Méthodes Itératives

Carola Kruse, Alena Kopanicakova et Ronan Guivarch

Cours 1, 10/03/2025
Méthodes Multigrilles

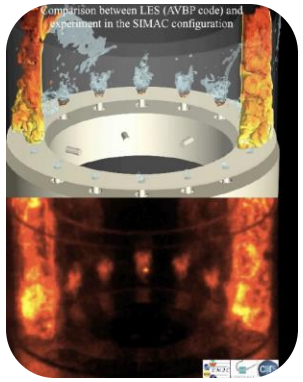
Organisation

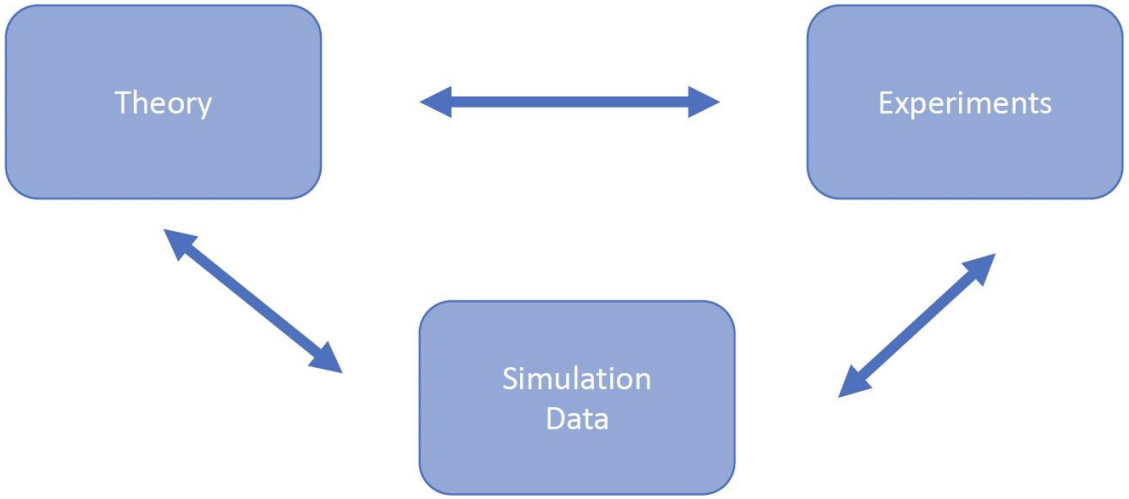
- 4 séances de Cours « Méthodes Multigrilles »
  - 10/03/25 – 10h
  - 11/03/25 – 10h
  - 14/03/25 – 14h
  - 18/03/25 – 18h

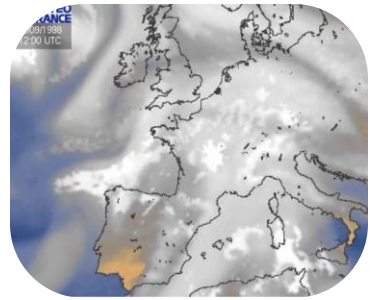
- 2 TP qui vont mettre en pratique la méthode multigrilles
  - 24/03/25 - 14h
  - 24/03/25 - 16h

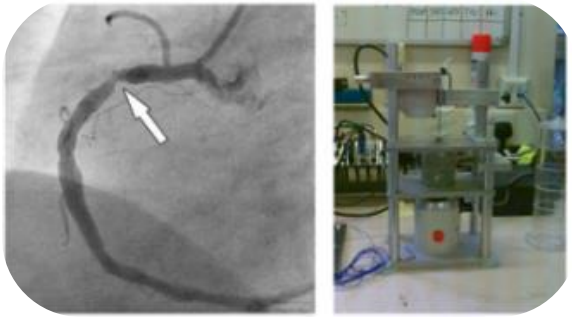
Numerical simulation

Third pillar for the development of scientific discoveries at the same level as theory and experiments.



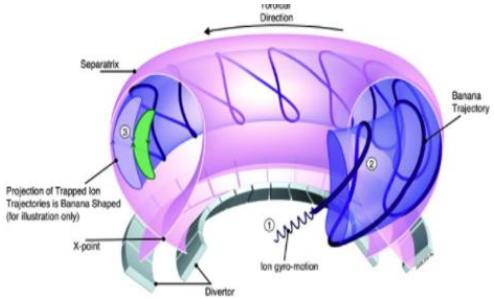






Many problems come down to **solving** a

- **Linear system** of type $Ax = b$
- An eigenvalue problem of type $Ax = \lambda b$



What kind of solvers exists?

Direct methods

- Accurate to machine precision
- High computational cost
- Efficient, parallel sparse direct solvers exist

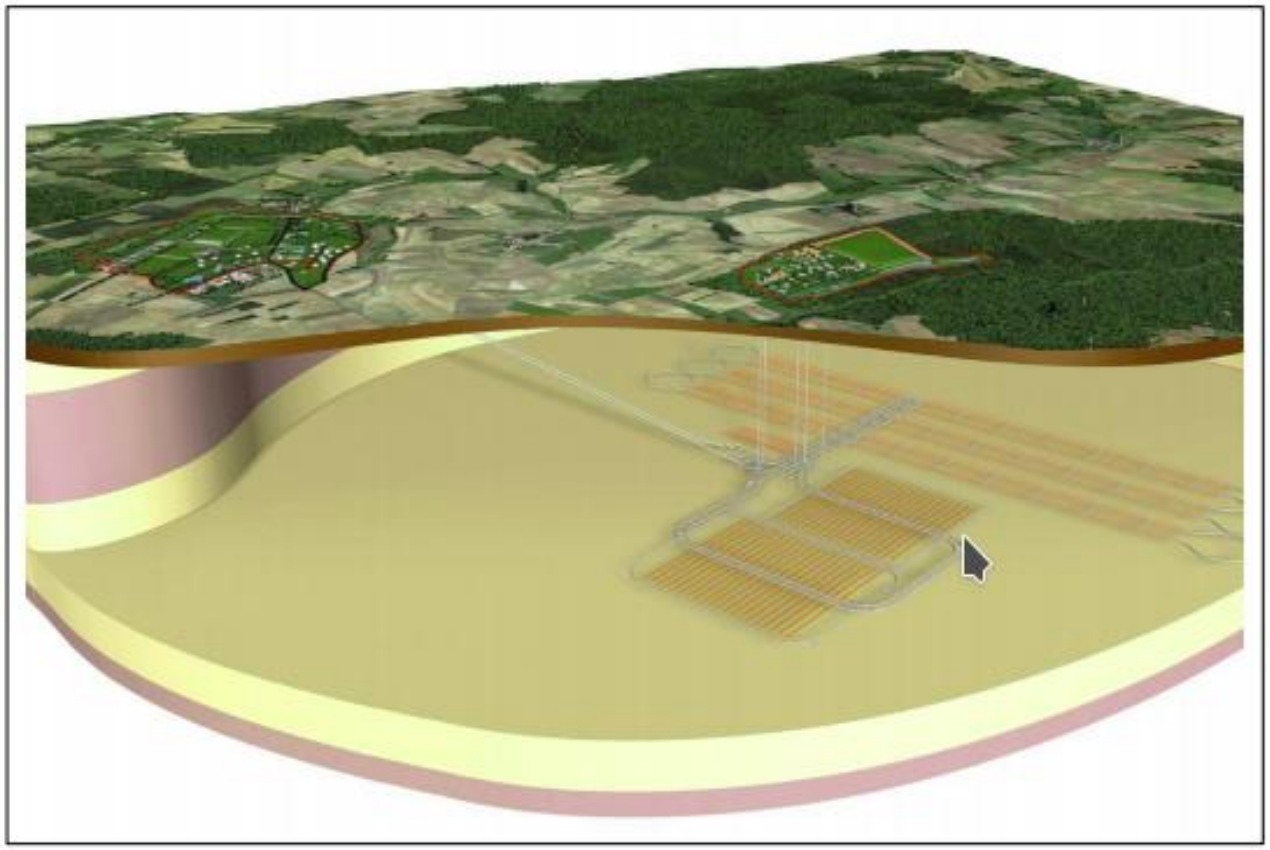
Iterative methods

- Classical relaxation schemes
- Krylov methods
- Preconditioning
- Multigrid methods
- Accurate to a predefined precision
- (Generally) lower computational cost

→ We start with a reminder

→ This will be the first part of the course

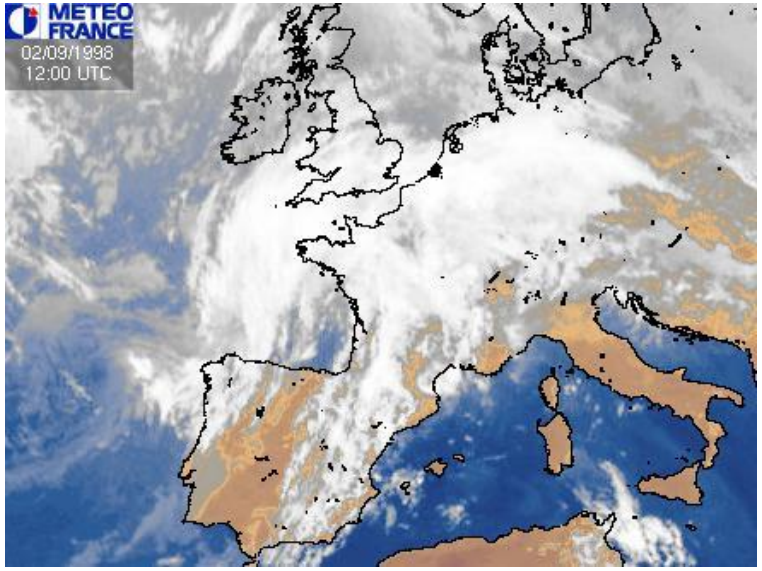
Examples – Discretization of PDE

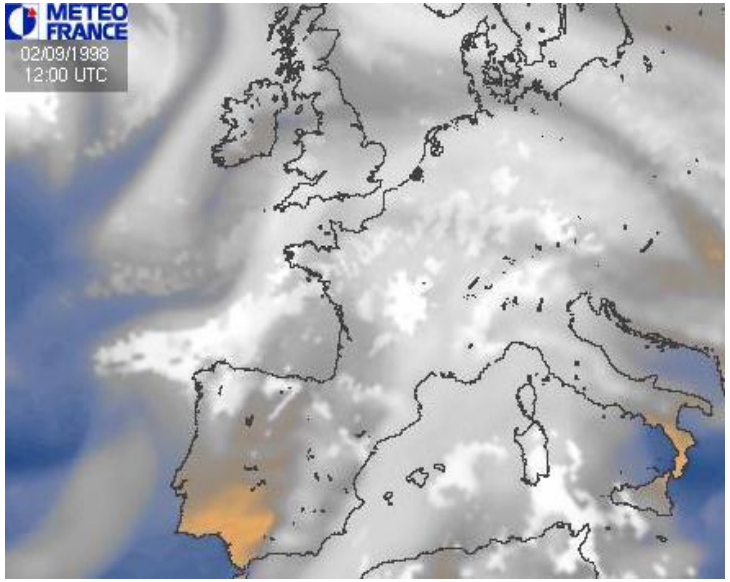


- Gallery crossing model
- 3D thermo-hydro-mechanics modelling (Biot's consolidation problem)
- Discretization of PDE leads to non-symmetric systems with about 100 million unknowns.
- We need fast iterative solvers!

Examples – Weather forecasting

- Time constraints: The weather forecast has to be done for ’soonish’





What will we look at

- Solving linear systems of the form $Ax = b$
- Matrix A shall be sparse (and large)
- Solve this problem iteratively (e.g. stationary methods, Krylov methods, multigrid)
- Do this in an inexpensive and fast way.

Large scale problems – mind experiment

- Let us take a matrix of size $N \times N$.

- Let us suppose that we have three algorithms with different computational complexities
  - **Algorithm 1**: $10^6 N$ operations
  - **Algorithm 2**: $10^3 N^2$ operations
  - **Algorithm 3**: N^3 operations

- Suppose that the problem size N is chosen, such that Algorithm 1 requires 1 second to compute the result.
- How long do the others need?

Speed of algorithms

N	Algorithm 1 $10^6 N$ operations	Algorithm 2 $10^3 N^2$ operations	Algorithm 1 N^3 operations	Computer Speed (ops/sec)
1	1 sec	0.001 sec	0.000001 sec	1M ($\sim 10^6$) (1980's)
$\mathbf{10^3}$	1 sec	1 sec	1 sec	1G ($\sim 10^9$) (1990's)
$\mathbf{10^6}$	1 sec	17 min	12 days	1T ($\sim 10^{12}$) (2000's)
$\mathbf{10^9}$	1 sec	12 days	31710 years	1P ($\sim 10^{15}$) (2010's)

Stronger computers ➡ Greater advantage (necessity) of efficient algorithms!

Message

- Computational complexity is important!

- For very large problems, an algorithm with optimal complexity (i.e., the work grows only linearly with problem size) wins dramatically against an algorithm whose complexity is quadratic.

- Not seen, but: Also the energy consumption will become a bottleneck.

Classic schemes

Classical iterative methods - basic theory

- Let $A \in \mathbb{R}^{n\times n}$ be a non-singular matrix and $\mathbf{f} \in \mathbb{R}^n$.

Problem: Find $\mathbf{u} \in \mathbb{R}^n$, such that

$$A\mathbf{u} = \mathbf{f} \qquad (1)$$

- Decompose the matrix into a non-singular matrix M and a matrix N as

$$A = M - N$$

- Equation (1) can then be written as

$$M\mathbf{u} = N\mathbf{u} + \mathbf{f}$$

or

$$\mathbf{u} = \underbrace{M^{-1}N}_{:=S}\mathbf{u} + M^{-1}\mathbf{f}$$

Classical iterative schemes – fixed point iterations

Let $\boldsymbol{u}^0$ be an initial guess. Then a fixed point iteration can be applied to the equation

$$\mathbf{u}^{(m+1)} = S\mathbf{u}^{(m)} + M^{-1}\mathbf{f}$$

For $\boldsymbol{u}$ being a solution, $\boldsymbol{u}$ must be a fixed point.

This basic iterative approach might also be damped.

$$\mathbf{u}^{\star} = S\mathbf{u}^{(m)} + M^{-1}\mathbf{f}$$
$$\mathbf{u}^{(m+1)} = \omega\mathbf{u}^{\star} + (1-\omega)\mathbf{u}^{(m)}$$

Then we get

$$\mathbf{u}^{(m+1)} = (\omega S + (1-\omega)\mathbf{I})\mathbf{u}^{(m)} + \omega M^{-1}\mathbf{f}$$

The error and residual equation

- Let $\boldsymbol{u}^{(m)}$ be the approximation at step *m*. The error is defined by

$$\mathbf{e}^{(m)} = \mathbf{u} - \mathbf{u}^{(m)}$$

- The residual at step *m* is defined by

$$\mathbf{r}^{(m)} = \mathbf{f} - A\mathbf{u}^{(m)}$$

- For both the weighted and the non-weighted iterative schemes above, the residual equation has the form

$$A\mathbf{e}^{(m)} = \mathbf{r}^{(m)}$$

- In some methods (e.g. multigrid), this residual equation is used to update the new $\boldsymbol{u}^{(m+1)}$ by computing an approximation $\tilde{\boldsymbol{e}}^{(m)}$ and then the new iterate

$$\boldsymbol{u}^{(m+1)} = \boldsymbol{u}^{(m)} + \tilde{\boldsymbol{e}}^{(m)}.$$

The error equation

- Let $\boldsymbol{u}$ be the solution of $Au = f$ and $\boldsymbol{u}^{(m)}$ an approximation. We have

$$\mathbf{u}^{(1)} = S\mathbf{u}^{(0)} + M^{-1}\mathbf{f}$$

- The solution $\boldsymbol{u}$ is a fixed point

$$\mathbf{u} = S\mathbf{u} + M^{-1}\mathbf{f}$$

- Substraction of the initial error

$$\mathbf{e}^{(1)} = S\mathbf{u} - S\mathbf{u}^{(0)} = S\mathbf{e}^{(0)}$$

- Repeating this argument, we have

$$\mathbf{e}^{(m)} = S\mathbf{u} - S\mathbf{u}^{(m-1)} = S\mathbf{e}^{(m-1)}$$

- Recursively, we then get

$$\boxed{\mathbf{e}^{(m)} = S^m\mathbf{e}^{(0)}}$$

Convergence

From the previous equation follows

$$\|\mathbf{e}^{(m)}\| \leq \|S^m\| \|\mathbf{e}^{(0)}\|$$

The iteration is called convergent, if

$$\lim_{m \to \infty} \|S^m\| = 0$$

$\| S \|$ is called contraction number of the fixed point iteration.

Theorem

Let $\rho(S) = \max(\lambda(S))$ be the spectral radius of *S*. The iteration associated with the matrix S converges for **all** initial guesses, **if and only if**

$$\rho(S) < 1.$$

Convergence – Error reduction

- Suppose the matrix is symmetric positive definite (spd). We have

$$\frac{\|\mathbf{e}^{(m)}\|}{\|\mathbf{e}^{(0)}\|} \leq \|S^m\| \approx \rho(S^m) = \rho(S)^m \approx 10^{-1}$$

- How many iterations do we need to guarantee the reduction of the error by a factor of 10?

$$m \geq -\frac{1}{\log_{10}|\rho(S)|}$$

- Convergence **factor** $\rho(S)$ → Loosely speaking: The worst factor for the Reduction of the error in each step.
- Convergence **rate** $-\log_{10}|\rho(S)|$ → The convergence becomes the faster, the higher the convergence rate.

Jacobi method

- Correct x_i^{old} by x_i^{new} such that the *i-th* equation of $A\boldsymbol{x} = f$ is correct.

$$\begin{aligned} a_{11} \cdot x_1 + \cdots + a_{1n} \cdot x_n &= b_1 \\ a_{21} \cdot x_1 + \cdots + a_{2n} \cdot x_n &= b_2 \\ &\vdots \\ a_{n1} \cdot x_1 + \cdots + a_{nn} \cdot x_n &= b_n \end{aligned}$$

- Then the Jacobi method is given by

$$x_i^{(m+1)} := \frac{1}{a_{ii}} \left(b_i - \sum_{i \neq j} a_{ij} x_j^{(m)} \right), \qquad i = 1, .., n$$

Jacobi Method

- Let D be the diagonal part and $-L$ and $-U$ the strictly lower and upper triangular parts of A, thus

$$A = D - L - U$$

Jacobi Method

- Calculate the relaxations simultaneously for all $i = 1, \dots, n$. In matrix notation

$$\mathbf{x}^{(m+1)} = D^{-1}(L + U)\mathbf{x}^{(m)} + D^{-1}\mathbf{f}$$

Define
$S_J := D^{-1}(L + U)$

Weighted Jacobi Method

- Let $\omega \in \mathbb{R}$ be a weighting factor. Then

$$\mathbf{x}^{(m+1)} = [(1 - \omega)\mathbf{I} + \omega S_J]\,\mathbf{x}^{(m)} + \omega D^{-1}\mathbf{r}^{(m)}$$

Define
$S_\omega := (1 - \omega)I + \omega S_J$

Gauss-Seidel Method

- The Gauss-Seidel method is given by

$$x_k^{(m+1)} := \frac{1}{a_{kk}} \left(b_k - \sum_{i=1}^{k-1} a_{ki} x_i^{(m+1)} - \sum_{i=k+1}^{n} a_{ki} x_i^{(m)} \right)$$

$$\begin{aligned} a_{11} \cdot x_1 + \cdots + a_{1n} \cdot x_n &= b_1 \\ a_{21} \cdot x_1 + \cdots + a_{2n} \cdot x_n &= b_2 \\ &\vdots \\ a_{n1} \cdot x_1 + \cdots + a_{nn} \cdot x_n &= b_n \end{aligned}$$

- Components of the new approximation are used as soon as they are computed
- Gauss-Seidel method is sequential: To compute a new entry, we first need to compute all previous entries.

Relaxation schemes

Gauss-Seidel method

- Calculate an entry x_k of the new iteration and use it in the computation of $x_i, i = k+1, .., n$.

$$\mathbf{x}^{(m+1)} = (\mathbf{D} - \mathbf{L})^{-1}\mathbf{U}\mathbf{x}^{(m)} + (\mathbf{D} - \mathbf{L})^{-1}\mathbf{f}$$

Define
$S_{GS} := (D - L)^{-1}U$

SOR method

$$\mathbf{x}^{(m+1)} = (\mathbf{D} - \omega\mathbf{L})^{-1}(\omega\mathbf{U} - (\omega - 1)\mathbf{D})\mathbf{x}^{(m)} + (\frac{1}{\omega}\mathbf{D} - \mathbf{L})^{-1}\mathbf{f}$$

Define
$S_{GS\omega} := (D - \omega L)^{-1}(-\omega U + (\omega - 1)D)$

- If $\omega = 1$, then the Gauss-Seidel method is recovered.

Properties

Weighted Jacobi

- For $A \in \mathbb{R}^{n \times n}$, weighted Jacobi converges for all initial $\boldsymbol{x}^0$, if $\omega \in (0, 2/(\lambda_{\max} \quad (D^{-1}A))$.
- The rate of convergence depends on ω.
- The convergence does not depend on the numbering of the unknowns.
- Entries can be computed all at once.

$$\begin{aligned} a_{11} \cdot x_1 + \cdots + a_{1n} \cdot x_n &= b_1 \\ a_{21} \cdot x_1 + \cdots + a_{2n} \cdot x_n &= b_2 \\ &\vdots \\ a_{n1} \cdot x_1 + \cdots + a_{nn} \cdot x_n &= b_n \end{aligned}$$

SOR

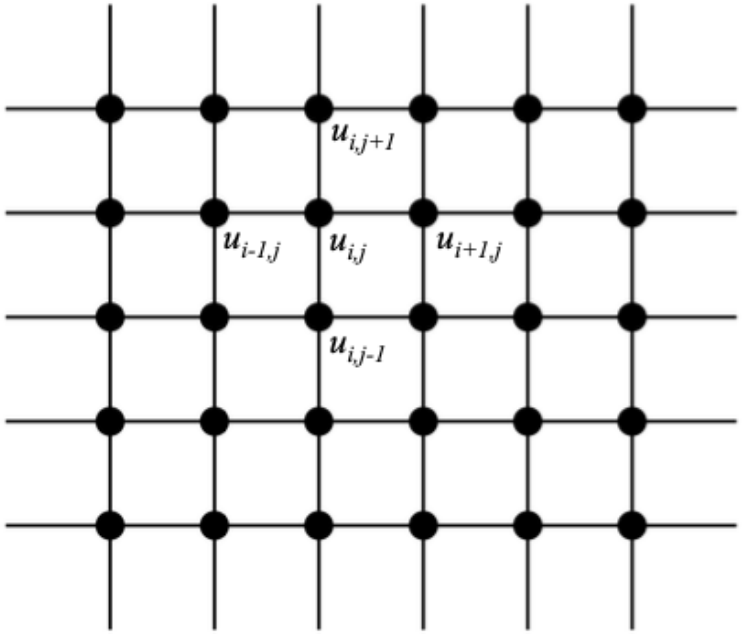
- For $A \in \mathbb{R}^{n \times n}$, SOR converges for all initial $\boldsymbol{x}^0$, if $\omega \in (0, 2)$.
- The rate of convergence depends on ω.
- The convergence depends on the numbering of the unknowns.
- We could also first solve for x_{nn} instead of x_{11}
- We could also alternate between ascending and descending order (symmetric Gauss-Seidel)

Example: 2D Poisson equation

$$
\begin{aligned}
-\Delta u &= f, && \text{in } \Omega = [0,\pi] \times [0,\pi] \\
u &= u_0, && \text{on } \Gamma = \partial\Omega
\end{aligned}
$$

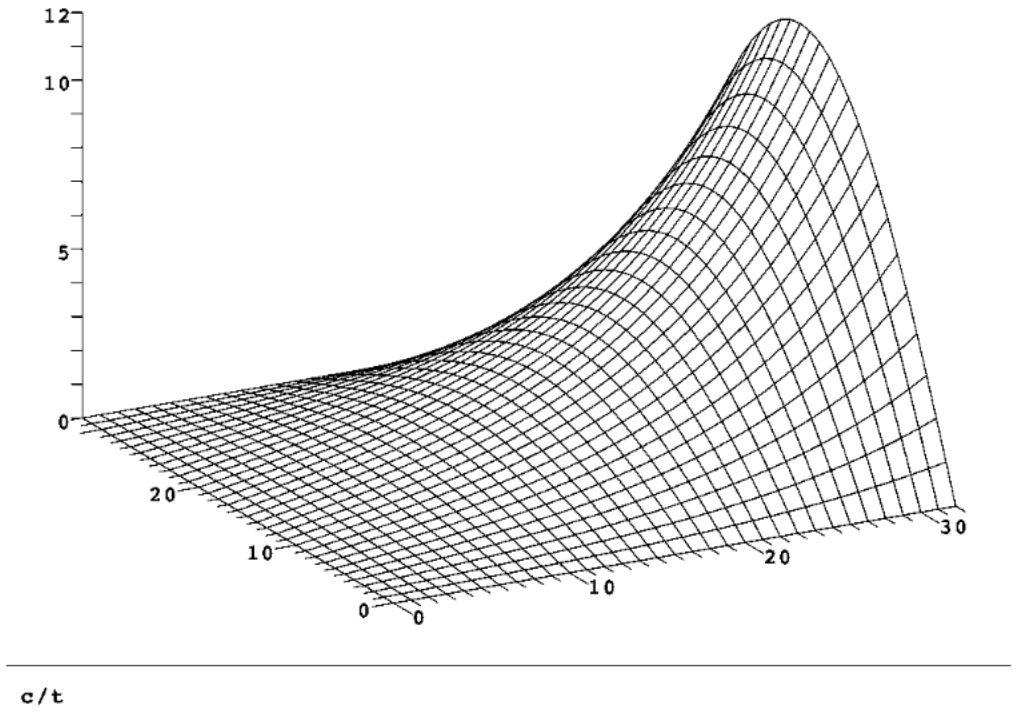
- Discretize Ω into a regular grid with *N* intervals in *x* and *y*
- We discretize using a 5-point stencil

$$\Delta u_{ij} \approx \frac{1}{h^2}(u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{ij}), \qquad \text{for } i,j = 1,...,n$$



- Boundary values are given.
- Imagine: Temperature at each node = average of neighboring temperatures.

Choice of an exact solution



We choose

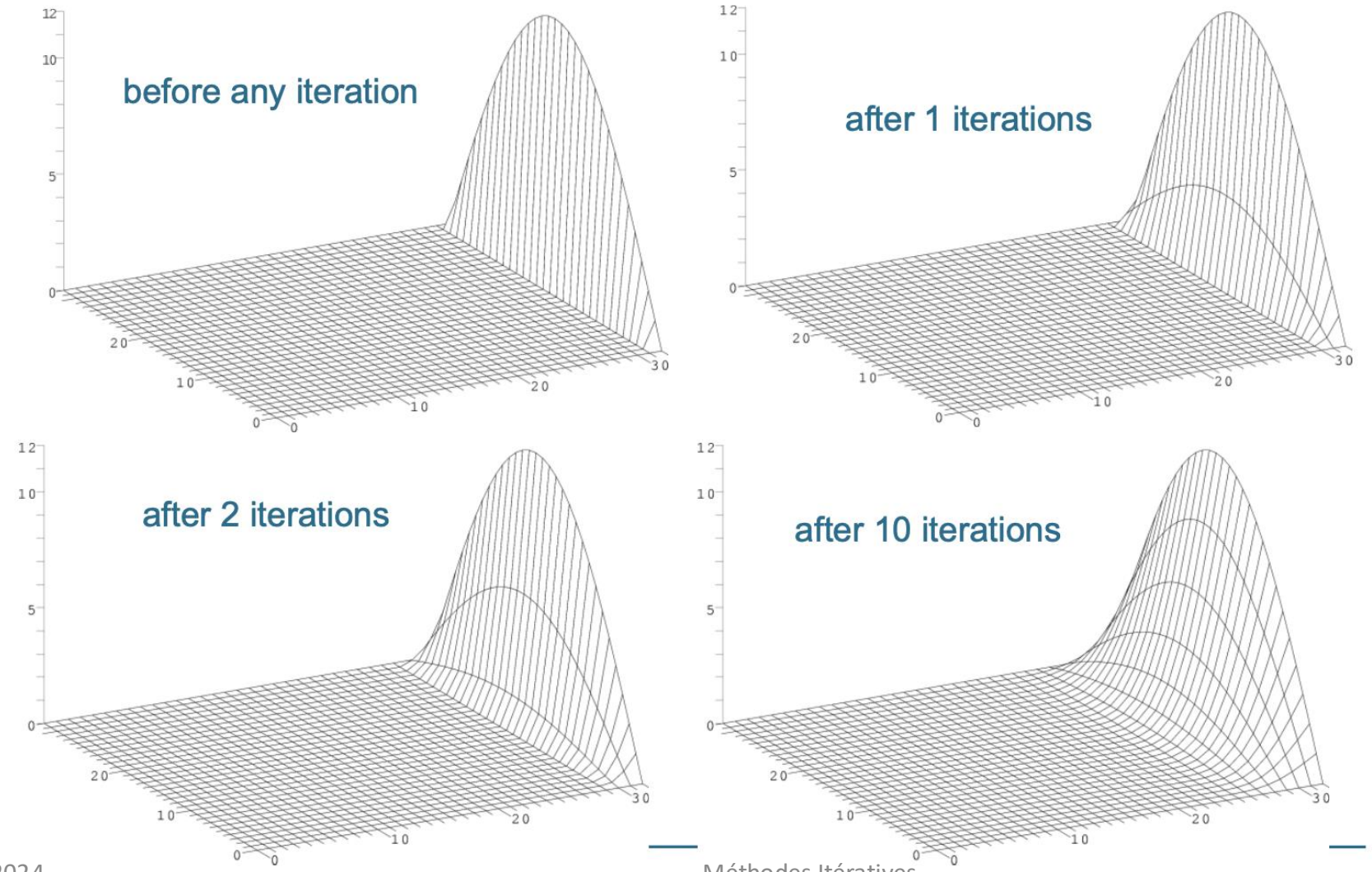
$$u = \sinh(x)\sin(y)$$

and obtain the right-hand side

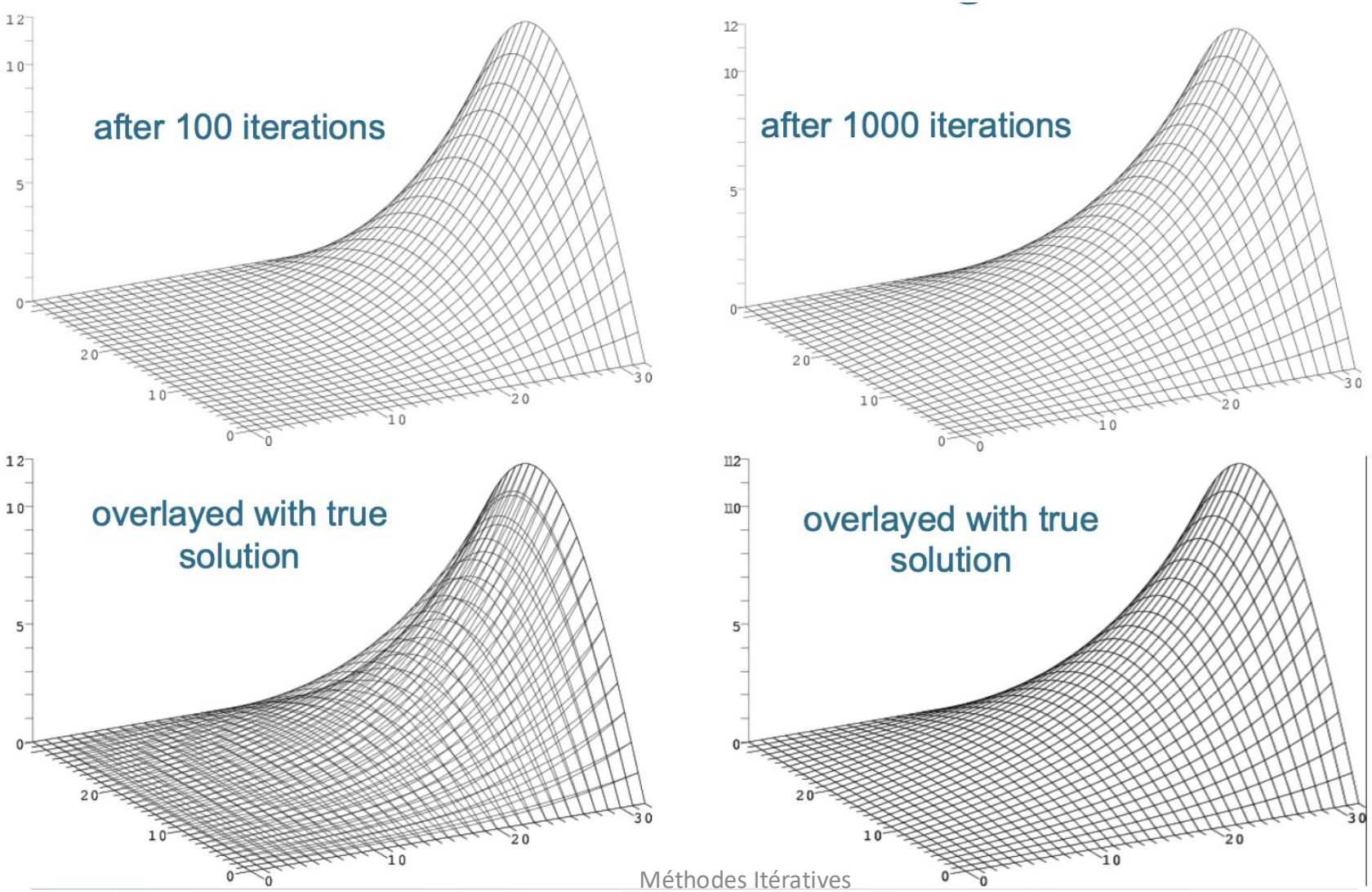
$$f = 0$$

- We can thus compute the error at each node.
- We discretize the domain $\Omega = [0, \pi] \times [0, \pi]$ into a mesh with n intervals, so the matrix is of size $N = n^2$ and $h = \frac{\pi}{n}$.

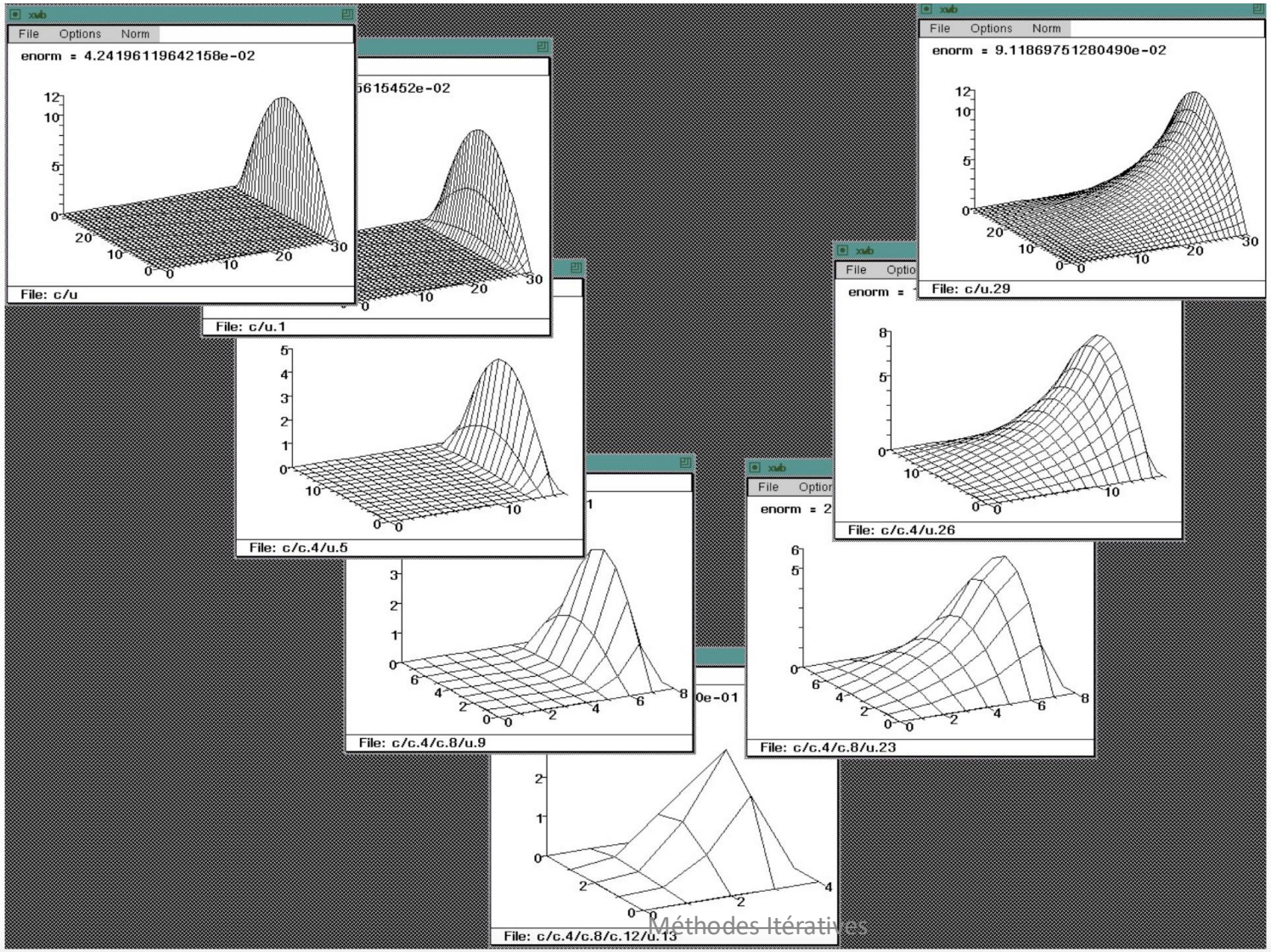
Visualization of Gauss-Seidel iterations



Visualization of Gauss-Seidel iterations



Multigrid


File Options Norm
enorm = 4.24196119642158e-02
File: c/u
File: c/u.1
File: c/u.29
File: c/c.4/u.5
File: c/c.4/u.26
File: c/c.4/c.8/u.9
File: c/c.4/c.8/u.23
File: c/c.4/c.8/c.12/u.13

Trying out algorithms

- Let us now apply some algorithms to the 2D Poisson problem.

level	h	d.o.f.	SSOR		PCG		MG		FGMRES+MG		UMFPACK	
			ite	time	ite	time	ite	time	ite	time	ite	time
1	1/4	25	49	0	3	0	11	0	6	0	1	0
2	1/8	81	164	0	9	0	13	0	8	0	1	0
3	1/16	289	543	0	31	0	13	0	8	0	1	0
4	1/32	1089	2065	0.07	66	0.01	14	0.03	8	0.01	1	0.01
5	1/64	4225	7998	0.92	132	0.02	14	0.11	8	0.03	1	0.03
6	1/128	16641	31054	14.61	263	0.16	13	0.35	8	0.21	1	0.12
7	1/256	66049	> 100000		524	1.79	13	1.55	8	1.06	1	0.75
8	1/512	263169			1038	16.55	12	6.09	8	3.90	1	5.40
9	1/1024	1050625			1986	127.76	12	27.46	7	18.32	1	46.46
10	1/2048	4198401			3944	1041.68	12	111.03	7	68.38		
factor $\approx$		4	4	16	2	8	1	4	1	4	1	

Table taken from the lecture notes 'Multigrid Methods by Volker John, Winter Semester 2013/2014.

Multigrid

- Framework: common concepts, different methods
- Efficient: usually *O(N*) or *O(N log N)* operations
- The importance of efficient methods becomes greater, as computers grow stronger!
- Iterative: Most nontrivial problems in our field cannot be solved directly
- Many variables: the larger the number of variables, the greater the gain of efficient methods

A framework of efficient iterative methods for solving problems with many variables and many scales.

This lecture is based on:

- A Multigrid Tutorial – W.L. Briggs, V.E. Henson, S.F. McCormick
- Why Multigrid Methods Are So Efficient - Irad Yavneh, Computing in Science and Engineering, 8(6):12 – 22
- Multigrid Methods, the basics - Luke Olson, Copper Mountain Multigrid Conference tutorial, 2021
- Multi-Grid Methods and Applications – W. Hackbusch

Some nice illustrations in jupyter notebooks:

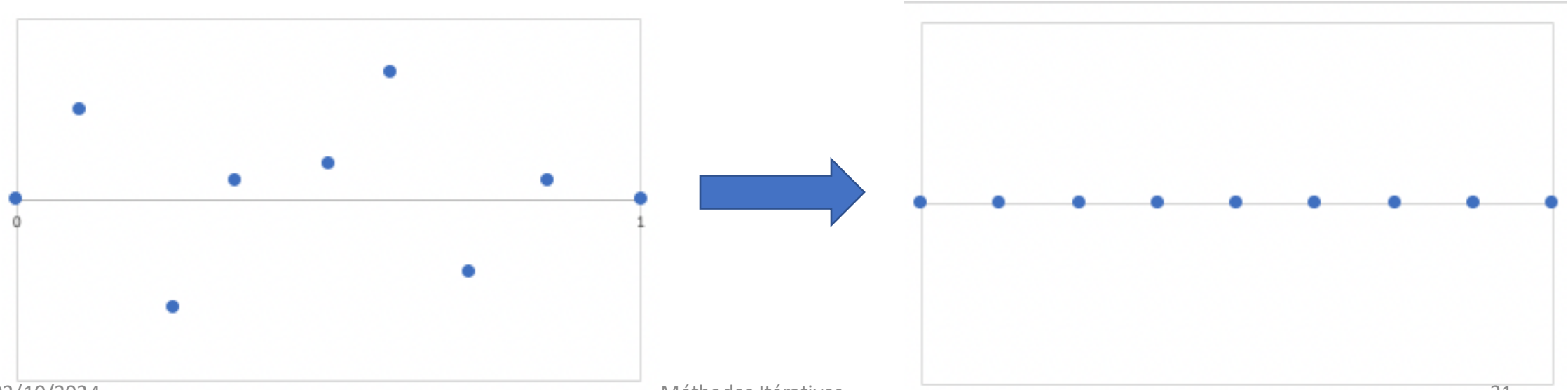
- https://github.com/lukeolson/imperial-multigrid/tree/master/lecture-1-mg-basics

A multiscale approach

Following Irad Yaneh (*Why Multigrid Methods Are So Efficient)*

Imagine:

- A football coach wants to line up his players in equal distances on the goal line.
- Player *0* and player *N* shall stand on the left and right goal post.
- The remaining players stand in between them. He now moves them according to some rule.



1. possibility: Global processing

- The coach numbers its players from *0* to *N*.
- He tells player *j* to move to the goal line connecting player 0 to player *N* and then at a distance $j\frac{L}{N}$ from player 0.
- This solves the problem directly.

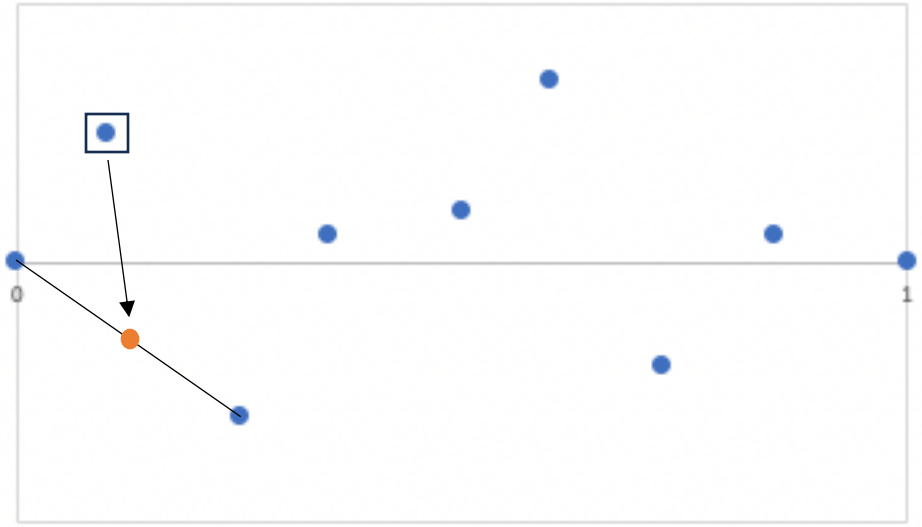
But

- Each player has to recognize the extreme players to find the 'goal line'.
- He has to do some 'fancy' arithmetic $\left(\frac{L}{N}\right)$, and has to move *j* times of this.

Let us see how we could achieve the same goal with some local processing.

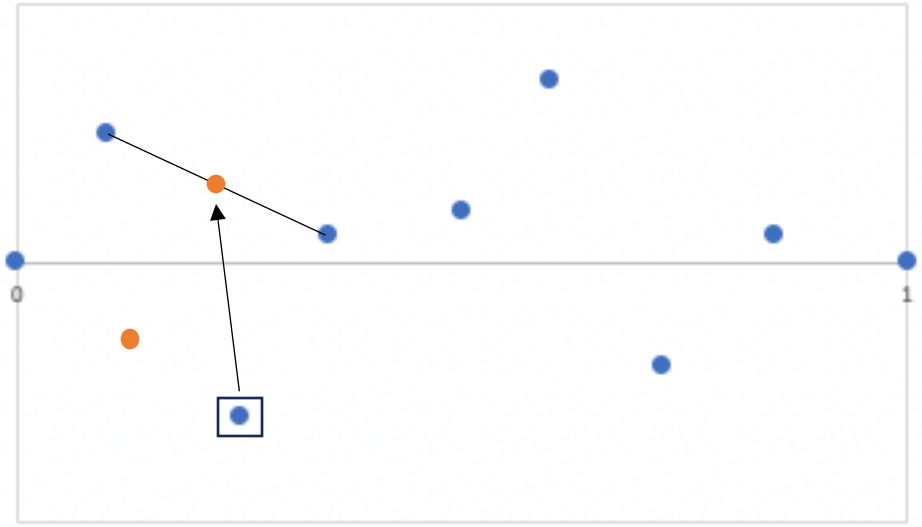
Local movements

- We now require that each player detects the two players next to him and moves to the midpoint of the straight line connecting them.
- This is done at the whistle blow of the coach.
- Blue: initial position, Orange: new position



Local movements

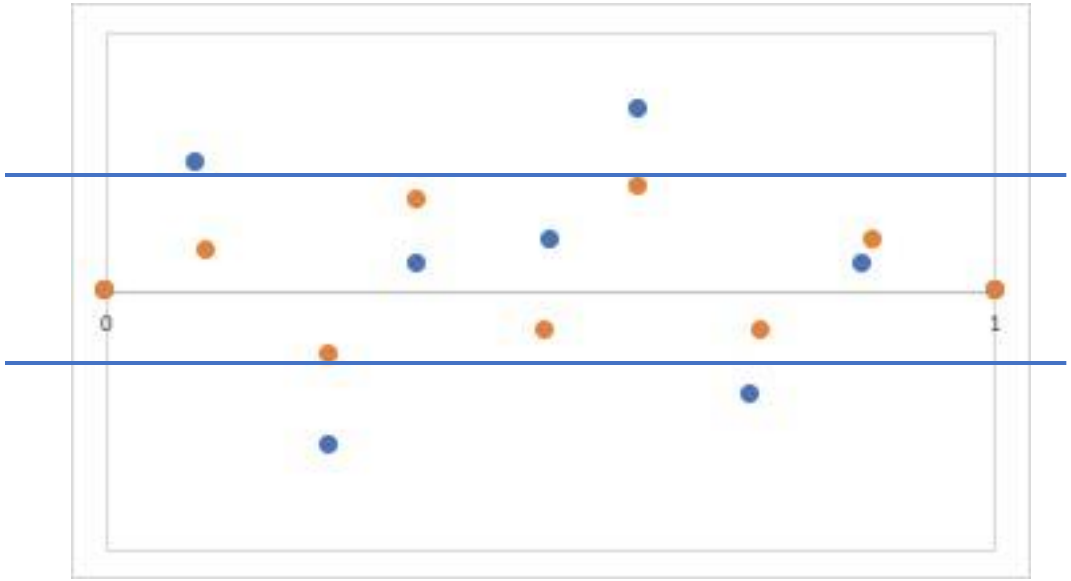
- We now require that each player detects the two players next to him and moves to the midpoint of the straight line connecting them.
- This is done at the whistle blow of the coach.
- Blue: initial position, Orange: new position



Local movements

- We now require that each player detects the two players next to him and moves to the midpoint of the straight line connecting them.
- This is done at the whistle blow of the coach.
- Blue: initial position, Orange: new position

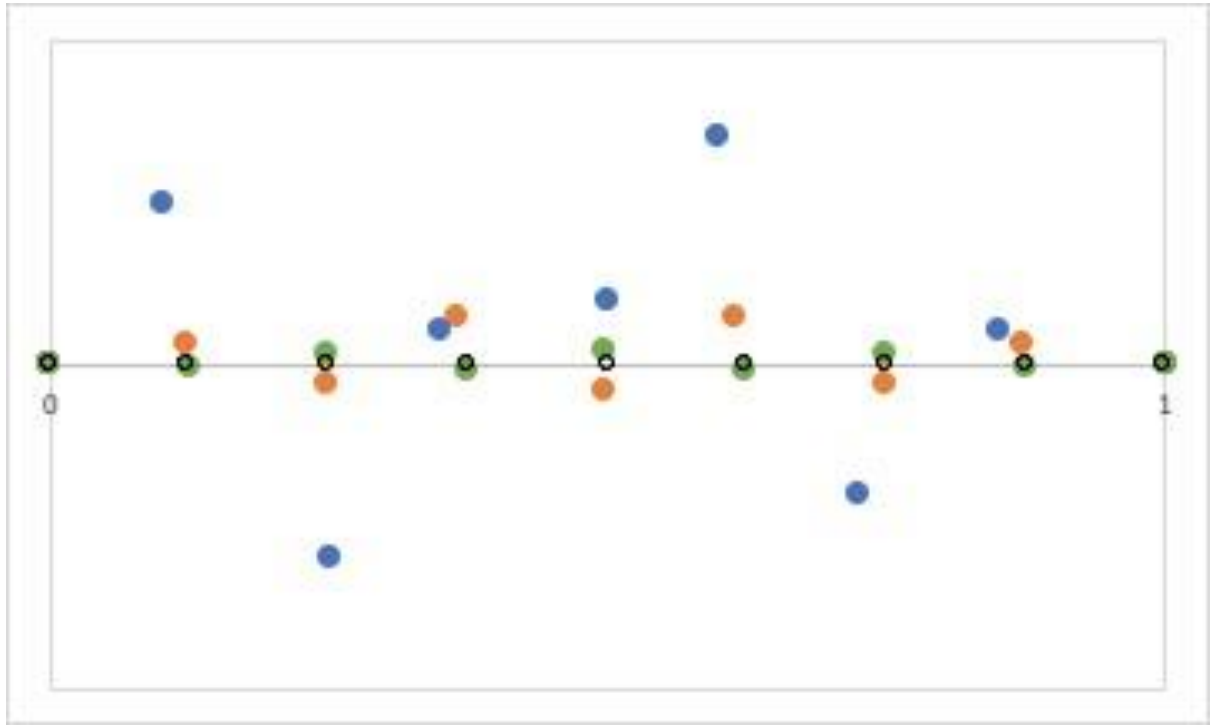
After one iteration



After two iterations

Local movements

- Blue: initial position, black: final position
- orange: after 15 iterations, green: after 30 iterations, black: after 64 iterations ($err \approx 5 \cdot 10^{-3}$)



Local movements – as formula

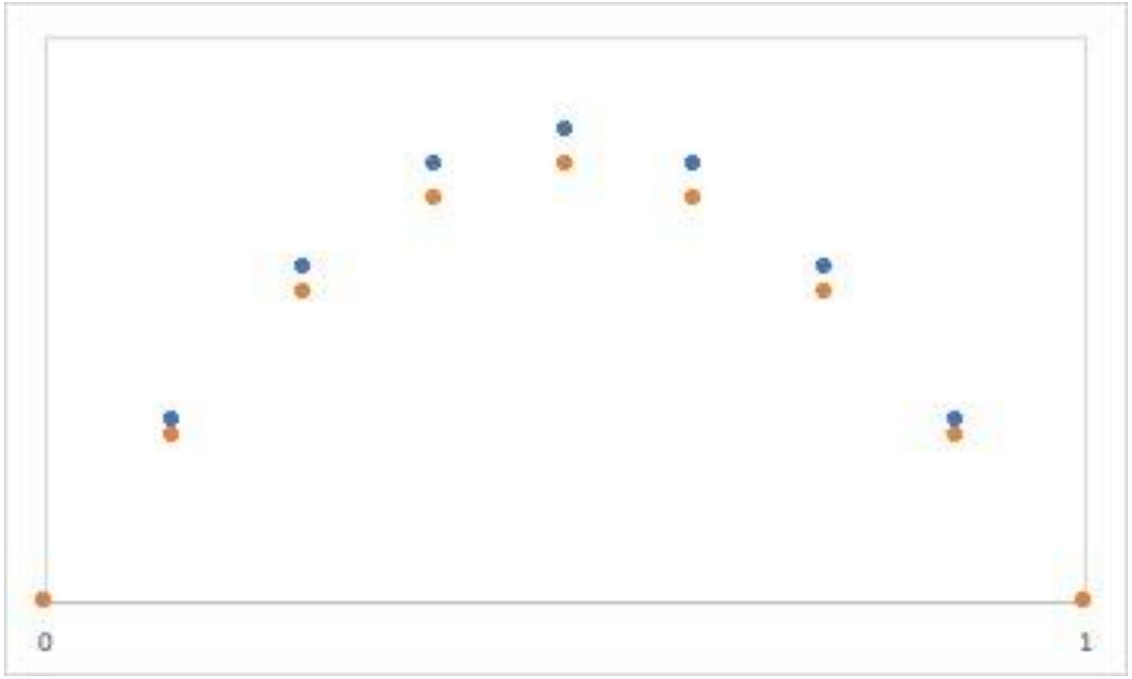
- Let us look at the distance of each player to the goal line. We have 9 players (N=8), with 7 players moving.
- At each whistle blow, the following update is done

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_7 \end{pmatrix}^{(m+1)} = \frac{1}{2} \begin{pmatrix} 0 & 1 & 0 & \dots & 0 \\ 1 & 0 & 1 & \ddots & \vdots \\ 0 & 1 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & 0 & 1 \\ 0 & \dots & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_7 \end{pmatrix}^{(m)}$$

- This is nothing else than the Jacobi method for the 1D Poisson problem with mesh size 1. We derive this later on.

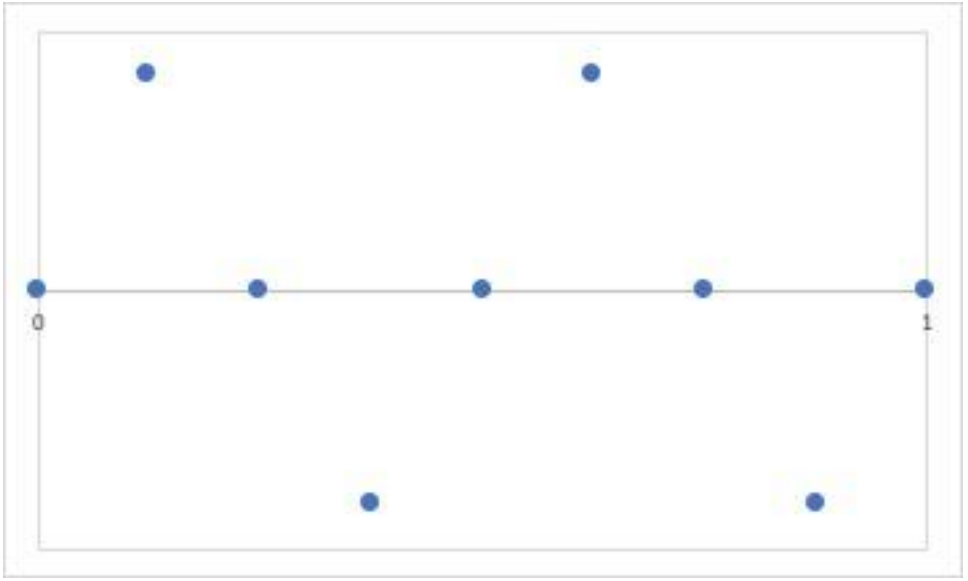
Some slow convergence

- Blue: initial position,
- Orange: new position after the whistle blow



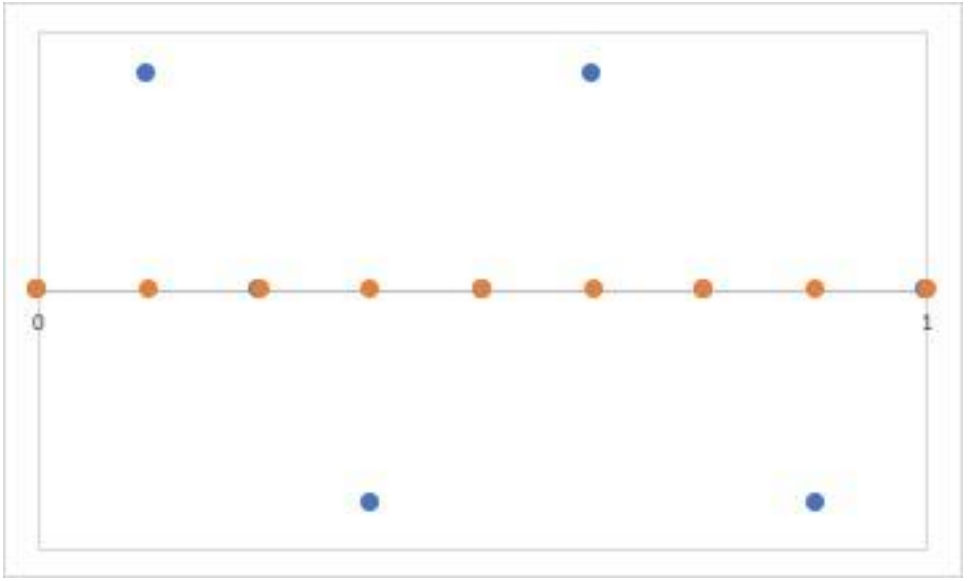
Fast convergence

- Blue: initial position,
- Orange: new position after the whistle blow



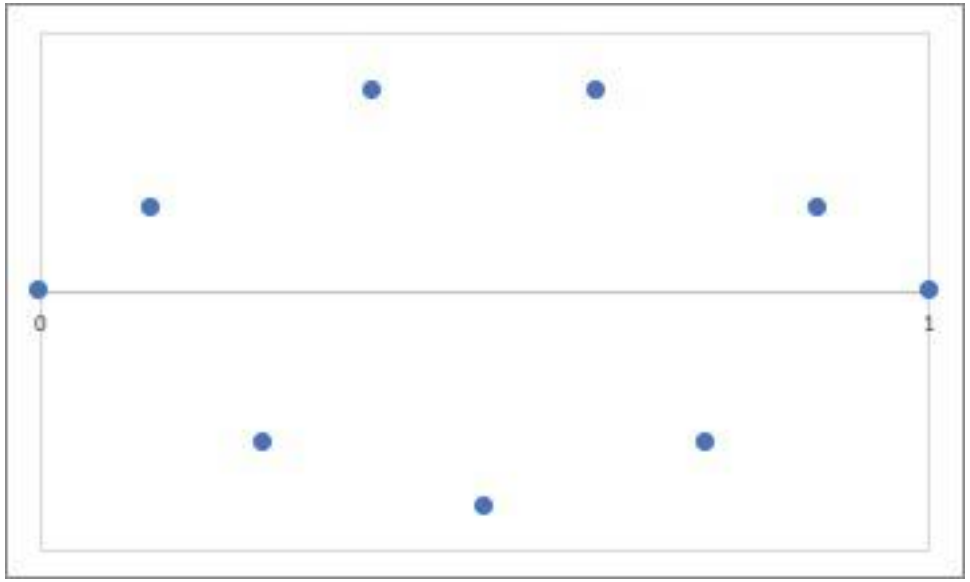
Fast convergence

- Blue: initial position,
- Orange: new position after the whistle blow



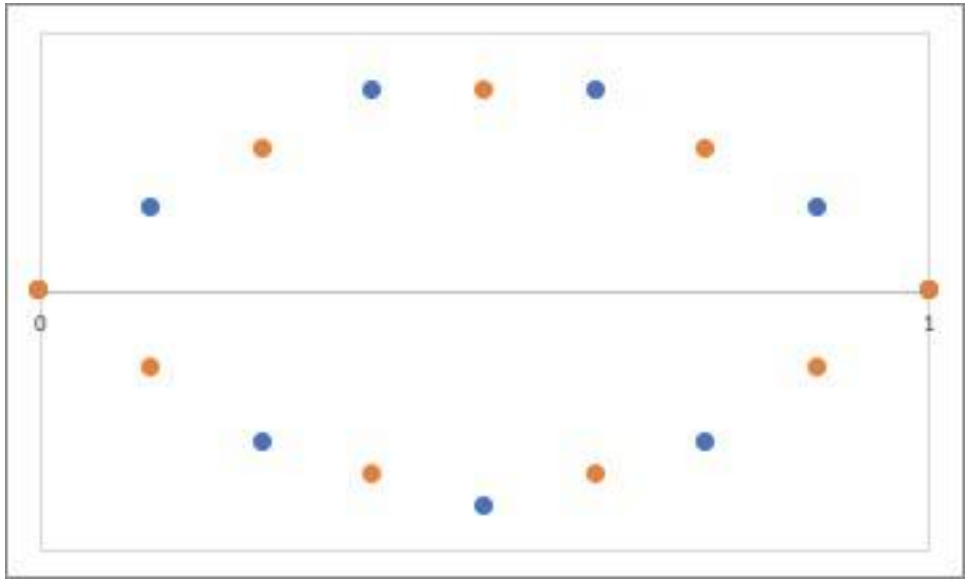
Slow convergence

- Blue: initial position,
- Orange: position after one whistle blow
- Green: position after two whistle blows



Slow convergence

- Blue: initial position,
- Orange: position after one whistle blow
- Green: position after two whistle blows

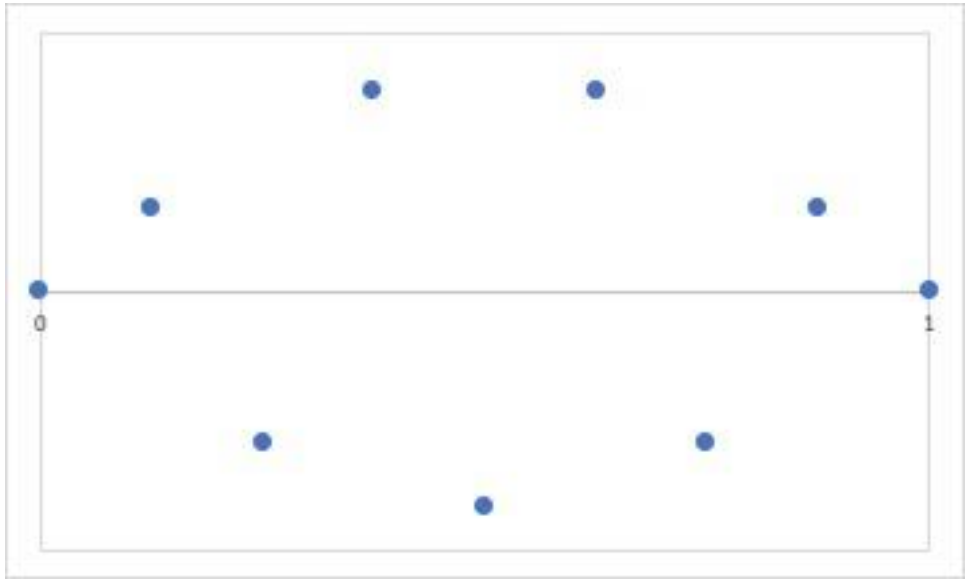


Slow convergence

- Blue: initial position,
- Orange: position after one whistle blow
- Green: position after two whistle blows

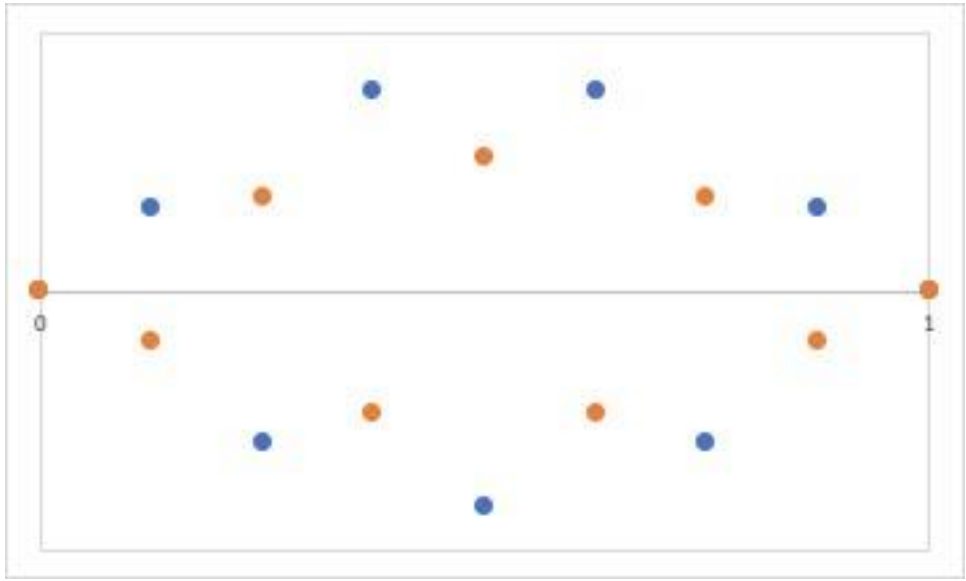
Slow convergence - improvement through damping

- Blue: initial position,
- Orange: position after one whistle blow
- Green: position after two whistle blows



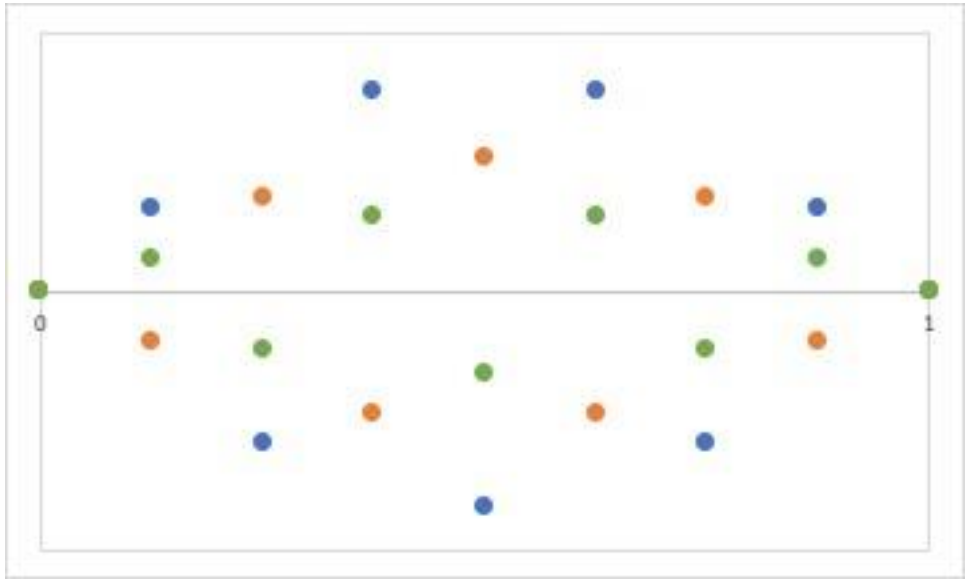
Slow convergence - improvement through damping

- Blue: initial position,
- Orange: position after one whistle blow
- Green: position after two whistle blows



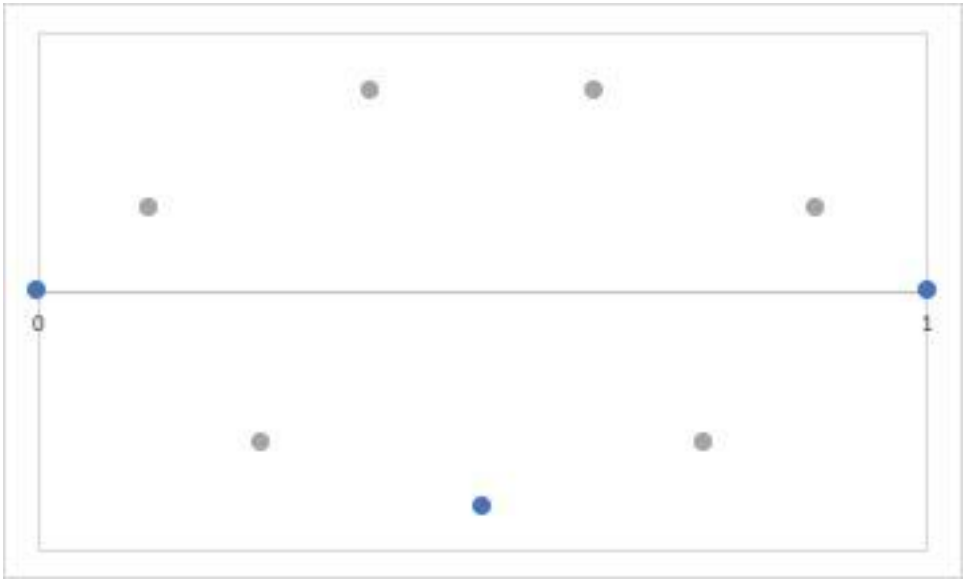
Slow convergence - improvement through damping

- Blue: initial position,
- Orange: position after one whistle blow
- Green: position after two whistle blows



The Multi-scale approach

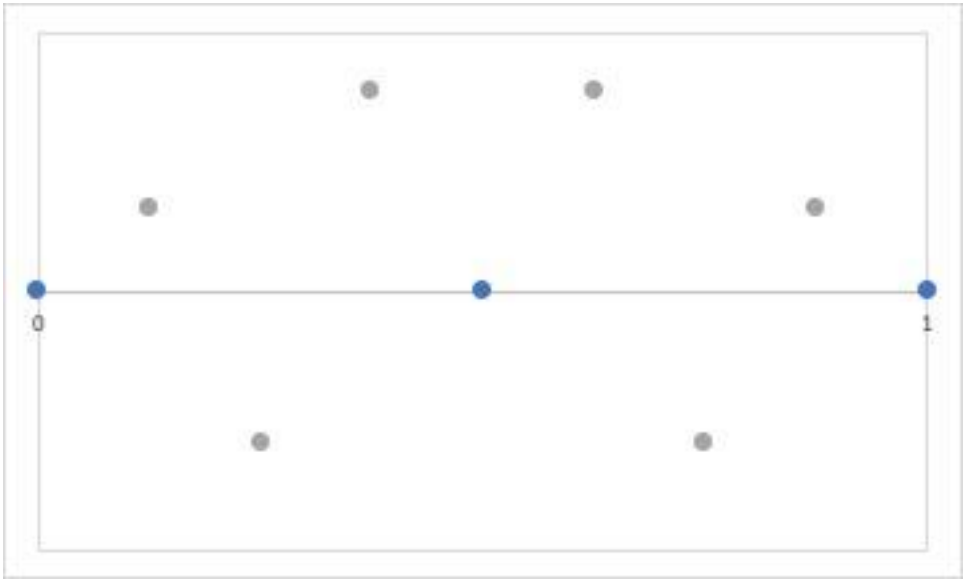
- Employ the local processing with simple arithmetic.
- Do this on different scales.



Large scale

The Multi-scale approach

- Employ the local processing with simple arithmetic.
- Do this on different scales.



Large scale

The Multi-scale approach

- Employ the local processing with simple arithmetic.
- Do this on different scales.

Intermediate scale

The Multi-scale approach

- Employ the local processing with simple arithmetic.
- Do this on different scales.

Intermediate scale

The Multi-scale approach

- Employ the local processing with simple arithmetic.
- Do this on different scales.

Small scale

The Multi-scale approach

- Employ the local processing with simple arithmetic.
- Do this on different scales.

Small scale

How much do we save?

- The local movements (Jacobi method) require about N^2 iterations and $N^2 \cdot N = N^3$ operations to improve the accuracy by an order of magnitude

- The multiscale approach solves the problem in about $\log_2(\mathrm{N})$ iterations (whistle blows) and only about *N* operations.

- Example: For *N=1000* we require about
  - Multiscale: 10 iterations and 1000 operations
  - Jacobi: 1 000 000 iterations and 1 000 000 000 operations

Geometric Multigrid methods

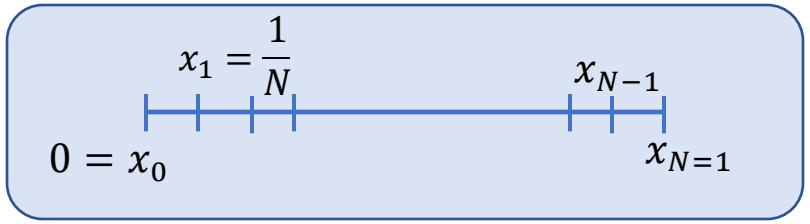
Elements of multigrid

Relaxation schemes and smoothing properties

Model problem: 1D Poisson equation

- The 1D Poisson equation:

$$-u'' = f$$
$$u(0) = u(1) = 0$$



- Grid points

$$h = \frac{1}{N}, \quad x_i = ih, \quad i = 0, .., N$$

- Finite difference scheme

$$u_i'' \approx \frac{-u_{i-1} + 2u_i - u_{i+1}}{h^2} = f_i, \qquad i = 1, .., N-1$$

$$u_0 = u_N = 0$$

- We obtain a linear system

$$A\mathbf{u} = \frac{1}{h^2} \begin{pmatrix} 2 & -1 & 0 & \dots & 0 \\ -1 & 2 & -1 & \ddots & \vdots \\ 0 & -1 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & 2 & -1 \\ 0 & \dots & 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_{N-1} \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_{N-1} \end{pmatrix}$$

Application to our example: Jacobi method

- Remember the discretization at point i (this gives one matrix row):

$$-u_{i-1} + 2u_i - u_{i+1} = h^2 f_i$$

$$A = \frac{1}{h^2}\begin{pmatrix} 2 & -1 & 0 & \dots & 0 \\ -1 & 2 & -1 & \ddots & \vdots \\ 0 & -1 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & 2 & -1 \\ 0 & \dots & 0 & -1 & 2 \end{pmatrix}$$

- Solving for u_i:

$$u_i = \frac{1}{2}(u_{i-1} + u_{i+1} + h^2 f_i)$$

- We can write this iteratively as

$$u_i^{(m+1)} = \frac{1}{2}(u_{i-1}^{(m)} + u_{i+1}^{(m)} + h^2 f_i)$$

- This can be expressed in matrix form as

$$\begin{aligned} \mathbf{u}^{(m+1)} &= D^{-1}(L+U)\mathbf{u}^{(m)} + D^{-1}h^2\mathbf{f} \\ &= \underbrace{(I - D^{-1}A)}_{:=R}\mathbf{u}^{(m)} + D^{-1}h^2\mathbf{f} \end{aligned}$$

$$R = \frac{1}{2}\begin{pmatrix} 0 & 1 & & \\ 1 & 0 & \ddots & \\ & \ddots & \ddots & 1 \\ & & 1 & 0 \end{pmatrix}$$

Weighted Jacobi method

To increase convergence properties, we next look at the weighted Jacobi method.

We compute

$$u_i^\star = \frac{1}{2}(u_{i-1} + u_{i+1} + h^2 f_i)$$

This is now only an intermediate step, and we define the new weighted iterate by

$$u_j^{(m+1)} = (1-\omega)u_j^{(m)} + \omega u_i^\star, \qquad i \le j \le N-1$$

In matrix notation, we can express it by

$$\begin{aligned}\mathbf{u}^{(m+1)} &= \left[(1-\omega)\mathbf{I} + \omega S_J\right]\mathbf{u}^{(m)} + \omega D^{-1}\mathbf{f}^{(m)} \\ &= (1-\omega D^{-1})A\mathbf{u}^m + \omega D^{-1}\mathbf{f}\end{aligned}$$

Define
$R_\omega := (I - \omega D^{-1}A)$

Eigenvectors and Eigenvalues

- The matrix is spd and sparse (not more than 3 non-zero entries per row and column).

$$A = \frac{1}{h^2}\begin{pmatrix} 2 & -1 & 0 & \dots & 0 \\ -1 & 2 & -1 & \ddots & \vdots \\ 0 & -1 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & 2 & -1 \\ 0 & \dots & 0 & -1 & 2 \end{pmatrix}$$

- It has the eigenvalues $(\lambda_k \mathbf{w}_k = A\mathbf{w}_k)$

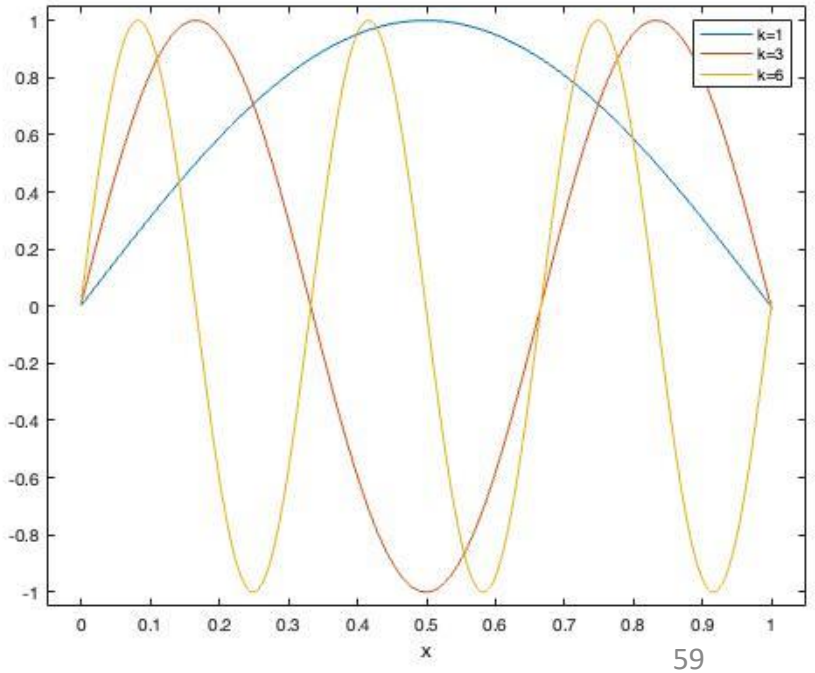
$$\lambda_k = \frac{4}{h^2}\sin^2\left(\frac{k\pi}{2N}\right)$$

- And the eigenvectors

$$(w_k)_j = \sin\left(\frac{jk\pi}{N}\right)$$

- This denotes the *j*-th component of the *k*-th eigenvector.

- The eigenvectors are Fourier modes.



Definition

Modes in the upper half of the spectrum, i.e.

$$\frac{N}{2} \leq k < N - 1$$

are called high-frequency or oscillatory modes.

Modes in the lower half of the spectrum, i.e.

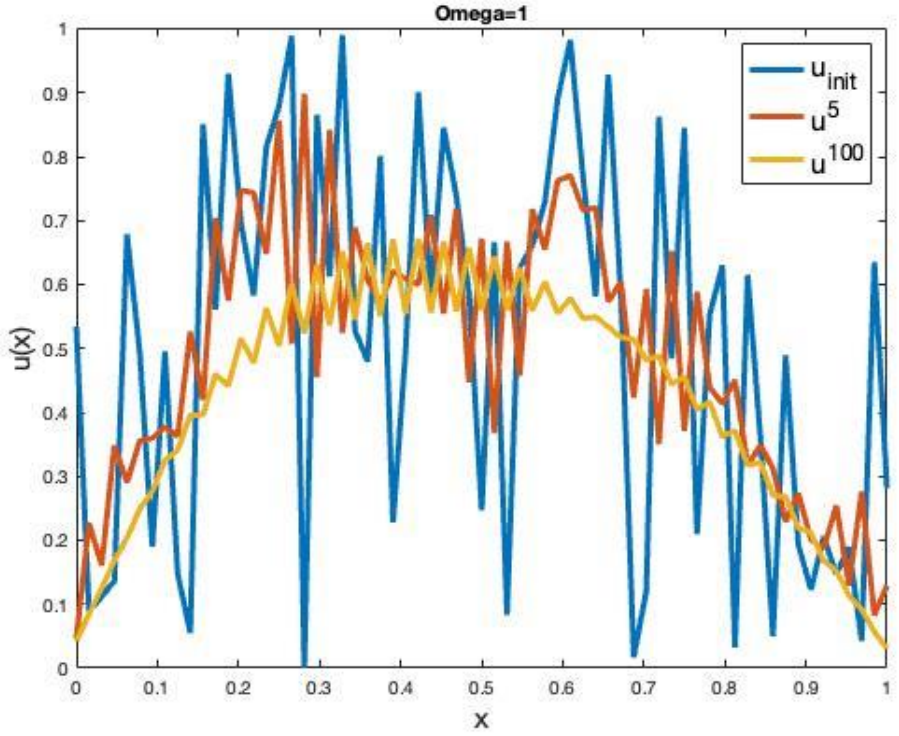
$$1 \leq k < \frac{N}{2}$$

are called low-frequency or smooth modes.

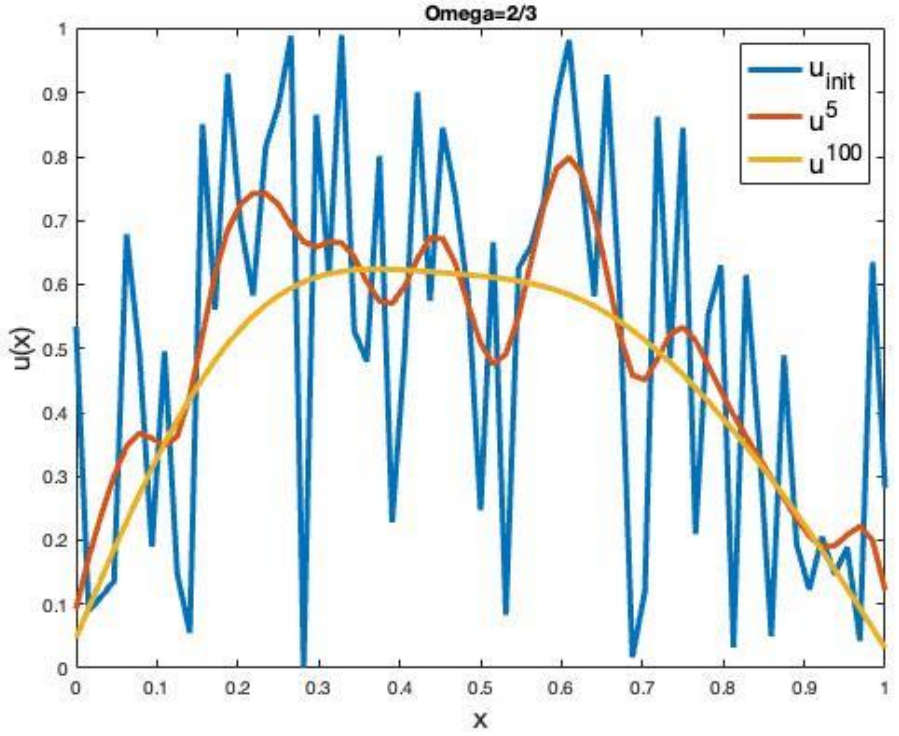
Note that the classification of smooth or oscillatory wave number depends on the total number N of grid points. A fixed wave number *k* might thus be smooth on one, but oscillatory on another grid.

Smoothing with Jacobi and weighted Jacobi

- Let $f = 0$ (thus solution $u = 0$). Then random initial guess → random error



Jacobi $R = I - D^{-1}A$

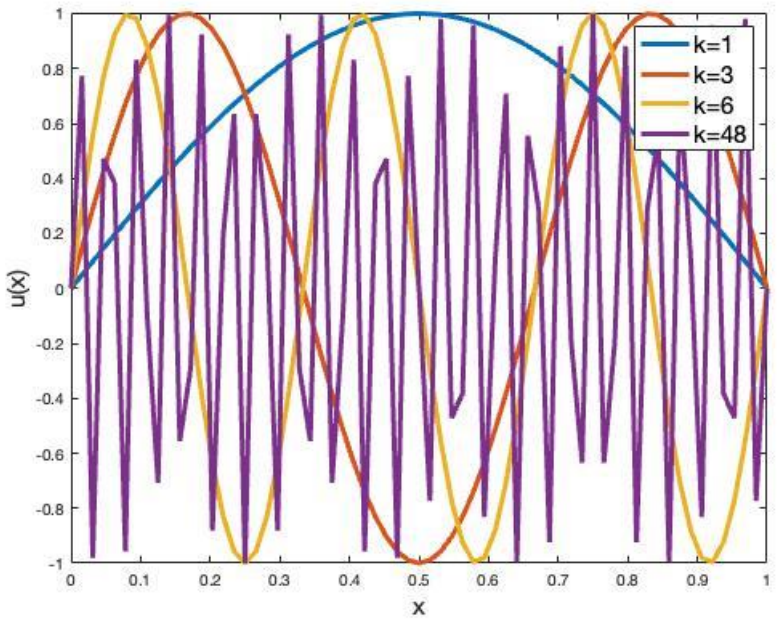


Weighted Jacobi $R_\omega = I - \frac{2}{3}D^{-1}A$

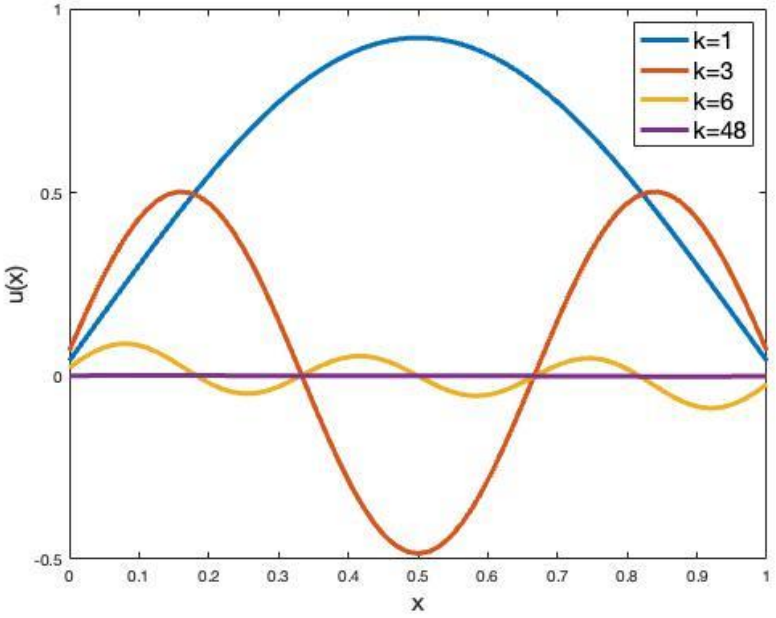
Weighted Jacobi

- Let us use the four modes: k = 1, 3, 6, 48, as initial guess.
- Then smooth modes will dampen less quickly than higher ones.

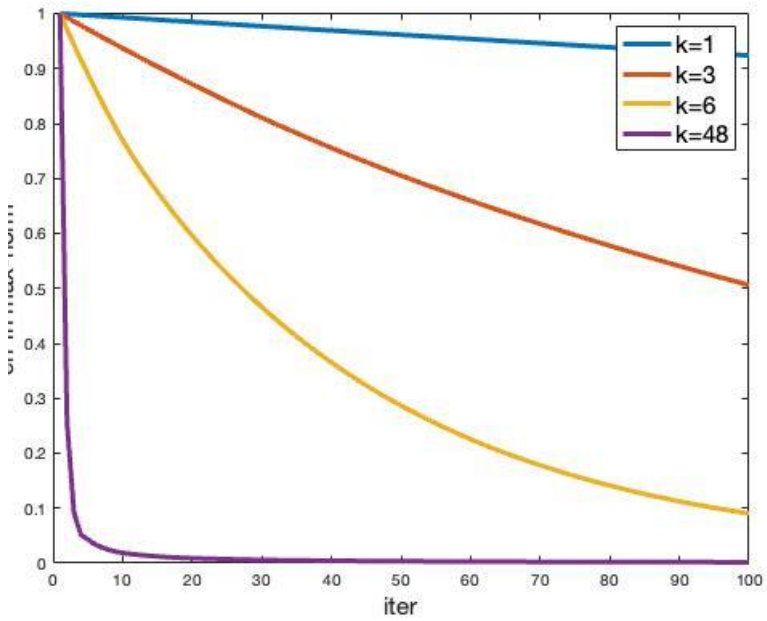
Initial error



Error after 100 iterations



Error decrease



WHY?