Soluția ecuației neliniare $f(x) = 0$. Rădăcinile polinoamelor. Lucrul cu polinoame în Octave. Rezolvarea sistemelor de ecuații neliniare.

Colaboratori: Andrei STAN, Sorin Ciolofan

February 17, 2025

Cuprins

1 Obiective laborator

- Însuşirea noţiunilor privitoare la determinarea aproximativă a soluţiilor unei ecuaţii neliniare;
- Rezolvarea iterativă a unui sistem de ecuaţii neliniare;
- Comenzi Octave pentru lucrul cu polinoame;
- Rădăcinile polinoamelor.

2 Noțiuni teoretice

2.1 Soluţia ecuaţiei neliniare $f(x) = 0$

Vom studia două tipuri de metode:

a) Metode bazate pe interval

Se porneşte de la observaţia că o funcţie îşi schimbă semnul în vecinătatea unei rădăcini. Iniţial, se consideră două valori de o parte şi de cealaltă a rădăcinii, apoi se micşorează acest interval care încadrează rădăcina până ce se ajunge la rădăcină, cu o anumită precizie. Metodele de acest tip sunt întotdeauna convergente către soluţie deoarece aplicarea repetată a algoritmului duce la o estimare mai precisă a rădăcinii.

În continuare, două metode bazate pe interval vom detalia: metoda bisecţiei şi metoda secantei.

Metoda bisecţiei

Dacă o funcţie îşi schimbă semnul pe un interval, adică $f(a) \cdot f(b) < 0$, atunci se evaluează valoarea funcţiei în punctul de mijloc al intervalului, $c = \frac{a+b}{2}$. Locaţia rădăcinii se estimează ca fiind la mijlocul subintervalului în care funcţia îşi schimbă semnul, noul subinterval având la unul dintre capete pe c. Procedura se repetă până ce se obţin estimări mai precise ale rădăcinii.

Se poate defini eroarea relativă procentuală folosind relaţia:

$$\epsilon = \left| \frac{x_r^{new} - x_r^{old}}{x_r^{new}} \right|$$

.

Când $\epsilon < prag$, algoritmul iterativ se termină, iar x_r^{new} este considerată valoarea calculată a rădăcinii.

Un alt avantaj al acestei metode este faptul că pentru o eroare acceptată tol, se poate calcula numărul de iteraţii ce trebuie parcurse pentru a se ajunge la aproximarea dorită a rădăcinii, conform formulei:

$$n = \log_2(\frac{b-a}{tol}).$$

Metoda secantei

În această metodă, aproximarea rădăcinii, în intervalul $[a_i, b_i]$, se consideră a fi intersecţia dreptei care trece prin punctele $(a_i, f(a_i))$ şi $(b_i, f(b_i))$ cu axa Ox, adică:

$$x_{i+1} = \frac{a_i f(b_i) - b_i f(a_i)}{f(b_i) - f(a_i)}.$$

Apoi, se consideră subintervalul care are la unul dintre capete pe x_{i+1}, în manieră similară cu metoda bisecţiei. Acelaşi criteriu de oprire, eroarea relativă procentuală ϵ, pot fi folosite.

b) Metoda care nu se bazează pe un interval, în care este suficientă cunoaşterea unei valori iniţiale x_i care este folosită mai departe pentru estimarea valorii următoare x_{i+1}. Aceste metode, spre deosebire de primele, pot fi convergente sau pot fi divergente. Atunci când ele converg, convergenţa este mult mai rapidă decât în cazul metodelor bazate pe interval. Pentru această categorie, menţionăm metoda tangentei (Newton-Raphson) şi metoda aproximaţiilor succesive (contracţiei).

Metoda tangentei

După cum s-a menţionat anterior, se porneşte cu o valoare de început x_i, apoi se deduce o estimare îmbunătăţită x_{i+1}. În cazul metodei tangentei, se duce o tangentă la curba din punctul de coordonate $[x_i, f(x_i)]$. Punctul de intersecţie a tangentei cu Ox se consideră x_{i+1}.

Relaţia de recurenţă este:

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

Se poate arăta că eroarea la iteraţia curentă este proporţională cu pătratul erorii la iteraţia precedentă, ceea ce, aproximativ, înseamnă că la fiecare iteraţie numărul de zecimale corect calculate din rădăcină se dublează.

Metoda aproximaţiilor succesive

În cazul acestei metode, ecuaţia $f(x) = 0$ se rescrie ca $x = g(x)$, ceea ce are avantajul de a furniza o formulă prin care se poate calcula o nouă valoare a lui x ca o funcţie g aplicată vechii valori a lui x, adică:

$$x_{i+1} = g(x_i)$$

Această metodă converge liniar (eroarea la pasul i este proporţională cu eroarea la pasul $i-1$ inmulţită cu un factor subunitar) dacă $\left|g^{'}(x)\right| < 1, \forall x \in (a, b)$. Altfel, metoda este divergentă.

2.2 Rădăcinile polinoamelor

Fie p un polinom de grad n, $p(x) = a_n x^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + ... + a_1 x + a_0$, *cu* $a_n \neq 0$. Conform cu *teorema fundamentală a algebrei*, polinomul p are n rădăcini reale sau complexe (numărând şi multiplicităţile). În cazul în care coeficienţii a_i sunt toţi reali, rădăcinile complexe apar conjugate (de forma $c + id$ şi $c - id$).

Folosind regula semnelor a lui Descartes, putem număra câte rădăcini reale pozitive are polinomul p. Fie v numărul variaţiilor de semn ale coeficienţilor $a_n, a_{n-1}, a_{n-2}, .., a_1, a_0$, ignorând coeficienţii care sunt nuli. Fie n_p numărul de rădăcini pozitive. Avem următoarele două relaţii:

1) $n_p \leq v$;

2) $v - n_p$ este un număr par.

Analog, numărul de rădăcini reale negative ale lui $p(x)$ se obţine folosind numărul de schimbări de semn ale coeficienţilor polinomului $p(-x)$. Pentru determinarea rădăcinilor se pot aplica metodele descrise anterior la punctul b), dacă nu se cunoaşte localizarea rădăcinilor pe intervale.

2.3 Lucrul cu polinoame în Octave

În Octave, un polinom este reprezentat prin coeficienţii săi (în ordine descrescătoare). Vectorul

```
> p = [-2, -1, 0, 1, 2]
```

reprezintă polinomul $-2x^4 - x^3 + x + 2$.

Funcţia *polyout* generează o reprezentare a funcţiei de o variabilă dată (de exemplu x)

```
> polyout(p,'x')
```

are ca rezultat $-2 * x^4 - 1 * x^3 + 0 * x^2 + 1 * x^1 + 2$.

Evaluarea unui polinom:

```
> y = polyval(p, x)
```

returnează $p(x)$. Dacă x este vector sau matrice, polinomul este evaluat în fiecare dintre elementele lui x.

Înmulţirea a două polinoame:

```
> r = conv(p, q)
```

returnează un vector r care conţine coeficienţii produsului dintre p şi q.

Împărţire a două polinoame:

```
> [b, r] = deconv(y, a)
```

returnează coeficienţii polinoamelor b şi r a.î. $y = ab + r$, unde b este câtul şi r este restul împărţirii.

Rădăcinile unui polinom:

```
> roots(p)
```

returnează un vector ce conţine toate rădăcinile polinomului p.

Derivata unui polinom:

```
> q = polyder(p)
```

returnează un vector ce conţine coeficienţii derivatei polinomului p.

Integrarea unui polinom:

```
> q = polyint(p)
```

returnează un vector ce conţine coeficienţii rezultaţi din integrarea polinomului p.

Polinomul de interpolare de gradul n care aproximează setul de date (x, y):

```
> p = polyfit(x, y, n)
```

Exemplu: Adunarea polinoamelor

Presupunem că dorim să adunăm polinoamele $p(x) = x^2 - 1$ şi $q(x) = x + 1$. Următoarea încercare va da eroare:

```
> p = [1, 0, -1];
> q = [1, 1];
> "p + q error: operator +: nonconformant arguments (op1 is 1x3, op2 is 1x2)
error:evaluating binary operator ‘+’ near line 22, column 3"
```

Operaţia de adunare a doi vectori de dimensiuni diferite în Octave dă eroare. Pentru a evita aceasta, trebuie adăugate zerouri la q.

```
> q = [0, 1, 1];
> p + q
ans = 1 1 0
> polyout(ans, 'x')
1*x^2 + 1*x^1 + 0
```

2.4 Sisteme de ecuații neliniare

Un sistem de ecuaţii neliniare are forma:

$$\begin{cases} f_1(x_1, x_2, x_3, ..., x_{n-1}, x_n) = 0 \\ f_2(x_1, x_2, x_3, ..., x_{n-1}, x_n) = 0 \\ ... \\ f_n(x_1, x_2, x_3, ..., x_{n-1}, x_n) = 0 \end{cases}$$

unde f_i reprezintă funcţii cunoscute de n variabile $x_1, x_2, ..., x_n$, presupuse continue, împreună cu derivatele lor parţiale până la un ordin convenabil (de obicei, până la ordinul doi). Se va urmări găsirea soluţiilor reale ale sistemului într-un anumit domeniu de interes, domeniu în care se consideră valabile proprietăţile de continuitate impuse funcţiilor f_i şi derivatelor lor. Rezolvarea sistemului este un proces iterativ în care se porneşte de la o aproximaţie iniţială pe care algoritmul o va îmbunătăţi până ce se va îndeplini o condiţie de convergenţă. În cazul de faţă, localizarea apriori a soluţiei nu mai este posibilă (nu există o metodă analoagă metodei înjumătăţirii intervalelor).

Metoda Newton

Pentru simplificarea notaţiei, considerăm $F = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ ... \\ f_n \end{pmatrix}$ şi $x = (x_1, x_2, ..., x_n)$. Sistemul îl putem rescrie ca $F(x) = 0$. Notăm cu $x^{(k)}$ estimarea la pasul k a soluţiei x^*, deci $F(x^*) = 0$.

Se poate deduce relaţia:

$$x^{(k+1)} = x^{(k)} - J^{-1}F(x^{(k)}), k = 0, 1, 2, ...$$

unde J este matricea *Jacobiană*

$$J = \begin{pmatrix} \frac{\partial F_1}{\partial x_1} & ... & \frac{\partial F_1}{\partial x_n} \\ ... & ... & ... \\ \frac{\partial F_n}{\partial x_1} & ... & \frac{\partial F_n}{\partial x_n} \end{pmatrix}$$

Dacă matricea J este neinversabilă, atunci pasul este nedefinit. Vom presupune că $J(x^*)$ este inversabilă, iar continuitatea lui J va asigura că $J(x^{(k)})$ este inversabilă pentru orice $x^{(k)}$ suficient de apropiat de x^*. Secvenţa definită iterativ converge spre solutia x^*. Condiţia de oprire la iteraţia k, $\left\|x^* - x^{(k)}\right\| < tol$, unde tol este o toleranţă dată, se poate arăta că revine la $\left\|x^{(k)} - x^{(k-1)}\right\| < tol$.

3 Probleme rezolvate

3.1 Problema 1

Scrieţi o funcţie Octave care primeşte ca parametri a, b (capetele intervalului în care se află soluţia reală), tol (toleranţa), respectiv un handler la funcţia care defineşte ecuaţia neliniară de rezolvat şi o rezolvă prin metoda bisecţiei.

Soluţie:

```
function sol = bisect(a, b, f, tol)
  m = 0;
  while(((b-a)/(2^m)) > tol)
    sol = (a+b)/2;
    if (f(a)*f(sol) < 0)
      b = sol;
    elseif (f(sol)*f(b) < 0)
      a = sol;
    endif
    m = m+1;
  endwhile
endfunction
```

Listing 1: Metoda bisecţiei.

Date de intrare:	**Date de ieşire:**
$a = 1, b = 2, @ecuatie, tol = 10^{-10}$	$sol = 1.6038$

Handler-ul care defineşte funcţia este:

```
function f = ecuatie(x)
  f = x^3-2*x^2+5*x-7;
endfunction
```

Listing 2: Handler funcţie.

3.2 Problema 2

Scrieţi o funcţie Octave care să rezolve o ecuaţie neliniară folosind metoda secantei. Parametrii pe care îi primeşte funcţia sunt: a, b (capetele intervalului în care se află soluţia reală), tol (toleranţa), respectiv un handler la funcţia care defineşte ecuaţia de rezolvat.

Soluţie:

```
function sol = secanta(a, b, f, tol)
  m = 0;
  while(((b-a)/(2^m)) > tol)
    sol = (a*f(b)-b*f(a))/(f(b)-f(a));
    if (f(a)*f(sol) < 0)
      b = sol;
    elseif (f(sol)*f(b) < 0)
      a = sol;
    endif
    m = m+1;
  endwhile
endfunction
```

Listing 3: Regula secantei.

Date de intrare:	**Date de ieşire:**
$a = 1, b = 2, @ecuatie, tol = 10^{-5}$	$sol = 1.6038$

3.3 Problema 3

Scrieţi o funcţie Octave care să rezolve o ecuaţie neliniară prin metoda tangentei. Parametrii pe care îi primeşte funcţia sunt: x_0 (aproximaţia iniţială a soluţiei), *tol* (toleranţa), respectiv un handler la funcţia care defineşte ecuaţia de rezolvat şi un handler la derivata acesteia.

Soluţie:

```
function sol = tangenta(x0, f, fd, tol)
  while(1)
    sol = x0-f(x0)/fd(x0);
    if (abs(sol-x0) < tol && f(sol) < tol)
      break;
    endif;
    x0 = sol;
  endwhile
endfunction
```

Listing 4: Metoda tangentei.

Date de intrare:	**Date de ieşire:**
$x_0 = 0, @ecuatie, @ecDer, tol = 10^{-5}$	$sol = 1.6038$

Handler-ul care defineşte derivata funcţiei:

```
function fd = ecDer(x)
  fd = 3*x^2-4*x +5;
endfunction
```

Listing 5: Handler pentru derivata funcţiei.

3.4 Problema 4

Scrieţi o funcţie Octave care să rezolve o ecuaţie neliniară prin metoda aproximaţiilor succesive. Parametrii pe care îi primeşte funcţia sunt: x_0 (aproximaţia iniţială a soluţiei), *tol* (toleranţa), respectiv un handler la funcţia care defineşte ecuaţia de rezolvat.

Soluţie:

```
function sol = aprox_succesive(x0, f, tol)
  while(1)
    sol = (-x0^3+2*x0^2+7)/5;
    if (abs(sol-x0)<tol && f(sol)<tol)
      break;
    endif;
    x0 = sol;
  endwhile
endfunction
```

Listing 6: Metoda aproximaţiilor succesive.

Date de intrare:	**Date de ieşire:**
$x_0 = 0, @ecuatie, tol = 10^{-5}$	$sol = 1.6038$

4 Probleme propuse

4.1 Problema 1

Să se determine tipul (dacă sunt reale pozitive sau negative, complexe) rădăcinilor polinomului $p(x) = 3x^6 + x^4 - 2x^3 - 5$.

4.2 Problema 2

Să se scrie în Octave un program care rezolvă un sistem de ecuaţii neliniare prin metoda Newton. Ca intrare, se consideră un vector coloană care reprezintă $x^{(0)}$, un pointer (handler) la o funcţie care evaluează F într-un vector generic x, un pointer la o funcţie care calculează Jacobiana într-un vector generic x, o toleranţă dată ϵ. Metoda se opreşte atunci când $\left\|x^{(k)} - x^{(k-1)}\right\| < \epsilon$ şi returnează vectorul soluţie x^* şi numărul de iteraţii n care au fost necesare pentru producerea soluţiei.