0.1 Iterative methods

As seen in the previous lecture, direct methods can theoretically compute exact solutions $\boldsymbol{x}\mathbb{R}^N$ to linear systems in the form of:

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

with matrix with real coefficients $\mathrm{A} \in M_N(\mathbb{R})$ and given data $\boldsymbol{b} \in \mathbb{R}^N$, in a determined finite number of steps.

As computing the inverse of the matrix is unrealistic, several methods were introduced based on factorizations of the type $\mathrm{A} = \mathrm{P}\ \mathrm{Q}$ where P and Q have a structure simplifying the resolution of the system: diagonal, banded, triangular.

Methods like LU, Cholevski take advantage of the existence of a decomposition involving triangular matrices while QR for example, involves the construcion of an orthogonal basis. All methods prove to be quite expensive, hard to parallelize due to the sequential nature of the algorithm and prone to error propagation.

Iterative methods have been developed for:

- solving very large linear systems with direct methods is in practice not possible due to the complexity in term of computational operations and data,
- taking advantage of sparse system for which the structure of the matrix can result in dramatic speed-up (this is the case for numerical schemes for PDEs),
- using the fact that some systems like PDEs discretizations are already formulated in an iterative fashion.

In this section, we discuss briefly the computational properties of iterative methods for solving linear systems. Computing the exact solution is not a requirement anymore but instead the algorithm is supposed to converge asymptotically to the exact solution: the algorithm is stopped when the approximate solution is deemed *close enough* to the exact solution in a sense to be defined. A parameter used as stopping criterion triggers the completion of the algorithm.

The general idea of these methods is to introduce a splitting of the form:

$$\mathrm{A} = \mathrm{G} - \mathrm{H}$$

such the solution $\boldsymbol{x}$ satisfies:

$$\mathrm{G}\boldsymbol{x} = \boldsymbol{b} + \mathrm{H}\boldsymbol{x}$$

Similarly to fixed-point methods we can define a sequence of approximate solutions $\left(\boldsymbol{x}^k\right)$ satisfying relations of the form:

$$\mathrm{G}\hat{\boldsymbol{x}}^{k+1} = \boldsymbol{b} + \mathrm{H}\hat{\boldsymbol{x}}^k$$

with G invertible.

The matrix viewed as a linear mapping in $\mathbb{R}^N$, the counterpart of such approaches is given by the Brouwer Theorem in finite dimension, where a continuous mapping $f : \Omega \to \Omega$ with Ω compact of $\mathbb{R}^N$ admits a fixed-point $\boldsymbol{x}^\star$ satisfying $f(\boldsymbol{x}^\star) = \boldsymbol{x}^\star$ and is contracting.

Methods introduced depend on the iteration defined by the splitting and call for several questions regarding the computational aspects:

1. How can the convergence be ensure?
2. How fast is the convergence?
3. How expensive is each iteration?
4. How does the algorithm behave with respect to numerical error?

The question of the convergence is addressed by proving an estimate on error vectors in terms of iteration error $\hat{\boldsymbol{\epsilon}}^k = \hat{\boldsymbol{x}}^{k+1} - \hat{\boldsymbol{x}}^k$ or global error: $\boldsymbol{\epsilon}^k = \hat{\boldsymbol{x}}^k - \boldsymbol{x}$. The convergence rate α means that $C > 0$, $|\boldsymbol{\epsilon}^{k+1}| \leq C|\boldsymbol{\epsilon}^k|^\alpha$.

For example, substituting $\hat{\boldsymbol{x}}^{k+1} = \mathrm{G}^{-1}\boldsymbol{b} + \mathrm{G}^{-1}\mathrm{H}\hat{\boldsymbol{x}}^k$ in $\hat{\boldsymbol{\epsilon}}^k = \hat{\boldsymbol{x}}^{k+1} - \hat{\boldsymbol{x}}^k$ gives a relation between successive iteration errors:

$$\hat{\boldsymbol{\epsilon}}^k = \mathrm{G}^{-1}\mathrm{H}\ \hat{\boldsymbol{\epsilon}}^{k-1}$$

with $\mathrm{M} = \mathrm{G}^{-1}\mathrm{H}$ the iteration matrix, and recursively $\hat{\boldsymbol{\epsilon}}^k = (\mathrm{G}^{-1}\mathrm{H})^{k+1}\hat{\epsilon}^0$. Convergence is then conditioned to the existence of a contraction factor $K < 1$ such that $\|\hat{\boldsymbol{\epsilon}}^k\|_\infty \leq K\ \|\hat{\boldsymbol{\epsilon}}^{k-1}\|_\infty$ ensuring decrease of the error.

In terms of the matrix M, this translate for the spectral radius $\rho(\mathrm{M})$ as $\rho(\mathrm{M}) < 1$ since in that case $\lim_{k\to\infty} M^k\hat{\boldsymbol{\epsilon}}^0 = 0_{\mathbb{R}^N}$. The smaller the spectral radius, the faster the convergence.

Each method is described briefly and qualitatively with just the necessary ingredients to discuss practical implementations.

0.2 Relaxation methods

Consider the relations for each row $i = 1, \ldots, N$:

$$\boldsymbol{x}_i = \frac{1}{a_{ii}}\Big(b_i - \sum_{i\neq j} a_{ij}\boldsymbol{x}_j\Big) \tag{1}$$

Let us introduce two methods based on constructing sequences of approximate solutions $\left(\hat{\boldsymbol{x}}^k\right)$, $k \geq 1$ given an initial guess $\hat{\boldsymbol{x}}^0 \in \mathbb{R}^N$ and then associated relaxation methods.

0.2.1 Jacobi, methods of simultaneous displacements

$$\hat{\boldsymbol{x}}_i^{k+1} = \frac{1}{a_{ii}}\Big(b_i - \sum_{i\neq j} a_{ij}\hat{\boldsymbol{x}}_j^k\Big) \tag{2}$$

Convergence: the global error $\boldsymbol{\epsilon}^k$ is controlled by

$$\|\boldsymbol{\epsilon}^{k+1}\| \leq \sum_{i\neq j}\left|\frac{a_{ij}}{a_{ii}}\right| \|\boldsymbol{\epsilon}^k\| \leq K^k\ \|\boldsymbol{\epsilon}^1\|$$

It is then enough if the matrix is strictly diagonally dominant. Expressing the iteration error gives directly that $\mathrm{M} = \mathrm{G}^{-1}\mathrm{H}$ such that $\rho(\mathrm{M}) < 1$.

Algorithm: the splitting is

$$\mathrm{A} = \mathrm{D} - \mathrm{H}$$

with $\mathrm{D} = \mathrm{diag}(A)$, thus

$$\hat{\boldsymbol{x}}^{k+1} = \mathrm{D}^{-1}(\boldsymbol{b} + \mathrm{H}\hat{\boldsymbol{x}}^k)$$

Implementation:

1. Parallelization component by component is possible since there is only dependency on $\hat{\boldsymbol{x}}^k$.
2. Memory requirement for storing both $\hat{\boldsymbol{x}}^{k+1}$ and $\hat{\boldsymbol{x}}^k$ at each iteration.

0.2.2 Gauss–Seidel, methods of sucessive displacements

In Jacobi iterations, notice that sequential ordered computation of terms

$$\hat{\boldsymbol{x}}_i^{k+1} = \frac{1}{a_{ii}}\Big(b_i - \sum_{i\neq j} a_{ij}\hat{\boldsymbol{x}}_j^k\Big) \tag{3}$$

involves components $\hat{\boldsymbol{x}}_j^k$ which are also computed for $\hat{\boldsymbol{x}}^{k+1}$ if $j < i$.

$$\hat{\boldsymbol{x}}_i^{k+1} = \frac{1}{a_{ii}}\Big(b_i - \sum_{i<j} a_{ij}\hat{\boldsymbol{x}}_j^{k+1} - \sum_{i>j} a_{ij}\hat{\boldsymbol{x}}_j^k\Big) \tag{4}$$

Algorithm: the splitting is

$$\mathrm{A} = \mathrm{L} - \mathrm{R}_0$$

with $\mathrm{L} = \mathrm{D} + \mathrm{L}_0$ lower-triangular matrix and R_0 strict upper-triangular matrix, thus

$$\hat{\boldsymbol{x}}^{k+1} = \mathrm{D}^{-1}(\boldsymbol{b} - \mathrm{L}_0\hat{\boldsymbol{x}}^{k+1} + \mathrm{R}_0\hat{\boldsymbol{x}}^k)$$

or

$$\mathrm{L}\ \hat{\boldsymbol{x}}^{k+1} = \mathrm{D}^{-1}(\boldsymbol{b} + \mathrm{R}_0\hat{\boldsymbol{x}}^k)$$

Recast under the usual form:

$$\hat{\boldsymbol{x}}^{k+1} = \mathrm{L}^{-1}(\boldsymbol{b} + \mathrm{R}_0\hat{\boldsymbol{x}}^k)$$

and the iteration matrix is $\bar{\mathrm{M}} = \mathrm{L}^{-1}\mathrm{R}_0$.

Convergence: the global error $\boldsymbol{\epsilon}^k$ is controlled by

$$\|\boldsymbol{\epsilon}^{k+1}\| \leq \frac{\displaystyle\sum_{i>j}\left|\frac{a_{ij}}{a_{ii}}\right|}{1 - \displaystyle\sum_{i<j}\left|\frac{a_{ij}}{a_{ii}}\right|}\ \|\boldsymbol{\epsilon}^k\| \leq \bar{K}^k\ \|\boldsymbol{\epsilon}^1\|$$

If the Jacobi contraction factor $K < 1$ then $\bar{K} < 1$. Expressing the iteration error gives directly that $\bar{\mathrm{M}} = \mathrm{L}^{-1}\mathrm{R}_0$ such that $\rho(\bar{\mathrm{M}}) < 1$.

Implementation:

1. Parallelization component by component is not possible easily since there is serialization for each row i due to the dependency on $\hat{\boldsymbol{x}}_j^{k+1}$, $j < i$.
2. Memory requirement is only for storing one vector of $\mathbb{R}^N$ at each iteration.

0.2.3 Relaxation of Jacobi and Gauss-Seidel

Relaxation methods consists of adding a linear combination of the approximate solution at the previous iteration to minimize the spectral radius for convergence, using the relaxation parameter $\gamma \in (0, 1)$.

1. Jacobi Over-Relaxation (JOR):

$$\hat{\boldsymbol{x}}_i^{k+1} = (1-\gamma)\ \hat{\boldsymbol{x}}_i^k + \gamma\frac{1}{a_{ii}}\Big(b_i - \sum_{i\neq j} a_{ij}\hat{\boldsymbol{x}}_j^k\Big) \tag{5}$$

   which reads in matricial form

$$\hat{\boldsymbol{x}}^{k+1} = \mathrm{M}_\gamma\hat{\boldsymbol{x}}^k + \gamma\mathrm{D}^{-1}\ \boldsymbol{b}$$

   with $\mathrm{M}_\gamma = (1-\gamma)\mathbb{I} + \gamma\mathrm{D}^{-1}\mathrm{H}$

2. Successive Over-Relaxation (SOR):

$$\hat{\boldsymbol{x}}_i^{k+1} = (1-\gamma)\ \hat{\boldsymbol{x}}_i^k + \gamma\frac{1}{a_{ii}}\Big(b_i - \sum_{i<j} a_{ij}\hat{\boldsymbol{x}}_j^{k+1} - \sum_{i>j} a_{ij}\hat{\boldsymbol{x}}_j^k\Big) \tag{6}$$

   which reads in matricial form

$$\hat{\boldsymbol{x}}^{k+1} = \mathrm{M}_\gamma\hat{\boldsymbol{x}}^k + \gamma\mathrm{C}\ \boldsymbol{b}$$

   with $\mathrm{M}_\gamma = (1+\gamma\mathrm{D}^{-1}{\mathrm{L}_0}^{-1})^{-1}\big[(1-\gamma)\mathbb{I}+\gamma\mathrm{D}^{-1}\mathrm{R}_0\big]$ and $\mathrm{C} = (1+\gamma\mathrm{D}^{-1}{\mathrm{L}_0}^{-1})^{-1}\mathrm{D}^{-1}$

The relaxation parameter γ cannot be known *a priori* and is usually determined by heuristics.

0.2.4 Parallelization of Gauss–Seidel

Overcoming the serialization in Gauss–Seidel is possible if the matrix is sparse. Taking advantage of the fact that components does not all possess connectivities with each other: such dependencies can be built from the sparsity pattern then decoupled graphs identified:

1. Component Dependency-Graph: generate a graph to reorder entries such that dependencies are avoided.
2. Red–Black coloring: special case for two-dimensional problems.

0.3 Krylov-subspace methods

The idea of these methods is that the solution is decomposed on a sequence of orthogonal subspaces.

If A is symmetric definite positive it induces the corresponding scalar product:

$$\langle\, \boldsymbol{x}\,,\, \boldsymbol{y}\,\rangle = (\,\mathrm{A}\boldsymbol{x}\,,\, \boldsymbol{y}\,) = \boldsymbol{y}^T \mathrm{A}\boldsymbol{x}$$

with $(\,\mathrm{A}\cdot\,,\,\cdot\,)$ canonical scalar product in $\mathbb{R}^N$. The vectors $(\boldsymbol{e}_1, \ldots, \boldsymbol{e}_N)$ are said A-conjugate if $\boldsymbol{e}_j^T \mathrm{A}\boldsymbol{e}_i = 0$ for $i \neq j$: they are orthogonal for the scalar-product induced by A.

0.3.1 Principle of descent methods: Steepest Gradient

Minimisation of the residual:

$$\boldsymbol{x}^\star = \mathrm{argmin}_{\boldsymbol{x}}\ \mathrm{J}(\boldsymbol{x}) = \frac{1}{2}\langle\, \boldsymbol{x}\,,\, \boldsymbol{x}\,\rangle - \langle\, \boldsymbol{b}\,,\, \boldsymbol{x}\,\rangle$$

Construct a sequence of solutions to approximate minimization problems, given $\hat{\boldsymbol{x}}^k$:

$$\mathrm{J}(\hat{\boldsymbol{x}}^{\mathrm{k}+1}) \leq \mathrm{J}(\hat{\boldsymbol{x}}^{\mathrm{k}})$$

where $\hat{\boldsymbol{x}}^{k+1} = \hat{\boldsymbol{x}}^k + \alpha_{k+1}\boldsymbol{e}^{k+1}$, with α_{k+1} a descent factor and $\boldsymbol{e}_{k+1}$ a direction.

For the Steepest Gradient:

1. take the direction given by $-\nabla\mathrm{J}(\hat{\boldsymbol{x}}^{\mathrm{k}}) = \boldsymbol{b} - \mathrm{A}\hat{\boldsymbol{x}}^{\mathrm{k}}$ which is the residual $\boldsymbol{r}_k = \boldsymbol{b} - \mathrm{A}\hat{\boldsymbol{x}}^k$, thus $\hat{\boldsymbol{x}}^{k+1} = \hat{\boldsymbol{x}}^k + \alpha_{k+1}\boldsymbol{r}_k$.
2. choose the descent factor α^{k+1} minimizing the functional $\mathrm{J}(\hat{\boldsymbol{x}}^{\mathrm{k}} + \alpha_{\mathrm{k}+1}\boldsymbol{r}_{\mathrm{k}})$:

$$\alpha_{k+1} = \frac{\boldsymbol{r}_k^T \boldsymbol{b}}{\boldsymbol{r}_k^T \mathrm{A}\boldsymbol{r}_k}$$

The speed of convergence is bounded by $\mathcal{O}\left(1 - \mathcal{C}(\mathrm{A})^{-1}\right)$ with $\mathcal{C}(\mathrm{A})$ the conditioning of A. The gradient direction may not be optimal, Conjugate Gradient methods improve the choice of $\left(\boldsymbol{e}_k\right)$.

0.3.2 Conjugate Gradient

The Conjugate Gradient (CG) is a Krylov-subspace algorithm for symmetric positive definite matrices.

Given $\hat{\boldsymbol{x}}^0$, $\left(\hat{\boldsymbol{x}}^k\right)$ is q sequence of solutions to approximate k-dimensional minimisation problems.

For the Conjugate Gradient:

1. take the direction $\boldsymbol{e}_{k+1}$ such that $\left(\boldsymbol{e}_1, \dots, \boldsymbol{e}_k, \boldsymbol{e}_{k+1}\right)$ is A-conjugate, thus $\hat{\boldsymbol{x}}^{k+1} = \hat{\boldsymbol{x}}^k + \alpha_{k+1}\boldsymbol{e}_{k+1}$.
2. choose the descent factor α^{k+1} minimizing the functional $\mathrm{J}(\hat{\boldsymbol{x}}^{\mathrm{k}} + \alpha_{\mathrm{k+1}}\boldsymbol{r}_{\mathrm{k}})$, which is defined by
$$\alpha_j = \frac{\boldsymbol{e}_j^T \boldsymbol{b}}{\boldsymbol{e}_j^T \mathrm{A} \boldsymbol{e}_j}$$
and with $\boldsymbol{e}_j^T \boldsymbol{b} \neq 0$ (unless the exact solution is reached).

The construction of $\left(\boldsymbol{e}_1, \dots, \boldsymbol{e}_{k+1}\right)$ is done by orthogonalization of residuals by Gramm–Schmidt:
$$\boldsymbol{e}_{k+1} = \boldsymbol{r}_k - \frac{\boldsymbol{e}_k^T \mathrm{A} \boldsymbol{r}_{k-1}}{\boldsymbol{e}_k^T \mathrm{A} \boldsymbol{e}_k}$$
so that $\boldsymbol{r}_{k+1} = \boldsymbol{b} - \mathrm{A}\hat{\boldsymbol{x}}^{k+1} = \boldsymbol{r}^k - \alpha_{k+1}\mathrm{A}\boldsymbol{e}_{k+1}$

After N steps, the A-conjugate basis of $\mathbb{R}^N$ is done and the exact solution is reached:
$$\boldsymbol{x} = \sum_{j=1}^{N} \alpha_j \hat{\boldsymbol{x}}^j$$

For any k, the speed of convergence is bounded by
$$\mathcal{O}\left(\frac{1 - \sqrt{\mathcal{C}(\mathrm{A})}}{1 + \sqrt{\mathcal{C}(\mathrm{A})}}\right)^{2k}$$
in the norm induced by A, with $\mathcal{C}(\mathrm{A})$ the conditioning of A.

The Conjugate Gradient can therefore be seen as a direct methods but in practice:

- the iterative computation of the A-conjugate basis suffers from the same issue of numerical error propagation as the QR factorization leading to a loss of orthogonality,
- the convergence is slow, which makes it unrealistic to compute the exact solution for large systems,

so it is used as an iterative method.

Example algorithm on first steps:

1. Given $\hat{\boldsymbol{x}}^0 = 0$, set $\boldsymbol{r}_0 = \boldsymbol{b} - \mathrm{A}\hat{\boldsymbol{x}}^0$ and $\boldsymbol{e}_1 = \boldsymbol{r}_0$,
2. Take $\hat{\boldsymbol{x}}_1 = \alpha_1 \boldsymbol{e}_1$, then $\alpha_1 \boldsymbol{e}_1^T \mathrm{A} \boldsymbol{e}_1 = \boldsymbol{e}_1^T \boldsymbol{b}$, thus
$$\alpha_1 = \frac{\boldsymbol{r}_0^T \boldsymbol{b}}{\boldsymbol{r}_0^T \mathrm{A} \boldsymbol{r}_0}$$
3. Compute the residual:
$$\boldsymbol{r}_1 = \boldsymbol{b} - \mathrm{A}\hat{\boldsymbol{x}}^1$$
4. Compute the direction:
$$\boldsymbol{e}_2 = \boldsymbol{r}_1 - \frac{\boldsymbol{e}_1^T \mathrm{A} \boldsymbol{r}_0}{\boldsymbol{e}_1^T \mathrm{A} \boldsymbol{e}_1}$$
5. Compute the factor:
$$\alpha_2 = \frac{\boldsymbol{e}_2^T \boldsymbol{b}}{\boldsymbol{e}_2^T \mathrm{A} \boldsymbol{e}_2}$$
6. Update the solution:
$$\hat{\boldsymbol{x}}^2 = \hat{\boldsymbol{x}}^1 + \alpha_2 \boldsymbol{e}_2$$
7. ...

The algorithm iteration reads:

1. Compute the residual:
$$\boldsymbol{r}_k = \boldsymbol{b} - \mathrm{A}\hat{\boldsymbol{x}}^k$$
2. Compute the direction:
$$\boldsymbol{e}_{k+1} = \boldsymbol{r}_k - \frac{\boldsymbol{e}_k^T \mathrm{A} \boldsymbol{r}_{k-1}}{\boldsymbol{e}_k^T \mathrm{A} \boldsymbol{e}_k}$$
3. Compute the factor:
$$\alpha_{k+1} = \frac{\boldsymbol{e}_{k+1}^T \boldsymbol{b}}{\boldsymbol{e}_{k+1}^T \mathrm{A} \boldsymbol{e}_{k+1}}$$
4. Update the solution:
$$\hat{\boldsymbol{x}}^{k+1} = \hat{\boldsymbol{x}}^k + \alpha_{k+1} \boldsymbol{e}_{k+1}$$

which requires two matrix-vector multiplications per loop, $\mathrm{A}\hat{\boldsymbol{x}}^k$ then $\mathrm{A}\boldsymbol{e}_{k+1}$ Using $\boldsymbol{r}_{k+1} = \boldsymbol{r}^k - \alpha_{k+1} \mathrm{A} \boldsymbol{e}_{k+1}$ saves one matrix-vector multiplication.

While the residual norm $\varrho_k = \|\boldsymbol{r}_k\|_2^2$ is big:

1. Compute the projection:
$$\beta_k = \frac{\varrho_k}{\varrho_{k-1}}$$
2. Compute the direction:
$$\boldsymbol{e}_{k+1} = \boldsymbol{r}_k + \beta_k \boldsymbol{e}_k$$
3. Compute the factor:
$$\boldsymbol{w} = \mathrm{A}\boldsymbol{e}_{k+1}; \quad \alpha_{k+1} = \frac{\varrho_k}{\boldsymbol{e}_{k+1}^T \boldsymbol{w}}$$
4. Update the solution:
$$\hat{\boldsymbol{x}}^{k+1} = \hat{\boldsymbol{x}}^k + \alpha_{k+1}\boldsymbol{e}_{k+1}$$
5. Update the residual:
$$\boldsymbol{r}^{k+1} = \boldsymbol{r}^k - \alpha_{k+1}\boldsymbol{w}$$

0.3.3 Preconditioners

While seeing the Conjugate Gradient as a pure iterative method relieves from concerns regarding orthogonality loss, the convergence is still slow as soon as the condition number of the matrix is bad.

Preconditioning the system consists in finding a non-singular symmetrix matrix C such that $\tilde{\mathrm{A}} = \mathrm{C}^{-1}\mathrm{A}\mathrm{C}^{-1}$ and the conjugate gradient is applied to
$$\tilde{\mathrm{A}}\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{b}}$$
with $\tilde{\boldsymbol{x}} = \mathrm{C}^{-1}\boldsymbol{x}$ and $\tilde{\boldsymbol{b}} = \mathrm{C}^{-1}\boldsymbol{b}$.

With:

- $\mathrm{M} = \mathrm{C}^2$
- $\boldsymbol{e}_k = \mathrm{C}^{-1}\tilde{\boldsymbol{e}}_k$
- $\hat{\boldsymbol{x}}_k = \mathrm{C}^{-1}\tilde{\hat{\boldsymbol{x}}}_k$
- $\boldsymbol{z}_k = \mathrm{C}^{-1}\tilde{\boldsymbol{r}}_k$
- $\boldsymbol{r}_k = \mathrm{C}\tilde{\boldsymbol{r}}_k = \boldsymbol{b} - \mathrm{A}\hat{\boldsymbol{x}}_k$

and M is a symmetric positive definite matrix called the preconditioner.

While the residual norm $\varrho_k = \|\boldsymbol{r}_k\|_2^2$ is big:

1. Solve:
$$\mathrm{M}\boldsymbol{z}_k = \boldsymbol{r}_k$$

2. Compute the projection:

$$\beta_k = \frac{\boldsymbol{z}_k^T \boldsymbol{r}_k}{\boldsymbol{z}_{k-1}^T \boldsymbol{r}_{k-1}}$$

3. Compute the direction:

$$\boldsymbol{e}_{k+1} = \boldsymbol{z}_k + \beta_k \boldsymbol{e}_k$$

4. Compute the factor:

$$\alpha_{k+1} = \frac{\boldsymbol{z}_k^T \boldsymbol{r}_k}{\boldsymbol{e}_{k+1}^T \mathrm{A} \boldsymbol{e}_{k+1}}$$

5. Update the solution:

$$\hat{\boldsymbol{x}}^{k+1} = \hat{\boldsymbol{x}}^k + \alpha_{k+1} \boldsymbol{e}_{k+1}$$

6. Update the residual:

$$\boldsymbol{r}^{k+1} = \boldsymbol{r}^k - \alpha_{k+1} \boldsymbol{w}$$

The linear system $\mathrm{M}\boldsymbol{z}_k = \boldsymbol{r}_k$ should be easy to solve and can lead to fast convergence, typically $\mathcal{O}\left(()\sqrt{N}\right)$. Since

$$\mathrm{M}\boldsymbol{z}_k = \boldsymbol{b} - \mathrm{A}\hat{\boldsymbol{x}}_k$$

Then an iterative relation appears:

$$\hat{\boldsymbol{x}}_{k+1} = \mathrm{M}^{-1}\big(\boldsymbol{b} - \mathrm{A}\hat{\boldsymbol{x}}_k\big)$$

therefore iterative methods like Jacobi, Gauss-Seidel and relaxation methods can be used.

0.4 Power method

This method is used for finding the dominant eigenvalues of a matrix $\mathrm{A} \in M_N(\mathbb{R})$ of N eigenvectors $\big(\boldsymbol{v}_i\big)$ with associated eigenvalues $\big(\lambda_i\big)$ ordered in decreasing module. The eigenvalues are either real or conjugate complex pairs.

Given a random vector $\boldsymbol{x}^0$, construct a sequence of vectors $\big(\hat{\boldsymbol{x}}^k\big)$ such that

$$\hat{\boldsymbol{x}}^{k+1} = \mathrm{A}\hat{\boldsymbol{x}}^k$$

then $\forall k \geq 0$

$$\hat{\boldsymbol{x}}^k = \sum_{i=0}^{N-1} \lambda_i^k \xi_i \boldsymbol{v}_i$$

for some coefficients $\big(\boldsymbol{v}_i\big)$.

Assume that λ_0 is a dominant real eigenvalue and $\xi_0 \neq 0$, then

$$\hat{\boldsymbol{x}}^k = \lambda_0^k \big(\xi_0 \boldsymbol{v}_0 + \boldsymbol{r}_k\big)$$

with the residual $\boldsymbol{r}_k$ defined as

$$\boldsymbol{r}_k = \lambda_0^{-k} \sum_{i=1}^{N-1} \lambda_i^k \xi_i \boldsymbol{v}_i$$

and $\lim_{k\to\infty} \boldsymbol{r}_k = O_{\mathbb{R}^N}$. To the limit $\hat{\boldsymbol{x}}_{k+1} \approx \lambda_0 \hat{\boldsymbol{x}}^k \approx \lambda_0 \xi_0 \boldsymbol{v}0$ almost parallel to the first eigenvector.

- This method is fast to compute the spectral radius for the Jacobi method and relaxation parameters.
- The convergence is geometric and the speed depends on the ratio $|\lambda_1/\lambda_0|$.
- If the matrix is symmetric, the convergence speed can be doubled.
- If λ_0 is very large or very small then taking high powers lead to numerical issues, the algorithm requires a normalization.