Week 2

Iterative Algorithms
Q1

Box Number	Decision / input / output	N	g
1	Start…	?	?
2	Input N as 27	27.0	
3			1.0
4	l g^3 - N l > 0.01 is true	27	
5	g <— 1/3 (N/g^2 + 2g)		9.6667
4	l g3 - N l > 0.01 is true	27	
5	g <— 1/3 (N/g^2 + 2g)		6.541
4	l g3 - N l > 0.01 is true	27	
5	g <— 1/3 (N/g^2 + 2g)		4.571
4	l g3 - N l > 0.01 is true	27	
5	g <— 1/3 (N/g^2 + 2g)		3.478
4	l g3 - N l > 0.01 is true	27	
5	g <— 1/3 (N/g^2 + 2g)		3.063
4	l g3 - N l > 0.01 is true	27	
5	g <— 1/3 (N/g^2 + 2g)		3.001
4	l g3 - N l > 0.01 is true	27	
5	g <— 1/3 (N/g^2 + 2g)		3.000
4	l g3 - N l > 0.01 is false	27	
6	print “Cube Root is” g		3.00
7	End CubeRoot function	27	3

Q2 Cube Root calculation using Newton's Method

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	27	
5	$g \leftarrow g - ((g^3 - N) / 3g^2)$		9.6667
4	$\lvert g3 - N \rvert > 0.01$ is true	27	
5	$g \leftarrow g - ((g^3 - N) / 3g^2)$		6.541
4	$\lvert g3 - N \rvert > 0.01$ is true	27	
5	$g \leftarrow g - ((g^3 - N) / 3g^2)$		4.571
4	$\lvert g3 - N \rvert > 0.01$ is true	27	
5	$g \leftarrow g - ((g^3 - N) / 3g^2)$		3.478
4	$\lvert g3 - N \rvert > 0.01$ is true	27	
5	$g \leftarrow g - ((g^3 - N) / 3g^2)$		3.063
4	$\lvert g3 - N \rvert > 0.01$ is true	27	
5	$g \leftarrow g - ((g^3 - N) / 3g^2)$		3.001
4	$\lvert g3 - N \rvert > 0.01$ is true	27	
5	$g \leftarrow g - ((g^3 - N) / 3g^2)$		3.000
4	$\lvert g3 - N \rvert > 0.01$ is false	27	
6	print "Cube Root is" g		3.00
7	End CubeRoot function	27	3

This method produces the same result with the same number of iterations however, Newton's method is supposedly faster and more robust method for larger and more complex functions.

Q3 Parallax Distance

Come back!!

Ryan Mitchell
4185 0178

Q4 Parallax Distance Code

```
from math import *
def parallax_dist(alpha, beta, c):
    alpha = float(alpha)
    beta = float(beta)
    c = float (c)
    h = c * (sin(radians(alpha)) * sin(radians(beta))) / (
        sin(radians(alpha + beta)))
    #print float("%.4f" %h)
    return h
```

Q5.1 Manning's Equation

```
term1 = 1.49 * A / n
term2 = Rh ** (2/3)
term3 = S0 ** (2/3)
Q = term1 * term2 * term3
```

Q5.2 Equivalent Resistance

```
R34 = (1/R3 + 1/R4) ** -1
R234 = R2 + R34
R2345 = 1/R234 + 1/R5
Rtot = R2345 ** -1
```

Q5.3 Static Pressure

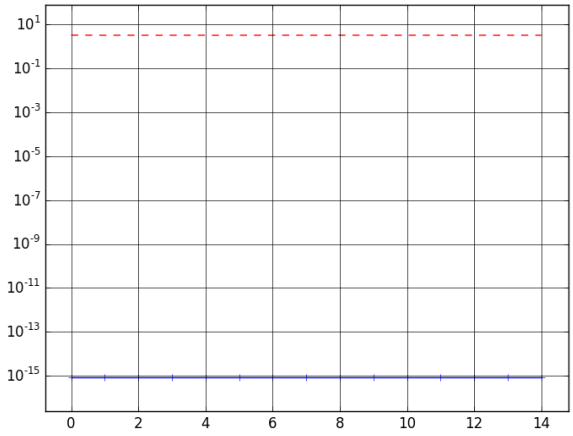
```
P0 = P0
M2K = 1 + M*M * ((k-1) / 2)
K = -k / (k-1)
p = P0 * M2K ** K
```

Q6 MATPLOTLIB

Semilog - is a function that allows one axis to have a log scale. The scale letter is appended directly to the smiling function i.e. semiology or semilogx. For both axis to have a log scale loglog must be used.

hold - is a function that allows the user to add multiple graphs onto a single set of axis. This is a rather cumbersome way of handling the data when plot(), semilog[x or y]() and loglog() both allow multiple data input. However, this is excellent for when the data sets are different in size.

On the graph the red dotted line is the estimation and the solid blue line is the error. If more than 10 terms are used then the error value dies off and the series estimation converges.



```
from pylab import *

# Matplotlib section
x_vals = [] # empty list for x values
y_vals = [] # empty list for y values
y_error = [] # empty list for y-error values

def atan_series(x, n):
    """Computes atan(x) with a truncated series expansion of n terms.
    Runs in the background allowing the machin function to execute."""
    xpower = x
    my_sum = x
    sign = 1
    for i in range(1, n):
        xpower = xpower * x * x
        sign = -1 * sign
        term = sign * xpower / (2 * i + 1)
        my_sum = my_sum + term

    #print("Pi is: ", my_sum)
    return my_sum

def machin(n):
    """Computes pi using Machin's formula. Utilises atan_series."""
    pi = 4 * (4 * atan_series(0.2,n) - atan_series(1.0/239,n))
    return pi

def machin_error(n):
    "Computes the error between pi and Machin's Formula"
    #print("Begin program machin_error")
    for i in range(1, n):
        error = fabs(machin(n) - pi)
        y_error.append(error)
    return error

if __name__ == "__main__":

    print("Begin program...")
    text = input("Enter a value for the number of terms: ")
    z = int(text)
    machin_error(z+1)
    machin(z+1)

    for i in range(1, z+1):
        pi = machin(i)
        x_vals.append(i)
        y_vals.append(pi)

    semilogy(y_error, '+-', y_vals, 'r--')
    grid(b=True, which='major', color='k', linestyle='-')
    show()
```

Q7 Cantilever Beams

Struggling to get the Legend command to work properly but the red is the theoretical and the blue is the experimental.