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

Matthew J. Zahr

**CME 292**
Advanced MATLAB for Scientific Computing
Stanford University

30th April 2015
1 Symbolic Math Toolbox
   • Symbolic Computations
   • Mathematics
   • Code generation

2 Parallel Computing Toolbox

3 Ordinary Differential Equations

4 Partial Differential Equations
   • Overview
   • Mesh Generation in MATLAB
   • PDE Toolbox

5 Conclusion
Outline

1. Symbolic Math Toolbox
   - Symbolic Computations
   - Mathematics
   - Code generation

2. Parallel Computing Toolbox

3. Ordinary Differential Equations

4. Partial Differential Equations
   - Overview
   - Mesh Generation in MATLAB
   - PDE Toolbox

5. Conclusion
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

```matlab
>> % 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 can be constructed from symbolic variables

```matlab
>> 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

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>finverse</td>
<td>Functional inverse</td>
</tr>
<tr>
<td>linsolve</td>
<td>Solve linear system of equations</td>
</tr>
<tr>
<td>poles</td>
<td>Poles of expression/function</td>
</tr>
<tr>
<td>solve</td>
<td>Equation/System of equations solver</td>
</tr>
<tr>
<td>dsolve</td>
<td>ODE solver</td>
</tr>
</tbody>
</table>
## Formula Manipulation and Simplification

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>simplify</code></td>
<td>Algebraic simplification</td>
</tr>
<tr>
<td><code>simplifyFraction</code></td>
<td>Symbolic simplification of fractions</td>
</tr>
<tr>
<td><code>subexpr</code></td>
<td>Rewrite symbolic expression in terms of common subexpression</td>
</tr>
<tr>
<td><code>subs</code></td>
<td>Symbolic substitution</td>
</tr>
</tbody>
</table>
## Calculus

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>diff</td>
<td>Differentiate symbolic</td>
</tr>
<tr>
<td>int</td>
<td>Definite and indefinite integrals</td>
</tr>
<tr>
<td>rsums</td>
<td>Riemann sums</td>
</tr>
<tr>
<td>curl</td>
<td>Curl of vector field</td>
</tr>
<tr>
<td>divergence</td>
<td>Divergence of vector field</td>
</tr>
<tr>
<td>gradient</td>
<td>Gradient vector of scalar function</td>
</tr>
<tr>
<td>hessian</td>
<td>Hessian matrix of scalar function</td>
</tr>
<tr>
<td>jacobian</td>
<td>Jacobian matrix</td>
</tr>
<tr>
<td>laplacian</td>
<td>Laplacian of scalar function</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>potential</td>
<td>Potential of vector field</td>
</tr>
<tr>
<td>vectorPotential</td>
<td>Vector potential of vector field</td>
</tr>
<tr>
<td>taylor</td>
<td>Taylor series expansion</td>
</tr>
<tr>
<td>limit</td>
<td>Compute limit of symbolic expression</td>
</tr>
<tr>
<td>fourier</td>
<td>Fourier transform</td>
</tr>
<tr>
<td>ifourier</td>
<td>Inverse Fourier transform</td>
</tr>
<tr>
<td>ilaplace</td>
<td>Inverse Laplace transform</td>
</tr>
<tr>
<td>iztrans</td>
<td>Inverse Z-transform</td>
</tr>
<tr>
<td>laplace</td>
<td>Laplace transform</td>
</tr>
<tr>
<td>ztrans</td>
<td>Z-transform</td>
</tr>
</tbody>
</table>
Most matrix operations available for numeric arrays also available for symbolic matrices

- `cat`, `horzcat`, `vertcat`, `diag`, `reshape`, `size`, `sort`, `tril`, `triu`, `numel`

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>adjoint</code></td>
<td>Adjoint of symbolic square matrix</td>
</tr>
<tr>
<td><code>expm</code></td>
<td>Matrix exponential</td>
</tr>
<tr>
<td><code>sqrtm</code></td>
<td>Matrix square root</td>
</tr>
<tr>
<td><code>cond</code></td>
<td>Condition number of symbolic matrix</td>
</tr>
<tr>
<td><code>det</code></td>
<td>Compute determinant of symbolic matrix</td>
</tr>
<tr>
<td><code>norm</code></td>
<td>Norm of matrix or vector</td>
</tr>
<tr>
<td><code>colspace</code></td>
<td>Column space of matrix</td>
</tr>
</tbody>
</table>

CME 292: Advanced MATLAB for SC  Lecture 8
# Linear Algebra

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>null</td>
<td>Form basis for null space of matrix</td>
</tr>
<tr>
<td>rank</td>
<td>Compute rank of symbolic matrix</td>
</tr>
<tr>
<td>rref</td>
<td>Compute reduced row echelon form</td>
</tr>
<tr>
<td>eig</td>
<td>Symbolic eigenvalue decomposition</td>
</tr>
<tr>
<td>jordan</td>
<td>Jordan form of symbolic matrix</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>chol</td>
<td>Symbolic Cholesky decomposition</td>
</tr>
<tr>
<td>lu</td>
<td>Symbolic LU decomposition</td>
</tr>
<tr>
<td>qr</td>
<td>Symbolic QR decomposition</td>
</tr>
<tr>
<td>svd</td>
<td>Symbolic singular value decomposition</td>
</tr>
<tr>
<td>inv</td>
<td>Compute symbolic matrix inverse</td>
</tr>
<tr>
<td>linsolve</td>
<td>Solve linear system of equations</td>
</tr>
</tbody>
</table>
## Assumptions

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>assume</code></td>
<td>Set assumption on symbolic object</td>
</tr>
<tr>
<td><code>assumeAlso</code></td>
<td>Add assumption on symbolic object</td>
</tr>
<tr>
<td><code>assumptions</code></td>
<td>Show assumptions set on symbolic variable</td>
</tr>
</tbody>
</table>
## Polynomials

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>charpoly</td>
<td>Characteristic polynomial of matrix</td>
</tr>
<tr>
<td>coeffs</td>
<td>Coefficients of polynomial</td>
</tr>
<tr>
<td>minpoly</td>
<td>Minimal polynomial of matrix</td>
</tr>
<tr>
<td>poly2sm</td>
<td>Symbolic polynomial from coefficients</td>
</tr>
<tr>
<td>sym2poly</td>
<td>Symbolic polynomial to numeric</td>
</tr>
</tbody>
</table>
Mathematical Functions

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>log, log10, log2</td>
<td>Logarithmic functions</td>
</tr>
<tr>
<td>sin, cos tan, etc</td>
<td>Trigonometric functions</td>
</tr>
<tr>
<td>sinh, cosh tanh, etc</td>
<td>Hyperbolic functions</td>
</tr>
</tbody>
</table>

- 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

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>digits</td>
<td>Variable-precision accuracy</td>
</tr>
<tr>
<td>double</td>
<td>Convert symbolic expression to MATLAB double</td>
</tr>
<tr>
<td>vpa</td>
<td>Variable precision arithmetic</td>
</tr>
</tbody>
</table>
## Functions

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ccode</code></td>
<td>C code representation of symbolic expression</td>
</tr>
<tr>
<td><code>fortran</code></td>
<td>Fortran representation of symbolic expression</td>
</tr>
<tr>
<td><code>latex</code></td>
<td>\LaTeX\ representation of symbolic expression</td>
</tr>
<tr>
<td><code>matlabFunction</code></td>
<td>Convert symbolic expression to function handle or file</td>
</tr>
<tr>
<td><code>texlabel</code></td>
<td>TeX representation of symbolic expression</td>
</tr>
</tbody>
</table>
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

\[
\nabla \cdot \mathbf{P} + \rho_0 \mathbf{b} = 0 \\
\mathbf{P} = \mathbf{S} \cdot \mathbf{F}^T \\
\mathbf{S} = \lambda \text{tr}(\mathbf{E}) \mathbf{I} + 2\mu \mathbf{E} \\
\mathbf{E} = \frac{1}{2} (\mathbf{F}^T \mathbf{F} - \mathbf{I}) \\
\mathbf{F} = \mathbf{I} + \frac{\partial \mathbf{u}}{\partial \mathbf{X}}
\]
Exercise: Method of Manufactured Solutions

- For the displacement field, \( u(X) = [X_1 X_2 X_3, X_1^2 + X_2^2, \sin(X_3)]^T \), compute the corresponding forcing term.

- Generate code in MATLAB, C, and Fortran to compute the forcing term from above.
CME 292: Advanced MATLAB for SC  Lecture 8

Outline

1. Symbolic Math Toolbox
   - Symbolic Computations
   - Mathematics
   - Code generation

2. Parallel Computing Toolbox

3. Ordinary Differential Equations

4. Partial Differential Equations
   - Overview
   - Mesh Generation in MATLAB
   - PDE Toolbox

5. Conclusion
Programming Parallel Applications

Level of control

Minimal
Some
Extensive

Required effort
None
Straightforward
Involved
Programming Parallel Applications

**Level of control**
- Minimal
- Some
- Extensive

**Parallel Options**
- Support built into Toolboxes
- High-Level Programming Constructs: (e.g. parfor, batch, distributed)
- 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'

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

MATLAB®
client

MATLAB®
workers

parfor
The Mechanics of `parfor` Loops

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

<table>
<thead>
<tr>
<th>Classification</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loop</td>
<td>Loop index</td>
</tr>
<tr>
<td>Sliced</td>
<td>Arrays whose segments operated on by different iterations</td>
</tr>
<tr>
<td>Broadcast</td>
<td>Variable defined outside loop (not changed inside)</td>
</tr>
<tr>
<td>Reduction</td>
<td>Accumulates value across iterations</td>
</tr>
<tr>
<td>Temporary</td>
<td>Variable created inside loop (not available outside)</td>
</tr>
</tbody>
</table>

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
parallel_demo.m
Outline

1. Symbolic Math Toolbox
   - Symbolic Computations
   - Mathematics
   - Code generation

2. Parallel Computing Toolbox

3. Ordinary Differential Equations

4. Partial Differential Equations
   - Overview
   - Mesh Generation in MATLAB
   - PDE Toolbox

5. Conclusion
A system of Ordinary Differential Equations (ODEs) can be written in the form

\[ \frac{\partial y}{\partial t}(t) = F(t, y) \]

\[ y(0) = y_0 \]

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 \([t_0, t_1, \ldots, t_N]\)
- At step \( n \), \( y_n \) is known and \( y_{n+1} \) is sought

\[
\begin{align*}
    k_1 &= F(t_n, y_n) \\
    k_2 &= F(t_n + 0.5\Delta t, y_n + 0.5\Delta tk_1) \\
    k_3 &= F(t_n + 0.5\Delta t, y_n + 0.5\Delta tk_2) \\
    k_4 &= F(t_n + \Delta t, y_n + \Delta tk_3)
\end{align*}
\]

\[
y_{n+1} = y_n + \frac{\Delta t}{6} (k_1 + 2k_2 + 2k_3 + k_4)
\]

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

Requires 4 evaluations of \( F \) 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 $n$, $y_n$ is known and $y_{n+1}$ is sought

$$y_{n+1} = y_n + F(t_{n+1}, y_{n+1})$$  \hspace{1cm} (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

<table>
<thead>
<tr>
<th>Command</th>
<th>Type</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>ode45</td>
<td>Nonstiff</td>
<td>Medium</td>
</tr>
<tr>
<td>ode23</td>
<td>Nonstiff</td>
<td>Low</td>
</tr>
<tr>
<td>ode113</td>
<td>Nonstiff</td>
<td>Low - High</td>
</tr>
<tr>
<td>ode15s</td>
<td>Stiff</td>
<td>Low - Medium</td>
</tr>
<tr>
<td>ode23s</td>
<td>Stiff</td>
<td>Low</td>
</tr>
<tr>
<td>ode23t</td>
<td>Moderately stiff</td>
<td>Low</td>
</tr>
<tr>
<td>ode23tb</td>
<td>Stiff</td>
<td>Low</td>
</tr>
</tbody>
</table>

Assignment

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

\[ y'(t) = y^2(1 - y), \quad 0 \leq t \leq 2/\epsilon, \quad y(0) = \epsilon \]

- 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

1. Symbolic Math Toolbox
   - Symbolic Computations
   - Mathematics
   - Code generation

2. Parallel Computing Toolbox

3. Ordinary Differential Equations

4. Partial Differential Equations
   - Overview
   - Mesh Generation in MATLAB
   - PDE Toolbox

5. Conclusion
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

\[
\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} = \epsilon \frac{\partial^2 u}{\partial x^2}
\]

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

\[
\frac{\partial U}{\partial t} = F(U(t), t),
\]

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 obtain
  \[
  \frac{\partial u}{\partial x}(x_i, t) \approx \frac{u(x_i, t) - u(x_{i-1}, t)}{\Delta x}
  \]  
  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
  \[
  \frac{\partial^2 u}{\partial x^2}(x_i, t) \approx \frac{u(x_{i+1}, t) - 2u(x_i, t) + u(x_{i-1}, t)}{\Delta x^2}
  \]  
  for \( i = 1, \ldots, N - 1 \)
At the last equation, a first-order, leftward bias of the second-order derivative is applied

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

The boundary condition is applied as

\[
u(x_0, t) = u(0, t)
\] (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

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>initmesh</td>
<td>Create initial triangular mesh</td>
</tr>
<tr>
<td>adaptmesh</td>
<td>Adaptive mesh generation and PDE solution</td>
</tr>
<tr>
<td>jigglemesh</td>
<td>Jiggle internal points of triangular mesh</td>
</tr>
<tr>
<td>reinemesh</td>
<td>Refine triangular mesh</td>
</tr>
<tr>
<td>tri2grid</td>
<td>Interpolate from PDE triangular mesh to rectangular grid</td>
</tr>
<tr>
<td>pdemesh</td>
<td>Plot PDE triangular mesh</td>
</tr>
<tr>
<td>pdetriq</td>
<td>Triangle quality measure</td>
</tr>
</tbody>
</table>

```matlab
>> [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
    \[-\nabla \cdot (c\nabla u) + au = f\]  
    \[(9)\]
  - Parabolic
    \[
d\frac{\partial u}{\partial t} - \nabla \cdot (c\nabla u) + au = f\]  
    \[(10)\]
  - Hyperbolic
    \[
d\frac{\partial^2 u}{\partial t^2} - \nabla \cdot (c\nabla u) + au = f\]  
    \[(11)\]
  - Eigenvalue
    \[-\nabla \cdot (c\nabla u) + au = \lambda du\]  
    \[(12)\]
- PDE coefficients \(a, c, d, f\) can vary with space and time (can also depend on the solution \(u\) or the edge segment index)
Boundary conditions available for both scalar and vector available in PDE toolbox

Here we focus on scalar PDEs

- Dirichlet (essential) boundary conditions
  \[ hu = r \text{ on } \partial\Omega \]  
  \[ (13) \]

- Generalized Neumann (natural) boundary conditions
  \[ n \cdot (\nabla u) + qu = g \text{ on } \partial\Omega \]  
  \[ (14) \]

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**
  - \(u=\text{parabolic}(u0,tlist,b,p,e,t,c,a,f,d);\)
  - **Demo**: workflow

- **Hyperbolic**
  - \(u=\text{hyperbolic}(u0,\text{ut}0,tlist,b,p,e,t,c,a,f,d);\)

- **Systems of Equations**
Outline

1. Symbolic Math Toolbox
   - Symbolic Computations
   - Mathematics
   - Code generation

2. Parallel Computing Toolbox

3. Ordinary Differential Equations

4. Partial Differential Equations
   - Overview
   - Mesh Generation in MATLAB
   - PDE Toolbox

5. Conclusion
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)