Pracownia nr 3

7.11.2009

1 Zadanie : Przybliżanie liczby π

Obliczymy przybliżoną wartość liczby π. Skorzystamy w tym celu z następującego wzoru na pole koła

$$p = \pi r^2$$

Biorąc zbiór punktów o współrzędnych całkowitych z kwadratu $[-r, r] \times [-r, r]$, dla każdego punktu możemy sprawdzić, czy leży on w kole o środku w punkcie $(0, 0)$ i promieniu r. Liczba takich punktów przybliża pole p koła wpisanego w kwadrat. Zatem przybliżoną wartość liczby π możemy uzyskać ze wzoru

$$\pi = \frac{p}{r^2}$$

1.1 Pierwsze przybliżenie $r = 2$

1. Tworzymy zbiór punktów w kwadracie $[-r, r] \times [-r, r]$. W tym celu utworzymy dwie tablice **x** i **y** zawierające wartości odpowiednich współrzędnych punktów w kwadracie. Proszę zwrócić uwagę na zastosowanie operatora '**:**' oraz '**'**

   ```
   >> r = 2;
   >> x = [-r : r]
   >> y = -x'
   ```

   Operator '**'** to tak zwana *transpozycja* - ustawia tablicę **x** "na głowie".

2. Następnie tworzymy macierze - czyli tablice tablic - **X** i **Y** odpowiadające zbiorom współrzędnych x i y punktów kwadratu. Współrzędnym każdego punktu kwadratu odpowiada para elementów znajdujących się na tym samych miejscach w macierzach.

   ```
   >> X = [x;x;x;x;x]
   >> Y = [y,y,y,y,y]
   ```

3. Zliczamy punkty leżące w kole o promieniu r. Liczbę tych punktów zapiszmy w **p**. W tym celu musimy sprawdzić dla każdego punktu o współrzędnych (x, y), czy $\sqrt{x^2 + y^2} \leq r$. Możemy to zrobić w następujący sposób:

   ```
   >> sqrt( X.^2 + Y.^2) <= r
   ```

4. O co chodzi? Dla każdego elementu macierzy **X** i odpowiadającemu mu elementowi macierzy **Y** sprawdzamy warunek napisany w poprzednim punkcie. Używamy funkcji **sqrt** (od angielskiego *square root*) obliczającej pierwiastek kwadratowy. Potrafi ona działać na całych tablicach, a nawet macierzach. Aby wykonać operację podniesienia do potęgi '**^**', dla każdego elementu macierzy po kolei musimy użyć operatora '**.**'.

5. Napisana przez nas instrukcja jest zdaniem logicznym. W wyniku otrzymujemy macierz zer i jedynek - wartości zdania logicznego dla każdego punktu o współrzędnych (x, y). Zero oznacza zdanie fałszywe - punkt nie leży w kole, jedynka prawdziwe.

6. Liczymy punkty leżące w kole. Wystraczy tylko policzyć ile jest jedynek w wynikowej macierzy. Służy do tego funkcja **nnz** (od angielskiego *number of non zeros*):

```
>> p = nnz(sqrt( X.^2 + Y.^2) <= r);
```

7. Sprawdźmy, ile wynosi przybliżone pole koła.

```
>> p
```

8. Obliczamy przybliżoną wartość π zgodnie ze wzorem $\pi \approx \frac{p}{r^2}$.

```
>> moje_pi = p / (r^2)
```

1.2 Kolejne przybliżenia

1. Bazując na wykonanych już operacjach spróbujmy poprawić przybliżenie naszych obliczeń. Zapiszmy je w pliku **przybliz_pi.m**. Stworzymy teraz funkcję, za pomocą której obliczymy kolejne przybliżenia ($r = 3, 4, ...$) π według naszego schematu. Funkcja będzie nazywać się **przybliz_pi**, przyjmować jeden argument r i zwracać liczbę zapisaną w zmiennej **moje_pi**.

```
function moje_pi = przybliz_pi(r)
```

2. Musimy teraz zmodyfikować nasz algorytm aby działał dla dowolnych wartości **r**. W tym celu usuńmy linię:

```
r = 2;
```

oraz zmodyfikujmy definicje macierzy **X** i **Y**:

```
X = ones(2*r + 1);
Y = ones(2*r + 1);
for i = 1 : 2*r + 1
   X(i, :) = x;
   Y(:, i) = y;
end
```

Zakończmy wszystkie linie, poza pętlą, średnikiem aby funkcja nie wypisywała niepotrzebnie wartości zmiennych oraz zakończmy kod funkcji słowem kluczowym **end**.

1.3 Ilustracja graficzna

1. Zobaczmy na rysunku jak pracuje nasza funkcja. Chcemy zobaczyć jak wygląda zbiór punktów, którego elementy zliczamy. Wprowadzimy jedną dodatkową instrukcję w naszej fukcji, w ostatniej linii, przed słowem **end**:

```
spy(sqrt( X.^2 + Y.^2) <= r)
```

Zobaczmy jak zmienia się rysunek, gdy zmnieniamy w funkcji parametr **r**.

1.4 Jak dobrze umiemy przybliżać?

1. Sprawdźmy teraz jak dobrze przybliża nasza funkcja. Wypiszemy wartości kolejnych przybliżeń policzonych za pomocą naszej funkcji oraz błąd przybliżenia równy:

$$|\mathbf{moje\_pi} - \pi|$$

2. Utwórzmy tablice **r**, w którym zapiszemy wartości r, dla których chcemy wykonać obliczenia.

```
r = [2 3 4 5 10 20 100 1000];
```

3. Wykonajmy teraz w pętli naszą funkcję dla kolejnych wartości r.

```
>> for k = 1 : size(r,2)
     disp( [k, przybliz_pi(k), abs(przybliz_pi(k)-pi)] )
 end
```

4. Wyniki wydają się być mało dokładne. To nie prawda, musimy tylko zwiększyć dokładność wypisywania wyników. Napiszmy polecenie:

```
>> format long
```

   i powtórzmy wykonanie pętli z poprzedniego punktu. Jaką różnicę możemy zaobserwować?

5. Zastanów się jak narysować dokładność (czyli wielkość błędu) kolejnych przybliżeń na wykresie.

2 Zadanie : Pole koła

Obliczymy pole koła w inny sposób niż w poprzednim zadaniu. Rozważmy n punktów na okręgu o promieniu 1. Łącząc punkty łamaną otrzymamy pewien wielokąt. Im większa wartość n, tym więcej punktów i tym dokładniej wielokąt wypełnia koło ograniczone naszym okręgiem. Zatem pole wielokąta jest przybliżeniem pola koła.

2.1 Konstrukcja wielokąta

1. Jak można opisać punkt leżący na okręgu? Jego współędne x, y spełniają równania:

$$x = r \cdot cos(\alpha) \quad y = r \cdot sin(\alpha)$$

   dla pewnego kąta α o wartości od 0 do 360 stopni i promienia r (w naszym przypadku $r = 1$).

2. Stworzymy zbiór n punktów leżących na okręgu. W tym celu utworzymy ciąg liczb, które będą odpowiadać różnym wartościom kąta α dla różnych punktów. Czyli $a_1 = \alpha_1, a_2 = \alpha_2, ...a_n = \alpha_n$.

3. Chcemy aby punkty na okręgu były położone równomiernie. Dlatego różnice pomiędzy kolejnymi wyrazmi ciągu (a_n) muszą być równe. Czyli będzie to ciąg arytmetyczny. Do stworzenia ciągu arytmetycznego możemy użyć funkcji napisanej podczas pierwszej pracowni.

4. Aby użyć funkcji **f** musimy znać jej parametry: numer elementu **n**, różnicę ciągu **r** oraz pierwszy wyraz **a1**.

5. Chcemy, aby $a_n = 360$ i $a_1 = 0$. Jak przy danym n wyznać r? Prosto:

$$a_n = a_1 + (n-1) \cdot r \quad \text{czyli} \quad r = \frac{a_n - a_1}{n-1}$$

6. Nasz program będzie miał postać funkcji, o nazwie **pole_kola** i parametrze **n**:

```
function p = pole_kola(n)
```

7. Na początku musimy wyznaczyć wartość różnicy ciągu zgodnie z wcześniejszym wzorem:

```
r = (360-0)/(n-1);
```

8. Następnie używamy funkcji **f** do utworzenia tablicy **a** zawierającej pierwsze n wyrazów ciągu arytmetycznego:

```
a = f(1:n,r,0);
```

9. Używając tablicy **a** możemy utworzyć ciągi liczb odpowiadające współrzędnym x, y kolejnych punktów na okręgu, zapiszemy je odpowiednio w tablicach **x** i **y**:

```
x = cosd(a);
y = sind(a);
```

10. Możemy teraz narysować punktyi połączone odcinkami:

```
plot(x,y,'+-')
```

11. Zakończy funkcję słowem **end** i zapiszmy plik zawierający funckję. Zobaczmy jak działa nasza nowa funkcja:

```
pole_kola(11)
```

2.2 Obliczenie pola wielokąta

Chcemy teraz uzupełnić naszą funkcję tak, aby obliczać pole wielokąta wyznaczonego przez punkty z okręgu. Zauważmy, że pole wielokąta jest sumą pól $n-1$ trójkątów o dwóch wierzchołkach w kolejnych punktach z okręgu i trzecim w środku okręgu.

Znamy wartość kąta leżącego przy wierzchołku w środku okręgu - to wyliczone przez nas wcześniej r. Z konstrukcji ciągu (a_n) wynika równość wszystkich tych kątów. Z kolei długość ramion tego kąta jest równa 1 ponieważ jest ona równa promieniowi okręgu. Zatem pola wszystkich trójkątów są równe. Aby policzyć pole wielokąta musimy tylko pomnożyć pole trójkąta przez liczbę trójkątow.

1. Pole trójkąta możemy policzyć ze wzoru:

$$p = \frac{1}{2} \cdot ab \cdot sin(\alpha)$$

dla a i b oznaczających długości boków wychodzących z wierzchołka o kącie α (w naszym przypadku $a = 1, b = 1$). Zatem pole wielokąta jest równe:

$$p = (n-1) \cdot \frac{1}{2} \cdot sin(r)$$

2. Uzupełniamy funkcję wpisując przed końcem linię:

```
p = (n - 1)*(1/2)*sind(r);
```

3. Przyjrzyjmy się jak wyglądają wyniki dla różnych wartości **n**:

```
>> for n = 10:10:100
 pole_kola(n)
 drawnow
 pause
end
```