Python Programming

Chapter 7 :

Advanced Topic : Plotting using PyLab

Introduction

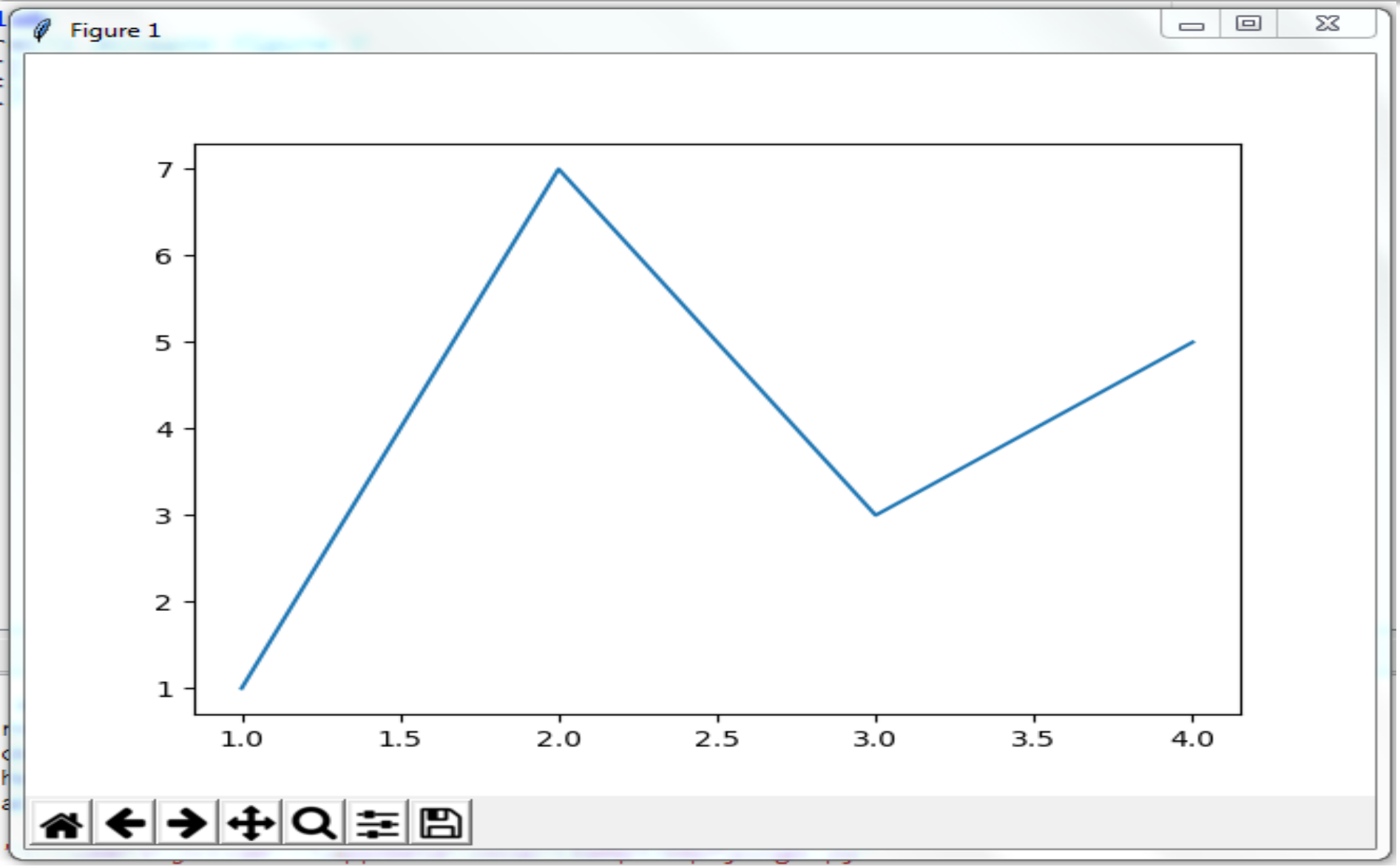
- PyLab is a Python standard library module that provides many of the facilities of MATLAB, “a high-level technical computing language and interactive environment”
  - for algorithm development, data visualization, data analysis, and numeric computation.
- A complete user’s guide for PyLab is at the Web site matplotlib.sourceforge.net/users/index.html.

Example

- Let's start with a simple example that uses pylab.plot to produce two plots.
- Executing

```
import pylab

pylab.figure(1) #create figure 1
pylab.plot([1,2,3,4], [1,7,3,5]) #draw on figure 1
pylab.show() #show figure on screen
```

Output



Example

```
pylab.figure(1) #create figure 1
pylab.plot([1,2,3,4], [1,2,3,4]) #draw on figure 1
pylab.figure(2) #create figure 2
pylab.plot([1,4,2,3], [5,6,7,8]) #draw on figure 2
pylab.savefig('Figure-Addie') #save figure 2
pylab.figure(1) #go back to working on figure 1
pylab.plot([5,6,10,3]) #draw again on figure 1
pylab.savefig('Figure-Jane') #save figure 1
```

Figure 1

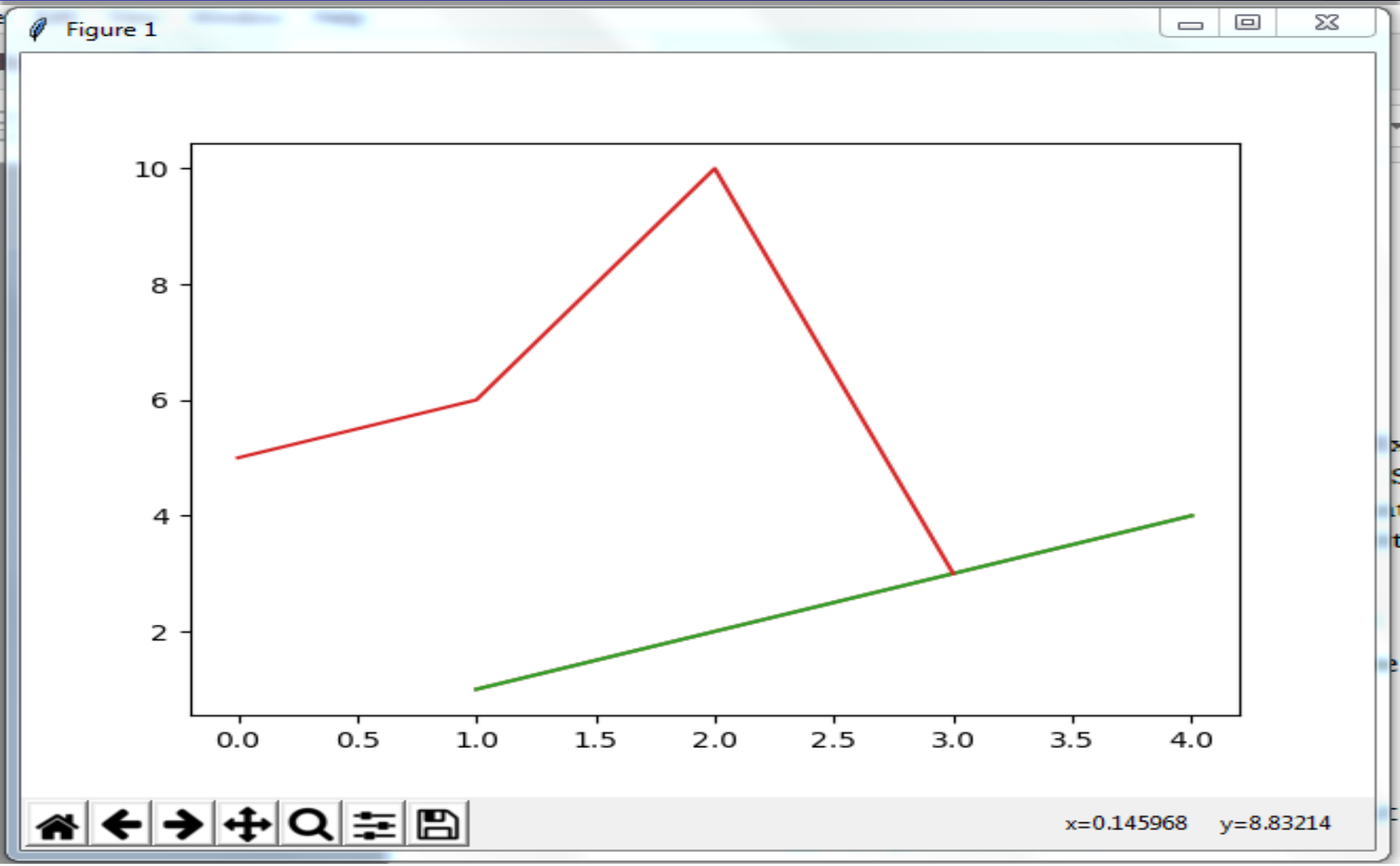
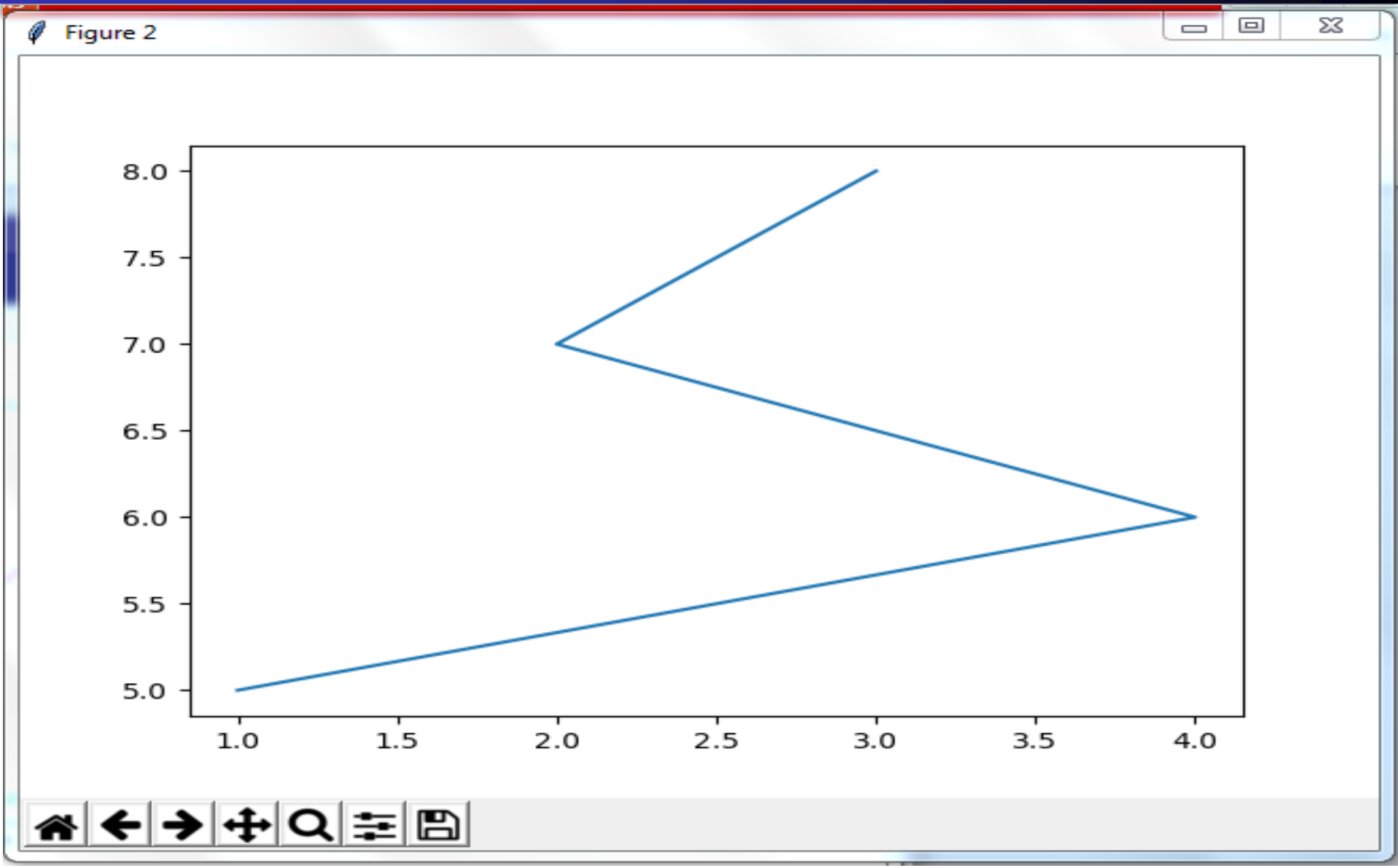


Figure 2



Example

```
principal = 10000 #initial investment
interestRate = 0.05
years = 20
values = []
for i in range(years + 1):
    values.append(principal)
    principal += principal*interestRate
pylab.plot(values)
```

Example

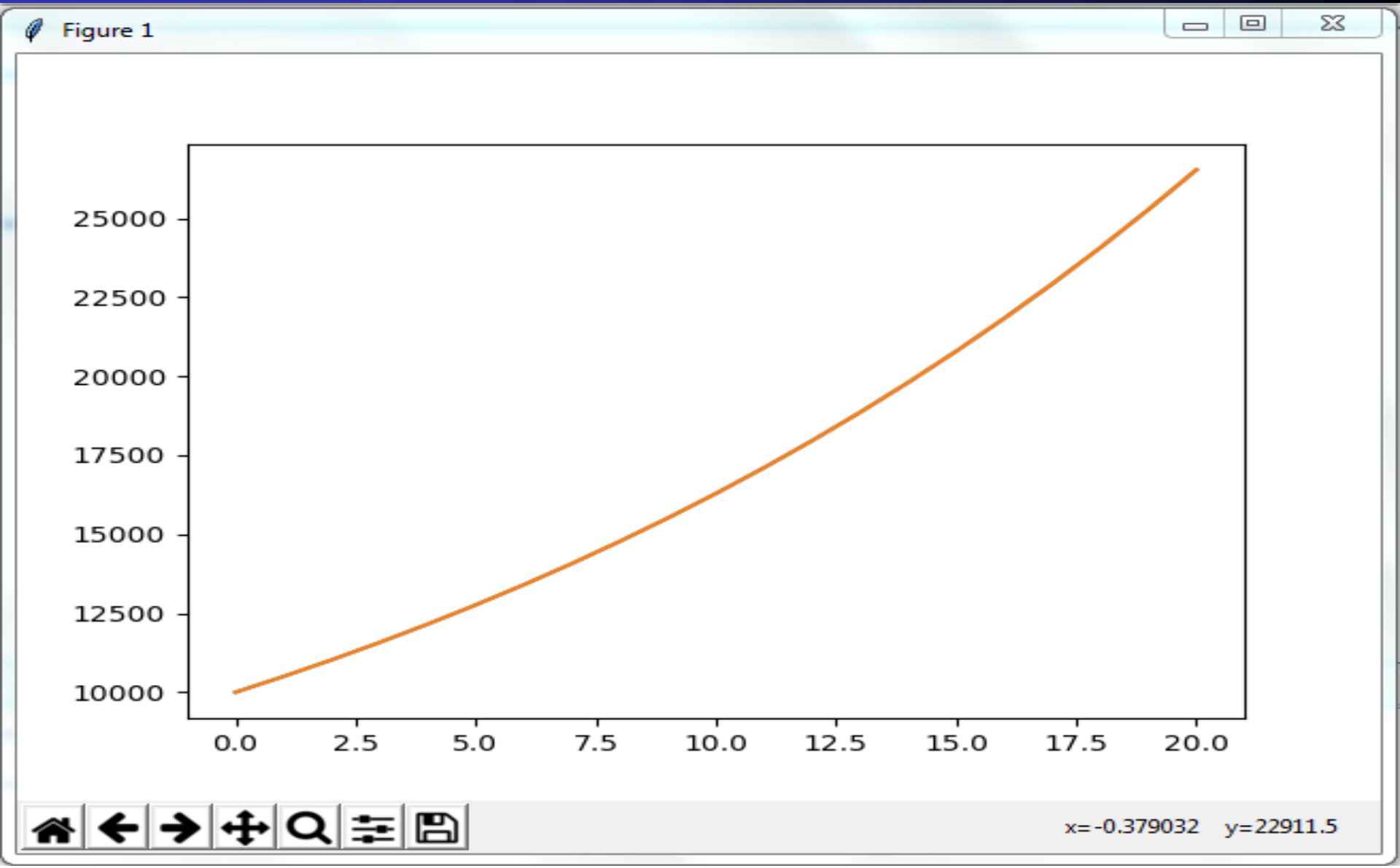
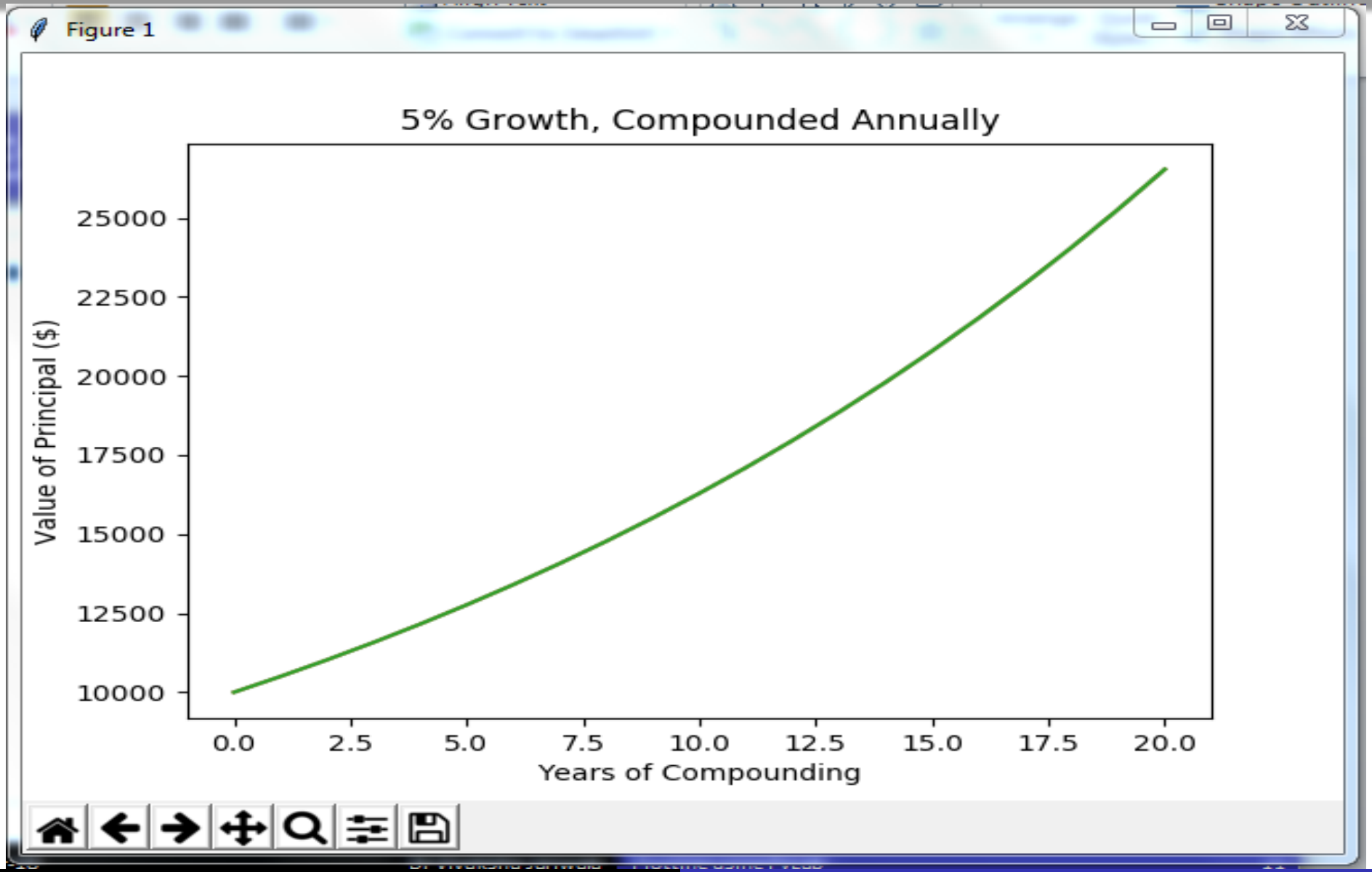


Figure 1
25000
22500
20000
17500
15000
12500
10000
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
x=-0.379032 y=22911.5

Example : Interest

```
import pylab
principal = 10000 #initial investment
interestRate = 0.05
years = 20
values = []
for i in range(years + 1):
    values.append(principal)
    principal += principal*interestRate
pylab.title('5% Growth, Compounded Annually')
pylab.xlabel('Years of Compounding')
pylab.ylabel('Value of Principal ($)')
pylab.plot(values)
pylab.show()
```

Example : Interest …



PyLab

```
import pylab
principal = 10000 #initial investment
interestRate = 0.05
years = 20
values = []
for i in range(years + 1):
    values.append(principal)
    principal += principal*interestRate
pylab.title('5% Growth, Compounded Annually')
pylab.xlabel('Years of Compounding')
pylab.ylabel('Value of Principal ($)')
pylab.plot(values, 'ro')
pylab.show()
```

PyLab

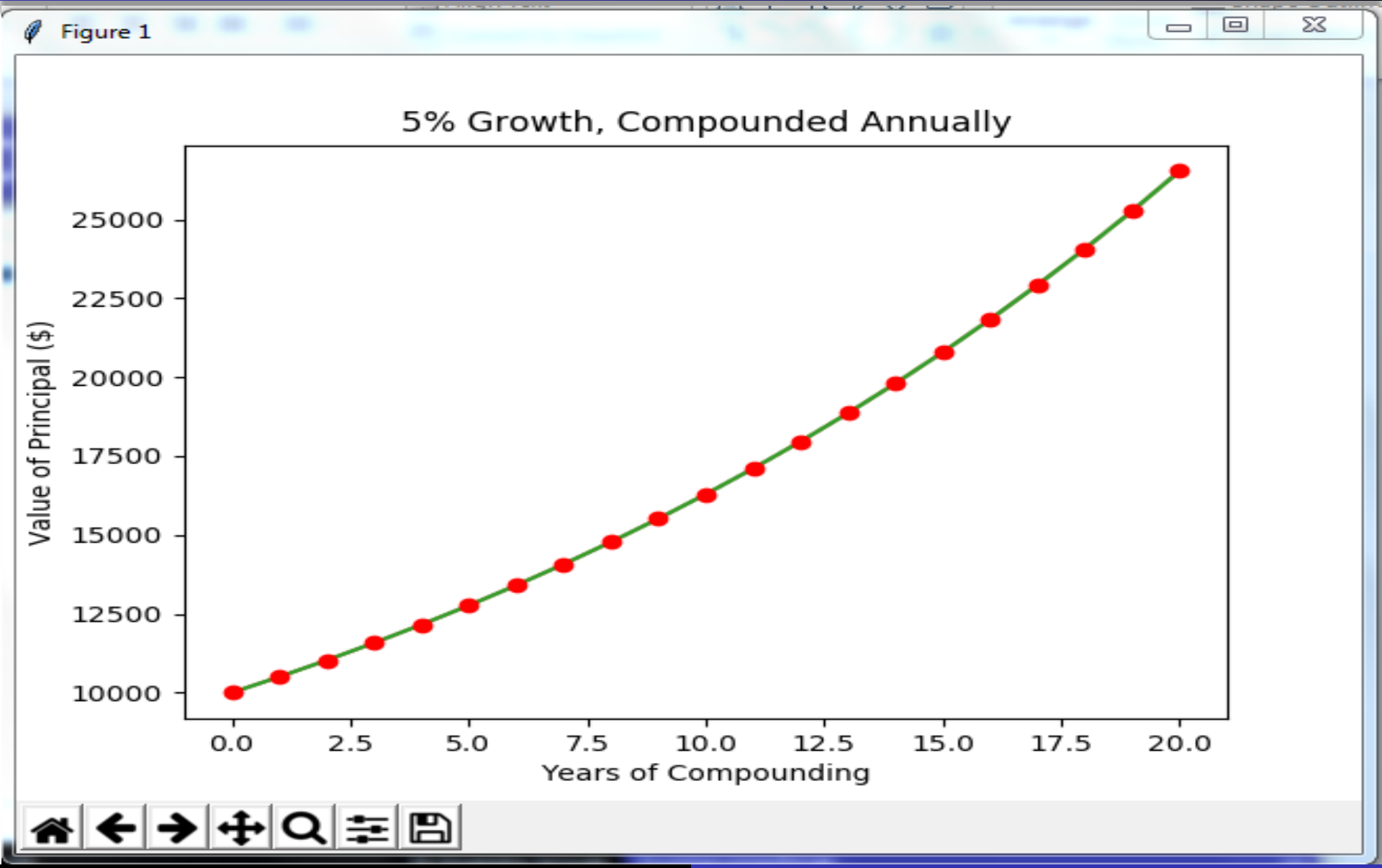


Figure 1
5% Growth, Compounded Annually
Value of Principal ($)
25000
22500
20000
17500
15000
12500
10000
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Years of Compounding

PyLab

```
import pylab
principal = 10000 #initial investment
interestRate = 0.05
years = 20
values = []
for i in range(years + 1):
    values.append(principal)
    principal += principal*interestRate
pylab.plot(values, linewidth = 30)
pylab.title('5% Growth, Compounded Annually', fontsize = 'xx-large')
pylab.xlabel('Years of Compounding', fontsize = 'x-small')
pylab.ylabel('Value of Principal ($)')
pylab.show()
```

PyLab

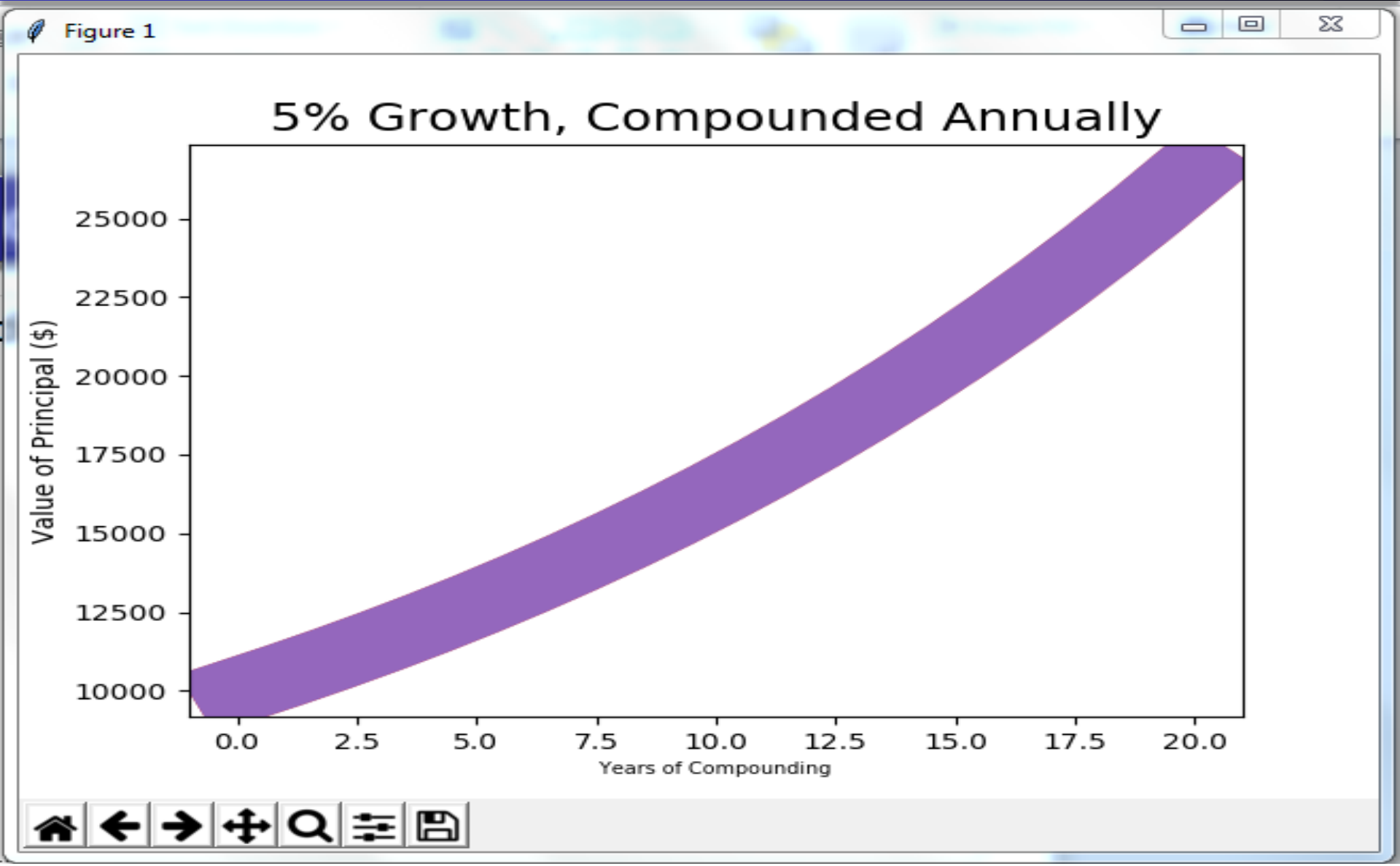


Figure 1
5% Growth, Compounded Annually
Value of Principal ($)
25000
22500
20000
17500
15000
12500
10000
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Years of Compounding

PyLab

```
#set line width
pylab.rcParams['lines.linewidth'] = 4
#set font size for titles
pylab.rcParams['axes.titlesize'] = 20
#set font size for labels on axes
pylab.rcParams['axes.labelsize'] = 20
#set size of numbers on x-axis
pylab.rcParams['xtick.labelsize'] = 16
#set size of numbers on y-axis
pylab.rcParams['ytick.labelsize'] = 16
#set size of ticks on x-axis
pylab.rcParams['xtick.major.size'] = 7
#set size of ticks on y-axis
pylab.rcParams['ytick.major.size'] = 7
#set size of markers
pylab.rcParams['lines.markersize'] = 10
```

numpy

```
import pylab
import numpy as np
x = np.linspace(0, 20, 1000)  # 100 evenly-spaced values from 0 to 50
y = np.sin(x)

pylab.plot(x, y)
pylab.show()
```

numpy…

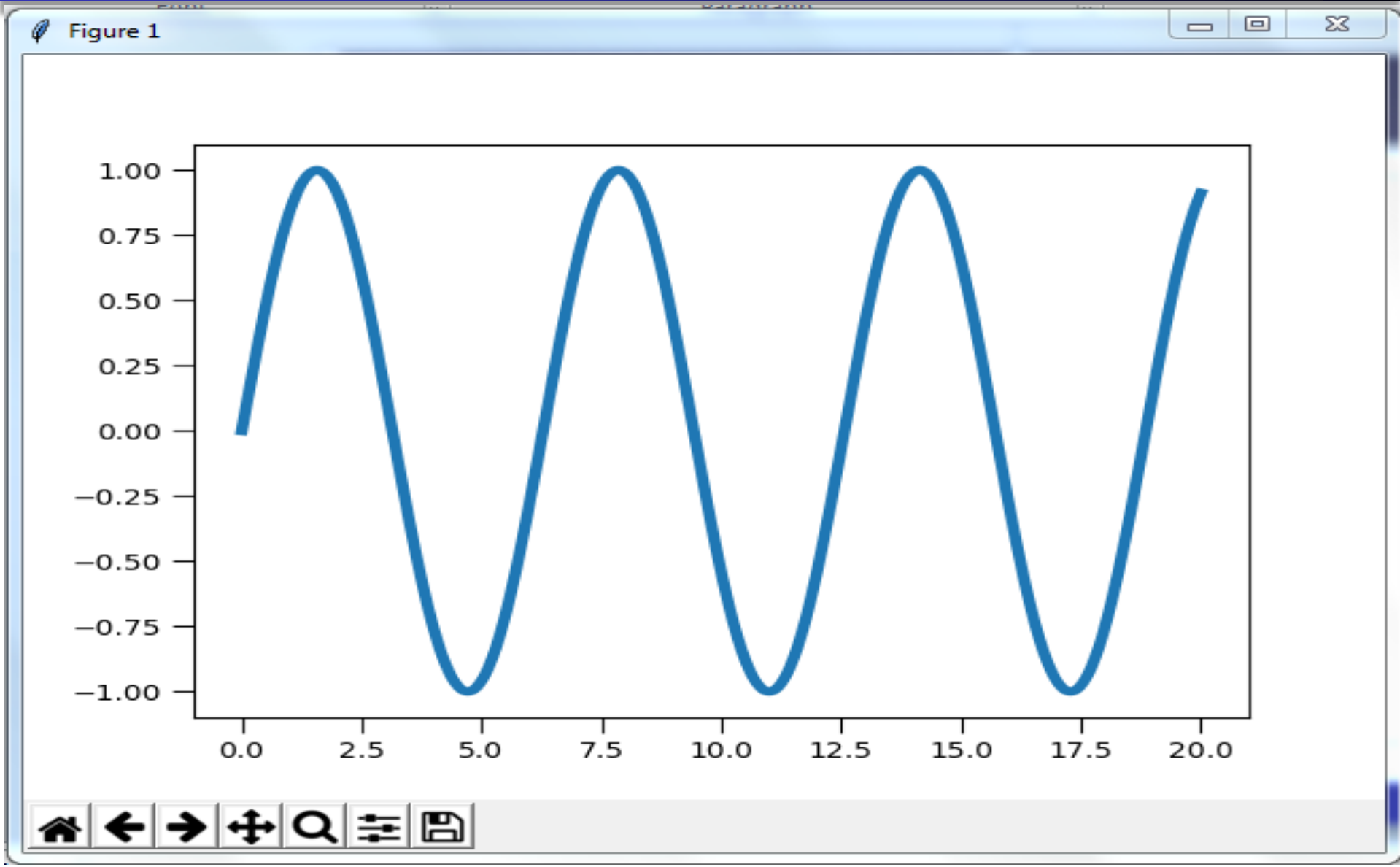
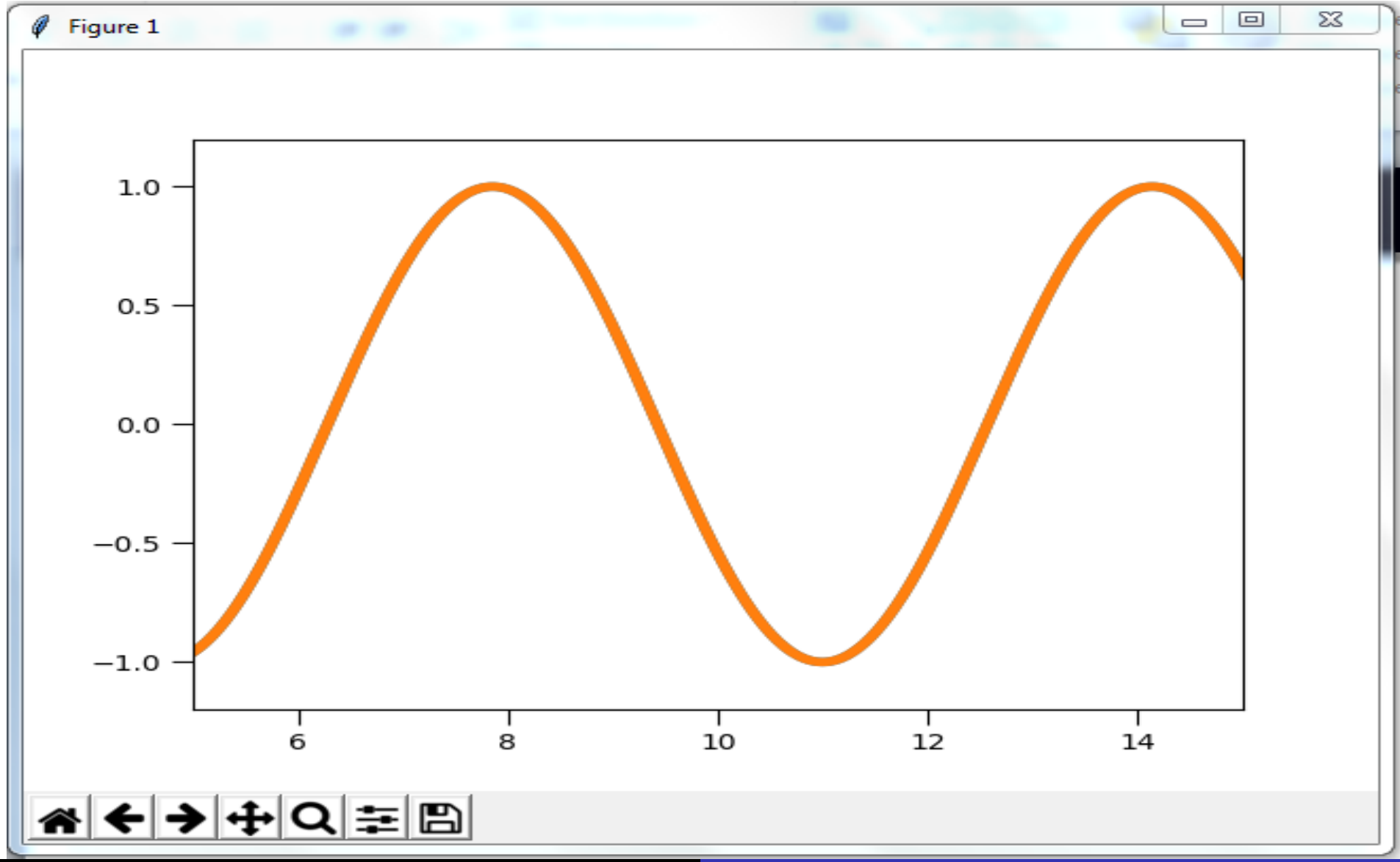


Figure 1
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

numpy…

```
import pylab
import numpy as np
x = np.linspace(0, 20, 1000)  # 100 evenly-spaced values from 0 to 50
y = np.sin(x)
pylab.plot(x, y)
pylab.xlim(5, 15)
pylab.ylim(-1.2, 1.2)
pylab.plot(x, y)
pylab.show()
```

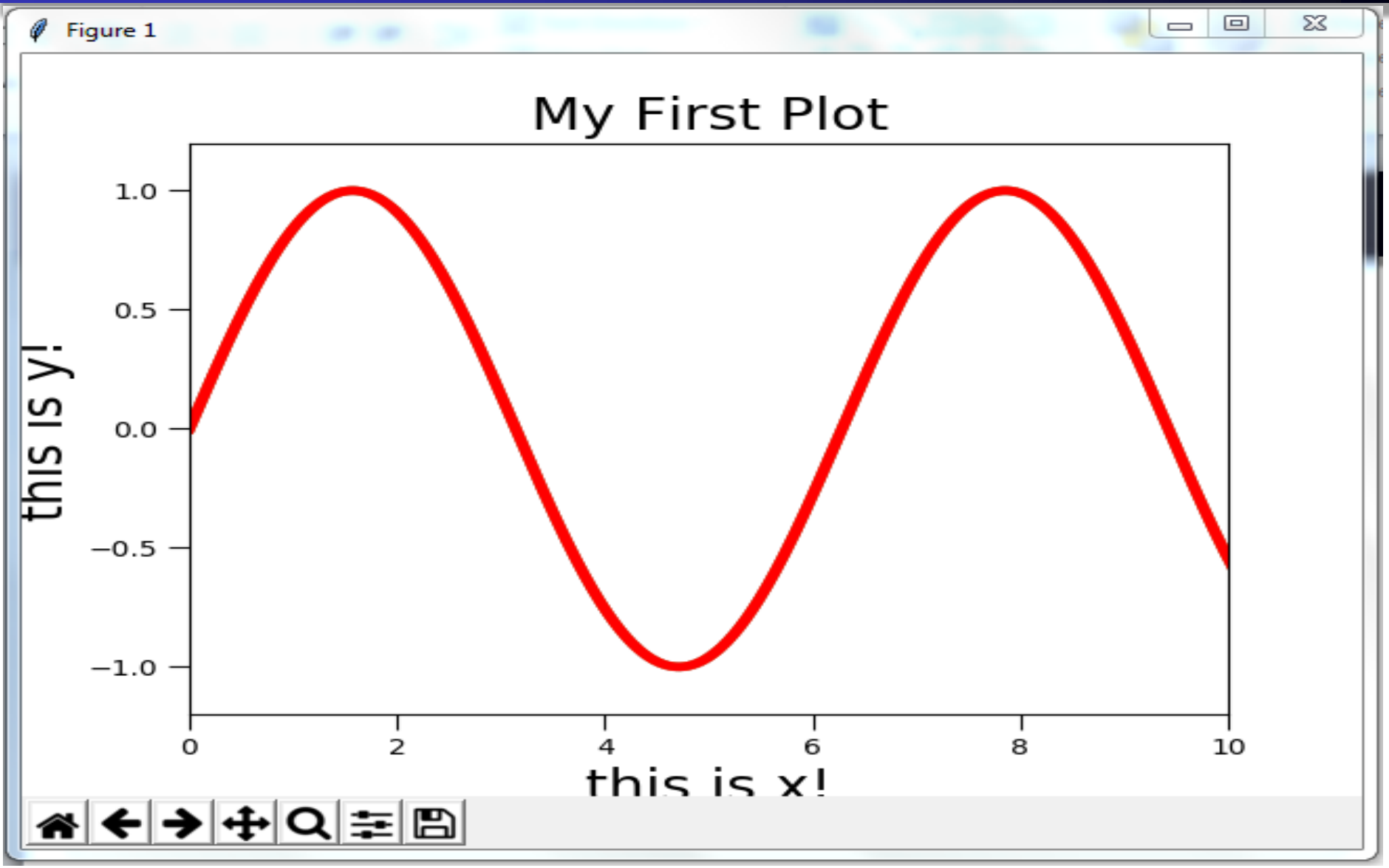
numpy…



Color

```
pylab.plot(x, y, '-r')  # solid red line ('r' comes from RGB color scheme)
pylab.xlim(0, 10)
pylab.ylim(-1.2, 1.2)

pylab.xlabel('this is x!')
pylab.ylabel('this is y!')
pylab.title('My First Plot')
```

Color…



Color…

- Other options for the color characters are:
- 'r' = red
- 'g' = green
- 'b' = blue
- 'c' = cyan
- 'm' = magenta
- 'y' = yellow
- 'k' = black
- 'w' = white

Line Styles

- Options for line styles are
- '-' = solid
- '--' = dashed
- ':' = dotted
- '-.' = dot-dashed
- '.' = points
- 'o' = filled circles
- '^' = filled triangles

Legend

```
import pylab
x = np.linspace(0, 20, 1000)
y1 = np.sin(x)
y2 = np.cos(x)

pylab.plot(x, y1, '-b', label='sine')
pylab.plot(x, y2, '-r', label='cosine')
pylab.legend(loc='upper right')
pylab.ylim(-1.5, 2.0)
pylab.xlabel('')
pylab.ylabel('')
pylab.show()
```

Legend…

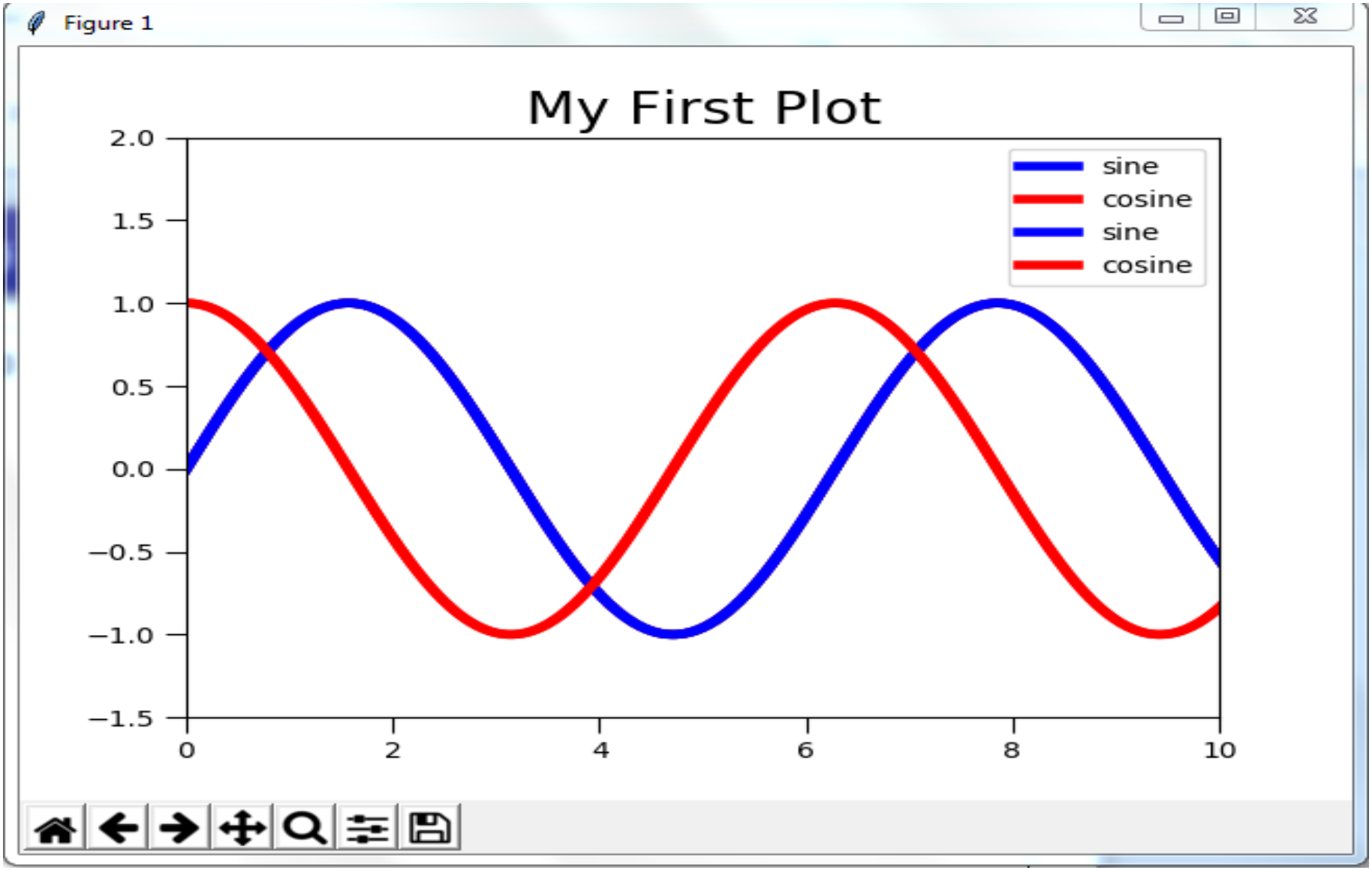


Figure 1
My First Plot
sine
cosine
sine
cosine
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0
2
4
6
8
10

Legend…

```
import pylab
x1 = np.linspace(0, 10, 20)
y1 = np.sin(x1)

x2 = np.linspace(0, 10, 1000)
y2 = np.sin(x2)
pylab.plot(x1, y1, 'bo', label='sampled')
pylab.plot(x2, y2, ':k', label='continuous')
pylab.legend()

pylab.ylim(-1.5, 2.0)
pylab.show()
```

Legend…

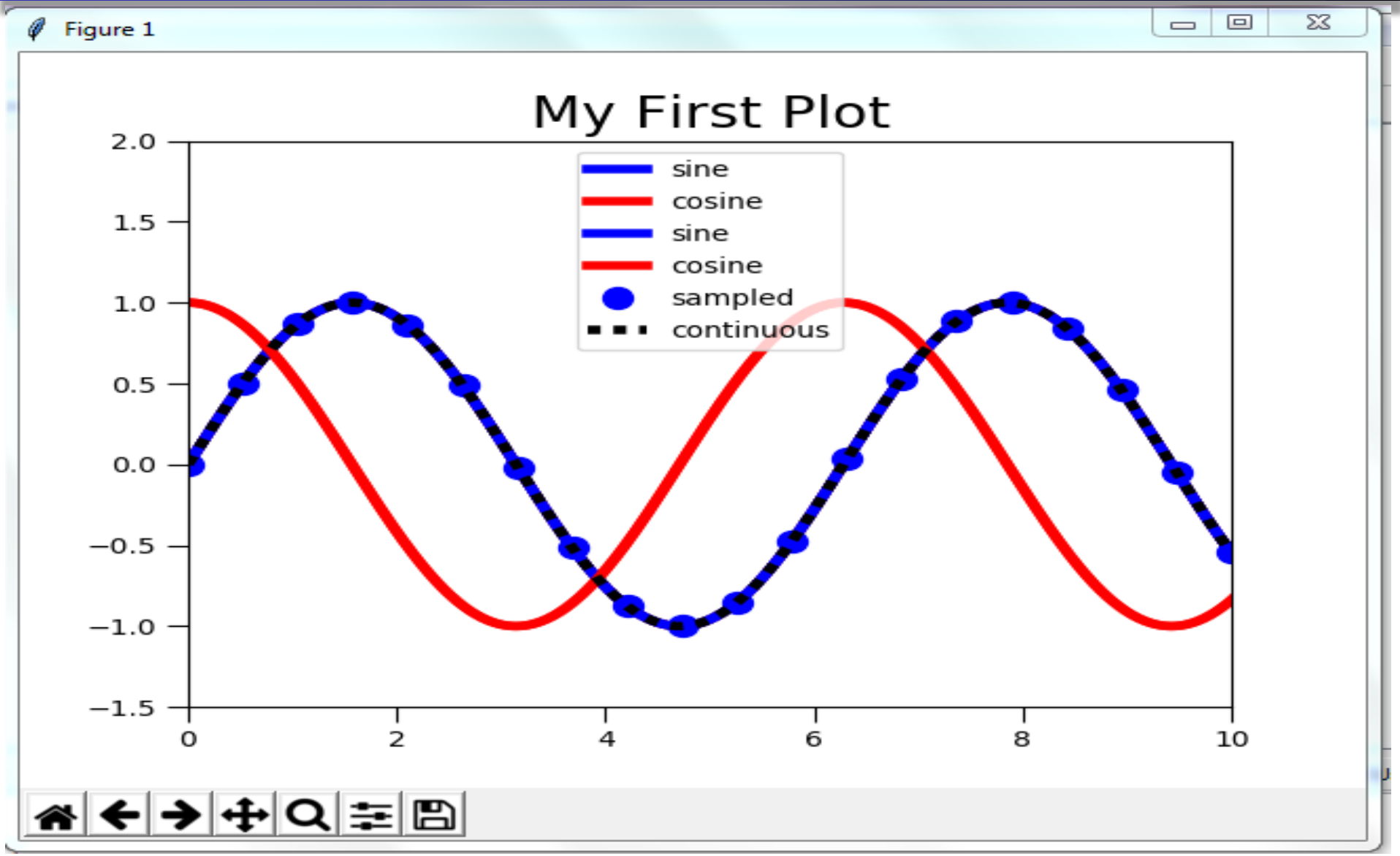
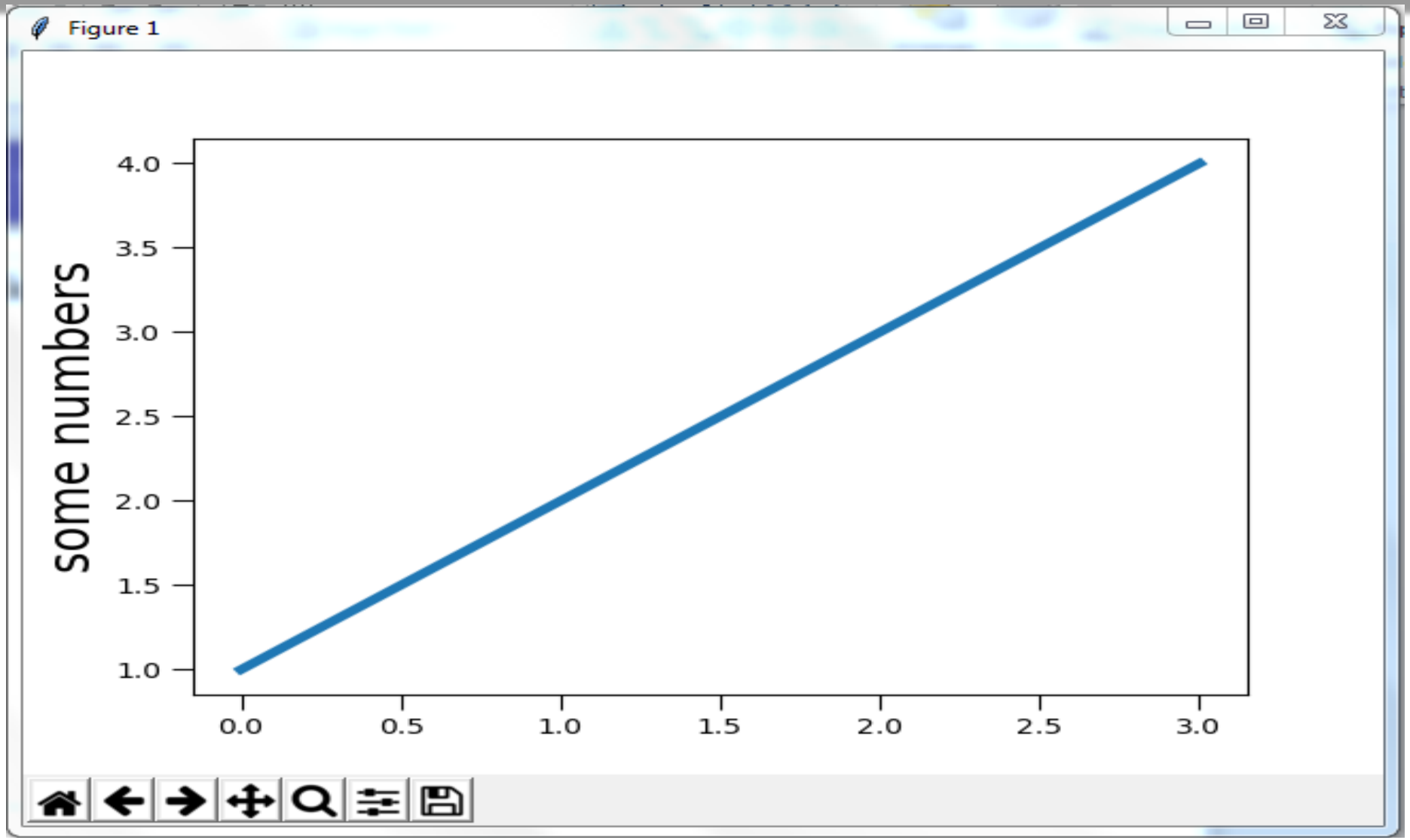


Figure 1
My First Plot
sine
cosine
sine
cosine
sampled
continuous
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0
2
4
6
8
10

matplotlib

```
import matplotlib.pyplot as plt
plt.plot([1,2,3,4])
plt.ylabel('some numbers')
plt.show()
```

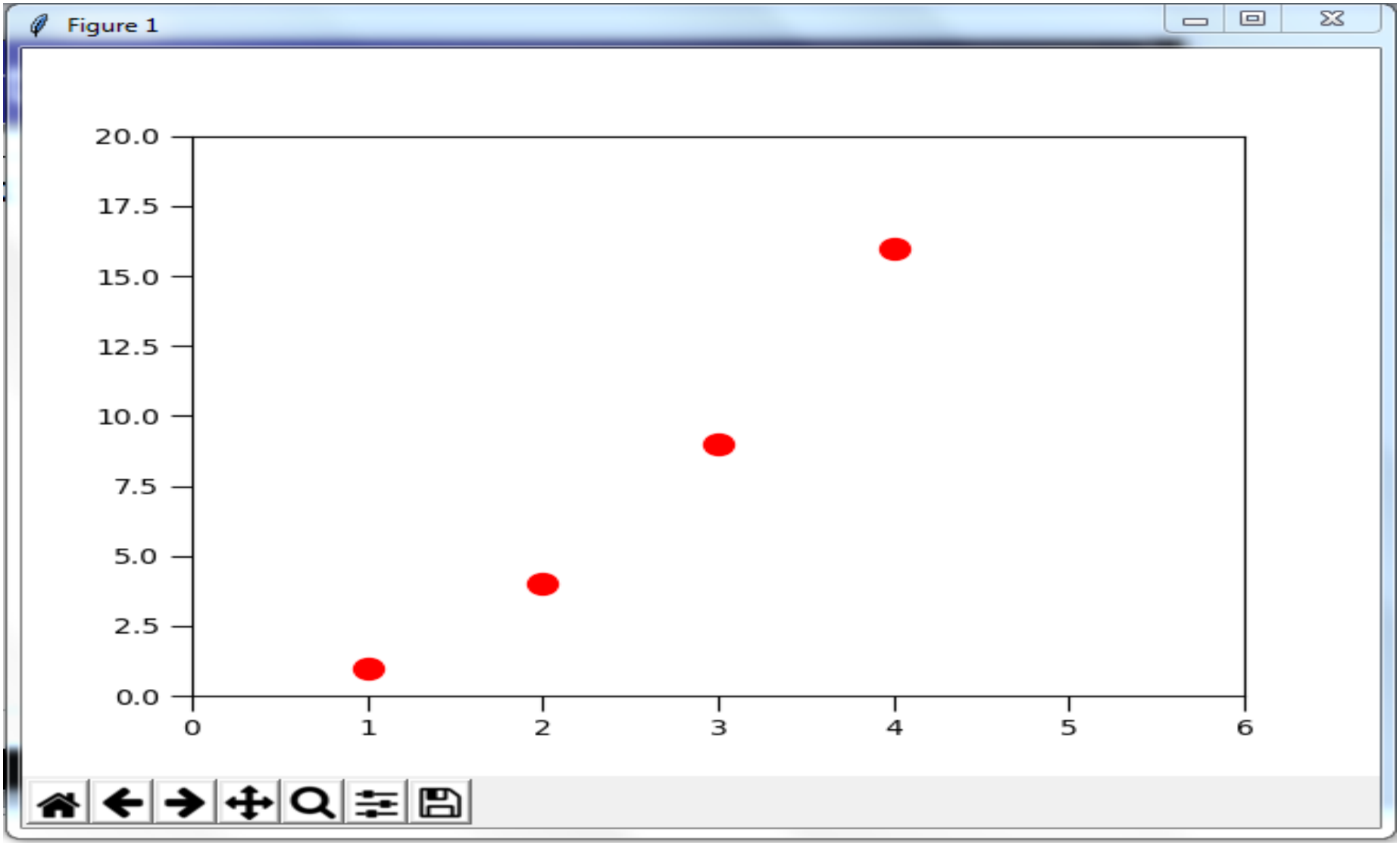
matplotlib…



matplotlib…

```
import matplotlib.pyplot as plt
plt.plot([1,2,3,4], [1,4,9,16], 'ro')
plt.axis([0, 6, 0, 20])
plt.show()
```

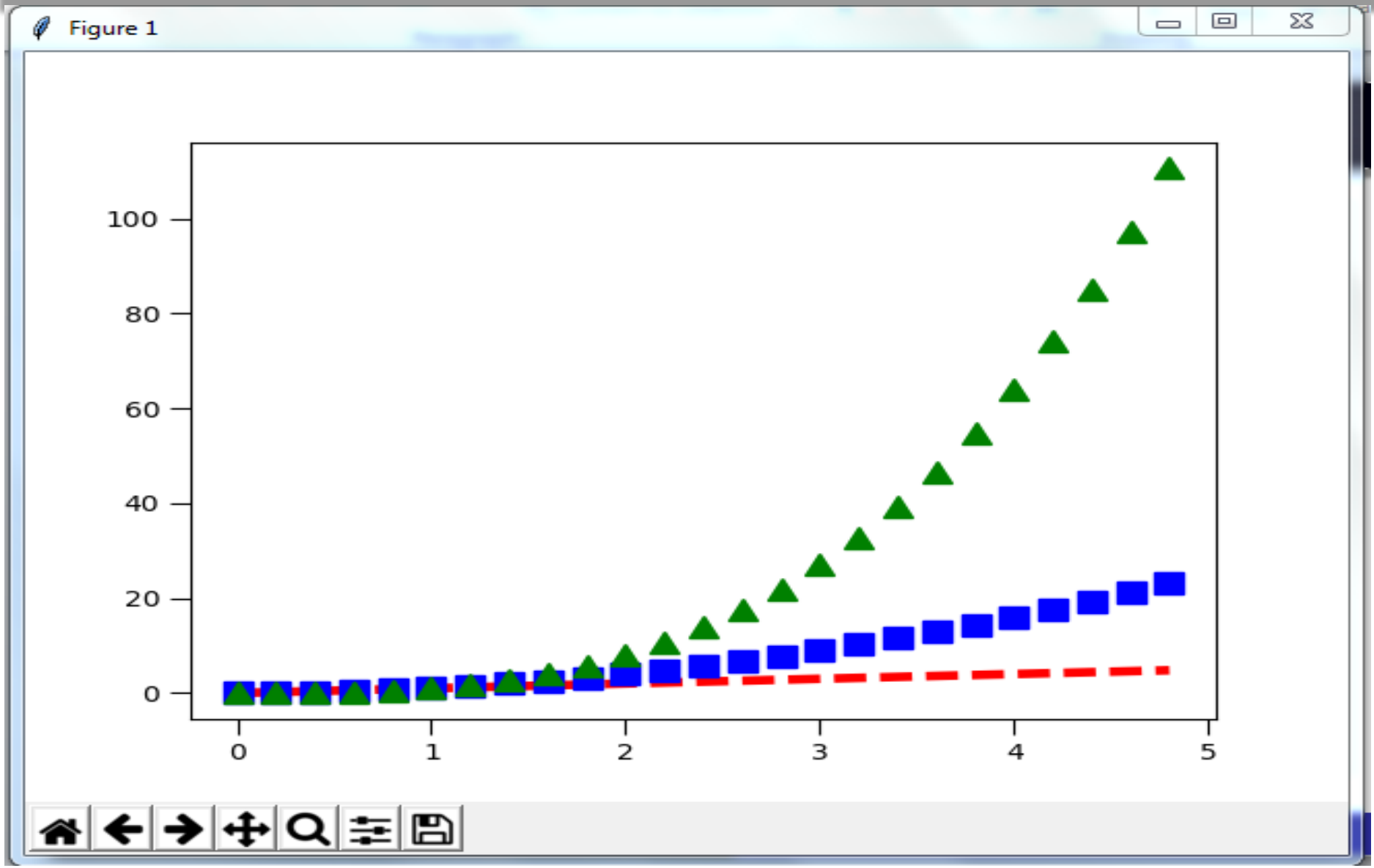
matplotlib…



matplotlib…

```
import numpy as np
import matplotlib.pyplot as plt

# evenly sampled time at 200ms intervals
t = np.arange(0., 5., 0.2)

# red dashes, blue squares and green triangles
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
plt.show()
```

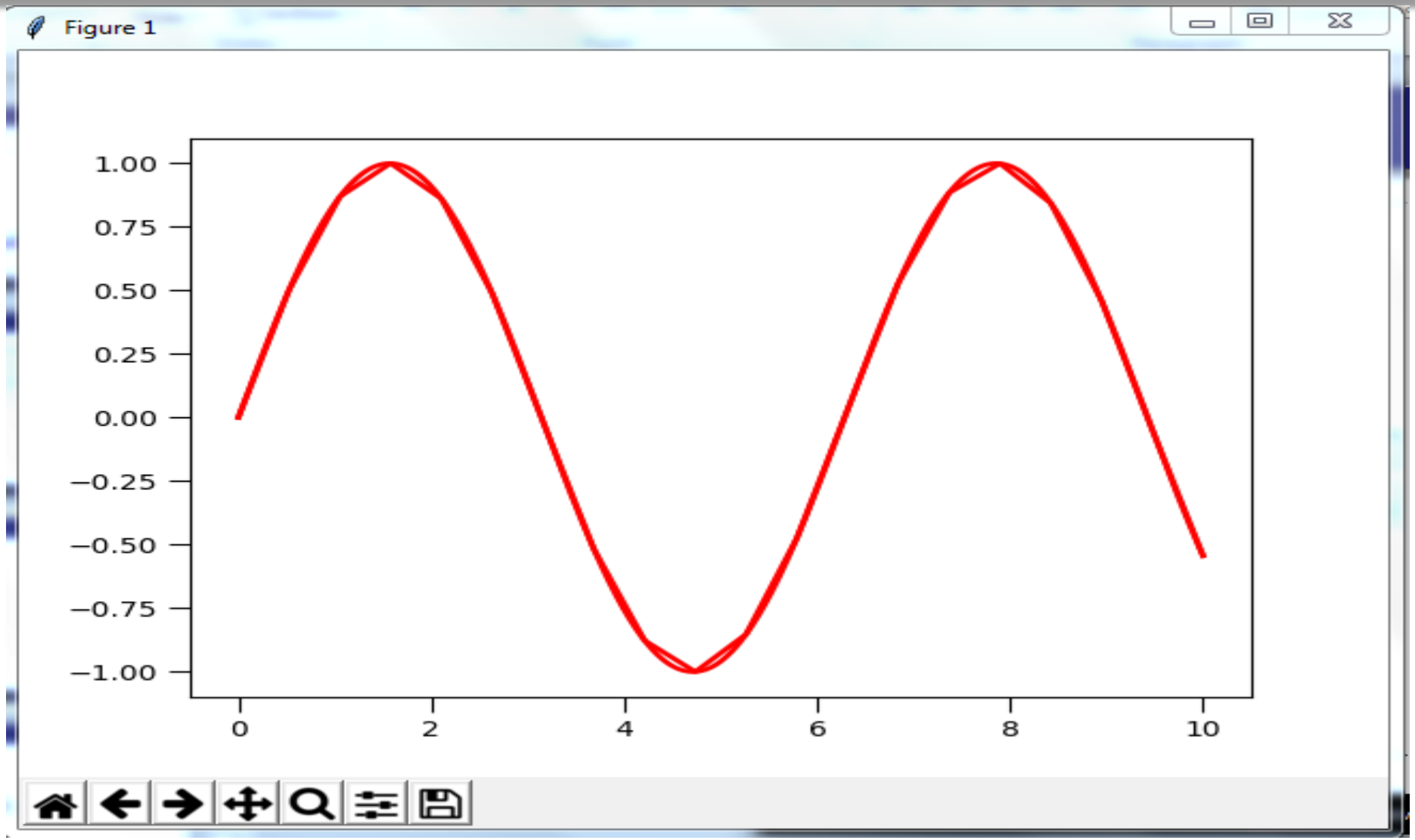
matplotlib…



matplotlib…

```
import numpy as np
import matplotlib.pyplot as plt

lines = plt.plot(x1, y1, x2, y2)
# use keyword args
plt.setp(lines, color='r', linewidth=2.0)
# or MATLAB style string value pairs
plt.setp(lines, 'color', 'r', 'linewidth', 2.0)
plt.show()
```

matplotlib…



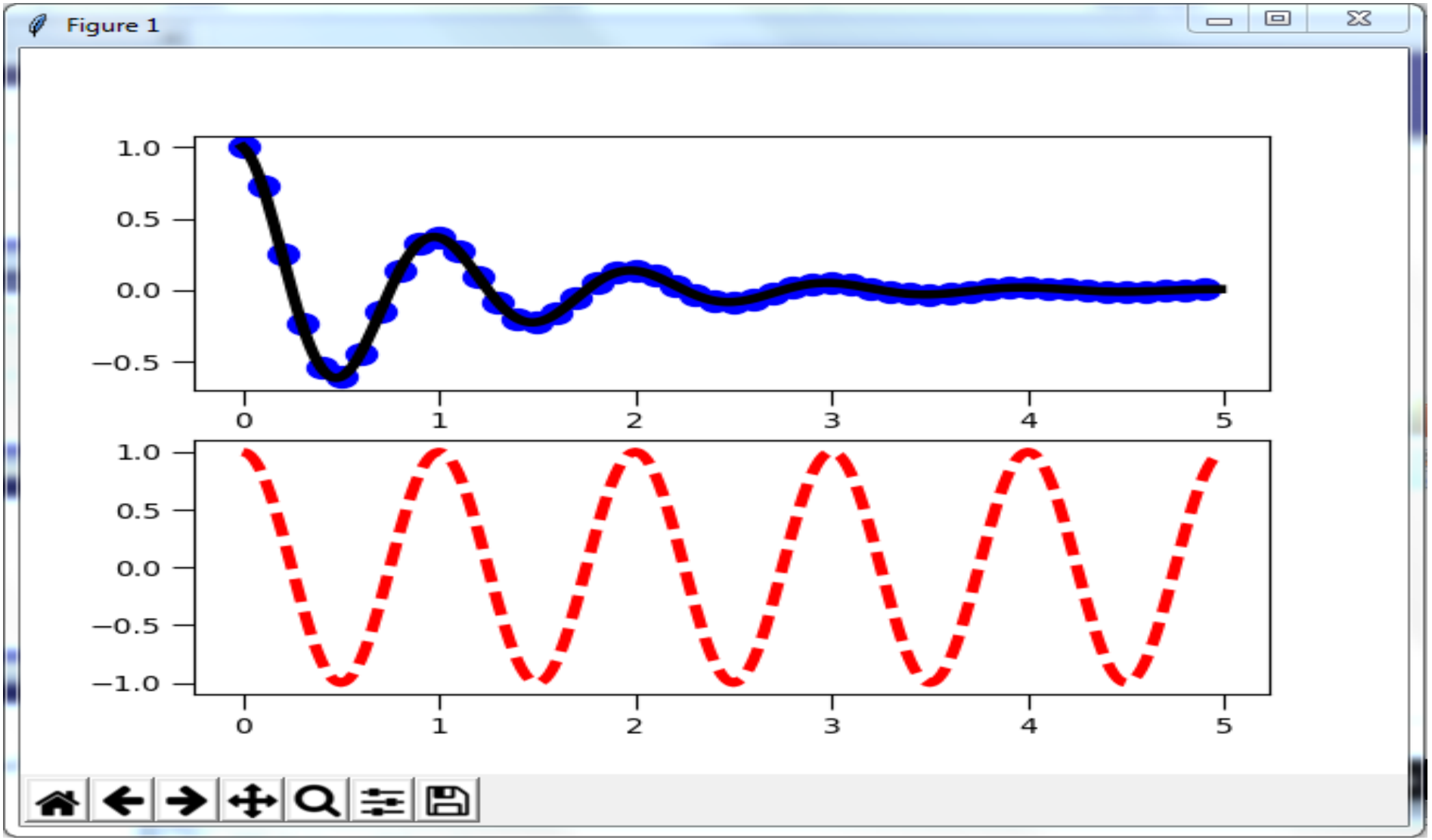
Property	Value Type
alpha	float
animated	[True | False]
antialiased or aa	[True | False]
clip_box	a matplotlib.transform.Bbox instance
clip_on	[True | False]
clip_path	a Path instance and a Transform instance, a Patch
color or c	any matplotlib color
contains	the hit testing function
dash_capstyle	['butt' | 'round' | 'projecting']
dash_joinstyle	['miter' | 'round' | 'bevel']
dashes	sequence of on/off ink in points
data	(np.array xdata, np.array ydata)
figure	a matplotlib.figure.Figure instance
label	any string
linestyle or ls	['-' | '--' | '-.' | ':' | 'steps' | ...]
linewidth or lw	float value in points
lod	[True | False]
marker	['+' | ',' | '.' | '1' | '2' | '3' | '4']
markeredgecolor or mec	any matplotlib color
markeredgewidth or mew	float value in points
markerfacecolor or mfc	any matplotlib color
markersize or ms	float
markevery	[None | integer | (startind, stride)]

picker	used in interactive line selection
pickradius	the line pick selection radius
solid_capstyle	['butt' | 'round' | 'projecting']
solid_joinstyle	['miter' | 'round' | 'bevel']
transform	a matplotlib.transforms.Transform instance
visible	[True | False]
xdata	np.array
ydata	np.array
zorder	any number

Working with multiple figures and axes

```
import numpy as np
import matplotlib.pyplot as plt

def f(t):
    return np.exp(-t) * np.cos(2*np.pi*t)

t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)

plt.figure(1)
plt.subplot(211)
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')

plt.subplot(212)
plt.plot(t2, np.cos(2*np.pi*t2), 'r--')
plt.show()
```

Working with multiple figures and axes…



Subplot

```
import matplotlib.pyplot as plt
plt.figure(1)                # the first figure
plt.subplot(211)             # the first subplot in the first figure
plt.plot([1, 2, 3])
plt.subplot(212)             # the second subplot in the first figure
plt.plot([4, 5, 6])


plt.figure(2)                # a second figure
plt.plot([4, 5, 6])          # creates a subplot(111) by default

plt.figure(1)                # figure 1 current; subplot(212) still current
plt.subplot(211)             # make subplot(211) in figure1 current
plt.title('Easy as 1, 2, 3') # subplot 211 title
plt.show()
```

Subplot…

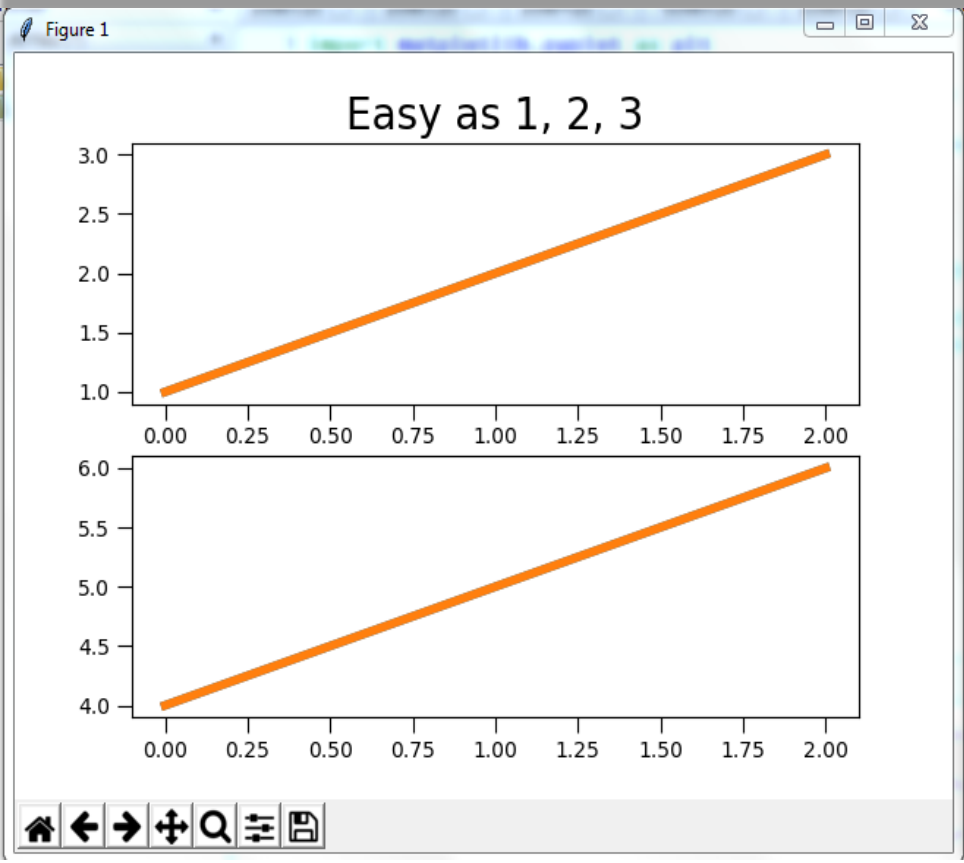


Figure 1
Easy as 1, 2, 3
3.0
2.5
2.0
1.5
1.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
6.0
5.5
5.0
4.5
4.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

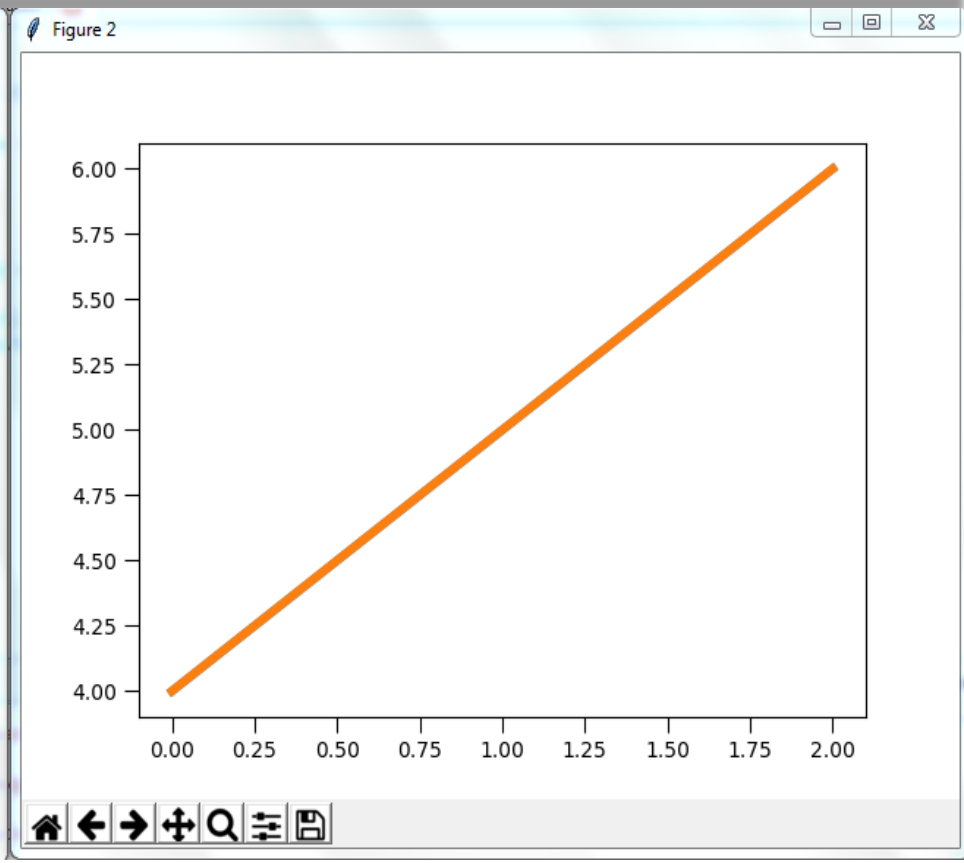
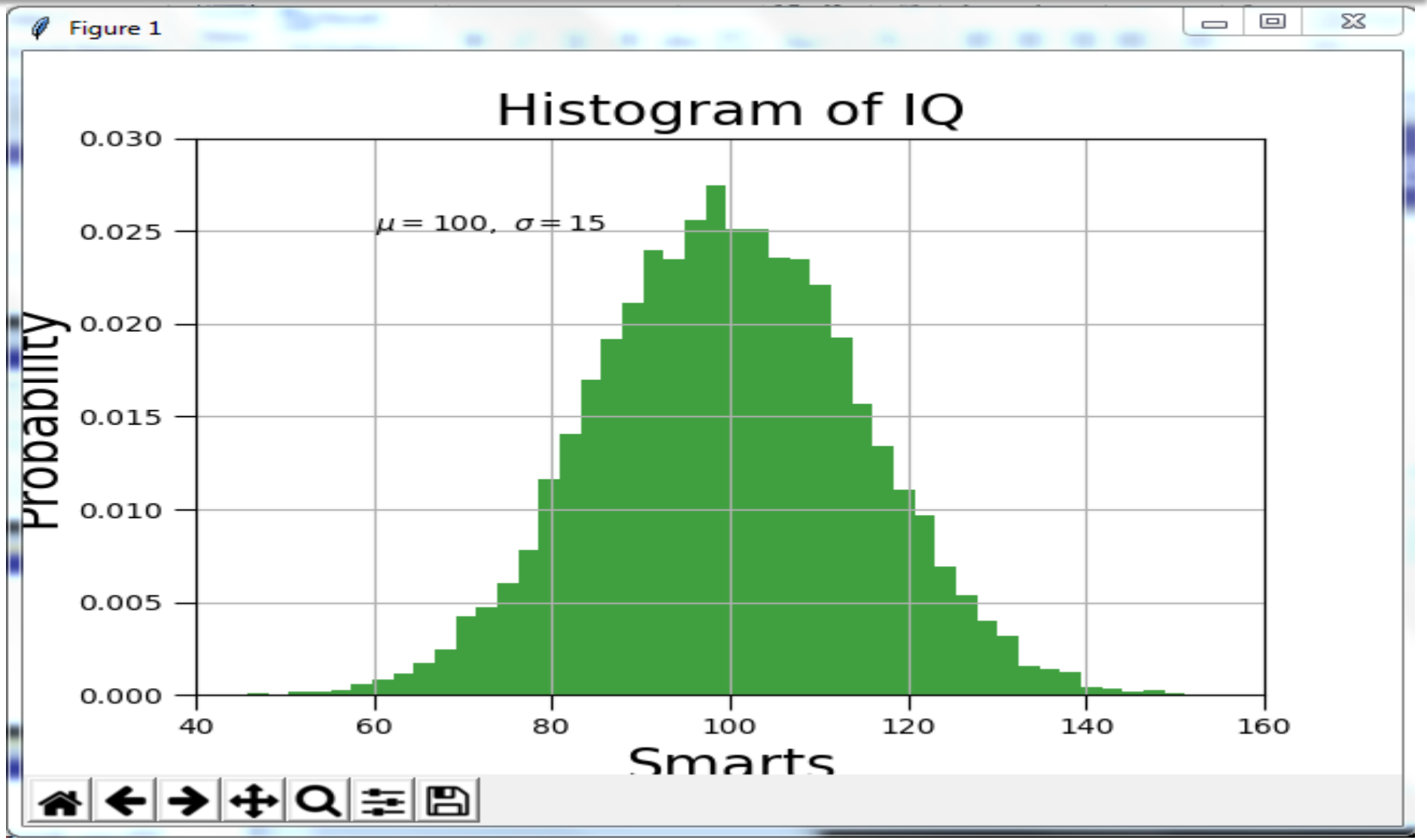


Figure 2
6.00
5.75
5.50
5.25
5.00
4.75
4.50
4.25
4.00
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

Working with text

```
import numpy as np
import matplotlib.pyplot as plt

# Fixing random state for reproducibility
np.random.seed(19680801)

mu, sigma = 100, 15
x = mu + sigma * np.random.randn(10000)

# the histogram of the data
n, bins, patches = plt.hist(x, 50, normed=1, facecolor='g', alpha=0.75)


plt.xlabel('Smarts')
plt.ylabel('Probability')
plt.title('Histogram of IQ')
plt.text(60, .025, r'$\mu=100,\ \sigma=15$')
plt.axis([40, 160, 0, 0.03])
plt.grid(True)
plt.show()
```

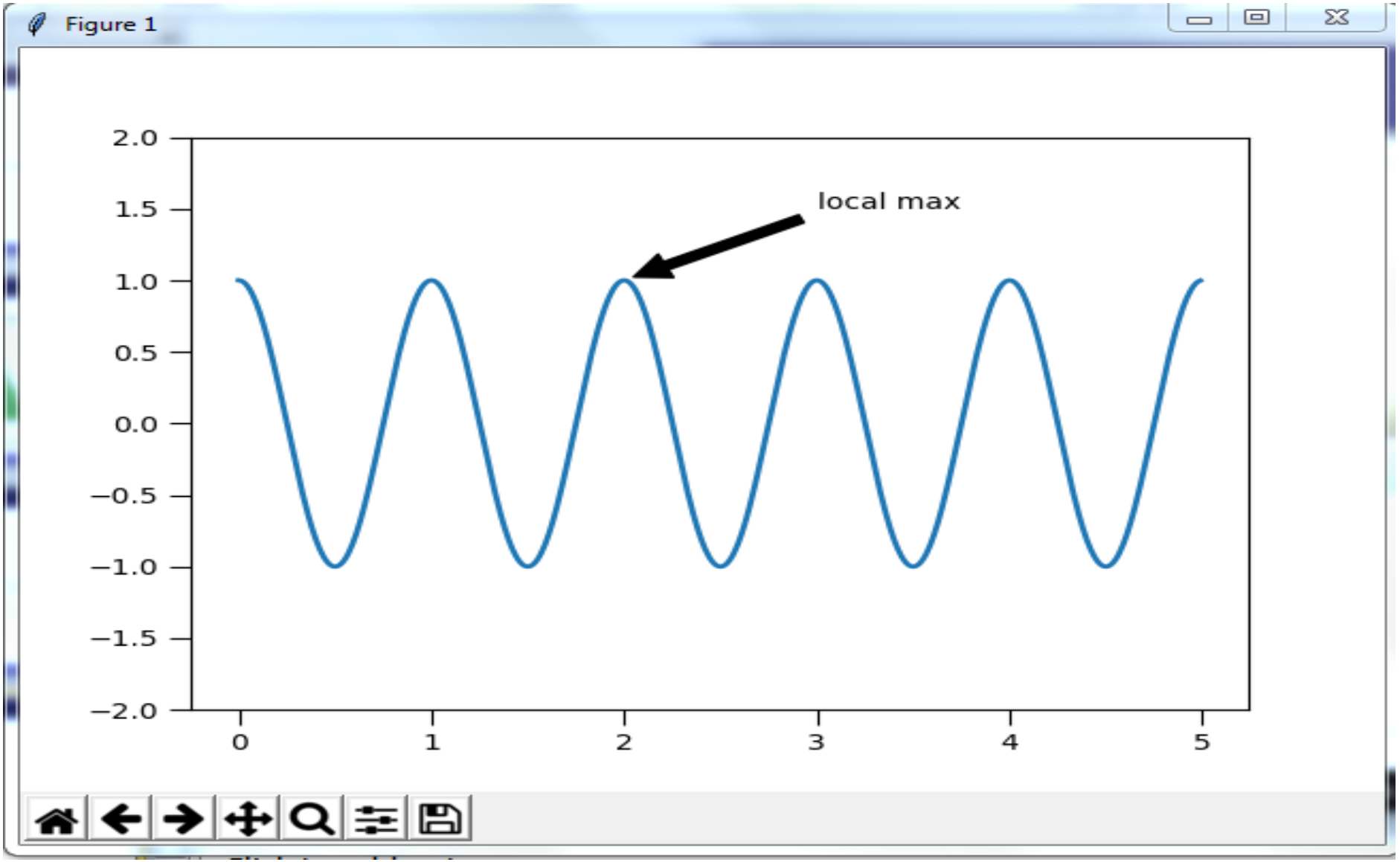
Working with text…



Annotating text

```
import numpy as np
import matplotlib.pyplot as plt

ax = plt.subplot(111)

t = np.arange(0.0, 5.0, 0.01)
s = np.cos(2*np.pi*t)
line, = plt.plot(t, s, lw=2)

plt.annotate('local max', xy=(2, 1), xytext=(3, 1.5),
             arrowprops=dict(facecolor='black', shrink=0.05),
             )

plt.ylim(-2,2)
plt.show()
```

Annotating text

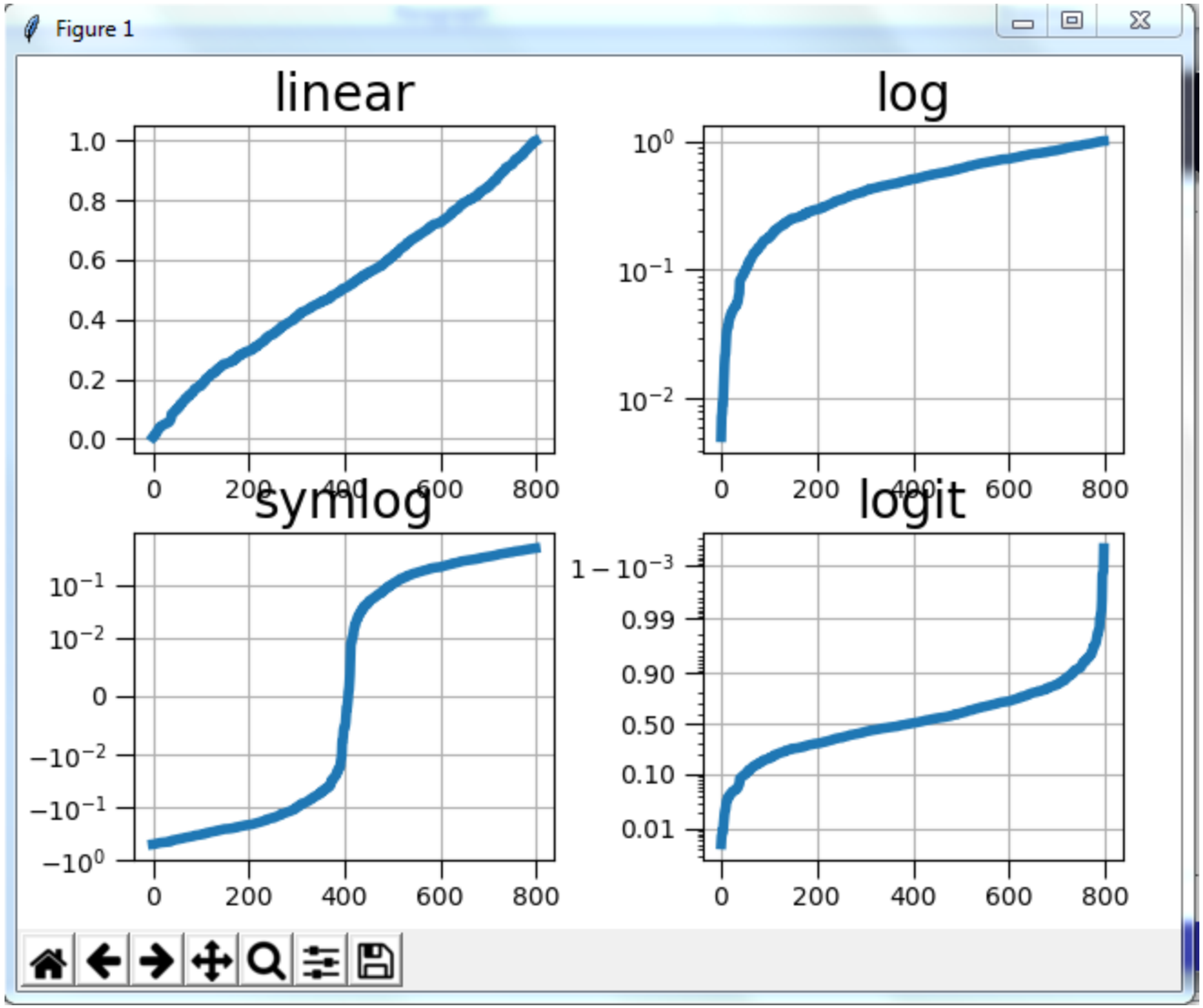


```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import NullFormatter  # useful for `logit` scale
# Fixing random state for reproducibility
np.random.seed(19680801)
# make up some data in the interval ]0, 1[
y = np.random.normal(loc=0.5, scale=0.4, size=1000)
y = y[(y > 0) & (y < 1)]
y.sort()
x = np.arange(len(y))
# plot with various axes scales
plt.figure(1)
# linear
plt.subplot(221)
plt.plot(x, y)
plt.yscale('linear')
plt.title('linear')
plt.grid(True)
# log
plt.subplot(222)
plt.plot(x, y)
plt.yscale('log')
plt.title('log')
plt.grid(True)
# symmetric log
plt.subplot(223)
plt.plot(x, y - y.mean())
plt.yscale('symlog', linthreshy=0.01)
plt.title('symlog')
plt.grid(True)
```

Working with multiple figures

```
# symmetric log
plt.subplot(223)
plt.plot(x, y - y.mean())
plt.yscale('symlog', linthreshy=0.01)
plt.title('symlog')
plt.grid(True)
# logit
plt.subplot(224)
plt.plot(x, y)
plt.yscale('logit')
plt.title('logit')
plt.grid(True)
# Format the minor tick labels of the y-axis into empty strings with
# `NullFormatter`, to avoid cumbering the axis with too many labels.
plt.gca().yaxis.set_minor_formatter(NullFormatter())
# Adjust the subplot layout, because the logit one may take more space
# than usual, due to y-tick labels like "1 - 10^{-3}"
plt.subplots_adjust(top=0.92, bottom=0.08, left=0.10, right=0.95, hspace=0.25,
                    wspace=0.35)
plt.show()
```

Working with multiple figures…



Thank you!!!