Soluția ecuației neliniare $f(x) = 0$. Analiza convergenței. Lucrul cu polinoame în MATLAB. Rezolvarea sistemelor de ecuații neliniare.

Cuprins

1 Obiective laborator

În urma parcurgerii acestui laborator, studentul va fi capabil să:

- determine aproximativ soluţiile unei ecuaţii neliniare;
- rezolve iterativ un sistem de ecuaţii neliniare;
- aplice metoda gradientului descendent și conjugat;
- lucreze în MATLAB cu polinoame.

2 Noțiuni teoretice

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

2.1.1 Metode bazate pe interval

Puncte fixe. Dezvoltăm ideea de la laboratorul trecut. Pentru a fi contracție pe intervalul $[a, b]$, funcţia $f(x)$ trebuie să fie continuă pe $[a, b]$ şi pentru orice $c \in [a, b]$ să fie satisfăcută inegalitatea:

$$|f'(c)| < 1$$

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, $c = \frac{a+b}{2}$. Dacă $f(c) \cdot f(b) < 0$ atunci rădăcina se află în intervalul (c, b) unde se continuă căutarea. La fel, dacă $f(c) \cdot f(a) < 0$ atunci rădăcina se află în intervalul (a, c). Procedura se repetă până ce se obţin estimări mai precise ale rădăcinii.

Se poate defini toleranța relativă folosind relaţia:

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

Când $\epsilon < tol$, algoritmul iterativ se termină, iar x_{new} este considerată valoarea calculată a rădăcinii. Metoda aceasta este întotdeauna convergentă deoarece aplicarea repetată a algoritmului duce la o estimare mai precisă a rădăcinii.

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

$$\frac{b-a}{2^n} \leq tol \Rightarrow \frac{b-a}{tol} \leq 2^n \Rightarrow n \geq \log_2(\frac{b-a}{tol})$$

Algoritmul poate fi gândit ca o căutare binară a rădăcinii într-un vector cu $\frac{b-a}{tol}$ elemente, de unde și complexitatea.

2.1.2 Metode care nu se bazează pe un interval

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.

Algorithm 1 Metoda Bisecției

```
while |f(c)| > tol do
    c ← (a+b)/2
    if f(a) · f(c) < 0 then
        b ← c
    else
        a ← c
    end if
end while
```

Metoda tangentei (Newton). 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}. Această metodă nu are garanția convergenței, existând pericolul, printre altele, de a ajunge într-un punct extrem unde $f'(x) = 0$ sau în apropierea unuia.

Relaţia de recurenţă este:

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

Se poate arăta prin dezvoltare în serie Taylor că eroarea la iteraţia curentă este proporţională cu pătratul erorii la iteraţia precedentă [1], ceea ce, aproximativ, înseamnă că la fiecare iteraţie numărul de zecimale corect calculate din rădăcină se dublează.

Definim eroarea ca $e_k = x_k - x^*$, astfel încât $x_k = e_k + x^*$. Aplicând Teorema lui Taylor, pentru $x = x_k$ și $h = -e_k$, avem:

$$0 = f(x^*) = f(x_k) + (x^* - x_k)f'(x_k) + \frac{1}{2}f''(\xi_k)(x^* - x_k)^2$$

Prima presupunere pe care o facem este că $f'(x^*) \neq 0$. Împărțim la $f'(x_k)$:

$$\frac{f(x_k)}{f'(x_k)} + (x^* - x_k) = -\frac{1}{2}\frac{f''(\xi_k)}{f'(x_k)}(x^* - x_k)^2$$

Din definiția metodei lui Newton ajungem la:

$$e_{k+1} = -\frac{1}{2}\frac{f''(\xi_k)}{f'(x_k)}e_k^2$$

Se observă că rata de convergență este cel puțin pătratică dacă atât f' și f'' sunt continue iar eroarea după primul pas este mai mică decât 1.

Algorithm 2 Metoda Newton

$i \leftarrow 1$
while $i \leq$ max_iter **do**
    $x_{prev} \leftarrow x$
    $x \leftarrow x - \frac{f(x)}{f'(x)}$
    if $|x - x_{prev}| <$ tol **then**
        break
    end if
    $i \leftarrow i + 1$
end while

Metoda secantei. De multe ori nu se poate calcula derivata funcțiilor neliniare, de aceea tangenta se aproximează prin secanta care trece prin două puncte apropiate. Astfel x_i devine:

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

Algorithm 3 Metoda Secantei

$i \leftarrow 1$
while $i \leq$ max_iter **do**
    $x \leftarrow x_1 - \frac{f(x_1)(x_1 - x_0)}{f(x_1) - f(x_0)}$
    if $|x - x_1| <$ tol **then**
        break
    end if
    $x_0 \leftarrow x_1$
    $x_1 \leftarrow x$
    $i \leftarrow i + 1$
end while

2.1.3 Viteza de convergență a metodelor prezentate

În tabelul de mai jos se observă numărul de iterații prin care a trecut fiecare metodă pentru a aproxima rădăcina unei funcții neliniare. Valoarea toleranței alese a fost 10^{-15}. Ca valori inițiale, pentru o rădăcină c, au fost alese fie valorile $[[c], [c] + 1]$ în cazul metodelor ce necesită două valori inițiale, fie $[c] + 1$ în cazul celor cu o singură valoare inițială, unde $[c]$ reprezintă partea întreagă a numărului c.

Se poate observa creșterea în eficiență de la metoda bisecției la următoarele două, care au avut de departe cel mai mic număr de iterații. Între metoda tangentei și cea a secantei prima se dovedește mai rapidă datorită formulelor exacte ale derivatelor funcțiilor.

Funcție	**Bisecție**	**Secantă**	**Tangentă**
$0.25e^x - 2$	48	7	7
$3\cos(x) - 4x$	50	7	5
$x^2 - 2$	49	7	6
$\ln(x) - 2$	46	6	4
$x^2 + \sqrt{x} - 6$	48	7	5

Tabelul 1: Numărul iterațiilor necesare pentru fiecare metoda

2.2 Metoda Gradientului Descendent și a Gradientului Conjugat

Pentru sisteme de forma $Ax = b$, dacă matricea A este pozitiv semidefinită și simetrică, putem utiliza atât metoda pașilor descrescători, cât și metoda gradientului conjugat pentru a ajunge la soluția x.

Ambele metode se formulează ca niște probleme de optimizare în care ne dorim să găsim minimul funcției definite de:

$$f(x) = \frac{1}{2}x^T Ax - b^T x$$

Gradientul lui f în acest caz este $\nabla f(x) = Ax - b$. Asta înseamnă că soluția sistemului $Ax = b$ este echivalentă cu minimizarea funcției $f(x)$.

2.2.1 Gradientul Descendent

Gradientul ne oferă direcția de creștere maximă a funcției. Pentru un vector v cu $||v|| = 1$, direcția de creștere maximă este dată de $\theta = 0$.

$$\langle \nabla f(x), v \rangle = ||\nabla f(x)|| \cdot ||v|| \cdot \cos(\theta) = ||\nabla f(x)|| \cdot \cos(\theta)$$

Asta înseamnă că pentru a obține direcția de scădere maximă, trebuie să ne deplasăm în direcția opusă gradientului. Notăm cu $r^{(k)}$ reziduul, adică $r^{(k)} = b - Ax^{(k)}$. Asta înseamnă că $r^{(k)}$ este direcția de scădere maximă fiind egală cu $-\nabla f(x^{(k)})$. La fiecare pas al algoritmului se calculează $x^{(k+1)}$.

$$x^{(k+1)} = x^{(k)} + \alpha r^{(k)}$$

Pentru parametrul α ne dorim să alegem un pas optim, așa că folosim tehnica *line search*. Ne dorim să minimizăm funcția $g(\alpha)$ definită ca:

$$g(\alpha) = f(x^{(k)} + \alpha r^{(k)}) = \frac{1}{2}(x^{(k)} + \alpha r^{(k)})^T A(x^{(k)} + \alpha r^{(k)}) - b^T(x^{(k)} + \alpha r^{(k)})$$

$$g'(\alpha) = r^{(k)T} A(x^{(k)} + \alpha r^{(k)}) - b^T r^{(k)}$$

$$g'(\alpha) = r^{(k)T} Ax^{(k)} + \alpha r^{(k)T} Ar^{(k)} - b^T r^{(k)}$$

$$g'(\alpha) = r^{(k)T} Ax^{(k)} - b^T r^{(k)} + \alpha r^{(k)T} Ar^{(k)}$$

$$g'(\alpha) = r^{(k)T}(Ax^{(k)} - b) + \alpha r^{(k)T} Ar^{(k)}$$

$$g'(\alpha) = -r^{(k)T} r^{(k)} + \alpha r^{(k)T} Ar^{(k)}$$

$$g'(\alpha) = 0 \equiv \alpha = \frac{r^{(k)T} r^{(k)}}{r^{(k)T} Ar^{(k)}}$$

Pentru a nu calcula de fiecare dată $r^{(k)} = b - Ax^{(k)}$ se poate înmulți ecuația $x^{(k)} = x^{(k-1)} + \alpha r^{(k-1)}$ la stânga cu $-A$. În urma acestei operații se ajunge la formula

$$r^{(k+1)} = r^{(k)} - \alpha Ar^{(k)}$$

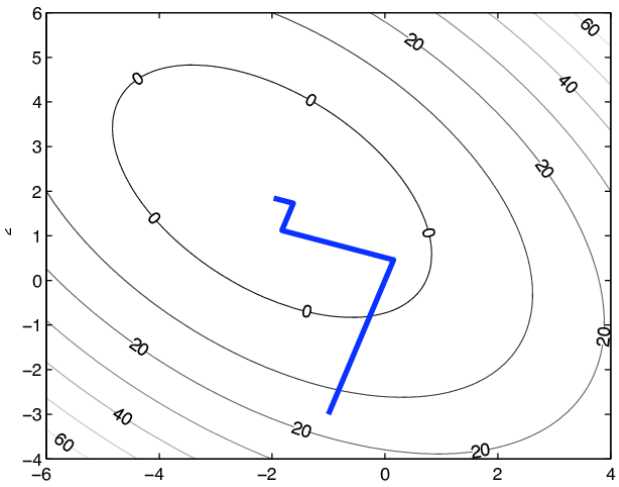

Figura 1: Metoda Gradientului Descendent

Algorithm 4 Metoda Gradientului Descendent

$r \leftarrow b$
$x \leftarrow 0$
$i \leftarrow 1$
while $i \leq$ max_iter **do**
  if $\|r\| <$ tol **then**
    break
  end if
  $ar \leftarrow Ar$
  $\alpha \leftarrow \frac{r^T r}{r^T ar}$
  $x \leftarrow x + \alpha \cdot r$
  $r \leftarrow r - \alpha \cdot ar$
  $i \leftarrow i + 1$
end while

2.2.2 Metoda Gradientului Conjugat

O metodă mult mai eficientă este cea a gradientului conjugat, ea având proprietatea de a converge garantat după cel mult n iterații. Ideea din spatele acestei metode este de a construi un set de direcții de căutare conjugate între ele în raport cu matricea A. Spre deosebire de metoda gradientului descendent, unde fiecare direcție de căutare este ortogonală cu cea anterioară, în metoda gradientului conjugat direcțiile sunt $A - conjugate$, adică satisfac relația:

$$p^{(i)T} A p^{(j)} = 0, \quad \text{pentru } i \neq j.$$

Această proprietate asigură că soluția exactă este atinsă în cel mult n pași pentru un sistem de dimensiune n, în cazul în care nu există erori numerice.

Algoritmul începe similar cu metoda gradientului descendent, însă în loc să ne deplasăm în direcția dată de gradient, construim un nou vector $p^{(k)}$ care este conjugat cu toate cele anterioare (Gram-Schmidt):

$$p^{(k)} = r^{(k)} + \sum_{i=0}^{k-1} \beta^{(i)} p^{(i)}$$

Știm de la laboratorul 3 că procesul Gram-Schmidt nu este unul eficient. Totuși, se pare că este suficient

să calculăm doar pe $\beta^{(k)}$ pentru a obține direcții conjugate.

Subspații Krylov. Subspațiul de dimensiune k îl generăm ca fiind

$$K_k = \text{span}\{r^{(0)}, Ar^{(0)}, A^2r^{(0)}, \ldots, A^{k-1}r^{(0)}\}$$

Dacă A este inversabilă atunci folosind teorema Cayley-Hamilton, toți vectorii din K_k sunt liniari independenți. Se poate observa că $r^{(0)}$ și $p^{(k)}$ progresează în același subspațiu Krylov. Având în vedere că la pasul k, vectorul r avansează în subspațiul Krylov K_k, scăzând din el proiecția acestuia pe vectorii p calculați anterior din subspațiul K_{k-1}, rămânem doar cu componenta corespunzătoare lui β_k. Gândiți-vă la algoritmul Gram-Schmidt modificat: la fiecare iterație scădem din toți vectorii proiecțile lor vectorul curent.

Coeficientul $\beta^{(k)}$ este

$$\beta^{(k)} = -\frac{\mathbf{r}^{(k)T} A\mathbf{p}^{(k-1)}}{\mathbf{p}^{(k-1)T} A\mathbf{p}^{(k-1)}}$$

Iar după ce se prelucrează se ajunge la formula lui β:

$$\beta^{(k)} = \frac{r^{(k)T} r^{(k)}}{r^{(k-1)T} r^{(k-1)}}$$

Algorithm 5 Metoda Gradientului Conjugat

```
r ← b
p ← r
x ← 0
i ← 1
while i ≤ max_iter do
    ap ← Ap
    pap ← p^T ap
    rr ← r^T r
    α ← rr / pap
    x ← x + α · p
    r ← r − α · ap
    if ‖r‖ < tol then
        break
    end if
    β ← (r^T r) / rr
    p ← r + β · p
    i ← i + 1
end while
```

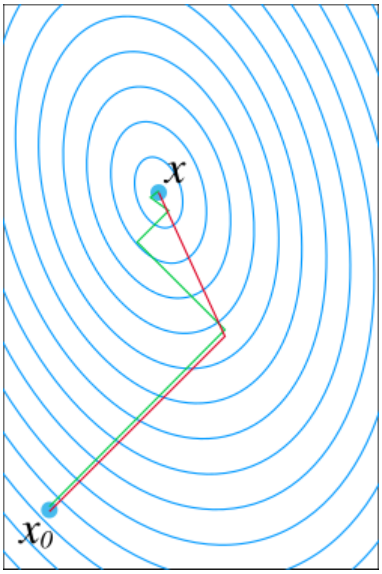

Figura 2: Metoda Gradientului Conjugat și a Pașilor descrescători

2.3 Sisteme de ecuații neliniare

Un sistem de ecuaţii neliniare are forma:

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

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).

2.3.1 Puncte fixe

Și în acest caz putem încerca să transformăm sistemul de ecuații neliniare într-un sistem de ecuații de tip punct fix:

$$
\begin{aligned}
&x_1 = g_1(x_1, x_2, x_3, ..., x_{n-1}, x_n) \\
&x_2 = g_2(x_1, x_2, x_3, ..., x_{n-1}, x_n) \\
&... \\
&x_n = g_n(x_1, x_2, x_3, ..., x_{n-1}, x_n)
\end{aligned}
$$

Pentru ca funcțiile g_i să fie contracții avem nevoie ca gradienții lor să aibă normă mai mică decât 1 pe domeniul ales.

2.3.2 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:

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

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 soluția 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 $\left\|x^{(k)} - x^{(k-1)}\right\| < tol$.

2.4 Polinoame

2.4.1 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.

2.4.2 Lucrul cu polinoame în MATLAB

În MATLAB, un polinom este reprezentat prin coeficienţii săi (în ordine descrescătoare). Vectorul `p = [-2, -1, 0, 1, 2]` reprezintă polinomul $-2x^4 - x^3 + x + 2$.

Funcții MATLAB

- `polyval(p, x)` - calculează $p(x)$.

- `conv(p, q)` - calculează convoluția celor două polinoame (le înmulțește).
- `polyder(p)` - calculează derivata polinomului.
- `polyint(p)` - calculează integrala polinomului.

3 Probleme propuse

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

Referințe

[1] Michael Overton. Quadratic convergence of newton's method.