Scientific Computing Summary

Rebecca Sela

May 7, 2005

1 Analyzing General Problems

1.1 The Condition of a Problem

Let P be a problem with data d and solution $s(P, d)$. We have a criterion function, $\Delta(\cdot, \cdot, \cdot)$ such that $\Delta(s, P, d) = 0$. (Ideally, $\Delta(s', P, d)$ is small when the distance from s to s' is small. This is not always true.)

Definition The *condition number* of a problem is a measure that reflects the largest possible relative change in the exact solution of a problem resulting from changes in the data. The condition number may depend on the problem and the criterion function for the solution, but does not depend on the numerical method used for solving it. (That is an issue of *stability*.)

Definition A problem is *well-conditioned* if small changes in the data always lead to small changes in the exact solution. A problem is *ill-conditioned* if small changes in the data may lead to large changes in the exact solution.

1.2 Error Analysis

Definition In *forward error analysis*, a solution is good if the computed solution is close to the exact solution.

This is hard to check, since the exact solution is generally not known. Also, this will reject any solution method for ill-conditioned problems.

Definition In *backward error analysis*, one interprets the computed solution as the exact solution of a different problem and checks how the data has changed.

In general,

$$forward - error \leq backward - error \times condition - number$$

Definition An algorithm is *backward stable* if the result of the algorithm is the exact solution of a nearby problem.

1.3 Errors in Finite Precision

Numbers in computers must be stored in finite precision. Generally, they are stored as a number of significant digits and a number that indicates where the decimal goes, so that 0.43 and 43 have the same precision. Numbers are made finite precision by *rounding* (to either the nearest even or the nearest odd if the part being rounded is exactly 1/2) and *truncation.*

Definition Suppose x is an exact number and x' is an approximation to it. Then, the *absolute error* is $|x - x'|$, and the *relative error* is $\frac{|x-x'|}{x}$, if $x \neq 0$. The *mixed error* is $\frac{|x-x'|}{1+|x|}$.

Definition *Cancellation* is the subtraction of nearly equal numbers. *Cancellation error* is the error due to subtracting nearly equal numbers due to rounding (since many digits of the difference may have been lost in the rounding).

"Cancellation reveals the error of rounding."

In *error analysis*, for any number, x, we have $fl(x) = x(1+\xi_x)$ which is the floating point value of x (where $|\xi_x| \leq \epsilon_m$ is the relative error which is less than machine precision). We then see how the errors combine:

$$\begin{aligned} fl(fl(x)+fl(y)) &= (fl(x)+fl(y))(1+\delta) \\ fl(\frac{fl(x)}{fl(y)}) &= \frac{fl(x)}{fl(y)(1+\delta)} \\ \delta &\leq \epsilon_m \end{aligned}$$

This gives a worst case bound on the errors (they may not actually be this bad).

The *maximum achievable accuracy* is the range of solutions that cannot be distinguished from the true solution because of machine precision. (For example, values very close to the true root of an equation may still evaluate to 0 in finite precision.) The accuracy may depend on the condition number of the problem and the computation method, as well as on machine precision.

1.4 Order notation

Definition Let $\phi(h)$ be a function of $h > 0$. Let p be fixed. If there exists a constant, $M > 0$ such that $|\phi(h)| \leq Mh^p$ for all sufficiently small h, we say that $\phi = O(h^p)$.

1.5 Rates of convergence

To judge an algorithm, we consider the sequence of iterates, $\{x_k\}$ that it generates. The first qualification is that $x_k \to x^*$, where x^* is the true solution to the problem. If that holds, then we want to know how fast it converges.

Definition Suppose $x_k \to x^*$. We say that $\{x_k\}$ is *linearly convergent* if:

$$\|x_{k+1} - x^*\| \leq \alpha \|x_k - x^*\|$$

with $0 < \alpha < 1$. α is called the *asymptotic error constant.*

Definition Suppose $x_k \to x^*$. We say that $\{x_k\}$ is *quadratically convergent* if:

$$\|x_{k+1} - x^*\| \leq \beta \|x_k - x^*\|^2$$

with $0 < \beta$. β is called the *asymptotic error constant.*

Definition Suppose $x_k \to x^*$. We say that $\{x_k\}$ is *super-linearly convergent* if:

$$\|x_{k+1} - x^*\| \leq \beta_k \|x_k - x^*\|$$

with $\beta_k \to 0$.

1.6 Richardson Extrapolation

Suppose we have an approximation method and we know the order of the error as $O(h^p)$ for a relevant h. Then, we may calculate the approximation with two different values of h and take a linear combination of the two approximations so that the highest order error (approximately) cancels out. This is called *Richardson extrapolation.* Continuing this with more terms to get rid of more orders of error is called *Extrapolation to the Limit.*

Similar methods may be used to estimate the error at any point (to know whether h really needs to be reduced) or to identify potential problems in the algorithm.

2 Solving One-dimensional Non-linear Equations

Problem: Solve the equation $f(x) = 0$. (It is not required that we have a closed form description of f.)

This problem has some limitations:

- Some functions have no solutions or multiple solutions.
- Some solutions occur when the function is tangent to the x-axis, which makes them harder to find.
- The solution of an arbitrary function (even a continuous one) cannot be found in a finite number of steps.
- Finite precision means that some functions look identical even though they are not exactly identical.
- Infinitely many functions have exactly the same solutions, and different methods may be more useful for different functions.

Algorithm: Bisection

- Suppose f is continuous on $[a_0, b_0]$, with $f(a_0)f(b_0) < 0$.

- Define $x_k = \frac{a_{k-1}+b_{k-1}}{2}$. If $f(x_k) = 0$, we have found the solution. If $f(a_{k-1})f(x_k) < 0$, set $a_k = a_{k-1}$ and $b_k = x_k$. Otherwise, we must have $f(x_k)f(b_{k-1}) < 0$, and we set $a_k = x_k$ and $b_k = b_{k-1}$.
- We terminate when either $f(x_k) = 0$ (in the finite precision sense) or the iterates are very close ($b_k - a_k < \epsilon_m$, for example).

This algorithm is guaranteed to halve the length of the interval each time and will therefore always find a zero. It is optimal in the worst case.

Theorem 2.1 *Suppose a function, f, and its first two derivatives exist and are continuous. Then, we may write:*

$$f(x+\delta) - f(x) = \delta f'(x) + \frac{1}{2}f''(\xi)\delta^2$$

where $\xi \in [x, x+\delta]$. This is called the Taylor expansion.

Algorithm: Newton's Method

- Suppose we know that f is differentiable and we can evaluate the derivative (numerically or otherwise). We also assume that we have a starting value, x_0, close enough to the solution that a Taylor series approximation is reasonable.
- Define $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$. (This solves the linear equation $f(x^*) \approx f(x_k) + (x^* - x_k)f'(x_k) = 0$.)

This has quadratic convergence in most cases. However, it has problems when the second-order term of the Taylor series is large, when $f'(x_k)$ is close to zero, or when f has a multiple root. For certain functions, it may also oscillate. In addition, evaluation of a derivative can be even harder numerically.

Algorithm: Secant Method

- Suppose we have any reasonably smooth function, f, and two starting values, a_0 and b_0.
- Define $x_{k+1} = x_k - f(x_k)\frac{x_k - x_{k-1}}{f(x_k)-f(x_{k-1})}$. (This finds the zero of the line passing through the points $(x_{k-1}, f(x_{k-1}))$ and $(x_k, f(x_k))$.)

This method converges super-linearly ($\|x_{k+1} - x^*\| \leq \beta\|x_k - x^*\|^{1.6180}$). However, this method can also get stuck in cycles or not converge. It also extrapolates in some case, which makes the interval bigger.

False Position (Regula Falsi) Method: Use the Secant Method, but choose to keep x_k if $f(x_k)f(x_{k+1}) < 0$ and x_{k-1} if $f(x_{k-1})f(x_{k+1}) < 0$. Though this seems to be an improvement, it can slow down convergence quite dramatically.

These methods can be combined using safeguarding; a “fast” method is used, but iterates are rejected if they fall outside a certain “interval of uncertainty”, and forced interval reductions occur from time to time. (These are more complicated, but are a way to get fast convergence in certain cases while being on the order of bisection in worse cases.)

3 Solving Linear Systems

3.1 Matrix Preliminaries

Definition Given a matrix, A, of dimension $m \times n$, the *singular value decomposition* is $A = U\Sigma V^T$, where U and V are square orthogonal matrices and Σ is an $m \times n$ diagonal matrix, with descending non-negative entries, $\sigma_1, ..., \sigma_{\min(m,n)}$. The values σ_i are called the *singular values* of A.

Some facts about singular values:

- If A is a square non-singular matrix, all of its singular values are non-zero. In general, $rank(A)$ is the number of non-zero singular values of A.
- The singular values of A are the square roots of the eigenvalues of A^TA (if $m \geq n$) or AA^T. If A is symmetric, its singular values are the absolute values of its eigenvalues.
- The columns of V corresponding to the zero singular values for an orthonormal basis for $null(A)$. The columns of V corresponding to non-zero singular values form an orthonormal basis of $range(A^T)$.
- The columns of U corresponding to the zero singular values for an orthonormal basis for $null(A^T)$. The columns of U corresponding to non-zero singular values form an orthonormal basis of $range(A)$.

Definition If $x \in R^n$, a *vector norm*, $\|x\|$, is a function such that:

1. $\|x\| > 0$, if $x \neq 0$.
2. $\|\alpha x\| = |\alpha|\|x\|$.
3. $\|x + y\| \leq \|x\| + \|y\|$ (the triangle inequality).

Three common norms are:

1. The Manhattan Norm (or the One Norm): $\|x\| = \sum_{i=1}^{n} |x_i|$
2. The Two Norm (the Euclidean Length): $\|x\| = \sqrt{\sum_{i=1}^{n} x_i^2}$
3. The Infinity Norm: $\|x\| = \max_{1 \leq 1 \leq n} |x_i|$.

Definition If A, B are matrices, a matrix norm is a function such that:

1. $\|A\| > 0$, if $A \neq 0$.
2. $\|\alpha A\| = |\alpha|\|A\|$.
3. $\|A + B\| \leq \|A\| + \|B\|$ (the triangle inequality).
4. $\|AB\| \leq \|A\|\|B\|$.

Definition Let a vector norm be given. The *subordinate induced matrix norm* is defined by:

$$\|A\| = \max \frac{\|Ax\|}{\|x\|}$$

over all $x \neq 0$. A matrix norm is *compatible* with a given vector norm if

$$\|A\|\|x\| \geq \|Ax\|$$

for all x and A.

The subordinate matrix norm induced by the one-norm is the maximum absolute column sum, and the subordinate matrix norm included by the infinity norm is the maximum absolute row sum. The subordinate matrix norm induced by the two-norm is the maximum singular value.

Definition The *Frobenius norm* of a matrix, A, is defined by $\|A\|_F = \sqrt{\sum_i \sum_j a_{ij}^2}$.

The Frobenius norm satisfies $\|A\|_F\|x\|_2 \geq \|Ax\|_2$ (so this norm is compatible with the two-norm, but is not induced by it).

Definition A square matrix, A, is *non-singular* if $Ax \neq 0$ whenever $x \neq 0$. A is *singular* if there exists some $x \neq 0$ such that $Ax = 0$.

The problem of solving a linear system is finding x for a given non-singular matrix, A, and a vector, b, such that $Ax = b$.

Definition The *condition number* of a matrix, A, is given by:

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

Note that $cond(A) = \|A\|\|A^{-1}\| \geq \|I\| = 1$, which gives a lower bound on the condition number. If a solution, x, is found, then we may find a lower bound of the condition number as:

$$cond(A) = \|A\|\|A^{-1}\| \leq \|A\|\frac{\|x\|}{\|b\|}$$

since $x = A^{-1}b$. In the two-norm, the condition of A is the ratio of its maximum to its minimum singular values.

3.2 Solving lower triangular systems

Algorithm: Solving $Lx = b$

- We are given a matrix, L, with entries l_{ij} on and below the diagonal only, and any vector, b.
- Set $x_1 = \frac{b_1}{l_{11}}$.

- Given $x_1, ..., x_{k-1}$, we compute $x_k = \frac{1}{l_{kk}}(b_k - l_{k1}x_1 - ... - l_{k,k-1}x_{k-1}$.

The k^{th} step of this process takes 1 division, $k-1$ multiplications, and $k-1$ additions. Thus, the total work is n divisions and $\frac{n(n-1)}{2}$ multiplications and additions (flops), and this algorithm is $O(n^2)$.

We conduct an error analysis. First, let x and y be n-vectors. Then,

$$fl(x^Ty) = fl(x_1y_1 + ... + x_ny_n)$$

Let $t_i = fl(x_iy_i) = x_iy_i(1+\xi_i)$. Then, the partial sums of the t_i are $s_i = fl(s_{i-1} + t_i) = (s_{i-1} + t_i)(1+\eta_i)$, and

$$fl(x^Ty) = x_1y_1(1+\epsilon_1) + ... + x_ny_n(1+\epsilon_n)$$

where $1+\epsilon_k = (1+\xi_i)(1+\eta_i)(1+\eta_{i+1})...(1+\eta_n)$. (The earlier terms are subject to more error because they are subjected to more additions.) We may write $fl(x^Ty)$ as the exact inner product of the two perturbed vectors, x and $(y_1(1+\epsilon_1), ..., y_n(1+\epsilon_n))^T$.

Applying this to $Lx = b$, we find that the computed solution is the exact solution to $(L+\delta L)x = b$, with the bound,

$$|\delta L_{ij}| \leq 1.06(i-j+2)\epsilon_m|l_{ij}|$$

Note that this bound does not depend on b or on the condition number of L. (Alternatively, x is the exact solutions of $Lx = b + \delta b$ as well.)

Definition A *normalized triangle* is a triangular matrix in which the largest element of each row is on the diagonal. A *unit triangle* is a matrix in which all the diagonal elements are one.

(Normalized unit triangles are generally but not always well-conditioned.)

3.3 General Non-singular Linear Systems

Recall that:

$$AB = A[b_1, ..., b_n] = [Ab_1, ..., Ab_n]$$

where Ab_i is a linear combination of the rows in the i^{th} column of B.

Definition An *elementary matrix* is a matrix of the form $I - \alpha uv^T$ where α is a scalar.

If u and v are column vectors, then uv^T is an $n \times n$ matrix with

$$uv^T = [v_1u, ..., v_nu] = \begin{bmatrix} u_1v^T \\ ... \\ u_nv^T \end{bmatrix}$$

An elementary matrix is non-singular if and only if $\alpha u^Tv \neq 1$. For any x,

$$(I - \alpha uv^T)x = x - \alpha u(v^Tx) = x - \alpha(v^Tx)u$$

and multiplying x by an elementary matrix subtracts a multiple of u from x.

Suppose we want to solve $Ax = b$ where A is non-singular.

Algorithm: Gaussian Elimination

- Step 1: Clear out all the entries in the first column of A; that is, reduce $a_1 = \begin{bmatrix} a_{11} \\ a_{22} \\ \dots \\ a_{n1} \end{bmatrix}$ to $\begin{bmatrix} a_{11} \\ 0 \\ \dots \\ 0 \end{bmatrix}$.
  - Choose $u_1 = \begin{bmatrix} 0 \\ a_{21}/a_{11} \\ \dots \\ a_{n1}/a_{11} \end{bmatrix}$, $\alpha_1 = 1$,$v_1 = e_1$.
  - Then, $\alpha_1 v_1^T a_1 = a_{11}$ and
  $$(I - u_1 e_1^T)a_1 = a_1 - u_1 a_{11} = \begin{bmatrix} a_{11} \\ 0 \\ \dots \\ 0 \end{bmatrix}$$
  - We call a_{11} the *pivot* and $m_{k1} = a_{k1}/a_{11}$ the *multipliers.* Define $M_1 = (I - u_1 e_1^T)$.
- Step 2: This leaves a square submatrix, $A^{(2)}$, based on all but the first row and column, on which we repeat the process, finding $M_2, \dots, M_{n-1}$.
- Step 3: This eventually leads to an upper triangular matrix, U, such that $U = M_{n-1} \dots M_1 A = MA$. Then, $Mb = MAx = Ux$, and we may solve for x as before.

Variant: LU Factorization

- We may write $A = M^{-1}U = LU$. M is lower triangular (because each M_i is lower triangular), so L is lower triangular. This allows L and U to take up only as much storage together as A.
- It turns out that the diagonal elements of L are 1 and the elements on the lower triangle are simply the multipliers, $m_{ij} = a_{ij}/a_{ii}$.
- Given the LU factorization, we first solve the lower triangular system, $Ly = b$, and then the upper triangular $Ux = y$.

The number of operations involved is $O(n^3/3)$, with $k-1$ divisions (for the multipliers) and $(k-1)^2$ multiplications and subtractions (for the other $k-1$ elements of the k^{th} row and the $(k-1)$ rows below) for each step.

LU factorization also solves systems involving A^T, since $A^T = U^T L^T$.

Definition A *permutation matrix* is a matrix of ones and zeroes with exactly one 1 in each row and each column (it is the identity matrix with the order of the rows changed).

If P is a permutation matrix, then PA interchanges the rows of A and AP interchanges the columns of A. In Gaussian elimination, we need the pivots to be non-zero. If A is non-singular, there is always some P such that $PA = LU$.

In finite precision, problems also occur if pivots are close to 0, since they cause the multipliers to be very large.

Definition The problem of large entries caused by small pivots is *growth.* We define the *growth factor* as:

$$g_n = \rho_n = \frac{\max_k \max_{i,j} |a_{ij}^{(k)}|}{\max_{i,j} |a_{ij}^{(1)}|}$$

where $a_{ij}^{(k)}$ is the i, j entry after the k^{th} step.

There are three strategies to eliminate small pivots and thereby reduce growth:

- Gaussian Elimination with Partial Pivoting (GEPP): At each step, choose the element of largest magnitude in the current column and switch rows so that element is the pivot. This ensures that the multipliers are always less than one in absolute value; the additional time required for all the comparisons is $O(n^2)$.
- Complete Pivoting: Find the largest element in magnitude in the entire submatrix and use both column and row interchanges to make it the pivot. This requires $O(n^3)$ comparisons.
- Threshold Pivoting (for sparse matrices): Interchange rows (as in partial pivoting) only when the current pivot is less than some fraction of the largest element.

In most cases, partial pivoting is good enough to keep growth minimal. In the worst case, however, partial pivoting still leads to growth factors that are $O(2^{n-1})$.

With partial pivoting, we end up with a string of elementary and partition matrices, $M_{n-1}P_{n-1}...M_1P_1A = U$. Because the P_i are orthogonal, we may move them together (by interchanging the rows and columns of the M_i as well, to keep them lower triangular), to get a factorization of the form $MPA = U$ or $PA = LU$.

Error analysis of GEPP:

- For the computed L and U, $A+\Delta_A = LU$, where $\|\Delta_A\|_\infty \leq n\gamma_a g_n \|A\|_\infty \epsilon_m$, where the growth factor, g_n, tends to be the big problem.

- The computed x based on the LU decomposition is the exact solution of $(A+E)x = b$, where $\|E\|_\infty \leq n^3 \gamma_E g_n \|A\|_\infty \epsilon_m + O(\epsilon_m^2)$. Again, the growth factor is the biggest problem.

Note that we have strong bounds on the residual, $r_G = b - Ax_g = Ex_G$. However, a small residual does not imply that x is close to the true solution, especially when A is ill-conditioned.

Computing A^{-1} is $O(n^3)$; the best way is to compute the LU factorization and then solve $Ax_i = e_i$, where e_i are the elementary vectors, which takes three times as long as the simple LU factorization. Even with the correctly rounded inverse, $A_I^{-1} = A^{-1} + F$ (with $\|F\| \leq \|A^{-1}\|\epsilon_m$), the residual is:

$$\begin{aligned} r_I &= b - A(A_I^{-1}b) = -AFb \\ \|r_I\| &\leq \|A\|\|F\|\|b\| \\ \frac{\|r_I\|}{\|A\|\|x\|} &\leq cond(A)\epsilon_m \end{aligned}$$

which now depends on the condition number.

Definition Suppose A is ill-conditioned. If $\|b\| \approx \|A\|\|x\|$, then we say that b *does not reflect the condition of A*. If $\|b\| \approx \frac{\|x\|}{\|A^{-1}\|}$, then we say that b *reflects the condition of A*.

Suppose we have $A = USV^T$. We may write $b = U\beta$, since the columns of U are an orthonormal basis. Then,

$$\begin{aligned} x &= A^{-1}b = VS^{-1}U^Tb \\ &= VX^{-1}U^TU\beta = VS^{-1}\beta \\ &= \sum v_i \frac{\beta_i}{s_i} \end{aligned}$$

Since V is orthonormal as well, $\|x\|^2 = \sum(\frac{\beta_i}{s_i})^2$. Thus, the solution of x is likely to be large if β_i is non-zero for small s_i and small if β_i is zero for small s_i. That is, b will not reflect the condition of A if it is close to a multiple of the first column of U (or any column with a relatively large singular value). b will reflect the condition of A if it is close to a multiple of the column of U corresponding to the a small singular value.

Similarly, to find x such that $\|b\|$ is small, one may write $x = V\gamma$ and then find that $Ax = \sum u_i s_i \gamma_i$.

3.3.1 Householder Triangularization

Definition For any non-zero vector, u, the corresponding *Householder transformation (elementary Hermitian matrix)* is $H(u) = I - \frac{2uu^T}{\|u\|_2^2}$, with *Householder vector*, u.

Householder transformations are orthogonal and depend only on the direction of u. Suppose $\|a\|_2 = \|b\|_2$, with $a \neq b$. Let u be any multiple of $b - a$. Then, $H(u)a = b$. The application of $H(u)$ to a vector, c, does not change the components corresponding to the 0 components of u. In particular, if u and c are orthogonal, then $H(u)c = c$.

Definition The QR factorization of a non-singular matrix, A, is $H_{n-1} ... H_1 A = R$, where each H_i is a Householder matrix and R is an upper triangular matrix.

Algorithm:QR factorization

- Let $u_1 = (a_{11} - r_{11}, a_{21}, ..., a_{n1})^T$, where $r_{11} = \|a_1\|_2$. Then, u_1 is a multiple of $\|a_1\|_2 e_1 - a_1$, and maps a_1 to a multiple of e_1; that is, $H_1 a_1 = r_{11} e_1$. This affects the first row of A, as well as transforming the first column.
- For the k^{th} step, let $u_k = (0, ..., 0, a_{kk} - r_{kk}, a_{k+1,k}, ..., a_{nk})^T$ and H_k the corresponding Householder transformation. Then, $H_k A^{(k-1)}$ has everything below A_{kk} cleared out.
- Let $Q = H_1 ... H_{n-1}$ (by orthogonality). Then, $A = QR$ is the *QR factorization.*

The QR factorization requires the storage of both R and the $n-k+1$ components of each u_k and of the β_k, which take more space than the LU decomposition. The calculation of the decomposition is $O(\frac{2}{3}n^3)$. Note that, since orthogonal transformations preserve the two-norm, $cond(R) = cond(A)$. For the computed R, there exists an orthogonal $\tilde{Q}$ such that $\tilde{Q}(A + \Delta_A) = R$, with $\|\Delta_A\|_F \leq \gamma(n)\|A\|_F \epsilon_m$, $\gamma(n)$ slow growing. Furthermore, Q is almost orthogonal, with $Q^T Q = I + \Delta_Q$ and $\|\Delta_Q\|_2 = O(\epsilon_m)$.

To solve a system of equations, $b = Ax = QRx$, using the QR factorization, we note that $Rx = Q^T b$ is an upper triangular system with can be solved in $O(\frac{1}{2}n^2)$ flops. Then, the computed solution satisfies $(A + \Delta)x = b + \delta b$, with $\|\Delta\|_F \leq (3n^2 + 41n)\|A\|_F \epsilon_m + O(\epsilon_m^2)$ and $\|\delta b\|_2 \leq (3n^2 + 40n)\|b\|_2 \epsilon_m + O(\epsilon_m^2)$.

3.3.2 Cholesky Factorization of a Positive Definite Matrix

Definition A matrix, A, is *symmetric* if $A = A^T$.

To preserve symmetry, any row interchanges must be accompanied by column interchanges; that is, we must transform A into PAP^T.

In some cases, we may write $PAP^T = LU = LDL^T$. However, this is not always possible (consider $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$).

Definition Suppose A is a symmetric matrix such that $x^T Ax > 0$ for all nonzero x. Then A is *positive definite.*

If A is symmetric and positive definite, then all its eigenvalues are positive and real. In addition, $\|A\|_2 = \lambda_{max} > 0$. In addition, all the diagonal elements are positive and the element of largest magnitude is on the diagonal.

Definition The *Cholesky factorization* of a symmetric and positive definite matrix, A, is $A = R^T R$, where R is upper triangular.

To calculate the Cholesky factorization, we write:

$$\begin{pmatrix} a_{11} & a_{12} & ... & a_{1n} \\ a_{21} & a_{22} & ... & a_{2n} \\ ... & ... & ... & ... \\ a_{n1} & a_{n2} & ... & a_{nn} \end{pmatrix} \begin{pmatrix} r_{11} & 0 & ... & 0 \\ r_{21} & r_{22} & ... & 0 \\ ... & ... & ... & ... \\ r_{n1} & r_{n2} & ... & r_{nn} \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & ... & r_{1n} \\ 0 & r_{22} & ... & r_{2n} \\ ... & ... & ... & ... \\ 0 & 0 & ... & r_{nn} \end{pmatrix}$$

Algorithm: Cholesky Decomposition

- $a_{11} = r_{11}^2$, so $r_{11} = \sqrt{a_{11}}$.
- $a_{12} = r_{11} r_{12}$, so that $r_{12} = a_{12}/r_{11}$.
- $r_{22}^2 = a_{22} - r_{12}^2$.
- And so on.

We may then solve the system $Ax = b$ using the two triangular systems from $R^T Rx = b$. This takes $\frac{1}{6}n^2$ steps instead of $\frac{1}{3}n^2$ using the LU decomposition. Furthermore, since $r_{1k}^2 + ... + r_{kk}^2 = a_{kk}$, $|r_{ik}| \geq \sqrt{a_{kk}}$ for all i, k, which gives a bound on the elements of R.

If A is not positive definite enough, then rounding errors may lead to negative values for r_{kk}^2.

In other cases, one may use Bunch-Paulette, which calculates $PAP^T = LBL^T$, where B is block diagonal with 2×2 blocks.

3.4 Solving Compatible Systems

Suppose $A = (a_1, a_2, ..., a_n)$ is an $m \times n$ matrix. Then, the range of A is of the form $Ax = a_1 x_1 + ... + a_n x_n$.

Definition A system of equations, $Ax = b$ is *compatible* if $b \in Range(A)$. Otherwise, the system is *incompatible*.

For general systems, we must consider both whether b is in the range of A (that is, whether a solution exists) and whether the solution is unique.

3.4.1 Pseudo-Inverses and Generalized Condition Numbers

Definition Let $A = U\Sigma V$ be the singular value decomposition of A. We define the *pseudo-inverse* by $A^+ = V^{-1}\Sigma^+ U^{-1}$, where Σ^+ is of size $n \times m$, the non-zero diagonal elements are inverted, and the zero diagonal elements are preserved.

Note that the pseudo-inverse agrees with the inverse when the inverse exists. The pseudo-inverse changes radically as the rank changes.

Definition If A is not invertible, then we define the generalized condition number by $cond(A) = \|A\|_2 \|A^+\|_2$.

3.4.2 Triangular Rank-Revealing Form

Definition Let $\hat{T} = \begin{bmatrix} T \\ 0 \end{bmatrix} = \begin{bmatrix} T_{11} & T_{12} \\ 0 & 0 \end{bmatrix}$, where T_{11} is upper triangular and non-zero (and both T_{12} and 0 may be empty). This is the *rank-revealing form.*

Only vectors of the form $b = \begin{bmatrix} b_1 \\ 0 \end{bmatrix}$ are compatible with $\hat{T}$.

To reduce a matrix, A, to triangular rank-revealing form, we may need to use complete pivoting, where each pivot is the largest element in the matrix and we apply both right and left permutations. Householder transformations may be used as well (in this case, we only need to do column pivoting).

3.4.3 Solution Method

Algorithm: Solving a Compatible System:

- Find an LU decomposition, $A = \tilde{P}^T \tilde{L} \tilde{U} P^T$, with $\tilde{L}$ unit lower triangular and $\tilde{U} = \begin{bmatrix} U_{11} & U_{12} \\ 0 & 0 \end{bmatrix}$ (U_{11} non-singular) in triangular rank revealing form, we have $\tilde{U} P^T x = \tilde{L}^{-1} \tilde{P} b$.
- Solve for $y = P^T x = \begin{bmatrix} y_r \\ y_{n-r} \end{bmatrix}$, by back substitution in the equation $\tilde{U} y = \begin{bmatrix} d_r \\ d_{m-r} \end{bmatrix} = \tilde{L}^{-1} \tilde{P} b$.
  - Case 1: If $d_{m-r} \neq 0$ (or is not "small") then the system is not compatible.
  - Case 2: If $d_{m-r} = 0$ (or is close enough), then we must solve $U_{11} y_r + U_{12} y_{n-r} = d_r$.
    - ∗ Suppose $b \neq 0$. Set $y_{n-r} = 0$. Then, solve $U_{11} y_r = d_r$ and set $y = \begin{bmatrix} U_{11}^{-1} d_r \\ 0 \end{bmatrix}$.
    - ∗ Suppose $b = 0 = d$ (and we want a non-zero y, since $y = 0$ is always a solution). In this case, we wish to solve $U_{11} y_r + U_{12} y_{n-r} = 0$.
      - · If $U_{12} = 0$, then we may choose any y_{n-r} and $y_r = 0$ is a solution.

· If $U_{12} \neq 0$, then we may choose a column, u_j which is nonzero. Then, $U_{12}e_j = u_j$. We may solve (through back substitution) $U_{11}y_r = -u_j$. Then, $y = \begin{bmatrix} -U_{11}^{-1}U_{12}e_j \\ e_j \end{bmatrix}$ is a non-zero solution.

- Finally, compute $x = Py$ to undo the permutations.

These techniques do not necessarily product the solution, x, of minimum length (which is only relevant when $b \neq 0$). The minimum length solution is found using the *triangular rank retaining form.*

3.4.4 Numerical Rank

Computation of the rank of a matrix can be problematic, since it can be hard to tell if a number is truly zero (and the rank is discrete). The smallest non-zero singular value gives the distance of the matrix from the nearest matrix of lower rank.

The choice of the when a singular value is "close enough" to zero affects the solutions:

- If we consider smaller values as nonzero, then we have larger rank estimates. This may lead to ill-conditioned and inaccurate matrices and larger solutions with smaller residuals.
- If we treat small numbers as 0, then we have lower rank estimates. This leads to smaller solutions and bugger residuals.

3.5 Least Squares Solutions to Overdetermined Linear Systems

Suppose we have the system $Ax \cong b$, where the system is overdetermined. That is, there are n equations in m unknowns with $m < n$. Since we cannot solve the system exactly, we wish to solve $x = \arg\min(\|Ax - b\|_2) = \arg\min(\sum_{i=1}^{n}((Ax - b)_i)^2$. We call $Ax - b$ the *residual.* This is equivalent to finding the closest point to b in the subspace spanned by A, which is the orthogonal projection of b onto $Range(A)$. Then, we may write $b = b_R + b_N$, with $b_R \in Range(A)$ and $b_N \in Null(A^T)$; this decomposition is unique. This gives us the normal equations:

$$A^T b = A^T A x$$

Assuming that A has a rank of m, this is a non-singular system of equations, with $A^T A$ symmetric and positive definite, so that we may use the Cholesky decomposition.

Algorithm: Least Squares Solution with Cholesky

- Compute $A^T b$ (in nm steps).
- Compute $A^T A$ (in nm^2 steps).

- Compute the Cholesky factorization, $A^TA = R^TR$ (in $O(\frac{1}{6}m^3)$ steps).
- Solve the systems $R^Ty = A^Tb$ and then $Rx = y$ (in $O(m^2)$ steps).

The most expensive step of this algorithm is computing A^TA, since $n > m$. Furthermore, $cond(A^T) = cond(A)^2$, which can lead to decreased accuracy.

Alternatively, we may compute a QR-factorization of A by applying Householder transformations to A, to find $Q_m^T...Q_1^TA = R$ or $A = QR$, where $R = \begin{bmatrix} \hat{R} \\ 0 \end{bmatrix}$ with $\hat{R}$ an $m \times m$ upper triangular matrix. Then,

$$\begin{aligned} \min \|Ax - b\|_2 &= \min \|Q^T(QRx - b)\|_2 \\ &= \min \| \begin{bmatrix} \hat{R} \\ 0 \end{bmatrix} x - \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \|_2 \end{aligned}$$

Then, $\hat{R}x = w_1$ is a triangular system which we may solve for x to set those terms to 0. $0x$ is constant, so the residual will have length $\|w_2\|$.

Algorithm: Least Squares Solution with QR-Factorization

- Compute the QR factorization of A (in $O(m^2n)$ steps).
- Solve $\hat{R}x = w_1$ (in $O(m^2)$ steps).

This is the same number of steps as the previous algorithm but tends to be more accurate.

If A is rank deficient or nearly rank deficient, then we may use the singular value decomposition of A, $A = U\Sigma V^T$, where Σ is non-square with zeroes below the diagonal containing the singular values. Then,

$$\begin{aligned} \min \|Ax - b\| &= \min \|\Sigma V^Tx - U^Tb\| \\ &= \min \| \begin{bmatrix} \hat{\Sigma}y \\ 0 \end{bmatrix} - \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \|_2 \end{aligned}$$

where $y = V^Tx$ and $w = U^Tb$. As before, we may solve $\hat{\Sigma}y = w_1$ (which is very easy since $\hat{\Sigma}$ is diagonal) and then solve $x = V^Ty = V\hat{\Sigma}^{-1}U_1^Tb$. If $\hat{\Sigma}$ has very small entries, we may set them to 0 to avoid the instability from a nearly singular matrix. However, computing the SVD is very costly.

In the least squares problems, perturbations to b that are in $Range(A)$ change x (proportionally to $cond(A)$) but do not change the residual. Perturbations to b that are in $Null(A)$ affect only the residual. (Most perturbations are a sum of the two and do affect both x and the residuals.) Perturbations to A that change the null space cause the biggest problems.

4 Approximating Functions

Let a function, $f(x)$, be given. We want to create $\tilde{f}(x)$ which approximates $f(x)$. We assume that f is continuous (and possibly differentiable).

Definition Let f be a function on $[a,b]$. We define the *Euclidean norm* as $\|f\|_2 = (\int_a^b |f(x)|^2 dx)^{1/2}$. We define the *maximum (Chebyshev) norm* as $\|f\|_\infty = \max_{[a,b]} |f(x)|$.

(These norms are usually not possible to evaluate numerically, since they depend on an infinite number of points.) If we are given a set of $n+1$ points, $x_0, ..., x_n$ and $f(x_0), ..., f(x_n)$, we may use a vector norm on $(f(x_0) - \tilde{f}(x_0), ..., f(x_n) - \tilde{f}(x_n))^T$ as well.

Suppose we have a polynomial $p_n(x) = a_0 + a_1 x + ... + a_n x^n$. To evaluate this with *Horner's Method*, set $b_n = a_n$ and $b_k = x b_{k+1} + a_k$. Then, $p_n(x) = b_0$. This is faster and has better error bounds than the obvious method.

Theorem 4.1 Fundamental Theorem of Algebra. *Let $p_n(x) = a_0 + a_1 x + ... + a_n x^n$, with $a_n \neq 0$. Then $p_n(x)$ has exactly n roots (some of which may be complex or multiple).*

Theorem 4.2 Weierstrass Approximation Theorem. *Let f be a continuous, real-valued function on $[a,b]$. For all $\epsilon > 0$, there exists $n(\epsilon) \in Z$ and a polynomial $p_{n(\epsilon)}$ of degree at most $n(\epsilon)$ such that $|f(x) - p_{n(\epsilon)}(x)| < \epsilon$ for all $x \in [a,b]$.*

4.1 Interpolating Polynomials

Definition Let $x_0, ..., x_n$ and $f(x_0), ..., f(x_n)$ be given. The polynomial, $p_n(x)$ such that $p_n(x_i) = f(x_i)$ for all i is called the *interpolating polynomial.*

Theorem 4.3 *The interpolating polynomial of degree n is always unique.*

Proof Suppose $p_n(x_i) = f(x_i) = q_n(x_i)$ for $i = 0, ..., n$. Let $d_n(x) = p_n(x) - q_n(x)$. Then, $d_n(x_0) = ... = d_n(x_n) = 0$ and d_n has $n+1$ zeroes. However, it is a polynomial of degree at most n. Thus, d_n must be the zero polynomial and $p_n = q_n$.

To compute the interpolating polynomial, we have the system of equations:

$$\begin{bmatrix} f(x_0) \\ ... \\ f(x_n) \end{bmatrix} \begin{bmatrix} 1 & x_0 & ... & x_0^n \\ ... & ... & ... & ... \\ 1 & x_n & ... & x_n^n \end{bmatrix} \begin{bmatrix} a_0 \\ ... \\ a_n \end{bmatrix}$$

(where the matrix of powers of $x_0, ..., x_n$ is called the *Vandermonde matrix*, V). Whenever the x_i are distinct, this system of equations has a unique solution. Note that V can be ill-conditioned. This will make the a_i less accurate, but the residual will be small as long as there is not much growth.

Definition Given $x_0, ..., x_n$, define:

$$\phi_j(x) = \frac{(x - x_0)...(x - x_{j-1})(x - x_{j+1})...(x - x_n)}{(x_j - x_0)...(x_j - x_{j-1})(x_j - x_{j+1})...(x_j - x_n)}$$

Then, $\phi_j(x_i) = 1(i = j)$. Then, we may define the interpolating polynomial as $p_n(x) = \sum_{j=0}^n f(x_j)\phi_j(x)$. This is the *Lagrangian Form of the Interpolating Polynomial.*

Note that this form is not computationally convenient, especially for adding additional data points.

Algorithm: Newton Interpolation

- Let $Q_0(x) = f(x_0)$.
- Define $Q_k(x) = Q_{k-1}(x) + q_k(x)$, with
  - $q_k(x_i) = 0$ for $i < k$, since we must have $Q_k(x_i) = f(x_i) = Q_{k-1}(x_i)$. This means that $q_k(x) = a_k(x - x_0)...(x - x_{k-1})$.
  - Since $f(x_k) = Q(x_k) = q_k(x_k) + Q_{k-1}(x_k)$, we may solve to find $a_k = \frac{f(x_k) - Q_{k-1}(x_k)}{(x_k - x_0)...(x_k - x_{k-1})}$.

  This gives us $Q_n(x) = a_0 + a_1(x - x_0) + ... + a_n(x - x_0)...(x - x_n)$, where the $a_i = f[x_0, ..., x_i]$ are called the *divided differences.*

Note that we may also solve for all the a_i at once using the triangular system of equations:

$$\begin{bmatrix} 1 & 0 & 0 & ... & 0 \\ 1 & (x_1 - x_0) & 0 & ... & 0 \\ 1 & (x_2 - x_0) & (x_2 - x_0)(x_2 - x_1) & ... & 0 \\ ... & ... & ... & ... & ... \\ 1 & (x_n - x_0) & (x_n - x_0)(x_n - x_1) & ... & (x_n - x_0)...(x_n - x_{n-1}) \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ ... \\ a_n \end{bmatrix} = \begin{bmatrix} f(x_0) \\ f(x_1) \\ ... \\ f(x_n) \end{bmatrix}$$

This is faster and usually better conditioned.

Proposition 4.4 *Suppose f has k continuous derivatives on $[\min(x_0, ..., x_{k-1}), \max(x_0, ..., x_{k-1})]$. Then, the divided difference obeys $f[x_0, ..., x_{k-1}] = f^{(k)}(\xi)/k!$ for some $\xi \in [\min(x_0, ..., x_{k-1}), \max(x_0, ..., x_{k-1})]$.*

Theorem 4.5 *Given $n + 1$ distinct points, $x_0, ..., x_n$, a continuous f with at least $n+1$ continuous derivatives and the interpolating polynomial, $p_n(x)$, for all x, there exists $\xi \in [\min(x_i), max(x_i)]$ such that $|f(x) - p_n(x)| = |f^{(n+1)}(\xi)\frac{(x-x_0)...(x-x_n)}{(n+1)!}|$*

Note that if $f^{(n+1)}$ is bounded, we may bound the error above.

Suppose we have equally spaced points, with $x_k = x_0 + kh$ and an interpolating polynomial, Q_n, of a function, f, based on them. Then, for any $x = x_0 + th$,

$$f(x) - Q_n(x) = h^{n+1}t(t-1)...(t-n)\frac{1}{(n+1)!}f^{(n+1)}(\xi) = h^{n+1}\pi_m(t)\frac{1}{(n+1)!}f^{(n+1)}(\xi)$$

where $\pi_n(t) = t(t-1)...(t-n)$ is the *error factor polynomial.* Since $\pi_n(t)$ is smallest in the middle of the interval and goes to infinity outside the interval, we see that interpolation in the middle of the interval is most reliable, and extrapolation can be very problematic.

Having more equally spaced points with which to interpolate is not always good. For example, with *Runge's function,* $f(x) = \frac{1}{1+x^2}$ on $[-5, 5]$, there are points (near the endpoints) in which the error goes to infinity as more points are added. However, the interpolation does well in the middle of the interval.

4.2 Minimax Approximating Functions

Definition Given a function, f, the *best approximation* or the *minimax approximation* is a function, $\tilde{f}$ that minimizes the Chebyshev norm, $\max_{x\in[a,b]} |f(x) - \tilde{f}(x)|$.

Theorem 4.6 *Let $f(x)$ be continuous on $[a,b]$. For any n, there exists a polynomial, p_n^*, of degree n that minimizes* $\max |f(x) - p_n^*(x)|$.

Theorem 4.7 Chebyshev. *A polynomial of degree at most n is a best approximation if and only if $f(x) - p_n(x)$ assumes the maximum value with alternating signs at least $n+2$ times in $[a,b]$. (This is called the* equal ripple quality*.) The best polynomial is unique.*

Suppose $f(x)$ has $n+1$ continuous derivatives. Then, the interpolating polynomial satisfies:

$$f(x) - p_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}(x - z_0)...(x - z_n)$$

for some ξ. In particular, if $f(x)$ is a polynomial of degree $n+1$, then $f^{(n+1)}(x) = (n+1)!a_{n+1}$, and the error is $a_{n+1}(x-z_0)...(x-z_n)$. Thus, well chosen z_i will lead to an interpolating polynomial that minimizes the function and gives the best approximation. Suppose $f(x) = x^{n+1}$ and we wish to approximate it on $[-1,1]$. The error polynomial must attain its maximum $n+2$ times. Let $x = \cos\theta$. Let $T_{n+1}(x) = \cos((n+1)\theta)$. Then, $T_{n+1}(x)$ has absolute maxima with alternating signs at $\theta = \frac{j\pi}{n+1}$, $j = 0, ..., n+1$, which is $n+2$ points. Furthermore, this is a polynomial in x (using the trigonometric addition rules). Thus, the interpolation points of the best approximation are the zeros of $T_{n+1}(x)$, which are $\cos(\frac{j\pi+\pi/2}{n+1}$. These are called the *Chebyshev points*. Note that these points are not equally spaced; they tend to crowd near the ends of the interval.

4.3 Interpolating Splines

Definition Let $[a,b]$ be an interval with $n+1$ distinct *knots* (points at which the function is known), with $a = x_0 < x_1 < ... < x_{n-1} < x_n = b$. A *spline function* (or *piecewise polynomial*) of degree p with knots $x_0, ..., x_n$ is a function, $s(x)$, such that:

- On each subinterval, $[x_i, x_{i+1}]$, $s(x)$ is a polynomial of degree p.
- $s(x)$ and its first $p-1$ derivatives are continuous on $[a,b]$.

$s(x)$ is an *interpolating spline* if $s(x_i) = f(x_i)$ for all i.

To calculate a cubic spline ($p = 3$), we consider each subinterval, $[x_{i-1}, x_i]$. Let $h_i = x_i - x_{i-1}$ and $u = x - x_{i-1}$ for all $x \in [x_{i-1}, x_i]$. On this interval, $s(x) = s_i(x) = \tilde{s}_i(u) = a_i + b_i u + c_i u^2 + d_i u^3$. (This is $4n$ unknowns.) Then,

- To match the left endpoint, define $a_i = s_i(x_{i-1}) = f(x_{i-1})$.
- By continuity at the interior knots and interpolation at the rightmost knot, $s_i(x_i) = f(x_i)$ for $i = 1, ..., n$. This means that $f(x_i) = s_i(x_i) = a_i + b_i h_i + c_i h_i^2 + d_i h_i^3$ (which is n equations).
- By the continuity of the first derivative at the interior knots, $s_i'(x_i) = s_{i+1}'(x_i)$, and $b_{i+1} = b_i + 2c_i h_i + 3d_i h_i^2$, which gives another $n-1$ linear equations.
- By the continuity of the second derivative at interior points, $s_i''(x_i) = s_{i+1}(x_i)$, and $2c_{i+1} = 2c_i + 6d_i h_i$, which is another $n-1$ linear equations.

This gives us $4n-2$ linear equations in $4n$ unknowns. This allows us to choose two extra conditions. For the *natural spline*, we assume a straight line outside the interval, so that $s''(x_0) = 2c_0 = 0$ and $s''(x_n) = 2c_n + 6d_n h_n = 0$. This gives a system of linear equations.

Splines can be harder to compute with that high-degree polynomials, because a search must be done to find the correct interval for any point. Furthermore, more storage is needed for $x_0, ..., x_n, a_1, ..., a_n, b_1, ..., b_n, c_1, ...c_n, d_1, ..., d_n$.

In practice, *B-splines*, which can be computed recursively and are more stable, are used. The associated linear systems are generally sparse and well-conditioned, and recursion makes it easier to add knots in places where where are large errors, since more knots leads to increased accuracy.

5 Numerical Quadrature

Problem: Evaluating $\int_a^b f(x)dx$.

Theorem 5.1 *Let $a = x_1 < x_2 < ... < x_{n+1} = b$. Let $R_n = \sum_{i=1}^n (x_{i+1} - x_i) f(\xi_i)$ where $\xi_i \in [x_i, x_{i+1}$. The, under mild conditions, if $|x_{i+1} - x_i| \to 0$ then for any $\xi_i \in [x_i, x_{i+1}]$, $\lim_{n\to\infty} R_n = \int_a^b f(x)dx$.*

There is no quadrature method based on sampling f at a finite number of points that can be correct for a general continuous f.

Error in Integration: Suppose $f(x)$ is bounded and continuous on $[a, b]$. Define $\|f\|_\infty = \max_{x\in[a,b]} |f(x)|$. Let $\tilde{f}(x) = f(x) + \delta(x)$ be a perturbed version. Then,

$$|\int_a^b \tilde{f}(x)dx - \int_a^b f(x)dx| \le \int_a^b |\tilde{f}(x) - f(x)|dx \le (b-a) \max_{x\in[a,b]} |\tilde{f}(x) - f(x)| = (b-a)\|\delta(x)\|_\infty$$

is a bound on the absolute error. (Note that a small absolute error may lead to a large relative error if $\int_a^b f(x)dx \approx 0$.)

Definition Let $a \le x_1 < x_2 < ... < x_n \le b$ (where the x_i are called the *nodes*). An *n-point quadrature rule* is a rule of the form: $Q_n(f) = \sum_{i=1}^n w_i f(x_i)$. If $x_1 = a$ and $x_n = b$, then this is a *closed formula.* If $x_1 > a$ and $x_n < b$, this is an *open formula.*

We find formulas that work exactly on polynomials up to some degree, using interpolating polynomials. Given f at the n points, $x_1, ..., x_n$, we may find the Lagrangian form of the interpolating polynomial, $p_{n-1}(x) = \sum_{i=1}^{n} f(x_j)\phi_j(x)$. With a fixed $x_1, ..., x_n$, ϕ_j and its integral do not depend on f at all. This means, for any f, we have:

$$\int_a^b f(x)dx \approx \int_a^b p_{n-1}(x)dx = \sum_{i=1}^{n} f(x_j) \int_a^b \phi_j(x)dx$$

This method interpolates f exactly if f is a polynomial of degree at most $n-1$.

Definition The *degree* or *order* of an interpolatory quadrature form is one more than the highest degree polynomial that it integrates exactly.

Suppose $p_{n-1}(x)$ is the interpolating polynomial for $f(x)$. Then, we have the error:

$$\int_a^b f(x)dx - \int_a^b p_{n-1}(x)dx = \frac{1}{n!}\int_a^b f'(\xi)(x-x_1)...(x-x_n)dx$$

which can be used to calculate an error bound, using the bounds on the derivative and the polynomial. In particular, if we have equally spaced points, $x_1, x_1+h, ..., x_1+(n-1)h$ and we write $x = x_1 + th$, then

$$\begin{aligned} f(x) - p_{n-1}(x) &= t(t-1)...(t-(n-1))\frac{f^{(n)}(\xi)}{n!} \\ \int_a^b |f(x) - p_{n-1}(x)|dx &\leq \frac{b-a}{4n}h^n \max|f^{(n)}(x)| \end{aligned}$$

This may be reduced to 0 by increasing n or decreasing h.

Definition Interpolatory quadrature forms using low-degree polynomials and equally-spaced points are called the *Newton-Cotes Formulae.*

- *Midpoint Rule*: We interpolate the function by a constant based at the midpoint of the interval, so that $f(x) \approx f(m)$ and $\int_a^b f(x)dx \approx (b-a)f(\frac{a+b}{2})$. This is an open Newton-Cotes formula of degree 2 (it integrates a line exactly). Using a Taylor expansion, we find the error as:

$$\begin{aligned} f(x) &= f(m) + (x-m)f'(m) + \frac{1}{2}f''(\xi_{m,x})(x-m)^2 \\ \int_a^b f(x)dx &= \int_a^b f(m)dx + \int_a^b f'(m)dx + \frac{1}{2}\int_a^b f''(\xi_{m,x})(x-m)^2dx \\ &= f(m)(b-a) + 0 + \frac{1}{2}\int_a^b f''(\xi_{m,x})(x-m)^2dx \end{aligned}$$

which gives an error bound of $\frac{(b-a)^3}{24}\max|f''(\xi)|$.

- *Trapezoid Rule*: We interpolate the function by a straight line from $f(a)$ to $f(b)$. Then, $\int_a^b f(x)dx \approx \frac{b-a}{2}(f(a)+f(b))$. This is a closed Newton-Cotes formula of order 2. The error is approximately $-f''(\xi)\frac{(b-a)^3}{12}$.
- *Simpson's Rule*: Use $x_1 = a$, $x_2 = \frac{a+b}{2}$, and $x_3 = \frac{b}{2}$ as the points for in interpolating quadratic. Then, $\int_a^b f(x)dx \approx \frac{b-a}{6}(f(a) + 4f(m) + f(b))$. The error is approximately $-\frac{(b-a)^5}{90}f^{(4)}(\xi)$. Note that Simpson's Rule is the sum of two-thirds of the midpoint rule and one-third of the trapezoidal rule (which makes their highest order errors cancel as well).

For any $c \in [a,b]$, $\int_a^c f(x)dx + \int_c^b f(x)dx = \int_a^b f(x)dx$. For larger intervals, it usually preferable to split it into smaller intervals and use one of the formulas above on each subinterval instead of using polynomials of higher degree. Using low order interpolatory quadrature rules on sub-intervals is called *composite integration formulas*. One method is *Adaptive Quadrature*, in which intervals that seem to have more error (based on the error estimates of the midpoint and trapezoidal rules, for example) are subdivided more. This can always fail, though, and there are tradeoffs between accuracy and speed.

If data is tabular (so that one cannot choose the points), one must use the trapezoidal rule. Alternatively, one may fit splines and integrate those.

6 Ordinary Differential Equations

Problem: Find a function, $y(t)$, such that $y'(t) = f(t, y(t))$ and $y(0) = y_0$ (because of the second condition, this is called an *initial value problem*).

Theorem 6.1 *If $f(t,y)$ is continuous in $0 \le t \le b$ and there is a constant, L, such that, for all $0 \le t \le b$ and y, z, $|f(t,y) - f(t,z)| \le L|y-z|$ (this is the* Lipschitz condition*), then there is a unique, continuous, and differentiable function, y, such that $y' = f(t, y(t))$ and $y(0) = y_0$.*

Proposition 6.2 *Let $h \le 0$, j an integer, $jh \le b$, and $hL \ge 0$. Then,*

$$1 + hL \le e^{hL}$$
$$(1+hL)^j \le e^{jhL} \le e^{Lb}$$

We discretize the problem by computing $y(t_1), ..., y(t_n)$ for given points. For now, we assume that the times are equally spaced, so that $t_{j+1} = t_j + h = (j+1)h$.

Definition An ODE is *well-posed* if small perurbations in the problem lead to small changes in the exact solution.

Definition A numerical method is *stable* if small changes in the initial values produce bounded changes in the approximate solutions.

(The stability of the method may depend on the problem.)

Definition The *test equation* is given by $y' = \lambda y$. Note that the exact solution is $y = y_0 e^{\lambda t}$. Note that if $\lambda = a + ib$, then $e^{\lambda t} = e^{at}(\cos(bt) + i\sin(bt))$, so that we have exponential growth in the modulus if $a > 0$ and exponential decay in the modulus if $a < 0$.

Definition A numerical method is *order p* if the local error is $O(h^{p+1})$ (since we must then add up all of the errors over all the intervals).

Definition A *one-step method* depends only on the last iterate, y_j. A *multi-step method* may depend on previous iterates as well.

Definition An *explicit method* defines y_{j+1} directly. An *implicit method* includes y_{j+1} on both sides of a (usually non-linear) equation.

Forward (Explicit) Euler Method: By a Taylor series, we have:

$$y(t_{j+1}) = y(t_j) + hf(t_j, y_j) + \frac{h^2}{2} f''(\xi)$$

We use the approximation, $y_{j+1} = y_j + hf(t_j, y_j)$. The *local truncation error* is the error in a single step. In this case, the local truncation error is $O(h^2)$. On the test equation, we have $y_j = (1 + h\lambda)^j y_0$. Note that this is stable when $|1 + h\lambda| \leq 1$; even if $\lambda < 0$ (so the true solution is stable), we must have $h \leq -2/\lambda$ for the computed solution to be stable.

Explicit Runge-Kutta Methods: In these methods, we try to match more terms of the Taylor series by applying the chain rule:

$$y''(t) = \frac{\partial f}{\partial t} + \frac{\partial f}{\partial y} y' = \frac{\partial f}{\partial t} + \frac{\partial f}{\partial y} f(t, y)$$

For example, *Heun's method* uses:

$$y_{j+1} = y_j + \frac{h}{2}(f(t_j, y_j) + f(t_{j+1}, y_j + hf(t_j, y_j)))$$

There is a family of methods with different parameters to match higher orders. the *Classical Runge-Kutta Method* is:

$$\begin{aligned}
K_0 &= f(t_j, y_j) \\
K_1 &= f(t_j + \frac{h}{2}, y_j + h\frac{K_0}{2}) \\
K_2 &= f(t_j + \frac{h}{2}, y_j + h\frac{K_1}{2}) \\
K_3 &= f(t_j + h, y_j + hK_2) \\
y_{j+1} &= y_j + \frac{h}{6}(K_0 + 2K_1 + 2K_2 + K_3)
\end{aligned}$$

This is a fourth order method, but it requires more evaluations of f. Runge-Kutta methods with different orders may also be combined to optimize the stepsize (one common method is RK45).

Explicit Linear Multistep (Multivalue) Methods (LMM): In these methods, we compute:

$$y_{j+1} = \sum_{i=1}^{n} \alpha_i y_{j+1-i} + h \sum_{i=1}^{m} \beta_i f(t_{j+1-i}, y_{j+1-i})$$

These methods are also called *predictors*. These methods do polynomial interpolation using both the function values and the derivatives to get to the next point. A two-step predictor of order two is:

$$y_{j+1} = y_j + \frac{h}{2}(3f(t_j, y_j) - f(t_{j-1}, y_{j-1}))$$

The *Adams-Bashforth Method* is:

$$y_{j+1} = y_j + \frac{h}{24}(55f(y_j, t_j) - 59f(t_{j-1}, y_{j-1}) + 37f(t_{j-2}, y_{j-2}) - 9f(t_{j-3}, y_{j-3}))$$

which is of fourth order. Note that alternative methods must be used for the first few steps, since we do not know y_{-1} or previous values.

Implicit (Backward) Euler: Now, we compute y_{j+1} from the equation $y_{j+1} = y_j + hf(t_{j+1}, y_{j+1})$, which is (possibly) a non-linear equation in y_{j+1}. On the test equation, we find that $y_j = y_0(1 - h\lambda)^{-j}$; this is stable for all $h > 0$ if $\lambda < 0$. In general, as well, implicit Euler is stable for a larger set of h values than explicit Euler.

Implicit Runge-Kutta Methods: Let $K_1 = f(t_j, y_j)$ and $K_2 = f(t_{j+1}, y_{j+1})$. Then, $y_{j+1} = y_j + \frac{h}{2}(K_1 + K_2)$, which is akin to the trapezoid rule. (And they get more complicated.)

Predictor-Corrector Methods (Implicit LMM): Estimate $\tilde{y}_{j+1}$ using the predictor method, and use it as the initial value in a non-linear equation solved with Newton's method. For example, the *Adams-Moulton Method* is:

$$y_{j+1} = y_j + \frac{h}{24}(9f(t_{j+1}, y_{j+1}) + 19f(y_j, t_j) - 5f(t_{j-1}, y_{j-1}) + f(t_{j-2}, y_{j-2}))$$

In higher dimensions, we have $y(t)$ equal to a vector of functions of y and t. We may also convert higher order ODE's into systems of first order ODE's.

Definition A differential equation is *stiff* when the solution varies slowly but there are nearby solutions that vary rapidly, so that the numerical method must take small steps. Alteratively, equations are stiff when components have different time scales.

This may require implicit methods (so that there are more options for h) and may be slower. *Stiff methods* are implicit methods with larger stability regions for h.

7 Linear Programming

Definition In *linear programming*, we minimize (or maximize) a linear *objective function*, $l(x) = c^T x = \sum_i c_i x_i$, subject to linear *constraints*, which may be either *equality constraints*, $Ax = b$, or *inequality constraints*, $Ax \leq b$. c is called the *normal vector.* x is *feasible* if it satisfies the constraints.

Notice that for any vector, p, and positive scalar, α, we have:

$$l(x + \alpha p) = c^T(x + \alpha p) = c^T x + \alpha c^T p$$

- If $c^T p = 0$, then the value of the function does not change as x moves along p.
- If $c^T p > 0$, the function increases as we move in the direction of p.
- If $c^T p < 0$, then the function decreases in the direction of p.

Unless $c = 0$, the function is unbounded and there are no (finite) absolute minima or maxima.

7.1 Equality Constraints

Suppose we have an equality constraint, $Ax = b$, and a feasible x_0 (which requires that the linear system $Ax = b$ is compatible; that is, the rows of A must be linearly independent). If we want to find a p such that $x_1 = x_0 + \alpha p$, then we must have $\alpha Ap = 0$; that is, the only vectors, p, that will allow a move to another feasible point lie in $Null(A)$.

The conditions for a point, x^*, to be optimal are that:

- x^* is feasible; that is $Ax^* = b$.
- There is no direction that stays feasible and decreases $c^T x$. That is, there is no p such that $c^T p < 0$ and $Ap = 0$.

Theorem 7.1 *If $y \in Range(A^T)$ and $Ap = 0$ then $y^T p = 0$. If $y \notin Range(A^T)$ then there exists p such that $Ap = 0$ and $Y^T p < 0$.*

This leads us to the following possible optimal solutions:

- Case 1: Suppose $c = A^T\lambda$ for some λ (where the λ are called the Lagrange multipliers). Then, there is not feasible descent direction from a feasible solution, x^*, and the optimal solution is:

  $$c^T x^* = \lambda^T A x^* = \lambda^T b$$

  Note that all feasible points lead to the same optimal solution.
- Case 2: If $c \neq A^T\lambda$ for all λ, then there exists p such that $Ap = 0$ and $c^T p < 0$. Then, $c^T(x + \alpha p) = c^T x + \alpha c^T p$ and there is no finite optimal point or optimal value (since we may just descend further in the direction of p).

7.2 Inequality Constraints

Suppose we have the inequality constraints, $Ax \geq b$. Let a be a row of A and β the corresponding element of b. If $a^T x \geq \beta$, then we say that x is *feasible with respect to this constraint.* If $a^T x > \beta$, the x is *strictly feasible.* If $a^T = \beta$, then the constraint is *active (binding, tight).* If $a^T x < \beta$, the constraint is *violated.*

Let x_0 be a feasible point. Let A_A be the rows of A that are active. Then, $A_A x_0 = b_A$. We want to find a *feasible direction*, p, such that $p \neq 0$ and there exists some $\gamma > 0$ such that $a^T(x_0 + \alpha p) \geq \beta$ for all $0 \leq \alpha \leq \gamma$ for all constraints. For each constraint, if $a^T p \geq 0$,then movement in that direction will always satisfy the constraint. If $a^T p < 0$, then there is a finite γ that determines when $a^T(x + \gamma p) = \beta$; that is, $\gamma = \frac{a^T x - \beta}{-a^T p}$. $\gamma > 0$ for each strictly feasible constraint, but $\gamma = 0$ for each active constraint. Thus, we must have $A_A p \geq 0$ in order for p to be a feasible direction.

Definition A *vertex*, x, is a point where the matrix of active constraints, $A_A(x)$, contains at least one subset of n linearly independent rows; that is, $rank(A_A(x)) = n$. A vertex is *nondegenerate* if there are exactly n linearly independent active constraints. Otherwise, the vertex is *degenerate.* Two vertices are *adjacent* if the matrices of the active constraints differ by one (at least in the case of non-degenerate matrices; it is more complicated otherwise).

Theorem 7.2 *Suppose we have a constraint $Ax \geq b$ with A of size $m \times n$ and at least one feasible point. A vertex exists if $rank(A) = n$.*

Theorem 7.3 Farkas's Lemma. $c^T p \geq 0$ *for all p such that* $Yp \geq 0$ *if and only if $c = Y^T \lambda$ and $\lambda \geq 0$ (that is, all the coefficients of the linear combination equalling c are positive). Equivalently, there exists p such that $c^T p < 0$ and $Yp \geq 0$ on if c cannot be written as $c = Y^T \lambda$ with all elements of λ nonnegative.*

Proof *(Only if, special case.)* For simplicity, assume that Y has linearly independent rows. Suppose that $c = Y^T \lambda$ and at least one component of λ is negative. (By linear independence, λ is unique.) Choose s to be one of the negative components. By the independence of the rows of Y, the system $Yp = e_s$ must be compatible. Then,

$$c^T p = \lambda^T Y p = \lambda^T e_s = \lambda_s < 0$$

Thus, this p stays on all but the s^{th} constraint; it moves in a strictly feasible direction for that constraint.

Theorem 7.4 *Suppose that $rank(A) = n$, that there exists a minimizer, and that the optimal objective value is finite. Then, there is a vertex minimizer.*

Proof Suppose that x_0 is a minimizer and not a vertex. Then, $Ax_0 \geq b$ and $c = A_{A0}^T \lambda_{A0}$ which $\lambda_{A0} \geq 0$. Choose p such that $A_{A0} p = 0$ and moving along p will hit an inactive constraint (this is possible since A_{A0} has fewer than n

linearly independent constraints and $rank(A) = n$). In this direction, $c^T p = \lambda_{A0}^T A_{A0} p = 0$ and the value of the function is constant (and optimal). This process may be repeated until the matrix of active constraints has rank n.

Definition Suppose we have two non-negative vectors. We say *complementarity* holds if it is always that case that whenever one element of a vector is non-zero, the corresponding elements of the other vector is zero. (That is, the dot product of the two non-negative vectors must be 0.)

If there is a solution, then we may always write $c = A^T \lambda$ with non-negative λ, since we may write $c = A_A^T \lambda$ and then set the $\lambda = 0$ for all the other constraints. Define $r = Ax - b$. At the solution, if $r_i > 0$ then we are not an an active constraint, and we must have $\lambda_i = 0$. If $\lambda_i > 0$, we are on an active constraint, and we must have $r_i = 0$. This gives the following optimality conditions:

$$\begin{aligned} c &= A^T \lambda \\ r_i \lambda_i &= 0 \\ \lambda &\geq 0 \\ r &\geq 0 \end{aligned}$$

Algorithm: Simplex Method (Nondegenerate Case)

- Suppose we have a vertex, x_0 (which can be found from "Phase One" below).
- For each k, let A_k be the matrix of active constraints at x_k. Note that, by non-degeneracy, A_k is an $n \times n$ non-singular matrix.
- Solve $A_k^T \lambda_k = c$ for the *Lagrange multipliers/dual variables*, λ_k. If $\lambda_k \geq 0$, then we have found an optimal point.
- Otherwise, choose s such that $\lambda_s < 0$. Solve $A_k p = e_s$ for p, which is a feasible direction.
- Move in the direction p until another constraint becomes active.

Because the value of $c^T x_k$ must decrease at every step and there are a finite (though large) number of vertices, this algorithm must terminate. However, the numbers of steps that will be required is unknown, and there is no way to measure the distance from the optimum or the rate of convergence.

For cases where there is more than one $\lambda_s < 0$, then we must specify a *constraint deletion rule.* The *Textbook Rule* simply chooses the most negative λ_s. (This is not necessarily the fastest because scales can vary, but it is reasonable.)

Note that A_k and A_{k-1} differ by exactly one row. Since this is a rank one change, there are algorithms for updating the LU decomposition instead of computing it from scratch, which can be done in $O(N^2)$ time. (After many steps, it should be recomputed from scratch, since numerical errors may have built up.)

In the case of degeneracy, more that n constraints might be active at any given vertex, so A_k will not be nonsingular or square. To modify the algorithm, one keeps a *working set* of n linearly independent constraints and uses them (the rest of the constraints are called the *idle constraints*). However, it is possible that the chosen feasible direction based on the working constraints will violate some of the idle constraints. In this case, one might drop the constraint on which one is moving and replace it with the violated constraint. This is not guaranteed to work; it may lead to *cycling.*

The worst case is the case in which there are 2^n vertices and all of them are visited (Klee and Minty showed that this is possible). Thus, this algorithm is not polynomial time, though it usually works much better than the worst case. Other methods, like *interior methods* (which don't require moving from one vertex to the next), are polynomial time.

To find an initial vertex, we use another linear programming algorithm (with a known initial vertex).

Algorithm: Phase 1

- Given the original variables, x, and original m constraints, $Ax \geq b$. Define
  - y a set of *artificial variables*
  - $A^* = \begin{pmatrix} A & I_m \\ 0 & I_m \end{pmatrix}$
  - $x^* = \begin{pmatrix} x \\ y \end{pmatrix}$
  - $b^* = \begin{pmatrix} b \\ 0 \end{pmatrix}$
- Choose an arbitrary point, x_0, and let $r_i = \min(0, a_i^T x_0 - b_i)$ for each of the original constraints. Then, $Ax_0 - r \geq b$ and $\begin{pmatrix} x_0 \\ y \end{pmatrix}$ is a feasible point.
- Run the linear program to optimize the objective function, $\sum y_i$. (This is called *minimizing the sum of infeasibilities.*

7.3 Standard-Form LP

In standard form linear programming, we minimize the objective function, $c^T x$, subject to the constraints, $Ax = b$ and $x \geq 0$, where the rows of A are linearly independent. The previous case can be converted to this (and vice versa) by adding *slack variables.* The m equality constraints are always active, so a vertex is determined by a place with $n - m$ of the variables equal to 0. Everything can be converted to this.