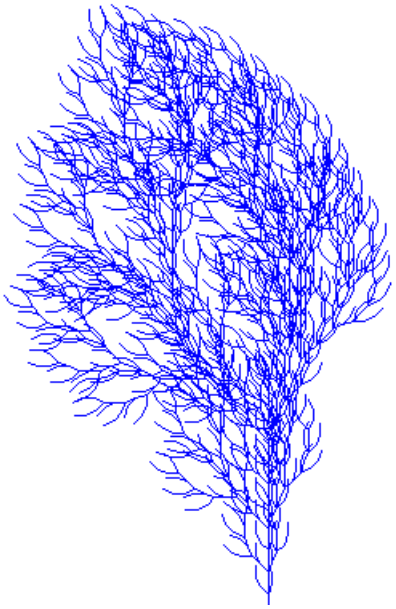
Technologia informacyjna

Dariusz Skibicki



Podręcznik wykonany w ramach projektu

POKL.04.01.01-00-013/09

Inżynieria biomedyczna - kierunek przyszłości

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

WYDZIAŁ INŻYNIERII MECHANICZNEJ
UNIWERSYTET TECHNOLOGICZNO-PRZYRODNICZY
IM. J.J.ŚNIADECKICH W BYDGOSZCZY

UNIA EUROPEJSKA
EUROPEJSKI FUNDUSZ SPOŁECZNY

Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego

SPIS TREŚCI

1. MATLAB DLA ZAAWANSOWANYCH

Wszystkie ćwiczenia i zadania w tym rozdziale należy wykonać zapisując kod Matlaba w m-plikach.

1.1. OBLICZANIE WARTOŚCI STAŁYCH MATEMATYCZNYCH

Ćwiczenie 1.1.

Wykorzystując wektoryzację obliczeń, za pomocą funkcji *sum()* obliczmy stałą e według wzoru:

$$e = \sum_{k=0}^{\infty} \frac{1}{k!} \tag{1.1}$$

Powyższa formuła przewiduje nieskończoną liczbę wyrazów, jednak dla naszych celów przyjmijmy 1001 pierwszych. Zdefiniujmy wektor k o takiej długości poczynając od 0:

```
k=0:1000;
```

Dzięki temu, że funkcja *fuctorial()* obliczy silnię dla każdej z wartości wektora *n*, oraz korzystając z operatora kropki przy znaku dzielenia, możemy wprost zapisać ułamek. Sumowanie wykonamy za pomocą funkcji *sum()*.

```
e=sum(1./factorial(k))
```

Ćwiczenie 1.2.

Obliczmy wartość π na podstawie zależności:

$$1 + \cfrac{1^2}{2 + \cfrac{3^2}{2 + \cfrac{5^2}{2 + \cfrac{7^2}{2+\ldots}}}} = \frac{4}{\pi} \tag{1.2}$$

Łatwo dostrzec, że zależność ma charakter rekurencyjny. Mianownik ułamka ma postać całej zależności. Według określonego klucza zmieniają się tylko wartości pewne wartości. Zaproponujmy, więc funkcję, która

ustaloną przez użytkownika liczbę razy wywołuje samą siebie, w celu wyliczenia kolejnych, coraz bardziej „zagłębionych” ułamków:

```
function my_pi = oblicz_pi(n,k)
% n – aktualna iteracja
% k – maksymalna liczba iteracji
if n < k
   my_pi  = 1 + (2*n-1)^2 / (2+oblicz_pi(n+1,k));
else
   my_pi = 0;
end
```

Po zapisaniu funkcji w pliku *oblicz_pi.m* możemy ją wywołać w następujący sposób:

```
n=1
moje_pi = 4 / oblicz_pi(n,100)
```

Zadanie 1.1.

Wykorzystując iteracje, za pomocą pętli *for* oblicz stałą e według wzoru:

$$e = \sum_{k=0}^{\infty} \frac{1}{k!} \tag{1.3}$$

Zadanie 1.2.

Wykorzystując wektoryzację obliczeń, za pomocą funkcji *sum()* oblicz stałą e według wzoru:

$$\mathrm{e} = \frac{1}{2}\sum_{\mathrm{k}=0}^{\infty} \frac{\mathrm{k}+1}{\mathrm{k}!} \tag{1.4}$$

Zadanie 1.3.

Wykorzystując wektoryzację obliczeń, za pomocą funkcji *sum()* oblicz stałą e według wzoru:

$$\mathrm{e}^{-1} = \sum_{\mathrm{k}=0}^{\infty} \frac{-1^{\mathrm{k}}}{\mathrm{k}!} \tag{1.5}$$

Zadanie 1.4.

Wykorzystując wektoryzację obliczeń, za pomocą funkcji *sum()* oblicz stałą e według wzoru:

$$e = \left(\sum_{k=0}^{\infty} \frac{1-2k}{(2k)!} \right)^{-1} \tag{1.6}$$

Zadanie 1.5.

Wykorzystując wektoryzację obliczeń, za pomocą funkcji *sum()* oblicz stałą π według wzoru:

$$\sum_{k=0}^{\infty} \frac{-1^k}{2n+1} = \frac{\pi}{4} \tag{1.7}$$

Zadanie 1.6.

Wykorzystując wektoryzację obliczeń, za pomocą funkcji *prod()* oblicz stałą π według wzoru:

$$\prod_{n=1}^{\infty} \frac{4n^2}{4n^2-1} = \frac{\pi}{2} \tag{1.8}$$

Zadanie 1.7.

Zaproponuj funkcję rekurencyjną obliczającą stałą π według wzoru:

$$\cfrac{4}{1+\cfrac{1^2}{3+\cfrac{2^2}{5+\cfrac{3^2}{7+\cfrac{4^2}{9+\cdots}}}}} = \frac{\pi}{2} \tag{1.9}$$

1.2. CHAOS W KOMPUTERZE

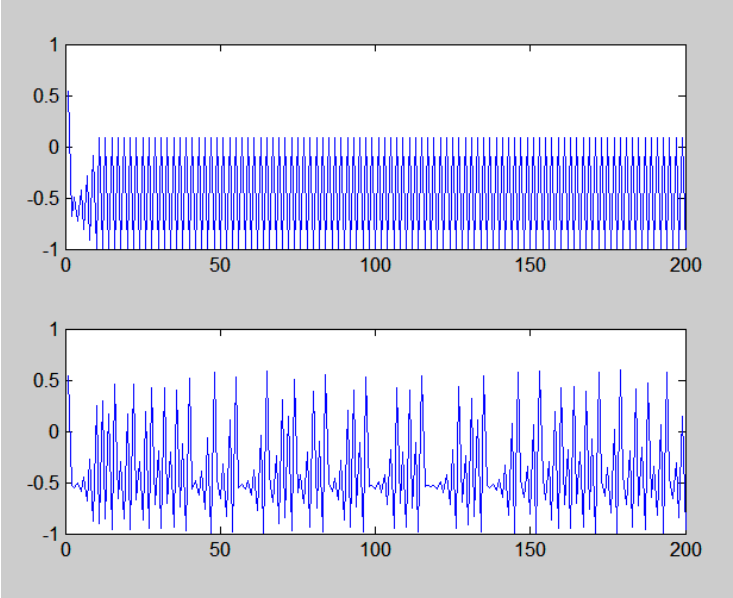
Zamiana liczb w systemie dziesiętnym, którymi się posługujemy na liczby binarne, którymi posługuje się komputer, powoduje utratę ich dokładności. Co gorsza przy wykonywaniu wielokrotnych obliczeń, błędy te mogą się kumulować powodując nieprzewidywalne wyniki obliczeń. W poniższym zadaniu wizualizowano tego rodzaju efekt.

Zadanie 1.8.

W jednym oknie dialogowym wykonaj dwa wykresy wg rekurencyjnej zależności:

$$x_{i+1} = kx_i^2 - 1 \tag{1.10}$$

Pierwszy wykres wykonaj przyjmując do obliczeń $k = 1.1$, a drugi dla $k = 1.6$. W obu przypadkach oblicz 200 pierwszych wartości, rozpoczynając od $x_1 = 0.54321$. Wykorzystaj funkcję subplot() do stworzenia dwóch wykresów w jednym oknie dialogowym.



Rys. 1.1.

1.3. LIŚĆ PAPROCI

Rysunek liścia paproci można uzyskać nanosząc na wykresie punkty, których współrzędne oblicza się wykorzystując cztery poniższe zależności iteracyjne:

Zależność 1

$$x_{n+1} = 0$$

Zależność 2

$$x_{n+1} = 0.2x_n - 0.26\,y_n$$
$$y_{n+1} = 0.23x_n + 0.22\,y_n + 1.6$$

Zależność 3

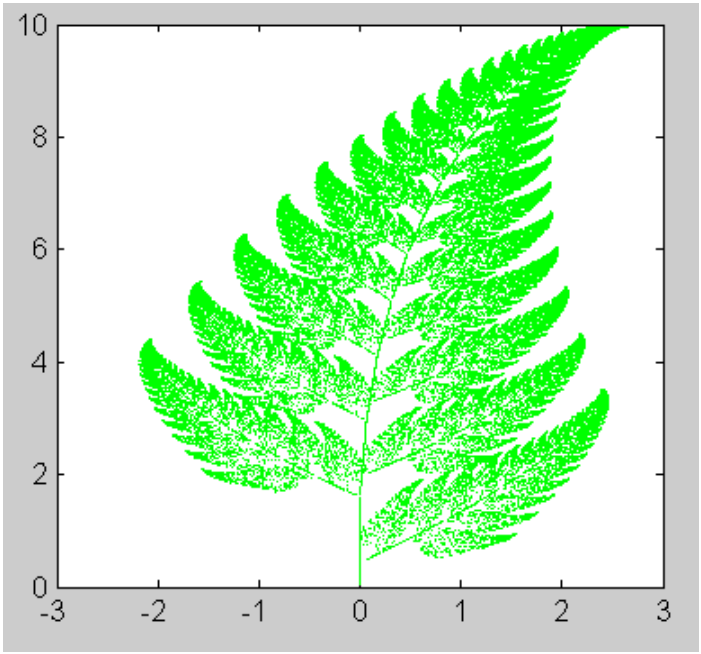
$$x_{n+1} = -0.15x_n + 0.28\,y_n$$

$$y_{n+1} = 0.26x_n + 0.24y_n + 0.44$$

Zależność 4

$$x_{n+1} = 0.85x_n + 0.04\,y_n$$
$$y_{n+1} = -0.04x_n + 0.85\,y_n + 1.6$$

Aby liść miał dokładnie taki kształt jak na rysunku zależność f_1 należy zastosować 1% razy, f_2 i f_3 po 7%, a zależność aż f_4 85% razy.



Rys. 1.2.

Zadanie 1.9.

Narysuj liść paproci wg następującego algorytmu:

1. Utwórz puste wektory X i Y.
2. Rozpocznij pętlę *for* wykonywaną 100 000 razy.
3. W pętli losuj wartość za pomocą funkcji *rand()*. Zapamiętaj wylosowaną wartość.
4. W zależności od wylosowanej wartości, za pomocą komendy *if*, wybieraj odpowiednią zależność iteracyjną 1, 2, 3 lub 4 i wyliczaj kolejne wartości współrzędnych, zapisując je w wektorach X i Y.
5. Zakończ pętlę *for*
6. Za pomocą komendy *plot()* narysuj liść paproci z punktów o wyliczonych współrzędnych X, Y.

Zadanie 1.10.

Modyfikuj postać paproci, eksperymentując z wartościami współczynników równań oraz częstotliwościami ich stosowania.

Zadanie 1.11.

Wykonaj rysunek liścia paproci ja wyżej, ale dla każdej z czterech zależności zastosuj inny kolor.

1.4. TRANSFORMACJE GEOMETRYCZNE

Dowolną transformację geometryczną na płaszczyźnie można uzyskać stosując trzy macierze transformacji. Przesunięcie o wektor $[t_x \quad t_y]$ odbywa się za pomocą macierzy przesunięcia w postaci:

$$T = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{1.11}$$

Skalowanie o współczynniki $s_x \quad s_y$ odbywa się za pomocą macierzy skalowania, która ma postać:

$$S = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1.12}$$

W celu dokonania obrotu o kąt φ należy zdefiniować macierz obrotu w postaci:

$$R = \begin{bmatrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1.13}$$

Wszystkie powyższe macierze można wymnożyć w odpowiedniej olejności uzyskując ogólna macierz transformacji:

$$M = T \cdot S \cdot R \tag{1.14}$$

W celu wykonania transformacji należy przemnożyć współrzędne wierzchołków figury płaskiej przez macierz *M*. Współrzędne wierzchołków zapisane muszą być w macierzy o formacie:

$$Wsp = \begin{bmatrix} x1 & x2 & \dots & xn \\ y1 & y2 & \dots & yn \\ 1 & 1 & \dots & 1 \end{bmatrix} \tag{1.15}$$

Wówczas współrzędne figury po transformacji uzyskujemy za pomocą mnożenia

$$Nowe\_wsp = M * Wsp \tag{1.16}$$

Zadanie 1.12.

Wielokąt w kształcie litery K opisany jest za pomocą następujących współrzędnych wierzchołków:

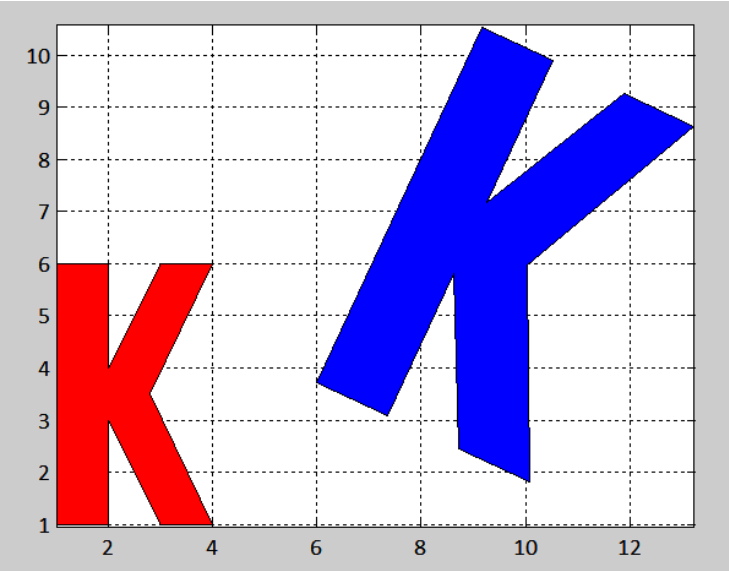
$$(1,1)(2,1)(2,3)(3,1)(4,1)(2.8,3.5)(4,6)(3,6)(2,4)(2,6)(1,6)(1,1)$$

Dokonaj transformacji litery K, dla następujących parametrów:

- przemieszczenia: $\mathrm{Tx} = 4, \mathrm{Ty} = 3,$
- skalowania: $\mathrm{Sx} = 1.5, \mathrm{Sy} = 1.5,$
- obrotu: $\varphi = 25°,$

Postępuj wg następującego algorytmu:

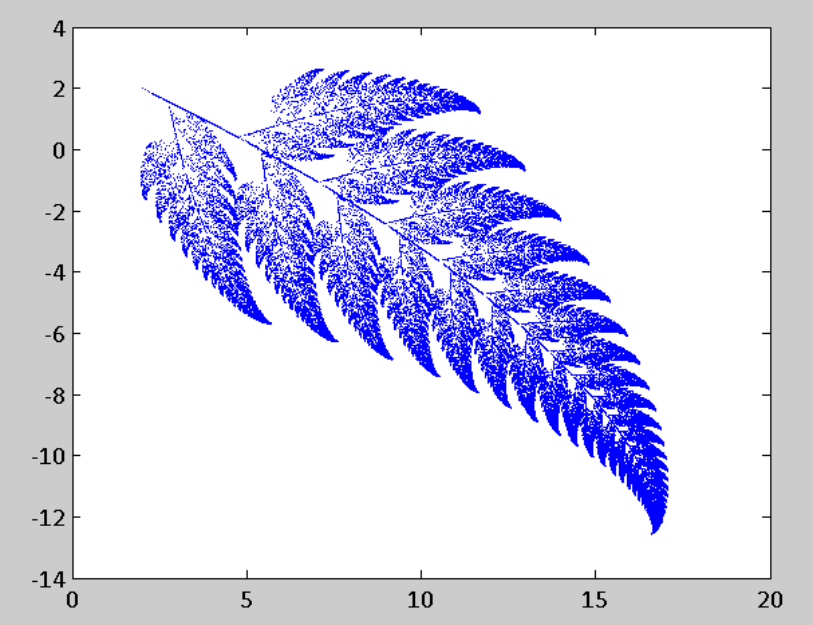
1. Utwórz macierz współrzędnych wierzchołków *Wsp* litery K w formacie (6.15).
2. Narysuj za pomocą komendy *fill()* literę K.
3. Utwórz macierze *T*, *S*, *R* oraz macierz *M*, według równań (6.11-6.14)
4. Oblicz nowe współrzędne *Wsp_nowe* według (6.16).
5. Na tym samym wykresie narysuj obróconą literę K.



Rys. 1.3.

Zadanie 1.13.

Wykonaj obrót o 120° rysunku liścia paproci.



Rys. 1.4.

1.5. L-SYSTEMY

L-systemy (systemy Lindemayera), znajdują zastosowanie w grafice komputerowej, szczególnie do generowania fraktali oraz kształtów roślin. L-systemy składają się ze zbioru symboli będących zakodowanym zapisem jakiejś geometrii oraz reguł przetwarzania tych symboli w celu generowania coraz bardziej złożonych kształtów.

Początkowy ciąg symboli zwany aksjomatem jest przetwarzany przy zastosowaniu reguł. Reguły w poniższych przykładach sprowadzają się do polecenia zamiany określonego znaku na inny znak lub ciąg innych znaków.

Dla przykładu przyjmijmy, że aksjomat ma wartość „B” oraz dane są dwie reguły „symbol A zastąp symbolem AB” oraz „symbol B zastąp symbolami A”. Przeanalizujmy 6 pierwszych kroków:

1. „B”.
2. „A”
3. „AB”
4. „ABA”
5. „ABAAB”
6. „ABAABABA”

Otrzymany w wyniku wielokrotnego stosowania reguł ciąg symboli, zawiera przepis na narysowanie figury geometrycznej. Każdy symbol ciągu jest bowiem określoną komendą rysowania. W poniższych przykładach przyjęto następującą konwencję:

1. Litery F, G, H oznaczają polecenie narysowania odcinka.
2. Znaki „+”, „-” kodują polecenie skrętu odpowiednio w prawo i w lewo o określone pochylenie.
3. Nawiasy „[”, „]” oznaczają odpowiednio rozpoczęcie i zakończenie nowej gałęzi.

Ćwiczenie 1.3.

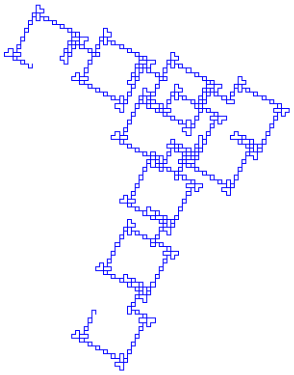
Dany jest aksjomat „F” oraz następująca reguła „F” zamień na „F-F-FF+F-F-F”. Narysuj wzór stosując czterokrotnie regułę i przyjmując pochylenie 90°.

W tym celu wykonaj następujący kod.

```
% zdefiniowanie danych
aksjomat='F'
regula_1_znak = 'F';
regula_1_zamien_na = 'F-F-FF+F-F-F';
powtorzen=4;
pochylenie = 90;

% tworzenie wzoru
wzor = aksjomat;
for i=1:powtorzen
   wzor = strrep(wzor,regula_1_znak,regula_1_zamien_na);
end

 % rysowanie
X = 0;  Y = 0;  kat = 0;
 licznik_pamieci = 1;
hold on

% dla każdego znaku wykonaj
```

```
for i=1:length(wzor)
  switch wzor(i)
  % jeśli to litera, oblicz współrzędne następnego punktu i narysuj
    odcinek
  case {'F','G','H'}
     nowyX = X + cos(kat);
     nowyY = Y + sin(kat);
     line([Y nowyY], [X nowyX],'color','b','linewidth',1);
     X = nowyX;
     Y = nowyY;
  % jeśli to znaki „+” lub „-” zmodyfikuj kierunek rysowania o
  przyjętą wartość pochylenia
  case '+'
     kat = kat + pochylenie*3.14/180;
  case '-'
     kat = kat - pochylenie*3.14/180;
  % jeśli to znaki „[” lub „]” to odpowiednio zapamiętaj aktualne
  współrzędne i kierunek rysowania lub przywróć zapamiętane
  współrzędne i kierunek rysowania
  case '['
     pamiec(licznik_pamieci,:) = [X Y kat];
     licznik_pamieci = licznik_pamieci +1 ;
  case ']'
     licznik_pamieci = licznik_pamieci -1 ;
     X=pamiec(licznik_pamieci,1);
     Y=pamiec(licznik_pamieci,2);
     kat=pamiec(licznik_pamieci,3);
  end
end
daspect([1,1,1])
```

Wykonanie kodu spowoduje narysowanie następującego wzoru.

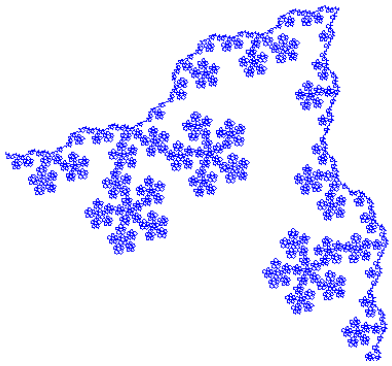


Rys. 1.5.

Zadanie 1.14.

Korzystając z powyższego kodu, narysuj wzór dla poniższych danych:

```
aksjomat='F'
regula_1_znak = 'F';
regula_1_zamien_na = 'F+F-F--F+F+F';
pochylenie = 72;
powtorzen=5;
```



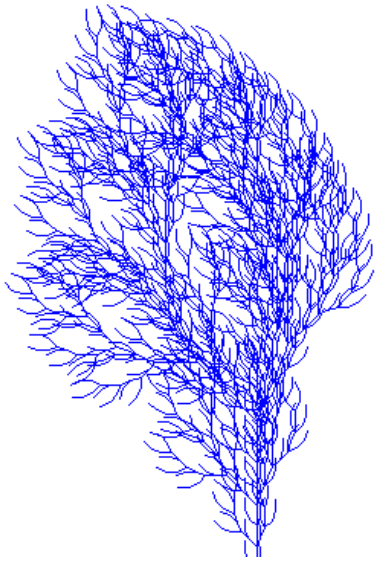
Rys. 1.6.

Zadanie 1.15.

Korzystając z powyższego kodu, narysuj wzór dla poniższych danych:

```
aksjomat='F'
```

```
regula_1_znak = 'F';
regula_1_zamien_na = 'FF-[-F+F+F]+[+F-F-F]';
pochylenie = 22.5;
powtorzen=4;
```

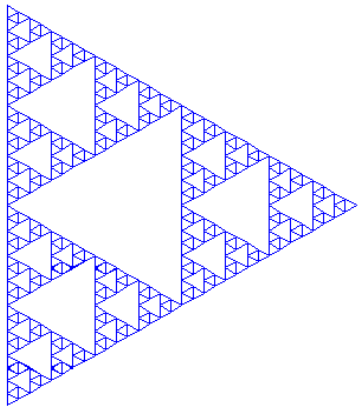


Rys. 1.7.

Zadanie 1.16.

Korzystając z powyższego kodu, narysuj wzór dla poniższych danych:

```
aksjomat='F+F+F'
regula_1_znak = 'F';
regula_1_zamien_na = 'F+F-F-F+F';
pochylenie = 120;
powtorzen=5;
```

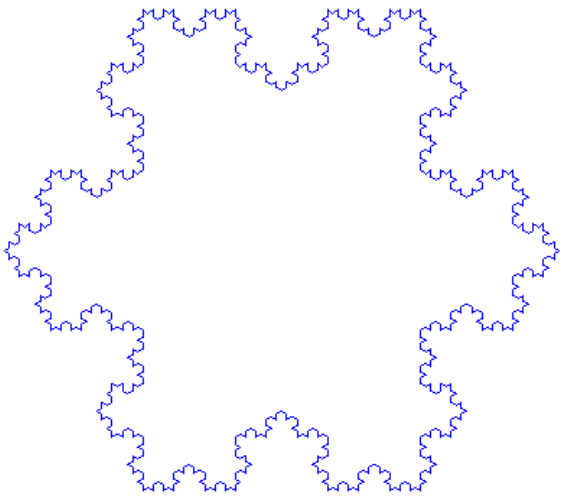


Rys. 1.8.

Zadanie 1.17.

Korzystając z powyższego kodu, narysuj wzór dla poniższych danych:

```
aksjomat='F++F++F'
regula_1_znak = 'F';
regula_1_zamien_na = 'F-F++F-F';
pochylenie = 60;
powtorzen=4;
```

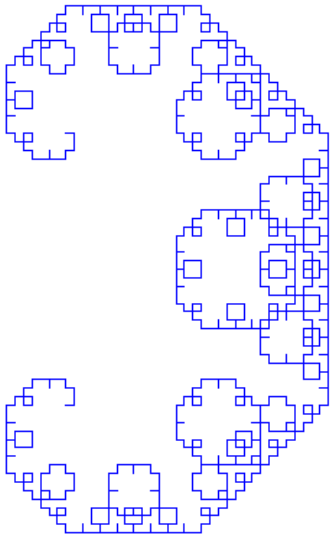


Rys. 1.9.

Zadanie 1.18.

Korzystając z powyższego kodu, narysuj wzór dla poniższych danych:

```
aksjomat='F'
regula_1_znak = 'F';
regula_1_zamien_na = '+F--F+';
pochylenie = 45;
powtorzen=10;
```



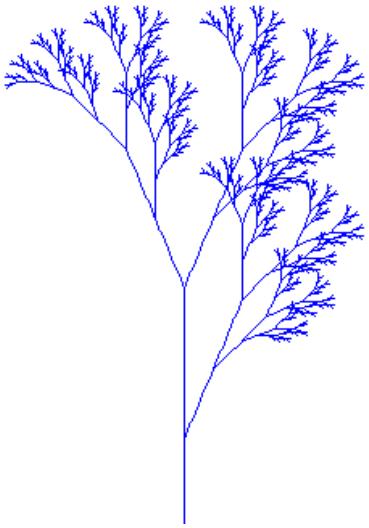
Rys. 1.10.

Zadanie 1.19.

Zmodyfikuj stosowany do tej pory kod tak, aby można było zastosować dwie reguły. W tym celu należy zmodyfikować fragment kodu opatrzony komentarzem „tworzenie wzoru”, przez dodanie w pętli wiersza umożliwiającego wykonanie drugiej reguły.

Następnie narysuj wzór dla danych:

```
aksjomat='G'
regula_1_znak = 'F';
regula_1_zamien_na = 'FF';
regula_2_znak = 'G';
regula_2_zamien_na = 'F[+G]F[-G]+G';
pochylenie = 22.5;
powtorzen=6;
```

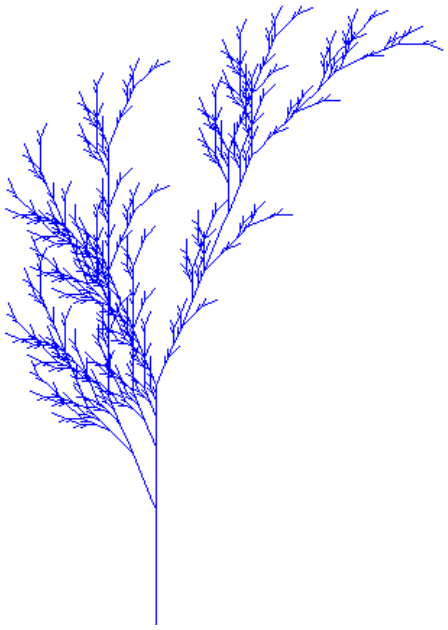


Rys. 1.11.

Zadanie 1.20.

Następnie narysuj wzór dla następujących danych:

```
aksjomat='G'
regula_1_znak = 'F';
regula_1_zamien_na = 'FF';
regula_2_znak = 'G';
regula_2_zamien_na = 'F-[[G]+G]+F[+FG]-G';
pochylenie = 22.5;
powtorzen=5;
```

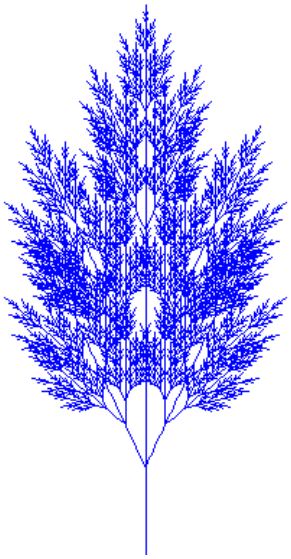


Rys. 1.12.

Zadanie 1.21.

Następnie narysuj wzór dla następujących danych:

```
aksjomat='G'
regula_1_znak = 'F';
regula_1_zamien_na = 'FF';
regula_2_znak = 'G';
regula_2_zamien_na = 'F[+G][-G]F[+G][-G]FG';
pochylenie = 25;
powtorzen=5;
```



Rys. 1.13.

1.6. SZYFROWANIE

W celu zaszyfrowania tekstu należy najpierw zamienić ciąg znaków komendą *uint16()* na ciąg liczb. Następnie do liczb należy bitowo, stosując funkcję *bitxor()* dodać klucz szyfrujący. Klucz szyfrujący również jest liczbą, czyli ciągiem bitów. Aby odkodować zaszyfrowany tekst, należy do niego ponownie dodać za pomocą funkcji *bitxor()* ten sam klucz. Następnie należy zamienić otrzymany ciąg liczb na ciąg znaków za pomocą komendy *char()*.

Ćwiczenie 1.4.

Dla przykładu zaszyfrujmy literę „a” stosując klucz w postaci liczby 3. Najpierw zamieniamy literę „a” na liczbę.

```
>> a_liczba=uint16('a')

a_liczba =

   97
```

Przedstawmy ją w postaci zero-jedynkowej.

```
>> a_liczba_binarnie=dec2bin(a_liczba)

a_liczba_binarnie =

   1100001
```

Klucz także zamieńmy na postać bitową.

```
>> klucz_binarnie=dec2bin(3)

klucz_binarnie =

   0000011
```

Wykonujemy operację bitową. Funkcja *dec2bin()* zwraca łańcuch znaków, więc zanim wykonamy dodawanie bitowe musimy przekonwertować uzyskane wyniki na liczby za pomocą funkcji *str2num()*.

```
>> zaszyfrowana_liczba_binarnie =
bitxor(str2num(a_liczba_binarnie), str2num( klucz_binarnie))

zaszyfrowana_liczba_binarnie =

   1100010
```

Przekonajmy się jaka to litera.

```
>> zaszyfrowana_liczba =
bin2dec(num2str(zaszyfrowana_liczba_binarnie))

zaszyfrowana_liczba =

  98

>> zaszyfrowana = char(zaszyfrowana_liczba)

zaszyfrowana =

  b
```

Okazuje się, że w wyniku szyfrowania otrzymaliśmy literę „b”. Jeśli do litery „b” dodamy ponownie ten sam klucz w postaci liczby 3 otrzymamy z powrotem literę „a".

```
>> odszyfrowana_liczba_binarnie =
bitxor(zaszyfrowana_liczba_binarnie, str2num( klucz_binarnie))

odszyfrowana_liczba_binarnie =

   1100001

>> odszyfrowana =
char(bin2dec(num2str(odszyfrowana_liczba_binarnie)))

odszyfrowana =

   a
```

Ćwiczenie 1.5.

Powyższe ćwiczenie jest wygląda dość skomplikowane jak na proste zadanie szyfrowania jednego znaku. Powodem jest każdorazowe przedstawianie liczb w postaci binarnej, wynikające tylko z potrzeby

dydaktycznej. Jeśli jednak zrezygnujemy z konwersji liczb na postać binarną zadanie stanie się bardziej przejrzyste:

```
>> klucz=3
klucz =
    3
>> a_liczba=uint16('a')
a_liczba =
    97
>> zaszyfrowana_liczba = bitxor(a_liczba, klucz)
zaszyfrowana_liczba =
    98
>> zaszyfrowana = char(zaszyfrowana_liczba)
zaszyfrowana =
b
>> odszyfrowana_liczba = bitxor(zaszyfrowana_liczba, klucz)
odszyfrowana_liczba =
    97
>> odszyfrowana = char(odszyfrowana_liczba)
odszyfrowana =
a
```

Zadanie 1.22.

Zaszyfruj tekst „To jest mój pin kod: 1234.”, stosując klucz w postaci liczby 23. Poniżej przedstawiono wyniki operacji dla każdego etapu.

Szyfrowany tekst:

```
tekst_oryginalny = ’To jest mój pin kod: 1234’.
```

Tekst zamieniony na ciąg liczb:

```
tekst_cyfry = 84   111   32   106   101   115   116   32   109
243   106   32   112   105   110   32   107   111 100   58   32
49   50   51   52   46
```

Tekst w postaci ciągu liczb do którego został dodany klucz:

```
tekst_cyfry_zakodowany = 67   120    55   125   114   100    99
55   122   228   125    55   103   126   121    55   124   120
115    45    55    38    37    36    35    57
```

Tekst zakodowany w postaci znaków:

```
tekst_zakodowany = Cx7}rdc7zä}7g~y7|xs-7&%$#9
```

Tekst zakodowany do którego ponownie dodany został klucz szyfrujący:

```
tekst_cyfry_rozkodowany = 84   111    32   106   101   115   116
32   109   243   106    32   112   105   110    32   107   111
100    58    32    49    50    51    52    46
```

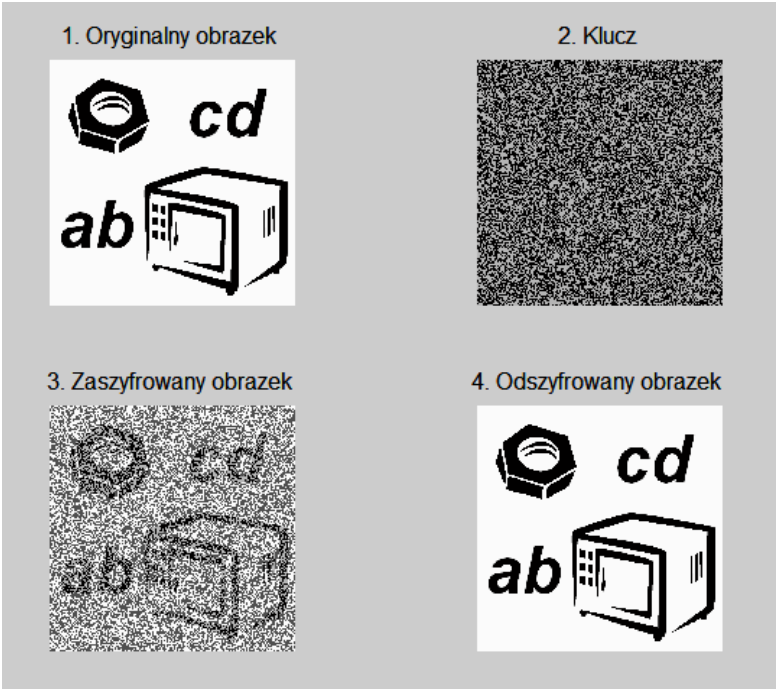
Tekst rozkodowany w postaci znaków:

```
tekst_rozkodowany = To jest mój pin kod: 1234.
```

Zadanie 1.23.

Zaszyfruj i odszyfruj obrazek „Obraz.gif”. W tym celu stwórz klucz o losowych wartościach w zakresie od 0 do 255, w rozmiarze obrazka. Za pomocą funkcji *bitxor()* dokonaj szyfrowania i deszyfrowania obrazka. Kolejne etapy zilustruj w oknie dialogowym.

Do wczytania obrazu do pamięci programu należy posłużyć się komendą *imread()* a w celu wyświetlenia go w oknie dialogowym funkcją *image()*. Funkcja *colormap('gray')* pozwoli na uzyskanie obrazka w odcieniach szarości. Do narysowania wielu obrazów w jednym oknie dialogowym wykorzystujemy komendę *subplot()*.



Rys. 1.14.

1.7. WYKRYWANIE KRAWĘDZI OBRAZU

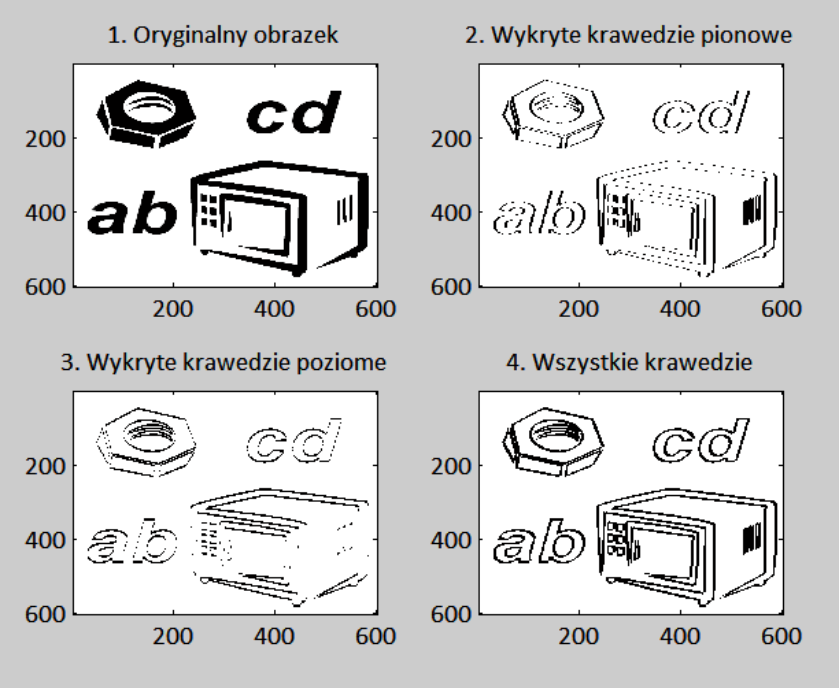
W celu odnalezienia krawędzi czarno-białego obrazu należy wykryć kolejno krawędzie w kierunku x i w kierunku y. Otrzymane w sten sposób obrazy należy w odpowiedni sposób do siebie dodać. Wykrycie krawędzi np. w kierunku x polega na przesunięciu obrazu w kierunku x o kilka (np. 5) pikseli a następnie na wykonaniu funkcji *~xor()* na tak przygotowanym obrazie i obrazie oryginalnym. Wynikiem wykonania tej operacji logicznej będą bity 0 lub 1 zgodnie z regułą ~xor. Aby otrzymać obraz w skali szarości należy przemnożyć uzyskaną macierz przez 255.

Do wczytania obrazu do pamięci programu należy posłużyć się komendą *imread()*, a w celu wyświetlenia go w oknie dialogowym *image()*. Należy pamiętać o wydaniu polecenia *colormap('gray')* aby wyświetlony obrazek był wyświetlony w odcieniach szarości. Do sumowania obrazów krawędzi x i y można posłużyć się komendą *and()*. Podobnie jak w przypadku komendy *xor()* wynikiem jest macierz bitowa. Trzeba, więc uzyskany wynik przemnożyć przez liczbę 255. Do narysowania wielu obrazów w jednym oknie dialogowym wykorzystujemy komendę *subplot()*.

Zadanie 1.24.

Wykryj krawędzie obrazka „Obraz.gif”. Postępuj wg następującego algorytmu:

1. Wczytaj obrazek „*Obraz.gif*” do macierzy „obrazek” za pomocą komendy imread().
2. Przesuń wczytany obraz w kierunku x dodając do macierzy „obrazek” 5 kolumn z lewej strony z wartościami 0. Wynika zapisz w macierzy *przesuniecie_x*.
3. Usuń 5 ostatnich kolumn z macierzy *przesuniecie_x* tak aby obrazek po przesunięciu posiadał tyle samo kolumn co oryginał.
4. Wykryj krawędzie w kierunku x wykonując operację *~xor()* na macierzach *obrazek* i *przesuniecie_x*. Wynik w postaci macierzy *krawedzie_pionowe*, przemnóż przez 255 i wyświetl w oknie dialogowym za pomocą polecenia *image()*.
5. Powtórz kroki 2, 3 i 4 w celu wykrycia krawędzi w kierunku y.
6. Dodaj macierze *krawedzie_pionowe* i *krawedzie_poziome* za pomocą komendy *and()*. Wynik wyświetl w oknie dialogowym.



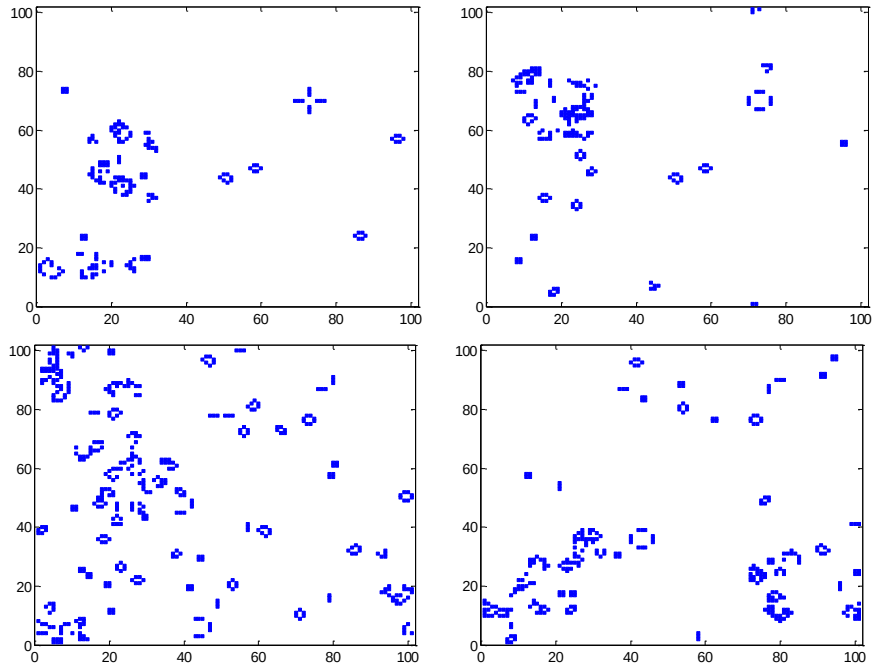
Rys. 1.15.

1.8. AUTOMAT KOMÓRKOWY I GRA ŻYCIE

Automat komórkowy to plansza podzielonej na kwadratowe komórki. Każda komórka ma ośmiu sąsiadów, czyli komórki przylegające do niej bokami i rogami. Każda komórka może znajdować się w jednym z dwóch stanów: może być albo "żywa" albo "martwa". Stany komórek zmieniają się w kolejnych cyklach ewolucji sztucznego świata. Stan wszystkich komórek w jednym cyklu jest używany do obliczenia stanu wszystkich komórek w następnym cyklu. Po obliczeniu wszystkie komórki zmieniają swój stan w tym samym momencie. Stan początkowy może zostać dowolnie skonfigurowany, np. wylosowany. Od stanu początkowego zależy, choć w sposób mało przewidywalny, dalsza ewolucja całego układu.

W grze „Życie” zaproponowanej przez Conway’a ewolucja automatu komórkowego przebiega według następujących reguł:

1. Martwa komórka, która ma dokładnie 3 żywych sąsiadów, staje się żywa w następnym cyklu.
2. Żywa komórka z 2 lub 3 żywymi sąsiadami pozostaje nadal żywa.
3. Żywa komórka z mniej niż 2 sąsiadami umiera z "samotności".
4. Żywa komórka z większą niż 3 sąsiadami umiera z "zatłoczenia".



Rys. 1.16.

Zadanie 1.25.

Zaprogramuj własną grę „Życie”:

1. Utwórz pustą, wypełnioną zerami, macierz „*Zycie*” o rozmiarach planszy.
2. Utwórz macierz pomocniczą „*pomZycie*” o takich samych rozmiarach.
3. Zapełnij przypadkowo macierz *Zycie* i wyświetl ją.
4. Rozpocznij pętlę *for* o rozmiarze liczy cykli ewolucji.
5. Dla każdej komórki w macierzy „*Zycie*” zastosuj reguły gry. Wyniki zapisuj kolejno w komórkach macierzy „pomZycie”.
6. Po zakończeniu stosowania reguł, skopiuj wyniki z „*pomZycie*” do „*Zycie*” i wyświetl macierz „*Zycie*” na wykresie. Do plotowania wygodnie będzie zastosować funkcję plot z parametrami *plot(i , j, '.', 'Marker', 's', 'MarkerFaceColor', 'b', 'MarkerSize', 3)* gdzie współrzędne *i* oraz *j* znajdujemy w następujący sposób *[i,j] = find(Zycie)*.
7. Zakończ pętle.

Spróbuj zidentyfikować charakterystyczne struktury do których ewoluuje gra „Życie”. Są to:

- struktury stale, niezmienne w trakcie gry układy komórek (chyba, że spotkają się z innymi wędrującymi po planszy strukturami),
- oscylatory – struktury, które zmieniają swą postać okresowo, np. wiatraczki w kształcie kreseczek,
- statki – struktury, które przesuwają się po planszy ruchem prostoliniowym,
- działa to oscylatory, które co jeden okres wyrzucają z siebie jeden statek.

Zadanie 1.26.

1. Zmodyfikuj powyższy kod stosując do wyświetlania wykresu technikę pobierania uchwytu wykresu i jego aktualizacji za pomocą polecenia *set()*.

1.9. FRAKTALE

Fraktal oznacza obiekt samopodobny, co oznacza, że jego części są podobne do całości. Cechą charakterystyczna fraktala jest to, że nie daje się łatwo opisać w języku tradycyjnej geometrii euklidesowej, natomiast ma względnie prostą definicję rekurencyjną.

Zadanie 1.27.

Narysuj fraktal zwany zbiorem Mandelbrota, kierując się następującymi zaleceniami:

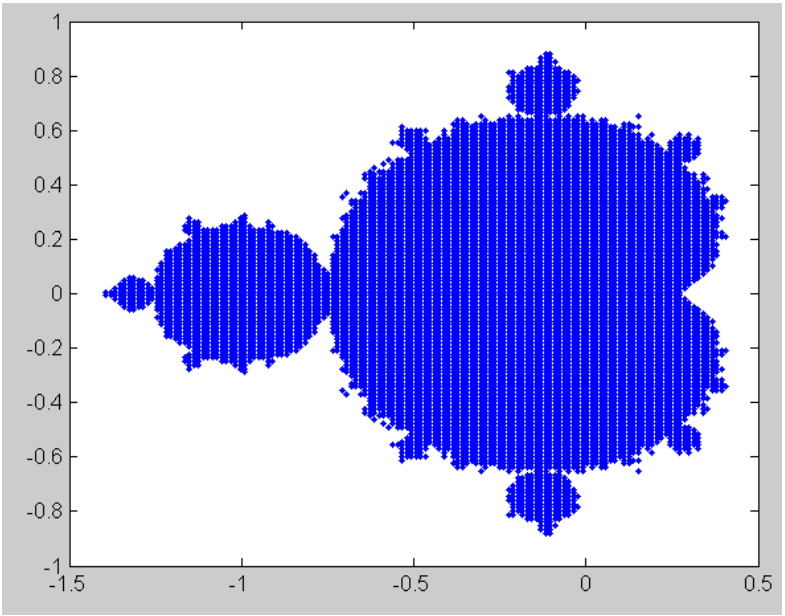
1. Dla każdego punktu C o współrzędnych w zakresie $C_x = \langle -1; 1 \rangle$ i $C_y = \langle -1.4; 0.5 \rangle$, wyliczaj zależności rekurencyjne:

   $$x_i = {x_{i-1}}^2 - {y_{i-1}}^2 + C_x$$
   $$y_i = 2 x_{i-1} y_{i-1} + C_y$$

   każdorazowo zaczynając od $x_i = 0, y_i = 0$. Zakresy C_x oraz C_y podziel na 100.
2. Jeśli dla punku w ciągu 100 iteracji spełniony jest warunek:

   $${x_i}^2 + {y_i}^2 \leq 2$$

   tzn. że punkt należy do zbioru Mandelbrota i wówczas zapamiętaj jego współrzędne C_x i C_y.
3. Utwórz wykres rysując niebieskie punkty dla wszystkich zapamiętanych punktów.



Rys. 1.17.

Zadanie 1.28.

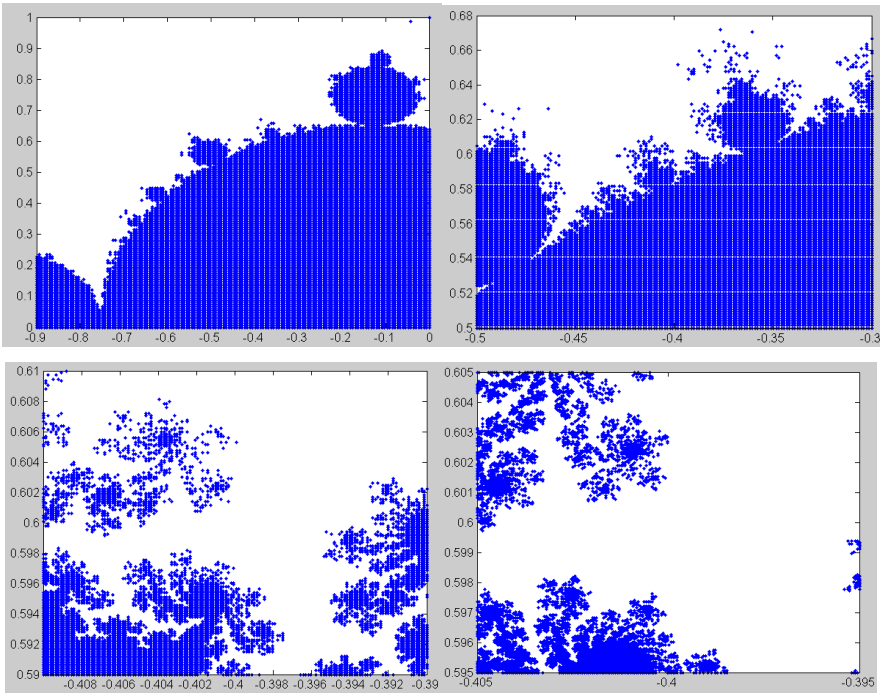
Zmieniając zakres zmiennych C_x, C_y wykonuj kolejne powiększenia:

$$C_x = \langle -0.9; 0 \rangle, C_y = \langle 0; 1 \rangle$$

$$C_x = \langle -0.5; -0.3 \rangle, C_y = \langle 0.5; 0.68 \rangle$$

$$C_x = \langle -0.41; -0.39 \rangle, C_y = \langle 0.59; 0.61 \rangle$$

$$C_x = \langle -0.405; -0.395 \rangle, C_y = \langle .595; 0.605 \rangle$$



Ćwiczenie 1.6.

Wyrażenia algebraiczne przedstawione powyżej wynikają z rachunku liczb zespolonych. Zbiór Mandelbrota jest bowiem podzbiorem płaszczyzny zespolonej i tworzą go te punkty na płaszczyźnie liczb zespolonych, dla których ciąg opisany jest równaniem rekurencyjnym:

$$\begin{cases} z_0 = 0 \\ z_{n+1} = z_n^2 + p \end{cases}$$

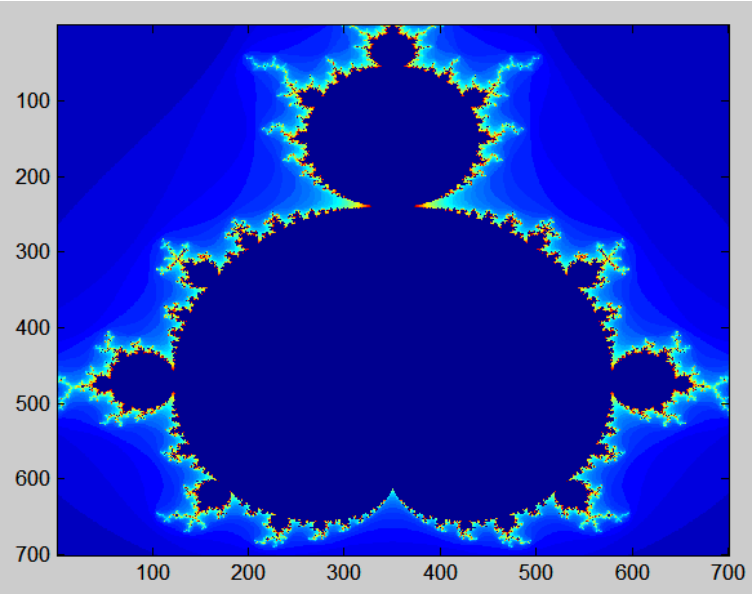
spełnia zależność:

$$\forall_{n \leq m} |z_n| < 2$$

Punkty, które tej zależności nie spełniają, czyli te których moduły dla wartości n mniejszej lub równej m mają wartość większą lub równą 2, leżą poza zbiorem Mandelbrota. Punktom tym można przypisać kolor równy liczbie iteracji n. Im szybciej z dąży do nieskończoności tym wyższy (cieplejszy) kolor.

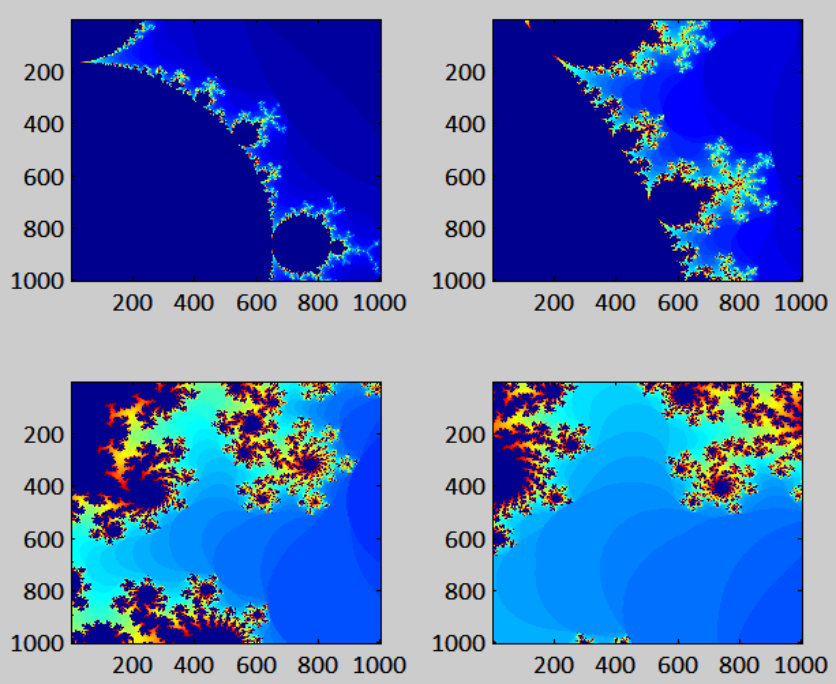
Stosując działania na liczbach zespolonych można procedurę rysującą zbiór Mandelbrota zapisać następująco:

```
clear all; clc;
rozdz    = 700;
zakres_x  = linspace(-1.4,0.5,rozdz);
zakres_y  = linspace(-1,1,rozdz);
obraz    = zeros(rozdz);
m        = 50;
 for Cx = zakres_x
  for Cy = zakres_y
    z = 0;
    C = Cx + i*Cy;
    n = 0;
    while (norm(z) < 2) && (n < m)
            z = z^2 + C;
            n = n + 1;
    end
    obraz(find(Cx==zakres_x),find(Cy==zakres_y)) =  n;
  end
end
 imagesc(obraz)
```



Rys. 1.18.

Wykonując kolejne powiększenia uzyskujemy.



Rys. 1.19.

2. ŹRÓDŁA

[1] Buchanan, W., Sieci komputerowe, WKiŁ, Warszawa 1999.
[2] Metzger, P., Jełowicki, A., Anatomia PC Wydanie X, Helion, 2006
[3] Thayer, R., Visual Basic 6 – księga eksperta, Helion, 1999.
[4] Serwis edukacyjny, Historia komputerów, http://edu.i-lo.tarnow.pl/inf/hist.php
[5] Systemy operacyjne, http://www.mimuw.edu.pl/~kubica/sop/index.html
[6] Programowanie w Visual Basic, www.e-programme.info