Marcin Rociek
Informatyka, II rok

Metody Obliczeniowe w Nauce i Technice

laboratorium

zestaw 7: Równania nieliniowe i układy równań nieliniowych

Zadanie 1: Stosując metodę Newtona oraz metodę siecznych wyznaczyć pierwiastek dodatni równania $2x^2+1=2^x$. Porównać liczbę operacji dla obu tych metod.

Szacowanie miejsca zerowego metodą Newtona polega na obliczaniu kolejnych wyrazów ciągu:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)},$$

rozpoczynając od jakiegoś x_0.

W metodzie siecznych natomiast korzysta się z wzoru:

$$x_{n+1} = x_n - \frac{f(x_n)(x_n - x_{n-1})}{f(x_n) - f(x_{n-1})}.$$

Obie te metody są realizowane przez odpowiednie funkcje z biblioteki *Numerical Recipes*, tzn.: **rtnewt** dla metody Newtona oraz **rtsec** dla metody siecznych.

Treść programu, który wykorzystuje te funkcje w celu poszukiwania dodatniego pierwiastka danej funkcji:

```
#include <math.h>
#include "stdio.h"

extern double rtnewt(void (*funcd)(double, double *, double *), double x1, double x2, double
xacc);
extern double rtsec(double (*func)(double), double x1, double x2, double xacc);

double func(double x)
{
  return 2*x*x+1.0-pow(2.0, x);
}

void funcd(double x, double *y, double *yy)
{
  *y = func(x);
  *yy = 4.0*x-pow(2.0, x)*log(2.0);
}

void main(void)
{
  double eps = 0.0001;
  double x;

  printf("Przyjety blad: %f\n\n", eps);
  printf("Metoda Newtona:\n");
  x = rtnewt((void*)&funcd, 6.0, 7.0, eps);
  printf("\nwynik:\n");
  printf("x=%f +/- %f, f(x)=%f\n", x, eps, func(x));
  printf("\nMetoda siecznych:\n");
  x = rtsec((void*)&func, 6.0, 7.0, eps);
  printf("\nwynik:\n");
  printf("x=%f +/- %f, f(x)=%f\n", x, eps, func(x));
}
```

Funkcje znajdujące pierwiastek zatrzymują się, gdy wyrażenie odejmowane w danej iteracji od x_n jest mniejsze od zadanego e. Zostały one również nieco zmodyfikowane tak, aby wypisywały na wyjściu wartości x_n oraz $f(x_n)$ w kolejnych iteracjach.

Wynik działania programu dla e=0,0001:

```
Przyjety blad: 0.000100
```

```
Metoda Newtona:
     x         f(x)
 1   6.500000   -5.009668
 2   6.363632   -0.354695
 3   6.352416   -0.002231

wynik:
x=6.352345 +/- 0.000100, f(x)=-0.000000

Metoda siecznych:
     x         f(x)
 1   6.000000   9.000000
 2   6.236842   3.378131
 3   6.379158   -0.849985
 4   6.350548   0.056038
 5   6.352318   0.000843

wynik:
x=6.352345 +/- 0.000100, f(x)=-0.000001
```

Dla ε=0.000001:

```
Przyjety blad: 0.000001

Metoda Newtona:
     x         f(x)
 1   6.500000   -5.009668
 2   6.363632   -0.354695
 3   6.352416   -0.002231
 4   6.352345   -0.000000

wynik:
x=6.352345 +/- 0.000001, f(x)=0.000000

Metoda siecznych:
     x         f(x)
 1   6.000000   9.000000
 2   6.236842   3.378131
 3   6.379158   -0.849985
 4   6.350548   0.056038
 5   6.352318   0.000843
 6   6.352345   -0.000001

wynik:
x=6.352345 +/- 0.000001, f(x)=0.000000
```

Jak widać, metoda Newtona daje rezultat w mniejszej ilości kroków.

Dodatnim pierwiastkiem zadanego równania jest więc punkt: *6,352345±0,000001*.

Zadanie 2: Rozwiązać metodą Newtona układ równań:

$$x_1^2+x_2^2+x_3^2=1$$
$$2x_1^2+x_2^2-4x_3=0$$
$$3x_1^2-4x_2+x_3^2=0$$

dla przybliżenia początkowego $x=[1,0,0]^T$.

Metoda Newtona rozwiązywania układów równań nieliniowych polega na obliczaniu kolejnych wyrazów ciągu:

$$x^{(i+!)} = x^{(i)} - \left[F'\left(x^{(i)}\right)\right]^{-1} F\left(x^{(i)}\right)$$

Funkcją realizującą to zadanie jest funkcja **mnewt** z biblioteki *Numerical Recipes*.

Treść programu wykorzystującego tą funkcję do realizacji zadania:

```
#include "nrutil.h"
#include "matrix.h"
#include <math.h>
#include <stdio.h>

extern void mnewt(int ntrial, double x[], int n, double tolx, double tolf);

/*
```

```
 * funkcja wyliczajaca wartosc funkcji w punkcie, oraz wartosc pochodnej
 */
void usrfun(double *x, double **alpha, double *bet)
{
  bet[1] = x[1]*x[1] +x[2]*x[2] +x[3]*x[3] -1.0;
  bet[2] = 2.0*x[1]*x[1] +x[2]*x[2] -4.0*x[3];
  bet[3] = 3.0*x[1]*x[1] -4.0*x[2]*x[2] +x[3]*x[3];
  alpha[1][1] = -2.0*x[1];
  alpha[1][2] = -2.0*x[2];
  alpha[1][3] = -2.0*x[3];
  alpha[2][1] = -4.0*x[1];
  alpha[2][2] = -2.0*x[2];
  alpha[2][3] = 4.0;
  alpha[3][1] = -6.0*x[1];
  alpha[3][2] = 4.0;
  alpha[3][3] = -2.0*x[3];
}

#define TOLX 1.0e-12
#define TOLF 1.0e-12

/*
 * glowna czesc programu
 */
void main(void)
{
  int i, ntrial = 20;
  double *x = dvector(1, 3);
  double **alpha = dmatrix(1, 3, 1, 3);
  double *bet = dvector(1, 3);

  x[1] = 1.0;
  x[2] = 0.0;
  x[3] = 0.0;

  usrfun(x, alpha, bet);
  printf("   x= ");
  vec_show(x, 3);
  printf("f(x)= ");
  vec_show(bet, 3);
  printf("\n");

  for(i=1; i<=ntrial; i++)
  {
    mnewt(1, x, 3, TOLX, TOLF);
    usrfun(x, alpha, bet);
    printf("i=%d\n   x= ", i);
    vec_show(x, 3);
    printf("f(x)= ");
    vec_show(bet, 3);
    printf("\n");
  }
  free_dvector(x, 1, 3);
  free_dvector(bet, 1, 3);
  free_dmatrix(alpha, 1, 3, 1, 3);
}
```

Poniżej znajduje się wynik działania tego programu, tj. wygląd wektora *x* oraz wartość funkcji w tym punkcie dla kolejnych iteracji:

```
x= 1.0000000000 0.0000000000 0.0000000000
f(x)= 0.0000000000 2.0000000000 3.0000000000

i=1
   x= 1.0000000000 0.7500000000 0.5000000000
f(x)= 0.8125000000 0.5625000000 1.0000000000

i=2
   x= 0.7669270833 0.6145833333 0.3567708333
f(x)= 0.0931752523 0.1269836426 0.3809661865

i=3
   x= 0.7002075187 0.6304777913 0.3422319374
f(x)= 0.0049155136 0.0091556344 -0.0020145746

i=4
   x= 0.6985056501 0.6282544817 0.3426283111
f(x)= 0.0000079966 0.0000107358 0.0023098140

i=5
   x= 0.6982594766 0.6285615302 0.3425554938
```

```
f(x)= 0.0000001602 0.0000002155 -0.0003152328

i=6
   x= 0.6982926468 0.6285191596 0.3425653928
f(x)= 0.0000000030 0.0000000040 0.0000435740

i=7
   x= 0.6982880532 0.6285250071 0.3425640237
f(x)= 0.0000000001 0.0000000001 -0.0000060121

i=8
   x= 0.6982886868 0.6285242001 0.3425642126
f(x)= 0.0000000000 0.0000000000 0.0000008297

i=9
   x= 0.6982885994 0.6285243115 0.3425641865
f(x)= 0.0000000000 0.0000000000 -0.0000001145

i=10
   x= 0.6982886114 0.6285242961 0.3425641901
f(x)= 0.0000000000 0.0000000000 0.0000000158

i=11
   x= 0.6982886098 0.6285242982 0.3425641896
f(x)= 0.0000000000 0.0000000000 -0.0000000022

i=12
   x= 0.6982886100 0.6285242979 0.3425641897
f(x)= -0.0000000000 0.0000000000 0.0000000003

i=13
   x= 0.6982886100 0.6285242980 0.3425641897
f(x)= -0.0000000000 -0.0000000000 -0.0000000000
```

W następnych iteracjach wartość wektora x nie ulega zmianie.

Rozwiązaniem jest wektor $x=[0.6982886100 \quad 0.6285242980 \quad 0.3425641897]^T$.