Numerical methods – part 1

Alain Hébert

alain.hebert@polymtl.ca

Institut de génie nucléaire

École Polytechnique de Montréal

Content (week 5)

- Solution of a linear system
  - Gauss elimination
  - Cholesky factorization
  - Crout factorization
  - The iterative approach

Solution of a linear system

A linear matrix system of order I is written

$$\mathbb{A}\,\boldsymbol{\Phi} = \boldsymbol{S} \tag{1}$$

where $\mathbb{A} = \{a_{i,j}\ ;\ i = 1, I \text{ and } j = 1, I\}$ is the coefficient matrix, $\boldsymbol{\Phi} = \text{col}\{\phi_i\ ;\ i = 1, I\}$ is the unknown vector and $\boldsymbol{S} = \text{col}\{s_i\ ;\ i = 1, I\}$ is the source vector.

- This linear system can be solved by a direct approach, such as the Gauss elimination or the Cholesky factorization or by an iterative approach.
- The discretization of the transport or diffusion equation over a 1D domain generally leads to a linear system whose coefficient matrix has a dense profile of non-zero values surrounding its diagonal. Moreover, the number of unknowns I is sufficiently small to justify the use of a direct approach.
- The linear system produced by 2D and 3D cases is generally solved using an iterative approach.

Gauss elimination

The Gauss elimination method proceeds in two steps. The first step is the forward elimination which consists in transforming the linear system (1) into an upper triangular system such as

(2)
$$\begin{pmatrix} 1 & h_{12} & h_{13} & \dots & h_{1I} \\ 0 & 1 & h_{23} & \dots & h_{2I} \\ 0 & 0 & 1 & \dots & h_{3I} \\ \vdots & & & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{pmatrix} \begin{pmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \vdots \\ \phi_I \end{pmatrix} = \begin{pmatrix} g_1 \\ g_2 \\ g_3 \\ \vdots \\ g_I \end{pmatrix} .$$

A number of I sub-steps, denoted with index (k), are required to obtain this upper triangular system. We are limiting our presentation to the case where $\mathbb{A}$ is positive-definite, in order to avoid the pivot maximization operation at each sub-step.

Gauss elimination

We set $a_{i,j}^{(0)} = a_{i,j}$ and $S_i^{(0)} = S_i$, the right terms being the components of matrix $\mathbb{A}$ and vector $\boldsymbol{S}$. We proceed line-by-line, starting with the first. The algorithm is

1. Set $k = 1$
2. Define the pivot $p^{(k)} = a_{k,k}^{(k-1)}$ and compute:

$$
\begin{aligned}
a_{k,j}^{(k)} &= \frac{a_{k,j}^{(k-1)}}{p^{(k)}} \;\; ; \;\; j = k, I \\
S_k^{(k)} &= \frac{S_k^{(k-1)}}{p^{(k)}} \\
a_{i,j}^{(k)} &= a_{i,j}^{(k-1)} - a_{i,k}^{(k-1)} a_{k,j}^{(k)} \;\; ; \;\; i = k+1, I \text{ and } j = k+1, I \\
S_i^{(k)} &= S_i^{(k-1)} - a_{i,k}^{(k-1)} S_k^{(k)} \;\; ; \;\; i = k+1, I \;\; . \qquad (3)
\end{aligned}
$$

3. If $k = I$, go to 4. Otherwise, set $k = k + 1$ and return to 2.
4. Compute:

$$
\begin{aligned}
h_{i,j} &= a_{i,j}^{(i)} \;\; ; \;\; i = 1, I \text{ and } j = 1, I \\
g_i &= S_i^{(i)} \;\; ; \;\; i = 1, I \;\; .
\end{aligned}
$$

Gauss elimination

In the particular case where $\mathbb{A}$ is tri-diagonal, this process is simplified as

$$
\begin{aligned}
h_{12} &= \frac{a_{12}}{a_{11}} \\
g_1 &= \frac{S_1}{a_{11}} \\
h_{i,i+1} &= \frac{a_{i,i+1}}{a_{i,i} - a_{i,i-1} h_{i-1,i}} \quad ; \quad i = 2, I \\
g_i &= \frac{S_i - a_{i,i-1} g_{i-1}}{a_{i,i} - a_{i,i-1} h_{i-1,i}} \quad ; \quad i = 2, I \\
h_{i,j} &= 0 \quad \text{if } j > i+1 \quad . \qquad (4)
\end{aligned}
$$

Gauss elimination

The second step of the Gauss elimination method is the backward substitution. The solution of linear system (2) is obtained by substitution, starting with the last equation. We write

$$\begin{aligned} \phi_I &= g_I \\ \phi_i &= g_i - \sum_{j=i+1}^{I} h_{i,j}\ \phi_j \quad ;\ i = I-1, I-2, \ldots, 1 \quad . \end{aligned} \tag{5}$$

A native implementation of the general Gauss elimination method is available in Matlab, using a matrix left division. If `A` is a matrix and `s` is the source column vector, then the solution of the linear system is simply obtained as

```
f = A \ s ;
```

The Matlab script `f=aslv3(a,s)` implements the recursion (4), for solving a linear system using Gauss elimination in the particular case where matrix $\mathbb{A}$ is

- symmetric,
- positive-definite and
- tri-diagonal.

Such a system is obtained from the mesh-corner or mesh-centered finite difference discretization of the 1D diffusion equation.

Cholesky factorization

- Numerical analysis algorithms may require the resolution of many linear systems, all with the same coefficient matrix $\mathbb{A}$, but with different source terms $\boldsymbol{S}$.
- We now present the Cholesky factorization method that is designed to perform the greatest part of arithmetic operations from the knowledge of $\mathbb{A}$, not of $\boldsymbol{S}$.
- We will limit our presentation to the case where $\mathbb{A}$ is symmetric and positive-definite.

The Cholesky factorization method proceeds in two steps. The first step consists in factorizing matrix $\mathbb{A}$ as a product of three matrices:

$$\mathbb{A} = \mathbb{L}\,\mathbb{D}\,\mathbb{L}^{\top} \tag{6}$$

where $\mathbb{D} = \mathrm{diag}\{d_i \ ; \ i = 1, I\}$ is a diagonal matrix and $\mathbb{L}$ is a lower triangular matrix written

$$\mathbb{L} = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ \ell_{21} & 1 & 0 & \dots & 0 \\ \ell_{31} & \ell_{32} & 1 & \dots & 0 \\ \vdots & & & \ddots & \vdots \\ \ell_{I1} & \ell_{I2} & \ell_{I3} & \dots & 1 \end{pmatrix} . \tag{7}$$

Cholesky factorization

The components d_i and $\ell_{i,j}$ are computed with the following algorithm:

$$d_1 = a_{11} \tag{8}$$

$$\ell_{i,1} = \frac{a_{i,1}}{d_1} \quad ; \; i = 2, I \tag{9}$$

$$d_i = a_{i,i} - \sum_{k=1}^{i-1} \ell_{i,k}^2 \, d_k \quad ; \; i = 2, I \tag{10}$$

$$\ell_{i,j} = \frac{1}{d_j} \left[a_{i,j} - \sum_{k=1}^{j-1} \ell_{i,k} \, d_k \, \ell_{j,k} \right] \quad ; \; i = 2, I \text{ and } j = 2, i-1 \; . \tag{11}$$

This algorithm involves no square root operations, making it a better alternative than the $\mathbb{A} = \mathbb{L}\mathbb{L}^\top$ factorization presented in some textbooks.

Cholesky factorization 2

- We note that matrix $\mathbb{L}$ is characterized by the same external profile as matrix $\mathbb{A}$, so that

$$\text{if} \quad \{a_{i,j} = 0 \ ; \ j = 1, k_i\} \quad \text{then} \quad \{\ell_{i,j} = 0 \ ; \ j = 1, k_i\} \tag{12}$$

where k_i is an integer taken in interval $1 \leq k_i < i$ and equal to the number of zero components on the left of the first non-zero component on the i–th row.

- The property of preserving the external profile is closely related to the introduction of the compressed diagonal storage mode for matrices $\mathbb{A}$ and $\mathbb{L}$.
- Instead of storing matrix $\mathbb{A}$ in a $I \times I$ 2D array, we can store the components, located inside the external profile, in a 1D row array. The components of $\mathbb{A}$, located inside the external profile, are stored in the row vector $\boldsymbol{A}' = \{a'_k \ ; \ k = 1, u_I\}$ as

$$a'_{u_i - i + j} = a_{i,j} \ ; \ i = 1, I \text{ and } j = k_i + 1, i \ . \tag{13}$$

- A second integer row array of dimension I is required to keep track of the matrix component positions. This array $\boldsymbol{u} = \{u_i \ ; \ i = 1, I\}$ is defined as

$$\begin{aligned} u_1 &= 1 \\ u_i &= u_{i-1} + i - k_i \ ; \ i = 2, I \ . \end{aligned} \tag{14}$$

Cholesky factorization

In the following example, a matrix $\mathbb{A}$ is defined as

$$\mathbb{A} = \begin{pmatrix} 3 & 2 & 0 & 0 & 0 \\ 2 & 4 & 1 & 1 & 2 \\ 0 & 1 & 5 & 0 & 0 \\ 0 & 1 & 0 & 4 & 0 \\ 0 & 2 & 0 & 0 & 6 \end{pmatrix} . \tag{15}$$

This matrix is symmetric and has three zero components, located outside its external profile, in its low-diagonal section. It can be written in compressed diagonal storage mode as

$$\boldsymbol{A}' = (\, 3 \quad 2 \quad 4 \quad 1 \quad 5 \quad 1 \quad 0 \quad 4 \quad 2 \quad 0 \quad 0 \quad 6 \,) \tag{16}$$

in association with the index vector $\boldsymbol{u}$ required to reconstruct $\mathbb{A}$. This vector is written

$$\boldsymbol{u} = (\, 1 \quad 3 \quad 5 \quad 8 \quad 12 \,) \ . \tag{17}$$

The factorization can be performed effectively for a matrix in compressed diagonal storage mode. The components $\ell_{i,j}$ can be computed in such a way as to replace components $a_{i,j}$ one–by–one. The Matlab script `ldlt=alldlf(asm,mu1)` exploits this idea.

Cholesky factorization

Next comes the resolution step, used to compute the unknown vector $\boldsymbol{\Phi}$ from the knowledge of the source vector $\boldsymbol{S}$. The computer resources required for this resolution step are much smaller than those required to factorize $\mathbb{A}$. The linear system (1) can now be written

$$\mathbb{L}\,\mathbb{D}\,\mathbb{L}^{\top}\,\boldsymbol{\Phi} = \boldsymbol{S} \quad . \tag{18}$$

We define an auxiliary vector as

$$\boldsymbol{h} = \mathbb{D}\,\mathbb{L}^{\top}\,\boldsymbol{\Phi} \tag{19}$$

and we first solve a lower-triangular matrix system,

$$\mathbb{L}\;\boldsymbol{h} = \boldsymbol{S} \quad , \tag{20}$$

using forward substitution. A simple recursion, written as

$$\begin{aligned} h_1 &= S_1 \\ h_i &= S_i - \sum_{j=1}^{i-1} \ell_{i,j}\; h_j \quad ; \; i = 2, I \end{aligned} \tag{21}$$

is required to obtain $\boldsymbol{h}$.

Cholesky factorization

The unknown vector $\mathbf{\Phi}$ is finally obtained using backward substitution in the upper-triangular linear system (19). This recursion is written

$$\begin{aligned} \phi_I &= \frac{h_I}{d_I} \\ \phi_i &= \frac{h_i}{d_i} - \sum_{j=i+1}^{I} \ell_{j,i}\, \phi_j \;\; ; \; i = I-1, I-2, \ldots, 1 \;\; . \end{aligned} \tag{22}$$

- The overall resolution step is implemented in the Matlab script `f=alldls(ldlt,mu1,s)`.
- The factorization step, as implemented in `alldlf`, is performed only once, for a given coefficient matrix $\mathbb{A}$.
  - The factorization step requires a number of arithmetic operations proportional to I^3
- The resolution step, as implemented in `alldls`, must be performed for each different value of the source vector $\boldsymbol{S}$.
  - The resolution step requires a number of arithmetic operations proportional to I^2

The Crout factorization

The Crout factorization is a generalization of the Cholesky factorization that could be used in cases where matrix $\mathbb{A}$ is non-symmetric. It is written

(23) $$\mathbb{A} = \mathbb{L}\ \mathbb{U}$$

where

(24) $$\mathbb{L} = \begin{pmatrix} \ell_{11} & 0 & 0 & \dots & 0 \\ \ell_{21} & \ell_{22} & 0 & \dots & 0 \\ \ell_{31} & \ell_{32} & \ell_{33} & \dots & 0 \\ \vdots & & & \ddots & \vdots \\ \ell_{I1} & \ell_{I2} & \ell_{I3} & \dots & \ell_{II} \end{pmatrix} \quad \text{and} \quad \mathbb{U} = \begin{pmatrix} 1 & u_{12} & u_{13} & \dots & u_{1I} \\ 0 & 1 & u_{23} & \dots & u_{2I} \\ 0 & 0 & 1 & \dots & u_{3I} \\ \vdots & & & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{pmatrix} .$$

The solution of a linear system after Crout factorization is similar to the approach already presented in this Subsection.

The iterative approach 1

- The Cholesky factorization method allows us to get rid of the zero components located outside the external profile of $\mathbb{A}$, but has no effect on those located inside this profile.
- This limited capability becomes a problem for the resolution of matrix systems originating from the discretization of 2D and 3D domains, as the number of internal zero components is high in these cases.
- The transformation of internal zero components of $\mathbb{A}$ into non-zero components in $\mathbb{L}$ is identified as factorization fill-in and is a major issue in numerical analysis.
- The iterative method is one numerical analysis technique permitting us to avoid fill-in in the resolution of a linear system.

The iterative approach 2

The iterative approach consists of introducing a preconditioning matrix $\mathbb{M}$ sufficiently close to $\mathbb{A}^{-1}$ so that the number of iterations of a fixed point method remains acceptable. The multiplication of the left and right sides of Eq. (1) by $\mathbb{M}$ leads to

(25) $$\mathbb{M}\,\mathbb{A}\boldsymbol{\Phi} = \mathbb{M}\boldsymbol{S} \ .$$

A fixed point iterative strategy is set-up, based on Eq. (25). An asymptotic series of approached solutions, denoted $\{\boldsymbol{\Phi}^{(j)} \ ; \ j \geq 0\}$, is set-up using

(26) $$\begin{aligned} &\boldsymbol{\Phi}^{(0)} \ \text{given} \\ &\boldsymbol{\Phi}^{(j+1)} = \boldsymbol{\Phi}^{(j)} + \mathbb{M}\,(\boldsymbol{S} - \mathbb{A}\boldsymbol{\Phi}^{(j)}) \ \text{ if } j \geq 0 \end{aligned}$$

where $\boldsymbol{\Phi}^{(0)}$ is an initial estimate of the solution that can be set to the zero vector if no a priori information is available.

The convergence analysis of the fixed point (26) requires the introduction of a residual matrix $\mathbb{R}$, defined as

(27) $$\mathbb{R} = \mathbb{I} - \mathbb{M}\,\mathbb{A}$$

where $\mathbb{I}$ is the identity matrix.

The iterative approach

The fixed point algorithm (26) generates the following asymptotic series:

$$
\begin{aligned}
\boldsymbol{\Phi}^{(1)} &= \mathbb{M}\boldsymbol{S} + (\mathbb{I} - \mathbb{M}\mathbb{A})\,\boldsymbol{\Phi}^{(0)} = (\mathbb{I} - \mathbb{R})\;\mathbb{A}^{-1}\,\boldsymbol{S} + \mathbb{R}\,\boldsymbol{\Phi}^{(0)} \\
\boldsymbol{\Phi}^{(2)} &= (2\,\mathbb{I} - \mathbb{M}\,\mathbb{A})\;\mathbb{M}\,\boldsymbol{S} + (\mathbb{I} - \mathbb{M}\mathbb{A})^2\,\boldsymbol{\Phi}^{(0)} = (\mathbb{I} - \mathbb{R}^2)\;\mathbb{A}^{-1}\,\boldsymbol{S} + \mathbb{R}^2\boldsymbol{\Phi}^{(0)} \\
\boldsymbol{\Phi}^{(3)} &= [3\,\mathbb{I} - (3\,\mathbb{I} - \mathbb{M}\,\mathbb{A})\;\mathbb{M}\,\mathbb{A}]\;\mathbb{M}\,\boldsymbol{S} + (\mathbb{I} - \mathbb{M}\mathbb{A})^3\,\boldsymbol{\Phi}^{(0)} = (\mathbb{I} - \mathbb{R}^3)\;\mathbb{A}^{-1}\,\boldsymbol{S} + \mathbb{R}^3\boldsymbol{\Phi}^{(0)} \\
&\;\;\vdots \\
\boldsymbol{\Phi}^{(j)} &= (\mathbb{I} - \mathbb{R}^j)\;\mathbb{A}^{-1}\,\boldsymbol{S} + \mathbb{R}^j\boldsymbol{\Phi}^{(0)} \quad .
\end{aligned}
\tag{28}
$$

We note that the asymptotic series converges to the exact solution $\mathbb{A}^{-1}\boldsymbol{S}$ of the linear system if and only if

$$
\lim_{j\to\infty} \mathbb{R}^j = \mathbb{O} \tag{29}
$$

where $\mathbb{O}$ is the zero matrix whose components are all set to zero. This condition is satisfied if a norm $\|\cdot\|$ of matrix $\mathbb{R}$ exists such that

$$
\| \mathbb{R} \| < 1 \quad . \tag{30}
$$

The iterative approach

- Equation (30) is the condition that must satisfy the preconditioning matrix $\mathbb{M}$ to guarantee the convergence of the fixed-point iterations.
- We are presenting preconditioning techniques that have proven useful for solving linear systems originating from the discretization of transport or diffusion equations.
- These three techniques all satisfy the condition (30) in the case where matrix $\mathbb{A}$ is symmetric and diagonally dominant.

The preconditioning techniques are based on a matrix splitting of $\mathbb{A}$ defined as

(31)
$$\mathbb{A} = \mathbb{U} + \mathbb{L} + \mathbb{L}^{\top}$$

where $\mathbb{U}$ is a diagonal matrix and $\mathbb{L}$ is a lower triangular matrix. They are written as

(32)
$$\mathbb{U} = \begin{pmatrix} a_{11} & 0 & 0 & \dots & 0 \\ 0 & a_{22} & 0 & \dots & 0 \\ 0 & 0 & a_{33} & \dots & 0 \\ \vdots & & & \ddots & \vdots \\ 0 & 0 & 0 & \dots & a_{II} \end{pmatrix} \qquad \mathbb{L} = \begin{pmatrix} 0 & 0 & 0 & \dots & 0 \\ a_{21} & 0 & 0 & \dots & 0 \\ a_{31} & a_{32} & 0 & \dots & 0 \\ \vdots & & & \ddots & \vdots \\ a_{I1} & a_{I2} & a_{I3} & \dots & 0 \end{pmatrix} .$$

The iterative approach

1. The Jacobi method consists in assuming that

$$\mathbb{A} \simeq \mathbb{U} \tag{33}$$

$$\text{so that} \quad \mathbb{M} = \mathbb{U}^{-1} \ . \tag{34}$$

2. The Gauss-Seidel method consists in assuming that

$$\mathbb{A} \simeq \mathbb{U} + \mathbb{L} \tag{35}$$

$$\text{so that} \quad \mathbb{M} = (\mathbb{U} + \mathbb{L})^{-1} \ . \tag{36}$$

3. The successive over-relaxation (SSOR) method consists in assuming that

$$\mathbb{A} \simeq (\mathbb{U} + \mathbb{L})\, \mathbb{U}^{-1}\, (\mathbb{U} + \mathbb{L}^{\top}) \tag{37}$$

$$\text{so that} \quad \mathbb{M} = (\mathbb{U} + \mathbb{L}^{\top})^{-1}\, \mathbb{U}\ (\mathbb{U} + \mathbb{L})^{-1} \ . \tag{38}$$

These three iterative methods are efficient as no inversion of matrices $(\mathbb{U} + \mathbb{L})$ and $(\mathbb{U} + \mathbb{L}^{\top})$ is required. Any linear system of the form $(\mathbb{U} + \mathbb{L})\, \boldsymbol{x} = \boldsymbol{S}$ ou $(\mathbb{U} + \mathbb{L}^{\top})\, \boldsymbol{x} = \boldsymbol{S}$ can be efficiently solved using forward or backward substitution.

The iterative approach

A closely related preconditioning technique is the alternating direction implicit (ADI) method. In the case of a Cartesian tri-dimensional domain, the ADI splitting is written

(39) $$\mathbb{A} = \mathbb{U} + \mathbb{P}_x \, \mathbb{X} \, \mathbb{P}_x^\top + \mathbb{P}_y \, \mathbb{Y} \, \mathbb{P}_y^\top + \mathbb{P}_z \, \mathbb{Z} \, \mathbb{P}_z^\top$$

where

$\mathbb{U} =$ matrix containing the diagonal elements of $\mathbb{A}$

$\mathbb{X}, \mathbb{Y}, \mathbb{Z} =$ symetrical matrices containing the nondiagonal elements of $\mathbb{A}$

$\mathbb{P}_x, \mathbb{P}_y, \mathbb{P}_z =$ permutation matrices that ensure a minimum bandwidth for matrices $\mathbb{X}$, $\mathbb{Y}$ and $\mathbb{Z}$.

The corresponding preconditioning matrix can be defined as

(40) $$\mathbb{M} = \left(\mathbb{P}_z \tilde{\mathbb{Z}}^{-1} \mathbb{P}_z^\top \right) \mathbb{U} \left(\mathbb{P}_y \tilde{\mathbb{Y}}^{-1} \mathbb{P}_y^\top \right) \mathbb{U} \left(\mathbb{P}_x \tilde{\mathbb{X}}^{-1} \mathbb{P}_x^\top \right)$$

where

(41) $$\tilde{\mathbb{X}} = \mathbb{X} + \mathbb{P}_x^\top \, \mathbb{U} \, \mathbb{P}_x \quad , \quad \tilde{\mathbb{Y}} = \mathbb{Y} + \mathbb{P}_y^\top \, \mathbb{U} \, \mathbb{P}_y \quad \text{and} \quad \tilde{\mathbb{Z}} = \mathbb{Z} + \mathbb{P}_z^\top \, \mathbb{U} \, \mathbb{P}_z \quad .$$

The iterative approach

- The Matlab script `free()` implements the basic iteration of Eq. (26) for solving a linear system.
- The script parameter `atv` is a user-defined function returning $\boldsymbol{\Phi}^{(j)} + \mathbb{M}\left(\boldsymbol{S} - \mathbb{A}\boldsymbol{\Phi}^{(j)}\right)$ as a function of $\boldsymbol{\Phi}^{(j)}$ and $\boldsymbol{S}$.
- The script parameter b is the source vector $\mathbb{M}\boldsymbol{S}$.

```
function [x, err, iter] = free(x, b, atv, errtol, maxit, varargin)
  errtol=errtol*norm(b);
  rho=Inf; err=[ ];
  iter=0;
  while((rho > errtol) && (iter < maxit))
    iter=iter+1;
    r=feval(atv,x,b,varargin{:})-x;
    rho=norm(r); err=[err;rho];
    x=x+r;
  end
```

The iterative approach

- The following Matlab script is an example of iterative solution of a linear system `A*x=b` with a coefficient matrix `A` selected among the `'wilk'` collection in Matlab.
- We also selected `inv(M1)` as preconditioning matrix, using an arbitrary choice for `M1`.

```
A=gallery('wilk',21);
M1=diag([10:-1:1 0 1:10]); M1(10:12,10:12)=A(10:12,10:12);
b=sum(A,2);
x0=zeros(21,1);
[out,err,total_iters]=free(x0, inv(M1)*b, 'atv', 1e-10, 100, ...
            inv(M1)*A);
```

where

```
function y=atv(x,b,A)
% compute x+(b-Ax) for an iterative algorithm.
  y=x+(b-A*x);
```

The iterative approach

A first convergence acceleration technique, known as the Livolant acceleration method, consists in introducing a dynamic acceleration factor $\mu^{(j)}$ in Eq. (26),

$$\boldsymbol{\Phi}^{(j+1)} = \boldsymbol{\Phi}^{(j)} + \mu^{(j)}\, \mathbb{M}\, (\boldsymbol{S} - \mathbb{A}\boldsymbol{\Phi}^{(j)}) \ , \tag{42}$$

and in adjusting it in such a way as to minimize the L^2 norm of residue $\boldsymbol{R}^{(j+1)}$ at the next iteration. A consistent definition of the residue is written as

$$\boldsymbol{R}^{(j)} = \mathbb{M}\,(\boldsymbol{S} - \mathbb{A}\boldsymbol{\Phi}^{(j)}) \ . \tag{43}$$

The value of the residue at the next iteration will be given as

$$\begin{aligned} \boldsymbol{R}^{(j+1)} &= \mathbb{M}\left(\boldsymbol{S} - \mathbb{A}\boldsymbol{\Phi}^{(j+1)}\right) \\ &= \mathbb{M}\left[\boldsymbol{S} - \mathbb{A}\left(\boldsymbol{\Phi}^{(j)} + \mu^{(j)}\, \mathbb{M}\, (\boldsymbol{S} - \mathbb{A}\boldsymbol{\Phi}^{(j)})\right)\right] \\ &= \boldsymbol{R}^{(j)} - \mu^{(j)}\, \mathbb{M}\, \mathbb{A}\boldsymbol{R}^{(j)} \ . \end{aligned} \tag{44}$$

The iterative approach

The L^2 norm of this residue will be minimum if

(45)
$$\frac{d}{d\mu^{(j)}} \left\langle \boldsymbol{R}^{(j+1)},\, \boldsymbol{R}^{(j+1)} \right\rangle = 0$$

so that

(46)
$$\mu^{(j)} = \frac{\left\langle \boldsymbol{R}^{(j)},\, \mathbb{M}\,\mathbb{A}\,\boldsymbol{R}^{(j)} \right\rangle}{\left\langle \mathbb{M}\,\mathbb{A}\,\boldsymbol{R}^{(j)},\, \mathbb{M}\,\mathbb{A}\,\boldsymbol{R}^{(j)} \right\rangle} \quad .$$

- The acceleration parameter can be recomputed dynamically as a function of the actual residue.
- In order to stabilize the iterative process, we suggest alternating cycles of three free and three accelerated iterations.
- Moreover, we must carefully detect any convergence instability where $\mu^{(j)} \leq 0$, especially near convergence where the denominator of Eq. (46) approaches zero. In case of convergence instability, we recommend setting $\mu^{(j)} = 1$.

The Matlab script implementing the Livolant acceleration method is implemented as

```
[x, err, iter] = livolant(x, b, atv, errtol, maxit, varargin)
```

The iterative approach

- The GMRES (Generalized Minimum RESidual) algorithm was proposed as a Krylov subspace method for nonsymmetric systems.
- The GMRES algorithm is a very powerful and stable technique to speed up iterations in solving large linear systems.
- The vector basis for the Krylov space is constructed and must be stored as the iteration progresses.
- This means that in order to perform k GMRES iterations, one must store k vectors of length N, leading to prohibitive computer resource requirements.
- One way to avoid this problem is to restart the GMRES algorithm every m (Matlab variable `nstart`) iterations, leading to the GMRES(m) algorithm.

The GMRES(m) algorithm is implemented in the Matlab script `gmres_m()` as

```
[x, err, iter] = gmres_m(x, b, atv, errtol, nstart, maxit, varargin)
```

The iterative approach