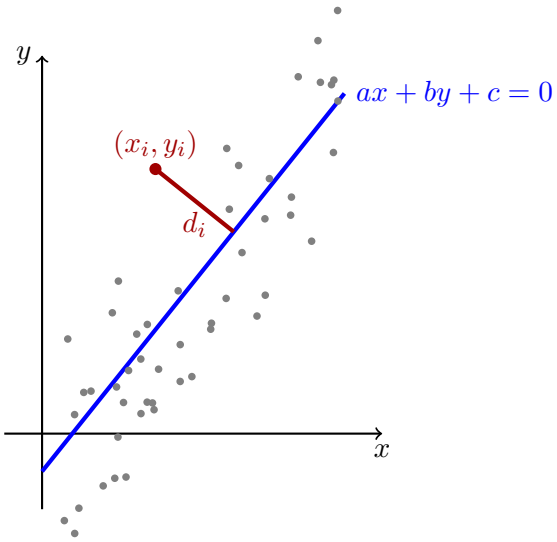
1 Background

1.1 Line Fitting

Today's session we will start working on optimization problems. The first problem we want to tackle is finding the best straight line that fits a set of n points $\{(x_i, y_i)\}_{i=1}^n$ in two dimensions



It is possible to show that the minimum distance d_i to any point i to the line with equation $ax+by+c = 0$ is given by

$$d_i^2 = \frac{(ax_i + by_i + c)^2}{a^2 + b^2}. \tag{1}$$

Note that finding the best values of (a, b, c) that describe the line is equivalent to reduce the distance between such line and the set of points, which then can be found by finding the minimum of the function

$$f(a, b, c) = \sum_{i=0}^{n} d_i^2 = \sum_{i=1}^{n} \frac{(ax_i + by_i + c)^2}{a^2 + b^2}. \tag{2}$$

1.2 Gradient Descent

The problem is reduced then to the general problem of finding the values of the vector $\boldsymbol{x}$ that minimize a function $g(\boldsymbol{x})$. It is possible to show that the vector $\nabla g(\boldsymbol{x})$ points along the direction of maximum variation of g at the location $\boldsymbol{x}$, that is if $\boldsymbol{x}_0$ is an initial guess close the position of the minimum value of g, then the vector

$$\boldsymbol{x}_1 = \boldsymbol{x}_0 - h\nabla g(\boldsymbol{x}_0), \tag{3}$$

will be even closer to the actual solution. The value of h is usually chosen to be a small number compared with the local variations of the function g. As an example consider the function

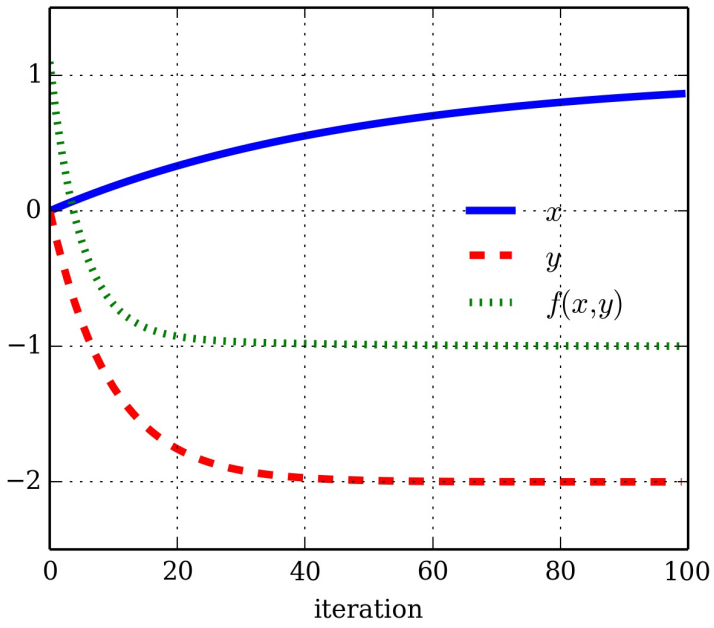
$$g(\boldsymbol{x}) = g(x, y) = 0.1(x - 1)^2 + 0.5(x + 2)^2 - 1. \tag{4}$$

The minimum value of g is -1 and it is reached at $(x, y) = (1, -2)$. In this example we will start with the guess $(x_0, y_0) = (0, 0)$ and with step-size $h = 0.1$.

The next code shows an implementation of this idea

```
x0, y0 = 0, 0
h = 0.1

def func(x, y):
    return -1 + 0.1 * (x - 1)**2 + 0.5 * (y + 2)**2

def grad(x, y):

    dfdx = 0.2 * (x - 1)
    dfdy = (y + 2)
    return dfdx, dfdy

n = 100
x, y = x0, y0

for i in range(n):

    f = func(x, y)
    dfdx, dfdy = grad(x, y)
    x = x - h * dfdx
    y = y - h * dfdx

    print i, x, y, f
```

It is interesting to see how indeed the values of x tend toward 1, while y approaches the value of -2



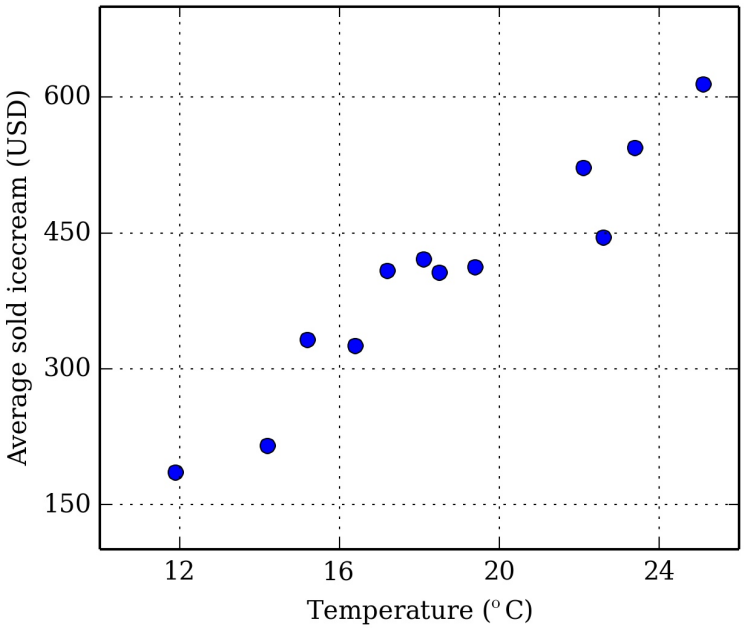
This plot shows this behavior graphically. In this example we run only for 100 iterations, but in reality we do not know how many iterations will it take to reach the final optimal solution. Instead another stopping criterion is preferred, notice for instead that the closer we get to the actual solution, the smaller is the gradient, we then opt to stop the algorithm when the the magnitude of the gradient becomes smaller than a given value $|\nabla g(\boldsymbol{x})| < \epsilon$, with ϵ being a small number.

2 Problems

(P1) In a given store the amount of sales S (in US dollars) of ice-creams seems to correlate with average temperature T

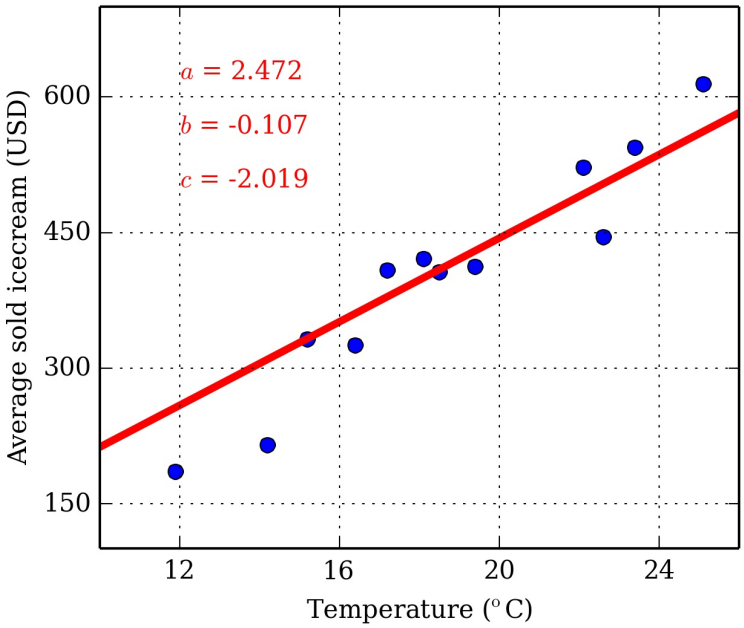
T (°C)	14.2	16.4	11.9	15.2	18.5	22.1	19.4	25.1	23.4	18.1	22.6	17.2
S (USD)	215	325	185	332	406	522	412	614	544	421	445	408

Make a plot of S vs T



(P2) Using Eq. (2) and its gradient find the optimal values of (a, b, c) that describe this dataset. To do that use $h = 10^{-6}$, start with $(a_0, b_0, c_0) = (3, -1, 0)$ and run the algorithm for $N_{\max} = 500000$ iterations

(P3) Overplot the best fit



(P4) What are the expected sales when the temperature is 20 °C?

```
>>> 443.736
```

3 Solutions

(P1)
```
T = np.array([14.2, 16.4, 11.9, 15.2, 18.5, 22.1, 19.4, 25.1, 23.4, 18.1, 22.6, 17.2
S = np.array([215, 325, 185, 332, 406, 522, 412, 614, 544, 421, 445, 408])
plt.plot(T, S, 'bo')
```

(P2)
```
def cost(a, b, c):

    f = (a * T + b * S + c)**2 / (a**2 + b**2)

    return f.sum()

def gradcost(a, b, c):

    dfda = 2 * T * (a * T + b * S + c) / (a**2 + b**2) \
    - 2 * a * (a * T + b * S + c)**2 / (a**2 + b**2)**2

    dfdb = 2 * S * (a * T + b * S + c) / (a**2 + b**2) \
    - 2 * b * (a * T + b * S + c)**2 / (a**2 + b**2)**2

    dfdc = 2 * (a * T + b * S + c) / (a**2 + b**2)

    return dfda.sum(), dfdb.sum(), dfdc.sum()

a, b, c = 3., -1., 0.
h = 1e-6
Nmax = 500000

i = 0

while True:

    f = cost(a, b, c)
    dfda, dfdb, dfdc = gradcost(a, b, c)

    a -= h * dfda
    b -= h * dfdb
    c -= h * dfdc

    i += 1
    if i == Nmax:
        break
```

(P3)
```
x = np.linspace(10, 30, 100)
y = ((-c - a * x) / b)
plt.plot(x, y)
plt.plot(T, S, 'bo')
```

(P4)
```
print (-c - a * (20)) / b
```