Programmazione Avanzata per il Calcolo Scientifico
Advanced Programming for Scientific Computing
Lecture title: Solution of large linear systems

Luca Formaggia

MOX
Dipartimento di Matematica
Politecnico di Milano

A.A. 2014/2015

The issue

Numerical methods for the solution of partial differential equations eventually lead to the solution of linear systems

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

where A is usually large and sparse

Sparse matrices

A sparse matrix is a matrix whose number of non-zero elements N_{nz} is $O(N)$, N being the size of the matrix. In other words is a matrix where the average number of non-zero elements per row is constant w.r.t. the matrix size. We will indicate this constant as m.

It is then convenient to store just the non-zero elements. The way of doing it, however, is not unique and several strategies are possible.

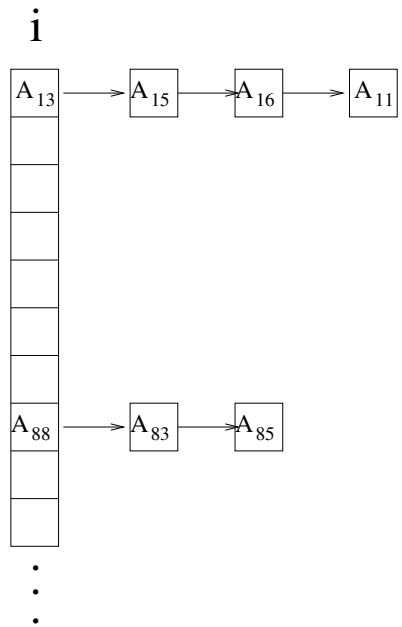
Storing sparse matrices

We do not intend to enter in the detail of sparse matrix storage, some information may be found in the book of Y. Saad.
We want to recall the general ideas. First of all we need to distinguish between

- dynamic storage system. Where it is possible to add new non-zero elements. This type of storage system is normally used only in the process of assembling the matrix. For efficiency reason is then converted to a static storage system.
- static storage system. The pattern of non-zero elements is fixed. It is not possible (or at least not easy) to add a new non-zero element.

Dynamic storage

It is based on the construction of a map $(i, j) \rightarrow \mathrm{A_{ij}}$. In practice the map is build either by using a hash table or a binary tree. The matrix can be stored either in row major or column major mode.



An example of hash table

Static storage

There are several formats, the most common ones are

- CSR Compressed sparse rows. Matrix is stored row major. As a consequence accessing an entire row is an $O(1)$ operation, while random access to an element is $O(m)$.
- CSC. Compressed sparse columns. Matrix is stored column major. Access to an entire row is $O(1)$.
- MSR. Modified sparse row. As CSR but more compact. Easy access to diagonal elements. Only square matrices.
- HB. Harwell-Boeing format. A rather old format, now less used.
- LINPACK banded format. The format used by LINPACK. It is a banded format so memory degrades for large bandwidths.

Variants are available for symmetric matrices that store only half matrix.

Matrix and vector norms

For a vector $\mathbf{v} \in \mathbb{R}^n$ and $p \in [1, \infty)$ we can define the p-norm

$$\|\mathbf{v}\|_p = \left(\sum_{i=1}^{n} |v_i|^p \right)^{1/p}$$

while

$$\|\mathbf{v}\|_\infty = \max_{i=1,\ldots,n} |v_i|$$

We can define the matrix p-norm induced by the corresponding vector norm, as

$$\|\mathrm{A}\|_\mathrm{p} = \sup_{\substack{\mathbf{v} \in \mathbb{R}^n \\ \mathbf{v} \neq \mathbf{0}}} \frac{\|\mathrm{A}\mathbf{v}\|_\mathrm{p}}{\|\mathbf{v}\|_\mathrm{p}}$$

The 2-norm

Of particular importance is the 2-norm which, for a vector, coincides with the so-called Euclidean norm. For the sake of notation we will omit the suffix 2.

For a matrix

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

where λ_{max} is the maximal eigenvalue.
If A is symmetric positive definite (spd) then

$$\|\mathrm{A}\| = \lambda_{\max}(\mathrm{A})$$

The M-norm

Given a symmetric and positive definite matrix M the M-dot product of vectors $\mathbf{v}$ and $\mathbf{w}$ is defined as

$$(\mathbf{v}, \mathbf{w})_{\mathrm{M}} = \mathbf{v}^T \mathrm{M} \mathbf{w} = \mathbf{w}^T \mathrm{M} \mathbf{v}$$

Correspondingly

$$\|\mathbf{v}\|_{\mathrm{M}} = \sqrt{(\mathbf{v}, \mathbf{v})_{\mathrm{M}}} = \sqrt{\mathbf{v}^T \mathrm{M} \mathbf{v}}$$

is the M-norm of $\mathbf{v}$.

The condition number

The p-condition number of an invertible matrix A is defined as

$$K_p(\mathrm{A}) = \|\mathrm{A}\|_\mathrm{p}\|\mathrm{A}^{-1}\|_\mathrm{p}$$

For a singular matrix $K_p(\mathrm{A}) = \infty$.

Of particular importance is the 2-condition number called simply condition number, which for a spd matrix can be written as

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

We have that $K_p(\mathrm{A}) \geq 1$.

The condition number represents, in some sense, the inverse of the distance of a matrix from the set of singular matrices.

The conditioning of the system

The condition number has a great importance in the analysis of linear system and in the behavior of the their numerical solution. If we perturb the matrix and the right hand side (and remember that by using floating point arithmetic we are almost always making a perturbation) the condition number is related on the resulting perturbation of the solution.

More precisely, let $\delta \mathrm{A}$ and $\delta \mathbf{b}$ be perturbations of A and $\mathbf{b}$ respectively, with $\|\delta \mathrm{A}\|/\|\mathrm{A}\| < \epsilon$ and $\|\delta \mathbf{b}\|/\|\mathbf{b}\| < \epsilon$, with ϵ small.

We have

$$\frac{\|\delta \mathbf{x}\|}{\|\mathbf{x}\|} \leq K(\mathrm{A}) \left(\|\delta \mathrm{A}\|/\|\mathrm{A}\| + \|\delta \mathbf{b}\|/\|\mathbf{b}\| \right) + o(\epsilon) \simeq K(A)\epsilon$$

Residual

If $\widehat{\mathbf{x}}$ is an approximate solution of $\mathrm{A}\mathbf{x} = \mathbf{b}$, we call residual $\mathbf{r} = \mathbf{r}(\widehat{\mathbf{x}})$ the quantity

$$\mathbf{r} = \mathbf{b} - \mathrm{A}\widehat{\mathbf{x}}$$

We have

$$\frac{\|\widehat{\mathbf{x}}\|}{\|\mathbf{x}\|} \leq K(A) \frac{\|\mathbf{r}\|}{\|\mathbf{b}\|}$$

The relative error is related to the (relative) residual through the condition number.

Direct methods

Direct methods for general matrices are mostly based on Gaussian elimination which is strictly related to the $LU(\mathcal{P})$ decomposition (LU with pivoting): for any matrix A there exist a permutation matrix $\mathcal{P}$, a lower triangular matrix L and a upper triangular matrix U such that

$$\mathcal{P}\mathrm{A} = \mathrm{LU}$$

If A is invertible then it admits an LU factorization (with $\mathcal{P}$ equal to the identity matrix) if and only if all its leading principal minors are non-zero. In this case, the factorization is unique if we impose that $L_{ii} = 1$ (or that $U_{ii} = 1$). Having an LU factorization the solution of the linear system reduces to

$$\mathbf{c} = \mathcal{P}\mathbf{b} \quad \mathrm{L}\mathbf{y} = \mathbf{c} \quad \mathrm{U}\mathbf{x} = \mathbf{y}$$

The last two systems can be solved with a $O(N^2)$ method, called forward and back-substitution.

spd matrices

A spd matrix admits an LU factorization (without pivoting). For a sdp matrix it is convenient to perform the Cholesky factorization (which is a particular case of LU):

$$\mathrm{A} = \mathrm{H}^{\mathrm{T}}\mathrm{H}$$

where H is upper-triangular (of course you may also do $\mathrm{A} = \mathrm{H}\mathrm{H}^{\mathrm{T}}$).

A note: the stability of the algorithm depends on the 2-norm of the matrix. If the matrix is badly conditioned round-off errors may produce null or negative pivots. In that case the algorithm fails. A possible remedy is to replace A by $\mathrm{A} + \alpha\mathrm{I}$ where α is a small number and I the identity matrix.

Disadvantages of standard Gauss elimination

Gauss elimination (LU factorization) is the most efficient direct method for general full matrices ($O(N^3)$). Moreover, if pivoting is performed its stability does not depend on the condition number and is, in general, good.

So it is an excellent algorithm, but unfortunately the L and U factors of a sparse matrix are not, in general, sparse!. We have the so-called fill-in.

For large sparse matrices standard Gauss elimination is thus unfeasible. Yet we have some alternatives....

Direct methods for sparse matrices

The fill-in depends on the bandwidth of the matrix, i.e. on how much the elements are clustered around the diagonal. If

$$A_{ij} \neq 0 \quad \text{only if } i - m_l \leq j \leq i + m_r$$

then the bandwidth is defined ad $b = 1 + m_l + m_r$.

Then a possibility is to reorder the unknowns (i.e. permuting the matrix) to minimize the bandwidth. There are several heuristic algorithm for this purpose, the most famous is the CuthillMcKee algorithm.

However, this can work fine only for spd matrices. For general matrices we may need to permute the matrix to avoid zero or too small pivots, potentially reintroducing a big fill-in.

Direct methods for sparse matrices

The most efficient algorithm for general sparse matrices is the multifrontal algorithm, implemented in the open-source libraries `UMFPACK` and `MUMPS`, and the `CSparse` package.
It is based on a two step procedure

- A symbolic analysis of the matrix. Only the graph of the matrix (i.e. the pattern of non-zero elements) is used, not the values. It identifies the order of elimination that minimizes fill-in.
- The elimination phase. The actual Gauss-type elimination is performed.

The algorithm does not make useless operations (like multiplication or addition of zeros), so it is rather efficient! Yet some fill-in is inevitable.

Note: If the size of your problem (and the memory of your computer) allows you to use direct methods (multifrontal) use them! In general, they beat in efficiency any iterative method!

Iterative refinement

We present a technique that in may cases may better the solution of a direct method when the matrix is ill conditioned (i.e. $K(\mathrm{A})$ is large, let's say above 10^5). We have seen that the stability of Gaussian elimination (and also multifrontal) does not depend on the condition number (or depends only weakly on it).

So we can try to eliminate the error induced by the round-off truncation by the following procedure

- For $i = 1, \ldots, M$ with M usually small (1 or 2) set $\mathbf{r}_1 = \mathbf{b}$ $\mathbf{x}_1 = \mathbf{0}$, and
  - Compute $\mathrm{A}\mathbf{e} = \mathbf{r}_\mathrm{i}$
  - $\mathbf{x}_{i+1} = \mathbf{x}_i + \mathbf{e}; \quad \mathbf{r}_{i+1} = \mathbf{b} - \mathrm{A}\mathbf{x}_{\mathrm{i}+1}$

and use the last value of $\mathbf{x}_i$ as the final solution.
If possible you should compute $\mathbf{r}_i$ using extended precision floating-point arithmetic.

A basic iterative method

We present the most basic iterative method, which encompasses some classical ones (Jacobi, Gauss-Siedel) because we can infer some basic properties that are relevant also for more advanced method.
Let M be an easily invertible matrix, called preconditioner (we will give details later). The preconditioned Richardson iteration reads:

Set $\mathbf{r}_0 = \mathbf{b}$, $\mathbf{x}_0 = \mathbf{0}$ and for $i = 0, \ldots$

- Compute $\mathbf{z}_i$ solution of $\mathrm{M}\mathbf{z}_i = \mathbf{r}_i$.
- $\mathbf{x}_{i+1} = \mathbf{x}_i + \alpha_i \mathbf{z}_i$

Where α_i are positive coefficients. If $\alpha_i = \alpha$ the method is stationary, in the simplest case $\alpha_i = 1$. If $\mathrm{M} = \mathrm{I}$ we have no preconditioning.

Preconditioner

Let's note that the preconditioned Richardson iteration is in-fact equal to the non preconditioned version on the modified system

$$\widetilde{\mathrm{A}}\mathbf{x} = \widetilde{\mathbf{b}}$$

with $\widetilde{\mathrm{A}} = \mathrm{M}^{-1}\mathrm{A}$ and $\widetilde{\mathbf{b}} = \mathrm{M}^{-1}\mathbf{b}$. $\widetilde{\mathrm{A}}$ will be called preconditioned matrix.

What we are presenting here is the left preconditioning. In general you have two matrices M^L and M^R and $\widetilde{\mathrm{A}} = (\mathrm{M}^L)^{-1}\mathrm{A}(\mathrm{M}^{\mathrm{R}})^{-1}$

Often M is not inverted: the system $\mathrm{M}\mathbf{z} = \mathbf{r}$ is solved instead.

The role of the preconditioner

To understand the role of the preconditioner let's take $\alpha_i = 1$ for simplicity and write the Richardson iteration in terms of the preconditioned system.

$$\mathbf{x}_{i+1} = \mathbf{x}_i + (\widetilde{\mathbf{b}} - \widetilde{\mathrm{A}}\mathbf{x}_i) = \mathbf{x}_i + \widetilde{\mathbf{r}}_i$$

By simple manipulation we note that $\widetilde{\mathbf{r}}_{i+1} = (\mathrm{I} - \widetilde{\mathrm{A}})^{\mathrm{i}+1}\widetilde{\mathbf{r}}_0$ So

$$\widetilde{\mathbf{r}}_{i+1} = \mathcal{Z}_{i+1}(\widetilde{\mathrm{A}})\widetilde{\mathbf{r}}_0 = \mathcal{Z}_{i+1}(\widetilde{\mathrm{A}})\widetilde{\mathbf{b}}$$

where $\mathcal{Z}_i(y) = \sum_{k=0}^{i}(1-y)^k$ is a polynomial of degree i with $\mathcal{Z}_i(0) = 1$. And

$$\mathbf{x} - \mathbf{x}_{i+1} = \mathcal{Z}_{i+1}(\widetilde{\mathrm{A}})\mathbf{x}$$

Important Note: Here and in the following we often take for simplicity $\mathbf{x}_0 = \mathbf{0}$. This does not cause any loss of generality since the case $\mathbf{x}_0 \neq \mathbf{0}$ is easily recovered by a simple shift.

What do we learn from this simple example?

First of all by applying Cauchy inequality

$$\|\widetilde{\mathbf{r}}_{i+i}\| \leq \|\mathrm{I} - \widetilde{\mathrm{A}}\|\|\widetilde{\mathbf{r}}_{\mathrm{i}}\|$$

so a sufficient condition for convergence is $\|\mathrm{I} - \widetilde{\mathrm{A}}\| < 1$.

Let's assume that $\widetilde{\mathrm{A}}$ has N eigenvectors $\mathbf{w}_i$ with corresponding eigenvalues λ_i. We can expand $\widetilde{\mathbf{r}}_0 = \widetilde{\mathbf{b}} = \sum_{i=1}^{N} \gamma_i \mathbf{w}_i$. Using the well known fact that $\mathcal{Z}(\widetilde{\mathrm{A}})\mathbf{w}_i = \mathcal{Z}(\lambda_i)\mathbf{w}_i$ we have

$$\widetilde{\mathbf{r}}_i = \mathcal{Z}(\widetilde{\mathrm{A}})\widetilde{\mathbf{r}}_0 = \sum_{j=1}^{N} \gamma_j \mathcal{Z}_i(\lambda_j)\mathbf{w}_j$$

Thus $\|\widetilde{\mathbf{r}}_i\| \leq \max_{j=1}^{N} |\mathcal{Z}_i(\lambda_J)| \sum_{j=1}^{N} |\gamma_j| \|\mathbf{w}_j\|$, which shows that a good preconditioner should generate a $\mathcal{Z}_i(x) = \sum_{k=1}^{i} (1-x)^i$ with small values in correspondence to the eigenvalues of the preconditioned matrix.

A question for the mathematician + engineers

If you can choose a set of real numbers (it works for symmetric matrices since the eigenvalues are real)

$$\lambda_{min} = x_1 < x_2 < \ldots < x_N = \lambda_{max}$$

with the objective of keeping $\mathcal{Z}_i(x) = \sum_{k=1}^{i}(1-x)^i$ "small" for $x \in [\lambda_{min}, \lambda_{max}]$ what would you choose?

Some classic results

The stationary Richardson iteration converges if and only if

$$\max_{i=1}^{N} \frac{2\,\mathrm{Re}(\widetilde{\lambda}_i)}{\alpha|\widetilde{\lambda}_i|} > 1$$

If A and $\widetilde{\mathrm{A}}$ are both spd, choosing $\alpha = 2/(\widetilde{\lambda}_{max} + \widetilde{\lambda}_{min})$ the method is monotonically convergent on the 2-norm and

$$\|\mathbf{x} - \mathbf{x}_{i+1}\| \leq \frac{K(\widetilde{\mathrm{A}}) - 1}{K(\widetilde{\mathrm{A}}) + 1}\|\mathbf{x} - \mathbf{x}_i\|$$

Which show that the role of the preconditioner is to reduce the condition number of the original matrix, i.e. we would like $K(\mathrm{M}^{-1}\mathrm{A}) << \mathrm{K}(\mathrm{A})$.

The Krylov space

We may note that $\mathcal{Z}_i(\mathrm{A})\mathbf{r}_0 = \sum_{\mathrm{k}=1}^{\mathrm{i}}(\mathrm{I}-\mathrm{A})^{\mathrm{i}}\mathbf{r}_0$ is a vector that may be written as

$$\mathcal{Z}_i(\mathrm{A})\mathbf{r}_0 = \mathbf{r}_0 + \mathrm{a}_1\mathrm{A}\mathbf{r}_0 + \mathrm{a}_2\mathrm{A}^2\mathbf{r}_0 + \dots$$

In other words $\mathcal{Z}_i(\mathrm{A})\mathbf{r}_0 \in \mathrm{span}(\mathbf{r}_0, \mathrm{A}\mathbf{r}_0, \dots, \mathrm{A}^{\mathrm{i}}\mathbf{r}_0))$

The space

$$\mathcal{K}_i = \mathcal{K}_i(\mathrm{A}; \mathbf{r}_0) = \mathrm{span}(\mathbf{r}_0, \mathrm{A}\mathbf{r}_0, \dots, \mathrm{A}^{\mathrm{i}-1}\mathbf{r}_0)$$

is the Krylov space of dimension i generated by A and $\mathbf{r}_0$. It plays a fundamental role in iterative methods for linear system!

An interesting property. If $\mathbf{x}_i \in \mathcal{K}_i$ then $\mathbf{r}_i \in \mathcal{K}_{i+1}$. Indeed $\mathbf{r}_i = \mathbf{b} - \mathrm{A}\mathbf{x}_{\mathrm{i}} = \mathbf{r}_0 + \mathrm{A}\mathbf{x}_{\mathrm{i}}$.

Krylov based methods

A vector $\mathbf{z} \in \mathcal{K}_k(\mathrm{A}, \mathbf{y})$ may be expressed in the form

$$\mathbf{z} = \sum_{j=0}^{i-1} \alpha_j \mathrm{A}^{\mathrm{j}} \mathbf{y}$$

that is $\mathbf{z} = q_{i-1}(\mathrm{A})\mathbf{y}$, where $q_{i-1}(x)$ is a polynomial of degree $i-1$ such that $q_{i-1}(0) = 1$.

So one may ask the question: how to choose q_{i-1} and $\mathbf{y}$ so that the solution $\mathbf{x}$ is "best approximated" (in some sense) by a member of $\mathcal{K}_i(\mathrm{A}, \mathbf{y})$?

The case of symmetric positive definite matices

Exploiting the Cayley-Hamilton theorem one may derive that, given an initial vector $\mathbf{x}^0$ and the residual $\mathbf{r}^{(0)} = \mathbf{b} - \mathrm{A}\mathbf{x}^0$ the solution $\mathbf{x}$ of $\mathrm{A}\mathbf{x} = \mathbf{b}$ satisfies

$$\mathbf{x} - \mathbf{x}^{(0)} = \hat{q}_{n-1}(A)\mathbf{r}^{(0)}$$

where $\hat{q}_{n-1}(A)$ is a particular polynomial defined as

$$\hat{q}_{n-1}(x) = \det(\mathrm{A})^{-1}\left[(-1)^{\mathrm{n}-1}\mathrm{x}^{\mathrm{n}-1} + \ldots + (\sum_{\mathrm{i}=1}^{\mathrm{n}} \prod_{\mathrm{j}=1,\mathrm{j}\neq\mathrm{i}}^{\mathrm{n}} \lambda_{\mathrm{i}})\right]$$

We deduce that a good choice is $\mathbf{y} = \mathbf{r}^{(0)}$.
It remains to decide how to choose a sequence of polynomials q_k so that the iterates $\mathbf{x}^{(k)} = \mathbf{x}^{(0)} + q_{k-1}(\mathrm{A})\mathbf{r}^{(0)} \in \mathbf{x}^{(0)} + \mathcal{K}(\mathrm{A}, \mathbf{r}^{(0)})$ converge rapidly to $\mathbf{x}$.

Conjugate Gradient method

We fist consider the case of symmetric positive definite matrices.

For simplicity we set $\mathbf{x}^{(0)} = \mathbf{0}$ The conjugate gradient method computes q_{k-1} such that

$$\|\mathbf{x} - \mathbf{x}^{(k)}\|_{\mathrm{A}} = \mathrm{argmin}_{\mathbf{z} \in \mathcal{K}(\mathrm{A}, \mathbf{r}^{(0)})} \|\mathbf{x} - \mathbf{z}\|_{\mathrm{A}}$$

If A is the stiffness matrix corresponding to the Poisson problem discretized by finite elements with unknown u_h, this corresponds the best approximation over the Krylov subspace, i.e. $u_h^{(k)} - u_h$ is the best approximation in the energy norm.

The CG algorithm

Algorithm 2.1: THE CONJUGATE GRADIENT METHOD

Choose $\mathbf{u}^{(0)}$, compute $\mathbf{r}^{(0)} = \mathbf{f} - A\mathbf{u}^{(0)}$, set $\mathbf{p}^{(0)} = \mathbf{r}^{(0)}$

for $k = 0$ until convergence do

$\alpha_k = \langle \mathbf{r}^{(k)}, \mathbf{r}^{(k)} \rangle / \langle A\mathbf{p}^{(k)}, \mathbf{p}^{(k)} \rangle$

$\mathbf{u}^{(k+1)} = \mathbf{u}^{(k)} + \alpha_k \mathbf{p}^{(k)}$

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

<Test for convergence>

$\beta_k = \langle \mathbf{r}^{(k+1)}, \mathbf{r}^{(k+1)} \rangle / \langle \mathbf{r}^{(k)}, \mathbf{r}^{(k)} \rangle$

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

enddo

Note: here **u** is used instead of **x** and **f** instead of **b**.

The GC method not only it has interesting convergence properties, as we will see, but also is efficient in terms of memory.

Preconditioned conjugate gradient method

The (preconditioned) conjugate gradient method is the method of choice for spd systems. Given a sdp preconditioning matrix M and its Chowlesky factorization $\mathrm{M} = \mathrm{H}\mathrm{H}^T$ the preconditioned iterates are equivalent to solve the preconditioned system

$$\widetilde{\mathrm{A}}\mathbf{v} = \mathrm{H}^{-1}\mathbf{b}, \quad \mathbf{v} = \mathrm{H}^T\mathbf{x}$$

where $\widetilde{\mathrm{A}} = \mathrm{H}^{-1}\mathrm{A}\mathrm{H}^{-\mathrm{T}}$.
In fact the algorithm does not require to build H explicitly!

The algorithm

Algorithm 2.2: THE PRECONDITIONED CONJUGATE GRADIENT METHOD

Choose $\mathbf{u}^{(0)}$, compute $\mathbf{r}^{(0)} = \mathbf{f} - A\mathbf{u}^{(0)}$, solve $M\mathbf{z}^{(0)} = \mathbf{r}^{(0)}$, set $\mathbf{p}^{(0)} = \mathbf{z}^{(0)}$

`for` $k = 0$ `until` convergence `do`

$\alpha_k = \langle \mathbf{z}^{(k)}, \mathbf{r}^{(k)} \rangle / \langle A\mathbf{p}^{(k)}, \mathbf{p}^{(k)} \rangle$

$\mathbf{u}^{(k+1)} = \mathbf{u}^{(k)} + \alpha_k \mathbf{p}^{(k)}$

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

<Test for convergence>

Solve $M\mathbf{z}^{(k+1)} = \mathbf{r}^{(k+1)}$

$\beta_k = \langle \mathbf{z}^{(k+1)}, \mathbf{r}^{(k+1)} \rangle / \langle \mathbf{z}^{(k)}, \mathbf{r}^{(k)} \rangle$

$\mathbf{p}^{(k+1)} = \mathbf{z}^{(k+1)} + \beta_k \mathbf{p}^{(k)}$

`enddo`

Preconditioned CG

The fact that we do not have to construct H is due to the fact that even if the left preconditioned matrix $\widetilde{\mathrm{A}} = \mathrm{M}^{-1}\mathrm{A}$ not not symmetric, if M is spd matrix it is symmetric with respect to the scalar product $(\mathbf{x}, \mathbf{y})_{\mathrm{M}} = \mathbf{x}^T \mathrm{M}\mathbf{y}$. Indeed

$$(\mathbf{x}, \widetilde{\mathrm{A}}\mathbf{y})_{\mathrm{M}} = \mathbf{x}^T \mathrm{M}\mathrm{M}^{-1}\mathrm{A}\mathbf{y} = (\mathrm{M}^{-1}\mathrm{A}\mathbf{x})^{\mathrm{T}}\mathrm{M}\mathbf{y} = (\widetilde{\mathrm{A}}\mathbf{x}, \mathbf{y})_{\mathrm{M}}$$

Thus, the basic algorithm has been modified for the preconditioned case by changing the scalar products.

Convergence properties of preconditioned CG

The i-th iteration of the conjugate gradient satisfies

$$\mathbf{x}_i - \mathbf{x} = \mathcal{Q}_i(\widetilde{\mathrm{A}})(\mathbf{x}_0 - \mathbf{x})$$

where $\mathcal{Q}_i$ minimizes $\|\mathbf{x}_i - \mathbf{x}\|_A$ among all polynomials $\mathcal{W}_i$ of degree i and such that $\mathcal{W}_i(0) = 1$.
As a consequence we have this interesting property: let $\mathbf{w}_i$ be the i-th eigenvector of $\widetilde{\mathrm{A}}$ and λ_i the corresponding eigenvalue. We can always write $\mathbf{b} = \sum_i \gamma_i \mathbf{w}_i$ (γ_i may be zero). Then

$$\|\mathbf{x}_i - \mathbf{x}\|_{\mathrm{A}}^2 = \sum_{j=1}^{N} \frac{\gamma_j^2}{\lambda_j} \mathcal{Q}_i^2(\lambda_j) \leq \max_{\substack{j=1\ldots,N \\ \gamma_j \neq 0}} \mathcal{W}_i^2(\lambda_j) \sum_{j=1}^{N} \frac{\gamma_j^2}{\lambda_j}$$

where $\mathcal{W}_i$ is any polynomial of degree i that takes the value 1 at 0.

The consequences

If m is the number of $\gamma_i \neq 0$ we have that In exact arithmetic the CG algorithm converges to the exact solution in m steps.

But more importantly: If l is the number of different eigenvalues of $\widetilde{\mathrm{A}}$ then in exact arithmetic the CG algorithm converges to the exact solution in l steps.

Thus the role of the preconditioner is to cluster the eigenvalues of the preconditioned matrix.

By exploiting some properties of polynomials we have also the classical result

$$\|\mathbf{x}_i - \mathbf{x}\|_{\mathrm{A}}^2 = 2 \left(\frac{\sqrt{K(\widetilde{\mathrm{A}})} - 1}{\sqrt{K(\widetilde{\mathrm{A}})} + 1} \right)^i \|\mathbf{x}_0 - \mathbf{x}\|_{\mathrm{A}}^2$$

Stopping criteria

Normally the stopping criteria is given in terms of the relative residual:

$$\|\mathbf{r}^{(k)}\|/\|\mathbf{r}^{(0)}\| \leq \epsilon$$

Since

$$\|\mathbf{e}^{(k)}\|_{\mathrm{A}}/\|\mathbf{e}^{(0)}\|_{\mathrm{A}} = \sqrt{\mathrm{cond}(\mathrm{A})}\|\mathbf{r}^{(k)}\|/\|\mathbf{r}^{(0)}\|$$

the relative residual is linked to the error in the A-norm, which for system arising from the solution of elliptic problems by finite elements is related to the error in the energy norm.

Practical rule: the error should be of the same order of that introduced by the discretization. For linear finite elements we should stop when $\|\mathbf{e}^{(k)}\|_{\mathrm{A}} \leq \tau h$, where $\tau > 0$ is a constant (related to $\|D^2 u\|/\|\nabla u\|$ and the interpolation constant). Using the previous relation and $\mathrm{cond}(\mathrm{A}) = \mathrm{Ch}^{-2}$ for linear finite elements

$$\|\mathbf{r}^{(k)}\|/\|\mathbf{r}^{(0)}\| \leq \epsilon h.$$

where ϵ is a user selected tolerance (for instance 10^{-4}).

Singular problems are not a problem

If

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

has an infinity of solutions, as for a purely Neumann problem in the Poisson equation, provided that the preconditioning matrix M is non-singular the PCG method converges to a solution.

Which solution depends on $\mathbf{x}^{(0)}$.

For a purely Neuman problem for the Poisson equation the rate of convergence is related to the effective condition number λ_n/λ_2, $0 = \lambda_1 < \lambda_2 \ldots \leq \lambda_n$ being the eigenvalues of A.

The Lanczos algorithm and MINRES

We now describe a method suitable for symmetric indefinite matrices. The Lanczos algorithm constructs an orthogonal basis for $\mathcal{K}_k(\mathrm{A}, \mathbf{r}^{(0)})$ $\mathbf{v}^i, i = 1, \ldots k$ by setting $\mathbf{v}^{(0)} = \mathbf{0}$, $\mathbf{v}^{(1)} = \mathbf{r}^{(0)}$ and using the recurrence

$$\gamma_{j-1}\mathbf{v}^{(j+1)} = \mathrm{A}\mathbf{v}^{(\mathrm{j})} - \delta_{\mathrm{j}}\mathbf{v}^{(\mathrm{j})} - \gamma_{\mathrm{i}}\mathbf{v}^{(\mathrm{j}-1)}$$

where $\delta_i = (\mathrm{A}\mathbf{v}^{(\mathrm{j})}, \mathbf{v}^{(\mathrm{j})})$ and γ_{j+1} is such that $\|\mathbf{v}^{(j+1)}\| = 1$. If

$$\mathrm{V}_k = [\mathbf{v}^{(1)}, \ldots \mathbf{v}^{(k)}] \quad \mathrm{T}_k = \mathrm{tridiag}(\gamma_j, \delta_j, \gamma_{j+1}), j = 1, \ldots, k$$

then the relation may be written as

$$\mathrm{AV_k} = \mathrm{V_k}\mathrm{T_k} + \gamma_{\mathrm{k}+1}[\mathbf{0}, \ldots, \mathbf{0}, \mathbf{v}^{(\mathrm{k}+1)}]$$

and, consequently, $\mathrm{V}_k^T \mathrm{AV_k} = \mathrm{T_k}$.

MINRES

Lanczos algorithm has been originally devised to construct approximations of the eigenvalues of A as the eigenvalues of T_k yet it is the basis for the MINRES method.

The idea is to minimize $\|\mathbf{r}^{(k)}\|$ using a least square technique, and the availability of an orthonormal basis of the Krylov space allow to have a stable procedure via a QR factorization. We omit the details and give just the algorithm.

Algorithm 2.4: THE MINRES METHOD

$\mathbf{v}^{(0)} = \mathbf{0}, \mathbf{w}^{(0)} = \mathbf{0}, \mathbf{w}^{(1)} = \mathbf{0}$

Choose $\mathbf{u}^{(0)}$, compute $\mathbf{v}^{(1)} = \mathbf{f} - A\mathbf{u}^{(0)}$, set $\gamma_1 = \|\mathbf{v}^{(1)}\|$

Set $\eta = \gamma_1, s_0 = s_1 = 0, c_0 = c_1 = 1$

`for` $j = 1$ `until` convergence `do`

$\mathbf{v}^{(j)} = \mathbf{v}^{(j)}/\gamma_j$

$\delta_j = \langle A\mathbf{v}^{(j)}, \mathbf{v}^{(j)} \rangle$

$\mathbf{v}^{(j+1)} = A\mathbf{v}^{(j)} - \delta_j \mathbf{v}^{(j)} - \gamma_j \mathbf{v}^{(j-1)}$ (Lanczos process)

$\gamma_{j+1} = \|\mathbf{v}^{(j+1)}\|$

$\alpha_0 = c_j \delta_j - c_{j-1} s_j \gamma_j$ (update QR factorization)

$\alpha_1 = \sqrt{\alpha_0^2 + \gamma_{j+1}^2}$

$\alpha_2 = s_j \delta_j + c_{j-1} c_j \gamma_j$

$\alpha_3 = s_{j-1} \gamma_j$

$c_{j+1} = \alpha_0/\alpha_1;\ s_{j+1} = \gamma_{j+1}/\alpha_1$ (Givens rotation)

$\mathbf{w}^{(j+1)} = (\mathbf{v}^{(j)} - \alpha_3 \mathbf{w}^{(j-1)} - \alpha_2 \mathbf{w}^{(j)})/\alpha_1$

$\mathbf{u}^{(j)} = \mathbf{u}^{(j-1)} + c_{j+1} \eta \mathbf{w}^{(j+1)}$

$\eta = -s_{j+1} \eta$

<Test for convergence>

`enddo`

GMRES

The Generalized Minimal Residual Method has been developed by Y. Saad and M.H. Schultz in 1986 as an attempt to find a good iterative method for general matrices.

At each iteration GMRES minimizes $\|\mathbf{r}_i\|$ over $\mathcal{K}_i(\mathrm{A}; \mathbf{r}_0)$. To do so it has to perform a projection procedure at each step using a basis of $\mathcal{K}_i(\mathrm{A}; \mathbf{r}_0)$, constructed via a Lanczos procedure.

We omit the details which are rather technical, just to say that the basis has now to be stored explicitly. It means that at each iteration we need to store an additional vector. Moreover at each iteration the projection procedure is more costly.

Thus, often one uses a restarted version of the algorithm, called GMRES(m): at every m iterations the algorithm is restarted taking as initial value the last computed value.
We will illustrate the not preconditioned version. With simple modifications one has the preconditioned version.

Algorithm 4.1: THE GMRES METHOD

Choose $\mathbf{u}^{(0)}$, compute $\mathbf{r}^{(0)} = \mathbf{f} - F\mathbf{u}^{(0)}$, $\beta_0 = \|\mathbf{r}^{(0)}\|$, $\mathbf{v}^{(1)} = \mathbf{r}^{(0)}/\beta_0$

`for` $k = 1, 2, \ldots$ `until` $\beta_k < \tau\beta_0$ `do`

  $\mathbf{w}_0^{(k+1)} = F\mathbf{v}^{(k)}$

  `for` $l = 1$ `to` k `do`

    $h_{lk} = \langle \mathbf{w}_l^{(k+1)}, \mathbf{v}^{(l)} \rangle$

    $\mathbf{w}_{l+1}^{(k+1)} = \mathbf{w}_l^{(k+1)} - h_{lk}\mathbf{v}^{(l)}$

  `enddo`

  $h_{k+1,k} = \|\mathbf{w}_{k+1}^{(k+1)}\|$

  $\mathbf{v}^{(k+1)} = \mathbf{w}_{k+1}^{(k+1)}/h_{k+1,k}$

  Compute $\mathbf{y}^{(k)}$ such that $\beta_k = \left\| \beta_0\mathbf{e}_1 - \widehat{H}_k\mathbf{y}^{(k)} \right\|$ is minimized,

    where $\widehat{H}_k = [h_{ij}]_{1 \le i \le k+1, 1 \le j \le k}$

`enddo`

$\mathbf{u}^{(k)} = \mathbf{u}^{(0)} + V_k\mathbf{y}^{(k)}$

Convergence of GMRES and GMRES(m)

Let $\widetilde{\mathrm{A}}$ be diagonalizable, i.e. we may write $\widetilde{\mathrm{A}} = \mathrm{X}^{-1}\mathrm{D}\mathrm{X}$, with $\mathrm{D} = \mathrm{diag}(\lambda_1, \ldots, \lambda_N)$. We define

$$e_i = \min_{\substack{\mathcal{W} \in \mathbb{P}_i \\ \mathcal{W}(0)=1}} \max_{i=1,\ldots,N} |\mathcal{W}(\lambda_i)|$$

We have that

$$\|\widetilde{\mathbf{r}}_i\| \leq e_i K(\mathrm{X}) \|\widetilde{\mathbf{r}}_0\|.$$

Therefore, if A is diagonalizable and has l distinct eigenvectors, in exact arithmetic GMRES converges in l iterations.

The convergence of GMRES(m) is not guaranteed in the general case. For a theory of convergence of GMRES see V. Simoncini et al., 1996.

Other methods

There are many other methods. We mention

- BICGStab. A method for general matrices with low memory requirement.
- ORTHOMIN For historical reasons it is very used in reservoir simulations. Yet, there is little or no advantage over GMRES. Also ORTHOMIN suffers the increase of memory storage at each iteration and has a restarted version.

Some classic preconditioners

- Jacobi or diagonal preconditioner: $\mathrm{M} = \mathrm{diag}(A)$
- Block Jacobi. If we are able to partition the matrix as

$$\mathrm{A} = \begin{bmatrix} A_{11} & A_{12} & \dots \\ A_{21} & A_{22} & \dots \\ \dots & \dots & \dots \end{bmatrix}$$

  with A_{ij} "small" matrices (that can be inverted by a direct method), we can choose $\mathrm{M} = \mathrm{diag}(A_{11}, A_{22}, \dots)$. Often used in a parallel setting.
- Gauss Siedel. If $\mathrm{A} = \mathrm{D} - \mathrm{E} - \mathrm{F}$ with D $-E$ and $-F$ the diagonal, strict lower and strict upper part, respectively. Then the GS preconditioner is $P = D - E$.

Some classic preconditioners

- Symmetric Gauss Siedel. Since the traditional Gauss Siedel gives a non-symmetric M for a symmetric A, a variant is $P = (D - E)D^{-1}(D - F)$, which is symmetric if A is symmetric.
- SOR. Successive over relaxation preconditioner. $\mathrm{M} = \frac{1}{\omega}D - E$, where ω is a relaxation parameter that has to be chosen carefully. If A is spd, if we have an estimate of $\xi = \rho(\mathrm{I} - \mathrm{D}^{-1}\mathrm{A}) = \max_{\mathrm{i}} |\lambda_{\mathrm{i}}((\mathrm{I} - \mathrm{D}^{-1}\mathrm{A}))|$ the optimal value for ω is $2/(1 + \sqrt{1 - \xi^2})$

Some less classic preconditioners

- Incomplete LU factorization, also called ILU(n). A very popular preconditioner. An approximate factorization $\mathrm{A} \simeq \hat{\mathrm{A}} = \hat{\mathrm{L}}\hat{\mathrm{U}}$ is constructed with the property that $\hat{\mathrm{A}}$ has at most $N_{nz} + nN$ non-zero entries (n is called fill-in factor). Then, $\mathrm{M} = \hat{\mathrm{L}}\hat{\mathrm{U}}$. It is based on LU factorization where elements are dropped. There are different strategies to do it. When it works can be very effective. Unfortunately it is not guaranteed to work always. If $n = 0$ the factored matrix has the same non-zero entries than the original one.

  A variant, called ILUT(n,τ) allows to prescribe a threshold tolerance for the selection of entries to be dropped.
  For spd matrices one can perform an Incomplete Cholesky Factorization.

IC(0)

Algorithm 2.3: IC(0) FACTORIZATION

for $i = 1$ until n do
  $m = \min\{k \mid a_{ik} \neq 0\}$
  for $j = m$ until $i - 1$ do
    if $a_{ij} \neq 0$ then
      $l_{ij} \leftarrow a_{ij} - \sum_{k=m}^{j-1} l_{ik} l_{jk} / l_{jj}$
    endif
  enddo
  $l_{ii} = \left(a_{ii} - \sum_{k=m}^{i-1} l_{ik} l_{ik} \right)^{1/2}$
enddo

Some less classic preconditioners

- Polynomial preconditioners. $\mathrm{M}^{-1} = p(\mathrm{A})$ where p is a polynomial (of low degree). If we set $\mathrm{A} = \mathrm{D} - \mathrm{C}$ if $\rho(CD^{-1}) < 1$ a possible polynomial preconditioner is given by

$$\mathrm{M}^{-1} = D^{-1}(\mathrm{I} + \mathrm{CD}^{-1} + (\mathrm{CD}^{-1})^2 2 + \ldots)$$

  Note that we can avoid computing the inverse of the preconditioner explicitly.

- Multilevel preconditioners. Let $I_N^M : \mathbb{R}^N \to \mathbb{R}^M$ be a restriction operator such that for a $\mathbf{v} \in \mathbb{R}^N$ we have $I_N^M \mathbf{v} = \mathbf{w} \in \mathbb{R}^M$, with $M << N$. Let $I_M^N = (I_N^M)^T$ Then we may choose $\mathrm{M}^{-1} = I_M^N [I_M^N \mathrm{A} \mathrm{I}_\mathrm{N}^\mathrm{M}]^{-1} \mathrm{I}_\mathrm{N}^\mathrm{M}$. Matrix $\mathrm{A}_\mathrm{M} = \mathrm{I}_\mathrm{M}^\mathrm{N} \mathrm{A} \mathrm{I}_\mathrm{N}^\mathrm{M}$ is of size M, if it is sufficiently small we can use a direct method to solve the system in A_M, otherwise we apply the same technique recursively. There are several ways of building the restriction operator, the simplest one is agglomeration.

Some less classic preconditioners

- Domain decomposition preconditioners. They are derived from the differential problem by partitioning the domain into subdomains Ω_i and transforming the original differential problems in problems on each subdomain plus interface condition. The preconditioner is then obtained by solving the local problems independently (in a sort of Jacobi or Gauss-Siedel fashion).

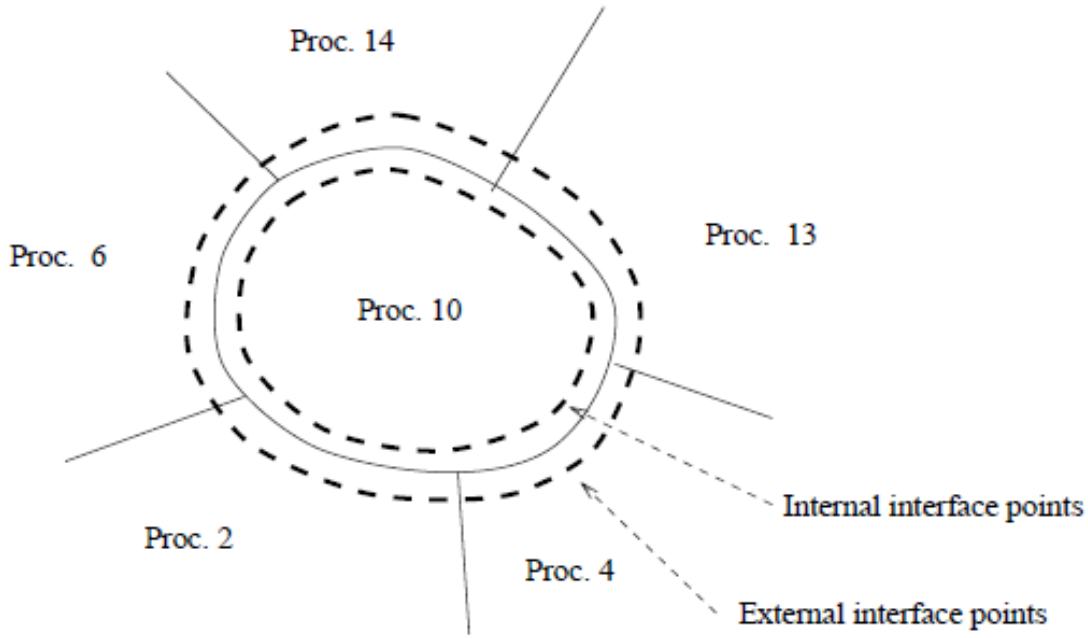
  Their detailed explanation goes beyond this notes, if you are interested you may look at the books of Quarteroni and Valli (1999), Toselli Widlund (2005) and Smith, Bjorstad, Gropp (2004).

Parallelization issues

Iterative methods can be implemented in parallel architecture more easily than direct method (even if parallel version of the multifrontal method are available).
The main ingredient are

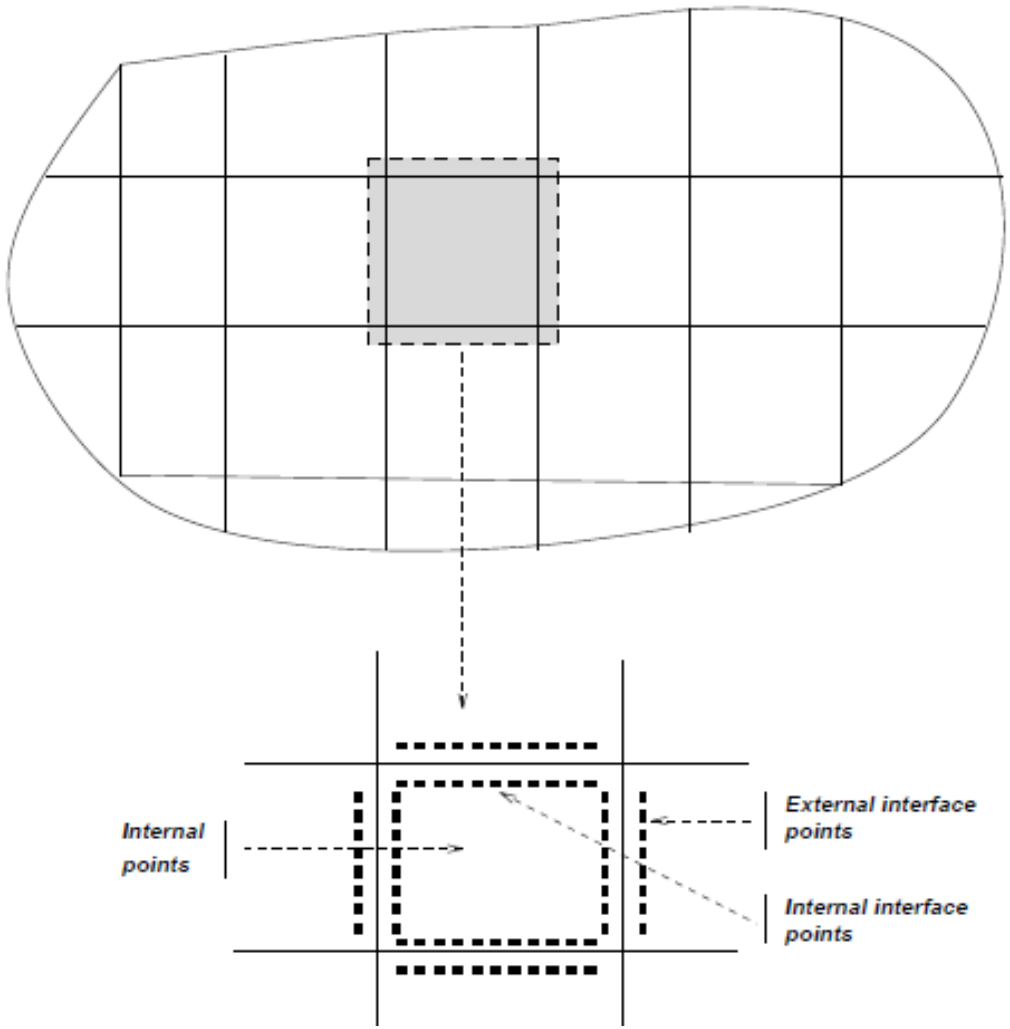
- Preconditioner setup
- Matrix-by vector operation
- Vector updates
- Dot products
- Preconditioning operations

Matrix assembly



Mesh is distributed among processors. The interface points are associated to dofs shared among processors and are used for interprocessor communication.
A strategy is that each processor assemble the rows of the matrix corresponding to the internal dofs. The interface nodes are needed to complete the local assembly.

Layout of dofs



Parallel matrix and vector operation

Since each processor stores the rows corresponding to the internal dofs and the value of the internal+interface dofs (communication needed!) every processor may compute its contribution to $\mathrm{A}\mathbf{x} = \mathbf{b}$.

The scalar product of a vector is also easily parallelizable: each processor computes its own part and then communicates by a broadcast its contribution to the other processors.

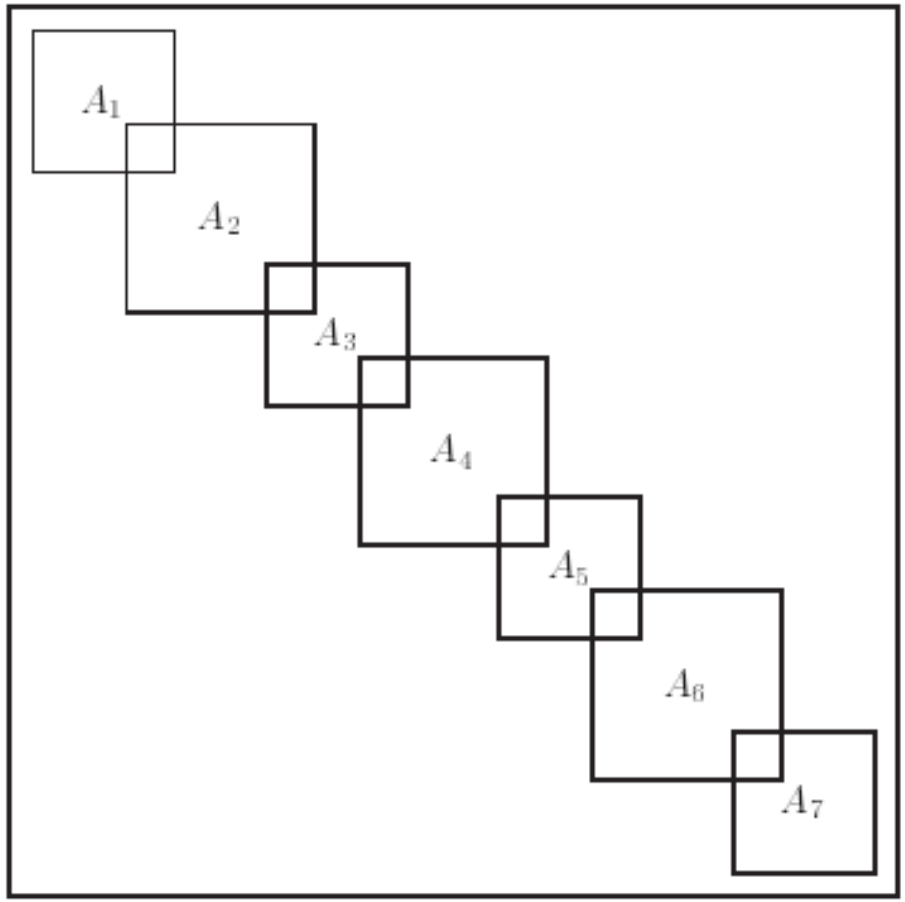
It remains to understand which preconditioners are suitable for parallel computations.

Block Jacobi preconditioners

Let $S_i = \{l_{i,1}, \ldots, l_{i,N_i}\}$ be the set containing the numbering of the internal and external interface dofs of processor i, for $i = 1, \ldots, p$. Let $\mathrm{V_i} = [\mathbf{e}_{\mathrm{l_{i,1}}}, \ldots, \mathbf{e}_{\mathrm{l_{i,N_i}}}]$, where $\mathbf{e}_i \in \mathbb{R}^N$ and

$$[\mathbf{e}_i]_j = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

The matrix $\mathrm{A_i} = \mathrm{V_i^T A V_i}$ is a $N_i \times N_i$ matrix. A block Jacobi preconditioner is of the form $\mathrm{M} = \sum_{k=1}^{p} \mathrm{V_i A_i^{-1} V_i^T}$.



A pictorial view of the block Jacobi matrix

Incomplete LU Block Jacobi preconditioners

The inversion of $\mathrm{A_i}$ may be still costly so we can replace $\mathrm{A_i}$ by its incomplete LU (or Cholesky) factorization $\hat{\mathrm{A}}_i = \hat{L}_i \hat{U}_i$, whose inversion is simple: $\mathrm{M} = \sum_{k=1}^{p} \mathrm{V_i} \hat{\mathrm{A}}_\mathrm{i}^{-1} \mathrm{V_i^T}$.

To implement the preconditioner in practice we compute $\hat{L}_i$ and $\hat{U}_i$ on each processor and then (if we use a simple Richardson iteration) at each iteration k

- Restrict the residual on each processor $\mathbf{r}_{k,i} = \mathrm{V_i^T} \mathbf{r}_\mathrm{k}$
- Solve $\hat{L}_i \hat{U}_i \mathbf{z}_{k,i} = \mathbf{r}_i$
- Sum the contribution (communication required) $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathrm{V_i} \mathbf{z}_{\mathrm{k,i}}$

Note: The matrices $\mathrm{V_i}$ are never build explicitly!

Parallel polynomial preconditioners

Polynomial preconditioners involve only multiplications of vectors with the original matrix A, so they can be effectively implemented in parallel.

Multigrid

Multigrid technique are based on a property of several iterative schemes (like Gauss-Siedel), valid for symmetric positive definite matrices arising from the discretization of elliptic problems.
If we decompose the residual $\mathbf{r}_0$ into discrete Fourier components one may note that high frequency modes are damped faster than low frequency modes.
From this observation the idea of having a series of grids of different level of coarsening and transfer the residual among them.
In this way the coarser grids will damp the lower frequency component.

Classic and algebraic multigrid

Multigrids methods subdivides into two main categories

- Grid based multigrids. A series of nested grids $\{\tau_1, \tau_2, \ldots, \tau_m\}$ (from finer to coarser) are built and the differential operator at hand discretized on each of them using the method of choice. Consequently, a series of matrices $\{\mathrm{A} = \mathrm{A}_1, \ldots, \mathrm{A}_\mathrm{m}\}$ and corresponding right hand side $\{\mathbf{b} = \mathbf{b}_1, \ldots, \mathbf{b}_m\}$ are produced.
- Algebraic multigrid (AMG). Here, the matrices $\mathrm{A}_1, \ldots, \mathrm{A}_\mathrm{m}$ are built from the original matrix A using algebraic manipulation, without the need of building the coarse grids explicitly. Since they are more practical are currently among the most used multigrid methods, even if sometimes they are less effective than the former.

The general layout

In both cases the general layout of a multigrid method is based on the following ingredients. For each level $\nu = 1, \ldots, m$ we have

- A smoother. i.e. an iterative method with the capability of eliminating rapidly high frequency components of the error. We indicate it as $S(\mathrm{A}_\nu, \mathbf{b}_\nu; \mathbf{x}_\nu)$. It returns the result of the iterative procedure starting from the initial value $\mathbf{x}_\nu \in \mathbb{R}^{N_\nu}$. A_ν is the matrix at level ν
- Restriction (coarsening) operators $I_{N_\nu}^{N_{\nu+1}} : \mathbb{R}^{N_\nu} \to \mathbb{R}^{N_{\nu+1}}$ that transfers vectors from level ν to level $\nu + i$.
- Prolongation operators $I_{N_{\nu+1}}^{N_\nu} : \mathbb{R}^{N_{\nu+1}} \to \mathbb{R}^{N_\nu}$ that transfers vectors from level $\nu + 1$ to level ν. Often $I_{N_{\nu+1}}^{N_\nu} = \alpha [I_{N_\nu}^{N_{\nu+1}}]^T$ (and in AGM usually $\alpha = 1$).
- A strategy of traversing the various grids. We can have V cycles or W cycles.

How to build the A_ν

In AMG the coarse matrix and coarse right hand side is usually build by Galerkin projection from the finer level (remind that $\mathrm{A}_1 = \mathrm{A}$)

$$\mathrm{A}_{\nu+1} = \mathrm{I}_{\mathrm{N}_\nu}^{\mathrm{N}_{\nu+1}} \mathrm{A}_\nu \mathrm{I}_{\mathrm{N}_{\nu+1}}^{\mathrm{N}_\nu}$$

$$\mathbf{b}_{\nu+1} = I_{N_\nu}^{N_{\nu+1}} \mathbf{b}_\nu$$

Note: Usually the last level matrix A_m is small enough so that the linear system can be solved by a direct method.

Two grid cycle

To explain the general idea let's suppose to have just two levels, 1 and 2 We start from a tentative solution $\mathbf{x}^{(0)} = \mathbf{x}_1^{(0)}$ at level 1 (finer level). The algorithm reads

- Pre-smooth $\mathbf{x}_1^{(1)} = S(\mathrm{A}_1, \mathbf{b}_1; \mathbf{x}_1^{(0)})$
- Get residual $\mathbf{r}_1 = \mathbf{b}_1 - \mathrm{A}_1 \mathbf{x}_1^{(1)}$
- Coarsen $\mathbf{r}_2 = I_{N_1}^{N_2} \mathbf{r}_1$
- Solve for the correction $\mathrm{A}_2 \mathbf{w}_2 = \mathbf{r}_2$
- Correct $\mathbf{x}_1^{(2)} = \mathbf{x}_1^{(1)} + I_{N_2}^{N_1} \mathbf{w}_2$
- Post-smooth $\mathbf{x}_1^{(3)} = S(\mathrm{A}_1, \mathbf{b}_1; \mathbf{x}_1^{(2)})$
- End Return $\mathbf{x}_1^{(3)}$ as approximate solution.

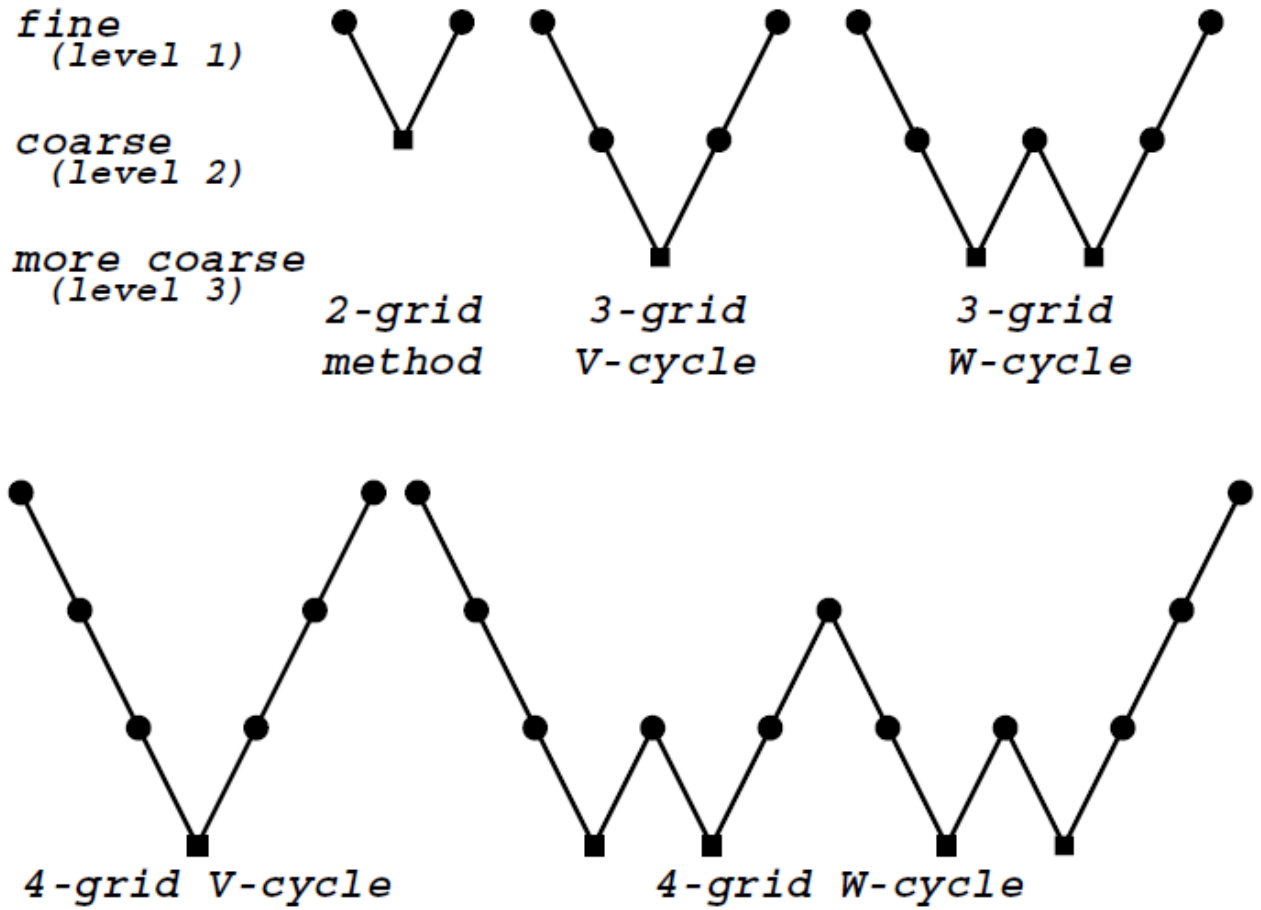
If $\mathbf{x}_1^{(3)}$ is not a satisfactory approximation the scheme is repeated setting $\mathbf{x}_1^{(0)} = \mathbf{x}_1^{(3)}$.

V and W cycles

If we have more levels we repeat the previous algorithm recursively. The step Solve for the correction is however replaced by a smoothing step, a part the coarsest level, where we solve (using a direct method).

The W cycle is a variation where we a V cycle from level 1 and 2 is followed by a V cycle from level 1 and 3 and so on, up to the coarsest level (many variations are in fact possible). Again the direct solution is performed only at the coarsest level.

V and W cycles



Restriction and prolongation operators

The construction of the restriction operator is clearly one of the key points of the method. in AMG setting one is the transpose of the other and a minimal requirement is that the prolongation operator be full rank.
Coarsening is often obtained by agglomeration. In practice the rows of $I_{N_\nu}^{N_{\nu+1}}$ contains either 0 or 1 (or another constant value) so that $I_{N_\nu}^{N_{\nu+1}}\mathbf{x}_\nu$ effectively sum-up a subset of the components of $\mathbf{x}_\nu$. It is called agglomeration because normally you sum up "nearby components", i.e. components i and j for which $[\mathrm{A_{nu}}]_{\mathrm{ij}} \neq 0$.

The way to choose the agglomeration strategy may depend on the problem at hand. For the Darcy problem we mention the works of K. Stüben.

Available software

There are plenty of good software tools for direct and iterative methods, both for scalar and parallel architectures. We give a list of the main ones.

- UMFPACK. A set of routines for solving unsymmetric sparse linear systems using the Unsymmetric MultiFrontal method. Written in ANSI/ISO C. Probably the most used package of this task. Adopted also by MATLAB. I do not know of any parallel implementation yet.
- MUMPS: a MUltifrontal Massively Parallel sparse direct solver. As the name says MUMPS can be effectively run on distributed memory parallel architectures.
- LAPACK. Linear Algebra PACKage. An old but well established Fortan library with several solvers. No parallelism and no support for sparse matrices.
- ARPACK. The ARnoldi PACKage. A set of Fortran routines for large scale eigenvalue problems. Support for sparse matrices.

Available software

- SuperLU: it is a general purpose library for the direct solution of large, sparse, nonsymmetric systems of linear equations on high performance machines. The library is written in C and is callable from either C or Fortran. Uses a modified version of LU decomposition to reduce fill-in. Scalar and parallel version (both for SIMD and MIMD architectures).
- SPARSEKIT. Sparse Matrix Utility Package. Written by Y. Saad, is a Fortran90 package that implements sparse matrix storage schemes and several iterative methods.
- Eigen. A C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms. Tailored for SSEx architectures (Intel). Supports sparse and dense matrices and it contains several solvers (wrappers for LAPACK, UMFPACK, SuperLU).

Available software

- PETSC. Portable, Extensible Toolkit for Scientific Computation. A vast collection of tools for parallel linear algebra, non-linear equations, numerical solution of PDE... Originally written in Fortran 77, has now wrappers for Fortran90, C, C++.
- TRILINOS. The Trilinos Project is an effort to develop algorithms and enabling technologies within an object-oriented software framework for the solution of large-scale, complex multi-physics engineering and scientific problems. Written in C++ has a wrapper for Fortran. It is a collection of libraries (too many to recall them) among which EPETRA/TPETRA for storing distributed matrices, BELOS for iterative solvers, ML for multilevel preconditioners, IFPACK algebraic preconditioners.... and many many others.

A simple library

If you want to have a look to the main iterative schemes with codes written "straight out of the book" you may have a look at the IML++ library. It is bases on the Sparselib++ package for matrices and vectors, but it is a template library so it can be readily adapted to other matrix libraries.
However, it does not use traits so the porting to other matrix libraries is not immediate (but not so complicated).

Domain partitioning

To partition a mesh for parallel computation there are rather well established open-source tools. For instance, the METIS-PARMETIS suite and the ZOLTAN software by Sandia National Labs.

They operate on the graph of the matrix/mesh and they allow to weight nodes in case of not equilibrated workload among nodes.

They also allow repartitioning, i.e. the dynamic transfer of nodes across processors. Very useful in the case of mesh adaption or when the workload changes in time and you need to redistribute the unknowns among processors to recalibrate it.

Bibliography

- Y. Saad. *Iterative methods for large linear systems*, 2nd Edition, SIAM, 2003.
- H. Elman, D. Silvester and A. Wathen. *Finite elements and fast iterative solvers, with applications in incompressible fluid dynamics*, Oxford Science Publications, 2005
- A Quarteroni and A Valli, *Domain decomposition methods for partial differential equations*, Oxford University Press, 1999.
- B Smith, P Bjorstad and W Gropp, *Domain decomposition*, Cambridge University Press, 1996.
- H. A. van der Vorst, *Iterative Krylov methods for large linear systems*, Cambridge University Press, 2003.
- V. Simoncini and E. Gallopoulos, *Convergence properties of block GMRES and matrix polynomials*, Linear Algebra and Applic., 1996