Minimization

Matteo Sani
matteosan1@gmail.com

November 13, 2019

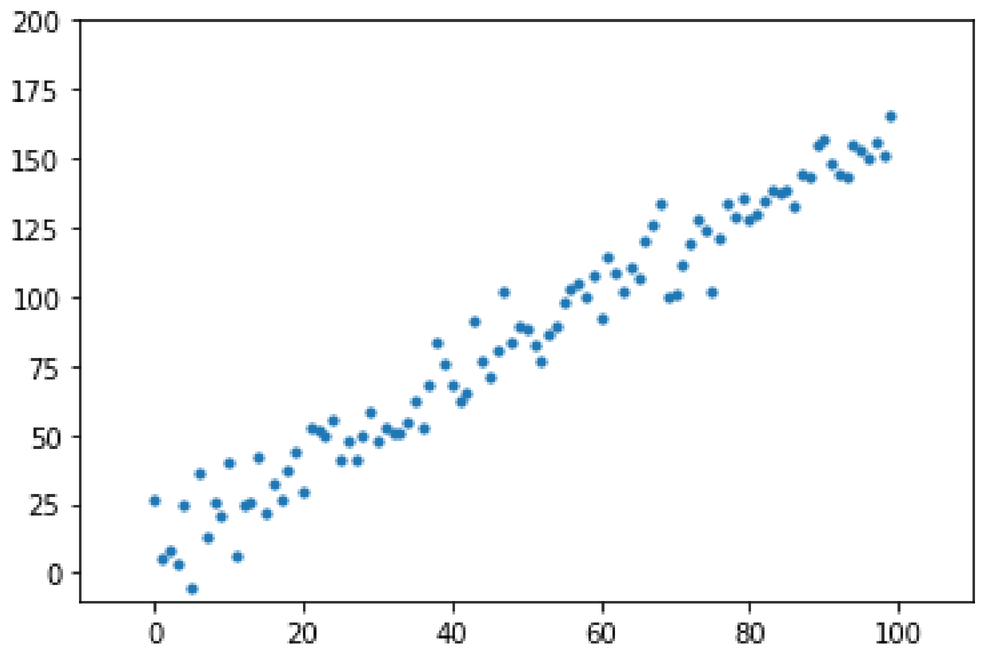
1 Minimization Example

1.1 How Minimization Works

With a simple example I would like to clarify how the minimization algorithm works. This is essentially the same algorithm used in the *bootstrap* except it is applied to a simpler case.

So let's assume we have a bunch of points and we want to detrmine the straight line that best fit to these points. These are the points:

[4]:
```
from numpy.random import randn, seed
from matplotlib import pyplot as plt

def generate_points(m, q, sigma=10):
    seed(1)
    points = []
    for i in range(100):
        points.append(m*i + q + randn()*sigma)

    return points

x_points = range(100)
y_points = generate_points(1.5, 10.0)
plt.scatter(x_points, y_points, s=10)
plt.xlim(-10, 110)
plt.ylim(-10, 200)
plt.show()
```



As said we would like to detrmine the "best" line to this points with a minimization algorithm (i.e. to find the paramters that best fit).

A straight line can be parametrize as follows:

$$y = m \cdot x + q \qquad m \text{ is the angular coefficient, } q \text{ the offset}$$

(In our example the points have been generated around a line with $m = 1.5$ and $q = 10$. So let's write a class that represents a straight line with this parametrization.

[11]:
```python
class Line:
    def __init__(self, m, q):
        # the only two parameters are m and q
        self.m = m
        self.q = q

    # this method is just to plot the line (returns Ys given Xs)
    def plot(self, r):
        points = []
        for x in r:
            points.append(self.m*x + self.q)

        return points
```

As in the *bootstrap* our goal is to find the best set of parameters (m and q) according to some input data (our points). An important issue is to define a criteria that tells us how good is a set of parameters (and to finally find the best).

In this case we define the best set the one that minimize *the sum of the squared distances of the line to each point*:

$$\sum_i^n (y_i - m \cdot x_i - q)^2$$

(in the bootstrap we use the fairness of the contracts, so we minimize the sum of the squared npvs, indeed fair implies npv=0). So we can define the *objective function*, the function that is actually minimized:

[8]:
```
def obj_function(x):
    line = Line(x[0], x[1])

    # here we compute the "distance" between line and points
    sum = 0
    for i, x in enumerate(x_points):
        sum += (y_points[i] - line.m*x - line.q)**2

    return sum
```

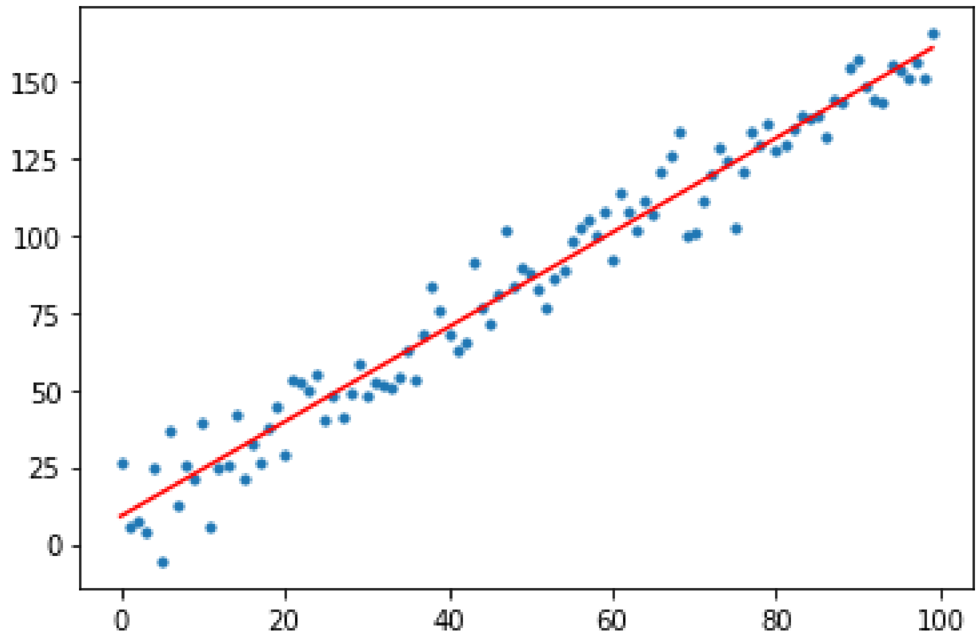
Finally we can run the minimization using `scipy.optimize.minimize`:

[12]:
```
from scipy.optimize import minimize

# set the initial guess m=0 and q=0
x_guess = [0, 0]

# set some limits for m and q
bounds = [(-3, 3), (-10, 10)]

# run the minimization
result = minimize(obj_function, x_guess, bounds=bounds)

print (result)
```

```
      fun: 7741.957339264212
 hess_inv: <2x2 LbfgsInvHessProduct with dtype=float64>
      jac: array([0.0005457, 0.0003638])
  message: b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'
     nfev: 30
      nit: 5
   status: 0
  success: True
        x: array([1.53341864, 8.95160579])
```

As you can see we have found parameters pretty close to our simulation $m = 1.53$ and $q = 8.95$.

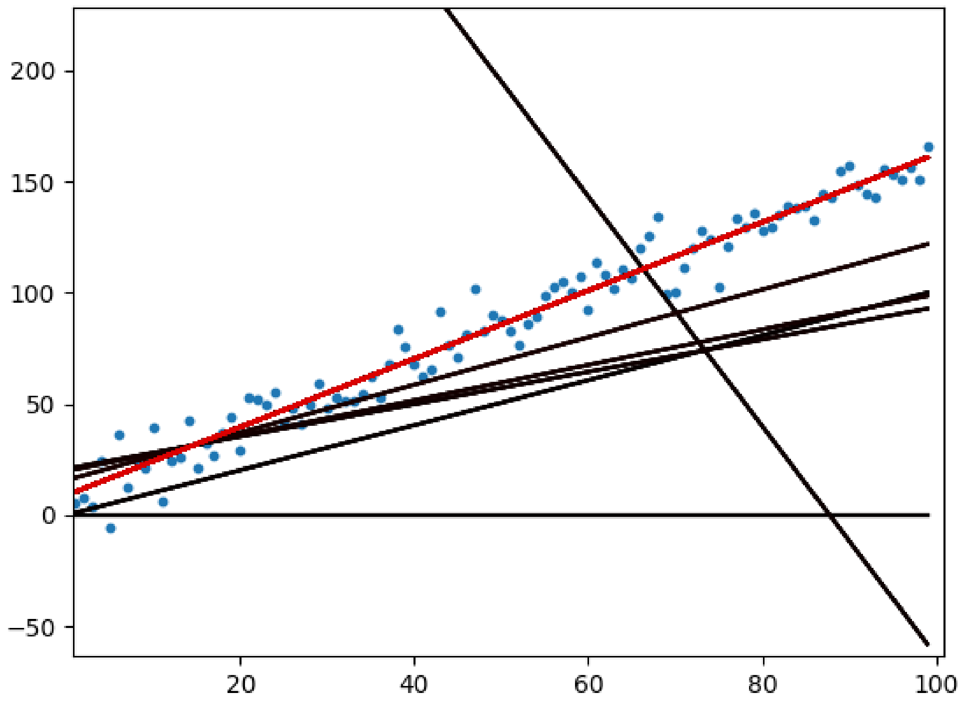
```
[13]: plt.scatter(range(100), points, s=10)
      line = Line(result.x[0], result.x[1])
      plt.plot(xs, line.plot(xs), color=(1, 0, 0))
      plt.show()
```



What actually does the minimization algorithm:

- Starts with our initial guess of the parameters and compute the objective function value;
- Move the parameters to reduce the value of the objective function;
- Repeat above step until further changes of the parameters do not bring additional reduction of the objective function.

Graphically we can check in this simple case what happened during the minimization plotting the points and the lines *tested* at the above step 2 during each iteration:



We can now compare side by side this example and the bootstrap of a `CreditCurve` or `DiscountCurve`.

	Example	`CreditCurve`	`DiscountCurve`
Inputs	X points, Y points	maturity dates market quotes CDS	maturity dates market quotes OIS
Output	m, q of a line	surv. prob. @ maturity dates	disc. factors @ maturity dates
Objective Func.	$\sum dist^2_{p_i l}$	$\sum_{\text{CDS}} \text{NPV}^2$	$\sum_{\text{OIS}} \text{NPV}^2$