2. Visualization

Computational Methods, Sept. 2017, Kenji Doya

Visualizatin is vital in data analysis and scientific computing, to

- get intuitive understanding
- come up with a new hypothesis
- detect a bug or data anormaly

That is why we'll cover this topic at the beginning of this course.

Matplotlib

Matplotlib is the standard graphics package for Python. It mimics many graphics functions of MATLAB.
The Matplotlib gallery (http://matplotlib.org/gallery.html (http://matplotlib.org/gallery.html)) illustrates variety of plots.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

A nice feature of Jupyter notebook is that you can embed the figures produced by your program within the notebook by the following *magic* command:

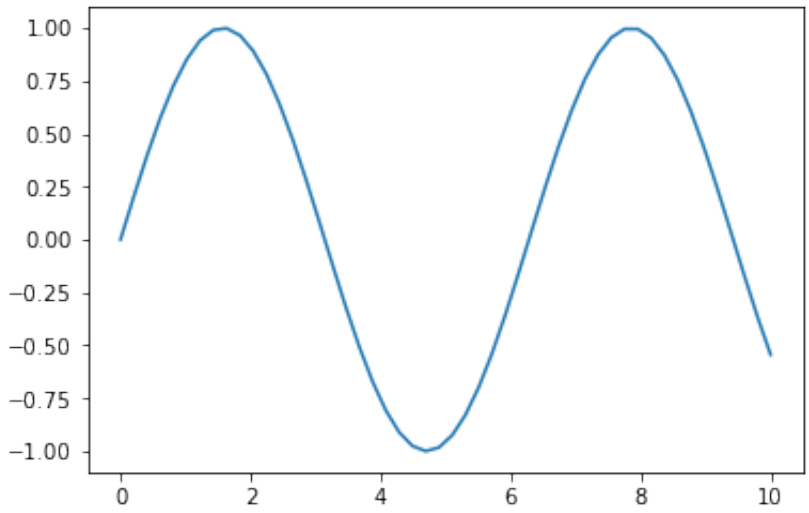
In [2]:
```
%matplotlib inline
```

Plotting functions

The standard way is to prepare an array for x values, compute y values, and pass x and y to plot( ) function.

In [3]:
```
# make an array from 0 to 10, the default is 50 points
x = np.linspace(0, 10)
# comupute a function for each point
y = np.sin(x)
# plot the points
plt.plot(x, y)  # by default, points are connected by a line
```

Out[3]:
```
[<matplotlib.lines.Line2D at 0x10fe3aeb8>]
```



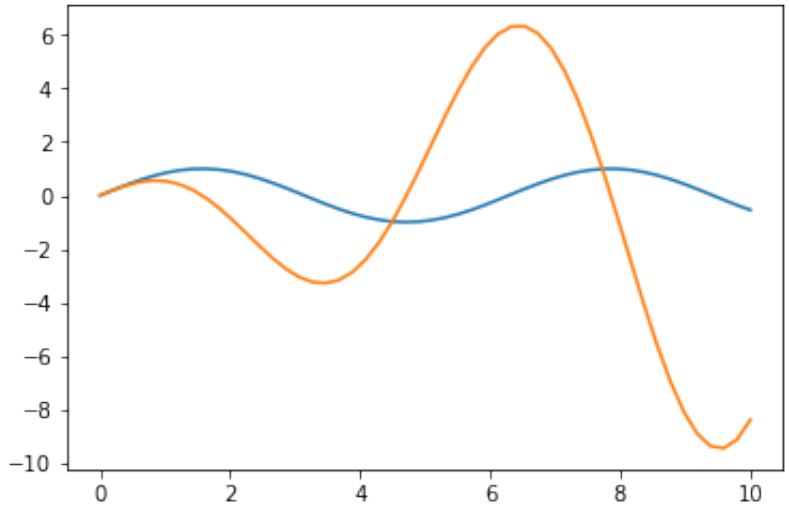
There are multiple ways to pass variables to plot():

- `plot(y)`: x is assumed to be [0, len(y)-1]
- `plot(x, y)`
- `plot(x1, y1, x2, y2,...)`: multiple lines
- `plot(x, Y)`: lines for columns of matrix Y

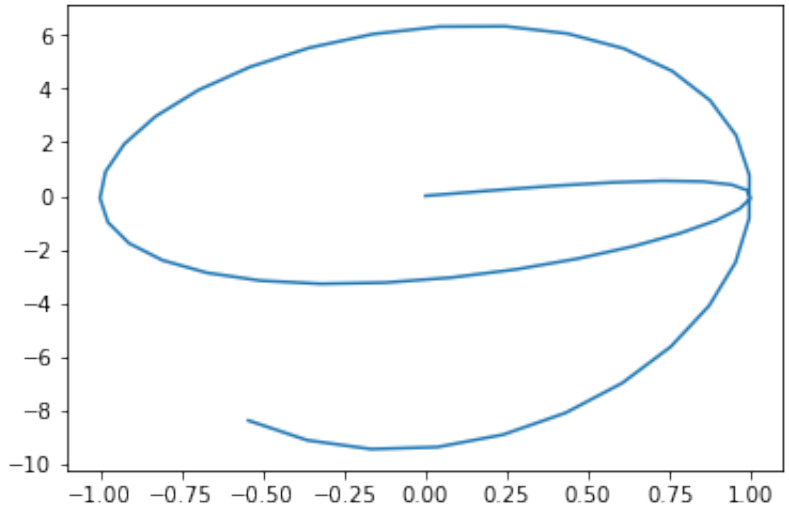
In [4]:
```
# another function of x
y2 = np.cos(x)*x
```

In [5]:
```
# plot two lines
plt.plot(x, y, x, y2)
```

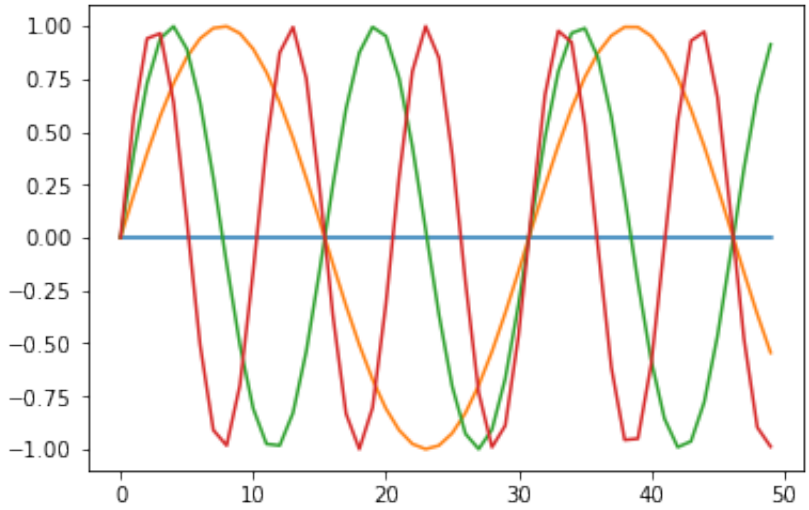
Out[5]:
```
[<matplotlib.lines.Line2D at 0x10ff6efd0>,
 <matplotlib.lines.Line2D at 0x10ff77978>]
```



In [6]:
```
# phase plot
plt.plot(y, y2);
# you can supress <matplotlib...> output by ;
```



In [7]:
```
# plot multiple lines by a matrix
Y = np.array([np.sin(k*x) for k in range(4)])
plt.plot(Y.T);
```

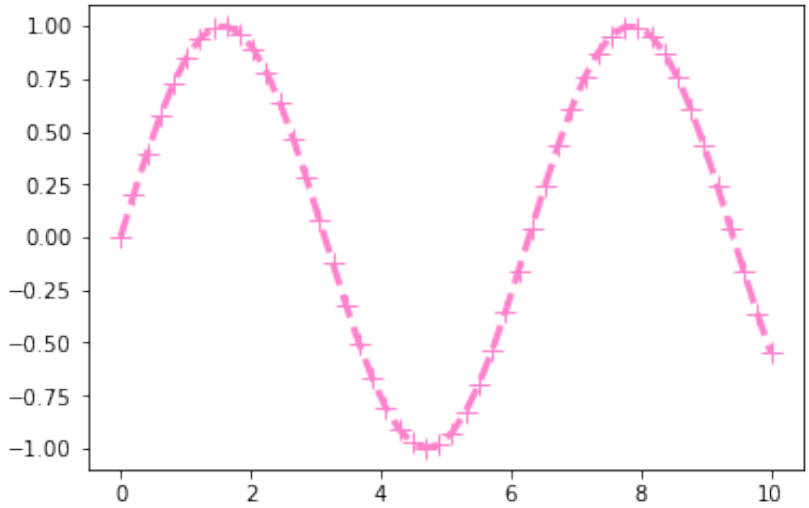


Options for plotting

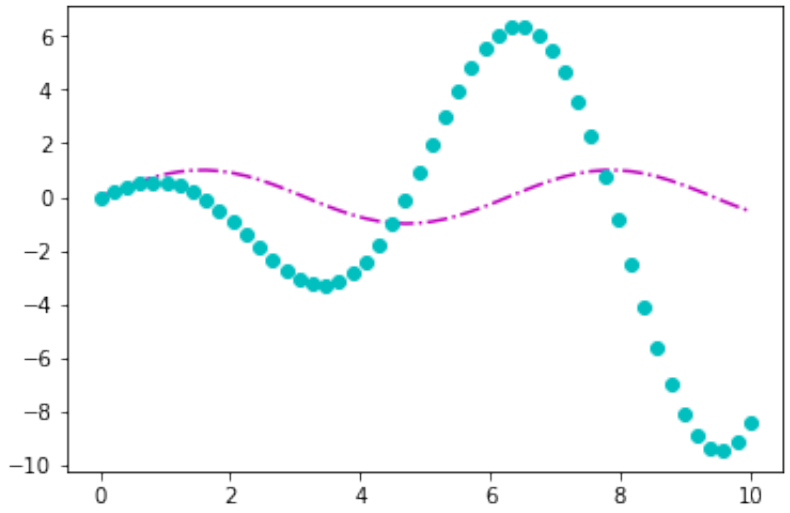
Line styles can be specified by

- `color=` (or `c=`) for color by code, name, RGB or RGBA
  - code: 'r', 'g', 'b', 'y', 'c', 'm', 'k', 'w'
- `marker=` for marker style
  - code: '.', 'o', '+', '*', '^', ...
- `linestyle=` (or `ls=`) for line style
  - code: '-', '--', ':', '-.', ...
- `linewidth=` (or `lw=`) for line width
- a string of color, marker and line sytel codes, e.g. 'ro:'

In [8]:
```
# using keyword=value
plt.plot(x, y, c=[1,0.5,0.8], marker='+', markersize=10, ls='--', lw=3
```



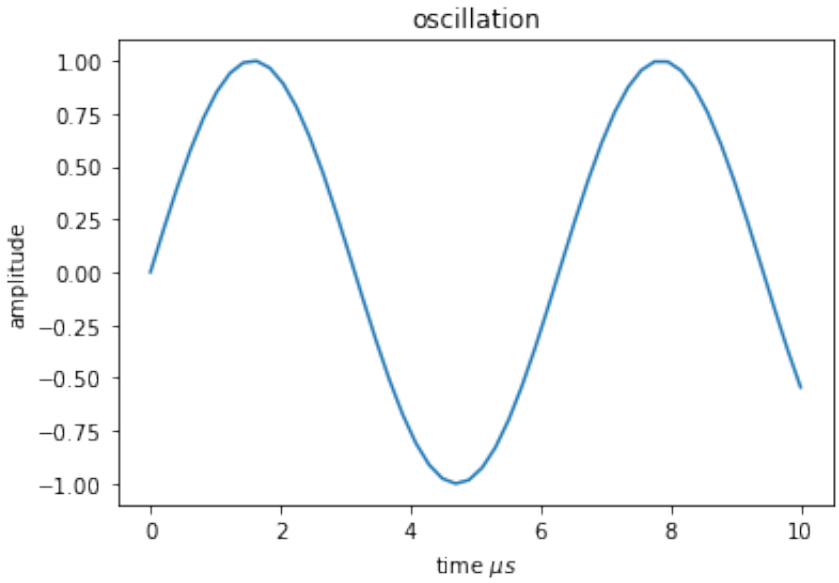
In [9]:
```
# using code string
plt.plot(x, y, 'm-.', x, y2, 'co');  # magenta dash-dot, cyan circle
```



It's a good habit to add axis lables and plot title.

In [10]:
```
plt.plot(x, y)
plt.xlabel('time $\mu s$')
plt.ylabel('amplitude')
plt.title('oscillation')
```

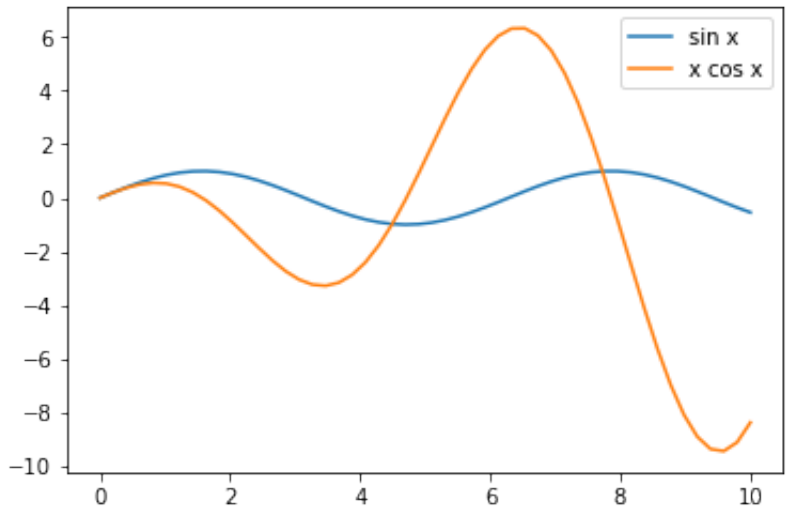
Out[10]: <matplotlib.text.Text at 0x1103e4f98>



It is also nice to add a legend box.

In [11]:
```
plt.plot(x, y, x, y2)
plt.legend(('sin x','x cos x'))
```

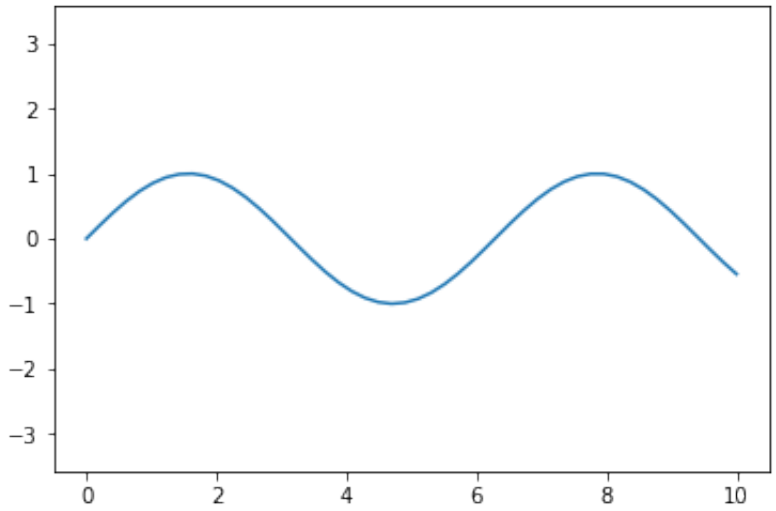
Out[11]: <matplotlib.legend.Legend at 0x1104a6358>



You can control axis ranges and scaling.

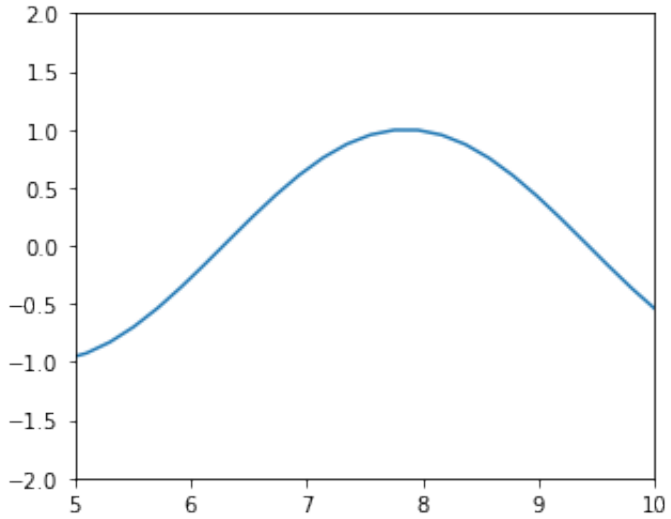
In [12]:
```
plt.plot(x, y)
plt.axis('equal')  # equal scale for x and y
```

Out[12]: (-0.5, 10.5, -1.09972447591003, 1.0979832896606587)



In [13]:
```
plt.plot(x, y)
plt.axis('square')  # change plot area for equal scaling
plt.xlim(5, 10)
plt.ylim(-2, 2)
```

Out[13]: (-2, 2)

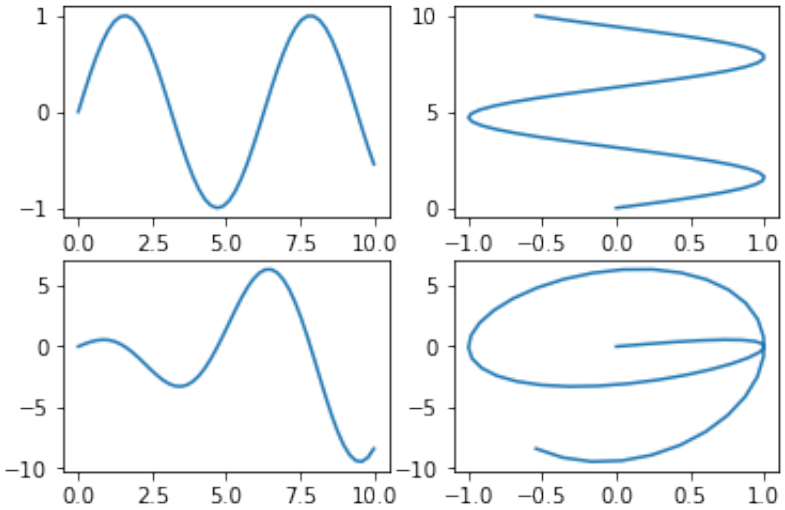


Subplot and axes

You can create multiple *axes* in a figure by subplot(rows, columns, index).
It uses a MATLAB legacy for index starting from 1.

In [14]:
```
plt.subplot(2, 2, 1)
plt.plot(x, y)
plt.subplot(2, 2, 2)
plt.plot(y, x)
plt.subplot(2, 2, 3)
plt.plot(x, y2)
plt.subplot(2, 2, 4)
plt.plot(y, y2)
```

Out[14]:
```
[<matplotlib.lines.Line2D at 0x1113c2a58>]
```



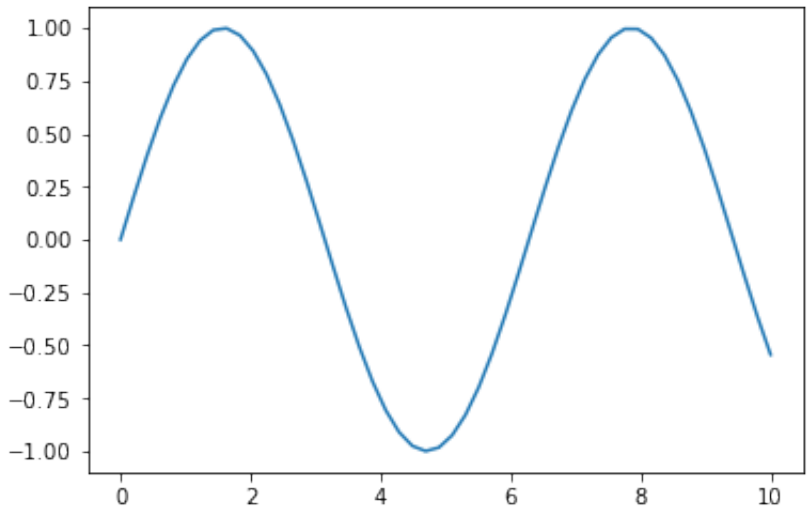
In [ ]:
```
```

Figure and axes

When you make a plot, matplotlib creates a figure object with an *axes* object. You can use gcf() and gca() to identify them and getp() and setp() to access their parameters.

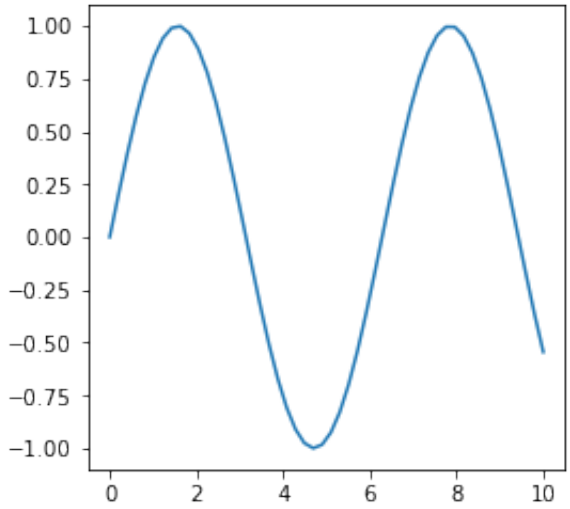
In [15]:
```
#fig = plt.figure(figsize=(6, 6))  # make a new figure
plt.plot(x, y)
fig = plt.gcf()  # get current figure
plt.getp(fig)  # show all parameters
```

```
    agg_filter = None
    alpha = None
    animated = False
    axes = [<matplotlib.axes._subplots.AxesSubplot object at ...
    children = [<matplotlib.patches.Rectangle object at 0x110397f...
    clip_box = None
    clip_on = True
    clip_path = None
    contains = None
    default_bbox_extra_artists = [<matplotlib.axes._subplots.AxesSub
plot object at ...
    dpi = 72.0
    edgecolor = (1.0, 1.0, 1.0, 0.0)
    facecolor = (1.0, 1.0, 1.0, 0.0)
    figheight = 4.0
    figure = None
    figwidth = 6.0
    frameon = True
    gid = None
    label =
    path_effects = []
    picker = None
    rasterized = None
    size_inches = [ 6.  4.]
    sketch_params = None
    snap = None
    tight_layout = False
    transform = IdentityTransform()
    transformed_clip_path_and_affine = (None, None)
    url = None
    visible = True
    window_extent = TransformedBbox(Bbox([[0.0, 0.0], [6.0, 4.0]]),
Af...
    zorder = 0
```



In [16]:
```
plt.plot(x, y)
fig = plt.gcf()
plt.setp(fig, size_inches=(4,4))
```

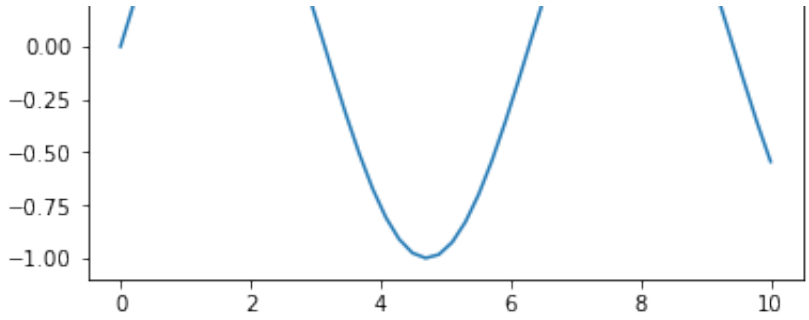
Out[16]: [None]



In [17]:
```
ax = plt.subplot(111)  # make a new axes
plt.plot(x, y)
#ax = plt.gca()  # get current axes
plt.getp(ax)
```

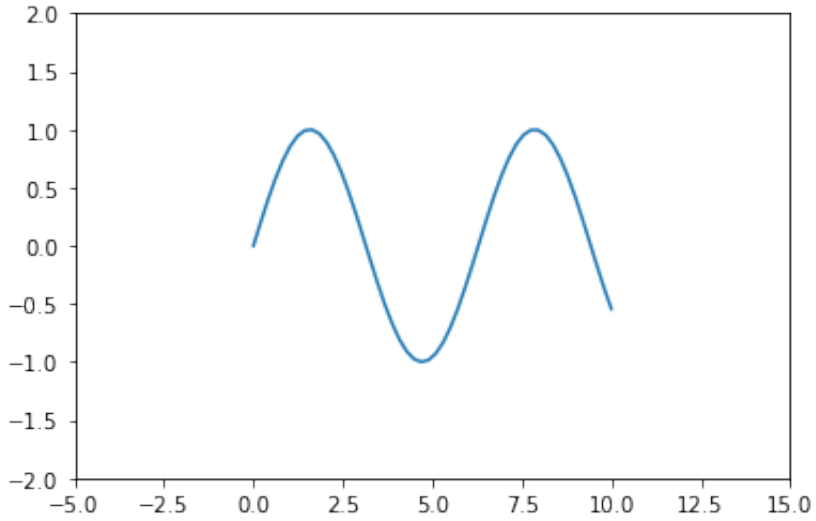
```
    adjustable = box
    agg_filter = None
    alpha = None
    anchor = C
    animated = False
    aspect = auto
    autoscale_on = True
    autoscalex_on = True
    autoscaley_on = True
    axes = Axes(0.125,0.125;0.775x0.755)
    axes_locator = None
    axis_bgcolor = (1.0, 1.0, 1.0, 1)
    axisbelow = line
    children = [<matplotlib.lines.Line2D object at 0x1114f7b00>, ...
    clip_box = None
    clip_on = True
    clip_path = None
    contains = None
    cursor_props = (1, (0.0, 0.0, 0.0, 1))
    data_ratio = 0.1997916150518808
    default_bbox_extra_artists = [<matplotlib.lines.Line2D object at
0x1114f7b00>, ...
    facecolor = (1.0, 1.0, 1.0, 1)
    fc = (1.0, 1.0, 1.0, 1)
    figure = Figure(432x288)
    frame_on = True
    geometry = (1, 1, 1)
    gid = None
    images = <a list of 0 AxesImage objects>
```

```
    label =
    legend = None
    legend_handles_labels = ([], [])
    lines = <a list of 1 Line2D objects>
    navigate = True
    navigate_mode = None
    path_effects = []
    picker = None
    position = Bbox(x0=0.125, y0=0.125, x1=0.9, y1=0.88)
    rasterization_zorder = None
    rasterized = None
    renderer_cache = None
    shared_x_axes = <matplotlib.cbook.Grouper object at 0x106da8898>
    shared_y_axes = <matplotlib.cbook.Grouper object at 0x106da88d0>
    sketch_params = None
    snap = None
    subplotspec = <matplotlib.gridspec.SubplotSpec object at 0x1114c
...
    title =
    transform = IdentityTransform()
    transformed_clip_path_and_affine = (None, None)
    url = None
    visible = True
    window_extent = Bbox(x0=50.5, y0=32.5, x1=392.3, y1=256.94)
    xaxis = XAxis(54.000000,36.000000)
    xaxis_transform = BlendedGenericTransform(CompositeGenericTransf
orm(...
    xbound = (-0.5, 10.5)
    xgridlines = <a list of 8 Line2D xgridline objects>
    xlabel =
    xlim = (-0.5, 10.5)
    xmajorticklabels = <a list of 8 Text xticklabel objects>
    xminorticklabels = <a list of 0 Text xticklabel objects>
    xscale = linear
    xticklabels = <a list of 8 Text xticklabel objects>
    xticklines = <a list of 16 Text xtickline objects>
    xticks = [-2.  0.  2.  4.  6.  8.]...
    yaxis = YAxis(54.000000,36.000000)
    yaxis_transform = BlendedGenericTransform(BboxTransformTo(Transf
orme...
    ybound = (-1.09972447591003, 1.0979832896606587)
    ygridlines = <a list of 11 Line2D ygridline objects>
    ylabel =
    ylim = (-1.09972447591003, 1.0979832896606587)
    ymajorticklabels = <a list of 11 Text yticklabel objects>
    yminorticklabels = <a list of 0 Text yticklabel objects>
    yscale = linear
    yticklabels = <a list of 11 Text yticklabel objects>
    yticklines = <a list of 22 Line2D ytickline objects>
    yticks = [-1.25 -1.   -0.75 -0.5  -0.25  0.  ]...
    zorder = 0
```



In [18]:
```
plt.plot(x, y)
ax = plt.gca()  # get current axes
plt.setp(ax, xlim=[-5, 15], ylim=[-2, 2])  # change parameters
```

Out[18]: [-2, 2, -5, 15]



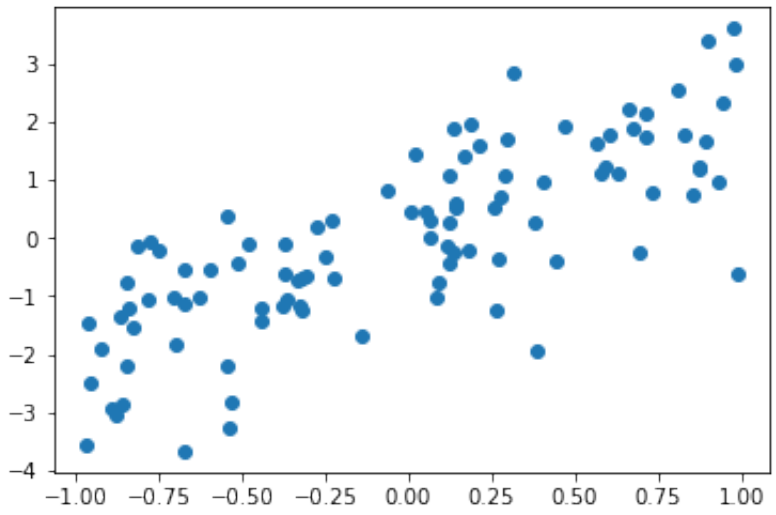
In [ ]:

Visualizing 2D data

A standard way for visualizing pariwise data is a scatter plot.

In [19]:
```
n = 100
x = np.random.uniform(-1, 1, n)  # n points in [-1,1]
y = 2*x + np.random.randn(n)  # scale and add noise
# plot
plt.plot(x, y, 'o')
#plt.scatter(x, y)
```

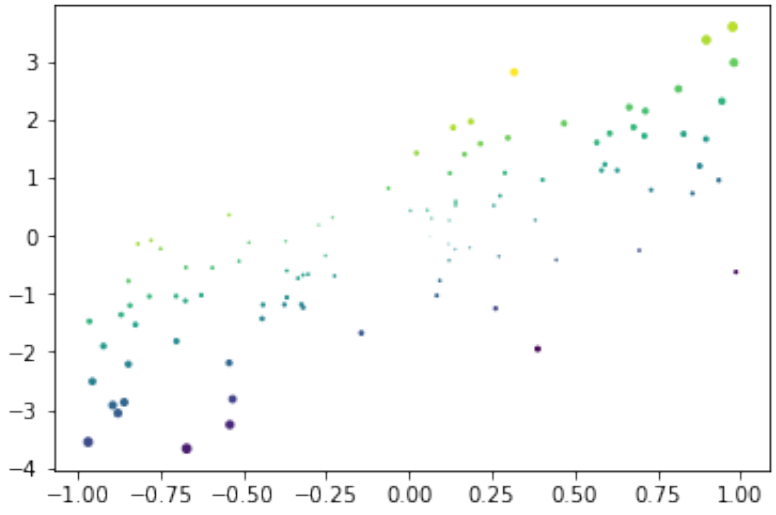
Out[19]:
```
[<matplotlib.lines.Line2D at 0x111535f60>]
```



By `scatterplot( )` you can specify the size and the color of each point to visualize higher dimension information.

In [20]:
```
z = x**2 + y**2
c = y - 2*x
# z for size, c for color
plt.scatter(x, y, z, c)
```

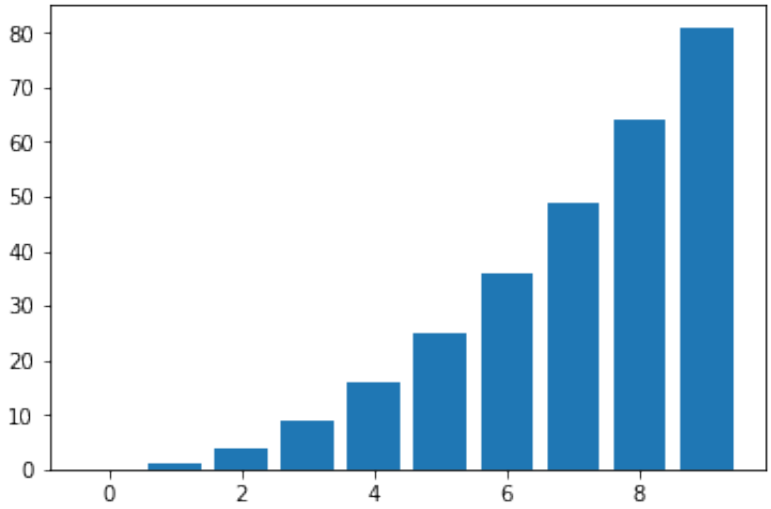
Out[20]:
```
<matplotlib.collections.PathCollection at 0x111804b38>
```



Bar plot and histogram

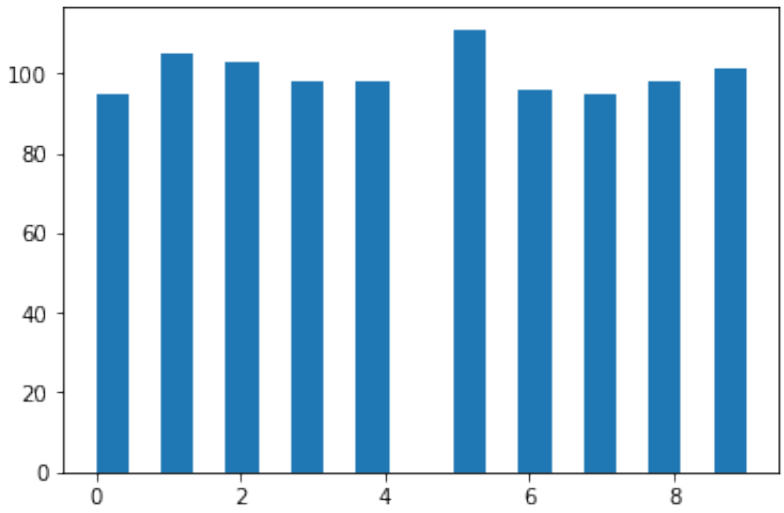
In [21]:
```
x = np.arange(10)
y = x**2
plt.bar(x, y)
```

Out[21]:
```
<Container object of 10 artists>
```



In [22]:
```
x = np.random.randint(0, 10, 1000)
plt.hist(x, 20)
```

Out[22]:
```
(array([  95.,    0.,  105.,    0.,  103.,    0.,   98.,    0.,   98
.,
          0.,    0.,  111.,    0.,   96.,    0.,   95.,    0.,   98
.,
          0.,  101.]),
 array([ 0.  ,  0.45,  0.9 ,  1.35,  1.8 ,  2.25,  2.7 ,  3.15,  3.6
,
        4.05,  4.5 ,  4.95,  5.4 ,  5.85,  6.3 ,  6.75,  7.2 ,  7.6
5,
        8.1 ,  8.55,  9.  ]),
 <a list of 20 Patch objects>)
```



In [ ]:

Visualizing a matrix or a function in 2D space

meshgrid() is for preparing x and y values in a grid.

In [23]:
```
x = np.linspace(-4, 4, 9)
y = np.linspace(-3, 3, 7)
print(x, y)
X, Y = np.meshgrid(x, y)
print(X, Y)
```

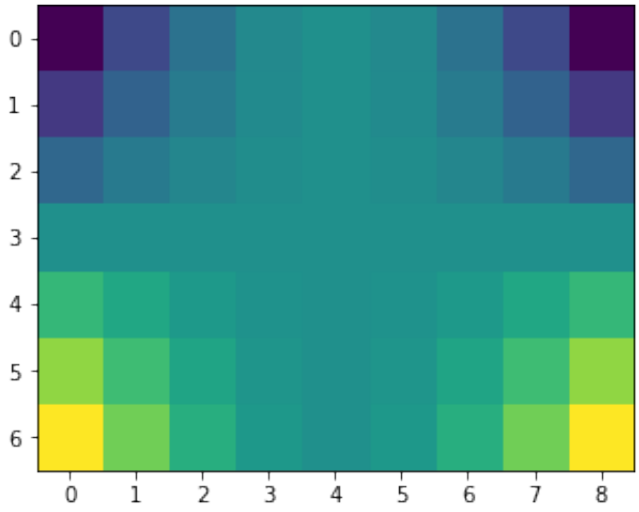
```
[-4. -3. -2. -1.  0.  1.  2.  3.  4.] [-3. -2. -1.  0.  1.  2.  3.]
[[-4. -3. -2. -1.  0.  1.  2.  3.  4.]
 [-4. -3. -2. -1.  0.  1.  2.  3.  4.]
 [-4. -3. -2. -1.  0.  1.  2.  3.  4.]
 [-4. -3. -2. -1.  0.  1.  2.  3.  4.]
 [-4. -3. -2. -1.  0.  1.  2.  3.  4.]
 [-4. -3. -2. -1.  0.  1.  2.  3.  4.]
 [-4. -3. -2. -1.  0.  1.  2.  3.  4.]] [[-3. -3. -3. -3. -3. -3. -3
. -3. -3.]
 [-2. -2. -2. -2. -2. -2. -2. -2. -2.]
 [-1. -1. -1. -1. -1. -1. -1. -1. -1.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.]
 [ 2.  2.  2.  2.  2.  2.  2.  2.  2.]
 [ 3.  3.  3.  3.  3.  3.  3.  3.  3.]]
```

We can use imshow() to visualize a matrix as an image.

In [24]:
```
Z = X**2 * Y
print(Z)
plt.imshow(Z)
```

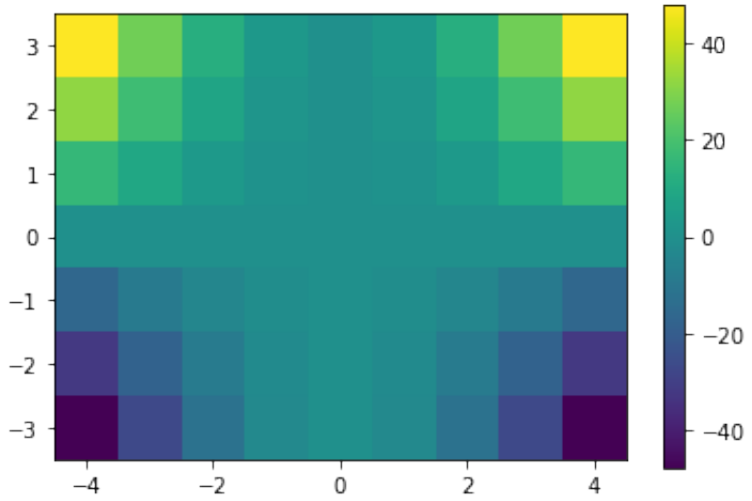
```
[[-48. -27. -12.  -3.  -0.  -3. -12. -27. -48.]
 [-32. -18.  -8.  -2.  -0.  -2.  -8. -18. -32.]
 [-16.  -9.  -4.  -1.  -0.  -1.  -4.  -9. -16.]
 [  0.   0.   0.   0.   0.   0.   0.   0.   0.]
 [ 16.   9.   4.   1.   0.   1.   4.   9.  16.]
 [ 32.  18.   8.   2.   0.   2.   8.  18.  32.]
 [ 48.  27.  12.   3.   0.   3.  12.  27.  48.]]
```

Out[24]: <matplotlib.image.AxesImage at 0x110066828>



In [25]:
```
# some more options
plt.imshow(Z, origin='lower', extent=(-4.5, 4.5, -3.5, 3.5))
plt.colorbar()
```

Out[25]: <matplotlib.colorbar.Colorbar at 0x110309a58>

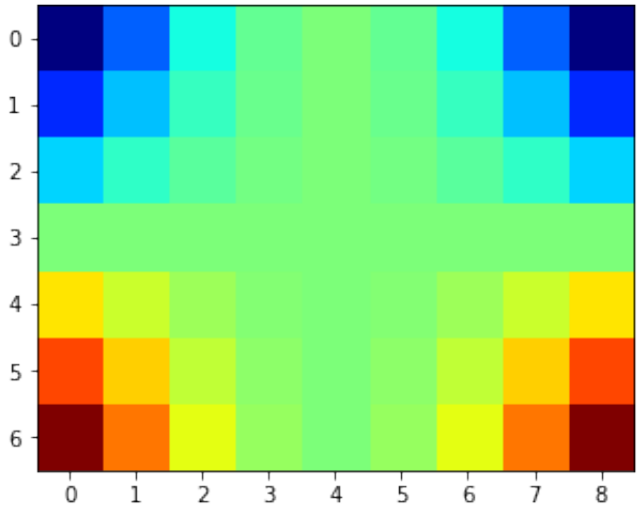


color maps

`imshow( )` maps a scalar Z value to color by a colormap. The standard color map *viridis* is friedly to color blindness and monochrome printing. You can also choose other color maps.

In [26]:
```
plt.imshow(Z, cmap='jet')
```

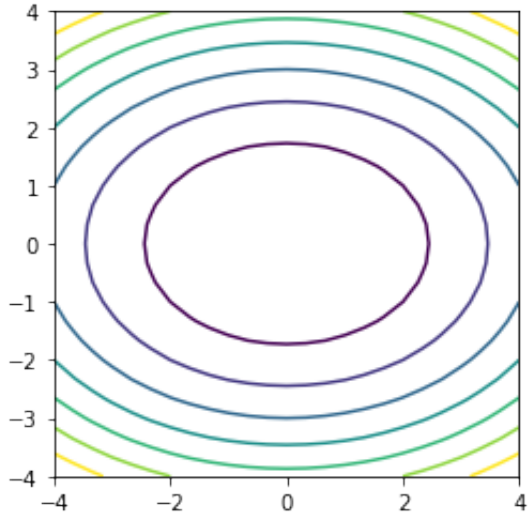
Out[26]:
```
<matplotlib.image.AxesImage at 0x111f2ca20>
```



contour plot

In [27]:
```
x = np.linspace(-4, 4, 25)
y = np.linspace(-4, 4, 25)
X, Y = np.meshgrid(x, y)
Z = X**2 + 2*Y**2
plt.contour(X, Y, Z)
plt.axis('square')
```

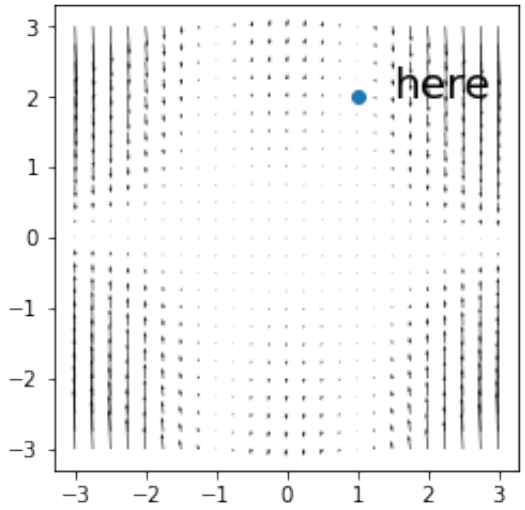
Out[27]:
```
(-4.0, 4.0, -4.0, 4.0)
```



vector field by `quiver( )`

In [28]:
```
x = np.linspace(-3, 3, 25)
y = np.linspace(-3, 3, 25)
X, Y = np.meshgrid(x, y)
# Van del Pol model
k = 5  # paramter
U = Y  # dx/dt
V = k*(1 - X**2)*Y - X  # dy/dt
plt.quiver(X, Y, U, V)
plt.axis('square')
plt.plot(1, 2, 'o')
plt.text(1.5, 2, 'here', size=20)
```

Out[28]:
```
<matplotlib.text.Text at 0x112068198>
```



In [ ]:
```
```

You can use `mplotlib.animation` library for more advanced animation and saving the result in a movie file.

In [ ]:
```
```

3D Visualization

matplotlib has a 3D toolkit called mplot3d, which can be imported as below.

In [29]:
```
from mpl_toolkits.mplot3d import Axes3D
```

You may want to use `notebook` option of the magic command to rotate the figure to pick the best viewpoint.

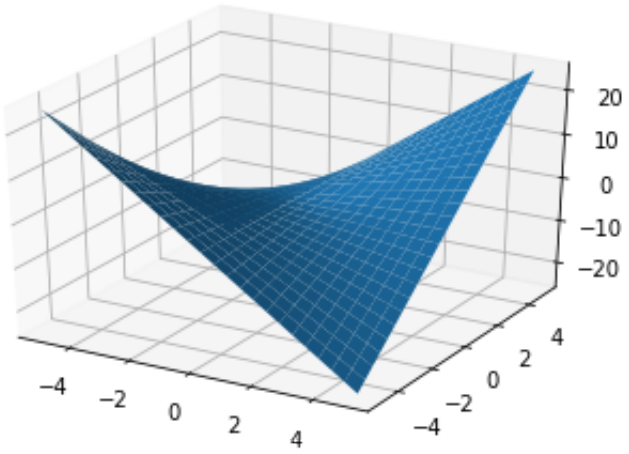
```
%matplotlib notebook
```

(You may have to run the magic command twice to take effect.)

surface plot

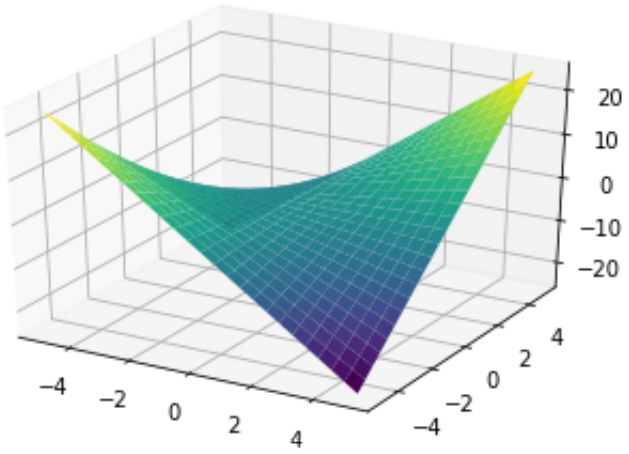
In [30]:
```
x = np.linspace(-5, 5, 25)
y = np.linspace(-5, 5, 25)
X, Y = np.meshgrid(x, y)
Z = X*Y
# create a figure and 3D axis
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
# call plot_surface method for the axis
ax.plot_surface(X, Y, Z)
```

Out[30]:
```
<mpl_toolkits.mplot3d.art3d.Poly3DCollection at 0x112299a90>
```



You can color the surface by the height.

In [31]:
```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
# map Z value to 'viridis' colormap
ax.plot_surface(X, Y, Z, cmap='viridis');
```

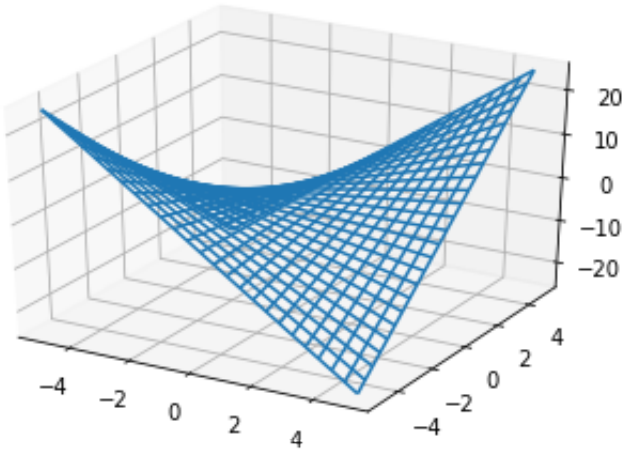


In [ ]:
```
```

surface by wire frame

In [32]:
```
# make a 3D axis in one line
ax = plt.figure().add_subplot(111, projection='3d')
# wireframe plot
ax.plot_wireframe(X, Y, Z)
```

Out[32]: <mpl_toolkits.mplot3d.art3d.Line3DCollection at 0x1120836a0>

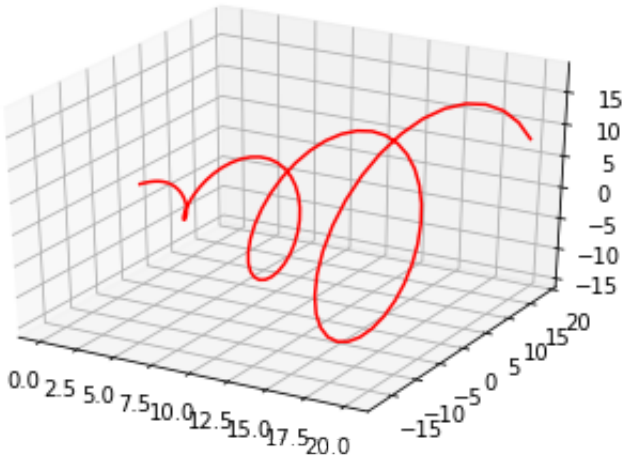


In [ ]:
```
```

lines and points in 3D

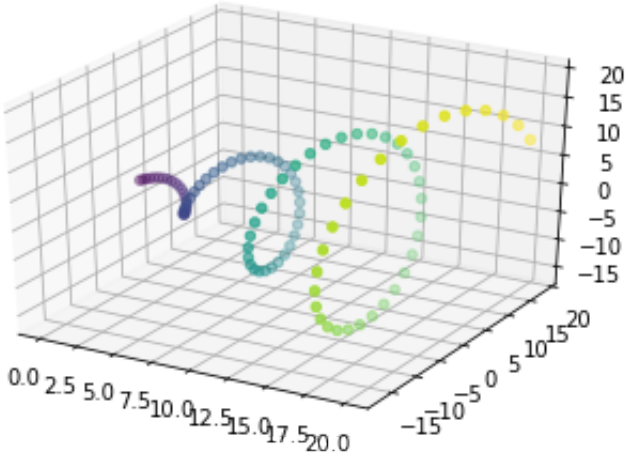
In [33]:
```
x = np.linspace(0, 20, 100)
y = x*np.sin(x)
z = x*np.cos(x)
ax = plt.figure().add_subplot(111, projection='3d')
# plot a line in 3D
ax.plot(x, y, z, 'r')
```

Out[33]: [<mpl_toolkits.mplot3d.art3d.Line3D at 0x11009ab70>]



In [34]:
```
ax = plt.figure().add_subplot(111, projection='3d')
# scatter plot with x value mapped to color
ax.scatter(x, y, z, c=x)
```

Out[34]: <mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x11208ef60>

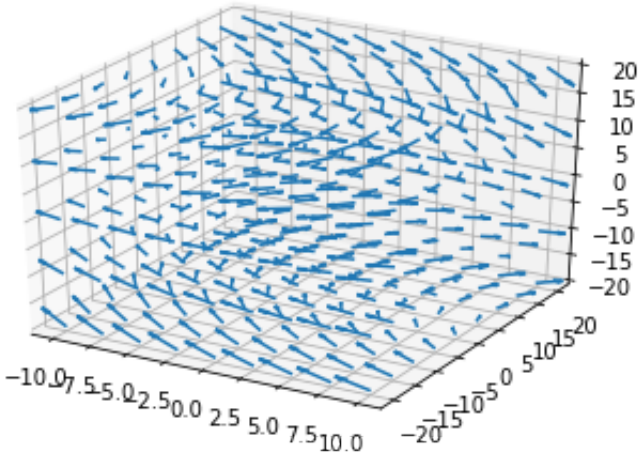


In [ ]:

3D vector field by `quiver( )`

In [35]:
```
## Lorenz equation
x = np.linspace(-10, 10, 9)
y = np.linspace(-20, 20, 9)
z = np.linspace(-20, 20, 5)
X, Y, Z = np.meshgrid(x, y, z)
#print(X)
# parameters
p, r, b = 10, 28, 8/3
U = Y  # dx/dt
V = -X # dy/dt
W = - Z   # dz/dt
ax = plt.figure().add_subplot(111, projection='3d')
ax.quiver(X, Y, Z, U, V, W, length=0.1)
```

Out[35]:
```
<mpl_toolkits.mplot3d.art3d.Line3DCollection at 0x111611e10>
```



For more advanced 3D visualization, you may want to use a specialized library like `mayavi` https://docs.enthought.com/mayavi/mayavi/ (https://docs.enthought.com/mayavi/mayavi/)

In [ ]:
```
```

Animation

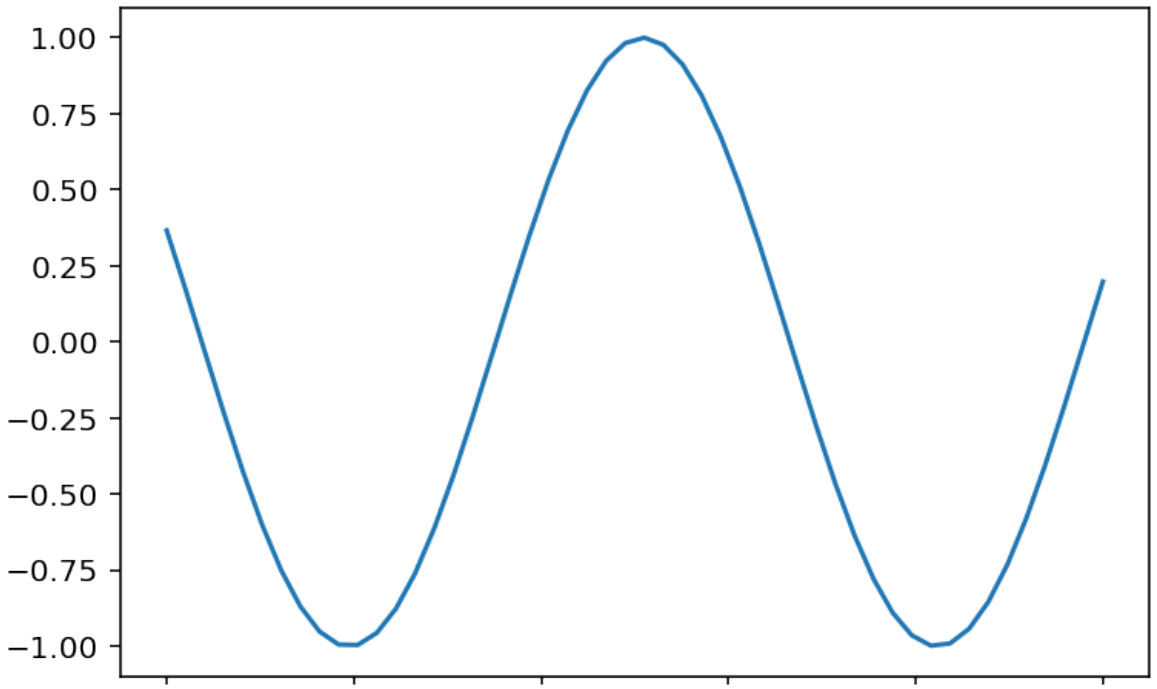
It is often helpful to visualize the result while it is computed, rather than after all computation. The simplest way is to repeat computing, drawing, and a short pause.

It does not work well with embedded figures, so open a figure window.

In [36]:
```
# use separate figure windows
%matplotlib
```

```
Using matplotlib backend: MacOSX
```

In [38]:
```
# set parameter
v = 0.2
# prepare x axis
x = np.linspace(0, 10)
for t in range(50):
    # compute the new result
    y = np.sin(x - v*t)
    # clear previous plot
    plt.cla()
    # draw a new plot
    plt.plot(x, y)
    # puase 0.01 sec
    plt.pause(0.01)
```



In [ ]:
```

```

Exercise

1) Plotting curves

a) Draw a spiral.

In [ ]:
```

```

b) Draw a "∞" shape.

In [ ]:
```

```

c) Draw a "flower-like" shape.

In [ ]:

2) Scatter plots

Let us take the famous *iris* data set.
First four columns are:

- SepalLength, SepalWidth, PetalLength, PetalWidth

The last column is the flower type:

- 1:Setosa, 2:Versicolor, 3:Virginica

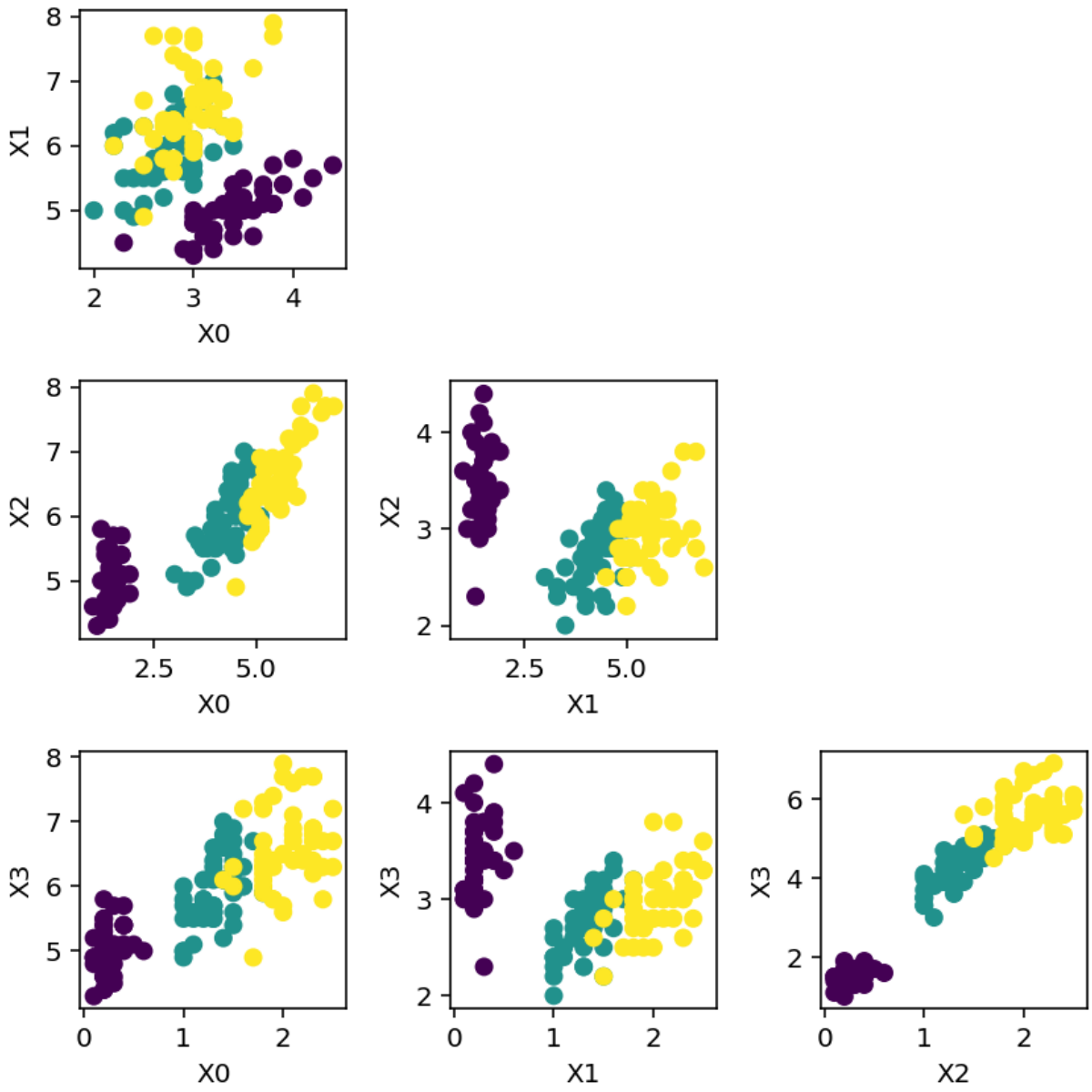
First, we'll read the data set from a text file.

In [ ]:
```
X = np.loadtxt('iris.txt', delimiter=',')
print(X.shape, X)
```

a) Make a scatter plot of the first two columns, with a distinct marker color for each flower type.

In [ ]:

b) Create a matrix of pair-wise scatter plots like this:



In [ ]:
```
plt.figure(figsize=(6, 6))  # a slightly larger area
d = 4  # data dimension
  # rows: X1 to Xd
     # columns: X0 to Xd-1

 # adjust the space between subplots
```

c) Make a `quiver` plot, representing sepal data by position, petal data by arrows, and flower type by arrow color.

In [ ]:
```
```

d) Make a 3D scatter plot of the sepal and petal data, with the 4th column represented by marker size.

In [ ]:

3) Surface plots

a) Draw a wavy surface (not just a sine curve extended in the 3rd dimension).

In [ ]:
```
x = np.linspace(-10, 10, 50)
y = np.linspace(-10, 10, 50)
X, Y = np.meshgrid(x, y)
R =
Z =
```

b) Draw the surface of a (half) cylinder.
Note that the mesh grid does not need to be square.

A half cylinder (0 <= theta <= pi), using a square mesh grid:

In [ ]:
```
r = 1
x = np.linspace(-r, r, 50)
y =
X, Y = np.meshgrid(x, y)
Z =
```

A full cylinder (0 <= theta < 2pi), using a cylindrical mesh grid:

In [ ]:
```
r = 1
x = np.linspace(-r, r, 50)
th =
X, Th = np.meshgrid(x, th)
Y =
Z =
```

c) Draw the surface of a sphere.

In [ ]:
```
r = 1
th =  # latitude
ph =   # longitude
Th, Ph = np.meshgrid(th, ph)
X =
Y =
Z =
```