Matematici Asistate de Calculator

Dr. Călin-Adrian POPA

Cursul 1

19 Februarie 2019

Despre mine

- Facultatea de Automatică şi Calculatoare, Universitatea Politehnica Timişoara
  - Inginer, Calculatoare şi Tehnologia Informaţiei, 2009
  - Master, Master of Software Engineering, 2011
  - Doctor, Calculatoare şi Tehnologia Informaţiei, 2015
  - Asistent, 2015
  - Şef de lucrări (Lector), 2016
- Facultatea de Matematică şi Informatică, Universitatea de Vest din Timişoara
  - Licenţiat, Matematică, 2013
  - Master, Modelări Analitice şi Geometrice ale Sistemelor, 2015

- **Curs**
  - 14 cursuri în primele 10 săptămâni
  - 2 examene scrise
    - săptămâna 5
    - săptămâna 11
  - N.E. $= \frac{\text{min(7 probleme} \times \text{10 puncte} + \text{7 probleme} \times \text{10 puncte} + \text{14 puncte pe prezenţă,140)}}{14}$
  - $2/3$ din nota finală
- **Laborator**
  - 10 laboratoare
  - 2 teste
    - săptămâna 6
    - săptămâna 12
  - N.L. $= \frac{\text{4 probleme} \times \text{10 puncte} + \text{5 probleme} \times \text{10 puncte} + \text{10 puncte din oficiu}}{10}$
  - $1/3$ din nota finală
- **Site**
  - http://www.cs.upt.ro/$\sim$cpopa

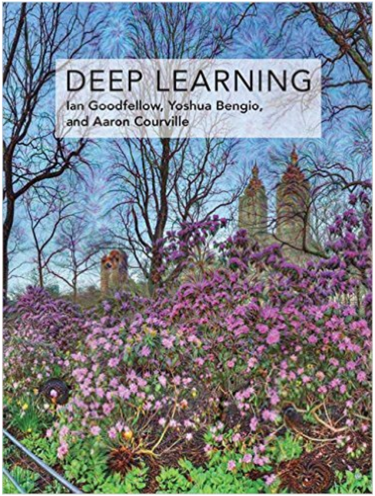
Cuprinsul cursului

Cuprinsul laboratorului

La ce ne foloseşte?

- fiecare capitol are aplicaţii în ştiinţa calculatoarelor
  - digital signal processing
  - image and video processing
  - computer graphics
  - computer vision
  - machine learning
  - data science
  - deep learning
- MATLAB – limbaj important în mai multe domenii
  - parallel computing
  - statistică, machine learning şi optimizare
  - signal processing
  - image processing şi computer vision

Deep learning



- Linear Algebra
- Probability and Information Theory
- Numerical Computation

1 Fundamente

- scopul acestui curs este de a prezenta şi discuta metode de rezolvare a unor probleme de matematică folosind calculatorul
- operaţiile aritmetice fundamentale sunt adunarea şi înmulţirea
- acestea sunt şi operaţiile necesare pentru a evalua un polinom $P(x)$ într-un punct x
- polinoamele stau la baza multor tehnici computaţionale pe care le vom construi în acest curs
- este important să ştim cum să evaluăm un polinom
- vom studia cum să implementăm evaluarea polinoamelor cât mai eficient posibil

1.1 Evaluarea unui polinom

- care este cea mai bună metodă pentru a evalua polinomul

$$P(x) = 2x^4 + 3x^3 - 3x^2 + 5x - 1,$$

  de exemplu, în punctul $x = 1/2$?
- coeficienţii polinomului şi numărul $1/2$ sunt stocaţi în memorie, şi dorim să minimizăm numărul de adunări şi înmulţiri necesare pentru a obţine $P(1/2)$
- **Metoda 1.** Prima şi cea mai directă metodă este:

$$P\left(\frac{1}{2}\right) = 2*\frac{1}{2}*\frac{1}{2}*\frac{1}{2}*\frac{1}{2}+3*\frac{1}{2}*\frac{1}{2}*\frac{1}{2}-3*\frac{1}{2}*\frac{1}{2}+5*\frac{1}{2}-1 = \frac{5}{4}. \quad (1)$$

- numărul de înmulţiri necesare este 10, la care se adaugă 4 adunări
- cu siguranţă există o metodă mai bună decât cea din (1)
- numărul de operaţii poate fi redus prin eliminarea înmulţirii repetate cu $1/2$
- o strategie mai bună este să calculăm mai întâi $(1/2)^4$, memorând şi rezultatele parţiale obţinute

- **Metoda 2.** Găsim mai întâi puterile numărului $x = 1/2$, pe care le stocăm pentru eventuale utilizări viitoare:

$$\begin{aligned} \frac{1}{2} * \frac{1}{2} &= \left(\frac{1}{2}\right)^2 \\ \left(\frac{1}{2}\right)^2 * \frac{1}{2} &= \left(\frac{1}{2}\right)^3 \\ \left(\frac{1}{2}\right)^3 * \frac{1}{2} &= \left(\frac{1}{2}\right)^4. \end{aligned}$$

- acum, putem face o simplă însumare a termenilor:

$$P\left(\frac{1}{2}\right) = 2 * \left(\frac{1}{2}\right)^4 + 3 * \left(\frac{1}{2}\right)^3 - 3 * \left(\frac{1}{2}\right)^2 + 5 * \frac{1}{2} - 1 = \frac{5}{4}.$$

- avem 3 înmulţiri ale lui $1/2$, plus alte 4 înmulţiri
- am redus numărul total de înmulţiri la 7, cu aceleaşi 4 adunări
- este reducerea de la 14 la 11 operaţii o îmbunătăţire semnificativă?
- dacă polinomul trebuie evaluat pentru diferite valori ale lui x, de mai multe ori pe secundă, atunci diferenţa poate fi esenţială

1.1 Evaluarea unui polinom

- **Metoda 3.** (Înmulţirea imbricată) Rescriem polinomul astfel încât să poată fi evaluat din interior spre exterior:

$$\begin{aligned} P(x) &= -1 + x(5 - 3x + 3x^2 + 2x^3) \\ &= -1 + x(5 + x(-3 + 3x + 2x^2)) \\ &= -1 + x(5 + x(-3 + x(3 + 2x))) \\ &= -1 + x * (5 + x * (-3 + x * (3 + x * 2))). \end{aligned} \tag{2}$$

- acum facem evaluarea dinspre interior înspre exterior:

$$\begin{aligned} &\text{înmulţim } \frac{1}{2} * 2, && \text{adunăm } +3 \to 4 \\ &\text{înmulţim } \frac{1}{2} * 4, && \text{adunăm } -3 \to -1 \\ &\text{înmulţim } \frac{1}{2} * -1, && \text{adunăm } +5 \to \frac{9}{2} \\ &\text{înmulţim } \frac{1}{2} * \frac{9}{2}, && \text{adunăm } -1 \to \frac{5}{4}. \end{aligned} \tag{3}$$

1.1 Evaluarea unui polinom

- această metodă, numită **înmulţire imbricată** sau **metoda lui Horner**, evaluează polinomul folosind 4 înmulţiri şi 4 adunări
- în general, un polinom de gradul d poate fi evaluat utilizând această metodă folosind d înmulţiri şi d adunări
- acest exemplu este caracteristic întregului domeniu al metodelor computaţionale implementate cu ajutorul calculatorului
- în primul rând, calculatoarele fac foarte rapid operaţii foarte simple
- în al doilea rând, este foarte important ca până şi aceste operaţii simple să fie făcute cât mai eficient, deoarece este posibil ca ele să fie executate de foarte multe ori
- în al treilea rând, cea mai bună metodă s-ar putea să nu fie şi cea mai evidentă metodă

1.1 Evaluarea unui polinom

- forma generală a unui polinom $c_1 + c_2 x + c_3 x^2 + c_4 x^3 + c_5 x^4$ poate fi scrisă în formă imbricată astfel:

$$c_1 + x(c_2 + x(c_3 + x(c_4 + x(c_5)))), \tag{4}$$

  însă anumite aplicaţii necesită o formă mai generală

- în particular, interpolarea va avea nevoie ca polinomul să fie scris sub forma

$$c_1 + (x - r_1)(c_2 + (x - r_2)(c_3 + (x - r_3)(c_4 + (x - r_4)(c_5)))), \tag{5}$$

  unde r_1, r_2, r_3, şi r_4 sunt numite **puncte de bază**

- luând $r_1 = r_2 = r_3 = r_4 = 0$ în (5), obţinem forma imbricată din (4)

1.1 Evaluarea unui polinom

Exemplul 1

- găsiţi o metodă eficientă pentru evaluarea polinomului $P(x) = 4x^5 + 7x^8 - 3x^{11} + 2x^{14}$
- ideea este de a factoriza x^5 din fiecare termen şi de a scrie polinomul în x^3:

$$\begin{aligned} P(x) &= x^5(4 + 7x^3 - 3x^6 + 2x^9) \\ &= x^5 * (4 + x^3 * (7 + x^3 * (-3 + x^3 * (2)))). \end{aligned}$$

- pentru fiecare punct x, trebuie să calculăm mai întâi $x * x = x^2$, $x * x^2 = x^3$, şi $x^2 * x^3 = x^5$
- aceste trei înmulţiri, împreună cu înmulţirea lui x^5, şi cele trei înmulţiri şi trei adunări necesare evaluării polinomului de gradul 3 în x^3 dau numărul total de operaţii necesar evaluării acestui polinom, şi anume 7 înmulţiri şi 3 adunări

1.2 Numere binare

- pentru a pregăti discuţia despre operaţiile aritmetice realizate în calculator, trebuie mai întâi să înţelegem sistemul de numeraţie binar
- numerele zecimale sunt convertite din baza 10 în baza 2 pentru a fi stocate în calculator şi a simplifica anumite operaţii ca adunarea sau înmulţirea
- pentru a furniza rezultatele ca numere zecimale, acest proces este inversat
- în această secţiune vom discuta diverse modalităţi de conversie între numere zecimale şi numere binare
- numerele binare sunt exprimate sub forma

$$\dots b_2 b_1 b_0 . b_{-1} b_{-2} \dots,$$

unde fiecare cifră binară, sau **bit**, este 0 sau 1
- echivalentul în baza 10 a acestui număr binar este

$$\dots b_2 2^2 + b_1 2^1 + b_0 2^0 + b_{-1} 2^{-1} + b_{-2} 2^{-2} \dots .$$

- de exemplu, numărul zecimal 4 se exprimă ca $(100)_2$ în baza 2
- $3/4$ este reprezentat ca $(0.11)_2$ în aceeaşi bază

1.2.1 Conversia din zecimal în binar

- numărul zecimal 53.7 va fi reprezentat ca $(53.7)_{10}$, pentru a sublinia faptul că va fi interpretat ca fiind în baza 10
- pentru a-l converti în binar, cel mai simplu este a fi împărţit în partea întreagă şi în partea fracţionară, şi a converti fiecare parte separat
- pentru numărul $(53.7)_{10} = (53)_{10} + (0.7)_{10}$, vom converti fiecare parte în binar, şi vom combina rezultatele
- **Partea întreagă.** Numerele zecimale se convertesc în binar prin împărţirea lor succesivă la 2 şi reţinerea resturilor acestor împărţiri
- resturile care pot fi 0 sau 1, sunt reţinute pornind de la virgula zecimală (sau, mai exact de la **rădăcină**) şi înaintând spre stânga
- pentru $(53)_{10}$, vom avea

$$\begin{aligned}
53 \div 2 &= 26 \text{ R } 1 \\
26 \div 2 &= 13 \text{ R } 0 \\
13 \div 2 &= 6 \text{ R } 1 \\
6 \div 2 &= 3 \text{ R } 0 \\
3 \div 2 &= 1 \text{ R } 1 \\
1 \div 2 &= 0 \text{ R } 1.
\end{aligned}$$

1.2.1 Conversia din zecimal în binar

- prin urmare, numărul 53 în baza 10 poate fi scris folosind biţi ca 110101, ceea ce se notează $(53)_{10} = (110101)_2$
- verificând rezultatul, avem că $110101 = 2^5 + 2^4 + 2^2 + 2^0 = 32 + 16 + 4 + 1 = 53$
- **Partea fracţionară.** Convertim $(0.7)_{10}$ în binar inversând paşii de mai sus
- înmulţim cu 2 succesiv şi reţinem părţile întregi, pornind de la virgula zecimală şi înaintând spre dreapta

$$
\begin{array}{ll}
0.7 \times 2 & 0.4 + 1 \\
0.4 \times 2 & 0.8 + 0 \\
0.8 \times 2 & 0.6 + 1 \\
0.6 \times 2 & 0.2 + 1 \\
0.2 \times 2 & 0.4 + 0 \\
0.4 \times 2 & 0.8 + 0 \\
& \vdots
\end{array}
$$

1.2.1 Conversia din zecimal în binar

- observăm că acest proces se repetă după patru paşi şi se va repeta exact la fel, la infinit
- prin urmare,

$$(0.7)_{10} = (0.1011001100110\ldots)_2 = (0.1\overline{0110})_2,$$

unde am folosit notaţia cu bară deasupra pentru a specifica o secvenţă de biţi care se repetă periodic

- punând împreună cele două părţi, concluzionăm că

$$(53.7)_{10} = (110101.1\overline{0110})_2.$$

1.2.2 Conversia din binar în zecimal

- pentru a converti un număr binar în zecimal, este din nou bine să-l despărţim în partea întreagă şi partea fracţionară
- **Partea întreagă.** Pur şi simplu adunăm puterile lui 2 înmulţite cu biţii corespunzători respectivelor puteri
- numărul binar $(10101)_2$ este $1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = (21)_{10}$
- **Parte fracţionară.** Dacă partea fracţionară este finită, procedăm de aceeaşi manieră
- de exemplu,

$$(0.1011)_2 = \frac{1}{2} + \frac{1}{8} + \frac{1}{16} = \left(\frac{11}{16}\right)_{10}.$$

- complicaţia apare atunci când partea fracţionară nu este finită, mai exact conţine o parte periodică
- cea mai simplă metodă de convertire a unei secvenţe binare periodice într-o fracţie zecimală este de a folosi proprietatea de deplasare a înmulţirii cu 2
- de exemplu, să presupunem că vrem să convertim numărul binar $x = (0.\overline{1011})_2$ în zecimal

1.2.2 Conversia din binar în zecimal

- înmulţim pe x cu 2^4, operaţie care realizează o deplasare cu 4 poziţii la stânga în binar
- apoi scădem valoarea iniţială a lui x:

$$\begin{aligned} 2^4 x &= 1011.\overline{1011} \\ x &= 0000.\overline{1011}. \end{aligned}$$

- obţinem astfel
$$(2^4 - 1)x = (1011)_2 = (11)_{10}.$$
- apoi îl scoatem pe x din această ecuaţie, şi obţinem $x = (0.\overline{1011})_2 = (11/15)_{10}$
- într-un alt exemplu, să presupunem că partea fracţionară nu este periodică imediat după virgula zecimală, ca în numărul binar $x = (0.10\overline{101})_2$
- înmulţirea cu 2^2 deplasează cu 2 poziţii la stânga, rezultând $y = 2^2 x = 10.\overline{101}$

1.2.2 Conversia din binar în zecimal

- partea fracţionară a lui y, notată $z = 0.\overline{101}$, este calculată ca mai sus:

$$\begin{aligned} 2^3 z &= 101.\overline{101} \\ z &= 000.\overline{101}. \end{aligned}$$

- prin urmare, $7z = 5$, şi $y = 2 + 5/7$, $x = 2^{-2}y = 19/28$ în baza 10
- numerele binare sunt elementele de bază ale operaţiilor aritmetice realizate în calculator, dar ele pot fi lungi şi greu de interpretat pentru factorul uman
- este folositor să folosim baza 16 pentru a reprezenta numerele binare mai uşor
- **numerele hexazecimale** sunt reprezentate de cele 16 numerale 0, 1, 2, . . . , 9, A, B, C, D, E, F
- fiecare număr hexazecimal poate fi reprezentat prin 4 biţi
- astfel, $(1)_{16} = (0001)_2$, $(8)_{16} = (1000)_2$, şi $(F)_{16} = (1111)_2 = (15)_{10}$

1.3 Reprezentarea în virgulă flotantă a numerelor reale

- în această secţiune, vom prezenta un model pentru operaţiile aritmetice realizate în calculator, folosind numere reprezentate în virgulă flotantă
- vom alege modelul standard de virgulă flotantă IEEE 754
- formatul pentru aritmetica în virgulă flotantă a devenit unul comun în industria calculatoarelor pentru operaţiile în simplă precizie şi dublă precizie
- erorile de rotunjire sunt comune atunci când locaţii de memorie cu precizie finită sunt folosite pentru a reprezenta numere reale care au precizie infinită
- deşi am spera ca erorile mici care apar într-un calcul lung să aibă doar un efect minor asupra rezultatului, acest lucru nu se întâmplă întotdeauna
- **algoritmi simpli pot amplifica erori microscopice la un nivel macroscopic**

1.3.1 Formate de virgulă flotantă

- standardul IEEE este format dintr-un set de reprezentări binare ale numerelor reale
- un **număr în virgulă flotantă** este format din trei părţi: **semnul** ($+$ sau $-$), **mantisa**, care conţine şirul de biţi semnificativi, şi **exponentul**
- cele trei părţi sunt stocate împreună într-un singur **cuvânt** de calculator
- există trei nivele de precizie utilizate în general pentru numerele în virgulă flotantă: precizia *simplă*, precizia *dublă* şi precizia *extinsă*, cunoscută şi ca precizia *dublă lungă*
- numărul de biţi alocaţi pentru fiecare număr în virgulă flotantă în cele trei formate este 32, 64, şi, respectiv, 80
- biţii sunt împărţiţi între cele trei părţi ale unui număr în virgulă flotantă astfel:

precizia	semnul	exponentul	mantisa
simplă	1	8	23
dublă	1	11	52
dublă lungă	1	15	64

1.3.1 Formate de virgulă flotantă

- toate cele trei tipuri de precizie funcţionează practic în acelaşi fel
- forma unui număr în virgulă flotantă **normalizat** IEEE este

$$\pm 1.bbb...b \times 2^p, \tag{6}$$

  unde fiecare dintre cei N de b este 0 sau 1, iar p un număr binar format din M biţi, reprezentând exponentul
- normalizarea înseamnă că, după cum se arată în (6), bitul cel mai din stânga (semnificativ) trebuie să fie 1
- când un număr binar este stocat ca un număr în virgulă flotantă normalizat, este aliniat la stânga, ceea ce înseamnă că bitul de 1 aflat cel mai la stânga este deplasat până pe prima poziţie dinaintea virgulei
- această deplasare este compensată printr-o schimbare a exponentului
- de exemplu, numărul zecimal 9, care este 1001 în binar, va fi stocat sub forma

$$+1.001 \times 2^3,$$

  deoarece o deplasare de 3 biţi, care este echivalentă cu o înmulţire cu 2^3, este necesară pentru a muta bitul de 1 aflat cel mai la stânga până la poziţia corectă

1.3.1 Formate de virgulă flotantă

- vom discuta doar cazul dublei precizii în cele ce urmează
- precizia simplă şi precizia dublă lungă sunt tratate în acelaşi mod, cu excepţia unor valori diferite ale lungimilor M şi N ale exponentului, respectiv mantisei
- în dubla precizie utilizată de majoritatea compilatoarelor de C şi de către MATLAB, avem $M = 11$ şi $N = 52$
- numărul 1 în dublă precizie este

$$+1.\boxed{0000000000000000000000000000000000000000000000000000}\times 2^0,$$

unde am pus într-un chenar cei 52 de biţi ai mantisei

- următorul număr în virgulă flotantă mai mare decât 1 este

$$+1.\boxed{0000000000000000000000000000000000000000000000000001}\times 2^0,$$

sau $1 + 2^{-52}$

1.3.1 Formate de virgulă flotantă

Definiţia 1

Numărul **epsilon maşină**, notat cu ϵ_{mach}, este distanţa dintre 1 şi cel mai mic număr în virgulă flotantă mai mare decât 1. Pentru standardul IEEE de virgulă flotantă în dublă precizie, avem că

$$\epsilon_{\text{mach}} = 2^{-52}.$$

- numărul zecimal $9.4 = (1001.\overline{0110})_2$ este aliniat la stânga astfel:

  $$+1.\boxed{0010110011001100110011001100110011001100110011001100}110\ldots \times 2^3,$$

  unde am pus într-un chenar primii 52 de biţi ai mantisei
- apare acum o nouă întrebare: cum poate fi stocat numărul binar infinit care reprezintă numărul zecimal 9.4 folosind un număr finit de biţi?
- acest număr trebuie trunchiat într-un anumit mod, şi făcând această operaţie va rezulta o mică eroare
- o metodă, numită **trunchiere**, este de a ignora biţii care se află dincolo de sfârşitul reprezentării, adică acei biţi care se află după cel de-al 52-lea bit de la dreapta virgulei binare

1.3.1 Formate de virgulă flotantă

- acest protocol este simplu, însă este părtinitor, în sensul în care întotdeauna deplasează rezultatul către zero
- metoda alternativă este **rotunjirea**
- în baza 10, numerele sunt de obicei rotunjite în sus dacă următoarea cifră este 5 sau mai mare, şi rotunjite în jos în caz contrar
- în binar, aceasta corespunde cu rotunjirea în sus dacă bitul este 1 şi în jos dacă bitul este 0
- bitul important în dublă precizie este al 53-lea bit de la dreapta virgulei binare, adică primul care se află în afara chenarului, în reprezentarea de mai sus
- tehnica implicită de rotunjire implementată de standardul IEEE este de a aduna un 1 la bitul 52 (rotunjire în sus) dacă bitul 53 este 1, şi de a nu face nimic (rotunjire în jos) bitului 52, dacă bitul 53 este 0
- cu o excepţie: dacă biţii care urmează bitului 52 sunt $10000\ldots$, adică exact la jumătate între rotunjirea în sus şi rotunjirea în jos, se va face o rotunjire în sus sau în jos, în funcţie de care dintre cele două alegeri va face bitul final 52 să fie egal cu 0
- de ce există această situaţie de excepţie care poate părea ciudată?

- cu excepţia acestui caz, regula se traduce prin rotunjirea la cel mai apropiat număr în virgulă flotantă normalizat faţă de numărul iniţial, de unde şi numele de regula rotunjirii la cea mai apropiată valoare
- eroarea de rotunjire făcută va fi la fel de probabil să fie în sus sau în jos
- prin urmare, în cazul de excepţie, în care există două numere în virgulă flotantă la care se poate face rotunjirea, care se află la distanţe egale faţă de numărul iniţial, decizia va fi luată astfel încât să nu fie preferată rotunjirea în jos sau în sus, în mod sistematic
- această măsură este menită să descurajeze o alunecare uşoară în calcule lungi, datorată unei rotunjiri părtinitoare
- alegerea de a face bitul final 52 egal cu 0 în cazul de egalitate menţionat mai sus este într-un fel arbitrară, dar are avantajul de a nu prefera nici rotunjirea în sus, nici rotunjirea în jos

1.3.1 Formate de virgulă flotantă

Algoritmul 1 (Regula IEEE a rotunjirii la cea mai apropiată valoare)

Pentru dubla precizie, dacă al 53-lea bit de la dreapta virgulei binare este 0, atunci rotunjim în jos (trunchiem după bitul 52). Dacă bitul al 53-lea este 1, atunci rotunjim în sus (adunăm 1 la bitul 52), cu excepţia cazului în care biţii de după 1 sunt 0, caz în care 1 este adunat la bitul 52 dacă şi numai dacă bitul 52 este 1.

- pentru numărul 9.4 discutat anterior, cel de-al 53-lea bit din dreapta virgulei binare este 1 şi este urmat de alţi biţi care nu sunt toţi zero
- conform regulii rotunjirii la cea mai apropiată valoare, trebuie să facem o rotunjire în sus, adică să adunăm un 1 la bitul 52
- prin urmare, numărul în virgulă flotantă care îl reprezintă pe 9.4 este

$$+1.\boxed{0010110011001100110011001100110011001100110011001101}\times 2^3. \tag{7}$$

1.3.1 Formate de virgulă flotantă

Definiţia 2

Notăm numărul în virgulă flotantă în dublă precizie IEEE asociat lui x, folosind regula rotunjirii la cea mai apropiată valoare, cu **fl(x)**.

- în operaţiile aritmetice realizate în calculator, numărul real x este înlocuit cu şirul de biţi $\mathrm{fl}(x)$
- conform definiţiei de mai sus, fl(9.4) este numărul în reprezentare binară dat de (7)
- am ajuns la această reprezentare în virgulă flotantă înlăturând partea infinită dată de $0.\overline{1100} \times 2^{-52} \times 2^3 = 0.\overline{0110} \times 2^{-51} \times 2^3 = 0.4 \times 2^{-48}$ din capătul din dreapta al numărului, şi apoi adunând $2^{-52} \times 2^3 = 2^{-49}$ în pasul de rotunjire
- prin urmare,

$$\begin{aligned} \mathrm{fl}(9.4) &= 9.4 + 2^{-49} - 0.4 \times 2^{-48} \\ &= 9.4 + (1 - 0.8)2^{-49} \\ &= 9.4 + 0.2 \times 2^{-49}. \end{aligned} \tag{8}$$

1.3.1 Formate de virgulă flotantă

- cu alte cuvinte, un calculator care foloseşte reprezentarea în dublă precizie şi regula rotunjirii la cea mai apropiată valoare face o eroare de 0.2×2^{-49} atunci când stochează numărul 9.4
- vom spune că 0.2×2^{-49} este **eroarea de rotunjire**
- ceea ce trebuie reţinut este faptul că numărul în virgulă flotantă care îl reprezintă pe 9.4 nu este egal cu 9.4, deşi este foarte aproape de această valoare
- pentru a cuantifica această apropiere, vom folosi definiţia standard a erorii

Definiţia 3

Fie x_c o versiune calculată a valorii exacte x. Atunci

$$\textbf{eroarea absolută} = |x_c - x|,$$

şi

$$\textbf{eroarea relativă} = \frac{|x_c - x|}{|x|},$$

dacă această din urmă cantitate există ($x \neq 0$).

Algoritmul 2 (Eroarea relativă de rotunjire)

În cadrul standardului IEEE, eroarea relativă de rotunjire $\mathrm{fl}(x)$ este mai mică decât jumătate din numărul epsilon maşină:

$$\frac{|\mathrm{fl}(x) - x|}{|x|} \leq \frac{1}{2}\epsilon_{\mathrm{mach}}. \tag{9}$$

- în cazul numărului $x = 9.4$, am găsit eroarea de rotunjire în (8), care trebuie să satisfacă (9):

$$\frac{|\mathrm{fl}(9.4) - 9.4|}{9.4} = \frac{0.2 \times 2^{-49}}{9.4} = \frac{8}{47} \times 2^{-52} < \frac{1}{2}\epsilon_{\mathrm{mach}}.$$

1.3.1 Formate de virgulă flotantă

Exemplul 2

- găsiţi reprezentarea în dublă precizie $\mathrm{fl}(x)$ şi eroarea de rotunjire pentru $x = 0.4$
- deoarece $(0.4)_{10} = (0.\overline{0110})_2$, alinierea la stânga a acestui număr binar ne dă:

$$\begin{aligned} 0.4 &= 1.10\overline{0110} \times 2^{-2} \\ &= +1.\boxed{1001100110011001100110011001100110011001100110011001} \\ &\quad 100110\ldots \times 2^{-2}. \end{aligned}$$

- prin urmare, în conformitate cu regula de rotunjire, $\mathrm{fl}(0.4)$ este

$$+1.\boxed{1001100110011001100110011001100110011001100110011010} \times 2^{-2}.$$

- aici, 1 a fost adunat la bitul 52, ceea ce a determinat o schimbare şi a bitului 51, ca urmare a transportului din adunarea binară
- analizând cu atenţie, am înlăturat $2^{-53} \times 2^{-2} + 0.\overline{0110} \times 2^{-54} \times 2^{-2}$ în cadrul trunchierii şi am adunat $2^{-52} \times 2^{-2}$ prin rotunjirea în sus

1.3.1 Formate de virgulă flotantă

- prin urmare,

$$\begin{aligned}
\mathrm{fl}(0.4) &= 0.4 - 2^{-55} - 0.4 \times 2^{-56} + 2^{-54} \\
&= 0.4 + 2^{-54}(-1/2 - 0.1 + 1) \\
&= 0.4 + 2^{-54}(0.4) \\
&= 0.4 + 0.1 \times 2^{-52}.
\end{aligned}$$

- observăm că eroarea relativă de rotunjire pentru 0.4 este $0.1/0.4 \times \epsilon_{\text{mach}} = 1/4 \times \epsilon_{\text{mach}}$, conform cu (9)

1.3.2 Reprezentarea numerelor în calculator

- până acum am descris reprezentarea în virgulă flotantă în abstract
- vom prezenta în continuare mai multe detalii despre cum este această reprezentare implementată într-un calculator
- vom discuta formatul în dublă precizie în această subsecţiune, celelalte formate fiind asemănătoare
- fiecărui număr în virgulă flotantă în dublă precizie îi este asignat un cuvânt de 8 octeţi (1 octet = 8 biţi), sau 64 de biţi, pentru a stoca cele trei părţi ale sale
- fiecare astfel de cuvânt are forma

$$\boxed{se_1 e_2 \ldots e_{11} b_1 b_2 \ldots b_{52}}, \tag{10}$$

  unde mai întâi este stocat semnul, urmat de 11 biţi reprezentând exponentul şi de 52 de biţi care urmează virgulei zecimale, care reprezintă mantisa
- bitul de semn s este 0 pentru un număr pozitiv şi 1 pentru un număr negativ
- cei 11 biţi reprezentând exponentul vin din numărul binar pozitiv care rezultă din adunarea lui $2^{10} - 1 = 1023$ la exponent, cel puţin pentru exponenţi între -1022 şi 1023

1.3.2 Reprezentarea numerelor în calculator

- aceasta acoperă valorile lui $e_1 \dots e_{11}$ de la 1 la 2046, lăsând valorile 0 şi 2047 pentru scopuri speciale, care vor fi detaliate ulterior
- numărul 1023 se numeşte **biasul exponent** al formatului în dublă precizie
- este folosit pentru a converti atât exponenţii pozitivi cât şi pe cei negativi în numere binare pozitive pentru a fi stocate în biţii de exponent
- pentru precizia simplă şi dublă lungă, valorile biasului exponent sunt 127 şi respectiv 16383
- formatul hexazecimal constă din exprimarea celor 64 de biţi ai reprezentării în virgulă flotantă (10) ca 16 numere hexazecimale
- prin urmare, primele 3 numerale hexazecimale reprezintă semnul şi exponentul combinate, în timp ce ultimele 13 conţin mantisa

1.3.2 Reprezentarea numerelor în calculator

- de exemplu, numărul 1, sau

  $$1 = +1.\boxed{0000000000000000000000000000000000000000000000000000} \times 2^0,$$

  are reprezentarea în dublă precizie în virgulă flotantă

  $$\boxed{0}\boxed{01111111111}\boxed{0000000000000000000000000000000000000000000000000000},$$

  de îndată ce biasul exponent 1023 este adunat la exponent
- primele trei cifre hexazecimale corespund lui

  $$001111111111 = 3FF,$$

  şi deci reprezentarea în format hexazecimal a numărului în virgulă flotantă 1 va fi $3FF0000000000000$

Exemplul 3

- găsiţi reprezentarea în virgulă flotantă în dublă precizie a numărului real 9.4
- din (7), avem că semnul este $s = 0$, exponentul este 3, şi cei 52 de biţi ai mantisei de după virgula zecimală sunt

$$\boxed{0010}\boxed{1100}\boxed{1100}\boxed{1100}\boxed{1100}\boxed{1100}\boxed{1100}\boxed{1100}\boxed{1100}\boxed{1100}\boxed{1100}\boxed{1100}\boxed{1101} \to (2CCCCCCCCCCCD)_{16}.$$

- adunând 1023 la exponent obţinem $1026 = 2^{10} + 2$, sau $(10000000010)_2$
- combinaţia semn – exponent este $(010000000010)_2 = (402)_{16}$, dând formatul hexazecimal $4022CCCCCCCCCCCD$

1.3.2 Reprezentarea numerelor în calculator

- ne întoarcem acum la valorile speciale ale exponentului 0 şi 2047
- ultima, 2047, este folosită pentru a reprezenta ∞ dacă şirul de biţi ai mantisei este format doar din zerouri şi `NaN`, adică Not a Number, altfel
- deoarece 2047 este reprezentat prin unsprezece biţi de 1, sau $e_1e_2\ldots e_{11} = (111\,1111\,1111)_2$, primii doisprezece biţi ai lui `Inf` şi `-Inf` sunt $\boxed{0111}\boxed{1111}\boxed{1111}$ şi, respectiv, $\boxed{1111}\boxed{1111}\boxed{1111}$, şi cei 52 de biţi rămaşi (mantisa) sunt zero
- numărul `NaN` începe de asemenea cu $\boxed{1111}\boxed{1111}\boxed{1111}$ dar are o mantisă nenulă
- putem rezuma cele de mai sus în tabelul următor:

numărul	exemplu	format hexazecimal
`+Inf`	$1/0$	$7FF0000000000000$
`-Inf`	$-1/0$	$FFF0000000000000$
`NaN`	$0/0$	$FFFxxxxxxxxxxxxx$

unde x-urile denotă biţi care nu sunt toţi egali cu zero

- exponentul special 0, adică $e_1 e_2 \ldots e_{11} = (000\,0000\,0000)_2$, denotă de asemenea o abatere de la forma standard a unui număr în virgulă flotantă
- în acest caz, numărul este interpretat ca numărul în virgulă flotantă nenormalizat
$$\pm 0.\boxed{b_1 b_2 \ldots b_{52}} \times 2^{-1022}. \tag{11}$$
- adică, *doar în acest caz, bitul cel mai din stânga nu este presupus a fi egal cu* 1
- aceste numere nenormalizate se numesc numere în virgulă flotantă **subnormale**
- ele extind gama numerelor foarte mici cu încă câteva ordine de mărime
- prin urmare, $2^{-52} \times 2^{-1022} = 2^{-1074}$ este cel mai mic număr diferit de zero care este reprezentabil în dublă precizie
- cuvântul corespunzător care va fi stocat în calculator este

$$\boxed{0}\boxed{00000000000}\boxed{0000000000000000000000000000000000000000000000000001}.$$

1.3.2 Reprezentarea numerelor în calculator

- trebuie făcută distincţia între cel mai mic număr reprezentabil 2^{-1074} şi $\epsilon_{\text{mach}} = 2^{-52}$
- sunt multe numere mai mici decât ϵ_{mach} care sunt reprezentabile, chiar dacă adunându-le la 1 nu vom obţine niciun efect
- pe de altă parte, numerele în dublă precizie mai mici decât 2^{-1074} nu pot fi reprezentate deloc
- numerele subnormale includ şi cel mai important număr, 0
- de fapt, reprezentarea subnormală include două numere în virgulă flotantă diferite, $+0$ şi -0, care sunt tratate în calcule ca fiind acelaşi număr real
- reprezentarea în virgulă flotantă a lui $+0$ are bitul de semn $s = 0$, biţii de exponent $e_1 \ldots e_{11} = 00000000000$, şi mantisa 52 de zerouri; pe scurt, toţi cei 64 de biţi sunt zero
- formatul hexazecimal pentru $+0$ este 0000000000000000
- pentru numărul -0, totul este exact la fel, cu excepţia bitului de semn $s = 1$
- formatul hexazecimal pentru -0 este 8000000000000000

1.3.3 Adunarea numerelor în virgulă flotantă

- adunarea în calculator constă din alinierea virgulelor zecimale ale celor două numere de adunat, adunarea lor, şi stocarea rezultatului tot ca un număr în virgulă flotantă
- adunarea propriu-zisă poate fi realizată cu precizie mai mare (cu mai mult de 52 de biţi) deoarece are loc într-un registru dedicat special pentru acest scop
- după adunare, rezultatul trebuie rotunjit înapoi la 52 de biţi după virgula binară pentru a fi stocat ca un număr în virgulă flotantă
- de exemplu, adunarea lui 1 la 2^{-53} va apărea după cum urmează:

$$
\begin{aligned}
& 1.\boxed{00\ldots0} \times 2^0 + 1.\boxed{00\ldots0} \times 2^{-53} \\
= \quad & 1.\boxed{0000000000000000000000000000000000000000000000000000} \times 2^0 \\
+ \quad & 0.\boxed{0000000000000000000000000000000000000000000000000000}1 \times 2^0 \\
= \quad & 1.\boxed{0000000000000000000000000000000000000000000000000000}1 \times 2^0.
\end{aligned}
$$

- acest număr este stocat ca $1.0 \times 2^0 = 1$, conform regulii de rotunjire
- prin urmare, $1 + 2^{-53}$ este egal cu 1 în aritmetica în dublă precizie IEEE

1.3.3 Adunarea numerelor în virgulă flotantă

- observăm că 2^{-53} este cel mai mare număr în virgulă flotantă cu această proprietate; orice alt număr adunat cu 1 va rezulta într-o sumă mai mare decât 1 în urma adunării din calculator
- faptul că $\epsilon_{\text{mach}} = 2^{-52}$ nu înseamnă că numerele mai mici decât ϵ_{mach} sunt neglijabile în modelul IEEE
- atâta timp cât acestea sunt reprezentabile în model, calculele cu numere de această mărime sunt la fel de exacte, presupunând că acestea nu sunt adunate la sau scăzute din 1
- este important să se înţeleagă că aritmetica din calculator, din cauza trunchierii şi a rotunjirii pe care le efectuează, poate da uneori rezultate surprinzătoare

1.3.3 Adunarea numerelor în virgulă flotantă

- de exemplu, dacă unui calculator care foloseşte regula IEEE a rotunjirii la cea mai apropiată valoare îi este dată instrucţiunea de a stoca numărul 9.4, apoi de a scădea 9 din acesta, şi apoi de a scădea 0.4 din rezultat, ceea ce se obţine va fi diferit de zero!
- ceea ce se întâmplă este că: mai întâi 9.4 este stocat ca $9.4 + 0.2 \times 2^{-49}$, după cum s-a arătat anterior
- când 9 este scăzut din această valoare (observăm că 9 poate fi reprezentat fără eroare), rezultatul este $0.4 + 0.2 \times 2^{-49}$
- acum, dând calculatorului instrucţiunea de a scădea 0.4 din acest număr, rezultă în scăderea numărului $\mathrm{fl}(0.4) = 0.4 + 0.1 \times 2^{-52}$ (după cum am arătat în Exemplul 2), ceea ce va da

$$0.2 \times 2^{-49} - 0.1 \times 2^{-52} = 0.1 \times 2^{-52}(2^4 - 1) = 3 \times 2^{-53},$$

  în loc de zero
- acesta este un număr mic, de ordinul lui ϵ_{mach}, dar nu este zero

1.3.3 Adunarea numerelor în virgulă flotantă

Exemplul 4

- găsiţi suma în virgulă flotantă în dublă precizie $(1 + 3 \times 2^{-53}) - 1$
- bineînţeles, în aritmetica reală, răspunsul este 3×2^{-53}
- însă calculele în aritmetica în virgulă flotantă pot da un rezultat diferit
- observăm că $3 \times 2^{-53} = 2^{-52} + 2^{-53}$
- prima adunare este

$$
\begin{aligned}
& 1.\boxed{00\ldots0} \times 2^0 + 1.\boxed{10\ldots0} \times 2^{-52} \\
= \quad & 1.\boxed{0000000000000000000000000000000000000000000000000000} \times 2^0 \\
+ \quad & 0.\boxed{0000000000000000000000000000000000000000000000000001}1 \times 2^0 \\
= \quad & 1.\boxed{0000000000000000000000000000000000000000000000000001}1 \times 2^0.
\end{aligned}
$$

1.3.3 Adunarea numerelor în virgulă flotantă

- acesta este cazul de excepţie pentru regula rotunjirii
- deoarece bitul 52 din sumă este 1, trebuie să facem o rotunjire în sus, ceea ce înseamnă să adunăm 1 la bitul 52
- după transport, obţinem

$$1.\boxed{0000000000000000000000000000000000000000000000000010} \times 2^0,$$

  care este reprezentarea lui $1 + 2^{-51}$
- prin urmare, după ce scădem 1, rezultatul va fi 2^{-51}, care este egal cu $2\epsilon_{\text{mach}} = 4 \times 2^{-53}$
- din nou, se poate observa diferenţa dintre aritmetica în calculator şi aritmetica exactă

2 Rezolvarea ecuaţiilor

- rezolvarea ecuaţiilor este una dintre problemele cele mai de bază în analiza numerică
- acest capitol introduce mai multe metode iterative pentru localizarea soluţiilor x ale ecuaţiei $f(x) = 0$, ele având o mare importanţă practică
- se poate pune întrebarea: de ce este necesar să cunoaştem mai mult de o metodă pentru rezolvarea ecuaţiilor?
- adesea, alegerea metodei va depinde de costul computaţional al evaluării funcţiei f şi probabil şi al derivatei ei
- dacă $f(x) = e^x - \sin x$, va dura mai puţin decât o milionime de secundă pentru a determina $f(x)$, şi derivata ei este disponibilă, dacă avem nevoie de ea
- dacă $f(x)$ reprezintă temperatura de îngheţ a unei soluţii de etilen glicol la o presiune de x atmosfere, fiecare evaluare a funcţiei va necesita un timp considerabil într-un laborator bine echipat, iar determinarea derivatei s-ar putea să nu fie fezabilă

2.1 Metoda bisecţiei

- să ne gândim cum căutăm un nume într-o carte de telefon
- pentru a căuta numele “Ion”, începem prin a deschide cartea la cea mai bună aproximaţie, să zicem, litera G
- apoi s-ar putea să întoarcem un set de pagini şi să ajungem la litera K
- acum putem spune că am găsit limitele între care se află numele Ion, şi va trebui să-l căutăm între limite din ce în ce mai mici, care până la urmă vor converge la acest nume
- metoda bisecţiei reprezintă acest tip de raţionament, realizat cât mai eficient posibil

2.1.1 Găsirea limitelor între care se află o rădăcină

Definiţia 4

Funcţia $f(x)$ are o **rădăcină** în $x = r$ dacă $f(r) = 0$.

- primul pas în rezolvarea unei ecuaţii este de a verifica dacă există o rădăcină
- un mod de a ne asigura de aceasta este de a găsi limitele între care se află o rădăcină, şi anume de a găsi un interval $[a, b]$ pe dreapta reală pentru care un număr din perechea $\{f(a), f(b)\}$ este pozitiv şi celălalt este negativ
- acest fapt poate fi exprimat ca $f(a)f(b) < 0$
- dacă f este o funcţie continuă, atunci va exista o rădăcină, şi anume un r între a şi b pentru care $f(r) = 0$
- acest fapt este rezumat în următorul corolar al teoremei valorii intermediare:

2.1.1 Găsirea limitelor între care se află o rădăcină

Teorema 1 (Teorema valorii intermediare)

Fie f o funcţie continuă pe intervalul $[a, b]$. Atunci f ia orice valoare între $f(a)$ şi $f(b)$. Mai precis, dacă y este un număr între $f(a)$ şi $f(b)$, atunci există un număr c care satisface $a \leq c \leq b$ astfel încât $f(c) = y$.

Teorema 2

Fie f o funcţie continuă pe $[a, b]$, care satisface $f(a)f(b) < 0$. Atunci f are o rădăcină între a şi b, adică există un număr r care satisface $a < r < b$ şi $f(r) = 0$.

2.1.1 Găsirea limitelor între care se află o rădăcină

- în Figura 1, $f(0)f(1) = (-1)(1) < 0$
- există o rădăcină imediat la stânga lui 0.7
- cum putem rafina presupunerea iniţială despre locaţia rădăcinii cu mai multe zecimale exacte?

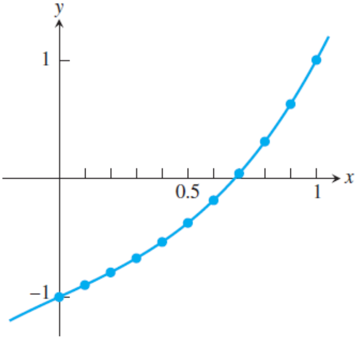


Figura 1: **Graficul funcţiei $f(x) = x^3 + x - 1$.** Funcţia are o rădăcină între 0.6 şi 0.7.

2.1.1 Găsirea limitelor între care se află o rădăcină

- ne vom inspira din felul în care ochiul nostru găseşte o soluţie atunci când i se dă graficul unei funcţii
- este puţin probabil că vom începe căutarea în capătul din stânga al intervalului şi vom înainta spre dreapta, oprindu-ne când găsim rădăcina
- probabil un model mai bun pentru ceea ce se întâmplă este că ochiul decide mai întâi asupra unei locaţii generale, cum ar fi de exemplu că rădăcina este în partea stângă sau în partea dreaptă a intervalului
- atunci continuă prin a decide mai exact cât de departe la dreapta sau la stânga se află rădăcina şi îmbunătăţeşte treptat precizia, la fel ca în căutarea unui nume în agenda telefonică
- această abordare generală devine destul de specifică în cadrul metodei bisecţiei, prezentată în Figura 2

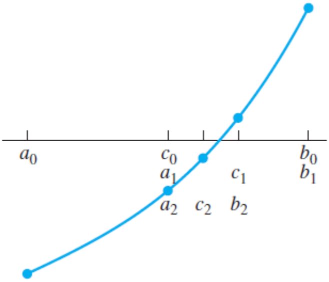


Figura 2: **Metoda bisecţiei.** În primul pas, semnul lui $f(c_0)$ este verificat. Deoarece $f(c_0)f(b_0) < 0$, asignăm $a_1 = c_0$, $b_1 = b_0$, şi intervalul este înlocuit cu jumătatea lui dreaptă $[a_1, b_1]$. În al doilea pas, subintervalul este înlocuit cu jumătatea lui stângă $[a_2, b_2]$.

Algoritmul 3 (Metoda bisecţiei)

Dându-se un interval iniţial $[a, b]$ astfel încât $f(a)f(b) < 0$

while $(b - a)/2 > \text{TOL}$

  $c = (a + b)/2$

  if $f(c) = 0$, **stop**, **end**

  if $f(a)f(c) < 0$

    $b = c$

  else

    $a = c$

  end

end

Intervalul final $[a, b]$ conţine o rădăcină.

Aproximarea rădăcinii este $(a + b)/2$.

2.1.1 Găsirea limitelor între care se află o rădăcină

- verificăm valoarea funcţiei la mijlocul $c = (a + b)/2$ al intervalului
- deoarece $f(a)$ şi $f(b)$ au semne opuse, ori $f(c) = 0$ (în care caz am găsit o rădăcină şi suntem gata), ori semnul lui $f(c)$ este opus semnului lui $f(a)$ sau $f(b)$
- dacă $f(c)f(a) < 0$, de exemplu, suntem asiguraţi că o soluţie se află în intervalul $[a, c]$, a cărui lungime este jumătate din cea a intervalului iniţial $[a, b]$
- dacă, însă, $f(c)f(b) < 0$, atunci putem spune acelaşi lucru despre intervalul $[c, b]$
- în ambele cazuri, un pas reduce problema la găsirea unei rădăcini pe un interval care are lungimea egală cu jumătate din lungimea intervalului iniţial
- acest pas poate fi repetat pentru a localiza rădăcina din ce în ce mai exact
- la fiecare pas, limitele între care se află rădăcina sunt altele, reducând incertitudinea despre localizarea soluţiei, pe măsură ce intervalul devine din ce în ce mai mic

Exemplul 5

- găsiţi o rădăcină a funcţiei $f(x) = x^3 + x - 1$ utilizând metoda bisecţiei pe intervalul $[0, 1]$
- observăm că $f(a_0)f(b_0) = (-1)(1) < 0$, prin urmare o rădăcină există în acest interval
- mijlocul intervalului este $c_0 = 1/2$
- primul pas constă în evaluarea lui $f(1/2) = -3/8 < 0$ şi alegerea noului interval ca fiind $[a_1, b_1] = [1/2, 1]$, deoarece $f(1/2)f(1) < 0$
- al doilea pas constă în evaluarea lui $f(c_1) = f(3/4) = 11/64 > 0$, conducând la noul interval $[a_2, b_2] = [1/2, 3/4]$

- continuând în această manieră, obţinem următoarele intervale:

i	a_i	$f(a_i)$	c_i	$f(c_i)$	b_i	$f(b_i)$
0	0.0000	$-$	0.5000	$-$	1.0000	$+$
1	0.5000	$-$	0.7500	$+$	1.0000	$+$
2	0.5000	$-$	0.6250	$-$	0.7500	$+$
3	0.6250	$-$	0.6875	$+$	0.7500	$+$
4	0.6250	$-$	0.6562	$-$	0.6875	$+$
5	0.6562	$-$	0.6719	$-$	0.6875	$+$
6	0.6719	$-$	0.6797	$-$	0.6875	$+$
7	0.6797	$-$	0.6836	$+$	0.6875	$+$
8	0.6797	$-$	0.6816	$-$	0.6836	$+$
9	0.6816	$-$	0.6826	$+$	0.6836	$+$

2.1.1 Găsirea limitelor între care se află o rădăcină

- concluzionăm din tabel că soluţia se află între $a_9 \approx 0.6816$ şi $c_9 \approx 0.6826$
- mijlocul acestui interval este $c_{10} \approx 0.6821$, care reprezintă aproximarea rădăcinii
- deşi problema a fost să găsim o rădăcină, am găsit de fapt un interval $[0.6816, 0.6826]$ care conţine o rădăcină; cu alte cuvinte, rădăcina este $r = 0.6821 \pm 0.0005$
- va trebui să fim mulţumiţi cu o aproximare
- bineînţeles, această aproximare poate fi îmbunătăţită, dacă este necesar, prin efectuarea mai multor paşi din metoda bisecţiei

2.1.1 Găsirea limitelor între care se află o rădăcină

- la fiecare pas al metodei bisecţiei, calculăm mijlocul $c_i = (a_i + b_i)/2$ al intervalului curent $[a_i, b_i]$, calculăm $f(c_i)$, şi comparăm semnele
- dacă $f(c_i)f(a_i) < 0$, asignăm $a_{i+1} = a_i$ şi $b_{i+1} = c_i$
- dacă, din contră, $f(c_i)f(a_i) > 0$, asignăm $a_{i+1} = c_i$ şi $b_{i+1} = b_i$
- fiecare pas necesită o nouă evaluare a funcţiei f şi împarte intervalul care conţine o rădăcină, reducându-i lungimea cu un factor de 2
- după n paşi de calcul al lui c şi $f(c)$, am realizat $n + 2$ evaluări de funcţie, şi cea mai bună estimare a soluţiei este mijlocul ultimului interval obţinut

2.1.2 Cât de exactă şi cât de rapidă este metoda bisecţiei?

- dacă $[a, b]$ este intervalul iniţial, atunci după n paşi ai metodei bisecţiei, intervalul $[a_n, b_n]$ are lungimea $(b - a)/2^n$
- alegând mijlocul $x_c = (a_n + b_n)/2$ obţinem o aproximare a soluţiei r, care este la jumătate din lungimea intervalului de soluţia adevărată
- rezumând, după n paşi din metoda bisecţiei, avem că

$$\text{Eroarea soluţiei} = |x_c - r| < \frac{b-a}{2^{n+1}}, \tag{12}$$

$$\text{Evaluări de funcţie} = n + 2. \tag{13}$$

- o modalitate bună de a evalua eficienţa metodei bisecţiei este de a ne întreba câtă acurateţe poate fi adusă de fiecare evaluare de funcţie
- fiecare pas, sau fiecare evaluare de funcţie, reduce incertitudinea în găsirea rădăcinii cu un factor de 2

Definiţia 3

O soluţie este **corectă cu** p **zecimale exacte** dacă eroarea este mai mică decât 0.5×10^{-p}.

2.1.2 Cât de exactă şi cât de rapidă este metoda bisecţiei?

Exemplul 6

- folosiţi metoda bisecţiei pentru a găsi o rădăcină a funcţiei $f(x) = \cos x - x$ pe intervalul $[0, 1]$ cu 6 zecimale exacte
- în primul rând, vom decide câţi paşi ai metodei bisecţiei sunt necesari
- conform cu (12), eroarea după n paşi este $(b-a)/2^{n+1} = 1/2^{n+1}$
- din definiţia celor p zecimale exacte, trebuie ca

$$\begin{aligned} \frac{1}{2^{n+1}} &< 0.5 \times 10^{-6} \\ n &> \frac{6}{\log_{10} 2} \approx \frac{6}{0.301} = 19.9. \end{aligned}$$

- prin urmare, $n = 20$ paşi vor fi necesari

2.1.2 Cât de exactă şi cât de rapidă este metoda bisecţiei?

- aplicând metoda bisecţiei, obţinem următorul tabel:

k	a_k	$f(a_k)$	c_k	$f(c_k)$	b_k	$f(b_k)$
0	0.000000	$+$	0.500000	$+$	1.000000	$-$
1	0.500000	$+$	0.750000	$-$	1.000000	$-$
2	0.500000	$+$	0.625000	$+$	0.750000	$-$
3	0.625000	$+$	0.687500	$+$	0.750000	$-$
4	0.687500	$+$	0.718750	$+$	0.750000	$-$
5	0.718750	$+$	0.734375	$+$	0.750000	$-$
6	0.734375	$+$	0.742188	$-$	0.750000	$-$
7	0.734375	$+$	0.738281	$+$	0.742188	$-$
8	0.738281	$+$	0.740234	$-$	0.742188	$-$
9	0.738281	$+$	0.739258	$-$	0.740234	$-$
10	0.738281	$+$	0.738770	$+$	0.739258	$-$
11	0.738769	$+$	0.739014	$+$	0.739258	$-$
12	0.739013	$+$	0.739136	$-$	0.739258	$-$
13	0.739013	$+$	0.739075	$+$	0.739136	$-$

2.1.2 Cât de exactă şi cât de rapidă este metoda bisecţiei?

k	a_k	$f(a_k)$	c_k	$f(c_k)$	b_k	$f(b_k)$
14	0.739074	$+$	0.739105	$-$	0.739136	$-$
15	0.739074	$+$	0.739090	$-$	0.739105	$-$
16	0.739074	$+$	0.739082	$+$	0.739090	$-$
17	0.739082	$+$	0.739086	$-$	0.739090	$-$
18	0.739082	$+$	0.739084	$+$	0.739086	$-$
19	0.739084	$+$	0.739085	$-$	0.739086	$-$
20	0.739084	$+$	0.739085	$-$	0.739085	$-$

- aproximarea rădăcinii cu şase zecimale exacte este 0.739085

2.1.2 Cât de exactă şi cât de rapidă este metoda bisecţiei?

- pentru metoda bisecţiei, întrebarea câţi paşi să facem este una simplă—trebuie doar să alegem precizia dorită şi să găsim numărul de paşi necesari, ca în (12)
- vom vedea că unii algoritmi mai puternici sunt adesea mai puţin predictibili şi nu au un analog al (12)
- în acele cazuri, va fi nevoie să stabilim anumite „criterii de oprire” care vor determina circumstanţele în care algoritmul se va opri
- chiar şi pentru metoda bisecţiei, precizia finită a aritmeticii în calculator va pune o limită numărului posibil de zecimale exacte

Vă mulţumesc!