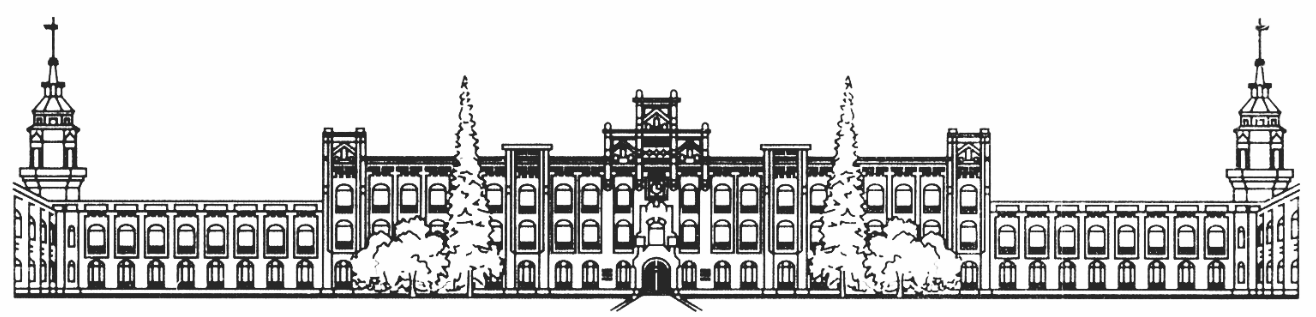


Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут ім. І. Сікорського»
Інститут прикладного системного аналізу

Лабораторна робота
з курсу «Методи оптимізації»
з теми «Чисельні методи безумовної оптимізації другого порядку. Метод Ньютона та його варіації»

Виконали студенти 3 курсу групи КА-81
Галганов Олексій
Єрко Андрій
Фордуй Нікіта

Перевірили
Спекторський Ігор Якович
Яковлева Алла Петрівна

Київ 2021

Варіант 1

Завдання. Скласти програму для мінімізації цільової функції одним з методів другого порядку (типу Ньютона). Конкретний тип методу обрати самостійно, ураховуючи особливості цільової функції.

Цільова функція: $f(x, y) = x^2 + 18y^2 + 0.01xy + x - y$

Результати роботи. Цільова функція є квадратичною, а тому використовувати узагальнені методи немає сенсу, оскільки класичний метод Ньютона $x^{k+1} = x^k - [f''(x^k)]^{-1} f'(x^k)$ для строго опуклих квадратичних функцій знаходить розв'язок за одну ітерацію. Доведемо це твердження.

Розглядаємо строго опуклі f виду

$$f(x_1, ..., x_n) = \frac{1}{2}(Ax, x) + (b, x) + c, \ A > 0$$

Тоді

$$f'(x_1, ..., x_n) = Ax + b$$

$$f''(x, y) = A$$

Нехай x^0 — довільне початкове наближення, x^* — шуканий розв'язок, єдиний з умови строгої опуклості. Запишемо перший крок метода Ньютона:

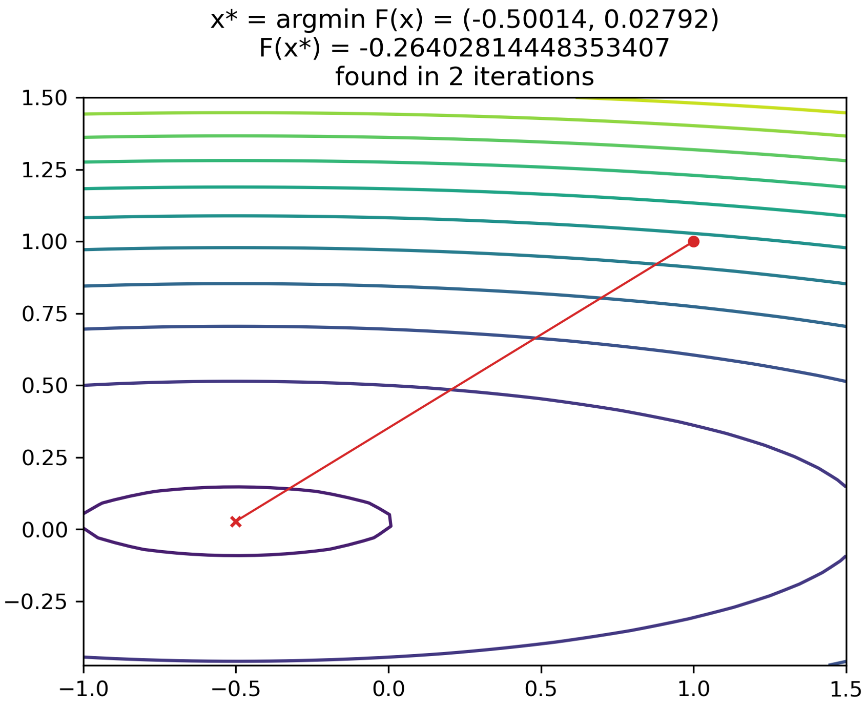
$$x^1 = x^0 - [f''(x^0)]^{-1} f'(x^0)$$

$$x^1 = x^0 - A^{-1}(Ax_0 + b)$$

$$x^1 = A^{-1}b$$

Отже, x^1 — розв'язок рівняння $Ax + b = 0$, що задає необхідну умову точки мінімуму f. Відомо, що у випадку мінімізації опуклої функції на опуклій множині (зокрема, на всьому просторі) необхідна умова мінімуму є достатньою. Звідки отримуємо рівність $x^1 = x^*$, тобто точку мінімуму отримано після виконання одного кроку. $\square$

Оскільки обраний критерій зупинки — $|f(x_{k+1}) - f(x_k)| < \varepsilon = 10^{-5}$ вимагає принаймні дві ітерації для порівняння, було отримано збіжність до точки мінімуму за 2 ітерації.



Лістинг. Текст програми було розділено на `Optimizer.py` з реалізацією власне методу Ньютона та `lab2.py`, де він викликається та зберігаються результати.

`Optimizer.py`

```
import itertools
import numpy as np


class NewtonOptimizer:
    def __init__(self, beta = 1, tol = 1e-5, max_iter = 100):
        """
        Parameteres
        -----------
        beta       : float
                     step parameter (default 1)
        max_iter   : int
                     maximum number of iterations,
                     default is 100
        tol        : tolerance for algorithm stopping
                     |f(x_prev) - f(x_new)| < tol
        """
        self.beta = beta
        self.tol = tol
        self.max_iter = int(max_iter)

    def minimize(self, target_func, x0, h = 0.005):
        """
        Parameteres
        -----------
        target_func : callable
                      function to minimize
        x0          : np.array of shape (n, )
                      initial point
        h           : step for computing gradients,
                      default is 0.005
        -----------
        Returns history - np.array with shape (n_iter, n+1), n - number of
    variables;
                history[:, -1] - values of target functions
                history[-1, :-1] - solution
```

```
                history[-1, -1] - value of target function at solution point
        """
        def compute_grad(target_func, x, h):
            n = len(x)
            grad = np.zeros_like(x)
            for i in range(n):
                x_plus = np.copy(x)
                x_plus[i] += h
                x_minus = np.copy(x)
                x_minus[i] -= h
                grad[i] = (target_func(x_plus) - target_func(x_minus))/(2*h)
            return grad

        def compute_hessian(target_func, x, h):
            n = len(x)
            hessian = np.empty((n,n))
            for k in range(n):
                for m in range(k+1):
                    dx = np.zeros_like(x)
                    dx[k] = h/2
                    dy = np.zeros_like(x)
                    dy[m] = h/2

                    # k=m:  (f(x+2h) - 2f(x) + f(x-2h)) / 4h^2
                    # k!=m: (f(x+h,y+h) - f(x+h,y-h) - f(x-h,y+h) + f(x-h,y-h))
    / 4h^2
                    hessian[k,m] = sum([i*j * target_func(x + i*dx + j*dy)
                                        for i,j in itertools.product([1,-1],
   repeat=2)
                                        ]) / h**2
                    hessian[m,k] = hessian[k,m]
            return hessian

        history = []
        x = np.copy(x0)
        history.append([*x0, target_func(x0)])
        for k in range(self.max_iter):
            grad = compute_grad(target_func, x, h)
            hessian = compute_hessian(target_func, x, h)
            alpha = self.beta
            x = x - alpha * np.linalg.pinv(hessian) @ grad  # x = x + (-1)*a*[f
   "]^(-1)*f'
            history.append([*x, target_func(x)])
            if k > 0 and np.abs(history[-1][1] - history[-2][1]) < self.tol:
                break
        return np.array(history)
```

lab1.py

```
import numpy as np
from Optimizer import NewtonOptimizer
from Plotter import PlotContour, PlotSurface

# define target function
def f(x_vect):
    x, y = x_vect[0], x_vect[1]
    return x**2 + 18*y**2 + 0.01*x*y + x - y

# set optimizer parameteres and create it
opt_params = {
    'beta' : 1,
    'tol' : 1e-5,
    'max_iter' : 100
}
opt = NewtonOptimizer(**opt_params)
```

```python
# initial point
x0 = np.array([1, 1], dtype=np.float64)
hist = opt.minimize(f, x0)
print(hist)

# save results
x_min, x_max = np.min(hist[:, 0])-0.5, np.max(hist[:, 0])+0.5
y_min, y_max = np.min(hist[:, 1])-0.5, np.max(hist[:, 1])+0.5
PlotContour((x_min, x_max), (y_min, y_max), f, fname = '../latex/pics/
    contour_init.png')
PlotSurface((x_min, x_max), (y_min, y_max), f, fname = '../latex/pics/
    surface_init.png')
PlotContour((x_min, x_max), (y_min, y_max), f, hist, fname = '../latex/pics/
    contour_final.png')
PlotSurface((x_min, x_max), (y_min, y_max), f, hist, fname = '../latex/pics/
    surface_final.png')
```

Висновки. При виконанні даної роботи ми дослідили застосування методу Ньютона до мінімізації заданої функції. Оскільки мінімум строго опуклої квадратичної функції може бути знайдений класичним методом Ньютона за одну ітерацію, було реалізовано саме його, а також — доведено відповідний факт.