Tuesday, 12 November

We wish to introduce two basic elements of control flow: the `for` statement and the `if` statement. The first allows us to automate repetitive tasks and the other helps us make basic decisions in the program. (If you have seen these in another language previously, they work much the same here in MATLAB.)

Example: Spring (curve fitting, plotting) • Physics • 20min

Curve fitting, interpolation, publication-quality plotting: spring example.

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

Let's use a `for` statement to set up the initial condition of the system, as well as to perform both the time iteration forward and the calculation at each point of the array per time step.

Recall that the if statement allows us to handle special cases. For instance, if we had a point heat source at $x = 2$ (like a soldering iron held to the beam), we could include that term with an if statement. In this case, we will use the if statement to generate our initial condition, a step function across the origin (imagine an iron which can instantly heat half of the beam to $T = 80$℃ and then is removed before the other half of the beam can respond).

Example: Radioactive decay chain (System of 1st-order ODEs) • Physics • 30min

This exercise will demonstrate the solution of first-order differential equations such as those used to describe radioactive decay processes. Given a large number of atoms, we may treat the number of atoms N as continuous. The rate of decay is proportional to the number of nuclei. We can write

$$\frac{dN}{dt} = -\lambda N$$

where λ is the decay constant characteristic of the material.

One advantage of modeling this system is that it has an analytical solution which we can use to assess the different methods of numerical solution that we will employ:

$$N(t) = N_0 \exp(-\lambda t)$$

where N_0 is the initial amount of the material.

- First, let's define a function for the analytical solution. Open a new file and save it as rad_analytic.m.
- Write a function in this file to return the analytical value for N given N_0, λ, t.

```
function N = rad_analytic(N0,lambda,t)
N = N0 * exp(-lambda * t);
```

- Test the function for a few cases:

```
rad_analytic(0,100)==100 % test the initial condition
rad_analytic(1,100)<36.79 && rad_analytic(1,100)>36.77 % test floating point
values with bounds not equality
t_ana=[0:0.1:10];
N_ana=rad_analytic(t_ana,100);
plot(t_ana,N_ana)
```

For the numerical solution, MATLAB requires that the right-hand side of a differential equation (the non-differential expression) be available as a function. In other words, for

$$\frac{dN}{dt} = -\lambda N$$

the function should return simply the value -λN.

- Open a new file and save it as dNdt.m. Write a function to return the analytical value for dN/dt given N, t. (Hard-code in the value of lambda as 1.0.)
- We will use the low-order ode23 method first.

```
[t_23,N_23] = ode23(@dNdt,[0,10],[100]);
```

- Compare the results visually.

```
plot(t_23,N_23,t_ana,N_ana);
```

ode23 is a low-order solver, however, and results can often be improved by moving to the slower but higher-order ode45 solver. (You will probably not perceive a difference in our work today.)

```
[t_45,N_45] = ode45(@dNdt,[0,10],[100]);
plot(t_23,N_23,t_ana,N_ana,t_45,N_45);
```

- Calculate the residual error at each point of the analytical solution. Because the time-domain points calculated by the ODE solvers are not at the same points as the analytical solution we acquired above, the calculation of residuals (errors) is slightly complicated. We can either interpolate directly (using interp) or we can use the handy built-in function deval, which will

be demonstrated here.

```
sol = ode45(@dNdt,[0,10],[100]);
N_int = deval(sol,t_ana,1);
resids = N_ana-N_int;
plot(t_ana,resids)
```

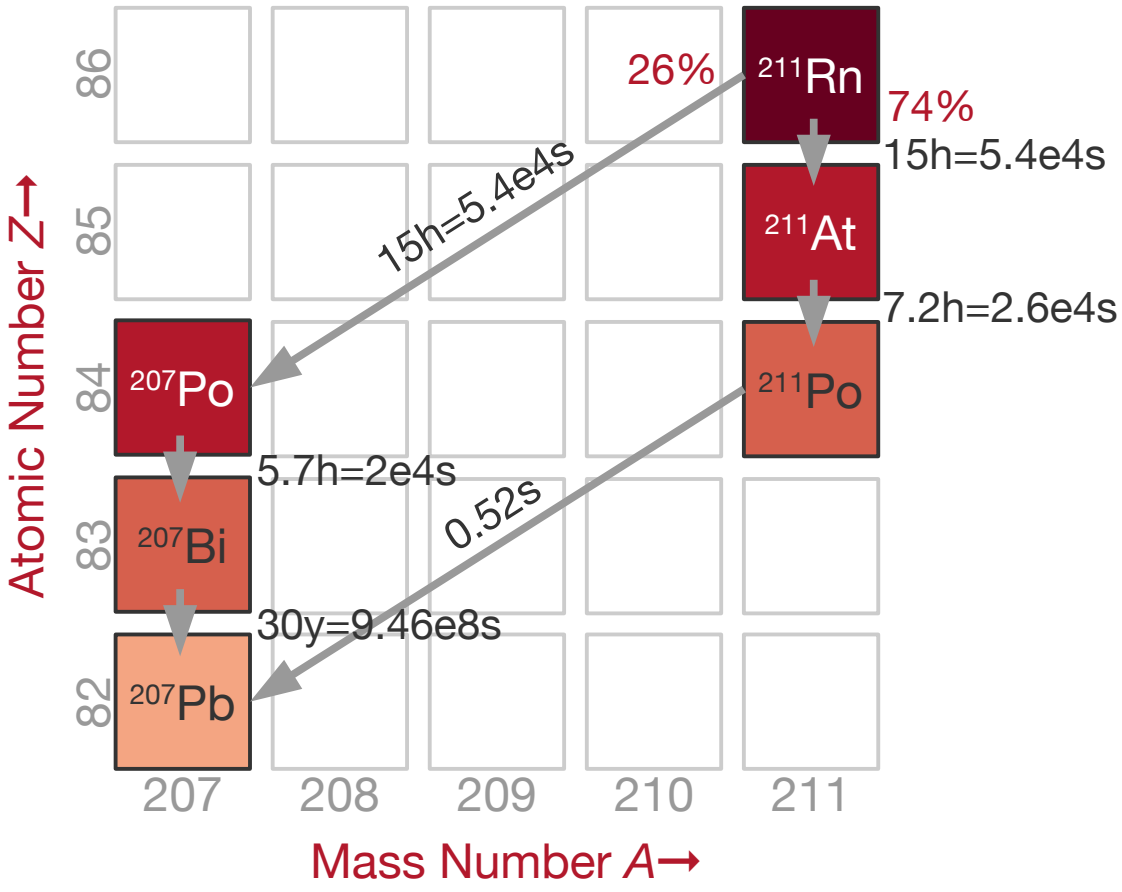
Now let's introduce another species (or model a decay chain). Because we wrote this as a function in MATLAB, it can be easily vectorized. Let's first construct a hypothetical decay chain in which element A turns into B with half-life $t_{½,A} = 1.0$ and B turns into C with half-life $t_{½,B} = 20$. (Recall that $\lambda_A = (\ln 2)/t_{½,A}$.)

- Write the governing equations for this system.
- Convert these equations into matrix form.
- Make a copy of dNdt.m as decay_chain.m. Define the decay constants as `l_A`, etc. Set up the matrix form of the equations and return the result.
- Introduce the matrix equation into `ode45` and plot the results. (We will forgo including the analytical solution at this time, but this can be readily done.)

```
N0=[100;0;0];
[T,N3]=ode45(@decay_chain,[0 50],N0);
plot(T,N3)
```

Example: Radioactive decay scheme (System of 1st-order ODEs) • Physics • 20min

Now that we have seen how to introduce a decay chain, let's explore a decay scheme consisting of six isotopes of five elements.



Radon-211 decays via two competing chains to the stable lead-207 as illustrated above. First we need to cast this reaction into a matrix form so that we can feed it into the ODE solver.

- Write the explicit first-order decay equations for this reaction.
- Again, put these into matrix form and write corresponding dNdt functions for each of them. Don't forget to convert the half-lives to decay constants.
- Put the resulting matrix equation into `ode45`. Note that the system takes an inordinately long time to resolve due to the wildly different time scales of the decay chain. Try using the stiff equation solver `ode15s` instead.

```
N0=[100;0;0;0;0;0];
tic;[T,N6]=ode45(@decay_scheme,[0 5e5],N0);toc  %45s
tic;[T,N6]=ode113(@decay_scheme,[0 5e5],N0);toc %63s
tic;[T,N6]=ode15s(@decay_scheme,[0 5e5],N0);toc %0.017s
plot(T,N6)
```

Example: Coolant flow in channel (2nd-order, PDE Toolbox) • Mech. Eng. • 20min

Now let's illustrate how you might set up a more complex geometry in MATLAB. The PDE Toolbox is a graphical interface for designing and calculating equations on a mesh geometry.

- Invoke the PDE Toolbox with `pdetool`.
- Create a rectangle and a circle overlapping a quarter of it. Change the set formula to R1-E1 to make the circle subtractive.
- First initialize a mesh, then refine the mesh. Now jiggle it.
- Change to Boundary Mode. Mixed boundary conditions can only be done in systems, so we'll cheat in this case and just set the outer and inner boundaries directly (perfect heat transfer at the surface).
    - Set the inner circle (coolant channel surface) to Dirichlet BC with h=1, r=400.
    - Set the outer (top) boundary (combustion surface) to Dirichlet BC with h=1, r=1700.
    - Set the three inner boundaries to Neumann BC with g=0, q=1.
- Set up the PDE. The types of PDE for which built-in solvers are available are shown here. Create an elliptic PDE with c = 1.0; a = 0.0; f = 0.0; and d = 1/3.6.

  (If, like me, you don't recall the mathematical terms for each class of PDE, here's a list:
    - elliptic = Laplace equation
    - parabolic = heat equation
    - hyperbolic = wave equation

  Other types (including nonlinear PDEs) require direct implementation of a numerical method to solve. Next time we will look at a nonlinear PDE, the Burgers' equation, which introduces a shock wave in its solution.)
- Solve the PDE.
- Plot it. Add contours (10 lines), flat shading, and change the colormap to 'hot'. Examine the abs(grad(u)) plot as well; if the sides are not horizontal, try playing with parameters such as the mesh resolution to solve that problem.