MathCore Compact Course on Scientific Computing

Thomas Richter
University of Magdeburg

July 17, 2018

Agenda

1. Numerical complexity of finite element simulations
2. Solving linear systems
3. Multigrid

Summary

- Weak formulation of Laplace

$$u \in H_0^1(\Omega) \quad (\nabla u, \nabla \phi) = (f, \phi) \quad \forall \phi \in H_0^1(\Omega)$$

- Finite element Galerkin discretization

$$V_h = \text{span}\{\phi_h^1, \ldots, \phi_h^N\} \subset \mathbb{R}^N, \quad u_h \in V_h \quad (\nabla u_h, \nabla \phi_h) = (f, \phi_h) \quad \forall \phi_h \in V_h$$

- With $u_h = \sum_i u_i \phi_h^i$ equivalent to

$$Au = b, \quad A_{ij} = (\nabla \phi_h^j, \nabla \phi_h^i), \quad b_i = (f, \phi_h^i)$$

- For linear finite elements it holds

$$\|u - u_h\|_{L^\infty(\Omega)} \leq ch^2 \log(h) \|\nabla^2 u\|_{L^\infty(\Omega)}$$

- We simplify by skipping the logarithm and assuming $c = 1$ and $\|\nabla^2 u\|_\infty = 1$

$$\|u - u_h\|_{L^\infty(\Omega)} \approx h^2$$

- **Goal:** reach tolerance $\epsilon > 0$, e.g. $\epsilon = 0.01$

$$\|u - u_h\|_{L^\infty(\Omega)} \approx h^2 = \epsilon \quad \Leftrightarrow \quad h = \epsilon^{\frac{1}{2}}$$

- Regular mesh: $h = 1/M$ and $N = (M-1)^d$

$$h = \epsilon^{\frac{1}{2}} \quad \Leftrightarrow \quad M = \epsilon^{-\frac{1}{2}} \quad \Leftrightarrow \quad N \approx \epsilon^{-\frac{d}{2}}$$

- The memory for storing matrix and solution grows with smaller tolerances

ϵ	N (2d)	Mem (2d)	N(3d)	Mem (3d)
0.1	10	560 B	30	2.4 kB
0.01	100	5.6 kB	1 000	72 kB
0.001	1 000	56 kB	30 000	2 MB
0.0001	10 000	1/2 MB	1 000 000	72 MB

- Most simple linear solvers are based on Gaussian elimination
- For a **full** $\mathbb{R}^{N\times N}$-matrix the effort scales like $\mathcal{O}\left(N^3\right)$
- If the matrix is a **band-matrix** with band-width B the effort scales like $\mathcal{O}\left(N\cdot B^2\right)$
- Our model-matrix is a band-matrix with band-width

$$B_{2d} = (M-1) \approx N^{\frac{1}{2}}, \quad B_{3d} = (M-1)^2 \approx N^{\frac{2}{3}}$$

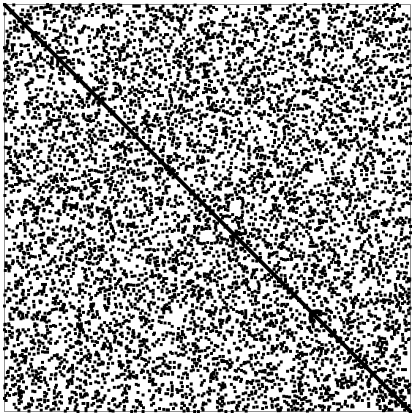
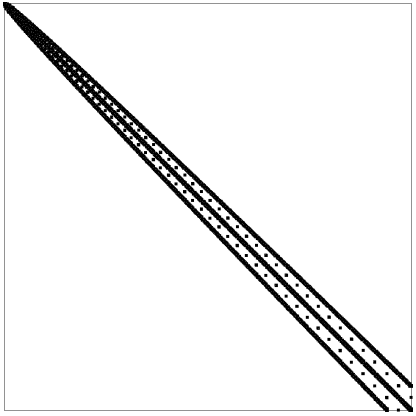
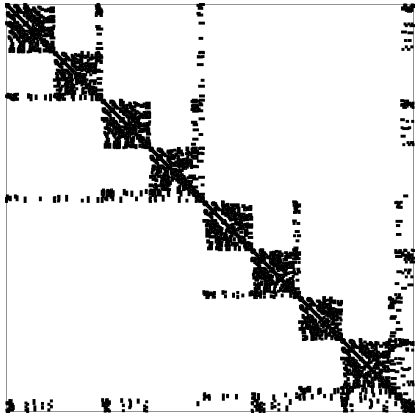
- The solution time T in terms of ϵ is

$$T_{2d} = \epsilon^{-1}, \quad T_{3d} = \epsilon^{-\frac{7}{2}}$$

- Number of operations in 2d and 3d & comparison with full Gaussian

	band-matrix		full matrix
ϵ	2d	3d	3d
0.1	10	$3\cdot 10^3$	$3\cdot 10^4$
0.01	100	10^7	10^9
0.001	1 000	$3\cdot 10^{10}$	$3\cdot 10^{14}$
0.0001	10 000	10^{14}	10^{18}

- The efficiency of the Gaussian elimination applied to a sparse matrix strongly depends on the sorting of the unknowns
- Use `spy` to show the sparsity pattern in matlab
- This is the result for a our model-matrix and three different sorting algorithms: random, CuthillMcKee, Adaptive Multifront (??? that's the algorithm in Matlab)

Agenda

1. Numerical complexity of finite element simulations
2. Solving linear systems
3. Multigrid

Alternatives to a direct solver

- We call the *Gaussian eliminination* a **direct solver**: it is a simple algorithm that we use once and we expect that the output is the solution
- Alternatively we use **iterative solvers**: instead of solving the problem we approximate the solution

$$u^0 \to u^1 \to u^2 \to \cdots$$

Many solvers exist

- Simple fixed-point iterations like *Jacobi*, *Gauss-Seidel*, *SOR*, ...

$$u^{n+1} = u^n + \omega C(b - Au^n)$$

  with the iteration matrix C and the relaxation parameter ω. Example Jacobi-iteration

$$u^{n+1} = u^n + \omega \operatorname{diag}(A)^{-1}(b - Au^n)$$

- *Krylov-Subspace methods* like CG (Conjugate Gradients), GMRES, BiCGStab, MinRes, ... These are excellent and very robust ... but not topic of this class
- With *Conjugate Gradient* we can reduce the operation count for solving to

$$O_{2d/3d} = \mathcal{O}\left(N^{\frac{3}{2}}\right), \quad T_{2d} = \epsilon^{-\frac{3}{4}}, \quad T_{3d} = \epsilon^{-\frac{9}{4}}$$

$$u^{n+1} = u^n + \underline{\underline{C}}(b - A u^n)$$

$C = A^{-1}$?

$$= u^n + A^{-1}(b - A u^n)$$

$$= u^n + A^{-1} b - u^n = A^{-1} b$$

$$= u$$

$C = \mathrm{diag}(A)^{-1}$ / $A = L + D + R$

$C = (L + D)^{-1}$

Given an initial guess $u^0 \in \mathbb{R}^N$ (usually $u^0 = 0$) iterate

$$u^{n+1} = u^n + \omega \operatorname{diag}(A)^{-1}(b - Au^n)$$

Lemma 1 (Error propagation) *For the error $e^n = u^n - u$ it holds*

$$e^{n+1} = [I - \omega D^{-1} A]e^n$$

where $D = \operatorname{diag}(A)$ is the diagonal of A.

What does "solving a linear system" mean?

- We must cancel the iteration after some steps
- Usually we prescribe a relative tolerance

$$Au^n \approx b \quad \Leftrightarrow \quad \|b - Au^n\|_\infty \leq tol \|b\|_\infty$$

- Here $tol = 10^{-6}$

$$\mathbf{A} = \begin{pmatrix} \mathbf{B} & -I & 0 & \cdots & 0 \\ -I & \mathbf{B} & -I & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & -I & \mathbf{B} & -I \\ 0 & \cdots & 0 & -I & \mathbf{B} \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} 4 & -1 & 0 & \cdots & 0 \\ -1 & 4 & -1 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & -1 & 4 & -1 \\ 0 & \cdots & 0 & -1 & 4 \end{pmatrix}.$$

Lemma 2 *The model matrix in 2d has the eigenvalues*

$$\lambda_i = 4 - 2\left(\cos(kh\pi) + \cos(lh\pi)\right), \quad i = (M-1)l + k$$

and in 3d

$$\lambda_i = 6 - 2\left(\cos(kh\pi) + \cos(lh\pi) + \cos(mh\pi)\right), \quad i = (M-1)^2 m + (M-1)l + k$$

$(\omega_i)_j = \sin(k m h \pi) \sin(l n h \pi)$

$j = (M-1)\,n + m$

The smallest and largest eigenvalues are bounded by

$$\lambda_{min}^{2d/3d} \approx h^2, \quad \lambda_{max}^{2d} \approx 8 - h^2, \quad \lambda_{max}^{3d} \approx 12 - h^2$$

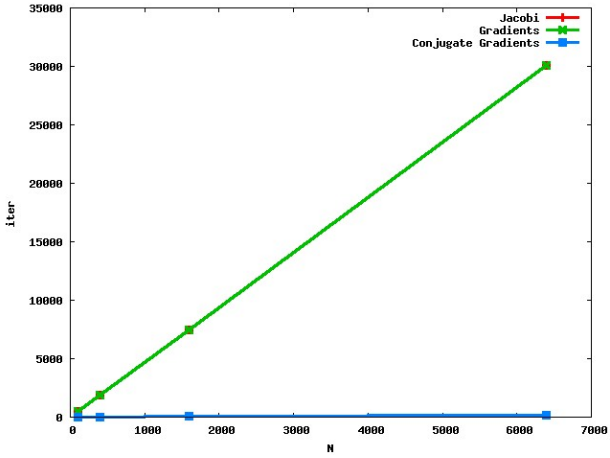
Lemma 3 (Error estimate) *Let $\omega = 1$. For the error $e^n = u^n - u$ it holds*

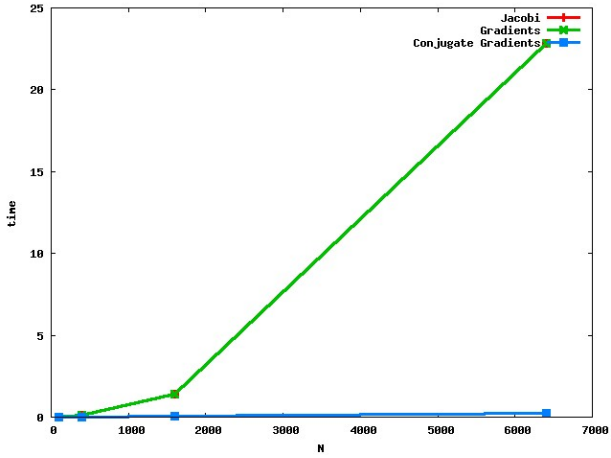
$$\|e^{n+1}\| \leq \left(1 - ch^2\right) \|e^n\|$$

$$e^n = \sum_{i=1}^{N} e_i^n w_i \qquad e_i^{n+1} = \left(1 - \frac{1}{4}\lambda_i\right) e_i^n$$

$$|e_i^{n+1}| \leq \left|1 - \frac{\lambda_i}{4}\right| |e_i^n|$$

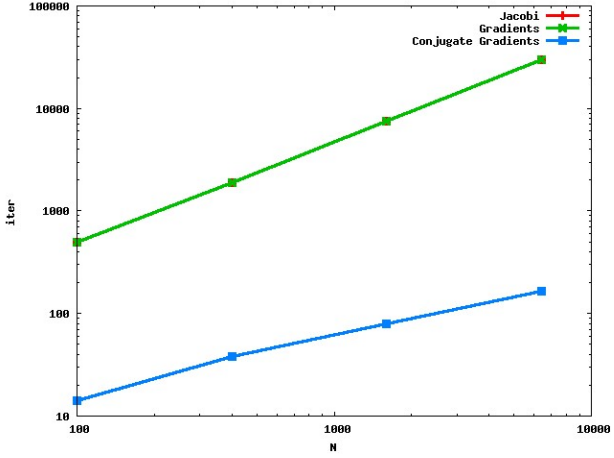
Jacobi			Gradient-Method			Conjugate Gradients		
N	iter	time(s)	N	iter	time(s)	N	iter	time(s)
100	499	0.001	100	499	0.02	100	14	0.001
400	1893	0.12	400	1893	0.11	400	38	0.01
1600	7460	1.42	1600	7461	1.43	1600	80	0.03
6400	30047	22.82	6400	30047	22.85	6400	164	0.21

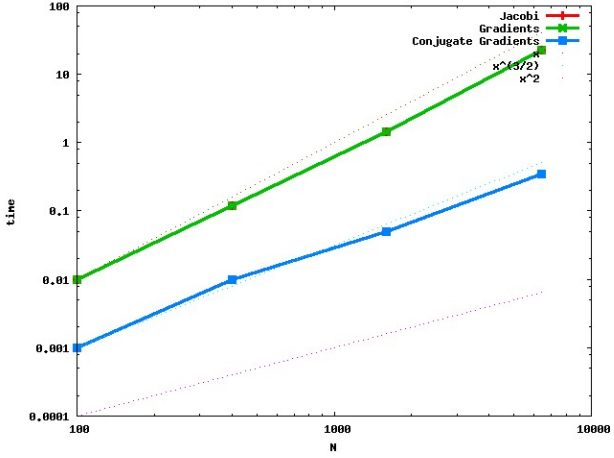




The same, just logarithmic

Jacobi			Gradient-Method			Conjugate Gradients		
N	iter	time(s)	N	iter	time(s)	N	iter	time(s)
100	499	0.001	100	499	0.02	100	14	0.001
400	1893	0.12	400	1893	0.11	400	38	0.01
1600	7460	1.42	1600	7461	1.43	1600	80	0.03
6400	30047	22.82	6400	30047	22.85	6400	164	0.21





Summary

- Jacobi is very simple
- Every iteration requires only one matrix-vector product

$$T_{2d} = 5N, \quad T_{3d} = 7N$$

- But, the number of iterations increases with the problem size as the convergence rate gets worse and worse

$$\rho = 1 - h^2 \quad \Rightarrow \quad \rho_{2d} = 1 - \frac{1}{N}, \quad \rho_{3d} = 1 - \frac{1}{N^{\frac{2}{3}}}$$

Agenda

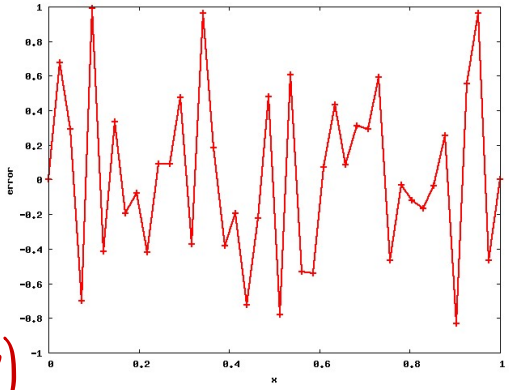
1. Numerical complexity of finite element simulations
2. Solving linear systems
3. Multigrid

Numerical complexity

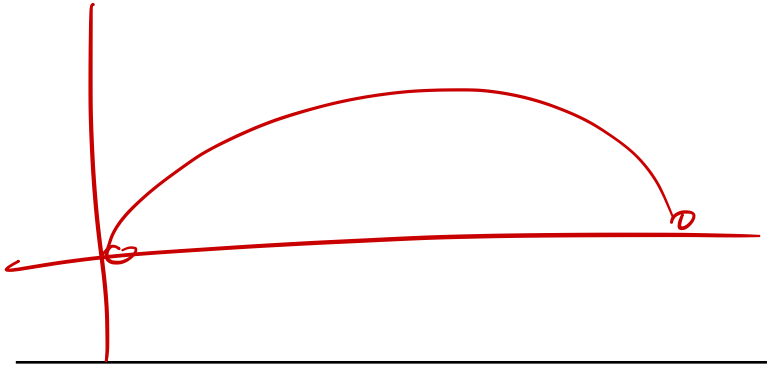
- There is no better method than
$$\mathcal{O}(N)$$
- General idea: cost = cost per iteration times number of iterations
- We have two options:
  - Few (costly) iterations but very good convergence = Gaussian elimination
  - Many (cheap) iterations but slow convergence = Jacobi

Simple test

- Solve 1d-Laplace
- Initial guess u^0 is random
- Perform some few steps

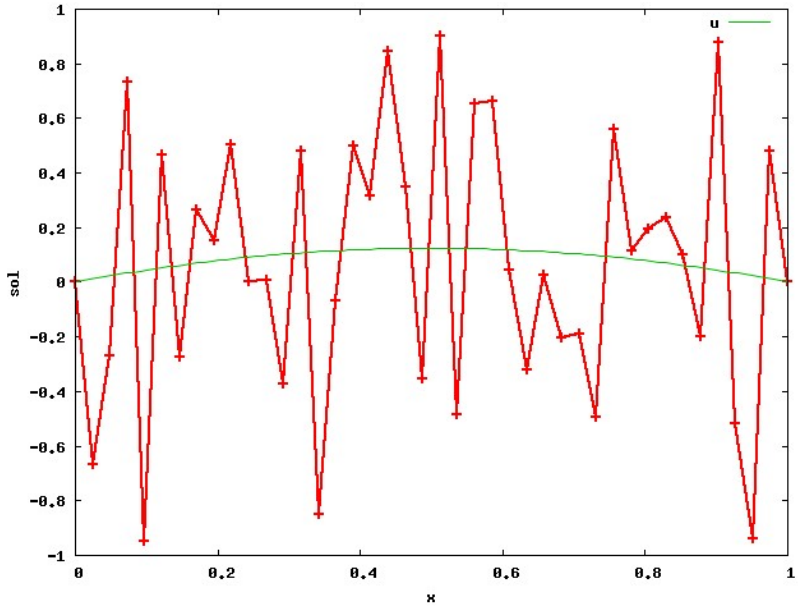


$-\partial_{xx} u = 1 \qquad u(0) = u(1) = 0$

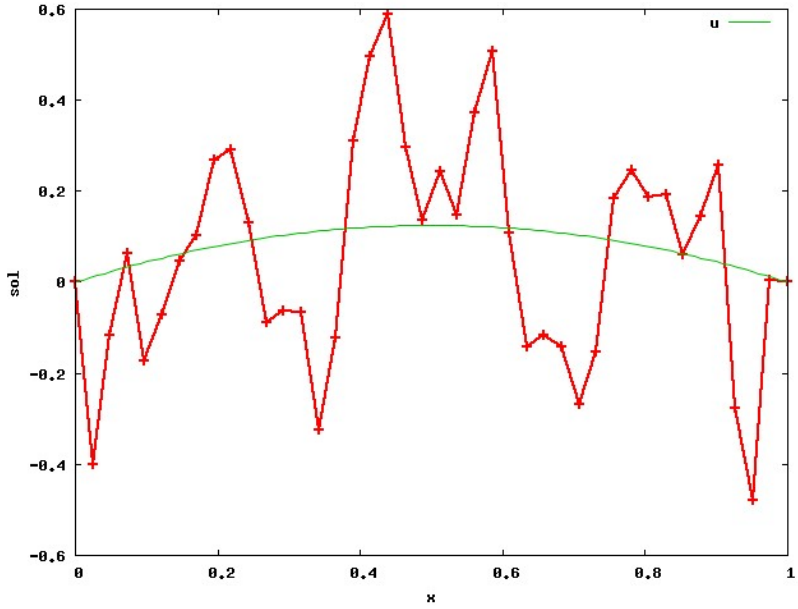


$Au = b \qquad u^0 = \begin{pmatrix} \cdot \\ \cdot \\ \cdot \end{pmatrix}$

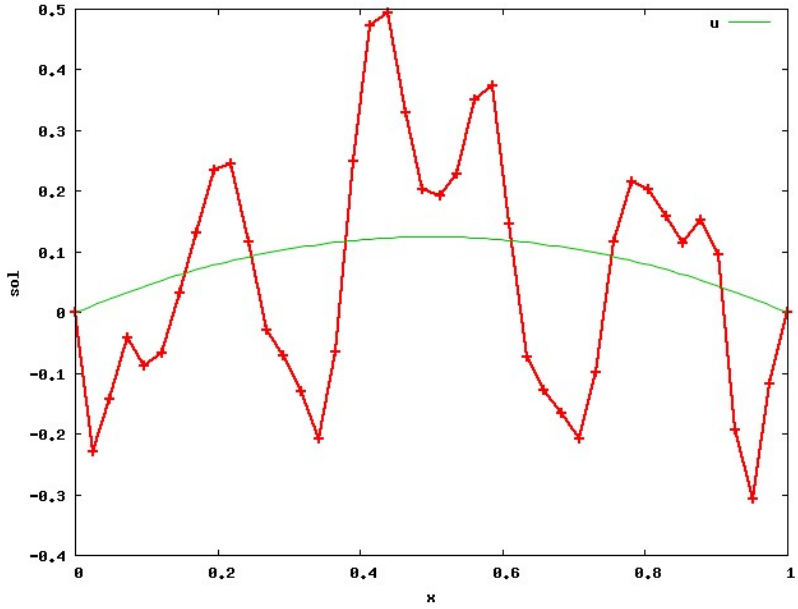
Step 0

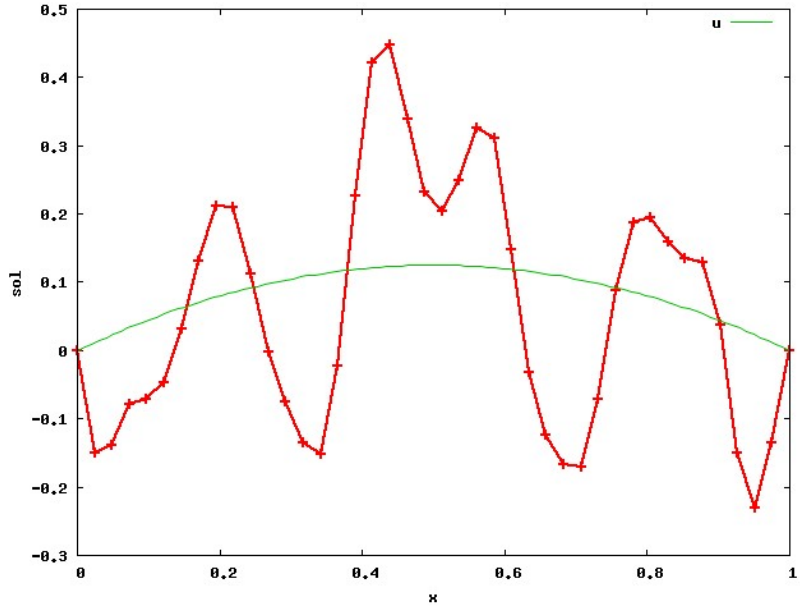


Step 1

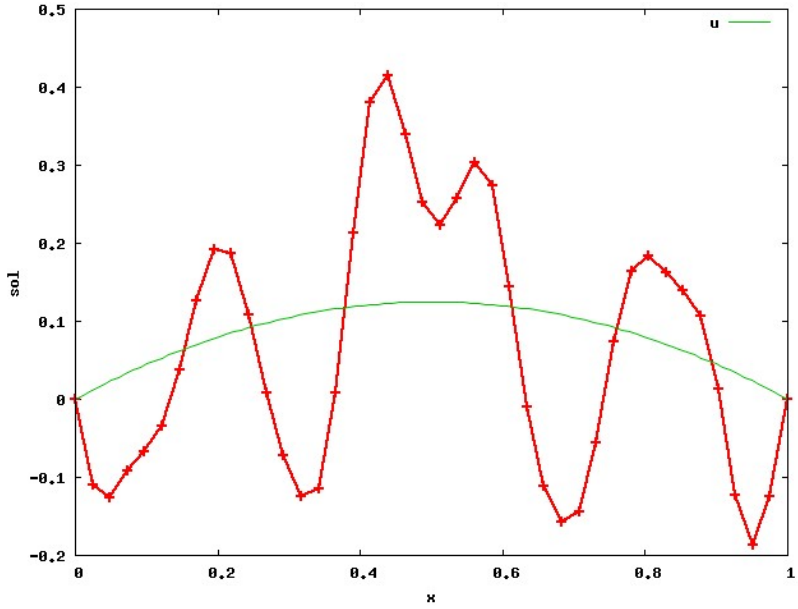


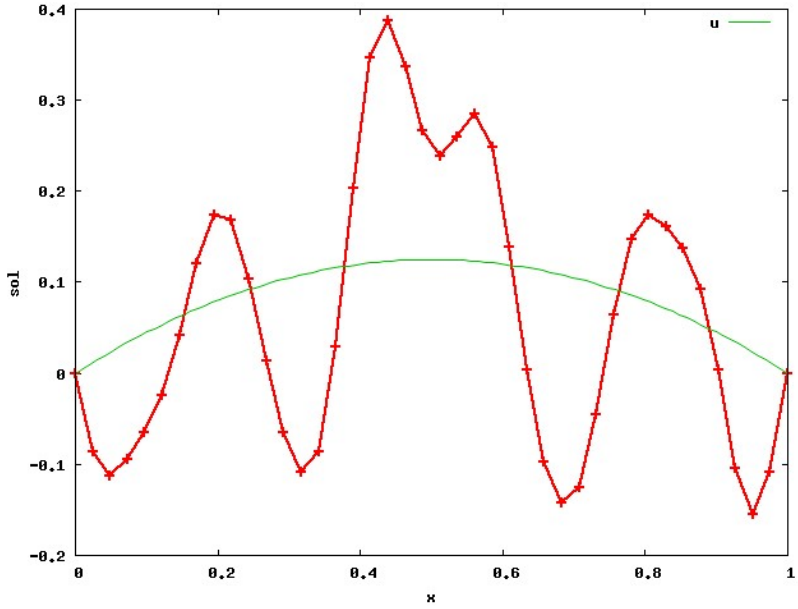
Step 2



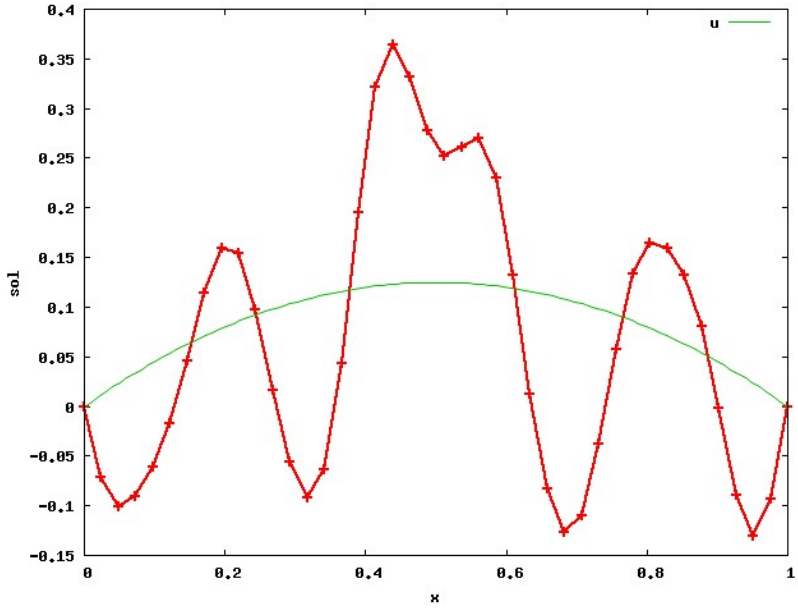

u
sol
x
0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
0
0.2
0.4
0.6
0.8
1

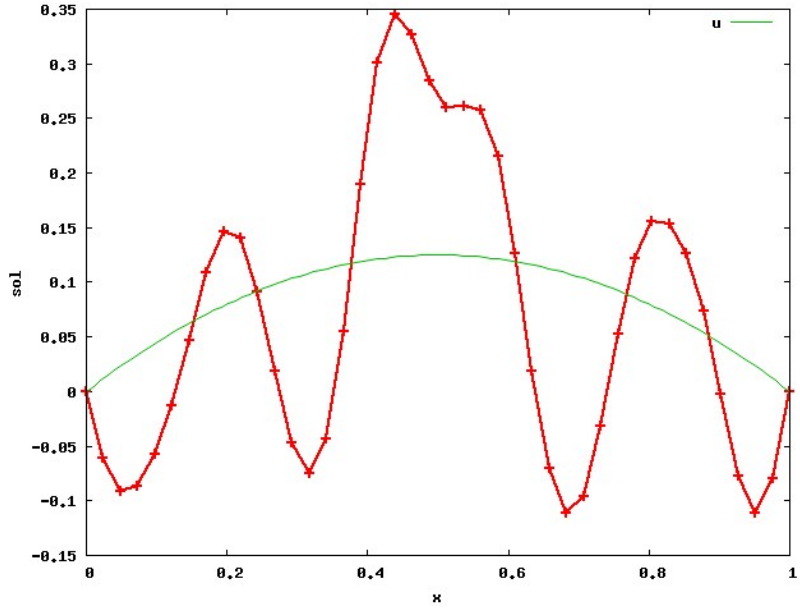
Step 4



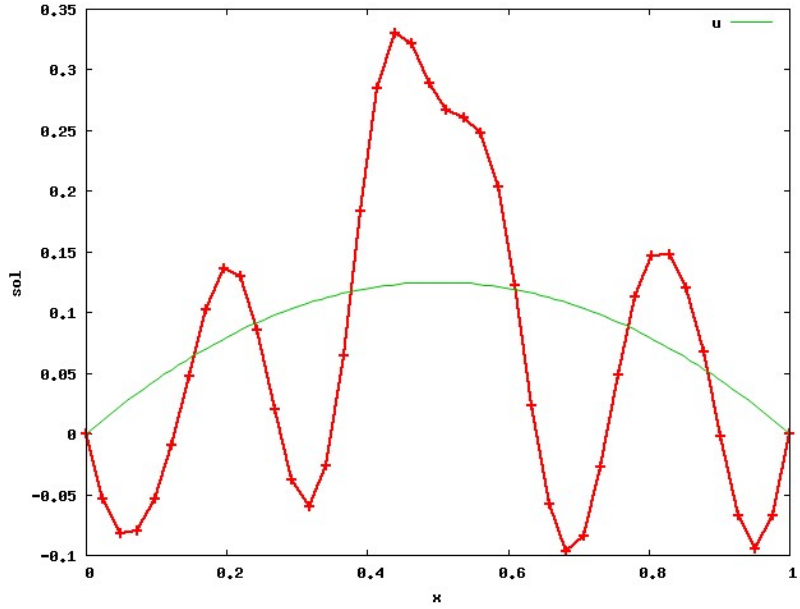

u
sol
x
0.4
0.3
0.2
0.1
0
-0.1
-0.2
0
0.2
0.4
0.6
0.8
1

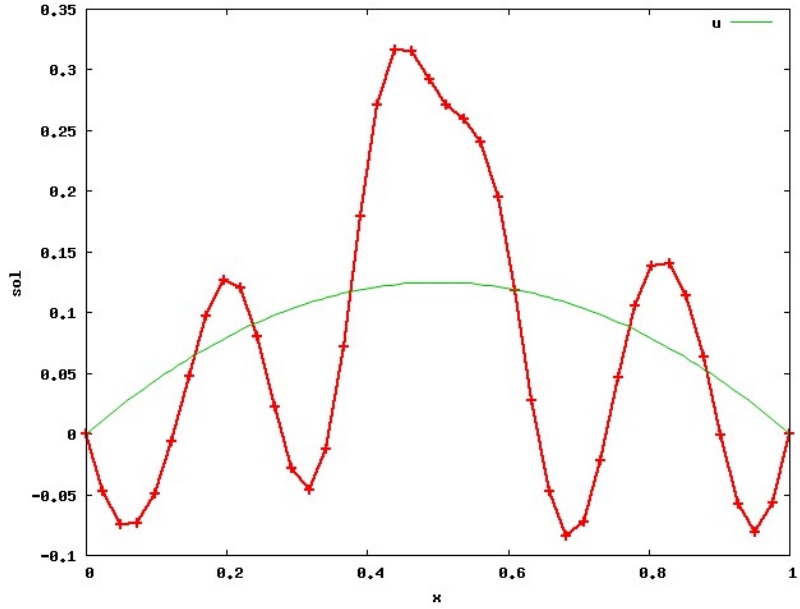
Step 6



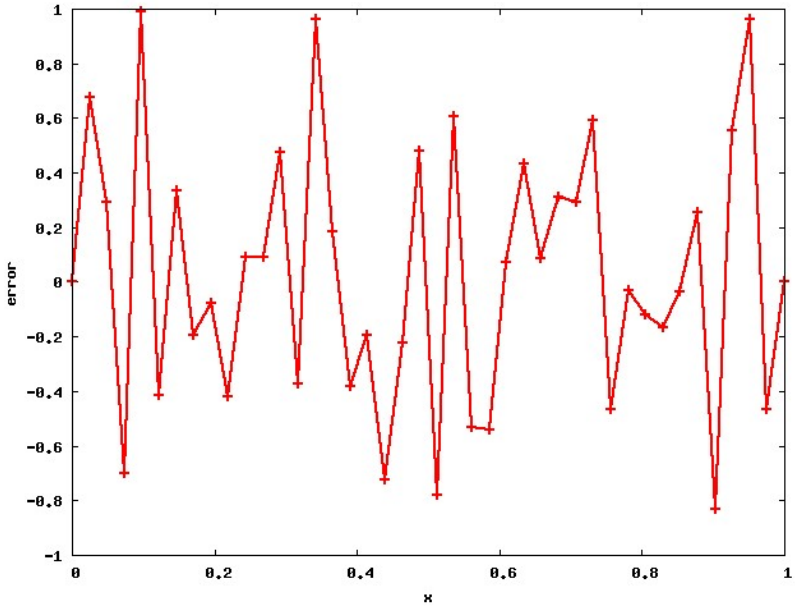

u
sol
x
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
0
0.2
0.4
0.6
0.8
1

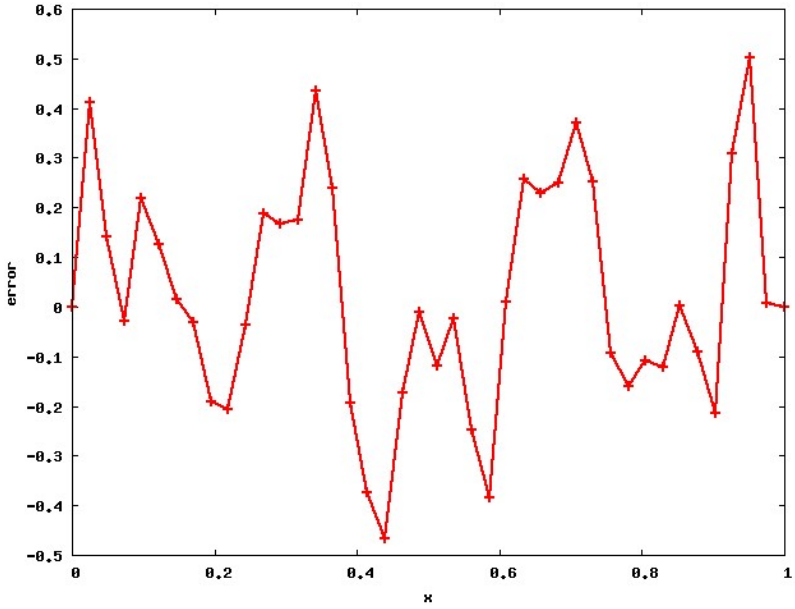
Step 8

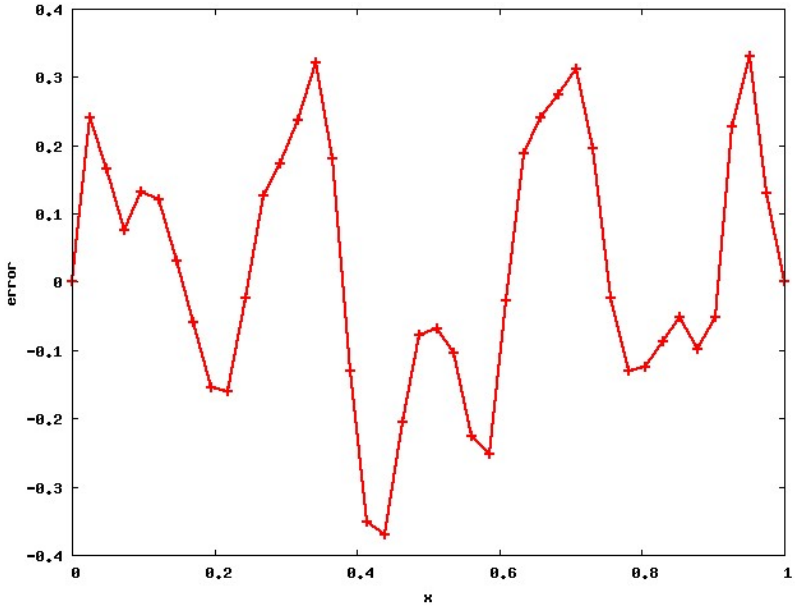



u
sol
x
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
-0.05
-0.1
0
0.2
0.4
0.6
0.8
1

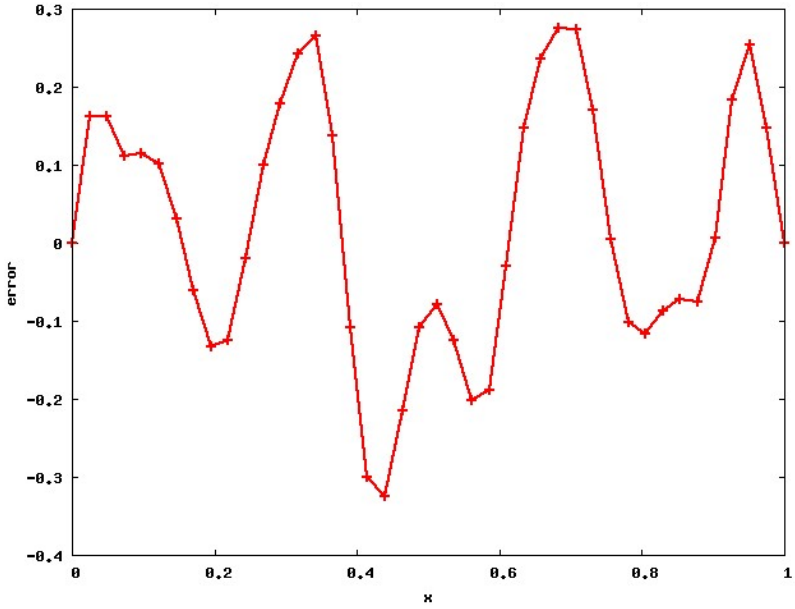
Same for the error - Step 0

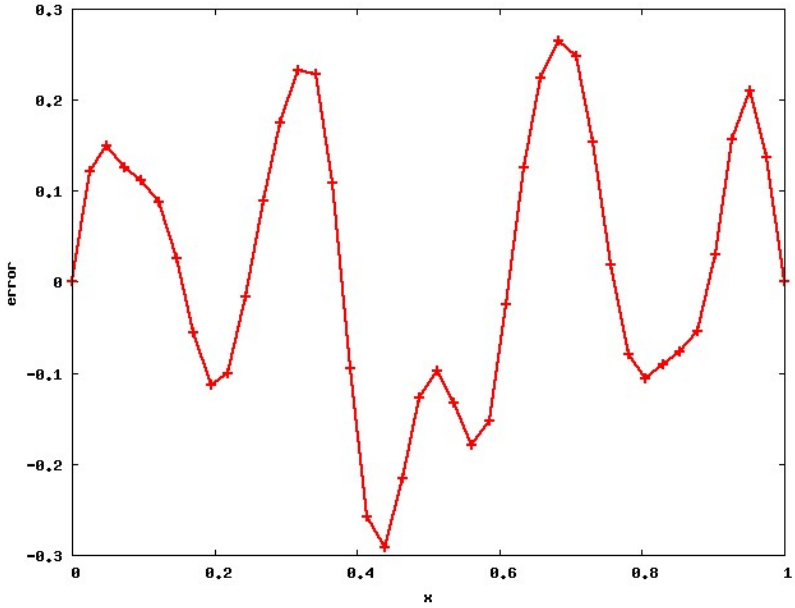


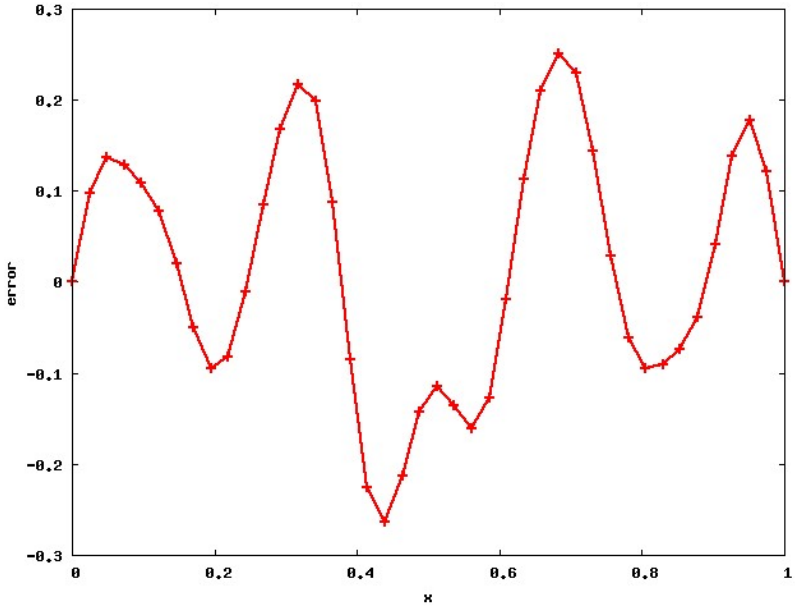

error
x
0.6
0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
0
0.2
0.4
0.6
0.8
1

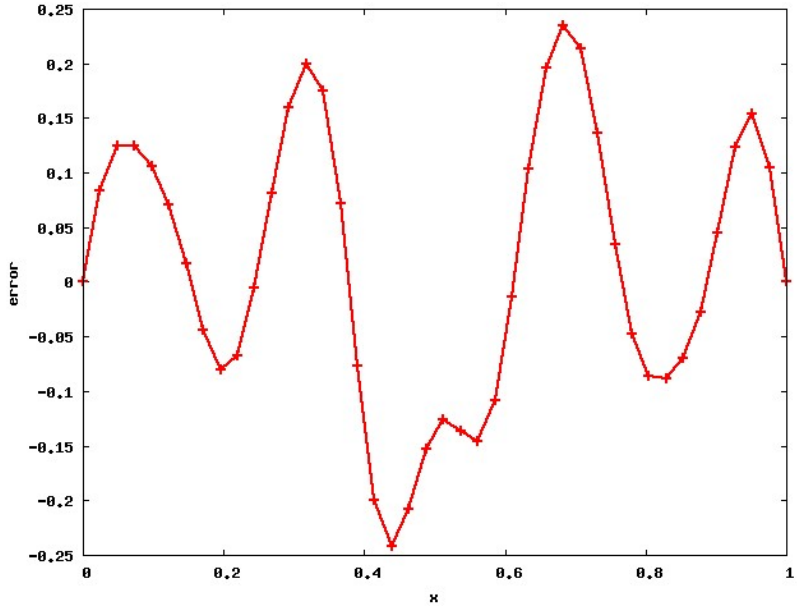

0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
error
0
0.2
0.4
0.6
0.8
1
x

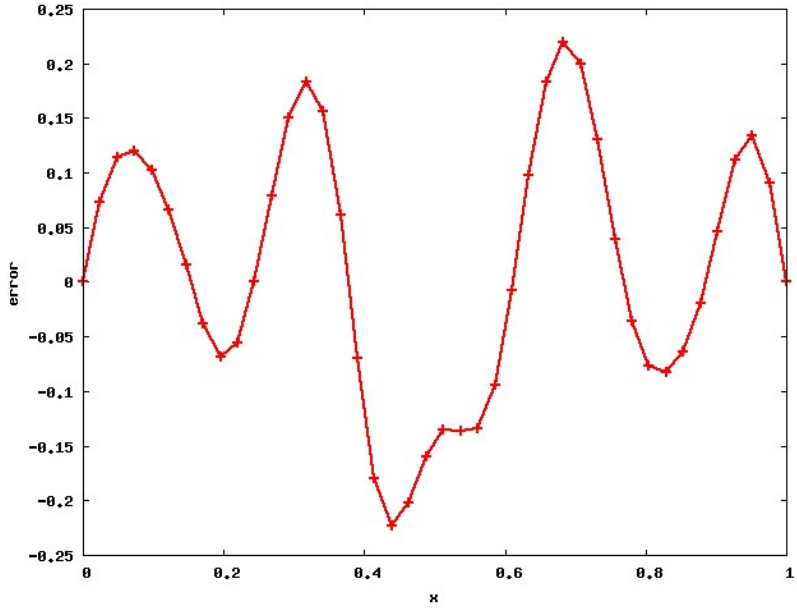
Step 3

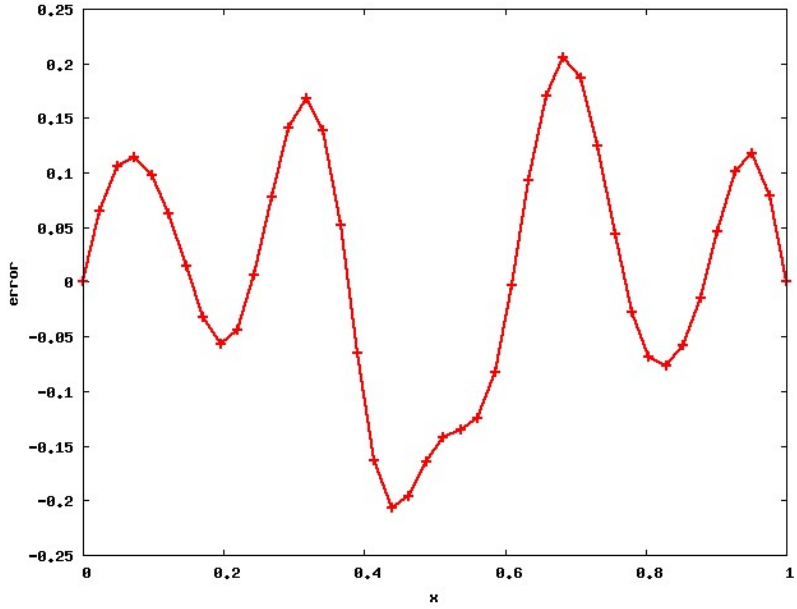


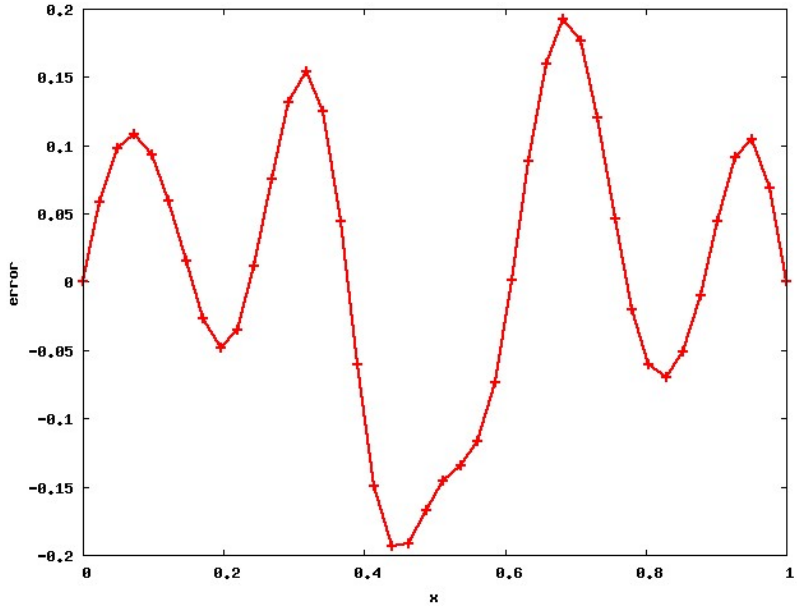

0.3
0.2
0.1
0
-0.1
-0.2
-0.3
error
0
0.2
0.4
0.6
0.8
1
x


0.3
0.2
0.1
0
-0.1
-0.2
-0.3
error
0
0.2
0.4
0.6
0.8
1
x

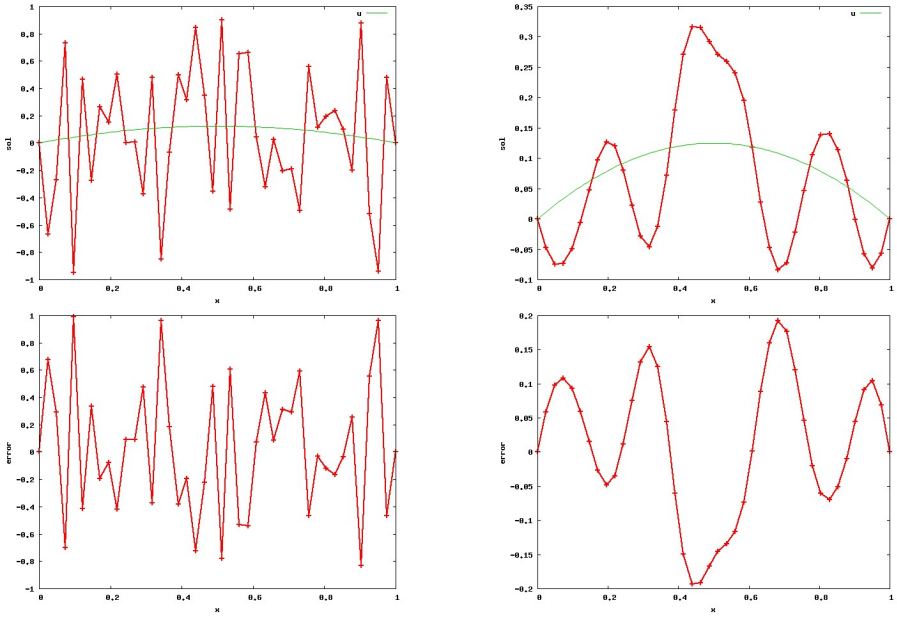

error
x
0.25
0.2
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
-0.2
-0.25
0
0.2
0.4
0.6
0.8
1


0.25
0.2
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
-0.2
-0.25
error
0
0.2
0.4
0.6
0.8
1
x


error
x
0.25
0.2
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
-0.2
-0.25
0
0.2
0.4
0.6
0.8
1


0.2
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
-0.2
error
0
0.2
0.4
0.6
0.8
1
x

What happens in step 0 to step 9

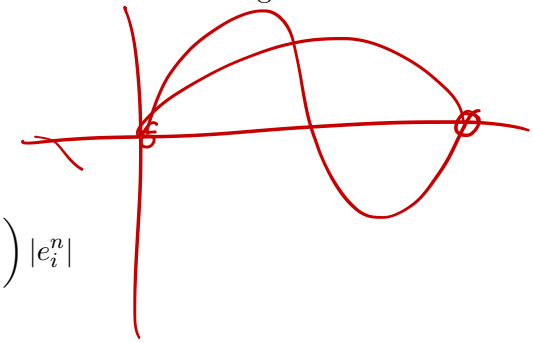


- Error decreases only slightly. But, approximation gets smooth

Jacobi is a bad solver but a good "smoother"

- The error analysis for Jacobi shows different convergence rates for different eigenvector contributions

$$e^n = \sum_{i=1}^{N} e_i^n \omega_i$$



The convergence of component e_i^n is given by

$$|e_i^{n+1}| \leq \left(1 - \omega \frac{\lambda_i}{A_{ii}}\right) |e_i^n|$$

- With the eigenvalues (2d)

$$\lambda_i = 4 - 2\cos(kh\pi) - 2\cos(lh\pi), \quad i = (M-1)l + k$$

and the eigenvectors

$$(\omega_i)_j = \sin(kmh\pi)\sin(lnh\pi), \quad i = (M-1)l + k, \quad j = (M-1)n + m$$

- High frequencies belong to large values of k and l (large values of i).

$$(\omega_i)_j = \sin\left(\frac{1}{M}\pi \cdot j\right)$$

Eigenvalues

$$\lambda_i = 4 - 2\cos(kh\pi) - 2\cos(lh\pi), \quad i = (M-1)l + k$$

and eigenvectors

$$(\omega_i)_j = \sin(kmh\pi)\sin(lnh\pi), \quad i = (M-1)l + k, \quad j = (M-1)n + m$$

- All frequencies i with $k > M/2$ and $l > M/2$ are called *high frequencies*
- It holds

$$k, l \leq \frac{M}{2}: \quad \lambda_{i=(M-1)l+k} \in (h^2, 4)$$
$$k, l \geq \frac{M}{2}: \quad \lambda_{i=(M-1)l+k} \in (4, 8 - h^2)$$

- For the reduction rate

$$|e_i^{n+1}| \leq \underbrace{\left(1 - \omega \frac{\lambda_i}{A_{ii}}\right)}_{=:\rho_i} |e_i^n|$$

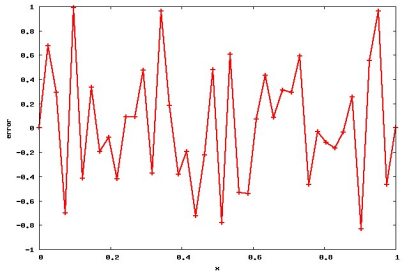
this is

$$\begin{aligned} k, l \leq \frac{M}{2}: &\quad \rho_i \in (1-\omega, 1-\omega h^2) \\ k, l \geq \frac{M}{2}: &\quad \rho_i \in (1-2\omega, 1-\omega) \end{aligned} \Rightarrow (\omega = 0.5) \Rightarrow \boxed{\begin{aligned} k, l \leq \frac{M}{2}: &\quad \rho_i \in (\frac{1}{2}, 1-h^2) \\ k, l \geq \frac{M}{2}: &\quad \rho_i \in (0, \frac{1}{2}) \end{aligned}}$$

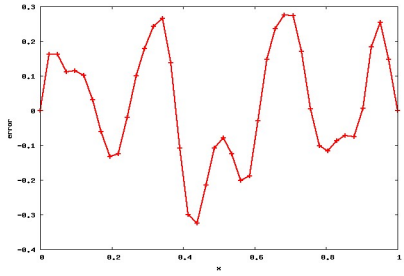
- All high frequencies are “smoothed” with a convergence rate that does not depend on the mesh size, the problem size or something else:

$$k, l \geq \frac{M}{2}: \quad \rho_i \in (0, \frac{1}{2}) \quad \Rightarrow \quad \rho_i \leq \frac{1}{2}$$

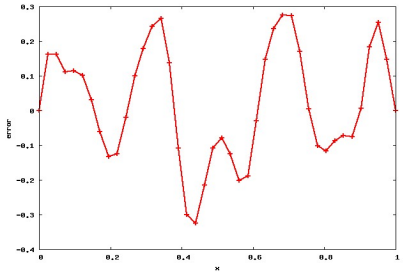
- *Smooth high frequencies* on fine mesh Ω_h (some few steps):



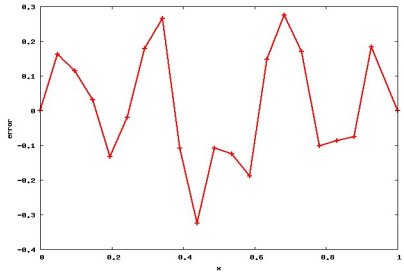
→



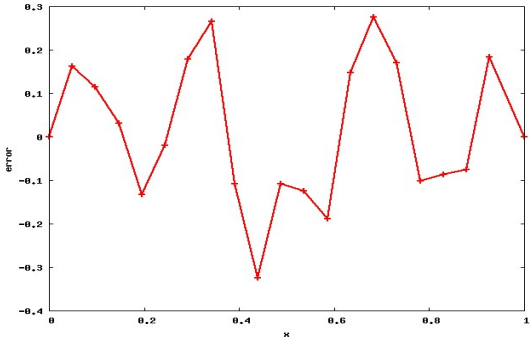
- Transfer to coarse Mesh Ω_{2h} with $M_{coarse} = M_{fine}/2$



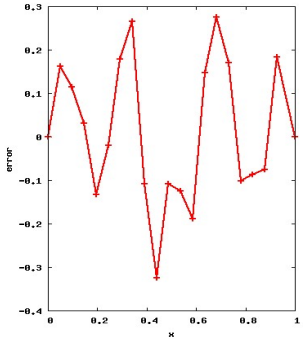
→



- *Low frequencies* on fine mesh Ω_h are *high frequencies* on coarse mesh Ω_{2h}



$\rightarrow$



- Only few (fixed number $C(1?)$) operations on fine mesh Ω_h
- Coarse mesh problem Ω_{2h} is smaller (reduction by 4 in 2d and by 8 in 3d). Perhaps direct solver possible?

2-Grid Method for solving $Au = b$ Initial guess $u^{(0)}$, iterate $t \geq 0$

1. Smooth: $y_h^{(t)} := \mathcal{S}(A_h, b_h, u_h^{(t)})$
2. Defect: $d_h^{(t)} := b_h - A_h y_h^{(t)}$
3. Restrict to coarse mesh: $d_H^{(t)} := \mathcal{R}_h d_h^{(t)}$
4. Coarse Mesh Solution: $u_H^{(t)} := A_H^{-1} d_H^{(t)}$
5. Prolongate to fine mesh: $u_h^{(t+1)} := u_h^{(t)} + \mathcal{P}_h u_H^{(t)}$

Theorem 1 (Two-Grid Iteration) *Let A be the model matrix. There is a number $\nu_0 > 0$ that does not depend on the mesh size h, such that the Two-Grid Iteration converges, if $\nu \geq \nu_0$ smoothing steps are considered. For the convergence rate it holds*

$$\rho_{TG} \leq \rho_0 < 1$$

on all meshes. Hence, to reduce the error by a certain fraction ϵ

$$\rho^t = \epsilon \quad \Rightarrow \quad t = O(1)$$

steps are required.

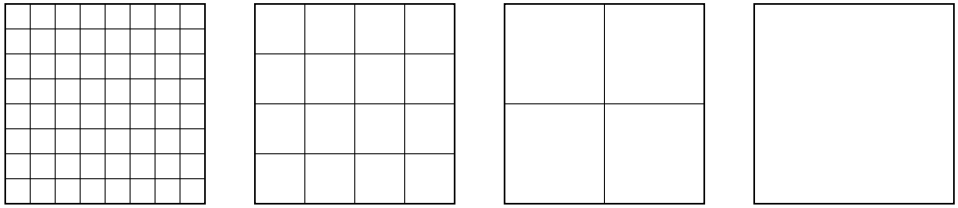
- The idea is good, but the effort is only reduced by a small factor:

$$T_{TG}(N) = \mathcal{O}(N) + T_{direct}(2^{-d}N)$$

- If the direct solver is used we still get a quadratic scaling, only the constant is better

- Similar to Two-Grid. But we replace the coarse mesh solution by the multigrid algorithm itself
- Hierarchy of Meshes

$$\Omega_h = \Omega_L \to \Omega_{L-1} \to \cdots \to \Omega_l \to \cdots \to \Omega_0 = \Omega_H$$



- Hierarchy of Problems

$$A_l u_l = b_l, \quad l = 0, \ldots, L, \; u_l \in \mathbb{R}^{N_l}$$

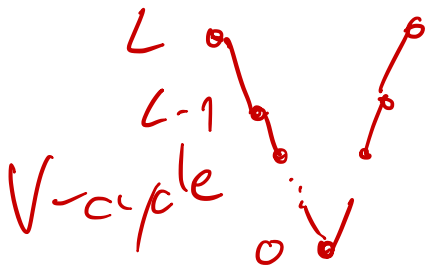
Multigrid-Method for solving $Au = b$ on Ω_h
Highest mesh level L. Initial guess $u_L^{(0)}$, iterate $t \geq 0$

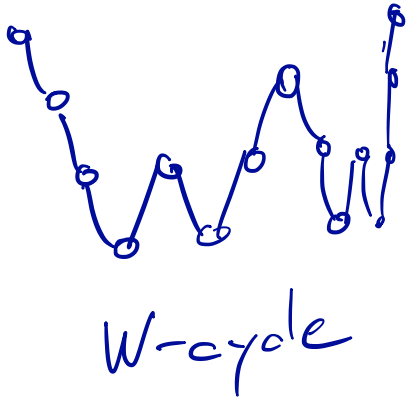
$$u_L^{(t)} = \mathrm{MG}(L, u_L^{(t-1)})$$

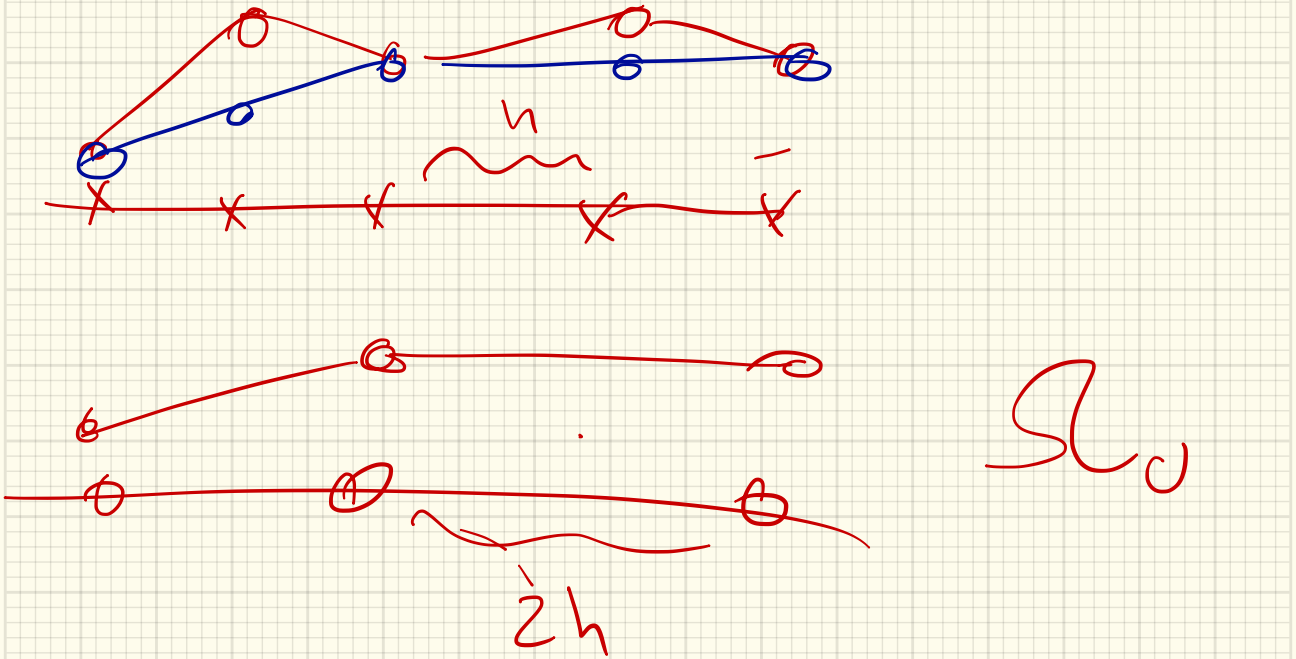
$$u^{n+1} = u^n + \omega \left(\tilde{L}\tilde{U}\right)^{-1}(b - Au^n)$$

MG(l, u_l, b_l)**:**
`if` $l = 0$
    `solve direct` $u_0 = A_0^{-1} b_0$
`else`

1. `smooth` $y_l^1 := \mathcal{S}^{\nu_1}(A_l, b_l, u_l)$
2. `residual` $d_l := b_l - A_h y_l$
3. `restrict` $d_{l-1} := \mathcal{R}_l d_l$
4. `coarsemesh solution with multigrid` $u_{l-1} = \mathrm{MG}(l-1, 0, d_{l-1})$
5. `prolongate` $z_l := u_l + \mathcal{P}_h u_{l-1}$
6. `smooth` $y_l^2 := \mathcal{S}^{\nu_2}(A_l, b_l, z_l)$
7. `return` y_l^2






h
2h

$$\Omega_n = \Omega_L > \Omega_{L-1} > \dots > \Omega_0$$

$$V_n = V_L \supset V_{L-1} \supset \dots \supset V_0$$

$$(\Omega_k u_k, \varphi_k) = (f, \varphi_k) \quad \forall \varphi_k \in V_k$$

$u_k \in V_k \qquad R_{k-1} u_k \in V_{k-1}$

$$(R_{k-1} u_k, \varphi_{k-1}) = (u_k, \varphi_{k-1}) \quad \forall \varphi_{k-1} \in V_{k-1}$$

Theorem 2 *Let, for a $\nu > \nu_0$, the Two-Grid iteration be uniformly convergent. Then, the Multigrid-Iteration is also uniformly convergent with a convergence rate*

$$\rho_{MG} < 1.$$

$$\rho_{MG} \leq \rho_0 < 1$$

Theorem 3 *The numerical effort for reducing the error by a given fraction ϵ scales like*

$$T = \mathcal{O}(N)$$

$$\rho_{mg}^t \approx \epsilon \Rightarrow t = \frac{\log(\epsilon)}{\log(\rho_{mg})}$$

$$E(\ell) = N_\ell + N_\ell + N_\ell + N_\ell + N_\ell$$

(smooth) (residual restr. prd. smooth)

$$+ E(\ell-1)$$

$$E(0) = c$$

$$E(L) = N_L + E(L-1)$$

$$= N_L + N_{L-1} + \dots + N_0 + c$$

$$= \sum_{\ell=0}^{L} N_L \left(\frac{1}{2^d}\right)^\ell + c \leq c + N \frac{1}{1-\left(\frac{1}{2^d}\right)}$$

thanks!