Discussion on Iterative Solvers

TMA4280—Introduction to Supercomputing

NTNU, IMF

April 6. 2018

Problem

— Solve

$$\mathrm{A}\boldsymbol{x} = \boldsymbol{b}, \qquad \boldsymbol{b}, \boldsymbol{x} \in \mathbb{R}^N, \qquad \mathrm{A} \in M_N(\mathbb{R})$$

where A can be the system resulting from discretizing a Poisson problem using finite differences.

— We use standard notation for matrices and vectors, i.e.

$$\begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,N} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N,1} & a_{N,2} & \cdots & a_{N,N} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{pmatrix}$$

Direct methods

— Computing the inverse of the matrix is unrealistic,

— Matrix inversion has the same time complexity as matrix multiplication (typically $\mathcal{O}(n^3)$).

— Direct methods can theoretically compute exact solutions $\boldsymbol{x}\mathbb{R}^N$ to linear systems in the form of:

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

— Several methods were introduced based on factorizations of the type $\mathrm{A} = \mathrm{P}\ \mathrm{Q}$

— P and Q have a structure simplifying the resolution of the system: diagonal, banded, triangular.

— The structure and properties of the matrix $\boldsymbol{A}$ determine which algorithms we can use.

Ex: LU, Cholevski involve triangular matrices, QR constructs an orthogonal basis.

Iterative methods

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.

General idea

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.

Link to linear mappings

The matrix viewed as a linear mapping in $\mathbb{R}^N$:

- the counterpart embodied by the Brouwer Theorem in finite dimension,
- Continuous mapping $f : \Omega \to \Omega$ with Ω compact of $\mathbb{R}^N$
  1. Admits a fixed-point $\boldsymbol{x}^\star$ satisfying $f(\boldsymbol{x}^\star) = \boldsymbol{x}^\star$,
  2. and is contracting.

Following a sequence of approximate solutions $\left(\boldsymbol{x}^k\right)$ in $\mathbb{R}^N$

Computational aspects

Methods introduced depend on the iteration defined by the splitting:

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

Estimate on error vectors in terms of iteration error $\hat{\epsilon}^k = \hat{\boldsymbol{x}}^{k+1} - \hat{\boldsymbol{x}}^k$ or global error: $\epsilon^k = \hat{\boldsymbol{x}}^k - \boldsymbol{x}$. With $\mathrm{A} = \mathrm{G} - \mathrm{H}$

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

Existence of a contraction factor $K < 1$ such that $\|\hat{\epsilon}^k\|_\infty \leq K\ \|\hat{\epsilon}^{k-1}\|_\infty$

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

Jacobi, methods of simultaneous displacements

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

Convergence: the global error ϵ^k is controlled by

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

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

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.

Gauss–Seidel, methods of sucessive displacements

In Jacobi iterations, notice that sequential ordered computation of terms

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

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

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

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

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

Gauss–Seidel, methods of sucessive displacements

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 ϵ^k is controlled by

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

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 $\rho(\bar{\mathrm{M}}) < 1$.

Gauss–Seidel, methods of sucessive displacements

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.

Parallelization possible if the matrix is sparse: components do not all possess connectivities with each other:

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.

Relaxation methods

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

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

Relaxation methods

1. Jacobi Over-Relaxation (JOR):

$$\hat{\boldsymbol{x}}_i^{k+1} = (1-\gamma)\,\hat{\boldsymbol{x}}_i^k + \gamma \frac{1}{a_{ii}} \left(b_i - \sum_{i \neq j} a_{ij} \hat{\boldsymbol{x}}_j^k \right) \tag{4}$$

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):

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

which reads in matricial form

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

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

Krylov-subspace methods

Idea: decomposition 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.

To bring the Conjugate Gradient method, first let us introduce the idea of descent method.

Descent methods

Minimisation of the residual:

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

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

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

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.

Steepest Gradient

For the Steepest Gradient:

1. take the direction given by $-\nabla \mathrm{J}(\hat{\boldsymbol{x}}^k) = \boldsymbol{b} - \mathrm{A}\hat{\boldsymbol{x}}^k$ 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 $\mathrm{J}(\hat{\boldsymbol{x}}^k + \alpha_{k+1}\boldsymbol{r}_k)$:

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

1. 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.
2. Gradient direction may not be optimal: Conjugate Gradient methods improve the choice of $\left(\boldsymbol{e}_k\right)$.

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 $\left(\boldsymbol{e}_1, \ldots, \boldsymbol{e}_k, \boldsymbol{e}_{k+1}\right)$ is A-conjugate, thus $\hat{\boldsymbol{x}}^{k+1} = \hat{\boldsymbol{x}}^k + \alpha_{k+1}\boldsymbol{e}_{k+1}$.
2. In doing so we build an orthogonal basis for the scalar product given by A (Gram-Schmidt)
3. choose the descent factor α^{k+1} minimizing the functional $\mathrm{J}(\hat{\boldsymbol{x}}^k + \alpha_{k+1}\boldsymbol{r}_k)$, 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).

Conjugate Gradient

The construction of $\left(\boldsymbol{e}_1, \ldots, \boldsymbol{e}_{k+1}\right)$ is done by orthogonalization of residuals by Gram–Schmidt:

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

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:

$$\boldsymbol{x} = \sum_{j=1}^{N} \alpha_j \hat{\boldsymbol{x}}^j$$

Conjugate Gradient

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.

Conjugate Gradient

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:

$$\boldsymbol{e}_2 = \boldsymbol{r}_1 - \frac{\boldsymbol{e}_1^T \mathrm{A} \boldsymbol{r}_0}{\boldsymbol{e}_1^T \mathrm{A} \boldsymbol{e}_1} \boldsymbol{e}_1$$

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. ...

Conjugate Gradient

The algorithm iteration reads:

1. Compute the residual:

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

2. Compute the direction:

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

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.

Conjugate Gradient

While the residual norm $\varrho_k = \|\boldsymbol{r}_k\|_2$ 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:
$$\boldsymbol{w} = \mathrm{A}\boldsymbol{e}_{k+1}; \alpha_{k+1} = \frac{\varrho_k}{\boldsymbol{e}_{k+1}^T \boldsymbol{w}}$$
4. Update the solution:
$$\hat{\boldsymbol{x}}^{k+1} = \hat{\boldsymbol{x}}^k + \alpha_{k+1} \boldsymbol{e}_{k+1}$$
5. Update the residual:
$$\boldsymbol{r}^{k+1} = \boldsymbol{r}^k - \alpha_{k+1} \boldsymbol{w}$$

Cost is one matrix-multiplication and five vector operations.

Preconditioners

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 $\tilde{\mathrm{A}} = \mathrm{C}^{-1}\mathrm{A}\mathrm{C}$ 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.

Preconditioned CG

While the residual norm $\varrho_k = \|\boldsymbol{r}_k\|_2$ 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:
$$\boldsymbol{r}^{k+1} = \boldsymbol{r}^k - \alpha_{k+1} \boldsymbol{w}$$

Preconditioned CG

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:

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

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

Performance Analysis: Poisson in 2D

Linear communication model:

$$\tau_C k = \tau_S + \gamma \dot{k}$$

For $N = n^2$ degrees of freedom:

- matrix-vector multiplication: 5 multiplications and 4 additions per row.
- scale-vector multplication: N multplications.
- innner-product: N multplications and $N - 1$ additions.

Total operational cost in serial is:

$$9N + 3N + 3N + 4N$$

thus $T_1 = 19N\mathcal{N}_{CG}\tau_A$.

For a one-directional paritioning:

- matrix-vector requires communication for $O(n)$ shared points per rank.
- inner-products require a global reduction.

Total communication cost is:

$$T_{comm} = 4\tau_C(8n) + 2(\tau_A + \tau_C(8))log_2(P)$$

Performance Analysis: Poisson in 2D

If $\tau_A << \tau_C(8)$ ans $\tau_C(8) \approx \tau_S$, total communication cost is:

$$T_{comm} = 4\tau_C(8n) + 2\tau_S log_2(P)$$

Power method

Find the dominant eigenvalues of a matrix $\mathrm{A} \in M_N(\mathbb{R})$ of N eigenvectors $(\boldsymbol{v}_i)$ with associated eigenvalues (λ_i) 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 $(\hat{\boldsymbol{x}}^k)$ 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 $(\boldsymbol{v}_i)$.

Power method

Find the dominant eigenvalues of a matrix $\mathrm{A} \in M_N(\mathbb{R})$ of N eigenvectors $(\boldsymbol{v}_i)$ with associated eigenvalues (λ_i) 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 $(\hat{\boldsymbol{x}}^k)$ 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 $(\boldsymbol{v}_i)$.

Power method

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

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

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.

Power method

— 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.

Software Packages

Libraries like PETSc and Trilinos offer interfaces to:

- a wide-range of iterative solvers based on Krylov-spaces,
- preconditioned by block Jacobi. ILU, AMG, . . .
- so better design your software packages in consequence.