Numerical Analysis

Sonia Reilly

October 15, 2023

These notes summarize Trefethen and Bau's Numerical Linear Algebra and Randall LeVeque's Finite Difference Methods for Ordinary and Partial Differential Equations.

Lecture 3 – Norms

- $||x||_1 = \sum_{i=1}^m |x_i|, \quad ||x||_2 = (\sum_{i=1}^m |x_i|^2)^{1/2}, \quad ||x||_\infty = \max_{1\leq i\leq m} |x_i|, \quad ||x||_p = (\sum_{i=1}^m |x_i|^p)^{1/p}$
- Cauchy-Schwartz Inequality: $|x^*y| \leq ||x||_2||y||_2$
- Hölder Inequality: $|x^*y| \leq ||x||_p||y||_q$ for any p, q such that $\frac{1}{p} + \frac{1}{q} = 1$.
- Induced matrix norm: $||A||_p = \sup \frac{||Ax||_p}{||x||_p}$
- Frobenius norm: $||A||_F = (\sum_{i=1}^m \sum_{j=1}^n |a_{ij}|^2)^{1/2} = \sqrt{\text{tr}(A^*A)} = \sqrt{\text{tr}(AA^*)}$
- $||AB|| \leq ||A|| \cdot ||B||$ for any of these norms

Lecture 4 – SVD

- $A = U\Sigma V^*$, where U and V are unitary and Σ is diagonal, so $Av_j = \sigma_j u_j$
- Reduced SVD $A = \hat{U}\hat{\Sigma}V^*$ drops extra orthogonal columns from U and extra rows of zeros from Σ

Lecture 5 – More on the SVD

- The SVD exists for every matrix, unlike the eigenvalue decomposition
- 2-norm of a matrix is the largest singular value, $||A||_2 = \sigma_1$
- Singular values are the square roots of eigenvalues of AA^* or A^*A

Lecture 6 – Projectors

- A projector P is orthogonal (projects away a component orthogonal to the components it retains) if P is hermitian/symmetric
- Given $\hat{Q}$ with orthonormal columns, $P = \hat{Q}\hat{Q}^*$ is an orthogonal projector onto the column space of $\hat{Q}$

Lecture 7 – QR Factorization

- $A = \hat{Q}\hat{R}$, where $\hat{Q} \in \mathbb{C}^{m\times n}$ has orthonormal columns and $\hat{R} \in \mathbb{C}^{n\times n}$ is upper-triangular
- Full QR adds extra orthogonal columns to Q and extra zero rows to R.
- All matrices $A \in \mathbb{C}^{m\times n}$ have QR factorizations, by following Gram-Schmidt. If A is not full rank, some r_{jj} will be 0, and q_j can be chosen to be an arbitrary orthogonal unit vector.
- If A is full rank, the reduced QR factorization is unique if we choose $r_{jj} > 0$ for all j.
- Fun fact: When A is the Vandermonde matrix, the columns of Q are (scalar multiples of) the Legendre polynomials
- Can solve $Ax = b$ quickly by solving $Rx = Q^*b$, since R is triangular and only requires substitution (but not as fast as LU).

Algorithm 1 Classical Gram-Schmidt (unstable)

for $j = 1 \to n$ **do**
  $v_j = a_j$
  for $i = 1 \to j-1$ **do**
    $r_{ij} = q_i^* a_j$
    $v_j = v_j - r_{ij} q_i$
  $r_{jj} = ||v_j||_2$
  $q_j = v_j / r_{jj}$

Lecture 8 – Gram-Schmidt Orthogonalization

- Modified Gram-Schmidt: instead of taking a column and removing the components that are already in the basis, take a column, normalize it, and remove the component in its direction from all the remaining columns.
- Equivalent to multiplying by an upper triangular matrix at each step.
- Both methods are $\approx 2mn^2$ flops

Algorithm 2 Modified Gram-Schmidt (stable)

for $i = 1 \to n$ **do**
  $v_i = a_i$
for $i = 1 \to n$ **do**
  $r_{ii} = ||v_i||_2$
  $q_i = v_i / r_{ii}$
  for $j = i+1 \to n$ **do**
    $r_{ij} = q_i^* a_j$
    $v_j = v_j - r_{ij} q_i$

Lecture 10 – Householder Triangularization

- QR method that is more numerically stable than Gram-Schmidt
- Instead of applying a sequence of triangular matrices, applies a sequence of unitary/orthogonal matrices.
- Construct $Q_n \cdots Q_1 A = R$, where each Q_i is of the form

$$Q_i = \begin{bmatrix} I & 0 \\ 0 & F \end{bmatrix}$$

  with $I \in \mathbb{C}^{(i-1)\times(i-1)}$. Each F is a Householder reflector that reflects the space in such a way that the relevant column $\mathbf{x}$ of the subset of A is mapped onto $\pm||\mathbf{x}||\mathbf{e}_1$ (whichever reflection is furthest from $\mathbf{x}$). Letting $\mathbf{v} = -\text{sign}(\mathbf{x}_1)||\mathbf{x}||\mathbf{e}_1$, the reflector is

$$F = I - 2\frac{\mathbf{v}\mathbf{v}^*}{\mathbf{v}^*\mathbf{v}}.$$

- Typically do not use this method to construct Q, but can apply it implicitly using projectors
- Operation count: $\approx 2mn^2 - \frac{2}{3}n^3$.

Lecture 11 – Least Squares Problems

- The least squares solution to minimizing $||r||_2 = ||Ax - b||_2$ occurs when $r \perp \text{range}(A)$, i.e., when $A^*r = 0 \implies A^*Ax = A^*b$.
- Classical method (technically the fastest) for solving the normal equations is to compute Cholesky factor $A^*A = R^*R$ and do two triangular solves.
- Alternate method (more numerically stable) computes a QR decomposition and solves

$$R^*Q^*QRx = R^*Q^*b \implies Rx = Q^*b.$$

- Another alternative (more stable if A is near rank-deficient) computes an SVD decomposition and solves

$$V\Sigma^*U^*U\Sigma V^*x = V\Sigma^*U^*b \implies \Sigma V^*x = U^*b.$$

- All three methods are $O(mn^2)$.

Nonlinear Least Squares and Gauss-Newton

- Instead of minimizing a linear function of x, now minimize $g(x) = \frac{1}{2}||F(x)||^2$.
- To do this using Newton's method, take $g'(x) = F'(x)F(x)$ and $g''(x) = F'(x)^TF'(x) + F''(x)^TF(x)$. Notice that near the correct solution $F(x)$ should be small if the data is close to compatible with the model, so we drop that term to avoid computing $F''(x)$.

- The resulting approach is the *Gauss-Newton method*, where the Newton update Δx^k is the solution to
$$F'(x^k)^T F'(x^k)\Delta x^k = -F'(x^k)^T F(x^k).$$
This is the normal equation for the linear least squares problem
$$\min_{\Delta x} ||F'(x^k)\Delta x^k + F(x^k)||,$$
so in practice we solve this with one of the methods above instead.
- The convergence of Gauss-Newton can approach quadratic if the data is very compatible with the model, otherwise it is linear.

Lecture 12 – Conditioning and Condition Number

- For a problem $x \to f(x)$, the *absolute condition number* $\hat{\kappa}(x)$ of the problem at x is
$$\hat{\kappa} = \lim_{\delta\to 0} \sup_{||\delta x||\le\delta} \frac{||\delta f||}{||\delta x||} = ||J(x)||.$$
The more commonly used *relative condition number* is defined by
$$\kappa = \lim_{\delta\to 0} \sup_{||\delta x||\le\delta} \left(\frac{||\delta f||}{||f(x)||} \Big/ \frac{||\delta x||}{||x||}\right) = \frac{||J(x)||}{||f(x)||/||x||}.$$
- Classical ill-conditioned problem: determine roots of a polynomial from coefficients. Also, finding eigenvalues of a non-symmetric matrix.
- The problem $x \to Ax$ has condition number
$$\kappa = ||A|| \frac{||x||}{||Ax||} \le ||A|| \left|\left|A^{-1}\right|\right|.$$
- The problem $A \to x = A^{-1}b$ has condition number
$$\kappa = ||A|| \left|\left|A^{-1}\right|\right|.$$
- The condition number of A is
$$\kappa(A) = ||A|| \left|\left|A^{-1}\right|\right| = \frac{\sigma_1}{\sigma_2}.$$

Lecture 14 – Stability

- Let $f(x)$ be the theoretical output of some algorithm and $\tilde{f}(x)$ the output in floating-point arithmetic.
- A stable algorithm gives "nearly the right answer to nearly the right question":
$$\frac{||\tilde{f}(x) - f(\tilde{x})||}{||f(\tilde{x})||} = O(\epsilon_{\text{machine}}) \quad \text{for some } \tilde{x} \text{ s.t.} \quad \frac{\tilde{x} - x}{x} = O(\epsilon_{\text{machine}}).$$
- A backward stable algorithm gives "exactly the right answer to nearly the right question":
$$\tilde{f}(x) = f(\tilde{x}) \quad \text{for some } \tilde{x} \text{ s.t.} \quad \frac{\tilde{x} - x}{x} = O(\epsilon_{\text{machine}}).$$

Lecture 15 – More on Stability

- Inner product: backward stable. Outer product: not backward stable (answer won't be exactly rank 1). Generally problems with solutions in a higher-dimensional space than the inputs are not backward stable.
- A backward stable algorithm has relative error

$$\frac{||\tilde{f}(x) - f(x)||}{||f(x)||} = O(\kappa(x)\epsilon_{\text{machine}})$$

where $\kappa(x)$ is the condition number of the problem.

Lecture 16 – Stability of Householder Triangularization

- Householder triangularization is backward stable in the sense that the input is $A = QR$ and the output is a new $\tilde{A} = \tilde{Q}\tilde{R}$. The $\tilde{Q}$ and $\tilde{R}$ computed in floating-point arithmetic may differ significantly from Q and R, so it is not forward stable, but $||A - \tilde{A}|| = O(\epsilon_{\text{machine}})$.

Lecture 17 – Stability of Back Substitution

- It's $O(m^2)$. It's backward stable. Trefethen and Bau prove this in agonizing detail. The point is to treat the error entirely as an error in the inputs.

Lecture 19 – Stability of Least Squares Algorithms

- Solving least squares using the normal equations is unstable. The relative error is $O(\kappa^2\epsilon_{\text{machine}})$ where κ is the condition number of A.
- The Householder and SVD methods are stable, with error $O(\kappa\epsilon_{\text{machine}})$. The Gram-Schmidt method is not automatically stable (Q is not accurately orthogonal), but it can be made to be stable easily.

Lecture 20/21/22 – Gaussian Elimination and Pivoting

- Solving a system of equations with Gaussian elimination / LU decomposition takes $\approx \frac{2}{3}m^3$ operations, vs. Householder QR's $\approx \frac{4}{3}m^3$ operations.
- LU is not backward stable without pivoting. L and U can be right to $O(\epsilon_{\text{machine}})$ while LU is off by a huge margin.
- Pivoting: at each step choose the row with x_{ik} the largest and swap it into the pivot position x_{kk}. Get $PA = LU$ decomposition where P is a permutation matrix (permuted identity matrix) and all the below-diagonal elements of L are ≤ 1.
- Pivoted Gaussian elimination is backward stable in the technical sense (for each m), but you can construct matrices such that $||U|| \approx 2^{m-1}||A||$ (e.g., all 1 on diagonal and right column, all -1 under diagonal, otherwise 0). This loses m digits of accuracy, so is useless for solving equations – "explosively unstable".

- In practice, these matrices never occur naturally and LU is extremely stable. It's still not proven why!

Lecture 23 – Cholesky Factorization

- For symmetric/hermitian positive definite matrices. Standard method for $Ax = b$ in s.p.d. cases.
- Idea: Zero out a column under a pivot by applying a lower-triangular matrix on the left (like in Gaussian elimination). Then to preserve symmetry, zero out row to the right of the pivot by applying an upper-triangular matrix to the right. Proceed until the middle matrix is the identity, to get $A = R^*R$, with R upper-triangular and $r_jj > 0$.
- Cholesky takes $\approx \frac{1}{3}m^3$ operations and is backward stable (but can be ill-conditioned in the forward sense of finding R).

Lecture 24 – Eigenvalue Problems

- Eigenvalue decomposition $A = X^{-1}\Lambda X$ exists only for square diagonalizable (nondefective) matrices.
- A matrix is defective if it has at least one eigenvalue whose algebraic multiplicity (multiplicity of the root in the characteristic polynomial) is greater than its geometric multiplicity (dimension of the corresponding eigenspace). Defective matrices are not similar to diagonal matrices, but they are similar to a matrix in *Jordan canonical form*, which is block diagonal with blocks of the form e.g.,
$$B = \begin{bmatrix} \lambda_1 & 1 & \\ & \lambda_2 & 1 \\ & & \lambda_3 \end{bmatrix}.$$
- Determinant and trace are the product and sum of the eigenvalues.
- A is unitarily/orthogonally diagonalizable (orthogonal eigenvectors) if and only if A is normal, i.e., $A^*A = AA^*$.
- Hermitian/symmetric matrices A are unitarily diagonalizable with real eigenvalues.
- The generalization of the eigenvalue decomposition to all square matrices is the *Schur factorization* $A = QTQ^*$, where Q is unitary and T is upper-triangular. The eigenvalues of A are the diagonal of T.

Lecture 25 – Overview of Eigenvalue Algorithms

- Nonstarters: solving the characteristic polynomial (ill-conditioned), power method (very, very slow).
- Alternative: construct eigenvalue or Schur decomposition. However, polynomial root-finding can be expressed as eigenvalue problem, and we know no algebraic formula exists for roots of degree ≥ 5 polynomials, so *any eigenvalue solver must be iterative.*

- Typically convert matrix to upper Hessenberg (zeros before first subdiagonal), then apply unitary $Q_j^* A Q_j$ to each side until the result approaches triangular.
- Takes $O(m^3)$ for the general case, and $O(m^2)$ for the hermitian case where we need only the eigenvalues.

Lecture 26 – Reduction to Hessenberg or Tridiagonal Form

- We'd like to compute the Schur factorization by triangularization, but if we apply a Householder reflection Q_i to zero out the elements under a_{11}, we need to apply Q_i^* on the right, which destroys the zeros.
- Instead apply Householder to zero out the elements under a_{21}, which leaves the top row unchanged. Then applying it on the right leaves the first column unchanged, so we preserve the zeros. Repeat for all columns to get $A = QHQ^*$ for H upper Hessenberg (or tridiagonal if A is hermitian).
- $O(m^3)$ and backward stable.

Lecture 27 – Rayleigh Quotient, Inverse Iteration

- Assume A is real and symmetric, and has been reduced to tridiagonal form.
- Rayleigh quotient,
$$r(x) = \frac{x^T A x}{x^T x}$$
takes an approximation x to an eigenvector and gives an approximation to the corresponding eigenvalue.
- Power iteration gives the eigenvector associated with the largest eigenvalue. Inverse iteration applies power iteration to $(A - \mu I)^{-1}$, which has the same eigenvectors and eigenvalues $(\lambda_i - \mu)^{-1}$. This method takes an estimate μ to an eigenvalue and gives the corresponding eigenvector.
- Rayleigh quotient iteration combines the two.
- Converges cubicly (triples digits of accuracy per iteration). Operation count is $O(m^3)$ for a full matrix $O(m^2)$ for upper Hessenberg, and $O(m)$ for tridiagonal.

Algorithm 3 Rayleigh Quotient Iteration

v^0 = some unit vector
$\lambda^0 = (v^0)^T A v^0$ = corresponding Rayleigh quotient
for $k = 1, 2, \ldots$ **do**
  Solve $(A - \lambda^{(k-1)} I) w = v^{(k-1)}$ for w ▷ apply $(A - \lambda^{(k-1)} I)^{-1}$
  $v^{(k)} = w / ||w||$ ▷ normalize
  $\lambda^{(k)} = (v^{(k)})^T A v^{(k)}$ ▷ Rayleigh quotient

Lecture 28 – QR Algorithm without Shifts

- Basic idea: Compute $A^{(k)} = QR$, set $A^{(k+1)} = RQ$. $A^{(k)}$ converges to diagonal (or generally triangular Schur) form.
- Comes from simultaneous iteration, the idea that given V with columns $v_1, \ldots, v_n$, the space $\langle A^k v_1, \ldots, A^k v_n \rangle$ converges to the space $\langle q_1, \ldots q_n \rangle$ of the eigenvectors corresponding to the n largest eigenvalues of A.
- Doing QR directly on $A^k V$ would be ill-conditioned, so instead do QR to orthonormalize V after each application of A, like normalizing in power iteration.
- The sequence of Q's and R's that QR algorithm generates is the same at each step as the ones in the simultaneous iteration algorithm, so the two algorithms are equivalent.
- In the QR algorithm,

$$A^k = Q^{(k)} R^{(k)} \quad \text{and} \quad A^{(k)} = (Q^{(k)})^T A Q^{(k)}.$$

- The diagonal elements of $A^{(k)}$ are Rayleigh quotients of the columns of $Q^{(k)}$, so they converge to eigenvalues.

Lecture 29 – QR Algorithm with Shifts

- There are three modifications made to make QR practical: reducing A to tridiagonal form first, factoring $A^{(k)} - \mu^{(k)} I$ instead of $A^{(k)}$ at each step, and breaking $A^{(k)}$ into submatrices whenever possible.

Algorithm 4 Practical QR Algorithm

$(Q^{(0)})^T A^{(0)} Q^{(0)} = A$ ▷ $A^{(0)}$ is a tridiagonalization of A
for $k = 1, 2, \ldots$ **do**
  Pick a shift $\mu^{(k)}$ ▷ e.g., choose $\mu^{(k)} = A^{(k)}_{mm}$
  $Q^{(k)} R^{(k)} = A^{(k-1)} - \mu^{(k)} I$ ▷ QR factorization of $A^{(k-1)} - \mu^{(k)} I$
  $A^{(k)} = R^{(k)} Q^{(k)} + \mu^{(k)} I$ ▷ Recombine factors in reverse order
  If any off-diagonal element $A^{(k)}_{j,j+1}$ is sufficiently close to 0,
    set $A_{j,j+1} = A_{j+1,j} = 0$ to get $\begin{bmatrix} A_1 & 0 \\ 0 & A_2 \end{bmatrix} = A^{(k)}$ and apply the QR algorithm to R_1 and R_2.

- This shifting gives cubic convergence. Cost is $O(m^2)$, and the algorithm is backward stable.

Lecture 30 – Other Eigenvalue Algorithms

- Jacobi, for full matrices. Zeros out a symmetric pair of elements at each step with a simple 2D (Jacobi) rotation, continues iteratively until off-diagonals are near 0. More parallelizable, but still not as fast as QR.
- Bisection search for zeros of characteristic polynomial – $O(m)$ per eigenvalue for a tridiagonal matrix, so faster than QR's $O(m^2)$ if only a few eigenvalues are needed.

- Divide and conquer (Cuppen 1981): split a tridiagonal matrix into two tridiagonal quadrants plus a rank-one correction, recurse on quadrants and recombine eigenvalues of the two with some nifty tricks. Faster than QR if both eigenvalues and eigenvectors are needed.

Lecture 31 – Computing the SVD

- Computing the SVD by computing the eigenvalue decomposition of A^*A is unstable, because it is governed by the squared condition number of A.
- Instead, we bidiagonalize A by Golub-Kahan bidiagonalization: applying (different) Householder reflectors to the left and right to zero out all but the diagonal and supradiagonal. (Can be done slightly faster by LHC bidiagonalization). Then use QR or divide-and-conquer to diagonalize the result.

Lecture 32 – Overview of Iterative Methods

- Krylov subspace methods (for **square** $m \times m$ matrices):

	$Ax = b$	$Ax = \lambda x$
$A = A^*$	CG	Lanczos
$A \neq A^*$	GMRES	Arnoldi

- These can be faster than the direct $O(m^3)$ both because of iteration and because they depend only on a black-box implementation of Ax, which can be fast if A is e.g., sparse.

Lecture 33 – The Arnoldi Iteration

- If $A = QHQ^*$, then $AQ_n = Q_{n+1}\tilde{H}_n$, where Q_n are the first n columns of Q and $\tilde{H}_n$ is the upper left $(n+1) \times n$ section of H. Then

$$Aq_n = h_{1n}q_1 + \cdots + h_{nn}q_n + h_{n+1,n}q_{n+1}.$$

  The Arnoldi iteration implements essentially Gram-Schmidt to isolate the last term at each step:

Algorithm 5 Arnoldi Iteration

$b =$ arbitrary, $q_1 = b/||b||$
for $n = 1, 2, 3, \ldots$ **do**
  $v = Aq_n$
  for $j = 1 \to n$ **do**
    $h_{jn} = q_j^* v$
    $v = v - h_{jn}q_j$
  $h_{n+1,n} = ||v||$
  $q_{n+1} = v/h_{n+1,n}$

- This recursion shows that the q_n are orthonormal bases of successive *Krylov subspaces*:

$$\mathcal{K}_n = \langle b, Ab, \ldots, A^{n-1}b \rangle = \langle q_1, q_2, \ldots, q_n \rangle \subseteq \mathbb{C}^m$$

- Arnoldi is equivalent to doing QR of the matrix $K_n = [b|Ab|A^2b|\cdots|A^{n-1}b]$, but stable.

Lecture 34 – How Arnoldi Locates Eigenvalues

- Essentially: do Arnoldi iteration, and at every iteration or certain iterations, use e.g., QR to compute eigenvalues of H_n. These are Arnoldi estimates or Ritz values, some of which converge rapidly to some of the more extreme eigenvalues. NOT for computing all eigenvalues.
- If x is in the Krylov space $\mathcal{K}_n$, then $x = p_n(A)b$ for some degree-n polynomial p_n.
- The polynomial that minimizes $||p_n(A)b||$ is the characteristic polynomial of H_n.
- (Skipped some stuff about Arnoldi lemniscates here.)
- Cautionary note: if the eigenvalues are highly sensitive to the matrix entries, this may not be a useful problem to solve.

Lecture 35 – GMRES

- Generalized minimal residuals: at each step n, x_n is the solution of the least-squares problem that minimizes $b - Ax_n$ for $x_n \in \mathcal{K}_n = \langle b, Ab, \ldots, A^{n-1}b \rangle$.
- By expressing x_n in Krylov basis as $x_n = K_n c$, we could express this as minimizing $||AK_n c - b||$, but this would be unstable. Instead Arnoldi iteration can give us orthogonalizations Q_n of the Krylov spaces so we can equivalently minimize $||AQ_n y - b||$. This $m \times n$ problem turns out to be equivalent to minimizing $||\tilde{H}_n y - ||b||e_1||$, which is an $(n+1) \times n$ problem.
- Updating the QR factorization of the previous $\tilde{H}_n$ for the next least squares only takes one Givens rotation, $O(n)$.
- GMRES, like Arnoldi, can be seen as minimizing $||p_n(A)b||$. Used for convergence analysis by bounding $||p_n(A)||$.
- GMRES will always converge in at most m iterations, but it is only useful if it converges in significantly less.
- In ideal conditions, GMRES converges as α^{-n}. It may converge poorly if A is poorly conditioned, if it is far from normal (i.e., if $A = V\Lambda V^{-1}$ and V is ill-conditioned), or if there are no normalized degree n polynomials whose values decrease rapidly with n at the eigenvalues of A. This may happen if the eigenvalues e.g., wrap around the origin rather than being clustered near it. Poor GMRES convergence means you need a good preconditioner.

Algorithm 6 GMRES

$q_1 = b/||b||$
for $n = 1, 2, 3, \ldots$ **do**
  Step n of Arnoldi iteration
  Find y to minimize $||\tilde{H}_n y - ||b||e_1||$ by QR least squares
  $x_n = Q_n y$

Lecture 36 – The Lanczos Iteration

- Like Arnoldi, but for real, symmetric matrices, so much faster.
- Again, will often converge to extreme eigenvalues first. Easier to quantify now that the eigenvalues are real. The rule of thumb is that if the eigenvalues are more spread out that Chebyshev points (as is usually the case), Lanczos will converge first to the extreme values. Otherwise it will converge first to the middle values.
- In the continuous analog where vectors become functions on $[-1, 1]$ and A becomes the linear operator that multiplies pointwise by x, Lanczos becomes the standard three-term recurrence relation for constructing orthogonal polynomials. The nodes and weights for Gauss quadrature fall out of a similar tridiagonal eigenvalue problem (see Lecture 37).

Lecture 38 – Conjugate Gradients

- For symmetric positive definite systems of equations (typically sparse, otherwise just use Cholesky.)
- CG is a set of recurrence relations that minimize the error of $x_n \in \mathcal{K}_n$ w.r.t. the true solution at each step, in the A norm ($||e_n||_A = \sqrt{e_n^T A e_n}$).

Algorithm 7 Conjugate Gradients

$x_0, r_0 = b, p_0 = r_0$
for $n = 1, 2, 3, \ldots$ **do**
  $\alpha_n = (r_{n-1}^T r_{n-1})/(p_{n-1}^T A p_{n-1})$ ▷ step length
  $x_n = x_{n-1} + \alpha p_{n-1}$ ▷ approximate solution
  $r_n = r_{n-1} - \alpha_n A p_{n-1}$ ▷ residual
  $\beta_n = (r_n^T r_n)/(r_{n-1}^T r_{n-1})$ ▷ improvement in this step
  $p_n = r_n + \beta_n p_{n-1}$ ▷ search direction

- At any unconverged step of CG,
$$\begin{aligned}\mathcal{K}_n &= \langle x_1, x_2, \ldots, x_n \rangle = \langle p_0, p_1, \ldots, p_{n-1} \rangle \\ &= \langle r_0, r_1, \ldots, r_{n-1} \rangle = \langle b, Ab, \ldots, A^{n-1} b \rangle,\end{aligned}$$
the residuals are orthogonal ($r_n^T r_j = 0$ for $j < n$) and the search directions are "A-conjugate" ($p_n^T A p_j = 0$ for $j < n$).
- This orthogonality combined with positive definiteness gives the optimality of each step.
- CG is also a minimization method for arbitrary functions. When applied to solve $Ax = b$, it can be viewed as minimizing $\phi(x) = \frac{1}{2} x^T A x - x^T b$.
- If A has only n distinct eigenvalues, CG converges in at most n steps. In general, given condition number κ of A, the convergence of CG is bounded by
$$\frac{||e_n||_A}{||e_0||_A} \leq 2 \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^n \sim \left(1 - \frac{2}{\sqrt{\kappa}} \right)^n,$$
so for κ large but not too large, convergence to a given tolerance takes at most $O(\sqrt{\kappa})$ iterations.

Lecture 39 – Biorthogonalization Methods

- Alternatives to GMRES:
- CGN: CG applied to the normal equations $A^*A = A^*b$ (without ever forming A^*A explicitly, since A^*A only needs to be applied to vectors). Takes $O(\kappa)$ iterations due to squaring. Useful only in cases where singular values are well behaved but eigenvalues are not.
- Lanczos does tridiagonal orthogonalization. Arnoldi generalizes this to Hessenberg orthogonalization for nonsymmetric matrices. BCG, CGS, QMR, etc. generalize it instead to tridiagonal biorthogonalization, where $A = VTV^{-1}$, V not necessarily orthogonal. Involves Krylov subspaces of both A and A^*.
- BCG's advantage over GMRES is that it involves three-term recurrences at each step, so work per step is much smaller. The disadvantages are that the convergence can be slower and is non-monotonic, and that it requires multiplication by A^* as well as A, which may or may not be available or efficient.

Lecture 40 – Preconditioners

- Idea: solve $M^{-1}A = M^{-1}b$ instead of $Ax = b$ iteratively, choosing an M that is close to A but much easier to solve (so that applying M^{-1} implicitly is easy).
- As a rule of thumb, a good preconditioner has $M^{-1}A$ near normal and with clustered eigenvalues.
- If A is hermitian positive definite, choose $M = CC^*$ also hermitian positive definite, and solve $(C^{-1}AC^{-*})C^*x = C^{-1}b$.
- Examples (many more in Trefethen and Bau): diagonal/Jacobi, incomplete Cholesky or LU (doing either on a sparse matrix, but not letting it overwrite zeros), courser-grid approximations in PDE problems, local approximations, block preconditioners (e.g., block diagonal/Jacobi), low-order discretization, a few steps of Jacobi or Gauss-Seidel, etc.

Polynomial Interpolation and Orthogonal Polynomials

Given $n+1$ function values of F, we want a function ϕ that passes through all of them, i.e., $\phi(x_i) = F_i$ for $i = 0, \ldots n$. One option of basis is Lagrange polynomials,

$$\phi_j(x) = \prod_{i=0, i\neq j}^{n} \frac{x - x_i}{x_j - x_i},$$

which have coefficients equal to the function values F_i. Lagrange polynomials are orthogonal for a vector inner product at the data points, but not for an integral inner product. The interpolant polynomial is also too expensive to compute in practice.

Any set of orthogonal polynomials satisfies a three-term recurrence relation of the form

$$P_{m+1}(x) = \alpha_m x P_m(x) - \beta_m P_m(x) - \gamma_m P_{m-1}(x),$$

since a new polynomial P_{m+1} can be created by multiplying P_m by x and then removing the components in the directions of previous polynomials. Since $\langle xP_m, P_n\rangle = \langle P_m, xP_n\rangle$, P_m is already orthogonal to P_{m-2} and the lower degree polynomials, so we only need three terms.

The orthogonal polynomials on $[-1, 1]$ with weight function $w(x) = 1$ are the Legendre polynomials. The roots of the Legendre polynomials are the Gaussian quadrature nodes.

The Chebyshev polynomials $T_m(x)$ are orthogonal on $[-1, 1]$ with respect to weight function $w(x) = (1 - x^2)^{-1/2}$, and satisfy the recurrence

$$T_{m+1}(x) = 2xT_m(x) - T_{m-1}(x).$$

On $[-1, 1]$, they have the values

$$T_m(x) = \cos(m \arccos x),$$

meaning that their roots

$$\xi_j = \cos\left(\frac{(j - 1/2)\pi}{m}\right) \qquad \text{for} j = 1, \ldots m$$

are the projections onto the x-axis of the set of equispaced points on a semicircle, and also that they are easy to compute using an FFT.

The interpolation problem is solved by making a Vandermonde matrix Φ of the chosen basis functions and solving $\Phi c = F$ for the coefficients c_i in the new basis. Interpolation on equispaced nodes tends to lead to large oscillations at the ends of the interval, known as *Runge's phenomenon*. This is avoided by using Chebyshev nodes (roots of the Chebyshev polynomials), for which the polynomial interpolants are guaranteed to converge to the function for any Lipschitz-continuous function (faster with more smoothness).

Interpolating polynomials are evaluated numerically using the Neville scheme based on Aitken's lemma, which provides a recurrence relation for a polynomial interpolant based on the interpolants passing through fewer points.

The *Hermite interpolation* is a polynomial of order n that coincides with the nodal values of a function at n possibly duplicated nodes, and also coincides with derivatives of the function at the duplicated nodes.

A common alternative: cubic splines, piecewise cubic interpolation.

Numerical Integration and Quadrature

Integration can be poorly conditioned if $I(f) = \int_a^b f(t)dt$ is small and the function changes sign (e.g., integral of $\sin(x)$ over a long interval).

A general quadrature formula has the form

$$\hat{I}(f) = \sum_{i=0}^{n} \lambda_i f(t_i)$$

with weights λ_i and nodal points t_i.

The *trapezoidal rule* integrates a piecewise linear interpolant at the nodes (error is $O(h^3)$). This is the simplest of the family of *Newton-Cotes formulas*, which use Lagrange polynomial approximations at equispaced points

$$\hat{f} = P_f(t|t_0, \ldots t_n) = \sum_{i=0}^{n} f(t_i)L_{in}(t)$$

where

$$\lambda_{in} = \frac{1}{b-a}\int_a^b L_{in}(t)dt.$$

The $n = 2$ rule is Simpson's rule, with weights 1/6, 4/6, 1/6. Newton-Cotes rules exactly integrate polynomials up to order n. They are suboptimal because high-degree polynomials suffer from Runge's phenomenon, i.e. oscillations at the ends of the interval. This can be solved by doing Newton-Cotes rules on lower-order piecewise polynomials.

Gauss quadrature is the unique set of nodes and weights such that the general quadrature formula integrates $\int_a^b \omega(x)f(x)dt$ exactly for polynomials of degree $2n+1$ or less. The nodes are the roots of the $n+1$-st polynomial with respect to the weight function ω and the weights are $\lambda_{in} = \frac{1}{b-a}\int_a^b L_{in}(t)dt$, where L are the Lagrange polynomials. For $\omega = 1$, the orthogonal polynomials are Legendre, so the method is Gauss-Legendre quadrature. Gauss quadrature has the optimal accuracy for the given number of points, but needs very smooth functions at high orders. It can be done on an infinite interval using Laguerre or Hermite polynomials.

Gauss quadrature is used in finite element methods, but otherwise interval partitioning (e.g., trapezoid method with an adaptive mesh) is more practical. Can be extended to multiple dimensions using bases like hat functions instead of trapezoid rule.

In high dimensions (10s to 100s), sparse grids can help circumvent the curse of dimensionality (Smolyak quadrature). An alternative is Monte Carlo.

1 Finite Difference Approximations

1.1 Truncation errors

We determine the error of a finite difference approximation by taking Taylor expansions around $u(x)$ of $u(x+h)$, etc., and choosing the leading power of h. One-sided approximations are first-order accurate, centered approximation is second-order accurate.

1.2 Deriving finite difference approximations

Method of undetermined coefficients: can make a higher-order accurate scheme by combining Taylor series with coefficients $a, b, c, ...$ and solving for coefficients such that the first several terms are 0, up to the desired power of h.

1.3 Second order derivatives

Standard second-order centered approximation is

$$D^2u(x) = \frac{1}{h^2}[u(x-h) - 2u(x) + u(x+h)].$$

Can be derived as a forward difference of a backward difference, vice versa, or from the interpolating quadratic polynomial at $x-h, x, x+h$, which suggests a finite difference rule based on any three non-uniformly spaced points.

1.4 Higher order derivatives

Typically derived by combining various lower order finite difference schemes, but can also be done with the method of undetermined coefficients, by solving a larger system of equations.

2 Steady States and Boundary Value Problems

Use

$$u''(x) = f(x)$$
$$u(a) = \alpha, \quad u(b) = \beta$$

as an example, viewing it as the steady state of the heat equation.

2.1 A simple finite-difference method

We want $U_0, \ldots, U_{m+1}$ which approximate $u(x_0), \ldots, u(x_{m+1})$, where $x_j = jh$ and $h = 1/(m+1)$. Using the standard second-order finite difference rule, we get a linear system $AU = F$ for $U_1, \ldots, U_m$, where

$$A = \frac{1}{h^2}\begin{bmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 1 & -2 \end{bmatrix}, \quad F = \begin{bmatrix} f(x_1) - \alpha/h^2 \\ f(x_2) \\ \vdots \\ f(x_{m-1}) \\ f(x_m) - \beta/h^2 \end{bmatrix}.$$

2.2 Error and stability

The *global error* is defined as the pointwise error $E_j = U_j - u(x_j)$. The *local truncation error* is $\tau = A\hat{U} - F$, where $\hat{U}$ is the true solution $\hat{U}_j = u(x_j)$. It satisfies $A^h E^h = -\tau^h$, so $||E^h|| \leq ||(A^h)^{-1}||||\tau^h||$. A finite difference method for a linear BVP is *stable* if there is a constant bound on $||(A^h)^{-1}||$ for all h sufficiently small, so that the global error is of the same order in h as the local truncation error.

Using the 2-norm, $||(A^h)^{-1}||$ is the inverse of the minimum eigenvalue of A^h. The eigenvalues in this case are $\lambda_p = \frac{2}{h^2}(\cos(p\pi h) - 1)$, with eigenvectors $u_j^p = \sin(p\pi jh)$. The smallest eigenvalue is $\lambda_1 = -\pi^2 + O(h^2)$, bounded away from 0, so the finite difference approximation is stable.

To bound the max of the global error (∞-norm), note that the columns of $(A^h)^{-1}$ are discrete counterparts to the Green's functions, because $A^h(A^h)^{-1} = I$. The Green's functions for the heat equation give us bounds on the elements of $(A^h)^{-1}$, so we can bound its ∞-norm to confirm stability.

2.3 Neumann boundary conditions

With one Neumann boundary condition $u'(0) = \sigma$, a one-sided approximation $\frac{1}{h}(-U_0 + U_1) = \sigma$ introduces local truncation error $O(h)$ and therefore global error $O(h)$, which is undesirable. Instead we use $\frac{1}{h}(-U_0 + U_1) = \sigma + \frac{h}{2}f(x_0)$, a centered approximation to $u'(x_0 + \frac{h}{2}) \approx u'(x_0) + \frac{h}{2}u''(x_0)$, or a second-order accurate one-side rule using U_0, U_1, U_2. The latter is more generalizable but breaks the tridiagonal structure slightly.

2.4 Existence and uniqueness

The matrix A may be singular when the problem is ill-posed, e.g., in the heat equation case with two Neumann boundary conditions (the solution is only defined up to an additive constant).

2.5 General second order linear equations

In the general case $a(x)u''(x) + b(x)u'(x) + c(x)u(x) = f(x)$, construct $AU = F$ using centered difference approximations to $u''(x)$ and $u'(x)$ to get a second-order accurate scheme. In some cases it may be better to do this differently, e.g., discretizing $(\kappa(x)u'(x))' = f(x)$ in two stages of centered difference to maintain symmetry and negative definiteness.

2.6 Nonlinear equations

Applying finite difference approximations to a nonlinear equation gives a nonlinear system of equations, which can be solved using Newton's method. If the solution is not unique, different initial guesses in Newton's method may converge to different solutions.

2.7 Singular perturbations and boundary layers

Consider the steady-state 1D advection-diffusion equations $au' = \kappa u'' + \psi(x)$ with 2-point Dirichlet BC's on $[0, 1]$. When $a >> \kappa$, advection overwhelms diffusion and the equation requires 2 boundary conditions, but is very close to a first-order equation in which only one boundary condition can be satisfied in general. This is a *singularly perturbed equation*, where a small added term of a high order changes the character of an equation. The solution is approximately linear with a sudden jump near $x = 1$, known as a *boundary layer*. While the local truncation error is only large in the boundary layer, the global error is large everywhere.

Problems with *interior layers*, such as $\epsilon u'' + u(u' - 1) = 0$, have sudden jumps on the interior of the domain. One approach is to use perturbation theory to obtain a good approximation to the solution, then use it to initialize Newton's method or devise an appropriate nonuniform grid. Alternatively can use adaptive mesh refinement, and/or *continuation/homotopy methods*, where the solution is calculated for sequence of larger ϵ_i approaching ϵ, and each solution is used as an initial guess for the next.

2.8 Methods with higher order accuracy

The obvious approach to fourth-order accuracy is a fourth-order finite difference scheme, which gives a pentadiagonal matrix that can still be solved in $O(m)$. In more general, e.g., higher dimensional, cases, we can instead use an *extrapolation method*, where we solve the problem with $O(h^2)$ accuracy on the h-spaced grid and on a finer $h/2$-spaced grid, then combine the solutions in such a way that the second-order error cancels and we are left with $O(h^4)$ error. Another approach is the method of *deferred corrections*, where we solve $AU = F$ using second-order finite difference, then use U to approximate the local truncation error $\tau_j = \frac{1}{12}h^2 u''''(x_j) + O(h^4)$, then solve $AE = -\tau$ to get an approximation to the global error which can be used to make a correction to U to obtain fourth-order accuracy.

2.9 Spectral methods

A *spectral method* is a method that can converge at a faster-than-polynomial rate in h. We seek an approximation in terms of orthogonal basis functions,

$$U(x) = \sum_{j=1}^{N} c_j \phi_j(x),$$

such that $U(x)$ satisfies the equation exactly at the grid points (we call this *spectral collocation* or the *pseudospectral method* as opposed to the *Galerkin approach*, where we minimize some norm of the residual for U.) Collocation is the logical limit of increasing accuracy by widening the stencil, until the stencil includes every grid point and the "finite difference approximation" is the derivative of a polynomial that passes through each grid point. This in theory would give $O(h^{1/h})$ accuracy, but that only happens when we choose the grid points to be Chebyshev points rather than equispaced points. Common orthogonal bases are Chebyshev polynomials or the Fourier basis.

3 Elliptic Equations

For 2D elliptic equations like the 2D Poisson equation, we can use a 5-point stencil to approximate the Laplacian. When the points are ordered rowwise, A for the Poisson equation is block-tridiagonal. The local truncation error can be found exactly as for the 1D case, and is $O(h^2)$, as is the global error, since the problem is stable (verify by computing min eigenvalue of A and bounding it away from 0). Computing also the max eigenvalue gives that the condition number of A is $O(h^{-2})$, which explains why iterative methods struggle. A 9-point stencil still gives $O(h^2)$ error, but can be modified using a special form of deferred corrections to give $O(h^4)$ error.

Direct solution methods for the 1D tridiagonal Poisson matrix take $O(m)$, and for the $m^2 \times m^2$ 2D block-tridiagonal Poisson matrix take $O(m^4)$ for Gaussian elimination, $O(m^3)$, or $O(m^2 \log m)$ for fast Poisson solvers based on the FFT.

4 Iterative Methods for Sparse Linear Systems

4.1 Richardson, Jacobi and Gauss-Seidel

For a general system $Ax = b$, the most obvious iterative solution method is the *Richardson iteration*,

$$x_{k+1} = (I - A)x_k + b,$$

which has as a fixed point the true solution $x = A^{-1}b$. This is slow and does not always converge, so we can precondition it with a matrix $Q \approx A$ that is easier to solve and use the Richardson iteration to solve $Q^{-1}Ax = Q^{-1}b$ instead. If we split $A = L + D + U$ into upper and lower triangular and diagonal pieces, setting $Q = D$ gives the *Jacobi iteration*, which converges when A is strictly diagonally dominant. Choosing $Q = D + L$ gives the *Gauss-Seidel iteration*, which converges when A is spd.

For the Poisson equation, the Jacobi iteration is given by

$$u_{ij}^{[k+1]} = u_{i-1,j}^{[k]} + u_{i+1,j}^{[k]} + u_{i,j-1}^{[k]} + u_{i,j+1}^{[k]} - \frac{h^2}{4} f_{ij}$$

and the slightly faster converging Gauss-Seidel iteration is given by

$$u_{ij}^{[k+1]} = u_{i-1,j}^{[k+1]} + u_{i+1,j}^{[k]} + u_{i,j-1}^{[k+1]} + u_{i,j+1}^{[k]} - \frac{h^2}{4} f_{ij}.$$

Both converge for any initial $u^{[0]}$ (proven by writing the iterations as a matrix equation and bounding the spectral radius of the matrix to be < 1). They converge linearly (error decreases by a factor of the spectral radius at each iteration), and the number of iterations to reach error on the order of the global error is $O(m^2 \log m)$, so the total work is $O(m^4 \log m)$. Gauss-Seidel requires about half as many iterations as Jacobi because of a smaller spectral radius.

The *method of successive overrelaxation* computes a Gauss-Seidel iteration and then takes a step in that direction with a larger, optimal step length. Its effectiveness is highly sensitive to step length but with the optimal value it can take only $O(m \log m)$ iterations. Not the most useful because the optimal step length is hard to compute in general.

4.2 Conjugate Gradients

LeVeque has a great geometric explanation of CG. In a 2D example of concentric ellipse level sets, steepest descent will move perpendicular to the level sets, while CG will take one initial step and then the A-conjugate direction will point directly towards the minimizer in the center of the ellipse. CG does this in every dimension until the answer is the minimizer in $\mathbb{R}^m$. (Go back to CG without the Agonizing Pain for more detail.)

4.3 Newton-Krylov methods for nonlinear problems

When you discretize a nonlinear problem you get a nonlinear equation, which we can solve with a Newton iteration. You could form the Jacobian and then use GMRES to solve for the Newton step, but Newton-Krylov often refers to *Jacobian-free Newton-Krylov methods* (JFNK). In JFNK the Jacobian is not constructed, and only computed using finite difference for the specific directions q_k needed by GMRES (since GMRES only requires matvecs).

4.4 Multigrid methods

Use 1D Poisson as an example. Jacobi iteration removes the middle frequencies of the error first, and takes forever to capture very low and very high frequencies. Underrelaxed Jacobi works quickly on the upper half of frequencies (but slower on the lower half). The idea of multigrid is to use underrelaxed Jacobi to remove the high frequency error, then downshift to a coarser grid so that the lower frequency error is now higher frequency, and then repeat. In particular:

Algorithm 8 V-cycle

Do a few iterations (e.g., $\nu = 3$) of a smoother such as underrelaxed Jacobi on $Au = f$.
Compute residual $r_\nu = f - Au_\nu$.
Coarsen residual to get $\tilde{r}$.
Approximately solve $\tilde{A}\tilde{e} = -\tilde{r}$ on the coarse grid (recursively).
Interpolate $\tilde{e}$ back to the original grid to get e_ν and compute a new guess $u_\nu - e_\nu$.
Using this as a starting guess, do a few more smoothing steps on $Au = f$.

This V-cycle can be repeated multiple times (for 1D Poisson, $O(\log m)$) or adapted to a W-cycle as needed to achieve the desired accuracy. *Full multigrid* (FMG) starts at the coarsest level and usually only needs one $V - cycle$, so the total work is the optimal $O(m)$. For 2D Poisson, it can do $O(m^2)$, beating the $O(m^2 \log m)$ of Fast Poisson Solvers.

5 The Initial Value Problem for ODEs

Consider the first-order IVP $u'(t) = f(u(t), t)$ for $t > t_0$ and $u(t_0) = \eta$ (higher-order equations can be decomposed into first-order systems).

5.1 Linear ODEs

A first-order linear ODE always has a unique solution. In the constant-coefficient case $u' = Au$, the solution is $u(t) = e^{A(t-t_0)}\eta$. Duhamel's principle gives an explicit solution to the inhomogeneous equation.

5.2 Nonlinear ODEs

The nonlinear ODE $u'(t) = f(u,t)$ is only guaranteed to have a unique solution if f is Lipschitz continuous in u. (E.g., $u' = \sqrt{u}$ near $t = 0$ has two solutions.) The Lipschitz constant indicates how the slope of the solution curve varies with perturbations of u, so it roughly determines how errors are magnified in time-stepping.

5.3 Some basic numerical methods

Given $U^0 = \eta$, we want $U^n \approx u(t_n)$, for $t_n = nk$.

Forward Euler / explicit method (from forward finite difference):

$$U^{n+1} = U^n + kf(U^n)$$

Backward Euler / implicit method (from backward finite difference; requires solving for U^{n+1}):

$$U^{n+1} = U^n + kf(U^{n+1})$$

Trapezoidal method (from averaging forward and backward Euler; second-order accurate):

$$U^{n+1} = U^n + \frac{k}{2}(f(U^n) + f(U^{n+1}))$$

Leapfrog method (from symmetric finite difference; second-order accurate):

$$U^{n+1} = U^{n-1} + 2kf(U^n)$$

You could also consider Taylor expanding $u(t_{n+1})$ around $u(t_n)$ and using the ODE to write the derivatives in terms of u. Keeping only the first derivative this gives Euler's method, but it's not very practical for deriving higher order methods since higher order derivatives would require repeated applications of f, which is cumbersome.

The LTE of these methods is computed using the finite difference form, e.g.,

$$\tau^n = \frac{u(t_{n+1}) - u(t_n)}{k} - f(u(t_n)).$$

5.4 Runge-Kutta methods

Two-stage explicit RK computes an intermediate $U^* \approx u(t_{n+1/2})$ by Euler's method and uses the slope at that point to calculate the update.

$$\begin{aligned} U^* &= U^n + \frac{1}{2}kf(U^n) \\ U^{n+1} &= U^n + kf(U^*) \end{aligned}$$

A general r-stage RK method has the form

$$\begin{aligned} Y_1 &= U^n + k\sum_{j=1}^{r} a_{1j} f(Y_j, t_n + c_j k), \\ &\vdots \\ Y_r &= U^n + k\sum_{j=1}^{r} a_{rj} f(Y_j, t_n + c_j k), \\ U^{n+1} &= U^n + k\sum_{j=1}^{r} b_j f(Y_j, t_n + c_j k), \end{aligned}$$

with coefficients summarized in a Butcher tableau

$$\begin{array}{c|ccc} c_1 & a_{11} & \cdots & a_{1r} \\ \vdots & \vdots & & \vdots \\ c_r & a_{r1} & \cdots & a_{rr} \\ \hline & b_1 & \cdots & b_r \end{array}$$

An explicit method such as RK4 only has nonzero values under the diagonal of the matrix a_{ij}, since Y_j only depends on previous Y_i:

$$\begin{array}{c|cccc} 0 & & & & \\ 1/2 & 1/2 & & & \\ 1/2 & 0 & 1/2 & & \\ 1 & 0 & 0 & 1 & \\ \hline & 1/6 & 1/3 & 1/3 & 1/6 \end{array}$$

A fully implicit RK method (all a_{ij} can be nonzero) is usually too expensive to compute. *Diagonally implicit* RK methods (DIRK) have a lower triangular a_{ij}, so they only require r implicit solves. The rows of a_{ij} should add up to c_i, and the b_j's should add up to 1. There is an additional condition for second-order accuracy, and the number of conditions required increases exponentially with the accuracy order desired.

Embedded RK methods uses two RK methods that differ only in their b_j values but have different orders to approximate the error of the lower order method without needed to compute extra function evaluations. The error estimation is used to adapt the time step. Matlab's ode45 is based on embedded 4th and 5th order RK methods.

5.5 Linear multistep methods (LMMs)

Multistep methods use previous steps (U^{n-1}, U^{n-2}, etc.) in addition to U^n. One-step methods have some advantages, such as not requiring an alternative method at U^0, handling nonsmoothness well when the problem is at a grid point, and handling adaptive time-stepping easily. However, Taylor series methods require nested function calls and RK methods require many function calls and/or implicit solves.

In an r-step LMM, U^{n+1} is given as a linear combination of the previous r U^j's and $f(U_j)$'s (including U^{n+1} for an implicit LMM).

The *Adams-Bashforth* methods have the structure

$$U^{n+r} = U^{n+r-1} + k\sum_{j=0}^{r} \beta_j f(U^{n+j}),$$

where the β_j's are chosen to obtain the maximum r^{th} order of accuracy by Taylor expanding the LTE and canceling as many terms as possible. (Adams-Bashforth 1 is forward Euler, 2 and 3 are common). The r-step *Adams-Moulton* methods are the implicit versions, and have accuracy order $r+1$. The *Nyström* methods are like the Adams methods but have a U^{n+r-2} term instead of U^{n+r-1} (an implicit example is Simpson's rule). *Predictor-corrector* methods use an Adams-Bashforth method to approximately predict U^{n+1} and use the approximation to replace the implicit term of an Adams-Moulton method, to get a new corrected U^{n+1}. The first approximation is lower order than the final answer, so the difference can be used to estimate error and adapt the time step.

LMMs typically need to be initialized with one-step methods. An LMM of order r can be initialized with a method of order $r-1$, which will only introduce order r error if used for a constant number of steps.

6 Zero-Stability and Convergence for IVPs

An r-step method is convergent if U^N, $Nk = T$, limits to $u(T)$ as $k \to 0$ for any T, for any ODE with $f(u)$ Lipschitz continuous, and for any r starting values that all limit to the initial value η. A convergent method must be consistent ($o(1)$ LTE as $k \to 0$) and *zero-stable*.

To prove zero-stability for some one-step method for some $u' = f(u)$, write $u(t_{n+1})$ in terms of the method and the local error, and subtract the expression for the U^{n+1} to get a recursive equation for the global error E^n. If the one-step method is Lipschitz-continuous in U_n with some constant L' that depends on the Lipschitz constant L of f, then

$$|E^n| \leq e^{L'T} T ||\tau||_\infty,$$

so the global error has the same order as the local error and the method is stable. The Lipschitz continuity is required to essentially bound the behavior of the ODE by the behavior of $u' = L'u$. Not all LMMs have the zero-stability that one-step methods have. It turns out that LMMs are zero-stable iff the characteristic polynomial of the LMM has no roots greater than 1, and no repeated roots equal to 1.

7 Absolute Stability for ODEs

Zero-stability does not guarantee that the global error does not explode for any given k, and k may need to be tiny to get good behavior. The region of *absolute stability* of a method is the set of complex points $z = k\lambda$ where an error in one time step of the method applied to $u' = \lambda u$ will not grow in future time steps. E.g., for Euler's method, $U^{n+1} = (1 + k\lambda)U^n$, so the region is the circle $|1 + z| \leq 1$.

For a general LMM of the form $\sum \alpha_j U^{n+j} = k \sum \beta_j f(U^{n+j})$, the *stability polynomial* is $\pi(\zeta; z) = \sum(\alpha_j - z\beta_j)\zeta^j$. The region of absolute stability is the set of z for which the polynomial has no roots greater than 1, and no repeated roots equal to 1. So a method is zero-stable if it is absolutely stable for $z = 0$. For a system of equations $u' = Au$, a method is absolutely stable if $k\lambda_p$ is in the region of absolute stability for every eigenvalue λ_p of A. For a general system $u' = f(u)$, the general rule of thumb is that $k\lambda_p$ should be in the region of absolute stability for every eigenvalue λ_p of the Jacobian matrix $f'(u)$, but this is not always sufficient.

If you need to choose k to be very small because otherwise the method is not stable, choose a different method (e.g., explicit methods for stiff ODEs run into stability issues).

8 Stiff ODEs

Roughly, a problem is stiff if the desired solution curve is slowly varying but perturbations of it are rapidly varying. E.g., $u'(t) = \lambda(u - \cos t) - \sin t$ has the solution $u(t) = \cos t$ for $u(0) = 1$, but for $u(t_0) = \eta$, the solution is $u(t) = e^{\lambda(t-t_0)}(\eta - \cos t_0) + \cos t$, which for $\lambda << 0$ converges very rapidly to $u(t) = \cos t$ after a perturbation η.

Stiff ODEs are hard for finite difference because the LTE at each step introduces a perturbation which introduces a large derivative in the solution, which must be handled by a tiny time step, so the region of absolute stability is tiny. For a scalar ODE, $f'(u) >> u'(t)$ is a sign of stiffness (the ODE changes a lot with u, but the solution is smooth), and for a system, and the largest-to-smallest eigenvalue ratio λ_m/λ_1 of the Jacobian $f'(u)$ is a measure of stiffness (longest timescale to shortest timescale ratio).

Generally the methods used are all implicit, in particular BDF (higher order generalizations of backward Euler). For some "mildly stiff" problems, Runge-Kutta-Chebyshev are explicit methods that can work.

9 Diffusion Equations and Parabolic Problems

To solve $u_t = u_{xx}$ we typically use an implicit time stepping method. We can use any combination of time and spatial discretization, but the classic is *Crank-Nicolson*:

$$\begin{aligned}\frac{U_i^{n+1} - U_i^n}{k} &= \frac{1}{2}(D^2 U_i^n + D^2 U_i^{n+1}) \\ &= \frac{1}{2h^2}(U_{i-1}^n - 2U_i^n + U_{i+1}^n + U_{i-1}^{n+1} - 2U_i^{n+1} + U_{i+1}^{n+1}).\end{aligned}$$

This creates a tridiagonal system of equations which take $O(m)$ work to solve. In more than one spatial dimension, the matrix has the same structure as for an elliptic equation. Crank-Nicolson has LTE $O(k^2 + h^2)$, so it is 2nd order in space and time.

The *method of lines* (MOL) refers to discretizing in space, which yields a linear system of ODEs, and then solving by discretizing in time (e.g., applying the trapezoidal method in time gives Crank-Nicolson). The implicit trapezoidal method is unconditionally stable (for any negative λ), and the eigenvalues of the finite difference matrix are negative, so it is stable for any time step, but it turns out to need $k = O(h)$ for accuracy.

The system produced by the MOL for the heat equation is stiff, because the largest magnitude eigenvalue of the finite difference matrix is $\lambda_m \approx -4/h^2$ and the smallest is $\lambda_1 \approx -\pi^2$. So for an explicit method the time step would have to go as $O(h^2)$, so instead we use implicit time stepping. The stiffness reflects the fact that high frequency oscillations diffuse on very short timescales relative to low frequencies.

9.1 Convergence and Stability

In general, k and h cannot go to 0 at any rates and still have guaranteed convergence at every point and time. If we fix h as some function of k (determined by the stability region of the time discretization), we can write a time discretization of the MOL system as

$$U^{n+1} = B(k)U^n + b^n(k).$$

The method is *Lax-Richtmyer* stable if for each T there is a constant $C_T > 0$ such that $||B(k)^n|| \leq C_T$ for all $k > 0$ and n for which $kn \leq T$. The *Lax Equivalence Theorem* states that a consistent linear method is convergent iff it is Lax-Richtmyer stable. (For many common methods,

including Crank-Nicolson, $||B|| \leq 1$, but methods with $||B|| \leq 1 + \alpha k$ for some α also satisfy the condition.) This analysis applies only for linear discretizations, i.e., typically only linear PDEs – PDE discretizations aren't bounded by a Lipschitz constant that is constant in h, so the nonlinear generalization used for ODEs does not apply.

9.2 Von Neumann analysis

To show that a finite difference method is Lax-Richtmyer stable, we want to show $||B||_2 \leq 1 + \alpha k$, or $||U^{n+1}||_2 \leq (1+\alpha k)||U^n||_2$ for all U^n for some α. The size of B grows with h and the U^n's are all coupled, so these are both hard formulations to work with. Instead we can take a Fourier transform by noting that the grid function $W_j = e^{ijh\xi}$ is an eigenfunction of any translationally-invariant finite difference operator, and get $||\hat{U}^{n+1}||_2 \leq (1 + \alpha k)||\hat{U}^n||_2$ by Parseval's relation $||\hat{V}||_2 = ||V||_2$.

The Fourier transform uncouples the $\hat{U}^n$'s and we get a recursion $\hat{U}^{n+1}(\xi) = g(\xi)\hat{U}^n(\xi)$ for every wave number ξ, so if $g(\xi) \leq 1 + \alpha k$ for all ξ and some α, the method is stable. To apply Von Neumann analysis, take an arbitrary wave numberξ and set $U_j^n = e^{ijh\xi}$ and $U_j^{n+1} = g(\xi)e^{ijh\xi}$. Plug into the full PDE discretization and solve for $g(\xi)$, then find the relation between k and h that ensures $g(\xi) \leq 1 + \alpha k$.

9.3 Multidimensional problems

The heat equation in 2 spatial dimensions can be solved using 2D Crank-Nicolson, where the discretization of u_{xx} in the 1D case is replaced by a 5-point discretization of the Laplacian. Solving the resulting block-tridiagonal system in every iteration is manageable because the condition number scales as $O(k/h^2)$, lower than the $O(1/h^2)$ for elliptic equations, and because previous time steps are great initial guesses when solving the system iteratively.

Alternatively, the *locally one-dimensional* (LOD) method instead uses Crank-Nicolson first in the x-direction, solving $u_t = u_{xx}$, then in the y-direction, solving $u_t = u_{yy}$, and alternating, essentially diffusing the solution in one direction then the other. This has the advantage of decoupling into $2m$ tridiagonal systems, so it has the optimal $O(m^2)$ complexity. It can have second-order accuracy if BCs are handled carefully. The *alternating direction implicit* (ADI) method is a variation of LOD where the first step of Crank-Nicolson is explicit in x and implicit in y, then the second step is the reverse, and the process repeats.

10 Advection Equations and Hyperbolic Systems

We will use the scalar advection equation $u_t + au_x = 0$ with initial data $u(x, 0) = \eta(x)$ and periodic boundary conditions $u(0, t) = u(1, t)$ as an example of a hyperbolic system. Hyperbolic systems can generally be solved with explicit time discretization using $k = O(h)$, unlike parabolic equations.

Applying the method of lines with a centered difference in space gives a skew-symmetric matrix with imaginary eigenvalues, so methods that do not include the imaginary axis in their stability regions (e.g., forward Euler) will be unstable for $k = O(h)$. We seek other explicit methods.

10.1 Leapfrog

The midpoint method is stable and gives *leapfrog method* for the advection equation,

$$U_j^{n+1} = U_j^{n-1} - \frac{ak}{h}(U_{j+1}^n - U_{j-1}^n).$$

This method is *marginally stable* because it is on the edge of the region of stability (which is just an interval of the imaginary axis). This makes it *nondissipative*, since no eigenmodes decay, which is good since the advection equation is nondissipative, but it may cause instabilities in more complicated equations. It is also a three-level method, which is inconvenient for various reasons (e.g., tricky BCs, more storage, etc.)

10.2 Lax-Friedrichs

The *Lax-Friedrichs* method modifies the forward Euler discretization of the MOL system:

$$U_j^{n+1} = \frac{1}{2}(U_{j-1}^n + U_{j+1}^n) - \frac{ak}{2h}(U_{j+1}^n - U_{j-1}^n).$$

It can be rewritten as a forward Euler discretization of the advection-diffusion equation $u_t + au_x = \epsilon u_{xx}$ with $\epsilon = h^2/2k$, which has an MOL system that can be stable with forward Euler. In fact Lax-Friedrichs is Lax-Richtmyer stable when $|ak/h| \leq 1$, but it is rarely used due to its first-order time accuracy. (Von Neumann analysis can also be used to derive the stability of all of these methods, but note that it is only applicable in the constant coefficient linear case of the Cauchy or periodic problem.)

10.3 Lax-Wendroff

To derive the *Lax-Wendroff* method, Taylor expand $u(x, t + k)$ around t to 2nd order, replace the first and second derivatives with spacial derivatives using the PDE, and substitute in centered approximations to u_x and u_{xx} to get

$$U_j^{n+1} = U - j^n - \frac{ak}{2h}(U_{j+1}^n - U_{j-1}^n) + \frac{a^2k^2}{2h^2}(U_{j-1}^n - 2U_j^n + U_{j+1}^n).$$

This is a two-level explicit method with second-order time accuracy. This can again be written as a discretization of advection-diffusion for analysis purposes, and it has the same condition $|ak/h| \leq 1$ for Lax-Richtmyer stability. It is not nondissipative, and high wave number modes can be damped significantly, requiring a finer grid to resolve.

10.4 Upwind methods

The advection equation is not symmetrical, so one-sided finite difference stencils can be useful. The first-order upwind method is

$$U_j^{n+1} = U_j^n - \frac{ak}{h}(U_j^n - U_{j-1}^n)$$

for $a > 0$, or the opposite for $a < 0$. For $a > 0$, this method is stable for $0 \leq ak/h \leq 1$. The Beam-Warming method is the 2nd-order analogue that uses one-sided 2nd-order finite difference stencils.

The upwind method can also be derived as a linear interpolation of the solution $u(x_j, t_{n+1}) = u(x_j - ak, t_n)$. A centered quadratic interpolation is the Lax-Wendroff method, and a one-sided quadratic interpolation is Beam-Warming.

10.5 Courant-Friedrichs-Lewy (CFL) condition

The *domain of dependence* of a point (X, T) for a PDE is the set of points at which the the initial data affects the solution $u(X, T)$. For the advection equation it's just the point $X - aT$. For the heat equation it's the entire domain (infinite propagation speed).

The domain of dependence of a finite difference method is the set of points at time 0 that affect the numerical value at (X, T). E.g., for the advection equation there is a tree of dependencies that converges to an interval at time 0 as $k, h \to 0$ with k/h fixed. The point $X - aT$ is in this interval iff $|ak/h| \leq 1$.

The CFL condition says that a convergent numerical method must have a numerical domain of dependence that contains the true domain of dependence of the PDE, in the limit as $k, h \to 0$.

For methods for hyperbolic equations, this puts constraints on the *Courant number* $\nu = ak/h$. For the heat equation, this is not satisfied by any 3-point explicit method for $k = O(h)$, but it is satisfied for $k = O(h^2)$ since as $k \to 0$, the numerical domain of dependence expands to the whole real line. For implicit methods like Crank-Nicolson, any k satisfies the CFL condition, because the tridiagonal system has a dense inverse that couples all the grid points.

10.6 Modified equations

Numerically, the upwind method is very dissipative (high wave numbers dissipate out), while the Lax-Wendroff method is dispersive (not all wave numbers travel at the same speed) and the leapfrog method is even more dispersive.

This is explained by analysis using the method of modified equations, in which we find a PDE such that the discretized solution solves the new PDE to higher order than the original one. Using Taylor expansions, the upwind method can be shown to satisfy an advection diffusion equation (added u_{xx} term) to 2nd order, while it only satisfies the advection equation to 1st order. So it is diffusive. The Lax-Wendroff solution satisfies the advection equation to 2nd order, but it satisfies an equation with an added u_{xxx} term to 3rd order, so it is dispersive.

10.7 Hyperbolic systems

The system $u_t + Au_x = 0$ is hyperbolic if A is diagonalizable with real eigenvalues. Each eigenvalue corresponds to a characteristic, and the domain of dependence includes one point for every eigenvalue. All the previous methods can be generalized to systems.

10.8 IBVPs

For the advection equation on a bounded domain we will have an inflow boundary condition but not an outflow boundary condition. To get the right number of equations, we'll need to impose a numerical absorbing/nonreflecting outflow BC, which needs to be chosen carefully to avoid numerical artifacts such as high wave number waves being reflected off the outflow boundary.

11 Mixed Equations

For example, advection-diffusion, advection-reaction, nonlinear hyperbolic like Navier-Stokes and viscous Burgers, dispersive like Korteweg-de Vries and nonlinear Schr odinger. Methods include

- fully coupled method of lines: uses same discretization method for all terms – wasteful for easier, non-stiff terms

- fully coupled Taylor series: first order always works, may not be possible to get second order
- fractional step: split each step into two or more that handle each term individually – usually just first order
- implicit-explicit: time discretizations that are implicit in some terms and explicit in others
- exponential time differencing: separate out the linear part to handle with a matrix exponential, and handle the hopefully non-stiff nonlinear part with an explicit method