Assignment 3

Doug Shi-Dong 260466662
MATH-578 Numerical Analysis

October 22, 2015

Problem 2 (10.18)

The code is show below where the estimation is calculated versus the actual product. A plot comparing both of them are shown in Figure 1-2. The estimation is close in orders of magnitude to the exact product.

```
#!/usr/bin/env python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.backends.backend_pdf import PdfPages

start = 3
end   = 21
m     = (end-start)/2
w1 = np.empty(m+1)
w2 = np.empty(m+1)
ea = np.empty(m+1)
eb = np.empty(m+1)

a = 1.0
b = 1.7
for i, n in enumerate(range(start,end+1,2)):
data1 = np.linspace(0, 1, num = n+1)
    x1 = 1/(2.0*n)
    x2 = 0.5
    w1[i] = 1
    w2[i] = 1
    for xi in data1:
        w1[i] *= (x1-xi)
        w2[i] *= (x2-xi)
    ea[i] = abs(np.exp(-a*n))
    eb[i] = abs(np.exp(-b*n))
```

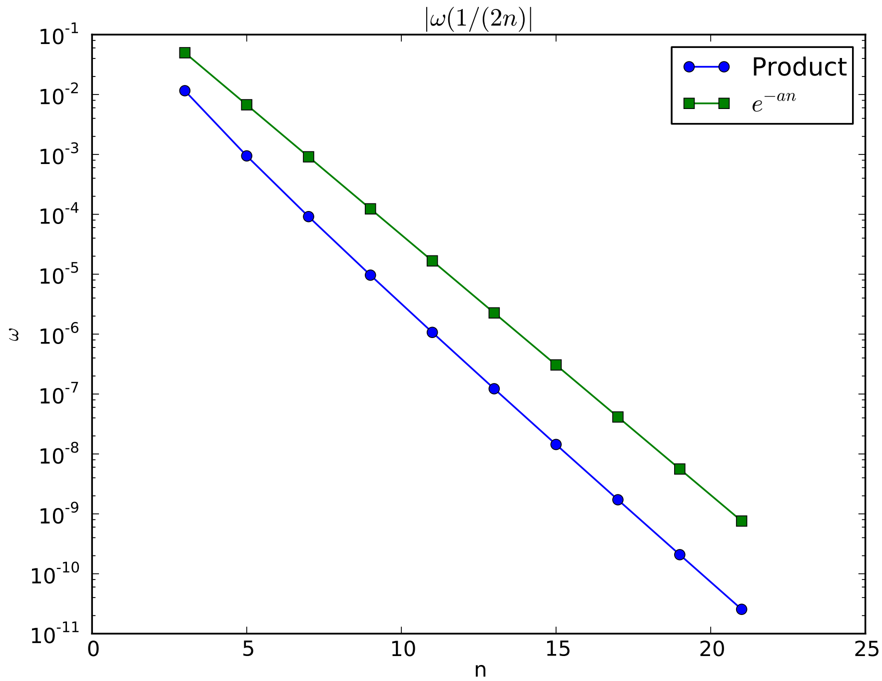

Figure 1: A figure

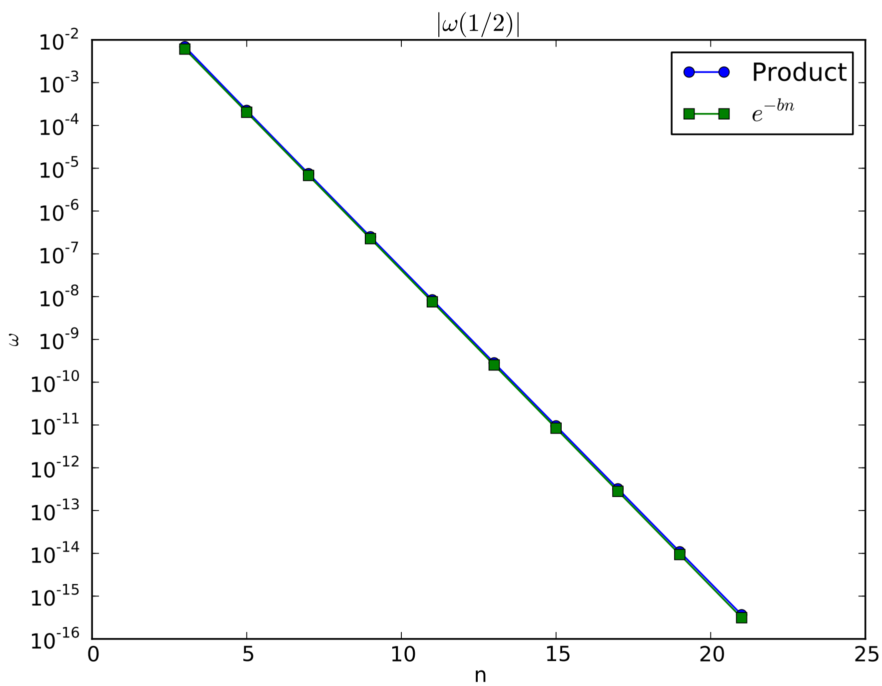

Figure 2: Another figure

Problem 5 (15.13)

The derivation has been done on paper. The code uses the same algorithm as shown in Additional Problems 9-11. The correct dominant value of -3.44948974278 has been found, as opposed to the other smaller eigenvalue of 1.44948974278. Note that the eigenvector keeps on flipping signs between [0.91209559 -0.40997761] and [-0.91209559 0.40997761]

```
Took 39 iterations
Eigenvalue
-3.44948974278
Eigenvector:
[ 0.91209559 -0.40997761]
```

Additional Problem 6

The code is shown below:

```
#!/usr/bin/env python
import numpy as np

data = np.array([[0,1,2,3],[3,5,-1,2]], dtype = np.float64)

n = data.shape[1]
x = 0.05

coeff = data[1,:]

for k in range(n-1):
    coeff[k+1:n] = (coeff[k+1:n] - coeff[k:n-1]) \
                   / (data[0,k+1:n] - data[0,0:n-k-1])

print 'Coefficients:'
print coeff

for ni in range (2,n+1):
    fx = 0
    for k in range(ni):
        fx += np.prod(x - data[0,0:k]) * coeff[k]
    print 'Degree %d Interpolation of %f is %f' % (ni-1,x, fx)
```

The interpolation for $x = 0.05$ is shown in Table 1.

Degree	x_c
1	3.1000
2	3.2900
3	3.5524

Table 1: Interpolation -8 $x = 0.05$

Additional Problem 7-8

Shown on paper.

Additional Problem 9

The same code has been used for problem 9, 10 and 11. It is shown after problem 11.
The convergence criterion is set to be the change in the Rayleigh's quotient. The power method converges to the dominant eigenvalue $\lambda_1 = 6$ in 46 power iterations.

```
Took 46 iterations
Eigenvalue
6.0
Eigenvector:
[ 0.80635149  0.57596535 -0.13439192]
```

Additional Problem 10

The Aitken's delta-squared method converged within 9 iterations. However, since this method computer two power iterations for each Aitken iteration, it is the equivalent of 18 power iterations.
Since the eigenvalue is not computed at each iteration, the convergence criterion is set to be the infinity norm of the change in the vector x.

```
Took 9 iterations
Eigenvalue
6.0
Eigenvector:
[ 0.80635149  0.57596535 -0.13439192]
```

Additional Problem 11

The eigenvalue converged to the eigenvalue of 3 as expected since it is the closest eigenvalue.

```
Took 22 iterations
Eigenvalue
3.0
Eigenvector:
[ 0.85714286  0.42857143 -0.28571429]
```

Power Method, Inverse Power Method and Aitken's Algorithms

```
#!/usr/bin/env python
import numpy as np

def rayleigh(A,x):
    return np.dot(x, np.dot(A,x))

def powerIt(A,x):
    y = np.dot(A,x)
    return y / np.linalg.norm(y,2)

nM = 2

# Matrix A
#A = np.array([[-4, 14, 0],
#              [-5, 13, 0],
#              [-1, 0 , 0]])

A = np.array([[-3, 1],
              [ 2, 1]])

# Convergence Information
tol = 1e-14
n = 50
eValResi  = 1
eVecResi  = 1
atkinResi = 1

# Inverse Power Method Options
useInv = 0 # Toggle on or off
if(useInv == 1):
    mu = 3.5
    B = A - np.identity(nM) * mu
    Binv = np.linalg.solve(B,np.identity(nM))

# Atkin Divided Difference to Speed Up Eigenvalue Convergence
useAtkin = 0 # Toggle on or off
if(useAtkin == 1):
    at = np.empty(n)
    xn0= np.empty(nM)
    xn1= np.empty(nM)
    xn2= np.empty(nM)

# Data Initialization
# Eigenvectors
x = np.empty((n,nM))
#x[0] = np.array([1,1,1])
x[0] = np.array([1,1])
# Eigenvalues
ra  = np.empty(n)
```

```
ra[0] = rayleigh(A,x[0])

if(useAtkin ==1):
    for i in range(n-1):
        xn0 = x[i]

        xn1 = powerIt(A,xn0)

        del1x = xn1-xn0
        if(max(abs(del1x)) < tol):
            break

        xn2 = powerIt(A,xn1)
        del2x = xn0-2*xn1+xn2

        x[i+1] = xn0 - del1x**2/del2x

    else:
        print 'Did not converge'
else:
    for i in range(n-1):
        if(useInv == 1):
            x[i+1] = powerIt(Binv,x[i])
        else:
            x[i+1] = powerIt(A,x[i])

        ra[i+1] = rayleigh(A,x[i+1])

        eValResi = abs(ra[i] - ra[i+1])
        if(eValResi < tol):
            break
    else:
        print 'Did not converge'

print x
print 'Took %d iterations' %i
print 'Eigenvalue'
if(useAtkin!=1):
    print ra[i+1]
else:
    print rayleigh(A,x[i])
print 'Eigenvector:'
print x[i-1]
```