GEOS 397: *Computation in the Geosciences*

Prerequisite/co-requisite: PHYS 211-211L (Physics w and w/o calculus), MATH 175 (Calculus II)

Meeting Time: Mon. and Weds., ERB 1100, 12:00 - 1:15 pm

This course aims to provide students with the opportunity to become competent[1] in the following computation areas:

Appropriate Assessment (Color code the following (Wiggins and McTighe, 1991 backward design))

- Big Ideas and Enduring Understanding (performance tasks, open-ended, complex, authentic)
- Important to know and do (typical assessment, selected and constructed responses)
- Worth being familiar with (typical assessment, selected and constructed responses)

Programming and Algorithms

Write code in the MATLAB programming language:

> - Understand the concept of syntax in a programming language
> - Describe the syntax of the programming language constructs
> - List the type of subprograms available in the language
> - Explain the concepts of argument pass-by-value and pass-by-reference
> - Understand the difference between a compiled and interpreted language
> - Understand the difference between a typed and an un-typed language
> - Write and run basic programs in the MATLAB language
> - Understand how to de-bug code and how to "sanity check" code using break points and code debugger
> - Understand the importance of user-interfaces: clear input instructions including physical units if needed and clearly formatted and labeled output
> - Understand the numerical limits of various data types and the implications for numerical accuracy of results
> - Explain the logic behind an if/then/else statement
> - Understand the iterative behavior of loops
> - Describe the difference between several looping constructs (e.g. *do* and *for*)
> - Understand how to document code
> - Use the MATLAB diary to keep track of MATLAB sessions
> - Modify a MATLAB graphic using the 'handle' and/or figure or axes properties
> - Explain the differences in MATLAB class types and gives examples of each

Describe the fundamentals of problem solving:

> - Understand Top-Down thinking and program design
> - Discuss breaking up a problem into its component tasks

- Understand how tasks acquire data
- Describe how tasks should be ordered
- Represent tasks in a flow-chart style format
- Understand the difference between high-level languages (for example Mathematica, Maple or MATLAB), medium level languages (for example FORTRAN or C) and low-level languages (assembler) and when each should be used
- Understand how to modularize code
- Understand the benefits of code re-use
- Write a program based on a story problems

Use subprograms/functions in program design:

- Describe how logical tasks can be implemented as subprograms
- Understand the logical distinction between functions and subroutines
- Explain the control flow when a function is called
- Understand how to represent data flow in and out of subprograms
- Define dummy and actual arguments
- Discuss the different relationships of dummy and actual arguments
- Explain how function output is used
- Understand how languages handle passed data into functions and subprograms, especially one- and two-dimensional arrays

Use different approaches to data I/O in a program:

- Explain the advantages and disadvantages of file I/O
- Describe the syntax for file I/O in the MATLAB programming language
- Compare binary and ASCII file I/O -- low level I/O
- Write code using file I/O and keyboard/monitor I/O
- Load various data file types (e.g. Microsoft Excel, XML, csv)

Understanding and use of programming algorithms:

- Explain an algorithm as an ordered series of solution steps
- Describe an algorithm for a simple programming problem
- Learn and use "classic" programming algorithms from Geoscience problems
- Describe what a software library is and how to access and utilize Mathworks (e.g. File Exchange) and custom libraries
- Understand how library functions implement algorithms
- Write code to implement your own version of a "classic" algorithm
- Compare with code using a library function (e.g. the mean of a vector)
- Understand data flow into library functions and implications of selecting any "tuning parameters" or options that may be required

Numerical Aspects of Problem Solving

Understand number representation and computer errors:

- Understand the pros and cons of floating point and integer arithmetic
- Describe various kinds of computing errors (e.g., round-off, chopping)
- Describe absolute, relative error and percent error
- Discuss error propagation
- Describe loss of significance - methods to avoid loss of significance
- Perform mathematical manipulations of matrices and vectors
- Complex numbers

Describe interpolation and approximation methods:

- Compare and contrast interpolation methods (e.g., Lagrange, Chebyshev, FFT)
- Describe interpolation with spline functions (e.g., piecewise linear, quadratic, natural cubic)
- Discuss approximation using the method of least squares (linear vs. non-linear)

Scientific Visualization

Overview of computer graphic concepts:

- Overview of SciVis concepts (pixels, rgb colors, 3D coordinate system, mapping 3D data to a 2Dscreen, continuous vs. discrete)
- Discuss polygonal representation
- Discuss lighting/shading
- Explore colormaps and examine conceptual definitions for different color maps (pertaining to color spaces HSV, RGB, etc.) as related to representing data and relationships to perception
- Describe and explore the use of different file formats for sharing data (netCDF, XML, TIFF, GIF, JPEG, Wavefront OBJ)
- Make ternary and tetrahedral plots
- Make rose plots
- Make stereonet plots for structural geology interpretation
- Make spider plots

Describe approaches to visualization for different scientific problems:

- Discuss different sources of data for scientific visualization and explain the terms applied to data types (i.e. scalar, vector, normal, tensor)
- Discuss different types of grids (e.g., regular vs. irregular grids)
- Changing from dimensionless "computational units" to physics units and from physical units to dimensionless units.
- Examine different computational solutions to scientific problems
- Explain the different techniques used in visualization (i.e. glyphs, iso-contours, streamlines, image processing, volume-

data)

- Examine the application of problems to visualization techniques
- Utilize MATLAB tools to implement visual image of a solution
- Discuss the use of time in animation

MATLAB Mapping toolbox:

- Display gridded (geo)datasets in multiple formats (e.g. contours, 2D and 3D surfaces)
- Display changes with time
- Access built in MATLAB data sets (e.g. global topography)
- Map projections
- Map objects

MATLAB Date and Time toolbox:

- Date, time and duration objects
- Plot time series data against sample number
- Plot time series data against time in seconds (with a start time of zero, i.e. relative time)
- Plot time series data against absolute time, after extracting the file start time from meta-data
- Date and time arithmetic

MATLAB Statistics toolbox:

- Geodata distributions and Gaussian statistics

MATLAB Signal Processing toolbox:
MATLAB Optimization toolbox:

Parallel Programming

Describe the fundamental concepts of parallel programming:

- Describe the differences between distributed and shared memory architectures
- Describe the difference between domain and functional decomposition in parallel
- Describe a parallel programming approach to an introductory problem
- Compare parallel, distributed, and grid computing concepts
- Use mathematical formulas to determine speed-up and efficiency metrics for a parallel algorithm
- Demonstrate the use of graphical systems such as MATLAB to display speed-up and efficiency graphs Demonstrate the use of performance tools for profiling programs (e.g., MATLAB profiler)

Other

Demonstrate technical communication:

- Demonstrate technical writing skills in the comprehensive report
- Demonstrate verbal communication skills in an oral presentation
- Create and present visual representation of model and results
- Address all components of a comprehensive technical report
- Respond to peer review

Technical Professional Skills

- The basics of a UNIX/LINUX operating system as well as a windows based system (i.e. Windows or OS X or similar)
- Experience with version control, such as Git
- Awareness/Insight into picking an algorithm suitable for the problem
- Understand the limitations of the algorithm; understand that all algorithms have limitations
- Understand accuracy and precision and the difference between them.
- Know how to figure out where your code is "spending its time", and knowing where it is worth spending your time to speed up a code

---

If time

Explain various approaches to program design:

- Define Object-Oriented Programming (OOP)
- Contrast OOP with functional decomposition
- Explain the power of Inheritance in OOP
- GPU vs. CPU

Describe techniques for solving systems of linear equations:

- Describe the naive Gauss elimination and the partial pivoting method
- Understand the concepts of condition number and ill-conditioning problems
- Discuss and contrast factorization methods (e.g., LU, QR, Cholesky, SVD)
- Discuss and contrast iterative methods (e.g., Jacobi, Gauss Siedel)
- Describe convergence and stopping criteria of iterative methods

Analyze techniques for computing eigenvalues/eigenvectors (Optional):

- Describe and give examples of eigenvalue/eigenvector problems using specific, applied examples and their significance
- Describe canonical forms of matrices
- Describe and contrast direct methods for computing eigenvalues (e.g., power method, inverse power method)
- Describe and contrast transformation methods (e.g., QR algorithm)

---

1. Competencies derived from a more comprehensive list at the HPC University ↩