Przedmiot: Metody Numeryczne, (zajęcia laboratoryjne)
Projekt: Układy równań nieliniowych, **Termin realizacji: 18 maja, 2025 r.**

Układy równań nieliniowych

Projekt dotyczy metod numerycznych rozwiązywania nieliniowych układów równań. W równaniach takich występują co najmniej dwie niewiadome oraz mogą pojawić się człony nieliniowe z tymi niewiadome takie, jak xy, x^2+y, $\sin(x)$, $1/(x^3y+\ln y)$.

Przykładem nieliniowego układu równań jest następujący układ

$$\begin{cases} x^2+y^2-2x+4y+4=0, \\ 9x^2+4y^2+16y-20=0. \end{cases} \tag{1}$$

Układ powyższy zawiera dwie niewiadome x, y.

W przypadku występowania dwóch lub trzech niewiadomych często stosujemy oznaczenia x, y, z, ale gdy niewiadomych jest więcej, to taka konwencja nie jest dobra – lepiej jest wtedy numerować zmienne $x_1, x_2, x_3, \ldots$ Przy takich oznaczeniach układ (1) zapiszemy w postaci

$$\begin{cases} x_1^2+x_2^2-2x_1+4x_2+4=0, \\ 9x_1^2+4x_2^2+16x_2-20=0. \end{cases} \tag{2}$$

W tym momencie nie jesteśmy jeszcze w stanie rozwiązać numerycznie układu (2), gdyż dotychczas omówione metody dotyczyły albo pojedynczych równań z *jedną* niewiadomą albo układów równań *liniowych*. Dlatego do przykładu tego wrócimy, gdy już zapoznamy się z odpowiednimi metodami.

Ogólna postać układu nieliniowego.

Równania takie jak (2) możemy opisać następująco. Dane są funkcje wielu zmiennych $f_1:\mathbb{R}^n\to\mathbb{R},\ldots f_n:\mathbb{R}^n\to\mathbb{R}$. Liczba funkcji jest równa n tak samo jak liczba zmiennych $x_1,\ldots,x_n$. Tak więc $f_1=f_1(x_1,\ldots,x_n)$, $f_2=f_2(x_1,\ldots,x_n)$ itd. Rozwiązaniem układu równań

$$\begin{cases} f_1(x_1,\ldots,x_n)=0, \\ \vdots \\ f_n(x_1,\ldots,x_n)=0, \end{cases} \tag{3}$$

jest n liczb $x_n^*,\ldots,x_n^*\in\mathbb{R}$, takich że po podstawieniu do (3) uzyskamy równości, czyli

$$\begin{cases} f_1(x_1^*,\ldots,x_n^*)=0, \\ \vdots \\ f_n(x_1^*,\ldots,x_n^*)=0. \end{cases} \tag{4}$$

Tak sformułowany problem można zapisać w sposób jeszcze bardziej zwarty wprowadzając funkcje o wartościach wektorowych, czyli zestawienie funkcji $f(x)=(f_1(x),\ldots,f_n(x))$, gdzie $x=(x_1,\ldots,x_n)$. Tak więc dla danej funkcji wielu zmiennych o wartościach wektorowych

$$f:\mathbb{R}^n\to\mathbb{R}^n, \tag{5}$$

szukamy $x^*\in\mathbb{R}^n$ takiego, że

$$f(x^*)=0. \tag{6}$$

Wracając do naszego przykładu układu równań wyrażonego w (2) możemy omówione funkcje zapisać tak $(n=2)$. Funkcje $f_1,\ f_2 : \mathbb{R}^2 \to \mathbb{R}$

$$\begin{aligned} f_1(x_1,x_2) &= x_1^2 + x_2^2 - 2x_1 + 4x_2 + 4, \\ f_2(x_1,x_2) &= 9x_1^2 + 4x_2^2 + 16x_2 - 20. \end{aligned} \tag{7}$$

Funkcja $f : \mathbb{R}^2 \to \mathbb{R}^2$ ma więc postać

$$f(x_1,x_2) = (x_1^2 + x_2^2 - 2x_1 + 4x_2 + 4,\ 9x_1^2 + 4x_2^2 + 16x_2 - 20). \tag{8}$$

Metoda prostych iteracji

Jedna z najprostszych metod polega na iterowanym obliczaniu wartości funkcji. Jest to metoda blisko związana z **Twierdzeniem Banacha o punkcie stałym.**[1] Punkt stały to taki punkt, który nie jest zmieniany przez funkcję, czyli $x^* = F(x^*)$.

Twierdzenie (Banacha o punkcie stałym). Niech X będzie przestrzenią metryczną zupełną. Jeżeli odwzorowanie $F : X \to X$ spełnia warunek Lipschitza

$$\rho(F(x),F(x)) \le L\rho(x,y),$$

ze stałą $0 \le L < 1,$ to wtedy odwzorowanie F posiada dokładnie jeden punkt stały $x^* \in X$.

Metoda prostych iteracji polega na wyborze jakiegoś punktu startowego $x^{(0)}$ i obliczaniu kolejnych wartości wg formuły

$$x^{(k+1)} = F(x^{(k)}) \ \text{ dla } \ k = 0,1,2,\ldots \tag{9}$$

W przypadku sytuacji opisanej powyżej ciąg (9) zmierza do punktu stałego odwzorowania układu równań niezależnie od wyboru punktu początkowego $x^{(0)} \in X$.

Przy zastosowaniu tej metody do numerycznego rozwiązywania układu (4) należy pamiętać, że Zbieżność ciągu (9) mamy zagwarantowaną, gdy jest spełniony warunek Lipschitza ze stałą L mniejszą od 1 $(L<1)$. W praktyce nie jest łatwo sprawdzić ten warunek, dlatego często po prostu eksperymentujemy. Jeżeli ciąg iteracji stabilizuje się, to mamy podstawy przypuszczać, że jest to przybliżenie punktu stałego, a zatem przybliżenie rozwiązania układu (4), (lub w zwartym zapisie (6)).

Przykład. Stosując metodę iteracji prostych znaleźć rozwiązanie układu równań

$$\begin{cases} x = -0,5\cos x + 0,1y^2, \\ y = 0,34\sin x + 0,5y^4. \end{cases}$$

Używając notacji $x = x_1,\ y = x_2$ możemy układ przepisać następująco

$$\begin{cases} x_1 = -0,5\cos x_1 + 0,1x_2^2, \\ x_2 = 0,34\sin x_1 + 0,5x_2^4. \end{cases} \tag{10}$$

[1] Stefan Banach (1892–1945) wybitny polski matematyk. Urodził się w Krakowie. Po odzyskaniu przez Polskę niepodległości Banach związał się ze Lwowem, gdzie był profesorem na Uniwersytecie im. Jana Kazimierza. W matematyce stworzył podstawy nowej dyscypliny o nazwie *Analiza Funkcjonalna*.

Widać, że jest to układ zapisany w formie problemu punktu stałego

$$x = f(x),$$

gdzie $f(x) = f(x_1, x_2) = (-0,5\cos x_1 + 0,1x_2^2,\ 0,34\sin x_1 + 0,5x_2^4).$

Poniżej jest przedstawiona prosta implementacja znajdowania przybliżenia punktu stałego. Funkcja `fixedPointSimpleIter` jest ogólna procedurą. W prezentowanym listingu wprowadzono jako przykład powyższe równania. Jeżeli chcemy rozwiązywać inny przykład należy zmodyfikować fragmenty zaznaczone dodatkowym tłem. (W języku C/C++ zmienne zapisujemy w tablicy, dlatego są numerowane od zera, tj. `x[0]`=x_1 itd.).

Fixed point simple iterations for systems.cpp

```
#include <iostream.h>
#include <math.h>
//Tutaj definiujemy prawą stronę równań dla zagadnienia punktu stałego x=F(x)
const int n = 2;
double f0(double x[n]) { return -0.5*cos(x[0])+0.1*x[1]*x[1]; }
double f1(double x[n]) { return 0.34*sin(x[0])+0.2*x[1]*x[1]*x[1]*x[1]; }

void fixedPointSimpleIter(double (*f[n])(double x[n]), double x[n], int numIter) {
     double x1[n];
     int i, j, k;
     //Wykonujemy numIter iteracji x_{k+1} = F(x_k)
     for (i=0; i < numIter; i++) {
         for (j=0; j < n; j++) x1[j] = (*f[j])(x);
         for (j=0; j < n; j++) x[j] = x1[j];
     }
}

int main() {
    double (*f[n])(double x[n]), x[n], diff = 0.0;
    int i, j, iterNum;

    x[0] = 0.4; x[1] = 0.5;
    f[0] = &f0; f[1] = &f1;

    fixedPointSimpleIter(f, x, 10);

    for (i=0; i < n; i++) cout << "x[" << i << "] = " << x[i] << "\n";
    for (i=0; i < n; i++) diff = diff + fabs(x[i]-(*f[i])(x));
    diff = diff/n;
    cout << "\nŚrednia różnica: |x-f(x)| = " << diff << "\n";
    system("pause");
}
```

Po uruchomieniu powyższego kodu otrzymamy następujący rezultat:

```
E:\Metody numeryczne\skrypt\Fixed point simple iterations for systems.exe
x[0] = -0.448401
x[1] = -0.147304

Średnia różnica: |x-f(x)| = 2.33575e-009
```

Dla innego punktu startowego możemy uzyskać drugi punkt stały układu (10).

Metoda Newtona–Raphsona dla układów równań nieliniowych

Metoda ta jest uogólnieniem metody Newtona dla pojedynczego równania na przypadek układów równań nieliniowych. Inne metody jednowymiarowe nie mają takiego uogólnienia, a w każdym razie

nie są one tak bezpośrednie.[2] W przypadku jednowymiarowej metody wykonywaliśmy dzielenie przez pochodną

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})},$$

ale dla przypadku wielowymiarowego odpowiednikiem zwykłej pochodnej $f'(x)$ jest macierz Jacobiego oznaczana zazwyczaj jako $Df(x)$. Jej elementami są pochodne cząstkowe odpowiednich składowych (czyli możemy powiedzieć nieco nieformalnie, że różniczkujemy kolejne równania układu (3) względem kolejnych niewiadomych):

$$Df(x) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(x) & \dots & \frac{\partial f_1}{\partial x_n}(x) \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1}(x) & \dots & \frac{\partial f_n}{\partial x_n}(x) \end{bmatrix}. \tag{11}$$

W tym przypadku zamiast dzielenia przez pochodną występują mnożenie przez macierz odwrotną do macierzy Jacobiego. Zatem podstawowy schemat metody Newtona–Raphsona będzie wyglądał następująco

$$x^{(k+1)} = x^{(k)} - \left[Df(x^{(k)}) \right]^{-1} f(x^{(k)}) \qquad (k = 0,1,2,\dots). \tag{12}$$

Tak jak w przypadku jednowymiarowej metody Newtona, tak i teraz musimy wybrać punkt startu $x^{(0)} \in \mathbb{R}^n$, aby można było rozpocząć iteracje opisane w (12). W ogólności teoria mówi, że jeżeli spełniony jest podstawowy warunek dla zbieżności metody Newtona–Raphsona, tj. odwracalność macierzy Jacobiego $Df(x^*)$ i funkcje f_i są klasy C^2, to metoda jest zbieżna jeżeli wystartujemy dostatecznie blisko rozwiązania.[3]

W praktyce obliczeniowej ciąg (12) wyznaczamy inaczej niż przez odwracanie $Df(x^*)$. Należy rozwiązywać układ równań liniowych. Równość ta może być bowiem równoważnie zapisana następująco

$$Df(x^{(k)})(x^{(k+1)} - x^{(k)}) = -f(x^{(k)}) \ (k = 0,1,2,\dots). \tag{13}$$

Takie sformułowanie jest podstawą do następującego algorytmu.

wybierz wektor startowy $x^{(0)} \in \mathbb{R}^n$
while (rel_err > ε **or** licznik < MaxIter **do** {
  rozwiąż układ liniowy $Df(x^{(k)}) \cdot \Delta x = -f(x^{(k)})$ względem Δx
  oblicz nowe przybliżenie $x^{(k+1)} = x^{(k)} + \Delta x$
}

---

2 Na przykład z metodą bisekcji jest taki kłopot, że dla funkcji, których dziedzina leży w $\mathbb{R}^n$, $(n \geq 2)$ nie za bardzo wiadomo co mogłoby zastąpić wybór „lewego" lub „prawego" podprzedziału. Takie pojęcia bezpośrednio odwołują się do liniowego porządku na prostej $\mathbb{R}$. Takiego naturalnego uporządkowania nie ma już na płaszczyźnie.

[3] Twierdzeniem, które precyzuje te warunki jest *Tw. Newtona-Kantorowicza*. W praktyce numerycznej na ogół trudno je sprawdzić, dlatego najczęściej eksperymentujemy z różnymi punktami startowymi.

Jak widać z tego algorytmu podstawowa praca wykonywana jest przy rozwiązywaniu układu liniowego

$$A \cdot \Delta x = b, \tag{14}$$

gdzie $A = Df(x^{(k)})$ oraz kolumna $b = -f(x^{(k)})$. Układ taki możemy rozwiązać jedną z metod opisanych w rozdziale o układach liniowych (np. eliminacja Gaussa, rozkład LU czy metody iteracyjna dla układów liniowych). Ponadto przy zastosowaniu metody Newtona–Raphsona pojawia się jeszcze problem obliczenia macierzy Jacobiego $A = Df(x^{(k)})$, która zmienia się w kolejnych krokach. W zależności od sytuacji możemy wykonać to na dwa sposoby.

Metoda 1

Jeżeli układ jest znany w postaci „analitycznej" czyli funkcje podane są wzorami (tak, jak np. (2)), to możemy obliczyć wszystkie składowe macierzy Jacobiego tez analitycznie (wzorami), a następnie obliczać macierz $A = Df(x^{(k)})$ podstawiając do tych wzorów aktualne przybliżenie. Przykładowo dla układu (2) mamy dwa równania opisane wyrażeniami (7) czyli

$$f_1(x_1, x_2) = x_1^2 + x_2^2 - 2x_1 + 4x_2 + 4,$$
$$f_2(x_1, x_2) = 9x_1^2 + 4x_2^2 + 16x_2 - 20,$$

Obliczamy teraz pochodne cząstkowe

$$\frac{\partial f_1}{\partial x_1}(x_1, x_2) = 2x_1 - 2, \ \frac{\partial f_1}{\partial x_2}(x_1, x_2) = 2x_2 + 4,$$
$$\frac{\partial f_2}{\partial x_1}(x_1, x_2) = 18x_1, \ \frac{\partial f_2}{\partial x_2}(x_1, x_2) = 8x_2 + 16.$$

Zatem ogólnie macierz Jacobiego dla tego układy ma postać

$$Df(x) = \begin{bmatrix} 2x_1 - 2 & 2x_2 + 4 \\ 18x_1 & 8x_2 + 16 \end{bmatrix}.$$

Jeżeli na jakimś etapie obliczeń uzyskamy $x^{(k)} = (0,5, 1,0)$, to podstawiając do powyższego wyrażenia uzyskamy macierz A do obliczenia kolejnego przybliżenie z układu (14)

$$A = Df(0,5, 1,0) = \begin{bmatrix} 2 \cdot 0,5 - 2 & 2 \cdot 1 + 4 \\ 18 \cdot 0,5 & 8 \cdot 1 + 16 \end{bmatrix} = \begin{bmatrix} -1 & 6 \\ 9 & 24 \end{bmatrix}.$$

Metoda 2

Jeżeli funkcja nie jest dana w postaci analitycznej lub nawet jeżeli jest dana, ale obliczanie macierzy Jacobiego wzorami jest niepraktyczne (np. 1000 równań, to pochodnych będzie milion), to możemy przybliżyć elementy tej macierzy (czyli pochodne cząstkowe) poprzez odpowiednie ilorazy różnicowe. Możemy np. wykorzystać zwykłe „różnice skończone w przód" (poniższe wzory dotyczą funkcji jednej zmiennej $g : \mathbb{R} \to \mathbb{R}$)

$$g'(x) \approx \frac{g(x+h) - g(x)}{h}, \tag{15}$$

lub „różnice centralne" (dokładniejsze)

$$g'(x) \approx \frac{g(x+h)-g(x-h)}{2h}. \tag{16}$$

Ponieważ pochodne cząstkowe funkcji wielu zmiennych $g:\mathbb{R}^n \to \mathbb{R}$ są zdefiniowane podobnie jak dla jednej zmiennej, tj.

$$\frac{\partial g}{\partial x_j}(x_1,\ldots,x_n) = \lim_{h\to 0}\frac{g(x_1,\ldots,x_{j-1},x_j+h,x_{j+1},\ldots x_n)-g(x_1,\ldots,x_n)}{h}, \tag{17}$$

więc analogicznie do wzorów (15) lub (16) możemy aproksymować pochodne cząstkowe. Biorąc pod uwagę, że w *i*-tym wierszu mamy pochodne funkcji f_i otrzymujmy (różnica w przód)

$$\frac{\partial f_i}{\partial x_j}(x_1,\ldots,x_n) \approx \frac{f_i(x_1,\ldots,x_{j-1},x_j+h_j,x_{j+1},\ldots x_n)-f_i(x_1,\ldots,x_n)}{h_j}, \tag{18}$$

lub (różnica centralna)

$$\frac{\partial f_i}{\partial x_j}(x_1,\ldots,x_n) \approx \frac{f_i(x_1,\ldots,x_{j-1},x_j+h_j,x_{j+1},\ldots x_n)-f_i(x_1,\ldots,x_{j-1},x_j-h_j,x_{j+1},\ldots x_n)}{2h_j}. \tag{19}$$

Zauważmy, że we wzorach (18), (19) do różnic skończonych dla różnych zmiennych możemy używać innych wartości h stąd we wzorach występują h_j. Oczywiście najprostszy wybór jest, gdy dla każdej zmiennej jest ten sam tj. $h_j = h$ dla $j = 1,\ldots,n.$ Należy jednak pamiętać, że zbyt mała wartość h może prowadzić do dużych błędów zaokrągleń arytmetyki komputerowej.[4] Przy wyborze możemy posłużyć się następującą formułą

$$h_j = \sqrt{\varepsilon_M}\,\max\{|x_j|, M_j\}\operatorname{sgn}(x_j), \tag{20}$$

gdzie $M_j > 0$ jest parametrem, który charakteryzuje typowy rozmiar składowej x_j szukanego rozwiązania x^*, ε_M oznacza epsilon maszynowy. Jest to najmniejsza reprezentowalna liczba maszynowa taka, że $1+\varepsilon_M > 1$ w arytmetyce maszynowej. W typowych środowiskach komputerów osobistych $\varepsilon_M \sim 1{,}19\cdot 10^{-16}$. Na przykład w układzie (7) będzie to $M_j \approx 2$. Oczywiście w zastosowaniach na ogół wiemy coś o szukanym rozwiązaniu (wiemy czego się spodziewać), więc możemy podać oszacowanie parametru M_j. Jeżeli mamy z tym trudność, to możemy spróbować uproszczonej wersji (bez tego parametru), czyli

$$h_j = \begin{cases} \sqrt{\varepsilon_M}\,|x_j|, & \text{gdy } x_j \neq 0, \\ \sqrt{\varepsilon_M}, & \text{gdy } x_j = 0. \end{cases} \tag{21}$$

Przykład. Należy znaleźć numerycznie przybliżenie rozwiązania następującego układu równań

---

[4] Należy pamiętać, że sama definicja pochodnej wyrażona granicą (17) jest pewną abstrakcją dlatego, że operujemy w niej na zbiorze liczb rzeczywistych $\mathbb{R}$, który w komputerze jest reprezentowany tylko przez skończony podzbiór liczb reprezentowalnych. W szczególności h nie może być dowolnie bliskie zeru. Ponadto, gdy h jest bardzo małe (ale nadal reprezentowalne), a x_j zbyt duże to w arytmetyce komputerowej może się okazać, że $x_j + h = x_j$. To spowoduje, że zamiast lepszego przybliżenia pochodnej otrzymamy zero (bo różnica w liczniku wyrażenia (18) będzie zero).

$$\begin{cases} x_0^2 + x_1^2 + x_2^2 = 3, \\ x_0 + 2x_1 - 7x_2 = 0, \\ 0{,}5x_0 - 3x_2 = 0. \end{cases} \tag{22}$$

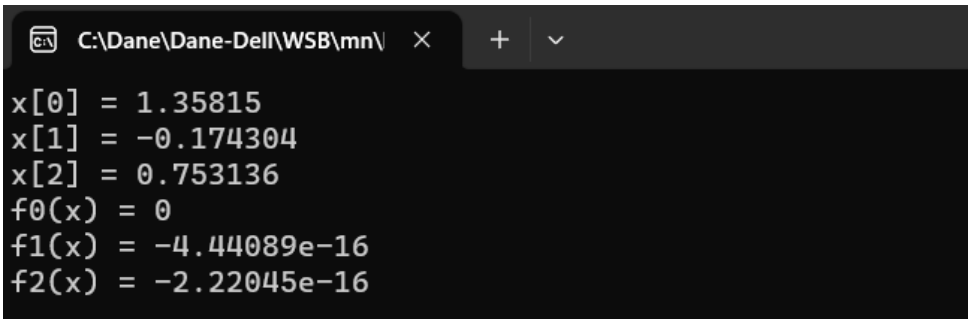
Jest to jak przykład testowy, który stosunkowo łatwo można go rozwiązać analitycznie i otrzymać

$$x_0 \approx \pm 1{,}702742, \quad x_1 = 0{,}141895, \quad x_2 = 0{,}28379$$

Poniżej jest przedstawiona przykładowa implementacja metody Newtona–Raphsona dla układów równań nieliniowych. Procedura rozwiązujące nie jest zrealizowana w postaci odrębnej funkcji, ale jest wprost zapisana w funkcji `main()`. Odwołuje się do funkcji `gauss` (rozwiązywania układu liniowego). Może też użyć rozkładu LU. Aby otrzymać numerycznie macierz Jacobiego $Df(x^{(k)})$ użyto różnic centralnych postaci (19). Jeżeli chcemy rozwiązać układ równań, to należy zdefiniować układ równań podając definicje funkcji występujących po lewej stronie układu. Należy też podać poprawną wartość zmiennej n (liczba równań; dla naszego przykładu $n = 3$).

```
const int n = 3;
void gauss(double a[n][n], double b[n], double x[n]);

double f0(double x[n]) { return pow(x[0],4)+x[1]*x[1]+x[2]*x[2]-4; }
double f1(double x[n]) { return x[0]*x[0]+x[1]*x[1]-2*tan(x[2]); }
double f2(double x[n]) { return x[0]*x[0]-x[2]*x[2]+sin(x[0]*x[1]*x[2])-1.1; }

int main() {
    double a[n][n], deltaX[n], b[n], x[n], x_plus_h[n], x_minus_h[n], h;
    double (*f[n])(double x[n]);
    int i, j, k, r, iterNum;

    f[0] = &f0;
    f[1] = &f1;
    f[2] = &f2;

    h = 0.01; iterNum = 60;
    x[0] = 1.0; x[1] = -1.0; x[2] = 1.0;

    for (k=1; k <= iterNum; k++) {
        for (i=0; i <n ; i++) {
            b[i] = -(*f[i])(x);
            for (j=0; j < n; j++) {
                for (r=0; r < n; r++)
                    if (r == j) { x_plus_h[j] = x[j]+h; x_minus_h[j] = x[j]-h; }
                    else x_plus_h[r] = x_minus_h[r] = x[r];
                a[i][j] = ((*f[i])(x_plus_h) - (*f[i])(x_minus_h))/(2*h);
            }
        }

        gauss(a, b, deltaX);

        for (i=0; i < n; i++) x[i] = x[i] + deltaX[i];
    }

    for (i=0; i < n; i++) cout << "x[" << i << "]=" << x[i] <<"\n";
    for (i=0; i < n; i++) cout << "Równanie numer " << i << ": " << (*f[i])(x) <<"\n";
}
```

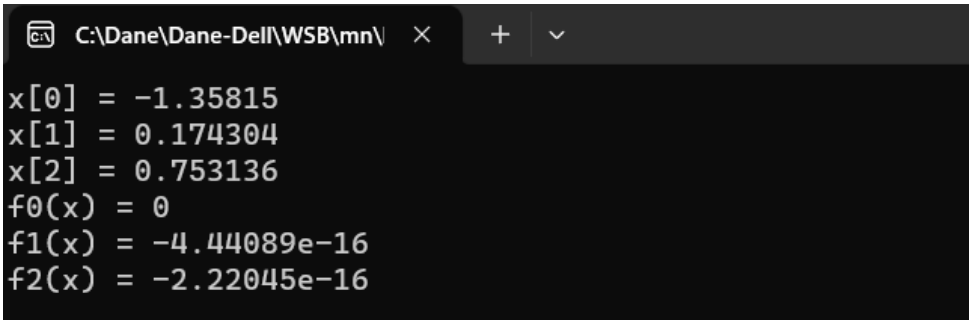
Po uruchomieniu programu otrzymamy następujące wyniki:



Jak widać – zwłaszcza na podstawie sprawdzenia poprzez podstawienie – uzyskane przybliżone rozwiązania ($x[0]=1.35815,\ x[1]=-0.174304,\ x[2]=0.753136$) spełniają układ bardzo dobrze. Aby uzyskać drugie rozwiązanie

$$x^* \approx (-1.35815,\ -0.174304,\ 0.753136),$$

należy wybrać inny punkt startowy $x^{(0)}$. W programie oznacza to zmianę inicjalizacji zmiennych `x[0], x[1], x[2]`. Na przykład dla inicjalizacji `x[0] = -1.0; x[1] = 1.0; x[2] = 1.0` otrzymamy to drugie rozwiązanie:



Projekt

Projekt ma wykorzystywać bibliotekę GSL do rozwiązywania pomocniczych układów równań liniowych. Wszystkie pliki projektu (plik źródłowy i ewentualne pliki tekstowe z danymi maja być skompresowane w jednym pliku.

Nazwa pliku: ***MetNum_Proj1_Nazwisko_Imię_Grupa***

Plik projektu załączamy do emaila i wysyłamy na adres: **szyszkin@agh.edu.pl**

Tytuł listu: ***MetNum_Proj1_Nazwisko_Imię_Grupa***

Zadania do wykonania w ramach projektu

Napisać program (jeden) w języku C++, który będzie implementować metodę Newtona–Raphsona dla układów równań nieliniowych. Rozwiązywanie układu liniowego z macierzą Jacobiego zrealizować funkcjami biblioteki GSL (*gsl_linalg_LU_decomp*, *gsl_linalg_solve*). Maksymalnie zautomatyzować zadawanie układu: podajemy liczbę niewiadomych (n) oraz wprowadzamy w kodzie taką sama liczbę funkcji definiujących równania. Oczywiście punkt startu także zadajemy. Obliczenia kończymy gdy liczba iteracji przekroczy zadany parametr ($MaxIter$) lun gdy zostanie osiągnięta zadana dokładność ($errTol$):

$$\| x^{(k+1)} - x^{(k)} \| \leq errTol \| x^{(k)} \|.$$

Przetestować program rozwiązując numerycznie podane niżej osiem układów równań. Po uruchomieniu program wykonuje odpowiednią procedurę rozwiązującą, a następnie wypisuje na ekranie numer zadania, punkt startowy użyty do iteracji, wartości uzyskanych rozwiązań oraz sprawdzenie poprzez podstawienie rozwiązań do równań i wypisanie otrzymanych wartości (powinny być bliskie zeru).

W krótkim raporcie (maks. dwie strony) podać informacje o metodzie Newtona–Raphsona, implementacji, a w przypadku **Zadania 1** pokazać rachunek (przekształcenia) prowadzące do równania z jedną niewiadomą.

Zadanie 1. Znaleźć numerycznie rozwiązania poniższych układów dwóch równań z dwoma niewiadomymi dokonując najpierw redukcji do równania z jedną niewiadomą. Przy rozwiązywaniu równania z jedną niewiadomą zastosować metodę bisekcji i metodą Newtona.

a) $\begin{cases} 2x^2 + 5y^2 - 2 = 0, \\ -3x^3 + 2y = 0. \end{cases}$

b) $\begin{cases} \cos(2x^3) - 3y^5 + 1 = 0, \\ 2\ln(1+x^2) - y = 0. \end{cases}$

Zadanie 2. Znaleźć numerycznie rozwiązania poniższych układów stosując metodę iteracji prostej.

a) $\begin{cases} 2x - \sin(0,5x + y) = 0, \\ y - 0,5\cos(x - 0,5y) = 0. \end{cases}$

b) $\begin{cases} x + \dfrac{x}{1+x^2} - \dfrac{1}{3}\sin(y) = 0, \\ 0,4\cos(x) + \dfrac{y}{3+y^2} - 2y = 0. \end{cases}$

Zadanie 3. Znaleźć numerycznie po jednym rozwiązaniu poniższych układów stosując metodę Newtona–Raphsona.

a) $\begin{cases} 2x^2 + 3y^2 - 5 = 0, \\ x^3 - \ln y = 0. \end{cases}$

b) $\begin{cases} 2x^4 + y^5 - xy - 1 = 0, \\ x^2 - y^3 - y = 0. \end{cases}$

c) $\begin{cases} 8x^3 + y^3 - 12xy = 0, \\ \cos 2x - y^2 + 1 = 0. \end{cases}$

d) $\begin{cases} \frac{1}{5}(x-1)^2 + y^2 + \frac{1}{3}z^2 - 6 = 0, \\ x^2 - 2,4x + 2y^2 - z + 0,06 = 0, \\ \sqrt[3]{z} + 21\sqrt{x} + 20\sqrt{y} - 25 = 0. \end{cases}$

e) $\begin{cases} x^4 + y^2 + z^2 = 4, \\ x^2 + y^2 - 2\,\mathrm{tg}(z) = 0, \\ x^2 - z^2 + \sin(xyz) = 1{,}1. \end{cases}$

f) $\begin{cases} 3.2x - \cos(yz) = 0.55, \\ x^2 - 80.45(y + 0{,}095)^2 + \sin z + 1.07 = 0, \\ e^{-xy} + 19.5z + 30.5 = 0. \end{cases}$

Z wektorem początkowym $x^{(0)} = [1,\ 1,\ -1]$ oraz $x^{(0)} = [0.1,\ 0.1,\ -0.1]$