Linear Solvers

Andrew Hazel

Introduction

- Thus far we have talked about the formulation and discretisation of physical problems ...
- ... and stopped when we got to a discrete linear system of equations.

Introduction

- Thus far we have talked about the formulation and discretisation of physical problems ...
- ... and stopped when we got to a discrete linear system of equations.
- In explicit timestepping methods, may not have to solve a linear system (although for Navier–Stokes one typically solves a pressure-Poisson equation).

Introduction

- Thus far we have talked about the formulation and discretisation of physical problems ...
- ... and stopped when we got to a discrete linear system of equations.
- In explicit timestepping methods, may not have to solve a linear system (although for Navier–Stokes one typically solves a pressure-Poisson equation).
- Solve these discrete systems (often repeatedly) to find the (discrete) solution to the problem of interest:
- Find $\mathbf{x} \in \mathbb{R}^n$, where

$$A\,\mathbf{x} = \mathbf{b},$$

A is a known $n \times n$ matrix and $\mathbf{b} \in \mathbb{R}^n$ is a known right-hand side vector.

Direct methods

- Direct methods find solutions of the linear system by determining explicit formulæ for the unknowns.
- The conceptually simplest direct method is Gaussian elimination, which should be familiar.

$$\begin{pmatrix} 1 & 4 & 5 \\ 2 & 7 & 6 \\ 3 & 3 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

Direct methods

- Direct methods find solutions of the linear system by determining explicit formulæ for the unknowns.
- The conceptually simplest direct method is Gaussian elimination, which should be familiar.

$$\begin{pmatrix} 1 & 4 & 5 \\ 0 & -1 & -4 \\ 0 & -9 & -14 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 1 \\ -2 \\ -3 \end{pmatrix} \quad \begin{matrix} \\ -2 \times A_{1i} \\ -3 \times A_{1i} \end{matrix}$$

Direct methods

- Direct methods find solutions of the linear system by determining explicit formulæ for the unknowns.
- The conceptually simplest direct method is Gaussian elimination, which should be familiar.

$$\begin{pmatrix} 1 & 4 & 5 \\ 0 & -1 & -4 \\ 0 & 0 & 22 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 1 \\ -2 \\ 15 \end{pmatrix} \qquad -9 \times B_{2i}$$

Direct methods

- Direct methods find solutions of the linear system by determining explicit formulæ for the unknowns.
- The conceptually simplest direct method is Gaussian elimination, which should be familiar.

$$\begin{pmatrix} 1 & 4 & 5 \\ 0 & -1 & -4 \\ 0 & 0 & 22 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 1 \\ -2 \\ 15 \end{pmatrix} \qquad -9 \times B_{2i}$$

- Once the matrix is upper triangular simple back substitution gives the answer $x = 1/2$, $y = -8/11$, $z = 15/22$.

Direct methods

- Direct methods find solutions of the linear system by determining explicit formulæ for the unknowns.
- The conceptually simplest direct method is Gaussian elimination, which should be familiar.

$$\begin{pmatrix} 1 & 4 & 5 \\ 0 & -1 & -4 \\ 0 & 0 & 22 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 1 \\ -2 \\ 15 \end{pmatrix} \qquad -9 \times B_{2i}$$

- Once the matrix is upper triangular simple back substitution gives the answer $x = 1/2$, $y = -8/11$, $z = 15/22$.
- Typical operation count is $O(n^3)$, which grows dramatically as n increases.

LU Decomposition

- Gaussian elimination is usually implemented via an LU factorisation which "keeps" the factors used in the elimination.

$$\begin{pmatrix} 1 & 4 & 5 \\ 2 & 7 & 6 \\ 3 & 3 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 2 & 9 & 1 \end{pmatrix} \begin{pmatrix} 1 & 4 & 5 \\ 0 & -1 & -4 \\ 0 & 0 & 22 \end{pmatrix}.$$

LU Decomposition

- Gaussian elimination is usually implemented via an LU factorisation which "keeps" the factors used in the elimination.

$$\begin{pmatrix} 1 & 4 & 5 \\ 2 & 7 & 6 \\ 3 & 3 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 2 & 9 & 1 \end{pmatrix} \begin{pmatrix} 1 & 4 & 5 \\ 0 & -1 & -4 \\ 0 & 0 & 22 \end{pmatrix}.$$

- Using the LU decomposition

$$A\,\mathbf{x} = \mathbf{b} \quad \Rightarrow \quad LU\,\mathbf{x} = \mathbf{b}$$

- The solution proceeds via two backsolves:

$$\text{Solve} \quad L\,\mathbf{y} = \mathbf{b} \quad \text{followed by} \quad U\,\mathbf{x} = \mathbf{y}.$$

LU Decomposition

- Gaussian elimination is usually implemented via an LU factorisation which "keeps" the factors used in the elimination.

$$\begin{pmatrix} 1 & 4 & 5 \\ 2 & 7 & 6 \\ 3 & 3 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 2 & 9 & 1 \end{pmatrix} \begin{pmatrix} 1 & 4 & 5 \\ 0 & -1 & -4 \\ 0 & 0 & 22 \end{pmatrix}.$$

- Using the LU decomposition

$$A\,\mathbf{x} = \mathbf{b} \quad \Rightarrow \quad LU\,\mathbf{x} = \mathbf{b}$$

- The solution proceeds via two backsolves:

$$\text{Solve} \quad L\,\mathbf{y} = \mathbf{b} \quad \text{followed by} \quad U\,\mathbf{x} = \mathbf{y}.$$

- Once LU factorisation is known, solution for alternative right-hand sides is quick.
- Operation count is still $O(n^3)$.

Using structure in the matrix

- Finite-element, finite-difference and finite-volume methods typically lead to large, but sparse matrices.
- Spectral or boundary-element methods typically lead to small(er) but dense matrices.

Using structure in the matrix

- Finite-element, finite-difference and finite-volume methods typically lead to large, but sparse matrices.
- Spectral or boundary-element methods typically lead to small(er) but dense matrices.
- Direct solvers can take advantage of sparsity structure.
- Computing LU factors of sparse matrices requires fewer operations (don't need to eliminate all entries in all rows).

Using structure in the matrix

- Finite-element, finite-difference and finite-volume methods typically lead to large, but sparse matrices.
- Spectral or boundary-element methods typically lead to small(er) but dense matrices.
- Direct solvers can take advantage of sparsity structure.
- Computing LU factors of sparse matrices requires fewer operations (don't need to eliminate all entries in all rows).
- SuperLU (`http://eicrd-legacy.lbl.gov/ xiaoye/SuperLU/`) is a direct LU solver designed to take advantage of sparsity.
- Typically analyse phase is run before factorisation and backsubstitution.

Frontal solvers

- Frontal solvers are a variant of direct solvers designed for element-based methods.
- The matrix is never completely assembled.
- Variables are removed as soon as every contributing element has been visited.
- Efficiency is dependent on order in which elements are assembled.
- Very good for long, thin domains ... can extend the domain at very minimal run-time cost.
- Multifrontal methods use several fronts at once
- MUMPS (`http://graal.ens-lyon.fr/MUMPS/`) is a parallel multi-frontal solver.

Direct solvers: summary

- Usually very robust.
- Efficient for matrices with particular sparsity structure.
- Expensive for large matrices.
- Memory is always an ultimate bottleneck, unless information is written to disk, which can be slow!
- Main problem is the fill-in of non-zero entries in the factorisation phase.
- SuperLU_Dist and MUMPS work well on multi-core processors.
- Direct methods are very competitive for two-dimensional and axi-symmetric problems and work well for three-dimensional methods until you run out of memory.

Iterative Methods

- Iterative methods are based on repeated updates to an initial guess, $\mathbf{x}^0$

$$\mathbf{x}^{k+1} = \mathbf{F}(\mathbf{x}^k)$$

- These methods do not necessarily require the storage of the matrix.
- Much cheaper in terms of memory usage than direct methods.
- If application of $\mathbf{F}$ is cheap *and* method converges in a few iterations, can be much cheaper in computation time.
- Aim is to find optimal $O(n)$ solver.

Iterative Methods

- Iterative methods are based on repeated updates to an initial guess, $\mathbf{x}^0$

$$\mathbf{x}^{k+1} = \mathbf{F}(\mathbf{x}^k)$$

- These methods do not necessarily require the storage of the matrix.
- Much cheaper in terms of memory usage than direct methods.
- If application of $\mathbf{F}$ is cheap *and* method converges in a few iterations, can be much cheaper in computation time.
- Aim is to find optimal $O(n)$ solver.
- **Questions:**
  - How do we choose $\mathbf{F}$?
  - How quickly does the method converge?

Conjugate Gradient Method

- Applies to symmetric, positive definite matrices A.
- Identify solution of linear system $A\mathbf{x} = \mathbf{b}$ with minimum of

$$\Phi(\mathbf{x}) = \frac{1}{2} x_i A_{ij} x_j - x_i b_i.$$

- Minimum occurs when

$$\frac{1}{2}\delta_{ik} A_{ij} x_j + \frac{1}{2} x_i A_{ij} \delta_{jk} - b_i \delta_{ik} = 0 \quad \Rightarrow \quad \frac{1}{2}(A + A^T)\mathbf{x} = \mathbf{b}$$

- A symmetric $\Rightarrow A = A^T$, so minimum occurs when $A\mathbf{x} = \mathbf{b}$.
- Want iteration that brings us close to the minimum as rapidly as possible.

Conjugate Gradient Method

- Consider an iterative improvement to the current solution along a line

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha \mathbf{p}^k,$$

where $\mathbf{p}^k$ is a "search" direction.

$$\Phi(\mathbf{x}^{k+1}) = \Phi(\mathbf{x}^k + \alpha \mathbf{p}^k) = \frac{1}{2}(x_i^k + \alpha p_i^k) A_{ij} (x_j^k + \alpha p_j^k) - (x_i^k + \alpha p_i^k) b_i$$

Conjugate Gradient Method

- Consider an iterative improvement to the current solution along a line

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha \mathbf{p}^k,$$

where $\mathbf{p}^k$ is a "search" direction.

$$\Phi(\mathbf{x}^{k+1}) = \Phi(\mathbf{x}^k + \alpha \mathbf{p}^k) = \frac{1}{2}(x_i^k + \alpha p_i^k) A_{ij} (x_j^k + \alpha p_j^k) - (x_i^k + \alpha p_i^k) b_i$$

$$= \frac{1}{2} x_i^k A_{ij} x_j^k + \frac{1}{2}\alpha \left(x_i^k A_{ij} p_j^k + p_i^k A_{ij} x_j^k \right) + \frac{1}{2}\alpha^2 p_i^k A_{ij} p_j^k - x_i^k b_i - \alpha p_i^k b_i$$

Conjugate Gradient Method

- Consider an iterative improvement to the current solution along a line

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha \mathbf{p}^k,$$

where $\mathbf{p}^k$ is a "search" direction.

$$\Phi(\mathbf{x}^{k+1}) = \Phi(\mathbf{x}^k + \alpha \mathbf{p}^k) = \frac{1}{2}(x_i^k + \alpha p_i^k) A_{ij} (x_j^k + \alpha p_j^k) - (x_i^k + \alpha p_i^k) b_i$$

$$= \Phi(\mathbf{x}^n) + \alpha p_i^k (A_{ij} x_j^k - b_i) + \frac{1}{2} \alpha^2 p_i^k A_{ij} p_j^k.$$

Conjugate Gradient Method

- Consider an iterative improvement to the current solution along a line

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha \mathbf{p}^k,$$

where $\mathbf{p}^k$ is a "search" direction.

$$\Phi(\mathbf{x}^{k+1}) = \Phi(\mathbf{x}^n) + \alpha p_i^k (A_{ij} x_j^k - b_i) + \frac{1}{2}\alpha^2 p_i^k A_{ij} p_j^k.$$

- Minimised when

$$\alpha = \frac{p_i^k (b_i - A_{ij} x_j^k)}{p_m^k A_{mn} p_n^k} = \frac{p_i^k r_i^k}{p_m^k A_{mn} p_n^k}$$

Conjugate Gradient Method

- Consider an iterative improvement to the current solution along a line

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha \mathbf{p}^k,$$

where $\mathbf{p}^k$ is a "search" direction.

$$\Phi(\mathbf{x}^{k+1}) = \Phi(\mathbf{x}^n) + \alpha p_i^k (A_{ij} x_j^k - b_i) + \frac{1}{2}\alpha^2 p_i^k A_{ij} p_j^k.$$

- Minimised when

$$\alpha = \frac{p_i^k (b_i - A_{ij} x_j^k)}{p_m^k A_{mn} p_n^k} = \frac{p_i^k r_i^k}{p_m^k A_{mn} p_n^k}$$

- Given $\mathbf{p}^k$ we can find α that minimises the next iterate.
- **Question:** How do we choose the set $\{\mathbf{p}^k\}$?

Conjugate Gradient Method

- The obvious choice for the search direction is the direction in which the solution decreases most rapidly — the steepest gradient $-\frac{\partial \Phi}{\partial x_i} = b_i - A_{ij}x_j^k = r_i^k$.
- Better to avoid repeats by choosing each search direction to be "different" from previous ones.
- If the search directions are linearly independent they form a basis of $\mathbb{R}^k$ and convergence is guaranteed in at most n steps.
- **Question:** What's the best basis to choose?

Conjugate Gradient Method

- Now $\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha_k \mathbf{p}^k = \mathbf{x}^0 + \sum_{m=1}^{k} \alpha_m \mathbf{p}^m$ and

$$\Phi(\mathbf{x}^{k+1}) = \frac{1}{2}(\mathbf{x}^0 + \sum_{m=1}^{k} \alpha_m \mathbf{p}^m)^T A(\mathbf{x}^0 + \sum_{m=1}^{k} \alpha_m \mathbf{p}^m)$$

$$-(\mathbf{x}^0 + \sum_{m=1}^{k} \alpha_m \mathbf{p}^m)^T \mathbf{b}.$$

Conjugate Gradient Method

- Now $\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha_k \mathbf{p}^k = \mathbf{x}^0 + \sum_{m=1}^{k} \alpha_m \mathbf{p}^m$ and

$$\Phi(\mathbf{x}^{k+1}) = \frac{1}{2}(\mathbf{x}^0 + \sum_{m=1}^{k} \alpha_m \mathbf{p}^m)^T A(\mathbf{x}^0 + \sum_{m=1}^{k} \alpha_m \mathbf{p}^m)$$

$$-(\mathbf{x}^0 + \sum_{m=1}^{k} \alpha_m \mathbf{p}^m)^T \mathbf{b}.$$

- Choose the search directions to be A-conjugate $p_i^k A_{ij} p_j^m = 0$ when $n \neq m$ ($\mathbf{p}^k$ and $A\mathbf{p}^m$ are orthogonal)
  $\Rightarrow$ All cross terms in the vector-matrix-vector product vanish.

$$\Phi(\mathbf{x}^{k+1}) = \Phi(\mathbf{x}^0) + \sum_{m=1}^{k} \left[\alpha_m \mathbf{p}^m \left(A\mathbf{x}^0 - \mathbf{b} \right) + \frac{1}{2} \alpha_m^2 \mathbf{p}^m A \mathbf{p}^m \right]$$

Conjugate Gradient Method

- Now $\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha_k \mathbf{p}^k = \mathbf{x}^0 + \sum_{m=1}^{k} \alpha_m \mathbf{p}^m$ and

$$\Phi(\mathbf{x}^{k+1}) = \frac{1}{2}(\mathbf{x}^0 + \sum_{m=1}^{k} \alpha_m \mathbf{p}^m)^T A(\mathbf{x}^0 + \sum_{m=1}^{k} \alpha_m \mathbf{p}^m)$$

$$-(\mathbf{x}^0 + \sum_{m=1}^{k} \alpha_m \mathbf{p}^m)^T \mathbf{b}.$$

- Choose the search directions to be A-conjugate $p_i^k A_{ij} p_j^m = 0$ when $n \neq m$ ($\mathbf{p}^k$ and $A\mathbf{p}^m$ are orthogonal)
  $\Rightarrow$ All cross terms in the vector-matrix-vector product vanish.

$$\Phi(\mathbf{x}^{k+1}) = \Phi(\mathbf{x}^0) + \sum_{m=1}^{k} \left[\alpha_m \mathbf{p}^m \left(A\mathbf{x}^0 - \mathbf{b} \right) + \frac{1}{2} \alpha_m^2 \mathbf{p}^m A \mathbf{p}^m \right]$$

- Hence minimisation in each direction decouples from others.

Conjugate Gradient Method

- We construct $\{\mathbf{p}^k\}$ by starting with the steepest gradient

$$\mathbf{p}^0 = \mathbf{r}^0 = \mathbf{b} - A\mathbf{x}^0.$$

- It can be shown (*e. g.* Golub & Van Loan) that

$$\mathbf{p}^{k+1} = \mathbf{r}^{k+1} + \beta_k \mathbf{p}^k.$$

and enforcing A-conjugacy gives

$$\beta_k = -\frac{p_i^k A_{ij} r_j^{k+1}}{p_m^k A_{mn} p_n^k} = \frac{r_i^{k+1} r_i^{k+1}}{r_j^k r_j^k},$$

Conjugate Gradient Method

- We construct $\{\mathbf{p}^k\}$ by starting with the steepest gradient

$$\mathbf{p}^0 = \mathbf{r}^0 = \mathbf{b} - A\mathbf{x}^0.$$

- It can be shown (*e. g.* Golub & Van Loan) that

$$\mathbf{p}^{k+1} = \mathbf{r}^{k+1} + \beta_k \mathbf{p}^k.$$

and enforcing A-conjugacy gives

$$\beta_k = -\frac{p_i^k A_{ij} r_j^{k+1}}{p_m^k A_{mn} p_n^k} = \frac{r_i^{k+1} r_i^{k+1}}{r_j^k r_j^k},$$

- The last equality is not obvious. It follows by induction using

$$\mathbf{r}^k = \mathbf{b} - A\mathbf{x}^k = \mathbf{b} - A(\mathbf{x}^{k-1} + \alpha_{k-1}\mathbf{p}^{k-1}) = \mathbf{r}^{k-1} - \alpha_{k-1} A\mathbf{p}^{k-1}$$

Conjugate Gradient Algorithm

```
A = [2 1 ; 1 3]; b = [5 7]'; n = 2; x = zeros(n,1);
r = b - A*x; %Initial residuals
p = r; %Initial search direction
rdotr = r'*r; %Inner product
%Loop over maximum of n possible steps
 for i=1:n
   Ap = A*p;
   alpha=rdotr/(p'*Ap);
   x += alpha*p; %Update guess
   r -= alpha*Ap; %Update residuals
   rdotr_new = r'*r;
   %Convergence criteria based on size of residuals
   if sqrt(rdotr_new) < 1e-10
     break;
   end
    p = r + (rdotr_new/rdotr)*p; %Update search direction
    rdotr = rdotr_new
  end
```

Convergence Analysis

- The space spanned by the vectors $\{\mathbf{p}^k\}$ is the same as the Krylov subspace $\{\mathbf{r}^0, A\mathbf{r}^0, \ldots, A^k\mathbf{r}^0\}$.
- We can therefore write the k-th iterate in the form

$$\mathbf{x}^k = \mathbf{x}^0 + \mathcal{P}_{k-1}(A)\,\mathbf{r}^0,$$

  where $\mathcal{P}_{k-1}$ is an order $k-1$th polynomial.
- Hence the error in the k-th iterate is

$$\mathbf{e}^k = \mathbf{x} - \mathbf{x}^k = \mathbf{x} - \mathbf{x}^0 - \mathcal{P}_{k-1}(A)\,\mathbf{r}^0$$

$$= [I - A\mathcal{P}_{k-1}(A)]\,\mathbf{e}^0 = \mathcal{P}_k(A)\,\mathbf{e}^0,$$

  where $A\mathbf{x} = \mathbf{b}$.

Convergence Analysis

- A-norm of error is minimised over the Krylov subspace (see Elman *et al*).

$$||\mathbf{e}^k||_A = e_i^k A_{ij} e_j^k = \min_{\mathcal{P}_k} ||\mathcal{P}_k(A)\mathbf{e}^0||_A$$

Convergence Analysis

- A-norm of error is minimised over the Krylov subspace (see Elman *et al*).

$$||\mathbf{e}^k||_A = e_i^k A_{ij} e_j^k = \min_{\mathcal{P}_k} ||\mathcal{P}_k(A)\mathbf{e}^0||_A$$

- Expand error in eigenbasis of A to show that

$$||\mathbf{e}^k||_A \leq \min_{\mathcal{P}_k} \max_i |\mathcal{P}_k(\lambda_i)| \, ||\mathbf{e}^0||_A$$

Convergence Analysis

- A-norm of error is minimised over the Krylov subspace (see Elman *et al*).

$$||\mathbf{e}^k||_A = e_i^k A_{ij} e_j^k = \min_{\mathcal{P}_k} ||\mathcal{P}_k(A)\mathbf{e}^0||_A$$

- Expand error in eigenbasis of A to show that

$$||\mathbf{e}^k||_A \leq \min_{\mathcal{P}_k} \max_i |\mathcal{P}_k(\lambda_i)| \, ||\mathbf{e}^0||_A$$

- Assuming eigenvalues lie in range $\lambda_j \in [\lambda_{\text{min}}(A), \lambda_{\text{max}}(A)]$ and fitting a Chebyshev polynomial to achieve the minimisation gives

$$||\mathbf{e}^k||_A \leq 2 \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^k ||\mathbf{e}^0||_A,$$

where $\kappa = \lambda_{\text{max}}(A)/\lambda_{\text{min}}(A)$ is the condition number of the matrix.

More general iterative methods

- CG cannot be applied to non-symmetric systems.
- GMRES is an iterative method that constructs approximate solutions within the Krylov subspace that minimise the Euclidean norm of the residual $||A\mathbf{x}^n - \mathbf{b}||$.
- Implementation is a little tricky because solving the minimisation problem requires some more linear algebra.

More general iterative methods

- CG cannot be applied to non-symmetric systems.
- GMRES is an iterative method that constructs approximate solutions within the Krylov subspace that minimise the Euclidean norm of the residual $||A\mathbf{x}^n - \mathbf{b}||$.
- Implementation is a little tricky because solving the minimisation problem requires some more linear algebra.
- A bound on the relative reduction on the size of the residuals at the k-th step can again be found in terms of a k-th degree polynomial

$$\frac{||\mathbf{r}^k||}{||\mathbf{r}^0||} \leq \kappa(V) \min_{\mathcal{P}_k} \max_i |\mathcal{P}_k(\lambda_i)|,$$

where V is the matrix consisting of the eigenvectors of the matrix, A.

More general iterative methods

- CG cannot be applied to non-symmetric systems.
- GMRES is an iterative method that constructs approximate solutions within the Krylov subspace that minimise the Euclidean norm of the residual $||A\mathbf{x}^n - \mathbf{b}||$.
- Implementation is a little tricky because solving the minimisation problem requires some more linear algebra.
- A bound on the relative reduction on the size of the residuals at the k-th step can again be found in terms of a k-th degree polynomial

$$\frac{||\mathbf{r}^k||}{||\mathbf{r}^0||} \leq \kappa(V) \min_{\mathcal{P}_k} \max_i |\mathcal{P}_k(\lambda_i)|,$$

  where V is the matrix consisting of the eigenvectors of the matrix, A.
- BiCGSTAB is an alternative method that generalises CG by imposing biorthogonality conditions on vectors associated with the Krylov subspaces of A and A^T.

Preconditioning

- Convergence of iterative methods depends on the spectrum of the matrix — the location of the eigenvalues.
- Idea of preconditioning is to modify the spectrum to improve convergence rate of the method.
- Solve the system

$$P^{-1}A\mathbf{x} = P^{-1}\mathbf{b},$$

  where P is the preconditioner.
- Obviously preconditioner should be cheap to compute and apply.
- Ideally want to ensure that convergence does not depend on mesh or any physical parameters of the problem.

Generic Preconditioners

- A number of generic possibilities exist:
  - Choose P to be the (block) diagonal of the matrix.
  - Use Incomplete LU decomposition (ILU).
  - Use a lower-order method to approximate the system (spectral).
  - Domain decomposition preconditioners.
  - Multigrid and algebraic multigrid.
- A bad solver can be a good preconditioner.

Problem-specific preconditioners

- Use the structure of the matrix.

Problem-specific preconditioners

- For Navier–Stokes equations the matrix structure is

$$\begin{pmatrix} F & B^T \\ B & 0 \end{pmatrix},$$

- Consider a diagonal preconditioner, spectrum of the preconditioned system follows from

$$\begin{pmatrix} D_1^{-1} & 0 \\ 0 & D_2^{-1} \end{pmatrix} \begin{pmatrix} F & B^T \\ B & 0 \end{pmatrix} \begin{pmatrix} u \\ p \end{pmatrix} = \lambda \begin{pmatrix} u \\ p \end{pmatrix}.$$

Problem-specific preconditioners

- For Navier–Stokes equations the matrix structure is

$$\begin{pmatrix} F & B^T \\ B & 0 \end{pmatrix},$$

- Consider a diagonal preconditioner, spectrum of the preconditioned system follows from

$$\begin{pmatrix} F & B^T \\ B & 0 \end{pmatrix} \begin{pmatrix} u \\ p \end{pmatrix} = \lambda \begin{pmatrix} D_1 & 0 \\ 0 & D_2 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix}.$$

Problem-specific preconditioners

- For Navier–Stokes equations the matrix structure is

$$\begin{pmatrix} F & B^T \\ B & 0 \end{pmatrix},$$

- Choose $D_1 = F$.

$$\begin{pmatrix} F & B^T \\ B & 0 \end{pmatrix} \begin{pmatrix} u \\ p \end{pmatrix} = \lambda \begin{pmatrix} F & 0 \\ 0 & D_2 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix}.$$

Problem-specific preconditioners

- For Navier–Stokes equations the matrix structure is

$$\begin{pmatrix} F & B^T \\ B & 0 \end{pmatrix},$$

- Choose $D_1 = F$.

$$\begin{pmatrix} F & B^T \\ B & 0 \end{pmatrix} \begin{pmatrix} u \\ p \end{pmatrix} = \lambda \begin{pmatrix} F & 0 \\ 0 & D_2 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix}.$$

- Have one eigenvalue $\lambda = 1$

Problem-specific preconditioners

- For Navier–Stokes equations the matrix structure is

$$\begin{pmatrix} F & B^T \\ B & 0 \end{pmatrix},$$

- Choose $D_1 = F$.

$$\begin{pmatrix} F & B^T \\ B & 0 \end{pmatrix} \begin{pmatrix} u \\ p \end{pmatrix} = \lambda \begin{pmatrix} F & 0 \\ 0 & D_2 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix}.$$

- Have one eigenvalue $\lambda = 1$
- If $\lambda \neq 1$, $u = \frac{1}{\lambda - 1} F^{-1} B^T p$, so

$$BF^{-1}B^T p = \lambda(\lambda - 1) D_2 p \quad \Rightarrow \quad D_2 = BF^{-1}B^T$$

Problem-specific preconditioners

- For Navier–Stokes equations the matrix structure is

$$\begin{pmatrix} F & B^T \\ B & 0 \end{pmatrix},$$

- Choose $D_1 = F$.

$$\begin{pmatrix} F & B^T \\ B & 0 \end{pmatrix} \begin{pmatrix} u \\ p \end{pmatrix} = \lambda \begin{pmatrix} F & 0 \\ 0 & D_2 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix}.$$

- Have one eigenvalue $\lambda = 1$
- If $\lambda \neq 1$, $u = \frac{1}{\lambda - 1} F^{-1} B^T p$, so

$$BF^{-1}B^T p = \lambda(\lambda - 1) D_2 p \quad \Rightarrow \quad D_2 = BF^{-1}B^T$$

- and $\lambda(\lambda - 1) = 1$, $\lambda = (1 \pm \sqrt{5})/2$
- Three eigenvalues ... convergence in three steps! (see Elman *et al.*)

Preconditioning Software

- For standard techniques use packages!
  - TRILINOS (`http://trilinos.sandia.gov/`)
  - PETSc (`http://www.mcs.anl.gov/petsc/`)
- These packages also have implementations of GMRES, BiCGSTAB that run in serial and parallel.
- Think about the human cost ... is it better to spend time developing a brilliant preconditioner or just use an off-the-shelf, sub-optimal, solver and wait longer.

Iterative solvers: summary

- Can give optimal solvers, $O(n)$.
- Not as robust unless good preconditioner available.
- Preconditioners can be general purpose or problem-specific.
- Typically problem-specific preconditioners perform better, but take much longer to develop.
- Resolve for different right-hand sides not much cheaper.
- Often the only possibility for large (realistic) problems.

Some practical advice on numerics

- Check the equations
  - Check all equations by constructing an "exact" solution that satisfies the boundary conditions and lump any left over terms into body force/source terms.
  - Check your "exact" solution gives zero (well suitably small) initial residuals, if not check your equations.

Some practical advice on numerics

- Check the equations
- Solve the linear system directly
  - Start by using a direct solver and for nonlinear problems, use Newton's method.
  - Initially compute associated Jacobian matrix using finite differences

$$\mathcal{J}_{ij} \approx \frac{\mathcal{R}_i(\mathbf{u} + \epsilon \mathbf{e}_j) - \mathcal{R}_i(\mathbf{u})}{\epsilon},$$

    (compute an entire column at a time)
  - Check system converges to your "exact" solution.
  - Modify Jacobian so that it is computed analytically.
  - If convergence is not quadratic check residuals and Jacobian (compare Jacobian to finite difference version).

Some practical advice on numerics

- Check the equations
- Solve the linear system directly
- Investigate iterative solvers
  - Investigate small unpreconditioned system. How well does it converge?
  - Try generic preconditioners and examine performance with problem size.
  - Develop/investigate problem-specific preconditioners.