Math 478 HW 7

Lucas Bouck

11/29/16

1 Problem 3.7.14

First, I'll import important modules and libraries for the exercises.

```
In [1]: import numpy as np
        from scipy.fftpack import fft, ifft, dst, idst, dct, idct, fftfreq
        import matplotlib.pyplot as plt
        %matplotlib inline
```

Here, we must use spectral differentiation to approximate the second derivative of $|\sin(2\pi t)|^5$. The function is 1 periodic, so fourier transforms on the function on the interval $[0, 1]$ will do.

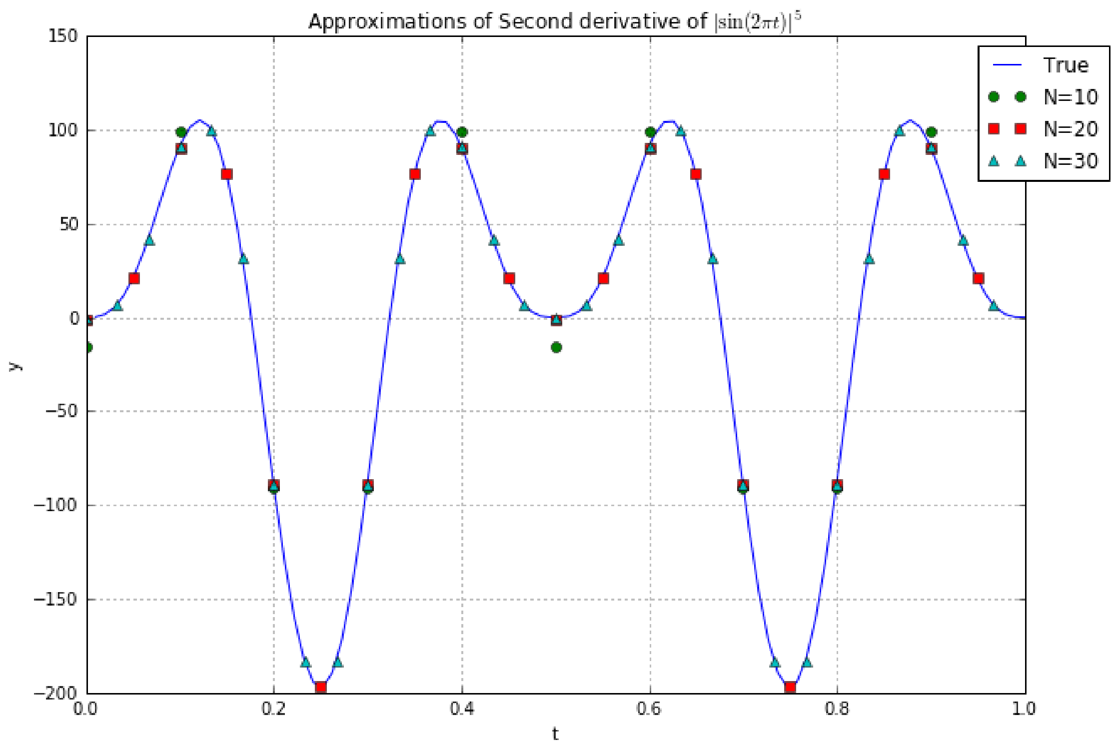
```
In [2]: def periodic_spectraldiff(f,N):
            '''In this function we approximate the second derivative
            of a function f. We update the answer vector and return
            a tuple of the tvals and the approximation.'''
            #use t values 0,1/n,...,(n-1)/n
            tvals=np.arange(0,N)/N
            #calculate our interpolation points
            answer=f(tvals)
            #calc the fft
            answer=fft(answer)
            #create the d vector using the fftfreq function
            d=fftfreq(N)*N*2*np.pi*1j
            d[int(N/2)]=0
            #compute the fft of the second derivative
            #note that python component wise mult is just *
            answer=(d**2)*answer
            #return the approxiamtion
            return (tvals, ifft(answer))
```

In the next cell, we compute and plot the second derivative approximations to $f(t) = |\sin(2\pi t)|^5$. The second derivative of f is

$$f''(t) = \frac{5\pi^2 \sin^2(2\pi t)\left(3\sin^2(4\pi t) + 4\sin^2(2\pi t)\cos(4\pi t)\right)}{|\sin(2\pi t)|}$$

```
In [3]: #create the f we are trying to approximate
        def f(t):
```

```python
    return np.abs(np.sin(2*np.pi*t))**5
#the true second derivative of f
def fpp(t):
    numerator1=5*np.pi**2*np.sin(2*np.pi*t)**2
    numerator2=(3*np.sin(4*np.pi*t)**2\
                +4*np.cos(4*np.pi*t)*np.sin(2*np.pi*t)**2)
    denominator=np.abs(np.sin(2*np.pi*t))
    return numerator1*numerator2/denominator

#create the mesh over which we will
#compare the true solution to the approximation
#starting this t value past 0 so we don't divide by zero
t=np.linspace(0,1,100)[1:]

#computing the approximations
periodic10=periodic_spectraldiff(f,10)
periodic20=periodic_spectraldiff(f,20)
periodic30=periodic_spectraldiff(f,30)

#plotting the approximations compared to the true f''
plt.figure(figsize=(10,7))
plt.plot(t,fpp(t),label='True')
plt.plot(periodic10[0],np.real(periodic10[1]),
         'o',label='N=10')
plt.plot(periodic20[0],np.real(periodic20[1]),
         's',label='N=20')
plt.plot(periodic30[0],np.real(periodic30[1]),
         '^',label='N=30')
plt.title('Approximations of Second derivative \
of $|\sin(2\pi t)|^5$')
plt.xlabel('t')
plt.ylabel('y')
plt.legend(bbox_to_anchor=(1.1, 1))
plt.grid()
plt.show()
```



2 Problem 3.7.15

In this problem, we are asked to approximate a second derivative using cosine transforms. In the cell below, the function computes the second derivative approximation of an arbitrary function f on the interval $[0, 1]$ with N interpolation points. Please note that, quoted from the scipy.fftpack documentation, they say "For a single dimension array x, dct(x, norm='ortho') is equal to MATLAB dct(x)."

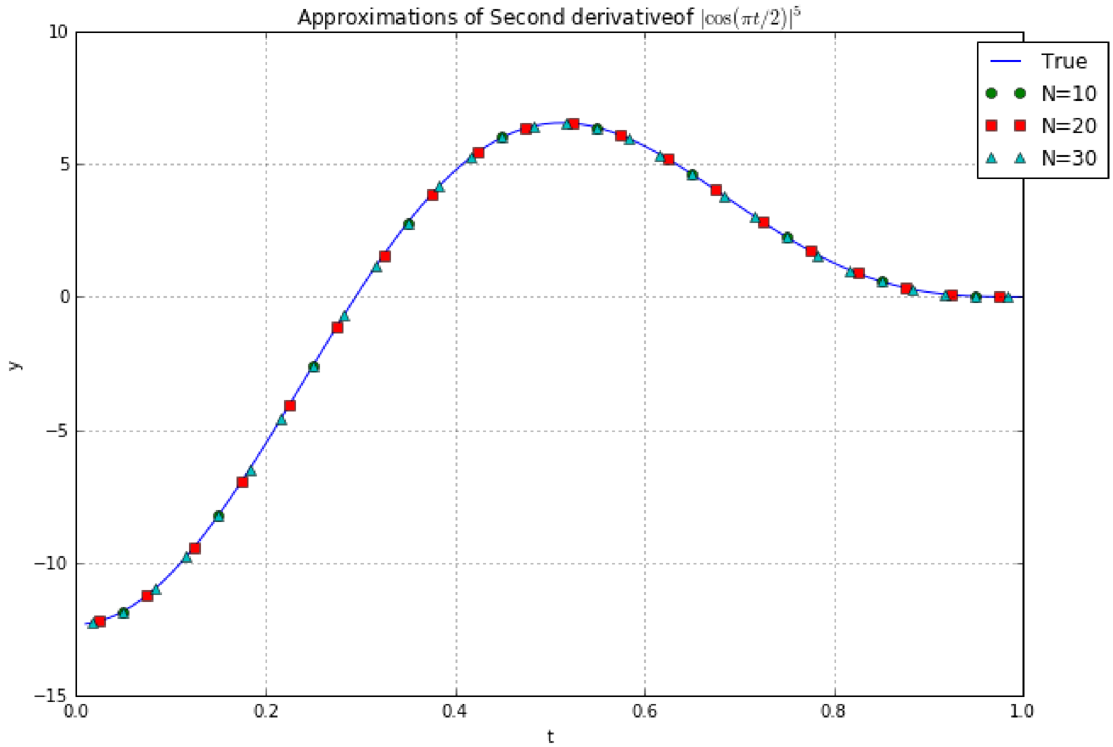
```
In [4]: def neumann_spectraldiff(f,N):
            '''In this function we approximate the second derivative
            of a function f. We update the answer vector and return
            the approximation.'''
            #use t values 0,1/n,...,(n-1)/n
            tvals=(2*np.arange(0,N)+1)/(2*N)
            #calculate our interpolation points
            answer=f(tvals)
            #calc the dct
            answer=dct(answer,norm='ortho')
            #create the d vector
            d=-(np.pi*np.arange(0,N))**2
            #compute the dct of the second derivative
            #note that python component wise mult is just *
            answer=d*answer
```

```
        #return the approxiamtion
        return (tvals,idct(answer,norm='ortho'))
```

In the next cell, we compute and plot the second derivative approximations to $|\cos(\frac{\pi t}{2})|^5$. The second derivative of f is

$$f''(t) = -\frac{5\pi^2\cos^2\left(\frac{\pi x}{2}\right)\left(4\cos^2\left(\frac{\pi x}{2}\right)\cos\left(\pi x\right) - 3\sin^2\left(\pi x\right)\right)}{16\left|\cos\left(\frac{\pi x}{2}\right)\right|}$$

```
In [5]: #create the f we are trying to approximate
        def f(t):
            return np.abs(np.cos(np.pi*t/2))**5
        def fpp(t):
            num1=5*np.pi**2*np.cos(np.pi*t/2)**2
            num2=(4*np.cos(np.pi*t/2)**2\
                  *np.cos(np.pi*t)-3*np.sin(np.pi*t)**2)
            denom=16*np.abs(np.cos(np.pi*t/2))
            return -num1*num2/denom

        #create the mesh over which we will
        #compare the true solution to the approximation
        t=np.linspace(0,1,100)[1:]

        #computing the approximations
        neumann10=neumann_spectraldiff(f,10)
        neumann20=neumann_spectraldiff(f,20)
        neumann30=neumann_spectraldiff(f,30)

        #plotting the approximations compared to the true f''
        plt.figure(figsize=(10,7))
        plt.plot(t,fpp(t),label='True')
        plt.plot(neumann10[0],np.real(neumann10[1]),
                 'o',label='N=10')
        plt.plot(neumann20[0],np.real(neumann20[1]),
                 's',label='N=20')
        plt.plot(neumann30[0],np.real(neumann30[1]),
                 '^',label='N=30')
        plt.title('Approximations of Second derivative\
        of $|\cos(\pi t/2)|^5$')
        plt.xlabel('t')
        plt.ylabel('y')
        plt.legend(bbox_to_anchor=(1.1, 1))
        plt.grid()
        plt.show()
```



3 Problem 3.7.16

Here, we must use spectral differentiation to approximate the second derivative of $|\sin(2\pi t)|^5$. We'll use dst to approximate the second derivative.

```
In [6]: def dirichlet_spectraldiff(f,N):
            '''In this function we approximate the second derivative
            of a function f. We update the answer vector and return
            the approximation.'''
            #use t values 0,1/n,...,(n-1)/n
            tvals=np.arange(1,N+1)/(N+1)
            #calculate our interpolation points
            answer=f(tvals)
            #calc the dst. please see the scipy documentation
            #to see that this is what Matlab would implement
            answer=dst(answer,type=1)/2
            #create the d vector
            d=-(np.pi*np.arange(1,N+1))**2
            #compute the dst of the second derivative
            #note that python component wise mult is just *
            answer=d*answer
            #return the approxiamtion
```

```
    #please note that the scaling by 1/(N+1) is
    #the idst based on the scipy.fftpack documentation
    #and what was implemented for dst
    return (tvals,idst(answer,type=1)/(N+1))
```

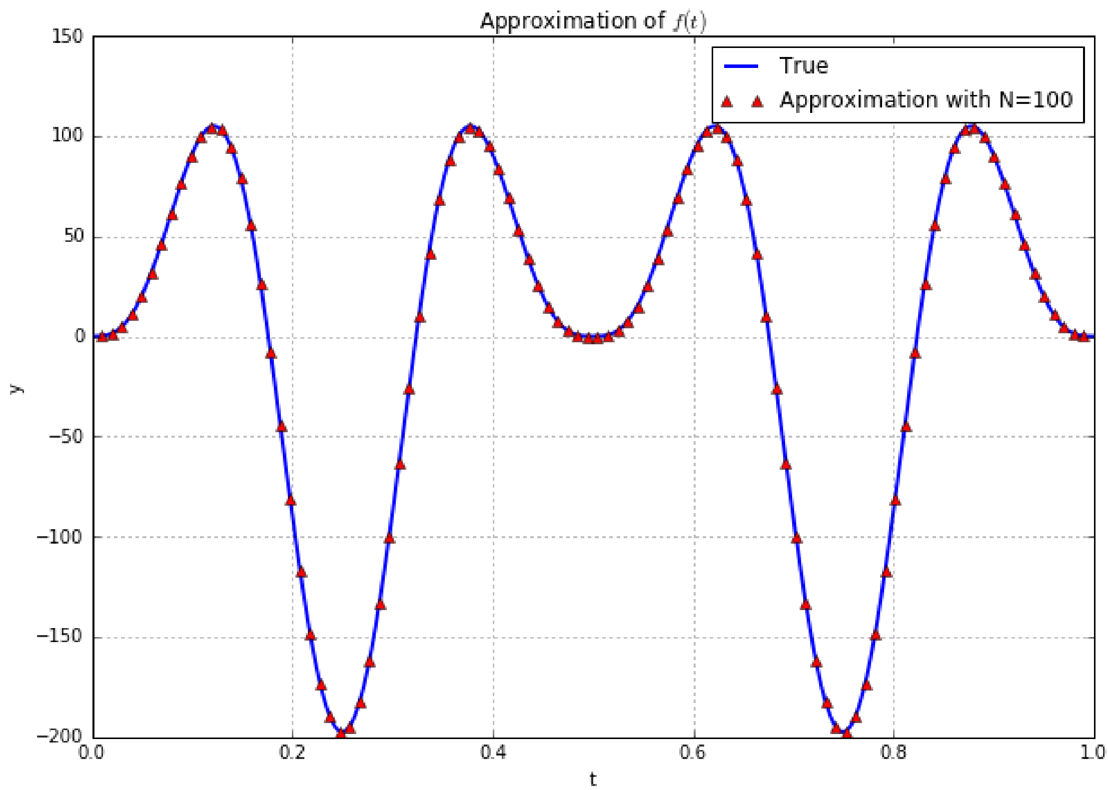
In the next cell, we compute and plot the second derivative approximations to $f(t) = |\sin(2\pi t)|^5$. The second derivative of f is

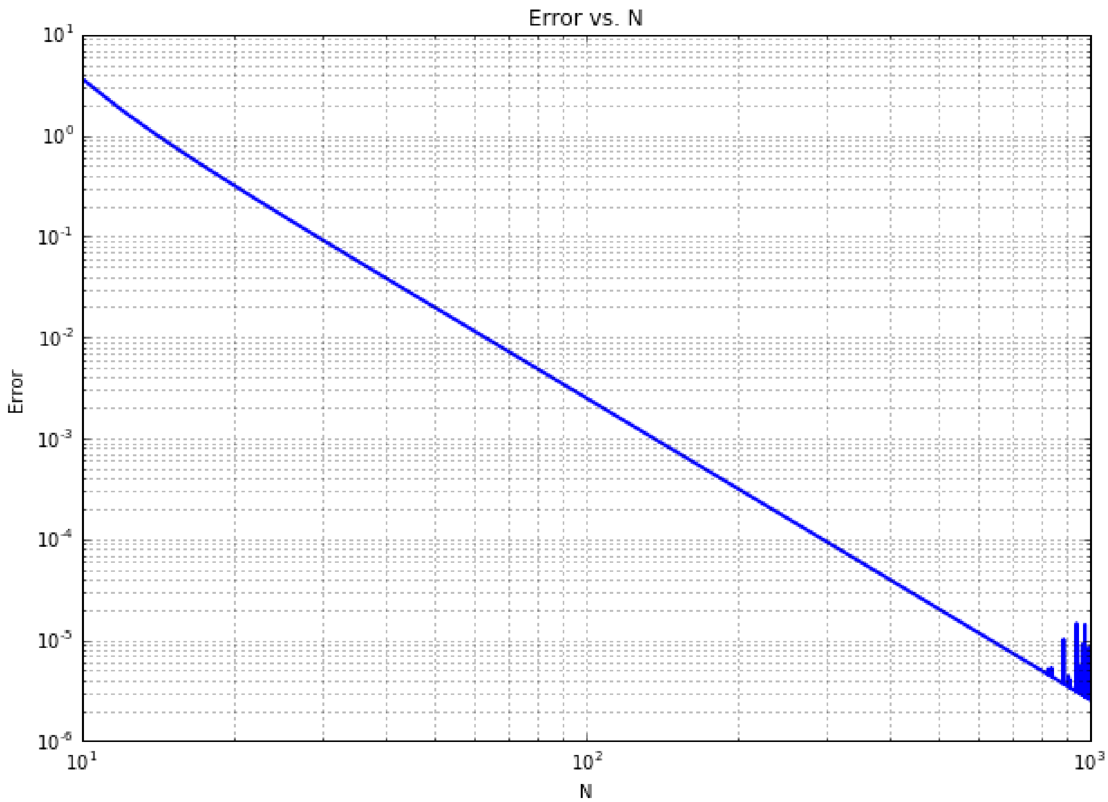
$$f''(t) = \frac{5\pi^2 \sin^2(2\pi t)\left(3\sin^2(4\pi t) + 4\sin^2(2\pi t)\cos(4\pi t)\right)}{|\sin(2\pi t)|}$$

We also loop through every even N starting at $N = 10$ all the way to $N = 1000$. We then find the maximum error between the true second derivative and our approximation. At $N = 100$, we plot the approximation over a graph of the true second derivative. We also create a log-log plot of the error as we increase N.

```
In [7]: #our f(t)
        def f(t):
            return np.abs(np.sin(2*np.pi*t))**5
        #the true second derivative
        def fpp(t):
            num1=5*np.pi**2*np.sin(2*np.pi*t)**2
            num2=(3*np.sin(4*np.pi*t)**2\
                +4*np.cos(4*np.pi*t)*np.sin(2*np.pi*t)**2)
            denom=np.abs(np.sin(2*np.pi*t))
            return num1*num2/denom

        #initialize the n and error arrays
        n_array=np.zeros(496)
        error_array=np.zeros(496)

        #this loop will calculate the error and make the two arrays
        #we need for the error loglog plot
        for n in range(0,496):
            #we calc our number of points
            N=n*2+10

            #if N=100, we make a plot of the approx
            # over the true deriv
            if N==100:
                t=np.linspace(0,1,1000)[1:]
                approx=dirichlet_spectraldiff(f,N)
                tvals=approx[0]
                deriv2=fpp(t)
                plt.figure(figsize=(10,7))
                plt.plot(t,deriv2,linewidth=2,label='True')
                plt.plot(tvals,approx[1],'r^',
                         label='Approximation with N=100')
                plt.xlabel('t')
```

```
        plt.ylabel('y')
        plt.title('Approximation of $f''(t)$')
        plt.grid()
        plt.legend()
        plt.show()

    #for every N, we make note of the max error
    n_array[n]=N
    approx=dirichlet_spectraldiff(f,N)
    tvals=approx[0]
    true=fpp(tvals)
    error_array[n]=np.max(np.abs(true-approx[1]))

#plotting log-log plot of the error vs N
plt.figure(figsize=(10,7))
plt.loglog(n_array,error_array,linewidth=2)
plt.xlabel('N')
plt.ylabel('Error')
plt.title('Error vs. N')
plt.grid(b=True, which='major')
plt.grid(b=True, which='minor')
plt.show()
```





Based on the slope of the graph, this method is an order 4 method.