PDE-Find: Reconstructing PDEs from data

The objective of this task is to use a regression method *PDE-FIND* [?] to discover the governing time-dependent partial differential equation (PDE) of an unknown system by using measurements of the solution to the PDE.

The *PDE-FIND* method is able to select, from a large library, the correct linear, nonlinear, and spatial derivative terms, resulting in the identification of PDEs from data. Only those terms that are most informative about the dynamics are selected as part of the discovered PDE. Let us assume that the unknown time-dependent PDE is given in the form of

$$u_t = \mathcal{D}(u, u_x, u_{xx}, u_y, u_{yy}, u_{xy}, ..., x, y, ..., t), \tag{1}$$

where subscripts denote partial differentiation, and we assume the solution is a scalar field, i.e. $u(x, y, ..., t) : \mathbb{R}^d \to \mathbb{R}^1$. An example of the operator $\mathcal{D}$ is Burgers' equation, given by $\mathcal{D} = -uu_x + \mu u_{xx}$, where μ is a scalar viscosity coefficient.

Suppose we have n observations of the solution to the PDE at many known coordinates in the domain. PDE-FIND begins by first constructing a column vector, $\mathbf{u} \in \mathbb{R}^{\mathbf{n}}$, containing all of the solution values. Next, similar column vectors are constructed which each compute the value of a possible (linear or non-linear) term in the PDE at each observational point. These column vectors are collected together to form a matrix $\Theta(\mathbf{u}) \in \mathbb{R}^{n \times D}$ of candidate terms in the PDE, where D is the total number of candidate terms, for example

$$\Theta(\mathbf{u}) = \begin{bmatrix} 1 & \mathbf{u} & \mathbf{u}^2 & \mathbf{u}_x & \mathbf{u}\mathbf{u}_x & \dots \end{bmatrix}. \tag{2}$$

Partial derivatives (such as $\mathbf{u}_x$) at each observational point can be estimated in a number of ways. If the observations are on a regular grid, a simple approach is to use finite differences. When the data is noisy, or irregularly spaced, polynomial interpolation can be used. Another approach is to use a neural network to fit the observational data, i.e. train $NN(x, y, ..., t; \theta) \approx u(x, y, t)$ and to estimate derivatives at query points using autodifferentiation.
Given the library of terms, we then assume that the PDE at each point can be written as

$$\mathbf{u}_t = \Theta(\mathbf{u})\xi, \tag{3}$$

where $\xi \in \mathbb{R}^D$ is a column vector of coefficients and each non-zero entry in ξ corresponds to a term in the PDE. It is assumed that the operator $\mathcal{D}$ may be expressed as a sum of a small number of terms (e.g. < 10 terms), which is certainly the case for the PDEs considered here and is widely used in practice. We therefore aim for a **sparse** vector ξ. Note the matrix $\Theta(\mathbf{u})$ must contain all the operators in the unknown PDE, so that the unknown PDE can always be written as a weighted sum of *a few* terms included in it. We require the sparsest vector ξ that satisfies 3 with a small residual.

Solving for ξ simply means solving a (large) linear system. To ensure we learn a sparse ξ, PDE-FIND uses **ridge regression** with hard thresholding (see the reference [?] for the exact method).

Your task: In this **folder** are 3 files containing observations of the solutions of 3 different PDEs. Your task is to predict the governing PDE for each file.

The files are roughly in order of increasing difficulty. Files 1 and 2 contain measurements of a 1+1D PDE (i.e. $u(x,t)$). File 3 contains measurements of a 2+1D PDE, where the solution is a vector field with two components, i.e. $u(x,y,t)$ and $v(x,y,t)$. In this case the PDE is a set of two coupled equations of the form

$$u_t = \mathcal{D}_1(u, u_x, v, v_x, u_{xy}, uv, ...), \tag{4}$$

$$v_t = \mathcal{D}_2(u, u_x, v, v_x, u_{xy}, uv, ...). \tag{5}$$

Thus, for file 3, you must work out how to generalise 3 so that it can represent this coupled PDE (hint: $\mathbf{u}_t$ and ξ should be replaced with matrices instead of column vectors).

Hint: You may assume for all files that the PDE only includes linear and non-linear combinations of the solution components and/or its (mixed) partial derivatives (and not the domain coordinates), and that only up to (and including) third order (mixed) partial derivatives are used. Carry out the following steps: first, write code which estimates mixed partial derivatives of the solution at each observational point. You can either use an interpolation-based, neural network-based, or finite difference-based approach. Then, decide on an appropriate library of possible PDE terms to include, and build the matrix Θ. Finally, either use an existing sparse linear system solver, or write your own solver to solve 3.

Deliverables: In your project report, state your guess of the PDE for each file. Describe how your algorithm works, the size of the library D you use for each file, what convergence issues you encounter, and the possible future extensions you would consider to improve the convergence and/or generality of your method.