Ridge Regression

```
from __future__ import division
import pandas as pd
import numpy as np
import random
#np.set_printoptions(edgeitems=3,infstr='inf', linewidth=75, nanstr='nan', precision=8, suppress=False, threshold=1000, formatter=None)

#numpy.eye(N, M=None, k=0, dtype=<class 'float'>, order='C')[source]
#Return a 2-D array with ones on the diagonal and zeros elsewhere.
#N : int           # Number of rows in the output.
#M : int, optional #Number of columns in the output. If None, defaults to N.
#k : int, optional #Index of the diagonal: 0 (the default) refers to the main diagonal,
#                  #a positive value refers to an upper diagonal, and a negative value to a lower diagonal.
print(np.eye(2, dtype=int))
print(np.eye(3, k=1))
[[1 0]
 [0 1]]
[[0. 1. 0.]
 [0. 0. 1.]
 [0. 0. 0.]]
```

In [ ]:

```
arr2D = np.array([[11 ,12,13,11], [21, 22, 23, 24], [31,32,33,34]])
print('Shape of arr2D: ', arr2D.shape)
print('2D Numpy Array\n',arr2D)
# get number of rows in 2D numpy array
print('numOfRows =', arr2D.shape[0])
# get number of columns in 2D numpy array
print('numOfColumns =', arr2D.shape[1])
print('Total Number of elements in 2D Numpy array : ', arr2D.shape[0] * arr2D.shape[1])
# Create a Numpy array from list of numbers
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
print(arr)
print('Shape of 1D numpy array : ', arr.shape)
```

```
print('length of 1D numpy array : ', arr.shape[0])
print("в Квадрате",np.power(arr, 2))
```

```
Shape of arr2D:  (3, 4)
2D Numpy Array
 [[11 12 13 11]
 [21 22 23 24]
 [31 32 33 34]]
numOfRows = 3
numOfColumns = 4
Total Number of elements in 2D Numpy array :  12
[1 2 3 4 5 6 7 8]
Shape of 1D numpy array :  (8,)
length of 1D numpy array :  8
в Квадрате [ 1  4  9 16 25 36 49 64]
```

Ridge Regression: Scikit-learn vs. direct calculation does not match for alpha > 0

```
np.linalg.solve(A.T*A + alpha * I, A.T*b)
```

yields the same as

```
np.linalg.inv(A.T*A + alpha * I)*A.T*b
```

In [3]:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
#https://gist.github.com/diogojc/1519756
import numpy as np

class RidgeRegressor(object):
    """
    Linear Least Squares Regression with Tikhonov regularization. More simply called Ridge Regression.
    We wish to fit our model so both the least squares residuals and L2 norm of the parameters are minimized.
    argmin(Theta) = ||X*Theta - y||^2 + alpha * ||Theta||^2
    A closed form solution is available.
    Theta = (X'X + G'G)^-1 X'y
    Where X contains the independent variables, y the dependent variable and G
```

```
    is matrix alpha * I, where alpha is called the regularization parameter.
    When alpha=0 the regression is equivalent to ordinary least squares.
    """
    def fit(self, X, y, alpha=0):
        """
        Fits our model to our training data.
        Arguments
        ----------
        X: mxn matrix of m examples with n independent variables
        y: dependent variable vector for m examples
        alpha: regularization parameter. A value of 0 will model using the
        ordinary least squares regression.
        """
        X = np.hstack((np.ones((X.shape[0], 1)), X))
        print("X.shape[0]",X.shape[0],"X.shape[1]",X.shape[1]) #print("np.ones((X.shape[0], 1)) ==\n",np.ones((X.shape[
0], 1)),end=''); print()
        print("alpha=",alpha)
        G = alpha * np.eye(X.shape[1])
        G[0, 0] = 0  # Don't regularize bias #print(G)
        self.params = np.dot(np.linalg.inv( np.dot(X.T,X) +np.dot(G.T,G)),
                             np.dot(X.T,y))
        print(self.params)

    def predict(self, X):
        """
        Predicts the dependent variable of new data using the model.
        The assumption here is that the new data is iid to the training data.
        Arguments
        ----------
        X: mxn matrix of m examples with n independent variables
        alpha: regularization parameter. Default of 0.
        Returns
        ----------
        Dependent variable vector for m examples
        """
        X = np.hstack((np.ones((X.shape[0], 1)), X))
        return np.dot(X, self.params)
```

In [4]:

```python
import matplotlib.pyplot as plt
from IPython.display import display, Math, Latex

# Create synthetic data
X = np.linspace(-4, 6, 31)
b0=0; b1=0; b2=1; y = b0 +b1*X +b2*X*X

# Plot synthetic data # plt.plot(X, y, label='$y = x^2$', linestyle='--',color='#00FF44')
fig, ax = plt.subplots(nrows=1,ncols=2, figsize=(12,4))
#----------1---------------
ax[0].plot(X, y,                 label=r"$y = x^2$",               linestyle='--',      color='#00FF44')
b0=4; b1=0; b2=1
ax[0].plot(X, b0 +b1*X +b2*X*X,  label=r'$y ={0} + x^2$'.format(b0,b1,b2), linestyle='dashdot', color='red')
b0=4; b1=4; b2=1
ax[0].plot(X, b0 +b1*X +b2*X*X,  label=r'$y ={0} {1:+}x + x^2$'.format(b0,b1,b2), color='blue')
b0=9; b1=-5; b2=2
ax[0].plot(X, b0 +b1*X +b2*X*X,  label=r'$y ={0} {1:+}x {2:+}x^2$'.format(b0,b1,b2), color='magenta')
ax[0].legend() #plt.legend()
ax[0].set_ylim(-1,60);ax[0].set_xlim(-5, 6)
ax[0].vlines([0],-1,100);ax[0].hlines([0],-5,56)
#----------2---------------
b0=9; b1=-5; b2=2
ax[1].plot(X, b0 +b1*X +b2*X*X,  label=r'$y ={0} {1:+}x {2:+}x^2$'.format(b0,b1,b2), color='magenta')
ax[1].legend()
ax[1].set_ylim(-1,60);ax[1].set_xlim(-5, 6)
ax[1].vlines([0],-1,60); ax[1].hlines([0],-5,15)
plt.show()
```

```
<Figure size 1200x400 with 2 Axes>
```

In [5]:

```python
# input array
in_arr1 = np.array([ 1, 2, 3] )
print ("1st Input array : \n", in_arr1)
in_arr2 = np.array([ 4, 5, 6] )
print ("2nd Input array : \n", in_arr2)
# Stacking the two arrays horizontally
out_arr = np.hstack((in_arr1, in_arr2))
```

```
print ("Output horizontally stacked array:\n ", out_arr)
#-----------------------------------
# input array
in_arr1 = np.array([[ 1, 2, 3], [ -1, -2, -3]] )
print ("1st Input array : \n", in_arr1)
in_arr2 = np.array([[ 4, 5, 6], [ -4, -5, -6]] )
print ("2nd Input array : \n", in_arr2)
# Stacking the two arrays horizontally
out_arr = np.hstack((in_arr1, in_arr2))
print ("Output stacked array :\n ", out_arr)
```

```
1st Input array :
 [1 2 3]
2nd Input array :
 [4 5 6]
Output horizontally stacked array:
  [1 2 3 4 5 6]
1st Input array :
 [[ 1  2  3]
 [-1 -2 -3]]
2nd Input array :
 [[ 4  5  6]
 [-4 -5 -6]]
Output stacked array :
  [[ 1  2  3  4  5  6]
 [-1 -2 -3 -4 -5 -6]]
```

In [6]:

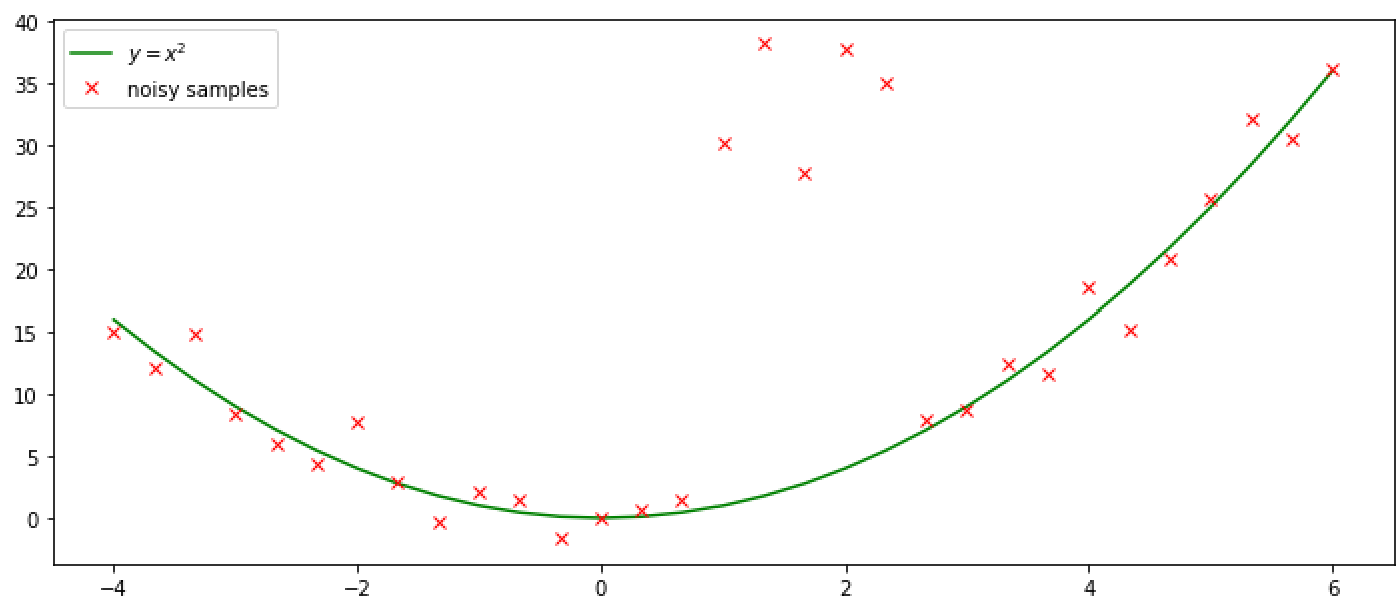
```
np.set_printoptions(formatter={'float': '{:8.2f}'.format})
#np.set_printoptions(edgeitems=3,infstr='inf', linewidth=75, nanstr='nan', precision=8, suppress=False, threshold=1000, formatter=None)
print(X,'\n'); print(y)
yhat = y + 2.5 * np.random.normal(size = len(X))
for ii in range(5):
    #print(size//2 +ii)
    yhat[len(X)//2 +ii] += 30
print(yhat)
```

```
[   -4.00    -3.67    -3.33    -3.00    -2.67    -2.33    -2.00    -1.67
    -1.33    -1.00    -0.67    -0.33     0.00     0.33     0.67     1.00
```

```
     1.33      1.67      2.00      2.33      2.67      3.00      3.33      3.67
     4.00      4.33      4.67      5.00      5.33      5.67      6.00]

[   16.00     13.44     11.11      9.00      7.11      5.44      4.00      2.78
     1.78      1.00      0.44      0.11      0.00      0.11      0.44      1.00
     1.78      2.78      4.00      5.44      7.11      9.00     11.11     13.44
    16.00     18.78     21.78     25.00     28.44     32.11     36.00]
[   15.06     12.11     14.90      8.43      5.94      4.33      7.75      2.90
    -0.34      2.09      1.44     -1.69     -0.06      0.62      1.51     30.26
    38.21     27.71     37.79     34.96      7.90      8.76     12.36     11.55
    18.50     15.13     20.90     25.73     32.21     30.57     36.09]
```

In [7]:

```python
plt.figure(figsize=(12,5))
# Plot synthetic data
plt.plot(X, y, 'g', label='$y = x^2$')
plt.plot(X, yhat, 'rx', label='noisy samples')
plt.legend()
plt.show()
```

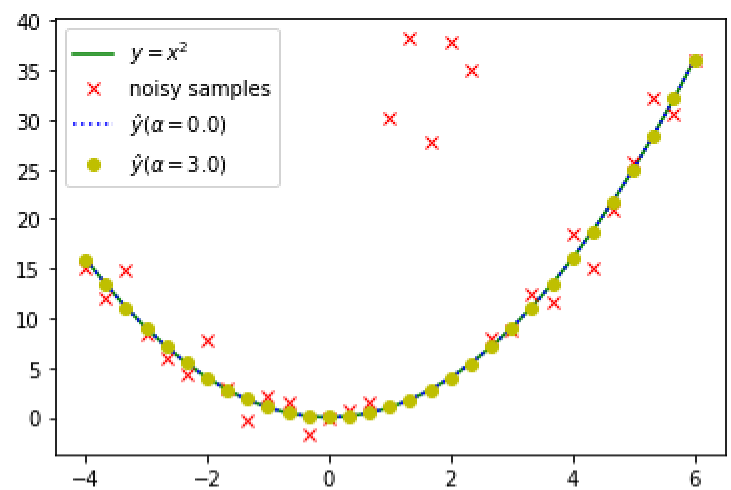


In [8]:

```
import matplotlib.pyplot as plt
from IPython.display import display, Math, Latex

np.set_printoptions(formatter={'float': '{:8.3f}'.format})
plt.plot(X, y, 'g', label='$y = x^2$')
plt.plot(X, yhat, 'rx', label='noisy samples')

# Create feature matrix
tX = np.array([X]).T
tX = np.hstack((tX, np.power(tX, 2), np.power(tX, 3)))

# Plot regressors
r = RidgeRegressor()
alpha = 0.0
r.fit(tX, y, alpha)
plt.plot(X, r.predict(tX), 'b:', label=r'$ŷ (\alpha={0:.1f})$'.format(alpha))

alpha = 3.0
r.fit(tX, y, alpha)
plt.plot(X, r.predict(tX), 'yo', label=r'$ŷ (\alpha={0:.1f})$'.format(alpha))
plt.legend()
plt.show()

#########################################################
fig, ax = plt.subplots(nrows=1,ncols=2, figsize=(14,4))
#----------1---------------
ax[0].plot(X, y, 'g', label='$y = x^2$')
ax[0].plot(X, yhat, 'rx', label='noisy samples')
#-------
alpha = 0.0
r.fit(tX, y, alpha)
ax[0].plot(X, r.predict(tX), 'b:', label=r'$ŷ (\alpha={0:.1f})$'.format(alpha))
#-------
alpha = 3.0
r.fit(tX, y, alpha)
ax[0].plot(X, r.predict(tX), 'yo', label=r'$ŷ (\alpha={0:.1f})$'.format(alpha))
ax[0].legend() #plt.legend()
ax[0].set_ylim(-5,40);ax[0].set_xlim(-5, 7) #ax[0].vlines([0],-1,100);ax[0].hlines([0],-5,56)
```

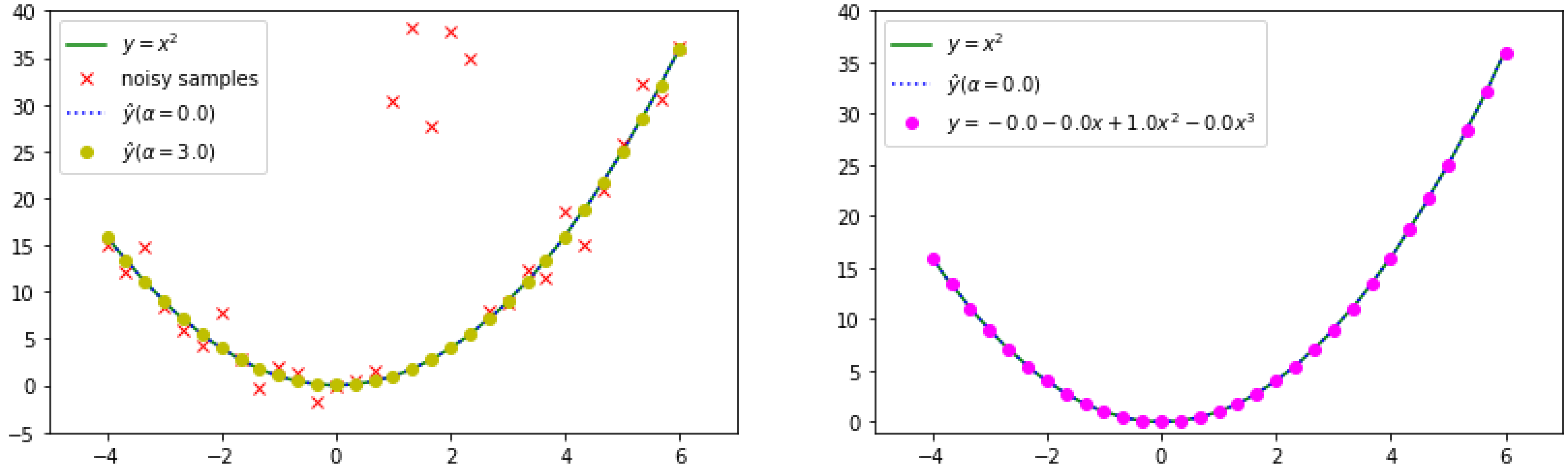
```
#----------2---------------
ax[1].plot(X, y, 'g', label='$y = x^2$')
alpha = 0.0
r.fit(tX, y, alpha)
ax[1].plot(X, r.predict(tX), 'b:', label=r'$ŷ (\alpha={0:.1f})$'.format(alpha))

b0, b1, b2, b3 = [round(sd,3) for sd in r.params]
ax[1].plot(X, b0 +b1*X +b2*X*X +b3*X*X*X, 'yo', label=r'$y ={0} {1:+}x {2:+}x^2 {3:+}x^3$'.format(b0,b1,b2,b3), color='
magenta')
ax[1].legend()
ax[1].set_ylim(-1,40);ax[1].set_xlim(-5, 7)
#ax[1].vlines([0],-1,60); ax[1].hlines([0],-5,15)
plt.show()
```

```
X.shape[0] 31 X.shape[1] 4
alpha= 0.0
[  -0.000   -0.000    1.000   -0.000]
X.shape[0] 31 X.shape[1] 4
alpha= 3.0
[   0.065   -0.015    0.990    0.002]
```




```
X.shape[0] 31 X.shape[1] 4
alpha= 0.0
```

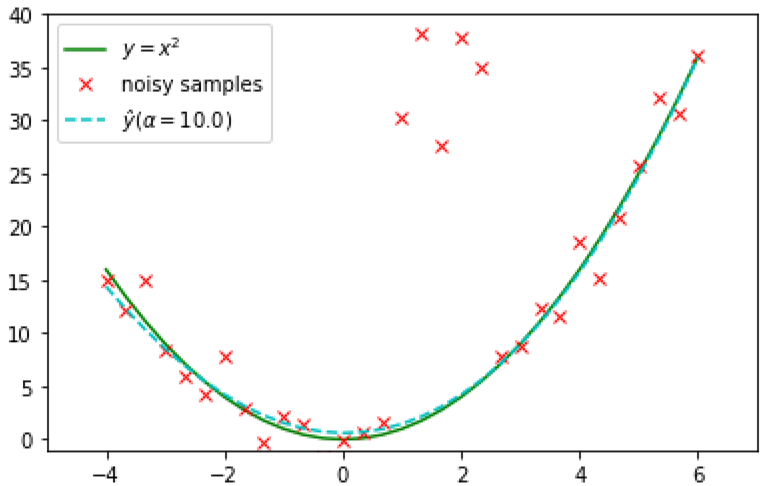
```
[  -0.000   -0.000    1.000   -0.000]
X.shape[0] 31 X.shape[1] 4
alpha= 3.0
[   0.065   -0.015    0.990    0.002]
X.shape[0] 31 X.shape[1] 4
alpha= 0.0
[  -0.000   -0.000    1.000   -0.000]
```

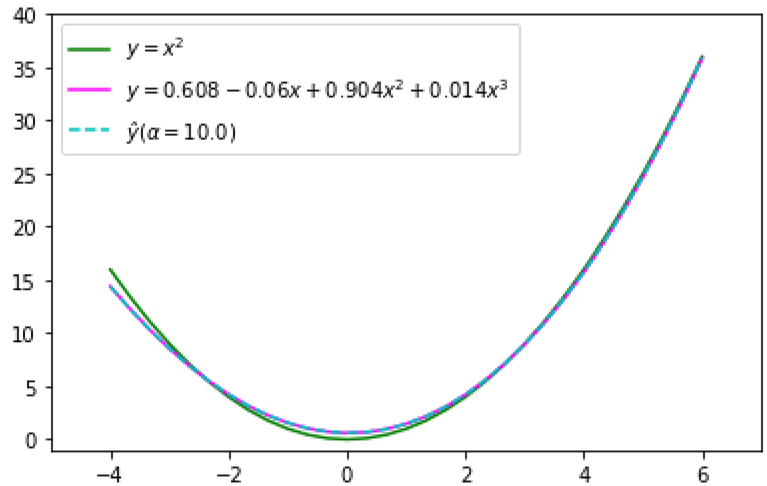


In [9]:

```
import matplotlib.pyplot as plt
from IPython.display import display, Math, Latex

fig, ax = plt.subplots(nrows=1,ncols=2, figsize=(14,4))
#----------1---------------
ax[0].plot(X, y, 'g', label='$y = x^2$')
ax[0].plot(X, yhat, 'rx', label='noisy samples')

alpha = 10.0
r.fit(tX, y, alpha)
ax[0].plot(X, r.predict(tX), 'c--', label=r'$ŷ (\alpha=%.1f)$' % alpha)
ax[0].legend() #plt.legend()
ax[0].set_ylim(-1,40);ax[0].set_xlim(-5, 7) #ax[0].vlines([0],-1,100);ax[0].hlines([0],-5,56)
#----------2---------------
```

```
ax[1].plot(X, y, 'g', label='$y = x^2$')

b0, b1, b2, b3 = [round(sd,3) for sd in r.params]
ax[1].plot(X, b0 +b1*X +b2*X*X +b3*X*X*X,  label=r'$y ={0} {1:+}x {2:+}x^2 {3:+}x^3$'.format(b0,b1,b2,b3), color='magen
ta')
ax[1].plot(X, r.predict(tX), 'c--', label=r'$\hat{y} (\alpha=%.1f)$' % alpha)
ax[1].legend()
ax[1].set_ylim(-1,40);ax[1].set_xlim(-5, 7)
#ax[1].vlines([0],-1,60); ax[1].hlines([0],-5,15)
plt.show()
X.shape[0] 31 X.shape[1] 4
alpha= 10.0
[   0.608   -0.060    0.904    0.014]
```

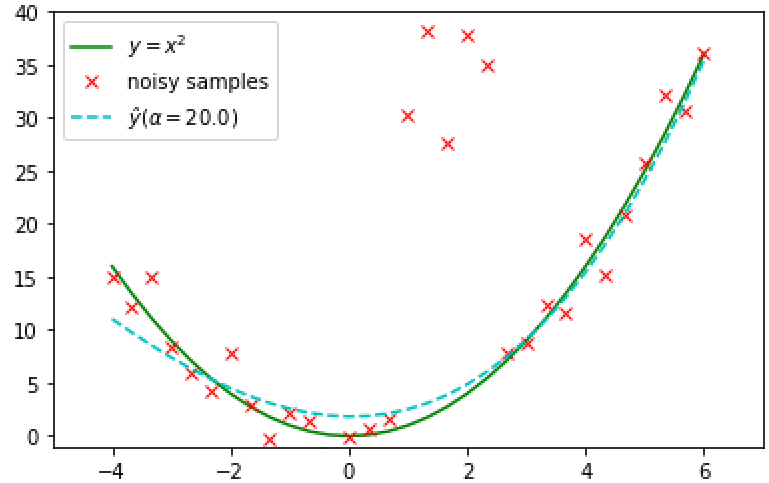


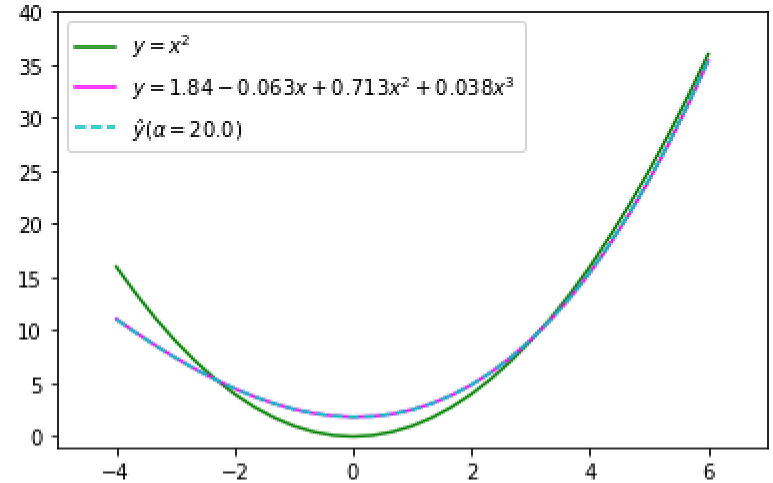


In [10]:

```
import matplotlib.pyplot as plt
from IPython.display import display, Math, Latex

fig, ax = plt.subplots(nrows=1,ncols=2, figsize=(14,4))
#----------1---------------
ax[0].plot(X, y, 'g', label='$y = x^2$')
ax[0].plot(X, yhat, 'rx', label='noisy samples')

alpha = 20.0
```

```
r.fit(tX, y, alpha)
ax[0].plot(X, r.predict(tX), 'c--', label=r'$ŷ (\alpha=%.1f)$' % alpha)
ax[0].legend() #plt.legend()
ax[0].set_ylim(-1,40);ax[0].set_xlim(-5, 7)
#----------2---------------
ax[1].plot(X, y, 'g', label='$y = x^2$')

b0, b1, b2, b3 = [round(sd,3) for sd in r.params]
ax[1].plot(X, b0 +b1*X +b2*X*X +b3*X*X*X,  label=r'$y ={0} {1:+}x {2:+}x^2 {3:+}x^3$'.format(b0,b1,b2,b3), color='magen
ta')
ax[1].plot(X, r.predict(tX), 'c--', label=r'$ŷ (\alpha=%.1f)$' % alpha)
ax[1].legend()
ax[1].set_ylim(-1,40);ax[1].set_xlim(-5, 7)
#ax[1].vlines([0],-1,60); ax[1].hlines([0],-5,15)
plt.show()
X.shape[0] 31 X.shape[1] 4
alpha= 20.0
[   1.840   -0.063    0.713    0.038]
```

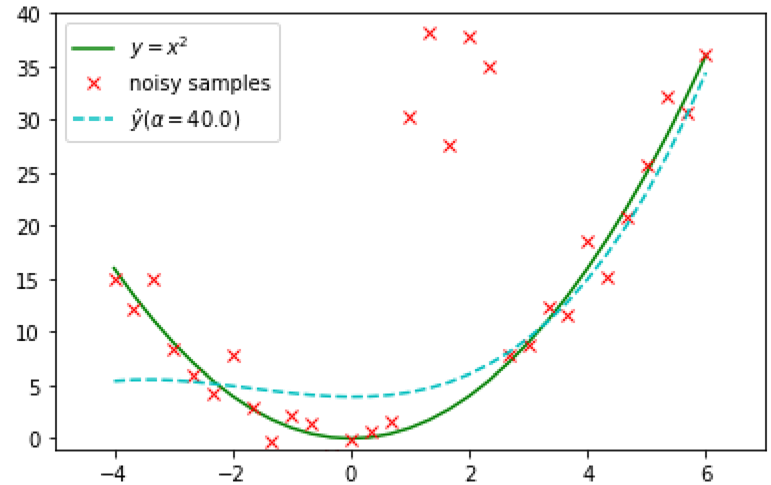


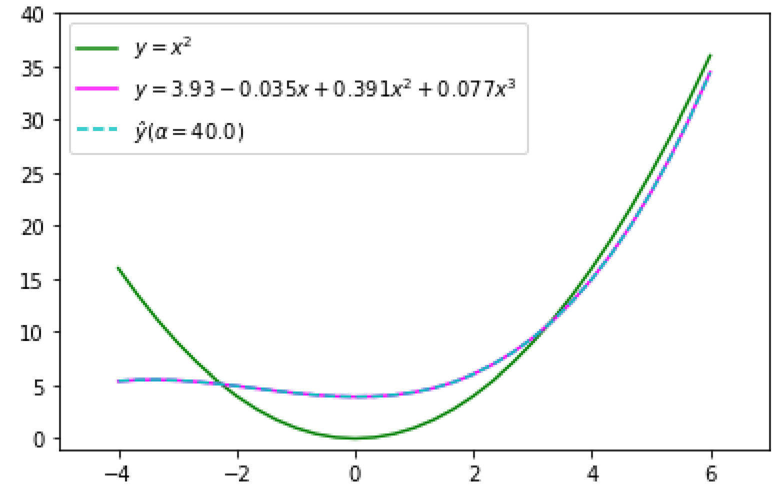


In [11]:

```
import matplotlib.pyplot as plt
from IPython.display import display, Math, Latex

fig, ax = plt.subplots(nrows=1,ncols=2, figsize=(14,4))
```

```
#----------1---------------
ax[0].plot(X, y, 'g', label='$y = x^2$')
ax[0].plot(X, yhat, 'rx', label='noisy samples')

alpha = 40.0
r.fit(tX, y, alpha)
ax[0].plot(X, r.predict(tX), 'c--', label=r'$ŷ (\alpha=%.1f)$' % alpha)
ax[0].legend() #plt.legend()
ax[0].set_ylim(-1,40);ax[0].set_xlim(-5, 7)
#----------2---------------
ax[1].plot(X, y, 'g', label='$y = x^2$')

b0, b1, b2, b3 = [round(sd,3) for sd in r.params]
ax[1].plot(X, b0 +b1*X +b2*X*X +b3*X*X*X,  label=r'$y ={0} {1:+}x {2:+}x^2 {3:+}x^3$'.format(b0,b1,b2,b3), color='magen
ta')
ax[1].plot(X, r.predict(tX), 'c--', label=r'$ŷ (\alpha=%.1f)$' % alpha)
ax[1].legend()
ax[1].set_ylim(-1,40);ax[1].set_xlim(-5, 7)
#ax[1].vlines([0],-1,60); ax[1].hlines([0],-5,15)
plt.show()
X.shape[0] 31 X.shape[1] 4
alpha= 40.0
[    3.930    -0.035     0.391     0.077]
```



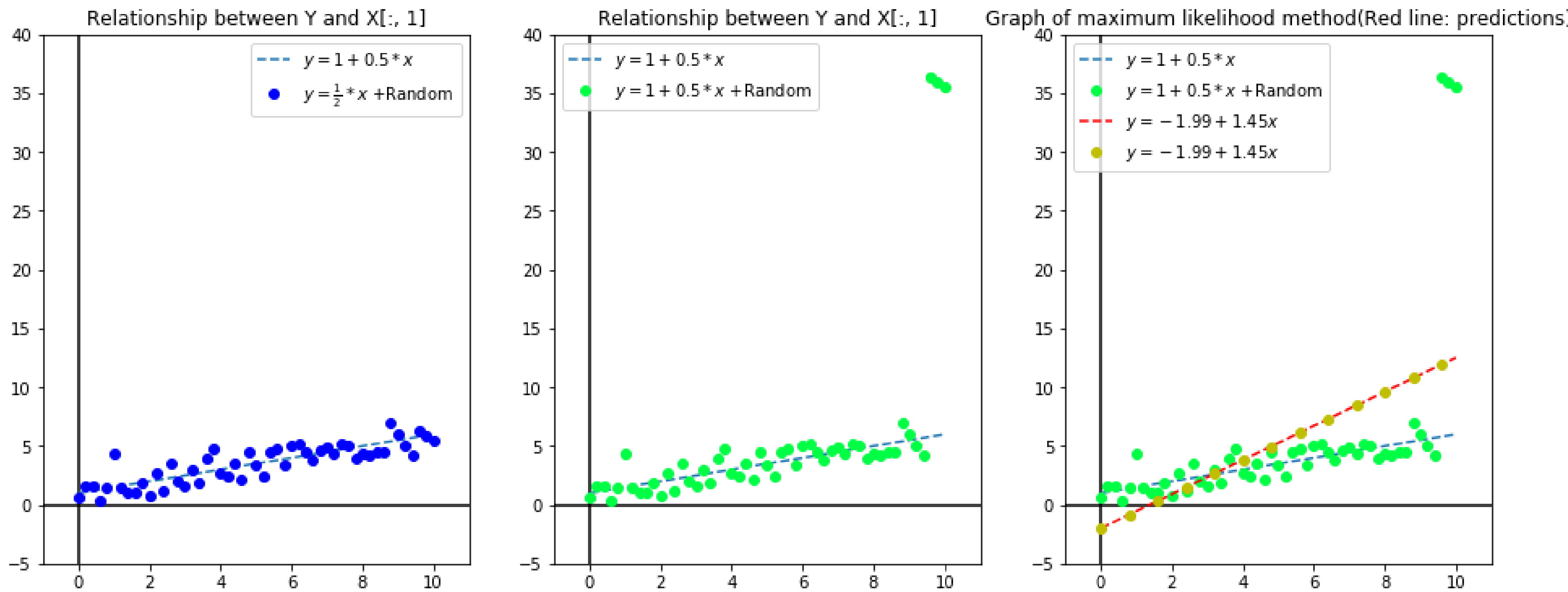


RIDGE REGRESSION AND ITS IMPLEMENTATION WITH PYTHON

In [12]:

```
random.seed(42)
np.set_printoptions(formatter={'float': '{:5.2f}'.format})
import numpy as np               #importing the numpy package with alias np
import matplotlib.pyplot as plt #importing the matplotlib.pyplot as plt

B0=1
B1=0.5
No_of_observations = 51                                #Setting number of observation = 50
X_input = np.linspace(0,10,No_of_observations)  #Generating 50 equally-spaced data points between 0 to 10.
Y_output =B0+B1*X_input + np.random.randn(No_of_observations)    #setting Y_outputi = 0.5X_inputi + some random noise
print(X_input)
print(Y_output)

fig, ax = plt.subplots(nrows=1, ncols=3, figsize=(16,6))
#----------1---------------
ax[0].plot(X_input,B0+B1*X_input , '--', label=r"$y ={0:} {1:+}*x$".format(B0,B1))
ax[0].plot(X_input, Y_output, 'bo', label=r"$y = \frac{1}{2}*x$ +Random",   color='#0000FF')
ax[0].title.set_text('Relationship between Y and X[:, 1]')
ax[0].legend()
ax[0].set_ylim(-5,40);ax[0].set_xlim(-1, 11)
ax[0].vlines([0],-10,100);ax[0].hlines([0],-5,56)
#----------2---------------
ax[1].plot(X_input,B0+B1*X_input , '--', label=r"$y ={0:} {1:+}*x$".format(B0,B1))
Y_output[-1]+=30   #setting last element of Y_output as Y_output + 30
Y_output[-2]+=30   #setting second last element of Y_output as Y_output + 30
Y_output[-3]+=30
ax[1].plot(X_input, Y_output, 'bo', label=r"$y ={0:} {1:+}*x$ +Random".format(B0,B1),   color='#00FF44')
ax[1].title.set_text('Relationship between Y and X[:, 1]')
ax[1].set_ylim(-5,40);ax[1].set_xlim(-1, 11)
ax[1].vlines([0],-10,100);ax[1].hlines([0],-5,56)
ax[1].legend()
#----------3---------------
ax[2].plot(X_input,B0+B1*X_input , '--', label=r"$y ={0:} {1:+}*x$".format(B0,B1))
ax[2].plot(X_input, Y_output, 'bo', label=r"$y ={0:} {1:+}*x$ +Random".format(B0,B1),   color='#00FF44')
```

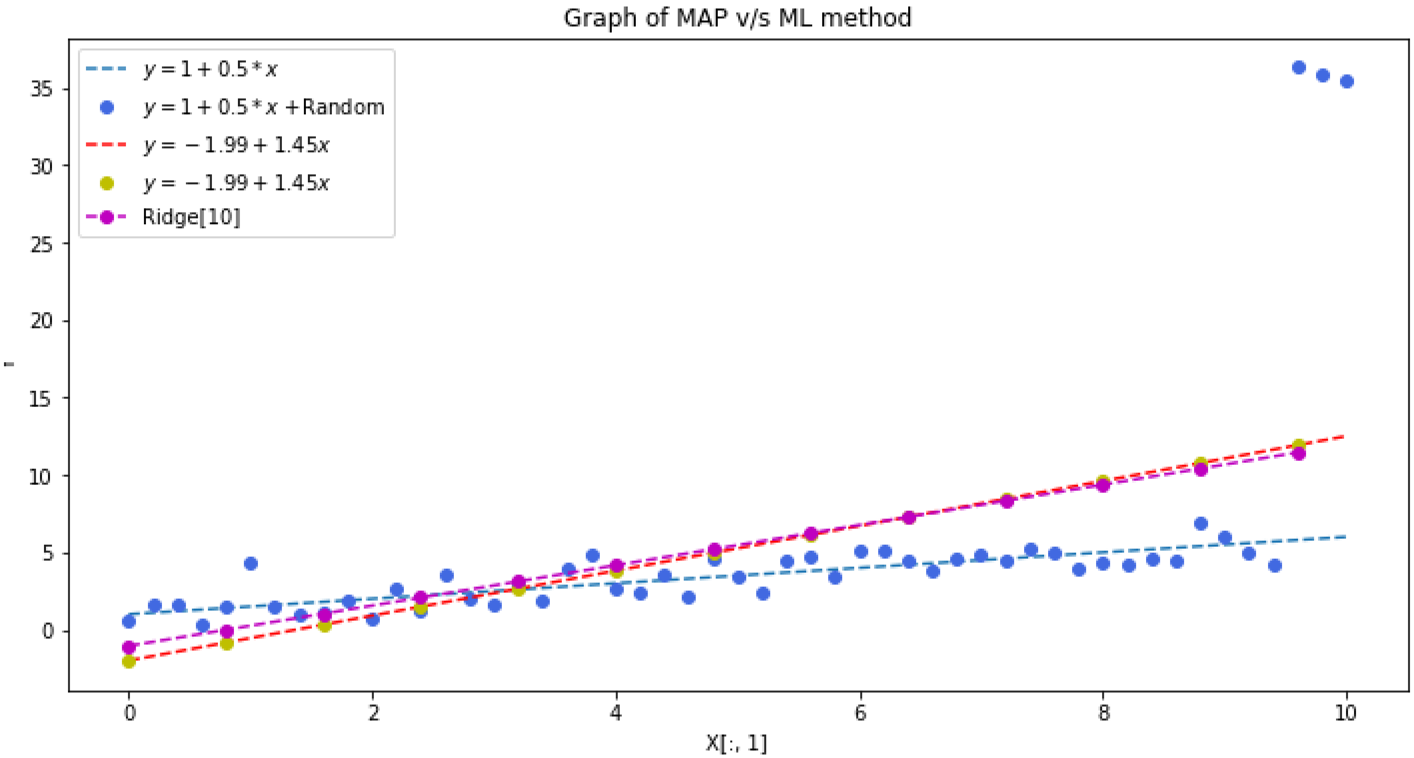
```
ax[2].title.set_text('Graph of maximum likelihood method(Red line: predictions)')
ax[2].set_ylim(-5,40);ax[2].set_xlim(-1, 11)
ax[2].vlines([0],-10,100);ax[2].hlines([0],-5,56)
##########Средняя Квадратичная #################################
X_input2 = np.vstack([np.ones(No_of_observations), X_input]).T       #appending bias data points colummn to X
for i in range(9): print(X_input2[i], end='')
#finding weights for maximum likelihood estimation
w_maxLikelihood = np.linalg.solve(np.dot(X_input2.T, X_input2), np.dot(X_input2.T, Y_output))
Y_maxLikelihood = np.dot(X_input2, w_maxLikelihood)

ll="$y={0:3.2f} {1:+3.2f}x$".format(w_maxLikelihood[0], w_maxLikelihood[1])
ax[2].plot(X_input, Y_maxLikelihood, '--', label=ll, color='red')
ax[2].plot(X_input[::4],[w_maxLikelihood[0] + w_maxLikelihood[1]*xx for xx in X_input[::4]], 'bo', label=ll, color='y'
)
ax[2].legend()

plt.show()
[ 0.00  0.20  0.40  0.60  0.80  1.00  1.20  1.40  1.60  1.80  2.00  2.20
  2.40  2.60  2.80  3.00  3.20  3.40  3.60  3.80  4.00  4.20  4.40  4.60
  4.80  5.00  5.20  5.40  5.60  5.80  6.00  6.20  6.40  6.60  6.80  7.00
  7.20  7.40  7.60  7.80  8.00  8.20  8.40  8.60  8.80  9.00  9.20  9.40
  9.60  9.80 10.00]
[ 0.60  1.56  1.60  0.32  1.49  4.31  1.49  1.01  1.08  1.83  0.77  2.65
  1.18  3.59  2.03  1.61  2.94  1.88  3.96  4.81  2.72  2.39  3.52  2.16
  4.54  3.44  2.39  4.43  4.74  3.42  5.08  5.17  4.43  3.76  4.62  4.87
  4.41  5.25  5.02  3.89  4.32  4.24  4.54  4.46  6.93  5.96  5.01  4.16
  6.33  5.93  5.50]
[ 1.00  0.00][ 1.00  0.20][ 1.00  0.40][ 1.00  0.60][ 1.00  0.80][ 1.00  1.00][ 1.00  1.20][ 1.00  1.40][ 1.00  1.60]
```



In [13]:

```
plt.figure(figsize=(12,6))
plt.plot(X_input,B0+B1*X_input , '--', label=r"$y ={0:} {1:+}*x$".format(B0,B1))
plt.plot(X_input, Y_output, 'bo', label=r"$y ={0:} {1:+}*x$ +Random".format(B0,B1), color='#4169E1') #color='royalblue'
)
ll="$y={0:3.2f} {1:+3.2f}x$".format(w_maxLikelihood[0], w_maxLikelihood[1])
plt.plot(X_input2[:,1],Y_maxLikelihood, '--', label=ll, color='red')
plt.plot(X_input[::4],[w_maxLikelihood[0] + w_maxLikelihood[1]*xx for xx in X_input[::4]], 'bo', color='y', label=ll)

L2_coeff = 10    #setting L2 regularization parameter to 1000
#Finding weights for MAP estimation
w_maxAPosterior = np.linalg.solve(np.dot(X_input2.T, X_input2)+L2_coeff*np.eye(2), np.dot(X_input2.T, Y_output))
Y_maxAPosterior = np.dot(X_input2, w_maxAPosterior) #Finding predicted Y corresponding to w_maxAPosterior
plt.plot(X_input[::4],Y_maxAPosterior[::4], '--o', color='m', label="Ridge[{0:}]".format(L2_coeff))
plt.title('Graph of MAP v/s ML method')
plt.legend()
plt.xlabel('X[:, 1]')
```

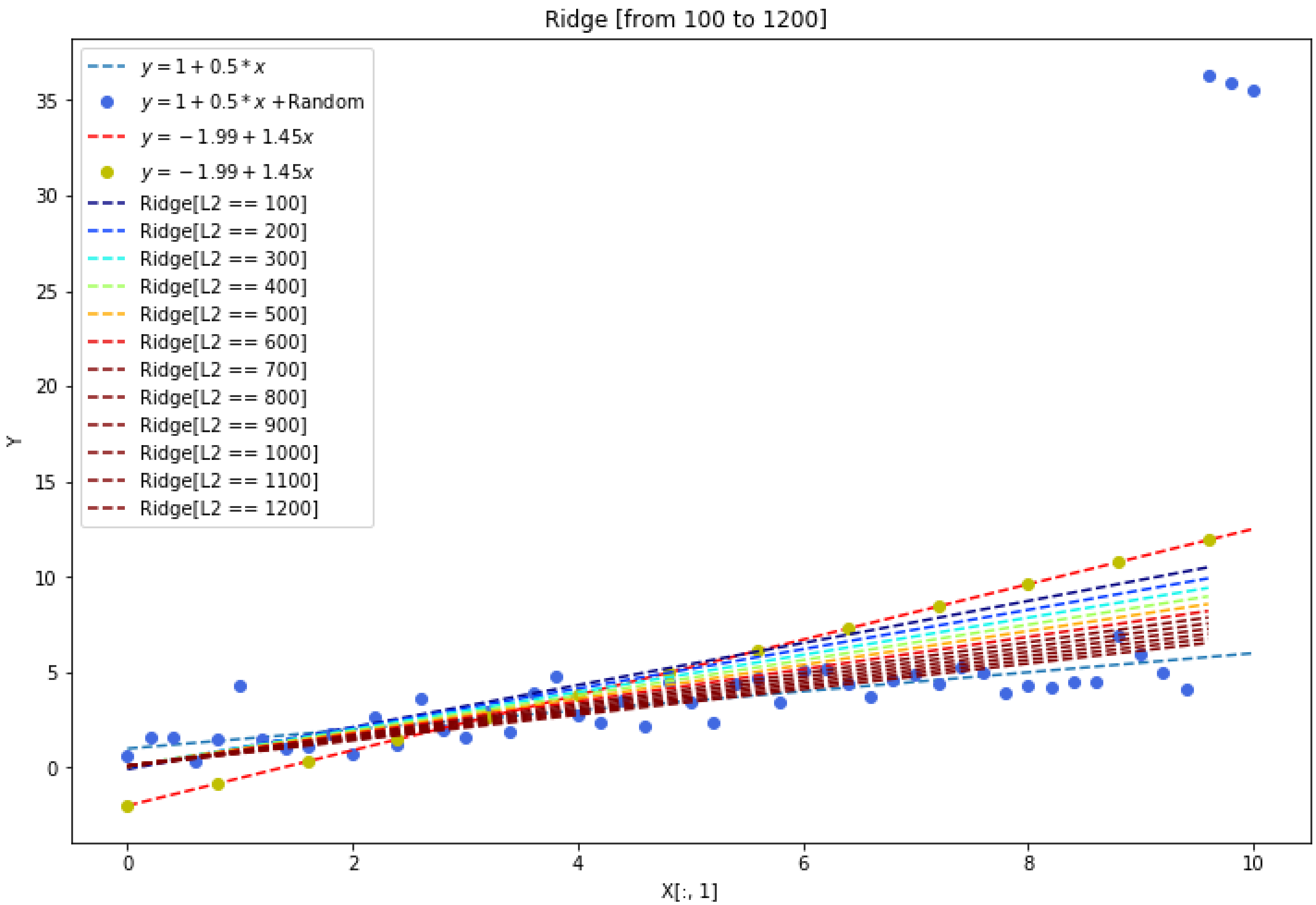
```
plt.ylabel('Y')
plt.show()
```



In [14]:

```
from matplotlib.colors import ListedColormap
import matplotlib.pylab as pl

plt.figure(figsize=(12,8))

plt.plot(X_input,B0 +B1*X_input , '--', label=r"$y ={0:} {1:+}*x$".format(B0,B1))
plt.plot(X_input, Y_output, 'bo', label=r"$y ={0:} {1:+}*x$ +Random".format(B0,B1), color='#4169E1') #color='royalblue'
)
ll="$y={0:3.2f} {1:+3.2f}x$".format(w_maxLikelihood[0], w_maxLikelihood[1])
plt.plot(X_input2[:,1],Y_maxLikelihood, '--', label=ll, color='red')
plt.plot(X_input[::4],[w_maxLikelihood[0] + w_maxLikelihood[1]*xx for xx in X_input[::4]], 'bo', color='y', label=ll)

steps = 12; rising =100
Y_maxAPosterior = []
```

```
w_maxAPosterior = []
colors = pl.cm.jet(np.linspace(0,2,steps))

for i in range(0,steps):
    L2_coeff = (i+1)*rising    #setting L2 regularization parameter to 1000
    w_maxAPosterior.append(np.linalg.solve(np.dot(X_input2.T, X_input2)+L2_coeff*np.eye(2), np.dot(X_input2.T, Y_output
)))
    Y_maxAPosterior.append(np.dot(X_input2, w_maxAPosterior[i])) #Finding predicted Y corresponding to w_maxAPosterior
    plt.plot(X_input[::4],Y_maxAPosterior[i][::4], '--', color=colors[i], label="Ridge[L2 == {0}]".format(L2_coeff))
    plt.title('Ridge [from {0:} to {1:}] '.format(1*rising,rising*steps))
    plt.legend()

plt.xlabel('X[:, 1]')
plt.ylabel('Y')
plt.show()
```

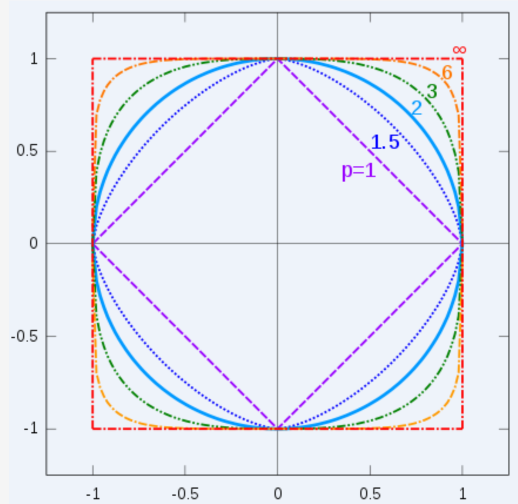


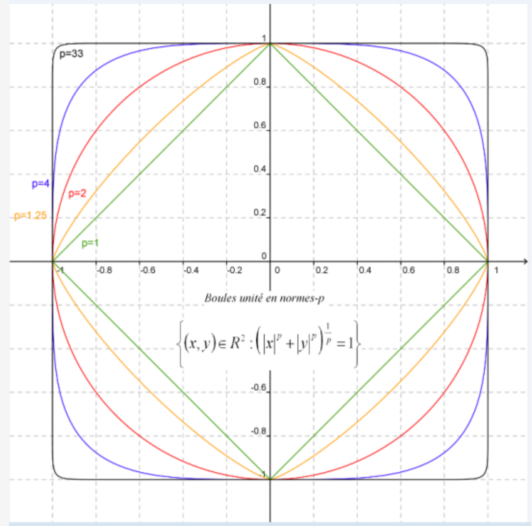
Норма (математика)

- $\|x\|_1 = \sum_i |x_i|$, что также имеет название *метрика* $L1$, норма ℓ_1 или [Расстояние_городских_кварталов| манхэттенское расстояние](). Для вектора представляет собой сумму модулей всех его элементов.
$\|x\| = |x|$

- $\|x\|_2 = \sqrt{\sum_i |x_i|^2}$, что также имеет название *метрика* $L2$, норма ℓ_2 или [евклидова норма](). Является геометрическим расстоянием между двумя точками в многомерном пространстве, вычисляемым по теореме Пифагора.
$\|x\|_2 := \sqrt{x_1^2 + \cdots + x_n^2}$

- $\|x\|_p = \sqrt[p]{\sum_{n=1}^{\infty} |x_n|^p}$ Пространства ℓ^p-norm
$\|x\|_p = \sqrt[p]{|x_1|^p + |x_2|^p + \cdots + |x_n|^p}$





[Собственный вектор]()

Со́бственный ве́ктор — понятие в [линейной алгебре](), определяемое для произвольного [линейного оператора]() как ненулевой [вектор](), применение к которому оператора даёт [коллинеарный]() вектор — тот же вектор, умноженный на некоторое скалярное значение. Скаляр, на который умножается собственный вектор под действием оператора, называется **собственным числом** (или **собственным значением**) линейного оператора, соответствующим данному собственному вектору. Одним из представлений линейного оператора является квадратная матрица, поэтому собственные векторы и собственные значения часто определяются в контексте использования таких матриц.

Понятия собственного вектора и собственного числа являются одними из ключевых в линейной алгебре, на их основе строится множество конструкций. Это связано с тем, что многие соотношения, связанные с линейными операторами, существенно упрощаются в системе координат, построенной на базисе из собственных векторов оператора. Множество собственных значений линейного оператора (спектр оператора) характеризует важные свойства оператора без привязки к какой-либо конкретной системе координат.

Понятие линейного векторного пространства не ограничивается «чисто геометрическими» векторами и обобщается на разнообразные множества объектов, таких как пространства функций (в которых действуют линейные дифференциальные и интегральные операторы). Для такого рода пространств и операторов говорят о собственных функциях операторов.

Множество всех собственных векторов линейного оператора, соответствующих данному собственному числу, дополненное нулевым вектором, называется собственным подпространством этого оператора.

$\mathbf{A}\vec{x} = \vec{y}$ Ax→=y→

$\mathbf{A}\vec{x} = \lambda\vec{x}$ Ax→=λx→

Предположим что существует такой вектор не равный $\vec{x} \neq \vec{0}$ тогда:

$$\mathbf{A}\vec{x} - \lambda\vec{x} = \vec{0}$$

$$\mathbf{A}\vec{x} - \lambda\mathbf{I}_n\vec{x} = \vec{0}$$

$$(\mathbf{A} - \lambda\mathbf{I}_n)\vec{x} = \vec{0}$$

Если имеется собственный вектор не равный 0 то тогда:

$$|\mathbf{A} - \lambda\mathbf{I}| = 0$$

Пусть имеется квадратная матрица AA разменостью $n \times n$:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{d1} & a_{d2} & a_{d3} & \cdots & a_{dn} \end{bmatrix}$$

собственное значение со связанным **собственным вектором**:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \lambda \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

Тогда эту систему можно преобразовать к:

$$\begin{bmatrix} a_{11} - \lambda & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} - \lambda & a_{23} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} - \lambda \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

В более компактной форме может быть записана как:

$$(\mathbf{A} - \lambda \cdot \mathbf{I})\mathbf{X} = 0$$

Ridge regression and L2 regularization - Introduction

Ordinary Least Squares

$$L_1(w) = \sum_{i=1}^{m}(y_i - \mathbf{x_i}^T w)^2 = (y - Xw)^T(y - Xw)$$

$$\hat{w}_{ridge} = argmin_{w\in\mathbb{R}^n} \sum_{i=1}^{m}(y_i - \mathbf{x_i}^T w)^2 ==$$

Берём производную и приравниваем к 0, в точке перегиба самый минимум:

$$\boxed{\frac{\partial L_1(w)}{\partial w} = \left[(b + wx - y)^2\right]\Big|_w' = 0}$$

$$2(b + wx - y)\cdot x = 0$$
$$(b + wx - y)\cdot x = bx + w\cdot x^2 - y\cdot x = 0$$
$$w\cdot x^2 = y\cdot x - bx$$
$$w\cdot x^2 = x(y - b)$$
$$w = \frac{x(y-b)}{(x^2)} = (x^2)^{-1}\cdot x(y-b) \iff \text{убрали b}$$
$$\mathbf{w} = \boxed{(\mathbf{X^T X})^{-1}\mathbf{X^T y}}$$

Ridge regression Добавляется маленькая добавка к матрице по диагонали чтобы eigen-values(собственное число) и матрица стала в более стабильном состоянии:

$$\hat{\mathbf{w}}_{ridge} = argmin_{\mathbf{w}\in\mathbb{R}^n} \sum_{i=1}^{m}(y_i - \mathbf{x_i}^T\mathbf{w})^2 + \lambda\sum_{j=1}^{n}\mathbf{w}^2$$

Quadratic cost function изменим чуть-чуть значение $\lambda\lambda$ - возведём в квадрат для облегчения вычислений:

$$L_2 = (b + wx - y)^2 + \lambda w^2$$

$$minimaze\|b + Xw - y\|_2^2 + \|\Gamma x\|^2 ==$$

$\Gamma = \alpha I$ и здесь $\alpha = \sqrt{\lambda}\frac{y}{x} = \sqrt{\lambda}\cdot w$

$$== \sum_{i=1}^{n}(\mathbf{x}_i^T\mathbf{w} - y_i)^2 + \lambda\|\mathbf{w}\|^2 = \sum_{i=1}^{n}(\mathbf{x}_i^T\mathbf{w} - y_i)^2 + \lambda\mathbf{w}^T\mathbf{w}$$

$$\|\mathbf{w}\|^2 = \sqrt{w_1^2 + w_2^2 + \cdots + w_n^2} = \mathbf{w}^T\mathbf{w}$$

Берём производную и приравниваем к 0, в точке перегиба самый минимум:

$$\boxed{\frac{\partial L_2(w)}{\partial w} = \left[(b + wx - y)^2 + \lambda w^2\right]\Big|_w' = 0}$$

$$2(b + wx - y)\cdot x + 2\lambda\cdot w = 0$$
$$(b + wx - y)\cdot x + \lambda\cdot w = bx + w\cdot x^2 - y\cdot x + \lambda\cdot w = 0$$
$$w\cdot x^2 + \lambda\cdot w = y\cdot x - bx$$
$$w(x^2 + \lambda) = x(y - b)$$
$$w = \frac{x(y-b)}{(x^2+\lambda)} = (x^2 + \lambda)^{-1}\cdot x(y-b) \iff \text{убрали b}$$

$$\boxed{\mathbf{w} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}}$$

$$\boxed{L_2(\mathbf{w}) = \sum_{i=1}^{m}(y_i - \mathbf{x_i}^T\mathbf{w})^2 + \lambda^2\sum_{j=1}^{n}\mathbf{w}^2}$$

$$= (\mathbf{y} - \mathbf{X}\mathbf{w})^\mathbf{T}(\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda^2\mathbf{w}^\mathbf{T}\mathbf{w}$$

$$= (\mathbf{y} - \mathbf{X}\mathbf{w})^\mathbf{T}(\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda^2\mathbf{w}^\mathbf{T}\mathbf{w}$$

$$= (\mathbf{y}^\mathbf{T} - \mathbf{w}^\mathbf{T}\mathbf{X}^\mathbf{T})(\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda^2\mathbf{w}^\mathbf{T}\mathbf{w}$$

$$= \mathbf{y}^\mathbf{T}\mathbf{y} - \mathbf{y}^\mathbf{T}\mathbf{X}\mathbf{w} - \mathbf{w}^\mathbf{T}\mathbf{X}^\mathbf{T}\mathbf{y} + \mathbf{w}^\mathbf{T}\mathbf{X}^\mathbf{T}\mathbf{X}\mathbf{w} + \lambda^2\mathbf{w}^\mathbf{T}\mathbf{w}$$

$$= \mathbf{y}^\mathbf{T}\mathbf{y} - 2\mathbf{w}^\mathbf{T}\mathbf{X}^\mathbf{T}\mathbf{y} + \mathbf{w}^\mathbf{T}\mathbf{X}^\mathbf{T}\mathbf{X}\mathbf{w} + \lambda^2\mathbf{w}^\mathbf{T}\mathbf{w}$$

$$\frac{\partial L_2(w)}{\partial w} = -2\mathbf{X}^\mathbf{T}\mathbf{y} + 2\mathbf{w}\mathbf{X}^\mathbf{T}\mathbf{X} + 2\lambda^2\mathbf{w} = 0$$

$$-\mathbf{X}^\mathbf{T}\mathbf{y} + \mathbf{w}\mathbf{X}^\mathbf{T}\mathbf{X} + \lambda^2\mathbf{w} = 0$$

$$\mathbf{w}\mathbf{X}^\mathbf{T}\mathbf{X} + \lambda^2\mathbf{I}\mathbf{w} = \mathbf{X}^\mathbf{T}\mathbf{y}$$

$$\mathbf{w}(\mathbf{X}^\mathbf{T}\mathbf{X} + \lambda^2\mathbf{I}) = \mathbf{X}^\mathbf{T}\mathbf{y}$$

$$\boxed{\mathbf{w} = (\mathbf{X}^\mathbf{T}\mathbf{X} + \lambda^2\mathbf{I})^{-1}\mathbf{X}^\mathbf{T}\mathbf{y}}$$

$$minimize\|Ax - y\|^2 + \lambda^2\|x\|^2 \rightarrow \text{тоже самое что и} \rightarrow minimize\left\|\begin{bmatrix} A \\ \lambda I \end{bmatrix} x - \begin{bmatrix} y \\ 0 \end{bmatrix}\right\|^2 :$$

[Factor selection and pivoted QR]

[Calculation of the squared Euclidean norm]

$$\left\|\begin{bmatrix} A \\ \lambda I \end{bmatrix} x - \begin{bmatrix} y \\ 0 \end{bmatrix}\right\|^2 = \left\|\begin{bmatrix} Ax \\ \lambda x \end{bmatrix} - \begin{bmatrix} y \\ 0 \end{bmatrix}\right\|^2 = \left(\begin{bmatrix} Ax \\ \lambda x \end{bmatrix} - \begin{bmatrix} y \\ 0 \end{bmatrix}\right)^T \cdot \left(\begin{bmatrix} Ax \\ \lambda x \end{bmatrix} - \begin{bmatrix} y \\ 0 \end{bmatrix}\right) ==$$

$$== \left\|\begin{bmatrix} Ax \\ \lambda x \end{bmatrix}\right\|^2 + \left\|\begin{bmatrix} y \\ 0 \end{bmatrix}\right\|^2 - \left(\begin{bmatrix} Ax \\ \lambda x \end{bmatrix}\right)^T \begin{bmatrix} y \\ 0 \end{bmatrix} - \begin{bmatrix} y \\ 0 \end{bmatrix}^T \left(\begin{bmatrix} Ax \\ \lambda x \end{bmatrix}\right) \Longrightarrow$$

$$u^T \cdot v == v^T \cdot u \rightarrow \left(\begin{bmatrix} A \\ \lambda I \end{bmatrix} x\right)^T \begin{bmatrix} y \\ 0 \end{bmatrix} == \begin{bmatrix} y \\ 0 \end{bmatrix}^T \left(\begin{bmatrix} A \\ \lambda I \end{bmatrix} x\right)$$

$$\Longrightarrow \left\|\begin{bmatrix} Ax \\ \lambda x \end{bmatrix}\right\|^2 + \left\|\begin{bmatrix} y \\ 0 \end{bmatrix}\right\|^2 - 2 \cdot \left(\begin{bmatrix} Ax \\ \lambda x \end{bmatrix}\right)^T \begin{bmatrix} y \\ 0 \end{bmatrix} == \left\|\begin{bmatrix} Ax \\ \lambda x \end{bmatrix}\right\|^2 + y^2 - 2 \cdot \left(\begin{bmatrix} Ax \\ \lambda x \end{bmatrix}\right)^T \begin{bmatrix} y \\ 0 \end{bmatrix} ==$$

$$== \left\|\begin{bmatrix} Ax \\ \lambda x \end{bmatrix}\right\|^2 + y^2 - 2 \cdot \left(\begin{bmatrix} Ax \\ \lambda x \end{bmatrix}\right)^T \begin{bmatrix} y \\ 0 \end{bmatrix} = \left(\begin{bmatrix} Ax \\ \lambda x \end{bmatrix}\right)^T \left(\begin{bmatrix} Ax \\ \lambda x \end{bmatrix}\right) + y^2 - 2 \cdot Axy ==$$

$$== (Ax)^2 + (\lambda x)^2 + y^2 - 2Axy = (Ax)^2 - 2(Ax)y + y^2 + (\lambda x)^2 \Longrightarrow$$

$$\boxed{\|Ax - y\|^2 + \lambda^2 \|x\|^2}$$

Regularized Cost Function

$$h_\theta(x^{(i)}) = \theta_0 x_0^{(i)} + \theta_1 x_1^{(i)} + \ldots + \theta_n x_n^{(i)}$$

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

$$J(\theta) = \frac{1}{2m}(X\theta - y)^T(X\theta - y) + \frac{\lambda}{2m}\theta^T\theta(\text{vectorized version})$$

Why divide by $2m$2m ?

- The $1/m$ is to "average" the squared error over the number of components so that the number of components doesn't affect the function.

Why do we have to divide by 2 ?

- The ½ is because when you take the derivative of the cost function, that is used in updating the parameters during gradient descent, that 2 in the power get cancelled with the 1/2 multiplier, thus the derivation is cleaner. These techniques are or somewhat similar are widely used in math in order ***"To make the derivations mathematically more convenient"***. You can simply remove the multiplier, see here for example, and expect the same result.

Where does 1/2m came from ?

```
def costFunctionReg(X,y,theta,lamda = 10):
    m = len(y); J = 0;

    h = X @ theta
    J = float((1./(2*m)) * (h - y).T @ (h - y)) + (lamda/(2*m)) * np.sum(np.square(theta))
    return(J)
```

@ PEP 465 -- A dedicated infix operator for matrix multiplication

```
#there are two different ways we might want to define multiplication.
#One is elementwise multiplication:
[[1, 2],      [[11, 12],      [[1 * 11, 2 * 12],
 [3, 4]]   x    [13, 14]]   =   [3 * 13, 4 * 14]]
#and the other is matrix multiplication
[[1, 2],      [[11, 12],      [[1 * 11 + 2 * 13, 1 * 12 + 2 * 14],     [[37, 40],
 [3, 4]]   x    [13, 14]]   =   [3 * 11 + 4 * 13, 3 * 12 + 4 * 14]]   =   [85, 92]]
```

The Python Data Model specifies that the `@` operator invokes `__matmul__` and `__rmatmul__`.

numpy.dot(a, b, out=None) Dot product of two arrays. Specifically,

- If both `a` and `b` are **1-D** arrays, it is inner product of vectors (without complex conjugation).
- If both `a` and `b` are **2-D** arrays, it is matrix multiplication, but using matmul or `a @ b` is preferred.
- If either `a` or `b` is **0-D** (**scalar**), it is equivalent to multiply and using `numpy.multiply(a, b)` or `a*b` is preferred.
- If `a` is an **N-D** array and `b` is a **1-D** array, it is a sum product over the last axis of `a` and `b`.
- If `a` is an **N-D** array and `b` is an **M-D** array (where **M>=2**), it is a sum product over the last axis of a and the second-to-last axis of `b`:

```
dot(a, b)[i,j,k,m] = sum(a[i,j,:] * b[k,:,m])
```

In [15]:

```
import numpy as np
A = np.matrix('1 2; 3 4')
B = np.matrix('11 12; 13 14')
C = A @ B; D = np.dot(A,B)
print(A); print(B)
print(C,C.shape)
print(D,D.shape)
```

```
[[1 2]
 [3 4]]
[[11 12]
 [13 14]]
[[37 40]
 [85 92]] (2, 2)
[[37 40]
 [85 92]] (2, 2)
```

Gradient

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{m} X^T (X\theta - y) + \frac{\lambda}{m}\theta$$

Gradient descent (vectorized)

$$\theta^{(t+1)} := \theta^{(t)} - \alpha \frac{\partial}{\partial \theta} J(\theta^{(t)})$$

```python
def gradient_descent_reg(X,y,theta,alpha = 0.0005,lamda = 10,num_iters=1000):
    #Initialisation of useful values
    m = np.size(y)
    J_history = np.zeros(num_iters)
    theta_0_hist, theta_1_hist = [], [] #Used for three D plot

    for i in range(num_iters):
        #Hypothesis function
        h = np.dot(X,theta)
        #Grad function in vectorized form
        theta = theta - alpha * (1/m)* (  (X.T @ (h-y)) + lamda * theta )
        #Cost function in vectorized form
        J_history[i] = costFunctionReg(X,y,theta,lamda)
        #Calculate the cost for each iteration(used to plot convergence)
        theta_0_hist.append(theta[0,0])
        theta_1_hist.append(theta[1,0])
    return theta ,J_history, theta_0_hist, theta_1_hist
```

Closed form solution

$$\boxed{\theta = (X^T X + \lambda I)^{-1} X^T Y}$$

```python
def closed_form_reg_solution(X,y,lamda = 10):
    '''Closed form solution for ridge regression'''
    m,n = X.shape
    I = np.eye((n))
    return (np.linalg.inv(X.T@X + lamda*I) @ X.T@y)[:,0]
```

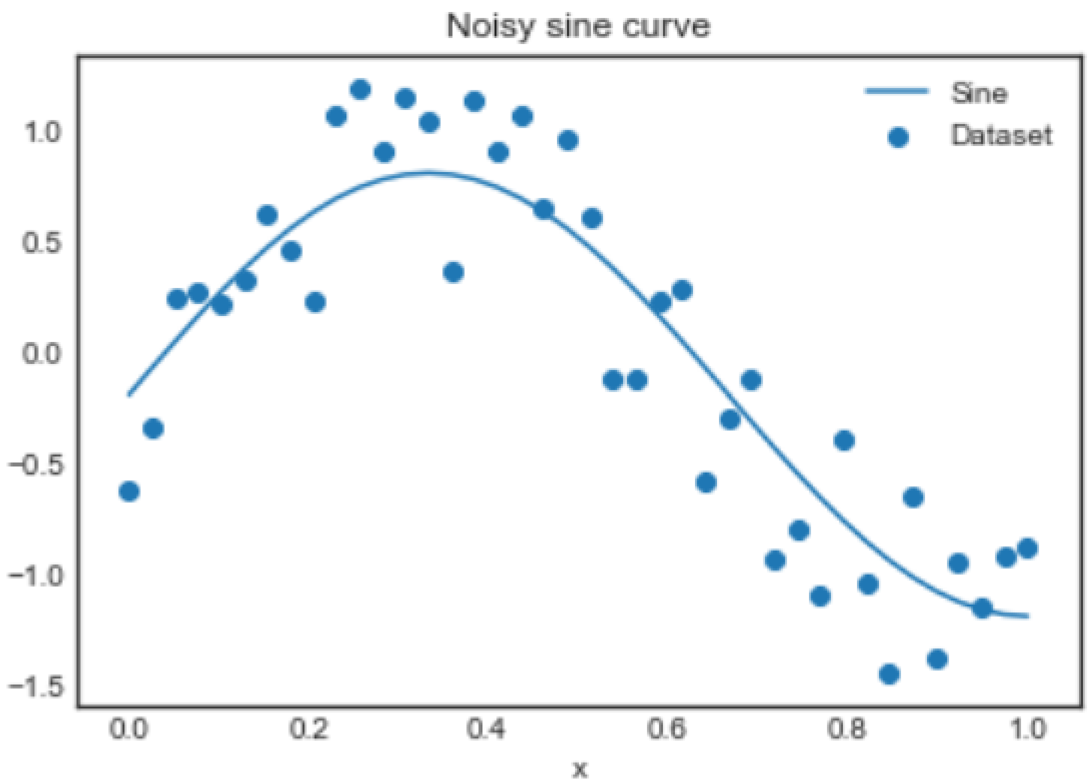
In [16]:

```python
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
#from sklearn import linear_model

%matplotlib inline
plt.style.use('seaborn-white')
```

In [17]:

```python
#Generating sine curve and uniform noise
x = np.linspace(0,1,40)
y = np.sin(x * 1.5 * np.pi )
```

```python
noise = 1*np.random.uniform(  size = 40)
y_noise = (y + noise).reshape(-1,1)

#Centering the y data
y_noise = y_noise - y_noise.mean()

#Design matrix is x, x^2
X = np.vstack((2*x,x**2)).T

#Nornalizing the design matrix to facilitate visualization
X = X / np.linalg.norm(X,axis = 0)

#Plotting the result
plt.scatter(x,y_noise, label = 'Dataset')
plt.plot(x,y - y.mean(),label = 'Sine')
plt.title('Noisy sine curve')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```



In [18]:

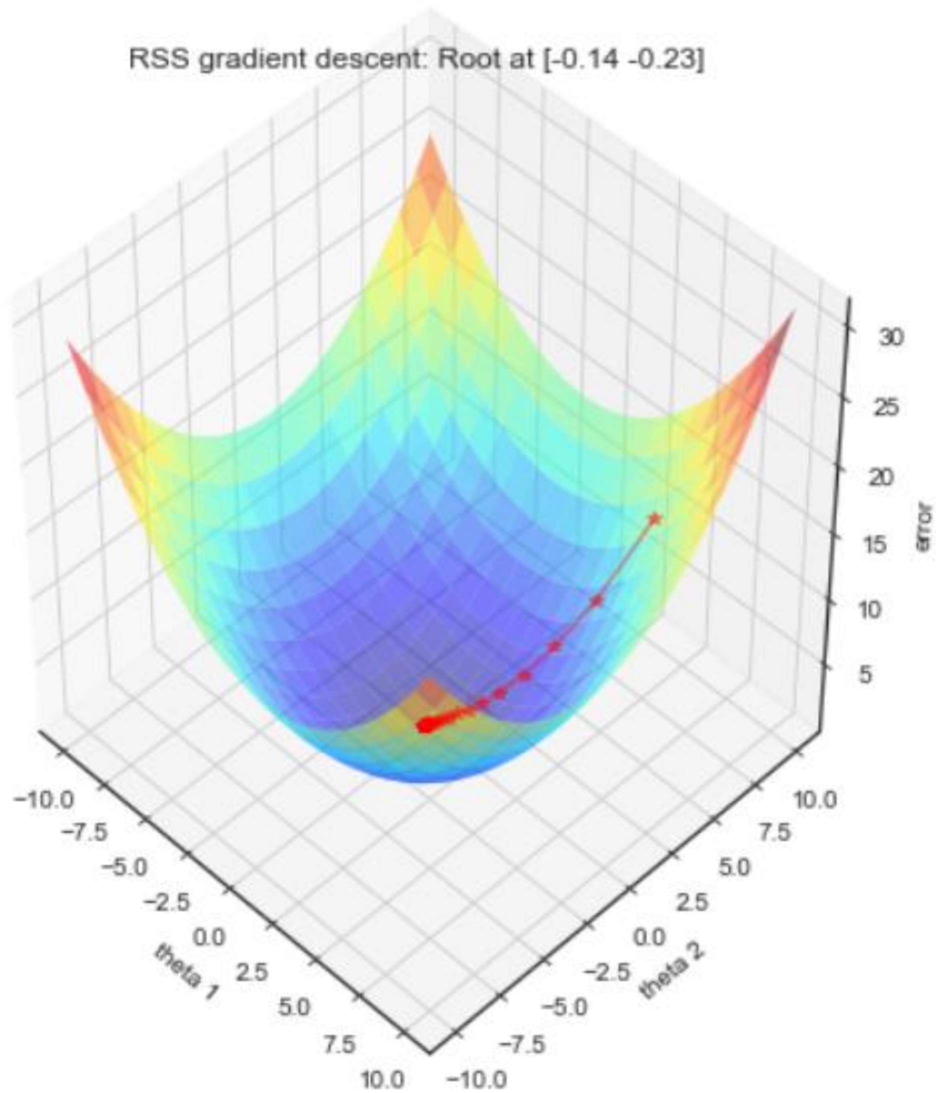
```
#Setup of meshgrid of theta values
T0, T1 = np.meshgrid(np.linspace(7,18,100),np.linspace(-18,-9,100))

def costFunctionReg(X,y,theta,lamda = 10):
    m = len(y); J = 0;

    h = X@theta
    J = float((1./(2*m)) *(h - y).T@(h - y)) +(lamda/(2*m)) *np.sum(np.square(theta))
    return(J)

def gradient_descent_reg(X,y,theta,alpha = 0.0005,lamda = 10,num_iters=1000):
    #Initialisation of useful values
    m = np.size(y)
    J_history = np.zeros(num_iters)
    theta_0_hist, theta_1_hist = [], [] #Used for three D plot
```

```
    for i in range(num_iters):
        #Hypothesis function
        h = np.dot(X,theta)
        #Grad function in vectorized form
        theta = theta - alpha * (1/m)* (  (X.T @ (h-y)) + lamda * theta )
        #Cost function in vectorized form
        J_history[i] = costFunctionReg(X,y,theta,lamda)
        #Calculate the cost for each iteration(used to plot convergence)
        theta_0_hist.append(theta[0,0])
        theta_1_hist.append(theta[1,0])
    return theta ,J_history, theta_0_hist, theta_1_hist

l = 10

#Setup of meshgrid of theta values
T1, T2 = np.meshgrid(np.linspace(-10,10,100),np.linspace(-10,10,100))

#Computing the cost function for each theta combination
zs = np.array([costFunctionReg(X, y_noise.reshape(-1,1),np.array([t1,t2]).reshape(-1,1),l)
               for t1, t2 in zip(np.ravel(T1), np.ravel(T2)) ]
             )
#Reshaping the cost values
Z = zs.reshape(T1.shape)


#Computing the gradient descent
theta_result_reg,J_history_reg, theta_0, theta_1 = gradient_descent_reg(X,y_noise,np.array([7.,10.]).reshape(-1,1),
                                                                          0.8,l,num_iters=5000)
```

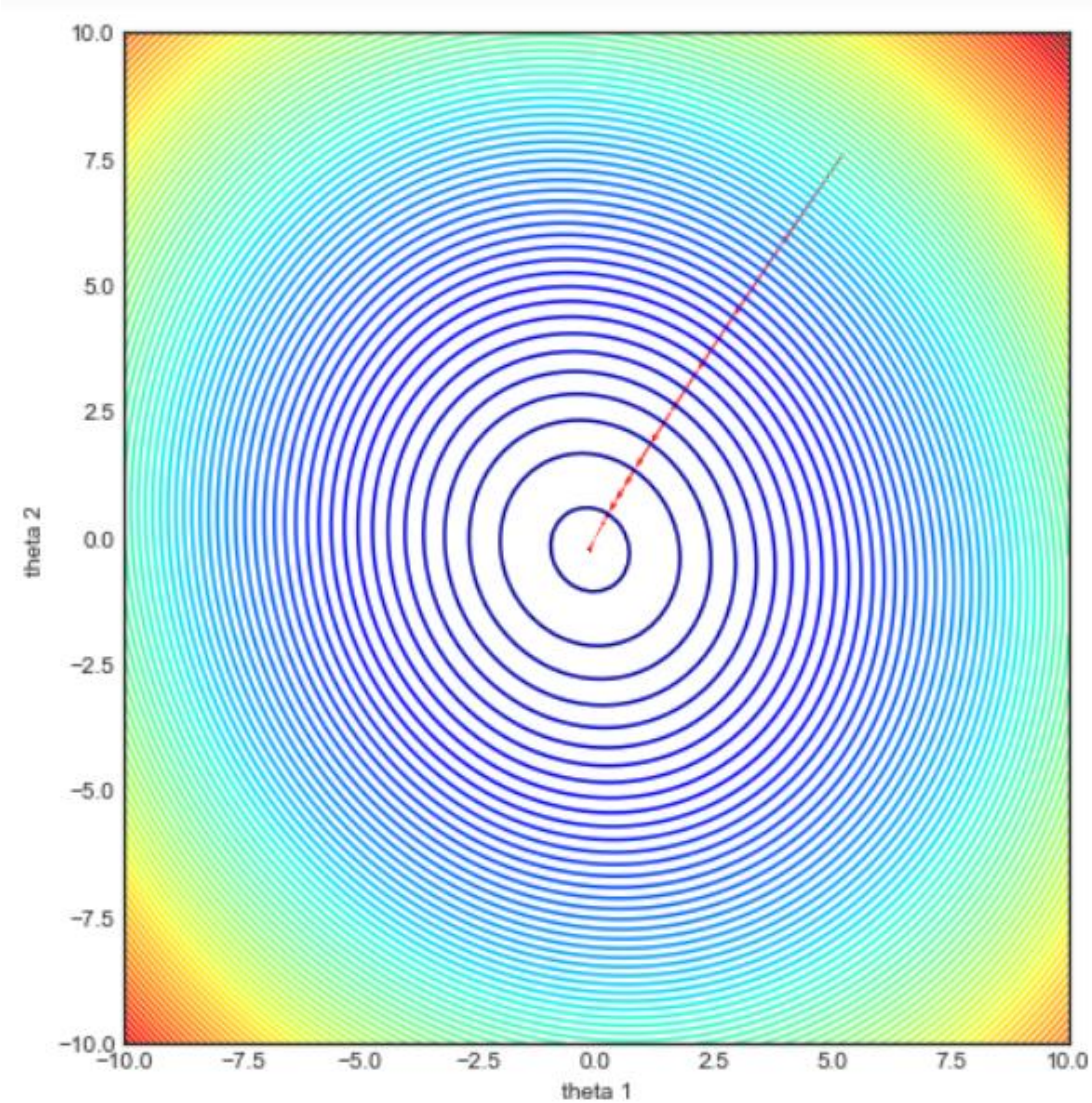
In [19]:

```
from mpl_toolkits.mplot3d import Axes3D
from mpl_toolkits import mplot3d

#Angles needed for quiver plot
anglesx = np.array(theta_0)[1:] - np.array(theta_0)[:-1]
anglesy = np.array(theta_1)[1:] - np.array(theta_1)[:-1]
```

```python
%matplotlib inline
fig = plt.figure(figsize = (16,8))

#Surface plot
ax = fig.add_subplot(1, 2, 1, projection='3d')
ax.plot_surface(T1, T2, Z, rstride = 5, cstride = 5, cmap = 'jet', alpha=0.5)
ax.plot(theta_0,theta_1,J_history_reg, marker = '*', color = 'r', alpha = .4, label = 'Gradient descent')

ax.set_xlabel('theta 1')
ax.set_ylabel('theta 2')
ax.set_zlabel('error')
ax.set_title('RSS gradient descent: Root at {}'.format(theta_result_reg.ravel()))
ax.view_init(45, -45)


#Contour plot
ax = fig.add_subplot(1, 2, 2)
ax.contour(T1, T2, Z, 100, cmap = 'jet')
ax.quiver(theta_0[:-1], theta_1[:-1], anglesx, anglesy, scale_units = 'xy', angles = 'xy', scale = 1, color = 'r', alph
a = .9)
ax.set_xlabel('theta 1')
ax.set_ylabel('theta 2')

plt.suptitle('Cost function and gradient descent: Ridge regularization')
plt.show()
```



RSS gradient descent: Root at [-0.14 -0.23]
theta 1
theta 2
error
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
5
10
15
20
25
30



The Simplest Machine Learning Algorithm

Wine Quality Datasets

$$min \sum_{i=1}^{n} (\mathbf{x}_i^T \mathbf{w} - y_i)^2 + \lambda \|\mathbf{w}\|^2$$

$$\boxed{\mathbf{w} = (\mathbf{X}^T\mathbf{X} + \lambda \mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}}$$

In [20]:

```
#The Simplest Machine Learning Algorithm
#https://simplyml.com/the-simplest-machine-learning-algorithm/
import numpy # as np
import sklearn
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
#from sklearn.cross_validation import train_test_split
#import matplotlib.pyplot as plt
#from IPython.display import display, Math, Latex

class RidgeRegression(object):
    def __init__(self, lmbda=0.1):
        self.lmbda = lmbda

    def fit(self, X, y):
        C = X.T.dot(X) + self.lmbda*numpy.eye(X.shape[1])
        self.w = numpy.linalg.inv(C).dot(X.T.dot(y))

    def predict(self, X):
        return X.dot(self.w)

    def get_params(self, deep=True):
        return {"lmbda": self.lmbda}

    def set_params(self, lmbda=0.1):
        self.lmbda = lmbda
        return self

Xy = numpy.loadtxt("winequality/winequality-white.csv", delimiter=";", skiprows=1)
```

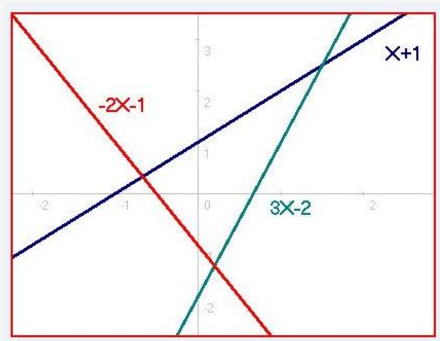
```python
#X = Xy[:, 0:-1]
X = Xy[:, :-1]
X = preprocessing.scale(X)

y = Xy[:, -1]
y -= y.mean()

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
ridge = RidgeRegression()
param_grid = [{"lmbda": 2.0**numpy.arange(-5, 10)}]
#learner = sklearn.model_selection.GridSearchCV(ridge, param_grid, scoring="mean_absolute_error", n_jobs=-1, verbose=0)
#sorted(sklearn.metrics.SCORERS.keys())
learner = sklearn.model_selection.GridSearchCV(ridge, param_grid, scoring="neg_mean_absolute_error", n_jobs=-1, verbose
=0)
learner.fit(X_train, y_train)

y_pred = learner.predict(X_test)
ridge_error = sklearn.metrics.mean_absolute_error(y_test, y_pred)
print(ridge_error)
```

```
0.5964294813409599
```

In [ ]:

Переопределённая система

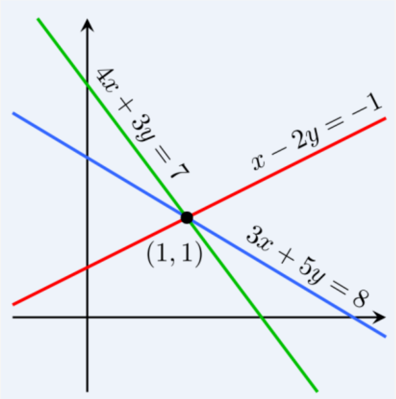
Переопределённая система — система, число уравнений которой больше числа неизвестных.
Для однозначного решения линейной системы уравнений нужно иметь n уравнений при n переменных величинах. Если уравнений меньше, чем число переменных величин, то такая система не определенане
определена (или несовместнанесовместна, см. следствие 2 в Метод Гаусса). Также система n (или больше) уравнений может быть недоопределенанедоопределена, если некоторые



The equations
$x-2y=-1$x−2y=−1,
$3x+5y=8$3x+5y=8,
$4x+3y=7$4x+3y=7
are linearly dependent.

уравнения не поставляют никакую дополнительную независимую от других уравнений информацию.

В силу отсутствия точного решения переопределённых систем, на практике принято вместо него отыскивать вектор, наилучшим образом удовлетворяющий всем уравнениям, то есть минимизирующий норму невязки системы в какой-нибудь степени. Этой проблеме посвящён отдельный раздел математической статистики — регрессионный анализ. Наиболее часто минимизируют квадрат отклонений от оцениваемого решения. Для этого применяют так называемый метод наименьших квадратов.



Невязка

Пусть требуется найти такое x, что значение функции:	$f(x)=b$
Подставив приближенное значение x_0 вместо x, получаем невязку	$b-f(x_0)$
а ошибка в этом случае равна	x_0-x

Если точное значение xx неизвестно, вычисление ошибки невозможно, однако при этом может быть определена невязка.

Невязка — величина ошибки (расхождения) приближённого равенства.

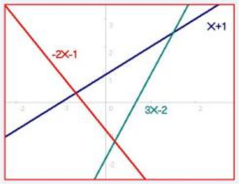
несовместные системы линейных уравнений

$$\begin{cases} Y = 2X - 1 \\ Y = 3X - 2 \\ Y = X + 1 \end{cases} =$$

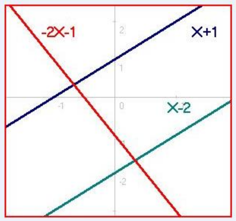
может быть записана в матричной форме в матричной форме

$$= \begin{bmatrix} 2 & 1 \\ -3 & 1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix} \begin{bmatrix} -1 \\ -2 \\ 1 \end{bmatrix}$$

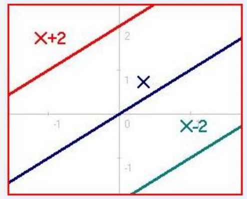
#1 A system of three linearly independent equations, three lines, no solutions



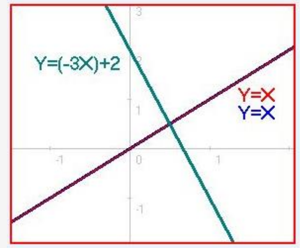
#2 A system of three linearly independent equations, three lines (two parallel), no solutions



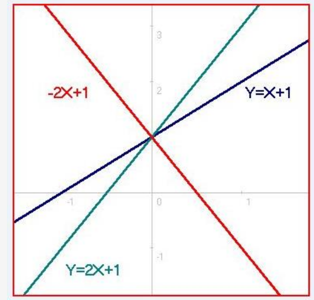
#3 A system of three linearly independent equations, three lines (all parallel), no solutions



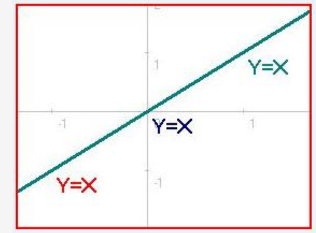
#4 A system of three equations (one equation linearly dependent on the others), three lines (two coinciding), one solution



#5 A system of three equations (one equation linearly dependent on the others), three lines (two coinciding), one solution



#6 A system of three equations (one equation linearly dependent on the others), three lines (two coinciding), one solution



In [ ]:

In [ ]:

```
#Ridge and Lasso: Geometric Interpretation
#https://www.astroml.org/book_figures/chapter8/fig_lasso_ridge.html

# Author: Jake VanderPlas
# License: BSD
#   The figure produced by this code is published in the textbook
#   "Statistics, Data Mining, and Machine Learning in Astronomy" (2013)
#   For more information, see http://astroML.github.com
#   To report a bug or issue, use the following forum:
#    https://groups.google.com/forum/#!forum/astroml-general
import numpy as np
from matplotlib import pyplot as plt
from matplotlib.patches import Ellipse, Circle, RegularPolygon

#------------------------------------------------------------
# This function adjusts matplotlib settings for a uniform feel in the textbook.
# Note that with usetex=True, fonts are rendered with LaTeX.  This may
# result in an error if LaTeX is not installed on your system.  In that case,
# you can set usetex to False.
```

```python
#if "setup_text_plots" not in globals():
#    from astroML.plotting import setup_text_plots

#------------------------------------------------------------
# Set up figure
fig = plt.figure(figsize=(5, 2.5), facecolor='w')

#------------------------------------------------------------
# plot ridge diagram
ax = fig.add_axes([0, 0, 0.5, 1], frameon=False, xticks=[], yticks=[])

# plot the axes
ax.arrow(-1, 0, 9, 0, head_width=0.1, fc='k')
ax.arrow(0, -1, 0, 9, head_width=0.1, fc='k')

# plot the ellipses and circles
for i in range(3):
    ax.add_patch(Ellipse((3, 5),
                         3.5 * np.sqrt(2 * i + 1), 1.7 * np.sqrt(2 * i + 1),
                         -15, fc='none'))

ax.add_patch(Circle((0, 0), 3.815, fc='none'))

# plot arrows
ax.arrow(0, 0, 1.46, 3.52, head_width=0.2, fc='k',
         length_includes_head=True)
ax.arrow(0, 0, 3, 5, head_width=0.2, fc='k',
         length_includes_head=True)
ax.arrow(0, -0.2, 3.81, 0, head_width=0.1, fc='k',
         length_includes_head=True)
ax.arrow(3.81, -0.2, -3.81, 0, head_width=0.1, fc='k',
         length_includes_head=True)

# annotate with text
ax.text(7.5, -0.1, r'$\theta_1$', va='top')
ax.text(-0.1, 7.5, r'$\theta_2$', ha='right')
ax.text(3, 5 + 0.2, r'$\rm \theta_{normal\ equation}$',
        ha='center', bbox=dict(boxstyle='round', ec='k', fc='w'))
```

```
ax.text(1.46, 3.52 + 0.2, r'$\rm \theta_{ridge}$', ha='center',
        bbox=dict(boxstyle='round', ec='k', fc='w'))
ax.text(1.9, -0.3, r'$r$', ha='center', va='top')

ax.set_xlim(-2, 9)
ax.set_ylim(-2, 9)

#------------------------------------------------------------
# plot lasso diagram
ax = fig.add_axes([0.5, 0, 0.5, 1], frameon=False, xticks=[], yticks=[])

# plot axes
ax.arrow(-1, 0, 9, 0, head_width=0.1, fc='k')
ax.arrow(0, -1, 0, 9, head_width=0.1, fc='k')

# plot ellipses and circles
for i in range(3):
    ax.add_patch(Ellipse((3, 5),
                         3.5 * np.sqrt(2 * i + 1), 1.7 * np.sqrt(2 * i + 1),
                         -15, fc='none'))

# this is producing some weird results on save
#ax.add_patch(RegularPolygon((0, 0), 4, 4.4, np.pi, fc='none'))
ax.plot([-4.4, 0, 4.4, 0, -4.4], [0, 4.4, 0, -4.4, 0], '-k')

# plot arrows
ax.arrow(0, 0, 0, 4.4, head_width=0.2, fc='k', length_includes_head=True)
ax.arrow(0, 0, 3, 5, head_width=0.2, fc='k', length_includes_head=True)
ax.arrow(0, -0.2, 4.2, 0, head_width=0.1, fc='k', length_includes_head=True)
ax.arrow(4.2, -0.2, -4.2, 0, head_width=0.1, fc='k', length_includes_head=True)

# annotate plot
ax.text(7.5, -0.1, r'$\theta_1$', va='top')
ax.text(-0.1, 7.5, r'$\theta_2$', ha='right')
ax.text(3, 5 + 0.2, r'$\rm \theta_{normal\ equation}$',
        ha='center', bbox=dict(boxstyle='round', ec='k', fc='w'))
ax.text(0, 4.4 + 0.2, r'$\rm \theta_{lasso}$', ha='center',
        bbox=dict(boxstyle='round', ec='k', fc='w'))
```

```
ax.text(2, -0.3, r'$r$', ha='center', va='top')

ax.set_xlim(-2, 9)
ax.set_ylim(-2, 9)

plt.show()
```