Scientific Computing assignment 6

Alfred Ajay Aureate R

December 20, 2018

Problem 1) The following is the code for the python module **DirectDFT.py** with functions **DFT** for computing DFT of a function f and **iDFT** for computing the inverse DFT of a function F:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Dec 14 03:24:57 2018

@author: alfred_mac
"""
import numpy as np
import matplotlib.pyplot as plt
import numpy.linalg as la

def DFT(f):
    N = len(f)
    F = (0+0j)*np.arange(N)
    for i in range(N):
        for j in range(N):
            F[i]+= (1/N)*f[j]*np.exp(-2*np.pi*1j*i*j/N)

    return F

def iDFT(F):
    N = len(F)
    f = (0+0j)*np.arange(N)
    for i in range(N):
        for j in range(N):
            f[i]+= F[j]*np.exp(2*np.pi*1j*i*j/N)

    return f
```

Problem 1.a) The following is the code for comparing the accuracy of the SFTW (code mentioned above) module with the inbuilt Python module for

FFT:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Dec 14 03:57:33 2018

@author: alfred_mac
"""

import numpy as np
import matplotlib.pyplot as plt
import numpy.linalg as la
import numpy.fft as fft
import DirectDFT as SFTW

#fi = [0,0,1,0,0,0,0]
#fi = [0,0,0,0,0,1,0]
fi = [0,0,0,0,1,0,0,0,0,0]
fo = SFTW.DFT(fi)

f = open("DFT_output_prob1a.txt","a+")
f.write("Input f is:\n")
np.savetxt(f,fi,fmt='%.2f')

fo_act = (1/N)*np.array(fft.fft(fi))

f.write("Euclidean norm of the difference between the
estimated result and the actual result as a measure of
accuracy is: ")
f.write(str(sum(abs(fo_act-fo))))
f.write("\n")
f.close()
```

The following shows the test results when the above code was tested with different examples. The accuracy of SFTW is mentioned along with it.

Input f is: 0.00 0.00 1.00 0.00 0.00 0.00 0.00 Euclidean norm of the difference between the estimated result and the actual result as a measure of accuracy is: 1.5435090268566047e-15 Input f is: 0.00 0.00 0.00 0.00 0.00 1.00 0.00 Euclidean norm of the difference between the estimated result and the actual result as a measure of accuracy is: 9.066215190086561e-15 Input f is: 0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00 Euclidean norm of the difference between the estimated result and the actual result as a measure of accuracy is: 4.7146948623413195e-15 Input f is: 0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00 Euclidean norm of the difference between the estimated result and the actual result as a measure of accuracy is: 9.000000000000002 Input f is: 0.00

0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00 Euclidean norm of the difference between the estimated result and the actual result as a measure of accuracy is: 4.734507369567292e-16

Problem 1.b) The mathematical derivation for finding the value of C to equal to n is given in the handwritten page. Here, the following code verifies that for a simple test case:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Dec 14 05:34:15 2018

@author: alfred_mac
"""

import numpy as np
import matplotlib.pyplot as plt
import numpy.linalg as la
import numpy.fft as fft
import DirectDFT as SFTW

fi = np.array([0,0,0,0,1,0,0,0,0,0])
fo = SFTW.DFT(fi)

C = sum(fi*np.conj(fi))/sum(fo*np.conj(fo))

print("The value of C for an arrray of 10 elements is: ",C)
```

The following is the output for the above mentioned code:

The value of C for an arrray of 10 elements is: (9.999999999999996+0j)

Problem 1.c) The mathematical derivation for determining the relation between $\hat{g_k}$ and $\hat{f_k}$ is given in the handwritten page. Here, the following code verifies that for a simple test case:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Dec 14 06:05:48 2018

@author: alfred_mac
"""

import numpy as np
import matplotlib.pyplot as plt
import numpy.linalg as la
```

```
import numpy.fft as fft
import DirectDFT as SFTW

N=10
fi = np.random.random([N,1])
gi = np.zeros([N,1])

gi[1:] = fi[0:-1]
gi[0] = fi[-1]

fo = SFTW.DFT(fi)
go = SFTW.DFT(gi)

go_est = fo*0

for i in range(N):
    go_est[i] = fo[i]*np.exp(-2*np.pi*1j*i/N)


print("go is: ",go)
print("go_est is: ",go_est)
diff = sum(abs(go-go_est))
print("Absolute difference between the estimated and
actual values of DFT of g is:",diff)
```

The following is the output for the above mentioned code:

go is: $[0.61932566 + 0.00000000e+00j - 0.03478393 - 1.28427372e-03j - 0.07382498 - 5.50837349e-02j - 0.09483726 - 6.86264145e-02j - 0.10475228 - 3.88661138e-02j 0.07684922 + 7.33402803e-17j - 0.10475228 + 3.88661138e-02j - 0.09483726 + 6.86264145e-02j - 0.07382498 + 5.50837349e-02j - 0.03478393 + 1.28427372e-03j]$

go-est is: $[0.61932566 + 0.00000000e+00j - 0.03478393 - 1.28427372e-03j - 0.07382498 - 5.50837349e-02j - 0.09483726 - 6.86264145e-02j - 0.10475228 - 3.88661138e-02j 0.07684922 + 8.31102647e-17j - 0.10475228 + 3.88661138e-02j - 0.09483726 + 6.86264145e-02j - 0.07382498 + 5.50837349e-02j - 0.03478393 + 1.28427372e-03j]$ Absolute difference between the estimated and actual values of DFT of g is: $1.0796099979596523e-15$

Problem 1.d) The following code verifies both the **DFT** and the **iDFT** function in the **DirectDFT.py** module for a simple test case:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Dec 14 06:49:54 2018

@author: alfred_mac
```

```
"""

import numpy as np
import matplotlib.pyplot as plt
import numpy.linalg as la
import numpy.fft as fft
import DirectDFT as SFTW

N=10
fi = (0+0j)*np.arange(N)
fi += np.random.random()

fo = SFTW.DFT(fi)

fi1 = SFTW.iDFT(fo)

print("Input f is: ",fi)
print("iDFT of DFT of f is: ",fi1)
diff = sum((fi-fi1)*np.conj(fi-fi1))
print("Euclidean norm of difference between f and iDFT of
DFT of f is:",diff)
```

The following is the output for the above mentioned code:

Input f is: [0.48158163+0.j 0.48158163+0.j 0.48158163+0.j 0.48158163+0.j 0.48158163+0.j 0.48158163+0.j 0.48158163+0.j 0.48158163+0.j 0.48158163+0.j 0.48158163+0.j] iDFT of DFT of f is: [0.48158163+7.80644855e-17j 0.48158163+1.94551855e-16j 0.48158163-3.16027451e-16j 0.48158163-6.94075799e-17j 0.48158163+1.65920312e-16j 0.48158163-3.48681348e-16j 0.48158163-2.59278682e-16j 0.48158163+3.44621896e-16j 0.48158163+1.83879486e-16j 0.48158163+2.63570262e-17j] Euclidean norm of difference between f and iDFT of DFT of f is: (1.1345371090849671e-30+0j)

Problem 1.e) The following code verifies how applying DFT four times on a function f gives back the same function again for a simple test case:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Dec 14 07:13:41 2018

@author: alfred_mac
"""


import numpy as np
import matplotlib.pyplot as plt
import numpy.linalg as la
import numpy.fft as fft
```

```
import DirectDFT as SFTW

N=10
f0 = (0+0j)*np.arange(N)
f0 += np.random.random()

f1 = SFTW.DFT(f0)
f2 = SFTW.DFT(f1)
f3 = SFTW.DFT(f2)
f4 = SFTW.DFT(f3)

print("Input f is: ",f0)
print("Applying DFT four times on f gives: ",f4)
diff = sum((f0-100*f4)*np.conj(f0-100*f4))
print("Euclidean norm of difference between f and the
result obtained after applying DFT 4 times is:",diff)
```

The following is the output for the above mentioned code:

Input f is: [0.44742176+0.j 0.44742176+0.j 0.44742176+0.j 0.44742176+0.j 0.44742176+0.j 0.44742176+0.j 0.44742176+0.j 0.44742176+0.j 0.44742176+0.j 0.44742176+0.j] Applying DFT four times on f gives: [0.00447422+2.35772819e-18j 0.00447422+2.54352449e-18j 0.00447422-1.49405621e-18j 0.00447422+2.17642095e-18j 0.00447422-1.24281656e-18j 0.00447422-6.26085601e-18j 0.00447422-1.45134260e-18j 0.00447422+1.69351020e-18j 0.00447422-9.09180979e-19j 0.00447422+2.58706853e-18j] Euclidean norm of difference between f and the result obtained after applying DFT 4 times is: (2.2846566502789448e-30+0j)

Problem 2) The following code computes DFT for the given two functions $g(x)$ and $h(x)$:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sat Dec 15 02:44:07 2018

@author: alfred_mac
"""



import numpy as np
import matplotlib.pyplot as plt
import numpy.linalg as la
import numpy.fft as fft

N = 10000
freqs = fft.fftfreq(N, 1.0/6.0)
freqs = fft.fftshift(freqs)
```

```
g = (1/np.sqrt(2*np.pi))*np.exp(-0.5*freqs*freqs)
h = 0.5*np.exp(-abs(freqs))

gk = fft.fft(g)
hk = fft.fft(h)

gk = fft.fftshift(gk)
hk = fft.fftshift(hk)
k = np.arange(N)-N/2
#plt.plot(freqs,g)
#plt.plot(freqs,h)

n=20
plt.plot(k[5000-n:5000+n+1],np.real(gk[5000-n:5000+n+1]))
plt.plot(k[5000-n:5000+n+1],np.real(hk[5000-n:5000+n+1]),'--')

#plt.title('$g(x)=(2\pi)^{-0.5}e^{-0.5x^{2}}$ and $h(x)=0.5e^{-|x|}$')
plt.title('$G(k)=DFT[g(x)]$ and $H(k)=DFT[h(x)]$')
#plt.legend(['$g(x)$','$h(x)$'],loc='upper right')
plt.legend(['$G(k)$','$H(k)$'],loc='upper right')
#plt.xlabel('$x$')
plt.xlabel('$k$')
#plt.ylabel('$g(x)$ or h(x)')
plt.ylabel('$G(k)$ or H(k)')
plt.show()
```

The following figures show the functions $g(x)$ and $h(x)$ in one plot and its corresponding DFTs - $\hat{g_k}$ and $\hat{h_k}$ in another plot.

From the second plot, its clear that the DFT coefficients of $g(x)$, i.e. $\hat{g_k}$ converge faster.

Problem 3) The following code interpolates both $g(x)$ and $h(x)$ using a trigonometric polynomial DFT for the given two functions $g(x)$ and $h(x)$:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sat Dec 15 05:03:40 2018

@author: alfred_mac
"""

import numpy as np
import matplotlib.pyplot as plt
import numpy.linalg as la
import numpy.fft as fft
```

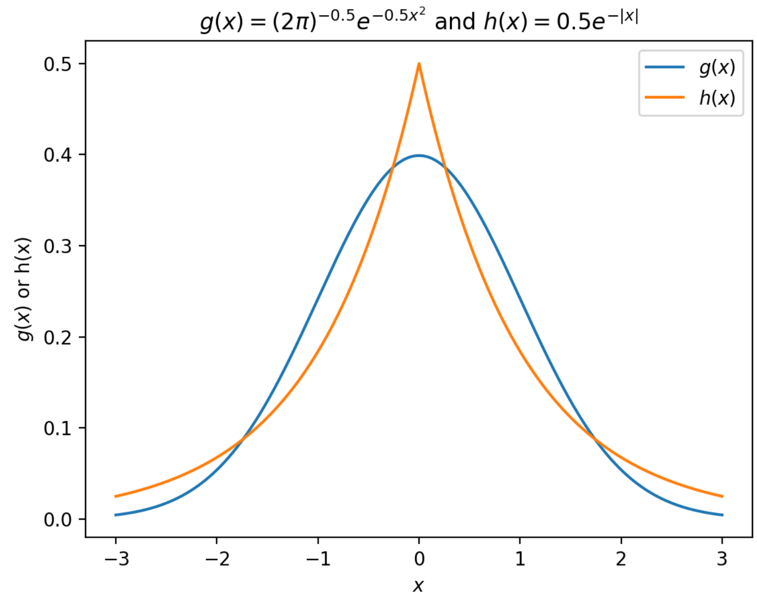

Figure 1: Functions $g(x)$ and $h(x)$

```
L = 6.0

def interpolate(A,freqs,M):
    n = len(freqs)
    p = np.zeros([M,1], dtype='complex')
    x = np.arange(M)*L/M - L/2
    for i in range(M):
        p[i] = 0
#       for j in range(n):
#           p[i] += A[j]*np.exp(2*np.pi*1j*j*x[i]/L)
        for j in range(n):
            if(j<n/2):
                p[i] += A[j]*np.exp(2*np.pi*1j*j*x[i]/L)
            elif(j==n/2):
                p[i] += A[j]*(np.exp(2*np.pi*1j*(n/2)*x[i]/L)+
                np.exp(-2*np.pi*1j*(n/2)*x[i]/L))/2
            else:
                p[i] += A[j]*np.exp(2*np.pi*1j*(j-n)*x[i]/L)


    p = fft.fftshift(p)
```

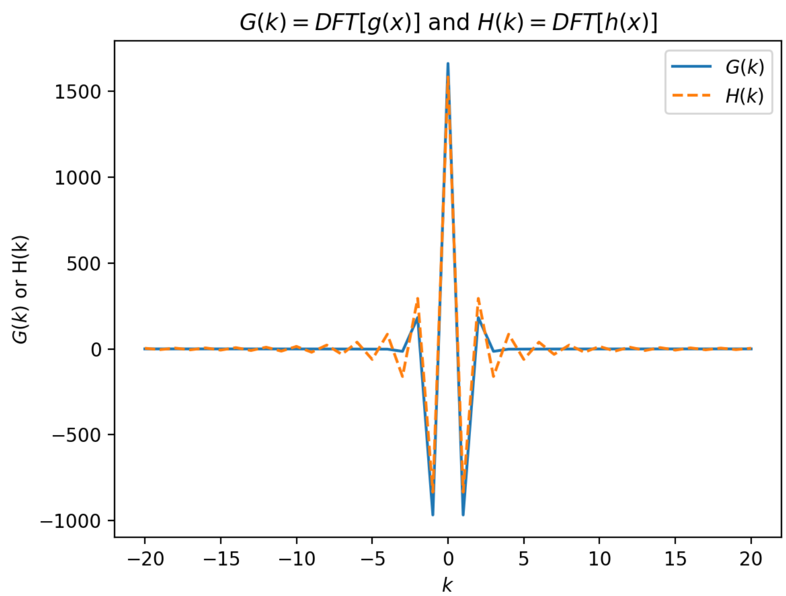

Figure 2: Functions $\hat{g_k}$ and $\hat{h_k}$

```
    return (p,x)

ng = 1000
nh = 1000
M = 1000
freqsg = fft.fftfreq(ng, 1.0/L)
freqsg = fft.fftshift(freqsg)

freqsh = fft.fftfreq(nh, 1.0/L)
freqsh = fft.fftshift(freqsh)

g = (1/np.sqrt(2*np.pi))*np.exp(-0.5*freqsg*freqsg)
h = 0.5*np.exp(-abs(freqsh))

gk = fft.fft(g)/ng
hk = fft.fft(h)/nh

(Pg,Xg) = interpolate(gk,freqsg,M)
(Ph,Xh) = interpolate(hk,freqsh,M)

#plt.plot(freqsg,g)
#plt.plot(Xg,np.real(Pg),'--')
```

```
plt.plot(freqsh,h)
plt.plot(Xh,np.real(Ph),'--')

#plt.title('$g(x)$ and its trigonometric interpolation $p_{g}(x)$')
plt.title('$h(x)$ and its trigonometric interpolation $p_{h}(x)$')
#plt.legend(['$g(x)$','$p_{g}(x)$'])
plt.legend(['$h(x)$','$p_{h}(x)$'])
plt.xlabel('$x$')
#plt.ylabel('$g(x)$ or $p_{g}(x)$')
plt.ylabel('$h(x)$ or $p_{h}(x)$')
plt.show()
```

The following figures show the functions $g(x)$ and $h(x)$ in one plot and its corresponding trigonometric polynomial interpolation for $M = 1000$ and $n = 1000$ for both $g(x)$ and $h(x)$. Such a low value seems to be enough for interpolation for both the functions.

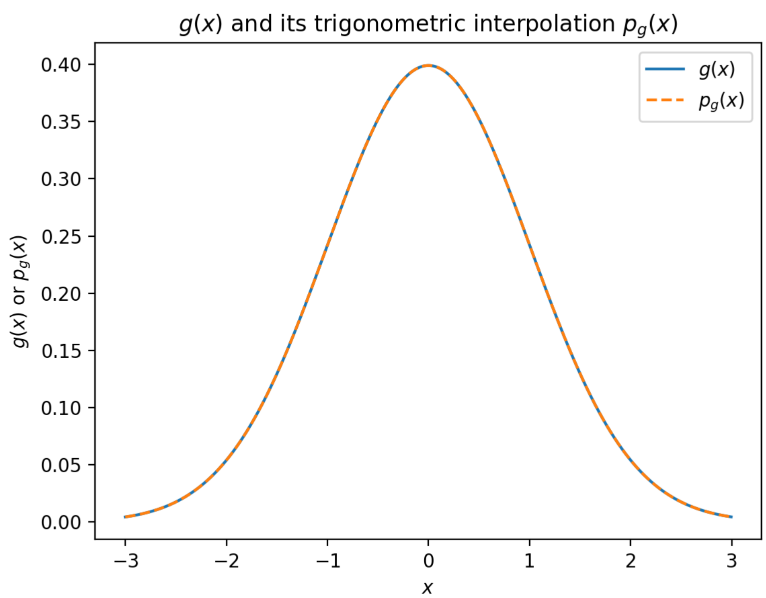

Figure 3: Function $g(x)$ and its trigonometric polynomial interpolation $p_g(x)$

Problem 4.a) The following code verifies the spectral derivative obtained from the interpolated trigonometric polynomial with the actual derivative of the function for the given two functions $g(x)$ and $h(x)$:

```
#!/usr/bin/env python3
```

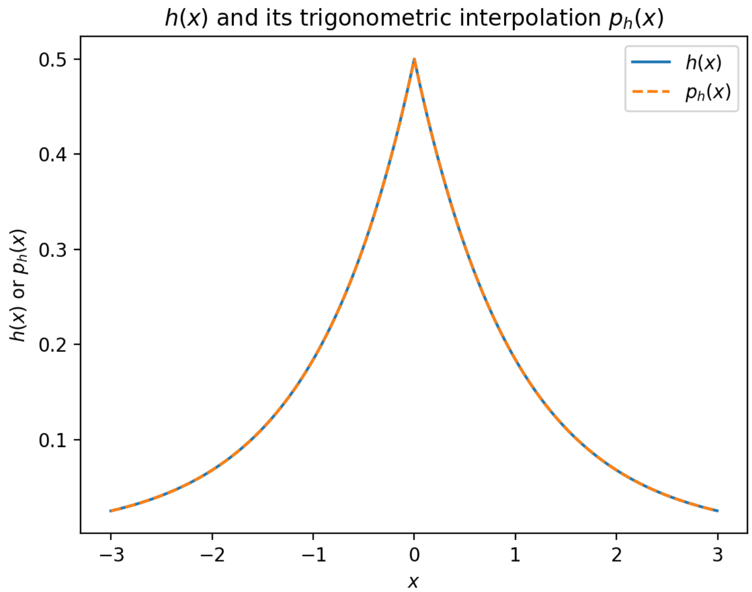

Figure 4: Function $h(x)$ and its trigonometric polynomial interpolation $p_h(x)$

```
# -*- coding: utf-8 -*-
"""
Created on Sat Dec 15 07:10:26 2018

@author: alfred_mac
"""

import numpy as np
import matplotlib.pyplot as plt
import numpy.linalg as la
import numpy.fft as fft

L = 6.0

def interpolate(A,freqs,M):
    n = len(freqs)
    p = (0+0j)*np.arange(M) #np.zeros([M,1], dtype='complex')
    x = np.arange(M)*L/M - L/2
    for i in range(M):
        p[i] = 0
#         for j in range(n):
#             p[i] += (2*np.pi*1j*(j-n/2)/L)*A[j]*np.exp(2*np.pi*1j*j*x[i]/L)
```

```
        for j in range(n):
            if(j<n/2):
                p[i] += (2*np.pi*1j*j/L)*A[j]*
                np.exp(2*np.pi*1j*j*x[i]/L)
            elif(j==n/2):
                p[i] += 0*A[j]*(np.exp(2*np.pi*1j*(n/2)*x[i]/L)+
                np.exp(-2*np.pi*1j*(n/2)*x[i]/L))/2
            else:
                p[i] += (2*np.pi*1j*(j-n)/L)*A[j]*
                np.exp(2*np.pi*1j*(j-n)*x[i]/L)

    p = fft.fftshift(p)
    return (p,x)

ng = 10000
nh = 10000
M = 1000
freqsg = fft.fftfreq(ng, 1.0/L)
freqsg = fft.fftshift(freqsg)

freqsh = fft.fftfreq(nh, 1.0/L)
freqsh = fft.fftshift(freqsh)

g = (1/np.sqrt(2*np.pi))*np.exp(-0.5*freqsg*freqsg)
h = 0.5*np.exp(-abs(freqsh))

gk = fft.fft(g)/ng
hk = fft.fft(h)/nh

dgdx = -(freqsg/np.sqrt(2*np.pi))*np.exp(-0.5*freqsg*freqsg)
dhdx = -(freqsh/abs(freqsh))*0.5*np.exp(-abs(freqsh))

#gk = fft.fft(dgdx)/ng
#hk = fft.fft(dhdx)/nh

#(Pg,Xg) = interpolate(gk,freqsg,M)
(Ph,Xh) = interpolate(hk,freqsh,M)



#plt.plot(freqsg,dgdx)
#plt.plot(Xg,np.real(Pg),'--')

#plt.plot(freqsh,dhdx)
#plt.plot(Xh,np.real(Ph),'--')
```

```
# Plotting error between the sampled derivative (f(xk)) and
spectral derivative (p(xk))
#plt.plot(Xg[10:990],dgdx[100:(ng-100):int(ng/M)]-
np.real(Pg[10:990]),'--') # Values at the edges are not
shown for visual reasons
plt.plot(Xh,dhdx[0:nh:int(nh/M)]-np.real(Ph),'--') # Values
at the edges are not shown for visual reasons

#plt.title('$g\'(x)$ and its trigonometric interpolation $p_{g}\'(x)$')
#plt.title('$h\'(x)$ and its trigonometric interpolation $p_{h}\'(x)$')
#plt.title('Error between $g\'(x)$ and its spectral derivative $p\'_{g}(x)$')
plt.title('Error between $h\'(x)$ and its spectral derivative $p\'_{h}(x)$')
#plt.legend(['$g\'(x)$','$p_{g}\'(x)$'])
#plt.legend(['$h\'(x)$','$p_{h\'}(x)$'])
plt.xlabel('$x$')
#plt.ylabel('$g\'(x)$ or $p_{g}\'(x)$')
#plt.ylabel('$h\'(x)$ or $p_{h\'}(x)$')
#plt.ylabel('$g\'(x) - p_{g\'}(x)$')
plt.ylabel('$h\'(x) - p_{h\'}(x)$')
plt.show()
```

The following first two figures show the spectral derivative obtained from the interpolated trigonometric polynomial with the actual derivative of the given two functions $g(x)$ and $h(x)$. Also, the next two figures show the error between the actual function and its interpolation for both the given functions:

Here we notice that, unlike for the function $g'(x)$, the error between $h'(x)$ and it's interpolation is too high even in the center region.

Problem 4.b) The following code computes the sequence of accuracy measures for different values of Δx for both the functions $g'(x)$ and $h'(x)$:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Dec 19 09:02:04 2018

@author: alfred_mac
"""


import numpy as np
import matplotlib.pyplot as plt
import numpy.linalg as la
import numpy.fft as fft

L = 6.0

def interpolate(A,freqs,M):
```

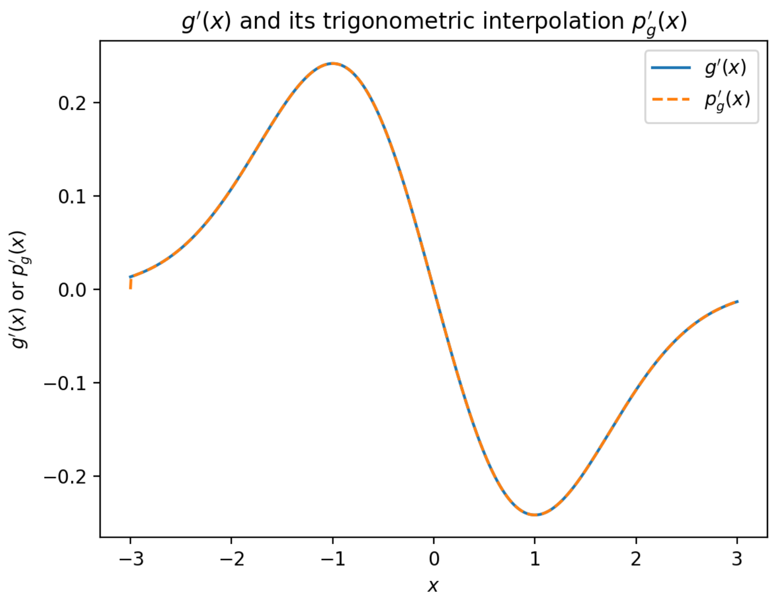

Figure 5: Function $g'(x)$ and its trigonometric polynomial interpolation $p_{g'}(x)$

```
    n = len(freqs)
    p = (0+0j)*np.arange(M) #np.zeros([M,1], dtype='complex')
    x = np.arange(M)*L/M - L/2
    for i in range(M):
        p[i] = 0
        for j in range(n):
            if(j<n/2):
                p[i] += (2*np.pi*1j*j/L)*A[j]*
                np.exp(2*np.pi*1j*j*x[i]/L)
            elif(j==n/2):
                p[i] += 0*A[j]*(np.exp(2*np.pi*1j*(n/2)*x[i]/L)+
                np.exp(-2*np.pi*1j*(n/2)*x[i]/L))/2
            else:
                p[i] += (2*np.pi*1j*(j-n)/L)*A[j]*
                np.exp(2*np.pi*1j*(j-n)*x[i]/L)

    p = fft.fftshift(p)
    return (p,x)

ng = 10000
nh = 10000
freqsg = fft.fftfreq(ng, 1.0/L)
```

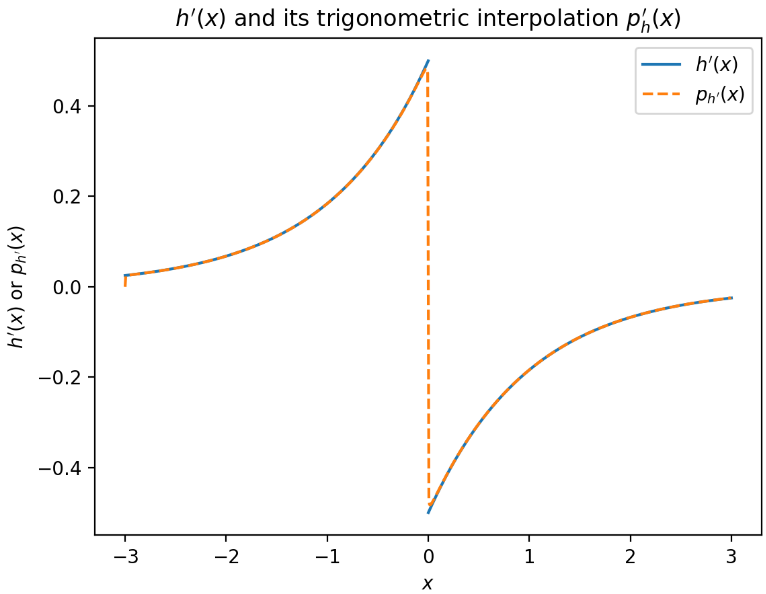

Figure 6: Function $h'(x)$ and its trigonometric polynomial interpolation $p_{h'}(x)$

```
freqsg = fft.fftshift(freqsg)

freqsh = fft.fftfreq(nh, 1.0/L)
freqsh = fft.fftshift(freqsh)

g = (1/np.sqrt(2*np.pi))*np.exp(-0.5*freqsg*freqsg)
h = 0.5*np.exp(-abs(freqsh))

gk = fft.fft(g)/ng
hk = fft.fft(h)/nh

dgdx = -(freqsg/np.sqrt(2*np.pi))*np.exp(-0.5*freqsg*freqsg)
dhdx = -(freqsh/abs(freqsh))*0.5*np.exp(-abs(freqsh))

M = np.array([1,2,5,10,20,25,40,50,80,100,125,200,250,400,
500,625,1000,2000,5000,10000])
D = np.zeros([len(M),1])

for i in range(len(M)):
    print("Doing for: ",M[i])
#    (Pg,Xg) = interpolate(gk,freqsg,M[i])
    (Ph,Xh) = interpolate(hk,freqsh,M[i])
```

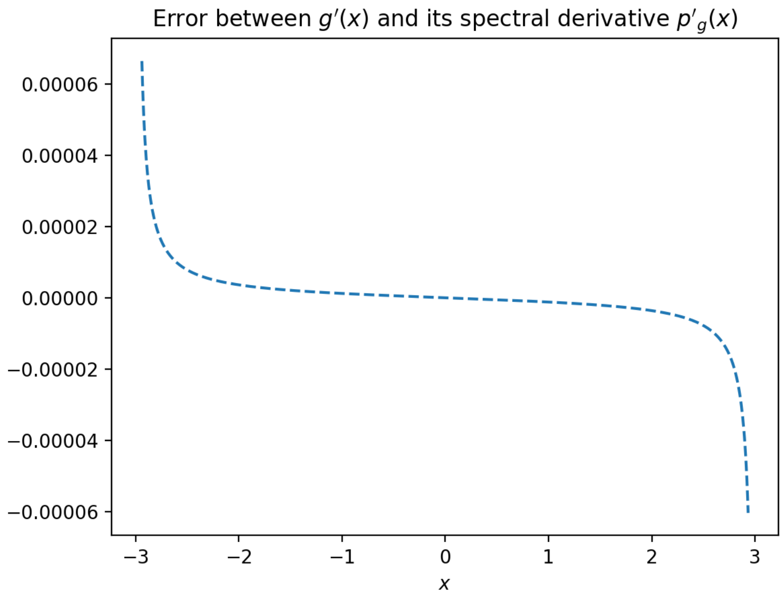

Figure 7: Error between $g'(x)$ and its trigonometric polynomial interpolation $p_{g'}(x)$

```
#     D[i] = max(abs(dgdx[0:ng:int(ng/M[i])]-np.real(Pg)))
    D[i] = max(abs(dhdx[0:nh:int(nh/M[i])]-np.real(Ph)))

plt.semilogx(L/M,D)

#plt.title('Accuracy (or error) vs $\Delta x$ for $g(x)$')
plt.title('Accuracy (or error) vs $\Delta x$ for $h(x)$')
plt.xlabel('$\Delta x$')
plt.ylabel('Accuracy (or error)')
plt.show()
```

The following figures show the variation of the accuracy measure for different values of Δx for both the functions $g'(x)$ and $h'(x)$:

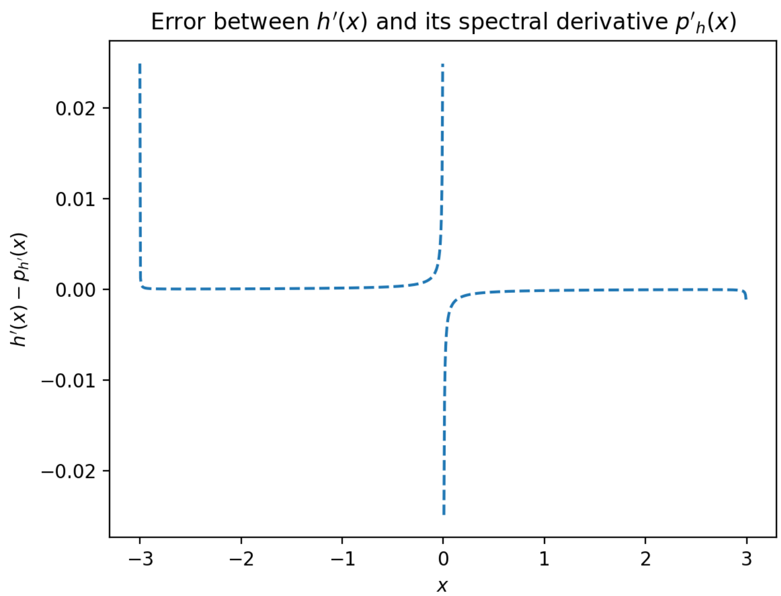

Figure 8: Error between $h'(x)$ and its trigonometric polynomial interpolation $p_{h'}(x)$

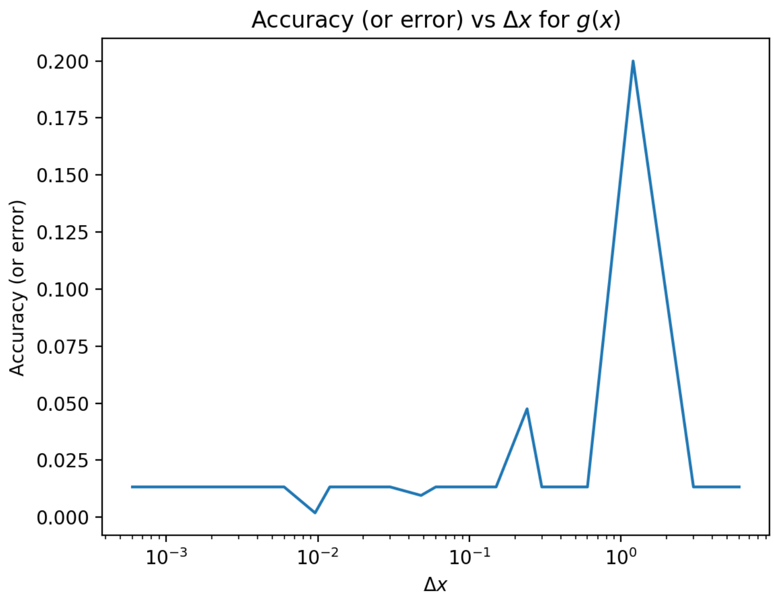

Figure 9: Accuracy vs Δx for the function $g'(x)$ and $p'_g(x)$

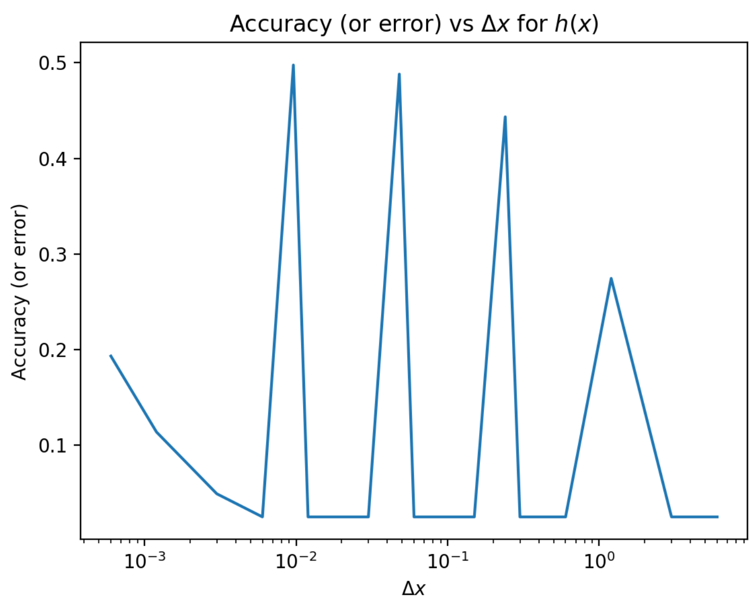

Figure 10: Accuracy vs Δx for the function $h'(x)$ and $p'_h(x)$