Python for Science and Engg: Interactive Plotting

FOSSEE

Department of Aerospace Engineering
IIT Bombay

25 September, 2010
Day 1, Session 1

Workshop Schedule: Day 1

Session 1	Sat 11:00–12:00
Session 2	Sat 12:00–13:00
Session 3	Sat 14:00–15:00
Session 4	Sat 15:10–16:10
Session 5	Thu 16:20–17:20

Workshop Schedule: Day 2

Session 1	Sun 09:00–10:00
Session 2	Sun 10:00–11:00
Session 3	Sun 11:10–12:10
Session 4	Sun 12:10–13:10
Exercises	Sun 14:25–15:10
Session 5	Sun 15:20–16:20

Outline

Checklist

1. IPython
2. Editor - we recommend scite
3. Data files:
   - `sslc1.txt`
   - `pendulum.txt`
   - `pos.txt`
   - `holmes.txt`
   - `anag.txt`
4. Python scripts:
   - `sslc_allreg.py`
   - `sslc_science.py`
5. Images
   - `lena.png`
   - `smoothing.gif`

About the Workshop

Intended Audience

- Engg., Mathematics and Science teachers.
- Interested students from similar streams.

Goal: Successful participants will be able to

- Use Python as plotting, computational tool.
- Understand how to use Python as a scripting and problem solving language.
- Train students for the same.

Outline

Starting up . . .

```
$ ipython -pylab
```

```
In []: print "Hello, World!"
Hello, World!
```

Exiting

```
In []: ^D(Ctrl-D)
Do you really want to exit([y]/n)? y
```

An alternative to IPython is bpython

Outline

Loops

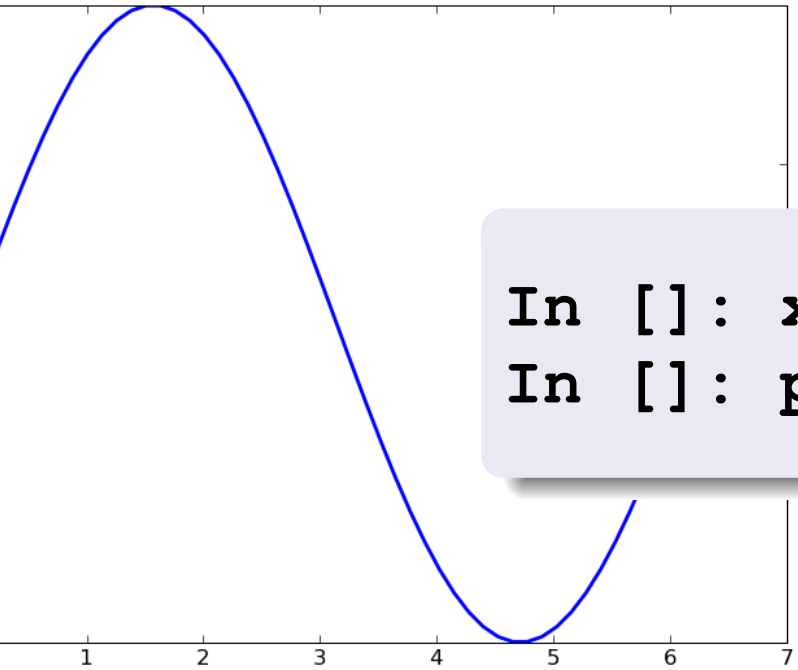
Breaking out of loops

```
In []: while True:
   ...:     print "Hello, World!"
   ...:
Hello, World!
Hello, World!^C(Ctrl-C)
---------------------------------------------------------------------------
KeyboardInterrupt
```

Outline

Outline

First Plot




```
In []: x = linspace(0, 2*pi, 50)
In []: plot(x, sin(x))
```

Walkthrough

`x = linspace(start, stop, num)`

returns **`num`** evenly spaced points, in the interval [**`start`**, **`stop`**].

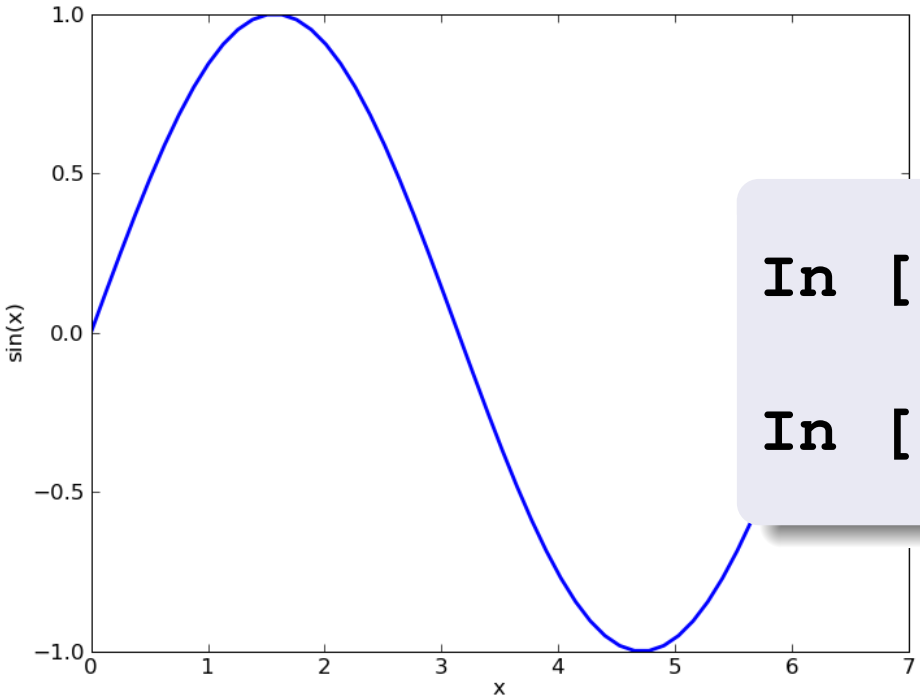
```
x[0] = start
x[num - 1] = end
```

`plot(x, y)`

plots **`x`** and **`y`** using default line style and color

Outline

Adding Labels




```
In [ ]: xlabel('x')

In [ ]: ylabel('sin(x)')
```

Another example

```
In []: clf()
```

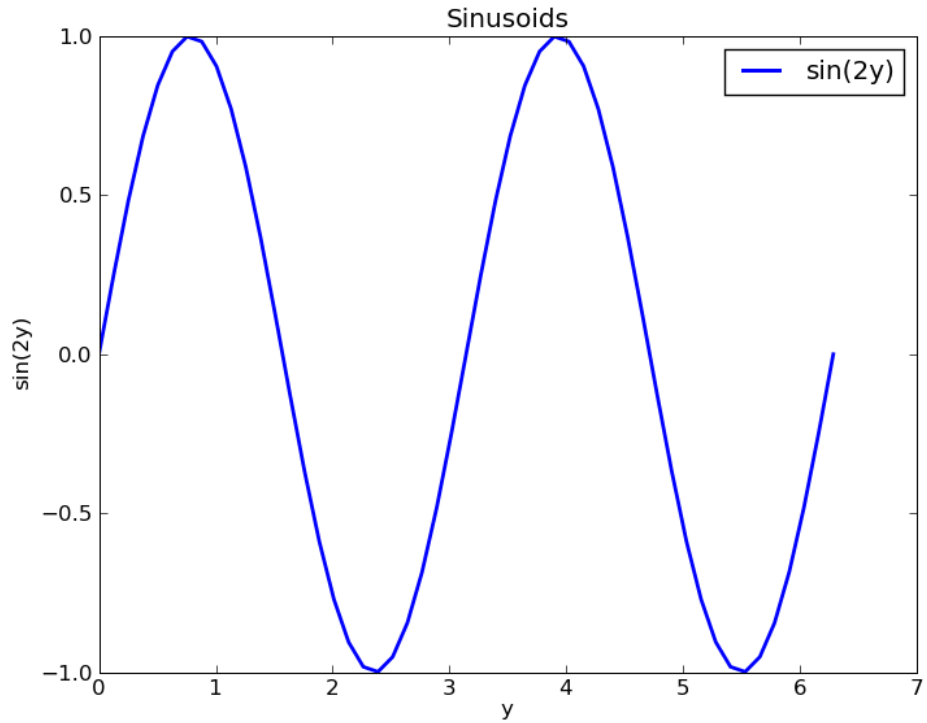
Clears the plot area.

```
In []: y = linspace(0, 2*pi, 50)
In []: plot(y, sin(2*y))
In []: xlabel('y')
In []: ylabel('sin(2y)')
```

Outline

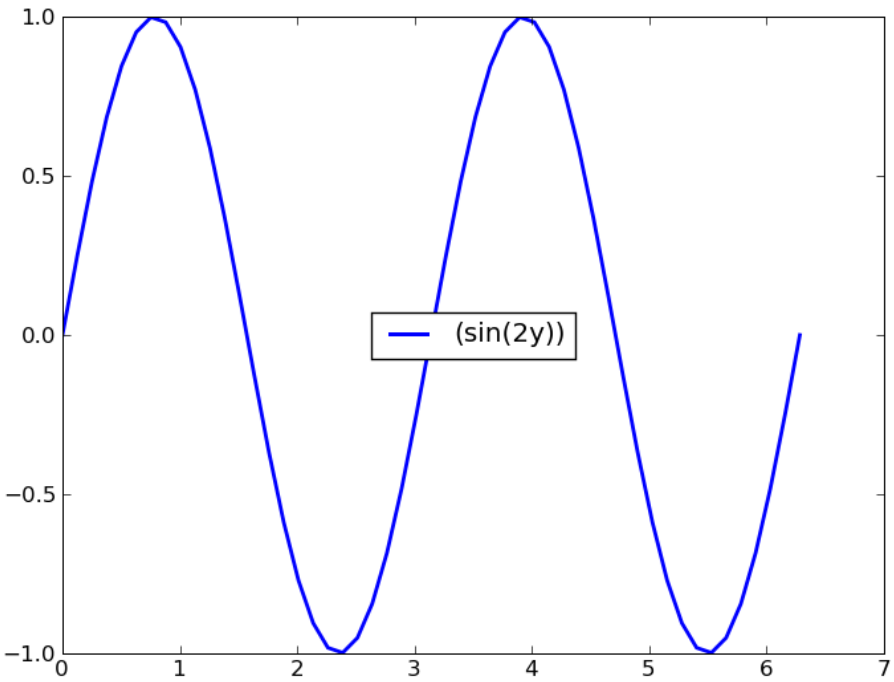
Title and Legends

```
In []: title('Sinusoids')
In []: legend(['sin(2y)'])
```



Legend Placement

```
In []: legend(['sin(2y)'], loc = 'center')
```




```
'best'
'right'
'center'
```

Saving & Closing

```
In []: savefig('sin.png')

In []: close()
```

Supported formats to store images:

- png
- eps - Easy to embed in LaTeX files
- emf
- pdf
- ps
- raw
- rgba
- svg

Outline

Overlaid Plots

```
In []: clf()
In []: plot(y, sin(y))
In []: plot(y, cos(y))
In []: xlabel('y')
In []: ylabel('f(y)')
In []: legend(['sin(y)', 'cos(y)'])
```

By default plots would be overlaid!

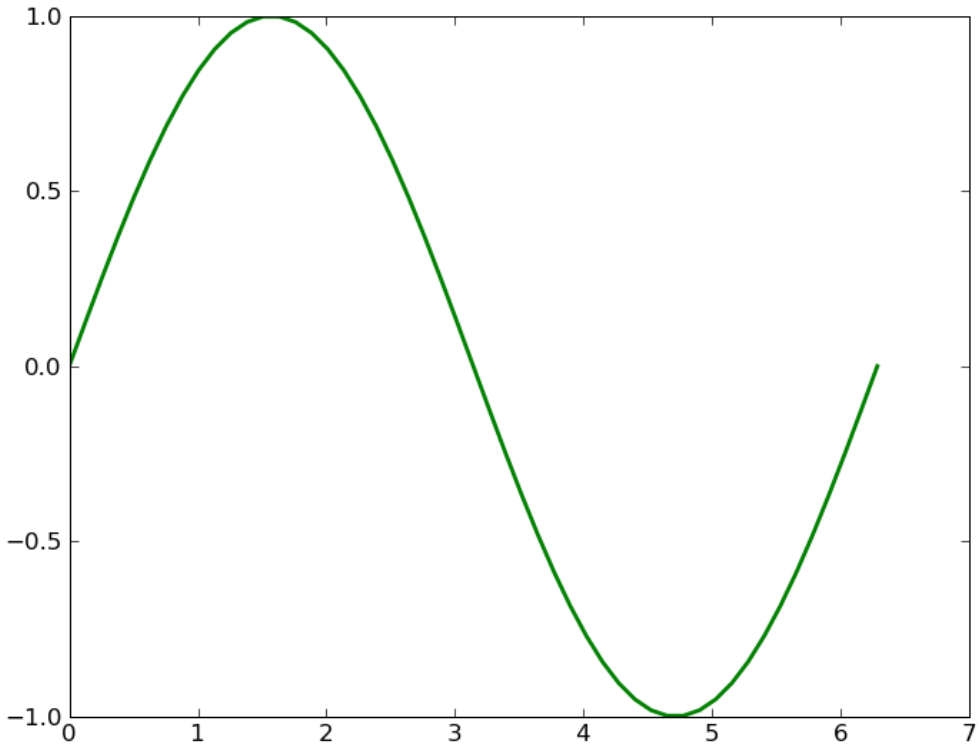
Plotting separate figures

```
In []: clf()
In []: figure(1)
In []: plot(y, sin(y))
In []: figure(2)
In []: plot(y, cos(y))
In []: savefig('cosine.png')
In []: figure(1)
In []: title('sin(y)')
In []: savefig('sine.png')
In []: close()
In []: close()
```

Showing it better

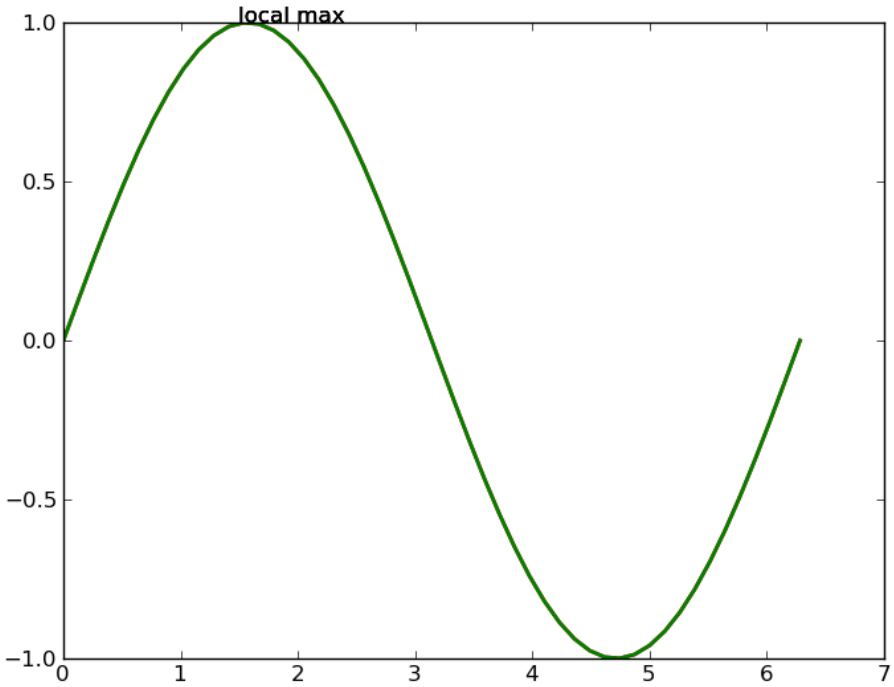
```
In []: plot(y, cos(y), 'r')

In []: clf()
In []: plot(y, sin(y), 'g', linewidth=2)
```



Annotating

```
In []: annotate('local max', xy=(1.5, 1))
```



Axes lengths

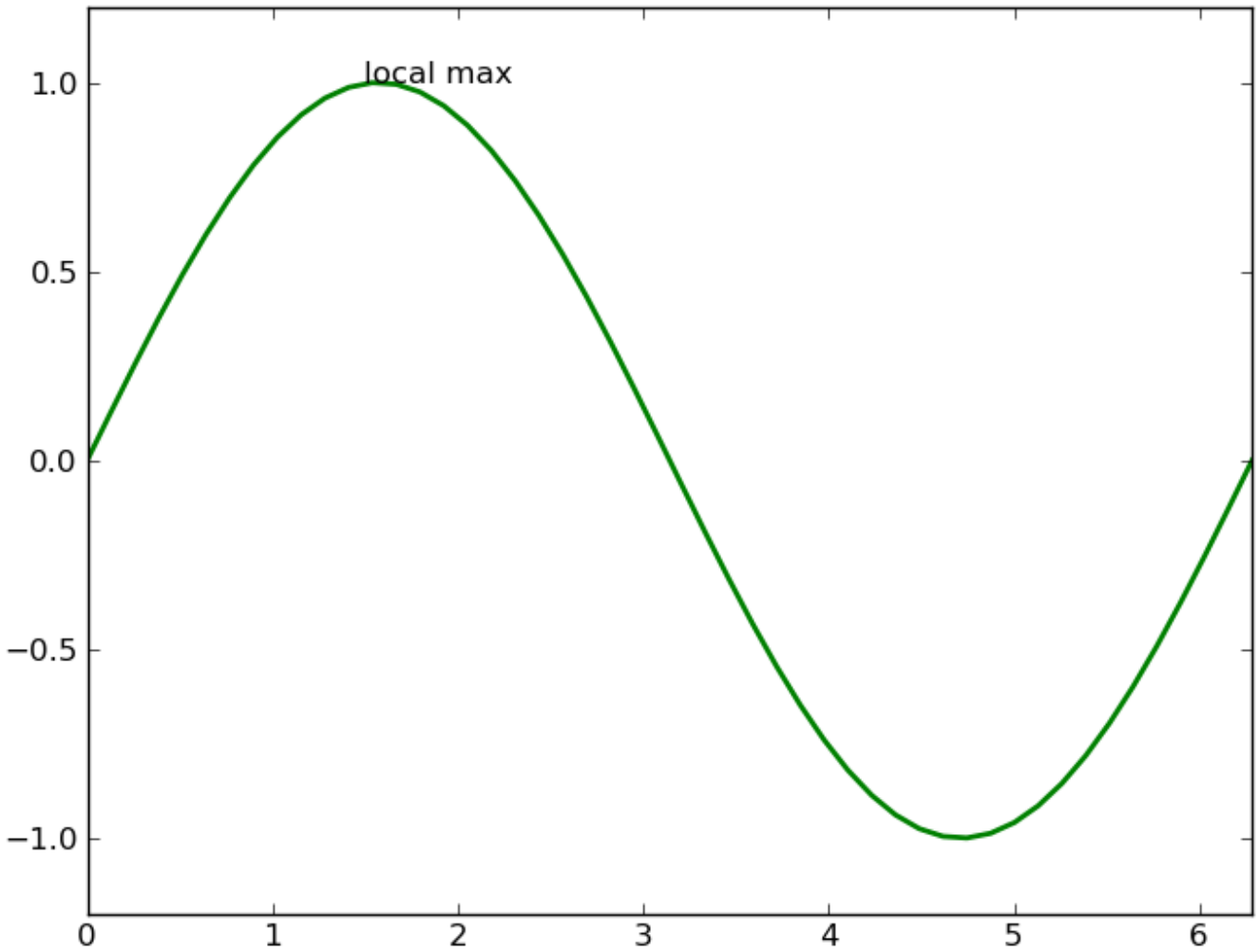
Getting axes lengths

```
In []: xmin, xmax = xlim()
In []: ymin, ymax = ylim()
```

Set the axes limits

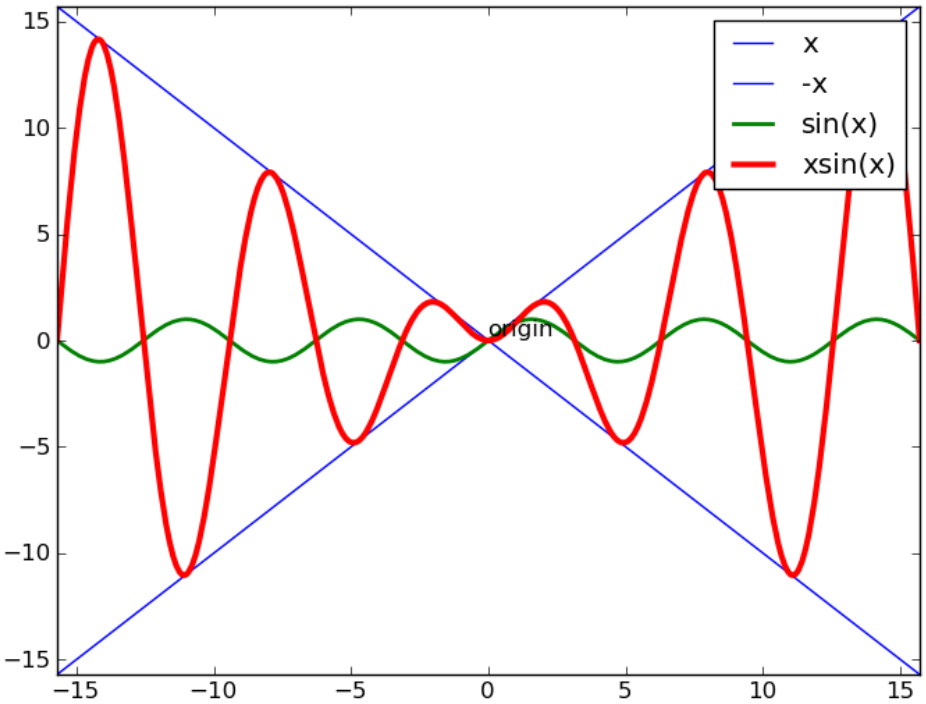
```
In []: xlim(xmin, 2*pi )
In []: ylim(ymin-0.2, ymax+0.2)
```

Axes lengths



Review Problem

1. Plot x, -x, sin(x), xsin(x) in range -5π to 5π
2. Add a legend
3. Annotate the origin
4. Set axes limits to the range of x



Review Problem . . .

Plotting . . .

```
In []: x=linspace(-5*pi, 5*pi, 500)
In []: plot(x, x, 'b')
In []: plot(x, -x, 'b')
In []: plot(x, sin(x), 'g', linewidth=2)
In []: plot(x, x*sin(x), 'r',
            linewidth=3)
```

⋮

Review Problem . . .

Legend & Annotation. . .

```
In [ ]: legend(['x', '-x', 'sin(x)',
                'xsin(x)'])
In [ ]: annotate('origin', xy = (0, 0))
```

Setting Axes limits. . .

```
In [ ]: xlim(-5*pi, 5*pi)
In [ ]: ylim(-5*pi, 5*pi)
```

Command History

Use the **`%hist`** magic command of IPython

```
In []: %hist
```

This displays all the commands typed in so far aka Command History.

Careful about errors!

`%hist` will contain the errors as well.

Magic Commands?

Magic commands are commands provided by IPython to make our life easier.

Saving commands into script

Use the **%save** magic command of IPython

```
%save script_name line_numbers
```

Line numbers can be specified individually separated by commas or as a range separated by a dash.

```
%save four_plot.py 16 18-27
```

This saves from the history the commands entered on line numbers 16, 18, 19, 20, . . . 27

Python Scripts. . .

Now, four_plot.py is called a Python Script.

- run the script in IPython using
  `%run four_plot.py`

```
NameError: name 'linspace' is not defined
```

To avoid this, run using `%run -i four_plot.py`

Python Scripts. . .

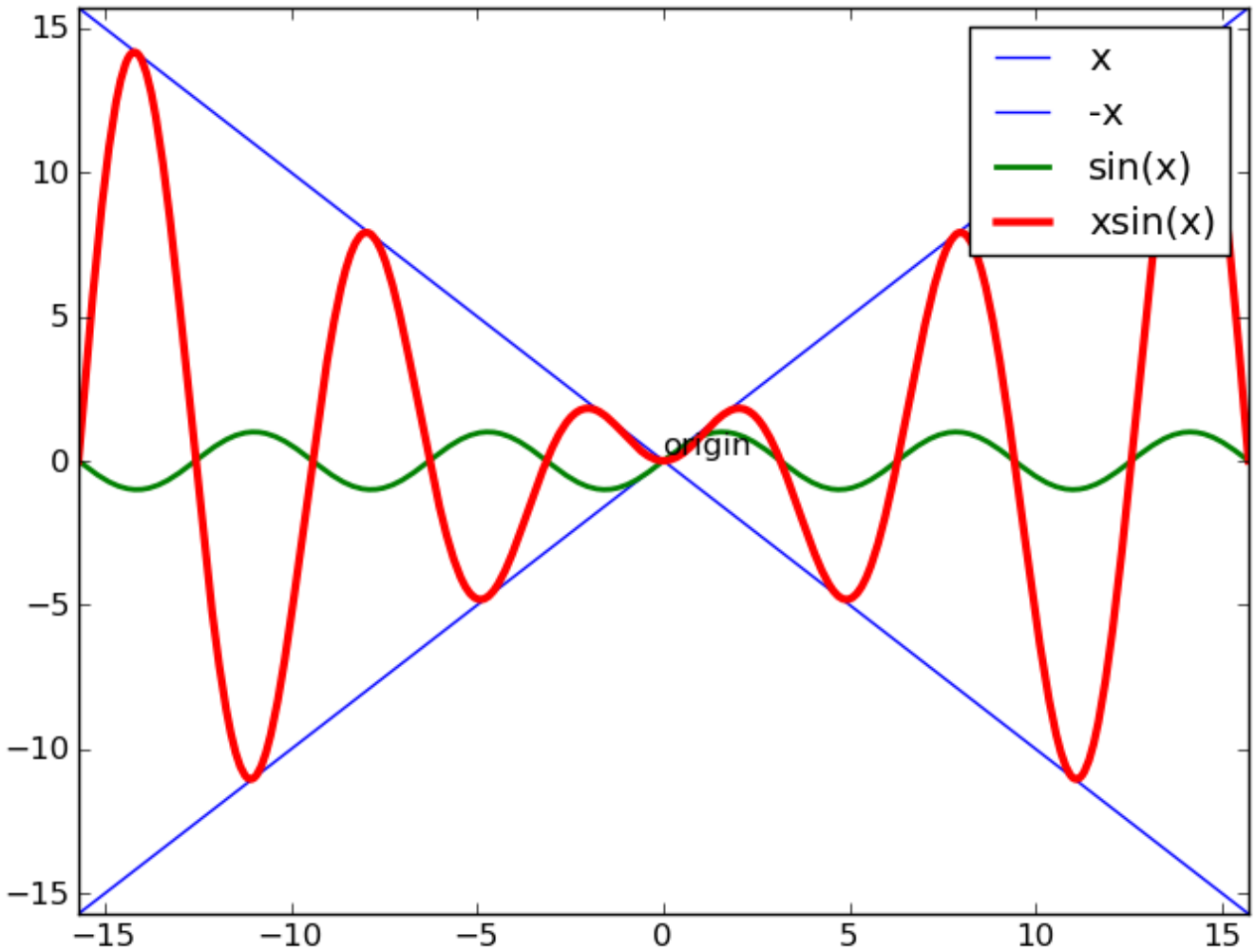
Now, four_plot.py is called a Python Script.

- run the script in IPython using
  `%run four_plot.py`

```
NameError: name 'linspace' is not defined
```

To avoid this, run using `%run -i four_plot.py`

Result graph



What did we learn?

- Starting up IPython
- Creating simple plots.
- Adding labels and legends.
- Annotating plots.
- Changing the looks: size, linewidth
- `%hist` - History of commands
- `%save` - Saving commands
- Running a script using `%run -i`