In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
```

Part 1

We wish to generate random functions that, contrary to conventional non-differentiable noise, are smooth. There are two limits of the normalization can be used. One used in the case where we care about the absolute value of the function and the other where we look at the case where we care about the integrated value of the function (as we do in the case of SDEs). In the second case, the normalization has an extra factor of $\sqrt{\frac{2}{\lambda}}$.

Part 2

Let us define some wavelength parameter λ. This quantity in some sense is giving a length scale to the oscillations in Fourier space. We limit the upper bound on the available frequencies in the Fourier series. Let L denote the length of the interval of interest.

We can now write a smooth periodic random function as

$$f(x) = \sum_{j=-m}^{m} c_j \exp(\frac{2\pi i j x}{L}) \qquad m = \lfloor \frac{L}{\lambda} \rfloor, c_j \in N(0, \frac{1}{2m+1}) + iN(0, \frac{1}{2m+1})$$

A real function can be written as

$$f(x) = a_0 + \sqrt{2}\sum_{j=1}^{m} \left(a_j \cos(\frac{2\pi j x}{L}) + b_j \sin(\frac{2\pi j x}{L}) \right) \qquad a_j, b_j \in N(0, \frac{1}{2m+1})$$

Note that in both cases, we ensure that the value of the function at any point has unit variance. There are two ways of constructing the function from some functional basis defined on the interval at hand. One is of course, the Fourier basis of complex exponentials, and the other is a basis consisting of Dirichlet Kernals. A Dirichlet Kernal is defined as

$$D(x) = \frac{\sin(\frac{(2m+1)\pi x}{L})}{(2m+1)\sin(\frac{\pi x}{L})}$$

Here, what we have done is that we have divided the space in to equally spaced points $x_j = j\Delta$ where $\Delta = \frac{L}{2m+1}$. The kernal is defined in such a way that it takes a value of 1 at $x = 0$ and 0 at all other interpolating points. Thus, we can decompose the function as

$$f(x) = \sum_{j=-m}^{j=m} d_j D(x - x_j) \qquad d_j = f(x_j)$$

We can relate the coefficients in the Fourier and the Dirichlet expansion via a matrix equation

$$\mathbf{d} = F\mathbf{c}$$

Here, we have the matrix F to be a Vandermonde matrix made from $1, \omega, \omega^2 \ldots$ where $\omega = \exp(\frac{2\pi i}{2m+1})$. This tells us that F is almost unitary i.e. $\frac{1}{\sqrt{2m+1}}F$ is unitary. Thus, the nature of the distributions of $\mathbf{c}$ and $\mathbf{d}$ are the same. In particular, we have

$$p(\mathbf{c}) = C\exp(-(2m+1)||\mathbf{c}||^2/2)$$

$$p(\mathbf{d}) = C\exp(-||\mathbf{d}||^2/2)$$

Simple demonstration of the two decompositions

In [131…

```
def randfourier(m,L,x):
    sarr=[]
    v=1/(1+2*m)**0.5
    a=np.random.normal(0,v,m+1)
    b=np.random.normal(0,v,m+1)
    for j in range(len(x)):
        s=0
        for i in range(0,m+1):
            s=s+np.sqrt(2)*a[i]*np.cos(2*np.pi*x[j]*i/L)+np.sqrt(2)*b[i]*np.sin(2*np
        sarr.append(s)
    return sarr
```

In [5]:

```
def dirichkern(m,L,x):
    if x==0:
        return 1
    else:
        return (np.sinc((2*m + 1)*x/L)/np.sinc(x/L))
```

In [49]:

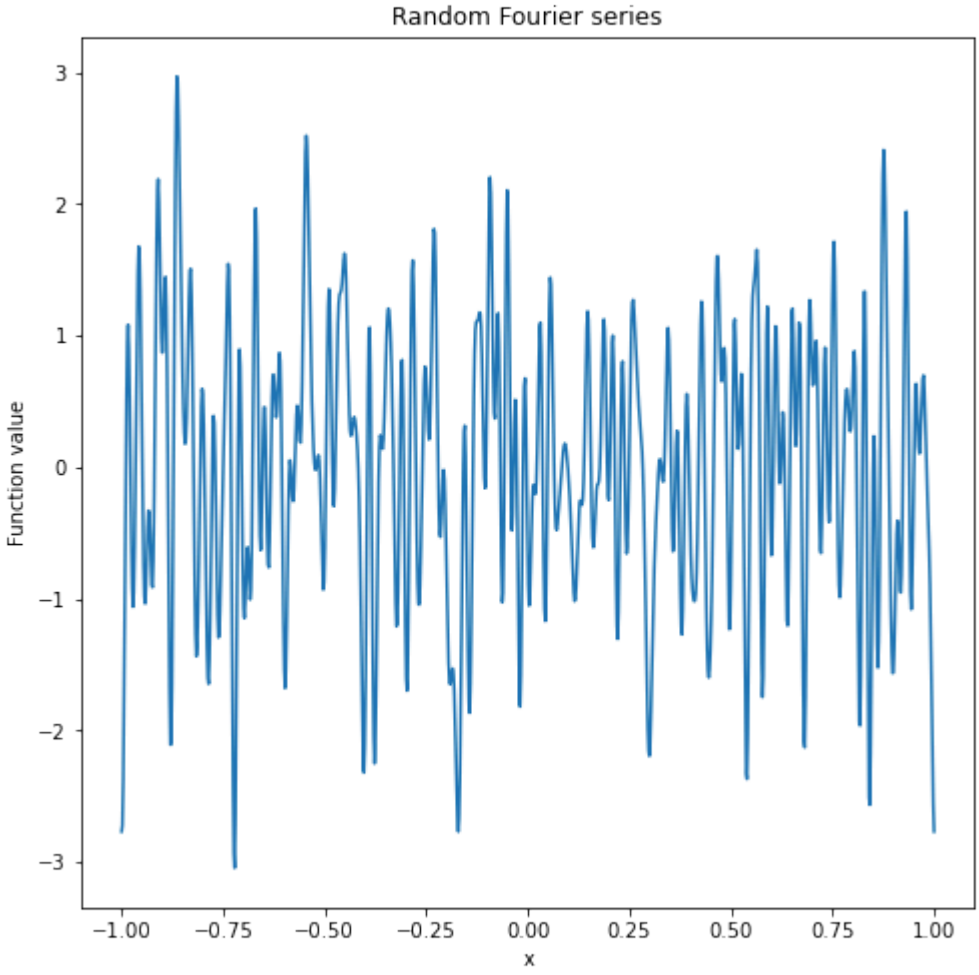
```
def randdirichlet(m,L,x):
    s=0
    v=1/(1+2*m)
    delta=L/(2*m+1)
    d=np.random.normal(0,v,2*m+2)
    for i in range(0,2*m+2):
        x1=(i*delta)-(L/2)
        s=s+d[i]*dirichkern(m,L,x-x1)
    return s
```

In [132…

```
x=np.linspace(-1,1,1000)
plt.figure(figsize=(8,8))
plt.title('Random Fourier series')
plt.xlabel('x')
plt.ylabel('Function value')
plt.plot(x,randfourier(100,2,x))
```

Out[132…

```
[<matplotlib.lines.Line2D at 0x1fe2014c940>]
```

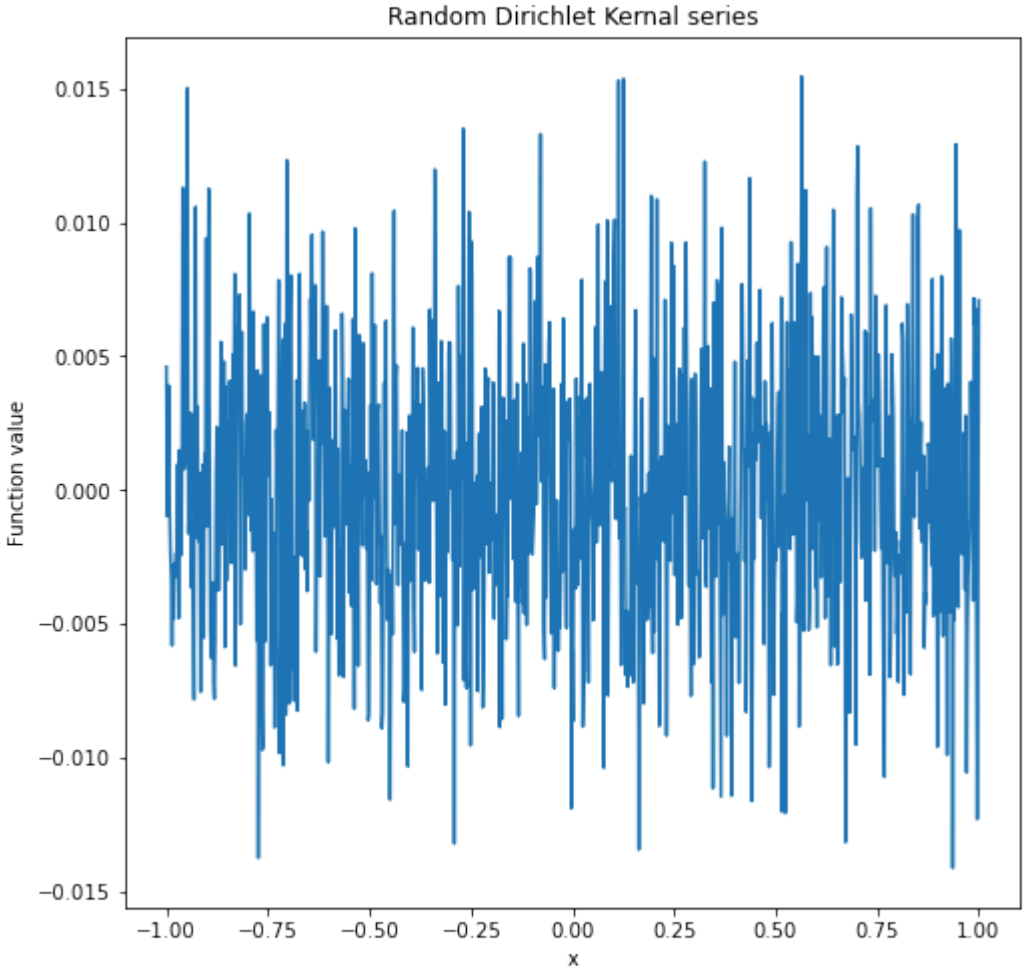


In [50]:

```
x=np.linspace(-1,1,1000)
plt.figure(figsize=(8,8))
plt.title('Random Dirichlet Kernal series')
plt.xlabel('x')
plt.ylabel('Function value')
plt.plot(x,[randdirichlet(100,2,x[i]) for i in range(len(x))])
```

Out[50]:

```
[<matplotlib.lines.Line2D at 0x1fe128313d0>]
```



Smooth Random Noise from Random Fourier series

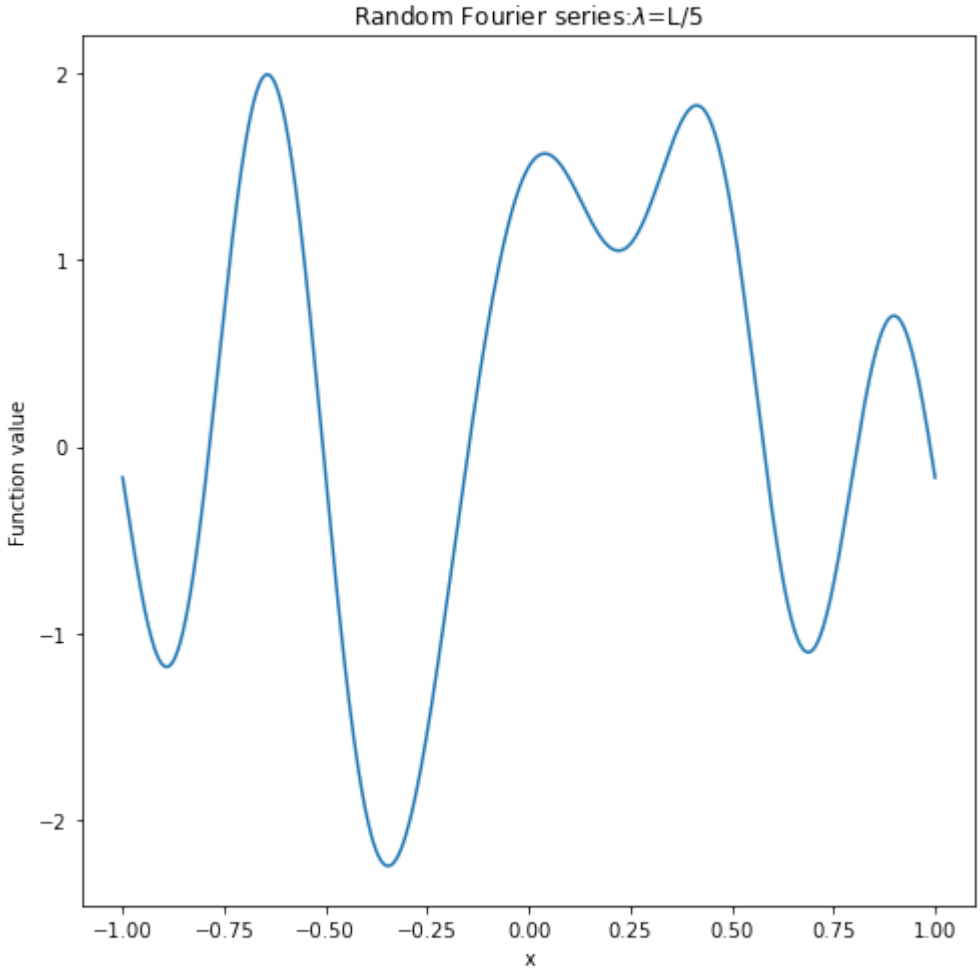
Varying λ

In [133…

```
x=np.linspace(-1,1,1000)
plt.figure(figsize=(8,8))
plt.title(r'Random Fourier series:$\lambda$=L/5 ')
plt.xlabel('x')
plt.ylabel('Function value')
plt.plot(x,randfourier(5,2,x))
```

Out[133…

```
[<matplotlib.lines.Line2D at 0x1fe1fe745e0>]
```

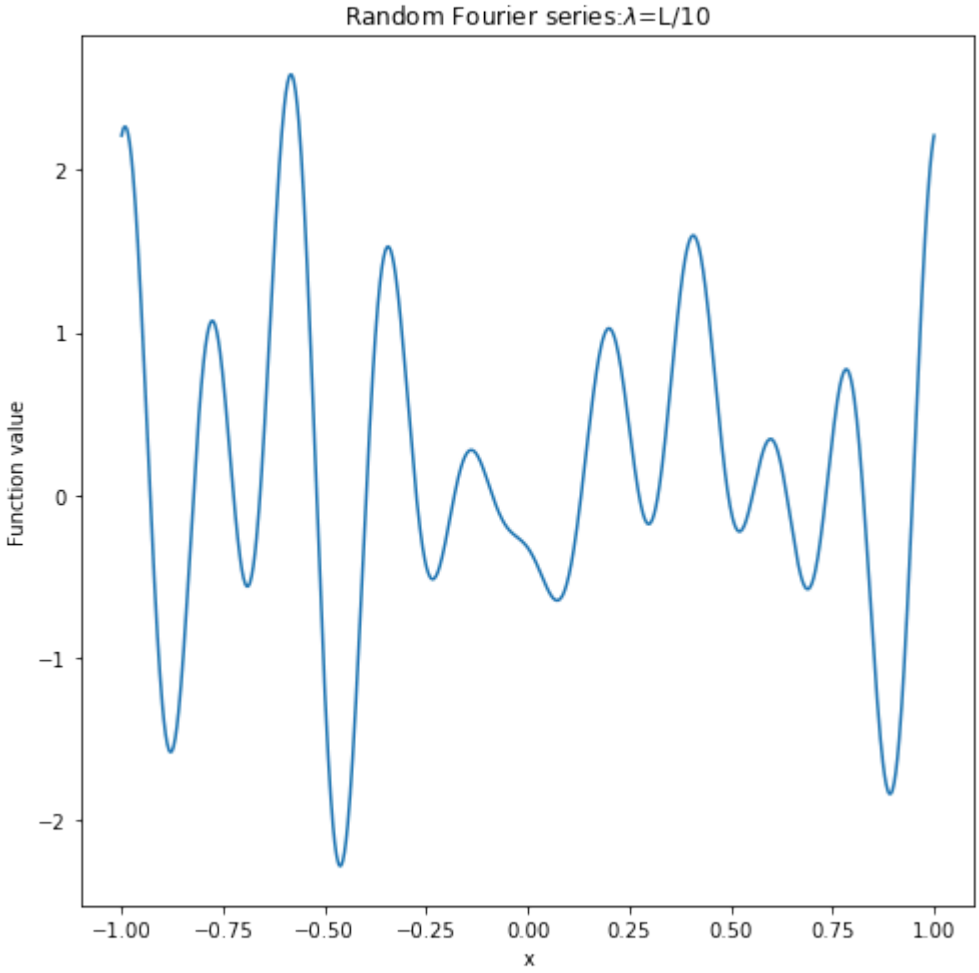


In [134…

```
x=np.linspace(-1,1,1000)
plt.figure(figsize=(8,8))
plt.title(r'Random Fourier series:$\lambda$=L/10 ')
plt.xlabel('x')
plt.ylabel('Function value')
plt.plot(x,randfourier(10,2,x))
```

Out[134…

```
[<matplotlib.lines.Line2D at 0x1fe1fee64c0>]
```

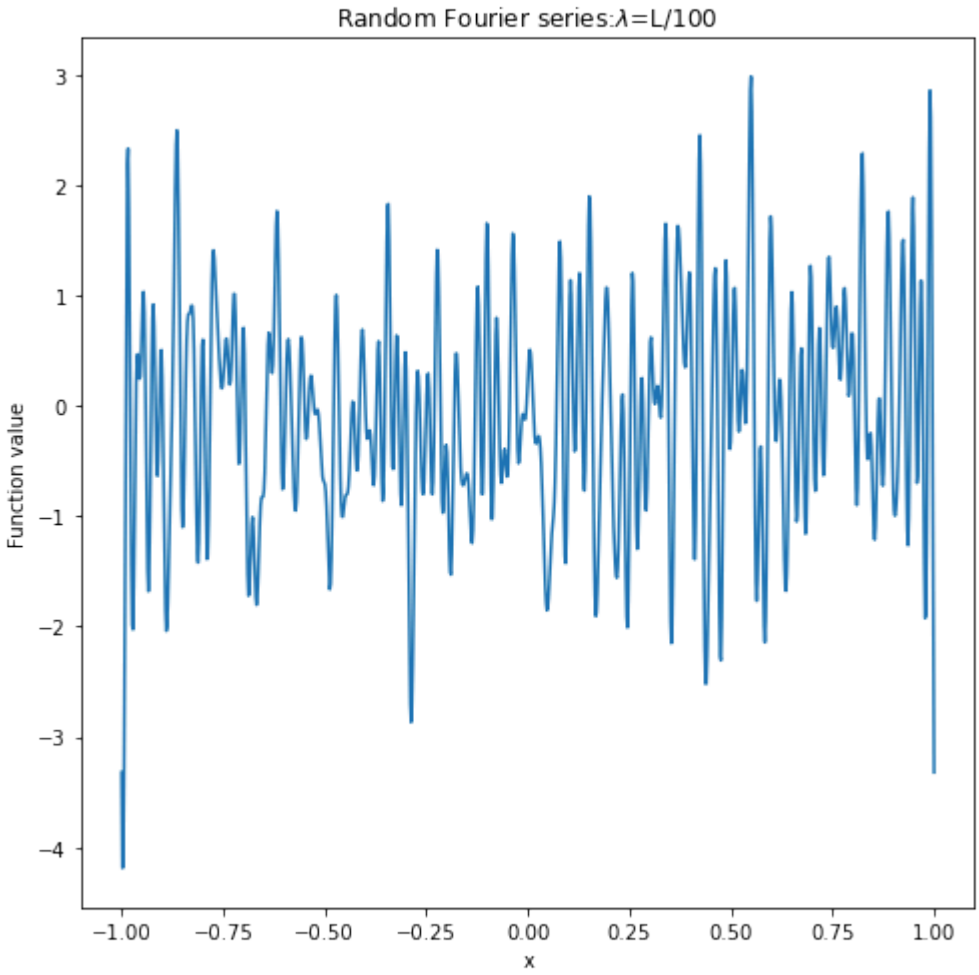


In [135…

```
x=np.linspace(-1,1,1000)
plt.figure(figsize=(8,8))
plt.title(r'Random Fourier series:$\lambda$=L/100 ')
plt.xlabel('x')
plt.ylabel('Function value')
plt.plot(x,randfourier(100,2,x))
```

Out[135…

```
[<matplotlib.lines.Line2D at 0x1fe1ff525e0>]
```

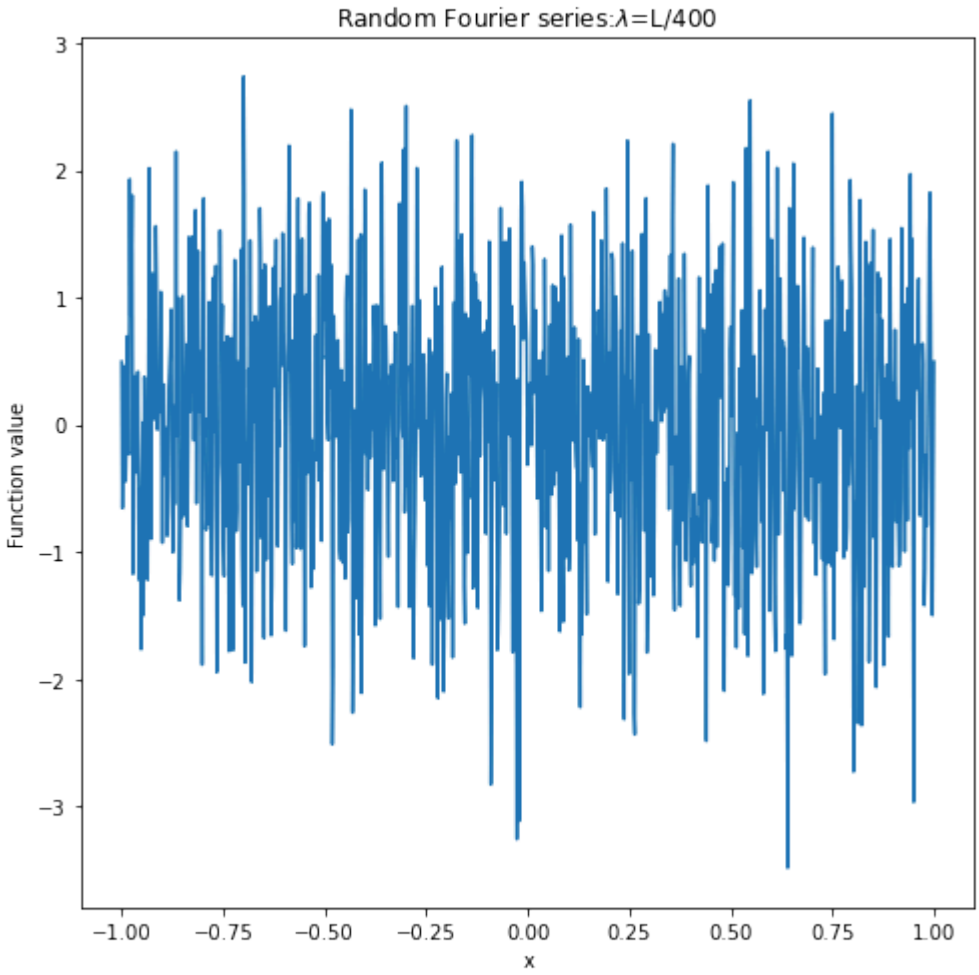


In [136…

```
x=np.linspace(-1,1,1000)
plt.figure(figsize=(8,8))
plt.title(r'Random Fourier series:$\lambda$=L/400 ')
plt.xlabel('x')
plt.ylabel('Function value')
plt.plot(x,randfourier(400,2,x))
```

Out[136…

```
[<matplotlib.lines.Line2D at 0x1fe212b3670>]
```



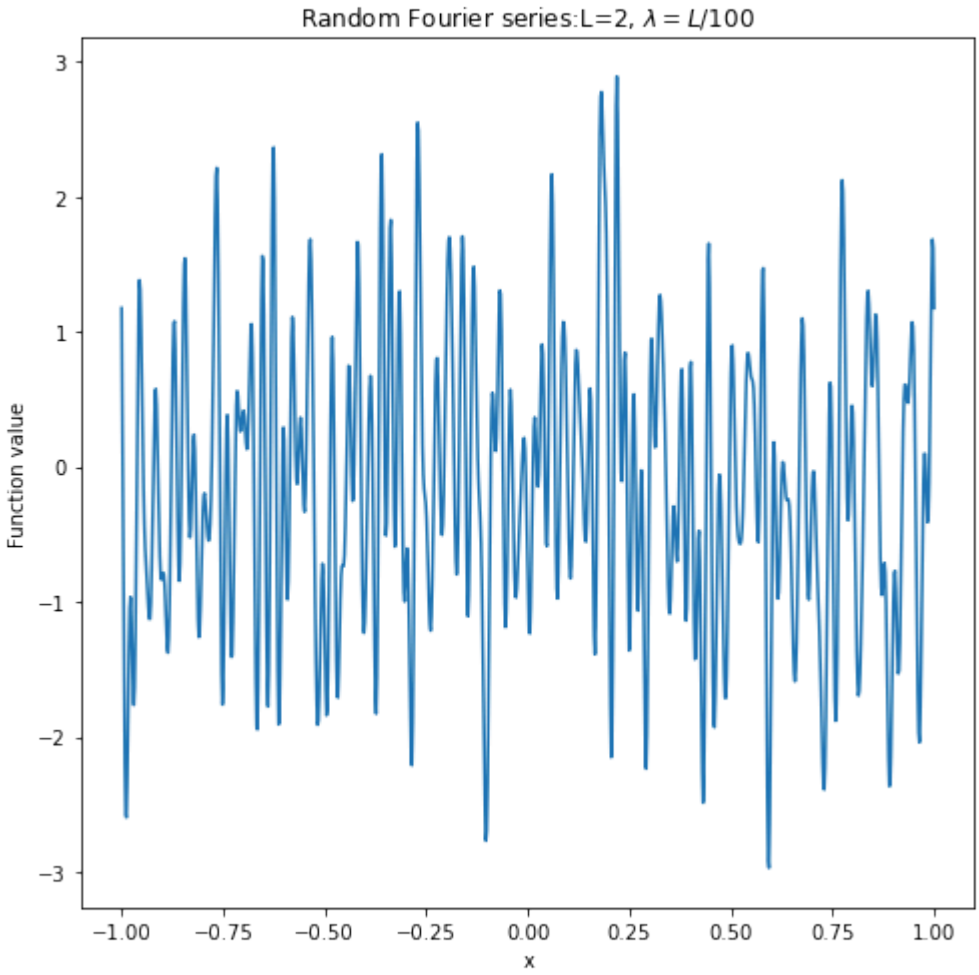
Varying L

In [137…

```
x=np.linspace(-1,1,1000)
plt.figure(figsize=(8,8))
plt.title(r'Random Fourier series:L=2, $\lambda=L/100$')
plt.xlabel('x')
plt.ylabel('Function value')
plt.plot(x,randfourier(100,2,x))
```

Out[137…

```
[<matplotlib.lines.Line2D at 0x1fe21196280>]
```

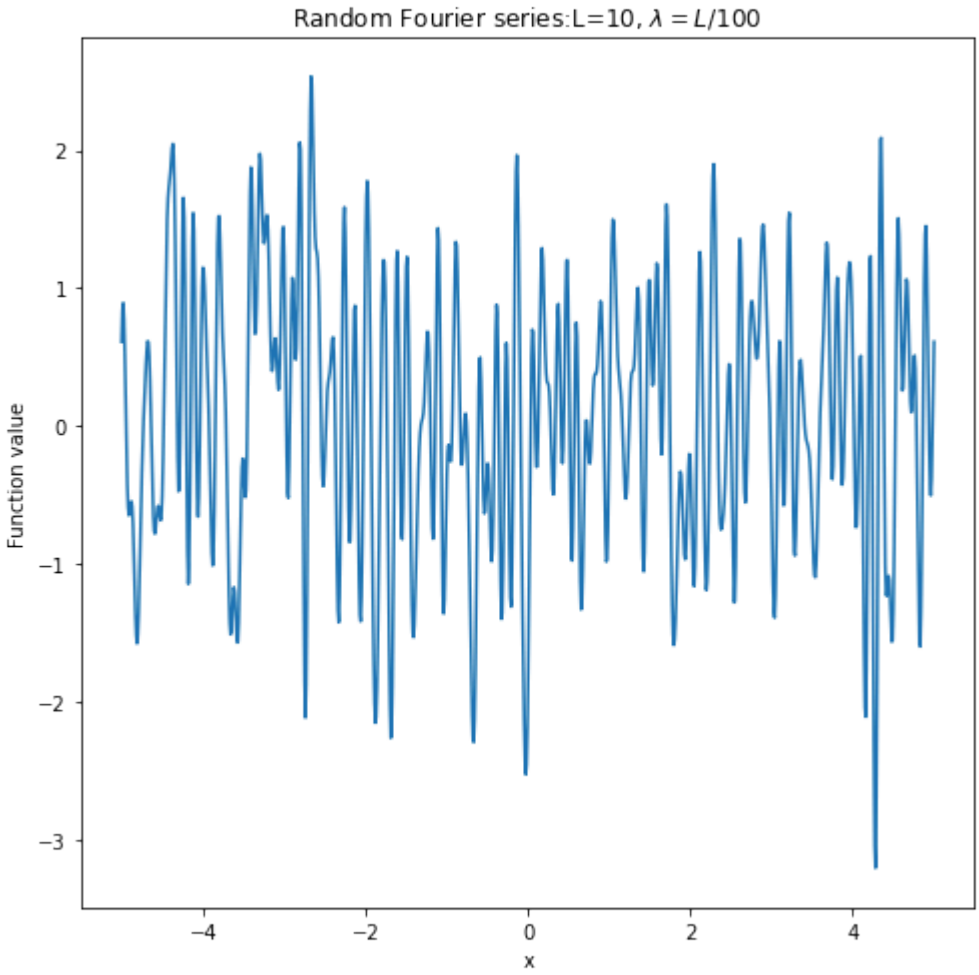


In [138…

```
x=np.linspace(-5,5,1000)
plt.figure(figsize=(8,8))
plt.title(r'Random Fourier series:L=10, $\lambda=L/100$')
plt.xlabel('x')
plt.ylabel('Function value')
plt.plot(x,randfourier(100,10,x))
```

Out[138…

```
[<matplotlib.lines.Line2D at 0x1fe211f4bb0>]
```

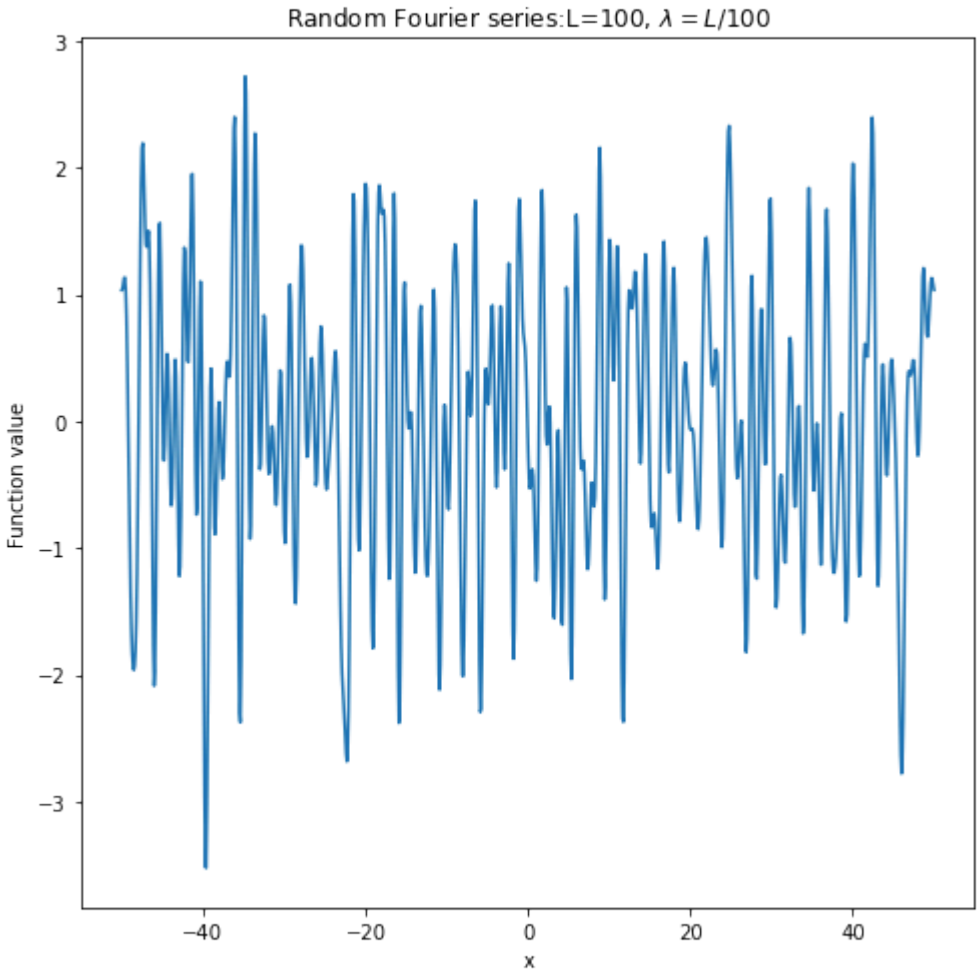


In [139…

```
x=np.linspace(-50,50,1000)
plt.figure(figsize=(8,8))
plt.title(r'Random Fourier series:L=100, $\lambda=L/100$')
plt.xlabel('x')
plt.ylabel('Function value')
plt.plot(x,randfourier(100,100,x))
```

Out[139…

```
[<matplotlib.lines.Line2D at 0x1fe21263a90>]
```



Part 3

As done in the paper, we can compare the amplitudes at a fixed λ

In [145…

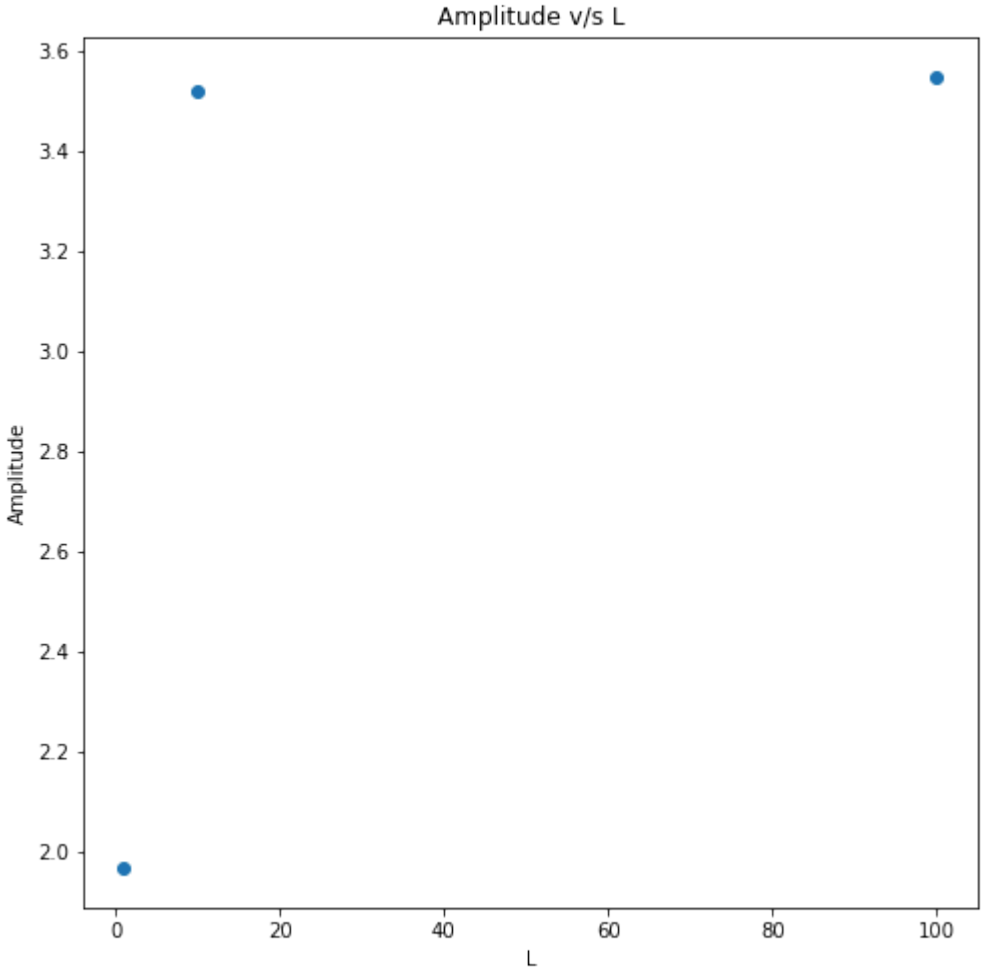
```
Larr=[1,10,100]
ld=0.05
amp=[]
for i in range(len(Larr)):
    x=np.linspace(-1*Larr[i]/2,Larr[i]/2,1000)
    m=20*Larr[i]
    func=randfourier(m,Larr[i],x)
    maxi=max(func)
    mini=min(func)
    amp.append((maxi-mini)/2)
```

In [146…

```
plt.figure(figsize=(8,8))
plt.title('Amplitude v/s L')
plt.xlabel('L')
plt.ylabel('Amplitude')
plt.scatter(Larr,amp)
```

Out[146…

```
<matplotlib.collections.PathCollection at 0x1fe212f78b0>
```

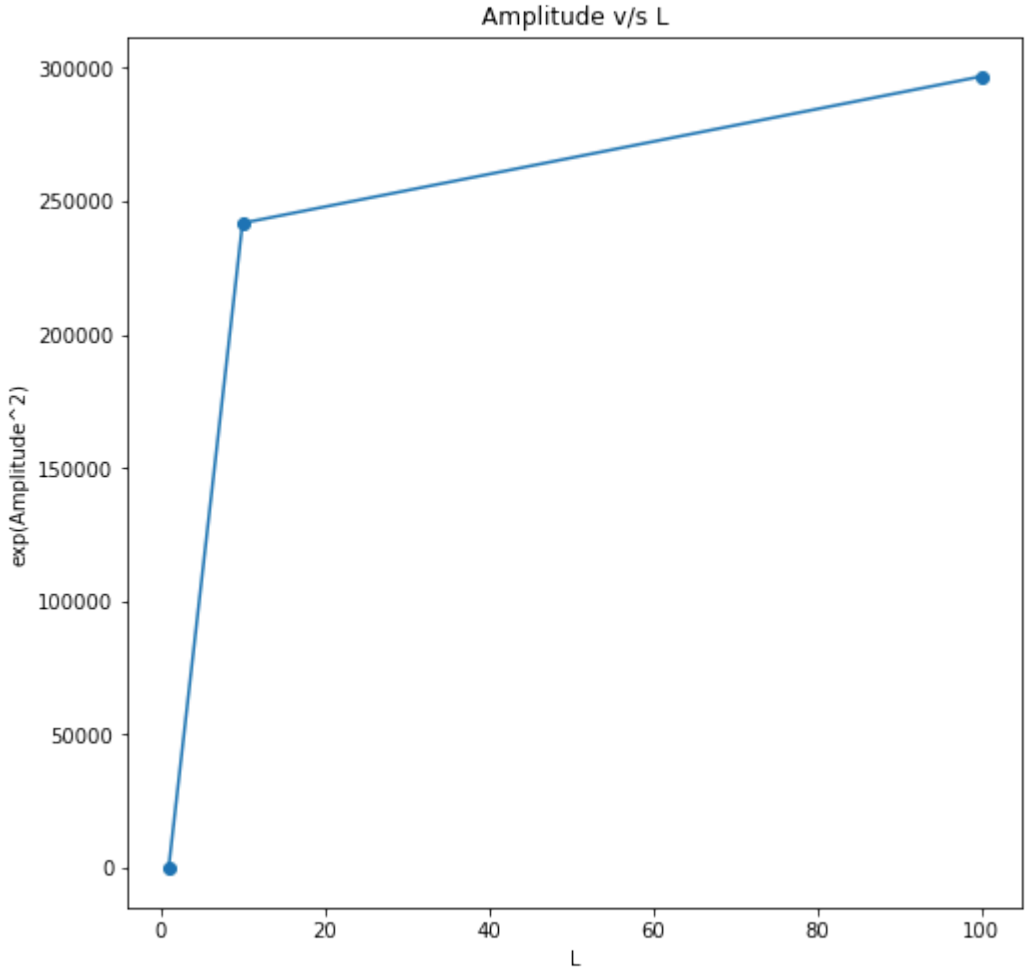


In [147…

```
plt.figure(figsize=(8,8))
plt.title('Amplitude v/s L')
plt.xlabel('L')
plt.ylabel('exp(Amplitude^2)')
plt.scatter(Larr,[np.exp(amp[i]*amp[i]) for i in range(len(amp))])
plt.plot(Larr,[np.exp(amp[i]*amp[i]) for i in range(len(amp))])
```

Out[147…

```
[<matplotlib.lines.Line2D at 0x1fe213979d0>]
```



Dyson Brownian motion

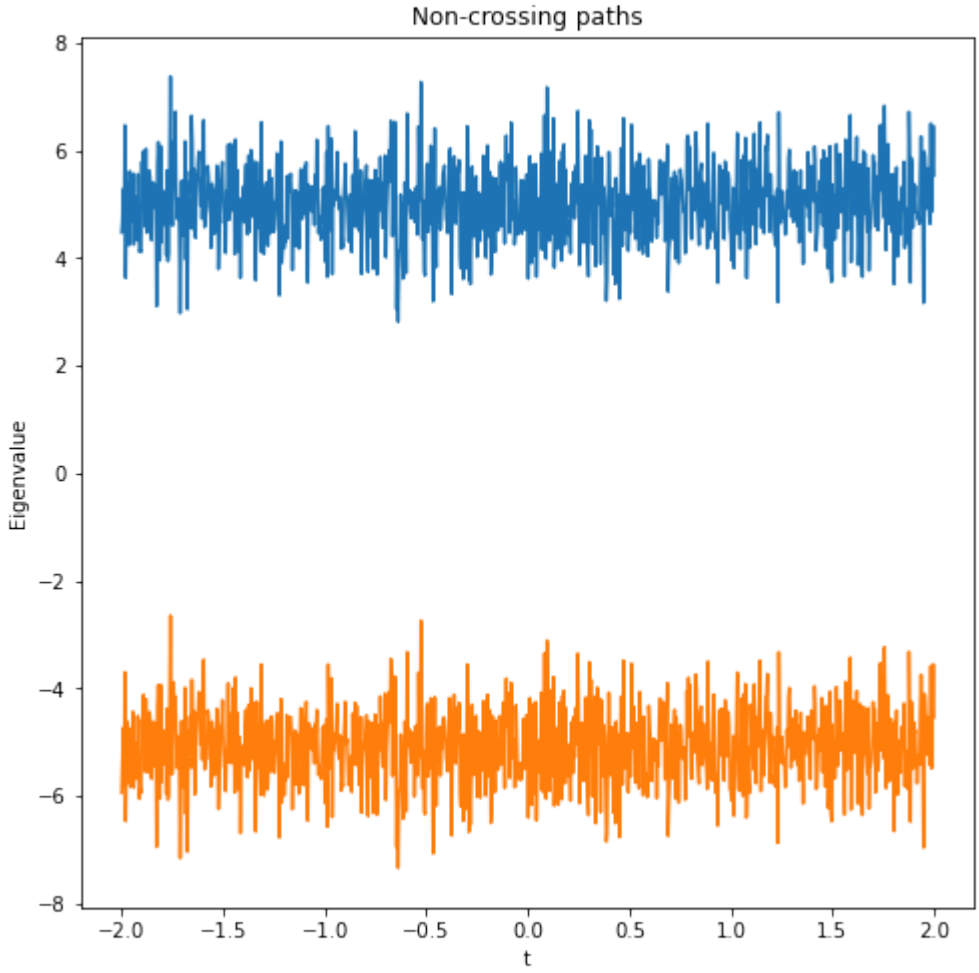
We can see two non-crossing paths that are generated from smooth random functions.

In [151…

```
eps=5
t=np.linspace(-2,2,1000)
e1=[]
e2=[]
for i in range(len(t)):
    tx=[t[i]]
    a11=randfourier(4,4,tx)
    a22=randfourier(4,4,tx)
    e1.append(((a11[0]+a22[0])+np.sqrt((a11[0]-a22[0])**2 + 4*eps*eps))/2)
    e2.append(((a11[0]+a22[0])-np.sqrt((a11[0]-a22[0])**2 + 4*eps*eps))/2)
plt.figure(figsize=(8,8))
plt.title('Non-crossing paths')
plt.xlabel('t')
plt.ylabel('Eigenvalue')
plt.plot(t,e1)
plt.plot(t,e2)
```

Out[151…

```
[<matplotlib.lines.Line2D at 0x1fe2176bc70>]
```



Part 4

We can define big normalization by replacing the variance of each of the coefficients in the initial definition by a rescaled verion of the original version as $V = \frac{2}{\lambda(2m+1)}$***. We also scale the entire thing by*** $\sqrt{\frac{2}{\lambda}}$.

In [156…

```
def bigrandfourier(m,L,x):
    sarr=[]
    v=2*m/(L*(1+2*m)**0.5)
    a=np.random.normal(0,v,m+1)
    b=np.random.normal(0,v,m+1)
    for j in range(len(x)):
        s=0
        for i in range(0,m+1):
            s=s+np.sqrt(2*m/L)*a[i]*np.cos(2*np.pi*x[j]*i/L)+np.sqrt(2*m/L)*b[i]*np.
        sarr.append(s)
    return sarr
```

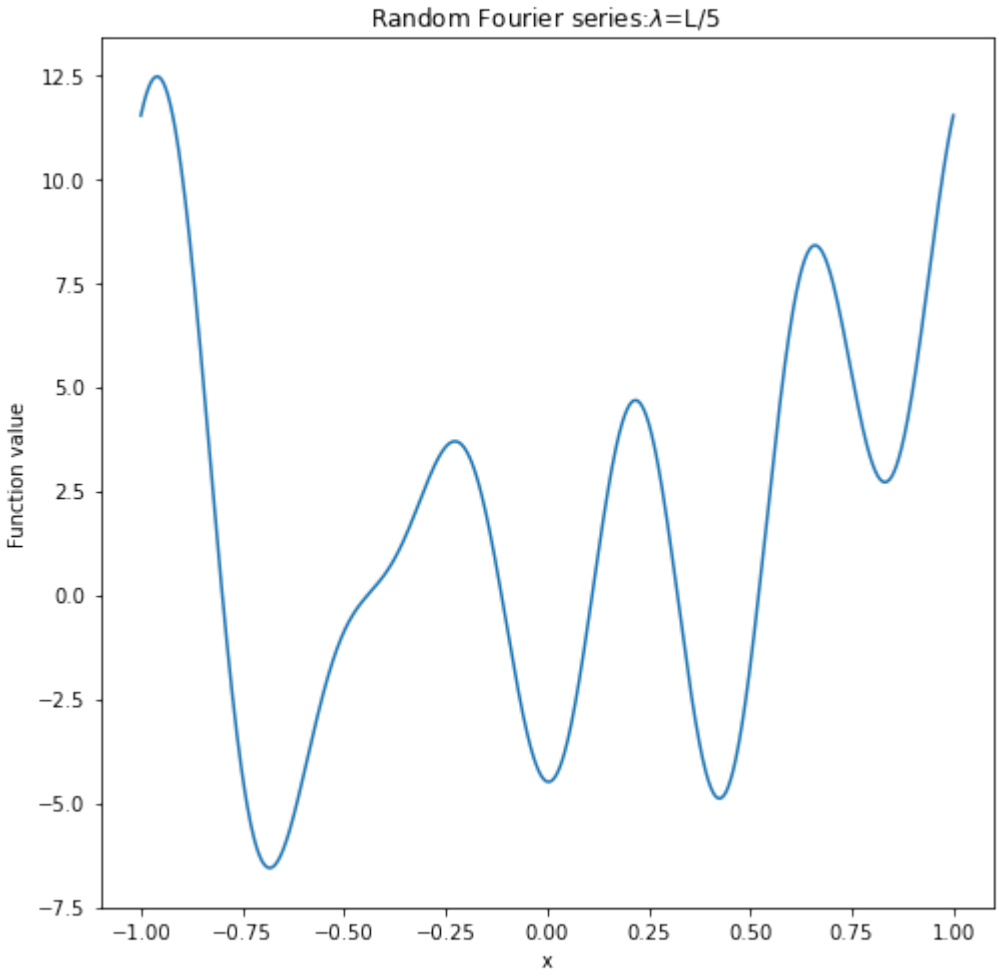
Varying λ

In [157…

```
x=np.linspace(-1,1,1000)
plt.figure(figsize=(8,8))
plt.title(r'Random Fourier series:$\lambda$=L/5 ')
plt.xlabel('x')
plt.ylabel('Function value')
plt.plot(x,bigrandfourier(5,2,x))
```

Out[157…

```
[<matplotlib.lines.Line2D at 0x1fe228cb8e0>]
```

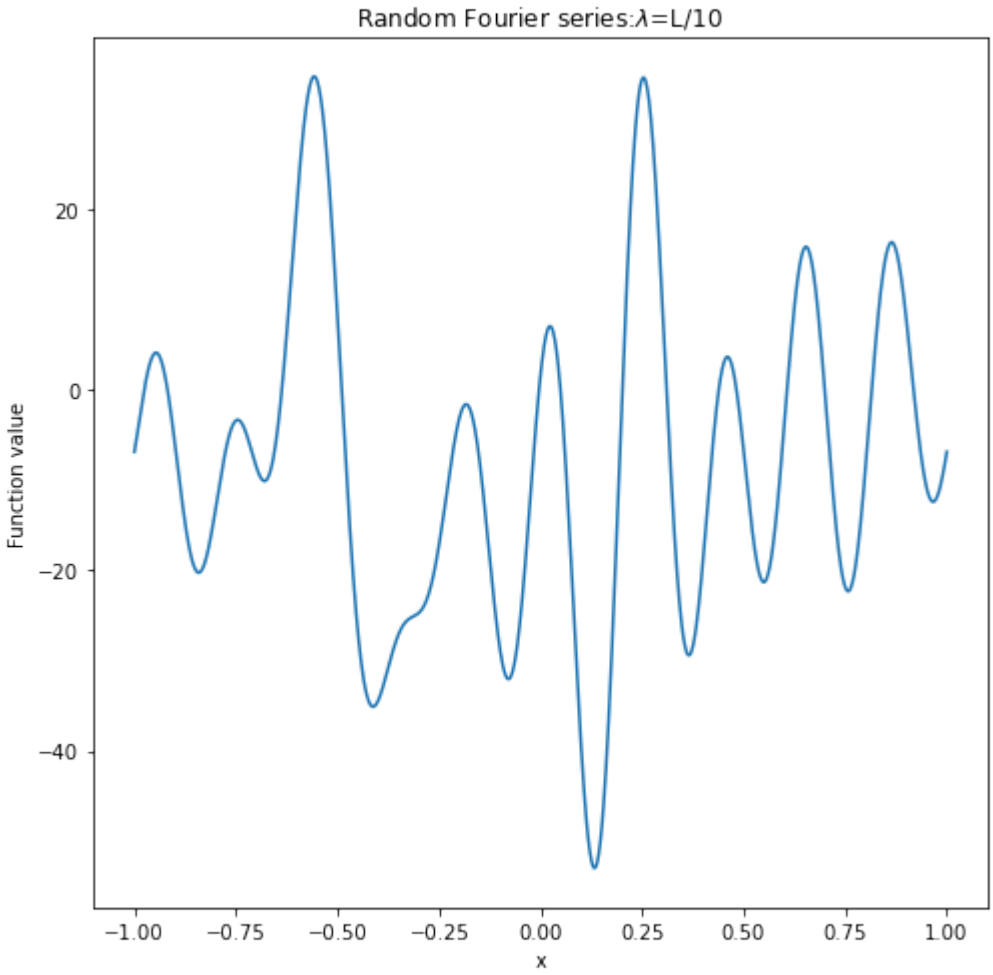


In [158…

```
x=np.linspace(-1,1,1000)
plt.figure(figsize=(8,8))
plt.title(r'Random Fourier series:$\lambda$=L/10 ')
plt.xlabel('x')
plt.ylabel('Function value')
plt.plot(x,bigrandfourier(10,2,x))
```

Out[158…

```
[<matplotlib.lines.Line2D at 0x1fe2277f940>]
```

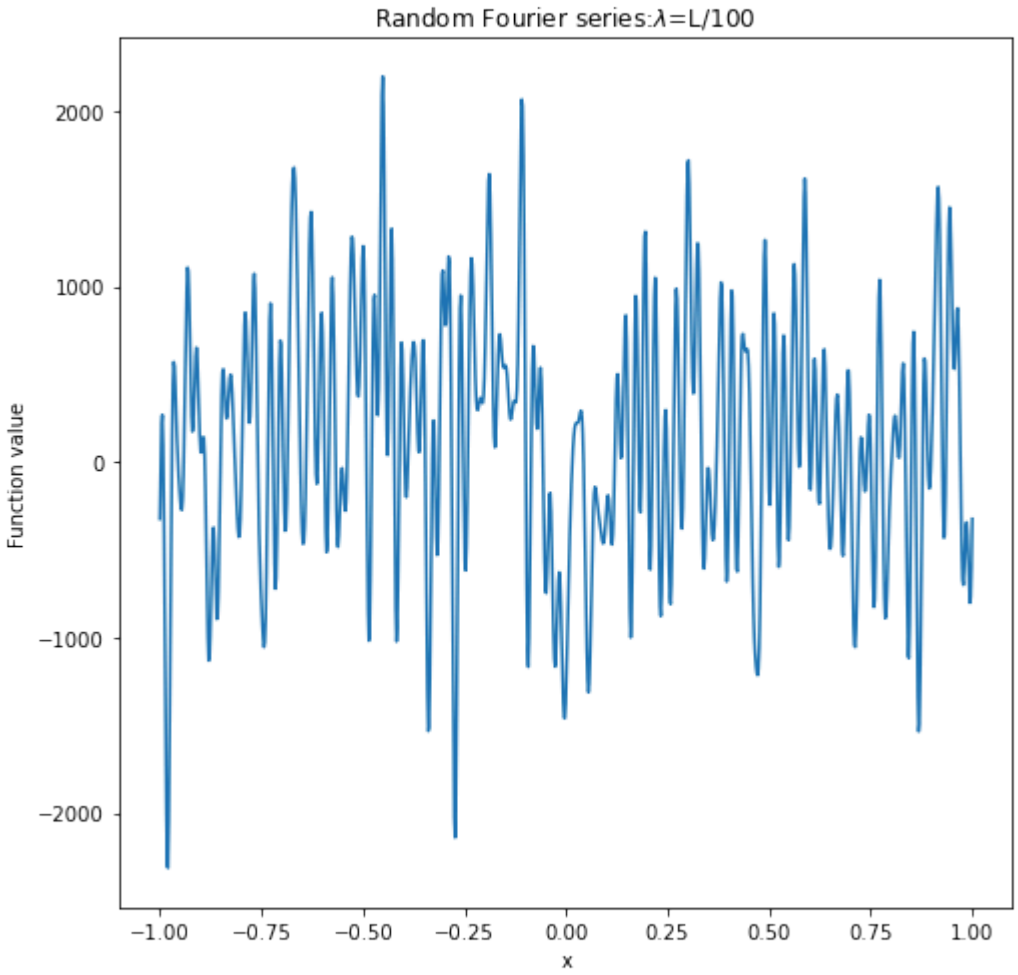


In [159…

```
x=np.linspace(-1,1,1000)
plt.figure(figsize=(8,8))
plt.title(r'Random Fourier series:$\lambda$=L/100 ')
plt.xlabel('x')
plt.ylabel('Function value')
plt.plot(x,bigrandfourier(100,2,x))
```

Out[159…

```
[<matplotlib.lines.Line2D at 0x1fe212ceca0>]
```



As is expected, the amplitude increases in order to maintain the integral's finiteness.

Complex Smooth Random Functions

As defined before, we can also compute smooth complex valued random functions.

However, due to the periodic nature of the function, the path it traces in the Argand plane is closed.

In [160…

```
def complexrandfourier(m,L,x):
    sarr=[]
    sari=[]
    v=1/(1+2*m)**0.5
    a=np.random.normal(0,v,2*m+1)
    b=np.random.normal(0,v,2*m+1)
    for j in range(len(x)):
        sr=0
        si=0
        for i in range(-m,m+1):
            sr=sr+a[i+m]*np.cos(2*np.pi*x[j]*i/L)-b[i+m]*np.sin(2*np.pi*x[j]*i/L)
            si=si+a[i+m]*np.sin(2*np.pi*x[j]*i/L)+b[i+m]*np.cos(2*np.pi*x[j]*i/L)
        sarr.append(sr)
        sari.append(si)
    return sarr,sari
```

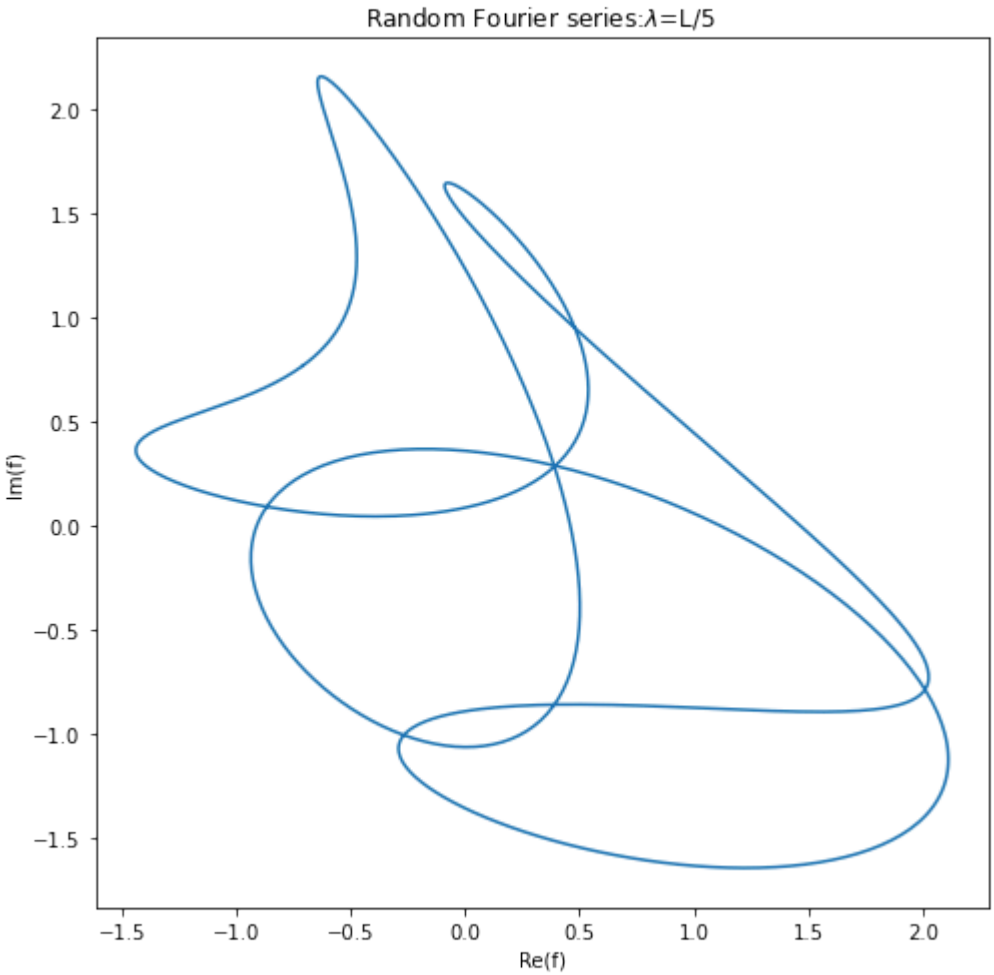
In [171…

```
x=np.linspace(-1,1,1000)
re,im=complexrandfourier(5,2,x)
plt.figure(figsize=(8,8))
plt.title(r'Random Fourier series:$\lambda$=L/5 ')
plt.xlabel('Re(f)')
```

```
plt.ylabel('Im(f)')
plt.plot(re,im)
```

Out[171…

```
[<matplotlib.lines.Line2D at 0x1fe22be2af0>]
```

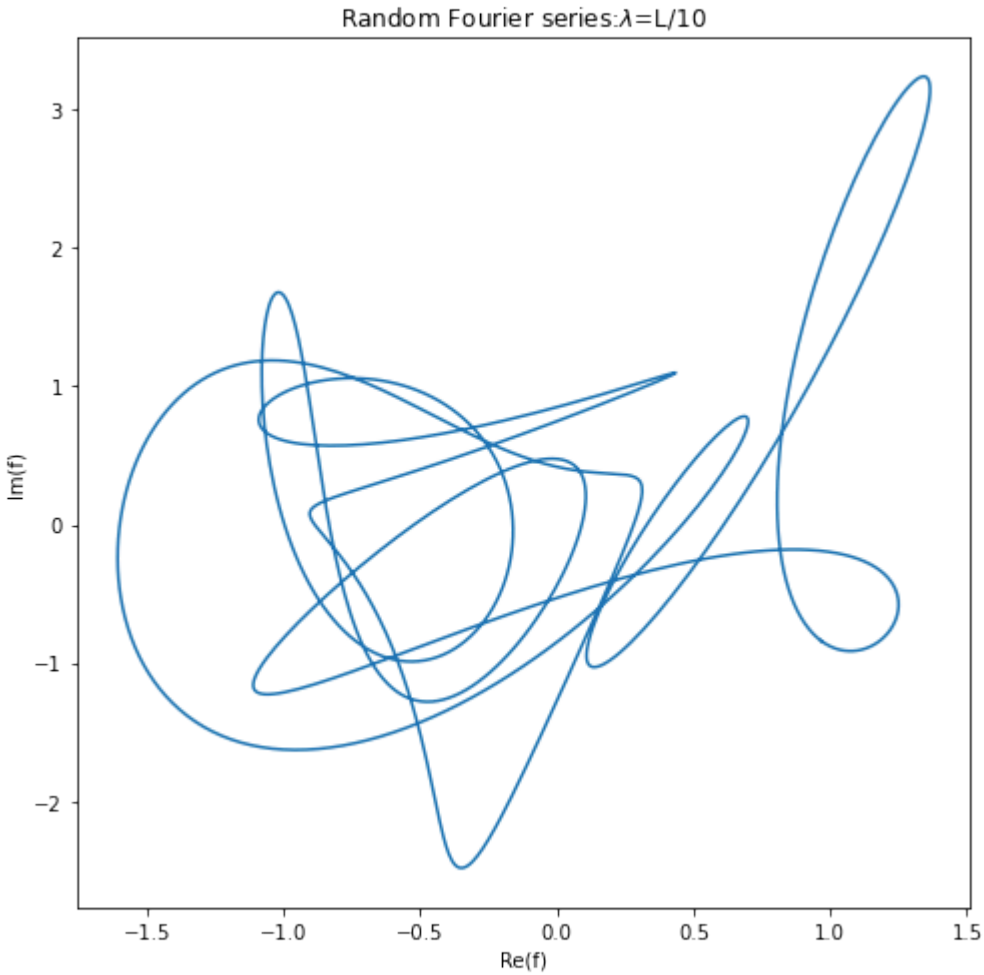


In [163…

```
x=np.linspace(-1,1,1000)
re,im=complexrandfourier(10,2,x)
plt.figure(figsize=(8,8))
plt.title(r'Random Fourier series:$\lambda$=L/10 ')
plt.xlabel('Re(f)')
plt.ylabel('Im(f)')
plt.plot(re,im)
```

Out[163…

```
[<matplotlib.lines.Line2D at 0x1fe2281cb80>]
```

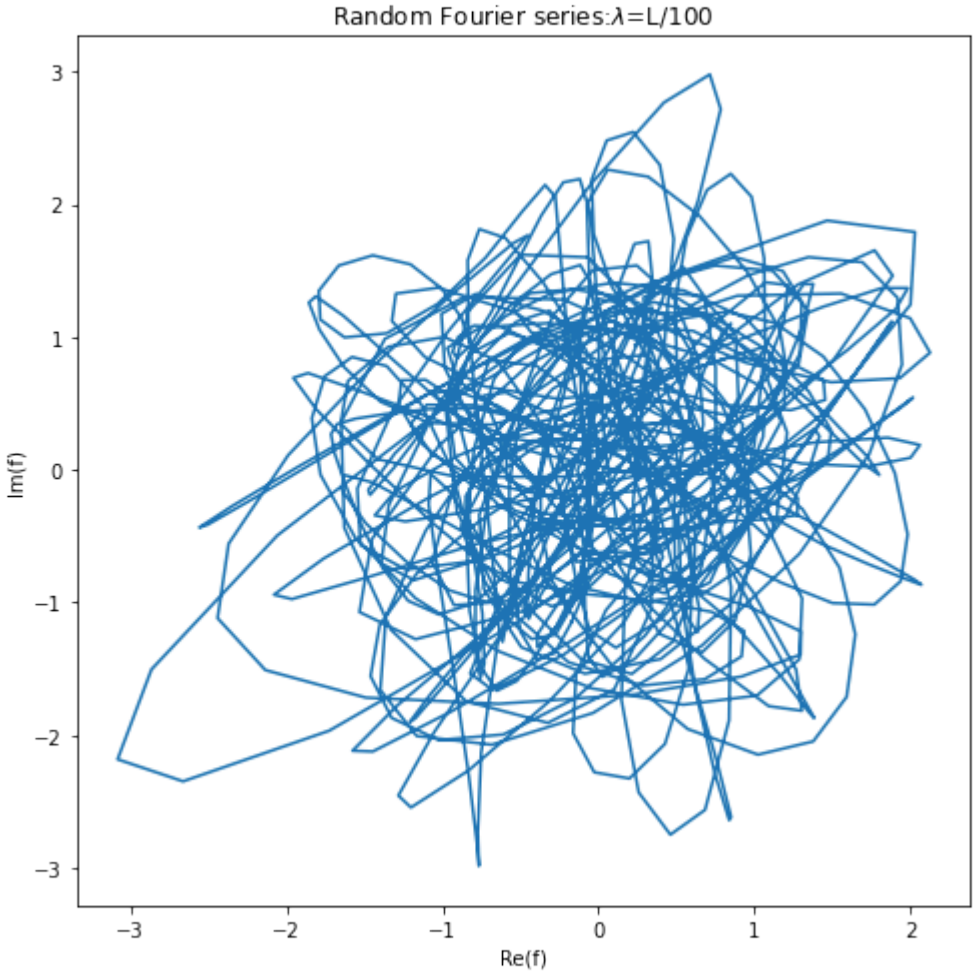


In [167…

```
x=np.linspace(-1,1,1000)
re,im=complexrandfourier(100,2,x)
plt.figure(figsize=(8,8))
plt.title(r'Random Fourier series:$\lambda$=L/100 ')
plt.xlabel('Re(f)')
plt.ylabel('Im(f)')
plt.plot(re,im)
```

Out[167…

```
[<matplotlib.lines.Line2D at 0x1fe229d8850>]
```


Random Fourier series:λ=L/100
Im(f)
Re(f)
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2