Министерство образования и науки Российской федерации

Федеральное государственное автономное образовательное учреждение высшего образования

САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

Лабораторная работа №3 по дисциплине

“Системное программное обеспечение”

Вариант №3

Выполнили:

студенты группы R3335

Кузьминов А. Ю.

Никитенко И. Р.

Гринжола Д. Ю.

Преподаватель:

кандидат технических наук

Капитонов А. А.

Санкт-Петербург

2018

Задание 1. Лабиринт.

Создать игру «лабиринт». Желательный размер лабиринта – 50 х 50 клеток (можно выбрать свой размер, но не менее 20х20), пользователь может перемещаться вверх, вниз, вправо и влево. Каждый шаг смещает пользователя на одну клетку, если не мешает стена. Вход и выход из лабиринта генерируются случайно при каждом запуске игры. При прохождении лабиринта пользователю выдается информация о затраченном количестве шагов, времени прохождения, предложение завершить игру и начать заново.

Пример работы реализованной программы:

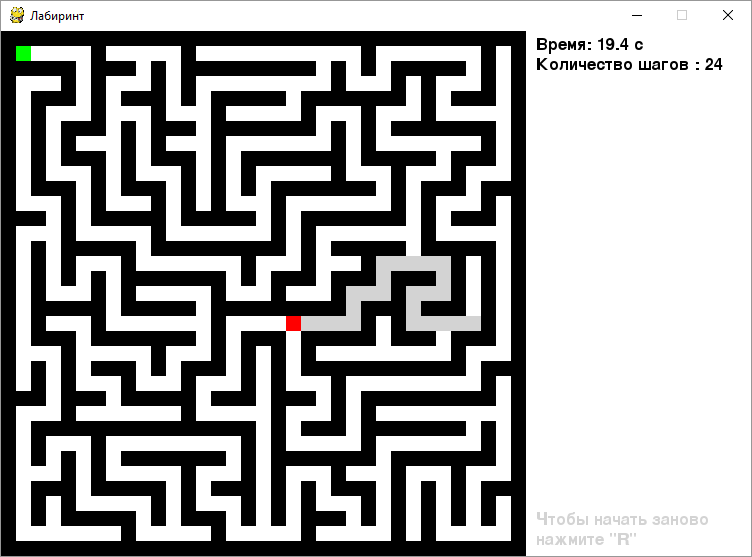


Рисунок 1. Пользовательский интерфейс.

Код программы:

|  |
| --- |
| **import** pygame  **import** time  **from** labyrinth **import** \*    *#задаем размеры окна*  display = (750, 525)  width = 35  height = 35    *#задаем цвета элементов игры*  wall\_color = (0, 0, 0)  hero\_color = (255, 0, 0)  exit\_color = (0, 255, 0)  path\_color = (211, 211, 211)  background\_color = (255, 255, 255)      **class** Interface:  **def** \_\_init\_\_(self):  pygame.init()  pygame.font.init()  pygame.display.set\_caption('Лабиринт')  self.screen = pygame.display.set\_mode(display)  self.clock = pygame.time.Clock()  self.FPS = 60  self.heroX = 1  self.heroY = 1  self.steps = 0  self.startPoint = (1, 1)  self.endPoint = (1, 1)  self.clock = pygame.time.Clock()  self.font = pygame.font.Font(None, 25)      **def** draw(self):  self.maze = Labyrinth(width, height)  self.screen.fill(background\_color)  pygame.draw.rect(self.screen, exit\_color, (self.endPoint[0] \* 15, self.endPoint[1] \* 15, 15, 15) )    *#вывод инструкции на экран*  restart1 = self.font.render("Чтобы начать заново ", True, path\_color)  restart2 = self.font.render("нажмите **\"**R**\"**", True, path\_color)  self.screen.blit(restart1, (535, 480))  self.screen.blit(restart2, (535, 500))    pygame.display.update()    *#построение лабиринта*  **for** i **in** range(width):  **for** j **in** range(height):  **if** self.maze.get(i, j) == 'wall':  pygame.draw.rect(self.screen, wall\_color, (i \* 15, j \* 15, 15, 15))    **while** True:  self.heroX = random.randint(1, width - 1)  self.heroY = random.randint(1, height - 1)  **if** self.maze.get(self.heroX, self.heroY) == 'cell':  self.startPoint = (self.heroX, self.heroY)  **break**    **while** True:  x = random.randint(1, width - 1)  y = random.randint(1, height - 1)  **if** self.maze.get(x, y) == 'cell':  self.endPoint = (x, y)  **break**      **def** main(self):  self.startTime = time.time()  self.draw()  success = False    **while** True:  self.clock.tick(self.FPS)    **if** (self.heroX, self.heroY) == self.endPoint:  **while** True:  **for** event **in** pygame.event.get():  **if** event.type == pygame.KEYDOWN:  **if** event.key == pygame.K\_r:  self.draw()  self.startTime = time.time()  self.heroX = self.startPoint[0]  self.heroY = self.startPoint[1]  **break**    **for** event **in** pygame.event.get():    *#выход из игры*  **if** event.type == pygame.QUIT:  pygame.quit()  quit()    *#настройка управления*  **if** event.type == pygame.KEYDOWN:  **if** event.key == pygame.K\_r:  self.draw()  self.startTime = time.time()    **while** True:  self.heroX = random.randint(1, width - 1)  self.heroY = random.randint(1, height - 1)  **if** self.maze.get(self.heroX, self.heroY) == 'cell':  **break**    **while** True:  x = random.randint(1, width - 1)  y = random.randint(1, height - 1)  **if** self.maze.get(x, y) == 'cell':  self.endPoint = (x, y)  **break**    **elif** event.key == pygame.K\_UP **or** event.key == pygame.K\_w:  **if** self.maze.get(self.heroX, self.heroY - 1) != 'wall':  pygame.draw.rect(self.screen, path\_color, (self.heroX \* 15, self.heroY \* 15, 15, 15))  self.heroY -= 1  self.steps += 1    **elif** event.key == pygame.K\_LEFT **or** event.key == pygame.K\_a:  **if** self.maze.get(self.heroX - 1, self.heroY) != 'wall':  pygame.draw.rect(self.screen, path\_color, (self.heroX \* 15, self.heroY \* 15, 15, 15))  self.heroX -= 1  self.steps += 1    **elif** event.key == pygame.K\_DOWN **or** event.key == pygame.K\_s:  **if** self.maze.get(self.heroX, self.heroY + 1) != 'wall':  pygame.draw.rect(self.screen, path\_color, (self.heroX \* 15, self.heroY \* 15, 15, 15))  self.heroY += 1  self.steps += 1    **elif** event.key == pygame.K\_RIGHT **or** event.key == pygame.K\_d:  **if** self.maze.get(self.heroX + 1, self.heroY) != 'wall':  pygame.draw.rect(self.screen, path\_color, (self.heroX \* 15, self.heroY \* 15, 15, 15))  self.heroX += 1  self.steps += 1    pygame.draw.rect(self.screen, hero\_color, (self.heroX \* 15, self.heroY \* 15, 15, 15))    *#вывод статистики на экран*  countedTime = time.time() - self.startTime  timeCounter = self.font.render("Время: {0:.1f} с ".format(countedTime), True, wall\_color, background\_color)  self.screen.blit(timeCounter, (535, 5))    stepsCounter = self.font.render("Количество шагов : {} ".format(self.steps), True, wall\_color, background\_color)  self.screen.blit(stepsCounter, (535, 25))    pygame.display.update()      **if** \_\_name\_\_ == '\_\_main\_\_':  interface = Interface()  interface.main() |

Файл labyrinth.py:

|  |
| --- |
| **import** random  **from** math **import** copysign    **class** Labyrinth:  **def** \_\_init\_\_(self, height, width):  self.w = width  self.h = height  self.maze = []  **for** i **in** range(self.w):  self.maze.append([])  **for** j **in** range(self.h):  **if** ((i % 2 != 0) **and** (j % 2 != 0) **and** (i < self.w **and** j < self.h)):  self.maze[i].append('cell')  **else**:  self.maze[i].append('wall')    self.createLabyrinth()    **def** removeWall(self, first, second, v, nv):  dx = second[0] - first[0]  dy = second[1] - first[1]    **if** dx == 0:  addx = 0  **else**:  addx = copysign(1, dx)    **if** dy == 0:  addy = 0  **else**:  addy = copysign(1, dy)    x = int(first[0] + addx)  y = int(first[1] + addy)    self.maze[x][y] = 'cell'  v.append( (x, y) )  nv.remove( (x, y) )    **return** v, nv    **def** getNeighbours(self, cell, v):  up = (cell[0], cell[1]+2)  down = (cell[0], cell[1]-2)  right = (cell[0]+2, cell[1])  left = (cell[0]-2, cell[1])  neighbours = [up, down, right, left]  cells = []    **for** n **in** neighbours:  **if** ( n[0] > 0 **and** n[0] < self.w ) **and** ( n[1] > 0 **and** n[1] < self.h ):    **if** ( **not** self.maze[n[0]][n[1]] == 'wall' ) **and** **not** ( n **in** v ):  cells.append(n)    **return** cells    **def** getUnvisited(self, v):  unvisited = []  **for** i **in** range(1, self.w):  **for** j **in** range(1, self.h):  **if** **not** (i, j) **in** v:  unvisited.append( (i, j) )    **return** unvisited    **def** createLabyrinth(self):  stack = []  visited = [(1,1)]  none\_visited = self.getUnvisited(visited)    current\_cell = (1,1)    **while** len(none\_visited) != 0:  neighbours = self.getNeighbours(current\_cell, visited)  **if** neighbours:  neighbour\_cell = random.choice(neighbours)  stack.append(current\_cell)  visited, none\_visited = self.removeWall(current\_cell, neighbour\_cell, visited, none\_visited)  current\_cell = neighbour\_cell  visited.append(current\_cell)  none\_visited.remove(current\_cell)  neighbours = 0  **elif** stack:  current\_cell = stack.pop()  **else**:  current\_cell = random.choice(none\_visited)  visited.append(current\_cell)  none\_visited.remove(current\_cell)    **def** get(self, i, j):  **return** self.maze[i][j] |

Задание 2. Поиск выхода.

Написать программу, которая будет автоматически передвигать пользователя из задания 1 и искать выход из лабиринта. При прохождении лабиринта выдается информация о количестве шагов.

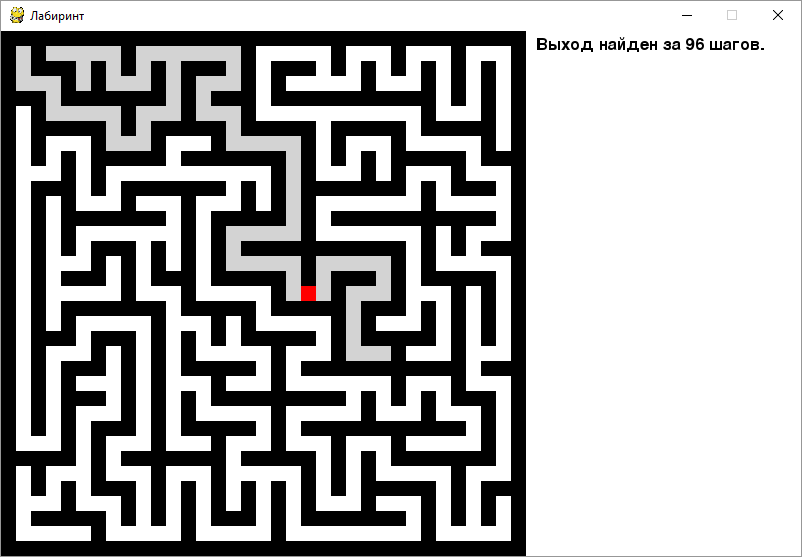
Пример работы реализованной программы: 

Рисунок 2. Пользовательский интерфейс.

Код программы:

|  |
| --- |
| **import** pygame  **import** sys  **from** labyrinth **import** \*  sys.path.append('../')    *#задаем размеры окна*  display = (800, 525)  width = 35  height = 35    *#задаем цвета элементов игры*  wall\_color = (0, 0, 0)  hero\_color = (255, 0, 0)  path\_color = (211, 211, 211)  background\_color = (255, 255, 255)    **class** auto\_walker:  **def** \_\_init\_\_(self):  *#создание лабиринта и условий игры*  self.visited = []  self.maze = Labyrinth(width, height)  self.steps = 0  self.success = False  self.start\_point, self.end\_point = (1, 1), (15, 15)    **while** True:  x = random.randint(1, width - 1)  y = random.randint(1, height - 1)  **if** self.maze.get(x, y) == 'cell':  self.start\_point = (x, y)  **break**    **while** True:  endX = random.randint(1, width - 1)  endY = random.randint(1, height - 1)  **if** self.maze.get(endX, endY) == 'cell':  self.end\_point = (endX, endY)  **break**    self.walk(self.start\_point)    **def** walk(self, start\_point):  *#автоматическое прохождение пути*  **if** start\_point == self.end\_point **or** self.success:  self.success = True  **return**    self.visited.append(start\_point)  neighbours = self.find\_neighbours(start\_point)  numberOfNeighbours = len(neighbours)    **while** numberOfNeighbours == 1:  self.steps += 1  start\_point = neighbours[0]  self.visited.append(start\_point)  **if** start\_point == self.end\_point **or** self.success:  self.success = True  **return**  neighbours = self.find\_neighbours(start\_point)  numberOfNeighbours = len(neighbours)    **for** neighbour **in** neighbours:  self.steps += 1  self.walk(neighbour)  **if** self.success:  **return**  self.steps += 1    **if** start\_point == self.end\_point:  **return**    **def** draw\_path(self):  pygame.init()  pygame.font.init()  pygame.display.set\_caption('Лабиринт')  self.font = pygame.font.Font(None, 25)  self.screen = pygame.display.set\_mode(display)  self.screen.fill(background\_color)    *#отрисовка пути (следа) персонажа*  **for** i **in** range(width):  **for** j **in** range(height):  **if** self.maze.get(i, j) == 'wall':  pygame.draw.rect(self.screen, wall\_color, (i \* 15, j \* 15, 15, 15))  **if** (i, j) **in** self.visited:  pygame.draw.rect(self.screen, path\_color, (i \* 15, j \* 15, 15, 15))    pygame.draw.rect(self.screen, hero\_color, (self.start\_point[0] \* 15, self.start\_point[1] \* 15, 15, 15))  pygame.draw.rect(self.screen, wall\_color, (self.end\_point[0] \* 15, self.end\_point[1] \* 15, 15, 15))    *#вывод результата прохождения лабиринта на экран*  stepsCounter = self.font.render("Выход найден за {} шагов.".format(self.steps), True, wall\_color)  self.screen.blit(stepsCounter, (535, 5))  pygame.display.update()    **while** True:  **for** event **in** pygame.event.get():  **if** event.type == pygame.QUIT:  pygame.quit()  quit()    **def** find\_neighbours(self, cell):  *#проверка соседних клеток*  up = (cell[0], cell[1] + 1)  down = (cell[0], cell[1] - 1)  left = (cell[0] - 1, cell[1])  right = (cell[0] + 1, cell[1])  neighbours = [up, down, left, right]  cells = []    **for** neighbour **in** neighbours:  **if** (neighbour[0] > 0 **and** neighbour[0] < width) **and** (neighbour[1] > 0 **and** neighbour[1] < height):  **if** (self.maze.get(neighbour[0], neighbour[1]) != 'wall') **and** **not** neighbour **in** self.visited:  cells.append(neighbour)    **return** cells    **if** \_\_name\_\_ == '\_\_main\_\_':  aw = auto\_walker()  aw.draw\_path() |

Задание 3. ТАУ.

Написать свои алгоритмы для вычисления управляемости, наблюдаемости и ранга матрицы. Сравните быстродействие полученных функций с аналогами NumPy.

Пример работы реализованной программы:

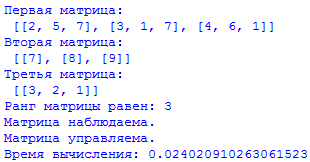


Рисунок 3. Программа, выполненная с помощью написанного нами алгоритма.

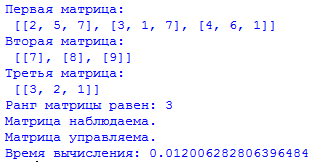


Рисунок 4. Программа, выполненная с помощью библиотеки NumPy.

Код программы:

|  |
| --- |
| **import** time    *#создание матриц для проверки*  matrix1 = [[2, 5, 7], [3, 1, 7], [4, 6, 1]]  matrix2 = [[7], [8], [9]]  matrix3 = [[3, 2, 1]]      **def** row(matrix):  **return** len(matrix)      **def** col(matrix):  **return** len(matrix[0])      **def** swap(matrix,i):  k = 0  **if** i == col(matrix) **or** i == row(matrix):  **pass**  **else**:  **for** step **in** range (col(matrix)-i):  **if** matrix[i][i] == 0:  matrix[i], matrix[i + step] = matrix[i + step], matrix[i]  **else**:  k = 1  **return** matrix, k      **def** check(matrix):  list = []  **for** i **in** range (0, row(matrix)):  flag = 0  **for** j **in** range(0, col(matrix)):  **if** matrix[i][j] != 0:  flag = 1  **if** flag == 0:  list.append(i)    step = 0  **for** i **in** list:  **del** matrix[i - step]  step += 1    list = []  **for** j **in** range(0,len(matrix[0])):  flag = 0  **for** i **in** range (0, row(matrix)):  **if** matrix [i] [j] != 0:  flag = 1  **if** flag == 0:  list.append(j)    step = 0  **for** j **in** list:  **for** i **in** range(0,row(matrix)):  **del** matrix[i][j - step]  step += 1    **return** matrix    **def** multiplication(matrix1, matrix2):  i, j, k = 0, 0, 0  resultLine = 0    **if** row(matrix2) == col(matrix1):  result = []  **for** i **in** range(row(matrix1)):  line=[]  **for** j **in** range (len(matrix2[i])):  **for** k **in** range(col(matrix1)):  resultLine += matrix1[i][k] \* matrix2[k][j]  line.append(resultLine)  resultLine = 0  result.append(line)  **else**:  **return** "Перемножение матриц возможно."    **return** result    **def** power(matrix,power):  i, j, k, p = 0, 0, 0, 1  result\_power = 0    **if** row(matrix) == col(matrix):  new\_matrix = []  **while** p < power:  p += 1  **for** i **in** range (row(matrix)):  line = []  **for** j **in** range(col(matrix)):  **for** k **in** range (row(matrix[i])):  result\_power += matrix[i][k] \* matrix[k][j]  line.append(result\_power)  resultPower = 0  new\_matrix.append(line)  **else**:  **return** "Матрица должна быть квадратной."    **return** new\_matrix      **def** rank(matrix,n):  checkMatrix = check(matrix)  fMatrix, k = swap(checkMatrix, n)  minim = row(fMatrix) **if** (row(fMatrix)) <= col (fMatrix) **else** col(fMatrix)  rank\_g = row(matrix)  N = n  **if** (n < minim - 1) **or** k != 0:  **for** i **in** range(n + 1, row(fMatrix)):  koef = -fMatrix[i][n] / fMatrix[n][n]  **for** j **in** range(n, col(fMatrix)):  fMatrix[i][j] = koef \* fMatrix[n][j] + fMatrix[i][j]  **return** rank(fMatrix, n + 1)  **else**:  **return** rank\_g      **def** observability\_of\_matrix(matrix1,matrix2):  n = len(matrix2) \* len(matrix2[0])  result = [0] \* n  **for** i **in** range(n):  result[i] = [0] \* len(matrix2[0])    result[0] = matrix2  i = 0  **while** i < (len(matrix2[0]) - 1):  i += 1  result[i] = multiplication(matrix2,matrix1)  matrix1 = power(matrix1,i + 1)    result1 = [0] \* n  **for** i **in** range(n):  result1[i] = []    i = 0  **for** j **in** range(len(matrix2[0])):  **for** k **in** range(len(result[0])):  result1[i] = result[j][k]  i +=1    **return** result1      **def** check\_observability(matrix1,matrix2):  rank\_of\_matrix = rank(observability\_of\_matrix(matrix1,matrix2),0)  Columns = col(matrix2)  **if** Columns == rank\_of\_matrix:  **return** "Матрица наблюдаема."  **else**:  **return** "Матрица не является наблюдаемой."      **def** controllability\_of\_matrix(matrix1,matrix2):  n = len(matrix2) \* len(matrix2[0])  result = [0] \* n  **for** i **in** range(n):  result[i] = [0] \* len(matrix2)    result[0] = matrix2  i = 0  **while** i < (len(matrix2) - 1):  i += 1  result[i] = multiplication(matrix1,matrix2)  matrix1 = power(matrix1,i + 1)    result1 = [0] \* len(matrix2)  **for** i **in** range(len(matrix2)):  result1[i] = []    **for** i **in** range(len(matrix2)):  **for** j **in** range(len(matrix2)):  result1[i] = result1[i] + result[j][i]    **return** result1      **def** check\_controllability(matrix1,matrix2):  rank\_of\_matrix = rank(controllability\_of\_matrix(matrix1,matrix2),0)  rows = row(matrix2)  **if** rows == rank\_of\_matrix:  **return** "Матрица управляема."  **else**:  **return** "Матрица не является управляемой."    *#вывод результатов на экран*  **print**("Первая матрица: **\n**", matrix1)  **print**("Вторая матрица: **\n**", matrix2)  **print**("Третья матрица: **\n**", matrix3)  start\_time = time.time()  **print**("Ранг матрицы равен:", rank(matrix1, 0))  **print**(check\_observability(matrix1, matrix3))  **print**(check\_controllability(matrix1, matrix2))  end\_time = time.time()  time = end\_time - start\_time  **print**("Время вычисления: " + str(time)) |

Файл 3.3\_np.py:

|  |
| --- |
| **import** numpy **as** np  **import** time    *#создание матриц для проверки*  matrix1 = [[2, 5, 7], [3, 1, 7], [4, 6, 1]]  matrix2 = [[7], [8], [9]]  matrix3 = [[3, 2, 1]]    **def** observability\_of\_matrix(matrix1,matrix2):  size = np.shape(matrix2)  n = size[0] \* size[1]  result = [0] \* n  **for** i **in** range (n):  result[i] = [0] \* size[1]    result[0] = matrix2  i = 0  **while** i < (size[1] - 1):  i += 1  result[i] = np.dot(matrix2,matrix1)  matrix1 = np.linalg.matrix\_power(matrix1, i+1)    result1 = [0] \* n  **for** i **in** range(n):  result1[i] = []    i = 0  **for** j **in** range(size[1]):  **for** k **in** range(len(result[0])):  result1[i] =result[j][k]  i += 1    resultend = np.array ([result1[0], result1[1], result1[2]])  **return** resultend      **def** check\_observability(matrix1,matrix2):  rank\_of\_matrix = np.linalg.matrix\_rank(observability\_of\_matrix(matrix1,matrix2))  size = np.shape(matrix2)  n = size[1]  **if** n == rank\_of\_matrix:  **return** "Матрица наблюдаема."  **else**:  **return** "Матрица не является наблюдаемой."      **def** controllability\_of\_matrix(matrix1,matrix2):  size = np.shape(matrix2)  n = size[0] \* size[1]  result = [0] \* n  **for** i **in** range (n):  result[i] = [0] \* size[0]    result[0] = matrix2  i = 0  **while** i < (size[0] - 1):  i += 1  result[i] = np.dot(matrix1,matrix2)  matrix1 = np.linalg.matrix\_power(matrix1, i + 1)    result1 = [0] \* size[0]  **for** i **in** range(size[0]):  result1[i] = []    **for** i **in** range(size[0]):  **for** j **in** range (size[0]):  result1[i] = np. concatenate( [result1[i], result[j][i]])    **return** result1      **def** check\_controllability(matrix1,matrix2):  rangMatrix = np.linalg.matrix\_rank(controllability\_of\_matrix(matrix1,matrix2))  size = np.shape(matrix2)  n = size[0]  **if** n == rangMatrix:  **return** "Матрица управляема."  **else**:  **return** "Матрица не является управляемой."      *#вывод результата вычислений на экран*  **print**("Первая матрица: **\n**", matrix1)  **print**("Вторая матрица: **\n**", matrix2)  **print**("Третья матрица: **\n**", matrix3)  start\_time = time.time()  **print**("Ранг матрицы равен:", np.linalg.matrix\_rank(matrix1))  **print**(check\_observability(matrix1, matrix3))  **print**(check\_controllability(matrix1, matrix2))  end\_time = time.time()  time = end\_time - start\_time  **print**("Время вычисления: " + str(time)) |

Сравнение методов:

На приведенном примере видно, что алгоритм, реализованный с помощью библиотеки NumPy, работает быстрее чем алгоритм, реализованный нами более, чем в два раза. Однако, в обоих случаях, затраченное время составляет менее 1 миллисекунды, что в условиях нашей работы не критично и разница не заметна.

Ссылка на GitHub: <https://github.com/nikotheflow/Python-labs>