Lucrarea 2

REZOLVAREA NUMERICĂ A ECUAȚIILOR ALGEBRICE

1. SCOPUL LUCRĂRII

Prezentarea unor metode de rezolvare a ecuațiilor algebrice, şi implementarea acestora în limbaje de nivel înalt (în particular, ANSI C*).*

2. PREZENTARE TEORETICĂ

Calculul rădăcinilor unei ecuații se face în două etape:

1) Separarea rădăcinilor.
2) Calculul lor cu o eroare impusă.

2.1. SEPARAREA RĂDĂCINILOR

Considerăm funcția $f:[a,b]\to \mathbf{R}$, $x\in[a,b]$ şi ecuația algebrică

$$f(x)=0 \tag{2.1}$$

Separarea rădăcinilor unei ecuații $f(x)=0$ constă în determinarea unor intervale în domeniul de definiție al funcției, în care să existe **o singură rădăcină** reală. Pentru separarea rădăcinilor reale există mai multe metode dintre care amintim: metoda şirului lui Rolle, metoda şirului lui Şturm şi metoda lui Budan- Fourier.

În continuare se prezintă metoda *şirului lui Şturm*.

2.1.1. METODA ŞIRULUI LUI ŞTURM

Considerăm funcția definită pe $f:[a,b]\to$ R, care îndeplineşte condițiile de continuitate şi derivabilitate pentru $x\in[a,b]$.

Definiția 2.1

Şirul de funcții $f_0,f_1, f_2.....f_m$ continue pe $[a,b]$ care satisfac condițiile:

a) $f_0(x)=f(x)$;
b) $f_m(x)\neq 0$ pentru $x\in[a,b]$;
c) dacă $f_i(x)=0$, $1\le i\le m-1$ şi $x\in[a,b]$, atunci $f_{i-1}(x)*f_{i+1}(x)<0$;
d) dacă $f_0(x)=0$ pentru $x\in[a,b]$, atunci $f'_0(x)*f_1(x)>0$

se numeşte şirul lui Şturm asociat funcției f(x).

Numărul rădăcinilor **ecuației f(x)** în intervalul $[a,b]$ este dat de următoarea teoremă:

Teorema 2.1

Fie şirul lui Şturm $f_0,f_1, f_2.....f_m$, ataşat funcției f(x) cu condițiile f(a) ≠0 şi f(b) ≠0, atunci numărul de rădăcini ale ecuației f(x)=0 în intervalul $[a,b]$ este dat de diferența numărului de variații de semn ale şirurilor (relația 2.2):

$$f_0(a),f_1(a), f_2(a).....f_m(a)$$
$$f_0(b),f_1(b), f_2(b).....f_m(b) \qquad (2.2)$$

În cazul **funcției polinom** P(x) care este definită pe R, Teorema 2.1 devine:

Teorema 2.2

Fie $P_0, P_1, P_2,.....P_m$ un şir de polinoame construit astfel $P_0=P$, $P_1=P'$, iar P_{i+1} este restul împărțirii lui P_{i-1} la P_i luat cu semn schimbat, pentru $2\le i\le m$. Atunci numărul de rădăcini ale ecuației P(x)=0 este egal cu diferența dintre numărul de schimbări de semn ale şirurilor:

$$P_0(-\infty), P_1(-\infty), P_2(-\infty),.....P_m(-\infty)$$
$$P_0(\infty), P_1(\infty), P_2(\infty),.....P_m(\infty)$$

2.1.1.1. Algoritmul 2.1. Şirul lui Şturm

```
void Sturm
(
   intreg grad,       // gradul polinomului
   real P1[],         // vectorul coeficienților polinomului
   real P2[],         // vectorul coeficienților derivatei
                      // polinomului;
   real Rez[],        // vectorul coeficienților restului
   real C1,           // pentru calculul coeficienților
                      // restului
   real C2            // coeficienții pentru calculul
                      // coeficienților restului
)
{ // corpul funcției
   pentru i=grad-1 pana la 0 calculează P2[i]=(i+1)P1[i+1];
   daca grad=0 Rez(0)=P1(0)-P1(1);
                           // Restul are gradul grad-2
   altfel
   {
      C1 = P1[grad]/P2[grad-1];
      C2 = (P1[grad-1]-C1*P2[grad-2])/P2[grad-1];
      pentru i= (grad-2 pana la 1) calculează
         Rez(i)=P1(i)-C1*P2[i-1]-C2*P2[i];
      Rez(0)=P1[0]=P1-C2*P2[0];
   }
tipăreşte coeficienții restului cu semn schimbat;
}
```

Notă: Programul şirului lui Şturm este implementat pe calculator şi poate fi utilizat direct.

2.2. ESTIMAREA ERORII

Pentru metodele numerice ce urmează să fie prezentate în continuare, soluția ecuației f(x)=0 se determină ca limită a unui şir de valori, $\{x_n\}$. Apar astfel două probleme, şi anume:

- cum se determină limita acestui şir;
- "cât de repede" are loc convergența sa.

Fixăm un număr *eps* în funcție de numărul de zecimale exacte cu care dorim să calculăm rădăcinile ecuației, şi determinăm termenii consecutivi ai şirului $\{x_n\}$ până se îndeplineşte condiția 2.3:

$$|x_n - alfa| < eps \tag{2.3}$$

unde *alfa* este limita şirului $\{x_n\}$ şi deci rădăcina ecuației.

2.2.1. ESTIMAREA APRIORI A ERORII

În acest caz dorim să obținem o inegalitate de forma:

$$|x_n - alfa| < eps(n) \tag{2.4}$$

Dacă inecuația `eps(n) < eps`, având necunoscuta n, poate fi rezolvată fără calculul termenilor şirului $\{x_n\}$, atunci vom şti apriori care termen al şirului aproximează soluția ecuației, cu precizia *eps*.

2.2.2. ESTIMAREA APOSTERIORI A ERORII

Acum urmărim să obținem o inegalitate de forma:

```
|xn - alfa| < const * |xn - xn-1| < eps          (2.5)
```

Deci vom testa realizarea condiției 2.6:

```
const * |xn - xn-1| < eps                        (2.6)
```

unde accentul se pune pe diferența ultimilor doi termeni ai şirului.

2.3. CALCULUL RĂDĂCINILOR REALE ALE ECUAȚIEI ALGEBRICE

Pentru calculul rădăcinilor reale ale unei ecuații algebrice se utilizează mai multe metode numerice.

Metodele de rezolvare a ecuațiilor le putem împărți în două categorii:

1. ***Metode directe*** care, teoretic, presupun un număr finit de operații, şi
2. ***Metode indirecte***, care implică , din punct de vedere teoretic, un număr infinit de operații (soluția este determinată ca limită a unui şir).

Vom studia:

- *pentru metodele directe*: metoda bisecției (bipartiției);
- *pentru metodele indirecte*:
  - metoda aproximațiilor succesive,
  - metoda lui Newton – Raphson (metoda tangentei)
  - metoda lui Bairstow.

2.4. METODE DIRECTE PENTRU REZOLVAREA ECUAȚIILOR

2.4.1. METODA BISECȚIEI (BIPARTIȚIEI)

Fie funcția continuă $f:[a,b]\to \mathbf{R}$ şi ecuația $f(x)=0$ *c*are are soluție unică pe intervalul $[a,b]$. Pentru condițiile date funcția satisface inecuația $f(a)\cdot f(b)\le 0$. Se pune problema determinării soluției α a ecuației $f(x)=0$ pe intervalul $[a,b]$, cu o anumită eroare ε.

Metoda are la bază principiul *Divide et Impera*. Problema dată se reduce la sub-probleme de aceeaşi natură, dar elementare (deci mai uşor de rezolvat), după care toate soluțiile intermediare ale sub-problemelor se combină în soluția finală Rezolvarea constă în înjumătățirea intervalului şi stabilirea acelei jumătăți de interval în care se găseşte soluția ecuației. Acest lucru se face testând semnul în capetele unuia din subintervale, cu decizia ca, în cazul în care semnele în capete sunt diferite, să alegem acea jumătate pentru pasul următor.

Avantajul metodelor directe, deci şi a acestei metode este acela că soluția unei ecuații, dacă există, poate fi determinată exact, fără aproximații de precizie. Această valoare exactă poate fi dată fie de limita stânga a intervalului inițial, fie de limita dreaptă a intervalului inițial, fie de unul din mijloacele succesiv determinate.

2.4.2. ESTIMAREA APRIORI ŞI APOSTERIORI A ERORILOR

Plecând de la intervalul [a, b] după n divizări se ajunge la intervalul $[a_n, b_n]$ în care căutăm soluția ecuației. Se poate arăta că (relația 2.7):

$$b_n-a_n = (b-a)/2^n \qquad (2.7)$$

pe baza faptului că la fiecare divizare, lungimea intervalului pe care se continuă calculele este jumătate din cea anterioară. În concluzie, după n divizări lungimea unui interval care conține soluția este de 2^n ori mai mică decât cea inițială.

Estimarea apriori a erorii se face ținând cont că *alfa* şi x_n aparțin $[a_n, b_n]$. Avem, deci:

$$|x_n - alfa| < |a_n - b_n| = b_n-a_n = (b-a)/2^n \qquad (2.8)$$

De aici rezultă că:

$$(b-a)/2^n < eps, \text{ adică } n = [\ln((b-a)/eps) / \ln 2] + 1 \qquad (2.9)$$

În cazul *estimării aposteriori* a erorii avem:

$$|a_n - x_n| = |b_n - x_n| = |b_n - a_n| / 2^n \quad (2.10)$$

de unde rezultă:

$$|b_n - a_n| = 2* |a_n - x_n| \text{ şi } |b_n - a_n| = 2*|b_n - x_n| \quad (2.11)$$

Deoarece în final $a_n = x_n$ (sau $b_n = x_n$) avem:

$$|a_n - b_n| = 2*|x_{n-1} - x_n| \quad (2.12)$$

Dar, deoarece *alfa* şi x_n aparţin $[a_n, b_n]$ putem scrie:

$$|x_n - alfa| < |a_n - b_n| = 2*|x_n - x_{n-1}| \quad (2.13)$$

Aşadar, pentru a obţine soluţia aproximativă a ecuaţiei cu eroarea *eps*, va trebui să calculăm termenii şirului până când condiţia devine:

$$2*|x_n - x_{n-1}| < eps \quad (2.14)$$

2.2.1.1. Algoritmul 2.1. Bisecţia pentru polinoame

```
// Funcţia întoarce:
//       True (1) – in caz de succes (s-a găsit rădăcina);
//       False (0) – in caz de eşec.
intreg BisPol
(
   intreg n,   // gradul polinomului
   real A[],   // coeficienţii polinomului
   real ls,    // limita stânga a intervalului
   real ld,    // limita dreapta a intervalului
   real er,    // eroarea de aproximare.
   real *pRad  // adresa rădăcinii (pointer)
)
{
// variabile locale
real xm;       // jumătatea intervalului.

// instrucţiunile ce modelează algoritmul metodei
daca (valPol(n,A,ls) * valPol(n,A,ld) > 0) return False;
daca (valPol(n,A,ls) == 0) {*pRad = ls; return True;}
daca (valPol(n,A,ld) == 0) {*pRad = ld; return True;}

xm=(ls+ld)/2; // condiţia de intrare in ciclul 'while'
cat timp((fabs(ld-ls) > er)  ŞI  valPol(n,A,xm) != 0)
{
   xm=(ls+ld)/2;
   daca (valPol(n,A,xm) * valPol(n,A,ls) <0) ld = xm;
   altfel ls = xm; // se stabilesc noile limite de interval
}
*pRad = xm; // stabilesc valoarea rădăcinii drept mijlocul
         // ultimului interval ce satisface condiţiile
         // buclei 'while()' anterioare.
return True; // succes: am găsit rădăcina .
}
```

Obs.: Funcția '`valPol()`', ce calculează valoarea unui polinom într-un punct (folosind *schema lui Horner*, sau echivalent, teorema împărțirii cu rest) este prezentată în *Anexa A*.

2.2.1.2. Algoritmul 2.2. Bisecția pentru ecuații transcendente

```
// functia intoarce:
//       True (1) - in caz de succes (s-a găsit rădăcina);
//       False (0) - in caz de eşec.
intreg Bis_Functie
(
   Adr_functie,      // adresa funcţiei din ecuaţia f(x)=0
   real ls,    // limita stânga a intervalului de calcul
   real ld,    // limita dreapta a intervalului de calcul
   real er,    // eroarea de calcul (precizia impusa)
   real *pRad  // adresa variabilei rădăcina
)
{
// declaraţiile variabilelor locale:
   real xm;    // mijlocul intervalului .

// instructiunile functiei
daca(*Adr_functie)(ls) * (*Adr_functie)(ld)>0) return False;
daca (*Adr_functie)(ls) == 0) {*pRad = ls; return True;}
daca (*Adr_functie)(ld) == 0) {*pRad = ld; return True;}


xm=(ls+ld)/2; // conditia de intrare in ciclul 'while'
cat timp((fabs(ld-ls) > er) SI (Adr_functie (xm) != 0) )
{
   xm=(ls+ld)/2;
   daca ((*Adr_functie)(xm) * (*Adr_functie)(ls)<0) ld = xm;
   altfel ls = xm; // se stabilesc noile limite de interval
}

*pRad = xm;    // stabilesc valoarea radacinii drept
               // mijlocul ultimului interval ce satisface
               // conditiile buclei 'while()' anterioare.
return True;   // succes: am gasit radacina .
}
```

Obs.: Notațiile '*Adr_functie*' şi '*(*Adr_functie)*' sunt simbolice. Ele reprezintă, respectiv, adresa unei funcții (deci un pointer la o funcție) şi apelul indirect al unei funcții, prin intermediul pointerului către acea funcție. Vedeți *Anexa B* - "*Noțiuni de C necesare desfăşurării lucrărilor de laborator*" pentru lămuriri.

2.5. METODE INDIRECTE PENTRU REZOLVAREA ECUAȚIILOR

2.5.1. METODA APROXIMAȚIILOR SUCCESIVE (metoda contracției)

Fie funcția $f:[a,b]\rightarrow$ continuă şi derivabilă pe (a,b) şi ecuația $f(x)=0$, care pe intervalul $(c,d)\subset(a,b)$ are o rădăcină unică α, $f(\alpha)=0$. Presupunem că ecuația

$$f(x)=0$$

se poate scrie sub forma:

$$x=\varphi(x) \tag{2.15}$$

Pentru x_o o aproximație inițială, avem următoarele succesiuni de aproximații:

$$x_1=\varphi(x_0),\ x_2=\varphi(x_1),\ x_3=\varphi(x_2),\ \ldots,\ x_n=\varphi(x_{n-1}) \tag{2.16}$$

Ca urmare formula $x_n=\varphi(x_{n-1})$ reprezintă o *formulă de iterație*.

Calculul soluției prin metoda aproximațiilor succesive se face funcție de o eroare impusă care dă criteriul de oprire al programului. Algoritmul se încheie atunci când:

$$|x_n-x_{n-1}|<\varepsilon \tag{2.17}$$

ceea ce arată că diferența dintre două valori consecutive calculate este mai mică decât ε, sau când:

$$|f(x_n)|<\eta \tag{2.18}$$

şi atunci soluția ecuației este aproximată de x_n, pentru η suficient de mic.

2.2.2.1. Algoritmul 2.3. Aproximații succesive

```
// funcția întoarce: True (1) - succes, s-a găsit rădăcina;
//                   False (0) - in caz de eşec.
intreg  Aprox_Succesive
(
   Adr_functie, // adresa functiei fi(x)(pointer la functie)
   real ls,     // limita stânga a intervalului de calcul
   real ld,     // limita dreapta a intervalului de calcul
   real x0,     // punctul de start - aproximația inițiala
                // necesara algoritmului
   real er,     // eroarea de calcul
   real h,      // pasul de derivare intre doua abscise
   real *pRad   // adresa variabilei radacina (pointer)
)
{
// declarațiile variabilelor locale:
real pc;       // punctul curent de derivare.
real xn, xn_1; // aproximatiile curentă şi respectiv
               // anterioara ale valorii radacinii.
real der;      // variabila in care se retine valoarea
               // derivatei funcției fi(x).
```

```
// instrucţiunile funcţiei
   pc = ls;     // pornesc de la valoarea 'ls' pentru
                // verificarea derivatei functiei fi(x) pe
                // intervalul dat.

// verifica daca functia fi(x) are derivata subunitara pe
// intreg intervalul considerat.
repeta
 {
   der = ((*Adr_functie)(pc+h) - (*Adr_functie)(pc)) / h;
   daca (fabs(der) >=1) return False;
   pc=pc+h;     // trec la urmatoarea abscisa, cu un pas cat
                // mai mic.
 } cat timp(pc<=ld);

xn = x0;
repeta
 { // modific pe 'x0' pana când este îndeplinita
   // condiţia de oprire.
   xn_1 = xn;
   xn = (*Adr_functie)(xn_1);
 } cat timp(fabs(xn-xn_1) >= er);

*pRad = xn;     // valoarea radacinii este valoarea lui 'xn'
                // cu care se încheie ciclul imediat anterior
return True;    // marchez succesul în găsirea rădăcinii
}
```

Obs.: Notaţiile '*Adr_functie*' şi '*(*Adr_functie)*' sunt simbolice. Ele reprezintă, respectiv, adresa unei funcţii (deci un pointer la o funcţie) şi apelul indirect al unei funcţii, prin intermediul pointerului către acea funcţie. Vedeţi *Anexa B* - "*Noţiuni de C necesare desfăşurării lucrărilor de laborator*" pentru lămuriri.

2.2.3. METODA NEWTON-RAPHSON (metoda tangentei)

Aplicând metoda aproximaţilor succesive cu *viteză de convergenţă mărită* pentru care considerăm $\xi = x_n$.

Rezultă:

$$x_{n+1} = x_n + \frac{1}{1-\varphi'(x_n)}\left(\varphi(x_n) - x_n\right) \tag{2.19}$$

Această expresie reprezintă o formulă de iteraţie pentru determinarea soluţiei ecuaţiei $x = \varphi(x)$ şi se numeşte *formula Newton - Raphson*. Formula de iteraţie poate fi reprezentată sub forma:

$$x_{n+1} = g(x_n), \quad \text{unde} \quad g(x) = \frac{\varphi(x) - x\,\varphi'(x)}{1-\varphi'(x)} \tag{2.20}$$

Condiţia de convergenţă a şirului spre soluţia ecuaţiei este dată de relaţia (teorema contracţiei):

$$\left|g'(x)\right| < 1 \tag{2.21}$$

Formula Newton-Raphson (2.19) poate fi scrisă şi pentru ecuaţia implicită $f(x) = 0$ astfel:

$$x_n = x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})} \tag{2.22}$$

Pentru anumite situaţii, relaţia (2.22) generează o nedeterminare de tip *Divide by zero*, marcată ca atare de către compilator. Este de exemplu cazul pentru care utilizatorul nu a studiat în prealabil condiţiile de existenţă ale unei anumite ecuaţii, pentru intervalul pe care l-a ales, şi într-unul din punctele de pe abscisă, derivata funcţiei se anulează.

Pentru aceste situaţii se poate adapta relaţia (2.22) în modul următor. Se va păstra numărătorul neschimbat, iar pentru numitor, acesta este stabilit la expresia $f(x_0)$.

Astfel, relaţia (2.22) devine:

$$x_n = x_{n-1} - \frac{f(x_{n-1})}{f'(x_0)} \tag{2.22'}$$

Geometric, acest lucru semnifică păstrarea unei pante constante, la fiecare pas. Panta la care ne referim este cea calculată în punctul de start (x_0).

Această adaptare a metodei tangentei poartă denumirea de *metoda coardei*. Denumirea de coardă provine din aceea că, într-o reprezentare grafică a situaţiei, vom observa că la fiecare pas mai puţin primul, se construiesc drepte paralele cu prima tangentă (doar prima dreaptă este tangentă la graficul funcţiei). Aceste drepte sunt apoi intersectate cu axa absciselor şi astfel determină valorile ulterioare de pe abscisă (x_k) din şirul de aproximaţii ale soluţiei.

Comparativ cu metoda tangentei, metoda coardei este mai lentă, dar mai generală. În rest, toate celelalte particularităţi ale metodei tangentei se păstrează.

2.2.3.1. Algoritmul 2.4. Newton-Raphson pentru ecuaţii transcendente

```
// functia intoarce: True (1) - succes, s-a gasit radacina;
//                   False (0) - in caz de esec.
intreg  NewtonRaphson
(
   Adr_functie, // referinta la functia f(x), din f(x)=0
   real x0,     // valoarea initiala necesara algoritmului
   real er,     // eroare de aproximare  radacinii
   real h,      // valoarea pasului de derivare
   int nMax     // numarul maxim de iteratii impus pentru
                // aflarea solutiei cu eroarea 'er' .
   real *pRad   // adresa variabilei rădăcină
)
{ // declaratiile variabilelor locale:
   real xn, xn_1; // valoarea aproximată curent si anterior
                  // a radacinii
   real der;   // valoarea derivatei
```

```
// instructiunile metodei
   xn=x0;
   repetă
   {
      xn_1=xn;
      der=((*Adr_functie)(xn_1+h)-(*Adr_functie)(xn_1))/h;
      dacă (der == 0) return False;

      xn = xn_1 - (*Adr_functie)(xn_1)/der;
      nMax = nMax-1;        // sau:     nMax--;
   } cat timp((fabs(xn-xn_1)>=er) SI (nMax>0));

   dacă (nMax= =0) return False;
   *pRad=xn;
   return True;       // succes.
}
```

Obs.: Notaţiile '*Adr_functie*' şi '*(*Adr_functie)*' sunt simbolice. Ele reprezintă, respectiv, adresa unei funcţii (deci un pointer la o funcţie) şi apelul indirect al unei funcţii, prin intermediul pointerului către acea funcţie. Pentru lămuriri consultaţi *Anexa B* - "*Noţiuni de C necesare desfăşurării lucrărilor de laborator*".

2.2.4. METODA NEWTON-RAPHSON PENTRU POLINOAME

Această metodă mai poartă numele şi de metoda Birge-Vieta.

Fie ecuaţia polinomială $P(x) = 0$:

$$P(x) = a_m x^m + a_{m-1} x^{m-1} + ... + a_1 x + a_0$$

Aplicăm formula de calcul (2.8) considerând o funcţie polinomială, astfel că avem:

$$x_n = x_{n-1} - \frac{P(x_{n-1})}{P'(x_{n-1})} \tag{2.23}$$

Şi pentru ecuaţiile polinomiale rămân valabile comentariile legate de metoda coardei.

2.2.4.1 Algoritmul 2.5. Newton-Raphson pentru polinoame

```
// funcţia întoarce: True (1) - success, s-a găsit rădăcina
//                   False (0) - in caz de eşec
intreg  Newton_Raphson_P
(
   intreg n,          // gradul polinomului
   intreg nMax,       // numărul maxim impus de iteraţii
   real  A[],         // vectorul coeficienţilor polinomului
   real x0,           // valoarea de start
   real er,           // eroarea de calcul
   real *pRad         // rădăcina ecuaţiei
)
{ // declaraţiile variabilelor locale
real xn, xn_1; // valoarea curentă respectiv anterioara a
               // rădăcinii
```

```
xn = x0;
repetă
{
   xn_1=xn;
   dacă (Derpol(n,A,xn_1)==0) return False;

   xn=xn_1-Valpol(n,A,xn_1)/Derpol(n,A,xn_1);
   nMax=nMax-1;      // sau:  nmax--;
} cât timp((fabs(xn-xn_1)>=er)  ŞI (nMax>0) );

dacă (nMax == 0)
   return False;     // numarul de pasi este insuficient
*pRrad = xn;
return True;
}
```

Obs.:

Funcţia '`valPol()`', ce calculează valoarea unui polinom într-un punct (folosind *schema lui Horner*, sau echivalent, teorema împărţirii cu rest) împreună cu funcţia `derpol()` - pentru calculul valorii derivatei unui polinom într-un punct, sunt prezentate în *Anexa A*.

2.2.5. METODA LUI BAIRSTOW

Această metodă determină toate rădăcinile atât reale cât şi complexe ale unei ecuaţii polinomiale cu coeficienţi reali. Fie ecuaţia:

$$P_n(x) = x^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + ... + a_1x + a_0 = 0 \qquad (2.24)$$

care admite soluţiile $x_1, x_2, x_3, ..., x_n$ care pot fi reale şi complexe. Se descompune polinomul (2.24) în produs de două polinoame unul de gradul doi şi altul de grad `n-2`.

Două soluţii ale polinomului iniţial sunt cele două rădăcini ale polinomului de gradul doi care pot fi calculate uşor şi ele pot fi reale sau complexe.

Vom descompune polinomul iniţial astfel:

$$a_n x^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + a_{n-3}x^{n-3} + ... + a_1x + a_0 =$$
$$\left(b_n x^{n-2} + b_{n-1}x^{n-3} + b_{n-2}x^{n-4} + ... + b_3x + b_2\right)\left(x^2 + s_n x + p_n\right) + Sx + R$$

Se rezolvă ecuaţia de grad 2, şi procesul se reia, pentru polinomul de grad n-2, care la rândul său va fi scris ca un produs de două polinoame, unul de grad 2 şi altul de grad (n-2)-2.

Algoritmul 2.6 respectă notaţiile de mai sus.

2.2.5.1. Algoritmul 2.6. Metoda lui Bairstow

```
   // funcţia întoarce:
   //     True  (1) – in caz de succes (s-a găsit rădăcina);
   //     False (0) – in caz de eşec.
   întreg Bairstow
   (
         întreg grad, // gradul polinomului, întreg
         real a[ ],   // vectorul coeficienţilor polinomului
         real eps,    // eroarea de calcul
         real s0,           // valoarea de start pentru sn
         real p0,           // valoarea de start pentru pn
         intreg niter,      // numărul de iteraţii
         real *solr,  // vectorul părţilor reale ale soluţiilor
         real *soli   // vectorul părţilor imaginare ale
                      // soluţiilor
)
   { // declaratiile variabilelor locale
      real b[ ];             // vectorul b conform descompunerii;
      real bs[ ], bp[ ];     // vectorii derivatelor polinomului
                             // in raport cu s, respectiv p;
      real S, R;             // coeficienţii restului;
      real Ds, Dp;     // variatiile lui sn respectiv pn
      real Ss, Sp;     // derivatele lui S în raport cu s
                       // respectiv p.
      real Rs, Rp;     // derivatele lui R în raport cu s
                       // respectiv p.
      real sn, pn;     // coeficienţii polinomului de gradul 2
      real irad;       // indicele rădăcinii curente întreg;
      int  i, k;       // indici întregi;

      pentru  i=0,grad  a[i]=a[i]/a[grad];// normarea lui a
      irad = 1;
      cât timp (grad>2)
      {
         sn = s0; pn = p0;
         repetă
         { // repeta-pana cand
         // calculul polinomului B
                  b[grad]=1;  b[grad-1]=a[grad-1]-sn;
                  pentru  k =[grad+2, 2]
                  b[k] = a[k]-sn*b[k+1]-pn*b[k+2];

         // calculul coeficienţilor restului
                  S=a[1]-b[2]*sn-b[3]*pn;  R=a[0]-b[2]*pn;

         // calculul derivatei polinomului b în raport cu s
                     bs[grad] =0;  bs[grad-1] =-1;
                     pentru  k=grad-2 până la  2 execută
                        bs[k] =-b[k+1]-sn*bs[k+1]-pn*bs[k+2];

   // derivarea coeficientilor restului în raport cu s
                     Ss=-b[2]-sn*bs[2]-pn*bs[3];
                     Rs=-bs[2]*pn;

   // calculul derivatei polinomului b în raport cu p
                     bp[grad] =0;   bp[grad-1] =0;
                     pentru  k=grad-2 până la  2 execută
                        bp[k] =-sn*bp[k+1]-b[k+2]-pn*bp[k+2];
```

```
    // derivarea coeficientilor restului în raport cu p
                      Sp=-bp[2]*sn-b[3]-pn*bp[3];
                      Rp=-bp[2]*pn-b[2];

    // Se aplică metoda Newton-Raphson pentru rezolvarea
    // sistemelor de ecuaţii neliniare
    dacă Ss*Rp-Sp*Rs=0  return False;
           // Ss*Rp-Sp*Rs este det. sistemului
              Ds=(-S*Rp+R*Sp)/(Ss*Rp-Sp*Rs);
                       Dp=(-Ss*R+Rs*S)/(Ss*Rp-Sp*Rs);
                       sn=sn+Ds;    pn=pn+Dp; // noile aproximaţii
                                              // ale lui sn şi pn
                       i=i+1; // incrementez contorul de iteraţii
                  }cât timp(((|Ds|>=eps)SI(|Dp|>=eps))SAU(i<=niter));
                  dacă (i>niter) atunci  return False;

EcGr2(1,sn,pn,solr[irad],soli[irad],solr[irad+1],soli[irad+1]);
// rezolvă ec. de gradul 2
       grad=grad-2;
       pentru k=grad până la 0  a[k]=b[k+2];// formez noua ecuaţie
       irad=irad+2;      // incrementez indicele  rădăcinii
    } // cat timp(grad>2) .

            dacă (grad = = 1) {
                 // rezolvă ecuaţia de gradul 1
                 solr[irad]=EcGr1(a[1],a[0]);
                 soli[irad]=0;
               return True;
               }

    dacă (grad = = 2) {
    // rezolvă ec. de grd. 2
    EcGr2(a[2],a[1],a[0],solr[irad],soli[irad],solr[irad+1],soli[irad+1]);

    return True;
    }
} // sfârşitul funcţiei.
```

3. DESFĂŞURAREA LUCRĂRII

3.1. ÎNTREBĂRI

1. De cine depinde viteza de convergenţă în cazul metodelor: bisecţiei, aproximărilor succesive, tangentei? Şi ce se poate face pentru a mări viteza de convergenţă în aceste cazuri?
2. Care sunt avantajele metodei bisecţiei faţă de tangentă şi invers?
3. Care sunt asemănările şi deosebirile între metoda aproximărilor succesive şi a tangentei?

4. Se poate estima apriori numărul de paşi necesari rezolvării ecuaţiei:
   $5e^{2x-1}-2x-4 = 0$

folosind metoda bisecției, când soluția se caută în intervalul `[0,2]` şi eroarea este: `0.00001`? Determinați.

5. Este corectă următoarea funcție fi(x) Pentru ce metodă este utilă?

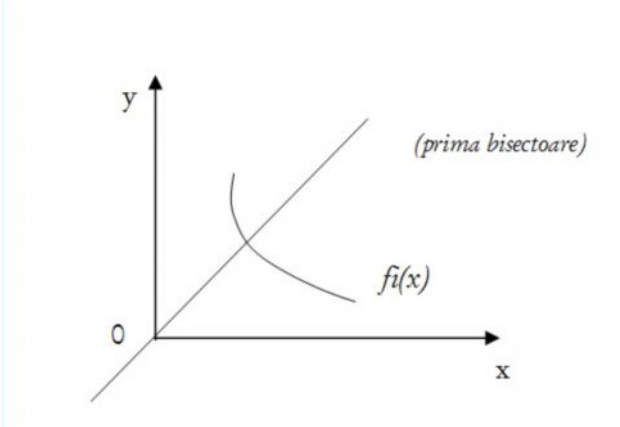


Fig. 2.1 - O caracteristică fi(x) *pentru care se pune problema de valabilitate.*

3.2. PROBLEME LABORATOR

1. Să se implementeze algoritmii 2.1-2.6 conform limbajului *C standard* (ANSI C). Pornind de la pseudocodul rutinei de calcul, se vor concepe programe complete.

2. Se determină numărul de rădăcini reale ale polinomului de la numitorul funcției de transfer. Se aplică programul Şturm care implementează metoda de formare a şirului lui Şturm pentru numitorul funcției de transfer. Programul Şturm este în directorul de lucru. Se scrie şirul lui Şturm. Primii doi termeni ai şirului sunt polinomul dat şi derivata lui. Se completează tabelul 1.

Tabelul 1

	P(s)	$P_1(s)$	$P_2(s)$	$P_3(s)$	n_-,n_+
P(-∞)					
P(+∞)					

Diferența dintre numărul de schimbări de semn a şirului lui Şturm la -∞ şi la +∞ reprezintă numărul de rădăcini reale ale polinomului.

3. Se încadrează soluțiile în intervale. Se scrie şirul
   P(s)=
   P`(s)=
   P``(s)=

Se rezolvă fiecare ecuaţie cu o metodă aleasă (de exemplu bisecţia) şi se completează tabelul 2. L_i şi L_s se aleg astfel ca soluţiile polinomului să fie cuprinse între ele (de exemplu -1000 şi 1000).

Tabelul 2

P''(s)	L_i		s_0	L_I
P'(s)	|	s_1	|	s_2 |
P(s)	| s_3	s_1 |	s_2	|

s_1,s_2,s_3 sunt polii funcţiei de transfer.

4. Se dă următorul *filtru*:

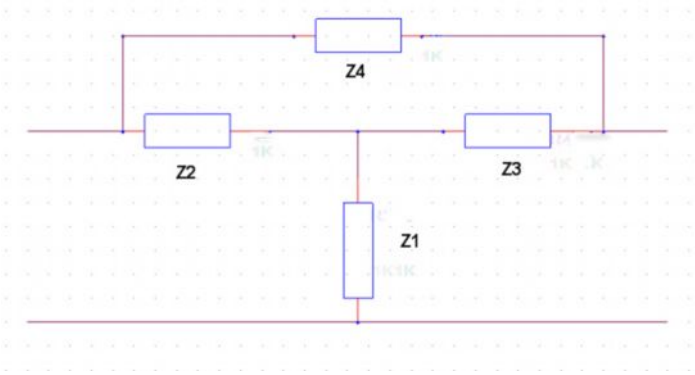


Fig. 2.2. - Filtru pasiv, pentru problema 4.

Cunoscând C_1=100mF, C_2=121mF, R=12.5 **KΩ** si $Z_1=1/j\omega C_1$, $Z_2=1/j\omega C_2$, $Z_3=Z_4=R$, să se determine modulul funcţiei de transfer şi valoarea frecvenţei pentru care acesta este de valoare 10. Folosiţi la alegere una din metodele de rezolvare numerică a ecuaţiilor polinomiale.

5. Să se calculeze creşterea de temperatură ΔT pentru care curentul invers al unei joncţiuni de siliciu se dublează (T=300K)
Indicaţie:
$I_r(T)=const*T^{3/2}exp(-Eg/2KT)$
şi
$Eg/KT=1.11/0.026$
Folosiţi metoda aproximărilor succesive, dacă acest lucru este posibil.

6. Se consideră un amplificator integrat, având în buclă deschisă o amplificare de forma:

$$H(s)=\frac{338.6260295}{s^3+0.5284s^2+0.05120153s+0.00096807437}$$

Să se determine polii amplificatorului.

7. Amplificarea buclei unui amplificator integrat prevăzut cu reacţie pur rezistivă este:

$$T(s)=T_0/(1+10s+30s^2+20s^3),$$

unde s este măsurat în microsecunde^{-1}. Valoarea mărimii T_0 este 12. Să se determine poziţia polilor câştigului în buclă închisă pentru această valoare a lui T_0 utilizându-se metoda bisecţiei, metoda aproximaţiilor succesive, metoda Bairstow şi metoda Newton-Raphson.

3.3. TEME DE CASĂ

1. Se consideră o *joncţiune pn* a cărei temperatură este menţinută la valoarea T=300K. La ce tensiune directă, V_a, aplicată acestei joncţiuni, curentul este de 10 ori mai mare decât curentul rezidual?
Indicaţie:
$I_a = I_o[\exp^{(qVa/mKT)}-1]$
şi $m=1...2$. Folosiţi *metoda tangentei*.

2. Fie un tranzistor cu efect de câmp a cărui lege este :
$i_D=I_{DSS}(1-U_{GS}/U_T)^2$
cu $I_{DSS}=9mA$ şi $U_T=-3V$, $R_S=0.25$ KΩ
   a) Să se determine curentul i_D.
   b) Între ce limite poate varia rezistenta R_d astfel încât tranzistorul să funcţioneze în regiunea de saturaţie? Folosiţi *metoda bisecţiei*.

3. Rezolvaţi ecuaţia de la problema 2 prin metodele bisecţiei si tangentei şi analizaţi rezultatele obţinute :
   a) după primii 3 paşi - prin calcul numeric, fără program;
   b) după finalizarea programelor.

4. Se consideră funcţia de transfer a unui filtru analogic de tip Cebîşev dată sub forma: $H(s)=\frac{0.04381}{s^4+0.6192s^3+0.61401692s^2+0.20379268s+0.0491588}$

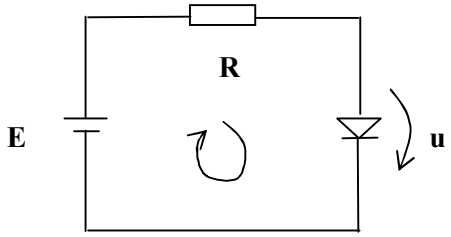
Să se determine polii filtrului.

5. Se dă ecuația:

$$e^x - 1/x = 0$$

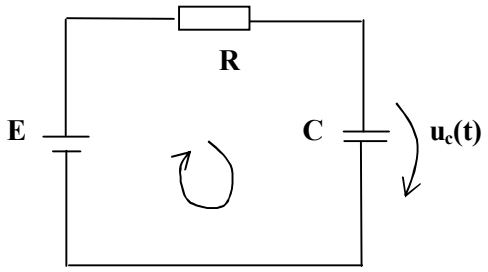
care are o soluție în intervalul [0.1, 1]. Să se calculeze soluția ecuației aplicând metoda bisecției pentru ecuații transcendente, metoda Newton-Raphson şi metoda aproximațiilor succesive. Eroarea de calcul se consideră $\varepsilon = 0.0000000001$ (ce se poate scrie şi `1e-10`, conform *notației ştiințifice*) pentru toate metodele. Să se compare rezultatele.

6. Să se determine valoarea tensiunii şi a curentului prin diodă, în situația următorului circuit (cel de *polarizare a diodei*). Ca metodă de rezolvare a ecuației, având necunoscuta *u* (tensiunea pe diodă), se va folosi metoda directă a Bisecției.



Valorile de calcul pot fi:
R=10Kohm, E = 10V
I_0 = 1pA sau 1nA
V_T = 25 mV.

7. Aplicând metodele indirecte de rezolvare a ecuațiilor studiate în laborator - Aproximații succesive şi metoda tangentei (*Newton-Raphson*) - calculați timpul după care tensiunea pe condensatorul din circuitul ce urmează, atinge valoarea 0.5*E (regimul tranzitoriu al condensatorului).



Valorile de calcul utile sunt:

$$E = \begin{cases} 0, & t<0 \\ 1, & t>0 \end{cases} \quad \text{în volți.}$$

$$\tau = RC = 1s$$

BIBLIOGRAFIE

1. I. Rusu - *Metode numerice în electronică. Aplicații în limbaj C*, Editura Tehnică, Bucureşti, 1997.
2. I. Rusu, Vlad. Al. Grosu, Grigore Țiplea - *Îndrumar de laborator pentru metode numerice, cu aplicații în limbaj C*, Litografia UPB, edițiile 1998 şi 2000.
3. W. Dorn, S. McCracken - *Metode numerice cu programare in FortranIV*, Ed. Tehnică, Bucureşti, 1976.
4. B. Demidovitch, I.A.Maron - *Computational Mathematics*, Ed. Mir, 1989
5. Catalog IPRS Băneasa - *Tranzistoare cu siliciu*, Bucureşti, 1989.