, then of course $x = A^{-1}b$ is the answer, but if A is singular or rectangular, life isn't as simple. One common case is that A is rectangular, taller than it is wide; then with some calculus (next section) you can show the answer is the solution to the "normal equations": $A^TAx = A^Tb$, i.e. $x = (A^TA)^{-1}A^Tb$. In this case, the matrix $(A^TA)^-1A^T$ is called the pseudo-inverse, and denoted A^+. Below we'll define the pseudo-inverse for an arbitrary matrix, and see different ways to compute it.

The singular value decomposition (SVD) of an arbitrary $m \times n$ matrix A is a way of writing it as the product $A = U\Sigma V^T$ where U is an $m \times m$ orthogonal matrix, V is an $n \times n$ orthogonal matrix, and Σ is a diagonal $m \times n$ matrix with non-negative entries. The diagonal of Σ is the vector $(\sigma_1, \sigma_2, \ldots)$ of so-called singular values. These are the square roots of the eigenvalues of A^TA (and AA^T). Typically they are written in decreasing order, so σ_1 is the biggest singular value. If the matrix A has rank

k, the first k singular values are nonzero and the rest are zero[1]. The right singular vectors are the columns of V, which happen to be the eigenvectors of A^TA, and the left singular vectors are the columns of U, which happen to be the eigenvectors of AA^T.

If A is SPD, then its singular values and eigenvalues are equal, and the eigenvectors and singular vectors (both left and right) are also equal. If A is just symmetric, then the singular values are the absolute values of the eigenvalues, and up to sign the eigenvectors match the singular vectors.

One can measure the magnitude of a matrix with a variety of norms. The 1-norm $|A|_1$ of a matrix A is the maximum 1-norm of any of its columns. The ∞-norm $|A|_\infty$ is the sum of the ∞-norms of its columns. These norms have the property that $|Ax|_1 \leq |A|_1|x|_1$ and $|Ax|_\infty \leq |A|_\infty|x|_\infty$ for all vectors x, and that these bounds are sharp (equality happens for some vector). You can define the 2-norm of a matrix by this property, $|Ax|_2 \leq |A|_2|x|_2$, but it's not as trivial to compute—it is equal to the maximum singular value. Finally the Frobenius norm $|A|_F$ is just the regular 2-norm of all the entries of the matrix: $|A|_F^2 = a_{11}^2 + a_{12}^2 + \ldots + a_{nn}^2$. An equivalent definition is in terms of the trace and the singular values: $|A|_F^2 = \text{tr}(A^TA) = \sigma_1^2 + \ldots + \sigma_n^2$.

Expanding the matrix product of the SVD, one can see it is a way of writing A as a sum of outer-products of the left and right singular vectors:

$$A = \sum_i \sigma_i \vec{u}_i \otimes \vec{v}_i$$

If you truncate this sum at the k leading terms, the result is a rank k approximation to A. The SVD has the nice property that this is the best rank k approximation possible, in both the 2-norm and Frobenius norm. This leads to another name for SVD, principle components analysis (PCA), which essentially says a good way to simplify some complicated and possible noisy linear transform is to take the first k terms in the SVD. These hopefully represent the most important parts of the transform, and the rest can safely be ignored as insignificant.

The SVD allows one to robustly solve least squares problems. The pseudo-inverse A^+ of any matrix A (even singular or rectangular matrices) is formed by taking the SVD $U\Sigma V^T$ and inverting it as much as possible: $A^+ = V\Sigma^+U^T$ where Σ^+ is the diagonal matrix with diagonal entries equal to the reciprocals of all the nonzero singular values and zeros for the rest. For an invertible matrix all the singular values are nonzero, and the pseudo-inverse is just the regular inverse. More generally, the pseudo-inverse solves the least-squares problem of minimizing $|Ax - b|_2$ and picking the x with minimum 2-norm among all possibilities: the answer is $x = A^+b$.

Solving the linear system $Ax = b$ can be accomplished with Gaussian elimination. This is a process where we combine equations, or equivalently add multiples of rows of A and entries of b to other ones to change the linear system to the easier form $Ux = y$, where U is upper triangular. This can be solved with back substitution (see below).

In fact, a little book-keeping shows that Gaussian elimination is actually factoring the matrix A into the product of a lower triangular matrix L with unit diagonal (all 1's) and an upper triangular matrix U: $A = LU$. Then solving the linear system is as easy as solving $Ly = b$ with forward substitution (see below) and then $Ux = y$ as before. One algorithm for finding L and U is to compute them row by row as: @@@

In the case of a symmetric positive definite matrix, with a little modification of the above algorithm we get the Cholesky factorization, $A = LL^T$. Note this is a different L, with diagonal entries that might not be one, and note that L^T is an upper-triangular matrix. Here's one algorithm for this:

Once an LU or Cholesky factorization for a matrix is found, linear systems can be solved with a sequence of triangular solves, namely forward then backward substitution.

Note that for LU, Cholesky, triangular solves, and in fact just about every common linear algebra operation, there are sophisticated implementations which run much faster and are much more robust than the naive approach—look for libraries called BLAS and LAPACK instead of coding these yourself. (One of the better implementations, available for just about any platform, is called ATLAS.)

[1]Numerically, the rank of a matrix a matrix is estimated by looking for a suddent drop from the first k singular values to much smaller values for the rest.

A sparse matrix is one where most of the entries are zero. It may be much more efficient to use special data structures to just store (and operate on) the nonzeros. These generally are written so it is fast and easy to multiply a sparse matrix times a vector. However, doing LU or Cholesky factorization creates new nonzeros and is not at all an easy thing to do fast (again, look for other people's optimized libraries rather than spend years trying to code this up). An alternative to solving a linear system by factorization by an iterative method, where an approximate guess is refined to be better and better in a number of steps, stopping when the error is small enough.

One way to measure how far away an estimate is from the true solution is to look at the residual, $r = b - Ax$. This is zero when the problem is solved; if the norm $|r|$ is very small, then the problem should be very close to solved. Typically we stop an iterative method as soon as $|r| < \texttt{tol}|b|$ where `tol` is some small tolerance such as 10^{-5}, or when the number of iterations exceeds some specified maximum. Some experimentation with parameters is almost always required.

One simple iterative method is Gauss-Seidel. It is guaranteed to converge (eventually) for SPD matrices, and many classes of unsymmetric matrices too, but may diverge for some problems. One way of writing the algorithm is:

will fill this in someday

For some problems, Gauss-Seidel is very efficient, but for others it is frustratingly slow.

For SPD matrices (and even semi-definite matrices), the method of Conjugate Gradients (CG) usually is best. It has many optimality properties, doesn't need much extra storage, and is quite robust.

will fill this in someday

To improve on CG, preconditioners are often used. These are linear transformations that approximate the action of A^{-1} cheaply. The most popular class of preconditioners are ones that approximate the Cholesky factorization of A: $LL^T \approx A$. For example, one can modify the algorithm to compute the Cholesky factorization so that no new nonzeros values are introduced (if you compute any, just throw them away)—this is called Incomplete Cholesky, level zero. The CG algorithm is then modified slightly, resulting in Preconditioned Conjugate Gradient (PCG).

Finally, often in physical applications it is convenient notationally to imbue matrices and vectors with a block structure. That is, grouping together parts of the matrix and vector into small submatrices and subvectors. Alternatively put, the entries of the matrix become small matrices instead of just real numbers. This is purely for convenience in notation, and does not change the mathematics above (though it usually makes sense to program with this block structure made explicit).

3 Multi-Variable Calculus

3.1 Derivatives

Let $q(\vec{x})$ be a scalar function of space, also called a scalar field. Then the gradient is ∇q (also written grad$q$), a vector composed of the partial derivatives with respect to all the dimensions of space:

$$\nabla q = \partial q / \partial \vec{x} = \left(\frac{\partial q}{\partial x_1}, \ldots, \frac{\partial q}{\partial x_n} \right)$$

A directional derivative is when we take a slice of space along one particular line, look at the function q as a one-dimensional function along that line, and take the regular derivative. If $\hat{n}$ is a unit vector in the direction of that line, the directional derivative is $\partial q / \partial \hat{n} = \nabla q \cdot \hat{n}$.

Look now at a vector valued function $\vec{f}(\vec{x})$ of space, also called a vector field. Say $\vec{f}$ is in $\mathbf{R}^m$ and $\vec{x}$ is in $\mathbf{R}^n$. Then the gradient of $\vec{f}$ is an $m \times n$ matrix called the Jacobian (often denoted J), each row being the gradient of one component of $\vec{f}$.

That is $\nabla \vec{f} = \partial \vec{f} / \partial \vec{x} = J$ has the ij'th entry $J_{ij} = \partial f_i / \partial x_j$. The directional derivative still makes sense, and is vector valued: $\partial \vec{f} / \partial \hat{n} = J\hat{n}$ using a regular matrix-vector multiply.

One common source for a vector field is the gradient of a scalar, $\vec{f}(\vec{x}) = \nabla q(\vec{x})$. The Jacobian of this vector-valued function is called a Hessian, usually denoted by H, and is the square symmetric matrix of all second partial derivatives of q: $H_{ij} = \partial^2 q / \partial x_i \partial x_j$.

At a minimum or maximum of a function, its gradient (or Jacobian) is zero. For a scalar function, at a maximum the Hessian is symmetric negative semi-definite and at a minimum the Hessian is symmetric positive semi-definite.

The divergence of a vector field $\vec{f}(\vec{x})$, written $\text{div}\vec{f} = \nabla \cdot \vec{f}$, is a special scalar combination of its partial derivatives:

$$\nabla \cdot \vec{f} = \frac{\partial f_1}{\partial x_1} + \frac{\partial f_2}{\partial x_2} + \ldots + \frac{\partial f_n}{\partial x_n}$$

This measures how much the vector field is "expanding" or "contracting": if $\vec{f}$ points outward from some point in all directions (expanding), its divergence will be positive, and if it points inward to some point from all directions (contracting), its divergence will be negative.

The curl of a 3D vector field in 3D, written $\text{curl}\vec{f} = \nabla \times \vec{f}$, is a vector combination of partial derivatives, related to the cross-product:

$$\nabla \times \vec{f} = \left(\frac{\partial f_2}{\partial x_3} - \frac{\partial f_3}{\partial x_2}, \frac{\partial f_3}{\partial x_1} - \frac{\partial f_1}{\partial x_3}, \frac{\partial f_1}{\partial x_2} - \frac{\partial f_2}{\partial x_1} \right)$$

The Laplacian of a scalar field $q(\vec{x})$ (or components of a vector field treated separately, one by one) is the divergence of its gradient ("div grad"). This can be written in several forms: $\Delta q = \nabla^2 q = \nabla \cdot \nabla q$. Written out, is is equal to:

$$\nabla^2 q = \frac{\partial^2 f_1}{\partial x_1^2} + \frac{\partial^2 f_2}{\partial x_2^2} + \ldots + \frac{\partial^2 f_n}{\partial x_n^2}$$

3.2 Integration

3.3 Approximation

A common operation in applied math is to approximate a smooth nonlinear function with a linear function in the neighbourhood of some point. In one dimension, where the nonlinear function is $f(x)$ and the point we're going to base the approximation on is x_0, we just have to evaluate $f(x_0)$ and figure out the slope of the the function there with a derivative. This gives us a value and a slope, which specifies a line. The approximation is then

$$f(x) \approx f(x_0) + \left. \frac{df}{dx} \right|_{x_0} (x - x_0)$$

This generalizes perfectly well to multiple dimensions. You can think of doing it dimension by dimension, e.g. fitting a line for the function along the x-axis, then another along y-axis, then the z-axis, etc. and putting a hyperplane through all those lines. All this boils down to is replacing the derivative in the above formula with a gradient (or Jacobian):

$$\vec{f}(\vec{x}) \approx \vec{f}(\vec{x}_0) + \left. \frac{\partial \vec{f}}{\partial \vec{x}} \right|_{\vec{x}_0} (\vec{x} - \vec{x}_0)$$

Note now that the product is between a matrix and a vector now. If $f(\vec{x})$ is scalar valued, its Jacobian is just a row vector, so sometimes people write this product as a dot-product:

$$f(\vec{x}) \approx f(\vec{x}_0) + \nabla f|_{\vec{x}_0} \cdot (\vec{x} - \vec{x}_0)$$

Sometimes it may be necessary to use a more accurate approximation to a nonlinear function. This is often done with Taylor series. We an think of this as matching a polynomial $p(x)$ to $f(x)$ so that the value is the same at x_0, the first derivative is the same at x_0, the second derivative is the same at x_0, and so on up to the degree we choose for the polynomial. The Taylor series is just a formula for the terms of such polynomials:

$$f(x) \approx f(x_0) + \frac{df(x_0)}{dx}(x - x_0) + \frac{1}{2}\frac{d^2f(x_0)}{dx^2}(x - x_0)^2 + \frac{1}{6}\frac{d^3f(x_0)}{dx^3}(x - x_0)^3 + \dots$$

The k'th term is

$$\frac{1}{k!}\frac{d^kf(x_0)}{dx^k}(x - x_0)^k$$

There are various theorems to show that if you truncate a Taylor series up to and including the k'th term (to get a degree k polynomial), the error you make behaves like the missing $k + 1$ term. This is often written with $O()$ notation, for example

$$f(x) = f(x_0) + \frac{df(x_0)}{dx}(x - x_0) + \frac{1}{2}\frac{d^2f(x_0)}{dx^2}(x - x_0)^2 + O((x - x_0)^3)$$

One important thing about this is that if the $k + 1$ derivative of the function is very large (or doesn't even exist) this indicates the approximation will be useless. So don't use high order Taylor series unless the function is appropriately smooth.

Note that the first-degree Taylor series approximation is the same as the linear approximation we made at the start of the section. Just as that could be generalized to multiple dimensions, so can Taylor series, but the notation gets a lot hairier (you really need tensors).

3.4 Nonlinear Methods

4 Quaternions

Quaternions are a generalization of complex numbers that have found a niche for efficiently working with rotations in 3D. We'll begin with the basics of quaternion algebra, then look at this application.

4.1 Complex number review

Complex numbers can be viewed as what you get when you include the square root of negative one with the real numbers, and let all the usual algebraic operations (plus, minus, multiply, divide) generate everything they can with it. Let i represent $\sqrt{-1}$. Then the complex numbers are of the form $a + bi$ where a and b are real. Recall the basic operations on complex numbers:

$$\begin{aligned} -(a + bi) &= (-a) + (-b)i \\ (a + bi) + (c + di) &= (a + c) + (b + d)i \\ (a + bi) * (c + di) &= ac + bci + adi + bdi^2 = (ac - bd) + (bc + ad)i \end{aligned}$$

and also the notion of the complex conjugate:

$$\bar{a + bi} = a - bi$$

which lets us define the absolute value (or "magnitude", or "modulus") of a complex number z:

$$|z| = \sqrt{z\bar{z}} \quad \text{or alternately} \quad |a + bi| = \sqrt{(a + bi)(a - bi)} = \sqrt{a^2 + b^2}$$

(Note the critical thing is that $z\bar{z}$ is always real and non-negative.) This also gives us an easy formula for complex division:

$$y/z = y\bar{z}\frac{1}{|z|^2}$$

4.2 Basic quaternion definition

Quaternions are what you get when you include in two more (different) square roots of negative one, which we'll call j and k. So $i^2 = j^2 = k^2 = -1$, but i, j, and k are all different (they are not equal to each other or their negatives). So all quaternions are of the form $a + bi + cj + dk$ where a, b, c and d are real numbers. Negation and addition work in the obvious way:

$$\begin{aligned} -(a + bi + cj + dk) &= (-a) + (-b)i + (-c)j + (-d)k \\ (a + bi + cj + dk) + (e + fi + gj + hk) &= (a + e) + (b + f)i + (c + g)j + (d + h)k \end{aligned}$$

Multiplication is where life gets complicated. It turns out that the algebra of quaternions cannot be *commutative*, that is for two quaternions p and q, the product pq might not be equal to qp. (Multiplication is commutative for complex numbers, which simplifies life a lot). We'll build up the multiplication rule from specifying that

$$\begin{array}{ccc} ij = k & jk = i & ki = j \\ ji = -k & kj = -i & ik = -j \end{array}$$

From those rules we can simplify down any product to the basic form. The general rule is then:

$$(a+bi+cj+dk)*(e+fi+gj+hk) = (ae-bf-cg-dh)+(af+be+ch-dg)i+(ag-bh+ce+df)j+(ah+bg-cf+de)k$$

The quaternion conjugate is just like the complex conjugate:

$$\bar{a + bi + cj + dk} = a - bi - cj - dk$$

and let's us define the absolute value (or magnitude) of a quaternion q:

$$|q| = \sqrt{q\bar{q}} \quad \text{or alternately} \quad |a + bi + cj + dk| = \sqrt{a^2 + b^2 + c^2 + d^2}$$

You can easily verify that the multiplication of $a+bi+cj+dk$ with its conjugate really does simplify down to the non-negative real number $a^2 + b^2 + c^2 + d^2$.

Finally, with the conjugate it's easy to come up with a formula for quaternion division:

$$p/q = p\bar{q}\frac{1}{|q|^2}$$

4.3 Vector definition of quaternions

There is another way of thinking about quaternions, as a real number together with a vector. That is, thinking of $q = a + bi + cj + dk$ as the real number a together with the vector $\vec{u} = (b, c, d)$: $q = (a, \vec{u})$. The basic operations then can be written a little more simply:

$$\begin{aligned} -(a, \vec{u}) &= (-a, -\vec{u}) \\ (a, \vec{u}) + (e, \vec{v}) &= (a + e, \vec{u} + \vec{v}) \\ (a, \vec{u}) * (e, \vec{v}) &= (ae - \vec{u} \cdot \vec{v}, a\vec{v} + e\vec{u} + \vec{u} \times \vec{v}) \\ \bar{(a, \vec{u})} &= (a, -\vec{u}) \end{aligned}$$

4.4 Rotations

5 Basic Physics

6 Ordinary Differential Equations

6.1 Numerical methods for first order systems

Here we suppose we have the first order differential equation

$$\frac{dx}{dt} = v(x,t)$$

Note that x and v could be vectors—not just three-dimensional vectors, but even extremely high dimension vectors that list out the three-dimension positions of many different particles for example. In the case where they are vectors, this is really a first order system of differential equations.

There are many numerical methods that can approximate the solution of this equation. Here we will go through the simplest and most popular ones.

6.1.1 Forward Euler

This is the simplest possible method. If we have the position x_n at time t_n, we get the next position at time $t_{n+1} = t_n + \Delta t$ from the formula:

$$x_{n+1} = x_n + \Delta t v(x_n, t_n)$$

6.2 Numerical methods for second order systems

Second order systems are really just a special case of first order systems. They almost always come from Newton's second law, $F = ma$, so we usually write them as:

$$\begin{aligned} \frac{dv}{dt} &= \frac{1}{m} f(x,v,t) \\ \frac{dx}{dt} &= v \end{aligned}$$

Note that if we rolled up x and v into one big vector, this would indeed be just a first order system. Thus you can use any method for first-order systems to solve second-order systems.

There are some methods which take advantage of the special structure. Here are some of the simpler and more popular ones.

6.2.1 Symplectic Euler

This goes by a number of different names, such as Forward-Backward Euler. It looks almost like Forward Euler, except that we first update velocity, then update position using the new value of velocity instead of the old one:

$$\begin{aligned} v_{n+1} &= v_n + \Delta t \frac{1}{m} f(x_n, v_n, t_n) \\ x_{n+1} &= x_n + \Delta t v_{n+1} \end{aligned}$$

Again, note that Forward Euler would be almost identical, just with a v_n in the second line instead. However, this makes a world of difference. Whereas Forward Euler is terribly unstable for undamped physical systems (e.g. springs), Symplectic Euler handles them just fine.

7 Partial Differential Equations

7.1 Laplace Equation

7.2 Heat Equation

7.3 Advection Equation

7.4 Second Order Wave Equation

7.5 Conservation Laws

7.6 Hamilton-Jacobi Equations