МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«Дальневосточный федеральный университет»
(ДВФУ)

ШКОЛА ЕСТЕСТВЕННЫХ НАУК

Кафедра информатики, математического и компьютерного моделирования

ОТЧЕТ

по лабораторной работе
по дисциплине «Методы оптимизации»

Выполнил студент
гр. Б9119-02.03.01сцт
Панченко Н.К. ____________
(ФИО) *(подпись)*

«17» мая 2022 г.

г. Владивосток
2022

Постановка задачи

Требуется найти минимум функции $f(x) = \dfrac{1}{2}x^T Ax + bx$ нескольких переменных при помощи методов Ньютона и градиентного спуска и сравнить результаты и время работы.
Для начала нужно сгенерировать симметричную и положительно определенную матрицу A. Сгенерировав нижнюю треугольную матрицу L, где все элементы на главной диагонали строго больше нуля, и умножив ее на транспонированную матрицу L мы получим необходимую симметрическую и положительно определенную матрицу. В правую часть можно записать любые числа.
В обоих методах необходима производная функции, так же в методе Ньютона используется вторая производная.

$$f(x) = \frac{1}{2}x^T Ax + bx$$

$$f'(x) = \left(\frac{1}{2}x^T Ax\right)' + (bx)' = \frac{1}{2}\left(A + A^T\right)x + b = Ax + b$$
$$f''(x) = (Ax)' + b' = A$$

Для метода Ньютона необходимо нахождение обратной матрицы. Будем искать обратную матрицу с помощью матрицы из алгебраических дополнений.

```
    def det(a):
  if(type(a[0]) != list or len(a) != len(a[0])):
    return None
  if(len(a) == 2):
    return a[0][0] * a[1][1] - a[0][1]*a[1][0]
  elif(len(a) == 3):
    sum = 0
    for i in range(3):
      p = 1
```

```
        m = 1
        for j in range(3):
            p *= a[(i + j)%3][j]
            m *= a[(i + j)%3][2-j]
        sum += p
        sum -= m
    return sum
  else:
    sum = 0
    for i in range(len(a)):
        sum += (-1)**i * a[0][i]*det(minor(a,i,0))
    return sum

def minor(a,x,y):
  to_ret = []
  for r in a:
    temp = r[:]
    temp.pop(x)
    to_ret += [temp]
  to_ret.pop(y)
  return to_ret

def inv(a):
  c = [r[:] for r in a]
  for i in range(len(a)):
    for j in range(len(a[0])):
        c[i][j] = (-1)**(i + j)*det(minor(a,j,i))
  return mul(transpose(c),1/det(a))
```

Метод Ньютона

Поиск значений x для нахождения минимума функции осуществляется по следующей формуле:

$$x^{k+1} = x^k - \frac{f'(x^k)}{f''(x^k)}$$

В нашем случае начальные значения x не влияют на ход работы и для вычисления значений x потребуется лишь одна итерация, потому как:

$$x^1 = x^0 - \frac{f'(x^0)}{f''(x^0)} = x^0 - A^{-1}(Ax^0 + b) = x^0 - A^{-1}Ax^0 - A^{-1}b = -A^{-1}b$$

Тогда подставив в производную найденные значения получим, что мы нашли стационарную точку, являющуюся минимумом функции:

$$f'(x^1) = Ax^1 + b = A(-A^{-1}b) + b = -b + b = 0$$

Вычисления происходили бы не за один шаг, если бы у нас была другая функция, соответственно другие производные.

Реализация алгоритма

Для реализации воспользуемся языком программирования Python и библиотекой Numpy для удобной работы с матрицами.

```
    def newtons_method(x,a,b):
  return x - dot(inv(a), dfunc(x,a,b))
```

dwdw

Метод градиентного спуска

Поиск значений x для нахождения минимума функции осуществляется по следующей формуле:

$$x^{k+1} = x^k - \lambda f'(x^k)$$

Будем брать $\lambda = 10^{-6}$, а вычисления проводить до тех пор, пока разница между значениями функций $f(x^k)$ и $f(x^{k+1})$ будет больше, чем 10^{-10}.

Реализация алгоритма

Для реализации воспользуемся языком программирования Python и библиотекой Numpy для удобной работы с матрицами.

```
    def grad_method(x,a,b,step = 0.0001):
  xn = [r[:] for r in x]
  while(step > 0.000000001):
    start = func(xn,a,b)
    d = dfunc(xn,a,b)
    right = mul(d ,-step)
    xn = plus(xn, right)
    end = func(xn,a,b)
    if(start < end):
      step /= 10
  return xn
```

dwdw

Тесты

Заключение