Module 1 : 22.520 Numerical Methods for PDEs : *Course Introduction*

David J. Willis

January 23, 2013

References and Acknowledgements

The following materials were used in the preparation of this lecture:

1. 16.920 Course notes and slides from MIT – Lecture 1
2. L.N. Treffethen and D.Bau III, *Numerical Linear Algebra*, SIAM.
3. G. Strang, *Introduction to Applied Mathematics*, Wellesley-Cambridge Press.

The author of these slides wishes to thank these sources for helping with the current lecture.

Table of contents

Introductions

- **22.520 Numerical Methods for PDEs** : Graduate & Seniors
- **Lecture**: W: 5:00pm-6:15pm
- **Office hours**:
- **Office**: EB-322 (Perry Hall – 322)
- **Email**: david_willis@uml.edu

What is the purpose of this course?

This course is a combination of:

- Mathematics, Engineering, & Computer science
- Specifically, we will cover:
  - **[FDM]** Finite Difference Methods: Theory, Discretization, and Numerical Solutions.
  - **[FVM]** Finite Volume Methods: Theory, Discretization, and Numerical Solutions.
  - **[FEM]** Finite Element Methods: Theory, Discretization, and Numerical Solutions.
  - **[BEM]** Boundary Integral Methods: Theory, Discretization, and Numerical Solutions.
- In the process, you will also learn about:
  - **[SOL]** Numerical methods for solving linear and nonlinear systems.
  - **[GEO]** Discrete Geometry Representation and Mesh Generation

Who should take this course?

ENGINEERS wanting to know useful math. MATHEMATICIANS wanting to know useful application of the math. COMPUTER SCIENTISTS to help us figure out the computer!
Knowing how to use computers to solve problems is a valuable skill in modern engineering and associated fields. This course is particularly valuable if:

- If you are performing research that requires numerical algorithm development
- If you are using computational tools for your research/analysis
- If you are interested in further developing your problem solving, logic and engineering skills
- If you are interested in approaches that allow you to solve complex math problems using computers
- If you want to be the computer's boss! :-).

Grades and Deliverables

- 60% 6-8 homework problems, each with a mandatory computer programming component
- 40% A directed project in an application field (more soon)
  - Homework policy:
    - You are encouraged to discuss homework assignments, projects, class concepts, and general programming together
    - You **may not** code/implement homework problems together.
    - Any computer code that is similar to other students' will result in a grade of zero.
    - All homeworks & projects will require matlab or another programming language (matlab/C++/Fortran etc.)
    - Homework will be due on paper, code by email, by the due date on the assignment.
    - Work hard, apply yourself to the challenge and you can get a great grade

Frequently Asked Questions

Questions?

- *Do I need to type my answers for questions that are not implemented on a computer?*
- *Do I need to type my project proposal and final report?*
- *How do I hand in the portions of the homework that are implemented on a computer?*
- *Do I need to hand in color printouts of matlab (or other coding language) results?*
- *Can I ask for additional computer coding help?*
- *Do I need to know matlab?*
- *What if I know no programming languages?*
- *How is this course diffferent than John White's? MIT's?*
- *Will the lecture videos be available for students to watch?*

Online Materials and Textbook(s)

- MIT opencourseware : 16.920 (and 16.910)
    - http://ocw.mit.edu/
- Suggestions for texts will appear as references in the lecture notes – buy if you want, when you want.
- Recordings of lectures TBD.

Approximate Schedule

- Introduction – today
- Finite difference methods for elliptical, parabolic and hyperbolic equations (4-5 weeks)
- Finite volume methods for conservation laws (2-3 weeks)
- Finite element methods (4-5 weeks)
- Boundary element methods (whatever time is left at the end)

Questions?

Any questions...?

Table of contents

Example PDEs

- PDE: A differential equation involving an unknown function(s) of several variables and their partial derivatives w.r.t those variables.
- Partial differential equations are convenient *models* of physical system behavior – thus they are found everywhere in engineering and mathematics A classical PDE is the convection diffusion equation:

$$\frac{\partial u}{\partial t} + U \cdot \nabla u = \kappa \nabla^2 u + f \tag{1}$$

Where $U, \kappa > 0, f$ are given functions in the space. U is typically a field velocity, f is a source term and κ is a physical parameter related to the physics of the problem.

PDEs : The Convection-Diffusion Equation

$$\frac{\partial u}{\partial t} + U \cdot \nabla u = \kappa \nabla^2 u + f \tag{2}$$

- u = T (temperature) – this is the Heat Transfer Equation
- u = P (polutant concentration) – Coastal/groundwater engineering
- u = p (price of an option) – Financial Engineering
- u = u (momentum per unit mass or velocity) – Navier Stokes Equation

Obviously, the convection-diffusion equation is an important PDE.

PDEs : The Convection-Diffusion Equation: DISCUSSION

PDEs : The Convection-Diffusion Equation: DISCUSSION

PDEs w/ Smooth Solutions

The Poisson Equation (Elliptic):

$$-\kappa \nabla^2 u = f \tag{3}$$

Laplace's Equation (Elliptic):

$$\nabla^2 u = 0 \tag{4}$$

Heat conduction equation (Parabolic):

$$\frac{\partial u}{\partial t} = \kappa \nabla^2 u + f \tag{5}$$

PDEs w/ Smooth Solutions

The Poisson and Laplace's Equations represent a wide variety of physical problems:

- Electrostatics and capacitance extraction
- Potential flow
- Torsion in a bar of constant cross section
- Gravity driven viscous flow in a channel
- Membrane deflection
- Ground water flow
- Stationary heat transfer
- ...

Why is Laplace's/Poisson's equation so common?

Laplace Operator

PDEs w/ Non-smooth Solutions

The 1st Order Wave Equation (Hyperbolic) – single direction of propagation:

$$\frac{\partial u}{\partial t} + U \cdot \nabla u = 0 \tag{6}$$

First Order Wave Equation

First Order Wave Equation

Additional Important PDEs : The Second Order Wave Equation

The wave equation:

$$\frac{\partial^2 u}{\partial t^2} - c^2 \frac{\partial^2 u}{\partial x^2} = 0 \tag{7}$$

This equation describes the propagation of a wave through the domain being considered (bi-directional).

PDEs : The KdV Equation

The KdV (Korteweg - de Vries) equation:

$$\frac{\partial u}{\partial t} + \frac{\partial^3 u}{\partial x^3} - 6u\frac{\partial u}{\partial x} = 0 \tag{8}$$

- Weakly non-linear, shallow water equation, optics signal propagation, etc.

PDEs : The KDV Equation

PDEs : The Hamilton-Jacobi equation and Eikonal equation

The H-J equation:

$$\frac{\partial u}{\partial t} + F(x, y, z)\|\nabla u(x, y)\| = 0 \tag{9}$$

The Eikonal equation:

$$F(x, y, z) \cdot \|\nabla T(x, y)\| = 1 \tag{10}$$

- Use heavily in level sets and path planning.

PDEs : The Hamilton-Jacobi equation and Eikonal equation

Boundary Conditions and Initial Values

Table of contents

Compiled vs. Interpreted Coding Languages

Computer programming languages are classified as:

- Interpreted languages (eg. Matlab, Python, Octave, etc.)
  - Code is saved as written
  - Intructions are sequentially compiled as they are reached
  - Slow run-time
  - Fast to prototype and code (less errors)
  - Usually used for prototyping
- Compiled languages (eg: C/C++/Fortran)
  - Code is converted to machine instructions and saved as an executable
  - Fast run-time $\rightarrow$ just need to run the instructions
  - Slower to prototype and code (more errors)
  - Usually used for production and released codes

Interpreted Coding Languages

Matlab is an interpreted language – designed for vector and matrix algebra. Thus, there are ways to make matlab interpreted code more efficient by making use of built in compiled functions.

- Many of the matrix-matrix, matrix-vector, and vector-vector operations are compiled.
- Writing vectorized code is much more efficient than loop strucutres
- example

Try the following:

- Generate a 1000×1000 matrix, A, and multiply it by a vector, b.
- Use for loops in one case and use the $*$ operator in the other case

We can see that matlabs built-in operations are much better. Try to keep **for**, **while**, etc. loops to a minimum.

Interpreted Coding Languages–Octave

For those who are interested in a 'cheaper' version of Matlab, octave is very similar to matlab (identical but slower for most things):

- http://www.gnu.org/software/octave/

Also, python is not a large departure from matlab and is gaining in popularity.

Discussion of computing hardware

- CPU
- GPU
- RAM
- HD
- ...

Table of contents

Vectors

A vector, is single magnitude, single direction in a multi-dimensional space:

$$V = \begin{pmatrix} V_1 \\ V_2 \\ \vdots \\ V_n \end{pmatrix} \tag{11}$$

- A vector can be used as one of several 'bases' or 'supports' for a given space
- A series of independent 'bases' or 'supports' can be scaled and added to span the entire space.

Vectors

- A 2-D example:
  - A set of vectors that span the 2D space:
  - A set of vectors that do not span the 2D space:

Dot Product or Innner Product

- The dot product of two vectors (a, and b) is a common operation, and is often called the $inner - product$, or $< a \cdot b >$. The result of a dot product is always a scalar.
- For example, $< a \cdot b >$ can be written as:

```
dot_prod = 0;
for i = 1:length(a)
dot_prod = dot_prod + a(i)*b(i);
end
```

- or:

```
dot_prod = a*b'; % this is using vectorization matlab
```

- or:

```
dot_prod = dot(a,b); % this is using internal compiled
```