

MATLAB the Calculator

1.1 Introduction

Programming is a powerful tool for every scientist, and in an era of data driven science and *computational thinking*, being able to solve problems using computational methods is more important than ever.

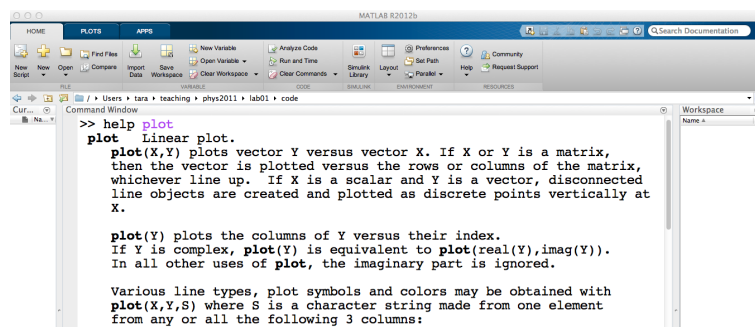
Some people find programming fun and exciting straight away. Others find it a bit intimidating, and find that it requires a new way of thinking. Whichever category you're in, the best way of learning and improving is by *actually writing code*. You can't learn Computational Physics by reading notes — you have to write code, run code, and experiment.

MATLAB is a great language for beginners but at the same time is a sophisticated programming environment that is widely used in physics and engineering. In its simplest form, you can think of MATLAB as an extremely good (and fast) calculator, and that's how we're going to start in the first lab session.

MATLAB has excellent built in documentation, so it is easy to look up how to use a particular function if you can't remember. For example, to get help on the **plot** function you can run:

```
>> help plot
```

and the results will look something like this:



```
>> help plot
plot Linear plot.
plot(X,Y) plots vector Y versus vector X. If X or Y is a matrix,
then the vector is plotted versus the rows or columns of the matrix,
whichever line up. If X is a scalar and Y is a vector, disconnected
line objects are created and plotted as discrete points vertically at
X.

plot(Y) plots the columns of Y versus their index.
If Y is complex, plot(Y) is equivalent to plot(real(Y),imag(Y)).
In all other uses of plot, the imaginary part is ignored.

Various line types, plot symbols and colors may be obtained with
plot(X,Y,S) where S is a character string made from one element
from any or all the following 3 columns:
```

MATLAB behaves in the same way whether you run it on Windows, Mac OS X or Linux, so you can practice in the Access labs or at home, and the code you write will run on the lab computers as well.

1.2 Lab objectives

The aim of this lab is to familiarise yourself with basic mathematical functionality and plotting in MATLAB. By the end of this lab session you should be able to:

1. Understand what MATLAB is and run the MATLAB interpreter.
2. Perform elementary operations.
3. Calculate expressions using variables.
4. Visualize simple datasets using **plot ()**.

If you still don't know how to do any of these things once you have completed the lab, please ask your tutor for help. You should also use these lists of objectives when you are revising for the end of semester exam.

1.3 Walkthrough: Doing maths with MATLAB

1.3.1 Getting started

The main window in the MATLAB interpreter contains the command prompt `>>` which is where you type commands. MATLAB runs each command you type and displays the result on the screen. You can think of MATLAB as an *extremely powerful* calculator. Let's try out some basic commands:

```
1 >> 1 + 3*5      % you type this
2 ans = 16        % the computer responds
```

The order of operations is the same as in a regular scientific calculator: multiplication (`*`) and division (`/`) are evaluated before addition (`+`) and subtraction (`-`). You can use parenthesis to control the order of operations:

```
1 >> (1 + 3)*5
2 ans = 20
```

Multiplication and division are evaluated from left to right, so:

```
1 >> 3*4/6*2
2 ans = 4
```

gives an answer of 4, not 1 as many people expect. To calculate powers you can use the caret `^` symbol:

```
1 >> 2^10
2 ans = 1024
```

There are a wide range of standard mathematical functions, such as **sin**, **mod**, **exp**, and **log**. For example:

```
1 >> cos(2*pi)
2 ans = 1
```

Note that the variables **pi**, **e** and **i** are built-in: MATLAB already knows the values of these constants:

```
1 >> pi*2
2 ans = 6.2832
```

See Section 1.5.3 for a list of useful mathematical functions. By default, trigonometric functions like **sin** take input in **radians** not **degrees**.

1.3.2 MATLAB variables

To write any useful program we need to store information in *variables*. A variable is a name given to a chunk of memory that contains a value. Variables allow you to save your results and use them later in the program. Like mathematical variables, they can be manipulated and operated on. We define variables using the `=` sign:

```
1 >> x = 1
2 x = 1
```

In English, this statement means: *set the value of x equal to 1*. We can now do basic arithmetic with *x*:

```
1 >> y = x + 2
2 y = 3
3 >> y / 2
4 ans = 1.500
```

The variable **ans** is the default variable name that MATLAB uses if you don't specify a variable in the interpreter.

You can suppress the displaying of the answer using a semi-colon at the end of each line. Compare the following two lines of code (note the semi-colon):

```
1 >> z = x - 5
2 z = -4
3 >> z = x - 5;
```

In both cases **z** has the value **-4**, but this is only displayed on the screen in the first case.

If you set a variable to a new value, it replaces the value previously held in that variable:

```
1 >> x = 100;
```

So **x** now has the value **100**, not **1** that we assigned to it earlier.

In physics we usually write numbers in scientific notation; for example the speed of light is often written as $c = 2.998 \times 10^8 \text{ ms}^{-1}$. To do this in MATLAB you can use the notation **e8** to represent $\times 10^8$. For example:

```
1 >> c = 2.998e8
2 c = 299800000
```

1.3.3 Scalars and Vectors

The power of computational science is in its application of multiple operations to large collections of numbers. MATLAB can store collections of numbers using n-dimensional *vectors*, also known as *arrays*. We can define a row or column vector in MATLAB as shown below:

```
1 >> a = [5 7 2]           % row vector
2 a = 5 7 2
```

When you define a column vector, the semicolon represents a new line:

```
1 >> b = [5; 7; 2]         % column vector
2 b = 5
3     7
4     2
```

We can access each element in a vector using its *index* (position in the vector). For example:

```
1 >> a = [100 22 pi];
2 >> a(1)
3 ans = 100
```

MATLAB vectors and arrays are indexed from 1, not from 0 as in some other languages. The first element of this vector is given by **a(1)** and the last element of this vector is **a(3)**. Alternatively, we can use the built-in variable called **end** to access the last element:

```
1 >> a(end)
2 a = 3.1416
```

Although MATLAB knows the value of π to much greater accuracy, only 4 decimal places are displayed by default. To show more, you can change the format :

```
1 >> format long
2 >> a(end)
3 a = 3.141592653589793
```

To get back to the default format setting type **format short**;

MATLAB vectors are a core part of the language, and so we can perform addition and subtraction on them just as we can with scalars (individual numbers):

```
1 >> a = [2 4 6];
2 >> b = [1 2 2];
3 >> c = a + b
4 c = 3 6 8
5 >> d = a - b
6 d = 1 2 4
```

These operations work on the vectors one element at a time, provided the vectors have the same dimensions. If they don't, MATLAB will return an error:

```
1 >> a = [2 4 6];
2 >> c = [3; 3; 3]
3 c = 3
```

```

4      3
5      3
6  >> a - c
7  ??? Error using ==> minus
8  Matrix dimensions must agree

```

In this case we have tried to subtract a column vector (3×1) from a row vector (1×3) which isn't possible.

Multiplication and division are slightly more tricky. By default MATLAB assumes everything is a matrix, and so it tries to do matrix multiplication when given two vectors:

```

1  >> a = [2 4 6];
2  >> b = [3; 3; 3];
3  >> c = a * b
4  c = 36

```

Note that the vectors must have the right dimensions for this to work — in this case a row vector (1×3) and a column vector (3×1) resulted in a (1×1) matrix. We'll do more on matrix multiplication in later weeks.

If we want to do element-by-element multiplication or division we have to use the *dot* version of the \star or $/$ operator, and the vectors must be of the correct dimensions:

```

1  >> a = [2 4 6];
2  >> b = [1 2 2];
3  >> c = a .* b
4  c = 2 8 12

```

If we want to add a scalar to every element in a vector we can do:

```

1  >> a = [7 5 1];
2  >> b = a + 10
3  b = 17 15 11

```

To divide a scalar by a vector, we have to use element-wise division (note the dot):

```

1  >> a = [2 5 -10];
2  >> b = 1./a
3  b = 0.5000 0.2000 -0.1000

```

1.3.4 Naming variables

One way of making your code easier to understand is to name your variables in a meaningful, but concise, way. For example, if we want to calculate the velocity, given a car travelled 10 km in 15 minutes, we could do:

```

1  >> dist = 10; % km
2  >> time = 15/60.0; % hr
3  >> vel = dist/time

```

MATLAB (and most other programming languages) doesn't keep track of the units. If you want to make a note of what units your variables are in, you can add a *comment* using the percent symbol, as we have done above (% km).

Another sensible choice of variable names for this example might be:

```

1  >> s = 10; % km
2  >> t = 15/60.0; % hr
3  >> v = s/t

```

as these are the standard symbols you probably used in high school. Choosing random names makes your code difficult to understand. For example:

```

1  >> cat = 10;
2  >> dog = 15/60.0;
3  >> pig = cat/dog

```

This code will give the correct answer, but people reading it (including yourself) will have no idea what it means! You can go too far in the other direction though. If you try to include a full description of each value your code

will also be difficult to read (and you'll have to do a lot of extra typing). For example:

```
1 >> distance_travelled_in_km = 10;  
2 >> time_taken_in_hours = 15/60.0;  
3 >> velocity_of_car = distance_travelled_in_km/time_taken_in_hours
```

Good coding is about finding a balance between these extremes.

Finally, you should avoid giving your variables names that conflict with well known physical constants, or other standard symbols like **pi**, **e** and **i**. For example, calling your velocity **c** would be confusing because **c** usually refers to the speed of light in a vacuum.

1.4 Exercises

Question 1

Given the two vectors **a** and **b** below:

```
1 a = [4 8 12 16 20];  
2 b = [2 4 2 4 2];
```

Write MATLAB code to

1. Subtract each element of **b** from the corresponding element of **a**.
2. Multiply each element of **a** by 7.
3. Add the first element of **a** to the last element of **b**.
4. Divide every element of **a** by 2.
5. Divide each element of **a** by the corresponding element of **b**.

Write your code in the box below.

```
1 a - b  
2 a*7  
3 a(1) + b(end)  
4 a/2  
5 a./b
```

Question 2

The speed of light in a transparent medium is given by $v = \frac{c}{n}$ where c is the speed of light in a vacuum ($c = 2.9979 \times 10^8 \text{ ms}^{-1}$) and n is the refractive index of the medium.

Two friends separated by 1000 km signal to each other using an extremely powerful torch. Use **MATLAB** as a calculator to work out how long the signal will take to arrive if the space between them is filled entirely by cold air ($n = 1.000293$ at 0° C and 1 atm).

```
1 >> c = 2.9979e8;
2 >> dist = 1000 * 1000;
3 >> n_air = 1.000293;
4 >> v_air = c/n_air;
5 >> time = dist/v_air
```

The signal will take 3.34 milliseconds to arrive.

Tutor note: Encourage students to use variables, rather than just 'magic numbers'.

Question 3

Work out how long the signal will take to arrive if, instead of cold air, the friends find themselves separated by each of the substances in the table below:

Substance	n
Carbon dioxide	1.00045
Water (at 20°)	1.3330
Ethyl alcohol	1.3600
60% Glucose solution in water	1.4394
Crown glass	1.52
Diamond	2.42

Hint: You should be able to do this in exactly the same way as the previous question, but using a vector of values for n . Remember the power of MATLAB is that you don't need to calculate each case individually.

```
1 c = 2.9979e8; % m/s
2 dist = 1000; % km
3 n = [1.00045 1.3330 1.3600 1.4394 1.52 2.42];
4 v = c./n;
5 t = dist*1000./v % s
```

Carbon dioxide = 0.0033s; Water = 0.0044s; Alcohol = 0.0045s; Glucose = 0.0048s; Glass = 0.0051s; Diamond = 0.0081s.

Tutor note: Remind students about ./ for vectors. You may need to help them define the vector of values.

Checkpoint 1:

1.5 Walkthrough: Arrays and plotting

1.5.1 Creating arrays

One of the advantages of MATLAB is its sophisticated visualisation capabilities. As a computational physicist this means you can plot and visualise your results directly, without having to learn or run a separate plotting package.

In this lab we will start with some very simple plotting. Firstly, we need to learn how to create vectors without manually inputting values.

MATLAB provides several ways of creating vectors of numbers over a specified range. For example, to make a vector with values ranging from 1 to 10 in steps of 2:

```
1 >> x = 1:2:10
2 x = 1 3 5 7 9
```

If we leave out the middle parameter (the step size), MATLAB will default to a step size of 1, for example:

```
1 >> d = 0:10
2 d = 0 1 2 3 4 5 6 7 8 9 10
```

The step size can be fractional:

```
1 >> x = 1:0.5:5
2 x = 1.0000 1.5000 2.0000 ... 4.5000 5.0000
```

and you can also step in reverse:

```
1 >> x = 1:-0.2:0
2 x = 1.0000 0.8000 0.6000 0.4000 0.2000 0
```

An alternative is the `linspace` function:

```
1 >> d = linspace(0, 10, 11)
2 d = 0 1 2 3 4 5 6 7 8 9 10
```

where the first argument is the starting value (0), the second is the end value (10) and the third is the number of values in the array (11). This creates a vector of 11 values linearly spaced between 0 and 10.

You can find more information by using `help linspace`.

1.5.2 Plotting

Creating 2D plots in MATLAB is straightforward. Typically you first create one array holding your x-values and another holding your y-values. You can then call the `plot()` function with various formatting options.

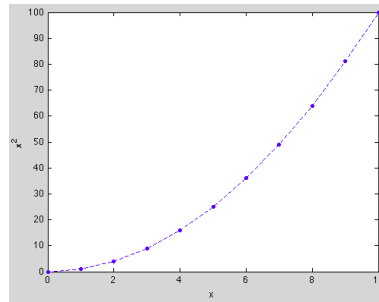
For example, to plot the function x^2 between 0 and 10 (inclusive):

```
1 >> x = [0:10]
2 x = 0 1 2 3 4 5 6 7 8 9 10
3 >> y = x.^2
4 y = 0 1 4 9 16 25 36 49 64 81 100
5 >> plot(x, y)
```

There are a couple of things to note about this syntax. Firstly, we define an array with values from 0 to 10 using `[0:10]`. We could change to increment to 2 by using `[0:2:10]` (experiment in MATLAB to see the effect of this). Secondly we want to square every individual value in the x vector, so we need to use the *dot* notation `x.^2` to do a scalar rather than matrix operation.

To make our plot scientifically acceptable we should add some axis labels. We might also want to change the formatting such as line colour. You can get more information on the formatting options using `help plot`.

```
1 >> plot(x, y, '-b.') % Note there are two dashes here
2 >> xlabel('x')
3 >> ylabel('x^2')
4 >> title('A plot of f(x) = x^2')
```



If we wanted to keep such plots, and overlay other plots, we can use the **hold** command. To start this process, make sure that the figure is still open, then type the commands:

```
1 >> hold on
2 >> plot(x, x.^3, 'r.')
3 >> hold off
```

hold on turns it on and **hold off** turns it off.

1.5.3 Built-in functions

MATLAB has a wide range of built-in functions that make mathematical calculations easy. These functions operate on single numbers (scalars) as well as vectors and matrices. For example to calculate the sin of integer angles between 1 and 10 radians:

```
1 >> x = [1:10];
2 >> y = sin(x)
3 y =
4     0.8415     0.9093     0.1411    -0.7568    -0.9589    -0.2794     0.6570     0.9894     0.4121    -0.5440
```

If we wanted to find the sum of all of these values we could do:

```
1 >> sumy = sum(y)
2 sumy = 1.4112
```

Some of the common mathematical functions you might need are listed in the table below.

Operation	MATLAB command
$\cos(a)$, $\sin(a)$, $\tan(a)$, $\cos^{-1}(a)$, $\sin^{-1}(a)$, $\tan^{-1}(a)$	cos(a) , sin(a) , tan(a) , acos(a) , asin(a) , atan(a)
\sqrt{a} ,	sqrt(a)
a^b ,	a^b
e^a , $\ln(a)$, $\log_{10}(a)$,	exp(a) , log(a) , log10(a)
Absolute value, $ a $	abs(a)
Summation: $\sum_i a_i$	sum(a)
Minimum value of a vector a	min(a)
Maximum value of a vector a	max(a)

Note that trigonometric functions work in radians by default. To work in degrees you can use the version with a **d** after the name, e.g. **sind(a)**.

1.5.4 Scripting

Typing in each command repeatedly is inefficient and error-prone, so a better option is to save these commands into a script so you can modify and reuse them. You can open the MATLAB editor by typing **edit** into the command window, or choosing New-Script from the menu. If you wrote your solution to Question 1 and saved it as, say, **lab1q1.m**, you can then run the solution at any time by typing the filename into the command window:

```
1 >> lab1q1
2 t = 0.0033
```


1.6 Exercises

Question 4

Write a MATLAB program to

1. Create a vector **a** containing the integers from 45 to 55 (inclusive).
2. Calculate the sum of the integers in vector **a**.
3. Create a vector **b** containing the odd numbers between 15 and -15.
4. Create a vector **c** containing the absolute value of the integers in **b**.
5. Create a vector **d** with numbers going from 1 to 2 in steps of 0.1.

Write your code in the box below. **Make sure you save it as a script.**

```
1 a = 45:55
2 sum(a)
3 b = 15:-2:-15
4 c = abs(b)
5 d = 1:0.1:2
```

Tutor note: From this point you should insist that students write all of their code as scripts, saved with a sensible name, e.g. lab1q4.m. Emphasise the importance of being able to reuse solutions (and keeping them for future study).

Question 5

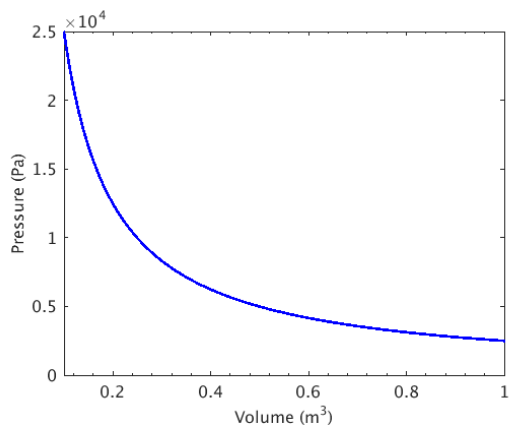
The equation of state for an *ideal* gas is

$$pV = nRT \quad (1.1)$$

where p is the pressure (in Pa), V is the volume (in m^3), n is the number of moles and T is the temperature (in K). R is the universal gas constant, $R = 8.31 \text{ J mol}^{-1} \text{ K}^{-1}$.

Create a vector **v** with volume values ranging from 0.1 to 1 m^3 (why not from 0 to 1 m^3 ?). Apply the ideal gas equation to calculate a vector with corresponding values of pressure for 1 mole of nitrogen at room temperature (300 K)

Plot pressure versus volume sketch your plot below. Now zoom in on your plot (click and drag using the zoom icon) to estimate the volume at a pressure of 1 atmosphere ($1 \times 10^5 \text{ Pa}$).



A calculation of the volume gives 0.02493 m^3 . If students get an answer of 0.0259 m^3 , they have not used small enough volume steps.

Tutor note: ensure all axes are labelled and numbered.

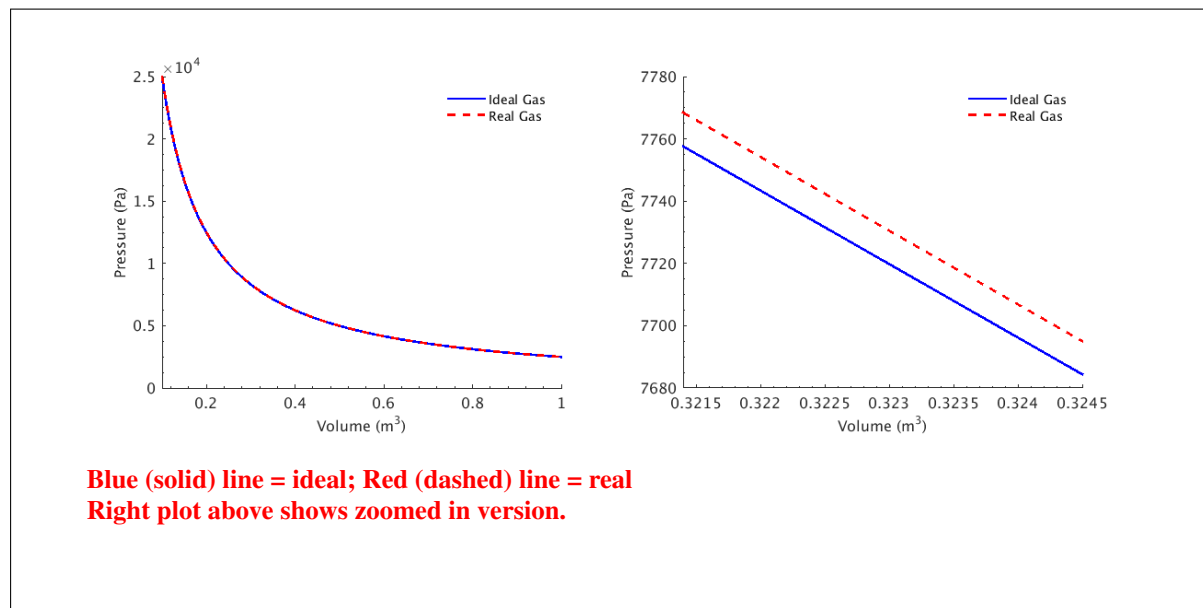
Question 6 (PHYS2011)

The equation of state for a *real* gas is given by the Van der Waals equation:

$$\left(p - \frac{n^2 a}{V^2}\right)(V - nb) = nRT \quad (1.2)$$

where constants a and b take account of long-range repulsive forces and short-range attractive forces between molecules.

On the same graph as in Question 5, plot p vs. V for 1 mole of nitrogen at room temperature (300 K) using this equation. For nitrogen, $a = 1.427 \times 10^{-4} \text{ Pa m}^6 \text{ mol}^{-2}$ and $b = 3.913 \times 10^{-4} \text{ m}^3 \text{ mol}^{-1}$. Sketch both plots on the same axes below.



You will have to zoom it to see any differences between these plots!

Question 7 (PHYS2911 and PHYS2921)

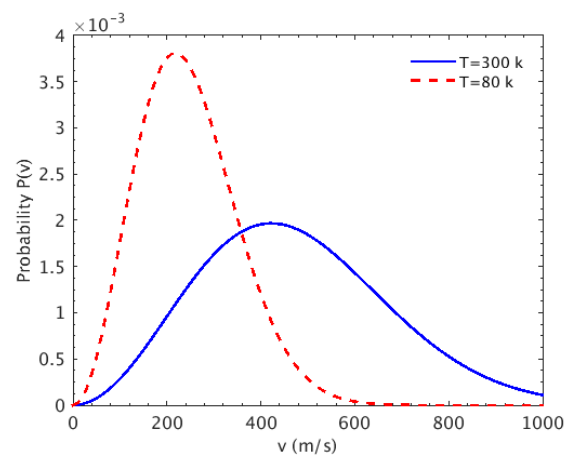
The distribution of molecular speeds v in a gas is given by

$$P(v) = 4\pi \left(\frac{M}{2\pi RT}\right)^{\frac{3}{2}} v^2 e^{-\frac{Mv^2}{2RT}} \quad (1.3)$$

where v is the speed (in m s^{-1}), M is the mass of 1 mole (in kg), and T is the temperature (in K). $R = 8.31 \text{ J mol}^{-1} \text{ K}^{-1}$ is the universal gas constant.

Plot $P(v)$ vs. v for nitrogen for two temperatures (80 K and 300 K) and sketch your plot below. Estimate the speed corresponding to the peak of the distribution in each case. For nitrogen, $M = 28.0 \times 10^{-3} \text{ kg mol}^{-1}$.

Try v ranging from 0 to 1000 m s^{-1} .



Blue (solid): $T = 300 \text{ K}$

Red (dashed): $T = 80 \text{ K}$

Speeds at peak 218 m/s and 424 m/s. They can also use $v1=v(P1==\max(P1))$ *Tutor note: They can just duplicate code for each T.*

Remember to explain hold on/off.

The equation is tricky to get right in this one - look for typos. A common error is to write $\wedge 3/2$ instead of $\wedge (3/2)$

Checkpoint 2:

1.7 Challenge: The Cosmic Microwave Background radiation

A *black body* is an idealised object that absorbs all incident electromagnetic radiation. No radiation is reflected or transmitted through the body. A black body in thermal equilibrium emits radiation with spectral radiance B_ν at frequency ν according to Planck's Law:

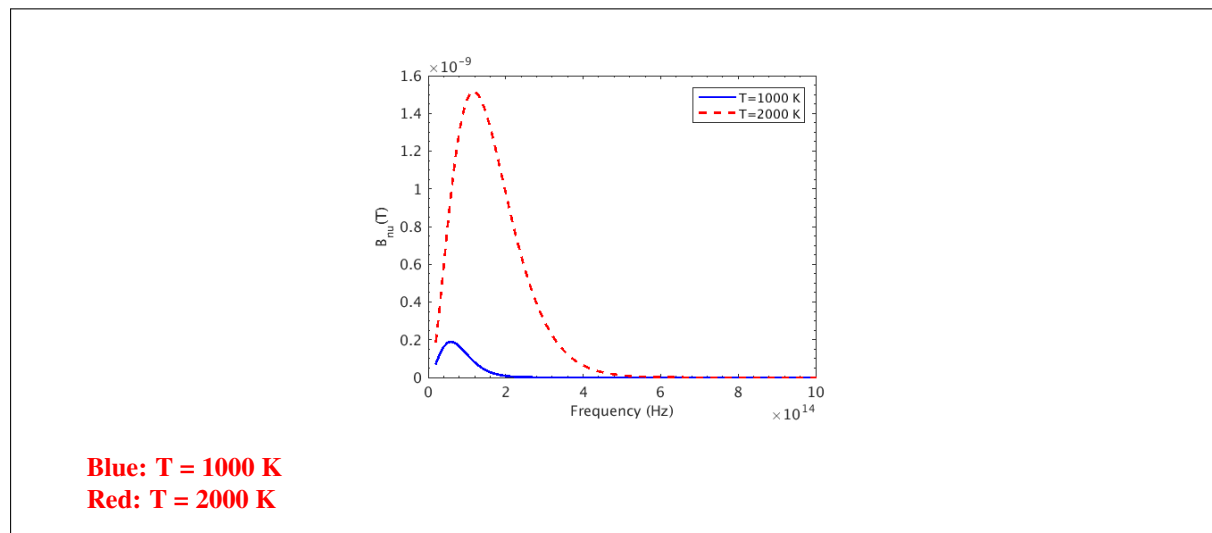
$$B_\nu(T) = \frac{2h\nu^3}{c^2} \frac{1}{e^{\frac{h\nu}{k_B T}} - 1} \quad (1.4)$$

where h is Planck's constant $h = 6.62606957 \times 10^{-34} \text{ m}^2 \text{ kg s}^{-1}$, k_B is the Boltzmann constant $k_B = 1.3806488 \times 10^{-23} \text{ m}^2 \text{ kg s}^{-2} \text{ K}^{-1}$ and c is the speed of light in a vacuum $c = 2.9979 \times 10^8 \text{ m s}^{-1}$. The frequency ν is measured in Hz, resulting in a spectral radiance B_ν in units of $\text{W sr}^{-1} \text{ m}^{-2} \text{ Hz}^{-1}$.

Note that in astronomy and astrophysics, spectral radiance is usually referred to as *intensity*.

Question 8

Plot the Planck spectrum for $T = 1000\text{K}$ and $T = 2000\text{K}$ in the frequency range $2 \times 10^{13} \leq \nu \leq 3 \times 10^{15} \text{ Hz}$ (approximately $100 \leq \lambda \leq 15\,000 \text{ nm}$). Sketch the plots below.



Question 9

In 1989 the COBE satellite was launched, with the aim of measuring the Cosmic Microwave Background (CMB) radiation — the thermal radiation left over from the Big Bang. The spectrum of this radiation was found to be an almost-perfect black body. Two of the COBE principle investigators, Smoot and Mather, won the 2006 Nobel Prize for Physics for this work.

You can read in the original COBE data from the file `blackbody.txt` provided using the following code

```
1 >> data = dlmread('blackbody.txt');
2 >> frequency = data(:, 1);
3 >> intensity = data(:, 2);
```

This produces two vectors `frequency` and `intensity` that contain the measured data. The frequency is in units of GHz and the intensity is in units of MJy sr^{-1} . Jy (Jansky) is a non-SI unit used in astronomy: $1\text{Jy} = 10^{-26} \text{ W m}^{-2} \text{ Hz}^{-1}$.

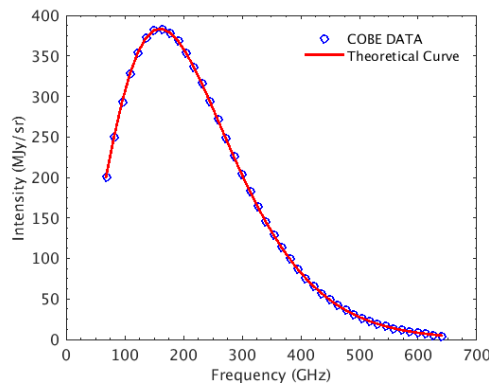
Plot this experimental data and sketch your plot below.

Hint: You should use points to plot experimental data, not lines. To do this you can use a plot command like `plot(frequency, intensity, 'bo')`.

```

1  c = 299792458;
2  figure()
3
4  % Read and plot COBE data
5  data = dlmread('test.out');
6  frequency = data(:,1);
7  intensity = data(:,2);
8  plot(frequency, intensity, 'bo');
9  xlabel('Frequency (GHz)');
10 ylabel('Intensity (MJy/sr)');
11 hold on;
12
13 % Plot theoretical BB spectrum
14 T = 2.725;
15 freq_hz = frequency * 1e9;
16 BnuT = planck_f(T, freq_hz);
17 BnuT_MJy = BnuT ./ (10^-26 * 1e6);
18 plot(frequency, BnuT_MJy, 'r')

```



Tutor note: students may need help downloading the blackbody.txt file into the correct directory.

Question 10

From this data, the temperature of the CMB was measured to be 2.725 K. Confirm this by plotting the theoretical Planck spectrum for $T = 2.725$ K. Add your theoretical line to the plot in the previous section.

You will have to do some unit conversions to make sure your values are in the same units as the experimental data (which has ν in GHz and B_ν in MJy sr^{-1}).

An interesting note is that the uncertainties on the COBE measurements are so small that the error bars are smaller than the thickness of the plotted line. This makes the measurement of the CMB one of the most precise scientific experiments ever conducted.

1.8 Extra information and references

The MathWorks website has a student centre with introductory tutorials and videos that cover MATLAB basics:

http://www.mathworks.com.au/academia/student_center/tutorials/launchpad.html

To look up how to use specific functions you can use the online documentation:

<http://www.mathworks.com.au/help/matlab>

The COBE black body spectrum comes from:

http://lambda.gsfc.nasa.gov/product/cobe/firas_prod_table.cfm

For more information about the COBE experiment and the Cosmic Microwave Background radiation, have a look at the NASA site:

http://lambda.gsfc.nasa.gov/product/cobe/c_edresources.cfm