Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

РЕШЕНИЕ ЗАДАЧ ЛИНЕЙНОЙ АЛГЕБРЫ

По лабораторной работе по дисциплине

«Численные методы»

Выполнил:

Обучающийся гр. 439-3
(группа)

_____ А. С. Мазовец
(подпись) *(И. О. Фамилия)*

« _____ » _____ 2021г.
(дата)

Проверил:

ассистент кафедры АСУ
(должность, ученая степень, звание)

_____ А. Е. Косова
(подпись) *(И. О. Фамилия)*

_____ « _____ » _____ 2021г.
(оценка) *(дата)*

Томск 2021

1 Введение

1.1 Цели

При помощи метода Гаусса необходимо реализовать решение следующих задач:

а) Решение СЛАУ.

б) Поиск определителя матрицы

в) Поиск обратной матрицы

1.2 Входные данные

– m – тип задачи;

– n – порядок матрицы;

– Коэффициенты матрицы и вектор свободных коэффициентов (при решении СЛАУ, т.е. при m = 1):

```
1| a11 .. a1n [b1]
2| a21 .. a2n [b2]
3| ..  ..  ..  ..
4| a2n .. ann [bn]
```

1.3 Выходные данные

– Если используется метод Гаусса, то в выходной файл выводятся матрицы A (1) , A (2), …, A (n) . Если решалась система СЛАУ, то еще и вектора b (1) , b (2), …, b (n) . Если вычислялась обратная матрица – вектора e1 (n) , e2 (n), …, en (n)

– x* – вектор решения;

– ε – вектор невязки;

– ||ε|| – норма вектора невязки.

– При поиске определителя – его значение.

– При вычислении обратной матрицы – следующие величины:

– X – обратная матрица;

– ε – матрица невязки (AX – E);

– ||ε|| – норма матрицы невязки.

2 Алгоритм

2.1 Решение СЛАУ

а) Объединить матрицу коэффициентов и решения уравнений

б) Выполнить прямой ход Гаусса

в) Выполнить обратный ход Гаусса, что и будет являться решением СЛАУ

2.2 Поиск определителя

а) Выполнить прямой ход Гаусса

б) Перемножить главную диагональ полученной матрицы, что и будет являться значением определителя

2.3 Обратная матрица

а) Объединить матрицу коэффициентов и единичную матрицу такого же порядка

б) Выполнить прямой ход Гаусса

в) Отделить столбцы, которые принадлежали единичной матрицы и для каждого столбца выполнить обратный ход гаусса.

3 Результат работы

3.1 Тестовый пример №1

Пример выполнения программы, для уравнения матрицы:

```
[ 1  4  2  1  4  1
  6 -2  1  2  5  3
  3 -3  1  5  4  2
  1  5  9  1  9  1
  1 -1  4  0  1  8 ]
```

Входные данные указаны в I, выходные в stdout, для решения СЛАУ на рисунке 3.1:

```
┤ Echelon matrix (iterations) ├
[  1.000,   4.000,   2.000,   1.000,   4.000,   1.000
   6.000,  -2.000,   1.000,   2.000,   5.000,   3.000
   3.000,  -3.000,   1.000,   5.000,   4.000,   2.000
   1.000,   5.000,   9.000,   1.000,   9.000,   1.000
   1.000,  -1.000,   4.000,   0.000,   1.000,   8.000 ]

[  1.000,   4.000,   2.000,   1.000,   4.000,   1.000
   0.000,  -4.333,  -1.833,  -0.667,  -3.167,  -0.500
   0.000,  -5.000,  -1.667,   0.667,  -2.667,  -0.333
   0.000,   1.000,   7.000,   0.000,   5.000,   0.000
   0.000,  -5.000,   2.000,  -1.000,  -3.000,   7.000 ]

[  1.000,   4.000,   2.000,   1.000,   4.000,   1.000
   0.000,   1.000,   0.423,   0.154,   0.731,   0.115
   0.000,   0.000,  -0.090,  -0.287,  -0.197,  -0.049
   0.000,   0.000,   6.577,  -0.154,   4.269,  -0.115
   0.000,   0.000,  -0.823,   0.046,  -0.131,  -1.515 ]

[  1.000,   4.000,   2.000,   1.000,   4.000,   1.000
   0.000,   1.000,   0.423,   0.154,   0.731,   0.115
   0.000,   0.000,   1.000,   3.200,   2.200,   0.543
   0.000,   0.000,   0.000,  -3.223,  -1.551,  -0.560
   0.000,   0.000,   0.000,  -3.256,  -2.041,   1.298 ]

[  1.000,   4.000,   2.000,   1.000,   4.000,   1.000
   0.000,   1.000,   0.423,   0.154,   0.731,   0.115
   0.000,   0.000,   1.000,   3.200,   2.200,   0.543
   0.000,   0.000,   0.000,   1.000,   0.481,   0.174
   0.000,   0.000,   0.000,   0.000,   0.146,  -0.573 ]

[  1.000,   4.000,   2.000,   1.000,   4.000,   1.000
   0.000,   1.000,   0.423,   0.154,   0.731,   0.115
   0.000,   0.000,   1.000,   3.200,   2.200,   0.543
   0.000,   0.000,   0.000,   1.000,   0.481,   0.174
   0.000,   0.000,   0.000,   0.000,   1.000,  -3.929 ]

┤ Inverse matrix A (equations) ├
[  1.000,   4.000,   2.000,   1.000,   4.000,   1.000
   6.000,  -2.000,   1.000,   2.000,   5.000,   0.000
   3.000,  -3.000,   1.000,   5.000,   4.000,   0.000
   1.000,   5.000,   9.000,   1.000,   9.000,   0.000
   1.000,  -1.000,   4.000,   0.000,   1.000,   0.000 ]
```

```
 44|
 45| [  1.000,   4.000,   2.000,   1.000,   4.000,   0.000
 46|    6.000,  -2.000,   1.000,   2.000,   5.000,   1.000
 47|    3.000,  -3.000,   1.000,   5.000,   4.000,   0.000
 48|    1.000,   5.000,   9.000,   1.000,   9.000,   0.000
 49|    1.000,  -1.000,   4.000,   0.000,   1.000,   0.000 ]
 50|
 51| [  1.000,   4.000,   2.000,   1.000,   4.000,   0.000
 52|    6.000,  -2.000,   1.000,   2.000,   5.000,   0.000
 53|    3.000,  -3.000,   1.000,   5.000,   4.000,   1.000
 54|    1.000,   5.000,   9.000,   1.000,   9.000,   0.000
 55|    1.000,  -1.000,   4.000,   0.000,   1.000,   0.000 ]
 56|
 57| [  1.000,   4.000,   2.000,   1.000,   4.000,   0.000
 58|    6.000,  -2.000,   1.000,   2.000,   5.000,   0.000
 59|    3.000,  -3.000,   1.000,   5.000,   4.000,   0.000
 60|    1.000,   5.000,   9.000,   1.000,   9.000,   1.000
 61|    1.000,  -1.000,   4.000,   0.000,   1.000,   0.000 ]
 62|
 63| [  1.000,   4.000,   2.000,   1.000,   4.000,   0.000
 64|    6.000,  -2.000,   1.000,   2.000,   5.000,   0.000
 65|    3.000,  -3.000,   1.000,   5.000,   4.000,   0.000
 66|    1.000,   5.000,   9.000,   1.000,   9.000,   0.000
 67|    1.000,  -1.000,   4.000,   0.000,   1.000,   1.000 ]
 68|
 69| ┤ Gauss solution (iterations) ├──
 70| [ -3.929
 71|    0.174,   0.481
 72|    0.543,   2.200,   3.200
 73|    0.115,   0.731,   0.154,   0.423
 74|    1.000,   4.000,   1.000,   2.000,   4.000 ]
 75|
 76| [  2.064
 77|    9.186,   3.200
 78|    2.987,   0.154,   0.423
 79|   16.716,   1.000,   2.000,   4.000 ]
 80|
 81| [  2.581
 82|    2.669,   0.423
 83|   14.651,   2.000,   4.000 ]
 84|
 85| [  1.577
 86|    9.489,   4.000 ]
 87|
 88| [  3.182 ]
 89|
 90| ──────────────────────────────────
 91|
 92|
 93| - Gauss solution (result)
 94| (  3.182,   1.577,   2.581,   2.064,  -3.929 )
 95|
 96| - Vector epsilon
 97| (-1.77636E-15, -1.42109E-14, -8.88178E-15,  7.10543E-15,  5.32907E-15 )
 98|
 99| - Normal vector epsilon
100| 1.90493E-14
101|
102| ------------------------------------
103|
104| - Det matrix A (result)
105| -1449.000
106|
107| - Inverse matrix A (result)
```

```
108|  [  0.404,   0.122,  -0.079,  -0.251,   0.353
109|     0.466,  -0.064,  -0.032,  -0.179,   0.193
110|     0.137,  -0.077,   0.016,  -0.063,   0.338
111|     0.398,  -0.193,   0.238,  -0.203,   0.246
112|    -0.484,   0.120,  -0.016,   0.324,  -0.512 ]
113|
114|  - Matrix epsilon (AX - E)
115|  [ 0.00000E0  ,  0.00000E0  , -1.38778E-17,  0.00000E0  ,  0.00000E0
116|   -3.55271E-15,  2.22045E-16,  1.94289E-16,  8.88178E-16,  8.88178E-16
117|   -2.22045E-15,  3.33067E-16, -1.11022E-16,  4.44089E-16,  0.00000E0
118|    1.77636E-15,  0.00000E0  , -5.55112E-17,  0.00000E0  ,  1.77636E-15
119|   -3.33067E-16,  1.11022E-16, -1.21431E-16, -1.11022E-16,  4.44089E-16 ]
120|
121|  - Normal matrix epsilon
122|  5.11851E-15
```

Рисунок 3.1 — Тестовый пример №1

4 Вывод

При помощи метода Гаусса мной были реализованы решения следующих задач :

а) Решение СЛАУ

б) Поиск определителя матрицы

в) Поиск обратной матрицы

5 Исходный код программы

5.1 Methods.hs

```
 1| module Methods
 2|     ( Matrix
 3|     , Vector
 4|     , matrixE
 5|     , echelonsTriangle
 6|     , echelons
 7|     , echelonTriangle
 8|     , echelon
 9|     , gausses
10|     , gauss
11|     , det
12|     , equationsE
13|     , inverse
14|     ) where
15|
16| import Data.List
17|
18| type Matrix = [ Vector ]
19| type Vector = [ Double ]
20|
21|
22| --- Echelon --- --- --- --- --- --- --- --- --- --- --- ---
23|
24| echelon :: Matrix -> Matrix
25| echelon = last . echelons
26|
27| echelons :: Matrix -> [Matrix]
28| echelons = zipWith (zipWith (++)) zeros . echelonsTriangle
29|   where
30|     zeros  = (++) <*> take 4. repeat . last <$> zeros'
31|     zeros' = scanl (\acc x -> acc ++ [x]) [[]] (tail . inits
    $ repeat 0)
32|
33|
34| echelonTriangle :: Matrix -> Matrix
35| echelonTriangle = last . echelonsTriangle
36|
37| echelonsTriangle :: Matrix -> [Matrix]
38| echelonsTriangle = (zipWith (++) <$> rows <*> id) . echelons'
39|   where
40|     rows = scanl (\acc x -> acc ++ [(1:) . divFst $ head x])
    []
41|
42|
43| echelons' :: Matrix -> [Matrix]
44| echelons' = (++[[]]) . takeWhile (/=[]) . iterate (subFst .
    map divFst)
45|
```

```
46| subFst :: Matrix -> Matrix
47| subFst m = do
48|     vi <- tail m
49|     return $ zipWith (-) vi (head m)
50|
51| divFst :: Vector -> Vector
52| divFst v = map (/ head v) (tail v)
53|
54|
55| --- Gauss --- --- --- --- --- --- --- --- --- --- --- --- ---
56|
57| gauss :: Matrix -> Vector
58| gauss = reverse . concat . map head . gausses
59|
60| gausses :: Matrix -> [Matrix]
61| gausses
62|   = takeWhile (/= [])
63|   . iterate step
64|   . reverse
65|   . map (reverse . tail)
66|   . echelonTriangle
67|   where
68|     step m' = do
69|         vi <- tail m'
70|         let vTail = tail . tail $ vi
71|         let vVarI = head . tail $ vi
72|         let vResI = head $ vi
73|         let vVarF = head . head $ m'
74|
75|         return $ vResI - vVarI * vVarF : vTail
76|
77|
78| --- Determinant -- --- --- --- --- --- --- --- --- --- ---
79|
80| det :: Matrix -> Double
81| det = product . concat . map (map head) . echelons'
82|
83|
84| --- Inverse Matrix --- --- --- --- --- --- --- --- --- ---
85|
86| matrixE :: Matrix
87| matrixE = do
88|     vi <- inits $ repeat 0
89|     return $ vi ++ (1: repeat 0)
90|
91| equationsE :: Matrix -> [Matrix]
92| equationsE mm = zipWith (\m e -> zipWith (\mi ei -> mi ++
    [ei]) m e) (replicate (length mm) mm) matrixE
93|
94| inverse :: Matrix -> Matrix
95| inverse = transpose . map gauss . EquationsE
```

5.2 Main.hs

```
 1| import Methods
 2| import System.IO
 3| import Data.List
 4| import Text.Printf
 5|
 6| main :: IO ()
 7| main = do
 8|     let inp = "I"
 9|
10|     hInp    <- openFile inp ReadMode
11|         matrixX  <-  map  (map  read  .  words)  .  lines  <$>
    hGetContents hInp :: IO Matrix
12|     matrix  <- return $ map init matrixX
13|     xs      <- return $ map last matrixX
14|
15|     putStrLn "\n┣┫ Echelon matrix (iterations) ┣━━━━"
16|     putStrLn . showMatrixIters "%7.3F" $ echelons matrixX
17|
18|     putStrLn "\n┣┫ Inverse matrix A (equations) ┣━━━━"
19|     putStrLn . showMatrixIters "%7.3F" $ equationsE matrix
20|
21|     putStrLn "\n┣┫ Gauss solution (iterations) ┣━━━━"
22|     putStrLn . showMatrixIters "%7.3F" $ gausses matrixX
23|
24|     putStrLn "\n━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━"
25|     putStrLn ""
26|
27|     putStrLn "\n━ Gauss solution (result)"
28|     putStrLn . showVector "%7.3F" $ gauss matrixX
29|
30|     putStrLn "\n━ Vector epsilon"
31|       let vEpsi = zipWith (-) (map sum $ map (zipWith (*)
    (gauss matrixX)) matrix) xs
32|     putStrLn $ showVector "% -12.5E" vEpsi
33|
34|     putStrLn "\n━ Normal vector epsilon"
35|      putStrLn . printf "%-12.5E" $ sqrt . sum $ map (** 2)
    vEpsi
36|
37|     putStrLn "\n-----------------------------------"
38|
39|     putStrLn "\n━ Det matrix A (result)"
40|     putStrLn . printf "%7.3F" $ det matrix
41|
42|     putStrLn "\n━ Inverse matrix A (result)"
43|     putStrLn . showMatrix "%7.3F" $ inverse matrix
44|
45|     putStrLn "\n━ Matrix epsilon (AX - E)"
46|     let mEpsi = subMatrix (mulMatrix matrix (inverse matrix))
    matrixE
47|     putStrLn $ showMatrix "% -12.5E" mEpsi
```

```
48|
49|     putStrLn "\n━ Normal matrix epsilon"
50|      putStrLn . printf "%-12.5E" . sqrt . sum . map (** 2) $
    concat mEpsi
51|
52|     return ()
53|
54|
55| mulMatrix :: Matrix -> Matrix -> Matrix
56| mulMatrix a b = do
57|     a' <- a
58|     b' <- return $ map (sum . zipWith (*) a') $ transpose b
59|     return b'
60|
61| subMatrix :: Matrix -> Matrix -> Matrix
62| subMatrix a b = zipWith (zipWith (-)) a b
63|
64|
65|
66| showMatrix :: String -> Matrix -> String
67| showMatrix f m = ('[':) . (++" ]") . intercalate "\n " $ map
    (intercalate ", " . map (printf f)) m
68|
69| showMatrixIters :: String -> [Matrix] -> String
70| showMatrixIters f ms = intercalate "\n\n" $ map (showMatrix
    f) ms
71|
72|
73| showVector :: String -> Vector -> String
74| showVector f v = ('(':) . (++" )") . intercalate ", " $ map
    (printf f) v
```