Module 01: Introduction to Geophysical Computing

For the modern professional geophysicists, the use of computers and algorithms is ubiquitous in all aspects of the daily routine. Whether using a "canned" commercial geophysical software application to or developing new code to address a particular task, employing/implementing some sort of computational algorithm is almost always at the heart of any geophysical task.

The challenge of canned software

One of the key challenges geophysicsts face when using commercial geophysical software is that it is often difficult to determine what is going on "under the hood". In many instances it may seem that geophysical results are found in the following way:

$$\boxed{\begin{array}{c}\text{Field}\\ \text{Geophysical}\\ \text{Data}\end{array}} \Longrightarrow \boxed{\begin{array}{c}\text{Magic Black}\\ \text{Box of Geophysical}\\ \text{Data Processing}\end{array}} \Longrightarrow \boxed{\begin{array}{c}\text{Top}-\text{notch}\\ \text{Geophysical}\\ \text{Result}\end{array}}$$

Often there are numerous parameters that go into the blackbox, various combinations of which can greatly affect the output results! It can be extremely frustrating for users - especially when the documentation and usage examples cannot be found, and there is no open-source code to help figure it out.

Python, Scientific Libraries and Beyond ...

One of the major goals of this course is to strip away some of the **magic black box** nature of how students are commonly introduced to geophysical computing, and to provide a better conceptual idea of what is going on when software is applied to tackle geophysical problems. That said, it is not our goal to "reinvent the wheel" on absolutely every algorithm because this would take far too long and one would never finish finding solutions to the problem at hand.

The approach we are taking in this course tries to find a middle way. In particular, the notes will focus on the **Python** language and leverage its open-source, diverse and well-validated computational toolkits including **Numpy** and **Scipy** as well as the strong plotting library **Matplotlib**.

However, to fully appreciate the theory behind some of these tools, it is important to code up some algorithms by oneself to help deepen the understanding of just what the algorithm is doing. Moreover, there will be scenarios where one has to develop a full computer program for a specialized task for which no community-based solution exists.

In these cases, Python may not be the optimal language in which to develop and thus there will be the need to include algorithms directly embedded into the code. In particular, when dealing with large-scale 3D geophysical computing problems, it is highly desirable to develop **parallel computing** solutions that allow you to leverage modern computational hardware to solve these problem in a fraction of the time that **serial computing** solutions will take. Having a deeper understanding of the geophysical algoritms will be greatly helpful in these scenarios.

In this course we will begin to explore these more advanced concepts using the student's choice of "low-level" programming language (https://en.wikipedia.org/wiki/High-level_programming) C/C++/F90 and OpenMP (https://www.openmp.org/). Overall, it is our hope that this will assist students in designing better geophysical computing solutions for read-world three-dimensional problems that will facilitate achievement of the student's geophysical research and data processing goals.

Course Aims

The course is intended to provide senior-level undergraduate and first-year graduate students with material that aimed at improving the algorithmic, programming and computing skills, while simultaneously enhancing skills in handling geophysical data sets. In some ways, these skills are those that are not really taught in any one particular geophysics course; however, they are often the "glue" that students need to help stick together disparate components learned in specific courses into a workable solution.

If we were to broadly classify the material presented in this course, it could be broken down into the following three skill sets:

- Computing Skills:
  - **Working in a Linux Environment and Cluster Computing Architecture**
  - **Introduction to Parallelism with OpenMP**
- Numerical Solutions of Geophysical Partial Differential Equations (PDEs):
  - **Elliptical PDEs (Laplace and Poisson Equations)**
  - **Parabolic PDEs (2D Heat Flow)**
  - **Hyperbolic PDEs (2D Acoustic wave equation)**
- Applied Numerical Algorithms and Data Handling
  - **Applied Linear Algebra for Geophysical Problems**
  - **Differentiation and Discretization**
  - **Regression**
  - **Introduction to Machine Learning**

We briefly summarize each of these sections below.

Module 02: Working in a Linux Cluster Environment

We will begin the course trying to establish a common computing framework for working on Mines computer clusters. While many of the computational tools can be developed through **prototyping** on your laptop, as we will see in this course it is often much more preferable to run your final large-scale and parallelized solutions (more later on that) in a cluster environment on HPC nodes with 24+ cores!

Figure 1. The CSM Mio cluster on which you will be doing much of your work.

Working on the CSM Mio cluster requires developing new skills (for some!) about working in a Linux environment; writing and compiling your code in a lower-level language (e.g., C/C++/F90); submitting jobs through a job scheduler (e.g., SLURM); and debugging, validating and optimizing your code to obtain the best results in the least amount of time.

To help facilitate this process, we are going to perform our tasks within the Madagascar (www.ahay.org) framework. While not a perfect environment for doing everything you might want to do in a computational sense, it does offer an extensive API (https://en.wikipedia.org/wiki/Application_programming_interface) that allows one to interact with (i.e., read in and write out) multi-dimensional array data stored in a regularly sampled format (RSF). There are also lots of code examples that provide a good starting point for developing your own applications.

Module 03: Numerical Differentiation / Discretization

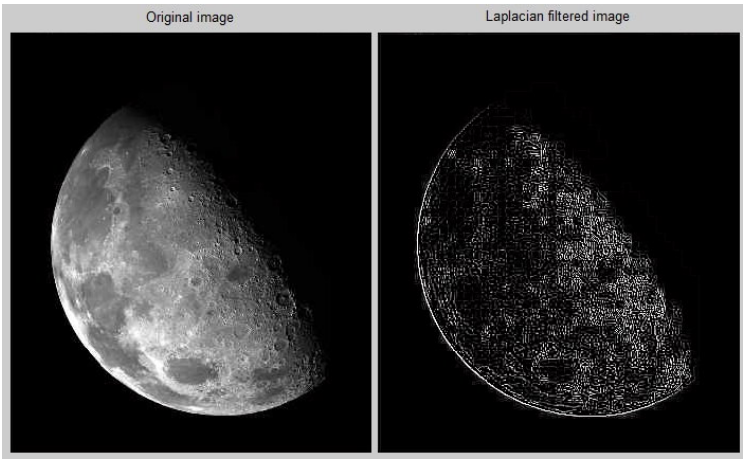
The next module begins our exploration of numerical methods used to solve partial differential equations (PDEs) such as the multi-dimensional advection/convection equation:

$$\mathbf{v} \cdot \nabla \mathbf{u} - \frac{\partial u}{\partial t} = 0 \tag{1}$$

where $\mathbf{v}$ is velocity and u is a scalar field. Examples in nature include the movement of, say, a pollutant in a stream in the absence of any diffusive process.

To solve these equations numerically in a computer, we need to solve PDEs on a discrete solution grid that necessarily requires the process of **discretizing the physical system** including the partial differential operators (e.g., $\frac{\partial^2}{\partial x^2}$) comprising the PDE. This is commonly done using **finite-difference** approximations, which commonly lead to numerical **stencils** and very **efficient** solution algorithms.

For those of you who have taken digital signal processing, you should be familiar with the concept of discretization. To remind you, the left panel below shows an image of the moon. The center panel shows this picture after applying a **convolutional Laplacian filter** for the purpose of emphasizing the discontinuous structure. The right panel shows the **2D finite-difference approximation** of the continuous Laplacian operators ∇^2.



0	−1	0
−1	4	−1
0	−1	0

Figure 2. Moon image before (left) and after (right) applying a numerical filter representing the 2D Laplacian operator, which is a fundamental component of many of the PDEs studied in this course.

However, we will go beyond this to explore how different discretizations affect the **stabilty** and **accuracy** of numerical solution of **time-dependent** PDEs. In this module we will be taking a deeper look at the numerical approximations and discretization with the goal of developing the skills required to solve PDEs of interest like those in Modules 5, 7 and 8.

Module 04 - Introduction to OpenMP

Having solved advection equations in 2D in Python and 3D in your lab assignment, you will quickly recognize the need for speeding up the solvers you developed to obtain your numerical solutions! In this case there is a clear need to move from Python-based serial (or automatically parallelized) solutions to ones where you can leverage modern computer architecture (i.e., the multi-core processor chips) to solve your PDE in **parallel**.

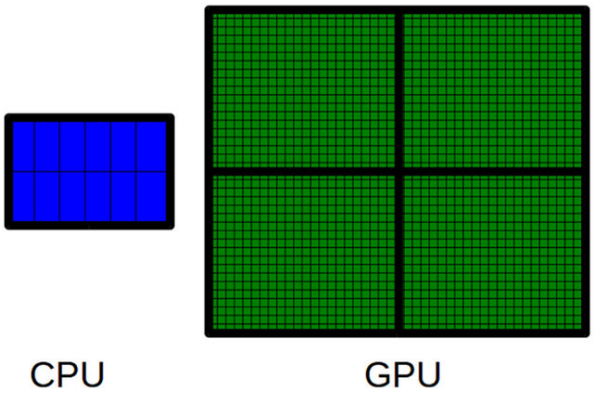


Figure 3. Illustrating modern hardware components in today's clusters, including 12 CPU cores (left) and a GPU comprised of thousands of stripped-down cores (right).

One of the most straightforward ways to do this is through **OpenMP**, which is an extension to C/C++/F90 languages. Combined with modern compilers like gcc/icc or gfortran/iform, these API-like extensions can be used to break up task-parallel components of your code (e.g., loops) so that each of your computer cores can contribute towards generating the solution.

In addition to developing parallel solutions through OpenMP, we will also look at how we can benchmark how we are doing in relation to the idealized speedup (i.e., if you have 24 cores can you do your task 24x faster?!?). We will look at both how your computational solutions scale with the increasing dimensionality of the problem (an example of **strong scaling**) as well as with the increasing number of computer cores (an example of **weak scaling**).

Module 05: Elliptical PDEs

Having tackled issues pertaining to discretization of partial differential operators and thereby PDEs, we are now ready to move on to actually solving some PDEs of interest in geophysics: **elliptical PDEs**. One of the most straightforward 2D PDEs to solve is Poisson's equation for a potential surface $U(x, y)$:

$$\nabla^2 U = \frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} = f(x, y) \tag{2}$$

which reduced to Laplace's equation in the instance where $f(x, y) = 0$ throughout the solution domain. You may recall that one must specify the **boundary conditions** prior to calcuating solutions to these types of equations.

The example below shows the potential surface solution related to the distribution of either four point masses (gravitational potential) or the response to four positive point charges (electrical potential).

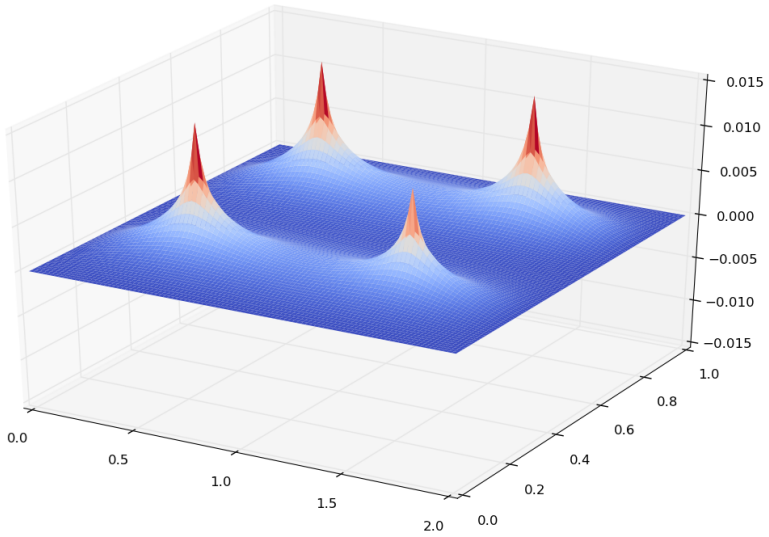


Figure 3. Numerical solution of Poisson's equation solution for four point charges within a domain held at zero potential.

In this module we will explore numerical approaches that will be couched in the framework of - and be used to motivate - applied linear algebra (Module 06).

Module 06 - Applied Linear Algebra for Geophysical Problems

In your undergraduate prepatory career you have probably taken a semester-long course in Linear Algebra. It is equally likely that this course was pretty abstract and didn't really make a strong connection with geophysics. However, applied linear algebra comes up all of the time in numerical geophysical computing - especially in the context of generating the solution to a system of linear equations.

In many situations we will investigate this semester, the geophysical problem can be represeted by a straightforward matrix equation:

$$\mathbf{Gm} = \mathbf{d}$$

where $\mathbf{m}$ is some geophysical **model parameter** (e.g., acoustic wave speed, thermal conductivity, electric charge), $\mathbf{d}$ is some sort of geophysical **data** (e.g., acoustic pressure, heat distibution, electric potential), and $\mathbf{G}$ is a numerical representation of the **physics** and often **experimental geometry** that is used to forward model $\mathbf{d}$ given $\mathbf{m}$. The following figure illustrates this concept using graphical depictions of $\mathbf{G}$, $\mathbf{m}$ and $\mathbf{d}$.

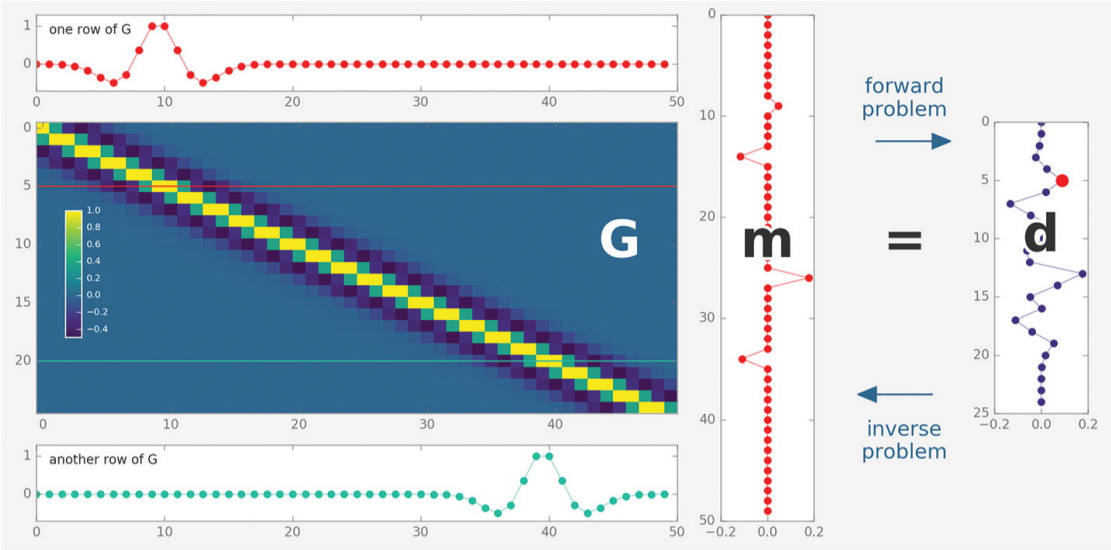


Figure 4. Illustration of a modeling operator (left) acting on a model vector (center) to develop data (right). You will see this paradigm constantly throughout this course!

A good portion of this course looks at how one can solve these types of forward modeling equations in an efficient and straightforward manner using different methods of **applied linear algebra**. We will also examine scenarios where solutions cannot actually be generated due to various numerical challenges.

Module 07: Parabolic PDEs

The next set of PDEs we will study fall into the family of **parabolic** equations. In particular, we are looking to generate numerical solutions of 2D heat flow and 2D diffusion equations, both of which can be written in the following form:

$$\frac{\partial \phi(x,y,t)}{\partial t} = \nabla \cdot \left[D(x,y)\, \nabla \phi(x,y,t) \right] \tag{3}$$

where $\phi(x,y,t)$ is, e.g., the distribution of heat through the 2D solution domain, and $D(x,y)$ is the heterogeneous thermal conductivity field. Computing solutions requires setting both the **initial condition** (i.e., at time $t = 0s$) and the **boundary conditions** at the edges of the computational domain.

The left panels shows example below shows the time evolution of the heat distribution $U(x,y,t)$ for a square computational domain where the four edges are held at 0°C and all interior points initially start at 100°C. The right panel shows a cross-section through the solution and is easier to see how the $U(x, y = 0.5, t)$ evolves over the computed solution time.

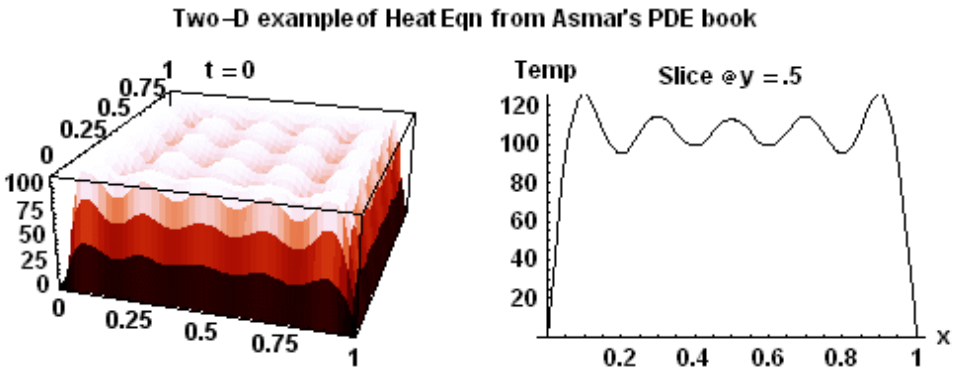


Figure 5. Illustrating of the numerical solution of the 2D heat equation.

In this module we will be developing numerical solutions to the 2D heat flow equation, validating them against analytical solutions to a PDE system, and then using your validated code to solve some interesting geophysical problems for which **no analytical solutions exist**.

Module 08: Hyperbolic PDEs

The final set of PDEs that we will study fall into the family of **hyperbolic** equations. In particular, we are looking to generate numerical solutions of 2D acoustic wave equation, which is written as the following:

$$\left[\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial x^2} - \frac{1}{c^2(x,y)} \frac{\partial^2}{\partial t^2} \right] U(x,y,t) = f(x,y,t) \tag{4}$$

where $U(x,y,t)$ is the displacement of the acoustic wave disturbance, and $c(x,y)$ is the heterogeneous acoustic wave speed, and $f(x,y,t)$ is the force source distribution (e.g., what is causing the acoustic wave disturbance). Computing solutions requires setting the initial conditions and the boundary conditions at the edges of the computational domain.

The example below shows a numerical solution to the 2D acoustic wave equation that is modeling the temporal and spatial evolution of a taut square drum head (i.e., clamped boundaries with zero displacement) and some initial amplitude and/or velocity distribution.

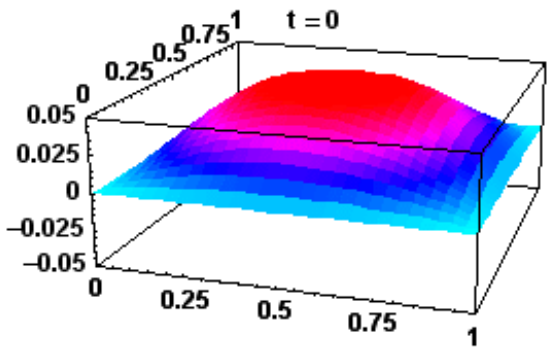


Figure 6. Illustration of the numerical solution of the 2D acoustic wave equation for a taut membrane on a square frame.

In this module we will be developing numerical solutions to the 2D acoustic, validating them against analytical solutions to a PDE system like in the example above, and then using our validated code to solve some interesting geophysical wave propagation problems for which there are no analytical solution.

Module 09: Interpolation

Geophysicists handle data that are acquired at regular and irregular (spatial and/or temporal intervals); however, most of the time the locations where data are acquired are insufficient in number or not in all of the desired locations required to make the corresponding geophysical or geological interpretation.

Let's look a the following example. For simplicitly, let's say that we have 30+ elevation data points that were acquired over a 1 km^2 area. An important question that will no doubt be asked is: *Based on this data, can we determine the elevation profile throughout the entire area?*

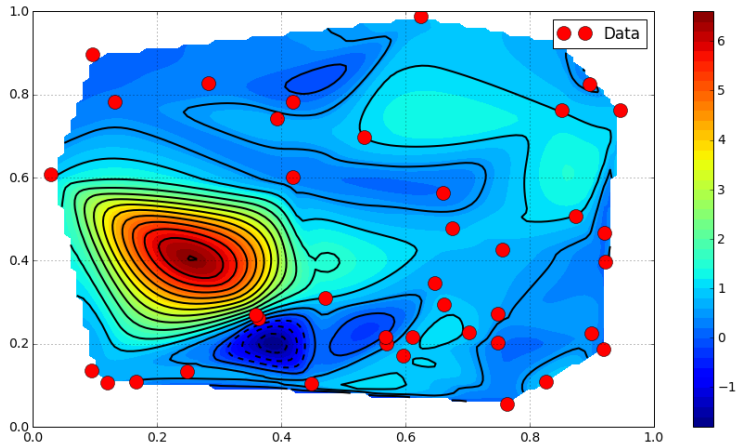


Figure 7. Illustration of an underlying interpolation solution developed from the given points in red.

The good news is that yes we can create high-density maps using the acquired data points; however, the challenge is that there is no one unique map that we can create from the data that would be arguably correct. To do this, we we will explore some of the fundamentals of **interpolation** and see a number of different ways in which plausible maps can be created in 1D and 2D. Many of these algorithms even can be exported to higher dimensions for those cases where 3D, 4D and even 5D interpolation is required; however, these fall beyond the scope of this module.

Module 10: Regression

Measured data contain various kinds of noise, and that's why models (numerical and analytical) can never fit data perfectly. In order to find the model that "best" fits the data, regression algorithms are needed.

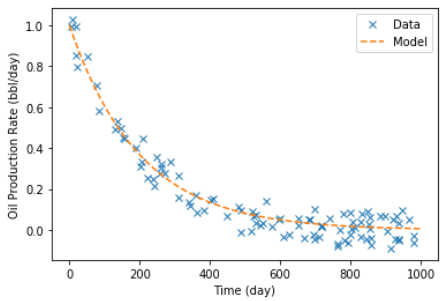


Figure 8: Fitting the production rate of an oil producer using a physical model (exponential decay).

Most regression problems can be presented like this:

- Find m such that $||G(m) - d|| + \lambda\Gamma(m)$ is minimized.

Where G is the model function that links the model parameter m with the observations d. $G(m)$ is the model-predict data, while d is the actually measured one. Symbol $||\ \dots\ ||$ represents some norm of the difference between the predicted data and real data. $\Gamma(m)$ is some regularization term, representing our a priori knowledge of the model.

Two catalogs of regression problems will be discussed in this section: linear and non-linear regression. For the linear regression problem, both model function G and regularization function Γ can be presented as a system of linear functions, and the problem usually has an analytical solution. On the other hand, the non-linear regression problem has to be solved numerically and several algorithms will be introduced.

Module 11: Introduction to Machine Learning

In this chapter, we will introduce some fundamentals of machine learning, as well as the associated python packages.

Machine-learning models, or statistical-learning models, can be considered as highly adaptive empirical functions that project the input data X with output y. In mathematics, it can be presented as:

- Find p such that $||G(X, p) - y||$ is minimized.

where p is the parameters to be "trained". You may find that this formula is quite similar to the regression problem from the last chapter: they are fundamentally the same, except machine-learning algorithms use statistical models instead of physical ones.

We will learn several popular machine learning models and basic machine learning workflow: data processing, feature engineering, hyperparameter selection, and model validation.

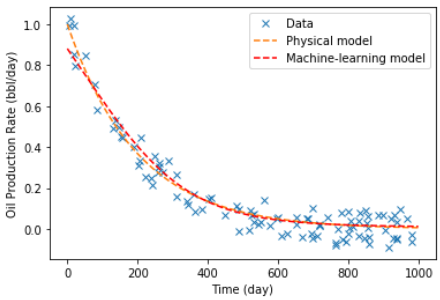


Figure 9: Fitting the production rate of an oil producer using a machine-learning model (multiple-layer neural network).

In [ ]: