Rozdział 7

Rozwiązywanie równań nieliniowych

W tym rozdziale zajmiemy się metodami rozwiązywania równań nieliniowych skalarnych.

Interesuje nas znalezienie zera nieliniowej funkcji $f : [a, b] \to \mathbb{R}$:

$$f(x) = 0.$$

Przetestujemy kilka metod.

7.1 Funkcja octave'a fzero()

Na początek zapoznamy się z funkcją octave'a służącą do znajdowania zer równania nieliniowego, czyli **fzero**().

Funkcja posiada dwa wywołania w pierwszym:

```
x=fzero(f,ap)
```

f jest wskaźnikiem (uchwytem) do funkcji

```
function y=f(x)
```

a $app = (a, b)$ jest wektorem z dwoma liczbami lokalizującymi zero funkcji, tzn. że to zero leży pomiędzy a i b.

Przetestujmy tę funkcję z tym wywołaniem na kilku prostych przykładach:

```
fzero(@(x) x^5,[-1,2])
fzero(@(x) x*x-2,[1,2])-sqrt(2)
fzero(@sin,[-1,2])
fzero(@sin,[3,4])-pi
```

Funkcja działa. Przetestujmy ją w sytuacji kiedy zer może być kilka np.:

```
fzero("sin",[-4,5])
```

Funkcja znalazła pierwiastek $-\pi$, ale może nie zadziałać:

```
fzero("sin",[-4,1])
```

Zwracając:

```
error: fzero: not a valid initial bracketing
```

Nietrudno się domyśleć, że dla tej funkcji właściwy przedział to taki, że funkcja w końcach przyjmuje przeciwne znaki.

Kolejnym możliwym wywołaniem tej funkcji jest podanie *ap* jako liczby -przybliżenia zera:

```
fzero(@(x) x*x-2,2)-sqrt(2)
fzero(@sin,1)
fzero(@sin,3)-pi
```

To wywołanie też działa. Inną funkcją octave'a służącą rozwiązywaniu równań nieliniowych jest fsolve(). Jej wywołanie jest takie samo, jak druga wersja wywołania funkcji **fzero**(), tzn. podajemy wskaźnik (uchwyt) do funkcji i liczbę będącą przybliżeniem pierwiastka, np.:

```
fsolve(@(x) x*x-4,2)
```

Przetestujmy na kilku innych przykładach:

```
fsolve(@(x) x*x*x-27,5)-3
fsolve(@sin,1)
fsolve(@sin,3)-pi
```

Wyniki są poprawne.

Funkcja fsolve() jest bardziej ogólna od **fzero**(), może służyć znajdowania zer układów równań nieliniowych, ale to wykracza poza standardowwy zakres wykładu z Matematyki Obliczeniowej.

7.2 Metoda bisekcji

Najprostszą metodą znajdowania zera jest metoda bisekcji. Dla funkcji ciągłej f - jeśli $f(a) * f(b) < 0$ - to istnieje zero pomiędzy punktami a, b.

Idea metody bisekcji (połowienia odcinka), to przyjęcie za przybliżenie zera $x = (a + b)/2$ i sprawdzenie znaku f w x. Albo $f(x)$ jest równe zero - wtedy otrzymaliśmy, że x jest szukanym zerem. W przeciwnym razie zastępujemy jeden z końców odcinka przez x - ten w którym znak wartości f

w tym końcu jest ten sam co dla $f(x)$. Dalej postępujemy analogicznie z nowym odcinkiem o długości równej połowie poprzedniego.

Poniżej widzimy prostą implementację metody bisekcji:

```
function x=bisekcja(f,a,b,eps=1e-4,itmax=200)
#[x]=bisekcja(f,a,b,eps=1e-4,itmax=200)
#f- wskaznik do funkcji
#a,b poczatek i koniec odcinka
#eps - tolerancja met zatrzymuje sie o ile (b-a)<eps
#itmax - max ilosc iteracji
#Output:
#x- przyblizone zero
#Zakladamy, ze uzytkownik poda dwa punkty a<b
# w ktorych f(a)f(b)<0
ya=f(a);
yb=f(b);
it=0;
x=(a+b)/2;
while( (b-a>eps) && (it<itmax))
y=f(x);
if(sign(y)==sign(ya))
 ya=y;
 a=x;
else
 yb=y;
 b=x;
end
 x=(a+b)/2;
endwhile
endfunction
```

Przetestujmy dla kilku funkcji:

```
bisekcja(@(x) x*x-2,1,2)-sqrt(2)
```

Wynik jest poprawny.

Jeśli $|b-a|$ jest duże to zbieżność jest dość wolna:

```
tic;bisekcja(@(x) x*x-2,1,1e30)-sqrt(2);toc
```

Czas potrzebny do obliczenia zera wydaje się mały, ale wyobraźmy sobie, że musimy rozwiązać kilka milionów takich równań. Spróbujmy znaleźć zero dla funkcji bardziej skomplikowanej $g(x) = \int_0^x \exp(-0.5 * t^2)\, dt = 0$:

```
f=@(x) exp(-x*x/2);
```

```
g=@(x) quad(f,0,x)-1;
x=bisekcja(g,1,2)
g(x)
```

7.3 Metoda Newtona

Kolejną bardzo ważną metodą jest metoda Newtona:

$$x_{n+1} = x_n - f(x_n)/f'(x_n) \qquad n \geq 0$$

poprawnie określona, o ile $f'(x_n) \neq 0$.

Zaimplementujmy w octave'ie metodę Newtona. Poniżej podajemy kod funkcji octave'a szukania zer za pomocą metody Newtona:

```
function [x,fval,info]=newton(f,df,x0,...
                              epsr=1e-7,epsa=1e-9,maxit=30)
#Metoda Newtona skalarna
#
#Input:
#f - function handle do funkcji postaci y=f(x),
#ktorej zera szukamy przy czym zwracana wartosc
#y to wartosc f(x)
#df - function handle do pochodnej funkcji postaci
#y=df(x) ktorej zera szukamy
#x - startowe przyblizenie
#
#Pozostale parametry opcjonalne
#epsr = wzgledna tolerancja
#epsa - bezwgledna tolerancja
#maxit - maksymalna ilosc iteracji
#Output:
#y - przyblizenie zera
#fval - wartosc f(x)
#info - wynik 0 - zbieznosc;
#              1   brak zbieznosci -
#                  przekroczyl max ilosc iteracji
#              2 - df(x) - zero nie mozna policzyc
#                          kolejnego przyblizenia
# iteracje zatrzymuja sie jesli
# zachodzi jedna z dwu
#   1/|f(x)|>epsa + epsr*|f(x0)|
```

```
#    lub
#    2/ilosc iteracji = maxit
#
# Przyklad:
# function y=f(x)
#  y=x*x-2;
# endfunction
#
# Wywolujemy:
# y=newton(@f,2)
# [y,fy,info]=newton(@f,4,1e-2,1e-3) itd

fx=f(x0);
fval=fx(1);
it=0;
info=0;
eps=0.5*(epsr*abs(fval)+epsa);

x=x0;
while((abs(fval)>eps ) && (it<maxit))
#liczymy pochodna
 dfx=df(x);
 if(abs(dfx)>1e-12)
  x=x - fval/dfx;
 else
  info=2;
  printf("Pochodna df w punkcie x_%d=%g zero\n",it,x);
  return;
 endif
 it++;
 fx=f(x);
 fval=fx(1);
endwhile
if((abs(fval)>eps ))
 info=1; #brak zbieznosci
 printf("Brak zbieznosci! Po ...
%d iteracjach  |f(%g)|=%g>tol=%g!\n",it-1,x,fval,eps);
endif
endfunction
```

Zbadajmy zbieżność metody Newtona dla modelowej funkcji $f(x) = x^2 -$

4, której pierwiastkami są $-2, 2$.

```
newton(@(x) x*x-4,@(x) 2*x,4)-2
```

Teraz wywołamy tę funkcję ze wszystkimi zwracanymi wartościami:

```
[x,fval,info]=newton(@(x) x*x-4,@(x) 2*x,4)
```

Przetestujmy, czy metoda zbiega dla $x_0 = 100$:

```
[x,fval,info]=newton(@(x) x*x-4,@(x) 2*x,100)
```

czy dla x_0 jeszcze bardziej oddalonych od pierwiastków:

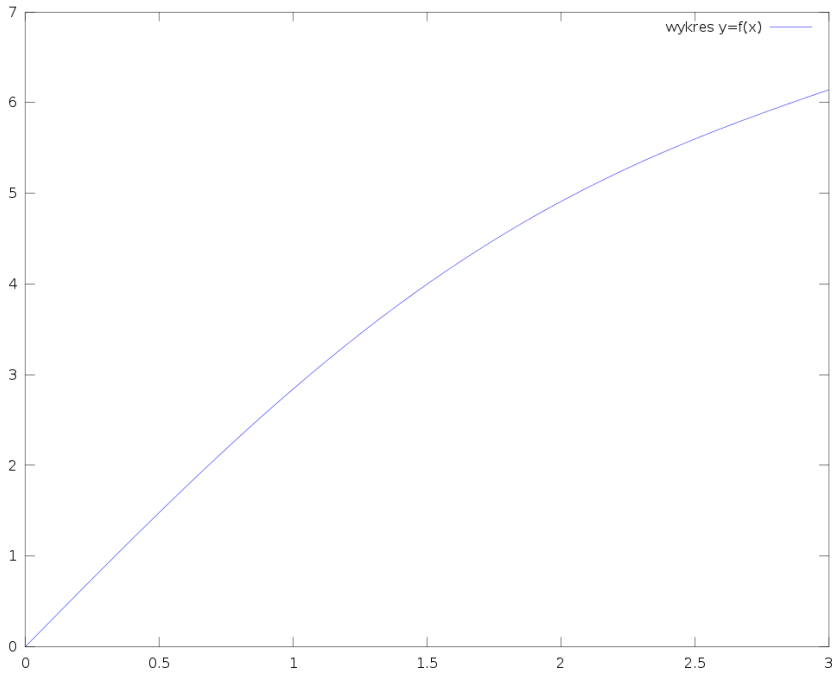
```
[x,fval,info]=newton(@(x) x*x-4,@(x) 2*x,1000)
[x,fval,info]=newton(@(x) x*x-4,@(x) 2*x,0.01)
[x,fval,info]=newton(@(x) x*x-4,@(x) 2*x,1e-9)
[x,fval,info]=newton(@(x) x*x-4,@(x) 2*x,1e6)
```

W ostatnim przypadku ilość iteracji w warunku stopu okazała się niewystarczająca.

Możemy też przetestować rząd zbieżności dla równania, którego rozwiązanie znamy, np. rozpatrzmy ostatnie równanie $x^2 = 4$: Poniższy kod testuje rząd zbieżności metody Newtona dla dodatniego pierwiastka funkcji $f(x) = x^2 - 4$:

```
M=6;
x=4;
e=0;
it=0;
for k=1:M,
 x=x-(x*x-4)/(2*x);
 it++;
 ep=e;
 e=x-2;
 if(it>1)
  printf("[%d] e/ep=%3.4e e/(ep*ep)=%3.4e \n",...
   it,e/ep,e/(ep*ep));
 endif
endfor
```

Otrzymaliśmy, że rząd zbieżności dla tej funkcji jest równy dwa, co jest zgodne z teorią, która mówi, że jeśli funkcja jest klasy C^2 na otoczeniu swojego pierwiastka x^*, oraz wartość pochodnej funkcji w pierwiastku jest różna od zera to metoda jest lokalnie zbieżna kwadratowo (zbieżna z rzędem równym dwa): tzn. istnieje takie $\epsilon > 0$ i stała $C \geq 0$, że jeśli x_0 startowe



Rysunek 7.1: Wykresy funkcji $f(x) = 2 * x + \sin(x)$ na $[0, 3]$.

przybliżenie metody Newtona spełnia: $|x^* - x_0| \leq \epsilon$, to

$$|x_{n+1} - x^*| \leq C|x_n - x^*|^2,$$

por. np. [11].

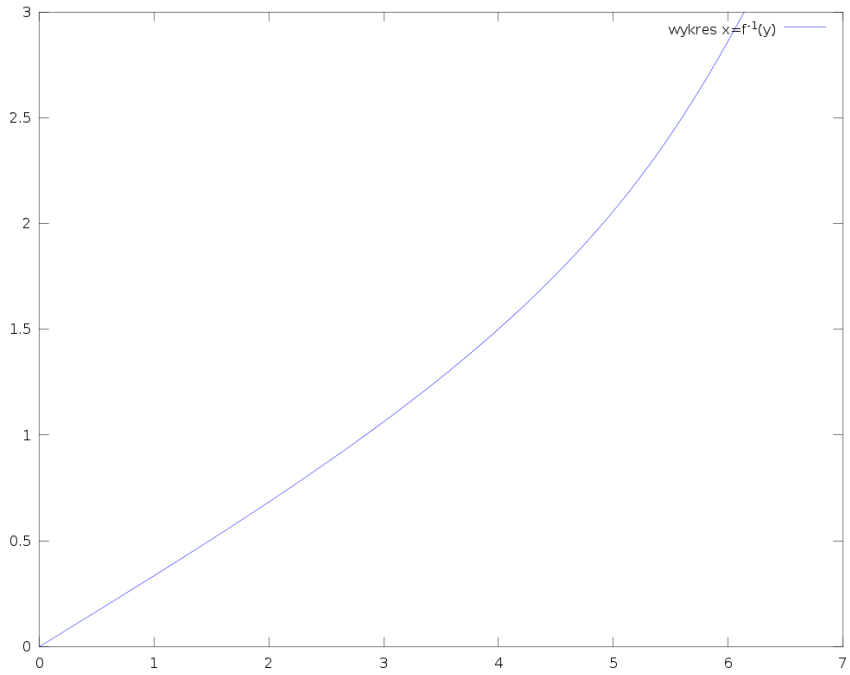
7.4 Odwracanie funkcji

Podamy jeden przykład zastosowania metod rozwiązywania równań nieliniowych do znajdowania przybliżonych wartości funkcji odwrotnej do danej. Załóżmy, że potrafimy policzyć wartości funkcji, i chcemy znaleźć wartości funkcji odwrotnej do danej funkcji, aby np. narysować wykres funkcji odwrotnej (zakładamy, że na danym odcinku ona istnieje). Zazwyczaj niestety nie znamy wzoru analitycznego na funkcję odwrotną.

Jak to zrobić z zastosowaniem funkcji **fzero**()?

Gdyby chodziło o sam wykres możemy zastosować proste odbicie:

```
x=linspace(a,b);
```



Rysunek 7.2: Wykresy funkcji odwrotnej do $f(x) = 2 * x + \sin(x)$ na $[0, 3]$.

```
y=f(x);
plot(x,y,";wykres y=f(x);");
plot(y,x,";wykres x=f^{-1}(y);");
```

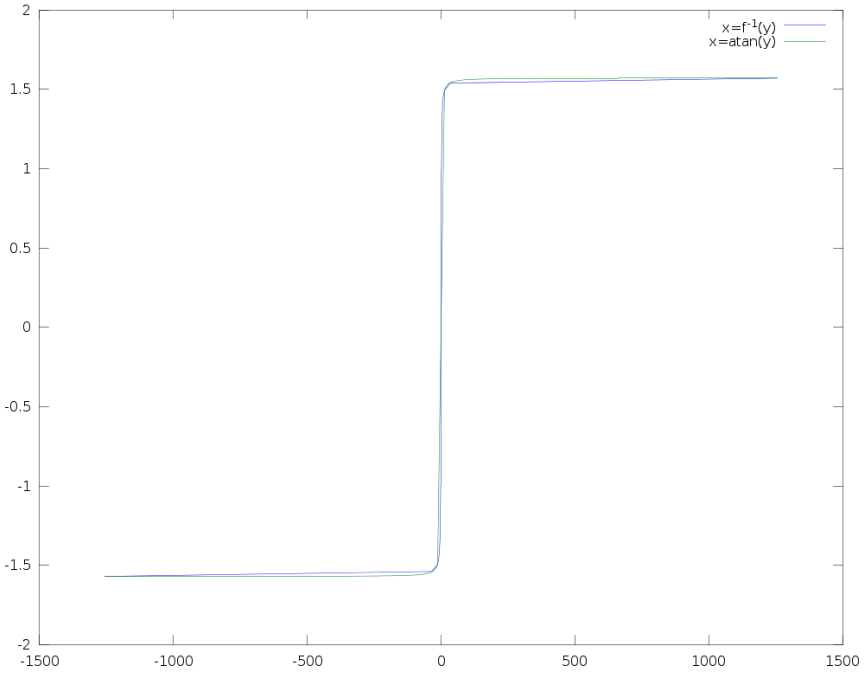
Na rysunku 7.1 widać wykres $f(x) = 2 * x + \sin(x)$ na $[0, 3]$, a na rysunku 7.2 wykres funkcji do niej odwrotnej.

Oczywiście wykres funkcji odwrotnej wygląda zgodnie z oczekiwaniami. Rozpatrzmy inną funkcje: $f(x) = \tan(x)$ na $[-1.57, 1.57]$ i popatrzmy na wykresy funkcji odwrotnej otrzymanej w analogiczny sposób i wykres rzeczywistej funkcji odwrotnej, której wzór analityczny w tym przypadku znamy: $(\tan)^{-1}(y) = \text{atan}(y)$: rysunek 7.3. Widać, że wykres funkcji odwrotnej jest trochę inny od wykresu uzyskanego naszym sposobem. Aby uzyskać wykres prawidłowy powinniśmy znać bardzo dokładne przybliżenia wartości funkcji odwrotnej w punktach siatki.

Aby obliczyć wartość $f^{-1}(y)$ dla konkretnego y należy rozwiązać równanie:

$$g(x) = y - f(x) = 0,$$

co w praktyce można w sposób przybliżony dokonać przy użyciu np. funkcji

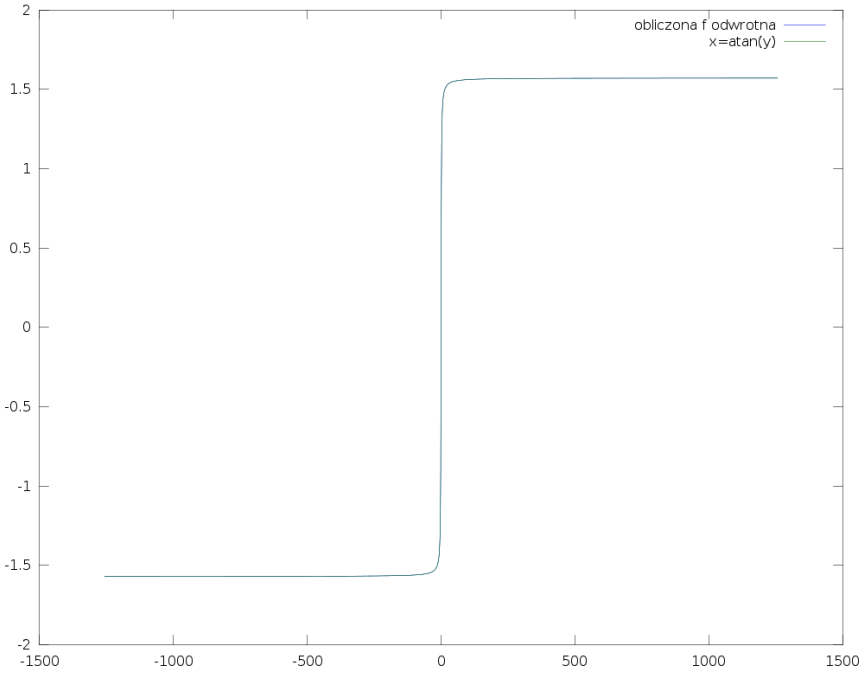


Rysunek 7.3: Wykresy funkcji odwrotnej do $f(x) = \tan(x)$ na dwa sposoby. kolor zielony rzeczywisty wykres funkcji odwrotnej, kolor niebieski wykres uzyskany przez odbicie.

fzero() w octave'ie.

Poniższy kod oblicza wartości funkcji odwrotnej do $f(x) = \tan(x)$ na siatce równomiernych punktów na $[-f(1.57), f(1.57)]$ (korzystamy z tego, że funkcja jest nieparzysta i obliczamy jej wartości tylko dla argumentów dodatnich):

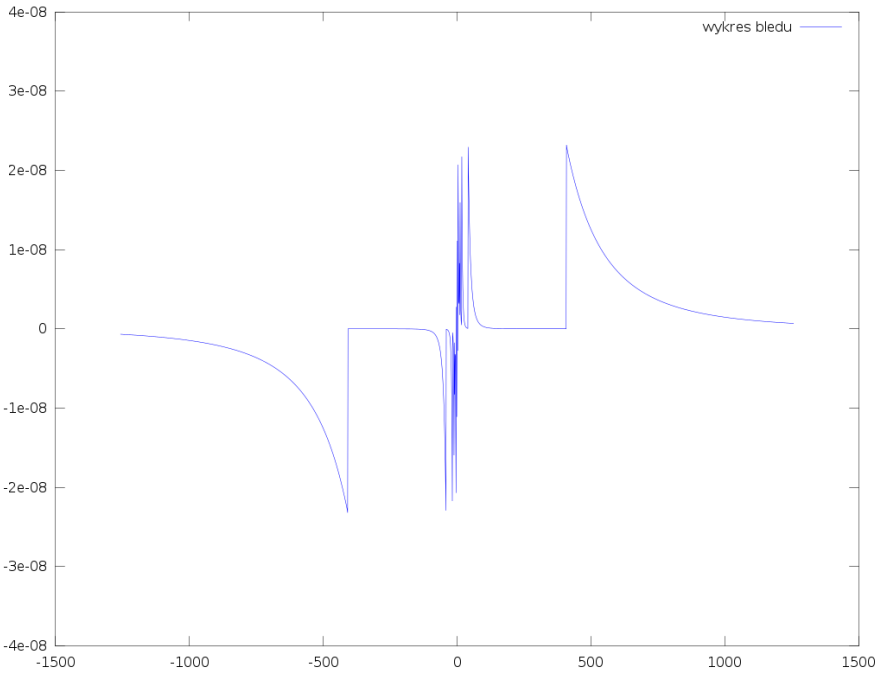
```
f=@tan;
c=f(-1.57);d=-c;
N=1000;
y=linspace(0,d,N);
x=zeros(N,1);
x(1)=0; #tan(0)=0
for k=2:N,
  x0=x(k-1);
  g=@(x) y(k) - f(x);
  x(k)=fsolve(g,x0);
```



Rysunek 7.4: Wykresy funkcji odwrotnej do $f(x) = \tan(x)$. Wykonane na dwa sposoby. Kolor zielony - rzeczywisty wykres funkcji odwrotnej, kolor niebieski - wykres uzyskany przez odbicie. Oba wykresy się pokrywają.

```
endfor
y=[y-d  y];x=[-flipud(x);x];
plot(y,x,";oblicz._f_odwrotna;",y,atan(y),";atan(y);")
pause(2);
plot(y,x'-atan(y),";wykres_bledu;")
```

Na rysunku 7.4 widać, że wykresy funkcji odwrotnej oraz obliczonej przez nas pokrywają się. Potwierdza to także wykres błędu, por. rysunek 7.5.



Rysunek 7.5: Wykresy błędu obliczania funkcji odwrotnej do $f(x) = \tan(x)$.