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:

\[ \nabla \times \mathbf{B} = \mu_0 \mathbf{j} + \epsilon_0 \mu_0 \frac{\partial \mathbf{E}}{\partial t} \]
\[ \nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t} \]
\[ \nabla \cdot \mathbf{B} = 0 \]
\[ \nabla \cdot \mathbf{E} = \frac{1}{\epsilon_0} \rho \]

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

\[ \mathbf{B} = \nabla \times \mathbf{A} \]
\[ \mathbf{E} = -\nabla \Phi - \frac{\partial \mathbf{A}}{\partial t} \]

together with the Coulomb Gauge condition:

\[ \nabla \cdot \mathbf{A} = 0 \]
With these definitions we get:

\[
\nabla \times \nabla \times \mathbf{A} = \mu_0 \mathbf{j} \\
\nabla \times (-\nabla \Phi - \frac{\partial \mathbf{A}}{\partial t}) = -\frac{\partial \nabla \times \mathbf{A}}{\partial t} \checkmark \\
\nabla \cdot \nabla \times \mathbf{A} = 0 \checkmark \\
\nabla \cdot (-\nabla \Phi - \frac{\partial \mathbf{A}}{\partial t}) = \frac{1}{\epsilon_0 \rho}
\]

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

\[
\nabla (\nabla \cdot \mathbf{A}) - \Delta \mathbf{A} = \mu_0 \mathbf{j} \\
-\Delta \Phi - \frac{\partial (\nabla \cdot \mathbf{A})}{\partial t} = \frac{1}{\epsilon_0 \rho}
\]

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

\[
-\Delta \mathbf{A} = \mu_0 \mathbf{j} \\
-\Delta \Phi = \frac{1}{\epsilon_0 \rho}
\]
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:

\[x_j = x_0 + j \Delta, \quad j = 0, 1, \ldots, J\]
\[y_l = y_0 + l \Delta, \quad l = 0, 1, \ldots, L\]

\[u_{j,l} \text{ for } u(x_j, y_l)\]
\[\rho_{j,l} \text{ for } \rho(x_j, y_l)\]

After discretisation we get the difference equation:

\[
\frac{u_{j+1,l} - 2u_{j,l} + u_{j-1,l}}{\Delta^2} + \frac{u_{j,l+1} - 2u_{j,l} + u_{j,l-1}}{\Delta^2} = \rho_{j,l}
\]
Equation holds on inner points only!
On the boundary we specify:

\[-u \text{ (Dirichlet B.C.) or} \]
\[-\text{Derivative of } u \text{ (von Neumann B.C.)} \]
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:

\[ A \cdot u = b \]

- 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:
  - $\mathbf{u}$ has $100 \times 100 = 10^4$ gridpoints, but
  - $A$ has $10^4 \times 10^4 = 10^8$ entries!

- For 3D-grids typically used in science application of about 300 x 300 x 300:
  - $\mathbf{u}$ has $300^3 = 2.7 \times 10^7$ gridpoints,
  - $A$ has $(2.7 \times 10^7)^2 = 7.29 \times 10^{14}$ entries!

=> Memory requirement for 300-cube to store
   $\mathbf{u} \sim 100$ MB,  $A \sim 3$ Million GByte
Structure of $A$?

Each block $(L + 1) \times (L + 1)$
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

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

• 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: \[ A \cdot x = b \]

We split \( A \) as: \[ A = L + D + U \]

\begin{align*}
\text{Lower Triangle} & \quad \text{Diagonal Elements} & \quad \text{Upper Triangle} \\
\text{Diagonal Elements} & \quad \text{Upper Triangle} & \quad \text{Lower Triangle}
\end{align*}

For the \( r \)th iteration step of the Jacobi method we get:

\[ D \cdot x^{(r)} = -(L + U) \cdot x^{(r-1)} + b \]
How fast do the methods converge?

We have to investigate the iteration matrix

\[-D^{-1} \cdot (L + U)\]

Eigenvalues of iteration matrix define how fast residual are suppressed. Slowest decaying Eigenmode (largest factor) defines convergence rate. => Spectral radius $\rho_s$ 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 reduce the overall error by a factor of $10^{-p}$?

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

For a $J \times J$ grid and Dirichlet B.C. one gets:

**Jacobi method**

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

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

**Gauss Seidel method**

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

$r \approx \frac{pJ^2 \ln 10}{\pi^2} \approx \frac{1}{4}pJ^2$
Can we do better?

\[ A \cdot x = b \quad A = L + D + U \]

Gauss Seidel iteration:

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

Can be rewritten as:

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

residual vector \( \xi^{(r-1)} \)
Successive Overrelaxation (SOR)

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

Now we overcorrect the residual error by

\[ x^{(r)} = x^{(r-1)} - \omega (L + D)^{-1} \cdot \xi^{(r-1)} \]

overrelaxation parameter

Method is only convergent for 0<\(\omega\)<2.
(for \(\omega\)<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:

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

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

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

\[ a = b = c = d = 1, \quad e = -4 \quad \text{for our example} \]

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 straightforward.
Summary: Relaxation methods

1.) Choose an initial solution \( u^0 \) (usually zeros)
2.) Relax for \( u^{\text{new}} \) from \( u^{\text{old}} \) (Jacobi, GS, SOR)
3.) Are \( u^{\text{old}} \) and \( u^{\text{new}} \) identical within some tolerance level?
   If No continue, If yes solution is found.
4.) \( u^{\text{old}} = u^{\text{new}} \) and go to step 2.)

Iterate only where \( u \) is unknown!!
-Dirichlet B.C.: \( u \) remains unchanged on boundaries.
-von Neumann: compute \( u \) from \( \text{grad}(u) = \text{known at each iteration step before 2.} \) [Ghost cells or one-sided derivatives]
Computing time for relaxation methods

- For a $J \times J$ 2D-PDE the number of iteration steps is $\sim J^2$ (Jacobi GS) or $\sim J$ (SOR)
- But: Each iteration step takes $\sim J^2$
- Total computing time: $\sim J^4$ (Jacobi, Gauss Seidel)
  $\sim J^3$ (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.

<table>
<thead>
<tr>
<th>J</th>
<th>k</th>
<th>1</th>
<th>10</th>
<th>20</th>
<th>40</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td></td>
<td>747</td>
<td>24</td>
<td>13</td>
<td>11</td>
</tr>
<tr>
<td>80</td>
<td>2615</td>
<td>67</td>
<td>26</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>160</td>
<td>8800</td>
<td>216</td>
<td>72</td>
<td>28</td>
<td></td>
</tr>
</tbody>
</table>

Gauss-Seidel method is very good smoother!
How fast are errors smoothed out?
Same test with SOR method

<table>
<thead>
<tr>
<th>J</th>
<th>k=1</th>
<th>k=10</th>
<th>k=20</th>
<th>k=40</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>81</td>
<td>109</td>
<td>112</td>
<td>119</td>
</tr>
<tr>
<td>80</td>
<td>213</td>
<td>141</td>
<td>146</td>
<td>152</td>
</tr>
<tr>
<td>160</td>
<td>844</td>
<td>173</td>
<td>179</td>
<td>189</td>
</tr>
</tbody>
</table>

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) than 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 \( \tilde{u}_h \) is an approximation and \( u_h \) the exact solution we have an error of:

\[ v_h = u_h - \tilde{u}_h \]

The residual or defect \( d_h = \mathcal{L}_h \tilde{u}_h - f_h \)

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:

\[ \hat{\mathcal{L}}_h \hat{v}_h = -d_h \]

and the next approximation \( \tilde{u}_h^{\text{new}} = \tilde{u}_h + \hat{v}_h \)

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

\[ \mathcal{L}_H \nu_H = -d_H \]

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 \( \tilde{\nu}_H \) on the coarser grid a prolongation operator to interpolate to the finer grid:

\[ \tilde{\nu}_h = \mathcal{P} \tilde{\nu}_H \]

Finally we update:

\[ \tilde{u}_h^{\text{new}} = \tilde{u}_h + \tilde{\nu}_h \]
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

2-grid

3-grid

4-grid

γ = 1

γ = 2
Full Multigrid cycles
Start on coarsest grid

4-grid
γ = 1

E E E E E

4-grid
γ = 2

E E E E E
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.

Elliptic PDEs

Summary

• Discretized differential equations lead to difference equations and \textit{algebraic equations}.

• System of coupled equations is way too large for direct solvers. $\Rightarrow$ Use \textit{Relaxation methods}.

• \textit{Gauss-Seidel} and \textit{SOR}-method are in particular suitable to solve algebraic equations derived from elliptic PDEs.

• Fastest solvers are based on \textit{Multigrid} methods.
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

\[ P1 : \begin{cases} \quad u'' = f \text{ in } (0, 1), \\ \quad u(0) = u(1) = 0, \end{cases} \]

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

\[
\int_0^1 f(x)v(x) \, dx = \int_0^1 u''(x)v(x) \, dx \quad \text{Partial integration of right side}
\]

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

\[
= -\int_0^1 u'(x)v'(x) \, dx = -\phi(u, v). \quad \text{Weak formulation of the PDE}
\]

Solution of weak problem and original PDE are identical.
Variational formulation: 2D example

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 \textit{piecewise linear functions}:
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:

  \[ \langle v_j, v_k \rangle = \int_0^1 v_j v_k \, dx \quad \int_\Omega v_j v_k \, ds \]

  \[ \phi(v_j, v_k) = \int_0^1 v'_j v'_k \, dx \quad \int_\Omega \nabla v_j \cdot \nabla v_k \, ds \]

  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:

\[
\begin{align*}
  u(x) &= \sum_{k=1}^{n} u_k \varphi_k(x) \\
  f(x) &= \sum_{k=1}^{n} f_k \varphi_k(x)
\end{align*}
\]

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

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

which leads to a system of equations which has to be resolved for \( u_k \).

We can write in matrix form:

\[ -Lu = Mf \]

This sparse matrix system can be solved with the method we studied for finite differences.
Lets remember all steps:

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

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.)