Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»
(УНИВЕРСИТЕТ ИТМО)

Факультет «Систем управления и робототехники»

ОТЧЕТ
О ЛАБОРАТОРНОЙ РАБОТЕ №5

По дисциплине «Практическая линейная алгебра»
на тему: «Спектральная теория графов»

Студенты:
Комарова О. И. группа R3235
Охрименко А.Д. группа R3235
Крамаренко Ю.А. группа R3237
Янушанец Р. Д. группа R3280

Проверил:
Догадин Егор Витальевич, ассистент

г. Санкт-Петербург
2024

Содержание

1 Задание 1. Кластеризация социальной сети

1.1 Task. Создание графа.

В этом пункте мы написали код (на *WolframMatematicaLanguage*) для построения смежного графа, состоящего из 20 вершин.

Он разбит на 3-4 условных сообщества. Переменная *edges* задает связи между вершинами *V1*, ..., *V20*.

Функция *Graph* рисует красивый граф с заданными параметрами.

```
edges = {
   V1 <-> V2, V1 <-> V3, V1 <-> V4, V2 <-> V3, V2 <-> V5, V2 <-> V6,
   V3 <-> V4, V3 <-> V6, V3 <-> V7, V4 <-> V7, V5 <-> V6, V6 <-> V7,
   V8 <-> V9, V8 <-> V10, V8 <-> V11, V9 <-> V10, V9 <-> V12,
   V10 <-> V11, V10 <-> V12, V10 <-> V13, V11 <-> V13,
   V14 <-> V15, V14 <-> V16, V15 <-> V16, V15 <-> V17, V15 <-> V18,
   V16 <-> V18, V17 <-> V19, V18 <-> V20, V19 <-> V20,

   V6 <-> V8, V11 <-> V15, V12 <-> V18 };

g = Graph[edges, VertexLabels -> "Name",
  GraphLayout -> "SpringElectricalEmbedding", ImageSize -> Large,
  EdgeStyle -> {Thick}
  , VertexSize -> Large, VertexStyle -> Red]
```

Листинг 1: Код для задания графа

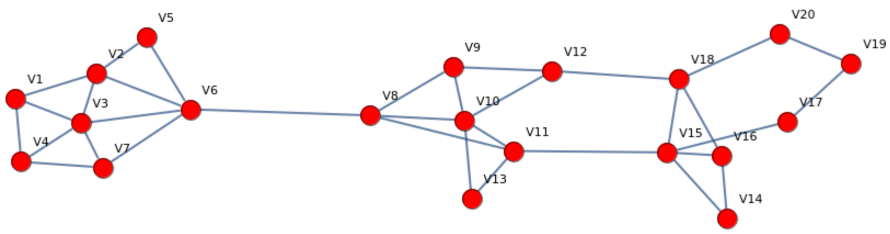

Рис. 1: Граф, который мы задумали

1.2 Task. Лапласиан неориентированного связного графа.

В этом задании мы воспользовались формулой из википедии для построения графа Лапласса:

$$L = D - A,$$

где D — матрица степеней, а A — матрица смежности графа.

```
adjacencyMatrix = AdjacencyMatrix[g];
degreeMatrix = DiagonalMatrix[Total[adjacencyMatrix, {2}]];
laplacianMatrix = MatrixForm[degreeMatrix - adjacencyMatrix]
```

Листинг 2: Код для Лапласиана графа

В результате нехитрых преобразований получилась следующая матрица 20 на 20:

Out[183]//MatrixForm=

$$\left(\begin{array}{cccccccccccccccccccc}
3 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 4 & -1 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & -1 & 5 & -1 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & -1 & 3 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 0 & 0 & 2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & -1 & 0 & -1 & 5 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & -1 & 0 & -1 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 & 4 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 3 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 5 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 & 0 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & -1 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 5 & -1 & -1 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 3 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 2 & 0 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & -1 & 0 & 4 & 0 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 2 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 2
\end{array}\right)$$

Рис. 2: Лапласиан графа

1.3 Task. Собственные числа и вектора Лапласиана.

В этом фрагменте мы с помощью кода нашли собственные значения и записали их в переменную *eigenvalues*.

Собственные вектора мы записали в переменную *eigenvectors*.

Поскольку вывод получился очень большим, мы предоставим только код. Но мы гарантируем, что все значения из множества $eigenvalues \geq 0$.

```
{eigenvalues , eigenvectors} = N[Eigensystem[laplacianMatrix]]
```

Листинг 3: Код для поиска собственных векторов и значений

1.4 Task. Выбор числа k желаемых компонент кластеризации графа.

Мы очень долго спорили и выбирали. Сошлись на числе *4*.

```
k = 4;
```

Листинг 4: Сложнейший выбор числа

1.5 Составление матрицы из векторов, соответствующих самым маленьким собственным числам.

В переменную *eigenpairs* мы поместили список, где каждая запись — это пара вида:

$$\{\text{собственное\_значение; собственный\_вектор}\}$$

Далее функцией *SortBy* мы отсортировали этот список по первому значению, то есть по собственному значению при помощи параметра *First*.

В переменную *smallestFourVectors* мы записали первые 4 собственных вектора из списка *sortedEigenpairs*. И вывели результат в матричном виде при помощи функции *MatrixForm*.

```
eigenpairs = Transpose[{eigenvalues, eigenvectors}];
sortedEigenpairs = SortBy[eigenpairs, First];
smallestFourVectors = Transpose[sortedEigenpairs[[;; k, 2]]];
V = MatrixForm[smallestFourVectors]
```

Листинг 5: Составление матрицы из векторов

Out[62]//MatrixForm=

$$\begin{pmatrix}
1. & -1.28317 & 0.385664 & -0.095979 \\
1. & -1.19313 & 0.267826 & -0.03695 \\
1. & -1.2185 & 0.307106 & -0.0621236 \\
1. & -1.2871 & 0.393665 & -0.103181 \\
1. & -1.15309 & 0.205266 & 0.0030081 \\
1. & -0.977567 & 0.0424338 & 0.0402807 \\
1. & -1.20838 & 0.295921 & -0.0593296 \\
1. & 0.000143157 & -0.884678 & 0.320839 \\
1. & 0.278574 & -1.12758 & 0.445569 \\
1. & 0.312782 & -1.1239 & 0.38596 \\
1. & 0.386766 & -0.897502 & 0.125129 \\
1. & 0.490066 & -0.823353 & 0.23214 \\
1. & 0.371606 & -1.33734 & 0.461571 \\
1. & 0.902245 & 0.309729 & -2.07346 \\
1. & 0.81709 & 0.194338 & -0.77956 \\
1. & 0.881386 & 0.273817 & -1.51634 \\
1. & 0.997785 & 1.00709 & 0.605146 \\
1. & 0.82126 & 0.183626 & -0.342345 \\
1. & 1.06124 & 1.32788 & 1.44963 \\
1. & 1. & 1. & 1.
\end{pmatrix}$$

Рис. 3: Матрица из собственных векторов

1.6 Task. Метод $k - means$.

В этом пункте мы применили метод $k - means$ к матрице из собственных векторов при помощи функции *FindClusters*.

Таким образом мы кластеризовали точки на 4 групки определенным алгоритмом.

```
clusters = FindClusters[smallestFourVectors, k]
```

Листинг 6: Применение метода $k - means$

1.7 Task. Кластеризация графа.

Первым делом, при помощи функции *VertexList* мы получили список (упорядоченный) вершин графа.

Потом мы вычисляем список *clusterLabels*, который содержит индексы кластеров, к которым принадлежит каждая вершина графа.

Затем определяем различные цвета.

```
vertices = VertexList[g];

```

```
clusterLabels =
  Table[With[{clusterIndex =
      Position[clusters,
       SelectFirst[clusters,
        MemberQ[#, smallestFourVectors[[i]]] &]]},
    If[clusterIndex === {}, Missing[],
     First[First[clusterIndex]]]], {i, Length[vertices]}];

vertexColors =
  Table[ColorData["Rainbow"][
    clusterLabels[[i]]/Max[clusterLabels]], {i, Length[vertices]}];
```

Листинг 7: Кластеризация

1.8 Task. Разноцветный граф. Результат кластеризации.

Передав в функцию *Graph* различные параметры, в том числе параметр *VertexStyle* мы определили цвет вершин и другие харктеристики графа.

```
coloredGraph =
 Graph[
  vertices,
  EdgeList[g],
  VertexStyle -> Thread[vertices -> vertexColors],
  VertexLabels -> "Name",
  VertexSize -> Large,
  EdgeStyle -> Thick,
  ImageSize -> Large
  ]
```

Листинг 8: Разноцветный граф

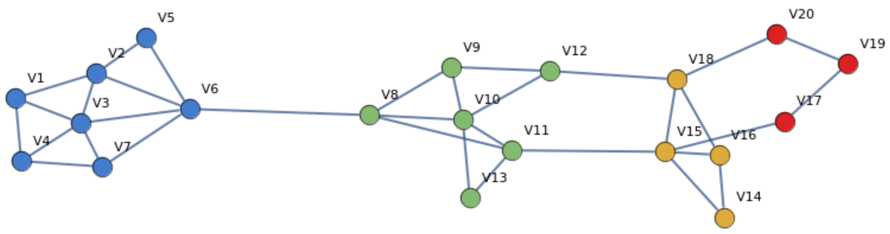

Рис. 4: Результат кластеризации графа

1.9 Task. Кластеризация для 5 разных значений k.

Код останется тем же, будет меняться только параметр k. Поэтому в этом пункте мы приведем только разноцветные картинки.

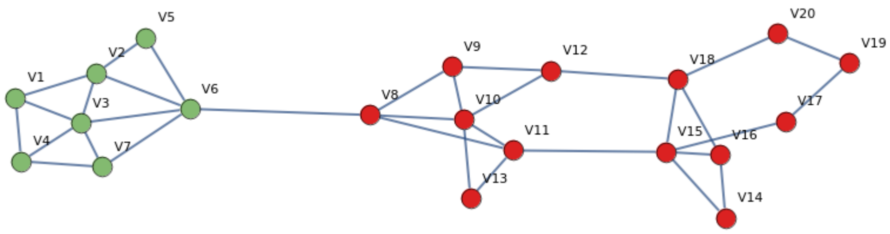

Рис. 5: Результат кластеризации. $k = 2$

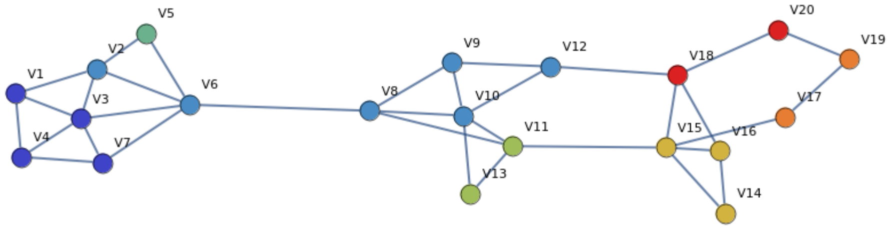

Рис. 6: Результат кластеризации. $k = 7$

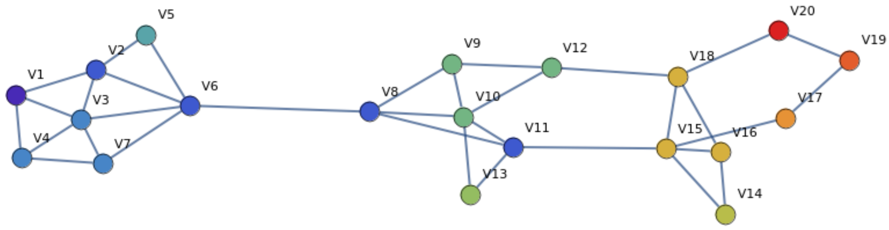

Рис. 7: Результат кластеризации. $k = 11$

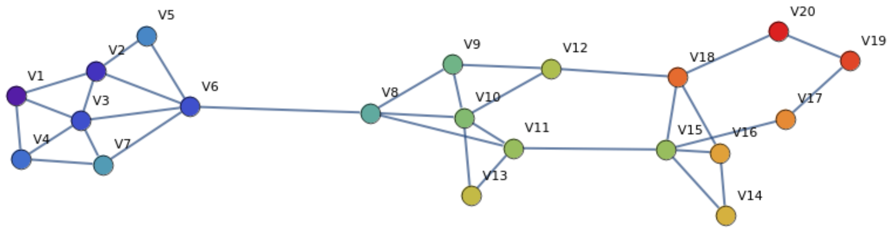

Рис. 8: Результат кластеризации. $k = 18$

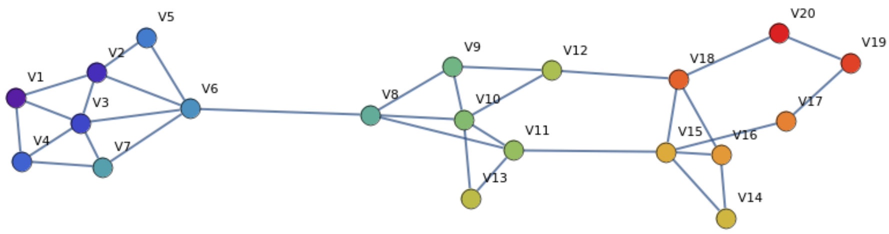

Рис. 9: Результат кластеризации. $k = 20$

1.10 Task. Почему это работает?

Кластеризация работает благодаря следующим причинам:

- **Матрица Лапласиана** описывает, как вершины графа связаны друг с другом.
- **Собственные векторы** создают новое представление вершин, где близкие вершины оказываются рядом.
- **Алгоритм k-means** группирует вершины на k кластеров в этом новом представлении.
- **Изменение k**:

  - При $k = 1$ весь граф — это один кластер.
  - При $k = 2$ граф делится на две крупные части.
  - При большем k крупные группы делятся на более мелкие.

- Это работает, потому что матрица Лапласиана точно показывает, какие вершины сильнее связаны между собой.

2 Задание 2. Google PageRank алгоритм

2.1 Task. Ориентированный связный граф.

В переменную *edges* мы поместили связи между вершинами, обозначив их стрелочками. Функцией *Graph* мы нарисовали получившийся граф.

```
edges = {v1 -> v2, v2 -> v3, v3 -> v4, v4 -> v1, v5 -> v6, v6 -> v7, v7 ->
   v8, v8 -> v5, v9 -> v10, v10 -> v11, v11 -> v12, v12 -> v9, v13 -> v14,
   v10 -> v13, v6 -> v10, v13 -> v3, v10 -> v14, v3 -> v14, v4 -> v14, v5 ->
    v14, v8 -> v14, v1 -> v14, v11 -> v14};

g = Graph[edges, VertexLabels -> "Name", DirectedEdges -> True,
  ImageSize -> Large, EdgeStyle -> Thick, VertexSize -> Large,
  VertexStyle -> Red]
```

Листинг 9: Код для графа

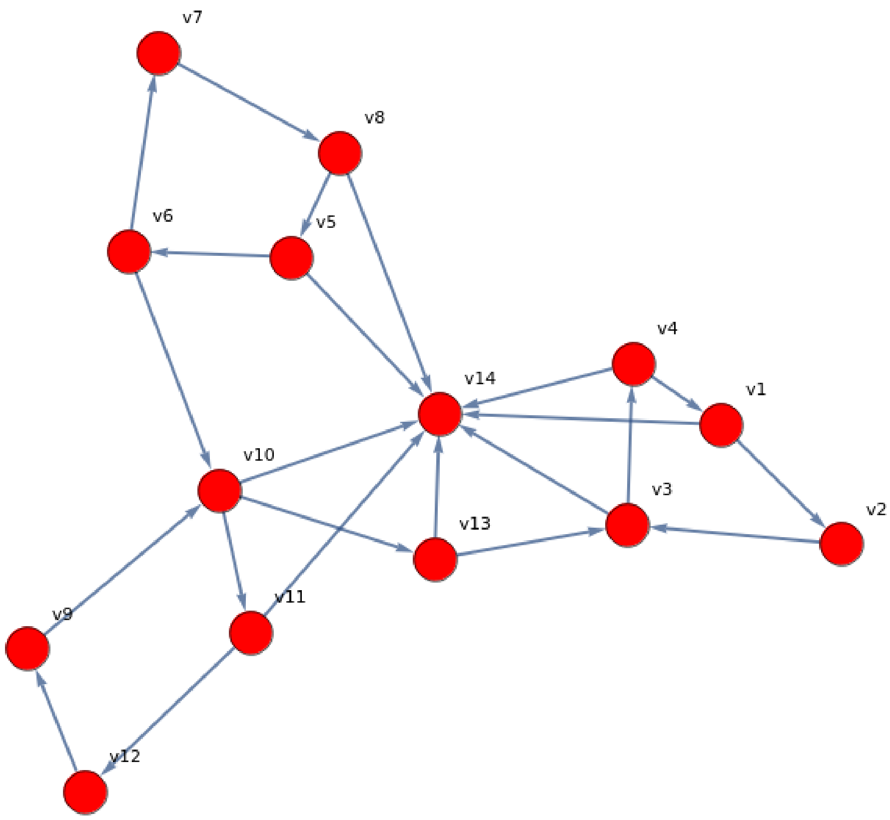

Рис. 10: Придуманный граф.

2.2 Task. Составление матрицы M.

В этом пункте мы составляли матрицу M следущим образом

$$M = \begin{bmatrix} m_{11} & m_{12} & \dots & m_{1n} \\ m_{21} & m_{22} & \dots & m_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ m_{n1} & m_{n2} & \dots & m_{nn} \end{bmatrix},$$

где

$$m_{ij} = \frac{\text{число стрелочек, выходящих из } j\text{-й вершины и входящих в } i\text{-ю вершину}}{\text{общее число стрелочек, выходящих из } j\text{-й вершины}}.$$

Для начала мы поместили в переменную *vertices* все вершины графа. Затем посчитали их количество.

Далее составили матрицу смежности функцией *Adjacency*.

Потом получили вектор исходящих степеней вершин графа и записали это в переменную *outDegree*.

Потом циклом *Table* нарисовали матрицу и функцией *MatrixForm* вывели ее в матричном и нормализованном виде.

```
vertices = VertexList[g];
n = Length[vertices];

adjMatrix = AdjacencyMatrix[g];
outDegree = Total[adjMatrix, {2}];

mMatrix =
  Table[If[outDegree[[j]] == 0, 1/n,
    adjMatrix[[i, j]]/outDegree[[j]]], {i, n}, {j, n}];
mMatrix = Map[#/Total[#] &, mMatrix];
MatrixForm[mMatrix]
```

Листинг 10: Код для матрицы M

Out[122]//MatrixForm=

$$
\begin{pmatrix}
0 & \frac{14}{15} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{15} \\
0 & 0 & \frac{7}{8} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{8} \\
0 & 0 & 0 & \frac{7}{8} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{8} \\
\frac{7}{8} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{8} \\
0 & 0 & 0 & 0 & 0 & \frac{7}{8} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{8} \\
0 & 0 & 0 & 0 & 0 & 0 & \frac{42}{59} & 0 & 0 & \frac{14}{59} & 0 & 0 & 0 & \frac{3}{59} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{7}{8} & 0 & 0 & 0 & 0 & 0 & \frac{1}{8} \\
0 & 0 & 0 & 0 & \frac{7}{8} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{8} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{14}{17} & 0 & 0 & 0 & \frac{3}{17} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{7}{15} & 0 & \frac{7}{15} & \frac{1}{15} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{14}{15} & 0 & \frac{1}{15} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{14}{15} & 0 & 0 & 0 & 0 & \frac{1}{15} \\
0 & 0 & \frac{7}{8} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{8} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}
$$

Рис. 11: Матрица M.

2.3 Task. Собственный вектор матрицы M, соответствующий наибольшему собственному числу.

В этом пункте мы для начала определили собственные значения и вектора при помощи функции *Eigensystem*. Далее мы поместили их в список, где каждая запись — это пара вида:

$$\{\text{собственное\_значение; собственный\_вектор}\}$$

Затем мы их отсортировали обратной сортировкой по первому значению из списка(то есть по собственному числу).

И непосредственно вытащили вектор из отсортированного массива(взяли самый первый вектор).

И вывели результат в матричном виде функцией *MatrixForm*.

```
{eigenvalues, eigenvectors} = N[Eigensystem[mMatrix]];

eigenpairs = Transpose[{eigenvalues, eigenvectors}];
sortedEigenpairs = ReverseSort[eigenpairs, First];

biggerVectors = Transpose[sortedEigenpairs[[;; 1, 2]]];
vector = MatrixForm[biggerVectors]
```

Листинг 11: Код для осбственного вектора

Out[127]//MatrixForm=

$$\begin{pmatrix} 1. \\ 1. \\ 1. \\ 1. \\ 1. \\ 1. \\ 1. \\ 1. \\ 1. \\ 1. \\ 1. \\ 1. \\ 1. \\ 1. \end{pmatrix}$$

Рис. 12: Собственный вектор

2.4 Task. *GooglePageRank* алгоритм.

PageRankCentrality вычисляет значения *PageRank* для вершин графа g, оценивая их значимость.

VertexList создаёт список вершин, а *AssociationThread* связывает их с соответствующими значениями PageRank.

MatrixForm отображает результаты в удобной таблице.

```
ranks = PageRankCentrality[g];
vertexRanks = AssociationThread[VertexList[g] -> ranks];
MatrixForm[vertexRanks // Normal]
```

Листинг 12: *GooglePageRank*

Out[112]//MatrixForm=

$$\begin{pmatrix} v1 \to 0.050936 \\ v2 \to 0.0463998 \\ v3 \to 0.0867231 \\ v4 \to 0.0616093 \\ v5 \to 0.0513651 \\ v6 \to 0.0465822 \\ v7 \to 0.0445494 \\ v8 \to 0.062619 \\ v9 \to 0.0649428 \\ v10 \to 0.0997508 \\ v11 \to 0.0530147 \\ v12 \to 0.0472833 \\ v13 \to 0.0530147 \\ v14 \to 0.23121 \end{pmatrix}$$

Рис. 13: Результат работы *GooglePageRank* алгоритма

2.5 Task. Почему это работает?

- Матрица M — это стохастическая матрица переходов, где элемент m_{ij} показывает вероятность перехода с вершины j на вершину i. Она строится из структуры графа ссылок, причём каждая строка нормализована, чтобы сумма вероятностей равнялась 1.

- Собственный вектор, соответствующий наибольшему собственному числу (1), описывает устойчивое состояние системы. Он показывает вероятности посещения вершин при бесконечном перемещении.
- Параметр d (обычно 0.85) вводит вероятность случайного перехода на произвольную вершину. Это помогает избежать зацикливания и учесть изолированные страницы.
- *PageRank* основан на марковских процессах: состояние системы определяется только текущим состоянием. Матрица M моделирует марковскую цепь, а собственный вектор — стационарное распределение вероятностей.