Samenvatting

Scientific Computing - Collegejaar 2015-2016

College aantekeningen

Disclaimer

Joris van Tatenhove

Samenvatting Scientific Computing 2015/2016

Chapter 1

Simulation:

- Setting up a model (system of diff. eq's)
- Analytical treatment (properties of solution)
- Numerical treatment (discretizing)
- Implementation
- Embedding
- Visualization
- Validation

Errors:

Errors. Absolute $|u - \hat{u}_h|$

Relative $\frac{|u - \hat{u}_h|}{|u|}$

- Modelling error
- Error in data
- Solution errors
- Round-off errors

Chapter 2

- $\sigma(A) = \{\lambda : \lambda \text{ eigenvalue of } A\}$ is the spectrum of A.
- Eigenvalues of a symmetric matrix are real
- A matrix is positive (semi-) definite if
  $\forall u \in \mathbb{R}^n \setminus \{0\}: \; u^T A u > 0 \quad (u^T A u \geq 0)$
- A symmetric positive definite (spd) matrix A has
  $\sigma(A) \subset \mathbb{R}^+$
- $\langle u, v \rangle_A := u^T A v$ is an inner product. If $\langle u, v \rangle_A = 0$, u and v are A-conjugate.
- $\|R\|_p := \max_{\|u\|_p = 1} \|Ru\|_p = \max_{u \in \mathbb{R}^n \setminus \{0\}} \frac{\|Ru\|_p}{\|u\|_p}$
- If A is spd, then $\|A\|_2 = \lambda_{max}(A)$.
- In general, $\|A\|_1 = \max_{1 \leq j \leq n} \sum_{i=1}^{m} |r_{ij}|$ max. abs. column sum

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

  $\|A\|_\infty = \max_{1 \leq i \leq m} \sum_{j=1}^{n} |r_{ij}|$ max. abs. row sum
- The condition number $K_p(A)$ controls Δu in terms of Δf in the system $Au = f$.

  $K_p(A) = \|A\|_p \cdot \|A^{-1}\|_p \Rightarrow K_2(A) = \frac{\sqrt{\lambda_{max}(A^T A)}}{\sqrt{\lambda_{min}(A^T A)}}$

  - A symm: $\frac{|\lambda_{max}(A)|}{|\lambda_{min}(A)|}$
  - A spd: $\frac{\lambda_{max}(A)}{\lambda_{min}(A)}$

- Spectral radius $\rho(A) := \max \{|\lambda_i| : \lambda_i \in \sigma(A)\}$

  $\rho(A) \leq \|A\|$: take a random $\lambda \in \sigma(A)$. Then

  $$|\lambda| \|u\| = \|\lambda u\| = \|Au\| \leq \|A\| \cdot \|u\| \Rightarrow |\lambda| \leq \|A\| \quad \square$$

- A matrix A is irreducible if no permutation P exists such that PAP^T is block upper triangular.

- A matrix is an M-matrix iff
  - $a_{ii} > 0 \quad \forall i$
  - $a_{ij} \leq 0 \quad \forall i \neq j$
  - A non-singular
  - $A^{-1} \geq 0$

  Also, the following are sufficient conditions:
  - $a_{ii} > 0 \quad \forall i$
  - $a_{ij} \leq 0 \quad \forall i \neq j$
  - A is irreducible and diagonally dominant

Chapter 3

- 2D Dirichlet boundary: $\frac{1}{h^2}\begin{bmatrix} 0 & -1 & 0 \\ 0 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$, $f^h_{2,j} + \frac{1}{h^2} b^h_{1,j}$

  Corner: $\frac{1}{h^2}\begin{bmatrix} 0 & -1 & 0 \\ 0 & 4 & -1 \\ 0 & 0 & 0 \end{bmatrix}$, $f^h_{2,2} + \frac{1}{h^2} b^h_{1,2} + \frac{1}{h^2} b^h_{2,1}$

Neumann boundary: $\frac{\partial u}{\partial n} = g$ on Γ

corner $(1,1)$: $4u_{1,1} - u_{2,1} - u_{0,1} - u_{1,2} - u_{1,0} = h^2 f_{1,1}$

1st order centered difference:

$$\begin{cases} u_{0,1} - u_{2,1} = 2h\, g_{1,1} \\ u_{1,0} - u_{1,2} = 2h g_{1,1} \end{cases}$$

$u_{0,1}$, $u_{1,0}$ ghost points $\Rightarrow$

$$u_{0,1} = 2h g_{1,1} + u_{2,1}$$

$$u_{1,0} = 2h g_{1,1} + u_{1,2}$$

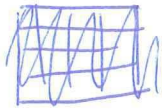
$$\Rightarrow 4u_{1,1} - 2u_{1,2} - 2u_{2,1} = h^2 f_{1,1} + 4h g_{1,1}$$

$$\Rightarrow \frac{1}{h^2}\begin{bmatrix} 0 & -2 & 0 \\ 0 & 4 & -2 \\ 0 & 0 & 0 \end{bmatrix}, \quad h f_{1,1} + \frac{4}{h} g_{1,1}$$

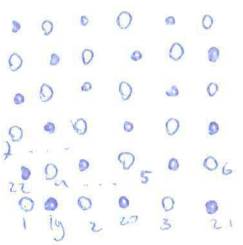
- h-scaled norm:

$$\|u(x) - u^h\|_{h,2} = \sqrt{h \sum_{i=0}^{N} \left(u(x_i) - u^h_i\right)^2} = O(h^2)$$

- Alternative Grid Orderings



Red/Black



Diagonal

- The resulting matrix is sparse, symmetric, irreducably diagonal dominant (so pd by Gershgorin → spd) so it is also an M-matrix

The matrix with $\frac{1}{h^2}[-1\ 2\ -1]$ as stencil has eigenvalues

$$\lambda_k^h = \frac{2}{h^2}[1-\cos(\pi h k)] \qquad k=1,\dots,N-1$$

eigenvectors

$$v_k^h = \frac{1}{\sqrt{N-1}}\begin{pmatrix}\sin(\pi k x_1)\\ \vdots \\ \sin(\pi k x_{N-1})\end{pmatrix} = \frac{1}{\sqrt{N-1}}\begin{pmatrix}\sin(\pi k h)\\ \vdots \\ \sin(\pi k (N-1) h)\end{pmatrix} \qquad k=1,\dots,N-1$$

Proof: • for nodes i not connected to the boundary:

Let $k \in \{1,\dots,N-1\}$.

$$[A^h v_k^h]_i = \sum_{\alpha=1}^{N-1} a_{i\alpha}[v_k^h]_\alpha$$

$$= \frac{1}{h^2}\frac{1}{\sqrt{N-1}}\left[-\sin(\pi(i-1)hk) + 2\sin(\pi i h k) - \sin(\pi(i+1)hk)\right]$$

As $\sin(\alpha+\beta) + \sin(\alpha-\beta) = 2\sin(\alpha)\cos(\beta)$: ($\alpha = \pi i h k$, $\beta = \pi h k$)

$$= \frac{1}{h^2}\frac{1}{\sqrt{N-1}}\left[2\sin(\pi i h k) - 2\sin(\pi i h k)\cos(\pi h k)\right]$$

$$= \frac{2}{h^2}\frac{1}{\sqrt{N-1}}\left[1-\cos(\pi h k)\right]\sin(\pi i h k)$$

$$= \frac{2}{h^2}\left[1-\cos(\pi h k)\right][v_k^h]_i$$

$$= \lambda_k^h [v_k^h]_i \qquad (2 \le i \le N-2)$$

• for $i=1$:

Let $k \in \{1,\dots,N-1\}$

$$[A^h v_k^h]_i = \sum_{\alpha=1}^{N-1} a_{1\alpha}[v_k^h]_\alpha$$

$$= \frac{1}{h^2}\frac{1}{\sqrt{N-1}}\left[2\sin(\pi h k) - \sin(\pi \cdot 2 h k)\right]$$

($\alpha = \beta = \pi h k$)

$$= \frac{1}{h^2}\frac{1}{\sqrt{N-1}}\left[2\sin(\pi h k) - 2\sin(\pi h k)\cos(\pi h k)\right]$$

$$= \frac{2}{h^2}\left[1-\cos(\pi h k)\right](v_h^k)_1$$

• for $i = N-1$

$$[A^h v_k^h]_{N-1} = \sum_{\alpha=1}^{N-1} a_{N-1,\alpha}[v_k^h]_{N-1}$$

$$= \frac{1}{h^2}\frac{1}{\sqrt{N-1}}\left[\sin(\pi(N-2)hk) + 2\sin(\pi(N-1)hk)\right]$$

$\alpha = (N+1)hk$, $\beta = \pi hk$

$$= \frac{4}{h^2}\frac{1}{\sqrt{N+1}}\left[2\sin(\pi(N-1)hk\cos(\pi hk) + 2\sin(\pi(N-1)hk)\right]$$

$$= \frac{2}{h^2}\frac{1}{\sqrt{N+1}}\left[1-\cos(\pi hk)\right]\sin(\pi(N-1)hk)$$

$$= \lambda_k^h \left[v_k^h\right]_{N-1}$$

Simarly for $\frac{1}{h^2}\begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix}$ we find

$$\lambda_{k,\ell}^h = \frac{4}{h^2}\left[1 - \frac{1}{2}\cos(\pi hk) - \frac{1}{2}\cos(\pi h\ell)\right]$$

$$v_{k,\ell}^h = \frac{1}{N-1}\begin{pmatrix} \sin(\pi kh)\sin(\pi \ell h) \\ \sin(\pi k(N-1)h)\sin(\pi \ell h) \\ \vdots \\ \sin(\pi k(N-1)h)\sin(\pi \ell(N-1)h) \end{pmatrix}$$

Proof: similar, but

$$\left[A^h v_{k,\ell}^h\right]_{i,j} = \sum_{\alpha,\beta=1}^{(N-1)^2} a_{\alpha\beta,ij}\left(v_{\alpha\beta,k,\ell}^h\right)_{\alpha\beta}$$

Other discretization tips

$$u''(x_i) \approx \frac{-u(x_{i-1}) + 2u(x_i) - u(x_{i+1})}{h^2}$$

approx. of second derivative

So in 2D

$$cu + \frac{\partial^2 u}{\partial x^2} + \varepsilon\frac{\partial^2 u}{\partial y^2} = f(x)$$ gives stencil

$$\frac{1}{h^2}\begin{pmatrix} 0 & -\varepsilon & 0 \\ -1 & 4+ch^2 & -1 \\ 0 & -\varepsilon & 0 \end{pmatrix}$$

Chapter 4

The analysis on stability of a sytem does not take a solution method into account.

Let f be replaced by $f+\Delta f$. Then $\exists \Delta u$:

$$A(u+\Delta u) = f + \Delta f, \text{ where } \Delta u \text{ solves } A\Delta u = \Delta f.$$

Then $$\frac{\|\Delta u\|}{\|u\|} \leq \|A^{-1}\| \cdot \|A\| \underbrace{\frac{\|\Delta f\|}{\|f\|}}_{:=\delta} = K(A) \cdot \delta$$

So if $K(A)$ is large, a small perturbation can lead to a big difference in the solution vector:

$$\begin{pmatrix} 1 & 1 \\ 1 & 0.999 \end{pmatrix} u = \begin{pmatrix} 2 \\ 1.999 \end{pmatrix} \Rightarrow u = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$$\text{---}''\text{---} = \begin{pmatrix} 2 \\ 2 \end{pmatrix} \Rightarrow u = \begin{pmatrix} 2 \\ 0 \end{pmatrix}$$

$$\left(\text{cond}\left(\begin{pmatrix} 1 & 1 \\ 1 & 0.999 \end{pmatrix}\right) = 3998\right)$$

Similarly for perturbations in A. Due to rounding, such errors will occur, so it is important to have a small condition number to guarantee better approximations.

A way to deal with this is by scaling matrix rows such that each row has similar $\|\cdot\|_\infty$-norm.

A direct way of solving $Au=f$ is by writing $A=LU$. L is upper lower triangular and U upper, and this factorization is unique. Instead of solving $Au=f$, we solve first $Ly=f$ and then $Uu=y$ is solved for u. This solve is numerically stable, but the factorization requires some deeper analysis. These solves are easy, as matrices are triangular

The LU-factorization can be computed by simply bringing $A^{n\times n}$ into echelon form (in Dutch: "vegen")
Doing this requires $n-1$ steps. The $n-1$ steps form an L, the adjusted A is the required U. Formally, we have

$$M_{n-1}\, M_{n-2} \cdots M_1\, A = U \Rightarrow A = \underbrace{(M_{n-1} \cdots M_1)^{-1}}_{L} U,$$

where $M_k = I - \alpha_k e_k^T = \begin{pmatrix} I & & \\ & -b_k/a_{k,k}^{k-1} & \\ & -b_{k+1}/a_{k+1,k+1}^{k-1} & I \end{pmatrix}$ where $\alpha = (0,0,\dots,0,\ b_k/a_{k,k}^{k-1},\ \dots)$

and $a_{i,j}^{k-1}$ represents the element on index i,j of the matrix A after $k-1$ steps.

The inverse of $M_k = I - \alpha_k e_k^T$ is equal to

$$M_k^{-1} = I + \alpha_k e_k^T$$

Note that this cannot deal with zeroes on the diagonal.

Another way to prevent the errors having too much impact on the system is by applying partial pivoting instead. Instead of just subtracting rows from one another, we first swap two rows such that the largest element in that column will be used as a pivot, so $b_k / a_{k,k} \leq 1$.

Then $M_{n-1} P_{n-1} \cdots M_1 P_1 A = U \Rightarrow PA = LU$

where $P = P_{n-1} \cdots P_1$

So we require more flops, but the system is now stable! This is called <u>partial pivoting</u>, and can deal with zeroes.

One can go a step further and also allow column swaps as well. In we have in step k:

$$A^{k-1} = \begin{matrix} k-1 \\ n-k+1 \end{matrix} \begin{pmatrix} A_{11}^{k-1} & A_{12}^{k-1} \\ \underbrace{0}_{k-1} & \underbrace{A_{22}^{k-1}}_{n-k+1} \end{pmatrix}$$, where we divided A into three blocks

Note that A_{11}^{k-1} is upper triangular.

<u>Complete pivoting</u> swaps rows and columns such that the largest element in the block A_{22}^{k-1} is now on element k,k.

Special cases

- It can be proven using induction that diagonally dominant matrices already have the pivots in the right place, so normal Gaussian elimination is sufficient.
- If A is spd, the LU factorization is the same as the Cholesky decomposition $A = CC^T$, so we can decompose and store the matrices using half the time / memory / flops.

- For band matrices, normal Gaussian elimination guarantees that the matrices L and U inherit the band width q and p respectively.

  Partial pivoting does not have this property, and thus should be avoided when possible.

- For tridiagonal systems, a direct method is possible: Let $A = LDL^T$ and solve

$$a_{11} = d_{11}$$
$$a_{k,k-1} = e_{k-1} d_{k-1} \quad \text{for} \quad 2 \le k \le n$$
$$a_{kk} = d_k + e_{k-1}^2 d_{k-1} = a_{kk} - e_{k-1} a_{kk-1} \quad \text{for} \quad 2 \le k \le n$$

- Smart ways exist to store general sparse matrices, such as the Yale format:

$$\begin{pmatrix} 10 & 20 & 0 & 0 & 0 & 0 \\ 0 & 30 & 0 & 40 & 0 & 0 \\ 0 & 0 & 50 & 60 & 70 & 0 \\ 0 & 0 & 0 & 0 & 0 & 80 \end{pmatrix}$$

(elements
$AA = (10\ 20\ \dots\ 80)$
row switches
$IA = (1\ 3\ 5\ 8)$
=> spots in row
$JA = (1\ 2\ 2\ 4\ 3\ 4\ 5\ 6)$

However, these LU methods are slow on large (sparse) matrices A

Chapter 5

Basic iterative methods start with an initial solution u^0 and keep improving that solution by iterating a certain process

BIMs themselves are not fast enough for real life applications, but they do serve as the building blocks for more powerful methods

The error after iteration k is defined as $e^k = u^* - u^k$

The residual " " $r^k = f - Au^k$

Note that knowing e^k requires u^* and that is very hard. Also, we know that $Ae^k = r^k$

A splitting of A are two matrices M, N such that $A = M - N$

$Au = f$ turns into $Mu = Nu + f \Rightarrow u^{k+1} = M^{-1}Nu^k + M^{-1}f$

$= u^k + M^{-1}r^k$

Then:

$e^{k+1} = u^* - u^{k+1} = \ldots e^k - M^{-1}Ae^k = \underbrace{(I - M^{-1}A)}_{B,\ \text{error propagation matrix, iteration matrix}} e^k$

Similarly, $r^{k+1} = \ldots = (I - AM^{-1})r^k$

Two popular BIMs are Jacobi and Gauss-Seidel.

Jacobi

$A = D - E - F$

$M_{JAC} = D, \quad N_{JAC} = E + F, \quad B_{JAC} = L + U$

An update $u_i^{k+1} = \left[f_i - \sum_{j=1, j \neq i} a_{ij} u_j^k \right] / a_{ii} \quad \forall i = 1, \ldots, n$

is independent of other n elements in u^{k+1} so they can be calculated in a parallel fashion. (in other words, the parallel complexity is n)

Gauss-Seidel

$M_{GS} = D - E, \quad N_{GS} = F, \quad B_{GS} = (I - L)^{-1} U$

An update $u_i^{k+1} = \left[f_i - \sum_{j=1}^{i-1} a_{ij} u_j^{k+1} - \sum_{j=i+1}^{n} a_{ij} u_j^k \right] / a_{ii}$

takes elements of the same iteration into account, and as such can converge more quickly than Jacobi. (parallel complexity: 1)

By changing the ordering to red/black or diagonal, the parallel complexity can be increased.

By calculating entire blocks of elements at once by means of dividing A into blocks and partitioning u and f likewise, solving the subsystems independently can also speed up the process.

Damped Jacobi with parameter ω is defined as

$$u^{k+1} = \omega u^{k+1,JAC} + (1-\omega) u^k$$

This is the same as Richardson ($M = \tau I$, $N = \tau I - A$) on the scaled system $D^{-1}Au = D^{-1}f$ with $\tau = \omega^{-1}$.

Dampening Gauss-Seidel results in a successive overrelaxation method:

$$u_i^{k+1} = (1-\omega) u_i^k + \omega u_i^{k+1,GS}$$ (note that it's defined componentwise)

If A is spd, it is useful to have an spd splitting as well. Jacobi is spd, but for GS, we need to set $M_1 = \frac{1}{\omega}D - E$, $M_2 = \frac{1}{\omega}D - F$, called Symmetric Successive overrelaxation (SSOR(ω))

Note that $e^k = Be^{k-1} = \dots = B^k e^0$. That means that

$$\rho(B) < 1 \iff \{u^k\}_{k=1}^{\infty} \text{ converges.}$$

If $\rho(B) \ll 1$, convergence is fast.

Any matrix norm is a bound for its ρ, so we need that

$\|B\|_1, \|B\|_2, \|B\|_\infty < 1$. For Jacobi:

$$\|B_k\|_1 = \|L+U\|_1 = \max_{1 \le j \le n} \sum_{i=1, i\neq j}^{n} \frac{|a_{ij}|}{|a_{ii}|} < 1 \quad \text{(max row sum)}$$

$\|B\|_\infty$ . . . . . (max column sum)

$$\|B\|_2 = \dots = \sum_{i,j=1, i\neq j}^{n} \left(\frac{a_{ij}}{a_{ii}}\right)^2 \quad \text{(square sum criterium)}$$

A splitting $A = M - N$ is regular iff M non-singular, $M^{-1} \ge 0$, $N \ge 0$

If $M-N$ is a regular splitting, then

$(\rho(B) =) \rho(M^{-1}N) < 1 \iff A$ is non-singular and A^{-1} nonnegative.

If A is strongly diagonal row dominant, then JAC and GS converge. 7/12

For SOR(ω), we have that $\rho(B_{SOR(\omega)}) \geq |1-\omega|$, so we need to have $0 < \omega < 2$. If A is spd, $0 < \omega < 2$ is also sufficient for guaranteeing convergence.

The rate of convergence / reduction rate is $\mu \in (0,1)$ s.th.

$$\|u^k - u^{k+1}\| \leq \mu \|u^{k-1} - u^{k-2}\| \quad \forall k.$$

For this μ, we find $\|e^k\|_2 \leq \frac{\mu^k}{1-\mu} \|e^0\|_2$

$$\boxed{\rho(B) \approx \mu}$$

So if $\rho(B) < 1$, we have convergence. If $\rho(B) \ll 1$, we have fast convergence

BIMs am such as Jac, GS and Jac(ω) have $\rho(B) \approx 1$, and

$$\rho(B) \to 1 \text{ as } h \to 0 \quad (\rho(B) = 1 - O(h^2))$$

SSOR(ω) has $\rho(B) = 1 - O(h)$, which is better.

A good choice of u^0 can be made by solving the model on a coarser mesh.
The process can be monitored by tracking $\|r^k\|_2$
A good stopping criterium takes into account the scale and does not depend on r^0: $\frac{\|r^k\|}{\|f\|} \leq \varepsilon$.

Building better methods is based on:

- Splitting
- Defect correction
- Preconditioning

Chapter 6

BIMs can deal with problems with low eigenvalues, as they will converge quickly. These eigenvalues (close to 0) have high frequency eigenmodes.
The eigenvalues close to 1 have slowly varying eigenmodes. That means they can be transformed to a smaller grid and then transformed back, without loss of information.
The starting vector is a combination of these modes. BIMs can act as a smoother for mutigrid methods to do their magic.
A basic multigrid algorithm estimates the error and looks like this:

(pre smoothing) $u^{i+1/3} = S_h^{\alpha} u_h^i + s_h$

$r_h = b_h - A_h u_h^{i+1/3}$

$r_{2h} = I_h^{2h} r_h$

$e_{2h} = (A_{2h})^{-1} r_{2h}$

$e_h = I_{2h}^h e_{2h}$

$u_h^{i+2/3} = u^{i+1/3} + e_h$

(post smoothing) $u_h^i = S_h u_h^{i+2/3} + s_h$

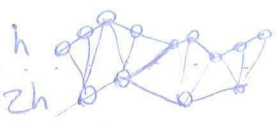
I_h^{2h} transforms a vector in $\mathbb{R}^{n-1} \to \mathbb{R}^{n/2-1}$

I_{2h}^h transforms 'back' into $\mathbb{R}^{n-1}$

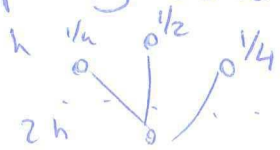
So we estimate the error in $\mathbb{R}^h$ by calculating the exact error on a coarser grid and bringing that to a fine grid.

Smoothers S can be any BIM iteration matrix.

I_h^{2h} makes each element a weighted average of corresponding elements in the finer grid.



1D:



$\frac{1}{4}$ $\frac{1}{2}$ $\frac{1}{4}$

2D: full weighting

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

half weighting

$$\frac{1}{6}\begin{bmatrix} 0 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

I_{2h}^{h} brings a smaller vector to a larger grid size:

1D:

$$\begin{pmatrix} 1/2 & 0 & 0 \\ 1 & 1/2 & 0 \\ 0 & 1 & 0 \\ 0 & 1/2 & 1/2 \\ 0 & 0 & 1 \\ 0 & 0 & 1/2 \end{pmatrix}$$

2D $\frac{1}{4}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$ (full weighting)

A_{2h} can either be constructed like A_h but in fewer grid points, or be created by the Galerkin coarsening:

$$A_{2h} = I_h^{2h} A_h I_{2h}^{h}$$

The algorithm can be improved: multiple coarsening and prolongation steps can be taken. • presmooth o postsmooth x sol \ restriction / prolongation

(3 grid) V-cycle W-cycle (4-grid) F-cycle

Chapter 7

The next methods are based on the Krylov subspace:

$$K^i(A; r_0) = \text{span}\{r_0, Ar_0, \dots, A^{i-1} r_0\}$$

The first method we'll discuss is the conjugate gradient method. This method is based on the Chebyshev method, and computes u^i such that $\|u - u^i\|_A = \min_{y \in K^i(A, r_0)} \|u - y\|_A$

Theoretically, this has some nice properties:

p is search direction
r residual

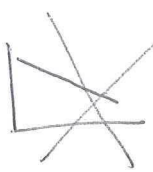
1) • $\text{span}\{p^1, \dots, p^k\} = \text{span}\{r^0, \dots, r^{k-1}\} = K^k(A, r^0)$

2) • $(r^j)^T r^i = 0 \quad 1 \le i < j \le k$
   • $(r^j)^T p^i = 0 \quad 1 \le i \le j \le k$
   • $(p^j)^T A p^i = 0 \quad 1 \le i < j \le k$

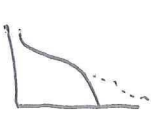
• 1, 2: $r^0, \dots r^{k-1}$ is an orthogonal basis for $K^k(A, r^0)$.
• CG is a finite method: $K^N(A, r^0) = \mathbb{R}^N \Rightarrow u^i = u^*$. In practice, this is not reached due to rounding errors, or N is too large to run N iterations.

The rate of convergence $\mu \le \dfrac{\sqrt{K_2(A)} - 1}{\sqrt{K_2(A)} + 1}$

In practice, we have superlinear convergence:



This happens because eigenvectors belonging to big eigenvalues (almost) enter the Krylov subspace, and thus no longer influence the convergence.



CG can only be applied to spd matrices A

By preconditioning the CG method, we can speed up the process: Instead of solving $Au = b$, we solve $M^{-1}Au = M^{-1}b$ instead. We need to find an M^{-1} such that:

- the eigenvalues of $M^{-1}A$ are clustered around 1
- $M^{-1}b$ can be obtained easily.

The first condition ensures that $\kappa_2(A) \geq \kappa_2(M^{-1}A)$, so it converges faster.

You can also write $(P^{-1}AP^{-T})(P^{T}u) = P^{-1}b$, and let $M = PP^T$.

Some options are:

- P is diagonal with $p_{ii} = \sqrt{a_{ii}}$ (Diagonal scaling)
- $M^{-1} = M_{GS}^{-1}$ or M_{JAC}^{-1} (BIM)
- Incomplete Cholesky decomposition: ICGC(m)

The normal Cholesky decomposition always exists for spd matrices: $A = LL^T$.

The incomplete forces 0 elements of A to be 0 elements of L as well, saving the sparseness of the system.

Cholesky decomposition is very stable.

ICGC(m) allows for m more m more bands around the diagonal to become non-zero.

MICGC forces equal rowsums as well, performing better if u and/or b are slowly varying vectors.

For general matrices, it is impossible to obtain a method that has the following three properties

- ~~[illegible]~~ $x^i \in K^i(A; r^0)$
- Optimality (wrt $\|\cdot\|_A$ norm)
- Short recurrencies.

But for each combination of 2 properties, there are methods.

Opt + short recurrencies:

CGNR (Conjugate Gradient normal residual)

Apply CG to $\underbrace{A^TA}_{spd}x = A^Tb$

However, $K_2(A^TA) = K_2(A)^2$, which is usually large, so convergence is slow. Also A^Tv is not always easily obtained.

A reliable improvement is Least Squares QR (LSQR), which solves $\begin{pmatrix} I & A \\ A^T & 0 \end{pmatrix}\begin{pmatrix} r \\ x \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix}$

Krglov + short recurrencies

Bi-CG type methods have two series of residuals:

$r^0, \dots, r^{k-1}$ form a basis for $K^k(A; r^0)$

$\hat{r}^0, \dots, \hat{r}^{k-1}$ " " $K^k(A, \hat{r}^0)$

~~The~~

r^k and $\hat{r}^k$ are orthogonal bases for $K^k(A; r^0)$ and $K^k(A, \hat{r}^0)$ respectively.

This method has bad convergence if A is far from spd ($A \approx -A^T$), doesn't guarantee optimality and requires A^T. It can also have unlucky (near) breakdowns.

There are two improvements:

- CGS (CG squared) is based on the fact that

$$(r^j, \hat{r}^i) = (P_i(A) P_j(A) r^0, \hat{r}^0) = 0 \quad \text{for } i<j \text{ and}$$

abuses this polynomial to get rid of explicit $\hat{r}^j$s and A^T. It converges faster than Bi CG usually, but shows very irregular convergence behavior.

- Bi CG STAB modifies the polynomials as well

Krylov + optimality

=

The GMRES type methods require a lot of memory. To deal with this, restart or truncation (only using the last ℓ vectors, for some ℓ) is required.

GMRES methods (generalized minimized residual) uses a different orthogonal basis for the Krylov subspace. To ensure orthogonality, all earlier vectors must be stored. Lucky breakdowns can occur. Due to reset only local minimization can be guaranteed and superlinear convergence behavior is lost.

GCR (Generalized conjugate residual) converges similarly to GMRES, but in some cases it breaks down. However, GCR allows for truncation, so if superlinear convergence is important, GCR is better than GMRES. In general, truncation is better than restarting.

Hybrid methods do exist. They mainly combine BiCG and GMRES type methods.

One method, IDR (Induced Dimension Reduction) has later been overshadowed by CGS and BiCGSTAB.

GMRESR (GMRES recursive). This method uses a GCR approach on search directions calculated by GMRES.

The search directions v^i are stored using a little trick:

GMRES(3): $v_1\ v_2\ v_3$ restart $v_1\ v_2\ v_3$ restart ...

GCR, trunc 3: $v_1\ v_2\ v_3\ [v_4\ v_5\ v_6]$ ...

GMRESR with GMRES(3) as inner loop: $\hat{v}_1\ \hat{v}_2\ \hat{v}_3 \rightarrow v_1$ $\hat{v}_1\ \hat{v}_2\ \hat{v}_3 \rightarrow v_2$ ...

So vital information does not get lost.

For GMRESR, truncation and restarting is possible as well.

Deciding which method to use depends on the expected amount of iterations (m_g) and the ratio

$$f = \frac{\text{CPU time for prec. matvec}}{\text{CPU time for vector update}}$$

Big m_g, small f: BiCGSTAB

Small m_g, big f: GMRES

In between: GMRESR

These methods can be preconditioned as well.

BLANK PAGE

BLANK PAGE

Chapter 8

Due to lack of time, we only need to know the Power method.

Wave Eigenvalues give a lot of practical and theoretical information, so it is important to be able to calculate them

The power method can calculate (multiple) extreme eigenvalues.

Let $q_0 \in \mathbb{C}$. Repeat the following process:

$$z_k = A q_{k-1}$$
$$q_k = z_k / \|z_k\|_2$$
$$\lambda^{(k)} = q_{k-1}^T z_k$$

For example, if the matrix A is $A = \begin{pmatrix} 10 & 0 \\ 0 & 2 \end{pmatrix}$ and $q_0 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$

then $z_1 = \begin{pmatrix} 10 & 0 \\ 0 & 2 \end{pmatrix}\begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 10 \\ 2 \end{pmatrix}$, $q_1 \approx \begin{pmatrix} 0.9806 \\ 0.1961 \end{pmatrix}$, $\lambda^{(1)} = (1\ 1)\begin{pmatrix} 10 \\ 2 \end{pmatrix} = 12$

$z_2 = \begin{pmatrix} 10 & 0 \\ 0 & 2 \end{pmatrix}\begin{pmatrix} 0.9806 \\ 0.1961 \end{pmatrix} = \begin{pmatrix} 9.8058 \\ 0.3922 \end{pmatrix}$, $q_2 \approx$ $\lambda^{(2)} = 9.6923$

$\lambda^{(3)} = 9.9872$

$\lambda^{(4)} = 9.9997$

$\lambda^{(5)} = 10.000$

This converges to the largest eigenvalue, as

(let $|\lambda_1| \geq |\lambda_2| \geq \dots \geq |\lambda_n|$)

$A^k q_0 = A^k(c_1 v_1 + \dots + c_n v_n) = c_1 \lambda_1^k v_1 + \dots + c_n \lambda_n^k v_n$, this will

so λ_1 will have the biggest impact.

12/12

Note that this fails if $c_1 = 0$ for your choice of q_0!

The convergence speed depends on $\left|\frac{\lambda_2}{\lambda_1}\right|$, and so applying the power method to $A - cI$ depends on $\left|\frac{\lambda_2^* - c}{\lambda_1^* - c}\right|$. (* means new largest eigenvalues) So shifting can increase convergence speed.

Taking $c \approx \lambda_1$ means that $\lambda_n - \lambda_1$ is now the largest eigenvalue so we can also calculate the smallest eigenvalue. (*)

If $|\lambda_1| = |\lambda_2|$, a fix is possible.

~~Taking A^{-1} instead of A ha~~

A^{-1} has eigenvalues $\frac{1}{\lambda_i}$, so applying the power method to A^{-1} will converge to $\frac{1}{\lambda_n}$.

Small eigenvalues are generally close to each other, so taking A^{-1} instead of the shift (*) will probably lead to a faster convergence.

To compute $z_k = A^{-1} q_{k-1}$, one solves $A z_k = q_{k-1}$.

To calculate the largest p eigenvalues, let $Q_0 \in \mathbb{C}^{n \times p}$ an orthogonal matrix and do

- $Z_k = A Q_{k-1}$
- orthonormalize columns of Z_k s.th. $Q_k R_k = Z_k$, $R_k \in \mathbb{R}^{k \times k}$ upp. triang. and $Q_k^T Q_k = I$