Siddhanth Kumar

Section 102

Homework 3: Arrays, File I/O and Plotting

Please complete this homework assignment in code cells in the iPython notebook. Include comments in your code when necessary. Please rename the notebook as SIS ID_HW03.ipynb (your student ID number) and save the notebook once you have executed it as a PDF (note, that when saving as PDF you don't want to use the option with latex because it crashes, but rather the one to save it directly as a PDF).

The homework should be submitted on bCourses under the Assignments tab (both the .ipynb and .pdf files). Please label it by your student ID number (SIS ID)

Problem 1: Sunspots

[Adapted from Newman, Exercise 3.1] At this link (and also in the current directory on datahub) you will find a file called `sunspots.txt`, which contains the observed number of sunspots on the Sun for each month since January 1749. The file contains two columns of numbers, the first being the month and the second being the sunspot number.

a. Write a program that reads in the data and makes a graph of sunspots as a function of time. Adjust the x axis so that the data fills the whole horizontal width of the graph.

b. Modify your code to display two subplots in a single figure: The plot from Part 1 with all the data, and a second subplot with the first 1000 data points on the graph.

c. Write a function `running_average(y, r)` that takes an array or list y and calculates the running average of the data, defined by

$$Y_k = \frac{1}{2r+1} \sum_{m=-r}^{r} y_{k+m},$$

where y_k are the sunspot numbers in our case. Use this function and modify your second subplot (the one with the first 1000 data points) to plot both the original data and the running average on the same graph, again over the range covered by the first 1000 data points. Use $r = 5$, but make sure your program allows the user to easily change r.

The next two parts may require you to google for how to do things. Make a strong effort to do these parts on your own without asking for help. If you do ask for help from a GSI or friend, first ask them to point you to the resource they used, and do your best to learn the necessary techniques from that resource yourself. Finding and learning from online documentation and forums is a very important skill. (Hint: Stack Exchange/Stack Overflow is often a great resource.)

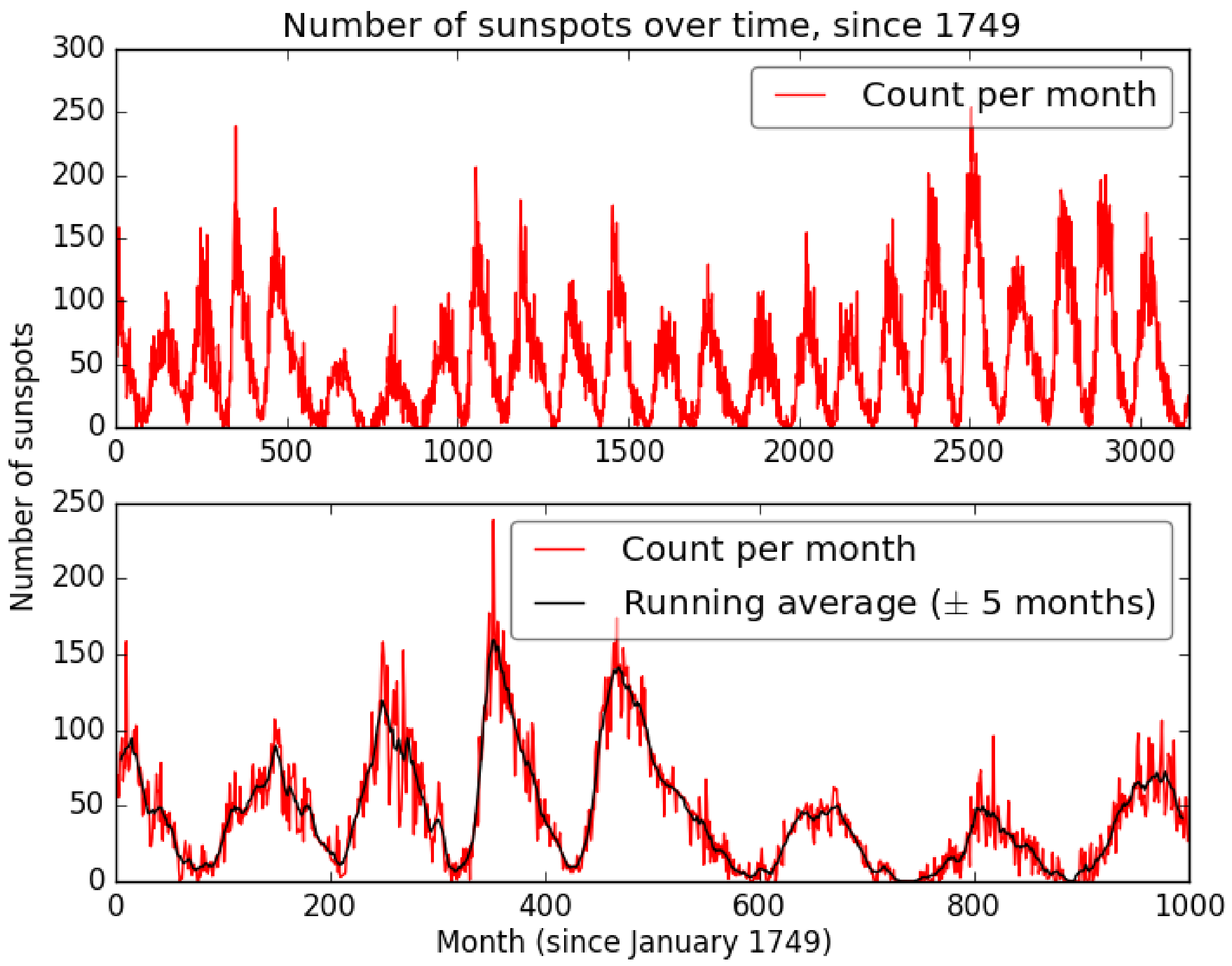
d. Add legends to each of your subplots, but make them partially transparent, so that you can still see any data that they might overlap. *Note: In your program, you should only have to change r for the running average in one place to adjust both the graph and the legend.*

e. Since the x and y axes in both subplots have the same units, add shared x and y labels to your plot that are centered on the horizontal and vertical dimensions of your figure, respectively. Also add a single title to your figure.

When your are finished, your plot should look something close to this:

In [1]:
```
# Don't rerun this snippet of code.
# If you accidentally do, uncomment the lines below and rerun

from IPython.display import Image
Image(filename="img/p1_output.png")
```

Out[1]:

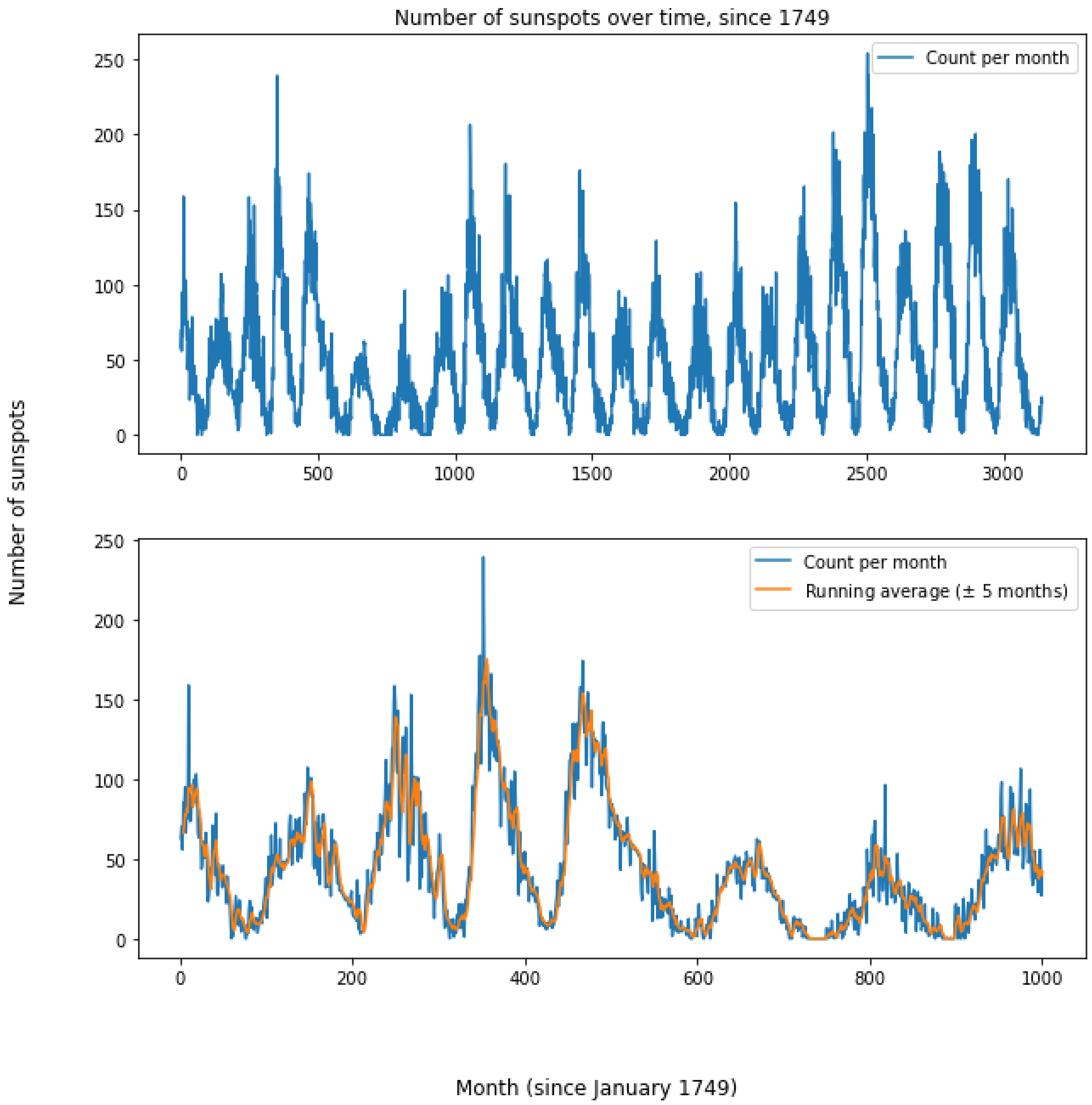


Hints

- The running average is not defined for the first and last few points that you're taking a running average over. (Why is that?) Notice, for instance, that the black curve in the plot above doesn't extend quite as far on either side as the red curve. For making your plot, it might be helpful if your `running_average` function returns an array of the x-values x_k (or their corresponding indices k) along with an array of the y-values Y_k that you compute for the running average.
- You can use the Latex code `$\pm$` for the $\pm$ symbol in the legend. You can also just write `+/-` if you prefer.

In [59]:
```
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline

def running_average(arr, window_size):
    i = 0
    moving_averages = []

    while i < len(arr) - window_size + 1:

        window = arr[i : i + window_size]

        window_average = sum(window) / window_size

        moving_averages.append(window_average)

        i += 1
    return moving_averages

#used reference from w3 schools

f = open('sunspots.txt', 'r')

month = []
sunspot = []

for line in f:
    tokens = line.split()
    month.append(float(tokens[0]))
    sunspot.append(float(tokens[1]))
f.close()

firstMonths = []
firstSunspots = []

for i in range(0, 1001):
    firstMonths.append(month[i])
    firstSunspots.append(sunspot[i])

averages = running_average(firstSunspots, 5)
for i in range(3):
    averages.insert(0, np.nan)

firstMonths.pop(0)
firstSunspots.pop(0)

print(len(averages))
print(len(firstMonths))

fig1, (ax1, ax2) = plt.subplots(2, 1)


ax1.plot(month, sunspot)
ax2.plot(firstMonths, firstSunspots)
ax2.plot(firstMonths, averages)


ax1.legend(['Count per month'])
ax1.set_title("Number of sunspots over time, since 1749")

ax2.legend(['Count per month', 'Running average ($\pm$ 5 months)'])

fig1.set_figheight(10)
fig1.set_figwidth(10)

fig1.supxlabel('Month (since January 1749)')
fig1.supylabel('Number of sunspots')


plt.show()
```

```
1000
1000
```



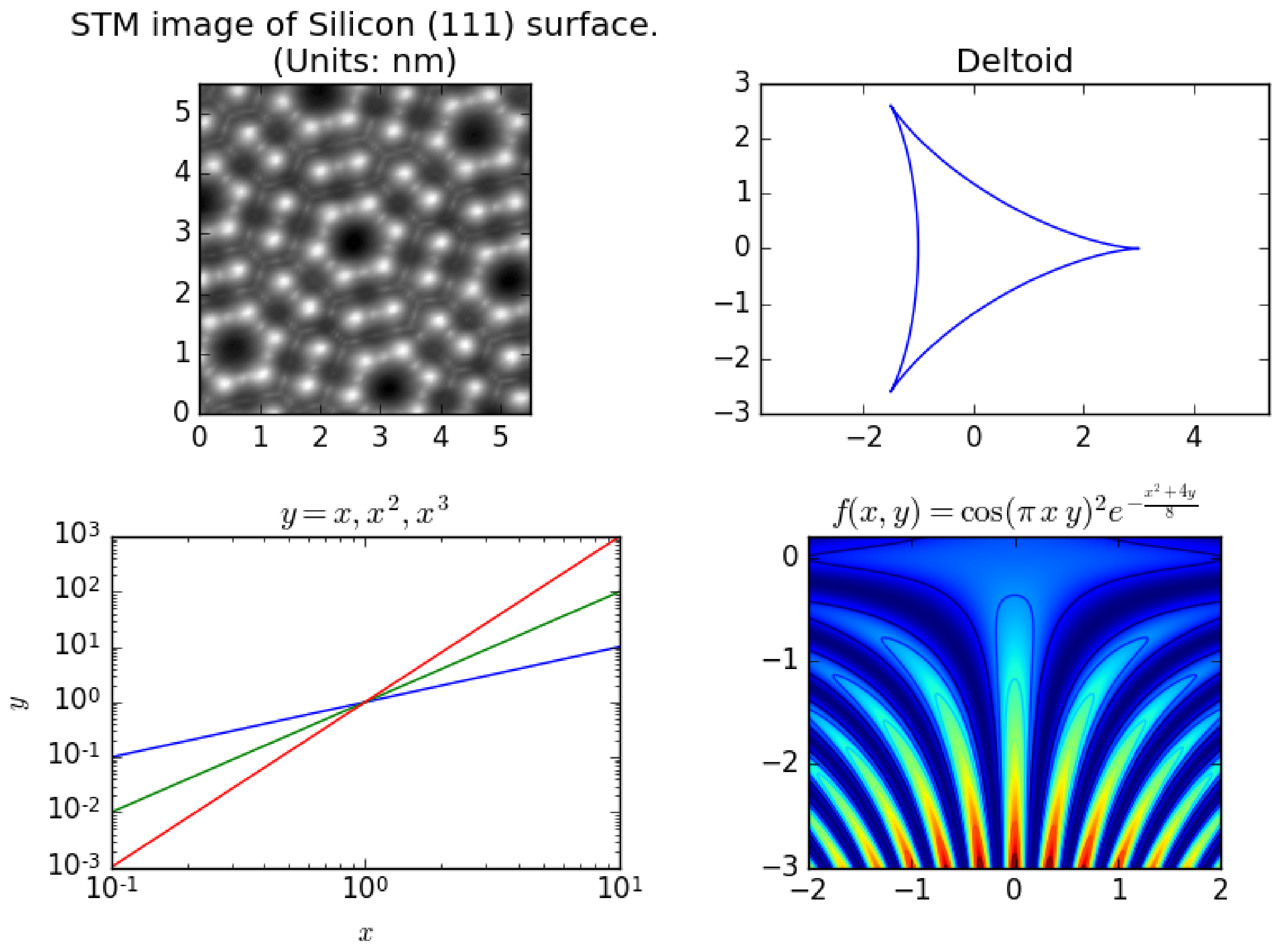
Problem 2: Variety Plot

In this problem, you will reproduce the following as a single figure with four subplots, as best you can:

In [2]:

```
# Don't rerun this snippet of code.
# If you accidentally do, uncomment the lines below and rerun

from IPython.display import Image
Image(filename="img/p2_output.png")
```

Out[2]:



Here are some hints and directions for each one:

Upper-left: This is an image of silicon taken with an electron microscope.

You can find the data file `stm.txt` here and in your datahub directory, among resources for the Newman text.

You may assume that the upper-left of the array is indeed the upper-left of the image.

Both axes should run from 0 to 5.5.

This subplot uses the `gray` colormap.

Upper-Right: Matplotlib can plot any list of (x, y) points you give it, including parametric or polar curves. The curve in this subplot is called a "deltoid", and is the result of the equations

$$\begin{aligned} x &= 2\cos\theta + \cos 2\theta \\ y &= 2\sin\theta - \sin 2\theta \end{aligned}$$

over a range of θ from 0 to 2π.

To get the aspect ratio equal with nice spacing around the curve, try one of the following, depending on how you are making your subplots:

- if you're using `plt.subplot(...)` to get each subplot (the "state-machine" approach), add the `aspect='equal'` and `adjustable='datalim'` arguments to the deltoid subplot, so your

command will look something like `plt.subplot(..., aspect='equal', adjustable='datalim')`.
- if you're using `... = plt.subplots(...)` (note the 's'!) or `ax = fig.add_subplot(...)` on a figure `fig` to get subplots with axes objects (the "object-oriented" approach), add the line `ax.set_aspect(aspect='equal', adjustable='datalim')`, where `ax` is the axes object you want to affect.

Lower-Left: This kind of plot is called a log-log plot, where both axes are on a logarithmic scale. Google or look in the matplotlib gallery to learn how to make this kind of plot.

The three curves are $y = x$, $y = x^2$, and $y = x^3$, where x ranges over 10^{-1} to 10^1. (Note: You can write powers of ten in python using the shorthand `1e-1` for 10^{-1}, `1e1` for 10^1, and so on.)

To make the pretty mathematical labels you see in the sample figure above, you can use

- `r'$y = x, x^2, x^3$'` for the title
- `r'$x$'` for the x-axis, and
- `r'$y$'` for the y-axis.

Just put these bits of code as you see them (with the `r` outside the quotes!) where you would normally put a string for the title or axes labels.

Lower-Right: Here you see a density plot with contours of the function

$$f(x, y) = \cos^2(\pi\, x\, y) e^{-\frac{x^2+4y}{8}},$$

over x from -2 to 2 and y from -3 to 0.2.

Use `meshgrid` to generate the x and y values. Be careful to make sure that the point $(-2, -3)$ is in the bottom left corner of the plot.

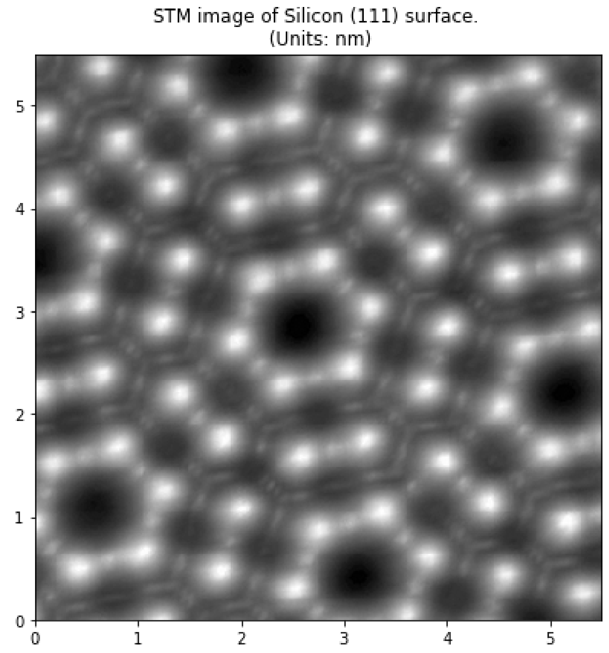
You'll need to use both `imshow` and `contour` to generate the density plot and then overlay it with contours. This plot uses the default contour spacing, so you don't need to worry about adjusting that. The colormap is `jet`, matplotlib's current default. (The default colormap will be changing to `viridis` in the next version.)

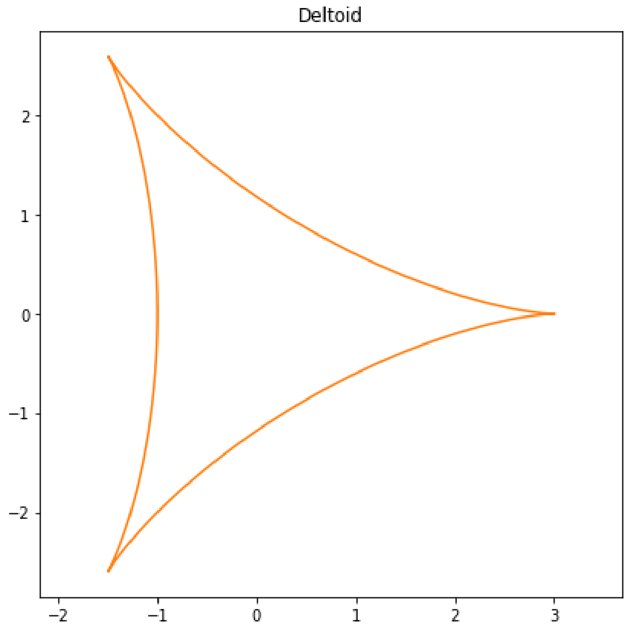
To get the ticks spaced out like you see here, use matplotlib's `xticks` or `set_xticks` functions for the x-axis (depending on how you're making your plots), and similar functions for the y-axis. You can pass each of these a single argument: a simple list or array of the numbers you want ticked on each axis.

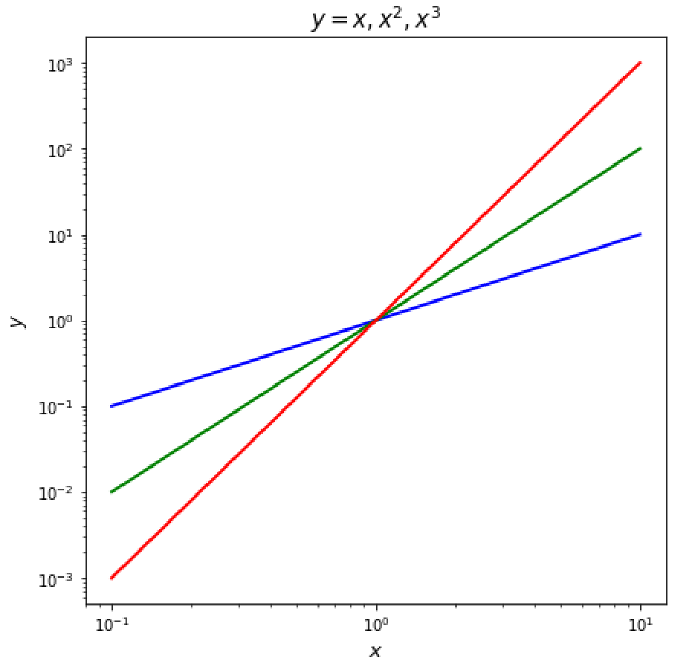
Spacing the subplots: Once all is said and done and you run `plt.show()`, you may notice your plots are cramped and overlapping each other. Add the line `plt.tight_layout()` before `plt.show()`, and matplotlib will space things out in an attempt to avoid overlapping subplots.

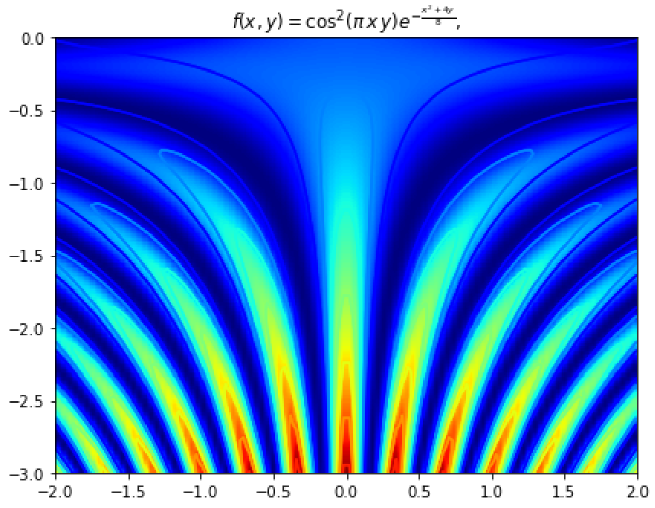
In [62]:
```
import numpy as np

fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(15, 15))

#ax1.plot(x, y)

array2D = np.loadtxt('stm.txt')
ax1.set_title('STM image of Silicon (111) surface. \n (Units: nm)')
plt.set_cmap('gray')
ax1.imshow(array2D, interpolation='none', extent=[0, 5.5, 0, 5.5])
```

```
#----------------------------------
x = []
y = []

ivals = np.arange(0, (2 * np.pi), 0.01)

for i in ivals:
    xval = (2*np.cos(i)) + np.cos(2*i)
    yval = (2*np.sin(i)) - np.sin(2*i)
    x.append(xval)
    y.append(yval)

ax2.set_title('Deltoid')
ax2.set_aspect(aspect='equal', adjustable='datalim')
ax2.plot(x, y, 'tab:orange')
#-----------------------------------
x1 = np.linspace(1e-1, 1e1, 2)
y1 = np.linspace(1e-1, 1e1, 2)

x2 = np.linspace(1e-1, 1e1, 2)
y2 = np.linspace(1e-2, 1e2, 2)

x3 = np.linspace(1e-1, 1e1, 2)
y3 = np.linspace(1e-3, 1e3, 2)

#plt.figure(figsize=[7, 5])


ax3.loglog(x1, y1, linewidth=2, color='blue')
ax3.loglog(x2, y2, linewidth=2, color='green')
ax3.loglog(x3, y3, linewidth=2, color='red')
ax3.set_title(r'$y = x, x^2, x^3$', fontsize=15)
ax3.set_xlabel(r'$x$', fontsize=13)
ax3.set_ylabel(r'$y$' , fontsize=13)
#-----------------------------------
xx = np.linspace(-2, 2, 100)
yy = np.linspace(-3, 0.2, 100)

X, Y = np.meshgrid(xx, yy)

Z = (np.cos(np.pi * X * Y)**2) * (np.e ** (-((X*X) + (4*Y))/8))

ax4.set_title(r'$f(x,y) = \cos^2(\pi\,x\,y ) e^{-\frac{x^2 + 4 y}{8}},$')

plt.jet()
con = ax4.contour(X, Y, Z)
ax4.set_aspect('equal')

plt.contour(X,Y,Z)
plt.imshow(Z, origin='lower', extent=[-2, 2, -3, 0])

plt.show()
```


STM image of Silicon (111) surface.
(Units: nm)
0
1
2
3
4
5


Deltoid
2
1
0
−1
−2
−2
−1
0
1
2
3


$y=x, x^2, x^3$
10^3
10^2
10^1
10^0
10^{-1}
10^{-2}
10^{-3}
y
10^{-1}
10^0
10^1
x


$f(x,y)=\cos^2(\pi x y)e^{-\frac{x^2+4y}{8}},$
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
−3.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

In [ ]: