Wstęp do bioinformatyki

Nr ćwiczenia: 1

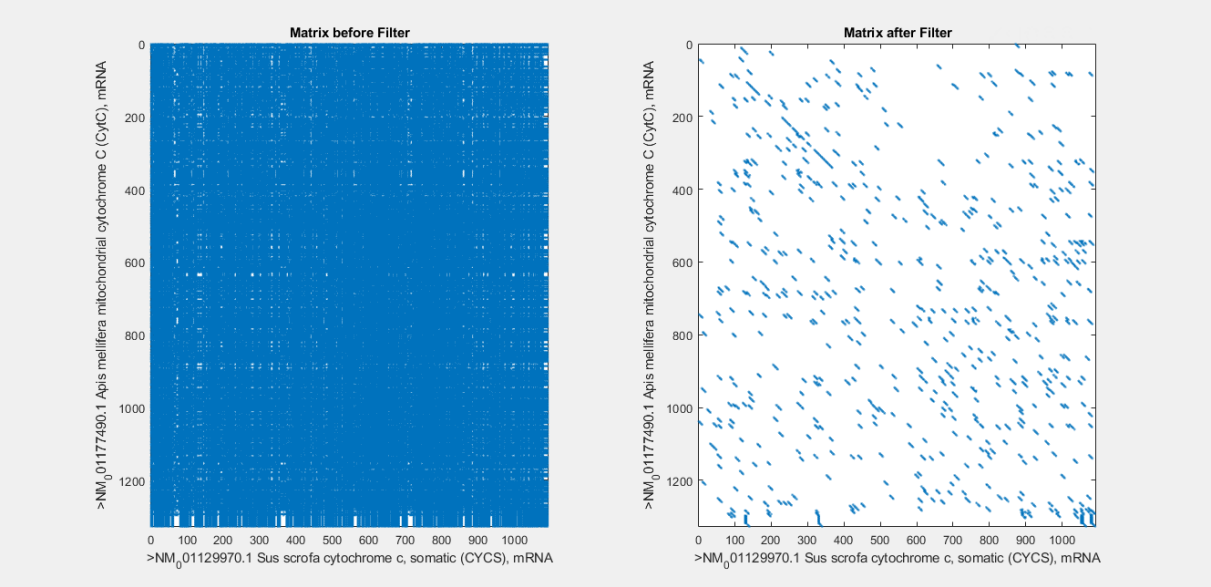
Temat ćwiczenia: Dopasowanie par sekwencji – algorytm kropkowy

Nazwisko i Imię prowadzącego kurs: dr inż. Witold Dyrka

Wykonawcy:	
Imię i Nazwisko **Nr indeksu, wydział**	**Edyta Krukowska 217097, WPPT**
Termin zajęć: dzień tygodnia, godzina	**Piątek 11.15**
Data oddania sprawozdania	**21.03.2019**

Repozytorium: https://github.com/Edie1995/Bioinformatyka/tree/zad1

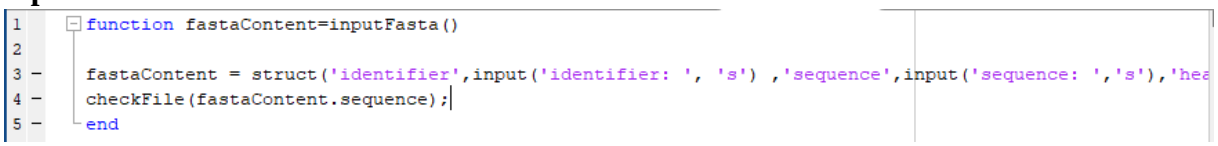
1. **Prezentacja działania programu**:



Rysunek 1 Przykład działania programu, dla genów świni i pszczoły.

2. Analiza złożoności czasowej obliczeniowej i pamięciowej kodu poszczególnych plików i całego programu:

inputFasta.m



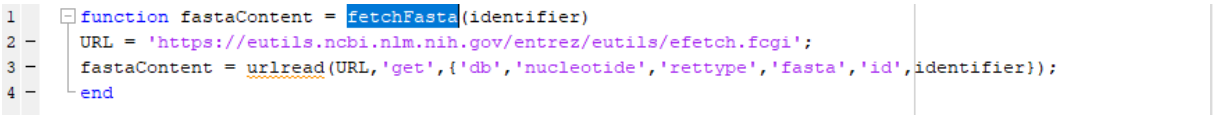
```
1   function fastaContent=inputFasta()
2
3 -   fastaContent = struct('identifier',input('identifier: ', 's') ,'sequence',input('sequence: ','s'),'hea
4 -   checkFile(fastaContent.sequence);
5 -   end
```

- **czasowa**
  - 1 podstawienie (linia 3)
  - 3 pobrania danych z klawiatury (linia 3)
  - 1 wywołanie funkcji

T(n, m)=1+3+1=5

- **pamięciowa**
  - n – rozmiar danych wpisanych z klawiatury (linia 3)
    O(5)

fetchFasta.m



```
1   function fastaContent = fetchFasta(identifier)
2 -   URL = 'https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi';
3 -   fastaContent = urlread(URL,'get',{'db','nucleotide','rettype','fasta','id',identifier});
4 -   end
```

- **czasowa**
  - 1 podstawienie (linia 2)
  - 1 podstawienie (linia 3)

T(n, m)=1+1=2

- **Pojemnościowa**
  - 1 zmienna przechowująca adres strony
  - n – długość łańcucha danych pobranych ze strony

$\exists(n_0, m_0, c > 0)\ \forall(n \geq n_0, m \geq m_0)\ S(n, m) = n + 1 \leq c \cdot n \in O(n)$

readFasta.m

```
1   function dataFasta = readFasta(file)
2   newStr = splitlines(file);
3   lines=0;
4   sequence='';
5   header="";
6   for i=1:size(newStr)
7       if(newStr(i,:)~="")
8           if(lines==0)
9           header=newStr(i,:);
10          lines=lines+1;
11          else
12          strpom=[sequence,newStr(i,:)];
13          sequence=strjoin(strpom);
14          end
15      end
16  end
17  id=idFasta(header);
18  sequence(sequence == ' ') = [];
19  checkFile(char(sequence));
20  dataFasta = struct('identifier',id ,'sequence',strtrim(sequence),'header',header);
21  end
```

- **czasowa**
  - 4 podstawienia (linie 2-5)
  - 1 podstawienie licznika pętli(linia 6)
  - Size(newStr) razy ilość inkrementacji licznika pętli (linia 6)
  - Size(newStr) +1 razy sprawdzeń warunku pętli (linia 6)
  - Size(newStr) powtórzeń pętli wewnętrznej (linia 7-13)
    - 1 Sprawdzenia zgodności znaków (linia 7)
      - 1 sprawdzenie zgodności znaków (linia 8)
      - 2 podstawienia (linie 9-10 lub 12-13)
  - 1 podstawienie
  - 1 usuwanie pustych znaków
  - 1 wywołanie funkcji
  - 1 formatowanie danych wyjściowych

  T(n,m)=4+1+size(newStr)+(size(newStr)+1)+
  size(newStr)(1+1+2)+1+1+1+1=10+6*size(newStr)

  Złożoność obliczeniowa konwertowania pliku FASTA do odczytu przez probram jest co najwyżej rzędu size(newStr), co oznacza, że jest ona rzędu co najwyżej takiego jaka jest długość pliku tekstowego wczytanego ze strony.

  O(size(newStr))

- **Pojemnościowa**
  - 1 zmienna przechowująca całą sekwencję o długości n
  - 1 zmienna przechowująca łańcuch znaków o długości m
  - 1 zmienna przechowująca łańcuch znaków o długości n-m
  - 1 zmienna przechowująca liczbę linii w tekście

- 1 zmienna przechowująca łańcuch znaków o maksymalnej długości n
- 1 zmienna przechowująca id o skończonej długości znaków ok.10
- n – długość danych wyjściowych

$\exists(n_0, m_0, c > 0)\ \forall(n \geq n_0, m \geq m_0)\ S(n, m) = 4n+2 \leq c \cdot n \in O(n)$

checkFile.m

```
1   function [] = checkFile(sequence)
2   permission='[A,C,G,T,U,"",newline]';
3   for i=1:length(sequence)
4       if(regexp(permission,sequence(i)))
5       else
6           error("zly kod");
7       end
8   end
9   end
10
```

➢ **czasowa**
- 1 podstawienie (linia 2)
- 1 podstawienie licznika pętli (linia 3)
- length(sequence)* ilość inkrementacji licznika pętli (linia 3)
- length(sequence)+1*sprawdzeń warunków pętli (linia 3)
- length(sequence)* wykonanie pętli wewnętrznej (linia 4-7)
    - 1 sprawdzenie czy tablica znaków zawiera określony znak z sekwencji (linia 4)
    - 1 podstawienie (linia 6)

Dla ułatwienia zapisu równania przyjmuję, że length(sequence)=n

T(n,m)=1+1+n+(n+1)+n(1+1)=3+3n

Wynika z tego, że podany algorytm jest co najwyżej rzędu n, czyli długości wczytanej sekwencji.

O(n)

➢ **Pojemnościowa**
- 1 zmienna przechowująca tablicę znaków dopuszczonych
  O(1)

createDotPlot.m

```
1   function dots = createDotPlot(matrix1,matrix2)
2 -  x=length(matrix1);
3 -  y=length(matrix2);
4
5 -  dots=zeros(x,y);
6 -  for i=1:x
7 -      for j=1:y
8 -      if(matrix1(i)==matrix2(j))
9 -          dots(i,j)=1;
0 -      end
1 -      end
2 -  end
3 -  end
```

- **czasowa**
  - 3 podstawienia (linie 2,3,5)
  - 1 podstawienie licznika pętli (linia 5)
  - x inkrementacji licznika pętli (linia 5)
  - x+1 sprawdzenia warunków pętli (linia 5)
  - x wykonań pętli wewnętrznej (linie 7-11)
    - 1 podstawienie licznika pętli (linia 7)
    - y inkrementacji licznika pętli (linia 7)
    - y+1 sprawdzenia warunków pętli (linia 7)
    - y wykonań pętli wewnętrznej (linie 8-9)
      - 1 sprawdzenie zgodności elementów macierzy wejściowych
      - 1 podstawienie wartości

Dla poprawienia czytelności w podstawieniu przyjęłam, że n=x, m=y.

T(n,m)=3+1+n+(n+1)+n(1+m+(m+1)+m(1+1))=5+2n+n+nm+nm+n+2nm=5+4n+4nm

Złożoność obliczeniowa tego algorytmu jest co najwyżej rzędu nm

O(nm)

- **Pojemnościowa**
  - x i y zmienne przechowujące rozmiary danych wejściowych matrix1, matrix2
  - xy tablica macierz kropkowa

$\exists(n_0, m_0, c > 0)\ \forall(n \geq n_0, m \geq m_0)\ S(n, m) = nm+n+m \leq c \cdot nm \in O(nm)$

filterFile.m

```
1   function [dotPlot, compareMatrix, label1, label2]=filterFile(okno, prog,fasta1, fasta2)
2
3 -   sequence1=fasta1.sequence;
4 -   sequence2=fasta2.sequence;
5
6 -   label1=fasta1.header;
7 -   label2=fasta2.header;
8
9 -   dotPlot=createDotPlot(sequence1, sequence2);
10 -  [x,y]=size(dotPlot);
11
12 -  if okno>x || okno>y
13 -      error('Zmniejsz okno!');
14 -  end
15 -  compareMatrix=zeros(x,y);
16
17 -  for i=okno:x
18 -      for j=okno:y
19 -         movingFrame=dotPlot(i-okno+1:i,j-okno+1:j) ;
20 -          if sum(diag(movingFrame))>=okno-prog
21 -           compareMatrix(i-okno+1:i,j-okno+1:j)=eye(okno);
22 -          end
23 -      end
24 -  end
25 -  makingPlots(dotPlot,compareMatrix,label1, label2);
26
27 -  end
28
```

- **czasowa**
  - 6 podstawień (linie 3-4,6-7,9-10)
  - 1 sprawdzenie warunku (linia 12)
  - 1 sygnalizacja błędu (linia 13)
  - 1 podstawienie (linia 15)
  - 1 podstawienie licznika pętli (linia 17)
  - x inkrementacji licznika pętli (linia 17)
  - x+1 sprawdzeń warunków pętli (linia 17)
  - x przejść pętli wewnętrznej (linie 18-23)
    - 1 podstawienie licznika pętli (linia 18)
    - y inkrementacji licznika pętli (linia 18)
    - y+1 sprawdzeń warunków pętli (linia 18)
    - y przejść pętli wewnętrznej (linie 19-22)
      - 1 podstawienie (linia 19)
      - 1 sprawdzenie warunku (linia 20)
      - 1 podstawienie (linia 21)
  - 1 wywołanie funkcji

Dla poprawienia czytelności n=x, m=y.

T(n,m)=6+1+1+1+1+n+(n+1)+n(1+m+(m+1)+m(1+1+1))+1=10+n+n+1+n+nm+nm+n+3nm=5nm+4n+11

Wynika z tego, że złożoność obliczeniowa czasowa tego algorytmu jest co najwyżej rzędu nm.

O(nm)

- **pojemnościowa**
  - 2 zmienne przechowujące sekwencje danych wejściowych o długościach m i n
  - tablica o wymiarach nm
  - macierz pomocnicza o wymiarach okna

$\exists(n_0, m_0, c > 0)\ \forall(n \geq n_0, m \geq m_0)\ S(n, m) = nm+n+m+okno \leq c \cdot nm \in O(nm)$

chooseFunction.m

```
1   function [] = chooseFunction(okno, prog,path,filename, varargin)
2   if(length(varargin)<1)
3       filterFile(okno,prog,inputFasta,inputFasta);
4   elseif(length(varargin)==1)
5       filterFile(okno, prog,readFasta(fetchFasta(varargi(1)),inputFasta))
6   elseif(length(varargin)==2)
7
8       filterFile(okno,prog, readFasta(fetchFasta(varargin(1))),readFasta(fetchFasta(varargin(2))));
9
10  end
11      saveFile(path,filename);
12  end
```

- **czasowa**
  - 1 sprawdzenie warunku (linia 2)
  - 1 wywołanie funkcji (linia 3 lub 5 lub 8)
  - 1 wywołanie funkcji (linia 11)

  T(n,m)=3
- **pojemnościowa**
  - Brak zmiennych przechowujących dane

idFasta.m

```
1   function id=idFasta(header)
2   idLine=split(header);
3   idFirst=idLine{1,1};
4   id=idFirst(2:end);
5   end
```

- **czasowa**
  - 3 podstawienia (linia 2-4)

  T(n,m)=3
- **pojemnościowa**

  brak

makingPlots.m

```
1   function []= makingPlots(plot, compareMatrix, label1, label2)
2   filtered=plot&compareMatrix;
3   figure
4   subplot(1,2,1)
5   spy(plot);
6   xlabel (label1);
7   ylabel(label2);
8   title('Matrix before Filter')
9   subplot(1,2,2)
10  spy(filtered);
11  xlabel (label1);
12  ylabel(label2);
13  title('Matrix after Filter')
14  end
```

- **czasowe**
  - 1 podstawienie (linia 2)
  - 10 dodatków do wykresu (linie 4-13)

T(n,m)=11

- **pojemnościowe**
  brak

saveFile.m

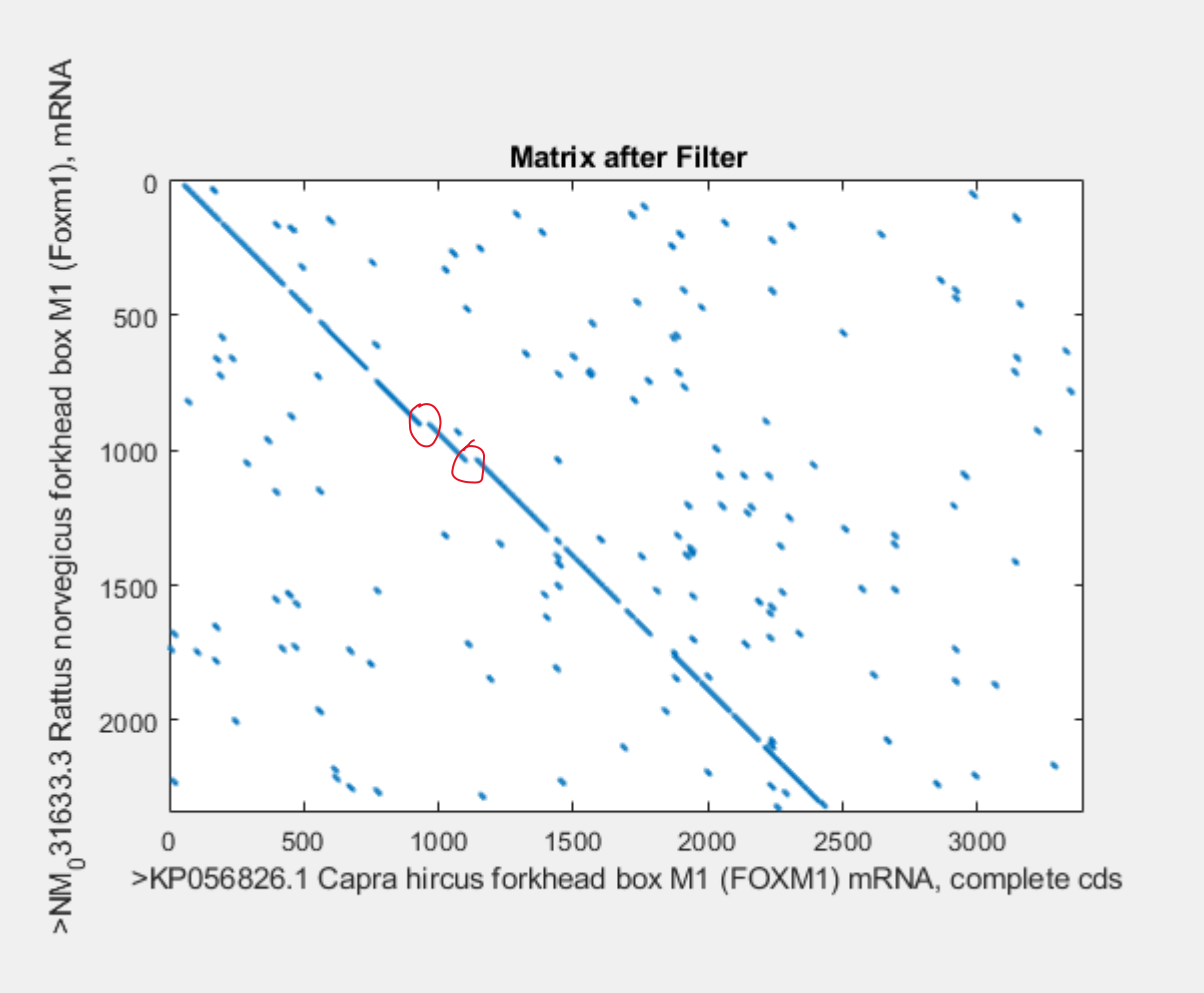
```
1   function []=saveFile(path,filename)
2   sciezka=path+"\"+filename;
3   print(sciezka,'-djpeg');
4   end
```

- **czasowe**
  - 1 podstawienie (linia 2)
  - 1 wywołanie funkcji (linia 3)
- **Pojemnościowe**
  - 1 zmienna przechowująca ścieżkę pliku do zapisu

3. Porównanie przykładowych par sekwencji

3.1. Ewolucyjnie powiązanych

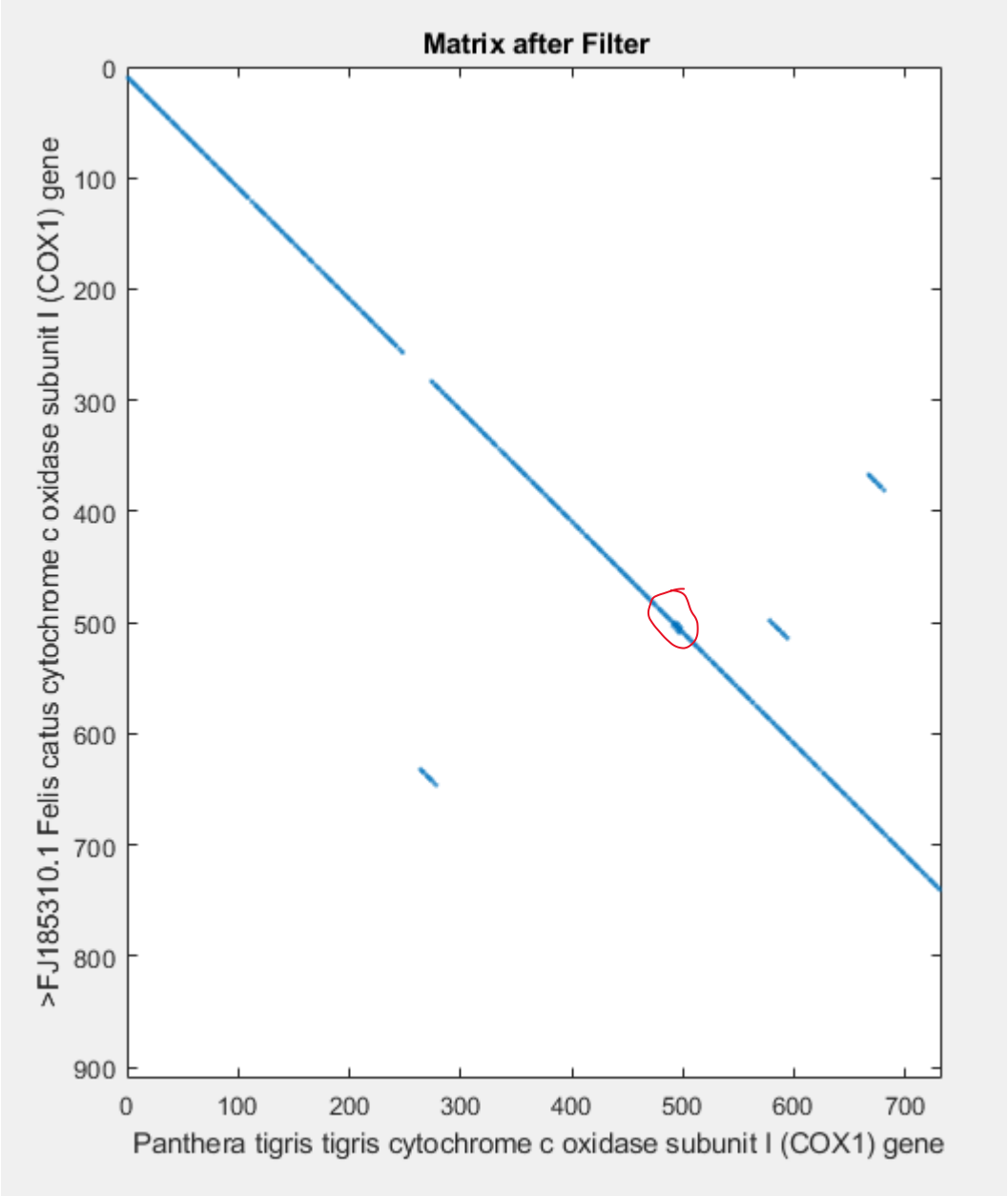
- **Porównanie sekwencji szczura z kozą (okno 15, próg 3)**



Rysunek 2 Macierz kropkowa porównująca szczura (oś y) i kozę (oś x)

Z przedstawionego powyżej wykresu możemy wywnioskować, że oba zwierzęta są ze sobą silnie spokrewnione. Wynika to z faktu, że zauważamy na przekątnej linię przekątną, która na to wskazuje. Sekwencje nukleotydowe szczura i kozy charakteryzują się analogią fragmentów łańcuchów nukleotydowych w większej części przekątnej. Fakt, że przekątna nie znajduje się na środku okna, wynika z faktu, że fragment łańcucha dla szczura posiada mniej elementów niż fragment pobrany dla kozy. Zauważamy, że a zaznaczonych na czerwono fragmentach występuje insercja/delecja, co powoduje charakterystyczne „przesunięcia” zgodnych fragmentów.

- **Porównanie kota z panterą (okno 15, próg 3)**

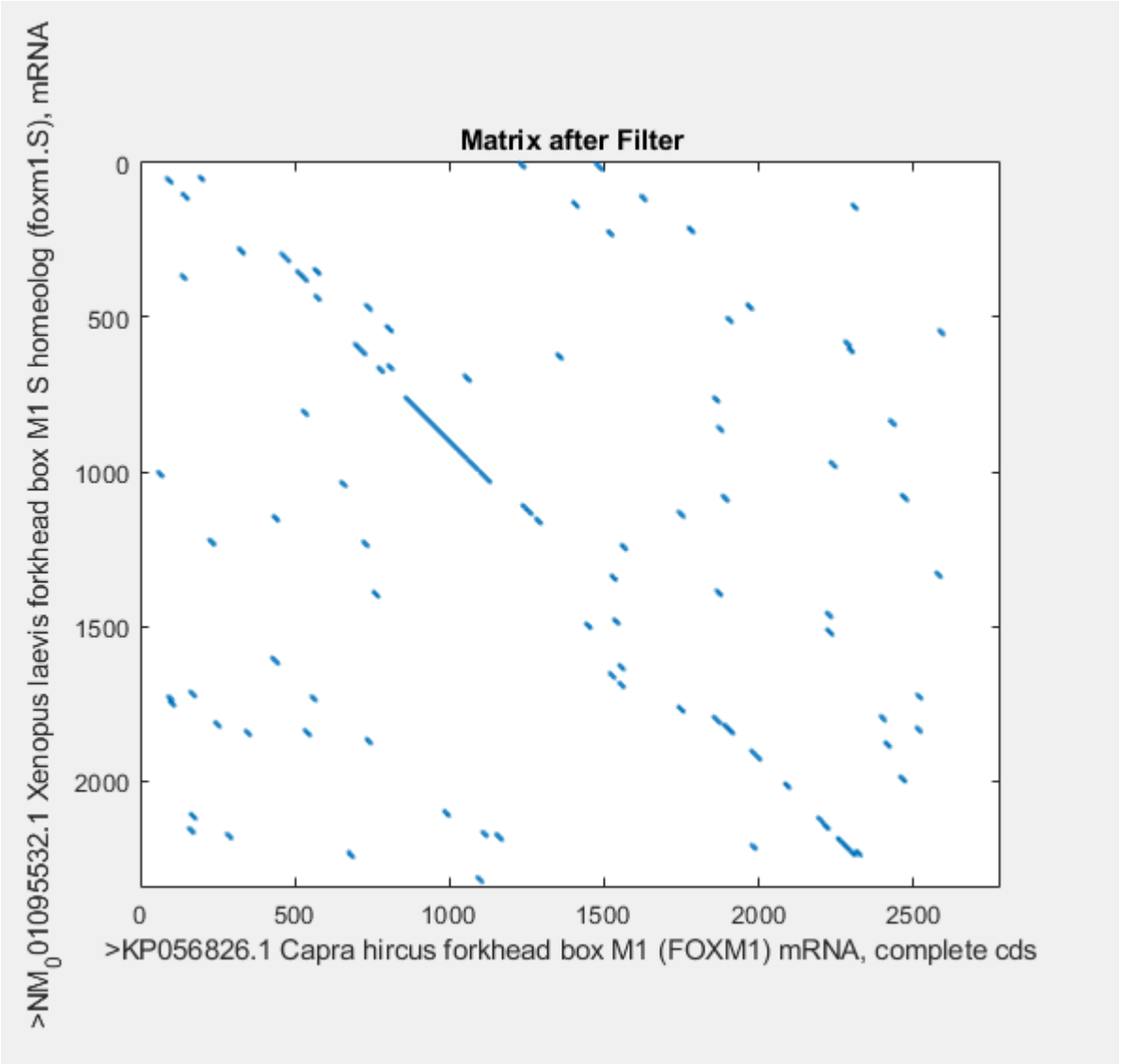


Rysunek 3 Macierz punktowa porównująca kota (oś y) z panterą (oś x)

Ponieważ, również w tym przypadku widzimy wyraźną przekątną oznacza to, że te gatunki są ze sobą silnie spokrewnione. Przekątna jest przesunięta, ponieważ, również w tym przypadku długości sekwencji różniły się od siebie. Na wykresie w miejscu zaznaczonym na wykresie możemy zaobserwować duplikację. Przerwy w linii podobieństwa gatunków mimo wspólnego pochodzenia mogą wynikać, z mutacji punktowych.

3.2. Ewolucyjnie niepowiązanych

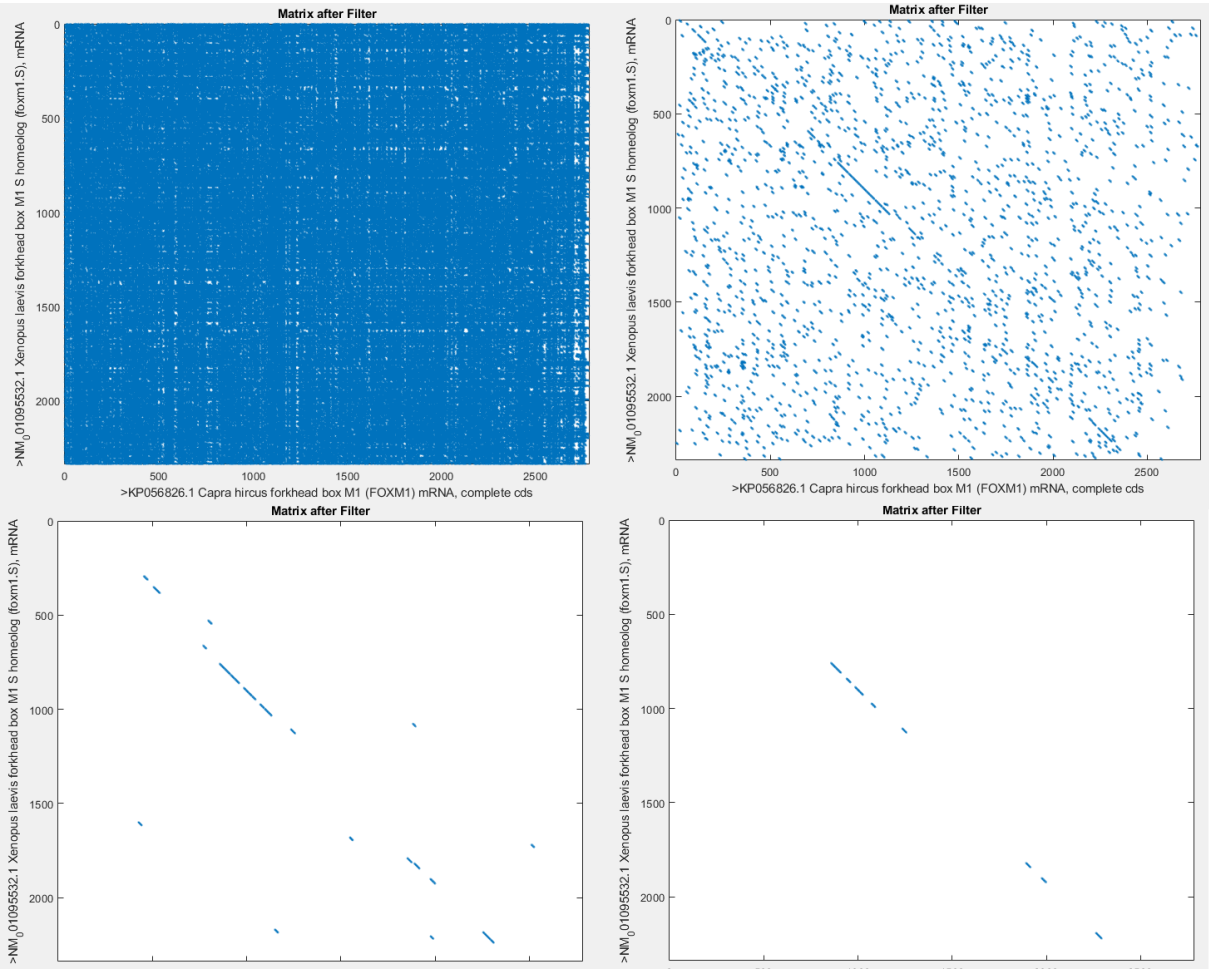
- **Porównanie żaby szponiastej z kozą (okno 15, próg 3).**



Rysunek 4 Macierz kropkowa wygenerowana dla żaby szponiastej (oś y) oraz kozy (oś x)

Jak zauważamy na powyższym wykresie, w przypadku tych dwóch gatunków, wykres pokrywa się w znacznie mniejszym stopniu, niż poprzednio interpretowane pary. Zauważamy, że przekątna jest zgodna tylko w niewielkich fragmentach, co oznacza, ze organizmy te są spokrewnione w bardzo niewielkim stopniu.

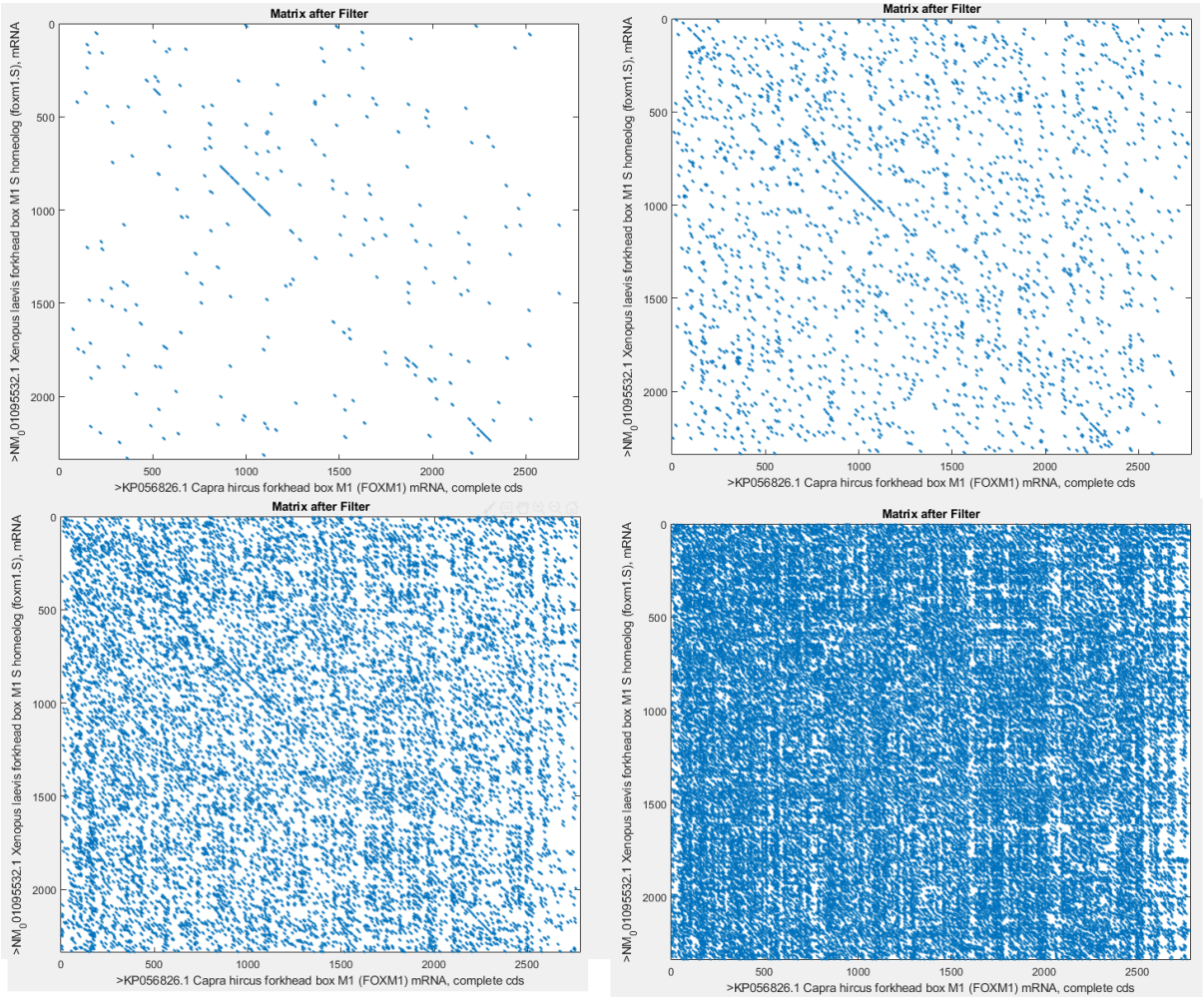
- **Porównanie wykresów przy różnych wielkościach okna 5,10,15,20**



Rysunek 5 Wygenerowane macierze kropkowe dla żaby szponiastej i kozy, przy różnych rozmiarach okna i jednym progu błędu.

W lewym górnym rogu widzimy wykres dla okna o rozmiarze 5, następnie po prawej okno =10. Na lewym dolnym rogu widzimy macierz kropkową dla okna =15, ostatnia macierz została wygenerowana dla okna 20. Jak zauważamy na powyższym obrazku, na czytelność wykresu i ilość wyświetlonych sekwencji zgodnych ma wpływ rozmiar okna. Im większe okno, tym mniej sekwencji zostanie zaprezentowanych. Wszystkie powyższe macierze zostały wygenerowane dla takiej samej wartości progu błędu.

- **Porównanie macierzy dla różnych progów błędu (1,2,3,4). W porównaniu użyto okna o rozmiarze 10.**



Rysunek 6 Wygenerowane macierze kropkowe dla żaby szponiastej i kozy, przy takich samych rozmiarach okna i różnych progach błędu..

Kolejno w lewym górnym rogu widzimy macierz dla progu błędu = 1, następnie po prawej stronie próg 2. W lewym dolnym rogu próg 3 i ostatecznie w prawym dolnym rogu macierz dla progu błędu =4.

Jak widzimy, jednoznacznie możemy stwierdzić, że na jakość czytelności takiej macierzy ma wpływ próg błędu. Wynika to z faktu, że im więcej może być „błędnych" par w ramach jednej przekątnej okna, tym więcej takich par zostanie zaznaczonych na przefiltrowanym wykresie.