High-**P**erformance **C**omputing 20**25**

Basics of Numerical Methods for PDEs

Aryan Eftekhari | **U**niversità della **S**vizzera **i**taliana (USI) | **I**nstitute of **C**omputing (CI)

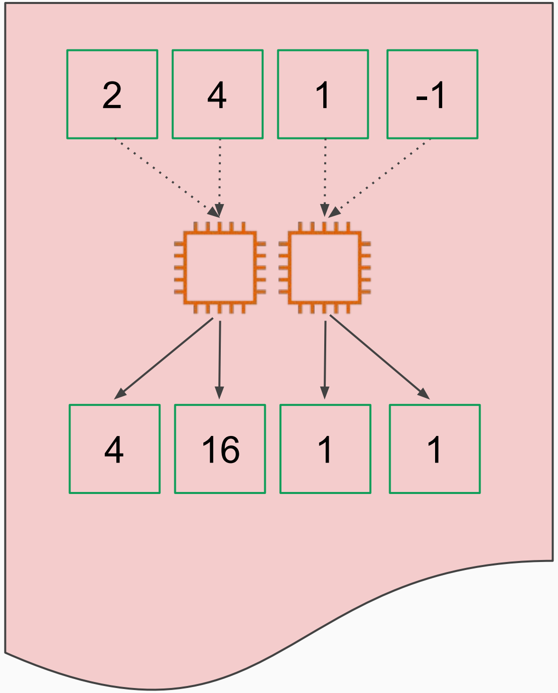
SIMD and **Memory Hierarchy** are fundamental to modern computing systems.

Review
OpenMP

Shared Memory Parallelization

Computation is distributed along ***threads****.*

Synchronization between threads.



OpenMP is easy to use …

```
#include <omp.h>
#include <vector>

int main(){
   std::vector<double> val(1e8,0);
   #pragma omp parallel for
   for (int i = 0; i < val.size(); i++)
      val[i] = COSTLY_OPERATION(i);
   return 0;
}
```

Parallel Region

```
// In Terminal/Command line
// Compile via command line (or makefile)
g++ -fopenmp -O3 main.cpp -o main.exe
// Run
export OMP_NUM_THREADS=2; ./main.exe
```

Overview

A **brief** and **basic** overview

of the numerical methods for solving PDEs.

Differential Equations
Ordinary and Partial

Ordinary Differential Equation (ODE)

Differentiation is with respect to *one variable*.

For example, exponential growth and decay, and Newton's second law of motion.

$$\frac{d^2y}{dx^2} - 3\frac{dy}{dx} + 2y = x^2$$

Partial Differential Equation (PDE)

Differentiation is with respect to *more than one variable*.

For example, heat equation, wave equation, and Fisher's equation.

$$\frac{\partial^2 s}{\partial x^2} + \frac{\partial^2 s}{\partial y^2} = \alpha\frac{\partial s}{\partial t}$$

Multivariate Calculus

Notation and Jargon

Gradient

Differentiation of scalar valued function with respect to a vector.

$$\nabla f = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n}\right]^\top$$

Jacobian

Differentiation of vector valued function with respect to more than one variables (i.e., vector).

$$\mathbf{J}_f = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

Hessian

Second-order differentiation of scalar valued function with respect to more than one variable .

$$\mathbf{H}_f = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

Laplacian (operator)

Divergence of the gradient or vector field (i.e., trace of the Hessian).

$$\Delta f = \sum_i^n \frac{\partial^2 f}{\partial x_i^2}$$

Solution Method (Finite Difference)

Basic Steps

1. **Formulation and representation**

Define the mathematical model of the problem including the domain, **initial condition**, **boundaries conditions**, of the governing equations.*

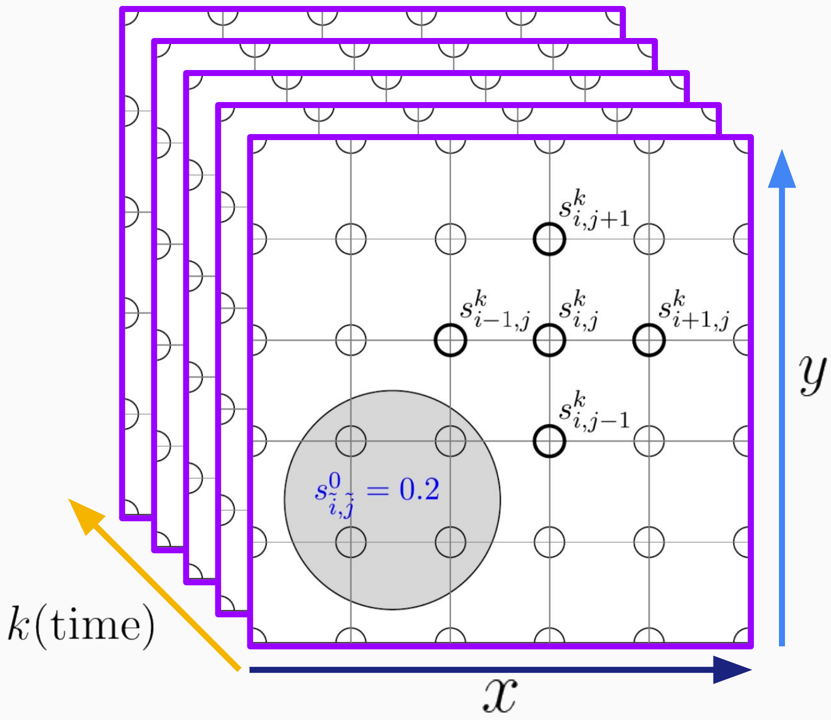
2. **Discretization in space and time**

Convert the continuous problem into a set of discrete equations using a chosen numerical method.

3. **Solve discretized problem in space and time**

Compute the solution of the discrete system over the defined domain.

**consider also stability of the solution.*



Representation of space and time for 2D.

Solution Method (Finite Difference)

Formulation/Discretization, e.g., Fisher's Equation

$$\frac{\partial s}{\partial t} = \delta \Delta s + \rho s(1 - s)$$

Used to describe biological populations: **spatial diffusion** with **reaction/growth**.

$$\frac{1}{\tau}(s_{i,j}^{k} - s_{i,j}^{k-1}) = \underbrace{\frac{\delta}{h^2}\left(-4 s_{i,j}^{k} + s_{i+1,j}^{k} + s_{i-1,j}^{k} + s_{i,j+1}^{k} + s_{i,j-1}^{k}\right)}_{\text{linear}} + \underbrace{\rho s_{i,j}^{k}(1 - s_{i,j}^{k})}_{\text{nonlinear}}$$

Boundary Conditions: What if the (i,j) is at the edge of the grid ?

Initial Condition: We always need the previous solution!

Solution Method (Finite Difference)

Solve, e.g., Fisher's Equation

$$\frac{1}{\tau}(s_{i,j}^{k}-s_{i,j}^{k-1}) = \underbrace{\frac{\delta}{h^2}\left(-4s_{i,j}^{k}+s_{i+1,j}^{k}+s_{i-1,j}^{k}+s_{i,j+1}^{k}+s_{i,j-1}^{k}\right)}_{\textbf{linear}}+\underbrace{\rho s_{i,j}^{k}(1-s_{i,j}^{k})}_{\textbf{nonlinear}}$$

Let $\mathbf{s}^k := [s_{i,j}^{k}, s_{i,j+1}^{k}, s_{i,j+2}^{k}, \ldots, s_{i+1,j}^{k}, s_{i+2,j}^{k}, \ldots]^\top$

The solution is than the <u>**root**</u> of :

$$f(\mathbf{s}^k|\mathbf{s}^{k-1}, \mathbf{A}, c_1, c_2) := \mathbf{s}^k - \mathbf{s}^{k-1} - c_1\mathbf{A}\mathbf{s}^k - c_2\mathbf{s}^k \cdot (1-\mathbf{s}^k)$$

Solution Method (Finite Difference)

Solve, e.g., Fisher's Equation

Newton Iteration–A Method for root finding:

$$\mathbf{s}^k \leftarrow \mathbf{s}^k - [\mathbf{J}_f]^{-1} f(\mathbf{s}^k)$$

Remark: We omit the "given" variables in the notation for clarity.

We need to <u>**solve a linear system**</u> of equations!

$$[\mathbf{J}_f]^{-1} f(\mathbf{s}^k) = \mathbf{x} \iff f(\mathbf{s}^k) = [\mathbf{J}_f]\mathbf{x}$$

Solution Method (Finite Difference)

Pseudo Algorithm

```
Input s_initail_value, K, iter_max, eps

// Initial Conditions
s_last ← s_initail_value
s      ← s_last

// Time loop
For k = 1 to K
      // Newton loop
      For iter=1 to iter_max
            // Linear Solve (will have its own loop)
            update ← lin_solve(J(s|s_last),f(s|s_last))
            s ← s - update
            // Convergence Check
            If norm(update)<eps
            break
            Endif
      Endfor
      // Swap Solution
      s_last ← s
Endfor

Return s
```

Common Options for `lin_solve`:

1. Direct Methods
Solve matrices in fixed steps with notable stability, especially for well-conditioned systems.

2. Iterative Methods
Memory-efficient with *adjustable accuracy*, though they demand careful considerations for stability.

Note: Iterative methods can be implemented in a matrix-free manner and rely on easily parallelizable operations.

Solution Method (Finite Difference)

Implicit vs Explicit

As before, the solution is than the **root** of:

$$f(\mathbf{s}^k|\mathbf{s}^{k-1}, \mathbf{A}, c_1, c_2) := \mathbf{s}^k - \mathbf{s}^{k-1} \underbrace{- c_1\mathbf{A}\mathbf{s}^k - c_2\mathbf{s}^k \cdot (1 - \mathbf{s}^k)}$$

What if we use $\mathbf{s}^{k-1}$ in place of $\mathbf{s}^k$?

… we get an "*explicit method*".

Explicit vs Implicit Methods

- Explicit Methods (in the above)
  No need to solve a system of equations, making them much easier to program.
  The solution can be unstable (may diverge), making them unsuitable for many serious applications.

- Implicit methods (what we showed before)
  Require solving a system of equations, increasing programming complexity.
  The solution is stable, making implicit methods essential for many challenging problems.