DIDACTICA MATHEMATICA, Vol. 33(2015), pp. 51–66

REZOLVAREA NUMERICĂ A ECUAŢIILOR NELINIARE CU MAPLE

Andra-Monica Manu

Abstract. The paper aims to present several iterative methods for solving algebraic equations. The speed of convergence and error estimates are given. The main Maple commands used in computations are summarized in the introduction.

MSC 2000. 97N40, 97N80.

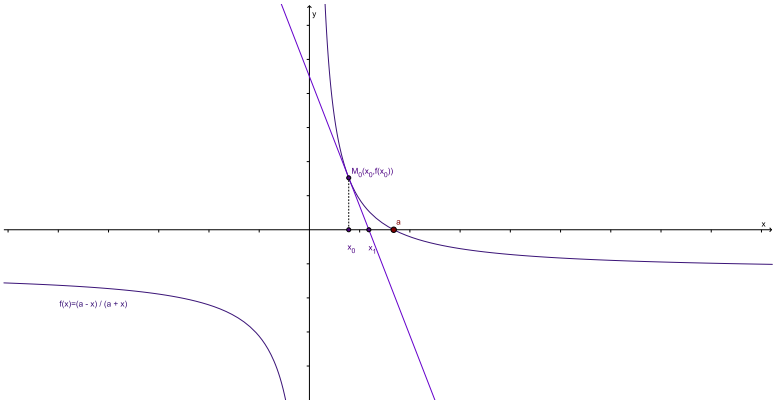
Cuvinte cheie. Iterative sequence, Newton's method, Maple programming.

1. INTRODUCERE

Una dintre problemele cele mai întâlnite în matematica aplicată este determinarea rădăcinilor unei ecuaţii de forma $f(x) = 0$. De cele mai multe ori nu poate fi găsită o soluţie explicită, caz în care ne mulţumim cu aflarea unei rădăcini α cu un anumit grad de exactitate. Metodele prin care se determină numeric soluţiile unei ecuaţii sunt cunoscute ca metode iterative. Ele presupun construirea unui şir de aproximaţii ale rădăcinii căutate, care să conveargă la aceasta. Vom considera o funcţie f de variabilă reală x şi α o rădăcina a ecuaţiei $f(x) = 0$. Cerem ca funcţia noastră să fie continuă şi diferenţiabilă. Metodele iterative pentru această clasă generală de funcţii impun cunoaşterea uneia sau mai multor valori iniţiale, ce reprezintă în fapt primele elemente ale şirului de aproximaţii. Valori iniţiale se determină din context sau pot fi estimate pe baza graficului funcţiei considerate.

Pentru a ilustra modalitatea de construcţie a unei metode iterative, vom prezenta un exemplu în care vom analiza convergenţa metodei, viteza de convergenţă şi eroarea.

EXEMPLUL 1. Fie funcţia $f(x) = \dfrac{a-x}{a+x}$, unde $a > 1$ este dat.



Putem observa că $x = a$ este soluţie a ecuaţiei. Notăm punctul iniţial cu x_0. În punctul $(x_0, f(x_0))$ ducem tagenta la curbă, care va intersecta axa absciselor în punctul $(x_1, 0)$. Abscisa obţinută ar trebui să fie o aproximaţie mai bună pentru rădăcina α. Pentru a obţine o ecuaţie în x_1, scriem ecuaţia tangentei şi înlocuim pe x cu x_1:

$$0 - f(x_0) = f^{'}(x_0)(x_0 - x_1)$$

Înlocuind expresia funcţiei şi făcând simplificările necesare, avem:

$$x_1 = x_0 + \frac{(a - x_0)(a + x_0)}{2a}.$$

Repetăm procesul prin înlocuirea lui x_0 cu x_1 la infinit şi obţinem următoarea formulă iterativă:

$$x_{n+1} = x_n + \frac{(a - x_n)(a + x_n)}{2a}.$$

Pentru a verifica dacă şirul de iterate converge la soluţia noastră, vom verifica dacă eroarea noastră converge la 0. Notăm $e_n = a - x_n$ şi prelucrăm formula iterativă pentru a ajunge la o expresie în funcţie de erorile a două iterate succesive.

$$a - x_{n+1} = a - x_n - \frac{(a - x_n)(a + x_n)}{2a} = (a - x_n)(1 - \frac{(a + x_n)}{2a}) = \frac{(a - x)^2}{2a}$$

Aşadar $e_{n+1} = \frac{e_n^2}{2a}$. Inductiv se obţine $e_n = \frac{e_0^{2^n}}{(2a)^{n+1}}$. Cum $\frac{1}{2a}$ este subunitar, şirul (e_n) converge la 0 dacă şi numai dacă $|e_0| < 1$, care în mod echivalent se scrie:

$$a - 1 < x_0 < a + 1.$$

Pentru ca şirul (x_n) să conveargă la a este necesar şi suficient ca x_0 să fie ales în intervalul de convergenţă $[a - 1, a + 1]$. În ceea ce priveşte viteza convergenţei, analizăm cât de repede descreşte eroarea relativă:

$$e_{n+1} = \frac{e_n^2}{2a} \iff \frac{e_{n+1}}{a} = \frac{e_n^2}{2a^2}$$

$$Rel(x_{n+1}) = \frac{1}{2} Rel(x_n)^2, n \geq 0,$$

unde prin $Rel(x_n)$ am notat eroarea relativă $\frac{\alpha - x_n}{\alpha}$. Deci, putem spune că eroarea descreşte pătratic la zero.

Exemplul furnizează construcţia unei metode iterative pentru rezolvarea unei ecuaţii. De asemenea s-a realizat şi o analiză a convergenţei, ea incluzând demonstrarea convergenţei, determinarea intervalului de convergenţă pentru alegerea punctului iniţial şi a vitezei de convergenţă. Vom introduce în paragraful următor câteva noţiuni, necesare caracterizării unui şir de iterate.

DEFINIŢIA 1. Spunem că şirul de iterate $(x_n)_{n \geq 0}$ converge cu ordinul $p \geq 1$ la un punct α, dacă

$$|\alpha - x_{n+1}| \leq c|\alpha - x_n|^p, n \geq 0 \tag{1}$$

pentru $c \geq 0$. Dacă $p = 1$, atunci spunem că şirul converge liniar la α, caz în care vom cere ca $c < 1$; constanta c este numită rata de convergenţă liniară a lui x_n la α.

După cum am văzut în exemplul de mai sus, ordinul de convergenţă este 2, numită şi convergenţă pătratică. În următoarele secţiuni vom expune patru metode iterative uzuale. Acestea se aplică unor de funcţii ce îndeplinesc condiţiile enumerate mai jos.

Mai întâi considerăm o funcţie $f : I \to \mathbb{R}$. Ne propunem să determinăm rădăcinile ecuaţiei $f(x) = 0$. Fie α o rădăcină izolată ecuaţiei, adică există un interval $[a, b]$ care conţine numai această rădăcină. Prin urmare, $f(\alpha) = 0$.

TEOREMA 1. **Prima teoremă a lui Bolzano-Cauchy.** Dacă funcţia $f : [a, b] \to (R)$este continuă pe $[a, b]$ şi $f(a)f(b) < 0$, atunci există cel puţin un punct $c \in (a, b)$ pentru care $f(c) = 0$.

Demonstraţie. Fie $a_1 = a, b_1 = b$ şi $c_1 = \dfrac{a_1 + b_1}{2}$ mijlocul intervalului $[a_1, b_1] = [a, b]$. Cum $f(a_1)f(b_1) < 0$ deducem că $f(a_1)f(c_1) < 0$ sau/şi $f(c_1)f(b_1) < 0$. Dacă $f(a_1)f(c_1) \leq 0$, atunci fie $a_2 = a_1, b_2 = c_1$, iar dacă $f(c_1)f(b_1) \leq 0$, atunci fie $a_2 = c_1, b_2 = b_1$. Evident

$$a = a_1 \leq a_2 < b_2 \leq b_1 = b, b_2 - a_2 = \frac{b-a}{2}, f(a_2)f(b_2) < 0.$$

Cu intervalul $[a_2, b_2]$ se procedează analog ca şi cu $[a_1, b_1]$. Fie $c_2 = \dfrac{a_2 + b_2}{2}$. Cum $f(a_2)f(b_2) \leq 0$ deducem că $f(a_2)f(c_2) < 0$ sau/şi $f(c_2)f(b_2) < 0$. Dacă $f(a_2)f(c_2) \leq 0$, atunci fie $a_3 = a_2, b_3 = c_2$, iar dacă $f(c_2)f(b_2) \leq 0$, atunci fie $a_3 = c_2, b_3 = b_2$. Evident

$$a = a_1 \leq a_2 \leq a_3 < b_3 \leq b_2 \leq b_1 = b, b_3 - a_3 = \frac{b-a}{2^2}, f(a_3)f(b_3) < 0.$$

Continuând tot aşa, obţinem două şiruri $(a_n), (b_n)$ de numere reale, caracterizate de următoarele proprietăţi:

1): $a \leq a_n \leq a_{n+1} < b_{n+1} \leq b_n \leq b$ oricare ar fi $n \in \mathbb{N}$;

2): $b_n - a_n = \dfrac{b-a}{2^n}$ oricare ar fi $n \in \mathbb{N}$;

3): $f(a_n)f(b_n) < 0$ oricare ar fi $n \in \mathbb{N}$;

Şirurilor $(a_n), (b_n)$, le aplicăm teorema lui Cantor; rezultă că acestea sunt convergente şi

$$\lim_{n\to\infty} a_n = \lim_{n\to\infty} b_n.$$

Fie $c = \lim_{n\to\infty} a_n$. Evident $c \in [a, b]$. Funcţia f fiind continuă, deducem că

$$\lim_{n\to\infty} f(a_n) = \lim_{n\to\infty} f(b_n) = f(c).$$

În baza proprietăţii 3), obţinem că $f^2(c) \leq 0$, deci $f(c) = 0$. Întrucât $f(a)f(b) < 0$, rezultă că $c \notin \{a, b\}$. Prin urmare $c \in (a, b)$ şi $f(c) = 0$. □

Interpretarea geometrică teoremei ar fi următoarea: dacă funcţia noastră ar avea semne diferite la capetele intervalului, atunci curba descrisă de aceasta ar trece de o parte şi de alta a axei Ox, intersectând-o cel puţin o dată.

Pentru a asigura existenţa rădăcinii, ţinând cont de teorema prezentată anterior, impunem asupra lui f următoarele condiţii:

- ➥ **1:** Funcţia f şi derivatele sale $f^{'}$ şi $f^{''}$ sunt continue pe $[a,b]$.
- ➥ **2:** Valorile funcţiei f la capetele intervalului sunt de semne diferite, adică $f(a)f(b)<0$.
- ➥ **3:** Derivatele $f^{'}$ şi $f^{''}$ păstrează semn constant în tot intervalul $[a,b]$.

Condiţiile 1 şi 2 ne asigură existenţa rădăcinii α, conform teoremei 1. Din condiţia 3, putem să deducem că funcţia f creşte sau descre c ste, deci se va anula o singură dată, adică rădăcina α este izolată. În plus, semnul lui $f^{'}$ indică faptul că f are aceeaşi direc tie, iar semnul lui $f^{"}$ arată că f este numai convexă sau numai concavă.

2. INTRODUCERE IN MAPLE

Aplicaţia Maple pune la dispoziţia utilizatorilor un mediu de calcul numeric şi simbolic, incluzând un număr de aproximativ 3000 de funcţii predefinite. Totodată oferă facilităţi pentru grafică 2D, 3D, inclusiv generare de animaţie. Maple include şi un limbaj de programare propriu, procedural, iar modalitatea de programare este una structurată. Utilizatorul are posibilitatea de a elabora propriile programe sau chiar să definească noi funcţii.

Maple dispune de trei componente de bază: *nucleul, biblioteca standard* şi aproximativ 40 de *pachete* specializate în rezolvarea unor tipuri de probleme mai complicate. Nucleul este scris în limbajul **C** şi realizează cea mai mare parte a calculelor efectuate de sistem. Biblioteca de bază se încarcă automat imediat ce a fost apelată o comandă, iar în cazul în care comanda nu face parte din bibliotecă, utilizatorul trebuie să încarce pachetul care o include. Orice pachet se încarcă în memorie cu comanda:

with(*nume pachet*):

Pentru început o vedem cum sunt constituite o serie de elemente ale sintaxei din Maple.

Expresiile sunt reprezentate ca secvenţe alcătuite din operanzi(variabile, constante, functii) şi operatori (de exemplu cei aritmetici). Ordinea efectuării operaţiilor se păstrează, iar în cazul în care vrem să evidenţiem o anumită ordine de efectuare, vom utiliza doar paranteze rotunde, deoarece parantezele drepte şi acoladele au semnificaţii predefinite.

Exemple:

> restart;

> x^3+ln(4)*x+y;

$$x^3 + 2\ln(2)\,x + y$$

> a^(n^2+2*n+1)-5*a;

$$a^{n^2+2\,n+1}-5\,a$$

> (x+y)/(x^4+y^4);

$$\frac{x+y}{x^4+y^4}$$

În Maple, variabilele sunt şiruri de caractere, care se iniţializează folosind comanda: *variabila := expresie*. În ceea ce priveşte declararea tipului acestora, ea poate fi omisă.

Există următoarele tipuri de de variabile:

1. **variabile legate:** - sunt cele care au primit o valoare în u mra unei atribuiri şi sunt folosite pentru a nota o expresie. De exemplu:
   > a:=x^3+4*x+y;

   $$a := x^3+4\,x+y$$
2. **variabile libere:** - joacă rolul unor necunoscute sau nedeterminate;
3. **variabile globale:** - au valori iniţiale predefinite, dar pot fi modificate pe parcursul operaţiilor de către utilizator sau de către sistem. De exemplu: **Digits**(numărul de cifre zecimale cu care vor fi afişate rezultatele numerice), **Order** (ordinul de trunchiere al unei serii), **libname** (locaţia bibliotecii standard), etc.

Constantele numerice în Maple sunt clasificate astfel:

1. **întregi:** - reprezintă numere întregi. Numărul maxim de zecimale nu trebuie să depăşească 500 000.
2. **raţionale:** - reprezentate sub forma $\frac{m}{n}$, cu m, n constante întregi. Se poate observa că Maple simplifică automat fracţiile.
3. **în virgulă mobilă:** - reprezentate prin următoarele câmpuri: partea întreagă, punctul zecimal, partea fracţionară, constanta e sau E şi un exponent. Constantele matematice nu pot fi folosite drept variabile. Exemplu:
   > -125/625);

   $$-1/5$$

   > 0.25678e-17;

   $$2.567800000\times 10^{-18}$$

   > 0.25678e-17;

   $$2.567800000\times 10^{-18}$$

Dacă o expresie conţine constante întregi, raţionale şi în virgulă mobilă, atunci constantele întregi şi raţionale sunt convertite în virgulă mobilă, rezultatul expresiei fiind în acesta caz o constantă în virgulă mobilă.

În scopul evaluării unei expresii în virgulă mobilă, chiar dacă toţi operanzii din expresie sunt întregi sau raţionali, se apelează comanda

evalf(*expresie, n*),

unde *expresie* reprezintă expresia ce trebuie evaluată, iar *n* este un parametru opţional, care specifică numărul de cifre semnificative. Dintre constantele matematice uzuale în Maple putem enumera:

Constanta	Nume **Maple**
Baza logaritmului natural *e*	**E**
Unitatea imaginară *i*	**I**
π	**Pi**
∞	**infinity**
- ∞	**-infinity**
Constanta lui Euler γ	**gamma**
Valoarea booleană *adevărat*	**true**
Valoarea booleană *fals*	**false**

Maple conţine o întreagă familie de funcţii de evaluare numerică şi algebrică a expresiilor:

1. **eval(*expr,* $x = x_0$):** - evaluează expresia în punctul dat x_0
2. **evalf(*expr*):** - evaluează numeric o expresie care conţine constante (*Pi, e, gamma*) sau funcţii matematice *ln, sin, cos, tan, exp,...*
3. **evala(*expr*):** - evaluează algebric o expresie
4. **evalb(*expr*):** - evaluează boolean o expresie
5. **evalm(*expr*):** - evaluează matriceal o expresie
6. **evalc(*expr*):** - evaluează o expresie în mulţimea numerelor complexe.

Exemple:

> eval(x^3-8, x = 2);

$$0$$

> evalf(Pi+ln(4));

$$4.527887015$$

> evala(Divide(x^2-5, x+sqrt(5)));

$$true$$

> evalb(-12 > 0);

$$false$$

> evalc(I^4-5*I+7);

$$8-5\,i$$

Maple pune la dispoziţia utilizatorului câteva din funcţiile elementare predefinite, după cum se poate vedea în tabelul de mai jos.

Funcţia	Nume **Maple**
e^x	*exp (x)*
ln x	ln(x)
$\log_{10} x$	*log10(x)*
sin x	*sin(x)*
cos x	*cos(x)*
tan x	*tan(x)*
arcsin x	*arcsin(x)*
arccos x	*arccos(x)*
arctg x	*arctan(x)*
arcctg x	*arccot(x)*

Pentru a defini o nouă funcţie se va utiliza una din metodele de mai jos:

1. ***nume funcţie funcţie := expresie*:** - funcţia este definită ca o expresie, iar *expresie* reprezintă o expresie analitică ce conţine parametrii şi variabile. Funcţia definită astfel se evaluează într-un anumit punct cu ajutorul comenzii **eval**.
2. ***nume funcţie := var → expresie*:** - unde *var* reprezintă variabila, iar *expresie* este expresia analitică ce conţine parametrii şi variabilele. Pentru a calcula valoarea funcţiei într-un punct este suficient să apelăm numele funcţiei având ca valoare a argumentului punctul respectiv.

Exemple:

> f := 5^x-4*x^2+12;

$$f := 5^x - 4\,x^2 + 12$$

> eval(f, x = 1/2);

$$\sqrt{5} + 11$$

> evalf(%);

$$13.23606798$$

> g := (x, y, z) → x^2+y^2+z^2-2*x*y*z;

$$g := (x, y, z) \mapsto x^2 + y^2 + z^2 - 2\,xyz$$

> g(1, 1, 1);

$$1$$

După cum am văzut la începutul capitolului, Maple oferă posibilitatea de a construi proceduri. Sintaxa unei proceduri în Maple este:

nume:=**proc**(param1,param2,...)
local lista declaraţii globale;
options lista de opţiuni;
instrucţiuni
end;

Parametrii care apar în scrierea unor proceduri se numesc **parametrii formali**, ei având un rol descriptiv (un parametru formal este o variabilă al cărei nume este

cunoscut, dar al cărei valori va fi identificat în momentul execuţiei). În cadrul listei, parametrii formali sunt separaţi prin virgulă. Dacă se urmăreşte ca procedura să întoarcă o valoare, atunci se utilizează funcţia predefinită

return(*v*)

în şirul de instrucţiuni din corpul procedurii. Apelul unei proceduri se face prin intermediul comenzii

nume(*lista parametrii actuali*);

În momentul execuţiei unei proceduri, parametrii actuali înlocuiesc parametrii formali definiţi în declaraţia de procedură.

Vom da un exemplu de procedura care returnează suma a două numere.

```
> suma := proc (a, b)
  local rezultat;
  rezultat := a+b;
  return rezultat;
end proc;
> suma(4, 5);
```

9

Instrucţiunile de decizie şi structurile repetitive în Maple sunt cele cunoscute din **C**, fiind utilizate în mod similar. Vom expune doar sintaxa lor, fără a le exemplifica (se va putea observa modul lor de folosire în implementarea metodelor iterative din ultima secţiune a acestui capitol).

```
if condiţie then
  instrucţiuni1;
else
  instrucţiuni2;
fi;


for i:=1 from expresie1 by expresie2 do
  instrucţiuni;
od;


while condiţie do
  instrucţiuni;
od;
```

3. FUNCII PREDEFINITE DE REZOLVARE A ECUAIILOR

În Maple ecuaţiile sunt expresii matematice scrise sub forma unor relaţii între anumite variabile şi valori cu ajutorul operatorului "=". Variabilele nu au valori explicite şi sunt neasignate. Cei doi membrii ai ecuaţiei pot fi referiţi cu ajutorul comenzilor **lhs** şi **rhs**. Maple poate să rezolve ecuaţii algebrice şi transcendente exact sau aproximativ, sau, poate să determine o soluţie simbolică a acestora. În

acest scop, softul dispune de o familie de funcţii predefinite, **solve**, care se folosesc ca şi parametrii.

Sintaxa acestor comenzi este

solve(*ecuatie,variabila*)

sau

solve(*ecuatii,variabile*)

Funcţia returnează soluţiile ecuaţiei sub forma unei secvenţe de expresii. Dacă comanda solve nu găseşte nicio soluţie sau nu poate determina o soluţie atunci se returnează secvenţa **NULL**.

Dacă rezultatul comenzii **solve** este o expresie definită pe ramuri, atunci poate fi folosită comanda **assuming** pentru a izola soluţiile dorite. În cazul în care rezultatul nu este definit pe ramuri, este posibil ca condiţiile puse asupra variabilelor independente în **assuming** să nu fie luate în considerare, însă dacă ecuaţiile date ca parametrii conţin necunoscute, atunci condiţiile din **assuming** vor fi folosite în efectuarea calculelor.

Pentru ecuaţiile polinomiale de grad cel mult trei este returnată o secvenţă a soluţiilor exacte, iar pentru cele de grad mai mare sau egal cu patru este utilizată reprezentarea cu **RootOff** a rădăcinilor, iar valorile lor sunt determinate cu ajutorul funcţiei predefinite **all-values**.

Indiferent de metoda utilizată pentru a determina soluţiile unei ecuaţii, Maple oferă posibilitatea de a le calcula cu un număr arbitrar de zecimale, ce se specifică prin variabila globală **Digits**.

Vom prezenta o serie de comenzi specializate în rezolvarea ecuaţiilor clasificate în funcţie de domeniul pe care se aplică.

fsolve(*ec, var, opt*) :: rezolvă ecuaţii în virgulă mobilă. Valorile *opt* disponibile acestei comenzi sunt : **complex**(implică aflarea unei soluţii complexe sau a tuturor în cazul ecuaţiilor algebrice), **fulldigits** (asigură menţinerea numărului de zecimale pe parcursul calculelor intermediare), **maxsols=n**(găseşte doar n rădăcini), **var=a..b** (caută doar rădăcinile care sunt localizate în intervalul $[a,b]$).

isolve(*ec, var*):: calculează rădăcinile întregi ale unei ecuaţii. Argumentul *var* se utilizează pentru a denumi parametri cu valori întregi, k̂in absenţa sa fiind utilizate simbolurile **N1, N2,...**.

msolve(*ec, var, m*): : rezolvă ecuaţii modulo m. Implicit, funcţia returnează doar cinci soluţii. Numărul acestora poate fi specificat prin asignarea variabilei globale **MaxSols**. Dacă soluţia este nedeterminată şi dacă este posibil se returnează o familie de soluţii exprimate în funcţie de numele variabilelor sau de simbolurile **N1, N2,...** .

rsolve(*ecr, f*):: rezolvă ecuaţii recurente. Argumentul *ecr* conţine o ecuaţie sau un sistem de ecuaţii recurente precum şi valorile iniţiale, iar f este o funcţie necunoscută din ecuaţia *ecr*.

dsolve(*ecd, var, ec*):: rezolvă ecuaţii şi sisteme de ecuaţii diferenţiale ordinare. Argumentul *ecd* este o ecuaţie diferenţială sau o mulţime de ecuaţii

diferenţiale, *var* reprezintă variabila, iar *ec* este un parametru opţional ce reprezintă o ecuaţie specială ce poate fi: **type = exact/ series/ numeric**, **explicit=true/false**, **method=rkf45/dverk78** (pentru metoda Runge-Kutta) sau **method=laplace** (dacă rezolvarea se face cu metoda transformării Laplace).

Exemple de utilizare a comenzilor de rezolvare de ecuaţii:

```
> restart:
> eq1 := x^5-x^3+3*x^2-2;
```

$$eq1 := x^5 - x^3 + 3x^2 - 2$$

```
> solve(eq1, x);
```

$$RootOf\left(\_Z^5 - \_Z^3 + 3\_Z^2 - 2\right)$$

```
> allvalues(RootOf(eq1));
```

$$RootOf\left(\_Z^5 - \_Z^3 + 3\_Z^2 - 2\right)$$

```
> Digits := 4;

> evalf(%%);
```

$$0.8505, 0.7379 + 1.187\,i, -0.7776, -1.549, 0.7379 - 1.187\,i$$

```
> fsolve(eq1, x);
```

$$-1.549, -0.7776, 0.8505$$

```
> fsolve(eq1, x, complex);
```

$$-1.549, -0.7776, 0.7379 - 1.187\,i, 0.7379 + 1.187\,i, 0.8505$$

```
> fsolve(eq1, x, -2 .. 1);
```

$$-1.549, -0.7776, 0.8505$$

```
> fsolve(eq1, x, maxsols = 3);
```

$$-1.549, -0.7776, 0.8505$$

```
> eq2 := 3*x+y-7*z;
```

$$eq2 := 3x + y - 7z$$

```
> isolve(eq2);
```

$$\{x = \_Z1, y = -3\_Z1 + 7\_Z2, z = \_Z2\}$$

```
> eq3 := y^2-5 = 0;
```

$$eq3 := y^2 - 5 = 0$$

```
> msolve(eq3, 10);
```

$$\{y = 5\}$$

```
> eqn := t(n)+t(n-1) = t(n+1);
```

$$eqn := t(n) + t(n-1) = t(n+1)$$

```
> rsolve({eqn, t(1 .. 2) = 1}, t);
```

$$-1/5\sqrt{5}\left(-1/2\sqrt{5} + 1/2\right)^n + 1/5\sqrt{5}\left(1/2\sqrt{5} + 1/2\right)^n$$

```
> simplify(%);
```

$$-1/5\sqrt{5}\left(\left(-1/2\sqrt{5}+1/2\right)^n-\left(1/2\sqrt{5}+1/2\right)^n\right)$$

```
> ode := diff(y(x), [$(x, 2)]) = 2*y(x)+1;
```

$$ode := \frac{d^2}{dx^2}y(x) = 2y(x)+1$$

```
> initcond := y(0) = 1, (D(y))(0) = 0;
```

$$initcond := y(0) = 1, \mathrm{D}\left(y\right)(0) = 0$$

```
> dsolve(ode);
```

$$y(x) = \mathrm{e}^{\sqrt{2}x}\_C2+\mathrm{e}^{-\sqrt{2}x}\_C1-1/2$$

```
> dsolve({ode, initcond});
```

$$y(x) = 3/4\,\mathrm{e}^{\sqrt{2}x}+3/4\,\mathrm{e}^{-\sqrt{2}x}-1/2$$

```
> dsolve({ode, initcond}, y(x), series);
```

$$y(x) = 1+3/2\,x^2+1/4\,x^4+O\left(x^6\right)$$

4. REZOLVAREA APROXIMATIVA A ECUATIILOR NELINIARE FOLOSIND METODE ITERATIVE

În această secţiune vom prezenta implementarea metodelor iterative în Maple. Totodată, dorim să studiem avantejele şi dezavantajele acestor metode prin intermediul unui exemplu concret.

Exemplul considerat presupune următoarele:

EXEMPLUL 2. Să se aproximeze soluţia reală a ecuaţiei $4(x-1)^3-12(x-1)^2-8x=0$ cu precizia $\varepsilon = 0.01$ folosind cele patru metode iterative prezentate.

Soluţie. Pentru a determina intervalul de pornire $[a,b]$ în care estimăm că se află soluţia aproximativă a ecuaţiei date considerăm funcţia $f(x) = 4(x-1)^3 - 12(x-1)^2 - 8x$ şi trasăm graficul acesteia cu ajutorul comenzii

$$\textbf{plot}(f, x_0..x_1);$$

unde x_0 şi x_1 reprezintă extremităţile intervalului considerat pe axa absciselor. Dacă al doilea argument lipseşte, atunci comanda trasează implicit graficul pe intervalul $[-10,10]$. Comanda **plot** trebuie să fie precedată de încărcarea pachetului **with(plots)**.

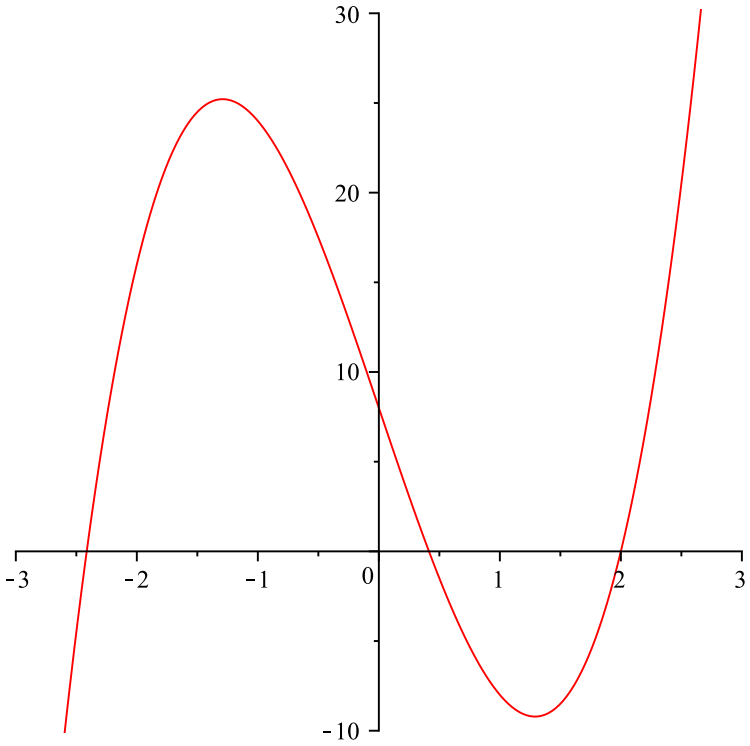


FIGURA. 4.1 – Graficul funcţiei $f(x) = 4(x-1)^3 - 12(x-1)^2 - 8x$

Pe baza graficului se poate observa că funcţia admite rădăcini în intervalele $[-3,-2]$, $[0,1]$ şi o rădăcină în punctul $x = 2$. Ne va interesa să aproximăm rădăcina din intervalul $[0,1]$.

Mai întâi trebuie să verificăm dacă funcţia noastră satiface condiţiile impuse în prima secţiune a lucrării(1). Putem să observăm cu ajutorul graficului că f este continuă pe $[0,1]$, are semne diferite la capetele intervalului, este strict descrescătoare şi convexă pe acest interval.

Vom prezenta pe rând cele patru proceduri corespunzătoare metodelor iterative amintite.

Procedura *înjumătăţire* presupune împărţirea succesivă în câte două subintervale a intervalului $[a,b]$ până când capetele intervalului vor fi suficient de apropiate de valorea aproximată a rădăcinii α, care va fi de fapt mijlocul intervalul, notat c. Parametrii procedurii sunt funcţia f, capetele intervalului $[a,b]$ în care se află rădăcina aproximativă α şi precizia maximă admisă. La fiecare iteraţie, procedura va afişa numărul iteraţiei, noua aproximaţie a intervalului în care se află rădăcina (adică un nou interval) şi mijlocul intervalului considerat.

```
> injumatatire := proc (f, a, b, eps)
  local c, a1, b1, n; a1 := a; b1 := b; n := 1;
```

```
    while n <= round(evalf(ln((b-a)/eps)/ln(2))) do
    c := (a1+b1)/2;
    print(cat("n=", n, ", c=", convert(c, string), ", (" , convert(a1, string), " , " ,con-
vert(b1,string), ") "));
    if evalf(f(c))*evalf(f(a1)) < 0 then
      b1 := c;
    else
      a1 := c;
    fi;
    n := n+1;
    od;
    c :=(a1+b1)/2;
    print(cat(" n=", n, ", c=", convert(c, string), ", (" , convert(a1, string), " , "
,convert(b1,string), ") "));
  end proc;
```

$$
\begin{gathered}
\text{“}n=1, c=1/2, (0,1)\text{”} \\
\text{“}n=2, c=1/4, (0,1/2)\text{”} \\
\text{“}n=3, c=3/8, (1/4,1/2)\text{”} \\
\text{“}n=4, c=7/16, (3/8,1/2)\text{”} \\
\text{“}n=5, c=13/32, (3/8,7/16)\text{”} \\
\text{“}n=6, c=27/64, (13/32,7/16)\text{”} \\
\text{“}n=7, c=53/128, (13/32,27/64)\text{”} \\
\text{“}n=8, c=107/256, (53/128,27/64)\text{”}
\end{gathered}
$$

Procedura care implementează metoda coardei poartă acelaşi nume *coarda*. Parametrii funcţiei sunt aceeaşi ca şi la metoda bisecţiei, având în plus un parametru care reprezintă minimul derivatei de ordin întâi al funcţiei f. Acest minim este determinat cu ajutorul unei alte proceduri, numită *minimum*. Procedura *coarda* afişează la fiecare iteraţie noul interval considerat, iar la încheierea ciclului va afişa rădăcina determinată.

```
> minimum := proc (f, a, b)
    local fd1, fd2;
    fd1 := x →diff(f(x), x);
    fd2 :=x → diff(fd1(x), x);
    if 0 <= evalf(eval(fd2(x), x = a)) then
      return evalf(eval(fd1(x), x = a));
    else
      return evalf(eval(fd2(x), x = b));
    fi;
  end proc;
```

```
> coarda := proc (f, x0, x1, eps, minim)
  local xn, fd1, fd2, xp, pf, final;
  fd1 := x → diff(f(x), x);
  fd2 :=x → diff(fd1(x), x);
  xp := x0;
  xn := x1;
  if 0 < evalf(f(xp))*evalf(eval(fd2(x), x = xp)) then
    pf := xp;
  else
  pf := xn;
  fi;
  xn := xn-(pf-xp)*evalf(f(pf))/(evalf(eval(f(x), x = pf))-evalf(eval(f(x), x = xp)));
  while eps <= abs(evalf(f(xn)))/abs(minim) do
    xp := xn;
    xn := xp-(pf-xp)*evalf(f(xp))/(evalf(eval(f(x), x = pf))-evalf(eval(f(x), x = xp)));
    print(cat('(',convert(min(xn, xp), string), ',', convert(max(xn, xp), string),')'))
  od;
  alpha := xp+abs(evalf(eval(f(x), x = xn)))/minim;
  print(cat("alpha=", convert(alpha, string)));
end proc;
coarda(f, 0, 1, 0.1e-1, minim);
```

$$\text{“(.3846153846,.5000000000)''}$$

$$\text{“(.3846153846,.4232081912)''}$$

$$\text{“}alpha = .3765669486\text{''}$$

Procedura corespunzătoare metodei lui Newton se numeşte *tangenta*. Signatura acestei proceduri este aceeaşi cu cea a procedurii *coarda*. Procedura tipăreşte la fiecare iteraţie intervalul considerat, iar la final aproximaţia rădăcinii α.

```
> tangenta := proc (f, x0, x1, eps, minim)
  local xn, xp, fd1, fd2, ptan, alpha;
  fd1 := x → diff(f(x), x);
  fd2 := x → diff(fd1(x), x);
  xp := x0;
  xn := x1;
  if 0 < evalf(f(xp))*evalf(eval(fd2(x), x = xp)) then
    ptan := xp;
  else
    ptan := xn ;
fi;
  xp := ptan;
  while eps <= abs(evalf(f(xn)))/abs(minim) do
    xn := xp-evalf(f(xp))/evalf(eval(fd1(x), x = xp));
  print(cat("(", convert(min(xn, xp), string), " , ", convert(max(xp, xn), string),
")"));
```

```
    xp := xn ;
    od;
    alpha := xn+abs(evalf(f(xn)))/minim;
    print(cat("alpha=", convert(alpha, string)));
  end proc;
  > tangenta(f, 0, 1, 0.1e-1, -20);
```

$$\text{“(0., 1)”}$$

$$\text{“(0., .4000000000)”}$$

$$\text{“(.4000000000, .4141592920)”}$$

$$\text{“}alpha = .4141106077\text{”}$$

Ultima procedură este cea aferentă metodei Newton-Fourier şi are o structura similară metodei tangentei.

```
  > NewtonFourier := proc (f, x0, x1, eps, minim)
    local xn, xp, fd1, fd2, ptan, ptanz, alpha, z0, z1, alpha1;
    fd1 :=x → diff(f(x), x);
    fd2 := x → diff(fd1(x), x);
    xp := x0;
    z0 := x0;
    xn := x1;
    z1 := x1;
    if 0 < evalf(f(xp))*evalf(eval(fd2(x), x = xp)) then
      ptan := xp;
      ptanz := xn
    else
      ptan := xn;
      ptanz := xp
    fi;
    xp := ptan;
    z0 := ptanz;
    while eps <= abs(evalf(f(z1)))/abs(minim) do
      xn := xp-evalf(f(xp))/evalf(eval(fd1(x), x = xp));
      z1 := z0-evalf(f(z0))/evalf(eval(fd1(x), x = xp));
      print(cat("(", convert(min(z0, z1), string), " , ", convert(max(z0, z1), string),
")"));
      xp := xn;
      z0 := z1;
    od;
    alpha := xn+abs(evalf(f(xn)))/minim;
    alpha1 := z1+abs(xn-z1);
    print(cat("alpha1=", convert(alpha1, string)));
  end proc;
  > NewtonFourier(f, 0, 1, 0.1e-1, -20);
```

$$“(0,1.000000000)''$$
$$“(.6000000000,1.000000000)''$$
$$“(.4265486726,.6000000000)''$$
$$“(.4142565047,.4265486726)''$$
$$“alpha1 = .4142994479''$$

Am prezentat cele patru proceduri în funcţie ordinul de convergenţă al metodelor. Dispunerea lor este în sensul creşterii ordinului de convergenţă. Din rezultatele obţinute, observăm că pe măsură ce ordinul creşte, metoda converge mai repede (numărul de iteraţii necesare scade), iar aproximările sunt tot mai bune, fapt pe care îl putem constata comparând cu rezultatul obţinut cu **fsolve**.

> fsolve(f(x) = 0);

$$-2.414213562, 0.4142135624, 2.0$$

BIBLIOGRAFIE

[1] MICULA, S., SOBOLU, R. and MICULA, M, *Analiză numerică cu Maple* , Editura AcademicPress **8** (2008).

[2] ATKINSON, K. E., *An introduction to numerical analysis*, John Wiley & Sons, **78** (1978).

Faculty of Mathematics and Computer Science
“Babeş-Bolyai” University
Str. Kogălniceanu, no. 1
400084 Cluj-Napoca, Romania
e-mail: manu_andra_monica@yahoo.com

Primit la redacţie: 25 Mai 2015