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**](https://stackoverflow.com/questions/38562701/ridge-regression-scikit-learn-vs-direct-calculation-does-not-match-for-alpha)

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]:

**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]:

*# 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]:

plt.figure(figsize**=**(12,5))

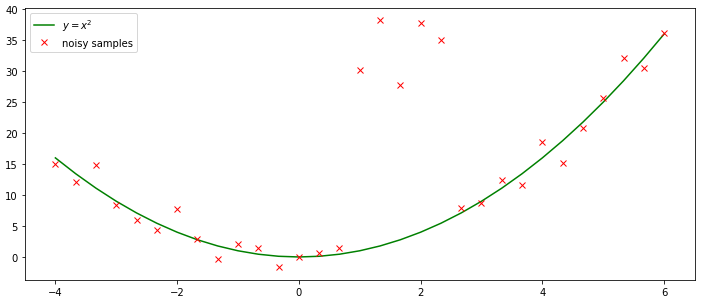
*# Plot synthetic data*

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

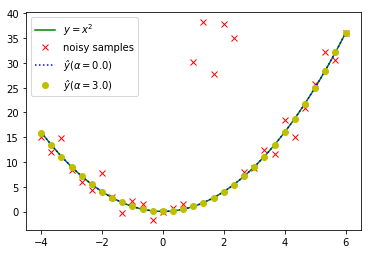
alpha= 0.0

[ -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

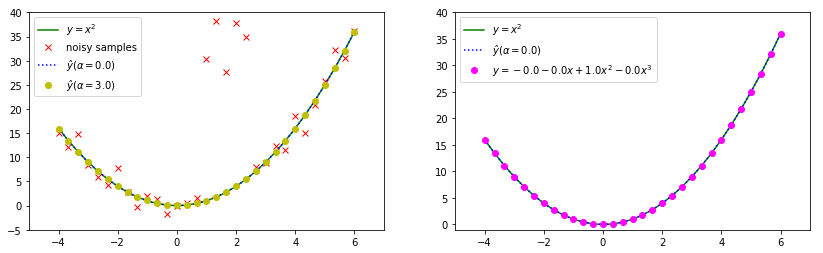
alpha= 3.0

[ 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**=**'magenta')

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)

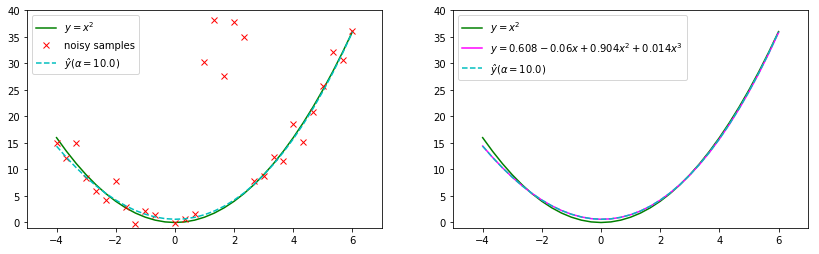
*#ax[1].vlines([0],-1,60); ax[1].hlines([0],-5,15)*

plt.show()

X.shape[0] 31 X.shape[1] 4

alpha= 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**=**'magenta')

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)

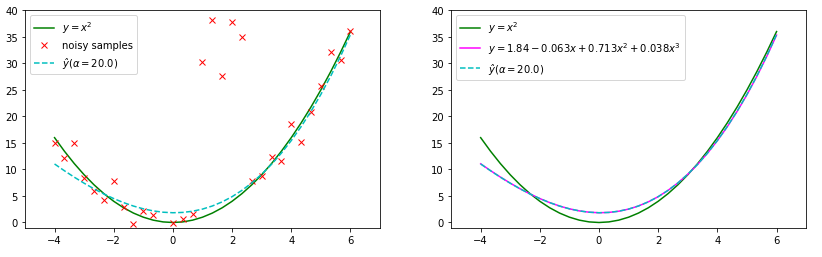
*#ax[1].vlines([0],-1,60); ax[1].hlines([0],-5,15)*

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**=**'magenta')

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)

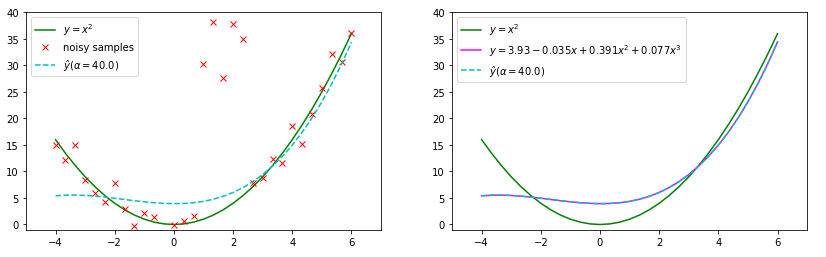
*#ax[1].vlines([0],-1,60); ax[1].hlines([0],-5,15)*

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**](https://mlforanalytics.com/2018/05/22/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

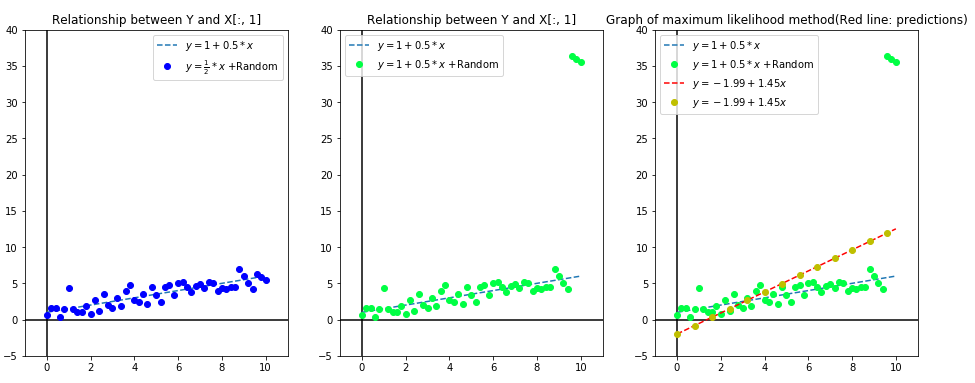
1.18 3.59 2.03 1.61 2.94 1.88 3.96 4.81 2.72 2.39 3.52 2.16

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))

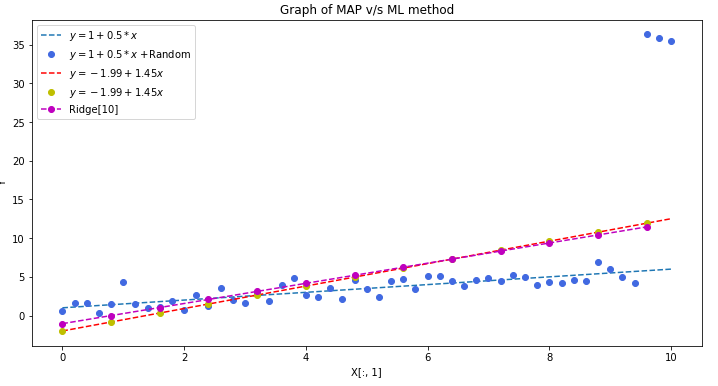
plt.title('Graph of MAP v/s ML method')

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))

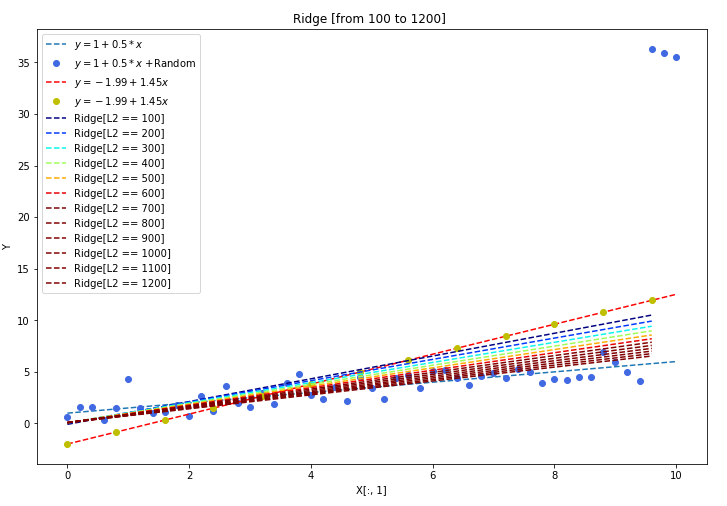
plt.legend()

​

plt.xlabel('X[:, 1]')

plt.ylabel('Y')

plt.show()



[**Норма (математика)**](https://ru.wikipedia.org/wiki/%D0%9D%D0%BE%D1%80%D0%BC%D0%B0_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0))

|  |  |  |
| --- | --- | --- |
|  |  |  |

[**Собственный вектор**](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D0%B1%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D1%8B%D0%B9_%D0%B2%D0%B5%D0%BA%D1%82%D0%BE%D1%80)

**Со́бственный ве́ктор** — понятие в [линейной алгебре](https://ru.wikipedia.org/wiki/%D0%9B%D0%B8%D0%BD%D0%B5%D0%B9%D0%BD%D0%B0%D1%8F_%D0%B0%D0%BB%D0%B3%D0%B5%D0%B1%D1%80%D0%B0), определяемое для произвольного [линейного оператора](https://ru.wikipedia.org/wiki/%D0%9B%D0%B8%D0%BD%D0%B5%D0%B9%D0%BD%D0%BE%D0%B5_%D0%BE%D1%82%D0%BE%D0%B1%D1%80%D0%B0%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5) как ненулевой [вектор](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%BA%D1%82%D0%BE%D1%80_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), применение к которому оператора даёт [коллинеарный](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BB%D0%BB%D0%B8%D0%BD%D0%B5%D0%B0%D1%80%D0%BD%D0%BE%D1%81%D1%82%D1%8C) 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](https://ru.wikipedia.org/wiki/%D0%9A%D0%B2%D0%B0%D0%B4%D1%80%D0%B0%D1%82%D0%BD%D0%B0%D1%8F_%D0%BC%D0%B0%D1%82%D1%80%D0%B8%D1%86%D0%B0), 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](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B8%D1%81) из собственных векторов оператора. Множество собственных значений линейного оператора ([спектр оператора](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B5%D0%BA%D1%82%D1%80_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80%D0%B0)) 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](https://ru.wikipedia.org/wiki/%D0%9D%D1%83%D0%BB%D0%B5%D0%B2%D0%BE%D0%B9_%D0%B2%D0%B5%D0%BA%D1%82%D0%BE%D1%80), называется собственным подпространством этого оператора.

𝐀𝑥⃗ =𝑦⃗ Ax→=y→

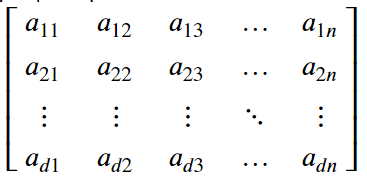
𝐀𝑥⃗ =𝜆𝑥⃗ Ax→=λx→

Предположим что существует такой вектор не равный 𝑥⃗ ≠0⃗  тогда:

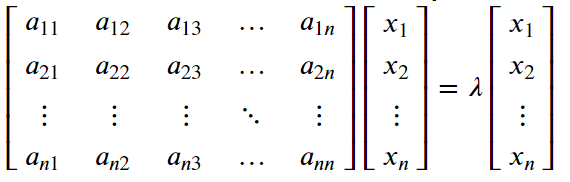
𝐀𝑥⃗ −𝜆𝑥⃗ =0⃗  
𝐀𝑥⃗ −𝜆𝐈𝑛𝑥⃗ =0⃗   
(𝐀−𝜆𝐈𝑛)𝑥⃗  =0⃗   
Если имеется собственный вектор не равный 0 то тогда:

|𝐀−𝜆𝐈|=0

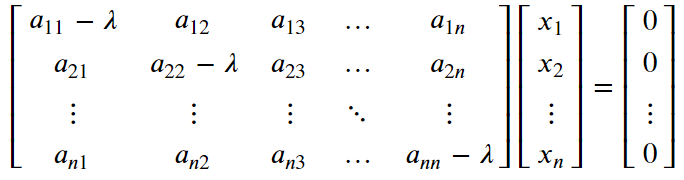
Пусть имеется квадратная матрица 𝐴A разменостью 𝑛×𝑛:



**собственное значение** со связанным **собственным вектором**:



Тогда эту систему можно преобразовать к:

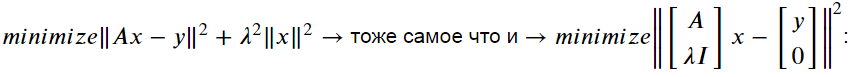


В более компактной форме может быть записана как:

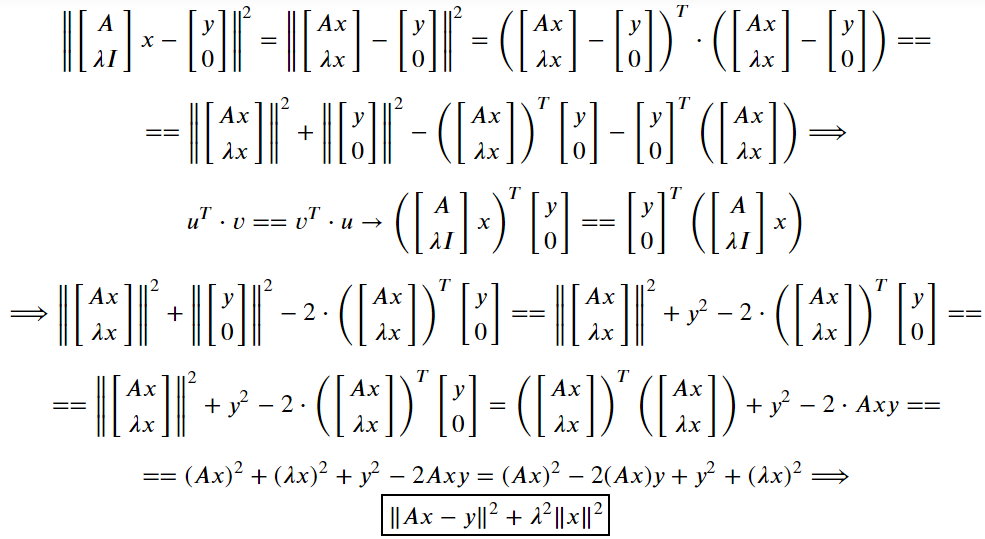
(𝐀−𝜆⋅𝐈)𝐗=0

[**Ridge regression and L2 regularization - Introduction**](https://xavierbourretsicotte.github.io/intro_ridge.html)

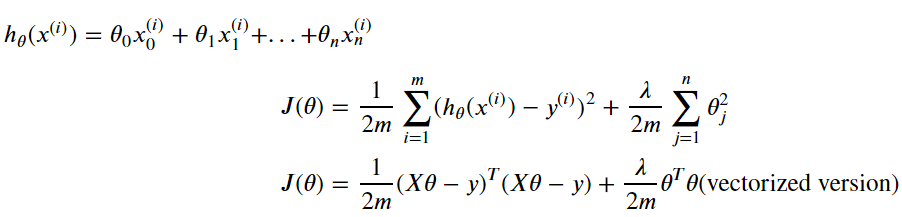
|  |  |
| --- | --- |
| **Ordinary Least Squares**    Берём производную и приравниваем к 0, в точке перегиба самый минимум:    **Ridge regression** Добавляется маленькая добавка к матрице по диагонали чтобы eigen-values(собственное число) и матрица стала в более стабильном состоянии:    **Quadratic cost function** изменим чуть-чуть значение 𝜆λ - возведём в квадрат для облегчения вычислений: |  |

[Factor selection and pivoted QR](http://www.cs.cornell.edu/courses/cs6210/2019fa/lec/2019-10-16.pdf)  

[Calculation of the squared Euclidean norm](https://math.stackexchange.com/questions/1865476/calculation-of-the-squared-euclidean-norm)



**Regularized Cost Function**



[Why divide by 2𝑚2m ?](https://math.stackexchange.com/questions/884887/why-divide-by-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 ?](https://datascience.stackexchange.com/questions/52157/why-do-we-have-to-divide-by-2-in-the-ml-squared-error-cost-function)

* 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](http://mccormickml.com/2014/03/04/gradient-descent-derivation/) for example, and expect the same result.

[Where does 1/2m came from ?](https://stackoverflow.com/questions/21099289/cant-understand-the-cost-function-for-linear-regression)

**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](https://www.python.org/dev/peps/pep-0465/)

*#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](https://docs.python.org/3/reference/datamodel.html#emulating-numeric-types) specifies that the @ operator invokes \_\_matmul\_\_ and \_\_rmatmul\_\_.

[numpy.dot(a, b, out=None)](https://docs.scipy.org/doc/numpy/reference/generated/numpy.dot.html) 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]]

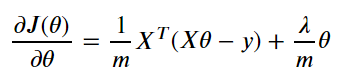
[[37 40]

[85 92]] (2, 2)

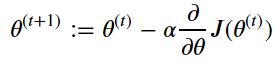
[[37 40]

[85 92]] (2, 2)

**Gradient**



**Gradient descent (vectorized)**



**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)

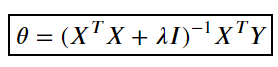
*#Calculate the cost for each iteration(used to plot convergence)*

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**



**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]:

**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]:

*#Generating sine curve and uniform noise*

x **=** np.linspace(0,1,40)

y **=** np.sin(x **\*** 1.5 **\*** np.pi )

​

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')

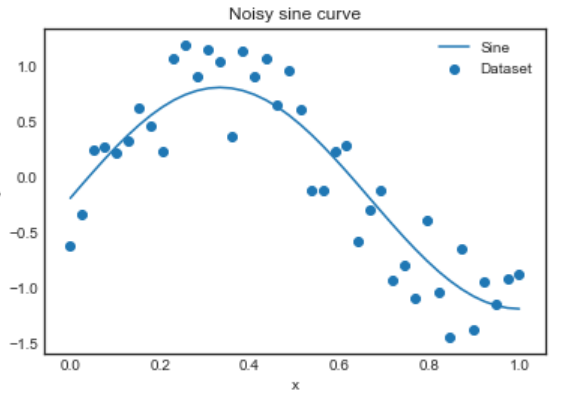
plt.title('Noisy sine curve')

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]

​

**%**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', alpha **=** .9)

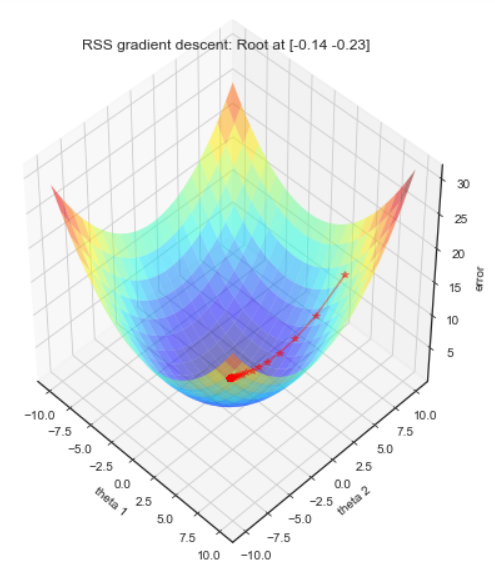
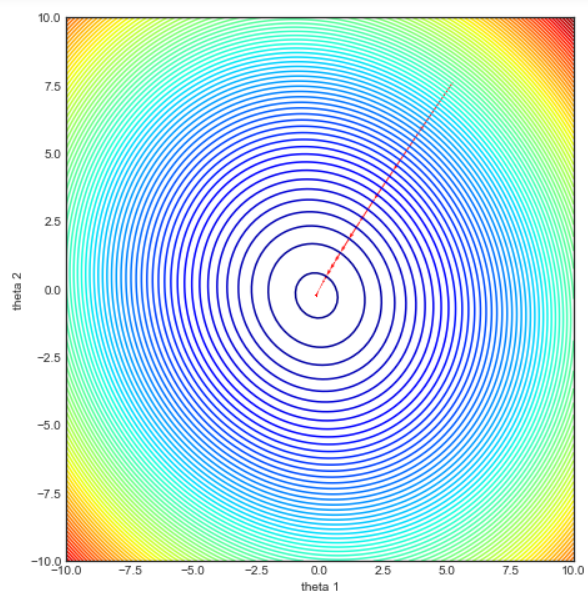
ax.set\_xlabel('theta 1')

ax.set\_ylabel('theta 2')

​

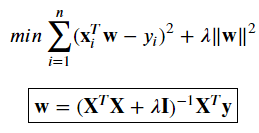
plt.suptitle('Cost function and gradient descent: Ridge regularization')

plt.show()

[**The Simplest Machine Learning Algorithm**](https://simplyml.com/the-simplest-machine-learning-algorithm/)

[Wine Quality Datasets](http://www3.dsi.uminho.pt/pcortez/wine/)



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)

​

*#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**](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D0%BE%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D1%91%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0)

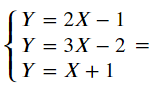
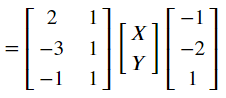
|  |  |  |
| --- | --- | --- |
| **Переопределённая система** — система, число уравнений которой больше числа неизвестных.  Для однозначного решения линейной системы уравнений нужно иметь 𝑛 уравнений при 𝑛 переменных величинах. Если уравнений меньше, чем число переменных величин, то такая система не определенане определена (или несовместнанесовместна, см. следствие 2 в [Метод Гаусса](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D0%93%D0%B0%D1%83%D1%81%D1%81%D0%B0)). Также система 𝑛 (или больше) уравнений может быть недоопределенанедоопределена, если некоторые 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](https://ru.wikipedia.org/wiki/%D0%9D%D0%B5%D0%B2%D1%8F%D0%B7%D0%BA%D0%B0) системы в какой-нибудь степени. Этой проблеме посвящён отдельный раздел [математической статистики](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%81%D1%82%D0%B0%D1%82%D0%B8%D1%81%D1%82%D0%B8%D0%BA%D0%B0) — [регрессионный анализ](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B3%D1%80%D0%B5%D1%81%D1%81%D0%B8%D0%BE%D0%BD%D0%BD%D1%8B%D0%B9_%D0%B0%D0%BD%D0%B0%D0%BB%D0%B8%D0%B7). Наиболее часто минимизируют квадрат отклонений от оцениваемого решения. Для этого применяют так называемый [метод наименьших квадратов](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D0%BD%D0%B0%D0%B8%D0%BC%D0%B5%D0%BD%D1%8C%D1%88%D0%B8%D1%85_%D0%BA%D0%B2%D0%B0%D0%B4%D1%80%D0%B0%D1%82%D0%BE%D0%B2). |  | The equations  𝑥−2𝑦=−1x−2y=−1, 3𝑥+5𝑦=83x+5y=8, 4𝑥+3𝑦=74x+3y=7 are linearly dependent. |

[**Невязка**](https://ru.wikipedia.org/wiki/%D0%9D%D0%B5%D0%B2%D1%8F%D0%B7%D0%BA%D0%B0)

**Невязка** — величина ошибки (расхождения) приближённого равенства.

|  |  |
| --- | --- |
| Пусть требуется найти такое 𝑥, что значение функции: | 𝑓(𝑥)=𝑏 |
| Подставив приближенное значение 𝑥0 вместо 𝑥, получаем невязку | 𝑏−𝑓(𝑥0) |
| а ошибка в этом случае равна | 𝑥0−𝑥 |
| Если точное значение 𝑥x неизвестно, вычисление ошибки невозможно, однако при этом может быть определена невязка. | |

[**несовместные системы линейных уравнений**](https://en.wikipedia.org/wiki/Overdetermined_system)

может быть записана в матричной форме в матричной форме

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| #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.*

*#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()

​

