Lecture 8
Scientific Computing: Symbolic Math, Parallel Computing, ODEs/PDEs

Matthew J. Zahr

CME 292
Advanced MATLAB for Scientific Computing
Stanford University

16th October 2014

Outline

Overview

- Symbolic computations in MATLAB
  - Symbolic variables, expressions, functions
- Mathematics
  - Equation solving, formula simplification, calculus, linear algebra
- Graphics
- Code generation (C, Fortran, Latex)

Symbolic variables, expressions, functions

- Create variables, expressions, functions with `sym`, `syms` commands

```
>> % Symbolic variables
>> syms x, y, z
>> % Symbolic expression
>> phi1 = sym('(1+sqrt(5))/2')
>> phi2 = sym('(1-sqrt(5))/2')
>> phi1*phi2
ans =
-(5^(1/2)/2 - 1/2)*(5^(1/2)/2 + 1/2)
>> simplify(phi1*phi2)
ans =
-1
>> % Symbolic function
>> syms f(u,v)
```

Symbolic matrices

- Symbolic matrices can be constructed from symbolic variables

```
>> syms a b c d
>> A = [a^2, b, c; d*b,c-a,sqrt(b)]
A =
[ a^2,      b,         c]
[ b*d, c - a, b^(1/2)]
>> b = [a;b;c];
>> A*b
ans =
                a^3 + b^2 + c^2
 b^(1/2)*c - b*(a - c) + a*b*d
```

Arithmetic, Relational, and Logical Operations

- Symbolic arithmetic operations
  - `ceil`, `cong`, `cumprod`, `cumsum`, `fix`, `floor`, `frac`, `imag`, `minus`, `mod`, `plus`, `quorem`, `real`, `round`
- Symbolic relational operations
  - `eq`, `ge`, `gt`, `le`, `lt`, `ne`, `isequaln`
- Symbolic logical operations
  - `and`, `not`, `or`, `xor`, `all`, `any`, `isequaln`, `isfinite`, `isinf`, `isnan`, `logical`

`http://www.mathworks.com/help/symbolic/operators.html`

Equation Solving

Command	Description
`finverse`	Functional inverse
`linsolve`	Solve linear system of equations
`poles`	Poles of expression/function
`solve`	Equation/System of equations solver
`dsolve`	ODE solver

Formula Manipulation and Simplification

Command	Description
`simplify`	Algebraic simplification
`simplifyFraction`	Symbolic simplification of fractions
`subexpr`	Rewrite symbolic expression in terms of common subexpression
`subs`	Symbolic substitution

Calculus

Command	Description
`diff`	Differentiate symbolic
`int`	Definite and indefinite integrals
`rsums`	Riemann sums
`curl`	Curl of vector field
`divergence`	Divergence of vector field
`gradient`	Gradient vector of scalar function
`hessian`	Hessian matrix of scalar function
`jacobian`	Jacobian matrix
`laplacian`	Laplacian of scalar function

Calculus

Command	Description
`potential`	Potential of vector field
`vectorPotential`	Vector potential of vector field
`taylor`	Taylor series expansion
`limit`	Compute limit of symbolic expression
`fourier`	Fourier transform
`ifourier`	Inverse Fourier transform
`ilaplace`	Inverse Laplace transform
`iztrans`	Inverse Z-transform
`laplace`	Laplace transform
`ztrans`	Z-transform

Linear Algebra

- Most matrix operations available for numeric arrays also available for symbolic matrices
  - `cat`, `horzcat`, `vertcat`, `diag`, `reshape`, `size`, `sort`, `tril`, `triu`, `numel`

Command	Description
`adjoint`	Adjoint of symbolic square matrix
`expm`	Matrix exponential
`sqrtm`	Matrix square root
`cond`	Condition number of symbolic matrix
`det`	Compute determinant of symbolic matrix
`norm`	Norm of matrix or vector
`colspace`	Column space of matrix

Linear Algebra

Command	Description
`null`	Form basis for null space of matrix
`rank`	Compute rank of symbolic matrix
`rref`	Compute reduced row echelon form
`eig`	Symbolic eigenvalue decomposition
`jordan`	Jordan form of symbolic matrix

Linear Algebra

Command	Description
`chol`	Symbolic Cholesky decomposition
`lu`	Symbolic LU decomposition
`qr`	Symbolic QR decomposition
`svd`	Symbolic singular value decomposition
`inv`	Compute symbolic matrix inverse
`linsolve`	Solve linear system of equations

Assumptions

Command	Description
`assume`	Set assumption on symbolic object
`assumeAlso`	Add assumption on symbolic object
`assumptions`	Show assumptions set on symbolic variable

Polynomials

Command	Description
`charpoly`	Characteristic polynomial of matrix
`coeffs`	Coefficients of polynomial
`minpoly`	Minimal polynomial of matrix
`poly2sm`	Symbolic polynomial from coefficients
`sym2poly`	Symbolic polynomial to numeric

Mathematical Functions

Command	Description
`log`, `log10`, `log2`	Logarithmic functions
`sin`, `cos` `tan`, etc	Trigonometric functions
`sinh`, `cosh` `tanh`, etc	Hyperbolic functions

- Complex numbers and operations also available in Symbolic toolbox
- Special functions
  - Dirac, Haviside, Gamma, Zeta, Airy, Bessel, Error, Hypergeometric, Whittaker functions
  - Elliptic integrals of first, second, third kinds

Precision Control

Command	Description
`digits`	Variable-precision accuracy
`double`	Convert symbolic expression to MATLAB double
`vpa`	Variable precision arithmetic

Functions

Command	Description
`ccode`	C code representation of symbolic expression
`fortran`	Fortran representation of symbolic expression
`latex`	LaTeX representation of symbolic expression
`matlabFunction`	Convert symbolic expression to function handle or file
`texlabel`	TeX representation of symbolic expression

Exercise: Method of Manufactured Solutions

- The *method of manufactured solutions* is a general method for constructing problems with *exact, known* solutions, usually for the purpose of verifying a code.
- Consider the structural equilibrium equations

$$
\begin{aligned}
\nabla \cdot \mathbf{P} + \rho_0 \mathbf{b} &= 0 \\
\mathbf{P} &= \mathbf{S} \cdot \mathbf{F}^T \\
\mathbf{S} &= \lambda \mathrm{tr}(\mathbf{E})\mathbf{I} + 2\mu \mathbf{E} \\
\mathbf{E} &= \frac{1}{2}\left(\mathbf{F}^T\mathbf{F} - \mathbf{I}\right) \\
\mathbf{F} &= \mathbf{I} + \frac{\partial \mathbf{u}}{\partial \mathbf{X}}
\end{aligned}
$$

Exercise: Method of Manufactured Solutions

- For the displacement field, $\mathbf{u}(\mathbf{X}) = [\mathbf{X}_1\mathbf{X}_2\mathbf{X}_3, \mathbf{X}_1^2 + \mathbf{X}_2^2, \sin(\mathbf{X}_3)]^T$, compute the corresponding forcing term
- Generate code in MATLAB, C, and Fortran to compute the forcing term from above

Outline

Programming Parallel Applications

Level of control	*Required effort*
Minimal	*None*
Some	*Straightforward*
Extensive	*Involved*

Programming Parallel Applications

Level of control	*Parallel Options*
Minimal	**Support built into Toolboxes**
Some	**High-Level Programming Constructs: (e.g. parfor, batch, distributed)**
Extensive	**Low-Level Programming Constructs: (e.g. Jobs/Tasks, MPI-based)**

Parallel support built into toolboxes

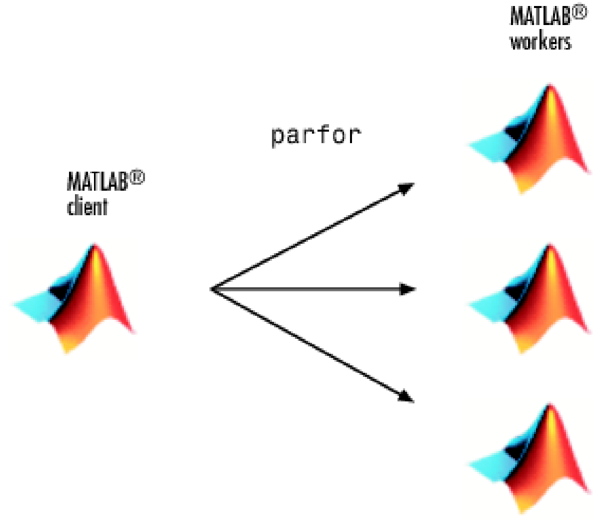
- Parallel Computations available with commands `fmincon`, `fminattain`, `fminimax`
  - Start MATLAB pool of workers
  - Set `UseParallel` option to `'always'`

```
>> matlabpool open 2
>> options = optimset('UseParallel','always');
>> x = fmincon( .., options);
```

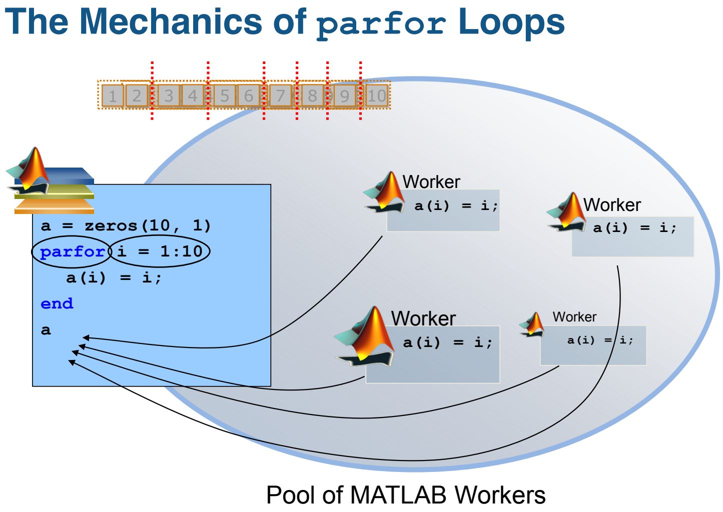
`parfor`

- MATLAB's `parfor` opens a parallel pool of MATLAB sessions (*workers*) for executing loop iterations in parallel
- Requires loop to be embarrassingly parallel
  - Iterations must be *task* and *order independent*
    - Parameter sweeps, Monte Carlo

parfor



parfor



```
a = zeros(10, 1)
parfor i = 1:10
   a(i) = i;
end
a
```

Figure : Courtesy of slides by Jamie Winter, Sarah Wait Zaranek (MathWorks)

Constraints on `parfor` body

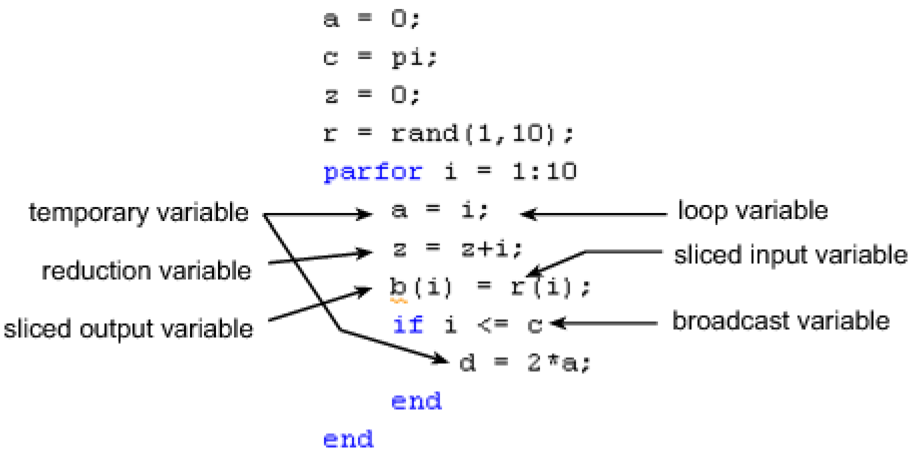
- There are constraints on the body of a `parfor` loop to enable MATLAB to automate the parallelization
  - Cannot introduce variables (`eval`, `load`, `global`, ...)
  - Cannot contain `break` or `return` statements
  - Cannot contain another `parfor` (nested `parfor` loops not allowed)

Parallel variable types

Classification	Description
`Loop`	Loop index
`Sliced`	Arrays whose segments operated on by different iterations
`Broadcast`	Variable defined outside loop (not changed inside)
`Reduction`	Accumulates value across iterations
`Temporary`	Variable created inside loop (not available outside)

`http://www.mathworks.com/help/distcomp/advanced-topics.html`

Parallel variable types



```
a = 0;
c = pi;
z = 0;
r = rand(1,10);
parfor i = 1:10
    a = i;
    z = z+i;
    b(i) = r(i);
    if i <= c
        d = 2*a;
    end
end
```

http://www.mathworks.com/help/distcomp/advanced-topics.html

Demo

`parallel_demo.m`

Outline

Introduction

- A system of *Ordinary Differential Equations* (ODEs) can be written in the form

$$\frac{\partial \mathbf{y}}{\partial t}(t) = \mathbf{F}(t, \mathbf{y})$$
$$\mathbf{y}(0) = \mathbf{y}_0$$

- The concept of *stiffness*
  - An ODE problem is stiff if the solution being sought is varying slowly, but there are nearby solutions that vary rapidly, so a numerical method must take small steps to obtain satisfactory results.
  - Numerical schemes applied to stiff problems have very restrictive time steps for stability

Numerical Solution of ODEs

- Various types/flavor of ODE solvers
  - Multi- vs single-stage
  - Multi- vs single-step
    - Number of time steps used approximate time derivative
  - Implicit vs. Explicit
    - Trade-off between ease of advancing a *single step* versus number of steps required
    - Implicit schemes usually require solving a system of equations
  - Serial vs. Parallel

Fourth-Order Explicit Runge-Kutta (ERK4)

- Multi-stage, single-step, explicit, serial ODE solver
- Consider the discretization of the time domain into $N+1$ intervals $[t_0, t_1, \dots, t_N]$
- At step n, $\mathbf{y}_n$ is known and $\mathbf{y}_{n+1}$ is sought

$$\begin{aligned}
\mathbf{k}_1 &= \mathbf{F}(t_n, \mathbf{y}_n)\\
\mathbf{k}_2 &= \mathbf{F}(t_n + 0.5\Delta t, \mathbf{y}_n + 0.5\Delta t \mathbf{k}_1)\\
\mathbf{k}_3 &= \mathbf{F}(t_n + 0.5\Delta t, \mathbf{y}_n + 0.5\Delta t \mathbf{k}_2)\\
\mathbf{k}_4 &= \mathbf{F}(t_n + \Delta t, \mathbf{y}_n + \Delta t \mathbf{k}_3)\\
\mathbf{y}_{n+1} &= \mathbf{y}_n + \frac{\Delta t}{6}(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4)
\end{aligned} \tag{1}$$

- Fourth-order accuracy: error $= \mathcal{O}(\Delta t^4)$

Requires 4 evaluations of $\mathbf{F}$ to advance single step; does not require solving linear or nonlinear equations

Backward Euler

- Single-stage, single-step, implicit, serial ODE solver
- Consider the discretization of the time domain into $N+1$ intervals $[t_0, t_1, \ldots, t_N]$
- At step n, $\mathbf{y}_n$ is known and $\mathbf{y}_{n+1}$ is sought

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \mathbf{F}(t_{n+1}, \mathbf{y}_{n+1}) \tag{2}$$

- First-order accuracy: error $= \mathcal{O}(\Delta t)$
- A-stable

Requires solving the (nonlinear) system of equations in (2)

MATLAB ODE Solvers

- `[TOUT,YOUT] = ode_solver(ODEFUN,TSPAN,Y0)`
    - Integrates the system of differential equations $y' = f(t, y)$ from time `T0` to `TFINAL` with initial conditions `Y0`
    - `TSPAN = [T0 TFINAL]`
    - `ODEFUN` is a function handle
        - For a scalar `T` and a vector `Y`, `ODEFUN(T,Y)` must return a column vector corresponding to $f(t, y)$
    - Each row in the solution array `YOUT` corresponds to a time returned in the column vector `TOUT`
    - To obtain solutions at specific times `T0,T1,..,TFINAL` (all increasing or all decreasing), use `TSPAN = [T0 T1 .. TFINAL]`

MATLAB ODE Solvers

Command	Type	Accuracy
`ode45`	Nonstiff	Medium
`ode23`	Nonstiff	Low
`ode113`	Nonstiff	Low - High
`ode15s`	Stiff	Low - Medium
`ode23s`	Stiff	Low
`ode23t`	Moderately stiff	Low
`ode23tb`	Stiff	Low

`http://www.mathworks.com/help/matlab/ref/ode45.html`

Assignment

Use `ode45` and `ode23s` to solve the simplified combustion model

$$y'(t) = y^2(1-y), \qquad 0 \leq t \leq 2/\epsilon, \qquad y(0) = \epsilon$$

- Try $\epsilon = 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1$
- How many time steps were required for `ode45` for each epsilon? How many for `ode23s`?
  - `length(TOUT)` using the notation from earlier
- For $\epsilon = 10^{-4}$, plot $y(t)$

Outline

Motivation

- Partial Differential Equations are ubiquitous in science and engineering
  - Fluid Mechanics
    - Euler equations, Navier-Stokes equations
  - Solid Mechanics
    - Structural dynamics
  - Electrodynamics
    - Maxwell equations
  - Quantum Mechanics
    - Schrödinger equation
- Analytical solutions over arbitrary domains mostly unavailable
- In some cases, existence and uniqueness not guaranteed

Numerical Solution of PDEs

- Classes of PDEs
    - Elliptic
    - Parabolic
    - Hyperbolic
- Numerical Methods for solving PDEs
    - Finite Difference (FD)
    - Finite Element (FE)
    - Finite Volume (FV)
    - Spectral (Fourier, Chebyshev)
    - Discontinuous Galerkin (DG)

Numerical Solution of PDEs

The (major) steps required to compute the numerical solution of to a system of Partial Differential Equations are

- Derive discretization of governing equations
  - Semi-discretization
  - Space-time discretization
  - Boundary conditions
- Construct spatial mesh (or space-time mesh)
  - Structured vs. Unstructured
    - Codes can be written to leverage structured mesh
    - Unstructured meshes more general
  - Requirements on mesh heavily depend on application of interest and code used
- If semi-discretized, define temporal mesh
- Implement and solve
- Postprocess

Example: Semi-Discretization

Consider the *viscous* Burger's equation

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \epsilon \frac{\partial^2 u}{\partial x^2} \tag{3}$$

for $x \in [0, 1]$, with the initial condition $u(x, 0) = 1$ and boundary condition $u(0, t) = 5$.

Spatial discretization of (3) yields a system of ODEs of the form

$$\frac{\partial \mathbf{U}}{\partial t} = \mathbf{F}(\mathbf{U}(t), t), \tag{4}$$

this is known as *semi-discretization* or the *method of lines*.

Example: Mesh

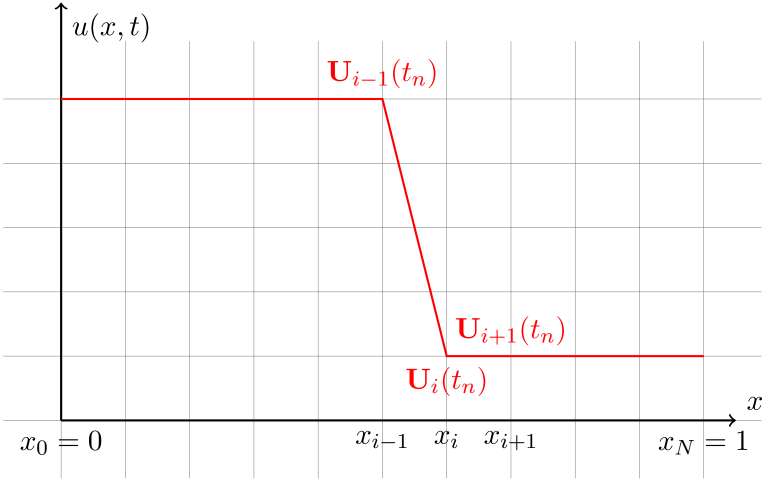


Figure : Discretized domain and solution for (3)

Example: Finite Difference Method

- We approximate the first-order derivative with a *backward* difference on the previous grid to objtain

$$\frac{\partial u}{\partial x}(x_i, t) \approx \frac{u(x_i, t) - u(x_{i-1}, t)}{\Delta x} \tag{5}$$

for $i = 1, \ldots, N$ where $\Delta x_i = x_i - x_{i-1} = \Delta x$ as the grid is assumed uniform.

- The standard central second-order approximation to the diffusive term is applied

$$\frac{\partial^2 u}{\partial x^2}(x_i, t) \approx \frac{u(x_{i+1}, t) - 2u(x_i, t) + u(x_{i-1}, t)}{\Delta x^2} \tag{6}$$

for $i = 1, \ldots, N - 1$

Example: Finite Difference Method

- At the last equation, a first-order, leftward bias of the second-order derivative is applied

$$\frac{\partial^2 u}{\partial x^2}(x_i, t) \approx \frac{u(x_N, t) - 2u(x_{N-1}, t) + u(x_{N-2}, t)}{\Delta x^2}. \tag{7}$$

- The boundary condition is applied as

$$u(x_0, t) = u(0, t) \tag{8}$$

in (5) and (7).

Mesh Generation

In 1D, mesh generation is trivial. Difficulties arise in 2D and higher.

- PDE Toolbox
  - 2D only
- `distmesh`
  - Both 2D (triangles) and 3D (tetrahedra)
  - Unstructured
  - Per-Olof Persson

PDE Toolbox

- Geometry definition (points, curves, surfaces, volumes)
- Mesh generation
- Problem definition
- Solution
- Postprocessing

PDE Toolbox

- Standard MATLAB distribution
  - `pdepe` for solving initial boundary-value problems for *parabolic-elliptic* PDEs in 1D
- PDE Toolbox
  - Graphical User Interface
    - `pdeapp`
    - Demo
  - Command Line

Geometry Definition

- Construct mesh interactively using `pdetool`
    - Unions and intersections of basic shapes
        - Rectangles, ellipses, circles, etc
- Use `pdegeom` to create geometry programmatically
    - Build parametrized, oriented boundary edges
    - Label left and right regions of edges
    - Geometry built from union of regions with similar labels
    - Demo: `naca`

Mesh Generation

Command	Description
`initmesh`	Create initial triangular mesh
`adaptmesh`	Adaptive mesh generation and PDE solution
`jigglemesh`	Jiggle internal points of triangular mesh
`reinemesh`	Refine triangular mesh
`tri2grid`	Interpolate from PDE triangular mesh to rectangular grid
`pdemesh`	Plot PDE triangular mesh
`pdetriq`	Triangle quality measure

```
>> [p,e,t]=initmesh('naca');
>> pdemesh(p,e,t), axis equal
```

Problem Definition: PDE

- Both scalar and vector PDEs available in PDE toolbox
- Here we focus on scalar PDEs
  - Elliptic
$$-\nabla \cdot (c\nabla u) + au = f \tag{9}$$
  - Parabolic
$$d\frac{\partial u}{\partial t} - \nabla \cdot (c\nabla u) + au = f \tag{10}$$
  - Hyperbolic
$$d\frac{\partial^2 u}{\partial t^2} - \nabla \cdot (c\nabla u) + au = f \tag{11}$$
  - Eigenvalue
$$-\nabla \cdot (c\nabla u) + au = \lambda du \tag{12}$$
- PDE coefficients a, c, d, f can vary with space and time (can also depend on the solution u or the edge segment index)

Problem Definition: Boundary Conditions

- Boundary conditions available for both scalar and vector available in PDE toolbox
- Here we focus on scalar PDEs
    - Dirichlet (essential) boundary conditions

$$hu = r \text{ on } \partial\Omega \tag{13}$$

    - Generalized Neumann (natural) boundary conditions

$$\mathbf{n} \cdot (\nabla u) + qu = g \text{ on } \partial\Omega \tag{14}$$

- Boundary coefficients h, c, r, q, g can vary with space and time (can also depend on the solution u or the edge segment index)

Specify Boundary Conditions

Boundary conditions can be specified:

- Graphically using `pdetool`
- Programmatically using `pdebound`
  - `[q,g,h,r] = pdebound(p,e,u,time)`
  - Demo: `nacabound`

Specify PDE Coefficients

PDE coefficients can be specified:

- Graphically using `pdetool`
- Programmatically via constants, strings, functions
  - `u = parabolic(u0,tlist,b,p,e,t,c,a,f,d);`
    - `u0` - initial condition
    - `tlist` - time instances defining desired time steps
    - `b` - function handle to boundary condition
    - `p`, `e`, `t` - mesh
    - `c`, `a`, `f`, `d` - PDE coefficients (numeric, string, functions)

PDE solvers

- Elliptic
  - `[u,res]=pdenonlin(b,p,e,t,c,a,f);`
- Parabolic
  - `u=parabolic(u0,tlist,b,p,e,t,c,a,f,d);`
  - Demo: `workflow`
- Hyperbolic
  - `u=hyperbolic(u0,ut0,tlist,b,p,e,t,c,a,f,d);`
- Systems of Equations

Outline

What Now?

- Classes (non-exhaustive)
    - Numerical Linear Algebra
        - EE 263, CME 200, CME 302, CME 335
    - Numerical Optimization
        - CME 304, CME 334, CME 338
    - Object-Oriented Programming
        - CS 106B, CS 108
    - ODEs/PDEs
        - CME 102, CME 204, CME 206, CME 303, CME 306
    - Additional Advanced MATLAB classes (none to my knowledge)
        - Interest in taking this class as a full quarter class (3 units)
        - Indicate in evaluations
        - Email Margot Gerritsen (margot.gerritsen@stanford.edu)
- Future MATLAB questions
    - You have my email!

Teaching Evaluations

- Very important so please complete them
- Detailed comments in evaluations regarding the pros and cons of the course will be *much* appreciated
- Not available until end of Quarter
- If you have something important you wish to convey
  - Make a note of it now so you don't forget in a month
  - Email Margot (margot.gerritsen@stanford.edu)