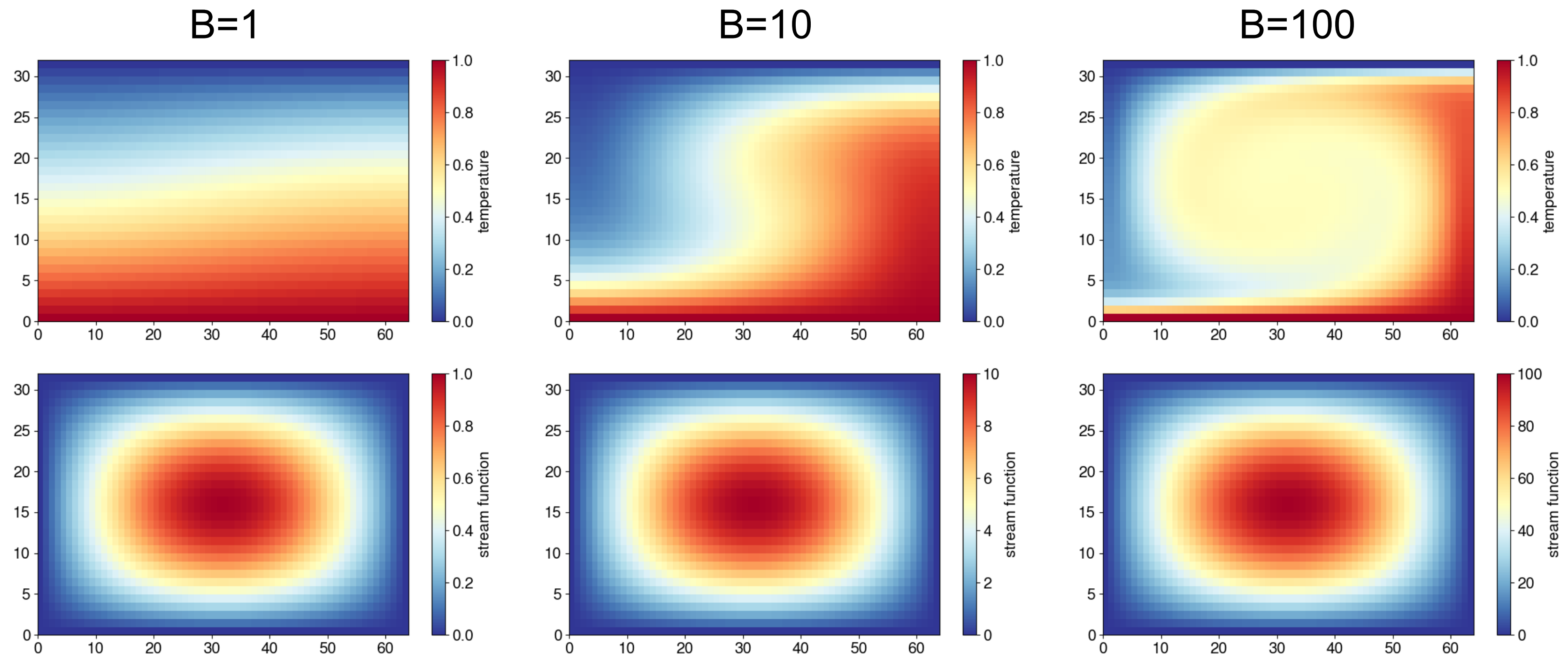
Exercise 6

- In the last exercise the flow was fixed:



- Next step: Make flow dependent on temperature → convection

Exercise 6

Boussinesq approximation of the Navier-Stokes equations for a fluid with high viscosity ($Pr \rightarrow \infty$):

$$-\nabla p + \nabla^2 \vec{v} = -\mathrm{Ra} \cdot T \cdot \vec{e}_y$$

$$\mathrm{Pr} = \frac{\nu}{\kappa} \qquad \mathrm{Ra} = \frac{gL^3 \beta \Delta T}{\nu\kappa}$$

Components:

$$-\frac{\partial p}{\partial x} + \nabla^2 u = 0 \qquad (1)$$

$$-\frac{\partial p}{\partial y} + \nabla^2 v = -\mathrm{Ra} \cdot T \qquad (2)$$

Vector product $\nabla \times F$ $\quad \dfrac{\partial(1)}{\partial y} - \dfrac{\partial(2)}{\partial x}$

Ra:	Rayleigh number
Pr :	Prandtl number
p:	Pressure
$\vec{v}$:	Velocity vector (u,v)
T:	Temperature
$\vec{e}_y$:	Unit vector in y direction
ν:	Kinematic viscosity
κ:	Diffusion coefficient
g:	Gravitational acceleration
L:	Characteristic length
β:	Thermal expansion coefficient

Exercise 6

→ Pressure terms cancel out:

$$\nabla^2\left(\frac{\partial u}{\partial y} - \frac{\partial v}{\partial x}\right) = \mathrm{Ra}\cdot\frac{\partial T}{\partial x} \qquad u = \frac{\partial \psi}{\partial y} \qquad v = -\frac{\partial \psi}{\partial x}$$

→ Insert stream function :

$$\nabla^2\left(\frac{\partial^2 \psi}{\partial y^2} + \frac{\partial^2 \psi}{\partial x^2}\right) = \mathrm{Ra}\cdot\frac{\partial T}{\partial x} \quad \rightarrow \quad \nabla^2\nabla^2\psi = \mathrm{Ra}\cdot\frac{\partial T}{\partial x}$$

→ Two Poisson equations (stream function-vorticity formulation)

$$\nabla^2\psi = -\omega \qquad -\nabla^2\omega = \mathrm{Ra}\cdot\frac{\partial T}{\partial x}$$

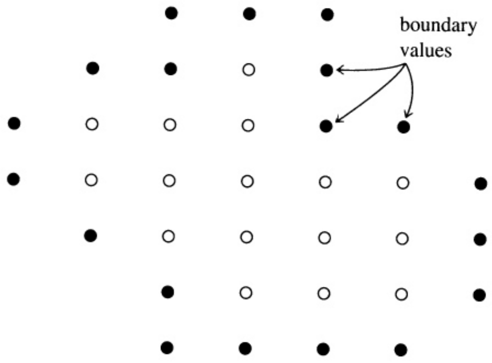
ω: vorticity

Poisson’s equation: a boundary value problem

Siméon Denis Poisson

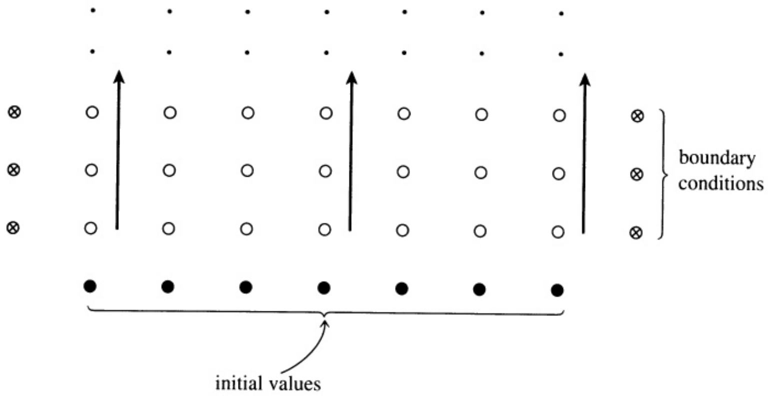
General form (2D): $\nabla^2 u = f$

wanted given



Boundary value problem: the solution is given by the boundary conditions

Initial value problem: the solution is given by the initial conditions (and boundary conditions)



→ We need a Poisson solver

Two ways to solve Poisson's equation numerically

1. Direct
   - Represent discretized equations as a matrix and solve them with a function/subroutine
   - Advantage: Only one (programming) step to the solution
   - Disadvantage: Requires a lot of computing time and memory for large model domains
     - Memory: (nx*ny)^2
     - Computing time: (nx*ny)^3
2. Iterative
   - Start with an initial estimate and improve it until the solution is good enough
   - Advantage: Requires less computing time and less memory than direct method
     - Storage space: nx*ny
     - Computing time: nx*ny
   - Disadvantage: Can be slow without multigrid process

Objective of Exercise 6: write an iterative Poisson solver

Example 1D Poisson: $$\frac{\partial^2 u}{\partial x^2} = f$$

Finite differences: $$\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} = f_i$$

Approximation → Residue $$R_i = f_i - \frac{\tilde{u}_{i+1} - 2\tilde{u}_i + \tilde{u}_{i-1}}{h^2}$$

→ Calculate correction for $\tilde{u}_i$

Correct $\tilde{u}_i$

Goal: $R_i = 0$ $\rightarrow \tilde{u}_i^{n+1} = \tilde{u}_i^n - \alpha \frac{1}{2} h^2 R_i$

α: Relaxation factor

$\alpha > 1$: Over-relaxation

$\alpha < 1$: Under-relaxation

For $\alpha = 1$:

$$R_i^{n+1} = f_i - \frac{\tilde{u}_{i+1} - 2\left(\tilde{u}_i - \frac{1}{2} h^2 R_i^n\right)_i + \tilde{u}_{i-1}}{h^2}$$

$$= f_i - \frac{\tilde{u}_{i+1} - 2\left(\tilde{u}_i - \frac{1}{2} h^2 \left(f_i - \frac{\tilde{u}_{i+1} - 2\tilde{u}_i + \tilde{u}_{i-1}}{h^2}\right)\right)_i + \tilde{u}_{i-1}}{h^2} = 0$$

Unfortunately, $\tilde{u}_{i+1}$ and $\tilde{u}_{i-1}$ also change, so R_i^{n+1} is not 0 but smaller.

Two iteration methods

- Gauss-Seidel: correct u immediately

```
DO (all grid points)
        res = ...
        u(i) = u(i) + f(res)
END DO
```

Advantage of Gauss-Seidel:

- Converges faster
- Residue does not have to be stored for all points

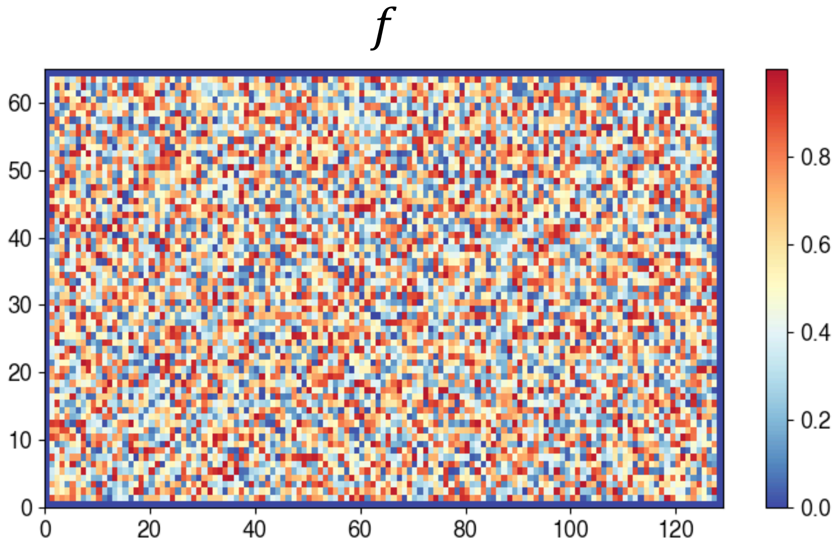
→ We will use Gauss-Seidel

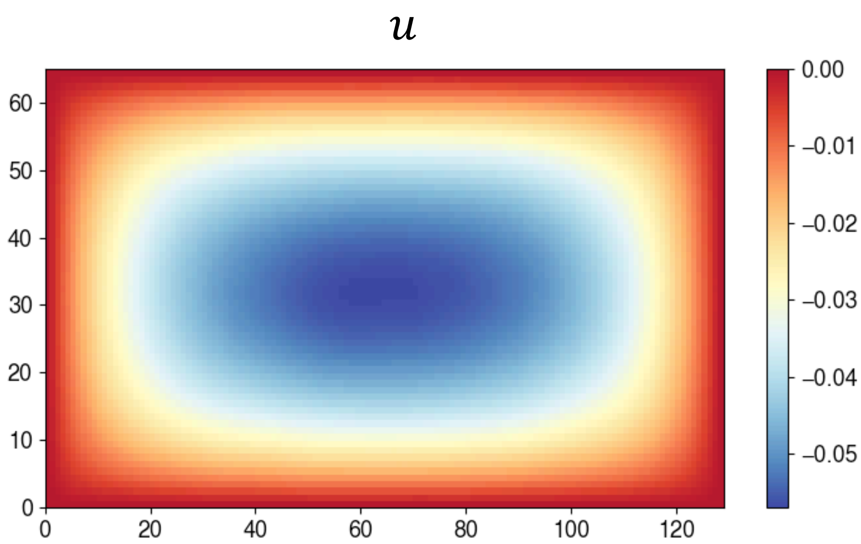
- Jacobi: first calculate residue for all points, then correct u

```
DO (all grid points)
        res(i) = ...
END DO
DO (all grid points)
        u(i) = u(i) + f(res(i))
END DO
```

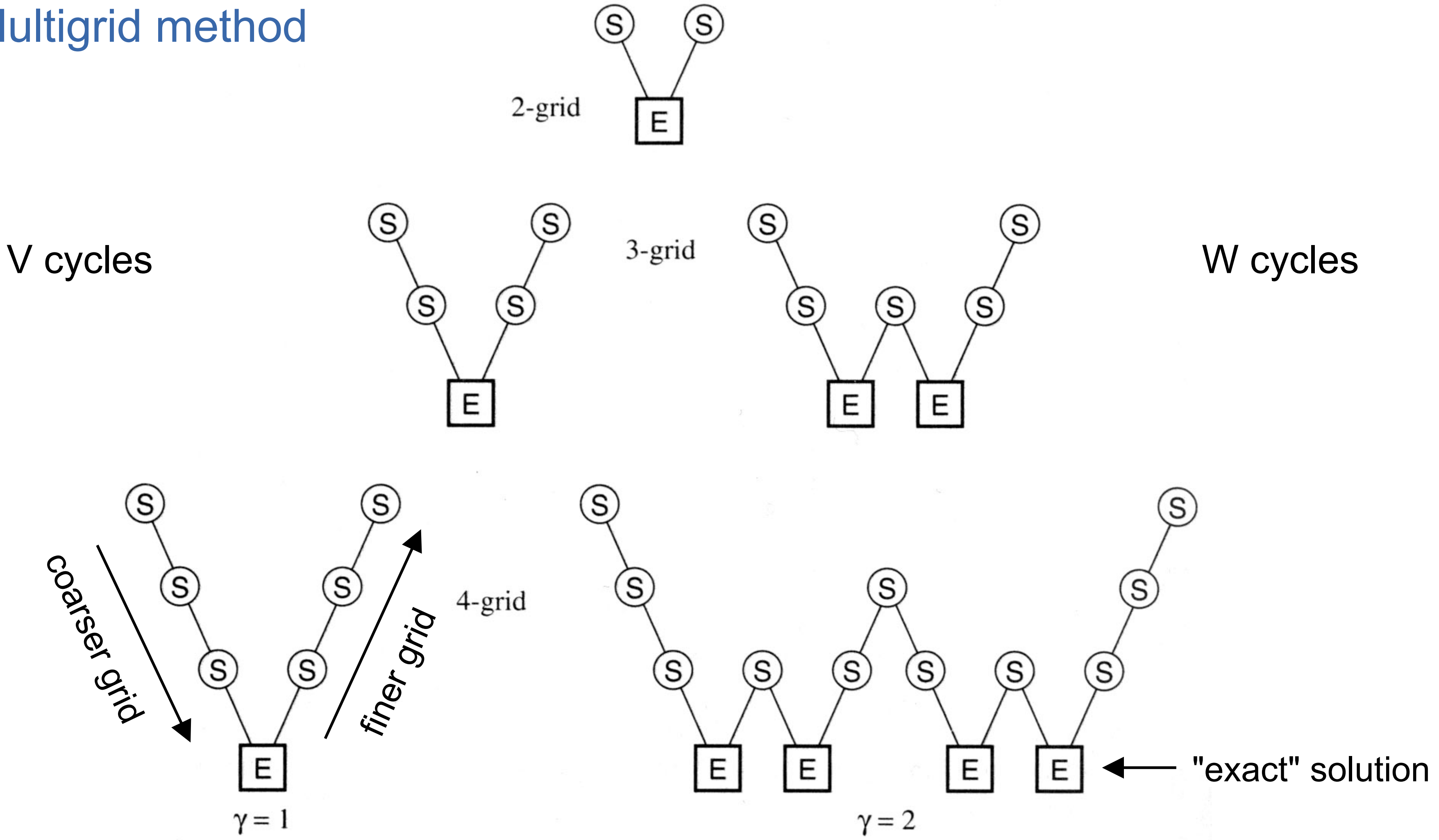
Example

$$\nabla^2 u = f$$





Multigrid method



2D Poisson solver with multigrid method

1. Initial estimate $\tilde{u} = 0$
2. Calculate residue: $R = f - \nabla^2 \tilde{u}$
3. Find correction e for $\tilde{u}$ such that $R \approx 0$: $e = u - \tilde{u}$

$$\nabla^2(\tilde{u} + e) = f$$

$$\nabla^2 e = f - \nabla^2 \tilde{u} = R$$

e becomes u and R becomes f

4. Transfer f to the coarser grid

5. On the coarsest grid (ny=5): Iterate until $R \approx 0$

Exercise 6

Write a Fortran program that iteratively solves the two-dimensional Poisson equation using the multigrid method:

$$\nabla^2 u = f$$

1. On page 3 (and on Moodle) you can find a recursive function called `Vcycle_2DPoisson` that performs the steps of the V cycle. The function calls itself to calculate the correction on the next coarser grid. It also calls other functions and subroutines that have the following tasks, but are still missing:

Exercise 6

- The function `iteration_2DPoisson(u,f,h,alpha)` performs a single Gauss-Seidel iteration on u:

$$R = f_{i,j} - \nabla^2 \tilde{u}_{i,j}$$

$$\tilde{u}_{i,j} = \tilde{u}_{i,j} - \alpha R \frac{h^2}{4}$$

- The subroutine `residue_2DPoisson(res)` calculates the residue $R_{i,j} = f_{i,j} - \nabla^2 \tilde{u}_{i,j}$
- The subroutine `restrict(fine,coarse)` copies every second grid point in `fine` to `coarse`
- The subroutine `prolongate(coarse,fine)` copies every grid point in `coarse` to `fine` and interpolates linearly in between to fill the remaining grid points

Your task is to add the missing functions / subroutines and create a Poisson solver module from them together with the function `Vcycle_2DPoisson`.

Exercise 6

2. Write a main program that tests the Poisson solver by running several iterations until the root mean square residue is less than 10^{-5} of *f* (note: 64-bit precision may be needed to satisfy this condition).

   The program should have the option to call either the iteration function directly (without the multigrid method) or the V-cycle function so it is possible to compare the speed of the two methods. It should perform the following steps:

   - Read input variables from a namelist:
     - Number of grid points `nx`, `ny` (should be $2^n + 1$)
     - Initialization `init_type` (random or spike)
     - Switch for `multigrid` (`.TRUE.` or `.FALSE.`)
     - Relaxation factor `alpha`

Exercise 6

- Initialize variables
  - Grid spacing `h=1./(ny-1.)`
  - Array `f` of size `(nx,ny)` with random numbers or delta function
  - Array `u` of size `(nx,ny)` with 0

- Repeatedly call either `iteration_2DPoisson` or `Vcycle_2DPoisson` until the solution converges (according to `res_rms` vs. `f`).

- Write out `f` and `u` for visualization.

Exercise 6

Deadline:

- Please hand in your solutions (.f90 files and plots of f and u) no later than Tuesday, **30 April 2024, 23:59**.

Questions?

- Email me (marina.duetsch@univie.ac.at)
  Or pass by my office (UZA II, 2G551).