1. **Задание**
2. Найти оценки параметров модели по данным, смоделированным в л/р3 (алгорим Баум-Велша, [4] стр 145-156)
3. Отобразить результаты оценивания в таблице:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Начальное приближение параметров модели | Оценки параметров  модели | Достигнутая точность по параметрам  ( , ) | Кол-во итераций  () | Достигнутая точность по значению невязки функции правдоподобия: |
|  |  |  |  |  |

Начальное приближение параметров модели выбирать как минимум три раза:

1. Близкими к истинным параметрам
2. Равными истинным параметрам
3. Далекими от истинных параметров
4. **Вариант задания**

|  |  |  |  |
| --- | --- | --- | --- |
| *Вариант* | *Алфавит: V* | *Матрица переходных вероятностей А* | *Матрица эмиссей*  *В* |
| 2 |  |  |  |

1. **Текст программы**

import numpy as np

import scipy as sp

from functools import reduce

import matplotlib.pyplot as plt

def get\_data(fname, type):

O = np.array([[i for i in line.split()] for line in open(fname, encoding="utf-8")], dtype=type)

return O

def get\_data1(fname, type):

O = np.array([i for i in open(fname, encoding="utf-8").readline().split()], dtype=type)

return O

def WritingInFile(names, sequences, fileName):

with open(fileName, "w") as file:

for line in sequences:

print(line, file=file)

#прямой ход

def forward\_path(O, pi, A, B, T, N, K):

alpha\_k = []

for k in range(K):

alpha = np.zeros((T, N))

alpha[0, :] = pi \* B[:, O[k, 0]]

for t in range(1, T):

for j in range(N):

tmp = np.zeros(N)

for i in range(N):

tmp[i] = alpha[t - 1, i] \* A[i, j]

alpha[t, j] = tmp.sum() \* B[j, O[k, t]]

alpha\_k.append(alpha)

return np.array(alpha\_k)

#обратный ход

def backward\_path(O, pi, A, B, T, N, K):

beta\_k = []

for k in range(K):

beta = np.zeros((T, N))

beta[T - 1, :] = 1

for t in range(T - 2, -1, -1):

for i in range(N):

tmp = np.zeros(N)

for j in range(N):

tmp[j] = beta[t + 1, j] \* A[i, j] \* B[j, O[k, t + 1]]

beta[t, i] = tmp.sum()

beta\_k.append(beta)

return np.array(beta\_k)

#вычисление гамма

def calculate\_gamma(alpha, beta, T, N, K):

gamma\_k = []

for k in range(K):

gamma = np.zeros((T, N))

for t in range(T):

for i in range(N):

gamma[t, i] = alpha[k, t, i] \* beta[k, t, i]

sum\_all = gamma[t, :].sum()

gamma[t, :] = gamma[t, :] / sum\_all

gamma\_k.append(gamma)

return np.array(gamma\_k)

#вычисление кси

def calculate\_ksi(O, alpha, beta, A, B, T, N, K):

ksi\_k = []

for k in range(K):

ksi = np.zeros((T, N, N))

for t in range(T - 1):

for i in range(N):

for j in range(N):

ksi[t, i, j] = alpha[k, t, i] \* A[i, j] \* beta[k, t + 1, j] \* B[j, O[k, t + 1]]

sum\_all = ksi[t, :, :].sum()

ksi[t, :, :] = ksi[t, :, :] / sum\_all

ksi\_k.append(ksi)

return np.array(ksi\_k)

#EM-алгоритм (вычисление оценок параметров модели)

def estimate\_parameter(O, pi\_0, A\_0, B\_0, T, N, M, K):

alp = forward\_path(O, pi\_0, A\_0, B\_0, T, N, K)

bet = backward\_path(O, pi\_0, A\_0, B\_0, T, N, K)

gam = calculate\_gamma(alp, bet, T, N, K)

ksi = calculate\_ksi(O, alp, bet, A\_0, B\_0, T, N, K)

#оценка начальных состояний

est\_pi = np.sum(gam[:, 0, :], axis=0) / K

#оценка переходной матрицы

est\_A\_k = np.zeros((K, N, N))

for k in range(K):

for i in range(N):

denom = gam[k, :-1, i].sum()

for j in range(N):

est\_A\_k[k, i, j] = ksi[k, :-1, i, j].sum() / denom

est\_A = np.sum(est\_A\_k, axis=0) / K

#оценка матрицы эмиссей

est\_B\_k = np.zeros((K, N, M))

for k in range(K):

for i in range(N):

denom = gam[k, :, i].sum()

for j in range(M):

numer = gam[k, :, i][O[k] == j].sum()

est\_B\_k[k, i, j] = numer / denom

est\_B = np.sum(est\_B\_k, axis=0) / K

return est\_pi, est\_A, est\_B

#вычисление функции правдоподобия

def log\_likelihood(O, pi, A, B, T, N, K):

alp = forward\_path(O, pi, A, B, T, N, K)

L = []

for k in range(K):

l = np.zeros((T, N))

for t in range(T):

for i in range(N):

l[t, i] = alp[k, t, i]

sum\_all = l[t, :].sum()

L.append(sum\_all)

lnL = np.sum(np.log(L))

return lnL

#условие выхода

def iter\_exit(O, pi\_old, A\_old, B\_old, pi\_new, A\_new, B\_new, T, N, K):

old = log\_likelihood(O, pi\_old, A\_old, B\_old, T, N, K)

new = log\_likelihood(O, pi\_new, A\_new, B\_new, T, N, K)

exit = abs(old - new)

if exit > 1e-3:

return False, exit

else:

return True, exit

#итерационный алгоритм Баума-Уэлша

def baum\_welch(O, pi, A, B, T, N, M, K):

iter = 0

exit = False

max\_iter = 100

ex = []

temp = []

temp.append(log\_likelihood(O, pi, A, B, T, N, K))

while exit == False:

iter += 1

new\_pi, new\_A, new\_B = estimate\_parameter(O, pi, A, B, T, N, M, K)

exit, tmp = iter\_exit(O, pi, A, B, new\_pi, new\_A, new\_B, T, N, K)

temp.append(log\_likelihood(O, new\_pi, new\_A, new\_B, T, N, K))

if iter > max\_iter:

exit = True

ex.append(tmp)

pi, A, B = new\_pi, new\_A, new\_B

plt.xlabel('iter')

plt.ylabel('lnl')

it = np.linspace(0, iter, iter)

f = plt.plot(it, np.array(ex1))

plt.show()

WritingInFile(['iter'], np.array([iter]), 'iter.txt')

WritingInFile(['ex'], ex, 'ex.txt')

return pi, A, B, ex

def ro\_lambd():

Q = get\_data('Q.txt', np.int)

O = get\_data('O.txt', np.int)

A\_0 = []

B\_0 = []

pi\_0 = []

A\_0.append(get\_data('A1.txt', np.double))

B\_0.append(get\_data('B1.txt', np.double))

pi\_0.append(get\_data1('pi1.txt', np.int))

A\_0.append(get\_data('A2.txt', np.double))

B\_0.append(get\_data('B2.txt', np.double))

pi\_0.append(get\_data1('pi2.txt', np.double))

A\_0.append(get\_data('A3.txt', np.double))

B\_0.append(get\_data('B3.txt', np.double))

pi\_0.append(get\_data1('pi3.txt', np.double))

A\_0.append(get\_data('A4.txt', np.double))

B\_0.append(get\_data('B4.txt', np.double))

pi\_0.append(get\_data1('pi4.txt', np.double))

A\_0.append(get\_data('A5.txt', np.double))

B\_0.append(get\_data('B5.txt', np.double))

pi\_0.append(get\_data1('pi5.txt', np.double))

est = []

lnL = []

#прогон алгоритма из разных приближений и выбор оценок параметров по максимальному #логарифму функции правдоподобия

for i in range(5):

est\_pi, est\_A, est\_B, ex = baum\_welch(O, np.array(pi\_0[i]), np.array(A\_0[i]), np.array(B\_0[i]), 100, 3, 2, 100)

est.append([est\_pi, est\_A, est\_B])

lnL.append(log\_likelihood(O, est\_pi, est\_A, est\_B, 100, 3, 100))

maxlnL = lnL.index(np.max(np.array(lnL)))

est\_param = est[maxlnL]

WritingInFile(['maxlnL'], np.array([np.max(np.array(lnL))]), 'maxlnL.txt')

WritingInFile(['est\_pi'], est\_param[0], 'est\_pi.txt')

WritingInFile(['est\_A'], est\_param[1], 'est\_A.txt')

WritingInFile(['est\_B'], est\_param[2], 'est\_B.txt')

ro\_A = np.linalg.norm(A\_0[maxlnL] - est\_param[1])

ro\_B = np.linalg.norm(B\_0 [maxlnL] - est\_param[2])

WritingInFile(['roA'], np.array([ro\_A]), 'roA.txt')

WritingInFile(['roB'], np.array([ro\_B]), 'roB.txt')

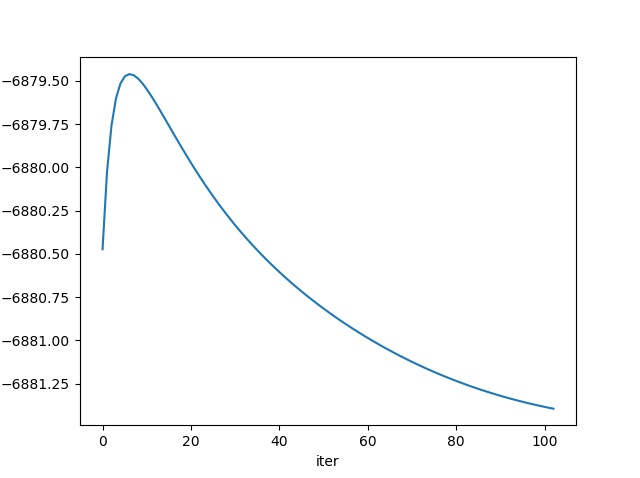
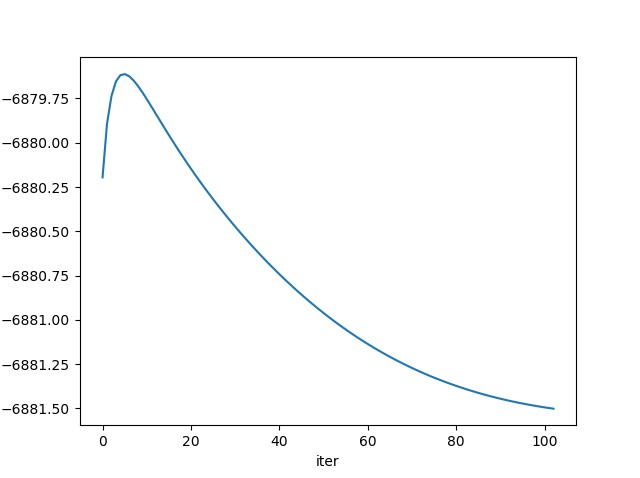
return est\_param

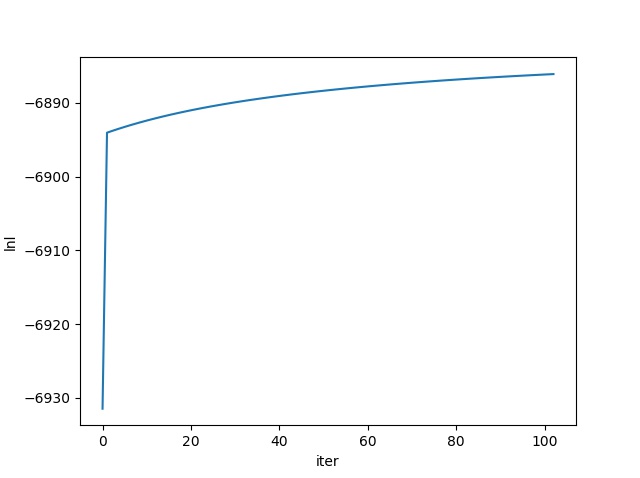
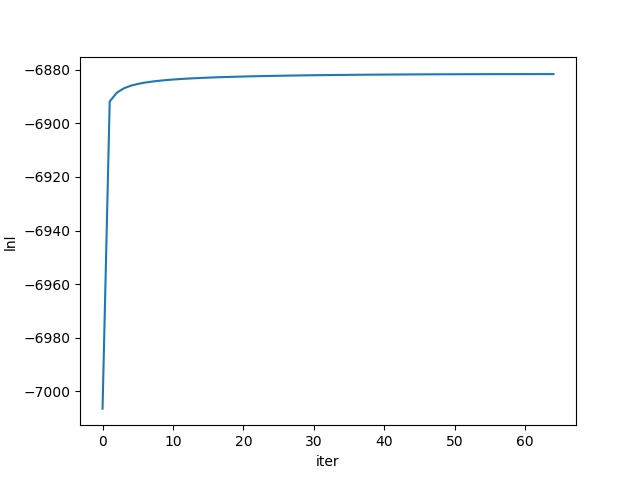
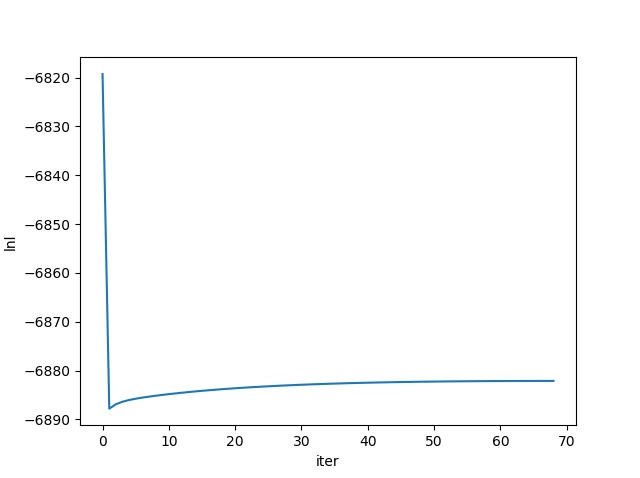
ro\_lambd()

1. **Исследования**

**Для различных начальных приближений**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Начальное приближение параметров модели | Оценки параметров  модели | Достигнутая точность по параметрам  ( , ) | Кол-во итераций  () | Достигнутая точность по значению невязки функции правдоподобия: |
| 1 |  |  | 0.123271  0.252465 | 101 | 0.0036643 |
| 2 |  |  | 0.125817  0.261109 | 101 | 0.005179 |
| 3 |  |  | 0.194036283978  0.3331186628713 | 67 | 0.0009105 |





Для различных начальных приближений (в качестве оценки параметров модели выбираются такие оценки, при которых логарифмическая функция правдоподобия максимальна)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Начальное приближение параметров модели | Оценки параметров  модели | Достигнутая точность по параметрам  ( , ) | Кол-во итераций  () | Достигнутая точность по значению невязки функции правдоподобия: |
| 1 |  |  | 0.125817087958  0.261109540218 | 101 | 0.0293165844805 |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |