Problem Class 3: Spectral/Fourier methods and combining with time evolution

David Dickinson d.dickinson@york.ac.uk

Semester 1

1 Fourier transforms in Python

In lecture 7 we looked at the use of spectral methods as an alternative to finite differences approximations of derivatives on a grid. The essential idea behind spectral methods is to represent our function, f, in terms of known *basis functions*:

$$f(x) = \sum_i \hat{f}_i \alpha_i(x)$$

Here the function f is represented in terms of the basis set $\{\alpha\}$ and the coefficients $\left\{\hat{f}\right\}$. which allows us to write

$$\frac{df}{dx} = \sum_i \hat{f}_i \frac{d\alpha_i}{dx}$$

First find f_hat, the weightings, to then find f

As $\{\alpha\}$ are known functions, their derivatives should also be known. This means that once we know $\left\{\hat{f}\right\}$ we can evaluate the derivatives of f by simply calculating the above sum.

The appropriate choice of basis function can be quite important. In this worksheet we will only consider periodic domains, in which the most suitable basis set is almost always the set of cos and sin waves. Given a periodic box of lenght L we will represent our basis functions as:

$$\alpha_j(x) = \exp\left(\frac{i2\pi jx}{L}\right) = \exp(ik_j x)$$

This means our spectral representation of f is

$$f(x) = \sum_j \hat{f}_j \exp(ik_j x)$$

which is the discrete Fourier transform. In other words, if we Fourier transform f we will find $\left\{\hat{f}\right\}$.

There are routines within scipy for performing the fast Fourier transform (FFT) of data. Before we look at how to use this to solve equations we'll get familiar with how to use these FFT routines in some simple problems.

1.1 Using SciPy to FFT

Doesn't solve full problem, just does first part (only fft, not inverse)

The FFT routines within scipy are in the fft sub-module. To find out what this sub-module provides you can do

```
from scipy import fft
help(fft)
```

This will tell you that there actually quite a few different FFT routines that are available. I also recommend that you look at the online documentation here as this discusses some of the important conventions involved (such as the normalisations used etc).

In this worksheet we'll just be using fft and ifft (which is the inverse FFT). For this example we'll consider a periodic box of length $L = 1.0$, which we discretise with $n_x = 100$ grid points. We'll say our function is

$$f(x) = 0.5\sin(2\pi x10) + \sin(2\pi x25)$$

wavelength = L/n

We can see this is the sum of two simple sin waves, one with wavenumber $k_x = 2\pi 10$ (wavelength $\lambda = 1.0/10.0$) and the other with wavenumber $k_x = 2\pi 25$ (wavelength $\lambda = 1.0/25.0$). The example below (based on the

documentation mentioned above) shows how we can FFT this simple function and plot it's components. The output of this example is shown in fig. 1.

```
from scipy.fft import fft, fftfreq
from numpy import linspace, sin, pi

#Define the length of our box
length = 1.0
nx = 100
x = linspace(0,length,nx,endpoint=False)
dx = x[1]-x[0]

#Make our function, sum two sin curves; f = sin(x k_25)+sin(x*k_10)/2
#where k_i is the ith wave-number = 2*pi/lambda_i
f = sin(x*2*pi*25)+sin(x*2*pi*10)*0.5

#FFT; Converts from x to k_x
fHat = fft(f)      returns frequency domain form of input (spectral representation) for each value input of x from array

#Now calculate the x values of the transformed problem, i.e. the inverse
#wavelengths lambda.
#Note we, only use the first half of the solution because of the fft
#symmetry (+ve and -ve wavelengths) -- See the documentation
lamb = fftfreq(nx,dx)      returns frequencies (1/wavelength??) associated with the values used so we can plot them together.
upLim = nx//2
#Note fftfreq is doing something like:
#lamb = linspace(1.0,1.0/(2*dx),nx/2,endpoint=False)

#Plot spectrum
import matplotlib.pyplot as plt
plt.plot(lamb[:upLim],(2.0/nx)*abs(fHat[:upLim]),'-x')
plt.xlabel(r'$1/\lambda$') ; plt.ylabel(r'$\hat{f}$')
plt.grid() ; plt.show()
```

There are several important things to note in this example code:

1. Because the box is periodic we don't want to include the last point in x because we know that this is the same as the first point. To avoid this we add the endpoint = False argument to linspace, which basically means don't include the last point.
2. There are two important points to consider when we've transformed to the Fourier space and we calculate λ. Firstly the maximum wavelength in the system is determined by the x-grid spacing. Secondly the number of wavelengths we use is half the number of x-grid points. This is because the FFT will tell us about both positive and negative wavelengths, which are the same here.
3. When we plot the coefficients there are also a few points to note. Once again we only use half the result so we only look at the positive wavelengths. We take the magnitude of the coefficients using abs as the coefficients are actually complex, whilst we only care about the magnitude. Finally, we must normalise the coefficients to get the correct values (0.5 and 1). The normalisation we're using is $2/n_x$, the factor two accounts for the negative wavelengths (e.g. half the amplitude for one wavelength goes to $\lambda = 1.0/10.0$ and the other half goes to $\lambda = -1.0/10.0$). The $1/n_x$ part of the normalisation is simply part of the FFT routine's definition.

Exercise:

1. Implement the example above and check you can reproduce the figure.
2. Modify this code to also plot the negative wavelengths. Hint: You will want to check the documentation for fft, maybe try printing the value of lamb. Advanced hint: Try looking at the documentation for scipy.fftpack.fftshift and scipy.fftpack.fftfreq.
3. Use the ifft function to convert back to the original solution. Is there any difference between the original f and back transformed function?
4. Try to filter out the $\lambda = 1/10.0$ component and compare the resulting function with the original. Hint: The coefficients, $\hat{f}$ control how much of each wavelength make up the final function (don't forget the negative wavelengths).

An example solution is given in section 4.1

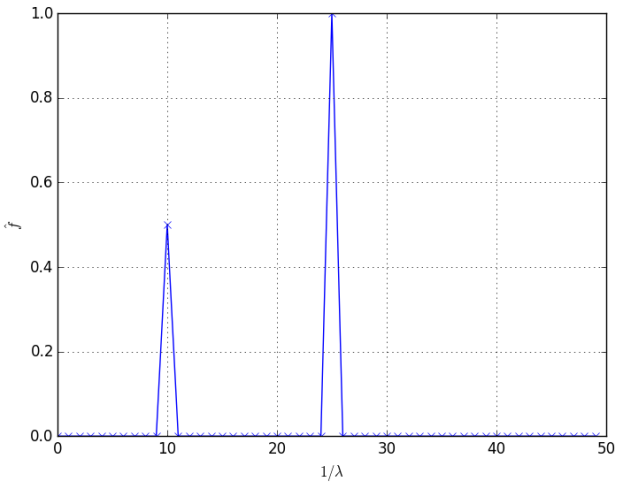

Figure 1: Output of the simple FFT example showing single peaks at 10 and 25.

2 Poisson's equation

In this task you're going to look at solving Poisson's equation, given by eq. (1), in a periodic 1D box.

$$-\frac{d^2\phi}{dx^2} = \rho(x) \tag{1}$$

Exercise:

1. Replace ϕ and ρ with their Fourier transform in eq. (1) and derive a simple relation between $\hat{\phi}$ and $\hat{\rho}$. Hint: We did this in the lecture! Lecture 7
2. Write a python program that calculates $\hat{\phi}$ given a specified ρ. Use the inverse fourier transform to calculate ϕ from $\hat{\phi}$ and plot this. plot phi against x?
3. Use the program you just wrote with $\rho = \sin(2\pi x/L)$ in a periodic box of length 1.0. Compare this to the analytic solution, does it agree? Hint: You'll need to make sure you've taken care of the FFT normalisations properly.
4. *Optional*: Return to the Poisson boundary value problem from the last problem class. Modify this to use periodic boundary conditions (you'll want to make sure ρ is periodic as well). Check that you get the same solution as using the Fourier approach. Compare the error as a function of number of grid points between these two approaches. Note, there's an example of using periodic boundary conditions using the matrix approach given in section 4.2.1.

An example solution is given in section 4.2

3 Including time evolution

Suppose our charge density $\rho(x)$ isn't fixed in time, i.e. $\rho = \rho(x,t)$, this means our ϕ values will change in time as well.

Exercise:

1. Convert the program you wrote in the previous task to solve Poisson's equation into a function which given x and ρ returns ϕ.
2. Let us suppose that the time evolution of ρ is determined by

$$\frac{\partial^2\rho(x,t)}{\partial t^2} = -\phi$$

Need to write as 1st order ode

Solve for phi at each step?

3. Write a program which uses the function you wrote in the task above and solve_ivp to evolve ρ from $t = 0.0$ to $t = 200.0$ in 100 steps using the initial conditions $\rho(t=0) = \sin(2\pi x/L)$ and $\partial\rho/\partial t = 0$.

Dirichlet lower boundary, and use Neumann for upper boundary

4. Try changing the initial conditions to be a sum of waves with different wavelengths. Fourier transform the resulting ρ in x at each time. Hint: Look at the documentation for fft (in particular the axis argument). Plot $\hat{\rho}(t)$ for each wavelength that you included in your initial conditions. Can you use this to understand the contour plot of $\rho(t)$?

An example solution is given in section 4.3

4 Example solutions

4.1 Fourier transforms

The below shows how to use the FFT routines from scipy. Example output is shown in fig. 2

```
from scipy.fft import fft, fftfreq, fftshift, ifftshift, ifft
import matplotlib.pyplot as plt
from numpy import linspace, sin, pi, sqrt, mean

#Define the length of our box
length = 1.0
nx = 1000
x = linspace(0,length,nx,endpoint=False)
dx = x[1]-x[0]

#Make our function, sum two sin curves; f = sin(x k_25)+sin(x*k_10)/2
#where k_i is the ith wave-number = 2*pi/lambda_i
f = sin(x*2*pi*25)+sin(x*2*pi*10)*0.5

#FFT; Converts from x to k_x
fHat = fft(f)
#Calculate the inverse wavelengths
lamb = fftfreq(nx,dx)
#Shift to put -ve values first then positive
fHat = fftshift(fHat)
lamb = fftshift(lamb)
#Plot full spectrum
#plt.plot(lamb,(1.0/nx)*abs(fHat),'-x')
#plt.xlabel(r'$1/\lambda$') ; plt.ylabel(r'$\hat{f}$')
#plt.grid() ; plt.show()

#/Now shift order back
fHat = ifftshift(fHat)
lamb = ifftshift(lamb)

#Let's calculate the rms error on the back transform
fBackTrans = ifft(fHat) ; err = f-fBackTrans
rmsErr = sqrt(mean(err*err.conjugate()))
print("The rms error is {err}".format(err=rmsErr))

#Now let's filter out the lambda = 1.0/10.0 wave
#Find the index where lamb is closest to 10.0
indPos = abs(10.0-lamb).argmin()
#Find the index where lamb is closest to -10.0
indNeg = abs(10.0+lamb).argmin()
#Zero out the coefficients at these locations
fHat[[indPos,indNeg]] = 0.0
#Now we can inverse transform
fFilt = ifft(fHat)
#Plot
plt.plot(x,f,label='Original function')
plt.plot(x,fFilt,label='Filtered function')
plt.xlabel(r'$x$') ; plt.ylabel(r'$f$')
plt.legend(loc='best') ; plt.show()
```

4.2 Poisson's equation

The following shows how to solve Poisson's equation in a periodic 1D domain using FFTs.

```
#Program to solve 1D Poisson using FFTs
from scipy.fft import fft, fftfreq, ifft
import matplotlib.pyplot as plt
from numpy import linspace, sin, pi, sqrt, mean, zeros

#Define the length of our box
length = 1.0
```

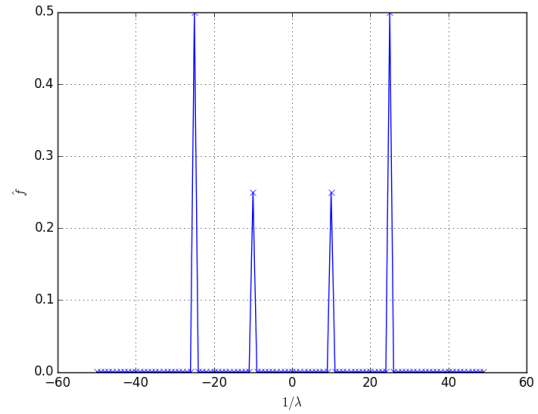

Figure 2: Example output for FFT task showing frequency spectrum with peaks at ± 10 and 25 as well as the real space structure pre and post filtering. Case: Full coefficient spectrum.

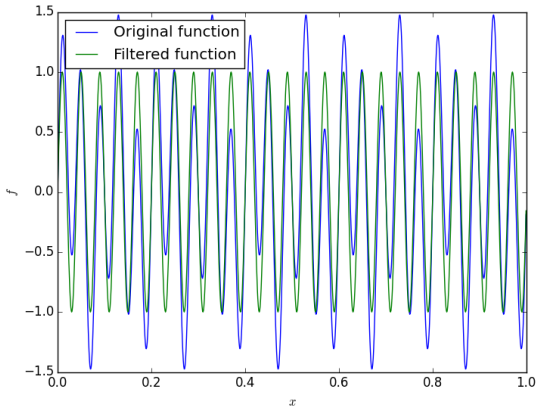

Figure 3: Example output for FFT task showing frequency spectrum with peaks at ± 10 and 25 as well as the real space structure pre and post filtering. Case: Original and filtered functions.

```
nx = 1000
x = linspace(0,length,nx,endpoint=False)
dx = x[1]-x[0]

#Make our rhs function, rho = sin(2 Pi (x/length))
rho = sin((x/length)*2*pi)

#Now FFT to get rhoHat
rhoHat = fft(rho)
#Get the inverse wavelengths
lambInv = fftfreq(nx,dx)
#Calculate wavenumbers
kx = lambInv*2*pi

#Now we can find phiHat using phiHat = rhoHat / kx^2 epsilon0
#Note: we've normalised such that epsilon0==1.0
phiHat = zeros(len(rhoHat),dtype='complex')#Have to make this complex!
#Note we skip the kx=0 component as this just defines a DC offset
#which we'll assume is zero
phiHat[1:] = rhoHat[1:] / kx[1:]**2
#Now we can find phi
phi = ifft(phiHat)

#Analytic solution
phiAn = sin(2*pi*x)/(2*pi)**2
err = phiAn-phi
rmsErr = sqrt(mean(err*err.conjugate()))
print("The rms error is {err}".format(err=rmsErr))
```

```
#Now plot
plt.plot(x,phi.real,'-',label=r'$\phi$ Numeric')
plt.plot(x[::10],phiAn[::10],'x',label=r'$\phi$ Analytic')
plt.xlabel(r'$x$') ; plt.ylabel(r'$\phi$')
plt.legend(loc='best'); plt.show()
```

4.2.1 Periodic boundaries in BVP

The following shows how to solve Poisson's equation in a periodic 1D domain using finite difference approximations. This approach is slower than the fft example shown above and results in a significantly bigger error for $n_x = 1000$.

```
#Imports
from scipy.sparse import lil_matrix
from scipy.sparse.linalg import spsolve
from numpy import zeros,linspace,abs,pi,sin,mean,sqrt
import matplotlib.pyplot as plt
from time import time

#Parameters
length = 1.0 ; nx = 1000
#Note we use nx-1 most places as we're skipping the repeated
#periodic point
xval=linspace(0,length,num=nx-1,endpoint=False)
dx = xval[1]-xval[0]

#Create matrix operator
M = lil_matrix((nx-1,nx-1))

#Set elements, -d^2/dx^2
rho = sin(xval*2*pi)
for i in range(0,nx-1):
    #Periodic indices
    im = i-1
    ip = (i+1) % (nx-1)
    M[i,im] = -1.0/dx**2
    M[i,i]   =  2.0/dx**2
    M[i,ip] = -1.0/dx**2

#Final setup and get solve routine
M = M.tocsr()
#Solve (and time)
t1 = time() ; phi = spsolve(M,rho) ; t2 = time()
#Note we can add any constant to our solution phi without
#changing the solution, hence we'll make phi[0] = 0.0
phi = phi-phi[0]

#Analytic solution
phiAn = sin(2*pi*xval)/(2*pi)**2
err = phiAn-phi
rmsErr = sqrt(mean(err*err.conjugate()))
print("The rms error is {err}".format(err=rmsErr))

#Plot solution
plt.plot(xval,phi,'-',label='Periodic BVP')
plt.plot(xval[::10],phiAn[::10],'x',label='Analytic solution')
plt.xlabel(r'$x$') ; plt.ylabel(r'$\phi$')
plt.legend(loc='best') ; plt.show()
```

4.3 Evolving equations

The following shows an example program to solve the system

$$\frac{\partial^2 \rho(x,t)}{\partial t^2} = -\phi \ ; -\frac{d^2\phi}{dx^2} = \rho(x)$$

```
from scipy.fft import fft, fftfreq, ifft, fftshift
from scipy.integrate import odeint
import matplotlib.pyplot as plt
from numpy import linspace, sin, pi, sqrt, mean, zeros, concatenate

def rhoToPhi(x,rho):
    """Takes the x grid and rho values and returns
        phi using an FFT integration approach
    """
    #Get properties of x-grid
    nx = len(x) ; dx = x[1]-x[0]
    #Now FFT to get rhoHat
    rhoHat = fft(rho)
    #Get the inverse wavelengths
    lambInv = fftfreq(nx,dx)
    #Calculate wavenumbers
    kx = lambInv*2*pi
    #Now we can find phiHat using phiHat = rhoHat / kx^2 epsilon0
    #Note: we've normalised such that epsilon0==1.0
    phiHat = zeros(len(rhoHat),dtype='complex')
    phiHat[1:] = rhoHat[1:] / kx[1:]**2
    phi = ifft(phiHat)
    return phi.real

def timeDeriv(state,time,x):
    """ Takes the state [rho,drho/dt=v] and returns
        [drho/dt,d^2rho/dt^2]. Also requires x to be passed
    """
    #Work out length of x
    nx = len(x)
    #Unpack
    rho = state[:nx]
    v = state[nx:]

    #Calculate the electrostatic potential
    phi = rhoToPhi(x,rho)
    #Return the time derivatives
    return concatenate([v,-phi])

#Setup the problem
nx = 50 ; length = 1.0
x = linspace(0.0,length,nx,endpoint=False)
dx = x[1]-x[0]

#Determine the initial conditions
rho0 = sin(2*pi*x/length)+sin(2*pi*5*x/length)
drhodt0 = 0.0*rho0
initVal = concatenate([rho0,drhodt0])
#Times for output
t = linspace(0,200.0,100)
#Get result
result = odeint(timeDeriv,initVal,t,args=(x,))

#Unpack result
rho = result[:,:nx] #All times and the first nx points
#Make a contour plot
plt.contourf(x,t,rho,64)
plt.xlabel(r'$x$') ; plt.ylabel(r'$t$')
plt.colorbar(label=r'$\rho$')
plt.show()
```

We can look at how the different coefficients evolve in time, and an example is shown below. This shows us that the period of oscillation depends on the wavelength, such that the product of the wavelength and the frequency is constant. Example output is shown in fig. 6

```
#Look at FFT coefficients
rhoHat = (1.0/nx)*(fft(rho,axis=1).imag)
```

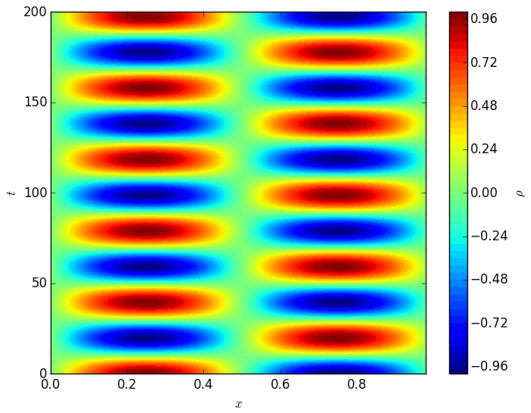

Figure 4: Example output for evolving ρ task showing different scale structures oscillating in time. Case: $\rho\left(t=0\right)=\sin\left(2\pi x/L\right)$

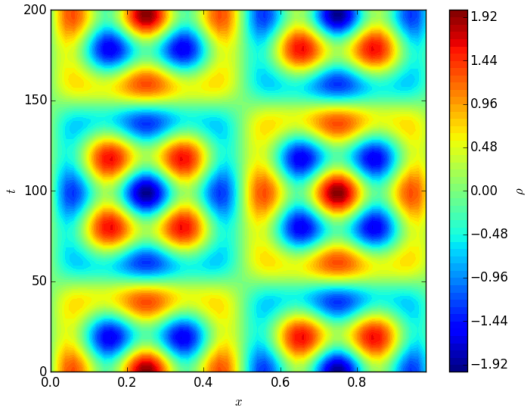

Figure 5: Example output for evolving ρ task showing different scale structures oscillating in time. Case: $\rho\left(t=0\right)=\sin\left(2\pi x/L\right)+\sin\left(2\pi 5x/L\right)$

```
invLamb = fftfreq(nx,dx)
plt.figure(2)
plt.contourf(fftshift(invLamb),t,fftshift(rhoHat,axes=1),64)
plt.xlabel(r'$1/\lambda$') ; plt.ylabel(r'$t$')
plt.colorbar(label=r'$\hat{\rho}$')
plt.show()
```

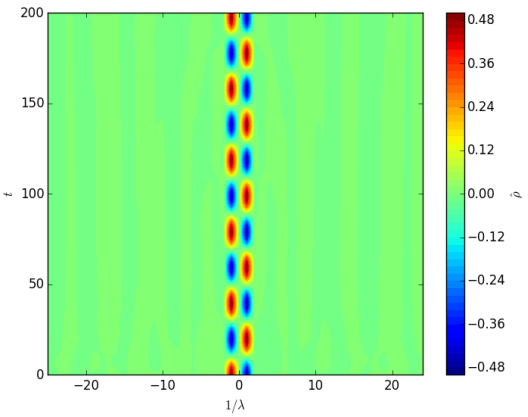

Figure 6: Example output for evolving $\hat{\rho}$ task showing oscillation of specific wavenlength components. Case : $\rho\left(t=0\right)=\sin\left(2\pi x/L\right)$

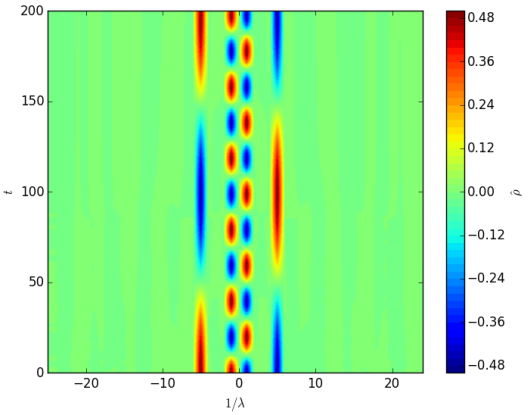

Figure 7: Example output for evolving $\hat{\rho}$ task showing oscillation of specific wavenlength components. Case : $\rho\left(t=0\right)=\sin\left(2\pi x/L\right)+\sin\left(2\pi 5x/L\right)$