Metoda Newtona

Język programowania: C++
Autor: Daniel Wilczak

Państwa zadaniem jest przygotowanie zestawu funkcji obliczających punkty stałe i zera funkcji wielu zmiennych. Chociaż mają Państwo zupełną dowolność w wyborze metody, zalecaną jest metoda Newtona. Nazwy funkcji, które należy napisać oraz ich argumenty zostaną opisane poniżej.

Sposób przesyłania rozwiązania

Do systemu BaCa należy przesłać plik o nazwie **source.cpp** zawierający definicje wszystkich wymaganych w zadaniu funkcji, ale **nie zawierający funkcji main**. Programy testujące Państwa rozwiązanie będą miały następującą strukturę:

```
#include "source.cpp"
// funkcje pomocnicze testu

int main(){
  // właściwy test rozwiązania
}
```

Drukowanie wyników

Wyniki należy wypisywać z dokładnością do 17 cyfr znaczących. Możecie Państwo użyć następującej funkcji

```
void printVector(const double* x, unsigned N){
  for(unsigned i=0;i<N;++i)
    printf("%17.17f ",x[i]);
  printf("\n");
}
```

Dokładność obliczeń

Wyniki generowane przez Państwa programy powinny być zgodne z wzorcowym rozwiązaniem. Dopuszczalne odchylenia będą ustawione w następujący sposób:

- tolerancja absolutna: $\mathbf{10^{-13}}$
- tolerancja relatywna: $\mathbf{10^{-6}}$

Oczywiście wynik policzony z błędem mniejszym niż maksymalny dopuszczalny będzie również zaakceptowany i zalecam obliczanie wyników z jak najlepszą dokładnością.

Argumenty funkcyjne

W tym zadaniu argumetami wszystkich funkcji, które należy napisać, będą wskaźniki do funkcji, dla których ma działać napisany przez Państwa algorytm. Przyjmujemy zasadę, że argumenty funkcyjne będą miały zawsze sygnaturę

```
void (*f)(const double* x, double* y, double* Df);
```

od tej pory oznaczaną jako

```
typedef void (*FuncPointer)(const double* x, double* y, double* Df);
```

Funkcja ta określa pewne odwzorowanie $f:R^N \to R^M$. Argumentami są

- **x** - tablica długości `N`, określająca wektor, dla którego liczona jest wartość funkcji
- **y** - tablica długości `M`, do której funkcja wpisze obliczone `f(x)`. Funkcja **zakłada, że tablica `y` ma co najmniej M elementów!**
- **Df** - tablica długości `M*N`, do której funkcja wpisze obliczoną macierz różniczki `Df(x)`. Funkcja **zakłada, że tablica `Df` ma co najmniej N*M elementów!**

Macierze będą reprezentowane jako jednowymiarowe tablice. Wiersze macierzy będą układane w tablicy jeden za drugim. Na przykład, macierz

```
| 1 2 3 4 |
| 5 6 7 8 |
```

będzie reprezentowana jako ośmioelementowa tablica

```
1 2 3 4 5 6 7 8
```

Lista funkcji, które należy napisać.

Poniżej podana jest lista funkcji, które należy napisać. Nadesłane rozwiązanie może oczywiście zawierać dowolną liczbę funkcji pomocniczych.

1. funkcja **findCurve** o sygnaturze

   ```
   int findCurve(FuncPointer f, double* x, unsigned k, double h);
   ```

   Funkcja ta ma obliczyć i wydrukować na ekran wybrane punkty $x_1,x_2,...,x_k$ spełniające równanie $f(x_i)=(f_1(x_i),f_2(x_i))=(0,0)$ z dokładnością co najmniej $\max(|f_i(x_i)|,|f_2(x_i)|)\leq 10^{-14}$. Argumentami tej funkcji są
   - **f** - wskaźnik do funkcji $R^3 \to R^2$
   - **x** - tablica liczb double długości 3, zawierająca punkt początkowy bliski miejsca zerowego funkcji (czyli f(x)≈(0,0))
   - **k** - określa liczbę punktów do wyznaczenia
   - **h** - krok zmiany parametru. Funkcję `f` możemy traktować jako

     $$f(a,b,c) = (f_1(a,b,c),f_2(a,b,c))$$

     Przy ustalonej wartości zmiennej **c** jest to funkcja $f_c:R^2 \to R^2$, której miejsca zerowe możemy wyliczać metodą Newtona. Państwa zadaniem jest obliczenie i wydrukowanie na ekran miejsc zerowych f_c oraz wartości parametru c dla c równych kolejno

```
x[2]+h
x[2]+2*h
....
x[2]+k*h
```

Zwracana wartość: jeśli dla pewnego **i=1,...,k** nie uda się wyznaczyć zera funkcji dla `c=x[2]+i*h` i z tolerancją absolutną 10^{-14} (na przykład metoda numeryczna będzie rozbieżna), to należy zakończyć działanie funkcji bez wypisywania błędnego miejsca zerowego i zwrócić indeks **i**, dla którego nastąpił problem. W przeciwnym przypadku zwracamy wartość **0** oznaczającą brak błędu.

Test jawny:

```
  #include "source.cpp"

  void implicitCurve(const double* x, double* y, double* Df){
    // funkcja dana jest wzorem f(a,b,c) =
(1-a^2-b^2-c^2,(a+b+c)/(a^2+b^2+c^2)-1)
    // zmienne pomocnicze
    const double n = x[0]*x[0] + x[1]*x[1] + x[2]*x[2];
    const double r = 1./n;
    const double s = (x[0] + x[1] + x[2])*r;

    // obliczam wartosc funkcji
    y[0] = 1. - n;
    y[1] = s - 1.;

    //obliczam pierwszy wiersz macierzy
    Df[0] = -2.*x[0];
    Df[1] = -2.*x[1];
    Df[2] = -2.*x[2];

    //obliczam drugi wiersz macierzy
    const double r2 = 2.*s*r;
    Df[3] = r - x[0]*r2;
    Df[4] = r - x[1]*r2;
    Df[5] = r - x[2]*r2;
  }

  int main(){
    double x[3] = {0.25*(1.+sqrt(5.)),0.25*(1.-sqrt(5.)),0.5};
    findCurve(implicitCurve,x,10,1./128);
    printf("\n");
    double x2[3] = {0.25*(1.-sqrt(5.)),0.25*(1.+sqrt(5.)),0.5};
    int i = findCurve(implicitCurve,x2,10,3./32);
    printf("%d",i);
    return 0;
  }

/**
spodziewane wyjscie:
0.80332000443603468 -0.31113250443603474 0.50781250000000000
0.79753481882962218 -0.31315981882962213 0.51562500000000000
0.79166053918176227 -0.31509803918176227 0.52343750000000000
```

```
0.78569621138679224 -0.31694621138679219 0.53125000000000000
0.77964082305207405 -0.31870332305207399 0.53906250000000000
0.77349330072459810 -0.32036830072459804 0.54687500000000000
0.76725250688009694 -0.32194000688009689 0.55468750000000000
0.76091723665304600 -0.32341723665304600 0.56250000000000000
0.75448621428335538 -0.32479871428335538 0.57031250000000000
0.74795808925263751 -0.32608308925263751 0.57812500000000000

-0.32835472974046714 0.73460472974046720 0.59375000000000000
-0.33288987007807902 0.64538987007807902 0.68750000000000000
-0.31824788073137528 0.53699788073137522 0.78125000000000000
-0.27407280044590648 0.39907280044590648 0.87500000000000000
-0.15906781074217108 0.19031781074217108 0.96875000000000000
6
*/
```

---

2. funkcja **findSurface** o sygnaturze

```
  int findSurface(FuncPointer f, double* x, unsigned k1, unsigned k2,
double h1, double h2);
```

Funkcja ta ma obliczyć i wydrukować na ekran wybrane punkty $x_1, x_2, ..., x_{k1*k2}$ spełniające równanie $f(x_i)=0$ z dokładnością $|f(x_i)|\leq 10^{-14}$. Argumentami tej funkcji są

- **f** - wskaźnik do funkcji R^3 -> R
- **x** - tablica liczb double długości 3, zawierająca punkt początkowy bliski miejsca zerowego funkcji (czyli f(x)≈0)
- **k1,k2** - określają liczby punktów do wyznaczenia
- **h1,h2** - kroki zmiany parametrów. Przy ustalonej wartości ostatnich dwóch zmiennych **b,c** jest to funkcja jednej zmiennej $f_{b,c}$:R ->R, której miejsca zerowe możemy wyliczyć na przykład za pomocą metody Newtona. Państwa zadaniem jest obliczenie i wydrukowanie na ekran miejsc zerowych $f_{b,c}$ oraz wartości **b,c** dla (b,c) równych (w podanej kolejności)

```
(x[2]+h1,x[3]+h2)
(x[2]+h1,x[3]+2*h2)
...
(x[2]+h1,x[3]+k2*h2)

(x[2]+2*h1,x[3]+h2)
(x[2]+2*h1,x[3]+2*h2)
...
(x[2]+2*h1,x[3]+k2*h2)

...

(x[2]+k1*h1,x[3]+h2)
(x[2]+k1*h1,x[3]+2*h2)
 ...
(x[2]+k1*h1,x[3]+k2*h2)
```

Dodatkowo, dla większej czytelności wyjścia, po każdym zestawie punktów z ustaloną wartością b wstawiamy wolną linię.

Zwracana wartość: jeśli dla pewnego **i=1,...,k1, j=1,...,k2** nie uda się wyznaczyć zera funkcji dla parametrów `b=x[1]+i*h1, c=x[2]+j*h2` z tolerancją absolutną 10^{-14}, to należy zakończyć działanie funkcji bez wypisywania błędnego miejsca zerowego i zwrócić wartość **i*k1+j**. W przeciwnym przypadku zwracamy wartość **0** oznaczającą brak błędu.

Test jawny:

```
#include "source.cpp"

void implicitSurface(const double* x, double* y, double* Df){
  // funkcja dana jest wzorem f(a,b,c) = (a+b+c)/(a^2+b^2+c^2)-1
  // zmienne pomocnicze
  const double n = x[0]*x[0] + x[1]*x[1] + x[2]*x[2];
  const double s = x[0] + x[1] + x[2];

  // obliczam wartosc funkcji
  *y = s/n - 1.;

  //obliczam pierwszy i jedyny wiersz macierzy
  const double r = 1./n;
  const double r2 = 2.*(*y)*r;
  Df[0] = r - x[0]*r2;
  Df[1] = r - x[1]*r2;
  Df[2] = r - x[2]*r2;
}

int main(){
  double x[3] = {0.25*(1.+sqrt(5.)),0.25*(1.-sqrt(5.)),0.5};
  findSurface(implicitSurface,x,4,4,1./32,1./32);
  return 0;
}
```

```
/**
spodziewane wyjscie:
0.12039238685063169 -0.27776699437494745 0.53125000000000000
0.12427103856508417 -0.27776699437494745 0.56250000000000000
0.13082600313543147 -0.27776699437494745 0.59375000000000000
0.14020352633612365 -0.27776699437494745 0.62500000000000000

0.06212351672030310 -0.24651699437494745 0.53125000000000000
0.06548172983245609 -0.24651699437494745 0.56250000000000000
0.07113718439412973 -0.24651699437494745 0.59375000000000000
0.07918248954042481 -0.24651699437494745 0.62500000000000000

0.01274589354441671 -0.21526699437494745 0.53125000000000000
0.01576154969082753 -0.21526699437494745 0.56250000000000000
0.02082981701922552 -0.21526699437494745 0.59375000000000000
0.02801681517470467 -0.21526699437494745 0.62500000000000000

-0.03023031684944523 -0.18401699437494745 0.53125000000000000
-0.02746042638880794 -0.18401699437494745 0.56250000000000000
-0.02281133203696253 -0.18401699437494745 0.59375000000000000
-0.01623226497988585 -0.18401699437494745 0.62500000000000000
*/
```

---

3. funkcja **findFixedPoints** o sygnaturze

```
int findFixedPoints(FuncPointer f, double* x, unsigned k1, unsigned k2,
```

```
double h1, double h2);
```

W tej funkcji zakładamy, że `f:R`^{4} `->` `R`^{2}. Nazwijmy zmienne funkcji `f` kolejno `x,y,a,b`. Przy ustalonej wartości ostatnich dwóch zmiennych **a,b** jest to funkcja $f_{a,b}$`:R`^{2} `->R`^{2}, która może mieć izolowany punkt stały. Państwa zadaniem jest obliczenie i wydrukowanie wektorów `(x,y,a,b)` dla których `f(x,y,a,b) =` $f_{a,b}$`(x,y)=(x,y)` z dokładnością $|f_{a,b}(x,y)-(x,y)|\leq 10^{-14}$ w normie maximum. Argumentami funkcji `findFixedPoints` są

- **f** - wskaźnik do funkcji `R`^{4} `->` `R`^{2}
- **x** - tablica liczb double długości 4, zawierająca punkt początkowy spełniający warunek `f(x[0],x[1],x[2],x[3])≈(x[0],x[1])`
- **k1,k2** - określają liczby punktów do wyznaczenia
- **h1,h2** - kroki zmiany parametrów. Państwa zadaniem jest obliczenie i wydrukowanie na ekran punktów stałych $f_{a,b}$ wraz z wartościami `a,b` dla (a,b) kolejno równych

```
(x[2]+h1,x[3]+h2)
(x[2]+h1,x[3]+2*h2)
...
(x[2]+h1,x[3]+k2*h2)

(x[2]+2*h1,x[3]+h2)
(x[2]+2*h1,x[3]+2*h2)
...
(x[2]+2*h1,x[3]+k2*h2)

...

(x[2]+k1*h1,x[3]+h2)
(x[2]+k1*h1,x[3]+2*h2)
 ...
(x[2]+k1*h1,x[3]+k2*h2)
```

Dodatkowo, dla większej czytelności wyjścia, po każdym zestawie punktów z ustaloną wartością `b` wstawiamy wolną linię.

Zwracana wartość: jeśli dla pewnego **i=1,...,k1, j=1,...,k2** nie uda się wyznaczyć punktu stałego funkcji $f_{a,b}$ dla parametrów `a=x[2]+i*h1, b=x[3]+j*h2` z tolerancją absolutną 10^{-14}, to należy zakończyć działanie funkcji bez wypisywania błędnego punktu stałego i zwrócić wartość **i*k1+j**. W przeciwnym przypadku zwracamy wartość **0** oznaczającą brak błędu.

Test jawny:

```
#include "source.cpp"

void henon(const double* x, double* y, double* Df){
  // funkcja dana jest wzorem henon(x,y,a,b) = (1+y-a*x^2,b*x)
  const double x2 = x[0]*x[0];

  y[0] = 1 + x[1] - x[2]*x2;
  y[1] = x[3]*x[0];
```

```
    //obliczam pierwszy wiersz macierzy
    Df[0] = -2*x[2]*x[0];
    Df[1] = 1.;
    Df[2] = -x2;
    Df[3] = 0.;

    //obliczam drugi wiersz macierzy
    Df[4] = x[3];
    Df[5] = 0.;
    Df[6] = 0.;
    Df[7] = x[0];
  }

  int main(){
    double x[4] = {-1.2807764064044151, -0.6403882032022076,
1.0000000000000000, 0.5000000000000000};
    findFixedPoints(henon,x,4,4,1./16,1./16);
    return 0;
  }

/**
spodziewane wyjscie:
-1.19763031176984502 -0.67366705037053787 1.06250000000000000
0.56250000000000000
-1.16253262436707128 -0.72658289022941946 1.06250000000000000
0.62500000000000000
-1.12828395985954577 -0.77569522240343769 1.06250000000000000
0.68750000000000000
-1.09489692504918534 -0.82117269378688895 1.06250000000000000
0.75000000000000000

-1.15709574728685860 -0.65086635784885805 1.12500000000000000
0.56250000000000000
-1.12409377442300484 -0.70255860901437794 1.12500000000000000
0.62500000000000000
-1.09187315546713570 -0.75066279438365580 1.12500000000000000
0.68750000000000000
-1.06044486059083676 -0.79533364544312757 1.12500000000000000
0.75000000000000000

-1.12017996019888111 -0.63010122761187060 1.18750000000000000
0.56250000000000000
-1.08904242173442811 -0.68065151358401754 1.18750000000000000
0.62500000000000000
-1.05862710283202821 -0.72780613319701948 1.18750000000000000
0.68750000000000000
-1.02894361972548642 -0.77170771479411493 1.18750000000000000
0.75000000000000000

-1.08638630667790914 -0.61109229750632399 1.25000000000000000
0.56250000000000000
-1.05691785736085264 -0.66057366085053293 1.25000000000000000
0.62500000000000000
-1.02811959340942205 -0.70683222046897776 1.25000000000000000
0.68750000000000000
-1.00000000000000000 -0.75000000000000000 1.25000000000000000
0.75000000000000000
*/
```