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

PDEs (Heat, Poisson & Laplace Equations) and Mathematical Software

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

Poisson, **Laplace**, and **Heat** equations are fundamental for modeling essential physical phenomena.

Their relationship can be enlightening …

Heat, Poisson, and Laplace Equations
Definitions

Heat Equation

Describes the diffusion of a quantity (e.g. heat) in a region over time.

$$\boxed{\frac{\partial u}{\partial t} = \Delta u}$$

Poisson Equation

Describes potential fields influenced by a given source (**source term**).

$$\boxed{\Delta u = \boxed{f}}$$

Written out as ...

$$\frac{\partial u}{\partial t} = \boxed{\alpha}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}\right)$$

A positive coefficient called "thermal diffusivity".

$$\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}\right) = \boxed{f(x, y, z)}$$

*If **source term** is equal to zero, the equation reduces to the "**Laplace's Equation**".*

Heat, Poisson, and Laplace Equations
We have seen this before …

Heat Equation: Can be viewed as a special case of *Fisher's Equation*.

Fisher's Equation (reminder):
Used to describe biological populations:
spatial diffusion with **reaction/growth**.

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

Heat Equation

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

* Notice that "S" is function of space, and time!

Ok but what what about *Poisson*, and *Laplace Equations* …
how are they related?

Consider the Heat Equation with a **source term**

$$\frac{\partial u(x,y,t)}{\partial t} = \alpha \Delta u(x,y,t) + \boxed{f(x,y)}$$

What will the "*steady-state*" solution (i.e. $t \to \infty$) look like ?

Heat, Poisson, and Laplace Equations
Steady-State of Heat Equation

Poisson Equation: Steady-state form of “Heat Equation with a source”.

The **steady-state** solution will follow:

$$\boxed{\frac{\partial u(x,y,t)}{\partial t}} = \alpha \Delta u(x,y,t) + f(x,y) = \boxed{0} \Rightarrow \Delta u(x,y,t) = -\frac{1}{\alpha} f(x,y)$$

In steady-state there is **no change induced by time**:

$$\boxed{\Delta u(x,y,\cancel{t}) = -\frac{1}{\alpha} f(x,y)}$$

Laplace Equation: Steady-state form of “Heat Equation <u>without</u> a source”.

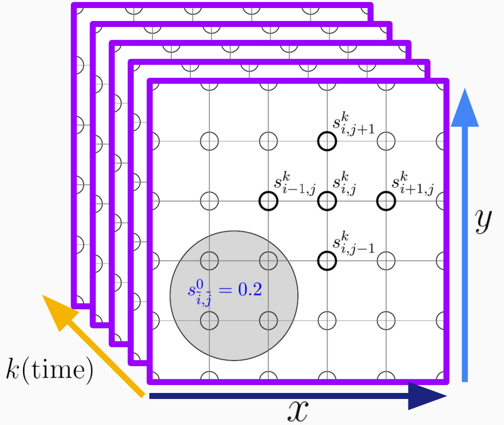
Consider the Heat Equation with **no heat source**:

$$\frac{\partial u(x,y,t)}{\partial t} = \alpha \Delta u(x,y,t) + \cancel{f(x,y)}$$

In steady-state there is **no change induced by time**:

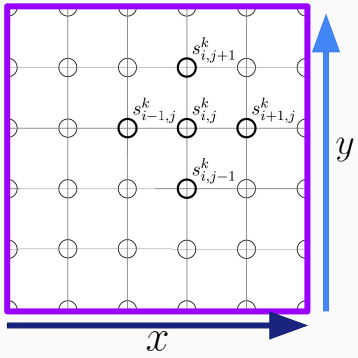
$$\boxed{\Delta u(x,y,\cancel{t}) = 0}$$

Solution Method
Basic Steps



Time-Dependent (e.g., Heat, Fisher's Eq.)

1. **Formulation & Representation:** Model the domain, **initial condition**, and **boundaries conditions**.
2. **Discretization Space & Time:** Convert the continuous problem into a set of discrete equations.
3. **Solve Space & Time:** Solution over domain per time-step.



Time-Independent (e.g., Poisson's and Laplace's Eq.)

1. **Formulation & Representation:** Model the domain and **boundaries conditions**.
2. **Discretization Space:** Convert the continuous problem into a set of discrete equations.
3. **Solve Space:** Solution over domain.

Solution Method
Linear or Nonlinear?

Fisher's Equation (Nonlinear)

** Note we use **s** in place **u**

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

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

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

Use Newton's Method → <u>multiple linear solves</u>

How would this change for the Poisson and Laplace equation?

Solution Method
Linear or Nonlinear?

Poisson's Equation (Linear):

For example consider a forcing function $\sin(x, y)$ which is discretized to $\mathbf{b} = [\sin(x_0, y_0), \sin(x_1, y_0), \ldots, \sin(x_{N-1}, y_{N-1})]$.

The solution is than the root of:

$$\Delta u(x, y) = \sin(x, y) \Rightarrow \; f(\mathbf{u}|\mathbf{A}, c) = \boxed{\mathbf{A}\mathbf{u} - c\mathbf{b}}$$

Or simply <u>single linear solve</u>

… and Laplace's Equation?

Solution Method

Pseudo Algorithm

Nonlinear w. time stepping

```
Input u_initail_value, K, iter_max, eps

// Initial Conditions
u_last ← u_initail_value
u      ← u_last

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

Return u
```

Linear wo. time stepping

```
Input ~~u_initail_value, K, iter_max, eps~~

// Initial Conditions
~~u_last ← u_initail_value~~
~~u      ← u_last~~

~~// Time loop~~
~~For k = 1 to K~~
      ~~// Newton loop~~
      ~~For iter=1 to iter_max~~
            // Linear Solve
            u ← lin_solve(A,b)
            ~~u ← u - update~~
            ~~// Convergence Check~~
            ~~If norm(update)<eps~~
            ~~break~~
            ~~Endif~~
      ~~Endfor~~
      ~~// Swap Solution~~
      ~~u_last ← u~~
~~Endfor~~

Return u
```

Solution Method
Linear Solve

Linear solve is central to the solution of both **linear** and **nonlinear** PDEs.

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

Utilize matrix factorizations (e.g., LU, Cholesky, QR) to reduce the problem to simpler triangular systems, which can then be solved through back-substitution.

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

1. **Conjugate Gradient (CG):** An iterative method for symmetric positive-definite systems of linear equations.
2. **Generalized Minimal Residual (GMRES)**: An iterative method for nonsymmetric (possibly indefinite) system of linear equations.

In the examples we have seen, **CG** is very much applicable.

For more on CG, see "An Introduction to the Conjugate Gradient Method Without the Agonizing Pain"

Toolkits
PETSc

Steps: (**i**) assemble matrices/vectors , (**ii**) solve and (**iii**) maybe repeat

… a toolkit would really help!

Portable, **E**xtensible **T**oolkit for **S**cientific **C**omputation: Scalable (parallel) solution of scientific applications modeled by partial differential equations (PDEs).

Supports **MPI** and GPU acceleration through CUDA, HIP, Kokkos, or OpenCL, as well as hybrid MPI-GPU parallelism

Other frameworks exist such as Trilinos

Toolkits
PETSc Code Structure

```
int main(int argc, char **argv){
 ...
// Boilerplate code for initial setup
PetscCall(KSPCreate(PETSC_COMM_WORLD, &ksp));
PetscCall(KSPSetComputeRHS(ksp, ComputeRHS, &user));
PetscCall(KSPSetComputeOperators(ksp, ComputeMatrix, NULL));
PetscCall(KSPSetDM(ksp, da));
PetscCall(KSPSetFromOptions(ksp));
PetscCall(KSPSetUp(ksp));
...
}

PetscErrorCode ComputeMatrix(KSP ksp, Mat A, Mat P, void *ctx) {
 DM             da;
 DMDALocalInfo info;

 // Retrieve the distributed array, grid information, and global grid dimensions
 PetscCall(KSPGetDM(ksp, &da));
 PetscCall(DMDAGetLocalInfo(da, &info));
 ...
}

PetscErrorCode ComputeRHS(KSP ksp, Vec b, void *ctx) {
 DM             da;
 DMDALocalInfo info;

 // Retrieve the distributed array, grid information, and global grid dimensions
 PetscCall(KSPGetDM(ksp, &da));
 PetscCall(DMDAGetLocalInfo(da, &info));
 ...
}
```

Main Idea:

- “KSP” linear solvers in PETSc.
- **ComputeMatrix** and **ComputeRHS** tells how to assemble the global matrix and RHS vector from the local data owned by each MPI rank.
- You can swap solvers and configurations very easily! *without changing your PDE code.*

Many tutorials available …
Use them!