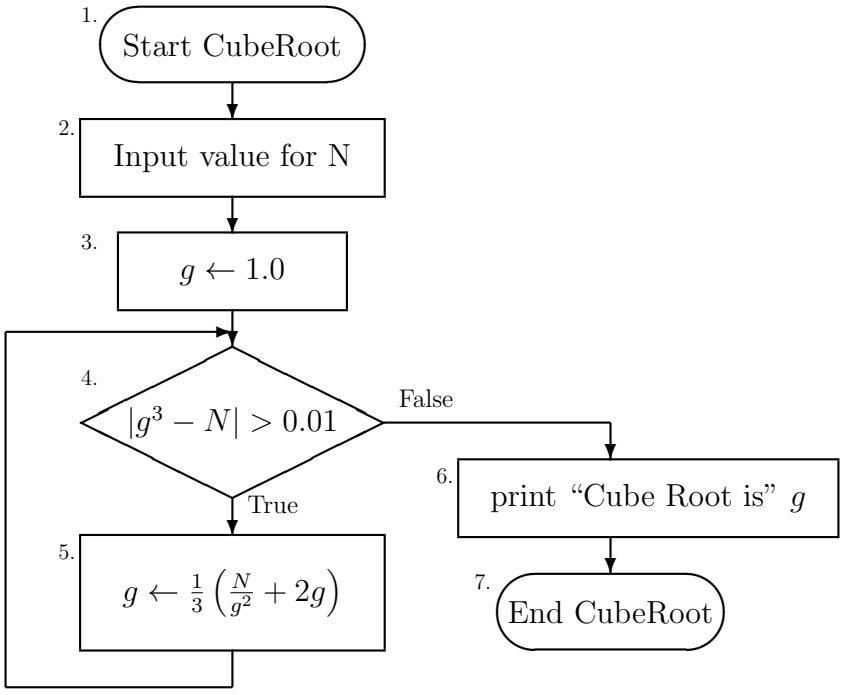
MECH2700 Engineering Analysis I, 2015
Exercise Sheet 2

Iterative algorithms and plotting using python

1. The flowchart below represents an algorithm for computing the cube root of a number N. An initial guess g is set to the value of 1.0 and this guess is improved (using the computation in box 5) until it is close enough. Trace the execution of the algorithm for $N = 27$.



Follow the flowchart above from start to finish, keeping track of the execution in a table as shown below. Of course, do this in your workbook, pasting a copy of the flowchart onto the facing page for reference.

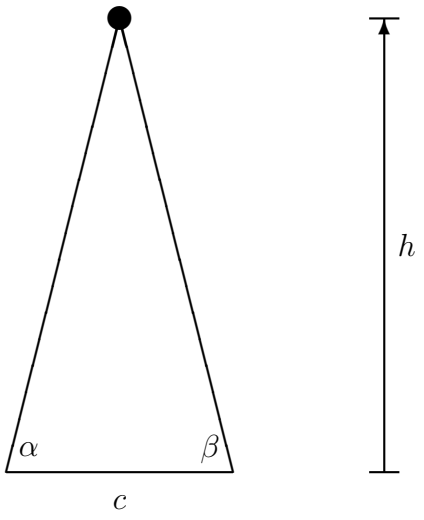
box number	decision/input/output	N	g
1	Start...	?	?
2	Input N as 27	27.0	
3			1.0
4	$\lvert g^3 - N \rvert > 0.01$ is true		
5	$g \leftarrow \frac{1}{3}\left(\frac{N}{g^2} + 2g\right)$		9.6667

2. Use Newton's Method to determine the expression in box 5 in the flowchart ($g \leftarrow \frac{1}{3}\left(\frac{N}{g^2} + 2g\right)$). Newton's method is an iterative method that is used to solve $f(x) = 0$. Each iteration is given by:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

Start with $f(x) = x^3 - N$.

3. One way of measuring the distance to a an object is by using parallax. Given a baseline of known length c, measure the angles α and β between the baseline and the lines of sight to the object.



Show that one may compute the distance from the baseline to the object as:

$$h = c\,\frac{\sin\alpha\ \sin\beta}{\sin(\alpha+\beta)}$$

The trigonometry identity $\sin(x+y) = \sin x \cos y + \cos x \sin y$ may be useful. Note that you don't have to work only from the diagram to the final expression. You may take the final expression and work backwards as well. The point is to verify a formula that you have been given or found in a text.

4. Write a Python program to compute the distance for the values $c = 10$ metres, $\alpha = 89.9$ degrees and $\beta = 89.8$ degrees. Paste your code and the results from running your code into your workbook.

5. Express the following algebraic equations as Python assignment statements. Long equations should be divided into several small statements. You may need to look up the relevant sections in your reference materials to complete this exercise.

   - Manning's equation for open channel flow

   $$Q = \frac{1.49}{n}(A)(R_h)^{\frac{2}{3}}S_0^{\frac{1}{2}}$$

   Start with these pieces:

   ```
   term1 = 1.49 * A/n
   term2 = Rh **
   term3 =
   Q =
   ```

   - The equivalent resistance for a certain parallel-series circuit

   $$R_{tot} = \frac{1}{\frac{1}{R_2+\left[\frac{1}{\frac{1}{R_3}+\frac{1}{R_4}}\right]} + \frac{1}{R_5}}$$

   ```
   R34 =
   R234 =
   Rtot =
   ```

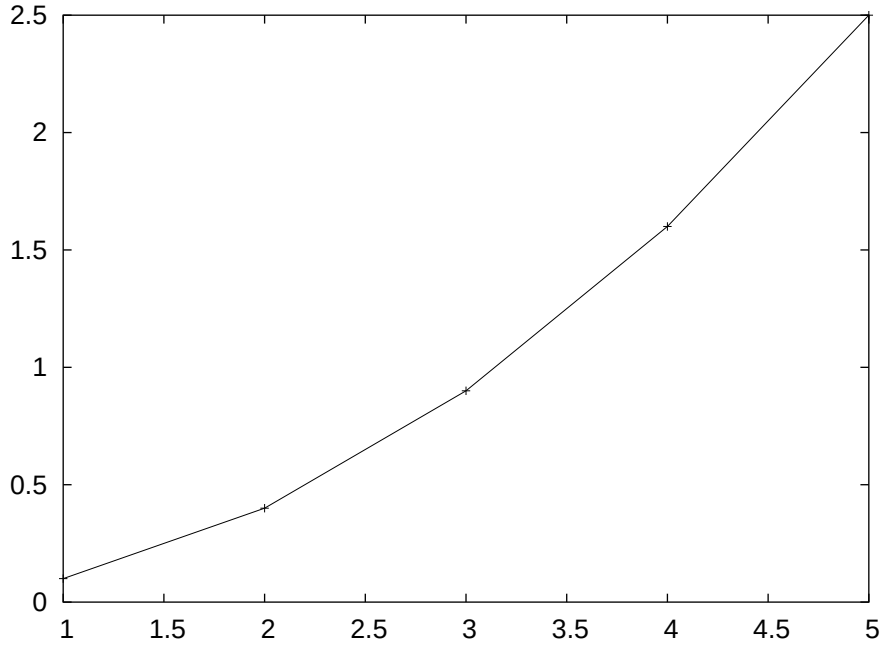
   - The static pressure in a gas relative to that at a stagnation point

   $$p = p_0\left[1 + M^2\left(\frac{k-1}{2}\right)\right]^{\left(\frac{-k}{k-1}\right)}$$

6. MATPLOTLIB is an extension to Python that provides a MATLAB-like analysis and plotting capability. The following transcript shows the use of the package to plot a set of 5 data points whose coordinates were stored as the lists x and y. Note that you need to enter the text on each line following the Python prompt >>>.

```
>>> from pylab import *
>>> x = [1, 2, 3, 4, 5]
>>> y = [0.1, 0.4, 0.9, 1.6, 2.5]
>>> plot(x, y, '+-')
>>> savefig('myplot')
```

   The function call `plot()` presents a graph on the screen while the `savefig()` call generates a *png* file that can be later printed or can be included in word-processor documents as done here:



There are many more options for controlling details such as axis labels, title, line styles and the like. If you want to know more about the matplotlib-pylab extension, there is a User's Guide on the web. For a brief description of a function such as `plot()`, type `help(plot)` at the Python prompt.

If your graph does not appear immediately, use the following command to make it appear.

```
>>> show()
```

The exercise here is to use the program from Exercise Sheet 1 to estimate the value of π using Machin's formula. Generate a number of estimates using between 3 and 14 terms in the arctangent expansions. The estimation error should have been tabulated at the end of Section 1. Plot the magnitude of the errors between the estimates and the correct value as a function of the number of terms in the arctangent expansions using MATPLOTLIB. Because we expect a large range of error values, use a logarithmic scale on that axis of the plot. What happens when you use more than 10 terms? Also, plot the errors for estimates of arctan(1.0) ($= \frac{\pi}{4}$) for the same number of terms. Functions `semilogy` and `hold` could be useful to generate this graph.

Write a brief description on how to use the function `semilogy`, and why this would be useful in this case

Explain how you would use the function `hold` to plot multiple data sets on the one graph.

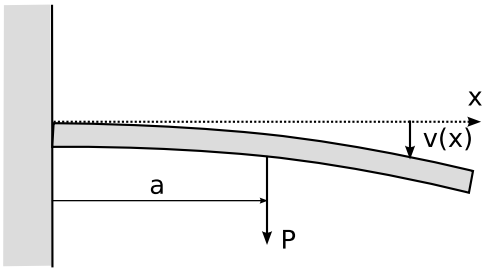
Paste in the code that you used to generate your plot of the approximation error. *Hint: You can load the error from the tables in Section 1 directly into lists. Use the example in this question as your guide* Paste the graph that your code generates into your workbook, as well.

What happens to the errors in Machin's formula when you use more than 10 terms?

Comment on the differences between the error curves for Machin's formula and the arctan curve.

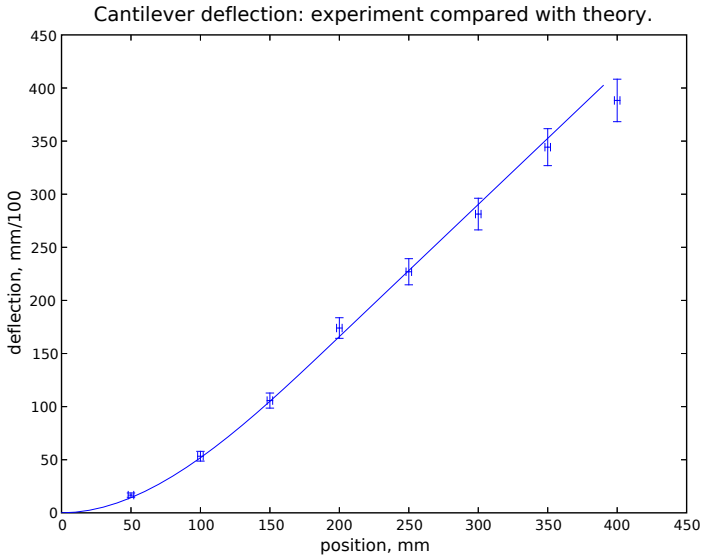
7. In one of your solid mechanics practicals, you might measure the deflection of a cantilever beam (length L) with a load P applied at $x = a$. The theoretical deflection for an elastic beam is

$$\begin{aligned} v(x) &= \frac{Px^2}{6EI}(3a - x) \quad , \quad 0 \le x \le a \\ v(x) &= \frac{Pa^2}{6EI}(3x - a) \quad , \quad a \le x \le L \end{aligned}$$



At various locations x along the beam, you will measure the vertical deflection $v(x)$ with a dial gauge. Here is a set of representative measurements with their estimated uncertainties and a plot of this data.

x, mm	δx, mm	$v(x)$, 10^{-2} mm	δv, 10^{-2} mm
50	2	16.7	2.0
100	2	53.3	4.6
150	2	105.7	7.1
200	2	174.0	9.7
250	2	227.0	12.3
300	2	281.3	14.9
350	2	344.3	17.4
400	2	388.3	20.0



Given these data, the following code snippet sets up lists of the experimental data and some theoretical data for comparison.

```
# cantilever.py
x  = [ 50.0, 100.0, 150.0, 200.0, 250.0, 300.0, 350.0, 400.0]
dx = [  2.0,   2.0,   2.0,   2.0,   2.0,   2.0,   2.0,   2.0]
v  = [ 16.7,  53.3, 105.7, 174.0, 227.0, 281.3, 344.3, 388.3]
dv = [  2.0,   4.6,   7.1,   9.7,  12.3,  14.9,  17.4,  20.0]

from pylab import *
def theoretical_v(x_mm):
    """
    Compute cantilever deflection for a fixed load.
    Input:   x_mm : position from fixed-end, in mm
    Returns: v    : deflection in hundredths of a mm
    """
    P = 1.96      # load, Newtons
    a = 200.0e-3  # position of load, metres
    E = 70.0e9    # Pascals
    I = 45.0e-12  # metres**2
    x = x_mm / 1000.0
    if x < a:
        v = P * x**2 / (6*E*I) * (3*a - x)
    else:
        v = P * a**2 / (6*E*I) * (3*x - a)
    return v * 1.0e5

# Build up a list of theoretical deflections
tx = []; tv = []
for position in arange(0.0, 400.0, 10.0):
    tx.append(position)
    tv.append(theoretical_v(position))

# Now, make the plot using the pylab functions...
# ...
```

Complete the code to produce the figure comparing experimental measurements with the theoretical deflection. Include your notes on the MATPLOTLIB/pylab functions and your completed program in your workbook.