Physics-Informed Neural Networks for Scientists and Engineers

Pre-enrolment quiz

Problem 1

Approximating an Unknown Function: Suppose $f : \Omega \to \mathbb{R}$ is an unknown function defined on a domain $\Omega \subset \mathbb{R}^d$. You are given a finite collection of observations

$$(x_i, f(x_i))_{i=1}^N,$$

where each $x_i \in \Omega$.

Your goal is to construct an approximation $\widehat{f}(x)$ that predicts the value of the function at any point $x \in \Omega$.

i) Describe at least three different approaches that could be used to approximate the unknown function $f(x)$.

ii) What are the advantages and disadvantages of each approach?

iii) Suppose the dimension d increases from 2 to 20. Which of your methods would become difficult to apply? Why?

iv) Now suppose the observations contain measurement noise. How might this influence your choice of approximation method?

v) Imagine that the function $f(x)$ is known to satisfy certain physical laws, such as a differential equation. How could this additional information improve your approximation?

vi) What properties would an ideal approximation method possess if it were to be used for solving scientific and engineering problems?

Problem 2

Consider the Poisson equation on the unit square $\Omega = [0, 1]^2$:

$$\begin{aligned} -\Delta u &= f, && \text{in } \Omega, \\ u &= g, && \text{on } \partial\Omega, \end{aligned} \tag{1}$$

where f is a given source term, and g is the prescribed Dirichlet boundary condition.

i) Draw the computational domain and divide it into a uniform square grid with mesh size

$$h = \frac{1}{N},$$

where N is a positive integer. Clearly indicate the interior grid points and the boundary grid points.

ii) Let

$$x_i = ih, \qquad y_j = jh,$$

for $i, j = 0, 1, \ldots, N$. Explain why the unknown solution can be approximated by the grid values

$$u_{i,j} \approx u(x_i, y_j).$$

iii) Show that the second derivatives may be approximated by the central difference formulas

$$\frac{\partial^2 u}{\partial x^2}(x_i, y_j) \approx \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2},$$

and

$$\frac{\partial^2 u}{\partial y^2}(x_i, y_j) \approx \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h^2}.$$

iv) Hence derive the five-point finite difference approximation

$$\frac{4u_{i,j} - u_{i-1,j} - u_{i+1,j} - u_{i,j-1} - u_{i,j+1}}{h^2} = f(x_i, y_j),$$

for every interior grid point.

v) Explain how the boundary conditions are incorporated into the finite difference equations.

vi) If there are $(N-1)^2$ interior grid points, how many unknowns must be solved for?

vii) Describe the structure of the resulting linear system

$$A\mathbf{u} = \mathbf{f}.$$

What properties does the matrix A have? (Hint: Think about sparsity, symmetry and positive definiteness.)

viii) Write a Python program to solve the resulting linear system for a simple test problem of your choice. Display the numerical solution using either a contour plot or a surface plot.

Problem 3

The diffusion of heat on a unit square $\Omega = [0,1]^2$ can be modelled by the following partial differential equation:

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

subject to zero boundary conditions and the initial condition

$$U(x, y, 0) = \sin(\pi x) \sin(\pi y).$$

Here $U(x, y, t)$ represents the temperature at time t of a given point $(x, y) \in \Omega$. In this question, you have to write a program to compute the numerical solution for the equation.

i) Draw the computational domain and divide it into a uniform square grid with spatial step size

$$h = \frac{1}{N},$$

where N is a positive integer. Clearly indicate the interior grid points and the boundary grid points.

ii) Introduce a uniform time step

$$\Delta t = \frac{T}{M},$$

where $T > 0$ is the final time and M is the number of time steps. Let

$$t^n = n\Delta t, \qquad n = 0, 1, \ldots, M.$$

Explain how the solution can be approximated by grid values

$$U_{i,j}^n \approx U(x_i, y_j, t^n),$$

where

$$x_i = ih, \qquad y_j = jh.$$

iii) Show that the second derivatives may be approximated at an interior point (x_i, y_j) by the central difference formulas

$$\frac{\partial^2 U}{\partial x^2}(x_i, y_j, t^{n+1}) \approx \frac{U_{i+1,j}^{n+1} - 2U_{i,j}^{n+1} + U_{i-1,j}^{n+1}}{h^2},$$

and

$$\frac{\partial^2 U}{\partial y^2}(x_i, y_j, t^{n+1}) \approx \frac{U_{i,j+1}^{n+1} - 2U_{i,j}^{n+1} + U_{i,j-1}^{n+1}}{h^2}.$$

iv) Explain the backward Euler approximation for the time derivative:

$$\frac{\partial U}{\partial t}(x_i, y_j, t^{n+1}) \approx \frac{U_{i,j}^{n+1} - U_{i,j}^{n}}{\Delta t}.$$

Why is this method called *implicit*?

v) Hence derive the full backward Euler finite difference scheme

$$\frac{U_{i,j}^{n+1} - U_{i,j}^{n}}{\Delta t} = \alpha \left[\frac{U_{i+1,j}^{n+1} - 2U_{i,j}^{n+1} + U_{i-1,j}^{n+1}}{h^2} + \frac{U_{i,j+1}^{n+1} - 2U_{i,j}^{n+1} + U_{i,j-1}^{n+1}}{h^2} \right].$$

vi) Explain how the zero boundary conditions are incorporated into the scheme at every time step.

vii) Show that, at each time step, the backward Euler method leads to a linear system

$$A\mathbf{U}^{n+1} = \mathbf{b}^n.$$

What is known on the right-hand side vector $\mathbf{b}^n$?

viii) Why is the backward Euler method stable for parabolic problems such as the heat equation, especially compared with an explicit time-stepping method?

ix) Write a Python program to solve the heat equation numerically using the backward Euler method and display the solution at several time levels.

Problem 4

Consider the one-dimensional viscous Burgers' equation

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu \frac{\partial^2 u}{\partial x^2}, \qquad -1 \le x \le 1,\ t > 0,$$

subject to the initial condition

$$u(x, 0) = -\sin(\pi x),$$

and the boundary conditions

$$u(-1,t) = u(1,t) = 0.$$

Here, $u(x,t)$ represents the velocity of a fluid, and $\nu > 0$ is the viscosity coefficient.

i) Which term in the equation represents convection (advection), and which term represents diffusion?

ii) Describe qualitatively how the solution is expected to change if the viscosity ν is large or if the viscosity ν is very small.

iii) Suppose you want to solve this problem using a classical numerical method. What additional choices must be made before the computation can begin?

iv) Suppose the viscosity coefficient ν is unknown, but measurements of $u(x,t)$ are available at only a small number of space-time locations. Is this still a forward problem? Explain your answer.

v) Imagine that the solution must be evaluated for thousands of different values of ν. Would repeatedly solving the PDE with a classical numerical solver be the most efficient approach? Explain your reasoning.

Solutions

Problem 1: Approximating an Unknown Function

Let $f : \Omega \to \mathbb{R}$ be an unknown function, with observations $\{(x_i, f(x_i))\}_{i=1}^{N}$.

i) **Three possible approximation approaches.** Common approaches include:

- polynomial interpolation or regression,
- spline interpolation,
- radial basis function (RBF) approximation,
- finite element basis expansions,
- Gaussian process regression,
- neural network approximation.

Any three of these are acceptable.

ii) **Advantages and disadvantages.**

- *Polynomials:* simple and classical, but can oscillate badly for high degree and are not very flexible in high dimensions.
- *Splines:* stable and local, but require knot placement and can become cumbersome in high dimensions.
- *RBF/FEM/GP methods:* flexible and often accurate, but basis selection, mesh generation, or covariance modelling may become expensive as dimension grows.
- *Neural networks:* very flexible and scalable, but require training and careful optimization.

iii) **Effect of increasing dimension from 2 to 20.** Grid-based and mesh-based methods become much harder because the number of degrees of freedom grows rapidly with dimension (the curse of dimensionality). This affects classical interpolation, finite difference, and finite element methods especially strongly. Methods based on sparse sampling or learned function classes are often more attractive in high dimensions.

iv) **Effect of measurement noise.** Noisy data usually calls for approximation methods with regularization or smoothing. Exact interpolation is often undesirable because it may fit the noise rather than the underlying function. One may prefer least-squares regression, smoothing splines, Gaussian processes, ridge regression, or regularized neural networks.

v) **How additional physical laws help.** If f is known to satisfy a differential equation or other physical constraints, then these constraints can be added to the approximation procedure. This reduces the set of admissible functions, acts as a regularizer, and can improve accuracy when data are sparse.

vi) **Ideal properties for scientific computing.** An ideal approximation method should be accurate, stable, scalable, able to handle noise, flexible in high dimension, and able to incorporate physical constraints and boundary data.

Problem 2: Poisson Equation by the Finite Difference Method

Consider

$$-\Delta u = f \quad \text{in } \Omega = [0,1]^2, \qquad u = g \quad \text{on } \partial\Omega.$$

i) **Grid on the unit square.** Let

$$h = \frac{1}{N}, \qquad x_i = ih, \qquad y_j = jh, \qquad i,j = 0,1,\ldots,N.$$

The interior grid points are those with $1 \le i,j \le N-1$, while the boundary grid points are those with $i = 0$ or $i = N$ or $j = 0$ or $j = N$.

ii) **Grid-value approximation.** We approximate the unknown solution by

$$u_{i,j} \approx u(x_i, y_j).$$

This replaces the continuous function by values at the grid nodes.

iii) **Central differences for second derivatives.** Using Taylor expansion about (x_i, y_j),

$$\frac{\partial^2 u}{\partial x^2}(x_i, y_j) \approx \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2},$$

and

$$\frac{\partial^2 u}{\partial y^2}(x_i, y_j) \approx \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h^2}.$$

iv) **Five-point stencil.** Substituting into $-\Delta u = f$ gives, at each interior point,

$$-\left(\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h^2}\right) = f(x_i, y_j),$$

which can be rearranged as

$$\frac{4u_{i,j} - u_{i-1,j} - u_{i+1,j} - u_{i,j-1} - u_{i,j+1}}{h^2} = f(x_i, y_j).$$

v) **Incorporating boundary conditions.** At boundary nodes, $u_{i,j} = g(x_i, y_j)$ is known. When a stencil touches the boundary, those known values are moved to the right-hand side. Thus only interior values remain as unknowns.

vi) **Number of unknowns.** There are $(N-1)^2$ interior grid points, so there are $(N-1)^2$ unknowns.

vii) **Structure of the linear system.** Ordering the interior unknowns into a vector $\mathbf{u}$ yields

$$A\mathbf{u} = \mathbf{f}.$$

For the standard Dirichlet Poisson problem, A is sparse, symmetric, and positive definite. Each row corresponds to a five-point stencil, so each row contains at most five nonzero entries (fewer near the boundary after boundary values are absorbed into the RHS).

viii) **Python program.** A typical implementation assembles the sparse matrix A from the five-point stencil and solves with a sparse linear solver such as `scipy.sparse.linalg.spsolve`. One may then reshape the solution vector into an $(N-1) \times (N-1)$ array and display it with a contour plot or surface plot.

Problem 3: Heat Equation by the Backward Euler Method

Consider

$$\frac{\partial U}{\partial t} = \alpha \left(\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} \right), \qquad (x, y) \in [0,1]^2,\ t > 0,$$

with zero boundary conditions and

$$U(x, y, 0) = \sin(\pi x) \sin(\pi y).$$

i) **Spatial grid.** Let

$$h = \frac{1}{N}, \qquad x_i = ih, \qquad y_j = jh, \qquad i, j = 0, 1, \ldots, N.$$

Interior points satisfy $1 \le i, j \le N-1$; boundary points lie on the edges of the square.

ii) **Time grid and grid values.** Let

$$\Delta t = \frac{T}{M}, \qquad t^n = n\Delta t, \qquad n = 0, 1, \ldots, M.$$

The solution is approximated by

$$U_{i,j}^n \approx U(x_i, y_j, t^n).$$

iii) **Central differences in space.** At time level t^{n+1},

$$\frac{\partial^2 U}{\partial x^2}(x_i, y_j, t^{n+1}) \approx \frac{U_{i+1,j}^{n+1} - 2U_{i,j}^{n+1} + U_{i-1,j}^{n+1}}{h^2},$$

$$\frac{\partial^2 U}{\partial y^2}(x_i, y_j, t^{n+1}) \approx \frac{U_{i,j+1}^{n+1} - 2U_{i,j}^{n+1} + U_{i,j-1}^{n+1}}{h^2}.$$

iv) **Backward Euler in time.** The time derivative is approximated by

$$\frac{\partial U}{\partial t}(x_i, y_j, t^{n+1}) \approx \frac{U_{i,j}^{n+1} - U_{i,j}^n}{\Delta t}.$$

This is called *implicit* because the unknown values at time t^{n+1} appear on the right-hand side of the discretized PDE.

v) **Full backward Euler scheme.** Substituting the spatial and temporal approximations gives

$$\frac{U_{i,j}^{n+1} - U_{i,j}^n}{\Delta t} = \alpha \left[\frac{U_{i+1,j}^{n+1} - 2U_{i,j}^{n+1} + U_{i-1,j}^{n+1}}{h^2} + \frac{U_{i,j+1}^{n+1} - 2U_{i,j}^{n+1} + U_{i,j-1}^{n+1}}{h^2} \right].$$

Equivalently, each time step requires solving a linear system for the unknown values U^{n+1}.

vi) **Boundary conditions.** The zero boundary conditions imply

$$U_{i,j}^{n+1} = 0 \quad \text{for all boundary nodes at every time step.}$$

These known values are inserted into the finite difference equations for neighboring interior nodes.

vii) **Linear system at each time step.** After ordering the interior unknowns into a vector $\mathbf{U}^{n+1}$, the backward Euler method yields

$$A\mathbf{U}^{n+1} = \mathbf{b}^n.$$

The right-hand side $\mathbf{b}^n$ is known from the previous time level $\mathbf{U}^n$ and from boundary data. For this problem the boundary data are zero, so $\mathbf{b}^n$ is determined entirely by $\mathbf{U}^n$.

viii) **Why backward Euler is stable.** Backward Euler is an unconditionally stable method for the linear heat equation. Intuitively, diffusion damps high-frequency modes, and the implicit method captures this damping without the restrictive time-step condition required by explicit schemes. By contrast, explicit methods typically need Δt to satisfy a CFL-type restriction, such as $\Delta t << Ch^2$.

ix) **Python program.** A standard implementation assembles the sparse matrix corresponding to the discrete Laplacian, then repeatedly solves

$$A\mathbf{U}^{n+1} = \mathbf{b}^n$$

for $n = 0, 1, \ldots, M-1$, plotting the solution at selected time levels.

Exact solution. Because the initial data are a single sine mode, the exact solution is

$$U(x, y, t) = e^{-2\pi^2 \alpha t} \sin(\pi x) \sin(\pi y).$$

Problem 4: One-Dimensional Viscous Burgers' Equation

Consider

$$u_t + u\, u_x = \nu u_{xx}, \qquad -1 \le x \le 1,\ t > 0,$$

with

$$u(x, 0) = -\sin(\pi x), \qquad u(-1, t) = u(1, t) = 0.$$

i) **Convection and diffusion terms.** The nonlinear convection (advection) term is

$$u\, u_x,$$

and the diffusion term is

$$\nu u_{xx}.$$

ii) **Effect of viscosity.** If ν is large, diffusion dominates and the solution remains smooth. If ν is very small, convection dominates, steep gradients develop, and the solution may form a shock-like layer.

iii) **Additional choices for a classical numerical method.** Before computation, one must choose:

- the spatial discretization (grid and step size),
- the time discretization (time step and final time),

- the numerical scheme (explicit, implicit, or semi-implicit),
- how the nonlinear term uu_x is discretized,
- how the resulting nonlinear algebraic system is solved, if needed.

So the PDE is not enough by itself; one must also specify the numerical method and discretization parameters.

iv) **Unknown viscosity and sparse observations.** If ν is unknown and only a few measurements of $u(x,t)$ are available, this is an *inverse problem* or parameter-identification problem, not a pure forward problem. One seeks to infer the unknown coefficient from partial data and the governing PDE.

v) **Repeated evaluation for many parameter values.** No. If the solution must be evaluated for thousands of different values of ν, repeatedly solving the PDE with a classical numerical solver would usually be expensive. In such settings, a surrogate model, reduced-order model, or physics-informed neural network may be more efficient after an initial training phase.

Overall lesson. The quiz highlights a central theme of the PINNs course: classical numerical methods are excellent for forward PDE simulation, while learning-based methods become attractive when the task involves high-dimensional approximation, inverse problems, sparse data, or repeated queries over parameter space.