МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«Дальневосточный федеральный университет»

(ДВФУ)

ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ

Департамент математического и компьютерного моделирования

ЛАБОРАТОРНАЯ РАБОТА №2

По основной образовательной программе подготовки бакалавров направлению 01.03.02 Прикладная математика и информатика профиль «Системное программирование»

Студент группы Б9121-02.03.01сцт

Москера Креспо Адриан Хосуэ

«24» декабря 2023 г.

Преподаватель кандидат физико-математических наук

________________________________ (подпись)

Яковлев Анатолий Александрович

«____» ______________ 2023 г.

г. Владивосток

2023

Постановка задачи

Найти минимум функции $\mathbb{R}^n$

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

с условием $\|x - x_0\| \leq r$.

Исходные данные

A — произвольная симметрическая, невырожденная матрица, $A \in \mathbb{R}^{4\times 4}$

b — произвольный ненулевой вектор, $b \in \mathbb{R}^4$

x_0 — произвольный начальный ненулевой вектор, $x \in \mathbb{R}^4$

r — радиус сферы

$$A = \begin{pmatrix} 8.917471161017044 & -5.881235445212047 & -3.748113781121384 & 3.0726928964843987 \\ -5.881235445212047 & -9.040202112715473 & -6.772212260246581 & 2.309845893033086 \\ -3.748113781121384 & -6.772212260246581 & 7.126513709691746 & 4.664488504098735 \\ 3.0726928964843987 & 2.309845893033086 & 4.664488504098735 & 5.724519936137147 \end{pmatrix}$$

Чтобы сгенерировать симметричную матрицу A, генерируется случайная временная матрица A_{temp} так, что $A = \frac{1}{2}\left(A_{\text{temp}} + A_{\text{temp}}^T\right)$.

$$b = \begin{pmatrix} -1.0034508506027269 \\ 8.494875723462954 \\ 3.3273150949612385 \\ 1.977512900042199 \end{pmatrix} \qquad x_0 = \begin{pmatrix} 9.902751075372013 \\ -9.113925983068228 \\ -1.288098758917016 \\ -5.869546741009071 \end{pmatrix}$$

$$r = 5.0$$

Решение

Найдём функцию Лагранжа:

$$L(x, y) = \frac{1}{2}x^T Ax + bx + y(\|x - x_0\|^2 - r^2)$$

Найдём точки минимума. Для этого возьмём частную производную по и приравняем её к нулю:

$$\frac{\partial L}{\partial x} = Ax + b + 2y(x - x_0) = 0$$

Рассмотрим два случая:

1. *Пусть* $y = 0$.

$Ax + b = 0$, тогда $x_* = -A^{-1}b$, где x_* — «подозрительная» на минимум точка.

$$x_* = \begin{pmatrix} 0.6147706419264252 \\ -6.822929360179975e-2 \\ 0.4621409715178318 \\ -1.024464312324732 \end{pmatrix}$$

$$f(x_*) = -0.8423471280686472$$

Проверим, подходит ли данная точка под условие $\|x - x_0\| \leq r$.

$$\left\| \begin{pmatrix} 0.6147706419264252 \\ -6.822929360179975e-2 \\ 0.4621409715178318 \\ -1.024464312324732 \end{pmatrix} - \begin{pmatrix} 9.902751075372013 \\ -9.113925983068228 \\ -1.288098758917016 \\ -5.869546741009071 \end{pmatrix} \right\| = 13.95096312032364$$

$$\|x - x_0\| = 13.95096312032364 \leq r = 5.0$$

Условие не выполняется. Таким образом, найденная точка не подходит под ограничения и не будет рассматриваться при выборе итогового ответа.

2. *Пусть* $y > 0$

Преобразуем L'_x и получим следующую систему уравнений из пяти уравнений:

$$\begin{cases} (A + 2Iy)x + (b - 2yx_0) = 0 \\ \|x - x_0\|^2 - r^2 = 0 \end{cases}$$

Для нахождения точек, подозрительных на оптимум, воспользуемся методом Ньютона:

$$x_{k+1} = x_k - {f'}^{-1}(x_k) - f(x_k)$$

где x_k — пятимерный вектор неизвестных, составленный из элементов вектора x и y.

$f(x_k)$ — левая часть данной системы, $f'(x_k)$ — матрица Якоби данной системы уравнений.

$$f'(x) = J = \begin{pmatrix} A + 2Iy & 2(x - x_0) \\ 2(x - x_0)^T & 0 \end{pmatrix}$$

Метод Ньютона будем запускать на нескольких начальных приближениях, т.к. функция может иметь несколько оптимальных точек. За начальное приближение берётся восемь точек:

$$x_1 = \begin{pmatrix} -6.5898083421499445 \\ 0.7592215884645075 \\ -4.226682137443012 \\ -6.119390680278865 \\ 8.627489224405448 \end{pmatrix} \qquad x_5 = \begin{pmatrix} -7.598796223848999 \\ 6.843339272245674 \\ -9.182634429570456 \\ -3.1893765108104084 \\ 9.206199154867335 \end{pmatrix}$$

$$x_2 = \begin{pmatrix} -2.6100865251413534 \\ 5.99242760986122 \\ -8.74927058223387 \\ -5.95634448776172 \\ 3.0078855539858225 \end{pmatrix} \qquad x_6 = \begin{pmatrix} 6.178983466733086 \\ 2.1890780465770114 \\ 4.1797457469462955 \\ -0.2392934769547459 \\ 1.816814793169577 \end{pmatrix}$$

$$x_3 = \begin{pmatrix} 1.3960738579100074 \\ 6.462717315110776 \\ -5.392336641442533 \\ 9.741069773234269 \\ 9.188512682496391 \end{pmatrix} \qquad x_7 = \begin{pmatrix} -2.292605667486429 \\ -6.746023009212916 \\ -4.696646180855078 \\ 7.4200281278856375 \\ 9.377417885952877 \end{pmatrix}$$

$$x_4 = \begin{pmatrix} 7.2108095713692295 \\ 6.869197562738942 \\ -9.983988083646999 \\ 4.03945099928122 \\ 4.600615303013031 \end{pmatrix} \qquad x_8 = \begin{pmatrix} 4.611794059659715 \\ 6.079254454194263 \\ 7.7980462506833135 \\ 7.846995291838499 \\ 2.9922584062467177 \end{pmatrix}$$

Условие для выхода из цикла:

$$\|x_{k+1} - x_k\| \leq \varepsilon,$$

где $\varepsilon = 10^{-6}$.

В результате получаем несколько точек x_i, подозрительных на оптимум:

i	**Начальное приближение**	x_i	y_i	$f(x_i)$
1	$\begin{pmatrix} -6.5898083 \\ 0.7592215 \\ -4.2266821 \\ -6.1193906 \\ 8.6274892 \end{pmatrix}$	$\begin{pmatrix} 6.061015807963965 \\ -11.876589956638057 \\ -2.111750020911596 \\ -4.480174039784763 \end{pmatrix}$	15.2331403	−138.21816218494826
2	$\begin{pmatrix} -2.6100865 \\ 5.9924276 \\ -8.7492705 \\ -5.9563444 \\ 3.0078855 \end{pmatrix}$	$\begin{pmatrix} 14.7481665631942 \\ -8.998465898153777 \\ -2.3565551605286443 \\ -6.475367374443787 \end{pmatrix}$	−17.7870006	1311.2159918285604
3	$\begin{pmatrix} 1.3960738 \\ 6.4627173 \\ -5.3923366 \\ 9.7410697 \\ 9.1885126 \end{pmatrix}$	$\begin{pmatrix} 6.758203006065774 \\ -13.175068797262252 \\ 11.216595144120248 \\ -21.872915990532427 \end{pmatrix}$	−1.0726899	1419.4616874561702

4	$\begin{pmatrix} 7.2108095 \\ 6.8691975 \\ -9.9839880 \\ 4.0394509 \\ 4.6006153 \end{pmatrix}$	$\begin{pmatrix} -12.799489605395173 \\ 3.9115356629638285 \\ 0.763414232449291 \\ 4.53999352717738 \end{pmatrix}$	-4.2064018	969.446868112623
5	$\begin{pmatrix} -7.5987962 \\ 6.8433392 \\ -9.1826344 \\ -3.1893765 \\ 9.2061991 \end{pmatrix}$	$\begin{pmatrix} 6.791822024079396 \\ -13.204440097576258 \\ 11.234563811710583 \\ -21.91831889661696 \end{pmatrix}$	-1.0673031	1426.0002410148386
6	$\begin{pmatrix} 6.1789834 \\ 2.1890780 \\ 4.1797457 \\ -0.2392934 \\ 1.8168147 \end{pmatrix}$	$\begin{pmatrix} -3.091764331166946 \\ -4.569631582300696 \\ 5.951991972588306 \\ -8.570411385348269 \end{pmatrix}$	-2.6509415	355.862001464922
7	$\begin{pmatrix} -2.2926056 \\ -6.7460230 \\ -4.6966461 \\ 7.4200281 \\ 9.3774178 \end{pmatrix}$	$\begin{pmatrix} 6.061015807964309 \\ -11.876589956638353 \\ -2.1117500209120843 \\ -4.4801740397843695 \end{pmatrix}$	15.2331403	-138.21816218496198
8	$\begin{pmatrix} 4.6117940 \\ 6.0792544 \\ 7.7980462 \\ 7.8469952 \\ 2.9922584 \end{pmatrix}$	$\begin{pmatrix} -36.51834885388397 \\ 24.63354895348917 \\ -11.913823909519436 \\ 36.57261085272549 \end{pmatrix}$	-8.0068537	9408.243073065663

Выясним, в какой из данных точек функция принимает минимальное значение. Отбросим результаты, полученные при $y < 0$, и получим, что минимальное значение функции $f(x)$ при заданных ограничениях достигается в точке:

$$x_{\min} = \begin{pmatrix} 6.061015807963965 \\ -11.876589956638057 \\ -2.111750020911596 \\ -4.480174039784763 \end{pmatrix}$$

Минимальное значение функции:

$$f_{\min}(x) = -138.21816218494826$$

Приложения (Я.П.: Haskell)

Весь исходный код этого приложения можно найти по адресу https://github.com/AJMC2002/opt-methods/tree/main.

Зависимости

- base ^>=4.17.2.0

- massiv >= 1.0.4 && < 1.1
- parallel >= 3.2.2 && < 3.3
- random >= 1.2.1 && < 1.3
- minimization2 — пользовательская библиотека с алгоритмами, используемыми для этой работы.

Библиотека

```haskell
-- lib/Minimization.hs
module Minimization (Minimization (..), mkMinimization) where

import Control.Parallel.Strategies (NFData)
import Data.Massiv.Array as A
import Function (Functions (..), mkFunctions)
import Utils (identity, inverse, split, zeros)
import Prelude as P

data Minimization r e = Minimization
    { getFunctions :: Functions r e
    , yIsZero :: Vector r e
    , yIsGreaterThanZero :: Vector r e -> Vector r e
    }

mkMinimization ::
    ( NumericFloat r e
    , Manifest r e
    , Load r Ix1 e
    , Load r Ix2 e
    , Ord e
    , Prim e
    , Show e
    , NFData e
    ) =>
    Matrix r e ->
    Vector r e ->
    Vector r e ->
    e ->
    e ->
    Minimization r e
mkMinimization matA vecB vecX0 r epsilon =
    Minimization
        { getFunctions = functions
        , yIsZero = yIsZero' n functions
        , yIsGreaterThanZero = yIsGreaterThanZero' n matA epsilon functions
        }
  where
    Sz2 n _ = size matA
    functions = mkFunctions matA vecB vecX0 r

yIsZero' :: (NumericFloat r e, Load r Ix1 e) => Int -> Functions r e -> Vector
r e
yIsZero' n functions = fPrimeInv functions $ zeros $ Sz1 n
```

```haskell
yIsGreaterThanZero' ::
    forall r e.
    ( NumericFloat r e
    , Manifest r e
    , Load r Ix1 e
    , Load r Ix2 e
    , Prim e
    , Ord e
    , Show e
    , NFData e
    ) =>
    Int ->
    Matrix r e ->
    e ->
    Functions r e ->
    Vector r e ->
    Vector r e
yIsGreaterThanZero' n matA epsilon functions vecXk0 = computeP $ recur vecXk0
(0 :: Int)
  where
    recur :: Vector r e -> Int -> Vector r e
    recur xk k
        | k >= 10000 || normL2 (xkNext !-! xk) <= epsilon = xk
        | otherwise = recur xkNext (k + 1)
      where
        (x, y) = split xk
        xkNext = xk !-! computeP (inverse fPrimeXk !>< fXk)
        fXk = computeP @P $ concat' 1 [up, down]
          where
            up = lagrangePrimeX functions x y
            down = singleton $ g functions x
        fPrimeXk = computeP @P $ concat' 2 [up, down]
          where
            up = computeP @P $ concat' 1 [upL, upR]
            down = computeP @P $ concat' 1 [downL, downR]
            upL = matA !+! ((2 * y) *. identity n)
            upR = resize' (Sz2 n 1) $ gPrime functions x
            downL = resize' (Sz2 1 n) $ gPrime functions x
            downR = singleton 0
```

Бинарный

```haskell
-- exe/Main.hs
module Main where

import Control.Parallel.Strategies (parMap, rpar)
import Data.Massiv.Array as A
import Function (Functions (..))
import Minimization (Minimization (..), mkMinimization)
import System.IO
import System.Random qualified as R
```

```haskell
import Utils (split)
import Prelude as P

main :: IO ()
main =
    let
        -- Initial values
        salt = 190902
        gen1 = R.mkStdGen salt
        gen2 = snd $ R.split gen1
        gen3 = snd $ R.split gen2
        rng = (-10 :: Double, 10)
        comp = ParN 0
        dim = 4
         tempA = computeP $ uniformRangeArray gen1 rng comp (Sz2 dim dim) ::
Matrix P Double
        matA = (tempA !+! computeP (transpose tempA)) ./ 2 -- this generates a
symmetric matrix
        vecB = computeP $ uniformRangeArray gen2 rng comp (Sz1 dim)
        vecX0 = computeP $ uniformRangeArray gen3 rng comp (Sz dim)
        r = 5
        epsilon = 1.0e-6
        minimization = mkMinimization matA vecB vecX0 r epsilon
        funs = getFunctions minimization
        -- Part 1 | When y = 0
        vecXSus = yIsZero minimization -- podozritel'niy
        fSus = f funs vecXSus
        distanceToCentre = normL2 (vecXSus !-! vecX0)
        isInSphere = distanceToCentre <= r
        -- Part 2 | When y > 0
        numPoints = 8
        gs = P.tail $ P.take (numPoints + 1) $ iterate (snd . R.split) gen3
        xy0s =
            P.map
                ( \g ->
                    let
                         xy' = computeP @P $ uniformRangeArray g rng comp (Sz1
dim + 1)
                        xy = makeArray @P (ParN 0) (Sz1 dim + 1) (\i -> if i ==
dim then abs (xy' ! i) else xy' ! i)
                     in
                        xy
                )
                gs
        xySols = parMap rpar (split . yIsGreaterThanZero minimization) xy0s
        fXY = parMap rpar (f funs . fst) xySols
     in
        do
            handle <- openFile "output.txt" WriteMode
            hPutStrLn handle "A"
            hPrint handle matA
            hPutStrLn handle "b"
```

```
hPrint handle vecB
hPutStrLn handle "x0"
hPrint handle vecX0
hPutStrLn handle "r"
hPrint handle r
hPutStrLn handle "** y = 0 **"
hPutStrLn handle "Solution"
hPrint handle vecXSus
hPutStrLn handle "Minimal value"
hPrint handle fSus
hPutStrLn handle "Distance to centre"
hPrint handle distanceToCentre
hPutStrLn handle "Is in sphere?"
hPrint handle isInSphere
hPutStrLn handle "** y > 0 **"
hPutStrLn handle "Initial vectors"
hPrint handle xy0s
hPutStrLn handle "Solutions"
hPrint handle xySols
hPutStrLn handle "Minimal values"
hPrint handle fXY
hClose handle
```