

Introduction to the mini-app code (Mini-Project 4)

Olaf Schenk
USI Lugano



The Application

- The code solves a **reaction diffusion** equation known as **Fischer's Equation**

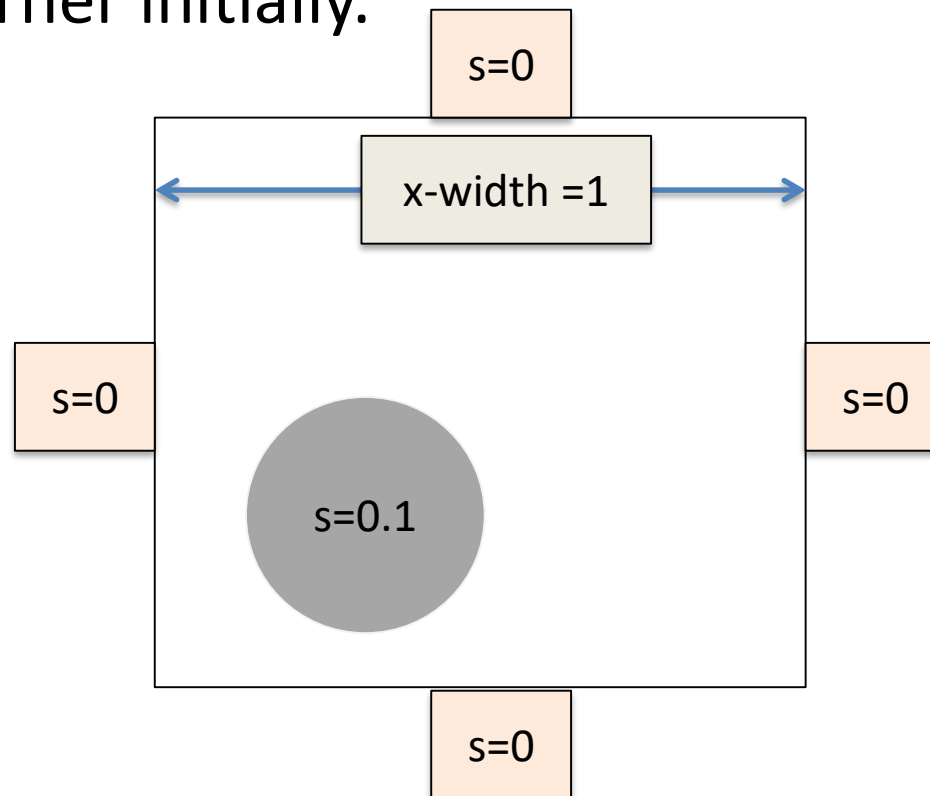
$$\frac{\partial s}{\partial t} = D \overbrace{\left(\frac{\partial^2 s}{\partial x^2} + \frac{\partial^2 s}{\partial y^2} \right)}^{\text{diffusion}} + \overbrace{Rs(1-s)}^{\text{reaction/growth}}$$

- Used to simulate travelling waves and simple population dynamics
 - The species **s** diffuses
 - And the population grows to a maximum of **s=1**



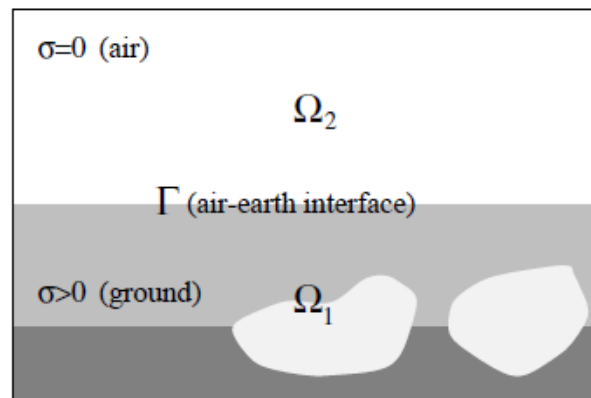
Initial and boundary conditions

- The domain is rectangular, with fixed value of $s=0$ on each boundary, and a circular region of $s=0.1$ in the lower left corner initially.

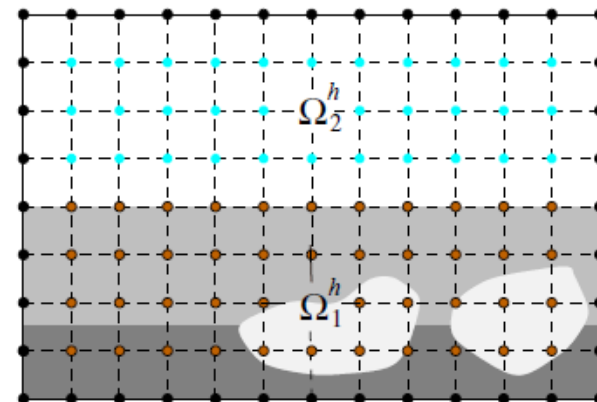


The need to discretize

$$\frac{\partial s(x,y)}{\partial t} = D \left(\frac{\partial^2 s(x,y)}{\partial^2 x} + \frac{\partial^2 s(x,y)}{\partial^2 y} \right) + R s(x,y) [1 - s(x,y)]$$



(a) An earthly domain

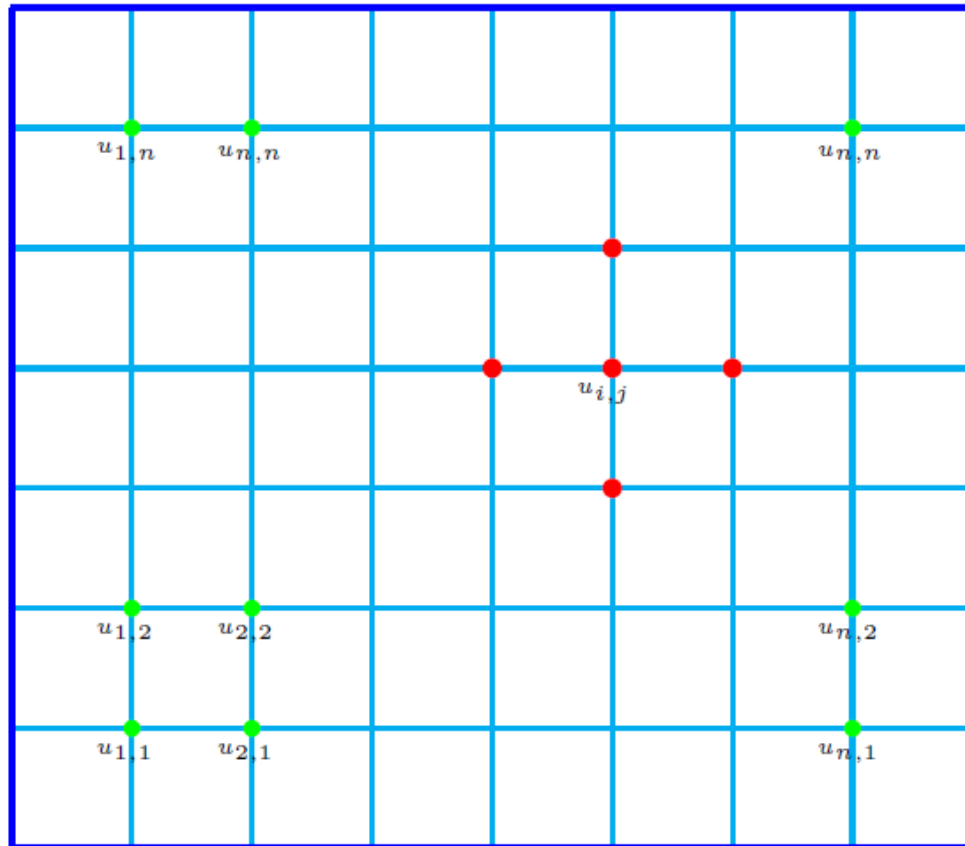


(b) With a discretization grid

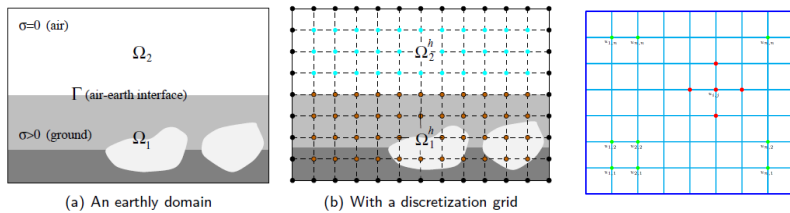
$$\frac{\partial s(x_i, y_i)}{\partial t} \approx \frac{s_{i,j}^{t+1} - s_{i,j}^t}{\Delta t} = \frac{s_{i,j} - s_{i,j}^t}{\Delta t}$$

$$s(x_i, y_i) := s^{t+1}(x_i, y_i) = s_{i,j}^{t+1}$$

The need to discretize



The need to discretize



$$\frac{\partial s(x,y)}{\partial t} = D \left(\frac{\partial^2 s(x,y)}{\partial^2 x} + \frac{\partial^2 s(x,y)}{\partial^2 y} \right) + R s(x,y) [1 - s(x,y)]$$



$$\frac{\partial s(x_i, y_i)}{\partial t} \approx \frac{s_{i,j}^{t+1} - s_{i,j}^t}{\Delta t} = \frac{s_{i,j} - s_{i,j}^t}{\Delta t} \quad s(x_i, y_i) := s^{t+1}(x_i, y_i) = s_{i,j}^{t+1}$$

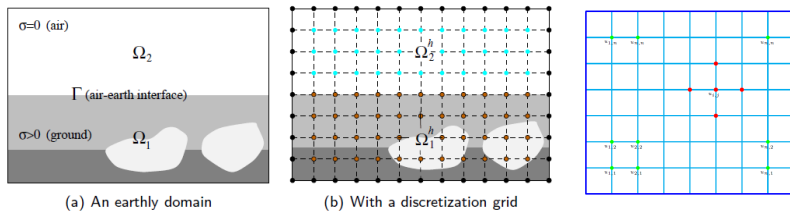
$$\frac{s_{ij} - s_{ij}^t}{\Delta t} = D \left(\frac{\partial^2 s(x,y)}{\partial^2 x} + \frac{\partial^2 s(x,y)}{\partial^2 y} \right) + R s(x,y) [1 - s(x,y)]$$



$$\begin{aligned} \frac{\partial^2 s(x,y)}{\partial^2 x} &\approx \frac{1}{\Delta x^2} (s_{i+1,j} - 2s_{i,j} + s_{i-1,j}) \\ \frac{\partial^2 s(x,y)}{\partial^2 y} &\approx \frac{1}{\Delta y^2} (s_{i,j+1} - 2s_{i,j} + s_{i,j-1}) \end{aligned}$$



The need to discretize



$$\frac{s_{ij} - s_{ij}^t}{\Delta t} = D \left(\frac{\partial^2 s(x,y)}{\partial^2 x} + \frac{\partial^2 s(x,y)}{\partial^2 y} \right) + R s(x,y) [1 - s(x,y)]$$



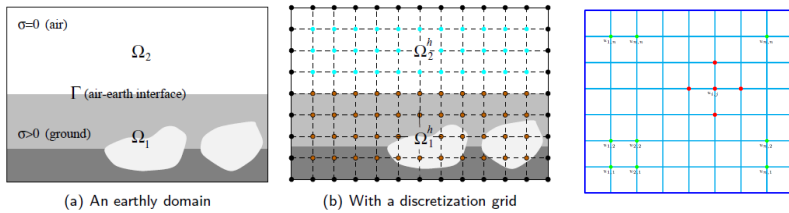
$$\frac{\partial^2 s(x,y)}{\partial^2 x} \approx \frac{1}{\Delta x^2} (s_{i+1,j} - 2s_{i,j} + s_{i-1,j})$$

$$\frac{\partial^2 s(x,y)}{\partial^2 y} \approx \frac{1}{\Delta y^2} (s_{i,j+1} - 2s_{i,j} + s_{i,j-1})$$

$$\frac{s_{ij} - s_{ij}^t}{\Delta t} = D \frac{1}{\Delta^2} (-4s_{i,j} + s_{i+1,j} + s_{i-1,j} + s_{i,j+1} + s_{i,j-1}) + R s_{i,j} [1 - s_{i,j}]$$



The need to discretize



$$\frac{s_{ij} - s_{ij}^t}{\Delta t} = D \frac{1}{\Delta^2} (-4s_{i,j} + s_{i+1,j} + s_{i-1,j} + s_{i,j+1} + s_{i,j-1}) + R s_{i,j} [1 - s_{i,j}]$$

$$D \frac{1}{\Delta^2} (-4s_{i,j} + s_{i+1,j} + s_{i-1,j} + s_{i,j+1} + s_{i,j-1}) + R s_{i,j} [1 - s_{i,j}] + \frac{s_{ij}}{\Delta t} = - \frac{1}{\Delta t} s_{ij}^t$$

$$-4s_{i,j} + s_{i+1,j} + s_{i-1,j} + s_{i,j+1} + s_{i,j-1} + \frac{R\Delta^2}{D} s_{i,j} - \frac{R\Delta^2}{D} s_{i,j}^2 + \frac{\Delta^2}{D\Delta t} s_{ij} = - \frac{\Delta^2}{D\Delta t} s_{ij}^t$$

$$(-4 + \alpha + \beta) s_{i,j} + s_{i+1,j} + s_{i-1,j} + s_{i,j+1} + s_{i,j-1} - \beta s_{i,j}^2 = - \frac{\Delta^2}{D\Delta t} s_{ij}^t$$

Nonlinear!

Solving nonlinear equations

$$\begin{aligned}s_1(x_1, x_2, \dots, x_n) &= 0, \\s_2(x_1, x_2, \dots, x_n) &= 0, \\&\vdots \\s_n(x_1, x_2, \dots, x_n) &= 0.\end{aligned}$$

Or, using vector notation, $\mathbf{s}(\mathbf{x}) = \mathbf{0}$.



Solving nonlinear equations

- Equipped with knowledge on how to **solve nonlinear equations** and **linear systems**, it is time to close the circle and consider systems of nonlinear equation

$$\begin{aligned}s_1(x_1, x_2, \dots, x_n) &= 0, \\s_2(x_1, x_2, \dots, x_n) &= 0, \\&\vdots \\s_n(x_1, x_2, \dots, x_n) &= 0.\end{aligned}$$

Or, using vector notation, $\mathbf{s}(\mathbf{x}) = \mathbf{0}$.



Solving nonlinear equations

Suppose we have one function, f :

- ▶ one variable: $s(x + p) = s(x) + s'(x)p + \frac{1}{2}s''(x)p^2 + \dots$
- ▶ n variables: $s(x + p) = s(x) + g(x)^T p + \frac{1}{2}p^T H(x)p + \dots$

Example: given the function

$$s = x_1^4 - 2x_1^3x_2^2 + 4x_1x_2^3,$$

Gradient:

$$g(x) = \nabla s = \begin{pmatrix} 4x_1^3 - 6x_1^2x_2^2 + 4x_2^3 \\ -4x_1^3x_2 + 12x_1x_2^2 \end{pmatrix};$$

Hessian:

$$H = \nabla^2 s = \begin{pmatrix} 12x_1^2 - 12x_1x_2^2 & -12x_1^2x_2 + 12x_2^2 \\ -12x_1^2x_2 + 12x_2^2 & -4x_1^3 + 24x_1x_2 \end{pmatrix}.$$

We obtain the Taylor expansion using the above formula, for a given $p = \begin{pmatrix} p_1 \\ p_2 \end{pmatrix}$.

m functions, n variables

$$\text{Let } x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}, \text{ function } s = \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_m \end{pmatrix}, \text{ direction } p = \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_n \end{pmatrix}.$$

Assume that s is sufficiently smooth (at least two bounded derivatives). Then the Taylor expansion is

$$s(x + p) = s(x) + J(x)p + \mathcal{O}(\|p\|^2),$$

where $J(x)$ is the **Jacobian** matrix of first derivatives of s at x , given by

$$J(x) = \begin{pmatrix} \frac{\partial s_1}{\partial x_1} & \frac{\partial s_1}{\partial x_2} & \cdots & \frac{\partial s_1}{\partial x_n} \\ \frac{\partial s_2}{\partial x_1} & \frac{\partial s_2}{\partial x_2} & \cdots & \frac{\partial s_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial s_m}{\partial x_1} & \frac{\partial s_m}{\partial x_2} & \cdots & \frac{\partial s_m}{\partial x_n} \end{pmatrix}.$$

Newton's method

By Taylor series,

$$s(\mathbf{x}) = s(\mathbf{x}_k) + J(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k) + \mathcal{O}(\|\mathbf{x} - \mathbf{x}_k\|^2).$$

For $\mathbf{x} = \mathbf{x}^*$, also $s(\mathbf{x}) = 0$.

Neglect nonlinear term and define method by

$$0 = s(\mathbf{x}_k) + J(\mathbf{x}_k)(\mathbf{x}_{k+1} - \mathbf{x}_k).$$

Given an initial guess \mathbf{x}_0 :

for $k = 0, 1, \dots$, until convergence

 solve $J(\mathbf{x}_k)p = -s(\mathbf{x}_k)$,

 set $\mathbf{x}_{k+1} = \mathbf{x}_k + p$.

end



Newton's method

By Taylor series,

$$s(\mathbf{x}) = s(\mathbf{x}_k) + J(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k) + \dots$$

For $\mathbf{x} = \mathbf{x}^*$, also $s(\mathbf{x}) = 0$.

Neglect nonlinear term and define \mathbf{p}

$$0 = s(\mathbf{x}_k) + J(\mathbf{x}_k)(\mathbf{x}_{k+1} - \mathbf{x}_k).$$

Given an initial guess \mathbf{x}_0 :

for $k = 0, 1, \dots$, until convergence

 solve $J(\mathbf{x}_k)\mathbf{p} = -s(\mathbf{x}_k)$,

 set $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{p}$.

end

```
// main timeloop
for (int timestep = 1; timestep <= nt; timestep++) {
    // set x_new and x_old to be the solution
    ss_copy(x_old, x_new, N);
    for (it=0; it<50; it++) {
        // compute residual : requires both x_new and x_old
        diffusion(x_new, b);
        residual = ss_norm2(b, N);
        // check for convergence
        if (residual < tolerance) { converged = true; break; }
        // solve linear system to get -deltax
        ss_cg(deltax, b, 200, tolerance, cg_converged);
        // check that the CG solver converged
        if (!cg_converged) break;
        // update solution
        ss_axpy(x_new, -1.0, deltax, N);
    }
    iters_newton += it+1;
}
```



Newton's method

By Taylor series,

$$s(\mathbf{x}) = s(\mathbf{x}_k) + J(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k) + \mathcal{O}(\|\mathbf{x} - \mathbf{x}_k\|^2).$$

For $\mathbf{x} = \mathbf{x}^*$, also $s(\mathbf{x}) = 0$.

Neglect nonlinear term and define method

$$0 = s(\mathbf{x}_k) + J(\mathbf{x}_k)(\mathbf{x}_{k+1} - \mathbf{x}_k).$$

Given an initial guess \mathbf{x}_0 :

for $k = 0, 1, \dots$, until convergence

 solve $J(\mathbf{x}_k)p = -s(\mathbf{x}_k)$,

 set $\mathbf{x}_{k+1} = \mathbf{x}_k + p$.

end

```
// the interior grid points
```

```
#pragma omp parallel for
```

```
for (int j=1; j < jend; j++) {
```

```
    for (int i=1; i < iend; i++) {
```

```
        S(i,j) = -(4. + alpha) * U(i,j) // central point
```

```
        + U(i-1,j) + U(i+1,j) // east and west
```

```
        + U(i,j-1) + U(i,j+1) // north and south
```

```
        + alpha * x_old(i,j)
```

```
        + dxs * U(i,j) * (1.0 - U(i,j));
```

```
    }
```

```
}
```



The CG method

Algorithm: Conjugate Gradient.

Given an initial guess \mathbf{x}_0 and a tolerance tol , set at first $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$, $\delta_0 = \langle \mathbf{r}_0, \mathbf{r}_0 \rangle$, $b_\delta = \langle \mathbf{b}, \mathbf{b} \rangle$, $k = 0$ and $\mathbf{p}_0 = \mathbf{r}_0$. Then:

while $\delta_k > \text{tol}^2 b_\delta$

// the interior grid points

#pragma omp parallel for

for (int j=1; j < jend; j++) {

for (int i=1; i < iend; i++) {

S(i,j) = -(4. + alpha) * U(i,j) // central point
 + U(i-1,j) + U(i+1,j) // east and west
 + U(i,j-1) + U(i,j+1) // north and south
 + alpha * x_old(i,j)
 + dxs * U(i,j) * (1.0 - U(i,j));

}

}

$\mathbf{s}_k = A\mathbf{p}_k$

$\alpha_k = \frac{\delta_k}{\langle \mathbf{p}_k, \mathbf{s}_k \rangle}$

$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$

$\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k \mathbf{s}_k$

$\delta_{k+1} = \langle \mathbf{r}_{k+1}, \mathbf{r}_{k+1} \rangle$

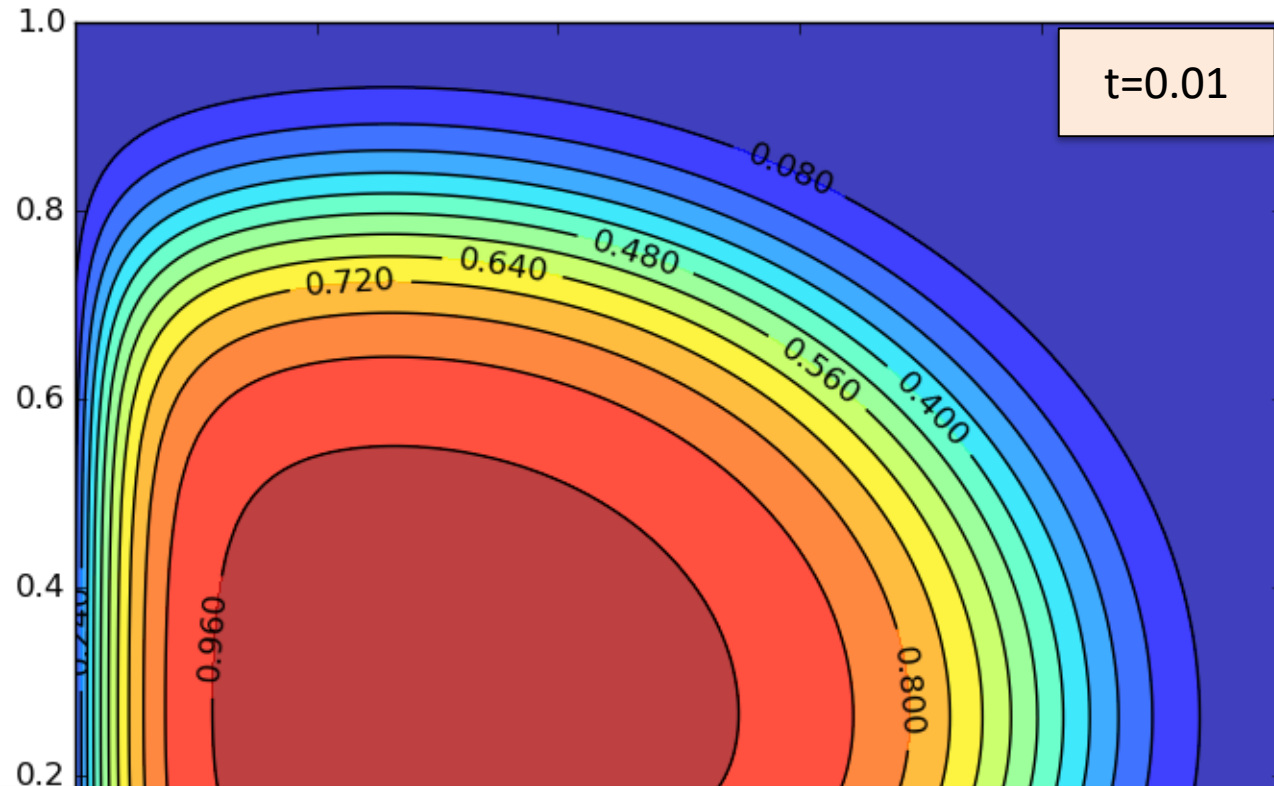
$\mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \frac{\delta_{k+1}}{\delta_k} \mathbf{p}_k$

$k = k + 1$

end



Time Evolution of Solution



- For most cases we will run the solution until $t=0.01$.
- Long enough for something interesting to happen
 - Will clearly show if there is a problem

Numerical Solution

- The rectangular domain is discretized with a grid of dimension $n_x * n_y$ points
- A **finite volume** discretization and **method of lines** gives the follow ordinary differential equation for each grid point

$$\frac{ds_{ij}}{dt} = \frac{D}{\Delta x^2} [-4s_{ij} + s_{i-1,j} + s_{i+1,j} + s_{i,j-1} + s_{i,j+1}] + Rs_{ij}(1 - s_{ij})$$

- Which we can express as the following nonlinear problem...

$$f_{ij} = [-(4 + \alpha) s_{ij} + s_{i-1,j} + s_{i+1,j} + s_{i,j-1} + s_{i,j+1} + \beta s_{ij}(1 - s_{ij})]^{k+1} + \alpha s_{ij}^k = 0$$



Numerical Solution

- One nonlinear equation for each grid point
 - together they form a system of $N=n_x \times n_y$ equations
 - Solve with Newton's method
- Each iteration of Newton's method has to solve a linear system
 - Solve with matrix-free Conjugate Gradient solver
- Solve the nonlinear system at each time step
 - This requires in the order of between 5-10 conjugate gradient iterations



- Don't worry if you don't understand it all!
- We don't need a deep understanding of the mathematics or domain problem to optimize the code
 - I often work on codes with little domain knowledge
- The mini-app has a handful of kernels that can be parallelized
 - And care was taken when designing it to make parallelization as easy as possible
- So let's look a little closer at each part of the code



Code Walkthrough

- There are three modules of interest
 - [main.f90/main.cpp](#) : initialization and main time stepping loop
 - [linalg.f90/linalg.cpp](#) : the BLAS level 1 (vector-vector) kernels and conjugate gradient solver
 - [operators.f90/operators.cpp](#) : the stencil operator for the finite volume discretization

the vector-vector kernels and diffusion operator are the only kernels that have to be parallelized



Linear algebra: linalg.f90/cpp

- This file defines simple kernels for operating on 1D vectors, including
 - dot product : $\mathbf{x} \bullet \mathbf{y}$: `ss_dot`
 - linear combination : $\mathbf{z} = \alpha * \mathbf{x} + \beta * \mathbf{y}$: `ss_lcomb`
- The kernels of interest start with `ss_XXXXX`
 - `ss ==` summer school
- For each parallelization approach that we will see (OpenMP, MPI, CUDA, ... etc), each of these kernels will have to be considered.



Stencil operator: operator.f90/cpp

- This file has a function/subroutine that defines the stencil operator

interior points

```
for j=2:ydim-1
```

```
  for i=2:xdim-1
```

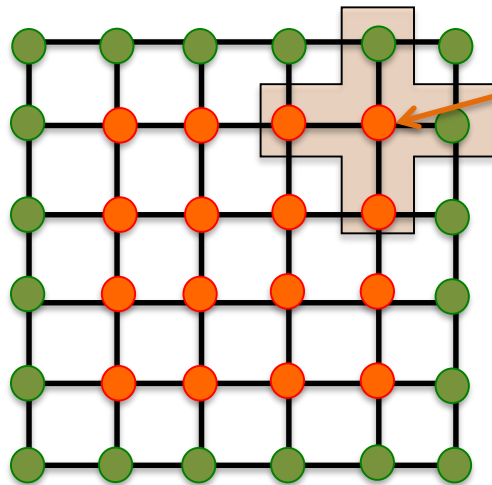
$$f_{ij} = [- (4 + \alpha) s_{ij} + s_{i-1,j} + s_{i+1,j} + s_{i,j-1} + s_{i,j+1} + \beta s_{ij} (1 - s_{ij})]^{k+1} + \alpha s_{ij}^k = 0$$

```
  end
```

```
end
```



Stencil: Interior Grid Points



interior points have all neighbours available

interior points

```
for j=2:ydim-1
```

```
  for i=2:xdim-1
```

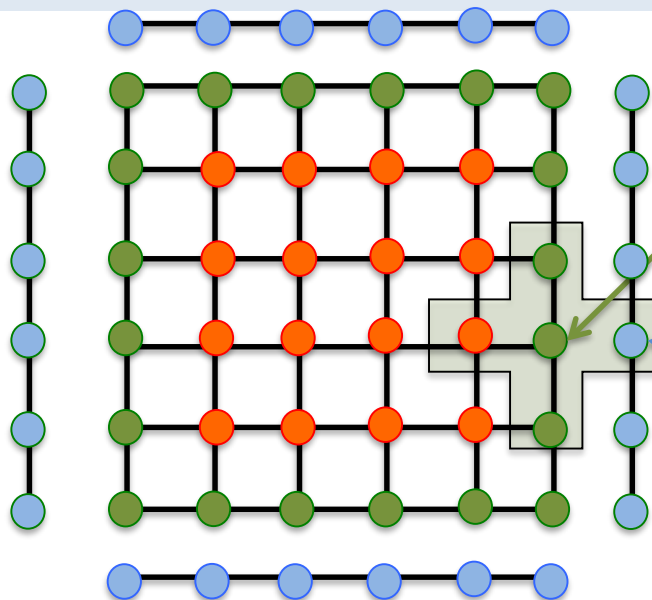
$$f_{ij} = [- (4 + \alpha) s_{ij} + s_{i-1,j} + s_{i+1,j} + s_{i,j-1} + s_{i,j+1} + \beta s_{ij} (1 - s_{ij})]^{k+1} + \alpha s_{ij}^k = 0$$

```
  end
```

```
end
```



Stencil: Boundary Grid Points



boundary points are missing 1 or 2 neighbours

create 4 halo buffers, that hold "ghost" buffers
bndN, bndE, bndS, bndW

east boundary

i=xdim

for j=2:ydim-1

$$f_{ij} = [-(4 + \alpha) s_{ij} + s_{i-1,j} + \text{bndE}_i + s_{i,j-1} + s_{i,j+1} + \beta s_{ij} (1 - s_{ij})]^{k+1} + \alpha s_{ij}^k = 0$$

end

Testing the code

- Get the code, by checking it out from github

```
> git clone https://github.com/rjanalik/HPC_2018.git  
> cd HPC_2018/Miniapp  
> cd cxx
```

I choose the C++ version here

- Compile and run

```
> make  
> srun -n1 ./main 128 128 100 0.01
```



Testing continued...

- Compile

```
> make
```

- Run interactively (use salloc beforehand)

```
> srun ./main 128 128 100 0.01
```

- the grid is 128 x 128 grid points
- take 100 time steps
- run simulation for $t=0.01$

It is possible to choose parameters that will make the simulation fail to converge! The code should tell you gracefully that it was unable to converge.

- Or run batch job

```
> sbatch job.daint
```

```
... when job is finished ...
```

```
> cat job.out
```

increasing the spatial resolution may require increasing the number of time steps