22.03.2018

Вычислительные модели с использованием научных библиотек Python

Методы оптимизации

Постановка задачи

$$f(x) \to \min, \quad x \in X.$$

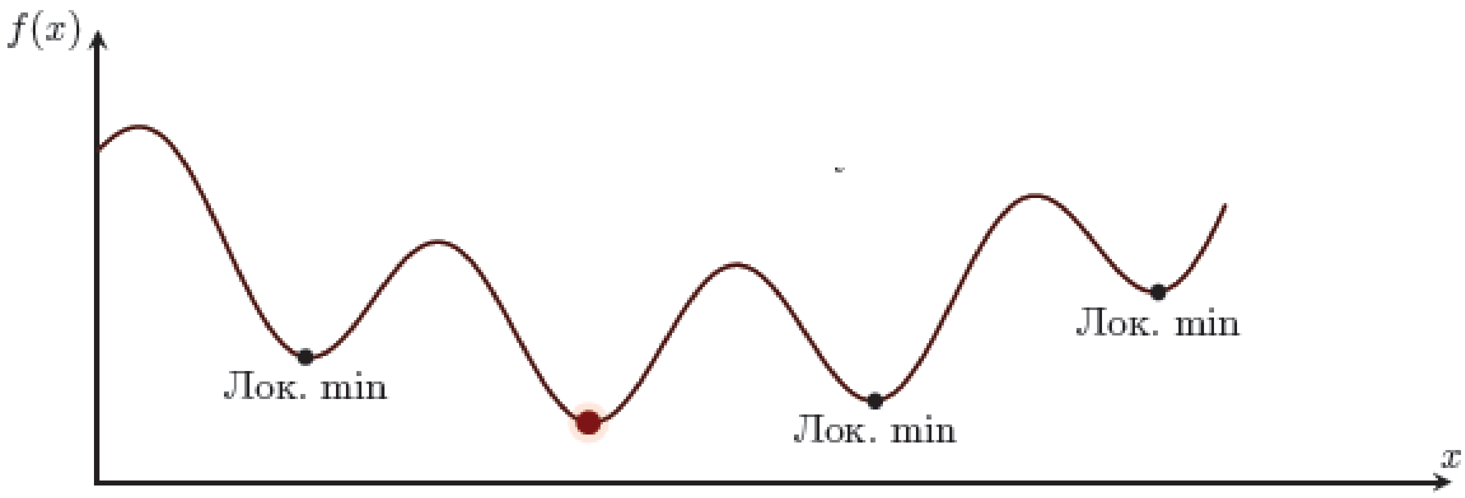
<u>Глобальный минимум</u>

$$f(x^*) \leqslant f(x) \quad \text{при всех} \quad x \in X;$$

<u>Локальный минимум</u>

$$f(x^*) \leqslant f(x) \quad \text{при всех} \quad x \in X \cap U_\varepsilon(x^*),$$

$$\text{где} \;\; U_\varepsilon(x^*) = \{x \in \mathbf{R}^n \mid \|x - x^*\| \leqslant \varepsilon\}$$



Градиентный метод

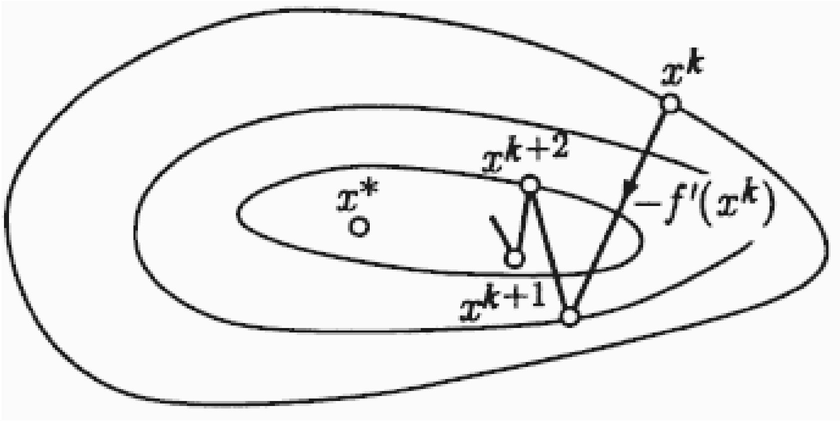
$$f(x) \to \min, \quad x \in X.$$

$$x^{k+1} = x^k - \alpha_k f'(x^k), \quad \alpha_k > 0, \quad k = 0, 1, 2, \ldots$$

Скорость сходимости

$$f(x^k) - f(x^*) \leqslant q^k (f(x^0) - f(x^*)),$$

$$\|x^k - x^*\| \leqslant C(\sqrt{q})^k,$$



Метод Ньютона

$$f(x) \to \min, \quad x \in X.$$

$$f(x) - f(x^k) = \langle f'(x^k), x - x^k \rangle + \frac{1}{2} \langle f''(x^k)(x - x^k), x - x^k \rangle + \\ + o(\|x - x^k\|^2).$$

$$x^{k+1} = x^k + h^k, \quad h^k = -(f''(x^k))^{-1} f'(x^k).$$

<u>Скорость сходимости</u>

$$\|x^k - x^*\| \leqslant \frac{1}{2\theta} \|f'(x^k)\|$$

Метод сопряженных градиентов (CG)

$$f(x) = \frac{1}{2}\langle Ax,\, x\rangle + \langle b,\, x\rangle$$

$$h^0 = -f'(x^0), \quad h^k = -f'(x^k) + \beta_{k-1} h^{k-1}, \quad k \geqslant 1.$$

$$\beta_{k-1} = \frac{\langle f'(x^k),\, Ah^{k-1}\rangle}{\langle h^{k-1},\, Ah^{k-1}\rangle}.$$

Для квадратичного функционала сходимость за n шагов

Алгоритм для общего случая

Given x_0;

Evaluate $f_0 = f(x_0), \nabla f_0 = \nabla f(x_0)$;

Set $p_0 = -\nabla f_0, k \leftarrow 0$;

while $\nabla f_k \neq 0$

Compute α_k and set $x_{k+1} = x_k + \alpha_k p_k$;

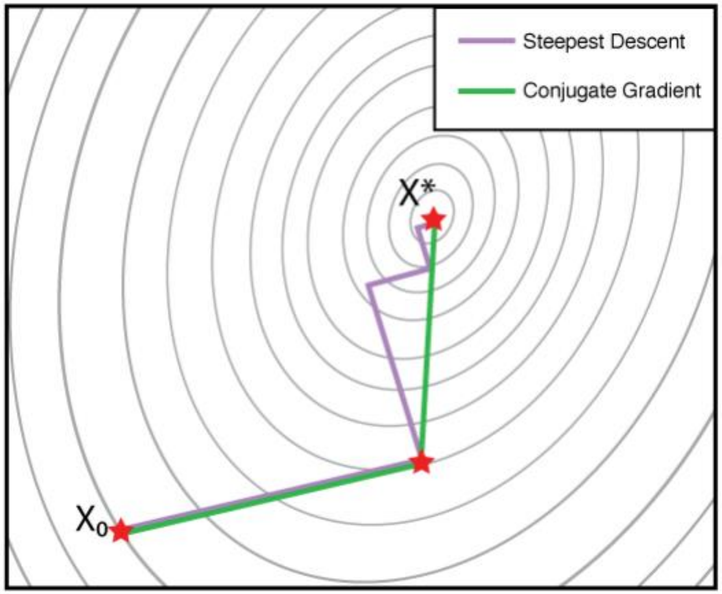
Evaluate ∇f_{k+1};

$$\beta_{k+1}^{\text{PR}} = \frac{\nabla f_{k+1}^T (\nabla f_{k+1} - \nabla f_k)}{\|\nabla f_k\|^2}$$

$$p_{k+1} \leftarrow -\nabla f_{k+1} + \beta_{k+1}^{\text{FR}} p_k;$$

$$k \leftarrow k + 1;$$

end (while)



Метод Бройдена – Флетчера-Гольдфарба – Шанно (BFGS)

$$f(x_k + p) = f(x_k) + \nabla f^T(x_k)p + \frac{1}{2}p^T H(x_k)p$$

$$p_k = -B_k^{-1}\nabla f(x_k)$$

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k}, \qquad y_k = \nabla f_{k+1} - \nabla f_k$$

$$s_k = x_{k+1} - x_k$$

$$C_k = B_k^{-1}$$

$$C_{k+1} = (I - \rho_k s_k y_k^T)C_k(I - \rho_k y_k s_k^T) + \rho_k s_k s_k^T,$$

$$\rho_k = \frac{1}{y_k^T s_k}.$$

Метод Пауэлла

АЛГОРИТМ

Шаг 1. Задать x_1 и шаг $h > 0$, точность $\varepsilon > 0$.

Шаг 2. Найти $x_2 = x_1 + h$, вычислить $f(x_1)$ и $f(x_2)$

Шаг 3 Если $f(x_1) > f(x_2)$, то $x_3 = x_1 + 2h$, иначе $x_3 = x_1 - h$.

Шаг 4. Вычислить $f(x_3)$; определить $F_{\min} = \min\{f(x_1), f(x_2), f(x_3)\}$,

определить соответствующее $x_{\min}$.

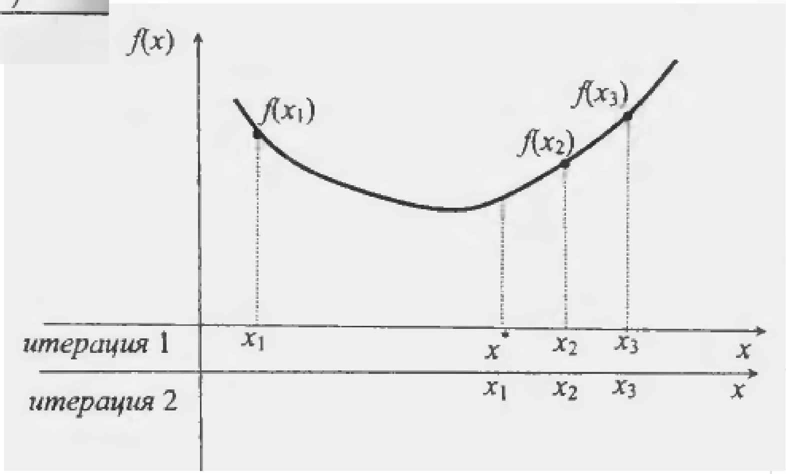
Шаг 5. Найти x^*

Шаг 6 Если $|x^* - x_{\min}| < \varepsilon$, то поиск окончен x^* является оценкой оптимума,

иначе перейти к шагу 7.

Шаг 7. Вычислить $f(x^*)$, если $f(x^*) < F_{\min}$, то $x_1 = x^*$, иначе $x_1 = x_{\min}$.

$$x^* = -\frac{1}{2}\frac{\left((x_2)^2 - (x_3)^2\right) f(x_1) + \left((x_3)^2 - (x_1)^2\right) f(x_2) + \left((x_1)^2 - (x_2)^2\right) f(x_3)}{(x_2 - x_3) f(x_1) + (x_3 - x_1) f(x_2) + (x_1 - x_2) f(x_3)}$$



Методы глобальной оптимизации

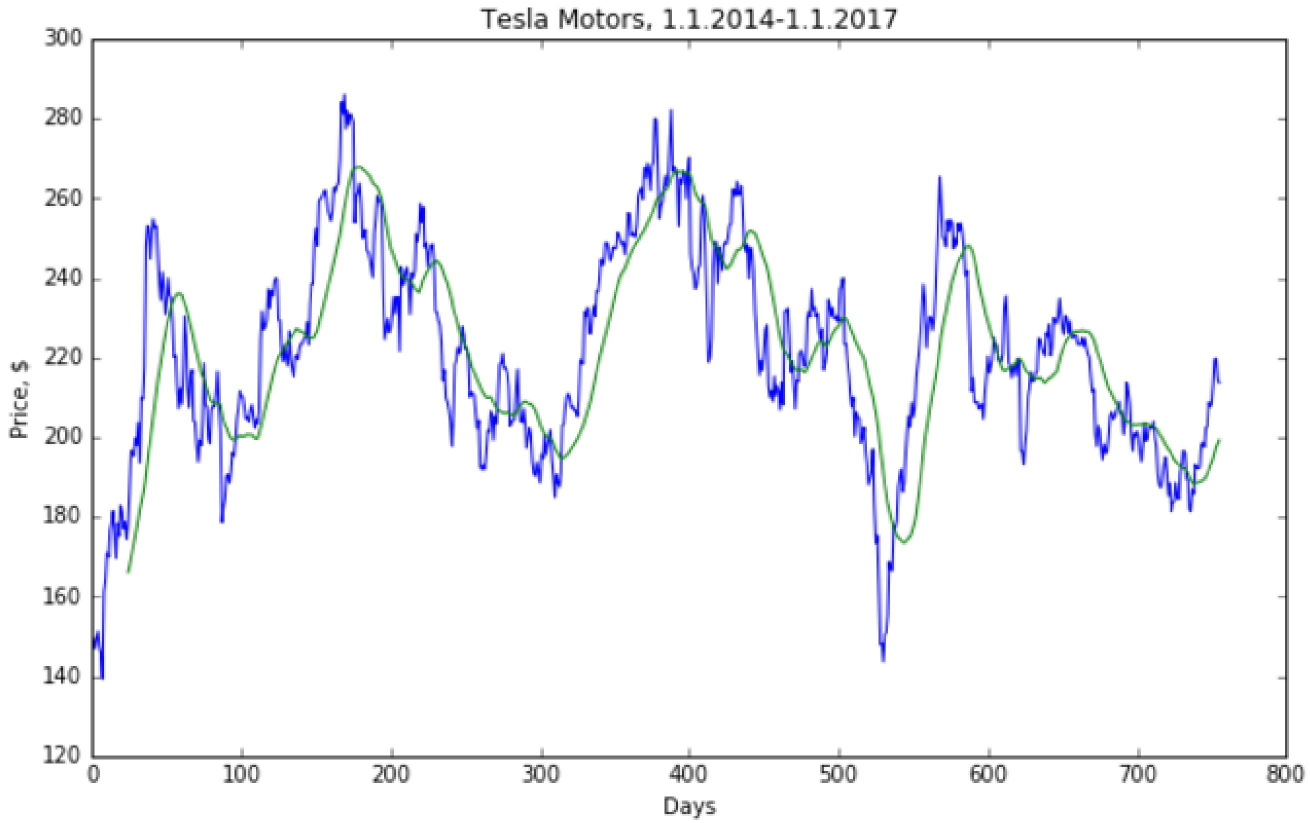
Basin-hopping

```
>>> from scipy.optimize import basinhopping
>>> def func2d(x):
...         f = np.cos(14.5 * x[0] - 0.3) + (x[1] + 0.2) * x[1] +
(x[0] + 0.2) * x[0]
...         df = np.zeros(2)
...         df[0] = -14.5 * np.sin(14.5 * x[0] - 0.3) + 2. * x[0]
+ 0.2
...         df[1] = 2. * x[1] + 0.2
...     return f, df
>>> minimizer_kwargs = {"method":"L-BFGS-B",
"jac":True}
>>> x0 = [1.0, 1.0]
>>> ret = basinhopping(func2d, x0,
minimizer_kwargs=minimizer_kwargs, ... niter=200)
>>> print("global minimum: x = [%.4f, %.4f], f(x0) =
%.4f" % (ret.x[0], ret.x[1], ret.fun))
global minimum: x = [-0.1951, -0.1000], f(x0) = -1.0109
```

Differential-evolution

```
>>> from scipy.optimize import differential_evolution
>>> import numpy as np
>>> def ackley(x):
...         arg1 = -0.2 * np.sqrt(0.5 * (x[0] ** 2 + x[1] ** 2))
...         arg2 = 0.5 * (np.cos(2. * np.pi * x[0]) + np.cos(2.
* np.pi * x[1]))
... return -20. * np.exp(arg1) - np.exp(arg2) + 20. + np.e
>>> bounds = [(-5, 5), (-5, 5)]
>>> result = differential_evolution(ackley, bounds)
>>> result.x, result.fun (array([ 0., 0.]),
4.440892098500626e-16)
```

Оптимизация торговой стратегии



- Покупать: цена поднимается выше скользящей средней
- Продавать: цена опускается ниже скользящей средней

Задача: определить оптимальное количество значений для расчета скользящей средней, долю от банка на покупку, долю бумаг на продажу