Chemistry 20262 - Mathematical Methods Spring 2018
Professor J. Daniel Gezelter

In-class Practical 4
More advanced Mathematica topics

A reminder about lists

1. Remember that Mathematica uses lists to represent sequences. If we have the three (x, y) points: (1, 2), (2, 3), and (3, 5), each of these points is represented as a list of two numbers. The list of the three points is then a nested list, or a list of lists:

```
{ {1, 2}, {2, 3}, {3, 5} }
```

2. To plot a graph of these three points, we use a new Mathematica command:

```
ListPlot[ { {1, 2}, {2, 3}, {3, 5} } , PlotJoined -> True]
```

This tells Mathematica to plot the list of points inside the square brackets. The spaces are optional and are included here mainly for the sake of clarity.

Fitting data points to polynomial expressions

1. Suppose we have a set of experimental data that we want to fit with a straight line. For specificity, imagine that we have measured absorbance as a function of concentration in a test of the Beer-Lambert law:

concentration (M)	absorbance
0.01	0.095
0.02	0.202
0.04	0.413
0.06	0.629
0.10	1.101
0.15	1.735

First let's put this data into a list of (x, y) points:

```
data = { {0.01, 0.095}, {0.02, 0.202}, {0.04, 0.413},
         {0.06, 0.629}, {0.10, 1.101}, {0.15, 1.735} }
```

We can plot the data with

```
ListPlot[data]
```

from which we see that the data looks fairly linear at low concentrations, but starts to curve upward as the concentration increases. (Well, it's actually a bit hard to see this, but it will become more evident once we try to fit the data with a straight line.)

2. To fit the data to a straight line, type

```
Fit[data, {1, conc}, {conc}]
```

This tells Mathematica to fit the data in terms of two functions of the variable **conc**; these functions are **1** and **conc**. You should get the result

```
-0.044375 + 11.6875 conc
```

from which we see that the slope and y-intercept of the line are 11.6875 and -0.044375, respectively. (Note that Mathematica uses its default of six significant figures even though our data has fewer significant figures.)

The general form of the **Fit** command is:

```
Fit[ <data-list>, <function-list>, <variable-list> ]
```

The first argument is a list of (x, y) data points, in the standard Mathematica form of a nested list. The second argument is a list of functions that Mathematica will use to try to fit the points. The third argument is a list of the independent variables in the data set. In this example, we have only one independent variable, so the variable list is a list with one element: the variable **conc**. (Of course, we could name the independent variable anything we want, as long as the same name appears in the list of functions and the list of variables.) Later we'll see how we can fit data as a function of two or more independent variables.

3. You might be asking yourself, "Why is there a 1 in the list of functions?" Let's try leaving the 1 out:

```
Fit[data, {conc}, {conc}]
11.2461 conc
```

Here we have instructed Mathematica to fit the data to a straight line that **goes through the origin**. The 1 in the variable list is used to fit the y-intercept in our original example. Think of the fit as finding the best constants A and B such that the data is described by the line

```
A * 1 + B * conc
```

When we leave out the 1, we are telling Mathematica not to look for a constant term in the fitting equation. This is equivalent to finding the best constant B such that the data is described by the line

```
B * conc
```

with A implicitly set to zero.

4. In the example presented here, we might **want** to force the y-intercept to be zero; after all, that is the sensible Beer-Lambert law prediction. But if our apparatus is dirty or malfunctioning, it may register a "baseline" absorbance even when there is nothing in our sample cell. This would show up as a constant error in the absorbance, independent of concentration. So the fit to the line

   ```
   A * 1 + B * conc
   ```

   lets us estimate this error. Notice that the slopes of the two straight lines are slightly different.

5. As mentioned above, the Beer-Lambert law can break down at high concentrations. Let's add a quadratic term to our fitting equation to see if the data is described better that way:

   ```
   linfit = Fit[data, {1, conc}, {conc}]
   -0.044375 + 11.6875 conc
   quadfit = Fit[data, {1, conc, conc^2}, {conc}]
   -0.00325599 + 9.91865 conc + 11.1374 conc^2
   ```

   Now let's plot both fits, along with the experimental data:

   ```
   points = ListPlot[data]
   curve1 = Plot[linfit, {conc, 0, 0.15}]
   curve2 = Plot[quadfit, {conc, 0, 0.15}]
   Show[points, curve1]
   Show[points, curve2]
   ```

   We see that the quadratic curve does seem to fit the data better. In addition, the quadratic fit leads to a y-intercept which is much closer to zero, in accord with our intuition.

Contour and surface plots

1. Mathematica can make very nice contour and surface plots of three-dimensional functions such as z = z(x, y). The commands that generate these plots are **ContourPlot** and **Plot3D** for contour and surface plots, respectively.

   These commands have the same basic form, although there are more "options" for **Plot3D**. To make a plot of some function of x and y in which x ranges from xmin to xmax and y ranges from ymin to ymax, enter

   ```
   ContourPlot[ <function>, {x, <xmin>, <xmax>},
                  {y, <ymin>, <ymax>} ]
   ```

or

```
Plot3D[ <function>, {x, <xmin>, <xmax>},
          {y, <ymin>, <ymax>} ]
```

For example, a contour plot of the function $e^{-(x^2 + 3 y^2)}$ near the origin can be generated as follows:

```
ContourPlot[ Exp[-(x^2 + 3 y^2)], {x, -2, 2}, {y, -2, 2} ]
```

A three-dimensional surface plot of the same function is generated with the command

```
Plot3D[ Exp[-(x^2 + 3 y^2)], {x, -2, 2}, {y, -2, 2} ]
```

Note that you can manipulate the three dimensional plots with your mouse to get a better view of the function.

2. If you try these commands, you might find that the surface plot seems "chopped off" near the origin. We can rectify this by including a larger range of z coordinates in the plot, using the **PlotRange** option that we first saw in two-dimensional plots:

```
Plot3D[ Exp[-(x^2 + 3 y^2)], {x, -2, 2}, {y, -2, 2},
PlotRange -> {0, 2} ]
```

This command tells Mathematica to include all parts of the function with z values between 0 and 2. We can also use the **PlotRange** option to specify boundaries on all three coordinates, for example to zoom into a portion of the surface:

```
Plot3D[ Exp[-(x^2 + 3 y^2)], {x, -2, 2}, {y, -2, 2} ]
Show[ %, PlotRange -> {{0, 1}, {0, 1}, {0, 2}} ]
```

3. You might notice that the resolution in this "zoomed" plot is fairly poor. Mathematica creates surface plots by scanning over a rectangular grid of points and calculating the height of the surface at each point. When we zoom in, Mathematica continues to use the original grid of points. We can make the grid finer by using the **PlotPoints** option in **Plot3D**. (Note that this option does **not** work in the **Show** command!) Here is how we might increase the resolution of this zoomed plot:

```
Plot3D[ Exp[-(x^2 + 3 y^2)], {x, -2, 2}, {y, -2, 2},
          PlotPoints-> 30 ]
Show[ %, PlotRange -> {{0, 1}, {0, 1}, {0, 2}} ]
```

Of course, if you decided to zoom in after viewing a surface plot, you can combine the **PlotPoints** and **PlotRange** options in the **Plot3D** command to do everything in one step:

```
Plot3D[ Exp[-(x^2 + 3 y^2)], {x, -2, 2}, {y, -2, 2},
          PlotPoints-> 30,
          PlotRange -> {{0, 1}, {0, 1}, {0, 2}} ]
```

Even this is wasteful in a sense; we are generating a 30-by-30 grid of points in which x and y both range from -2 to 2, and then throwing away all of the points except those where x and y are between 0 and 1. Instead, we might try

```
Plot3D[ Exp[-(x^2 + 3 y^2)], {x, 0, 1}, {y, 0, 1},
          PlotPoints-> 30, PlotRange -> {0, 2} ]
```

which gives us higher resolution in the region of interest.

4. The **PlotPoints** option applies to contour plots as well. Compare these two plots:

```
ContourPlot[ Exp[-(x^2 + 3 y^2)], {x, -2, 2}, {y, -2, 2} ]
ContourPlot[ Exp[-(x^2 + 3 y^2)], {x, -2, 2}, {y, -2, 2},
          PlotPoints -> 30 ]
```

The plot with more points is much smoother, and represents the oval shape of the surface more faithfully. But how can we get rid of the shades of gray, blue, and purple so that we can see the contours better? Try

```
ContourPlot[ Exp[-(x^2 + 3 y^2)], {x, -2, 2}, {y, -2, 2},
               PlotPoints -> 30, ContourShading -> False ]
```

Nice, isn't it? The only problem is that we don't know what z values the contour lines correspond to. With the **Contours** option we can specify exactly where we want the contours. Try entering

```
ContourPlot[ Exp[-(x^2 + 3 y^2)], {x, -2, 2}, {y, -2, 2},
               PlotPoints -> 30, ContourShading -> False,
               Contours -> {0.2, 0.4, 0.6, 0.8},
               PlotRange -> {0, 2} ]
```

to see how you can control the placement of contour lines. Note that we need to include **both** the **Contours** and the **PlotRange** options to make sure that all of the contour values fit inside the range of z values that Mathematica will plot. Remember that when you only give one pair of numbers for **PlotRange**, Mathematica applies this range to the z axis.

5. Finally, note that the variables x and y could be called anything we want. These two commands produce exactly the same results:

```
ContourPlot[ Exp[-(x^2 + 3 y^2)], {x, -2, 2}, {y, -2, 2} ]
ContourPlot[ Exp[-(u^2 + 3 v^2)], {u, -2, 2}, {v, -2, 2} ]
```

Vectors & Matrices

1. Lets define 3x3 matrices A, B, and vector c.

```
A = {{4, 0, 1}, {-2, 1, 0}, {-2, 0, 1}}
B = {{0, 1, 1}, {-3, 1, 5}, {0, 0, 1}}
c = {1, 2, 3}
```

2. We can look at these objects in Matrix Form:

```
MatrixForm[A]
MatrixForm[B]
MatrixForm[c]
```

3. We can get the individual elements of the matrix using the `[[ ]]` notation, where `A[[i,j]]` is the element A_{ij}. Note that c has only one index.

```
A[[1, 1]]
4
B[[3, 2]]
0
c[[3]]
3
```

4. We can also define a n x n identity matrix:

```
ii = IdentityMatrix[3]
MatrixForm[ii]
```

5. We can Transpose a matrix, and then show the result in MatrixForm:

```
Transpose[A] // MatrixForm
```

6. We can multiply two matrices using the "." operator. Note that A . B is another 3 x 3 matrix and B . c is a 3 x 1 vector:

```
A . B
{{0, 4, 5}, {-3, -1, 3}, {0, -2, -1}}
MatrixForm[%]

MatrixForm[B . c]
```

7. If we want to find the Eigenvalues of a matrix, we could set up the determinant:

```
Det[ A - lambda ii]
```

but we really need to solve for lambda when this determinant is zero, so:

```
Solve[ Det[ A - lambda ii] == 0, lambda ]
```

8. A much simpler way to get eigenvalues and eigenvectors is the built-in commands in Mathematica:

```
eval = Eigenvalues[A]
e1 = eval[[1]]
e2 = eval[[2]]
e3 = eval[[3]]
```

9. The Eigenvectors command returns all of the eigenvectors of the matrix as columns of a new matrix:

```
evec = Eigenvectors[A]
v1 = evec[[1]]
v2 = evec[[2]]
v3 = evec[[3]]
```

10. Now, we can verify that the eigenvectors all satisfy the eigenvalue equation $A\,v = e\,v$

```
A . v1 == e1 v1
A . v2 == e2 v2
A . v3 == e3 v3
```

11. Since A has distinct eigenvalues, A is *diagonalizable*. If we now let P be the matrix whose *columns* are the eigenvectors,

```
P = Transpose[ evec ]
Pinv = Inverse[ P ]
```

then $P^{-1}\,A\,P = D$, where D is the diagonal matrix whose diagonal entries are the respective eigenvalues. We verify this as follows:

```
diag = Pinv . A . P
MatrixForm[%]
```

Solving a linear system of n equations and n unknowns

1. Suppose we have a system of linear equations that we want to solve (don't type these in):

```
2x  +y  +z -2w =  1
3x -2y  +z -6w = -2
 x  -y  +z  +w = -1
3x  -y +2z -8w =  3
```

We first enter the coefficient matrix A and the constant matrix b:

```
A = {{2, 1, 1, -2}, {3,-2,1,-6}, {1, -1, 1, 1}, {3, -1, 2, -8}}
b = {1, -2, -1, -3}
```

2. Next we can solve the linear system using an inverse:

```
x = Inverse[A] . b
```

or alternatively

```
LinearSolve[A, b]
```

Problems (Due with Problem Set *9* on April 20):

Consider the linear CO_2 model we were talking about in class. In terms of the atomic coordinates, the total potential is given by:

$$V = \frac{1}{2}k\left(x_2 - x_1 - l_0\right)^2 + \frac{1}{2}k\left(x_3 - x_2 - l_0\right)^2$$

Here, k is the force constant for both of the C=O bonds. When the potential is harmonic like this, the equations of motion become much simpler, and can be arranged as a matrix equation. First, we define displacement coordinates,

$$\begin{aligned} q_1 &= x_1 + l_0 \\ q_2 &= x_2 \\ q_3 &= x_3 - l_0 \end{aligned}$$

Newton's equations of motion for the q coordinates are nearly the same as for the original coordinates:

$$\begin{aligned} m_O\frac{d^2q_1}{dt^2} &= F_1 = k(q_2 - q_1) \\ m_C\frac{d^2q_2}{dt^2} &= F_2 = k(q_1 - 2q_2 + q_3) \\ m_O\frac{d^2q_3}{dt^2} &= F_3 = -k(q_3 - q_2) \end{aligned}$$

Dividing by the masses, the matrix equation for the equations of motion are

$$\frac{d^2}{dt^2}\begin{pmatrix} q_1 \\ q_2 \\ q_3 \end{pmatrix} = \frac{k}{m_O}\begin{pmatrix} -1 & 1 & 0 \\ \gamma & -2\gamma & \gamma \\ 0 & 1 & -1 \end{pmatrix}\begin{pmatrix} q_1 \\ q_2 \\ q_3 \end{pmatrix}$$

where we define $\gamma = m_O/m_C$ or the mass ratio between oxygen and carbon. Solving for the eigenvectors of the matrix will *decouple* the three variables. That is, if $\vec{v}_1$ is an *eigenvector* of this matrix with associated *eigenvalue* λ_1 then,

$$\frac{d^2}{dt^2}\vec{v}_1 = \frac{k}{m_O}\begin{pmatrix} -1 & 1 & 0 \\ \gamma & -2\gamma & \gamma \\ 0 & 1 & -1 \end{pmatrix}\cdot\vec{v}_1 = \frac{k}{m_O}\lambda_1\vec{v}_1$$

This means we have three separate equations for v_{11}, v_{12}, v_{13}, and all of them have the same time dependence:

$$\frac{d^2 v_{11}}{dt^2} = \frac{k\lambda_1}{m_O} v_{11}$$

That was a big build up to the actual questions:

1. Find the general solution of the differential equation for $v_{11}(t)$ and leave it in terms of the constants k, m_O, and the eigenvalue λ_1. It may be useful to write $\omega_1 = \sqrt{-k\lambda_1/m_O}$ before doing this.

2. Find the three eigenvectors $\vec{v}_1, \vec{v}_2, \vec{v}_3$ and the associated eigenvalues $\lambda_1, \lambda_2, \lambda_3$ for this matrix. Do this symbolically in Mathematica (i.e. don't plug in any numbers).

3. Given your answer to number 1, what do these three *eigenvalues* tell you about the motion of the atoms in each of these three "modes" of motion?

4. Look at the eigenvectors for each of the modes. The elements of these vectors tell you how the three atoms move as the motion described by that eigenvector progresses. Can you describe what each of the eigenvectors will do to the molecule in each of these three modes?

5. If we freeze q_2 in place at the origin, you can make a surface plot the potential energy in terms of the atomic coordinates q_1, and q_3. You can set $k = 1$ arbitrarily, and let the coordinates range from -5 to +5. What does a larger value of k do? Can you show this effect on your surface plot?