Riešenie nelineárnych rovníc II

Na predchádzajúcej hodine sme si pripravili základný úvod do problematiky nelineárnych rovníc. Ukázali sme si, akým spôsobom je možné riešiť nelineárne rovnice priamym hľadaním a zároveň sme venovali čas základným postupom riešenia nelineárnych rovníc využitím **symbolic toolboxu**.
Dnes sa budeme venovať ďalším spôsobom, ktorými je možné riešiť nelineárne rovnice v MATLABe. Ukážeme si jednu z najznámejších metód na riešenie nelineárnych rovníc: Newtonovu metódu. V druhej časti hodiny si povieme o univerzálnom spôsobe riešenia nelineárnych rovníc v MATLABe s použitím funkcie **fsolve**.

Newtonova metóda

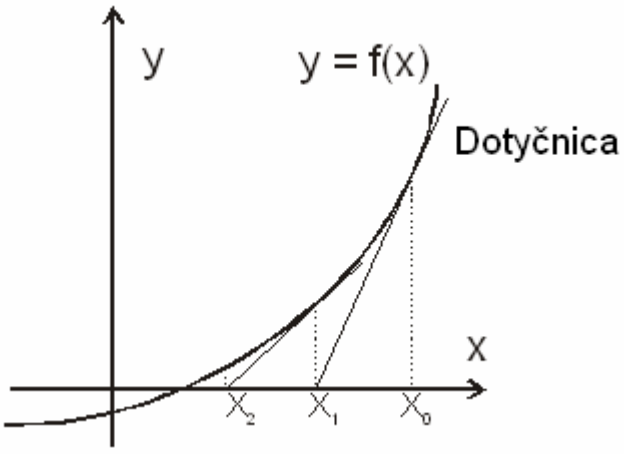
Ako bolo uvedené na predchádzajúcej hodine, spôsobov ktorými je možné riešiť nelineárne rovnice je mnoho. Na tomto základnom kurze nie je dostatok priestoru a času, aby sme sa venovali všetkým základným metódam a preto si predstavíme iba jednu a to Newtonovu metódu.
Newtonova metóda patrí medzi gradientové metódy na riešenie nelineárných rovníc. Niekedy sa tejto metóde hovorí aj metóda dotyčníc, pretože využíva na nájdenie koreňa nelineárnej rovnice rovnicu dotyčnice. Krivku v okolí koreňa nahradzujeme dotyčnicou v bode $[x_i, f(x_i)]$. Priesečník dotyčnice s osou x je nová aproximácia koreňa x_{i+1}. Vychádzame z toho, že z hodnoty prvej derivácie funkcie v určitom bode sa dá určiť rovnica dotyčnice prechádzajúcej týmto bodom. K výpočtu pomocou Newtonovej metódy sa využíva jednoduchý vzťah:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Pokiaľ je pre daný problém možné použiť Newtonovú metódu, je veľmi vhodná, nakoľko je veľmi jednoduchá.
Nevýhodou tejto metódy je skutočnosť, že nemusí konvergovať vždy. Takéto kritérium použiteľnosti môže značne obmedziť oblasť jej použitia:

- funkcia musí byť v okolí koreňov spojitá
- funkcia nesmie mať v okolí koreňa nulovú deriváciu

Princíp Newtonovej metódy pre konvexnú funkciu je znázornený na nasledujúcom obrázku:



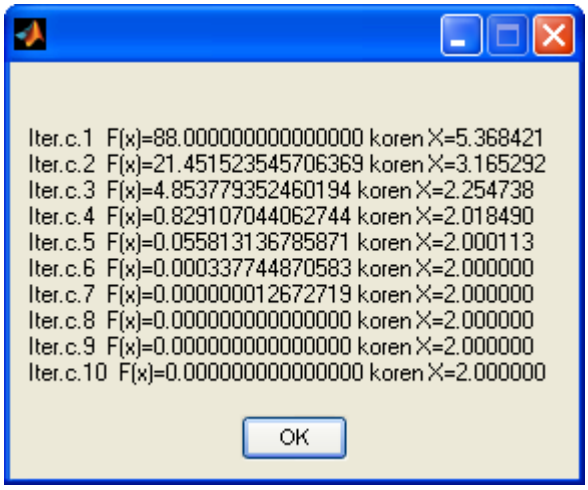
Teória možno vyzerá trochu „nepriateľsky", no príklad, na ktorom si ukážeme využitie Newtonovej metódy, Vás dúfam presvedčí o tom, že riešenie nelineárnych rovníc je v podstate sranda.

Príklad 1

Vytvorte program, ktorý vypočíta prvých 10 iterácii Newtonovou metódou rovnice x^2-x-2=0.

```
clc
xstare=10;
str='';
for iteracia=1:10
   fx=xstare^2-xstare-2;
   dfdx=2*xstare-1;
   xnove=xstare-fx/dfdx;
   xstare=xnove;
   str=sprintf('%s\nIter.c.%d F(x)=%1.15f koren
X=%f',str,iteracia,fx,xstare);
end
msgbox(str);
```

Výsledok vyzerá nasledovne:



Informácia o funkčných hodnotách rovnice jednoznačne ukazuje, že sa nám podarilo nájsť koreň cvičnej rovnice.
Ak si pozorne pozriete uvedený príklad, zistíte, že daný algoritmus má hneď niekoľko nevýhod:

- už siedma iterácia mala veľmi nízku hodnotu a preto nebolo nutné pokračovať vo výpočte
- príklad nie je dostatočne univerzálny, lebo v prípade, že by sme chceli uvedený algoritmus použiť na riešenie inej rovnice, bolo by nutné celý algoritmus prerobiť
- naviac ak by sme sa pokúšali riešiť komplikovanejšiu rovnicu, mohol by byť značný problém analytický určiť hodnotu prvej derivácie

Tieto nevýhody rieši druhý príklad. V tomto príklade prerobíme algoritmus tak, aby sme nemuseli priamo zadať analytickú hodnotu prvej derivácie, ale aby sa počítala numericky.

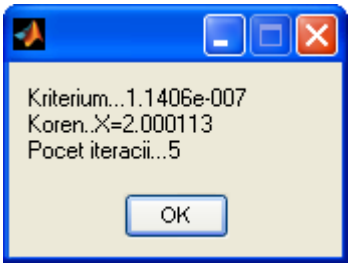
Ďalej algoritmus upravíme tak, aby vykonal práve toľko iterácií, aby hodnota koreňa bola dostatočne presná:

Príklad 2

```
function main
clc
xstare=10;
kriterium=funkcia(xstare)^2;
presnost=1e-6;
iteracia=0;
while kriterium>presnost
   fx=funkcia(xstare);
   dfdx=derivacia(xstare);
   xnove=xstare-fx/dfdx;
   xstare=xnove;
   kriterium=funkcia(xstare)^2;
   iteracia=iteracia+1;
end
str=sprintf('Kriterium...%1.4e\nKoren..X=%f\nPocet
iteracii...%d',kriterium,xstare,iteracia);
msgbox(str);

function [FX]=funkcia(X)
FX=X^2-X-2;

function DFDX=derivacia(X)
delta=1e-6;
DFDX=(funkcia(X)-funkcia(X-delta))/delta;
```



Niekoľko poznámok k uvedenému príkladu:

- v tomto príklade sme ako kvalitu riešenia použili štvorec/štvorce ľavej strany cvičnej rovnice (podobne ako na predchádzajúcej hodine) a žiadali sme aby hodnota nášho kritéria mala hodnotu nižšiu ako 1e-6
- výhoda takéhoto algoritmu je skutočnosť, že vykoná iba nevyhnutný počet iterácii (skúste meniť hodnotu požadovanej presnosti a sledovať ako sa mení počet potrebných iterácií, prípadne skúste meniť hodnotu počiatočného odhadu riešenia, ktorý takisto bude mať vplyv na počet iterácii)
- ďalšia výhoda je „univerzálnosť" uvedeného algoritmu, nakoľko v prípade, že by sme chceli zmeniť rovnicu, ktorej koreň potrebujeme vypočítať, stačí zmeniť jediný riadok v programe (funkciu funkcia)

Riešenie nelineárnych rovníc využitím funkcie *fsolve*

Pravdepodobne „najrobustnejší" spôsob, ktorý MATLAB ponúka na riešenie nelineárnych rovníc je funkcia ***fsolve***. Uvedený solver je schopný bez problémov riešiť relatívne rýchle stovky nelineárnych rovníc. My si teraz jeho využitie ukážeme na dvoch príkladoch rovníc, ktoré sme riešili na predchádzajúcich hodinách.

Prototyp funkcie je nasledujúci:

```
x = fsolve(fun,x0)
[x,fval] = fsolve(fun,x0)
[x,fval,exitflag] = fsolve(...)
[x,fval,exitflag,output] = fsolve(...)
```

pričom
fun je funkcia
x0 je počiatočný odhad riešenia
x je hľadané riešenie
fval je hodnota pravých strán rovníc v bode, v ktorom bol nájdený koreň
exitflag je číslo, ktoré udáva akým výsledkom bol výpočet ukončený, môže nadobúdať hodnoty od -4 po 4 (význam všetkých čísel nebudeme teraz rozoberať, nakoľko v prípade potreby je možné si ich pozrieť v nápovede k funkcii fsolve)
output je štruktúra, v ktorej sú uložené podrobné informácie o ukončení výpočtu, o použitom algoritme a počte iterácii

Existujú aj ďalšie prototypy volaní funkcie fsolve, ktorými sa teraz nebudeme zaoberať. Za zmienku azda stojí prototyp, ktorý umožňuje nastaviť podrobné parametre výpočtu nelineárnych rovníc:

```
[x,fval] = fsolve(fun,x0,options)
```

pričom
options je špeciálna štruktúra, do ktorej je možné zadať parametre výpočtu a tvorí sa pomocou funkcie **optimset** (viď help optimset).

Príklad 1

Nájdite riešenie nelineárnej rovnice x^2-x-2=0.

```
function main
riesenie=fsolve(@funkcia,5)

function F=funkcia(x)
F=x^2-x-2;
```

```
riesenie =

2.0000
```

Uvedený zápis je najjednoduchší a poskytne iba informácie o hodnote koreňa. Ak by sme potrebovali vedieť hodnotu účelovej funkcie pri skončení výpočtu, či podrobnejšie informácie o ukončení výpočtu, použijeme zápis:

```
function main
[riesenie,prava_strana,ukoncenie_vypoctu,info]=fsolve(@funkcia,5)

function F=funkcia(x)
F=x^2-x-2;
```

```
riesenie =

2.0000

prava_strana =

1.7053e-012

ukoncenie_vypoctu =

1

info =

iterations: 6
funcCount: 14
algorithm: 'trust-region dogleg'
firstorderopt: 5.1159e-012
message: 'Optimization terminated: first-order optimality is less than options.TolFun.'
```

Týmto zápisom sme priamo získali hodnotu, ktorú nadobúda naša funkcia v bode riešenia a funkcia fsolve nám poskytla informáciu o spôsobe ukončenia výpočtu (v tomto prípade hodnota 1, ktorej význam je: výpočet konverguje k riešeniu, viď help fsolve).
Štruktúra **info** obsahuje informácie o počte iterácii, ďalej o počte volaní funkcie, v ktorej je zadefinovaná naša funkcia, o použitom algoritme, atď.

Príklad 2

Riešte sústavu nelineárnych rovníc:

$2x_1-x_2-e^{-x1}=0$
$-x_1+2x_2-e^{-x2}=0$

```
function main
[riesenie,prava_strana,ukoncenie_vypoctu,info]=fsolve(@funkcia,[5 5])

function F=funkcia(x)
F(1)=2*x(1)-x(2)-exp(-x(1));
F(2)=-x(1)+2*x(2)-exp(-x(2));
```

```
riesenie =

0.5671 0.5671
```

prava_strana =

1.0e-008 *

-0.1082 -0.1082

ukoncenie_vypoctu =

1

info =

iterations: 6
funcCount: 21
algorithm: 'trust-region dogleg'
firstorderopt: 1.6956e-009
message: 'Optimization terminated: first-order optimality is less than options.TolFun.'