Задание 4. Алгоритмы безусловной нелинейной оптимизации. Стохастические и метаэвристические алгоритмы

Варвара Кошман, C4113, 26.12.2019

Задача: аппроксимация сгенерированных данных функцией, полученной

a) алгоритмом Нелдера-Мида
b) алгоритмом Левенберга-Марквардта
c) алгоритмом дифференциальной эволюции

Массив зашумленных данных сгенерирован по правилу:

$y_k = -100 + \delta_k,\ f(x_k) < -100,$

$y_k = f(x_k) + \delta_k, -100 \le f(x_k) \le 100,$

$y_k = 100 + \delta_k,\ f(x_k) > 100,$

$x_k = \frac{3k}{1000},$

где $k = 0, \ldots, 1000$, $\delta_k \sim N(0,1)$.

Функция находится путем минимизации функционала: $D(a,b,c,d) = \sum_{k=0}^{1000} (F(x_k, a, b, c, d) - y_k))^2$ (с точностью ε=0.001) при

$F(x,a,b) = ax + b/(x^2 + cx + d)$ (рациональная аппроксимирующая функция);

Общий код:

```
import numpy as np
import scipy.optimize as sp

from Scripts.utility import plot_dataset, epsilon, x_s, y_s, maxiter


rational_reg_f = lambda a, b, c, d: np.array((a * x_s + b) / (x_s * x_s + c * x_s + d))


def functional_rational(point):
    return np.sum(np.array([np.square((point[0] * x_s + point[1]) / (
            x_s * x_s + point[2] * x_s + point[3]) - y_s)]))


def helper_f(point):
    err = f(point) - y_s
    return err


def f(point):
    return (x_s * point[0] + point[1]) / (x_s * x_s + point[2] * x_s + point[3])


def main():
    x0 = np.array((np.random.rand(1)[0], np.random.rand(1)[0], np.random.rand(1)[0],
                   np.random.rand(1)[0]))  # initial guess
```

```
    nm_res = sp.minimize(functional_rational,
                         x0,
                         method="Nelder-Mead",
                         options={'xtol': 1e-3, 'ftol': 1e-3, 'maxiter': maxiter})
    nm_iter = nm_res.nit
    optimal_nm = nm_res.x
    print("coefficients with Nelder-Mead algorithm with ({} iterations): "
          .format(nm_iter), optimal_nm)
    print("functional value at found point (nm)", functional_rational(optimal_nm))
    print()

    lm_res = sp.least_squares(helper_f, x0,
                              method="lm",
                              max_nfev=maxiter,
                              xtol=epsilon)
    optimal_lm = lm_res.x
    lm_iterations = lm_res.nfev
    print("coefficients with Levenberg-Marquardt algorithm with ({} iterations): "
          .format(lm_iterations), optimal_lm)
    print("functional value at found point (lm)", functional_rational(optimal_lm))
    print()

    bounds = ((-3, 3), (-3, 3), (-3, 3), (-3, 3))
    res = sp.differential_evolution(functional_rational,
                                    bounds,
                                    maxiter=maxiter,
                                    tol=epsilon)
    optimal_de = res.x
    de_iterations = res.nit
    print("coefficients with Differential evolution algorithm with ({} iterations): "
          .format(de_iterations), optimal_de)
    print("functional value at found point (de)", functional_rational(optimal_de))
    print()

    plot_dataset(rational_reg_f, optimal_nm, optimal_lm, optimal_de)


if __name__ == '__main__':
    main()
```

Вывод программы (приведены несколько запусков):

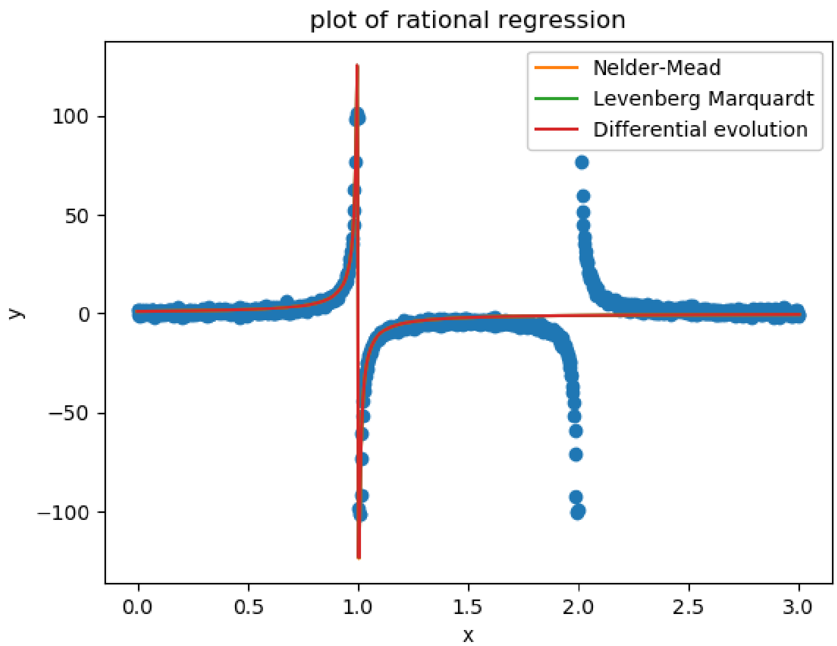
Оценивая работу в среднем, можно выделить 3 основных случая поведения:

1)все методы сходятся к одним значениям, различие в сумме квадратов невязок можно считать незначительными, на графиках линии регрессии для методов совпадают, алгоритмы Левенверга-Маквардта и метод дифференциальной эволюции по числу итераций в несколько раз быстрее сходятся, чем алгоритм Нелдера-Мида.

coefficients with Nelder-Mead algorithm with (505 iterations): [-1.00809623 1.00860099 -2.00097303
1.00098953]
functional value at found point (nm) 135481.41195102307

coefficients with Levenberg-Marquardt algorithm with (111 iterations): [-1.01065979 1.01114686 -2.00089705 1.00091364]
functional value at found point (lm) 135485.1456153138

coefficients with Differential evolution algorithm with (121 iterations): [-1.00505518 1.00553723 -2.00088878 1.00090513]
functional value at found point (de) 135485.9300668714

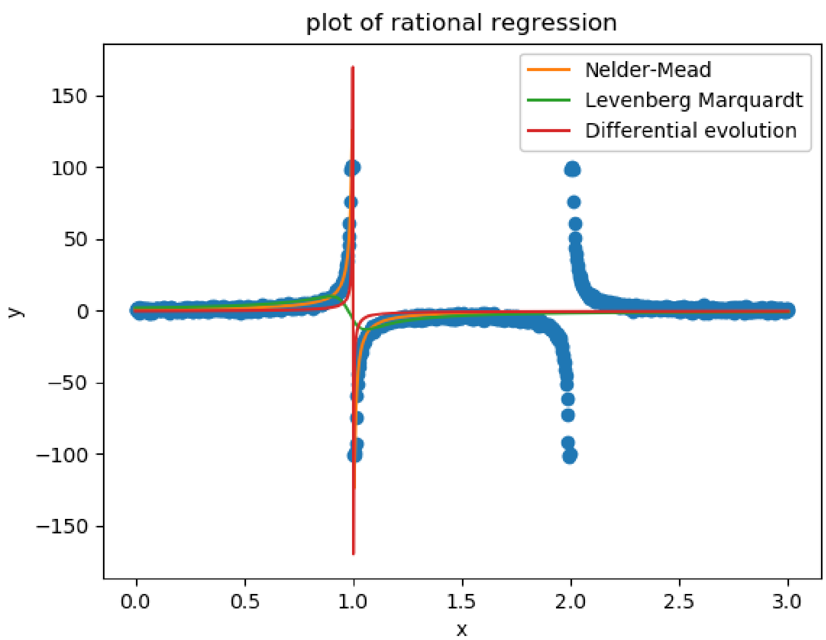


2) алгоритм Левенберга-Марквардта сходится быстро, однако с заметно большой суммой квадратов невязок, на графике видно, что данные им аппроксимируются плохо.

coefficients with Nelder-Mead algorithm with (482 iterations): [-1.0064518   1.00694651 -2.00092061 1.00093684]
functional value at found point (nm) 136625.79675749032

coefficients with Levenberg-Marquardt algorithm with (63 iterations): [-1.86534257 1.82328314 -1.97655941 0.98329521]
functional value at found point (lm) 245856.46406766662

coefficients with Differential evolution algorithm with (99 iterations): [-1.536093   0.52048551 1.99311796 -2.99510566]
functional value at found point (de) 198004.2213224115

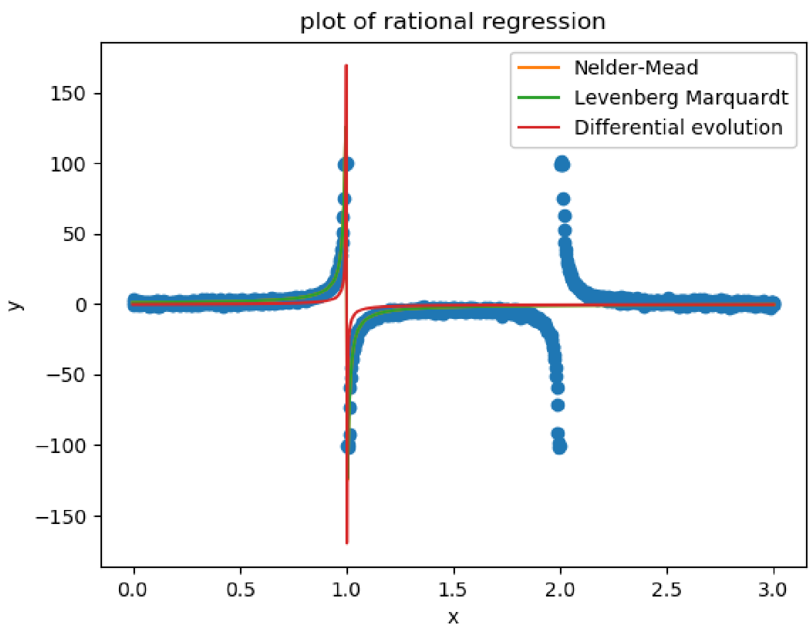


3) Случаи, когда случайная генерация в методе дифференциальной эволюции особенно удачна, этот метод по числу итераций превосходит оба других. Хотя сумма квадратов невязок в этом случае больше, чем у Нелдера-Мида за 455 итераций и у Левенберга-Марквардта за 148, по графику видно, что линия регрессии для этого метода хорошо аппроксимирует данные.

coefficients with Nelder-Mead algorithm with (455 iterations): [-1.00034984 1.00084791 -2.0009853 1.00100138]
functional value at found point (nm) 136310.48192946665

coefficients with Levenberg-Marquardt algorithm with (148 iterations): [-1.00126387 1.00173631 -2.00088464 1.00090073]
functional value at found point (lm) 136316.5445515288

coefficients with Differential evolution algorithm with (77 iterations): [-1.86219907 0.85345407 1.97222495 -2.97419988]
functional value at found point (de) 197273.7754467029



Выводы:

Алгоритм Нелдера-Мида сходится за число итераций, большее в несколько раз, чем Левенберга-Марквардта и дифференциальной эволюции, зато он довольно стабилен: сумма квадратов невязок меньше, данные аппроксимирует хорошо. Алгоритм Левенберга-Маркварда бывает нестабилен: может, как хорошо приближать данные, так и плохо, возможно, результат зависит от выбора начальной точки и ее близости к минимуму. Метод дифференциальной эволюции за счет своей стохастичности часто обходит два других метода по скорости сходимости, и в среднем хорошо аппроксимирует данные.

Вспомогательный код:

```
epsilon = 0.001
n = 1000
maxiter = 1000

function = lambda x: 1 / (x ** 2 - 3 * x + 2)

# generate dataset
x_s = np.array([3 * k / n for k in range(n + 1)])
y_s = np.array([random.normalvariate(0, 1) for _ in range(n + 1)]) + np.array(
    [-100 if function(x) < -100 else function(x) if function(x) <= 100 else 100 for x in
x_s])


def plot_dataset(regression, optimal_nm, optimal_lm, optimal_de):
    result_optimal_nm = regression(np.array(optimal_nm[0]), np.array(optimal_nm[1]),
np.array(optimal_nm[2]),
                                   np.array(optimal_nm[3]))
    result_optimal_lm = regression(np.array(optimal_lm[0]), np.array(optimal_lm[1]),
np.array(optimal_lm[2]),
                                   np.array(optimal_lm[3]))
    result_optimal_sa = regression(np.array(optimal_de[0]), np.array(optimal_de[1]),
np.array(optimal_de[2]),
```

```
                               np.array(optimal_de[3]))
plt.plot(x_s, y_s, 'o')
plt.xlabel('x')
plt.ylabel('y')
plt.plot(x_s, result_optimal_nm, label='Nelder-Mead')
plt.plot(x_s, result_optimal_lm, label='Levenberg Marquardt')
plt.plot(x_s, result_optimal_sa, label='Differential evolution')
plt.legend(framealpha=1, frameon=True)
plt.title('plot of rational regression')
plt.show()
```