Мухамадиев Владимир

Задание 8

Загрузка и предварительная обработка

In[1]:=
```
g1 = ConnectedGraphComponents[SimpleGraph[UndirectedGraph[
    Import[FileNameJoin[{NotebookDirectory[], "cat.graphml"}]]]][[1]];
```
связные граф-компоненты | простой граф | ненаправленный граф | импорт | соединить пути | директория файла блокнота

In[2]:=
```
g2 = ConnectedGraphComponents[SimpleGraph[UndirectedGraph[
    Import[FileNameJoin[{NotebookDirectory[], "c-elegans.graphml"}]]]][[1]];
```
связные граф-компоненты | простой граф | ненаправленный граф | импорт | соединить пути | директория файла блокнота

In[3]:=
```
g3 = ConnectedGraphComponents[SimpleGraph[UndirectedGraph[
    Import[FileNameJoin[{NotebookDirectory[], "drosophila.graphml"}]]]][[1]];
```
связные граф-компоненты | простой граф | ненаправленный граф | импорт | соединить пути | директория файла блокнота

In[4]:=
```
g4 = ConnectedGraphComponents[SimpleGraph[UndirectedGraph[
    Import[FileNameJoin[{NotebookDirectory[], "macaque.graphml"}]]]][[1]];
```
связные граф-компоненты | простой граф | ненаправленный граф | импорт | соединить пути | директория файла блокнота

In[5]:=
```
g5 = ConnectedGraphComponents[SimpleGraph[UndirectedGraph[
    Import[FileNameJoin[{NotebookDirectory[], "mouse.graphml"}]]]][[1]];
```
связные граф-компоненты | простой граф | ненаправленный граф | импорт | соединить пути | директория файла блокнота

In[6]:=
```
g6 = ConnectedGraphComponents[SimpleGraph[UndirectedGraph[
    Import[FileNameJoin[{NotebookDirectory[], "rat.graphml"}]]]][[1]];
```
связные граф-компоненты | простой граф | ненаправленный граф | импорт | соединить пути | директория файла блокнота

In[7]:=
```
sre = Import[FileNameJoin[{NotebookDirectory[], "sre.m"}]];
```
импорт | соединить пути | директория файла блокнота

In[8]:=
```
re = Import[FileNameJoin[{NotebookDirectory[], "re.m"}]];
```
импорт | соединить пути | директория файла блокнота

1. Random walk normalized Laplacian

In[9]:=
```
RWNL[graph_] := Inverse[DiagonalMatrix[VertexDegree[graph]]].KirchhoffMatrix[graph]
```
обратна… | диагональная ма… | степень вершины | матрица Кирхгофа

In[10]:=
```
HPDF[graph_, r_] := Module[{ei = Eigenvalues[N[RWNL[graph]]]},
                           программный··· собственные··· численное приближение
  {Range[0, 2, 1/VertexCount[graph]], Flatten[Append[MovingAverage[
   диапазон                           уплостить добав··· скользящее среднее
    Table[Evaluate[PDF[HistogramDistribution[ei, VertexCount[graph]], i]],
     табл··· вычислить пл··· распределение по гистограмме число вершин
      {i, 0, 2, 1/VertexCount[graph]}], r], ConstantArray[0, r - 1]]}^T]
                                        постоянный массив
```

In[11]:=
```
g1d = HPDF[g1, Round[VertexCount[g1]/10]];
      округлить
```

In[12]:=
```
g2d = HPDF[g2, Round[VertexCount[g2]/10]];
      округлить
```

In[13]:=
```
g3d = HPDF[g3, Round[VertexCount[g3]/10]];
      округлить
```

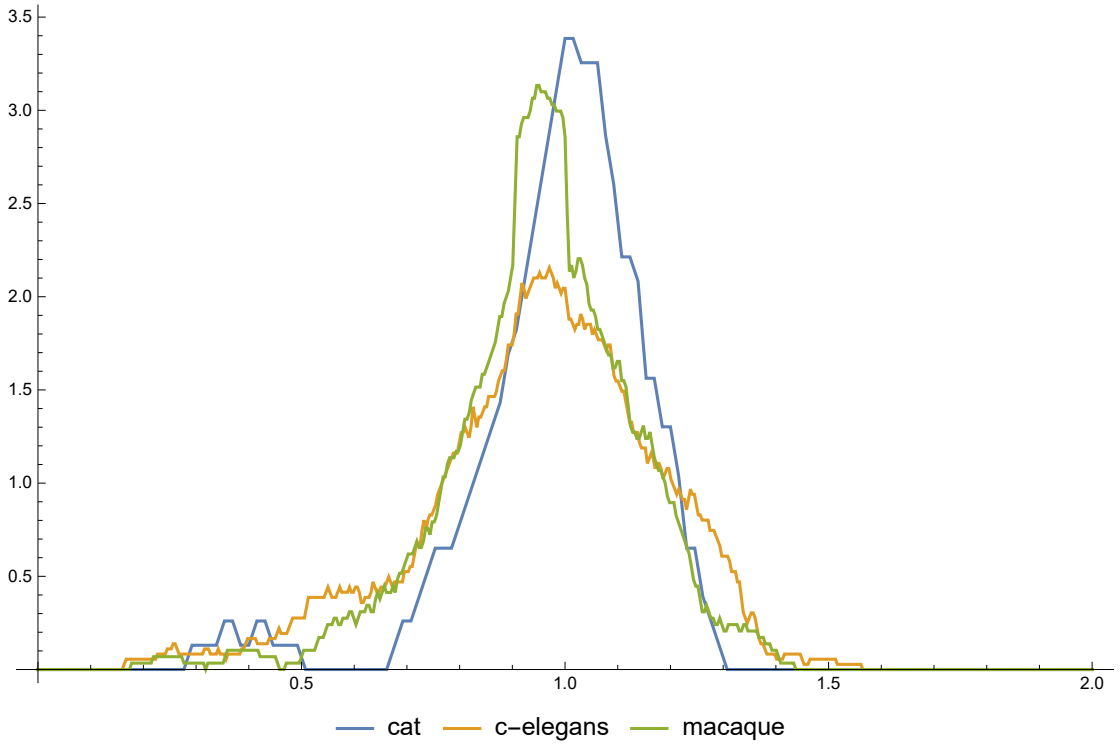
In[14]:=
```
g4d = HPDF[g4, Round[VertexCount[g4]/10]];
      округлить
```

In[15]:=
```
g5d = HPDF[g5, Round[VertexCount[g5]/10]];
      округлить
```

In[16]:=
```
g6d = HPDF[g6, Round[VertexCount[g6]/10]];
      округлить
```

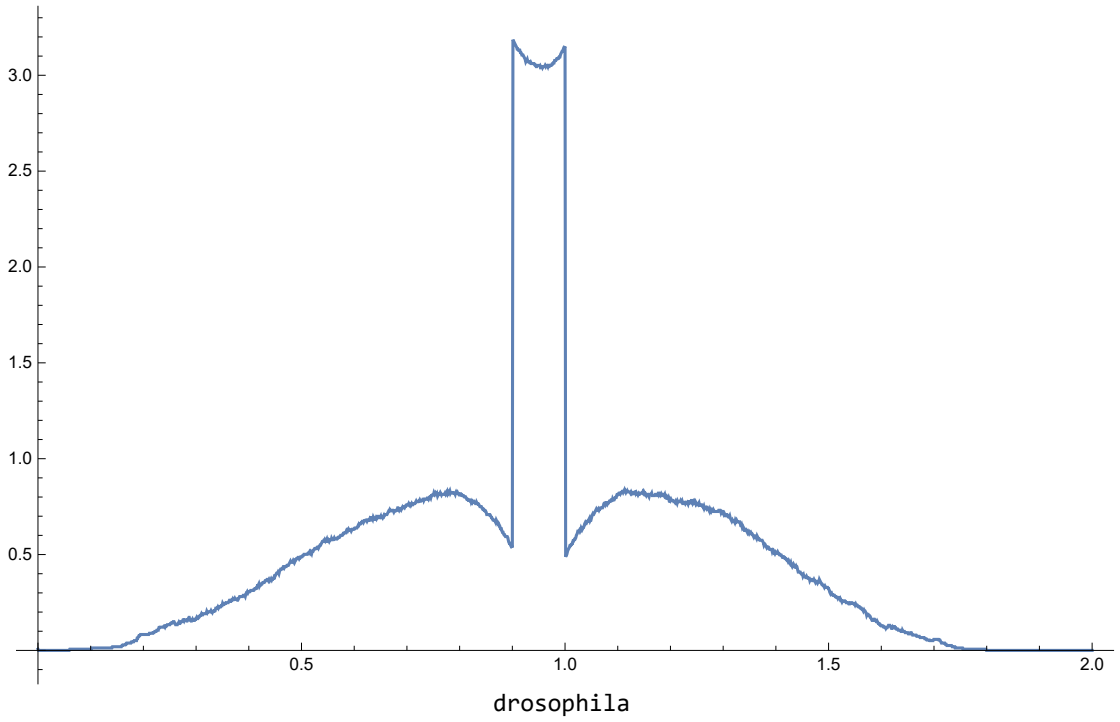
In[17]:=
```
ListLinePlot[{g1d, g2d, g4d}, PlotRange → All,
 PlotLegends → Placed[{"cat", "c-elegans", "macaque"}, Below], ImageSize → Large]
```

Out[17]=



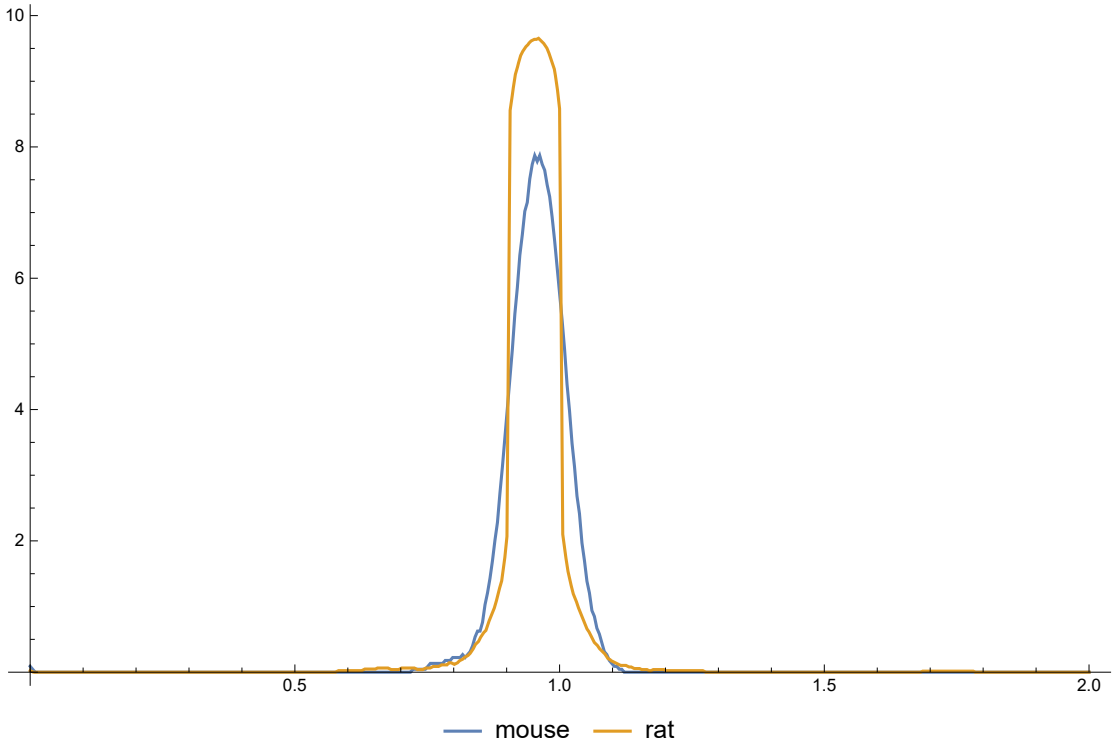
In[18]:=
```
ListLinePlot[g3d, PlotRange → All,
 PlotLegends → Placed["drosophila", Below], ImageSize → Large]
```

Out[18]=



In[19]:=
```
ListLinePlot[{g5d, g6d}, PlotRange → All,
 PlotLegends → Placed[{"mouse", "rat"}, Below], ImageSize → Large]
```
линейный график данных | отображаем… | всё | легенды графика | расположен | снизу | размер изоб… | крупный

Out[19]=



2. Устойчивость коннектомов

Функция которая приводит список ребер к виду, где в каждом ребре $v_1 \rightarrowtail v_2$, $v_1 \leq v_2$

In[20]:=
```
EdgeSort[edgelist_] := Module[{out = edgelist},
 Do[If[edgelist[[i, 1]] > edgelist[[i, 2]], out[[i]] = edgelist[[i, 2]] ⇀ edgelist[[i, 1]]],
  {i, 1, Length[edgelist]}];
 out]
```
программный модуль | … | условный оператор | длина

Рандомизация графа без сохранения степеней вершин

In[21]:=
```
GraphSimpleRandomization[graph_, n_] :=
 Module[{el = EdgeSort[EdgeList[graph]], vl = VertexList[graph], temp = {}},
  Do[temp = RandomChoice[el];
   While[VertexDegree[graph, temp[[1]]] == 1 || VertexDegree[graph, temp[[2]]] == 1,
    temp = RandomChoice[el]];
   el = DeleteCases[el, temp];
   temp = RandomSample[vl, 2];
   temp = EdgeSort[{temp[[1]] ⟷ temp[[2]]}][[1]];
   While[ContainsAll[el, {temp}] == True, temp = RandomSample[vl, 2];
    temp = EdgeSort[{temp[[1]] ⟷ temp[[2]]}][[1]]];
   AppendTo[el, temp], {i, 1, n}];
  Graph[el]]
```

Рандомизация графа с сохранением степеней вершин

In[22]:=
```
GraphRandomization[g_, n_] :=
 Module[{el = EdgeSort[EdgeList[g]], eltemp = {}, vltemp = {}, temp = {{}, {}}, k = 0},
  Do[temp[[1]] = RandomChoice[el](*Выбираем первое ребро/петлю*);
   eltemp = DeleteCases[el, temp[[1]]](*Создаем временную выборку из ребер/петель
    и удалаем из нее все мультиребра/мультипетли для выбранного ребра/петли*);
   If[temp[[1, 1]] == temp[[1, 2]], (*Алгоритм выбрал петлю*)
    eltemp = DeleteCases[eltemp, v_ ⟷ v_](*удаляем из временной выбрки все петли*);
    vltemp = Flatten[{Cases[eltemp, temp[[1, 1]] ⟷ _], Cases[eltemp, _ ⟷ temp[[1, 1]]]}];
    vltemp = DeleteDuplicates[Flatten[{vltemp[[All, 1]], vltemp[[All, 2]]}]]
    (*Список вершин с которыми граничит выбранная петля*);
    Do[eltemp = DeleteCases[eltemp, vltemp[[j]] ⟷ _];
     eltemp = DeleteCases[eltemp, _ ⟷ vltemp[[j]]], {j, 1, Length[vltemp]}](*Удаляем из
     временной выборки все ребра/петли, вершины которых связаны с данной петлей*);
    If[eltemp == {}, i++;
     Goto[end](*Алгоритм выбрал петлю которую нельзя изменить,
     так как до любой вершины из нее можно добраться в два шага. Переходим в
```

```
    конец цикла и считаем что на этом шаге ничего не поменялось*), temp[[2]] =
    RandomChoice[eltemp](*Выбираем второе ребро*)], (*Алгоритм выбрал ребро*)
   случайный выбор
  vltemp = Flatten[{Cases[eltemp, temp[[1, 1]] ⟼ _], Cases[eltemp, _ ⟼ temp[[1, 1]]],
           уплостить  случаи по образцу                случаи по образцу
     Cases[eltemp, temp[[1, 2]] ⟼ _], Cases[eltemp, _ ⟼ temp[[1, 2]]}];
    случаи по образцу                 случаи по образцу
  vltemp = DeleteDuplicates[Flatten[{vltemp[[All, 1]], vltemp[[All, 2]]}]]
           удалить дубликаты  уплостить  всё  всё
  (*Список вершин с которыми граничит выбранное ребро*);
  Do[eltemp = DeleteCases[eltemp, vltemp[[j]] ⟼ _];
  оператор ци…  удалить случаи по образцу
    eltemp = DeleteCases[eltemp, _ ⟼ vltemp[[j]]], {j, 1, Length[vltemp]}](*Удаляем из
             удалить случаи по образцу                 длина
    временной выборки все ребра/петли, вершины которых связаны с данным ребром*);
  If[eltemp == {}, i++;
  условный оператор
    Goto[end](*Алгоритм выбрал ребро которое нельзя изменить,
    перейти
    так как до любой вершины из него можно добраться в два шага. Переходим
     в конец цикла и считаем что на этом шаге ничего не поменялось*),
    temp[[2]] = RandomChoice[eltemp](*Выбираем второе ребро/петлю*)]];
                случайный выбор
 k = 1;
 While[(el[[k]] === temp[[1]]) == False, k++];
 цикл–пока                  ложь
 el = Delete[el, k];
      удалить элемент
 k = 1;
 While[(el[[k]] === temp[[2]]) == False, k++];
 цикл–пока                  ложь
 el = Delete[el, k](*Удаляем из списка ребер выбранные*);
      удалить элемент
 If[RandomInteger[{1, 2}] == 1(*Случайно выбираем как переключить ребра и добавляем
 …  случайное целое число
   новые к списку ребер*), AppendTo[el, EdgeSort[{temp[[1, 1]] ⟼ temp[[2, 1]]}][[1]]];
                           добавить в конец к
  AppendTo[el, EdgeSort[{temp[[1, 2]] ⟼ temp[[2, 2]]}][[1]]],
  добавить в конец к
  AppendTo[el, EdgeSort[{temp[[1, 1]] ⟼ temp[[2, 2]]}][[1]]];
  добавить в конец к
  AppendTo[el, EdgeSort[{temp[[1, 2]] ⟼ temp[[2, 1]]}][[1]]]];
  добавить в конец к
 Label[end], {i, 1, n}];
 отметка
Graph[el]]
граф
```

Эволюция расстояния между спектрами графа и графа после рандомизации

Без сохранения степеней вершин

In[23]:=
```
SRE[graph_, t_, Δt_] := Module[{temp = graph,
                        программный модуль
  ei0 = Eigenvalues[N[RWNL[graph]]], out = ConstantArray[{0, 0}, Round[t/Δt] + 1]},
        собственные··· численное приближение  постоянный массив  округлить
 Do[temp = GraphSimpleRandomization[temp, Round[Δt]];
 оператор цикла                    округлить
  out[[i + 1]] = {i Round[Δt], EuclideanDistance[ei0, Eigenvalues[N[RWNL[temp]]]]},
                  округлить    эвклидово расстояние  собственные··· численное приближение
  {i, 1, Round[t/Δt]}];
          округлить

  out]
```

In[24]:=
```
g1sre = sre[[1]];
(*g1sre=SRE[g1,2EdgeCount[g1], EdgeCount[g1]/100 ]^T;
                  число рёбер
g1sre[[1]]=N[g1sre[[1]]/EdgeCount[g1]];
          численное приближение
g1sre=g1sre^T;*)
```

In[25]:=
```
g2sre = sre[[2]];
(*g2sre=SRE[g2,2EdgeCount[g2], EdgeCount[g2]/100 ]^T;
                  число рёбер
g2sre[[1]]=N[g2sre[[1]]/EdgeCount[g2]];
          численное приближение
g2sre=g2sre^T;*)
```

In[26]:=
```
g3sre = sre[[3]];
(*g3sre=SRE[g3,2EdgeCount[g3], EdgeCount[g3]/100 ]^T;
                  число рёбер
g3sre[[1]]=N[g3sre[[1]]/EdgeCount[g3]];
          численное приближение
g3sre=g3sre^T;*)
```

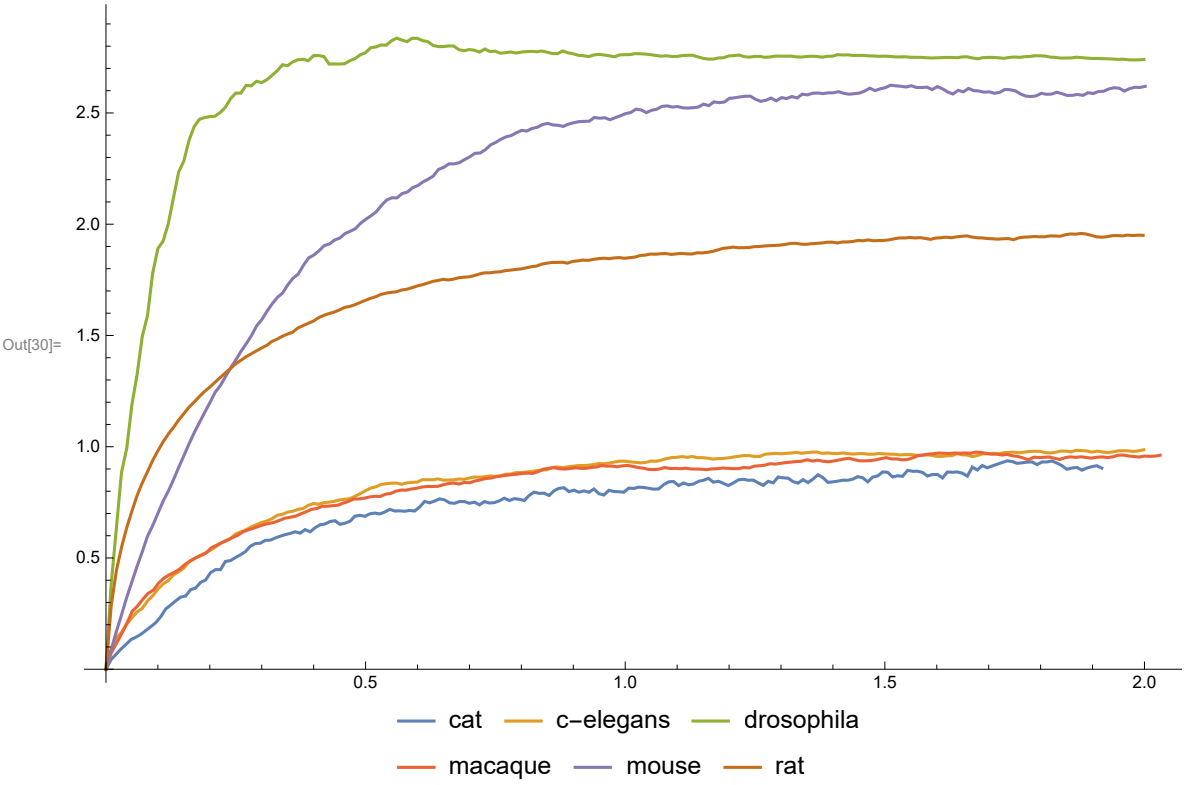
In[27]:=
```
g4sre = sre[[4]];
(*g4sre=SRE[g4,2EdgeCount[g4], EdgeCount[g4]/100 ]^T;
                  число рёбер
g4sre[[1]]=N[g4sre[[1]]/EdgeCount[g4]];
          численное приближение
g4sre=g4sre^T;*)
```

In[28]:=
```
g5sre = sre[[5]];
(*g5sre=SRE[g5,2EdgeCount[g5], EdgeCount[g5]/100 ]^T;
                  число рёбер
g5sre[[1]]=N[g5sre[[1]]/EdgeCount[g5]];
          численное приближение
g5sre=g5sre^T;*)
```

In[29]:=
```
g6sre = sre[[6]];
(*g6sre=SRE[g6,2EdgeCount[g6], EdgeCount[g6]/100]^T;
                  число рёбер
g6sre[[1]]=N[g6sre[[1]]/EdgeCount[g6]];
             численное приближение
g6sre=g6sre^T;*)
```

In[30]:=
```
ListLinePlot[{g1sre, g2sre, g3sre, g4sre, g5sre, g6sre},
 линейный график данных
 PlotLegends → Placed[{"cat", "c-elegans", "drosophila", "macaque", "mouse", "rat"},
  легенды графика  расположен
  Below], ImageSize → Large]
  снизу  размер изоб···  крупный
```

Out[30]=



С сохранением степеней вершин

In[31]:=
```
RE[graph_, t_, Δt_] := Module[{temp = graph,
                               программный модуль
  ei0 = Eigenvalues[N[RWNL[graph]]], out = ConstantArray[{0, 0}, Round[t/Δt] + 1]},
        собственные···  численное приближение  постоянный массив  округлить
 Do[temp = GraphRandomization[temp, Round[Δt]];
 оператор цикла                    округлить
  out[[i + 1]] = {i Round[Δt], EuclideanDistance[ei0, Eigenvalues[N[RWNL[temp]]]]},
                  округлить  эвклидово расстояние  собственные···  численное приближение
  {i, 1, Round[t/Δt]}];
          округлить

 out]
```

In[32]:=
```
g1re = re[[1]];
(*g1re=RE[g1,2EdgeCount[g1], EdgeCount[g1]/100 ]^T;
                  число рёбер
g1re[[1]]=N[ g1re[[1]]/EdgeCount[g1] ];
          численное приближение
g1re=g1re^T;*)
```

In[33]:=
```
g2re = re[[2]];
(*g2re=RE[g2,2EdgeCount[g2], EdgeCount[g2]/100 ]^T;
                  число рёбер
g2re[[1]]=N[ g2re[[1]]/EdgeCount[g2] ];
          численное приближение
g2re=g2re^T;*)
```

In[34]:=
```
g3re = re[[3]];
(*g3re=RE[g3,2EdgeCount[g3], EdgeCount[g3]/100 ]^T;
                  число рёбер
g3re[[1]]=N[ g3re[[1]]/EdgeCount[g3] ];
          численное приближение
g3re=g3re^T;*)
```

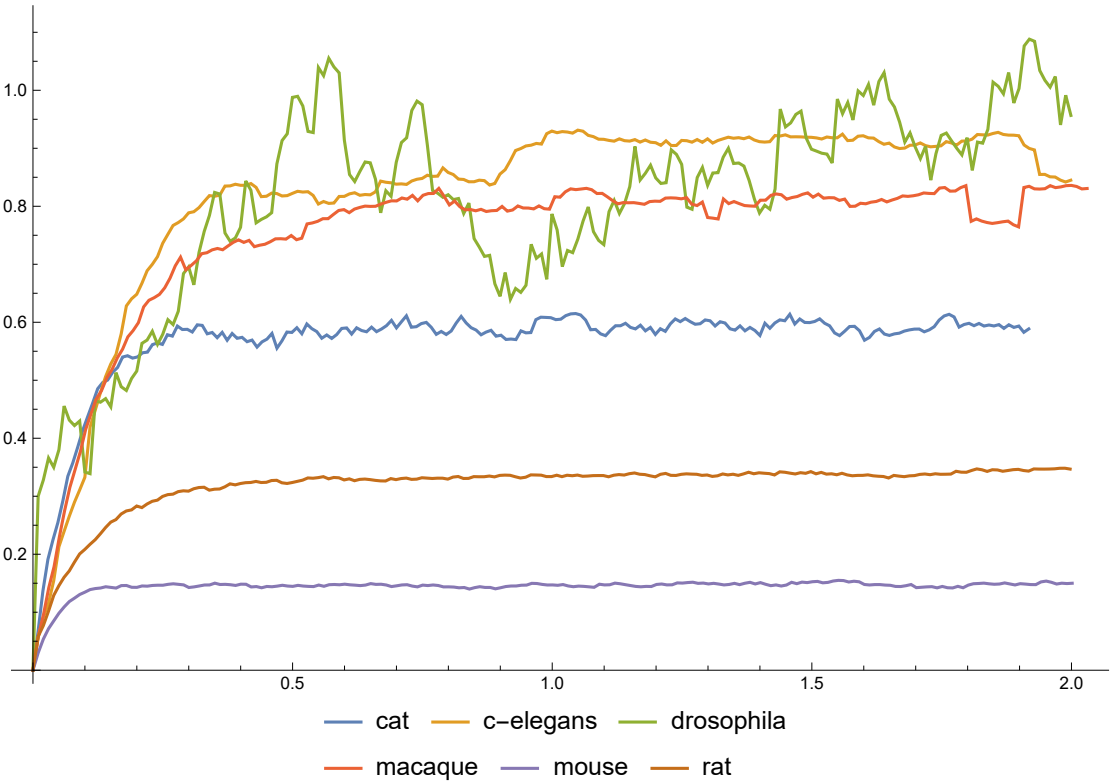
In[35]:=
```
g4re = re[[4]];
(*g4re=RE[g4,2EdgeCount[g4], EdgeCount[g4]/100 ]^T;
                  число рёбер
g4re[[1]]=N[ g4re[[1]]/EdgeCount[g4] ];
          численное приближение
g4re=g4re^T;*)
```

In[36]:=
```
g5re = re[[5]];
(*g5re=RE[g5,2EdgeCount[g5], EdgeCount[g5]/100 ]^T;
                  число рёбер
g5re[[1]]=N[ g5re[[1]]/EdgeCount[g5] ];
          численное приближение
g5re=g5re^T;*)
```

In[37]:=
```
g6re = re[[6]];
(*g6re=RE[g6,2EdgeCount[g6], EdgeCount[g6]/100 ]^T;
                  число рёбер
g6re[[1]]=N[ g6re[[1]]/EdgeCount[g6] ];
          численное приближение
g6re=g6re^T;*)
```

In[38]:=
```
ListLinePlot[{g1re, g2re, g3re, g4re, g5re, g6re},
 PlotLegends → Placed[{"cat", "c-elegans", "drosophila", "macaque", "mouse", "rat"},
   Below], ImageSize → Large]
```

Out[38]=



3. Схожесть спектров различных организмов

Распределение по гистограмме собственных значений RWNL

In[39]:=
```
RWNLHD[graph_, Δ_] :=
 Table[Evaluate[PDF[HistogramDistribution[Eigenvalues[N[RWNL[graph]]],
     VertexCount[graph]], i]], {i, 0, 2, Δ}]
```

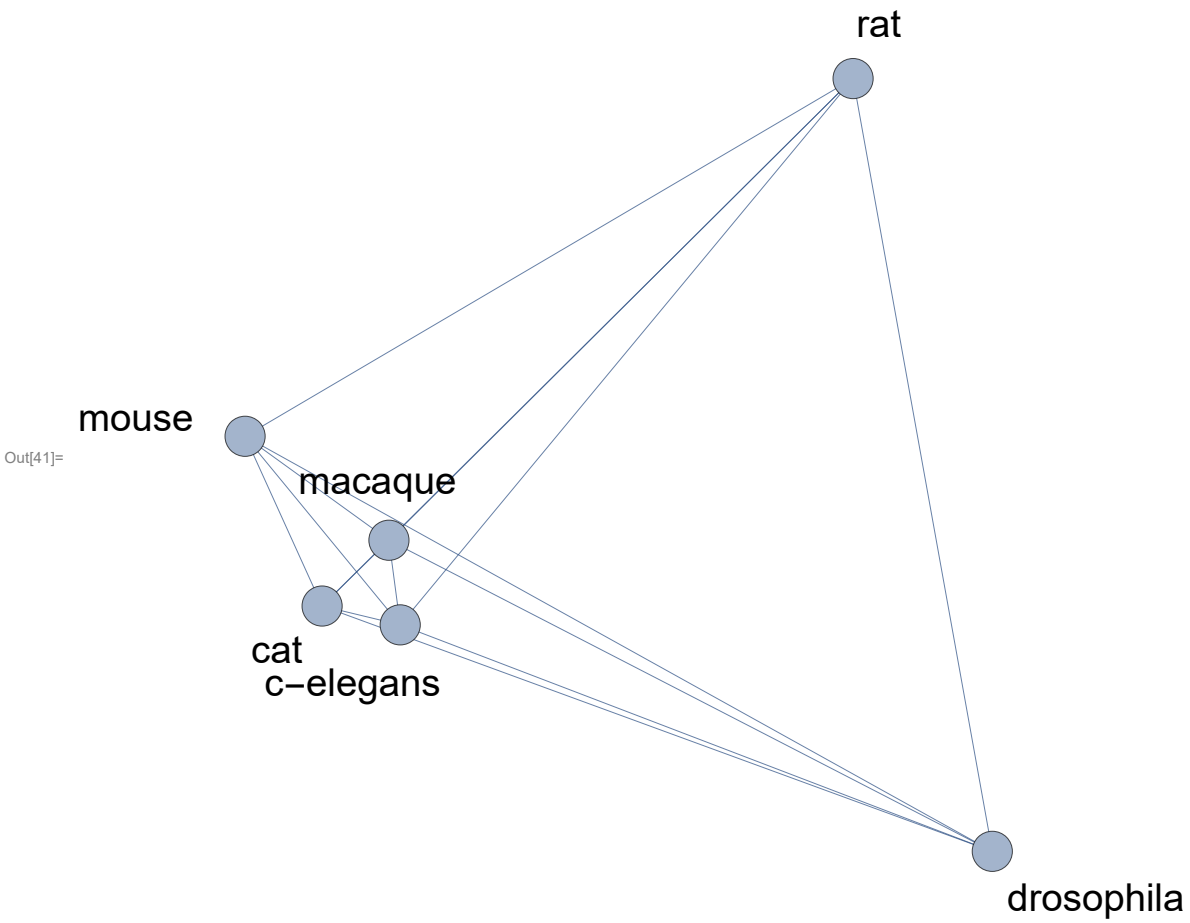
In[40]:=
```
dist = {RWNLHD[g1, 0.001], RWNLHD[g2, 0.001], RWNLHD[g3, 0.001],
   RWNLHD[g4, 0.001], RWNLHD[g5, 0.001], RWNLHD[g6, 0.001]};
```

Граф расстояний между распределениями

Евклидово расстояние

In[41]:=
```
CompleteGraph[6, EdgeWeight →
 полный граф   вес ребра
  Flatten[Table[EuclideanDistance[dist[[i]], dist[[j]]], {i, 1, 5}, {j, i + 1, 6}]],
  уплостить  табл···  эвклидово расстояние
 GraphLayout → {"VertexLayout" → {"SpringElectricalEmbedding", "EdgeWeighted" → True}},
  укладка графа                                                                 истина
 VertexLabels → {1 → Style["cat", 20], 2 → Style["c-elegans", 20],
  метки для вершин   стиль                  стиль
   3 → Style["drosophila", 20], 4 → Style["macaque", 20], 5 → Style["mouse", 20],
     стиль                        стиль                      стиль
   6 → Style["rat", 20]}, VertexSize → 0.5, ImageSize → Large]
     стиль                 размер вершины   размер изоб···  крупный
```

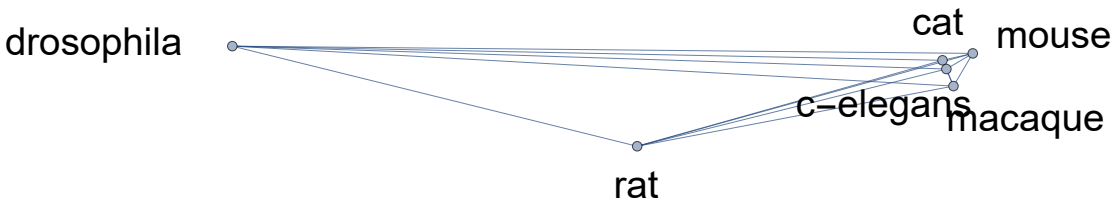
Out[41]=



Расстояние Чебышева

In[42]:=
```
CompleteGraph[6, EdgeWeight →
  Flatten[Table[ChessboardDistance[dist[[i]], dist[[j]]], {i, 1, 5}, {j, i + 1, 6}]],
 GraphLayout → {"VertexLayout" → {"SpringElectricalEmbedding", "EdgeWeighted" → True}},
 VertexLabels → {1 → Style["cat", 20], 2 → Style["c-elegans", 20],
   3 → Style["drosophila", 20], 4 → Style["macaque", 20], 5 → Style["mouse", 20],
   6 → Style["rat", 20]}, VertexSize → 1, ImageSize → Large]
```

Out[42]=



Манхэттенское расстояние

In[43]:=
```
CompleteGraph[6, EdgeWeight →
  Flatten[Table[ManhattanDistance[dist[[i]], dist[[j]]], {i, 1, 5}, {j, i + 1, 6}]],
 GraphLayout → {"VertexLayout" → {"SpringElectricalEmbedding", "EdgeWeighted" → True}},
 VertexLabels → {1 → Style["cat", 20], 2 → Style["c-elegans", 20],
   3 → Style["drosophila", 20], 4 → Style["macaque", 20], 5 → Style["mouse", 20],
   6 → Style["rat", 20]}, VertexSize → 0.25, ImageSize → Large]
```

Out[43]=