Пояснительная записка к зачётному заданию по курсу «Статистические методы обработки информации»

Работу выполни:
Студент группы Б14-501
Попцов П. А.

НИЯУ МИФИ 2017

1. Задание

Реализовать алгоритм МНК для нелинейных объектов.

2. Теоретическое описание алгоритма

Рассмотрим нелинейный статический объект (1.1.3). Соответствующая ему модель представлена в виде

$$\tilde{y}(i) = \Psi(\overline{U}(i), \tilde{\tilde{c}}), \quad i = \overline{1, N}. \qquad (1.3.52)$$

Оптимизируемый критерий соответствует методу наименьших квадратов:

$$J(\tilde{\tilde{c}}) = \sum_{i=1}^{N} [y(i) - \Psi(\overline{U}(i), \tilde{\tilde{c}})]^2 r(i):$$

$$\hat{\tilde{c}}_{LS} = \arg\min_{\tilde{\tilde{c}}} J(\tilde{\tilde{c}}). \qquad (1.3.53)$$

Задача минимизации критерия (1.3.53) при нелинейной функции (1.3.52) может быть решена каким-либо методом нелинейного программирования. Эти методы достаточно хорошо разработаны и описаны в соответствующей литературе [1], [4].

В настоящем пособии приведен метод оптимизации, предназначенный для нахождения экстремума функции вида (1.3.53). Получаемая в результате итерационная последовательность соответствует методу Ньютона — Гаусса [1].

Пусть имеется некоторая оценка параметра $\hat{\tilde{c}}(j)$, достаточно близкая к истинному значению параметра. Разложим функцию $\Psi(U(i), \hat{\tilde{c}}(j))$ для каждого $i \in [i, N]$, в ряд Тейлора относительно оценки $\hat{\tilde{c}}(j)$ до второго члена малости:

$$\Psi(\overline{U}(i),\widetilde{\widetilde{c}}) \cong \Psi(\overline{U}(i),\hat{\widetilde{c}}(j)) + \left.\frac{\partial\Psi(\overline{U}(i),\widetilde{\widetilde{c}})}{\partial\widetilde{\widetilde{c}}^{\,\mathrm{T}}}\right|_{\widetilde{\widetilde{c}}=\hat{\widetilde{c}}(j)} (\widetilde{\widetilde{c}} - \hat{\widetilde{c}}(j)) . \quad (1.3.54)$$

Подставим последнее выражение в формулу критерия (1.3.53):

$$J(\widetilde{\widetilde{c}}) =$$

$$= \sum_{i=1}^{N} \left[y(i) - \Psi(\overline{U}(i),\hat{\widetilde{c}}(j)) - \left(\frac{\partial\Psi(\overline{U}(i),\hat{\widetilde{c}})}{\partial\widetilde{\widetilde{c}}} \right)^{\mathrm{T}} \Bigg|_{\widetilde{\widetilde{c}}=\hat{\widetilde{c}}(j)} (\widetilde{\widetilde{c}} - \hat{\widetilde{c}}(j)) \right]^2 r(i) . \quad (1.3.55)$$

Введем обозначения:

$$\widetilde{\widetilde{\beta}} = \widetilde{\widetilde{c}} - \hat{\widetilde{c}}(j), \quad \overline{e}(j) = \begin{bmatrix} y(1) - \Psi(\overline{U}(1),\hat{\widetilde{c}}(j)) \\ \cdots \\ y(N) - \Psi(\overline{U}(N),\hat{\widetilde{c}}(j)) \end{bmatrix},$$

$$\Phi(j) = \left[\begin{array}{ccc} \dfrac{\partial\Psi(U(1),\widetilde{\widetilde{c}})}{\partial\widetilde{c}_1} & \cdots & \dfrac{\partial\Psi(\overline{U}(1),\widetilde{\widetilde{c}})}{\partial\widetilde{c}_m} \\ \cdots & \cdots & \cdots \\ \dfrac{\partial\Psi(\overline{U}(N),\widetilde{\widetilde{c}})}{\partial\widetilde{c}_1} & \cdots & \dfrac{\partial\Psi(\overline{U}(N),\widetilde{\widetilde{c}})}{\partial\widetilde{c}_m} \end{array} \right]_{\widetilde{\widetilde{c}}=\widetilde{\widetilde{c}}(j)} . \quad (1.3.56)$$

С учетом введенных обозначений, функция (1.3.55) принимает вид:

$$J(\widetilde{\widetilde{c}}) = (\overline{e}(j) - \Phi(j)\widetilde{\widetilde{\beta}})^{\mathrm{T}} R(\overline{e}(j) - \Phi(j)\widetilde{\widetilde{\beta}}) . \quad (1.3.57)$$

Последнее выражение имеет тот же вид, что и критерий (1.3.10). Минимизация критерия (1.3.57) по $\widetilde{\widetilde{\beta}}(j)$ приводит к итерационной последовательности:

$$\hat{\widetilde{e}}(j+1) = \hat{\widetilde{c}}(j) + (\Phi^{\mathrm{T}}(j)R\Phi(j))^{-1}\Phi^{\mathrm{T}}(j)R(\overline{y} - \Psi(\hat{\widetilde{c}}(j))) . \quad (1.3.58)$$

Необходимо отметить, что данный метод очень чувствителен к выбору начального приближения $\hat{\widetilde{c}}(0)$.

Это объясняется тем, что в основе метода лежит линеаризация функции $\Psi(\overline{U}(i),\tilde{\tilde{c}})$ относительно оценки $\hat{\tilde{c}}(j)$ на j-м шаге итерационного процесса.

Признаком окончания итерационного процесса (1.3.58) может служить выполнение условия:

$$\frac{\left|J(\hat{\tilde{c}}(j+1)) - J(\hat{\tilde{c}}(j))\right|}{J(\hat{\tilde{c}}(j+1))} \le \delta, \qquad (1.3.59)$$

где δ — заданное малое число.

В заключение данного раздела сравним рекуррентную формулу (1.3.47) и итерационную (1.3.58). Несмотря на то, что эти формулы имеют схожий вид, они отражают два совершенно различных принципа идентификации.

Рекуррентная формула (1.3.47) построена по принципу коррекции оценки на основе новых наблюдений.

Итерационная формула (1.3.58) обеспечивает движение к минимуму функции (1.3.53) по направлению антиградиента в точке $\hat{\tilde{c}}(j)$, при этом никаких новых наблюдений не поступает.

3. Листинг программы

```
from math import *
import numpy as np


#import matplotlib.pyplot as plt



#w_true = [ 1.0, 2.0, -1.0, pi, 5.0 ]
w_true = [ 0.01, 1.0, -pi, pi, 0.001 ]
input_params = [ [3.0, 0.1], [pi, 0.1], [1.0, 0.1], [e, 0.1] ]
input_dist = [ "normal", "normal", "normal", "normal" ]
#N_samples = 1000
#R_mat = []
#R_mat = np.identity(N_samples)
N_inputs = 4 # N_inputs = len(w_true)
N_samples = 30
```

```
R_mat = np.identity(N_samples)
N_iter = 100

global Debug_tmp
Debug_tmp = None
Doc = lambda obj: print(obj.__doc__)
IsList = lambda obj: "list" in str(type(obj))

def f(w, u):
          y = w[0]*exp(-u[0]**w[1])*sin(w[2]*u[1]) + cos(w[3]/u[2])\
                    + w[4]*log(u[3])
          return y

f_t = lambda u: f(w_true, u)

test_dfw_0 = lambda w, u: exp(-u[0]**w[1])*sin(w[2]*u[1])
test_dfw_1 = lambda w, u: -w[0]*exp(-u[0]**w[1])*sin(w[2]*u[1])\
                               *(u[0]**w[1])*log(u[0])
test_dfw_2 = lambda w, u: w[0]*exp(-u[0]**w[1])*cos(w[2]*u[1])*u[1]
test_dfw_3 = lambda w, u: -sin(w[3]/u[2])*(1.0/u[2])
test_dfw_4 = lambda w, u: log(u[3])

Test_dfw_a = [ test_dfw_0, test_dfw_1, test_dfw_2, test_dfw_3, test_dfw_4]

def dfw(w, u, n, func = f):
          h = 1e-06
          w1 = np.copy(w)
          w2 = np.copy(w)
#       w1[n] = w[n] + h
#       w2[n] = w[n] - h
#       df = (func(w1, u) - func(w2, u))/(2.0*h)
          w1[n] = w[n] - h
          w2[n] = w[n] - 2*h
          df = (3.0*func(w, u) - 4.0*func(w1, u) + func(w2, u))/(2.0*h)
          return df

def dfw2(w, u, n, dwf_a = Test_dfw_a):
          df = dwf_a[n](w, u)
          return df
```

```
def gen_u_(sz, params = [ 0.0, 1.0 ], dist = "normal"):
        if params == None:
                params = [ 0.0, 1.0 ]
        if dist == None:
                dist = "normal"
        cmd = "np.random." + dist + "(" + str(params[0]) + ", " +\
                str(params[1]) + ", " + str(sz) + ")"
        u_ = eval(cmd)
        return u_

def gen_u(sz, params_a = None, dist_a = None):
        N_samples = sz[0]
        N_inputs = sz[1]
        if params_a == None:
                params_a = N_inputs*[None]
        if dist_a == None:
                dist_a = N_inputs*[None]
        U = []
        tmp = []
        for i in range(0, N_inputs):
                tmp = gen_u_(N_samples, params_a[i], dist_a[i])
                U.append(tmp)
        U = np.array(U)
        return U.T

def calc_ft(w = w_true, U = [], func = f_t):
        if U == []:
                return []
        N_samples = len(U)
        Y = []
        for u in U:
                Y.append(func(u))
        Y = np.array(Y)
        return Y

def calc_f(w, U = [], func = f):
        if U == []:
```

```
                return []
        N_samples = len(U)
        Y = []
        for u in U:
                Y.append(func(w, u))
        Y = np.array(Y)
        return Y

def calc_df(w = w_true, U = [], n = -1, func = f):
        if U == [] or n == -1:
                return []
        N_samples = len(U)
        DY = []
        for u in U:
                DY.append(dfw(w, u, n, func))
        DY = np.array(DY)
        return DY

def calc_Phi_t_(w = w_true, u = [], func = f):
        if u == []:
                return u
        N_params = len(w)
        df = []
        for i in range(0, N_params):
                df.append(dfw(w, u, i, f))
        df = np.array(df)
        return df

def calc_Phi_t(w = w_true, U = [], func = f):
        if U == []:
                return U
        N_params = len(w)
        N_samples = len(U)
        DF = []
        tmp = []
        for u in U:
                tmp = calc_Phi_t_(w, u, f)
                DF.append(tmp)
```

```
        DF = np.array(DF)
        return DF

def calc_Phi_(w, u = [], func = f):
        if u == [] or w == []:
                return u
        N_params = len(w)
        df = []
        for i in range(0, N_params):
                df.append(dfw(w, u, i, f))
        df = np.array(df)
        return df

def calc_Phi(w, U = [], func = f):
        if U == [] or w == []:
                return U
        N_params = len(w)
        N_samples = len(U)
        DF = []
        tmp = []
        for u in U:
                tmp = calc_Phi_(w, u, f)
                DF.append(tmp)
        DF = np.array(DF)
        return DF

def calc_Phi2_(w, u = [], func = f):
        if u == [] or w == []:
                return u
        N_params = len(w)
        df = []
        for i in range(0, N_params):
                df.append(dfw2(w, u, i))
        df = np.array(df)
        return df

def calc_Phi2(w, U = [], func = f):
        if U == [] or w == []:
```

```
                return U
        N_params = len(w)
        N_samples = len(U)
        DF = []
        tmp = []
        for u in U:
                tmp = calc_Phi2_(w, u, f)
                DF.append(tmp)
        DF = np.array(DF)
        return DF

def calc_e_vec(w_params, Y_t = [], U_values = [], func = f):
        if U == []:
                return []
        if Y_t == []:
                Y_t = calc_ft(w = w_true, U = U_values)
        N_samples = len(U_values)
        Y_ = calc_f(w = w_params, U = U_values)
        E = Y_t - Y_
        return E

def calc_w_next(w_prev, Func, U_values, Y_exp):
        Epsilon = 1.0/10.0**256
        Phi = calc_Phi(w = w_prev, U = U_values, func = Func)
        Y_vec = Y_exp
        Psi_vec = calc_f(w = w_prev, U = U_values, func = Func)
        w_new = np.array(w_prev)
        Res = np.dot(np.dot(Phi.T, R_mat), Phi)
        if abs(np.linalg.det(Res)) == 0.0:
                return (Phi, w_new, Psi_vec, "stop")
        Res = np.dot(np.linalg.inv(Res), Phi.T)
        Res = np.dot(np.dot(Res, R_mat), Y_vec - Psi_vec)
        w_new = w_new + Res
        return w_new

def calc_w_next2(w_prev, Func, U_values, Y_exp):
        Epsilon = 1.0/10.0**256
        Phi = calc_Phi2(w = w_prev, U = U_values, func = Func)
```

```
        Y_vec = Y_exp
        Psi_vec = calc_f(w = w_prev, U = U_values, func = Func)
        w_new = np.array(w_prev)
        Res = np.dot(np.dot(Phi.T, R_mat), Phi)
        if abs(np.linalg.det(Res)) == 0.0:
                return (Phi, w_new, Psi_vec, "stop")
        Res = np.dot(np.linalg.inv(Res), Phi.T)
        Res = np.dot(np.dot(Res, R_mat), Y_vec - Psi_vec)
        w_new = w_new + Res
        return w_new

def show_inputs_info():
        print("_"*80)
        print("INPUTS INFO:")
        print("Num of inputs: ", N_inputs)
        print("Num of samples: ", N_samples)
        print("Max iterations: ", N_iter)
        for i in range(0, N_inputs):
                Str_f = ""
                Str_f += input_dist[i][0].upper()
                Str_f += str(tuple(input_params[i]))
                print("%d: U%d = %s" % (i, i, Str_f))
        print("_"*80)

'''
def nonlin_mls(w_start):
        w_begin = w_start
        global Debug_tmp
        U_values = gen_u((N_samples, N_inputs), input_params, input_dist)
        Y_true = calc_ft(U = U_values)
        w_current = w_begin
        Str_f = (len(w_begin)*("%.4f "))[:-1]
        print("True params:\t%s" % Str_f % tuple(w_true))
        print("Init params:\t%s" % Str_f % tuple(w_begin))
        for i in range(0, N_iter):
                print("Iteration:[%d]\t%s" % (i, Str_f % tuple(w_current)))
                w_new = calc_w_next(w_current, f, U_values, Y_true)
                if w_new[-1] == "stop":
```

```
                                Debug_tmp = w_new
                                w_new = []
                                break
                        w_current = w_new
                if Debug_tmp != None:
                        Phi_d, w_new_d, Psi_vec_d, _ = Debug_tmp
'''

def nonlin_mls(w_start, Print = 2):
        if Print > 0:
                print("_"*80)
        w_begin = w_start
        global Debug_tmp
        U_values = gen_u((N_samples, N_inputs), input_params, input_dist)
        Y_true = calc_ft(U = U_values)
        w_current = w_begin
        Str_f = (len(w_begin)*("%.6f "))[:-1]
        if Print > 0:
                print("True params:\t%s" % Str_f % tuple(w_true))
                print("Init params:\t%s" % Str_f % tuple(w_begin))
        LastIterIndex = 0
        for i in range(0, N_iter):
                if Print == 2:
                        print("Iteration:[%d]\t%s" % (i, Str_f % tuple(w_current)))
                w_new = calc_w_next(w_current, f, U_values, Y_true)
                if w_new[-1] == "stop":
                        Debug_tmp = w_new
                        w_new = []
                        break
                LastIterIndex = i
                w_current = w_new
        AbsError = np.abs(np.array(w_true) - np.array(w_current)).tolist()
        if Print > 0:
                print("Last iter:\t[%d]\nCurrent params:\t%s" % (LastIterIndex, Str_f %
tuple(w_current)))
                print("Abs errors:\t%s" % (Str_f % tuple(AbsError)))
                print("Squared sum error:%.6f" % (sum([ AbsErr**2 for AbsErr in AbsError
])/len(w_true)))
        if Debug_tmp != None:
```

```
                Phi_d, w_new_d, Psi_vec_d, _ = Debug_tmp
        if Print > 0:
                print("_"*80)

def nonlin_mls2(w_start, Print = 2):
        if Print > 0:
                print("_"*80)
        w_begin = w_start
        global Debug_tmp
        U_values = gen_u((N_samples, N_inputs), input_params, input_dist)
        Y_true = calc_ft(U = U_values)
        w_current = w_begin
        Str_f = (len(w_begin)*("%.6f "))[:-1]
        if Print > 0:
                print("True params:\t%s" % Str_f % tuple(w_true))
                print("Init params:\t%s" % Str_f % tuple(w_begin))
        LastIterIndex = 0
        for i in range(0, N_iter):
                if Print == 2:
                        print("Iteration:[%d]\t%s" % (i, Str_f % tuple(w_current)))
                w_new = calc_w_next2(w_current, f, U_values, Y_true)
                if w_new[-1] == "stop":
                        Debug_tmp = w_new
                        w_new = []
                        break
                LastIterIndex = i
                w_current = w_new
        AbsError = np.abs(np.array(w_true) - np.array(w_current)).tolist()
        if Print > 0:
                print("Last iter:\t[%d]\nCurrent params:\t%s" % (LastIterIndex, Str_f %
tuple(w_current)))
                print("Abs errors:\t%s" % (Str_f % tuple(AbsError)))
                print("Squared sum error:%.6f" % (sum([ AbsErr**2 for AbsErr in AbsError
])/len(w_true)))
        if Debug_tmp != None:
                Phi_d, w_new_d, Psi_vec_d, _ = Debug_tmp
        if Print > 0:
                print("_"*80)
```

```
def corr_w_b(w_start, delta_w):
          if IsList(delta_w):
                    return [ w_start[i] + delta_w[i] for i in range(0, len(w_start)) ]
          else:
                    return [ w_start[i] + delta_w for i in range(0, len(w_start)) ]



#w_true = [ 1.0, 2.0, -1.0, pi, 5.0 ]
#w_begin1 = [ 0.8, 1.7, -1.2, 3.0, 4.5 ]
#w_begin2 = [ 1.2, 2.3, -0.8, 3.5, 5.5 ]
#w_begin3 = [ 1.0, 1.0, 1.0, 1.0, 1.0 ]

show_inputs_info()

#w_true = [ 0.01, 1.0, -pi, pi, 0.001 ]
w_begin4 = [ 0.1, 0.5, -2.5, 3.0, 0.01 ]
w_begin5 = [ 0.0, 1.5, -3.8, 3.5, 0.09 ]
w_begin6 = [ 0.03, 0.8, -pi + pi/12.0, pi - pi/12.0, 0.05 ]
w_begin7 = [ 0.5, 0.5, -4.0, 3.2, 0.03 ]
dw = 0.01
delta_w =  len(w_true)*([ dw ])

nonlin_mls(w_begin4, Print = 1)
nonlin_mls(w_begin5, Print = 1)
nonlin_mls(w_begin6, Print = 1)
nonlin_mls(w_begin7, Print = 1)

if Debug_tmp != None:
                    Phi_d, w_new_d, Psi_vec_d, _ = Debug_tmp
```

4. Скриншоты работы программы

```
INPUTS INFO:
Num of inputs:  4
Num of samples:  30
Max iterations:  100
0: U0 = N(3.0, 0.1)
1: U1 = N(3.141592653589793, 0.1)
2: U2 = N(1.0, 0.1)
3: U3 = N(2.718281828459045, 0.1)
```

```
True params:    0.010000 1.000000 -3.141593 3.141593 0.001000
Init params:    0.100000 0.500000 -2.500000 3.000000 0.010000
Last iter:      [99]
Current params: -0.008971 0.847958 -2.147352 3.141588 0.000894
Abs errors:     0.018971 0.152042 0.994240 0.000004 0.000106
Squared sum error:0.202398
```

```
True params:    0.010000 1.000000 -3.141593 3.141593 0.001000
Init params:    0.000000 1.500000 -3.800000 3.500000 0.090000
Last iter:      [0]
Current params: 0.000000 1.500000 -3.800000 3.500000 0.090000
Abs errors:     0.010000 0.500000 0.658407 0.358407 0.089000
Squared sum error:0.163995
```

```
True params:    0.010000 1.000000 -3.141593 3.141593 0.001000
Init params:    0.030000 0.800000 -2.879793 2.879793 0.050000
Last iter:      [99]
Current params: -0.012677 0.962456 -2.115578 3.141595 0.000956
Abs errors:     0.022677 0.037544 1.026014 0.000002 0.000044
Squared sum error:0.210926
```

```
True params:    0.010000 1.000000 -3.141593 3.141593 0.001000
Init params:    0.500000 0.500000 -4.000000 3.200000 0.030000
Last iter:      [99]
Current params: -0.011156 1.108204 -4.132591 3.141593 0.001064
Abs errors:     0.021156 0.108204 0.990999 0.000001 0.000064
```

```
Max iterations:  100
0: U0 = N(3.0, 0.1)
1: U1 = N(3.141592653589793, 0.1)
2: U2 = N(1.0, 0.1)
3: U3 = N(2.718281828459045, 0.1)
```

```
True params:    0.010000 1.000000 -3.141593 3.141593 0.001000
Init params:    0.100000 0.500000 -2.500000 3.000000 0.010000
Last iter:      [1]
Current params: -0.004299 8.579206 62.800385 3.141604 0.000878
Abs errors:     0.014299 7.579206 65.941977 0.000011 0.000122
Squared sum error:881.157789
```

```
True params:    0.010000 1.000000 -3.141593 3.141593 0.001000
Init params:    0.000000 1.500000 -3.800000 3.500000 0.090000
Last iter:      [0]
Current params: 0.000000 1.500000 -3.800000 3.500000 0.090000
Abs errors:     0.010000 0.500000 0.658407 0.358407 0.089000
Squared sum error:0.163995
```

```
True params:    0.010000 1.000000 -3.141593 3.141593 0.001000
Init params:    0.030000 0.800000 -2.879793 2.879793 0.050000
Last iter:      [0]
Current params: 0.243218 5.264447 -4.836050 3.172965 0.023212
Abs errors:     0.233218 4.264447 1.694457 0.031373 0.022212
Squared sum error:4.222513
```

```
True params:    0.010000 1.000000 -3.141593 3.141593 0.001000
Init params:    0.500000 0.500000 -4.000000 3.200000 0.030000
Last iter:      [99]
Current params: -0.015131 1.192890 -4.103759 3.141598 0.001094
Abs errors:     0.025131 0.192890 0.962166 0.000005 0.000094
Squared sum error:0.192720
```

5. Заключение

В результате выполнения работы был реализован алгоритм метода МНК для нелинейных объектов средствами языка программирования Python. В результате проведенных численных экспериментов по работе данного алгоритма была выявлена плохая сходимость начальных значений итерационной формулы к истинным значениями параметров. Это объясняется особенностями численного метода вычисления частных производных (большой погрешностью, которая быстро возрастает) и также слабой обусловленностью получившейся матрицы Phi*Phi^T (лучше использовать алгоритм Алгоритм Левенберга — Марквардта).