**МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА**

**СПОРТУ УКРАЇНИ**

**Національний технічний університет України**

**“Київський політехнічний інститут”**

**Чисельні методи**

**Лабораторна робота №3**

Знаходження власних чисел методом Данилевського

Виконав:

Сочка Олександр Олександрович

Варіант 17

**Київ 2015**

**Постановка задачі**

Знайти форму Фробеніюса, та власні числа матриці:

**Варіант 17**

A:

7.03 0.94 1.13 1.135

0.94 3.39 1.3 0.16

1.13 1.3 6.11 2.1

1.135 0.16 2.1 5.33

MInv:

1.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,

0.0000000000, 1.0000000000, 0.0000000000, 0.0000000000,

1.1350000000, 0.1600000000, 2.1000000000, 5.3300000000,

0.0000000000, 0.0000000000, 0.0000000000, 1.0000000000,

M:

1.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,

0.0000000000, 1.0000000000, 0.0000000000, 0.0000000000,

-0.5404761905, -0.0761904762, 0.4761904762, -2.5380952381,

0.0000000000, 0.0000000000, 0.0000000000, 1.0000000000,

A:

6.4192619048, 0.8539047619, 0.5380952381, -1.7330476190,

0.2373809524, 3.2909523810, 0.6190476190, -3.1395238095,

2.7619932143, 3.2481342857, 12.1497857143, -30.6256328571,

0.0000000000, 0.0000000000, 1.0000000000, 0.0000000000,

MInv:

1.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,

2.7619932143, 3.2481342857, 12.1497857143, -30.6256328571,

0.0000000000, 0.0000000000, 1.0000000000, 0.0000000000,

0.0000000000, 0.0000000000, 0.0000000000, 1.0000000000,

M:

1.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,

-0.8503322127, 0.3078690448, -3.7405429227, 9.4286843348,

0.0000000000, 0.0000000000, 1.0000000000, 0.0000000000,

0.0000000000, 0.0000000000, 0.0000000000, 1.0000000000,

A:

5.6931591791, 0.2628908434, -2.6559721757, 6.3181508330,

7.4059240861, 16.1668408209, -75.9350264123, 108.0405941703,

0.0000000000, 1.0000000000, 0.0000000000, 0.0000000000,

0.0000000000, 0.0000000000, 1.0000000000, 0.0000000000,

MInv:

7.4059240861, 16.1668408209, -75.9350264123, 108.0405941703,

0.0000000000, 1.0000000000, 0.0000000000, 0.0000000000,

0.0000000000, 0.0000000000, 1.0000000000, 0.0000000000,

0.0000000000, 0.0000000000, 0.0000000000, 1.0000000000,

M:

0.1350270389, -2.1829606451, 10.2532817687, -14.5884015167,

0.0000000000, 1.0000000000, 0.0000000000, 0.0000000000,

0.0000000000, 0.0000000000, 1.0000000000, 0.0000000000,

0.0000000000, 0.0000000000, 0.0000000000, 1.0000000000,

A:

21.8600000000, -166.0284750000, 520.6808585000, -568.3005549875,

1.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,

0.0000000000, 1.0000000000, 0.0000000000, 0.0000000000,

0.0000000000, 0.0000000000, 1.0000000000, 0.0000000000,

Результат:

21.8600000000, -166.0284750000, 520.6808585000, -568.3005549875,

1.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,

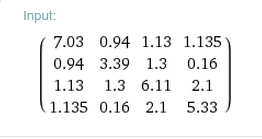
0.0000000000, 1.0000000000, 0.0000000000, 0.0000000000,

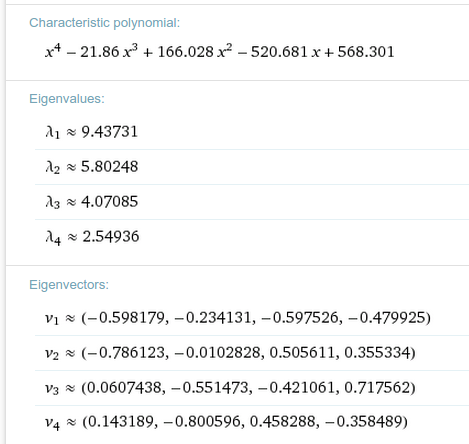
0.0000000000, 0.0000000000, 1.0000000000, 0.0000000000,

**Характеристичне рівняння:**

x^4 — 21.86 x^3 + 166.028 x^2 — 520.681 x + 568.301

**Результат в WolframAlpha**



****

**Висновок:**

Метод Данилевського є порівняно ефективним методом (відносно базової формули визначення через детермінант). Особливістю є вирішення повної проблеми знаходження повного харакетристичного рівняння. Базується на властивостях подібності матриць, та зведенні матриці до форми Фробеніуса.

**Лістинг програми**

**import** scala.io.Source

**object** Lab3 **extends** App {

**type** Vec = Seq[Double]

**type** Matrix = Seq[Vec]

**def** printMatrix(A: Matrix, msg: String = "") = {

**if** (msg.nonEmpty) *println*(s"**$**msg: ")

**for** (line <- A) {

**for** (elem <- line) *print*(f"**$**elem%16.10f,")

*println*()

}

*println*()

}

**def** mul(A: Matrix, B: Matrix): Matrix = {

**for** (i <- 0 until *N*) **yield** {

**for** (k <- 0 until *N*) **yield** {

A(i).zipWithIndex.map({ **case** (a, j) => a \* B(j)(k)}).sum

}

}

}

**val** *N* = 4

**var** *A*: Matrix =

**for** (*line* <- Source.*fromFile*("input/lab3.txt").getLines().toVector) **yield**

**for** (*word* <- line.split(' ').toVector) **yield** word.toDouble

**for** (i <- 0 until *N* - 1) {

**def** MInv(row: Int): Matrix = {

**val** NC = row - 1

**for** (j <- 0 until *N*) **yield** j **match** {

**case** NC => *A*(row)

**case** \_ => (*Vector*.fill(j)(0.0) :+ 1.0) ++ *Vector*.fill(*N* - j - 1)(0.0)

}

}

**def** M(row: Int): Matrix = {

**val** NC = row - 1

**for** (j <- 0 until *N*) **yield** j **match** {

**case** NC => *A*(row).zipWithIndex.map {

**case** (v, NC) => 1.0 / *A*(row)(NC)

**case** (v, col) => -v / *A*(row)(NC)

}

**case** \_ => (*Vector*.fill(j)(0.0) :+ 1.0) ++ *Vector*.fill(*N* - j - 1)(0.0)

}

}

**val** minv = MInv(*N* - i - 1)

**val** m = M(*N* - i - 1)

*A* = *mul*(*mul*(minv, *A*), m)

*printMatrix*(minv, "MInv")

*printMatrix*(m, "M")

*printMatrix*(*A*, "A")

}

}