MATLAB for Engineers (Beginner's Edition)

http://uiuc-cse.github.io/matlab-sp14/

Section 1, Session 2

Review of concepts (square v. round brackets; colons; semicolons) • 10min

Elementwise & matrix operators: Truss forces • 15min

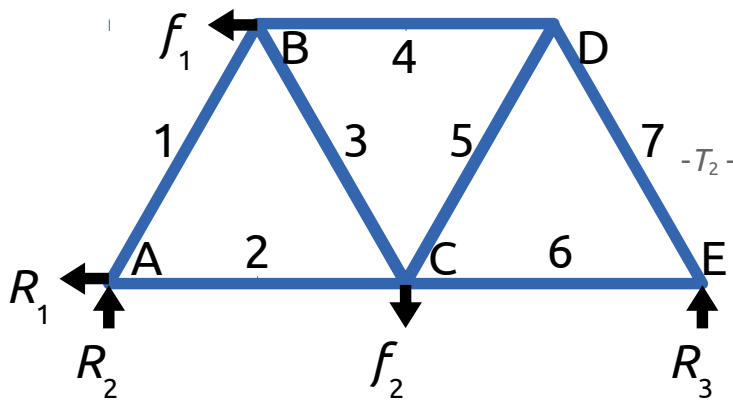
Vectorization, control flow: Falling ballistic object • 15min

Control flow: Matrix definition • 15min

Loops: Finite-difference heat equation • 25min

Example: Truss forces (Elementwise & matrix operators) • Mathematics • 5min

A common problem in mechanics is the solution of forces in a truss. This is solved statically by the *method of joints*, in which an equation is written for each node of the truss and resulting set of linear equations is solved.



$$
\begin{aligned}
0.5\ T_1 + T_2 &= R_1 = f_1 \\
0.866\ T_1 &= -R_2 = -0.433\ f_1 - 0.5\ f_1 \\
-0.5\ T_1 + 0.5\ T_3 + T_4 &= -f_1 \\
0.866\ T_1 + 0.866\ T_3 &= 0 \\
-T_2 - 0.5\ T_3 + 0.5\ T_5 + T_6 &= 0 \\
0.866\ T_3 + 0.866\ T_5 &= f_2 \\
-T_4 - 0.5\ T_5 + 0.5\ T_7 &= 0
\end{aligned}
$$

Let us write the matrix form of this equation as $\mathbf{T}x = f$. You could write the solution to this formally as $x = \mathbf{T}^{-1}f$. While attractive formally, it is often far too expensive to calculate and store an inverse matrix in memory for large problems. Matrix inversion is a brute-force solution to a linear algebra problem. MATLAB has a number of clever ways to solve matrices built into it. The most frequent method is *not* to invert the matrix, but instead to use what is called *right division*, written $x = \mathbf{T}\backslash f$.

- Let $f_1 = 1000$ and $f_2 = 2000$. Write the governing equations in matrix form.

$$
\begin{pmatrix}
0.5 & 1 & 0 & 0 & 0 & 0 & 0 \\
0.866 & 0 & 0 & 0 & 0 & 0 & 0 \\
-0.5 & 0 & 0.5 & 1 & 0 & 0 & 0 \\
0.866 & 0 & 0.866 & 0 & 0 & 0 & 0 \\
0 & -1 & -0.5 & 0 & 0.5 & 1 & 0 \\
0 & 0 & 0.866 & 0 & 0.866 & 0 & 0 \\
0 & 0 & 0 & -1 & -0.5 & 0 & 0.5
\end{pmatrix}
x =
\begin{pmatrix}
f_1 \\
-0.433 f_1 - 0.5 f_2 \\
-f_1 \\
0 \\
0 \\
f_2 \\
0
\end{pmatrix}
$$

- You can either define **T** and *f* using the MATLAB *New Variable...* interface or load it from the file `truss-matrix.mat`.

```
load('truss-matrix.mat', 'T')
load('truss-matrix.mat', 'f')
```

- Solve the matrix using inversion:

```
x = inv(T)*f
```

- Solve the matrix in MATLAB using right division:

```
x = T\f;
```

- Compare the speed of these two methods using `tic` and `toc`.

```
tic; x = T\f; toc
```

Example: Falling ballistic object (vectorization, functions, control) • Physics • 15min

- You are familiar with the equation for an object moving in a gravitational field (a.k.a. falling). The formula for vertical position y as a function of time t may be written

$$y(t) = y_0 + v\,t + a\,t^2$$

- where y_0 is the initial position (typically ground level, $y_0 = 0$); v is the initial y-velocity; and a is the acceleration (in this case due to gravity, a = -9.8 m/s^2). The classic ballistics example would be an object fired into the air such that a bullet or shell has a high initial velocity. Let's update this by using a railgun, with a projectile velocity of 2.52km/s.
- First, calculate the values directly.

```
a=-9.8; %m/s^2
v=2520; %m/s
x0=0;
t=1;
y=a*t^2+v*t+x0;
```

This will rapidly become cumbersome if we require many values.

- So let's *vectorize* the formula, or allow it to work on many values simultaneously.

```
t=linspace(0,5,101);
y=a*t^2+v*t+x0;      % comment on this error or ask for input
y=a*t.^2+v*t+x0;
```

Plot the vectorized formula: `plot(t,y);`

- Finally, let's abstract the formula out into its own function. This which will allow us to reuse it over and over again as necessary. Open a new blank file and write the function definition: `function y = a_fall(t,v,x0)`
- Test the function. When does a projectile traveling at 2.52km/s strike the ground?
- What is the maximum position (`max`)? Where is it (`[val, index] = max(y)`)?
- Modify the function to not yield y-values less than zero (`if` statement).
- Write another function, `v_fall`, to give the velocity at each position. Use this to find the time at which the highest point of the projectile is attained.

Control flow: Matrix definition • Computer Science • 15min

- A simple `for` loop merely cycles through elements of an array:

```
for counter = linspace(0,1,11)
      disp('Now at number:')
      disp(counter)
end
```

- This is useful in a number of applications both scientific and numerical. For instance, one can define a finite-difference matrix as follows:

```
% Preallocate a matrix
nrows = 10;
ncols = 10;
myData = ones(nrows, ncols);

% Loop through the matrix
for r = 1:nrows
   for c = 1:ncols
      if r == c
         myData(r,c) = 2;
      elseif abs(r - c) == 1
         myData(r,c) = -1;
      else
         myData(r,c) = 0;
      end
   end
end
```

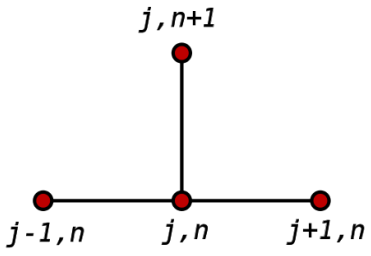
Example: Heat conduction (Numerics & linear algebra) • Mech. Eng. • 30min

A finite difference approximation is a numerical method which approximates a derivative (as in a differential equation) by a finite difference, as follows.

$$f'(x) \approx \frac{f(x+h)-f(x)}{h} \quad \text{thus} \quad f''(x) \approx \frac{f(x+h)-2f(x)+f(x-h)}{h^2}$$

Now, we can utilize this fact to solve a transient heat conduction problem (file `heat_FTCS.m`). (The picture replaces i by j and m by n but describes how each point is calculated.)

$$\frac{1}{\alpha}\frac{du}{dt} = \frac{d^2u}{dx^2} \quad \text{thus} \quad \frac{1}{\alpha}\frac{u_i^m}{\delta t} = \frac{u_{i-1}^{m-1} - 2u_i^{m-1} + u_{i+1}^{m-1}}{\delta x^2}$$



Given a bar of material with $\alpha = 0.1$ heated to the initial condition $u(x,t=0) = \sin(\pi x/L)$ and boundary conditions $u(x=0,t) = u(x=L,t) = 0$, with a length $L = 1$, find the temperature profile $u(x,t)$ from $t_0 = 0$ to $t_{max} = 0.5$.

Use a `for` statement to perform the time iteration forward and the calculation at each spatial point of the array per time step.