Machine Learning

6 Oct 2021

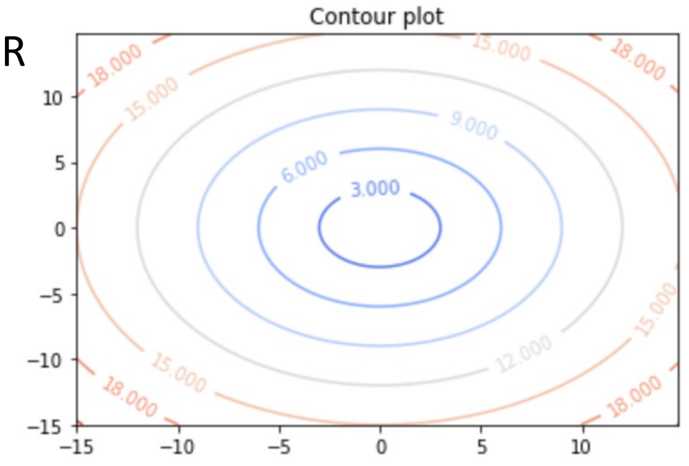
Prof Dr Arif Mahmood

Plotting Functions of Two Variables: Contour Plot

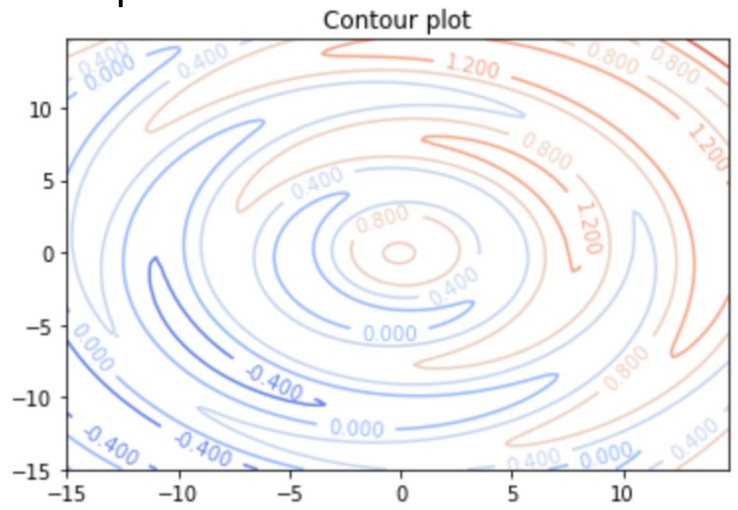
```python
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
from matplotlib.pyplot import *
from numpy import *
```

```python
x = arange(-15, 15, 0.25)
y = arange(-15, 15, 0.25)
X, Y = meshgrid(x, y)
R = sqrt(X**2 + Y**2)
Z = -.4 + (X+15)/30. + (Y+15)/40.+.5*sin(R)

figure(1)
CS = contour(X, Y, Z, cmap=cm.coolwarm)
clabel(CS, inline=1, fontsize=10)
title('Contour plot')

show()
```

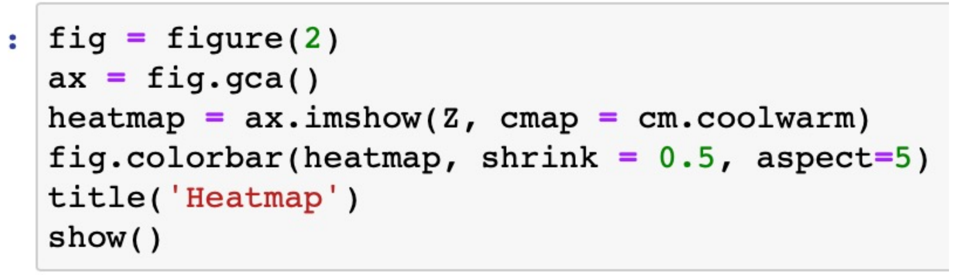
- Contour plot of R



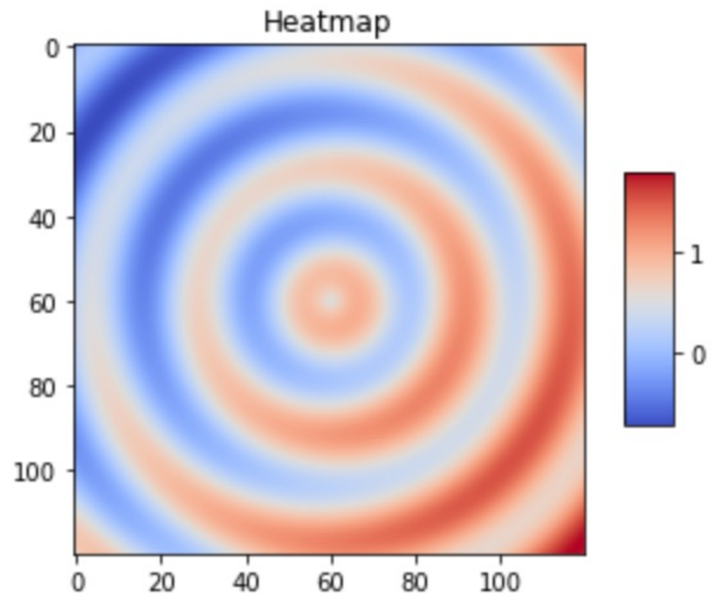
- Contour plot of Z



Plotting Functions of Two Variables: Heatmap Plot

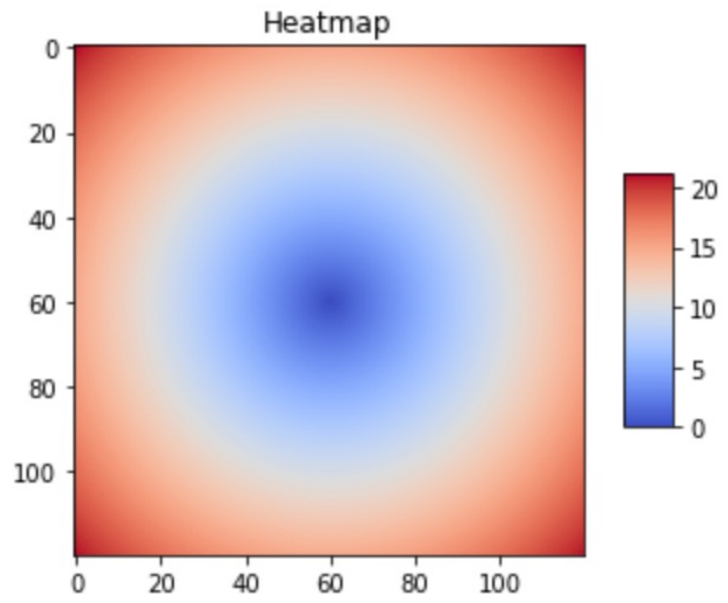


```
fig = figure(2)
ax = fig.gca()
heatmap = ax.imshow(Z, cmap = cm.coolwarm)
fig.colorbar(heatmap, shrink = 0.5, aspect=5)
title('Heatmap')
show()
```

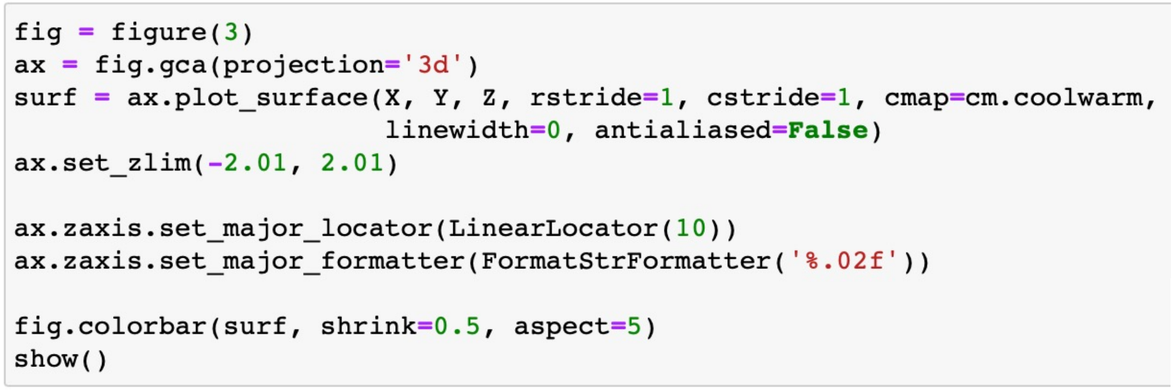


- Heat map of:

R = sqrt(X**2 + Y**2)



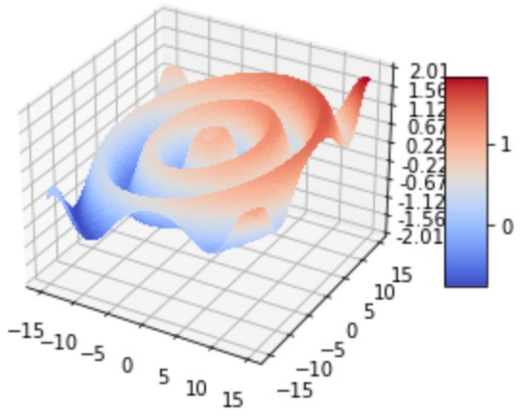
Plotting Functions of Two Variables: Surface Plot

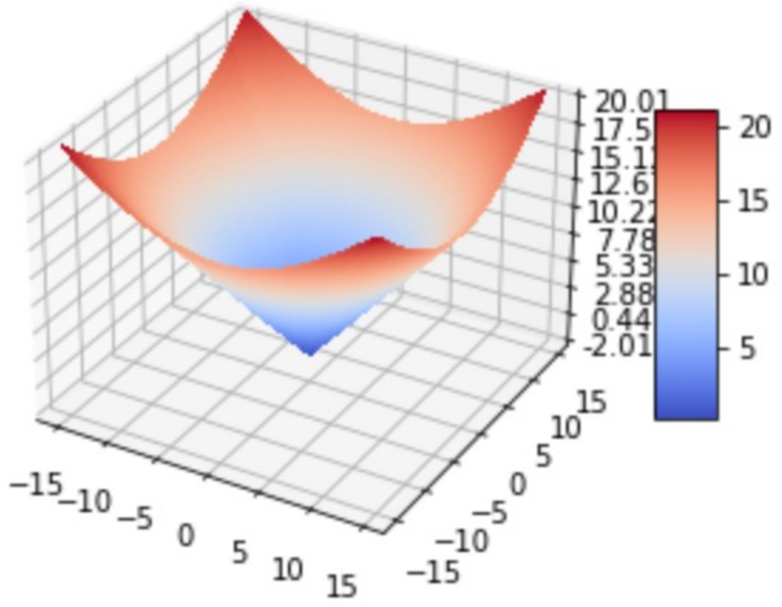


```
fig = figure(3)
ax = fig.gca(projection='3d')
surf = ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap=cm.coolwarm,
                       linewidth=0, antialiased=False)
ax.set_zlim(-2.01, 2.01)

ax.zaxis.set_major_locator(LinearLocator(10))
ax.zaxis.set_major_formatter(FormatStrFormatter('%.02f'))

fig.colorbar(surf, shrink=0.5, aspect=5)
show()
```

- Surface plot of:

R = sqrt(X**2 + Y**2)





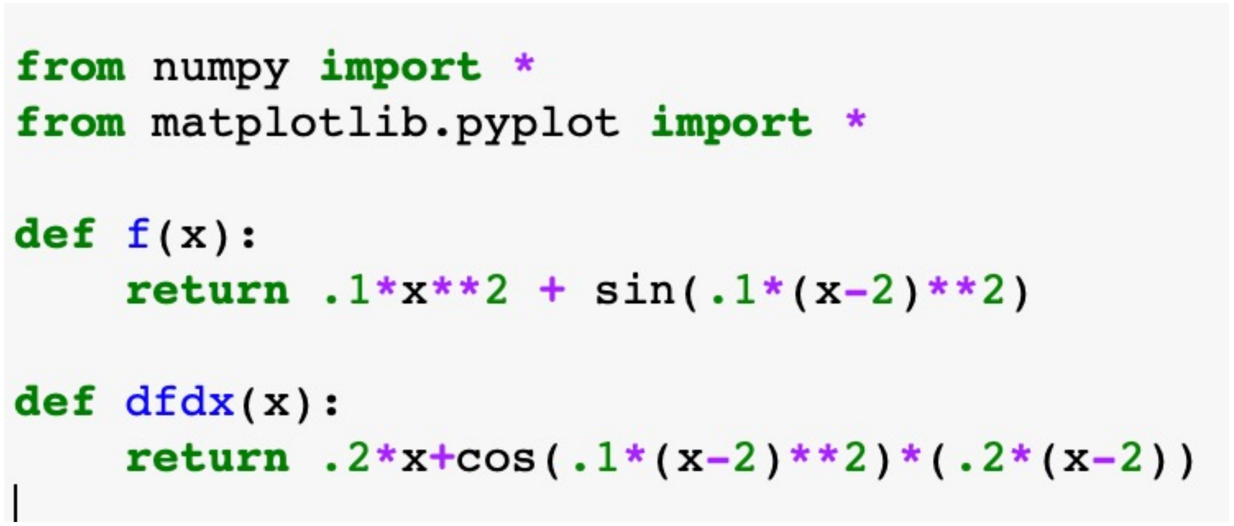
Gradient Descent for Functions of One Variable

Consider a function:

$$f(x)=0.1x^2+\sin(0.1(x-2)^2)$$

Its derivative:

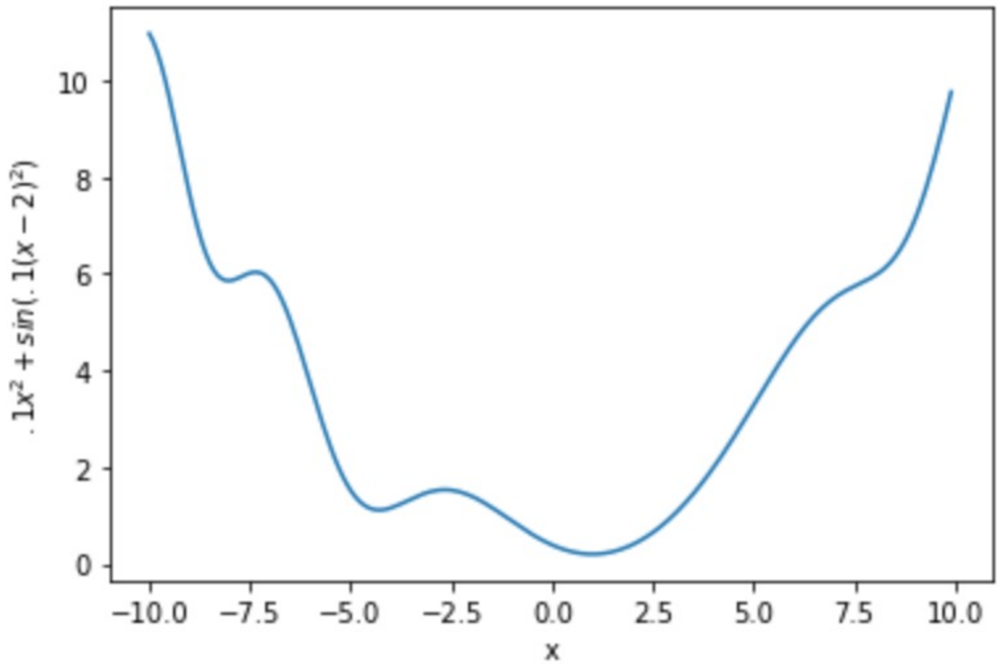
$$f'(x)=0.2x+\cos(0.1(x-2)^2)(0.2(x-2))$$



```python
from numpy import *
from matplotlib.pyplot import *

def f(x):
    return .1*x**2 + sin(.1*(x-2)**2)

def dfdx(x):
    return .2*x+cos(.1*(x-2)**2)*(.2*(x-2))
```

Visualization of Function

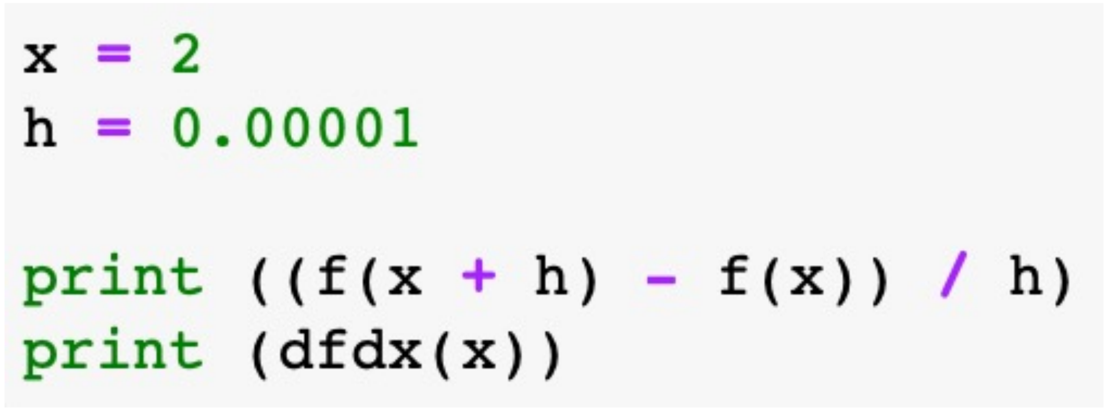
```
x = arange(-10, 10, .1)
y = f(x)

figure(1)
plot(x, y)
xlabel("x")
ylabel("$.1 x^2 + sin(.1 (x-2)^2)$")
show()
```



Neumerical Derivative Computation

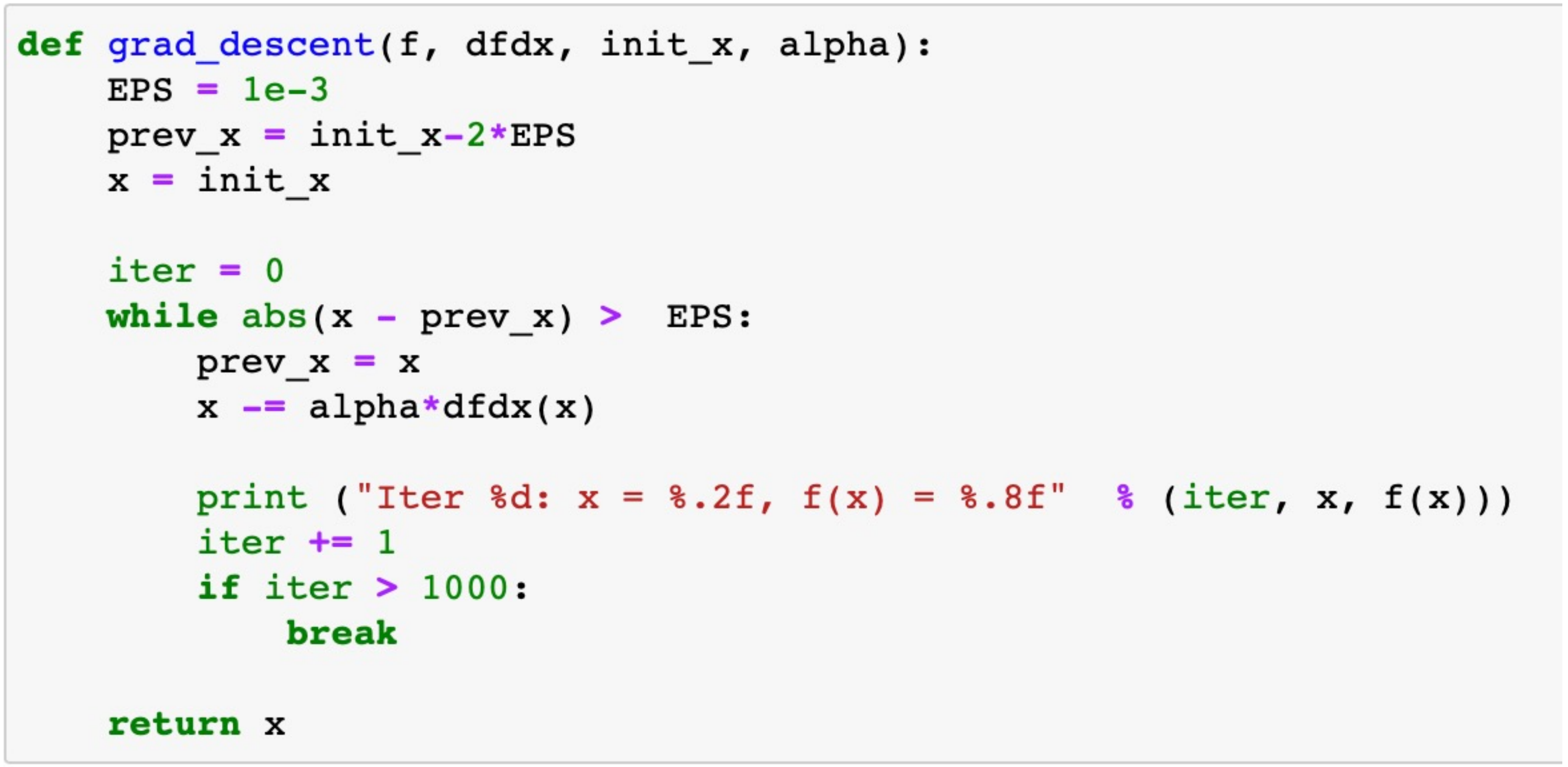
- Compute the slope of function at a particular location:

$$f'(x) = \frac{f(x+h) - f(x)}{h}$$



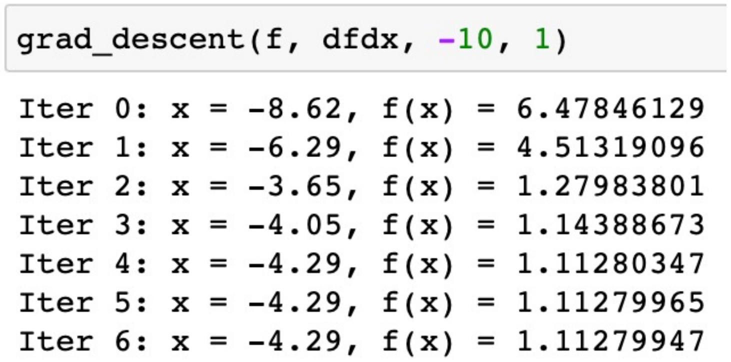
```
x = 2
h = 0.00001

print ((f(x + h) - f(x)) / h)
print (dfdx(x))
```

Gradient Descent Function



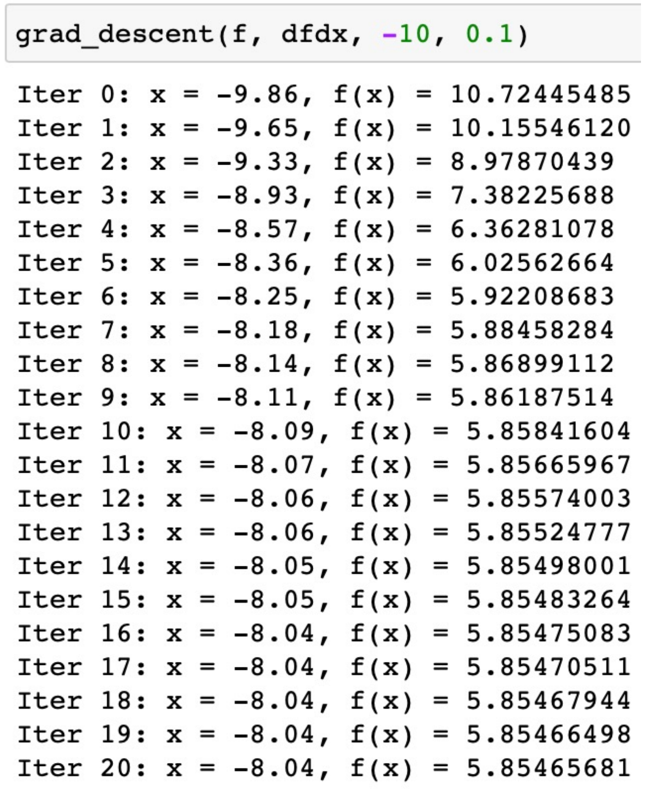
```python
def grad_descent(f, dfdx, init_x, alpha):
    EPS = 1e-3
    prev_x = init_x-2*EPS
    x = init_x

    iter = 0
    while abs(x - prev_x) >  EPS:
        prev_x = x
        x -= alpha*dfdx(x)

        print ("Iter %d: x = %.2f, f(x) = %.8f"  % (iter, x, f(x)))
        iter += 1
        if iter > 1000:
            break

    return x
```

- Effect of initial guess
- Effect of learning rate
- Final result may vary



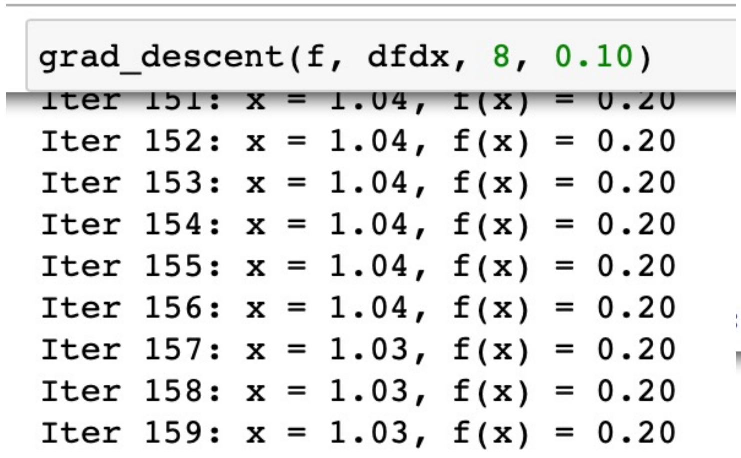
```
grad_descent(f, dfdx, -10, 1)
```

```
Iter 0: x = -8.62, f(x) = 6.47846129
Iter 1: x = -6.29, f(x) = 4.51319096
Iter 2: x = -3.65, f(x) = 1.27983801
Iter 3: x = -4.05, f(x) = 1.14388673
Iter 4: x = -4.29, f(x) = 1.11280347
Iter 5: x = -4.29, f(x) = 1.11279965
Iter 6: x = -4.29, f(x) = 1.11279947
```



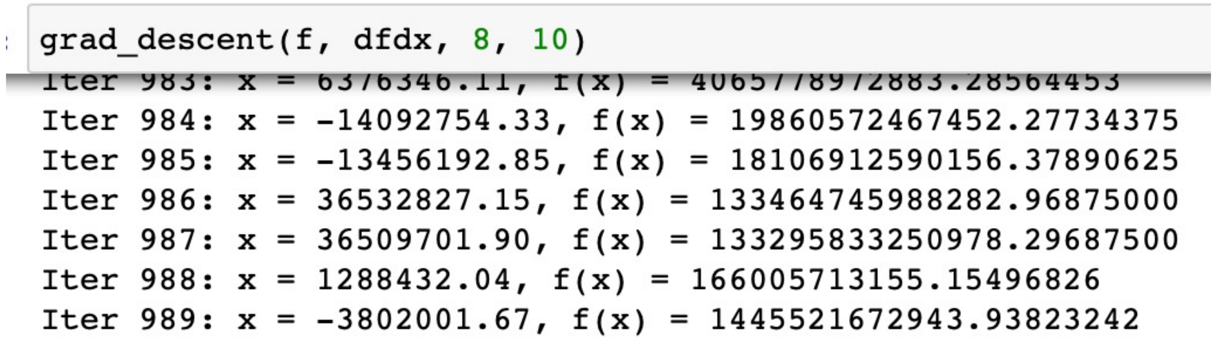
```
grad_descent(f, dfdx, -10, 0.1)
```

```
Iter 0: x = -9.86, f(x) = 10.72445485
Iter 1: x = -9.65, f(x) = 10.15546120
Iter 2: x = -9.33, f(x) = 8.97870439
Iter 3: x = -8.93, f(x) = 7.38225688
Iter 4: x = -8.57, f(x) = 6.36281078
Iter 5: x = -8.36, f(x) = 6.02562664
Iter 6: x = -8.25, f(x) = 5.92208683
Iter 7: x = -8.18, f(x) = 5.88458284
Iter 8: x = -8.14, f(x) = 5.86899112
Iter 9: x = -8.11, f(x) = 5.86187514
Iter 10: x = -8.09, f(x) = 5.85841604
Iter 11: x = -8.07, f(x) = 5.85665967
Iter 12: x = -8.06, f(x) = 5.85574003
Iter 13: x = -8.06, f(x) = 5.85524777
Iter 14: x = -8.05, f(x) = 5.85498001
Iter 15: x = -8.05, f(x) = 5.85483264
Iter 16: x = -8.04, f(x) = 5.85475083
Iter 17: x = -8.04, f(x) = 5.85470511
Iter 18: x = -8.04, f(x) = 5.85467944
Iter 19: x = -8.04, f(x) = 5.85466498
Iter 20: x = -8.04, f(x) = 5.85465681
```



```
grad_descent(f, dfdx, 8, 0.10)
```

```
Iter 151: x = 1.04, f(x) = 0.20
Iter 152: x = 1.04, f(x) = 0.20
Iter 153: x = 1.04, f(x) = 0.20
Iter 154: x = 1.04, f(x) = 0.20
Iter 155: x = 1.04, f(x) = 0.20
Iter 156: x = 1.04, f(x) = 0.20
Iter 157: x = 1.03, f(x) = 0.20
Iter 158: x = 1.03, f(x) = 0.20
Iter 159: x = 1.03, f(x) = 0.20
```



```
grad_descent(f, dfdx, 8, 10)
```

```
Iter 983: x = 6376346.11, f(x) = 40657789728883.28564453
Iter 984: x = -14092754.33, f(x) = 19860572467452.27734375
Iter 985: x = -13456192.85, f(x) = 18106912590156.37890625
Iter 986: x = 36532827.15, f(x) = 133464745988282.96875000
Iter 987: x = 36509701.90, f(x) = 133295833250978.29687500
Iter 988: x = 1288432.04, f(x) = 166005713155.15496826
Iter 989: x = -3802001.67, f(x) = 1445521672943.93823242
```

Gradient Descent with Two Variables

- Find minimma of this function using gradient descent algorithm:

$$f(x, y) = -0.4 + (x + 15)/30 + (y + 15)/40 + 0.5\sin(r), r = \sqrt{x^2 + y^2}$$

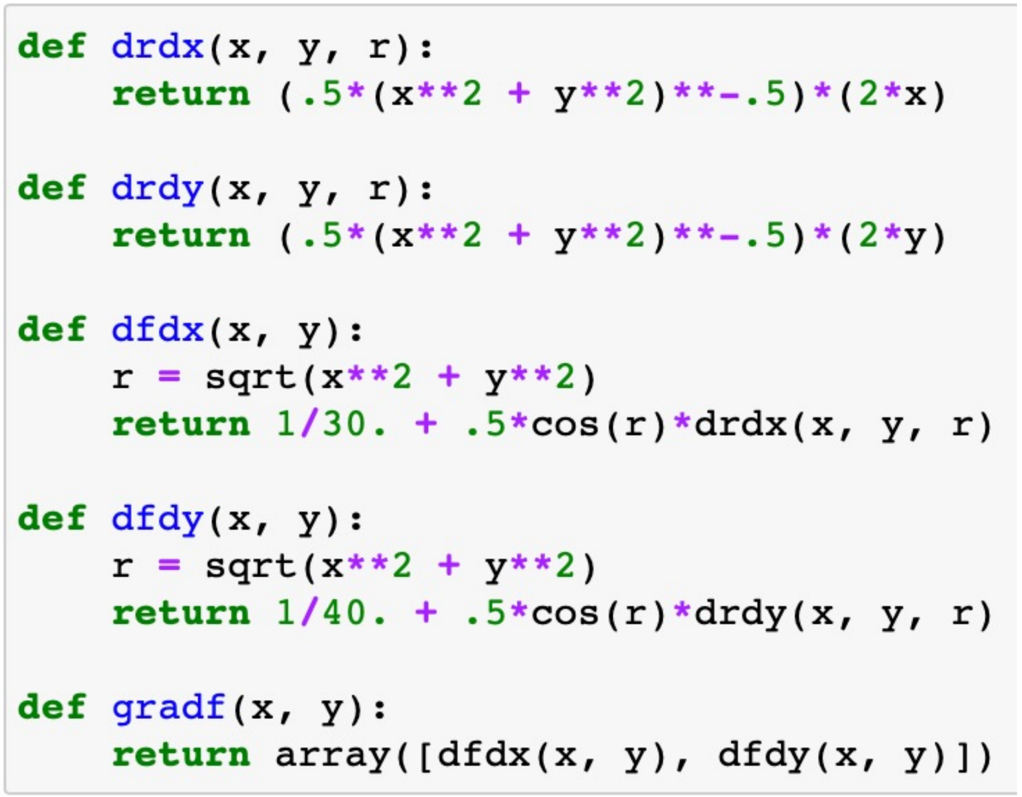
```
def f(x, y):
    r = sqrt(x**2 + y**2)
    return -.4 + (x+15)/30. + (y+15)/40.+.5*sin(r)
```

Partial Derivatives of the Function

- Partial derivatives w.r.t x:

$$\frac{\partial f}{\partial x} = 1/30 + 0.5\cos(r)\frac{\partial r}{\partial x}$$

$$\frac{\partial r}{\partial x} = (0.5(x^2 + y^2)^{-1/2})(2x)$$

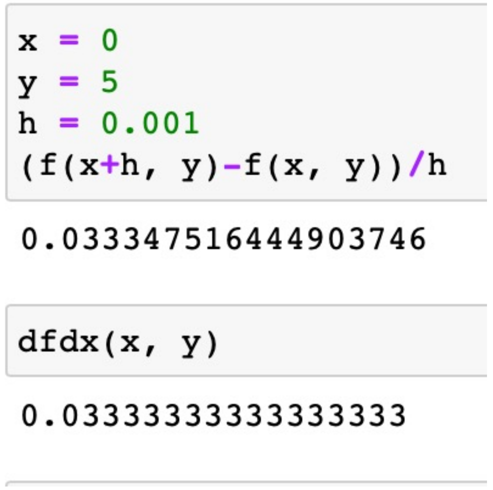


```python
def drdx(x, y, r):
    return (.5*(x**2 + y**2)**-.5)*(2*x)

def drdy(x, y, r):
    return (.5*(x**2 + y**2)**-.5)*(2*y)

def dfdx(x, y):
    r = sqrt(x**2 + y**2)
    return 1/30. + .5*cos(r)*drdx(x, y, r)

def dfdy(x, y):
    r = sqrt(x**2 + y**2)
    return 1/40. + .5*cos(r)*drdy(x, y, r)

def gradf(x, y):
    return array([dfdx(x, y), dfdy(x, y)])
```

Numerical Derivatives of Two Dimensional Functions

$$\frac{\partial f(x,y)}{\partial x} = \frac{f(x+\Delta x, y) - f(x,y)}{\Delta x}$$

$$\frac{\partial f(x,y)}{\partial y} = \frac{f(x, y+\Delta y) - f(x,y)}{\Delta y}$$

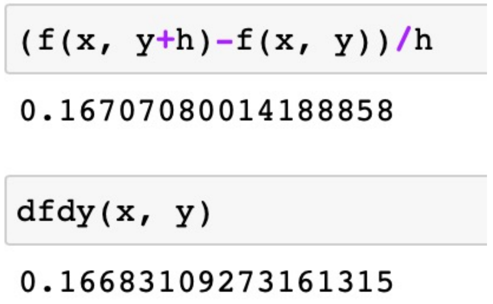


```
x = 0
y = 5
h = 0.001
(f(x+h, y)-f(x, y))/h
```

```
0.03334751644490374б
```

```
dfdx(x, y)
```

```
0.03333333333333333
```

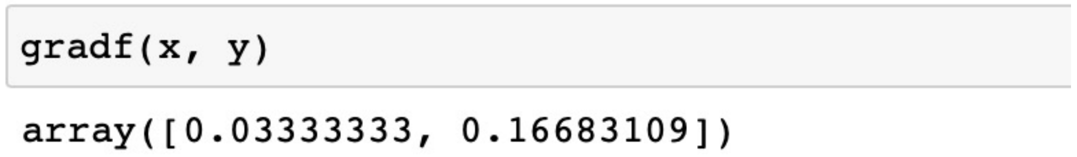


```
(f(x, y+h)-f(x, y))/h
```

```
0.16707080014188858
```

```
dfdy(x, y)
```

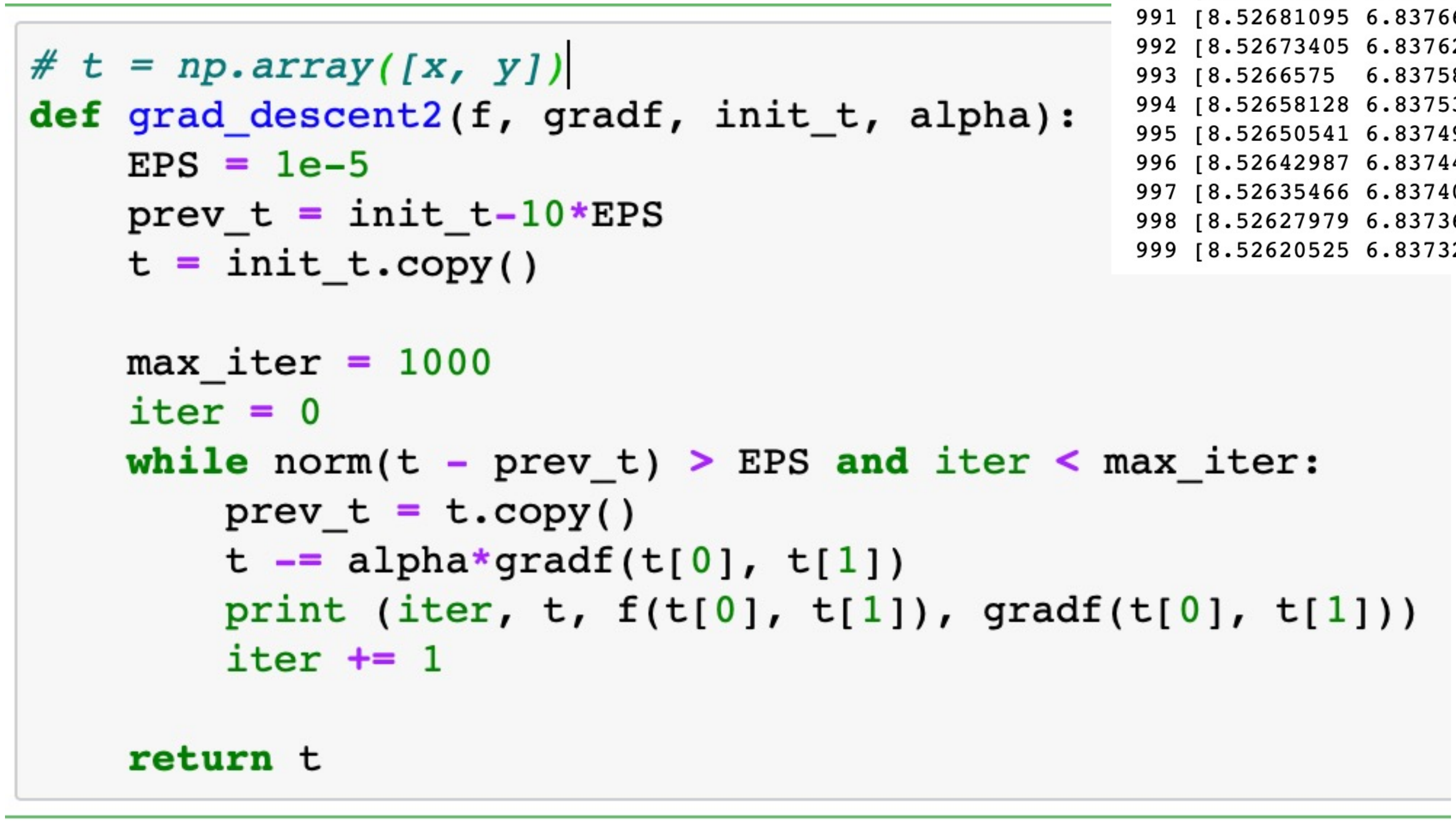
```
0.16683109273161315
```



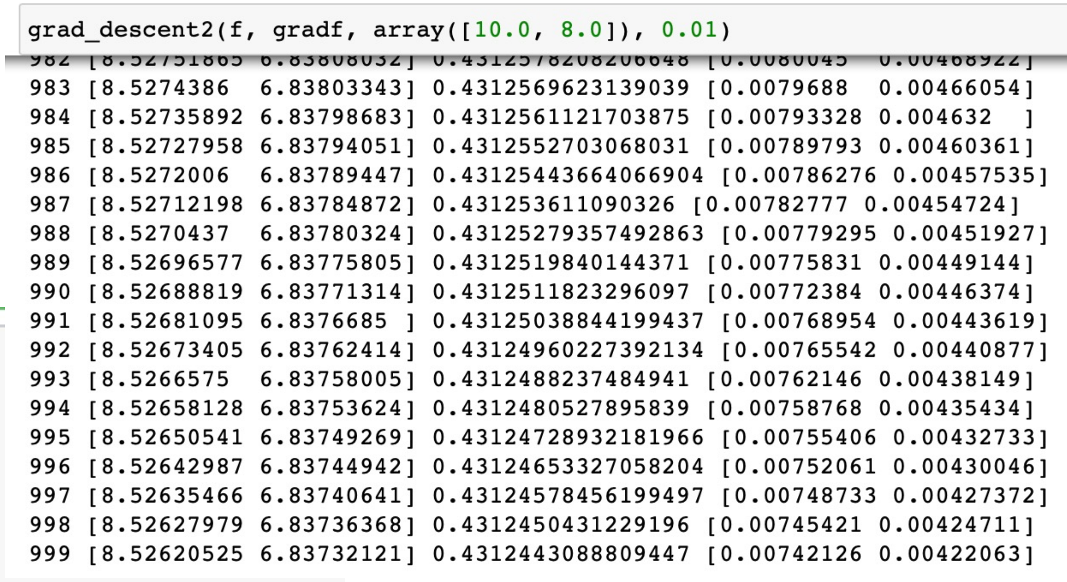
```
gradf(x, y)
```

```
array([0.03333333, 0.16683109])
```

2D Graduent Descent



```
# t = np.array([x, y])
def grad_descent2(f, gradf, init_t, alpha):
    EPS = 1e-5
    prev_t = init_t-10*EPS
    t = init_t.copy()

    max_iter = 1000
    iter = 0
    while norm(t - prev_t) > EPS and iter < max_iter:
        prev_t = t.copy()
        t -= alpha*gradf(t[0], t[1])
        print (iter, t, f(t[0], t[1]), gradf(t[0], t[1]))
        iter += 1

    return t
```



```
grad_descent2(f, gradf, array([10.0, 8.0]), 0.01)
```

```
982 [8.52751865 6.83808032] 0.4312578208206648 [0.0080045  0.00468922]
983 [8.5274386  6.83803343] 0.4312569623139039 [0.0079688  0.00466054]
984 [8.52735892 6.83798683] 0.4312561121703875 [0.00793328 0.004632  ]
985 [8.52727958 6.83794051] 0.4312552703068031 [0.00789793 0.00460361]
986 [8.5272006  6.83789447] 0.43125443664066904 [0.00786276 0.00457535]
987 [8.52712198 6.83784872] 0.431253611090326 [0.00782777 0.00454724]
988 [8.5270437  6.83780324] 0.43125279357492863 [0.00779295 0.00451927]
989 [8.52696577 6.83775805] 0.4312519840144371 [0.00775831 0.00449144]
990 [8.52688819 6.83771314] 0.4312511823296097 [0.00772384 0.00446374]
991 [8.52681095 6.8376685 ] 0.43125038844199437 [0.00768954 0.00443619]
992 [8.52673405 6.83762414] 0.43124960227392134 [0.00765542 0.00440877]
993 [8.5266575  6.83758005] 0.4312488237484941 [0.00762146 0.00438149]
994 [8.52658128 6.83753624] 0.4312480527895839 [0.00758768 0.00435434]
995 [8.52650541 6.83749269] 0.43124728932181966 [0.00755406 0.00432733]
996 [8.52642987 6.83744942] 0.43124653327058204 [0.00752061 0.00430046]
997 [8.52635466 6.83740641] 0.43124578456199497 [0.00748733 0.00427372]
998 [8.52627979 6.83736368] 0.4312450431229196 [0.00745421 0.00424711]
999 [8.52620525 6.83732121] 0.4312443088809447 [0.00742126 0.00422063]
```

Multiple variables (features) predict y

Size (feet²) x_1	Number of bedrooms x_2	Number of floors x_3	Age of home (years) x_4	Price (\$1000) y
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
…	…	…	…	…

$\boxed{x_0 = 1}$

Notation:

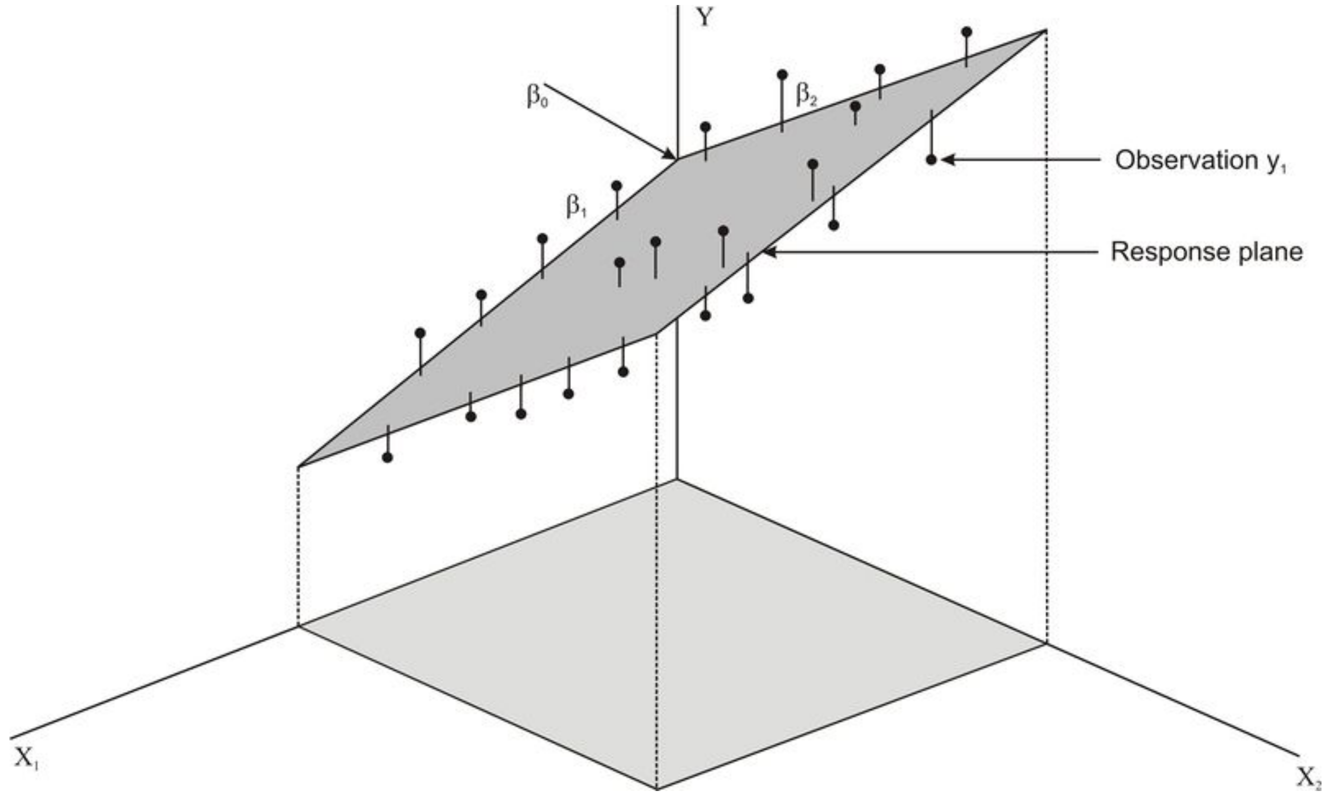
n = number of variables features

$x^{(i)}$ = input (features) of i^{th} training example.

$x_j^{(i)}$ = value of feature j in i^{th} training example.

$\boxed{x_0\theta_0 = \theta_0}$

$$\boxed{h_\theta(\boldsymbol{x}) = h_{\boldsymbol{\theta}}(x_0, x_1, x_2, \dots, x_n) = x_0\theta_0 + x_1\theta_1 + \cdots + x_n\theta_n = \boldsymbol{\theta}^{\boldsymbol{T}}\boldsymbol{x}}$$

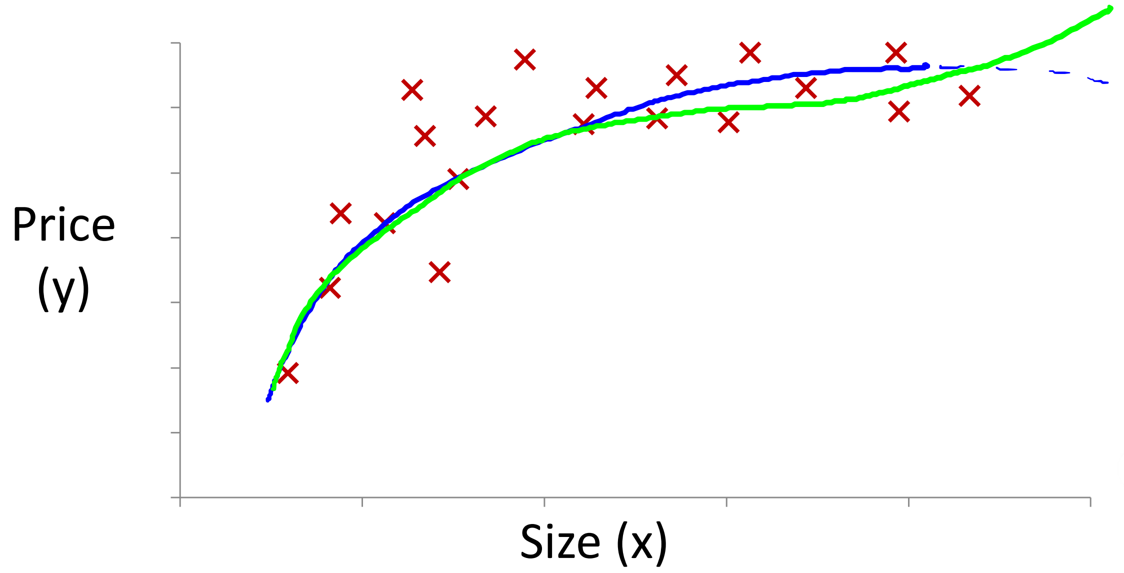


Minimize: $$J(\theta_0, \theta_1, \ldots, \theta_n) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Minimizing this cost function corresponds to minimizing the distance between
The observations and the hyperplane defined by $\boldsymbol{\theta}^{\boldsymbol{T}}\boldsymbol{X} - \boldsymbol{Y}$=0

Computed Features

It is sometimes useful to compute more features



$$\theta_0 + \theta_1 x + \theta_2 x^2$$

$$\boxed{\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3}$$

$$\begin{aligned} h_\theta(x) &= \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 \\ &= \theta_0 + \theta_1(size) + \theta_2(size)^2 + \theta_3(size)^3 \end{aligned}$$

$$\begin{aligned} x_1 &= (size) \\ x_2 &= (size)^2 \\ x_3 &= (size)^3 \end{aligned}$$

Much better fit than we would have gotten with linear regression

Computed Features Examples – cont’d

Basic Idea:

$$b_\theta(depth, frontage) = \theta_0 + \theta_1 depth + \theta_{\underset{2}{\cancel{1}}} frontage$$

depth
frontage

Better (if the price depends on the area):

$$h_\theta(depth, frontage) = \theta_0 + \theta_1 depth + \theta_2 frontage + \theta_3 (frontage \times depth)$$

Note: we could not represent the idea that the price is proportional to the area using The basic b_θ

Set:

$$\begin{aligned}&x_0: 1\\&x_1: depth\\&x_2: frontage\\&x_3: depth \times frontage\\&x_4: depth^2\\&\ldots\end{aligned}$$

(But there is a limit to how much we can do this without *overfitting:* more on that later), and find the best θ such that

$$h_\theta(\boldsymbol{x}) = \theta^T x$$