Session 2: Numerics

Curve fitting & interpolation (spring)

Linear algebra (truss)

First-order ODEs (radioactive decay)

Systems of ODEs (radioactive decay)

Electives: File operations (temperature); Publishing (any); Finite difference equations (heat transfer)

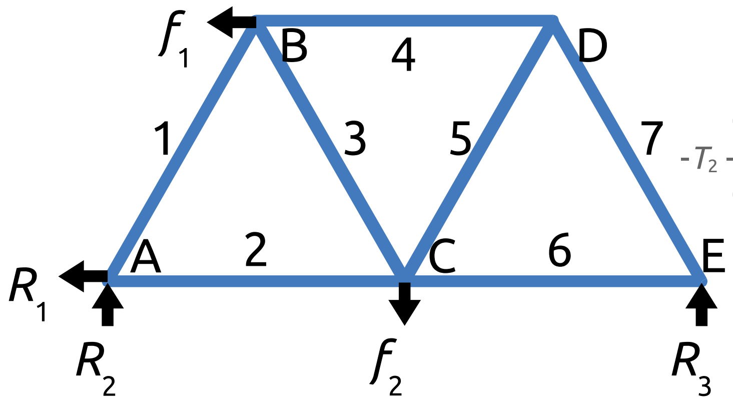
Curve fitting & Interpolation

Example: Hooke's law (spring constant)

Linear algebra

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 you write an equation for each node of the truss and solve the linear set of equations which results.



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

Let us write the matrix form of this equation as $\mathbf{M}x = f$. You could write the solution to this formally as $x = \mathbf{M}^{-1}f$. While attractive formally, it is often far too expensive to calculate and store an inverse matrix in memory for large problems (even if $\mathbf{M}$ is sparse, $\mathbf{M}^{-1}$ is in general dense).

In practice, 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 is not to invert the matrix, but instead to use what is called *left division*, written $x = \mathbf{M}\backslash f$.

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

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

- Define the matrix using the MATLAB *New Variable...* interface.
- Write the solution vector and call it *f*.
- Solve the matrix in MATLAB using left division:

```
x = M\f;
```

First-order ODEs and Systems of Equations

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 $-\lambda N$.

- 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)
```

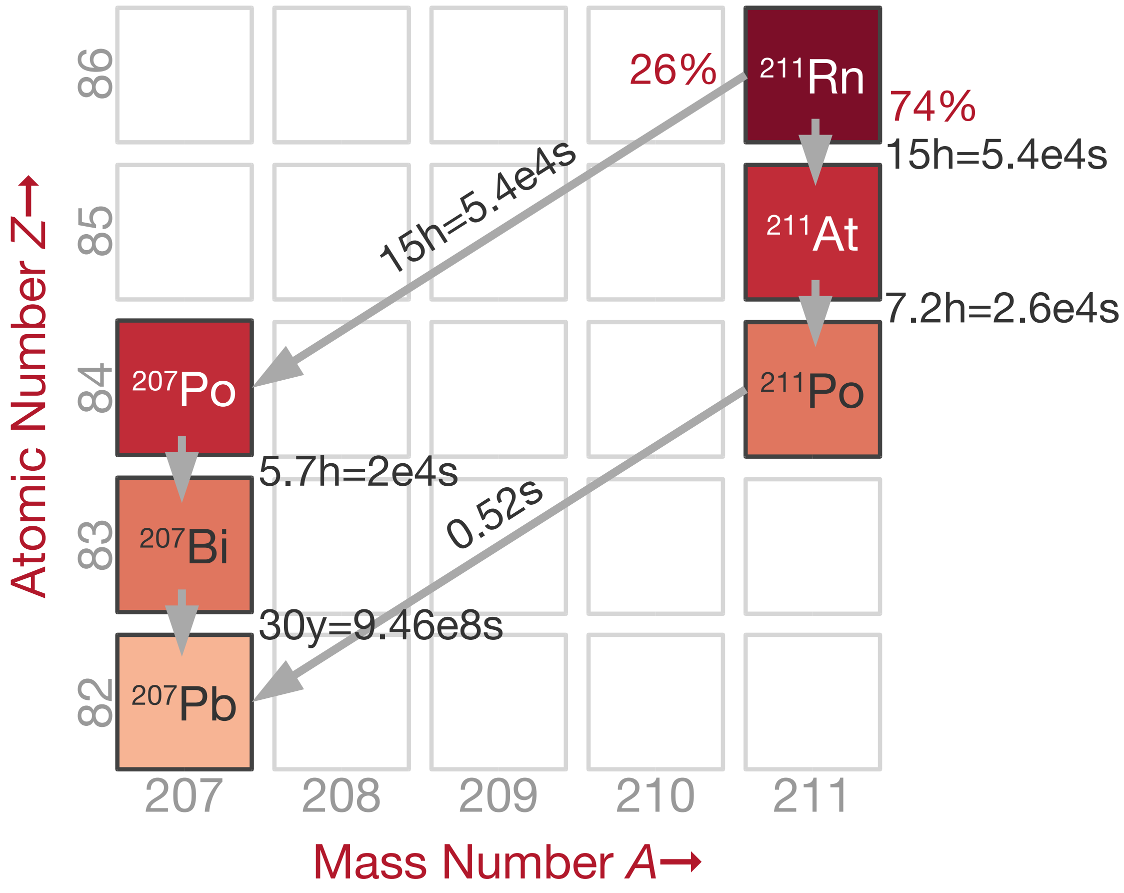
Now let's introduce another species (or model a decay chain). Because we wrote this as a function in MATLAB, it can be easily vectorized. Let's first construct a hypothetical decay chain in which element A turns into B with half-life $t_{½,\mathrm{A}}$ = 1.0 and B turns into C with half-life $t_{½,\mathrm{B}}$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

- 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 1^st-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)
```

Electives

Example: Fahrenheit/Celsius (Scripts, file operations) • Mathematics • 25min

You are familiar with the conversion formula from Fahrenheit to Celsius:

$$T_F(T_C) = T_C \frac{180}{100} + 32$$

- Write a function which performs this conversion and save it in an appropriate file.

  ```
  function Tf = TempC2F(Tc)
  ... your code here ...
  ```
- Write a script templist.m which generates a table of conversions for the range 0℃ to 200℃ and then writes that table to a file templist.txt.

  ```
  cList = linspace(0,200,101);
  fList = TempC2F(cList);

  fileID = fopen('templist.txt','w');
  fprintf(fileID,'%6s %12s\n','Tc','Tf');
  fprintf(fileID,'%6.2f %6.2f\n',cList,fList);
  fclose(fileID);
  ```
- Examine templist.txt. Is the output what you expect? Change the output line to the following:

  ```
  tList = [cList; fList];
  fprintf(fileID,'%6.2f %6.2f\n',tList);
  ```
  (Note that the row-major order of MATLAB translates into a transposed column-major order in the output.)

- Next, let's read the data in and plot them.

  ```
  tData = importdata('templist.txt')
  plot(tData.data(:,1),tData.data(:,2))
  ```

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 calcualtion 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).