MAST90026 Computational Differential Equations: Week 7

Jesse Collis
Modified from Hailong Guo (2022)

Semester 1 2024

The University of Melbourne

Sparse solvers

Finite difference and finite element methods (but not spectral solvers) we get large sparse linear systems.

Can't be solved efficiently using standard methods for dense systems e.g. LU factorisation or Cholesky factorisation.

For a finite difference grid, there are

- $m \times m$ unknowns, so $N = m^2$ in 2D
- $m \times m \times m$ unkowns, so $N = m^3$ in 3D

A dense Cholesky solver uses $\sim \frac{1}{6}N^3$ operations, so we need

- $O(m^6)$ operations in 2D
- $O(m^9)$ operations in 3D

But if we count the number of unzero elements in A we have

- $5m^2$ elements in 2D
- $7m^3$ elements in 3D

Two possible solvers

Sparse direct solvers, which are competitive in 2D

Iterative solvers which are ultimately more efficient as we move up in dimensions

Sparse Cholesky uses a re-ordering automatically and only handles non-zero elements, resulting in

- $O(m^3)$ operations in 2D (c.f. $O(m^6)$) and
- $O(m^6)$ operations in 3D (c.f. $O(m^9)$)

Iterative methods

Three main classes of iterative solvers:

- Classical stationary (1950s)
- Krylov space methods: (1970s-1990s)
- Multigrid methods (1980s-)

Two main reasons to look at iterative solvers for sparse systems:

- More Efficiency: each iteration typically involves Av, for FDM with a 5 point stencil, A has 5 non-zero entries per row and A is $m^2 \times m^2$ so to form Av takes $5m^2$ operations
  So in 2D as long as our solver converges in $O(m)$ iterations, we can achieve $O(m^3)$ complexity running time which matches (asymptotically) sparse Cholesky.
  In 3D we only have Lm^3 operations per iteration so we require $O(m^3)$ iterations for convergence to do equal to or better than sparse Cholesky.

Reasons for iterative solvers

- The matrix A is not exact.
  We already have a discretisation error $O(h^2)$ (local truncation error), so the residual is $O(h^2)$. Why use a solver whose relative residuals are $O(u)$ (unit round-off) when the backward error in A is $O(h^2)$? Instead, aim to solve $Au = F$ approximately till residual is $O(h^2)$ – this is not possible with direct methods.

Krylov subspace methods

Krylov subspace methods is a large class of methods. We will focus on (preconditioned) conjugate gradient method. Assume A is SPD.

Observation: find the solution of $A\vec{x} = \vec{b} \equiv$ find the minimum of $\phi(x)$, where $\phi(x) = \frac{1}{2}\vec{x}^T A\vec{x} - \vec{b}^T\vec{x}$.

Aim: find this minimum as fast as possible (in as few iterations as possible).

Steepest descent: idea

Since the directional derivative of ϕ in the direction $\vec{r}$ (a unit vector) is $\nabla\phi \cdot \vec{r}$, this is minimised if $\vec{r}$ is in the direction $-\nabla\phi$ which implies that $-\nabla\phi$ is the direction of steepest descent.

At current guess $\vec{x}_k$:

- Compute descent direction, $\vec{r}_k = -\nabla\phi\big|_{\vec{x}_k}$.
- Step along $\vec{r}_k$ in a line search a distance α_k, chosen to give the minimum possible value of ϕ.

$$\min_{\alpha \geq 0} \phi(\vec{x}_k + \alpha\vec{r}_k).$$

Steepest descent: derivation

$$
\begin{aligned}
&\frac{d}{d\alpha}\phi(\vec{x}_k + \alpha\vec{r}_k) = 0 \\
\implies &\nabla\phi\Big|_{\vec{x}_k+\alpha\vec{r}_k} \cdot \vec{r}_k = 0 \\
\implies &\nabla\phi\Big|_{\vec{x}_{k+1}} \cdot \vec{r}_k = 0 \implies \text{ so } \vec{r}_{k+1} \perp \vec{r}_k. \implies \text{ a zig-zag path.}
\end{aligned}
$$

Since $\nabla\phi = Ax - b$, $\vec{r}_k = -\nabla\phi = b - A\vec{x}_k$, the orthogonality of $\vec{r}_k$ and $\vec{}_{k+1}$

$$\vec{r}_k \cdot \underbrace{[b - A(\vec{x}_k + \alpha_k\vec{r}_k)]}_{\vec{r}_{k+1}} = 0,$$

give an equation for α_k:

$$\alpha_k = \frac{\vec{r}_k \cdot (\vec{b} - A\vec{x}_k)}{\vec{r}_k \cdot A\vec{r}_k} = \frac{\vec{r}_k^T\vec{r}_k}{\vec{r}_k^T A\vec{r}_k},$$

in terms of the *residual* $\vec{r}_k = \vec{b} - A\vec{x}_k = -\nabla\phi(\vec{x}_k)$.

Steepest descent: algorithm

```
Choose $x_0$
repeat
    $r_k = b - Ax_k$
    stopping criterion here
    $\alpha_k = r_k^T r_k / (r_k^T A r_k)$
    $x_{k+1} = x_k + \alpha_k r_k$
end
```

Uses s matrix-vector product per iteration

Steepest descent: algorithm

```
$Choose x_0$
$r_0 = b - Ax_0$
$repeat \quad k = 0, 1, 2, \dots$
    $w_k = Ar_k$
    $\alpha_k = r_k^T r_k / (r_k^T w_k)$
    $x_{k+1} = x_k + \alpha_k r_k$
    $r_{k+1} = r_k - \alpha_k w_k$
    stopping criterion here
```

Uses 1 matrix-vector product per iteration

It always finds the minimum, but it may be very slow.

Steepest descent: convergence

Theorem

If A is symmetric positive definite with condition number

$$\kappa_2(A) = \frac{\lambda_{\max}(A)}{\lambda_{\min}(A)},$$

then the steepest descent algorithm converges according to

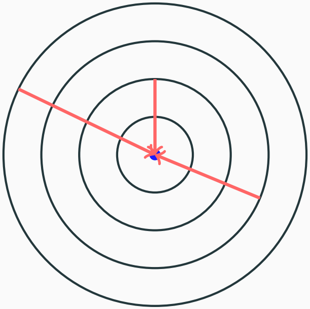
$$\|\vec{e}_k\|_A \leq \left(\frac{\kappa - 1}{\kappa + 1}\right)^k \|\vec{e}_0\|_A,$$

where we have used a new norm for the error

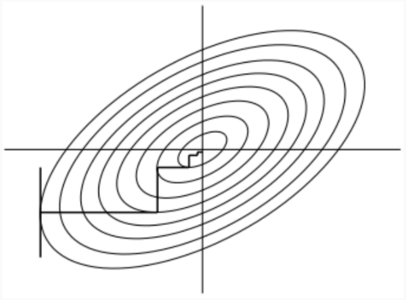
$$\|\vec{x}\|_A = (\vec{x}^T A \vec{x})^{\frac{1}{2}},$$

which has the properties of a norm only if A is spd.

Steepest descent: example



Ideal case $\kappa = 1$



When $\kappa \gg 1$

The method of steepest descent slows down for ill-condition matrices.

Steepest descent: FDM and FEM

For a finite difference 5-point stencil, the matrix A that we obtain has $\kappa_2(A) \sim h^{-2}$

For FEM + nodal basis gives matrices with $\kappa_2(A) \leq Ch^{-2}$

For a uniform mesh on a square we have $\kappa_2(A) = \frac{2}{\pi^2 h^2} + O(1)$

Refine our mesh to get smaller error, $\kappa_2(A)$ grows and steepest descent slows down.

Steepest descent: number of iteration

Want

$$\left(\frac{\kappa-1}{\kappa+1}\right)^n < \epsilon.$$

Where

$$\frac{\kappa-1}{\kappa+1} = 1 - \frac{2}{\kappa+1} \sim 1 - \frac{2}{\kappa}, \qquad \text{for large } \kappa.$$

So we have

$$\begin{aligned} & n \log\left(1 - \frac{2}{\kappa}\right) < \log(\epsilon), \\ \implies & \frac{-2n}{\kappa} < \log(\epsilon), \qquad \text{by Taylor approximation (as } \tfrac{2}{\kappa} \text{ is small)} \\ \implies & n > \log\left(\frac{1}{\epsilon}\right)\frac{\kappa}{2}, \\ & \text{i.e. } n \propto \kappa_2(A). \end{aligned}$$

Steepest descent: compare with sparse Cholesky factorization

In 2D, $\kappa_2(A) \sim h^{-2} \sim m^2$ for an $m \times m$ grid, so

$$\begin{aligned}\text{Total work} \sim& \ \#\text{iterations} \times \ \text{ops/iteration}\\ =& m^2 \times 5m^2\\ =& O(m^4).\end{aligned}$$

which is worse asymptotically than sparse Cholesky factorisation.

In 3D, $\kappa_2(A) \sim h^{-2}$ as well, so we use $O(m^5)$ operations to solve our system – this is already better than sparse Cholesky. But, for the 4th order biharmonic equation arising in elasticity, $\nabla^2\nabla^2 u = f$, $\kappa_2(A) \sim h^{-4}$ i.e. slowdown is worse.

Conjugate gradient method:idea

Idea: Choose different directions $\vec{p}_k$, constructed from steepest descent directions $\vec{r}_k$, which have very nice properties.

In particular, once we have searched along a particular direction, we never want to search in that direction again.

- $\vec{x}_1$ minimises ϕ when searching over $\{\vec{p}_1\}$ [as before]
- $\vec{x}_2$ minimises ϕ when searching over span$\{\vec{p}_1, \vec{p}_2\}$ [not true of steepest descent]
- $\vec{x}_3$ minimises ϕ when searching over span$\{\vec{p}_1, \vec{p}_2, \vec{p}_3\}$

$$\vdots$$

By making x_k minimise ϕ over nested subspaces, we never 'backtrack' or search along previous directions, so

$$\phi(\vec{x}_{k+1}) \leq \phi(\vec{x}_k),$$

i.e. ϕ decreases monotonically.

The hard part: making this cheap.

Conjugate gradient method: solution to hard problem

We want to construct $\{\vec{p}_j\}$ so that

$$\alpha_k = \min_{\alpha} \phi(\vec{x}_{k-1} + \alpha \vec{p}_k)$$

also minimises ϕ over $x \in \text{span}\{\vec{p}_1, \dots, \vec{p}_k\}$.

Let's try generating $\vec{p}_1$:

1. Guess $\vec{x}_0, \vec{p}_0$ [often $\vec{x}_0 = \vec{0}$, $\vec{p}_0 = \vec{r}_0 = \vec{b}$]

Conjugate gradient method: solution to hard problem

2. Find α_0 by a 1D minimisation of $\phi(\vec{x}_0 + \alpha\vec{p}_0)$

$$\begin{aligned}\vec{p}_1\phi(\vec{x}_0 + \alpha\vec{p}_0) =& \frac{1}{2}(\vec{x}_0 + \alpha\vec{p}_0)^T A(\vec{x}_0 + \alpha\vec{p}_0) - (\vec{x}_0 + \alpha\vec{p}_0)^T\vec{b} \\ =& \frac{1}{2}\vec{x}_0^T A\vec{x}_0 - \vec{x}_0^T\vec{b} + \frac{1}{2}\alpha^2\vec{p}_0^T A\vec{p}_0 + \alpha\vec{p}_0^T A\vec{x}_0 - \alpha\vec{p}_0^T\vec{b},\end{aligned}$$

$$\frac{d}{d\alpha}(\phi(\vec{x}_0 + \alpha\vec{p}_0) = \vec{0} \implies \alpha\vec{p}_0^T A\vec{p}_0 = \vec{p}_0^T(\vec{b} - A\vec{x}) = \vec{p}_0^T\vec{p}_0,$$

so that we get

$$\alpha_0 = \frac{\vec{p}_0^T\vec{p}_0}{\vec{p}_0^T A\vec{p}_0}, \quad \vec{x}_1 = \qquad \vec{x}_0 + \alpha_0\vec{p}_0, \quad \vec{r}_1 = \vec{r}_0 - \alpha_0\underbrace{A\vec{p}_0}_{\vec{w}_0},$$

as for the first step of steepest descent

Conjugate gradient method: solution to hard problem

3. Find $\vec{x}_2$ by 2D minimisation of

$$\begin{aligned}
\phi(\vec{x}_0 + \alpha_0\vec{p}_0 + \alpha_1\vec{p}_1) =& \phi(\vec{x}_1 + \alpha_1\vec{p}_1) \\
=& \frac{1}{2}(\vec{x}_1 + \alpha_1\vec{p}_1)^T A(\vec{x}_1 + \alpha_1\vec{p}_1) - (\vec{x}_1 + \alpha_1\vec{p}_1)^T\vec{b} \\
=& \underbrace{\frac{1}{2}\vec{x}_1^T A\vec{x}_1 - \vec{x}_1^T\vec{b}}_{\phi(\vec{x}_1)} + \frac{1}{2}\alpha_1^2\vec{p}_1^T A\vec{p}_1 + \underbrace{\alpha_1\vec{p}_1^T A\vec{x}_1 - \alpha_1\vec{p}_1^T\vec{b}}_{-\alpha_1\vec{p}_1^T\vec{r}_1} \\
=& \phi(\vec{x}_1) + \frac{1}{2}\alpha_1^2\vec{p}_1^T A\vec{p}_1 - \alpha_1\vec{p}_1^T(\vec{r}_0 - \alpha_0 A\vec{p}_0),
\end{aligned}$$

where we note that $\beta\alpha\vec{p}_1^T A\vec{p}_0$ is the only term that includes α_1 and α_0. We can de-couple the 2D minimisation of ϕ into two 1D minimizations if we set this term to zero by choosing

$$\vec{p}_1^T A\vec{p}_0 = 0.$$

Note: Local minimisation also gives us the global minimum.

Conjugate gradient method: conjugate direction

Two vectors that satisfy $\vec{u}^{\mathrm{T}} A \vec{v} = 0$ are called *A-conjugate*.

If A is symmetric we can define an inner product $(\vec{u}, \vec{v})_A \equiv \vec{u}^T A \vec{v}$. If A is also positive definite, this induces a norm $\|\vec{u}\|_A^2 \equiv < \vec{u}, \vec{u} >_A = \vec{u}^T A \vec{u}$. So A-conjugate vectors are orthogonal wrt the A-inner product.

Aim to produce a set of mutually A-conjugate vectors. To achieve this we do conjugate Gram-Schmidt to form $\{\vec{p}_0, \vec{p}_1, \ldots\}$, from $\{\vec{r}_0, \vec{r}_1, \ldots\}$.

Conjugate gradient method: algorithm

1. Guess $\vec{p}_0 = \vec{r}_0$.
2. Let

$$\alpha_0 = \frac{\vec{r}_0^T \vec{r}_0}{\vec{p}_0^T A \vec{p}_0}, \quad \vec{x}_1 = \vec{x}_0 + \alpha_0 \vec{p}_0, \quad \vec{r}_1 = \vec{r}_0 - \alpha_0 A \vec{p}_0,$$

   as before.
3. $\vec{p}_1 = \vec{r}_1 -$ component of $\vec{r}_1$ not $A-$ conjugate to $\vec{p}_0$, i.e. $\vec{p}_1 = \vec{r}_1 + \beta_0 \vec{p}_0$ (since span $\{\vec{r}_0, \vec{r}_1\}$ =span $\{\vec{p}_0, \vec{p}_1\}$) Determine β_0 by enforcing A-conjugacy

$$\vec{p}_1^T A \vec{p}_0 = 0 \implies (\vec{r}_1 + \beta_0 \vec{p}_0)^T A \vec{p}_0 = 0, \implies \quad \beta_0 = \frac{-\vec{r}_1^T A \vec{p}_0}{\vec{p}_0^T A \vec{p}_0}.$$

   Then since

$$\vec{r}_1 = \vec{r}_0 - \alpha_0 A \vec{p}_0, \qquad -A\vec{p}_0 = \frac{\vec{r}_1 - \vec{r}_0}{\alpha_0},$$

$$\beta_0 = \frac{\vec{r}_1^T (\vec{r}_1 - \vec{r}_0)/\alpha_0}{\vec{p}_0^T A \vec{p}_0} = \frac{\vec{r}_1^T \vec{r}_1}{\alpha_0 \vec{p}_0^T A \vec{p}_0} = \frac{\vec{r}_1^T \vec{r}_1}{\vec{r}_0^T \vec{r}_0},$$

(= reduction in the square residual.)

Conjugate gradient method: algorithm

4. So given $\vec{p}_1$, $\vec{x}_1$, $\vec{r}_1$, we calculate $\vec{p}_2$:

$$\alpha_1 = \frac{\vec{r}_1^T \vec{r}_1}{\vec{p}_1^T A \vec{p}_1}, \qquad x_2 = \vec{x}_1 + \alpha_1 \vec{p}_1, \qquad \vec{r}_2 = \vec{r}_1 - \alpha_1 A \vec{p}_1,$$

$$\beta_1 = \frac{\vec{r}_2^T \vec{r}_2}{\vec{r}_1^T \vec{r}_1}, \qquad \vec{p}_2 = \vec{r}_2 + \beta_1 \vec{p}_1.$$

This all leaves $\vec{r}_2 \perp \vec{r}_0, \vec{r}_1$; $\vec{p}_2$ conjugate to $\vec{p}_0, \vec{p}_1$; and $\vec{r}_2 \perp \vec{p}_0, \vec{p}_1$ (but not $\vec{p}_2$.

After k iterations,

$$\vec{x}_k = \text{solution that minimizes } \phi \text{ over } x \in \vec{x}_0 + \text{span}\{\vec{p}_0, \vec{p}_1, \ldots, \vec{p}_{k-1}\}$$

Conjugate gradient method: property

Minimising ϕ is same as minimising the error in the A-norm: $\|\vec{e}\|_A = \|\vec{x} - \vec{x}^*\|_A$.

Proof:

$$\begin{aligned}\|\vec{e}\|_A^2 =&\|\vec{x} - \vec{x}^*\|_A^2 = (\vec{x} - \vec{x}^*)^T A(\vec{x} - \vec{x}^*)\\ =&\vec{x}^T A\vec{x} - 2\vec{x}^T A\vec{x}^* + \vec{x}^{*T} A\vec{x}^*\\ =&\underbrace{\vec{x}^T A\vec{x} - 2\vec{x}^T\vec{b}}_{2\phi(\vec{x})} + \underbrace{\vec{x}^{*T} A\vec{x}^*}_{\text{constant}}\end{aligned}$$

$\vec{x}_k$ minimises $\|\vec{e}\|_A$ over $\vec{x}_0 + \text{span}\{\vec{p}_0, \dots, \vec{p}_{k-1}\}$. But

$$\vec{r}_0 = \vec{p}_0, \vec{r}_1 = \vec{r}_0 - \alpha_0 A\vec{p}_0, \vec{r}_2 = \vec{r}_1 - \alpha_1 A\vec{p}_1 \dots .$$

so

$$\text{span}\{\vec{p}_0, \vec{p}_1, \dots, \vec{p}_{k-1}\} = \text{span}\{\vec{r}_0, A\vec{r}_0, A^2\vec{r}_0, \dots, A^{k-1}\vec{r}_0\}.$$

Krylov Space

Conjugate gradient method: property

Since $A\vec{x} = \vec{b} - \vec{r}$ and $A\vec{x}^* = \vec{b}$, $A\vec{e} = -\vec{r}$, so

$$\begin{aligned}\text{span}\{\vec{p}_0, \vec{p}_1, \ldots, \vec{p}_{k-1}\} =& \text{span}\{\vec{r}_0, A\vec{r}_0, A^2\vec{r}_0, \ldots, A^{k-1}\vec{r}_0\} \\ =& \text{span}\{A\vec{e}_0, A^2\vec{e}_0, \ldots, A^k\vec{e}_0\}\end{aligned}$$

so

$$\vec{x}_k = \vec{x}_0 + \text{Linear combinations}\{A\vec{e}_0, \ldots, A^k\vec{e}_0\}$$

If we subtract $\vec{x}^*$ from both sides then

$$\vec{e}_k = \vec{e}_0 + \text{Linear combinations}\{A\vec{e}_0, \ldots, A^k\vec{e}_0\} = P_k(A)\vec{e}_0$$

where P_k is a polynomial of degree k with $p(0) = 1$.

Since conjugate-gradient minimises $\phi(x)$ and $\|\vec{e}\|_A$ over $\mathcal{K}_k$,

$$\|\vec{e}_k\|_A = \min_{p \in P_k} \|p(A)\vec{e}_0\|_A.$$

$\Rightarrow$ Conjugate gradient converges to the exact answer in N iterations (ignoring roundoff).

Conjugate gradient method: example

If A has only $k < N$ different eigenvalues then conjugate gradient converges in k iterations.

Proof The polynomial of degree k:

$$\prod_{i=1}^{k}\left(1 - \frac{x}{\lambda_i}\right),$$

has $p(0) = 1$, but $p(\lambda_i) = 0 \implies \vec{e}_k = \vec{0}$.

If we don't know the spectrum of A but only know $\lambda \in (\lambda_{\min}, \lambda_{\max})$, then the best result is:

$$\frac{\|\vec{e}_k\|_A}{\|\vec{e}_0\|} \leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k, \tag{1}$$

where $\kappa = \kappa_2(A) = \frac{\lambda_{\max}}{\lambda_{\min}}$.

So if κ is near 1, i.e. $\lambda_{\max} - \lambda_{\min} \ll \lambda_{\max} + \lambda_{\min}$, then convergence is rapid.

Conjugate gradient method: number of iteration

Estimate a (pessimistic) bound for the number of iterations to achieve a tolerance ϵ:

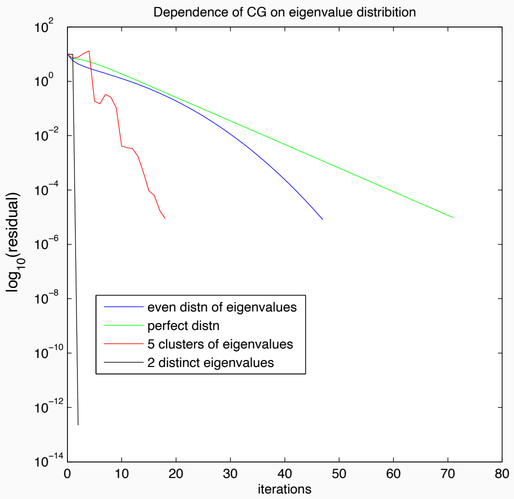
$$n > \frac{\sqrt{\kappa}}{2} \log\left(\frac{2}{\epsilon}\right).$$

So we need $\sim \sqrt{\kappa}$ iterations for CG to achieve convergence.

For a 5-point FD stencil or P_1 or Q_1 FEM, $\kappa_2 \sim h^{-2} \sim m^2$. So if we require $\sim \sqrt{\kappa_2}$ iterations then the total work is $O(m^3)$ to find the solution – the same complexity as $\backslash$ in 2D.

In 3D we have $\sim m$ iterations at $O(m^3)$ operations per iteration, so $O(m^4)$, compared to $O(m^6)$ for $\backslash$.

Conjugate gradient method: Numerical example



CG converges faster if the spectrum is ‘nice’, we look for ways to improve the spectrum, i.e. preconditioning.

Preconditioned conjugate gradient (PCG): idea

Instead of solving $A\vec{x} = \vec{b}$, solve $M^{-1}A\vec{x} = M^{-1}\vec{b}$ such that:

- it must be cheap to solve $M\vec{z} = \vec{r}$ (which is the only place M enters in the PCG algorithm)
- $M^{-1}A$ must be sparse and symmetric positive definite (if A is)
- $M^{-1}A$ has a 'nicer' spectrum, e.g. $\kappa_2(M^{-1}A) \ll \kappa_2(A)$

Choice of M:

- If $M = I$ then $M\vec{z} = \vec{r}$ is easy to solve, but $\Lambda(M^{-1}A) = \Lambda(A)$ and we haven't done anything.
- If $M = A$ now $\Lambda(M^{-1}A) = \Lambda(I)$ which is perfect but $M\vec{z} = \vec{r}$ is $A\vec{z} = \vec{r}$ which is just our original problem.

look for M that 'approximates' A so that the eigenvalues of $M^{-1}A$ are more clustered than A. We often look for a factorised $M = M_1 M_2$ where M_1, M_2 are cheap to invert, e.g. Cholesky factors $M = LL^T = R^T R$

PCG: choice of preconditioner

There are *no* magical preconditioners!

The best comes from a deep knowledge of A and where it came from. They often come from approximating A or solving a related simpler problem cheaply to get an approximation.

Some candidates that are symmetric positive definite and sparse:

- Jacobi (point), $M_J = D = \text{diag}(A)$.
  This has modest benefit unless the diagonal elements of A vary greatly. It has no effect on A_5 (since the diagonal elements are all the same).
- Incomplete factorisations
  For these we go through the process of (Cholesky) factorisation but prevent or reduce the amount of fill-in. So the matrix obtained is sparser than usual sparse Cholesky factorisation (less fill-in) but only gives an approximate factorisation.

PCG: incomplete Cholesky factorisation

Direct Cholesky (\)

$$A = R^T R \text{ or } LL^T.$$

Incomplete Cholesky

$$A = \bar{R}^T \bar{R} + \underbrace{\epsilon}_{\text{error}},$$

where $\bar{R}^T \bar{R} = M$ is a preconditioner, $M = M_1 M_2 = \bar{R}^T \bar{R}$ (or $M = \bar{L}\bar{L}^T$).

Matlab has (as at 2012):

- Incomplete Cholesky with no fill-in
  $L = ichol(A)$, which you then use in a call to *pcg*:
  $[, , ,] = pcg(A, b, tol, maxits, L, L')$
  This only allows non-zeroes of L where A has non-zeroes i.e. no fill-in. To call CG in MATLAB, use *pcg* with no preconditioner:
  $[, , ,] = pcg(A, b, tol, maxits)$

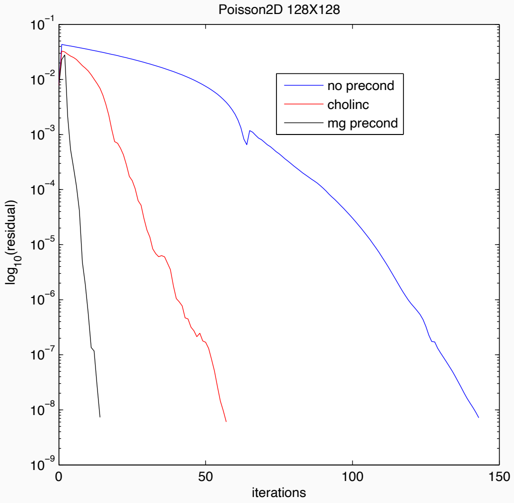
PCG: incomplete Cholesky factorisation

- Modified incomplete Cholesky
  Where the matrix A has come from an elliptic PDE (as in our cases), it is better when you delete an entry that would have produced fill-in to add that entry to the diagonal entry on the same row, so that $A\vec{x} = \bar{L}\bar{L}^T\vec{x}$ when $\vec{x}$ is a constant vector. This is called Modified incomplete Cholesky (MIC) factorization. This can be called in MATLAB by accessing the *options* structure in the call to *ichol*:
  $options.type =' nofill';$
  $options.michol =' on';$
  $L = ichol(A, options)$
  Supposedly, for A_5 this can reduce $\kappa_2(M^{-1}A)$ to $O(h^{-1})$.
  With this preconditioner, PCG has $\sqrt{\kappa} \sim m^{\frac{1}{2}}$ iterations, so the total work is $O(m^{2.5})$ in 2D, $O(m^{3.5})$ in 3D. This is called *optimal PCG* (for A_5) .

PCG: incomplete Cholesky factorisation

- Incomplete Cholesky with a drop tolerance
  $options.type =' ict';$
  $options.droptol = 0.05;$
  $L = ichol(A, options)$
  This throws away the fill-in entries smaller than a certain size determined by the drop tolerance. For large drop tolerance, this is the same as the no-fillin case (the default);

PCG: Numerical example



End of week 7!