## Łukasz Mielewczyk 224701

## Maciej Posłuszny 224574

## Michał Tyczyński 224596

# 

# Zadanie 1.5

Opis problemu

układ:

zadanie: Zbadać eksperymentalnie metodę Newtona dla powyższego układu równań, tj

wykonywać program dla różnych punktów startowych i dla zadanej przez użytkownika ilości wykonanych iteracji.

**Metoda Newtona dla układów równań**

Metoda Newtona dla układów równań polega na znalezieniu takiej wartości dla układu z funkcją (gdzie jest otarty podzbiorem przestrzeni). Metoda polega na wyborze punktu startowego , a następnie rekurencyjnym przekształceniu tego wektora do momentu kiedy otrzymamy satysfakcjonujący nas wynik. Wektory przekształcane są zgodnie z równaniem:

Zakładamy, że:

jest klasy .

Pochodna jest macierzą a jej odwrotną.

Ciąg Newtona będzie poprawnie zdefiniowany, gdy na każdym kroku macierz będzie odwracalna

Wektor można wyznaczyć jako rozwiązanie liniowego układu

**Zastosowanie teorii do zadania**

Dla układu równańwykonamy krok Newtona z, czyli musimy policzyć

Zapiszmy nasz układ równań w postaci macierzy

Następnie wyliczmy macierz pochodnych

Wyliczamy wartości macierzy oraz dla punktu startowego

Następnie musimy wyliczyć macierz odwrotną z

Teraz mamy już wszystkie potrzebne dane do wyliczenia wyniku pierwszej iteracji Podstawiamy więc dane do wzoru:

czyli

***Odp*.** Wynikiem pierwszej iteracji jest wektor:

Pseudokod

***Wejście****: tablica danych (punkt startowy) p[], ilość wykonywanych iteracji n*

***Uwaga****: pracujemy z bazą przestrzeni* , *zatem tablica p[] ma rozmiar 3 i przechowuje współrzędne wyniku kolejnych iteracji (początkowo punktu startowego który jest wprowadzany przy wejściu)*

***Uwaga****: tablice są indeksowana od 0*

*dajMacierz(p)*

*macierz[0] → (p[0]\*p[0]) + (p[1]\*p[1]) - (p[2]\*p[2]) - 1*

*macierz[1] → ((p[0]\*p[0])/4) + ((p[1]\*p[1])/9) + ((p[2]\*p[2])/4) - 1*

*macierz[2] → (p[0]\*p[0]) + (p[1]\*p[1] - 2\*p[1] + 1) - 1*

*zwróć macierz*

*dajMacierzPochodnych(p)*

*macierz[0][0] → 2\*p[0];*

*macierz[0][1] → 2\*p[1];*

*macierz[0][2] → -2\*p[2];*

*macierz[1][0] → p[0]/2;*

*macierz[1][1] → 2\*p[1]/9;*

*macierz[1][2] → p[2]/2;*

*macierz[2][0] → 2\*p[0];*

*macierz[2][1] → 2\*p[1] - 2;*

*macierz[2][2] → 0;*

*zwróć macierz*

*odwroc(macierz)*

*wyznacznik → (((macierz[0][0]\*macierz[1][1]\*macierz[2][2]) +*

*(macierz[0][1]\*macierz[1][2]\*macierz[2][0]) +*

*(macierz[0][2]\*macierz[1][0]\*macierz[2][1])) -*

*((macierz[0][2]\*macierz[1][1]\*macierz[2][0]) +*

*(macierz[0][0]\*macierz[1][2]\*macierz[2][1]) +*

*(macierz[0][1]\*macierz[1][0]\*macierz[2][2])))*

*odwrot[0][0] → (macierz[1][1]\*macierz[2][2]) - (macierz[1][2]\*macierz[2][1])*

*odwrot[0][1] → -((macierz[1][0]\*macierz[2][2]) - (macierz[1][2]\*macierz[2][0]))*

*odwrot[0][2] → (macierz[1][0]\*macierz[2][1]) - (macierz[1][1]\*macierz[2][0])*

*odwrot[1][0] → -((macierz[0][1]\*macierz[2][2]) - (macierz[0][2]\*macierz[2][1]))*

*odwrot[1][1] → (macierz[0][0]\*macierz[2][2]) - (macierz[0][2]\*macierz[2][0])*

*odwrot[1][2] → -((macierz[0][0]\*macierz[2][1]) - (macierz[0][1]\*macierz[2][0]))*

*odwrot[2][0] → (macierz[0][1]\*macierz[1][2]) - (macierz[0][2]\*macierz[1][1])*

*odwrot[2][1] → -((macierz[0][0]\*macierz[1][2]) - (macierz[0][2]\*macierz[1][0]))*

*odwrot[2][2] → (macierz[0][0]\*macierz[1][1]) - (macierz[0][1]\*macierz[1][0])*

*odwrot → transpozycja(odwrot)*

*i = 0*

*dopóki i < 3*

*j →0*

*dopóki j < 3*

*odwrot [i][j] = odwrot [i][j] \* (1/wyznacznik)*

*j → j +1*

*i → i +1*

*zwróć odwrot*

*transpozycja(macierz)*

*dopóki i < 3*

*j →0*

*dopóki j < 3*

*t [j][i] → macierz[i][j]*

*j → j +1*

*i → i +1*

*zwróć t*

*wylicz(p, f, fp)*

*i → 0*

*dopóki i < 3*

*wynik[i] → 0*

*i → i + 1*

*i → 0*

*dopóki i < 3*

*j → 0*

*dopóki j < 3*

*wynik[j] → wynik[j]+(fp[i][j] \* f[j])*

*j → j +1*

*i → i + 1*

*i → 0*

*dopóki i < 3*

*wynik[i] → p[i] - wynik[i]*

*i → i + 1*

*zwróć wynik*

*wykonajIteracje(p, n)*

*i → 0*

*dopóki i < n*

*f → dajMacierz(p)*

*fp → dajMacierzPochodnych(p)*

*fp → odwroc(fp)*

*p → wylicz(p, f, fp)*

*i → i+1*

*zwróć p*

Opis Programu

|  |
| --- |
| public void pobierzDane() {    p = new Double[3];  Scanner scanner = new Scanner(System.in);  try  {  System.out.println("Podaj punkt startowe:");  System.out.println("x0: ");  p[0] = Double.parseDouble(scanner.next());  System.out.println("y0: ");  p[1] = Double.parseDouble(scanner.next());  System.out.println("z0: ");  p[2] = Double.parseDouble(scanner.next());  System.out.println("Podaj ilosc iteracji: ");  n = Integer.parseInt(scanner.next());  }  catch(Exception e)  {  System.out.println("Blad");  }  scanner.close();  } |

Funkcja *pobierzDane()*  odpowiedzialna jest za pobranie danych startowych na wejście do programu.

Pobiera ona wartości punktu startowego - *{ x0,y0,z0}* oraz liczbę iteracji – *n*

|  |
| --- |
| public Double[] dajMacierz()  {  Double macierz[] = new Double[3];  macierz[0] = (p[0]\*p[0]) + (p[1]\*p[1]) - (p[2]\*p[2]) - 1;  macierz[1] = ((p[0]\*p[0])/4) + ((p[1]\*p[1])/9) + ((p[2]\*p[2])/4) - 1;  macierz[2] = (p[0]\*p[0]) + (p[1]\*p[1] - 2\*p[1] + 1) - 1;  return macierz;  } |

Funkcja dajMacierz() odpowiedzialna jest za wyliczenie macierzy z otrzymanych poprzednio wartości punktu startowego oraz układu równań danego w zadaniu. Rezultatem wykonania funkcji jest zwrócona macierz o wymiarach 1 x k , gdzie k jest liczbą równań w naszym układzie równań.

|  |
| --- |
| public Double[][] dajMacierzPochodnych()  {  Double macierz[][] = new Double[3][3];  macierz[0][0] = 2\*p[0];  macierz[0][1] = 2\*p[1];  macierz[0][2] = -2\*p[2];  macierz[1][0] = p[0]/2;  macierz[1][1] = 2\*p[1]/9;  macierz[1][2] = p[2]/2;  macierz[2][0] = 2\*p[0];  macierz[2][1] = 2\*p[1] - 2;  macierz[2][2] = (double)0;  return macierz;  } |

Funkcja dajMacierzPochodnych() odpowiedzialna jest za wyliczenie macierzy pochodnych z otrzymanych wartości punktu startowego. Funkcja ta zwraca macierz pochodnych o wymiarach 3x3

|  |
| --- |
| public Double[][] odwroc(Double[][] macierz)  {  Double[][] odwrot = new Double [3][3];  double wyznacznik=  (((macierz[0][0]\*macierz[1][1]\*macierz[2][2]) +  (macierz[0][1]\*macierz[1][2]\*macierz[2][0]) +  (macierz[0][2]\*macierz[1][0]\*macierz[2][1])) -  ((macierz[0][2]\*macierz[1][1]\*macierz[2][0]) +  (macierz[0][0]\*macierz[1][2]\*macierz[2][1]) +  (macierz[0][1]\*macierz[1][0]\*macierz[2][2])));  odwrot[0][0] = (macierz[1][1]\*macierz[2][2]) - (macierz[1][2]\*macierz[2][1]);  odwrot[0][1] = -((macierz[1][0]\*macierz[2][2]) - (macierz[1][2]\*macierz[2][0]));  odwrot[0][2] = (macierz[1][0]\*macierz[2][1]) - (macierz[1][1]\*macierz[2][0]);  odwrot[1][0] = -((macierz[0][1]\*macierz[2][2]) - (macierz[0][2]\*macierz[2][1]));  odwrot[1][1] = (macierz[0][0]\*macierz[2][2]) - (macierz[0][2]\*macierz[2][0]);  odwrot[1][2] = -((macierz[0][0]\*macierz[2][1]) - (macierz[0][1]\*macierz[2][0]));  odwrot[2][0] = (macierz[0][1]\*macierz[1][2]) - (macierz[0][2]\*macierz[1][1]);  odwrot[2][1] = -((macierz[0][0]\*macierz[1][2]) - (macierz[0][2]\*macierz[1][0]));  odwrot[2][2] = (macierz[0][0]\*macierz[1][1]) - (macierz[0][1]\*macierz[1][0]);  odwrot = transpozycja(odwrot);  for(int i = 0; i < 3; i++)  for(int j = 0; j < 3; j++)  odwrot[i][j] \*= (1/wyznacznik);  return odwrot;  } |

Funkcja odwroc() odpowiedzialna jest za odwracanie macierzy której wartość otrzymuję przy wywołaniu w parametrze – odwroc(Double[][] macierz). Najpierw wyliczany jest wyznacznik, a następnie poszczególne indeksy odwróconej już macierzy. Na koniec indeksy macierzy mnożone są w pętli przez wyznacznik. Otrzymana macierz jest zwracana przez funkcję w postaci zmiennej *odwrot*.

|  |
| --- |
| public Double[][] transpozycja(Double[][] macierz)  {  Double[][] transpozycja = new Double [3][3];  for(int i = 0; i < 3; i++)  for(int j = 0; j < 3; j++)  transpozycja[j][i] = macierz[i][j];  return transpozycja;  } |

Funkcja transpozycja() transponuję otrzymaną w parametrze macierz, czyli zamienia jej rzędy na kolumny.

|  |
| --- |
| public Double[] wylicz(Double[] f, Double[][] fp)  {  Double wynik[] = new Double[3];  for(int i = 0; i < 3; i++)  wynik[i] = (double)0;  for(int i = 0; i < 3; i++)  for(int j = 0; j < 3; j++)  wynik[i] += (fp[i][j] \* f[j]);  for(int i = 0; i < 3; i++)  wynik[i] = p[i] - wynik[i];  return wynik;  } |

Funkcja wylicz() odpowiada za obliczenie kolejnego punktu przybliżenia miejsca zerowego . Najpierw obliczamy iloczyn macierzy pochodnych oraz macierzy otrzymanej z naszego równania, a następnie otrzymaną macierz odejmujemy od wartości naszego punktu startowego, co w wyniku daje nam kolejny wektor przybliżenia miejsca zerowego.

|  |
| --- |
| public void wykonajIteracje()  {  Double f[];  Double fp[][];  for(int i = 0; i < n; i++)  {  f = dajMacierz();  fp = dajMacierzPochodnych();  fp = odwroc(fp);  p = wylicz(f, fp);  }  } |

Funkcja wykonajIteracje() odpowiada za poszczególne wywołania funkcji, by otrzymać wektor przybliżenia miejsca zerowego tyle razy ile podaliśmy na wejściu programu. W wyniku daje nam to większe przybliżenie poszukiwanego miejsca zerowego.