A Few Notes on Linear Solvers in IC-Ferst

James Percival

AMCG,
Department of Earth Science and Engineering
Imperial College London

IC-Ferst Training 2015

Imperial College London

Outline

Direct Methods

Iterative Methods
- Stationary methods
- Krylov subspace methods

Preconditioning

Multigrid

Halting

Parallel Solvers

Solver Failures & Troubleshooting

Numerical Solutions to PDEs

The majority of the work in generating an algorithm and set of discretizations to solve a system of PDEs numerically is aimed at reducing them to a series of linear equations. I.e. to a matrix problem like

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

or, in a more general form,

$$\mathrm{A}\boldsymbol{x} = \boldsymbol{b}.$$

Solving a linear system

Still actually get a solution for this. IC-Ferst uses linear solvers available from the PETSc framework (http://www.mcs.anl.gov/petsc/) to solve problem efficiently and in parallel.

Generally solution algorithms can be split into two sorts, direct methods and iterative methods.

Solving a linear system

1. Direct methods: Generate an explict representation of A^{-1} and then perform multiplication

$$\boldsymbol{x} = \mathrm{A}^{-1}\boldsymbol{b}.$$

These methods tend to be slow and memory inefficient, especially when only using the matrix once. Also difficult to do in parallel.

2. Iterative methods: given a previous guess, $\boldsymbol{x}_i$, generate a new guess, $\boldsymbol{x}_{i+1}$,such that the new residual

$$\boldsymbol{r}_{i+1} := \boldsymbol{b} - \mathrm{A}\boldsymbol{x}_{i+1}$$

is “smaller” than the old residual $\boldsymbol{r}_i$. Iterate over the algorithm until the residual error is small enough to be acceptable.

A Direct method: LU factorization

As an example of a direct method, lets look at LU factorization. This takes the original matrix, A, and creates two new ones, L & U such that

$$\mathrm{A} = \mathrm{LU},$$

where L is lower triangular and U is upper triangular

$$\mathrm{A} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix},$$

$$\mathrm{L} = \begin{bmatrix} a & 0 & 0 \\ d & e & 0 \\ g & h - \frac{gb}{a} & \dots \end{bmatrix}, \quad \mathrm{U} = \begin{bmatrix} 1 & b/a & c/a \\ 0 & 1 & \frac{ea(f - cd/a)}{ea - db} \\ 0 & 0 & 1 \end{bmatrix}$$

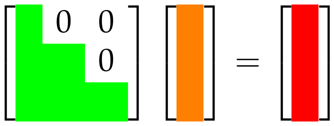
A Direct method: LU factorization

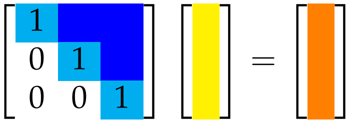
Now $\mathrm{A}^{-1} = \mathrm{U}^{-1}\mathrm{L}^{-1}$ and the triangular matrices can be quickly inverted by two elimination sweeps working from top-to-bottom or bottom to top.

$$L_{11}y_1 = b_1, \qquad x_N = y_N$$

$$L_{22}y_2 = b_2 - L_{21}y_1, \qquad x_{N-1} = y_{N-1} - U_{(N-1)N}x_N$$

$$y_n = \frac{1}{L_{nn}}\left(b_n - \sum_{i=1}^{n-1} L_{ni}y_i\right) \qquad x_n = y_n - \sum_{i=n+1}^{N} U_{ni}x_i$$





Iterative Methods

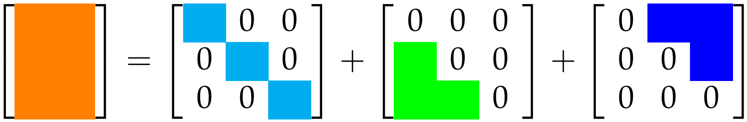
Iterative methods can be further subdivided into two groups:

1. Stationary methods: Split A into a bit which is easy to invert and a bit which is moved to the right hand side of the equation.
2. Krylov subspace methods: Get a "good" solution in a subspace smaller than the length of the vector $\boldsymbol{x}$, then extend the subspace until it spans all of $\boldsymbol{x}$.

Stationary method:Successive Over-Relaxation(sor)

For this method we use

$$\mathrm{A} = \mathrm{D} + \mathrm{L} + \mathrm{U}$$



and rewrite the original equation $A\boldsymbol{x} = \boldsymbol{b}$ into the form

$$\left(\mathrm{D} + \omega \mathrm{L}\right) \boldsymbol{x} = \omega \boldsymbol{b} - \left(\omega \mathrm{U} + \left(\omega - 1\right) \mathrm{D}\right) \boldsymbol{x}.$$

Where ω is a constant, $\omega > 1$. The iterative method is thus

$$\boldsymbol{x}_{k+1} = \left(\mathrm{D} + \omega \mathrm{L}\right)^{-1} \left[\omega \boldsymbol{b} - \left(\omega \mathrm{U} + \left(\omega - 1\right) \mathrm{D}\right) \boldsymbol{x}_k\right].$$

Knowing the best value of ω to use can be tricky.

Krylov subspace methods:Conjugate Gradients(cg)

If A is a symmetric $\left(\mathrm{A}^T=\mathrm{A}\right)$, positive definite matrix ($\boldsymbol{x}^T\mathrm{A}\boldsymbol{x}>0$ for all $\boldsymbol{x}\neq 0$), then the solution to $\mathrm{A}\boldsymbol{x}=b$ is also the value of $\boldsymbol{x}$ which minimises the quadratic form $I(x)=\frac{1}{2}\boldsymbol{x}^T\mathrm{A}\boldsymbol{x}-\boldsymbol{x}^T\boldsymbol{b}$.
Could just slide down gradient

$$\boldsymbol{r}_k=\mathrm{A}\boldsymbol{x}_k-\boldsymbol{b}$$

$$\boldsymbol{x}_{k+1}=\boldsymbol{x}_k-\alpha_k\boldsymbol{r}_k$$

Minimizing I implies

$$\boldsymbol{r}_k^T\boldsymbol{r}_{k+1}=\boldsymbol{r}_k^T\left(\mathrm{A}\boldsymbol{x}_{k+1}-\boldsymbol{b}\right)=\boldsymbol{r}_k^T\left(\mathrm{A}\boldsymbol{x}_k-\alpha\boldsymbol{r}_k\right)=0$$

$$\alpha=\frac{\boldsymbol{r}_k^T\mathrm{A}\boldsymbol{x}_k}{\boldsymbol{r}_k^T\boldsymbol{r}_k}.$$

Krylov subspace methods:Conjugate Gradients(cg)

Can do better! Use the new residual to make a seach direction “A conjugate” to all the previous ones

$$\boldsymbol{r}_k = \mathrm{A}\boldsymbol{x}_k - \boldsymbol{b}$$

$$\boldsymbol{p}_k = \boldsymbol{r}_k - \sum_{i<k} \frac{\boldsymbol{p}_i^T \boldsymbol{r}_k}{\boldsymbol{p}_i^T \mathrm{A}\boldsymbol{p}_i} \boldsymbol{p}_i$$

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \boldsymbol{p}_k$$

with

$$\alpha = \frac{\boldsymbol{r}_k^T \boldsymbol{r}_k}{\boldsymbol{p}_k^T \mathrm{A}\boldsymbol{p}_k},$$

which makes $\boldsymbol{r}_k^T \boldsymbol{r}_k = 0$.

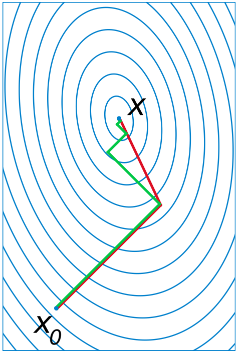


illustration by Oleg Alexandrov - via Wikimedia Commons

Ksp:Generalized minimum residual(gmres)

Conjugate gradients method relies on A being symmetric and positive (or negative) definite. In the general case where A is non-symmetric but an inverse exists then a similar approach can be used.

1. Work in space $\mathcal{K}_n = \text{span}\left\{\boldsymbol{r}_0, \mathrm{A}\boldsymbol{r}_0, A^2\boldsymbol{r}_0, \dots A^n\boldsymbol{r}_0\right\}$,
2. Find $\boldsymbol{x}_n \in \mathcal{K}_n$ which minimises $\|\boldsymbol{r}_n\|_{K_n}$.
3. If residual not small enough form $\mathcal{K}_{n+1} = \text{span}\left\{\boldsymbol{r}_0, \mathrm{A}\boldsymbol{r}_0, A^2\boldsymbol{r}_0, \dots A^{n+1}\boldsymbol{r}_0\right\}$.

In a practical implimentation, often impossible to work with $\mathcal{K}_n$ as n gets big. Answer: stop solve periodically and restart with

$$\boldsymbol{x}_0 = \boldsymbol{x}_{n_{max}},$$

$$\mathcal{K}_0 = \text{span}\left\{\boldsymbol{r}_{n_{max}}\right\}.$$

Convergence can be slow. Preconditioning is important.

Preconditioning: The Matrix Condition number

A square matrix can be written as

$$\mathrm{A} = \mathrm{PDP}^{-1}$$

where D is a diagonal matrix containing the eigenvalues of A and P has columns containing the eigenvectors of A. We define the matrix condition number as

$$\kappa := \frac{\max |\lambda_i|}{\min |\lambda_i|} \geq 1.$$

Generally low condition numbers mean a matrix is "easier" to solve numerically, that iterative methods convergence and that solutions are more robust to truncation error.

Preconditioning: The Preconditioning matrix

Need to solve $\mathrm{A}\boldsymbol{x} = \boldsymbol{b}$. Suppose we have another matrix M, “close” to A, but easier to invert. Can attempt to solve

$$\mathrm{M}^{-1}\mathrm{A}\boldsymbol{x} = \mathrm{M}^{-1}\boldsymbol{b}, \text{ (left preconditioning)}$$

or

$$\mathrm{A}\mathrm{M}^{-1}\mathrm{M}\boldsymbol{x} = \mathrm{M}^{-1}\boldsymbol{b} \text{ (right preconditioning)}$$

i.e. solve a two part system

$$M\boldsymbol{y} = \boldsymbol{b}, \qquad AM^{-1}\boldsymbol{y} = \boldsymbol{b},$$

$$M^{-1}A\boldsymbol{x} = \boldsymbol{y}. \qquad M\boldsymbol{x} = \boldsymbol{y}$$

Advantage is that matrix product is hopefully more “identity-like”, i.e. smaller condition number, so both problems are easier to solve.

Preconditioning: The Preconditioning matrix

Jacobi method

Precondition using $\mathrm{M} = \mathrm{D}$

SOR

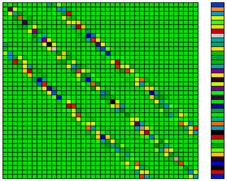
Precondition using $\mathrm{M} = \mathrm{D} + \omega \mathrm{L}$

Incomplete LU factorization

Like a lazy LU factorization. Do trickery to find an approximate $\tilde{\mathrm{L}}$ and $\tilde{\mathrm{U}}$ such that $\tilde{\mathrm{L}}\tilde{\mathrm{U}} \approx \mathrm{A}$ then use $\mathrm{M} = \tilde{\mathrm{L}}\tilde{\mathrm{U}}$ as preconditioner.

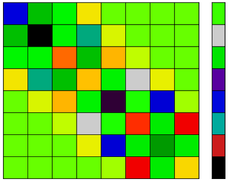
Multigrid

Preconditioning shows it is better to solve simple problems than more difficult ones. Multigrid takes this a step further: Big matrices are hard to solve for, so why not solve a smaller problem which is “like” the big one, and then use the update to correct the full problem?



$\Rightarrow$ (smoothing)

$\Leftarrow$ (prolongation)



Halting Criteria

Good:

Absolute tolerance acheived, $\|r_k\| \leq \tau_{abs}$,
Relative tolerance acheived, $\|r_k\| \leq \tau_{rel} \|r_0\|$,

Bad:

Maximum iterations reached, $k = k_{max}$
Matrix solve diverges $\|r_k\| \geq \tau_{div} \|r_{k-1}\|$,
NANs start appearing

Parallel Solvers

In principle a parallel matrix solve could work just like a serial one. In practice, multiplying out rows of a matrix is local and cheap, while exchanging column information requires expensive communication. PETSc modifies methods to be more efficient. This means

- Direct solvers don't work
- Parallel solvers may need more (fast) iterations for good answer.
- Serial & parallel solves can halt on different answers.

Imperial College
London

Solver failures

When a PETSc linear solve fails an error message usually gets reported

```
WARNING: Failed to converge.
PETSc did not converge for matrix solve of: DeltaP
Reason for non-convergence: KSP_DIVERGED_ITS
Number of iterations: 3000
Sending signal to dump and finish
```

This names the variable for which the solve failed, the halting criterion and the number of solve iterations successfully performed.

Solver failures

Possible reasons include

KSP_DIVERGED_ITS Specified limit on solver iterations reached

KSP_DIVERGED_DTOL The residual has increased too much between iterations

KSP_DIVERGED_NAN The problem PETSc has been asked to solve has stopped making sense.

KSP_DIVERGED_INDEFINITE_PC You've trying to solve a badly assymetric matrix using CG.

If you see a much longer error message involving PETSc then either you're trying to do something it can't (e.g. a direct solve in parallel), or something else has broken earlier.

Troubleshooting

- When trying to understand a solver failure, it's important to check that you're looking at the first thing which went wrong.
- If solver failures appear (almost) immediately, then it's possible there's a mistake in your input file.
- Solver failures can be the result of a bad mesh, either fixed or adaptive. Fix the mesh before trying to fix the solver.
- If you're really sure it's just the solver, then you can try
  - Increasing the maximum number of iterations
  - Adding an absolute tolerance limit (e.g. 1.0e-10)
  - Reducing the simulation timestep.

Final Summary

Robust option choice:

- `Iterative_method(gmres)`
  - `restarts: 30`
- `preconditioner(hypre)`
- `relative_error: 1.0e-7`
- `max_iterations: 1000`

References

- Y. Saad
  Iterative Methods for Sparse Linear Systems.
  SIAM 2003
- G. H. Golub & C. F. Van Loan
  Matrix Computations
  Johns Hopkins University Press 1996
- Fluidity Manual
  Applied Modelling & Computation Group
- Petsc Manual/online documentation
  http://www.mcs.anl.gov/petsc/