

Lecture 18: Preconditioned iterative methods

COMP5930M Scientific Computation

Today

Conjugate gradient examples

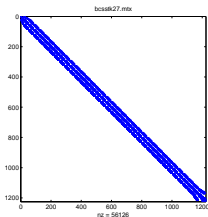
Preconditioning

Choosing a preconditioner

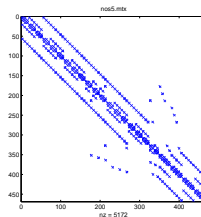
Matlab implementations

Preconditioned conjugate gradient examples

Conjugate gradient examples from MatrixMarket

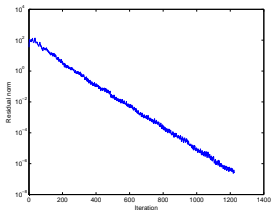


$n=1224, nz=56126$

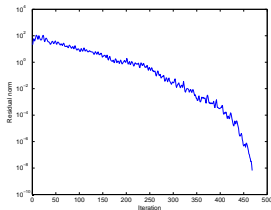


$n=468, nz=5172$

Sparsity pattern



Not converged in 1224 its



Not converged in 468 its

CG convergence

Preconditioning

- ▶ The convergence of our iterative methods is governed by the distribution of eigenvalues of \mathbf{A}
 - ▶ Measured as the number of iterations required to achieve a certain reduction of the residual
- ▶ Recall the condition number $\kappa(\mathbf{A}) = \frac{\max_i |\lambda_i|}{\min_i |\lambda_i|}$
- ▶ We can attempt to transform our linear system $\mathbf{Ax} = \mathbf{b}$ into one with a better eigenvalue distribution through the use of a preconditioning matrix \mathbf{M}
 - ▶ We apply the iterative method to a modified linear system
 - ▶ The goal is to reduce:
the number of iterations required, and the overall time taken

The preconditioned system

Given a linear system $\mathbf{Ax} = \mathbf{b}$ and preconditioning matrix \mathbf{M} we can define a modified system in a number of ways

- ▶ Left preconditioning

$$(\mathbf{M}^{-1}\mathbf{A})\mathbf{x} = \mathbf{M}^{-1}\mathbf{b}$$

- ▶ Right preconditioning

$$(\mathbf{AM}^{-1})(\mathbf{Mx}) = \mathbf{b}$$

- ▶ Left and right preconditioning (assuming $\mathbf{M} = \mathbf{M}_1\mathbf{M}_2$)

$$(\mathbf{M}_1^{-1}\mathbf{AM}_2^{-1})(\mathbf{M}_2\mathbf{x}) = \mathbf{M}_1^{-1}\mathbf{b}$$

The preconditioned conjugate gradient (PCG) algorithm

Assume we applied left preconditioning, $\mathbf{M}^{-1}\mathbf{Ax} = \mathbf{M}^{-1}\mathbf{b}$. Then:

- ▶ $\mathbf{r}_0 = \mathbf{b} - \mathbf{Ax}_0$
- ▶ $\mathbf{Mz}_0 = \mathbf{r}_0$ (preconditioned residual)
- ▶ $\mathbf{d}_0 = \mathbf{z}_0$ (preconditioned search direction)
- ▶ $i = 0, 1, 2, \dots$
 - ▶ $\alpha_i = \frac{\mathbf{r}_i^T \mathbf{z}_i}{\mathbf{d}_i^T \mathbf{Ad}_i}$
 - ▶ $\mathbf{x}_{i+1} = \mathbf{x}_i + \alpha_i \mathbf{d}_i$
 - ▶ $\mathbf{r}_{i+1} = \mathbf{r}_i - \alpha_i \mathbf{Ad}_i$
 - ▶ $\mathbf{Mz}_{i+1} = \mathbf{r}_{i+1}$ (preconditioned residual)
 - ▶ $\beta_{i+1} = \frac{\mathbf{r}_{i+1}^T \mathbf{z}_{i+1}}{\mathbf{r}_i^T \mathbf{z}_i}$
 - ▶ $\mathbf{d}_{i+1} = \mathbf{z}_{i+1} + \beta_{i+1} \mathbf{d}_i$

Notes on the PCG algorithm

- ▶ Largely unchanged (requires algebra to show this)
 - ▶ Although we apply the algorithm to the preconditioned system we can manipulate it back into the form $\mathbf{Ax} = \mathbf{b}$
- ▶ Only one matrix-multiply required per iteration: \mathbf{Ad}_i
- ▶ One linear system solution per iteration: $\mathbf{Mz}_{i+1} = \mathbf{r}_{i+1}$
 - ▶ Require this to be solved very efficiently
 - ▶ If possible in $\mathcal{O}(n^2)$ total cost per iteration does not increase

Remark on inverse matrices

- ▶ Here \mathbf{M} approximates the matrix \mathbf{A} so that $\mathbf{M}^{-1}\mathbf{A} \approx \mathbf{I}$
- ▶ We could also **approximate the inverse matrix \mathbf{A}^{-1}** , but:
 - ▶ We don't know anything about \mathbf{A}^{-1} in general
 - ▶ For many sparse matrices, the inverse \mathbf{A}^{-1} will be dense, e.g.

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{A}^{-1} = \begin{bmatrix} -\frac{1}{3} & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{2}{3} & -\frac{1}{3} & -\frac{1}{3} & -\frac{1}{3} \\ \frac{1}{3} & -\frac{1}{3} & \frac{2}{3} & -\frac{1}{3} & -\frac{1}{3} \\ \frac{1}{3} & -\frac{1}{3} & -\frac{1}{3} & \frac{2}{3} & -\frac{1}{3} \\ \frac{1}{3} & -\frac{1}{3} & -\frac{1}{3} & -\frac{1}{3} & \frac{2}{3} \end{bmatrix}$$

- ▶ Idea: Look for sparse preconditioner \mathbf{M} such that

$$\min_{\mathbf{M} \in \mathcal{G}_S} \|\mathbf{AM} - \mathbf{I}\|^2$$

where \mathcal{G}_S is the set of matrices with fixed sparsity pattern S .
 This strategy is called an approximate inverse preconditioner.

Properties of a good preconditioner

The choice of preconditioner is determined by several factors:

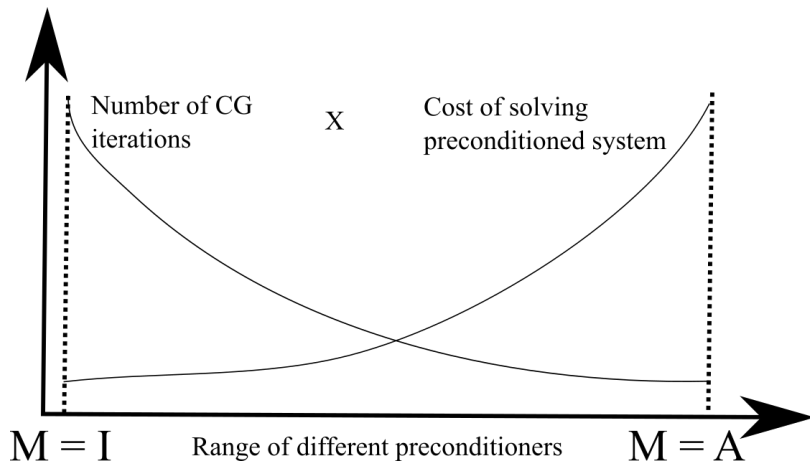
- ▶ Properties of the original system, eg. symmetry, pos.def.
- ▶ \mathbf{M} should be cheap to build
- ▶ $\mathbf{M}\mathbf{z} = \mathbf{r}$ should be cheap to solve
- ▶ Trade-off: the fewer iterations the more expensive \mathbf{M} is

A heuristic argument

M should approximate **A**

- ▶ If $\mathbf{M} = \mathbf{A}$ our preconditioned system will be the identity matrix \mathbf{I} and only 1 iteration is required
 - ▶ **Not practical**: inverting \mathbf{M} now has the same expense as inverting \mathbf{A}
- ▶ There are roughly two general approaches
 - ▶ Approximate factorisation of \mathbf{A}
 - ▶ Approximate inverse of \mathbf{A}
- ▶ Finding problem-specific preconditioners an art in itself

Finding an optimal preconditioner as a trade-off



Symmetric linear systems

- ▶ CG requires a SPD matrix \mathbf{A}
- ▶ Preconditioning must maintain the SPD property
- ▶ Left or right preconditioning are not guaranteed to maintain symmetry
 - ▶ Use left and right with $\mathbf{M1} = \mathbf{M2}^T$

Preconditioning SPD systems

- ▶ We can factorise a SPD matrix \mathbf{A} in the form

$$\mathbf{A} = \mathbf{L}\mathbf{L}^T$$

- ▶ This is known as Cholesky decomposition
a specialised form of LU decomposition
- ▶ For preconditioning we use a cheaper approximate process
 - ▶ Incomplete Cholesky decomposition

$$\mathbf{M} = \bar{\mathbf{L}}\bar{\mathbf{L}}^T$$

Incomplete factorisation

- ▶ **Recall:** The full factorisation process **introduces fill-in**
 - ▶ Sparse matrices can have non-sparse factorisations
 - ▶ Reordering algorithms help here too
- ▶ Incomplete factorisation ignores some of the fill-in
- ▶ Two approaches are adopted in practice
 - ▶ Structural - based on sparsity pattern of \mathbf{A}
 - ▶ Numerical - based on size of matrix entries

(i) Structural incomplete factorisation

- ▶ We know the sparsity pattern of \mathbf{A}
- ▶ We pre-define the sparsity pattern of $\bar{\mathbf{L}}$ and ignore terms that do not fit this pattern
- ▶ The simplest scheme assumes that $\bar{\mathbf{L}}$ has the same sparsity pattern as \mathbf{A}
 - ▶ Our factorisation produces no fill-in
 - ▶ It approximates the true factorisation \mathbf{L}
- ▶ We can define approximations with larger amounts of fill-in to achieve greater accuracy
 - ▶ at greater expense

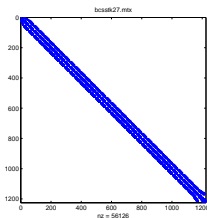
(ii) Numerical incomplete factorisation

- ▶ We modify the algorithm such that only “large” entries of $\bar{\mathbf{L}}$ are stored
 - ▶ Cannot be pre-defined, must be computed
- ▶ Controlled by a drop-tolerance d_{tol} that specifies the minimum acceptable size
 - ▶ Often also a parameter l_{max} that restricts storage
- ▶ Accuracy of $\bar{\mathbf{L}}$ is determined by d_{tol} and l_{max}
- ▶ If d_{tol} too large, the approximate factor $\bar{\mathbf{L}}$ will be singular and solving $\bar{\mathbf{L}}\mathbf{z} = \mathbf{r}$ will fail

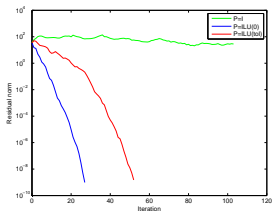
Using Matlab

- ▶ The Matlab function `ichol()` computes the **incomplete Cholesky factorisation**
 - ▶ `L = ichol(A,struct('type','nofill'))`
computes the structural factorisation with no fill-in
 - ▶ `L = ichol(A,struct('type','ict','droptol',0.01))`
computes the numerical factorisation with drop-tolerance d_{tol}
- ▶ The following symmetric examples demonstrate the benefits for the CG algorithm
 - ▶ The modified CG algorithm is covered next lecture

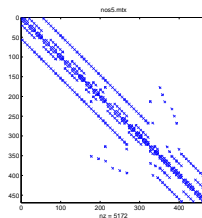
PCG examples - MatrixMarket



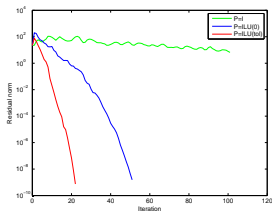
$n=1224, nz=56126$



Converged in 20-50 its



$n=468, nz=5172$



Converged in 30-50 its

General observations

- ▶ For a given amount of storage the numerical approach is almost always more effective than structural
 - ▶ The structural approach may store very small values and ignore much larger ones
 - ▶ The numerical approach will include fill-in but only if it is of significant size
 - ▶ The numerical approach requires hand-tuning
- ▶ The numerical approach is a more complex and costly algorithm but the reduction in iterations usually outweighs this cost

Non-symmetric linear systems

- ▶ GMRES is designed for non-symmetric, non-PD systems
 - ▶ The most general of the Krylov-subspace iterative methods
- ▶ There is no restriction on the preconditioner
- ▶ Can consider genuine left or right preconditioning
 - ▶ **Incomplete LU algorithms** for general non-symmetric linear systems

References

Templates for the Solution of Linear Systems

Richard Barrett *et al.*

[http://www.netlib.org/linalg/
html_templates/Templates.html](http://www.netlib.org/linalg/html_templates/Templates.html)

Iterative methods for sparse linear systems

Yousef Saad

<http://www-users.cs.umn.edu/~saad/books.html>