Numerical Integration of Partial Differential Equations (PDEs)

- Introduction to PDEs.
- Semi-analytic methods to solve PDEs.
- Introduction to Finite Differences.
- Stationary Problems, Elliptic PDEs.
- Time dependent Problems.
- Complex Problems in Solar System Research.

Stationary Problems, Elliptic PDEs.

- Example: 2D-Poisson equation.
- From differential equations to difference equations and algebraic equations.
- Relaxation methods:
  -Jacobi and Gauss-Seidel method.
  -Successive over-relaxation.
  -Multigrid solvers.
- Finite Elements.

Maxwell Equations:

$$\begin{aligned}
\nabla \times \mathbf{B} &= \mu_0 \mathbf{j} + \cancel{\epsilon_0 \mu_0 \frac{\partial \mathbf{E}}{\partial t}} \\
\nabla \times \mathbf{E} &= -\frac{\partial \mathbf{B}}{\partial t} \\
\nabla \cdot \mathbf{B} &= 0 \\
\nabla \cdot \mathbf{E} &= \frac{1}{\epsilon_0} \rho
\end{aligned}$$

For slowly varying temporal evolution we neglect the displacement current (popular in MHD) and use the electromagnetic potentials:

$$\begin{aligned}
\mathbf{B} &= \nabla \times \mathbf{A} \\
\mathbf{E} &= -\nabla \mathbf{\Phi} - \frac{\partial \mathbf{A}}{\partial t}
\end{aligned}$$

together with the Coulomb Gauge condition:

$$\nabla \cdot \mathbf{A} = 0$$

With these definitions we get:

$$\begin{aligned}
\nabla \times \nabla \times \mathbf{A} &= \mu_0 \mathbf{j} \\
\nabla \times (-\nabla \mathbf{\Phi} - \frac{\partial \mathbf{A}}{\partial t}) &= -\frac{\partial \nabla \times \mathbf{A}}{\partial t} \checkmark \\
\nabla \cdot \nabla \times \mathbf{A} &= 0 \checkmark \\
\nabla \cdot (-\nabla \mathbf{\Phi} - \frac{\partial \mathbf{A}}{\partial t}) &= \frac{1}{\epsilon_0}\rho
\end{aligned}$$

We use the vector identity $\nabla \times \nabla \times \mathbf{A} = \nabla(\nabla \cdot \mathbf{A}) - \Delta \mathbf{A}$

$$\begin{aligned}
\nabla(\nabla \cdot \mathbf{A}) - \Delta \mathbf{A} &= \mu_0 \mathbf{j} \\
-\Delta \mathbf{\Phi} - \frac{\partial(\nabla \cdot \mathbf{A})}{\partial t} &= \frac{1}{\epsilon_0}\rho
\end{aligned}$$

Finally we use the Coulomb Gauge $\nabla \cdot \mathbf{A} = 0$ and derive Poisson equations:

$$\boxed{\begin{aligned}
-\Delta \mathbf{A} &= \mu_0 \mathbf{j} \\
-\Delta \mathbf{\Phi} &= \frac{1}{\epsilon_0}\rho
\end{aligned}}$$

Boundary value problems for elliptic PDEs: Example: Poisson Equation in 2D

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = \rho(x, y)$$

We define short notation:

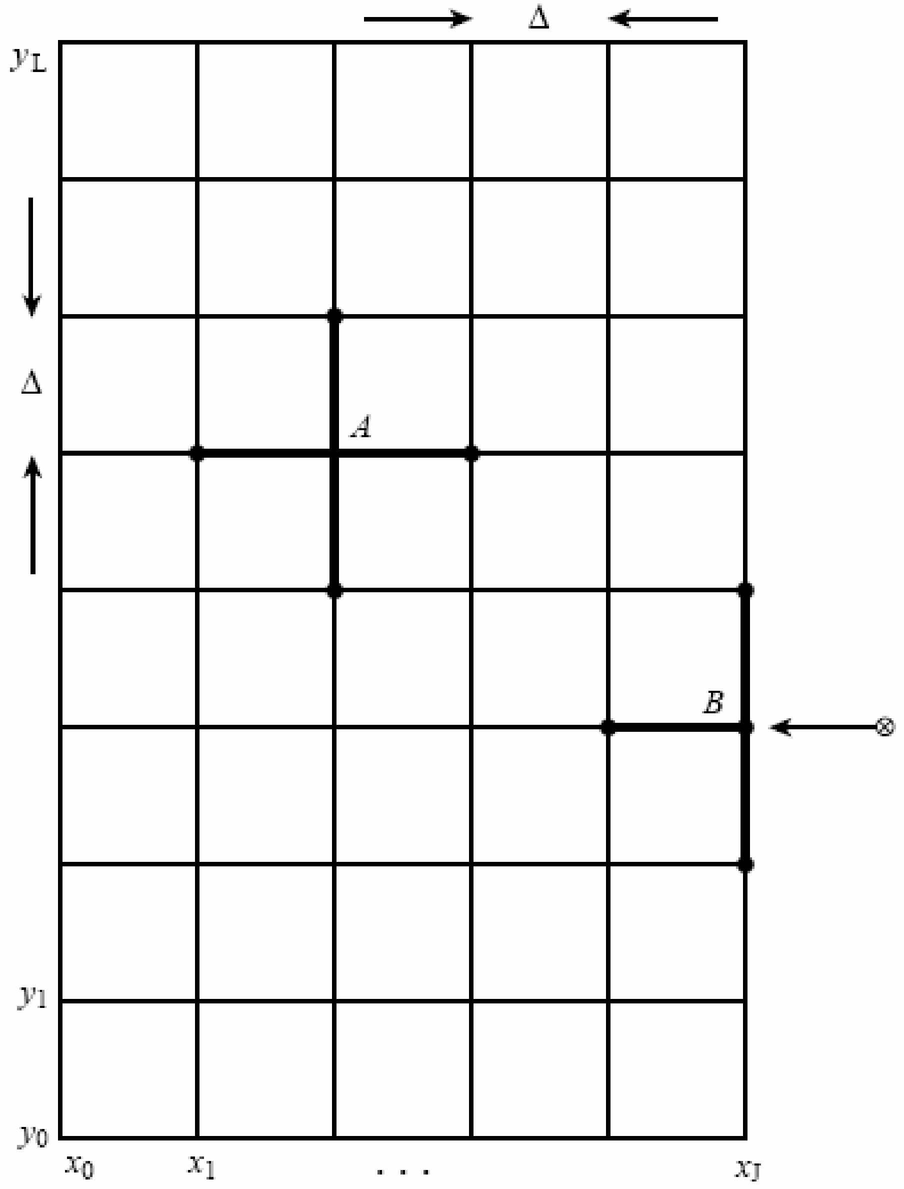
$$x_j = x_0 + j\Delta, \qquad j = 0, 1, ..., J \qquad u_{j,l} \text{ for } u(x_j, y_l)$$

$$y_l = y_0 + l\Delta, \qquad l = 0, 1, ..., L \qquad \rho_{j,l} \text{ for } \rho(x_j, y_l)$$

After discretisation we get the difference equation:

$$\frac{u_{j+1,l} - 2u_{j,l} + u_{j-1,l}}{\Delta^2} + \frac{u_{j,l+1} - 2u_{j,l} + u_{j,l-1}}{\Delta^2} = \rho_{j,l}$$

$$u_{i+L+1} + u_{i-(L+1)} + u_{i+1} + u_{i-1} - 4u_i = \Delta^2 \rho_i$$



Equation holds on inner points only!
On the boundary we specify:

-u (Dirichlet B.C.) or
-Derivative of u
(von Neumann B.C.)

How to solve the difference equation?

$$u_{i+L+1} + u_{i-(L+1)} + u_{i+1} + u_{i-1} - 4u_i = \Delta^2 \rho_i$$

We can interpret **u** as a vector and write the equation formally as an algebraic matrix equation:

$$\mathbf{A} \cdot \mathbf{u} = \mathbf{b}$$

- Theoretical one could solve this algebraic equation by well known algebraic equation solvers like Gauss-Jordan elimination.
- This is very unpractical, however, because A is very large and contains almost only zeros.

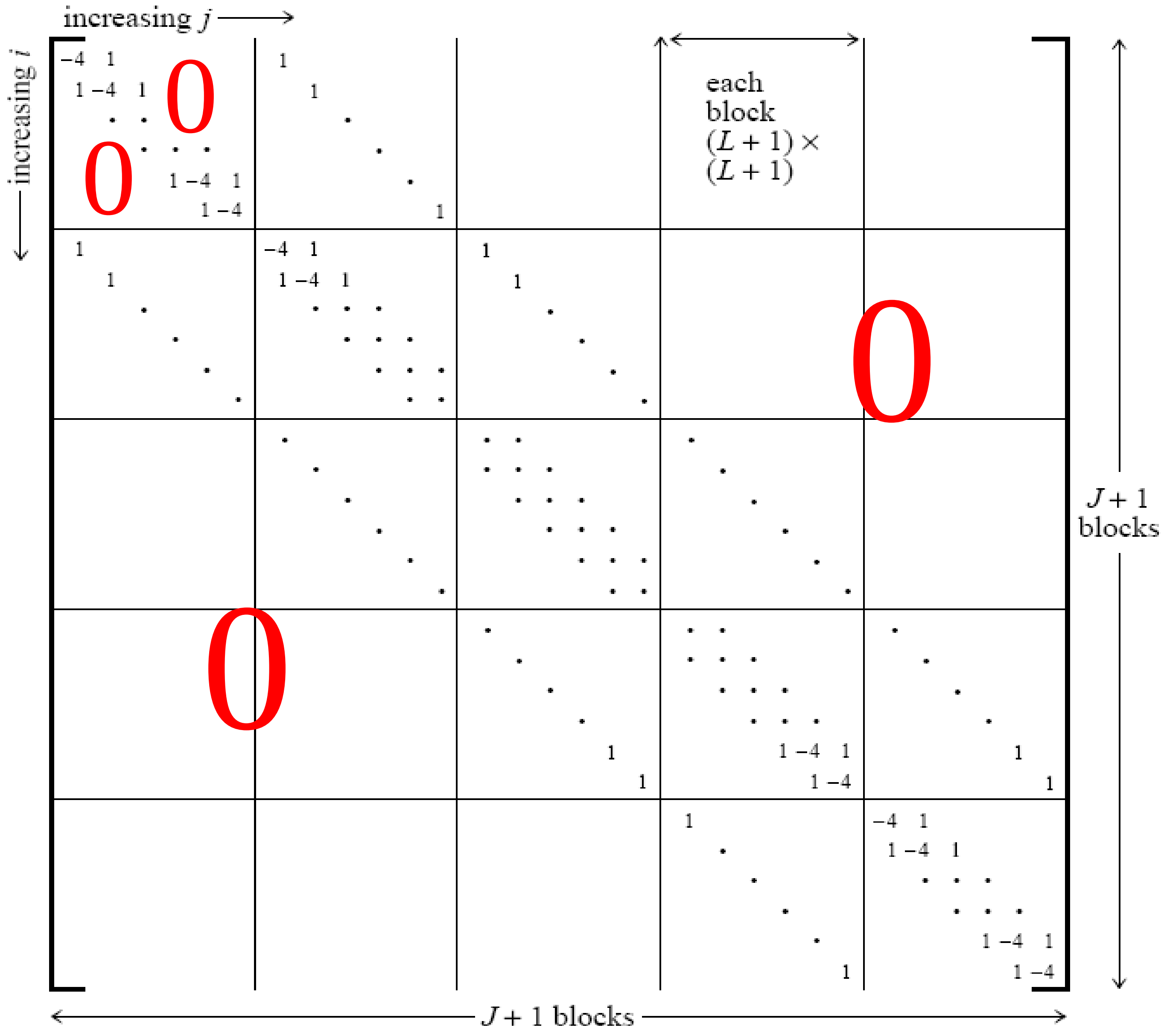
How large is **A** ?

- For a very moderate 2D-grid of 100x100
  -**u** has 100 x 100= 10^4 gridpoints, but
  -**A** has 10^4 x 10^4 =10^8 entries!
- For 3D-grids typically used in science application of about 300 x 300 x 300
  -**u** has 300^3= 2.7 *10^7 gridpoints,
  -**A** has (2.7 *10^7)^2 =7.29*10^{14} entries!

=> Memory requirement for 300-cube to store

u ~100 MB, **A**~3Million GByte

Structure of **A** ?



How to proceed?

- We have reduced our original PDE to algebraic equations (Here: system of linear equations, because we started from a linear PDE.)

- To do: Solve these equations.
- As exact Matrix solvers are of no much use we solve the equations numerically by **Relaxation methods**.

Relaxation: Jacobi method

Carl Jacobi
1804-1851

From $$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = \rho(x, y)$$

we derived the algebraic equations:

$$u_{i+L+1} + u_{i-(L+1)} + u_{i+1} + u_{i-1} - 4u_i = \Delta^2 \rho_i$$

Assume any initial value, say **u**=0 on all grid points (except the specified boundary values of course) and compute:

$$u^{n+1}_{j,l} = \frac{1}{4}\left(u^n_{j+1,l} + u^n_{j-1,l} + u^n_{j,l+1} + u^n_{j,l-1}\right) - \frac{\Delta^2}{4}\rho_{j,l}$$

Use the new values of **u** as input for the right side and repeat the iteration until **u** converges. (n: iteration step)

Relaxation: Jacobi method

- Jacobi method converge for diagonal dominant matrices A.
  (diagonal entries of A larger than the others)
- This condition is usually fulfilled for Matrix equations derived from finite differencing.
  (Tridiagonal block matrix: Most entries in A are zeros!)
- Jacobi method converges (but slowly) and can be used in principle, but maybe we can improve it?
- For practice: Method should converge fast!

Gauss Seidel method

C.F. Gauss
1777-1855

- Similar as Jacobi method.
- Difference: Use on the right-hand site already the new (and assumed to be better) approximation u^{n+1}, as soon as known.

$$u_{j,l}^{n+1} = \frac{1}{4}\left(u_{j+1,l}^{n} + u_{j-1,l}^{n+1} + u_{j,l+1}^{n} + u_{j,l-1}^{n+1}\right) - \frac{\Delta^2}{4}\rho_{j,l}$$

How fast do the methods converge?

To solve: $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$

We split **A** as: $\mathbf{A} = \underbrace{\mathbf{L}}_{\text{Lower Triangle}} + \underbrace{\mathbf{D}}_{\text{Diagonal Elements}} + \underbrace{\mathbf{U}}_{\text{Upper Triangle}}$

For the **r**th iteration step of the Jacobi method we get:

$$\mathbf{D} \cdot \mathbf{x}^{(r)} = -(\mathbf{L} + \mathbf{U}) \cdot \mathbf{x}^{(r-1)} + \mathbf{b}$$

How fast do the methods converge?

We have to investigate the iteration matrix

$$-\mathbf{D}^{-1} \cdot (\mathbf{L} + \mathbf{U})$$

Eigenvalues of iteration matrix define how fast residual are suppressed. Slowest decaying Eigenmode (largest factor) defines convergence rate. => Spectral radius ρ_s of relaxation operator. $0 < \rho_s < 1$

How many iteration steps r are needed to reduces the overall error by a factor of 10^{-p} ?

How many iteration steps **r** are needed to reduces the overall error by a factor of 10^{-p} ?

In general: $$r \approx \frac{p \ln 10}{(-\ln \rho_s)}$$

For a J x J grid and Dirichlet B.C. one gets:

Jacobi method

$$\rho_s \simeq 1 - \frac{\pi^2}{2J^2}$$

$$r \simeq \frac{2pJ^2 \ln 10}{\pi^2} \simeq \frac{1}{2} pJ^2$$

Gauss Seidel method

$$\rho_s \simeq 1 - \frac{\pi^2}{J^2}$$

$$r \simeq \frac{pJ^2 \ln 10}{\pi^2} \simeq \frac{1}{4} pJ^2$$

Can we do better?

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b} \qquad \mathbf{A} = \mathbf{L} + \mathbf{D} + \mathbf{U}$$

Gauss Seidel iteration:

$$(\mathbf{L} + \mathbf{D}) \cdot \mathbf{x}^{(r)} = -\mathbf{U} \cdot \mathbf{x}^{(r-1)} + \mathbf{b}$$

Can be rewritten as:

$$\mathbf{x}^{(r)} = \mathbf{x}^{(r-1)} - (\mathbf{L} + \mathbf{D})^{-1} \cdot \underbrace{\left[(\mathbf{L} + \mathbf{D} + \mathbf{U}) \cdot \mathbf{x}^{(r-1)} - \mathbf{b}\right]}_{\text{residual vector } \xi^{(r-1)}}$$

Successive Overrelaxation (SOR)

$$\mathbf{x}^{(r)} = \mathbf{x}^{(r-1)} - (\mathbf{L} + \mathbf{D})^{-1} \cdot \xi^{(r-1)}$$

Now we overcorrect the residual error by

$$\mathbf{x}^{(r)} = \mathbf{x}^{(r-1)} - \underbrace{\omega}_{\text{overrelaxation parameter}} (\mathbf{L} + \mathbf{D})^{-1} \cdot \xi^{(r-1)}$$

Method is only convergent for 0<w<2.
(for w<1 we have underrelaxation)
Aim: Find optimal overrelaxation parameter.
Often done empirically.

Successive Overrelaxation (SOR)

For the optimal overrelaxation parameter w the number of iteration steps to reduce the error by 10^{-p} are:

$$r \simeq \frac{pJ \ln 10}{2\pi} \simeq \frac{1}{3} pJ$$

Number of iteration steps increases only linear with the number of mesh points **J** for SOR method.
For Jacobi and Gauss Seidel it was ~$\mathbf{J^2}$

Successive Overrelaxation (SOR)

- SOR method only more effective when overrelaxation parameter w is close it’s optimum.
- Some analytic methods exist to estimate optimum w, but often one has to find it empirically.
- Unfortunately the optimum value w does not depend only on the PDE, but also on the grid resolution.
- The optimum asymptotic w is not necessarily a good initial choice.
- Chebyshev acceleration changes w during iteration.

Generalization of SOR-method.

Finite difference schemes from 2D-elliptic PDEs have the form:

$$a_{j,l}u_{j+1,l} + b_{j,l}u_{j-1,l} + c_{j,l}u_{j,l+1} + d_{j,l}u_{j,l-1} + e_{j,l}u_{j,l} = f_{j,l}$$

$a = b = c = d = 1, e = -4$ for our example

We iterate for the solution by

$$u^*{}_{j,l} = \frac{1}{e_{j,l}}\left(f_{j,l} - a_{j,l}u_{j+1,l} - b_{j,l}u_{j-1,l} - c_{j,l}u_{j,l+1} - d_{j,l}u_{j,l-1}\right)$$

and get:

$$u^{\text{new}}_{j,l} = \omega u^*{}_{j,l} + (1-\omega)u^{\text{old}}_{j,l}$$

Generalization to 3D is straight forward

Summary: Relaxation methods

1.) Choose an initial solution u^0 (usually zeros)

2.) Relax for u^{new} from u^{old} (Jacobi, GS, SOR)

3.) Are u^{old} and u^{new} identical within some tolerance level?

If No continue, If yes solution is found.

4.) $u^{old} = u^{new}$ and go to step 2.)

Iterate only where u is unknown!!

-Dirichlet B.C.: u remains unchanged on boundaries.

-von Neumann: compute u from grad(u)=known at each iteration step before 2.) [Ghost cells or one-sided derivatives]

Computing time for relaxation methods

- For a **J** x **J** 2D-PDE the number of iteration steps is ~$\mathbf{J^2}$ (Jacobi GS) or ~**J** (SOR)
- But: Each iteration step takes ~$\mathbf{J^2}$
- Total computing time: ~$\mathbf{J^4}$ (Jacobi, Gauss Seidel)

  ~$\mathbf{J^3}$ (SOR-method)

- Computing time depends also on other factors:
  -required accuracy
  -computational implementation
  -IDL is much slower as C or Fortran
  -Hardware and parallelization

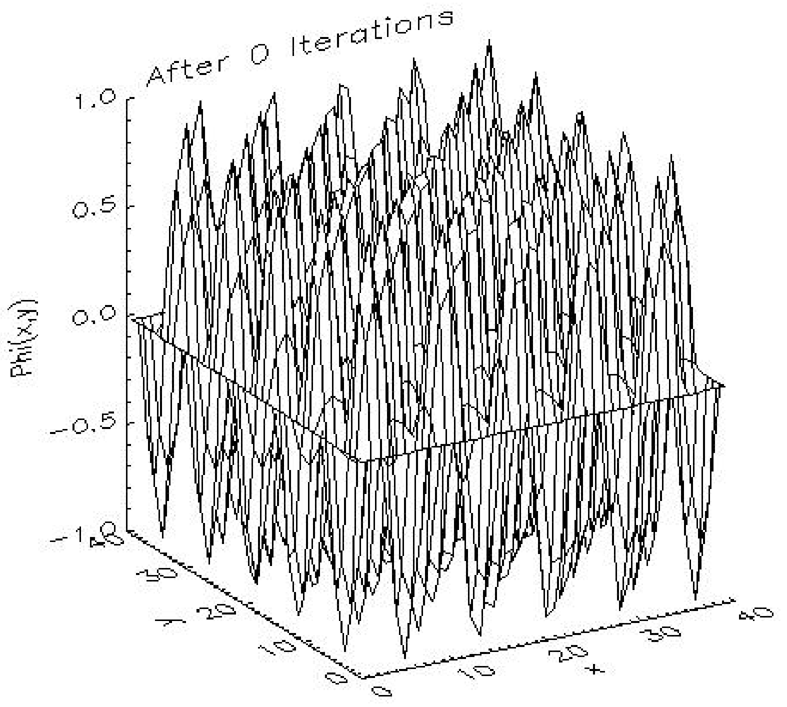
How fast are errors smoothed out?

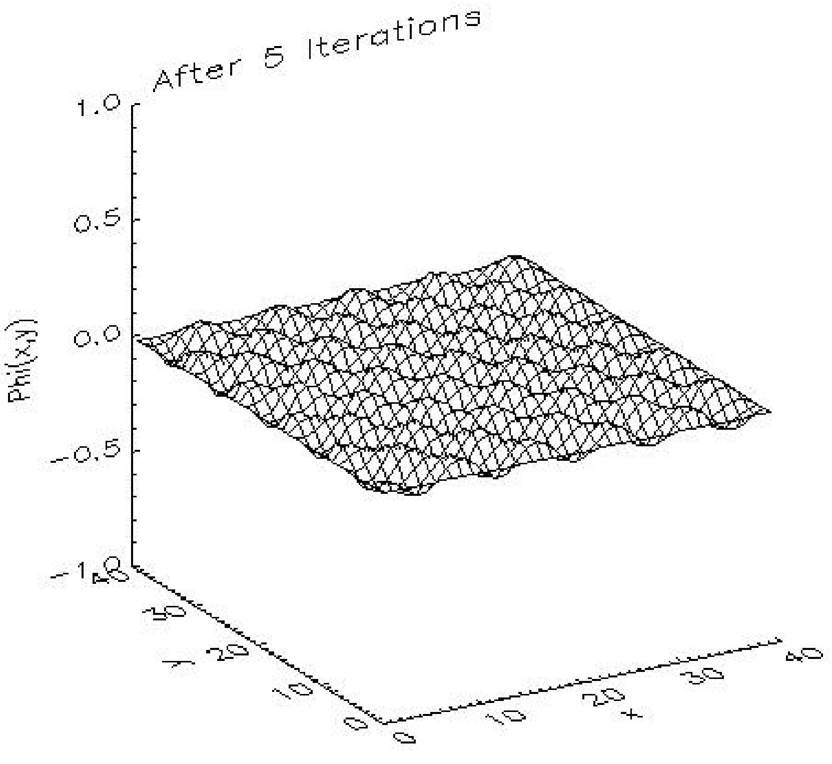
Show: demo_laplace.pro

This IDL program shows how fast or slow
Errors of different wave-length are relaxed
for Jacobi, Gauss-Seidel and SOR for
the homogenous Laplace-equation.

How fast are errors smoothed out?

We use Gauss-Seidel 40x40 box and investigate a **high frequency (k=10)** disturbance.

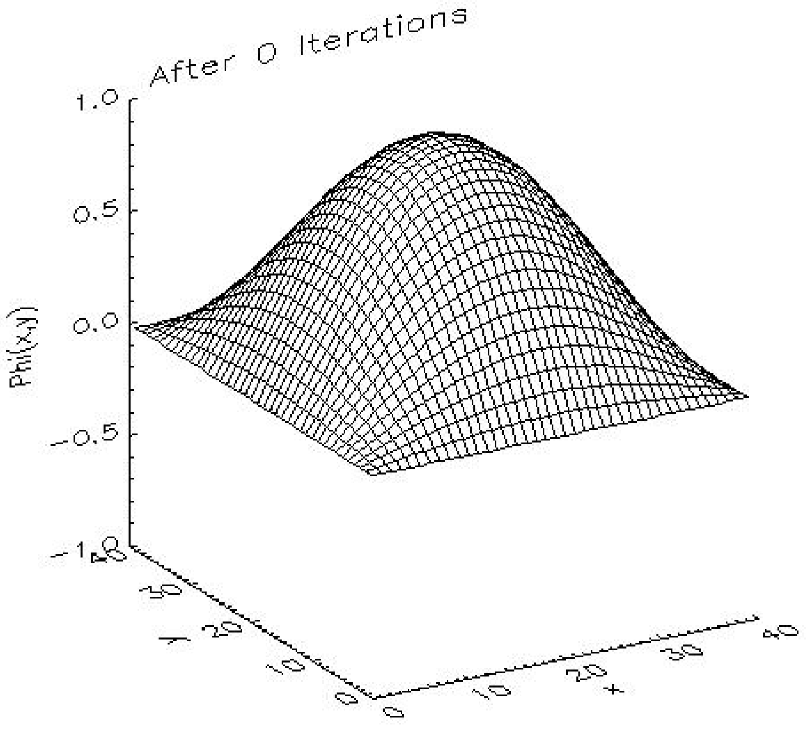


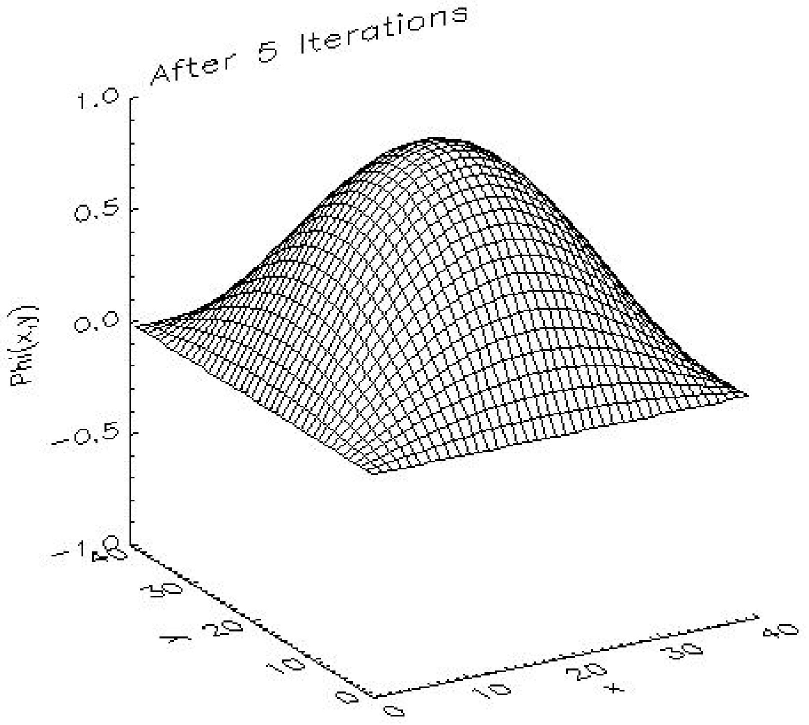


Converged (Error $<10^{-6}$) after **24** iteration steps)

How fast are errors smoothed out?

We use Gauss-Seidel 40x40 box and investigate a **low frequency (k=1)** disturbance.





Converged (Error $<10^{-6}$) after **747** iteration steps)

How fast are errors smoothed out?

We use **Gauss-Seidel** on JxJ boxes and investigate number of steps to converge for different frequencies

J \ **k**	1	10	20	40
40	747	24	13	11
80	2615	67	26	14
160	8800	216	72	28

Gauss-Seidel method is very good smoother!

How fast are errors smoothed out?

Same test with SOR method

k / **J**	1	10	20	40
40	81	109	112	119
80	213	141	146	152
160	844	173	179	189

SOR is a faster solver, but NOT good smoother!

How fast are errors smoothed out? (Gauss-Seidel)

- High frequency errors are smoothed out fast.
- Low frequency errors take very long to vanish.
- But the long frequency errors are reduced faster on low resolution grids.
- Can we use this property to speed up the relaxation?
- Yes! The answer is **Multigrid**.

Multigrid Methods

- Aim: Be even better (faster) then
  the SOR-method.
- From experience we know that any
  relaxation methods smoothes out errors
  fast on small length scales, but very slowly
  on large scales.
- Idea: compute solution on grids with reduced
  spatial resolution.
- Interpolate to finer grids.
- Need to swap between grids in a consistent way.

Multigrid Methods

We want to solve the linear elliptic PDE

$\mathcal{L}u = f$ discretized we get $\mathcal{L}_h u_h = f_h$

If $\widetilde{u}_h$ is an approximation and u_h the exact solution we have an error of:

$$v_h = u_h - \widetilde{u}_h$$

The residual or defect $d_h = \mathcal{L}_h \widetilde{u}_h - f_h$

and for the error $\mathcal{L}_h v_h = -d_h$

Multigrid methods

Any iteration methods now uses a simplified operator (e.g. Jacobi: diagonal part only, GS: lower triangle) to find error or correction: $\widehat{\mathcal{L}}_h \widehat{v}_h = -d_h$

and the next approximation $\widetilde{u}_h^{\text{new}} = \widetilde{u}_h + \widehat{v}_h$

Now we take a different approach. We do not simplify the operator, but approximate $\mathcal{L}_h$ on a coarser grid H=2h by

$$\mathcal{L}_H v_H = -d_H$$

which will be easier to solve, because of lower dimension.

Multigrid Methods

We need an restriction operator to compute the residual on the coarser grid:

$$d_H = \mathcal{R} d_h$$

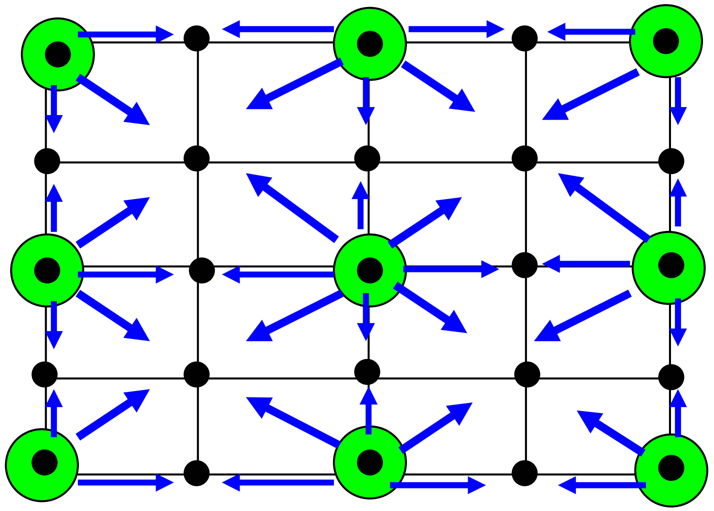
And after we find the solution $\widetilde{v}_H$ on the coarser grid a prolongation operator to interpolate to the finer grid:

$$\widetilde{v}_h = \mathcal{P} \widetilde{v}_H$$

Finally we update:

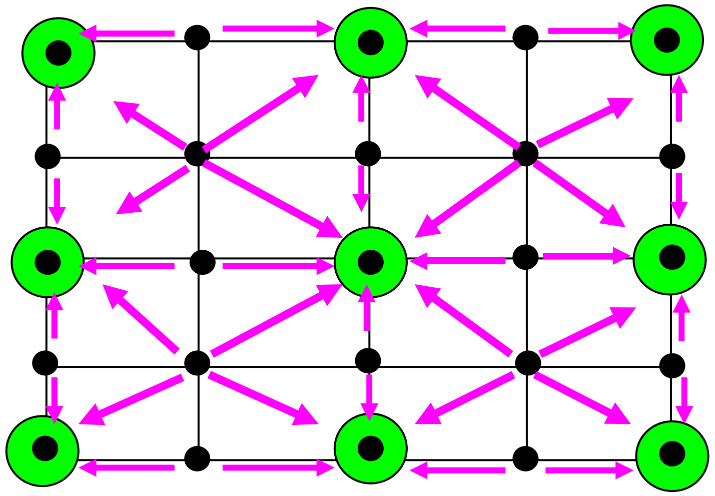
$$\widetilde{u}_h^{\text{new}} = \widetilde{u}_h + \widetilde{v}_h$$

Multigrid Methods



Prolongation (coarse to fine)

$$\begin{bmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ \frac{1}{2} & 1 & \frac{1}{2} \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \end{bmatrix}$$



Restriction (fine to coarse)

$$\begin{bmatrix} \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \\ \frac{1}{8} & \frac{1}{4} & \frac{1}{8} \\ \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \end{bmatrix}$$

Coarse grid correction

One coarse-grid correction step in
a 2-level Multigrid scheme contains:

- Compute defect on fine grid.
- Restrict defect to coarse grid.
- Solve correction exactly on coarse grid.
- Prolongate (interpolate) correction to fine grid.
- Update next approximation.

2-level Multigrid scheme

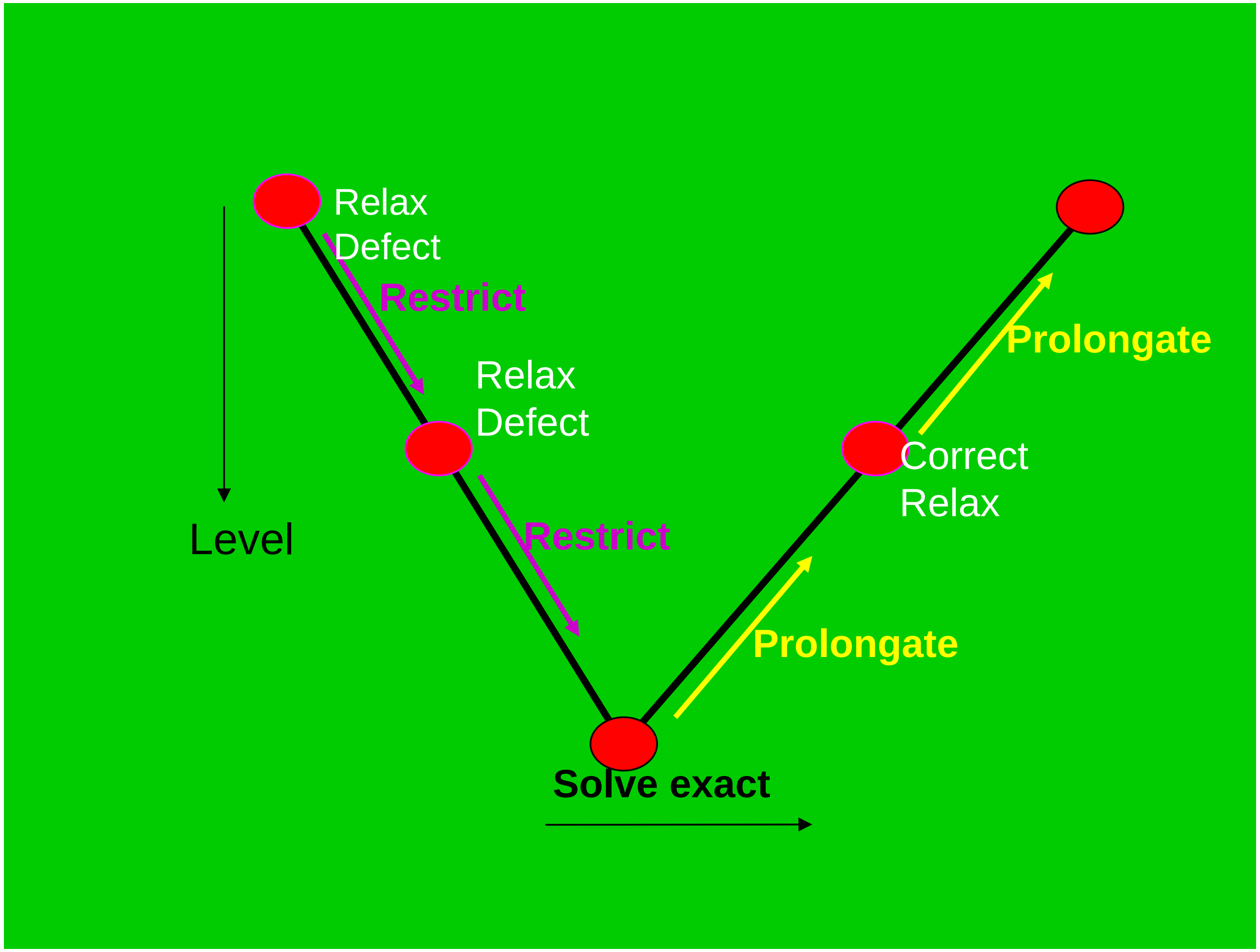
- Pre-smoothing: Apply some relaxation steps (usually with Gauss-Seidel method) on fine grid.
- Coarse grid correction.
- Post-smoothing: Relax some steps again on the fine grid to the updated solution.

-High frequency defects are smoothed out
fast on the fine grid.
- Low frequency defects (which took very long
to relax on fine grid) are taken care by on coarse grid.

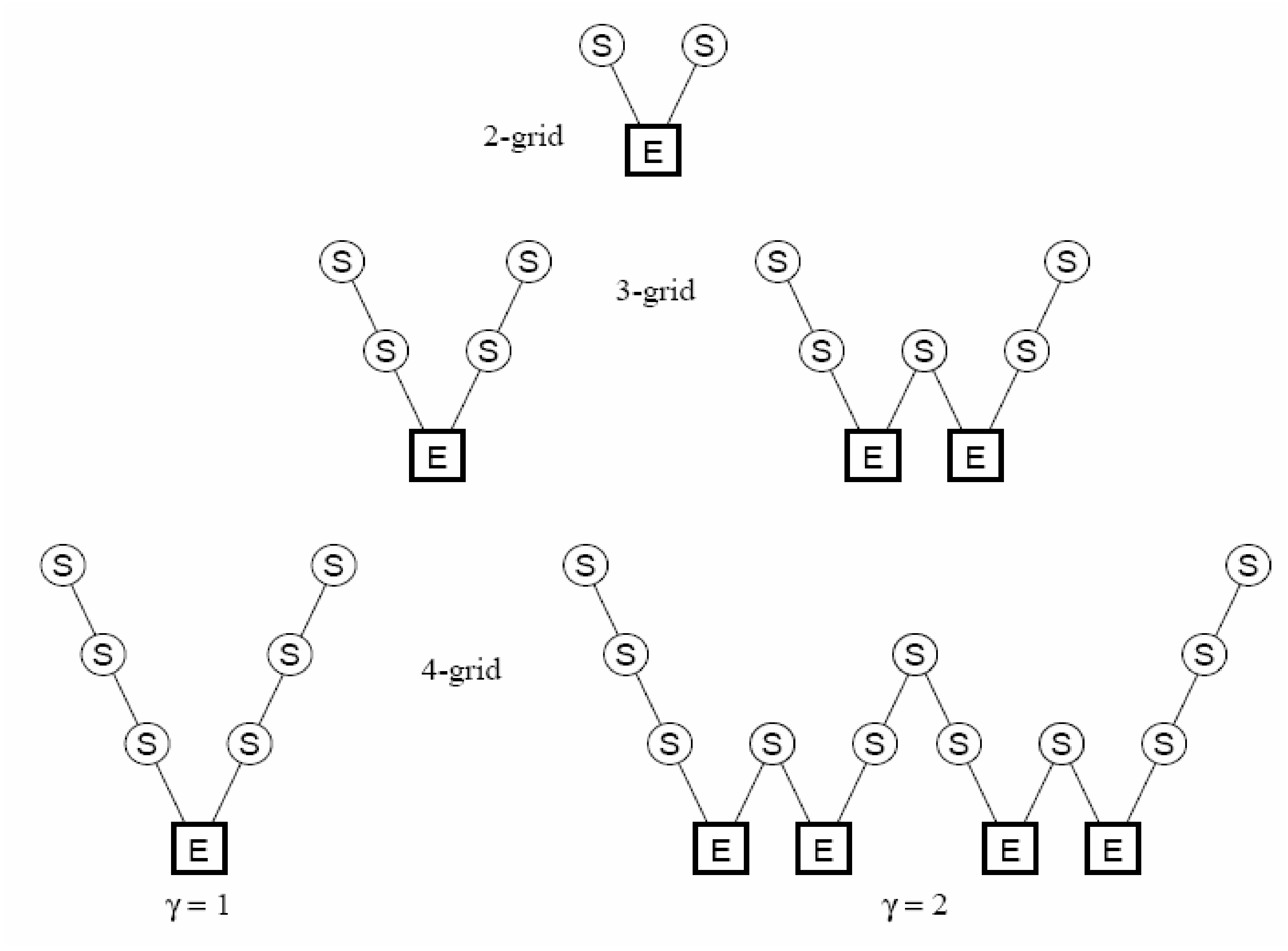
N-level Multigrid scheme

- Generalization of 2-level multigrid method.
- Instead of solving the equation on 2. grid exactly we approximate it on an even coarser grid.
- Very easy to solve on coarsest grid.
- Different possibilities cycles are possible:
  -V-cycle
  -W-cycle
  -Full multigrid
- Hint: Do not use the SOR-method for smoothing (but Gauss-Seidel). Overrelaxation in SOR-methods destroys the high-frequency smoothing.

V-cycle for 3 levels

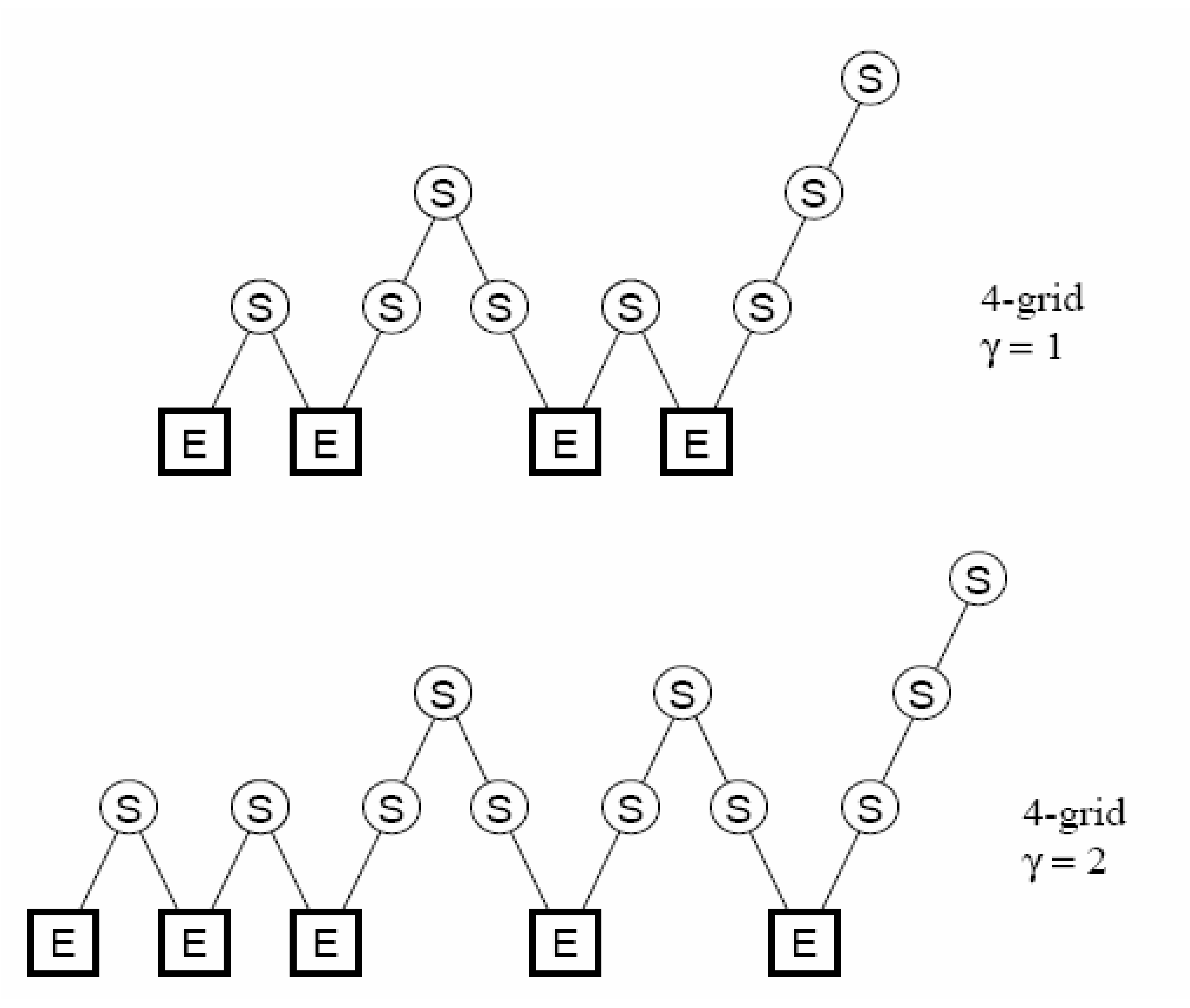


V-cycle W-cycle



Full Multigrid cycles

Start on coarsest grid



Multigrid and Full Multigrid

- Multigrid methods speed up the convergence of relaxation scheme.
- Number of cycles needed does not depend on grid size. (computing time for each cycle does of course)
- Way more demanding in programming afford.
- Multigrid computes only defect on coarser grid, but Full Multigrid (FMG) provides solution of the PDE on all grids.
- FMG can be generalized for nonlinear PDEs, Full Approximation Storage Algorithm (FAS). Discussion is outside scope of this lecture.

Summary: Relaxation Methods

- Methods are well suited to solve Matrix equations derived from finite difference representation of elliptic PDEs.
- Classic methods are easy to program and suitable not to large numerical grids. Computing time increases rapidly with grid size.
- Multigrid methods are much faster for large grids and should be first choice.
- Computational implementation of Multigrid Methods is way more demanding.

Alternatives to solve Matrix Equations derived from PDEs

- **Direct Matrix solvers:** Only for very small 2D-Problems or as exact solver on coarsest Multigrid.
- **Fast Fourier Transform Methods (FFT):** Suitable for linear PDEs with constant coefficients. Original FFT assumes periodic boundary conditions. Fourier series solutions look somewhat similar as what we got from separation of variables.
- **Krylov subspace methods:** Zoo of algorithms for sparse matrix solvers, e.g. Conjugate Gradient Method (CG).

Exercise:
2D-Poisson equation

lecture_poisson2d_draft.pro

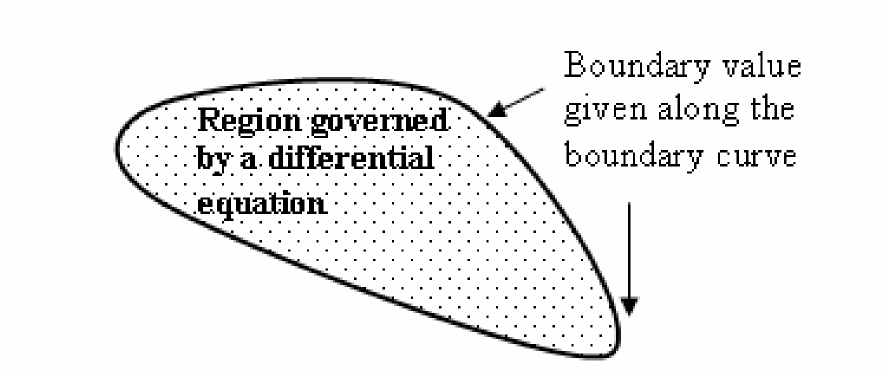
This is a draft IDL-program to solve the Poisson-equation for provide charge distribution.

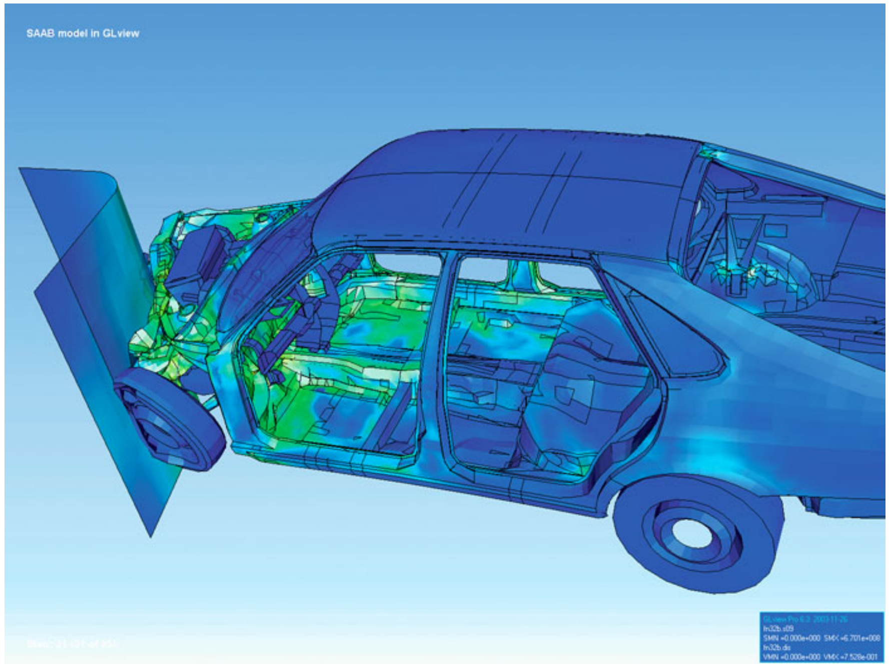
Task: implement Jacobi, Gauss-Seidel and SOR-method. Find optimal relaxation parameter for SOR-method.

Elliptic PDEs Summary

- Discretized differential equations lead to difference equations and **algebraic equations**.
- System of coupled equations is way to large for direct solvers. => Use **Relaxation methods.**
- **Gauss-Seidel** and **SOR**-method are in particular suitable to solve algebraic equations derived from elliptic PDEs.
- Fastest solvers are based on **Multigrid** methods.

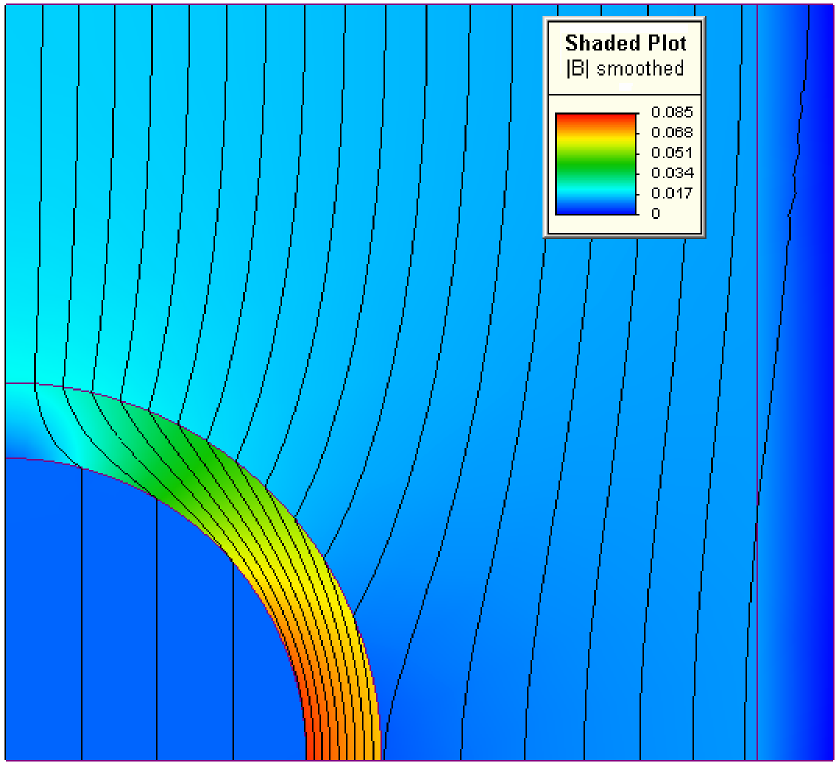
Finite Element Method (FEM)

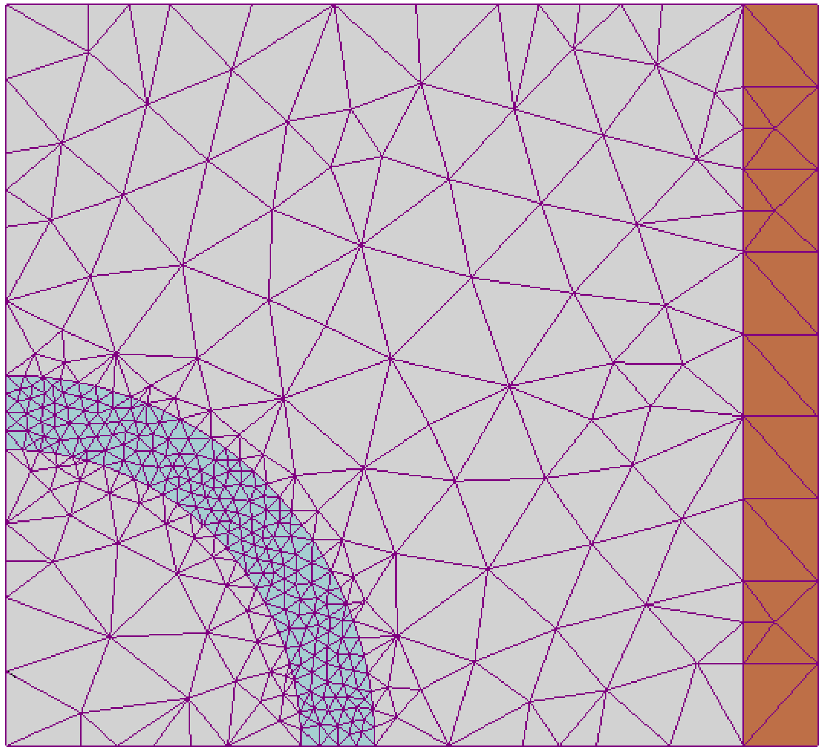

Region governed by a differential equation
Boundary value given along the boundary curve



Arbitrary shaped boundaries are difficult to implement in finite difference methods.
Alternative: Finite Elements, popular in particular to solve PDEs in engineering/structural mechanics.

Finite Elements





FEM covers the space with finite elements (in 2D often triangles, in 3D tetrahedra). The elements **do not need** to have the same size and shape.

This allows to use a higher resolution where needed.

Variational formulation: 1D example

$$\mathrm{P1} \ : \begin{cases} u'' = f \text{ in } (0,1), \\ u(0) = u(1) = 0, \end{cases}$$

If u fulfills P1 and v(x) is an arbitrary function which vanishes on the boundary:

$$\int_0^1 f(x)v(x)\,dx = \int_0^1 u''(x)v(x)\,dx$$

Partial integration of right side

$$= u'(x)v(x)|_0^1 - \int_0^1 u'(x)v'(x)\,dx$$

$$= -\int_0^1 u'(x)v'(x)\,dx = -\phi(u,v).$$

Weak formulation of the PDE

Solution of weak problem and original PDE are identical.

Variational formulation: 2D example

$$\text{P2} \ : \begin{cases} u_{xx} + u_{yy} = f & \text{in } \Omega, \\ u = 0 & \text{on } \partial\Omega, \end{cases}$$

Poisson equation

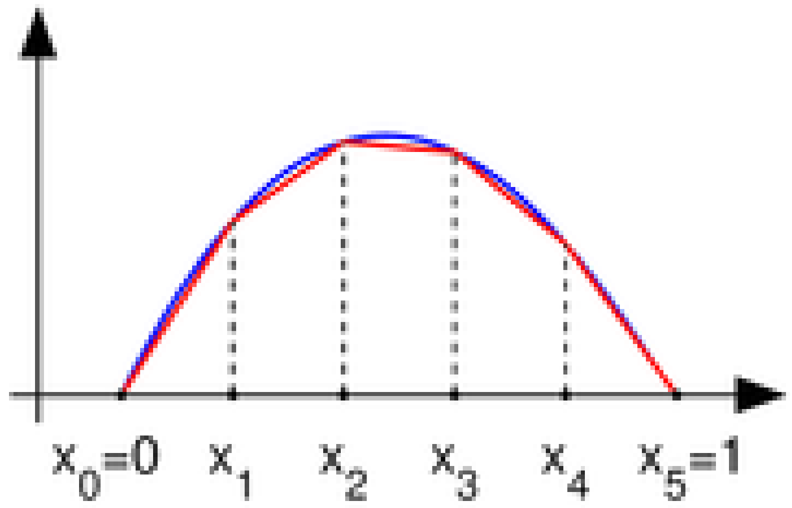
For an arbitrary function v the PDE can again be formulated in weak form (using Greens theorem):

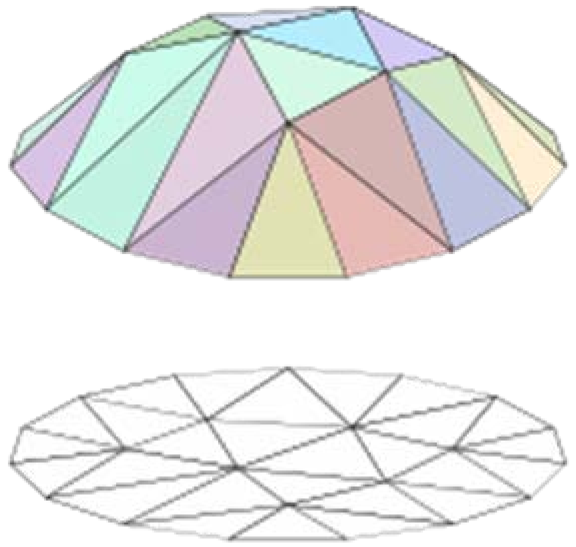
$$\int_\Omega f v \, ds = -\int_\Omega \nabla u \cdot \nabla v \, ds = -\phi(u, v),$$

If we find a solution for the weak problem, we solved our (strong form) original PDE. Order of derivatives is reduced in weak form, which is helpful to treat discontinuities.

Shape function v

- How to choose the function v ?
- v must be at least once differentiable.
- For FEM-approach one takes polynomials or in lowest order **piecewise linear functions**:





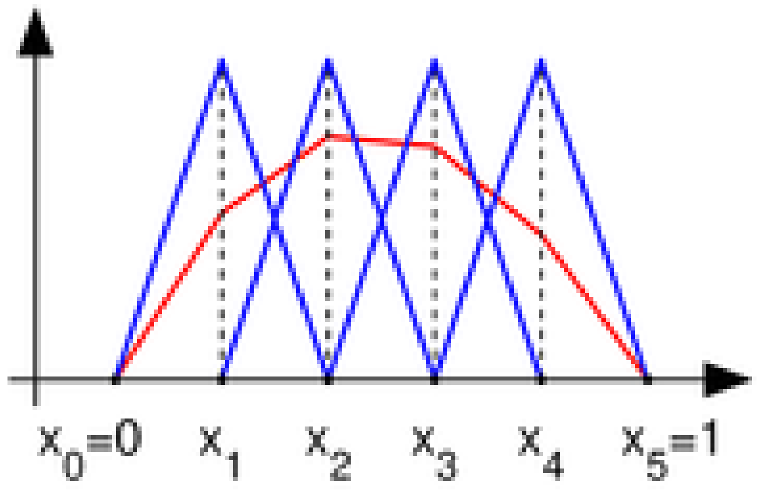
1D

2D

Basis of functions for v

We choose piecewise linear functions which are one at a particular grid-point and zero at all other grid-points (triangle or tent-function)

$$v_k(x) = \begin{cases} \frac{x-x_{k-1}}{x_k - x_{k-1}} & \text{if } x \in [x_{k-1}, x_k], \\ \frac{x_{k+1} - x}{x_{k+1} - x_k} & \text{if } x \in [x_k, x_{k+1}], \\ 0 & \text{otherwise}, \end{cases}$$



We get function value and derivative by interpolation.

Basic tent-function (blue) and superposition to piecewise linear function (red)

Basis of functions for v

- For such base-functions almost all integrals in the form:

1D

$$\langle v_j, v_k \rangle = \int_0^1 v_j v_k \, dx$$

$$\phi(v_j, v_k) = \int_0^1 v_j' v_k' \, dx$$

2D

$$\int_\Omega v_j v_k \, ds$$

$$\int_\Omega \nabla v_j \cdot \nabla v_k \, ds$$

are zero. Only integrals of elements sharing grid points (edges of triangles in 2D) are non-zero.

From FEM to matrix form

Let's try to describe the unknown function u(x) and the known f(x) with these basis functions:

$$u(x) = \sum_{k=1}^{n} u_k v_k(x) \qquad f(x) = \sum_{k=1}^{n} f_k v_k(x)$$

Aim: Find the parameters u_k !
This will be the solution in FEM-approach.

How to find this solution?
Insert this approaches for u and f into the weak formulation of the PDE.

From FEM to matrix form

$$-\sum_{k=1}^{n} u_k \underbrace{\phi(v_k, v_j)}_{\mathbf{L_{kj}}} = \sum_{k=1}^{n} f_k \underbrace{\int v_k v_j}_{\mathbf{M_{kj}}}$$

which leads to a system of equations which has to be resolved for u_k .
We can write in matrix form:

$$-L\mathbf{u} = M\mathbf{f}$$

This sparse matrix system can be solved with the method we studied for finite differences.

Lets remember all steps:

$$\text{P2} \;:\; \begin{cases} u_{xx} + u_{yy} = f & \text{in } \Omega, \\ u = 0 & \text{on } \partial\Omega, \end{cases}$$

Original PDE (strong form)

$$\int_\Omega f v \, ds = -\int_\Omega \nabla u \cdot \nabla v \, ds = -\phi(u, v),$$

PDE in weak form

$$-\sum_{k=1}^{n} u_k \phi(v_k, v_j) = \sum_{k=1}^{n} f_k \int v_k v_j$$

PDE in discretized form

Solve corresponding sparse Matrix system:
=> Solution of PDE in FEM-approach.

Finite Element Method Summary

- **Finite Elements** are an alternative to **finite differences**. Good for complicated boundaries.
- PDE is solved in **weak form**.
- More flexible as finite differences, but also more complicated to implement in code.
- Setting up the optimal grid can be tricky. (Some research groups only work on this.)