Referat 1

Tehnici şi metode numerice în securitatea informaţiei

Cod sursă: https://github.com/danielw98/numerical-methods/

Vizualizare site web: https://metode-numerice.danielwagner.ro/

Profesor universitar dr. habil. Mircea Merca

Student masterand: Ștefan-Daniel Wagner

Cuprins

1. Aproximarea soluțiilor ecuațiilor neliniare

1.1. Metoda bisecției

Scop

Metoda bisecției aproximează o rădăcină p a unei ecuații neliniare

$$f(x) = 0$$

pe un interval $[a, b]$, cu o precizie cerută $\varepsilon > 0$.

Ipoteze (condiția de bracketing)

Ca metoda să fie aplicabilă, folosim ipoteza clasică:

f este continuă pe $[a, b]$ și $f(a)$ și $f(b)$ au semne opuse (adică $f(a) \cdot f(b) < 0$).

Atunci există cel puțin o rădăcină în interval (teorema valorilor intermediare), iar algoritmul păstrează mereu o **încadrare** a rădăcinii.

Ideea metodei

Luăm mijlocul intervalului:

$$p = a + \frac{b - a}{2}$$

și păstrăm jumătatea în care semnul se schimbă: - dacă $\text{sgn}\big(f(a)\big) = \text{sgn}\big(f(p)\big)$ atunci rădăcina rămâne în $[p, b]$; - altfel rădăcina rămâne în $[a, p]$.

Notă practică: în implementare folosim formula $a + (b - a)/2$ (nu $(a + b)/2$), pentru a evita overflow/erori numerice când a și b sunt mari.

Algoritm (pseudocod)

```
Input: f, a, b, eps
Verifică: a < b și f(a), f(b) finite și sgn(f(a)) != sgn(f(b))

Pentru iter = 0..max:
    p  = a + (b - a)/2
    fp = f(p)

    Dacă |fp| <= eps:       return p
    Dacă (b - a)/2 <= eps:  return p   (garanție pe eroarea în x)

    Dacă sgn(fp) == sgn(f(a)):
        a = p
    altfel:
        b = p

Aruncă eroare NonConvergence dacă depășește numărul maxim de iterații.
```

Criterii de oprire (de ce sunt două)

În cod apar două condiții utile:

1) **Criteriu pe reziduu**: $|f(p)| \le \varepsilon$.
   - poate opri mai devreme când funcția este bine scalată;
   - nu oferă singur o garanție directă asupra erorii $|p - p^*|$ fără ipoteze suplimentare.
2) **Criteriu pe lățimea intervalului**: $(b - a)/2 \le \varepsilon$.
   - pentru bisecție avem o **limită garantată**:
     $$|p - p^*| \le \frac{b - a}{2}$$
     unde p^* este o rădăcină din interval;
   - acesta este criteriul „de precizie în soluție" tipic pentru bisecție.

În proiect, păstrăm ambele: intervalul îți dă garanția, iar reziduul poate scurta numărul de pași.

Estimarea erorii și numărul de iterații

După n pași, intervalul are lungimea:

$$b_n - a_n = \frac{b_0 - a_0}{2^n}$$

și eroarea maximă a aproximării prin mijloc este:

$$|p_n - p^*| \le \frac{b_n - a_n}{2} = \frac{b_0 - a_0}{2^{n+1}}.$$

De aici, un număr suficient de pași pentru o precizie în x este:

$$n \ge \left\lceil \log_2\left(\frac{b_0 - a_0}{\varepsilon}\right) \right\rceil.$$

Viteza de convergență

Bisecția are convergență **liniară**, cu factor $1/2$ (intervalul se înjumătățește la fiecare pas). Avantajul major este robustețea: dacă ipotezele sunt îndeplinite, metoda converge sigur.

Legătura cu implementarea din proiect

- Implementare: `RootFinding::bisection` în
  - `nm-lib/include/nonlinear/RootFinding.h`
  - `nm-lib/src/nonlinear/RootFinding.cpp`
- Verificări de intrare:
  - `validateEps(eps)` (cere $\varepsilon > 0$)
  - `validateBracket(eq, a, b)` (cere interval valid și semne opuse la capete)

- Pentru a urmări pașii metodei, putem activa trasarea (trace): la fiecare iterație se memorează valorile relevante, astfel încât să putem reconstrui evoluția algoritmului și să construim tabelul de iterații.
  - Structuri folosite: `BisectionTrace` / `BisectionTraceStep` (rețin a, b, mijlocul p, $f(p)$ și limita de eroare $(b-a)/2$).
- Programul de test `nm-lib/tests/tema1_rootfinding.cpp` poate emite rezultatele în JSON; opțiunea `--trace` include și pașii interni ai metodei.

Exemplu (ecuația 4)

Ecuația:

$$f(x) = 2x\cos(2x) - (x-2)^2$$

pe intervalul [2,3], cu $\varepsilon = 10^{-7}$.

Rulat din proiect: - `tema1_rootfinding.exe --json --trace 4 1`

Rezultat bisecție (din JSON): - $x \approx 2.370686948299408$ - limită garantată: $|x - x^*| \leq \frac{b-a}{2} \approx 5.96046447753906 \cdot 10^{-8} \leq 10^{-7}$ - (reziduu în acest punct) $f(x) \approx 2.69464500490812 \cdot 10^{-7}$

Primele iterații (valori rotunjite pentru lizibilitate):

iter	a	b	p(mijloc)	f(p)	err(b-a)/2
0	2.00000	3.00000	2.5 0000	1.16831	0.5
1	2.00000	2.50000	2.25000	-1.01108	0.25
2	2.25000	2.50000	2.37500	0.0379852	0.125
3	2.25000	2.37500	2.31250	-0.501316	0.0625
4	2.31250	2.37500	2.34375	-0.234819	0.03125
5	2.34375	2.37500	2.35938	-0.0991345	0.015625
6	2.35938	2.37500	2.36719	-0.0307450	0.0078125
...	...	...	...	...	...
23	2.370686889	2.370687008	2.370686948	2.694645e-07	5.960464e-08

Observație: intervalul se înjumătățește la fiecare pas, iar limita erorii scade ca $1/2^n$.

Pașii metodei

Se notează

$$f(x) = 2x\cos(2x) - (x - 2)^2,$$

cu intervalul inițial $[a_0, b_0] = [2,3]$.

Iterația 0

Mijlocul intervalului inițial este

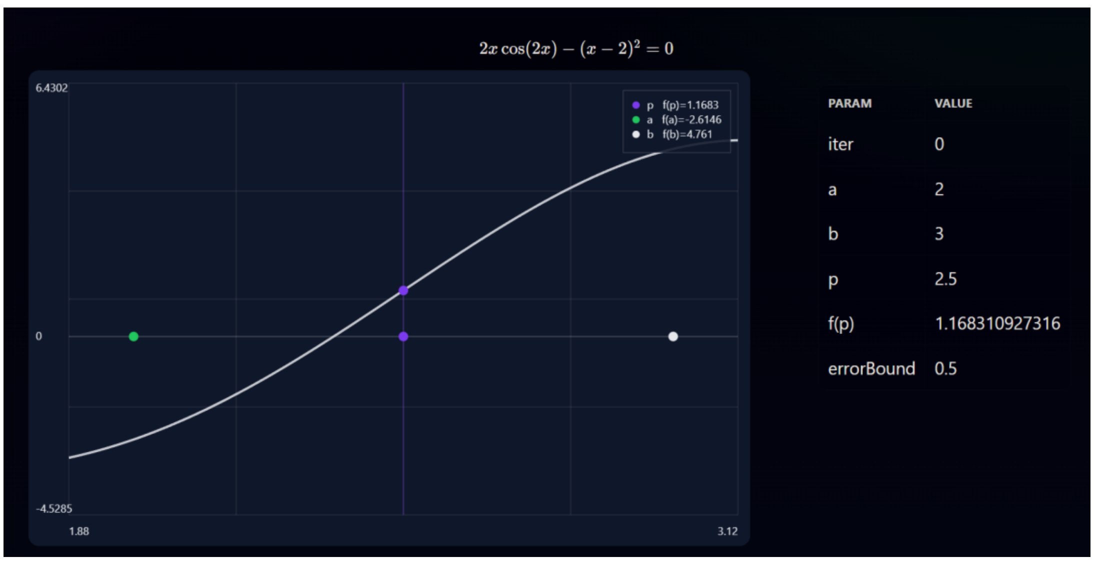
$$p_0 = a_0 + \frac{b_0 - a_0}{2} = 2 + \frac{3 - 2}{2} = 2.5.$$

Valorile funcției (rotunjite):

$$f(a_0) = f(2) < 0, \quad f(b_0) = f(3) > 0, \quad f(p_0) \approx 1.16831 > 0.$$

Semnul lui $f(p_0)$ este același cu al lui $f(b_0)$, deci se păstrează jumătatea stângă:

$$[a_1, b_1] = [a_0, p_0] = [2,2.5], \quad \text{eroare garantată } \frac{b_0 - a_0}{2} = 0.5.$$



Iterația 1

Mijlocul noului interval este

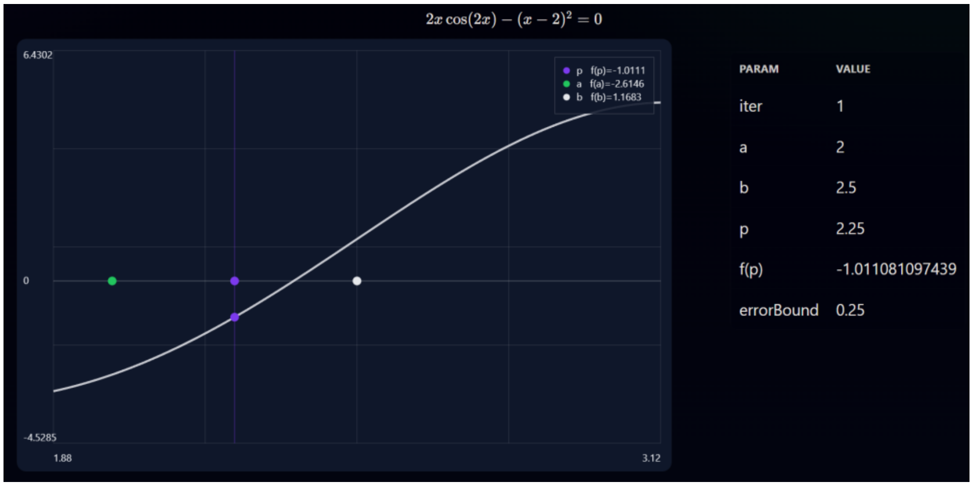
$$p_1 = a_1 + \frac{b_1 - a_1}{2} = 2 + \frac{2.5 - 2}{2} = 2.25.$$

Valoarea funcției în acest punct este

$$f(p_1) \approx -1.01108 < 0.$$

Valoarea $f(p_1)$ are același semn cu $f(a_1)$, deci se păstrează jumătatea dreaptă:

$$[a_2, b_2] = [p_1, b_1] = [2.25, 2.5], \quad \text{eroare garantată } \frac{b_1 - a_1}{2} = 0.25.$$



Iterația 2

Mijlocul intervalului curent este

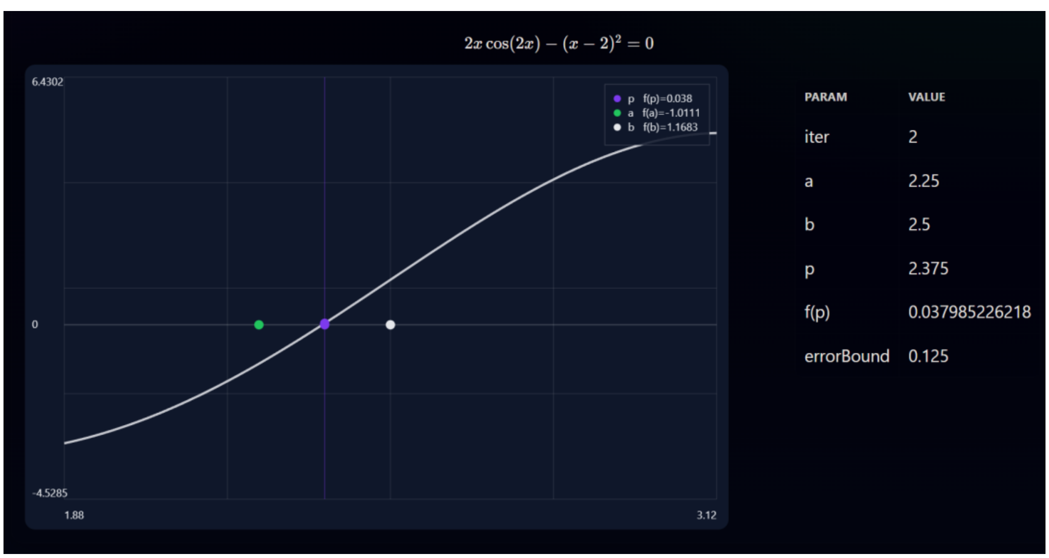
$$p_2 = a_2 + \frac{b_2 - a_2}{2} = 2.25 + \frac{2.5 - 2.25}{2} = 2.375.$$

Valoarea funcției:

$$f(p_2) \approx 0.0379852 > 0.$$

Semnul lui $f(p_2)$ coincide cu cel al lui $f(b_2)$, de unde

$$[a_3, b_3] = [a_2, p_2] = [2.25, 2.375], \quad \text{eroare garantată } \frac{b_2 - a_2}{2} = 0.125.$$



Iterația 3

Mijlocul intervalului este

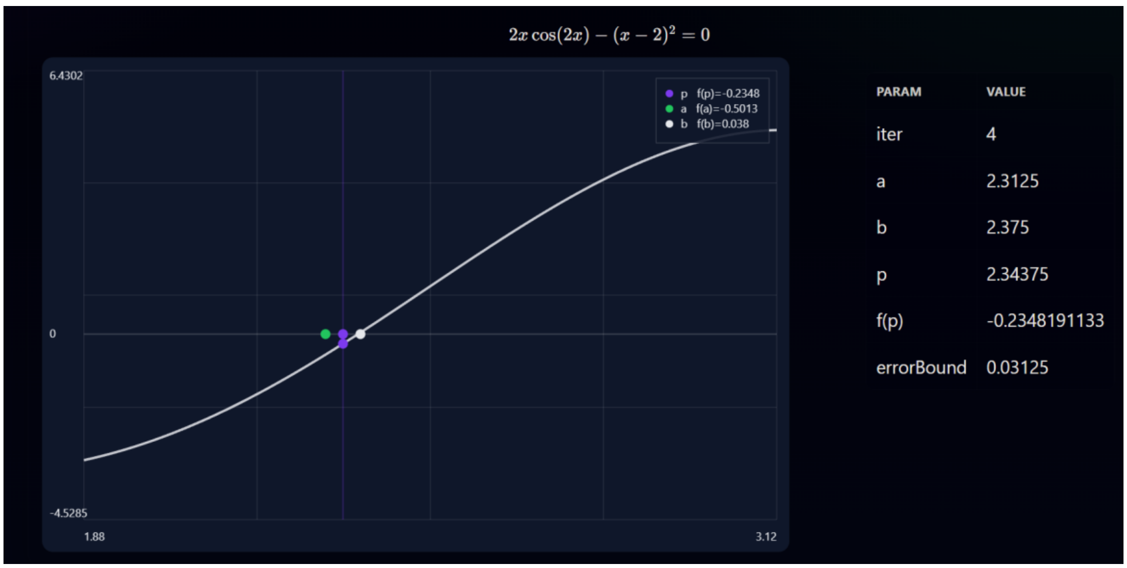
$$p_3 = a_3 + \frac{b_3 - a_3}{2} = 2.25 + \frac{2.375 - 2.25}{2} = 2.3125.$$

Valoarea funcției:

$$f(p_3) \approx -0.501316 < 0.$$

În acest caz, se păstrează jumătatea dreaptă:

$$[a_4, b_4] = [p_3, b_3] = [2.3125, 2.375], \quad \text{eroare garantată } \frac{b_3 - a_3}{2} = 0.0625.$$



Ultimul pas (iterația 23)

După 24 de iterații (de la $k = 0$ la $k = 23$), intervalul ajunge la

$$[a_{23}, b_{23}] = [2.370686888694763,\ 2.3706870079040527].$$

Mijlocul este

$$p_{23} = 2.370686948299408,$$

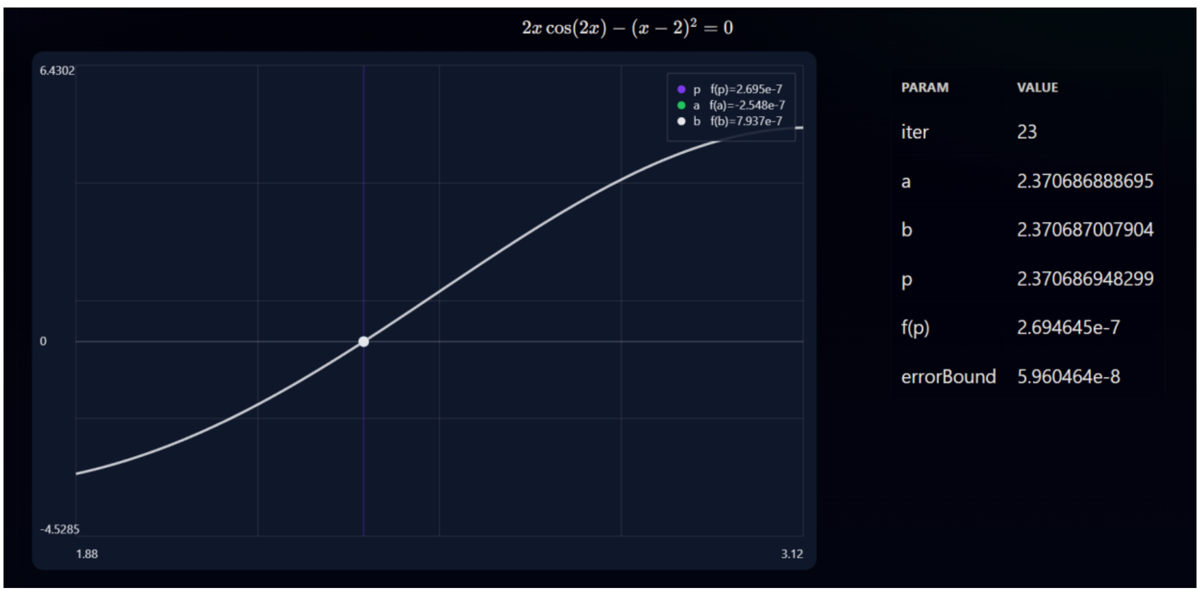
iar funcția în acest punct are valoare foarte mică:

$$f(p_{23}) \approx 2.694645 \cdot 10^{-7}.$$

Eroarea garantată pe soluția exactă p^* este dată de jumătatea lățimii intervalului curent:

$$|p_{23} - p^*| \leq \frac{b_{23} - a_{23}}{2} = 5.960464477539063 \cdot 10^{-8} \leq 10^{-7},$$

astfel încât aproximarea p_{23} respectă precizia cerută în x, iar $|f(p_{23})|$ este, de asemenea, foarte mic.



1.2. Metoda Newton (metoda tangentei)

Scop

Metoda Newton aproximează o rădăcină p a ecuației neliniare

$$f(x) = 0$$

plecând de la o aproximație inițială x_0 și folosind derivata $f'(x)$.

Ipoteze (când are sens să o folosim)

În practică, metoda este folosită când: - f este derivabilă în vecinătatea rădăcinii; - $f'(x)$ este definită și nu se anulează în punctele vizitate; - alegem un x_0 suficient de „aproape” de rădăcina căutată.

Spre deosebire de bisecție, Newton **nu este o metodă de bracketing**: nu păstrează un interval $[a, b]$ care garantează existența rădăcinii.

Ideea metodei

În punctul curent x_k, aproximăm funcția prin tangenta:

$$f(x) \approx f(x_k) + f'(x_k)(x - x_k).$$

Intersecția acestei drepte cu axa Ox (adică $f(x) = 0$) dă noul punct:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}.$$

Geometric: x_{k+1} este abscisa punctului unde tangenta în $\left(x_k, f(x_k)\right)$ taie axa Ox.

Algoritm (pseudocod)

```
Input: f, f', x0, eps
Verifică: eps > 0, f' definită

Pentru iter = 0..max:
    df = f'(x0)
    Dacă df == 0: aruncă NonConvergence (nu putem împărți)

    p  = x0 - f(x0)/df
    fp = f(p)

    Dacă |fp| <= eps:     return p
    Dacă |p - x0| <= eps: return p

    x0 = p

Aruncă eroare NonConvergence dacă depășește numărul maxim de iterații.
```

Criterii de oprire

În implementare apar două criterii (aceeași idee ca în bisecție: una pe reziduu, una pe pas):

1. **Criteriu pe reziduu**: $|f(x_{k+1})| \leq \varepsilon$.
2. **Criteriu pe schimbarea iteratelor**: $|x_{k+1} - x_k| \leq \varepsilon$.

Newton nu are, în general, o limită garantată a erorii în x de forma bisecției (nu avem un interval care se micșorează monoton).

Viteza de convergență (intuție)

- Pentru o rădăcină **simplă** (adică $f(p) = 0$ și $f'(p) \neq 0$) și un x_0 suficient de bun, Newton are convergență **cvadratică**.
- Dacă rădăcina este **multiplu** (de ex. $f'(p) = 0$), convergența poate deveni mult mai lentă.

Legătura cu implementarea din proiect

- Implementare: `RootFinding::newton` în
  - `nm-lib/include/nonlinear/RootFinding.h`
  - `nm-lib/src/nonlinear/RootFinding.cpp`
- Verificări de intrare:
  - `validateEps(eps)` (cere $\varepsilon > 0$)
  - verificare că derivata este furnizată și produce valori finite
  - verificare `df != 0` (altfel nu putem aplica formula)
- Pentru a urmări pașii metodei, putem activa trasarea (trace): la fiecare iterație se memorează valorile relevante, astfel încât să putem reconstrui evoluția metodei și să construim tabelul de iterații.
  - Structuri folosite: `NewtonTrace` / `NewtonTraceStep` (rețin x_k, $f(x_k)$, $f'(x_k)$, x_{k+1}, $f(x_{k+1})$).
- Programul de test `nm-lib/tests/tema1_rootfinding.cpp` poate emite rezultatele în JSON; opțiunea `--trace` include și pașii interni ai metodei.

Exemplu (ecuația 4)

Ecuația:

$$f(x) = 2x\cos(2x) - (x-2)^2,$$

cu derivata:

$$f'(x) = 2\cos(2x) - 4x\sin(2x) - 2(x-2).$$

Pe intervalul [2,3], cu $\varepsilon = 10^{-7}$, în programul de test Newton pornește din

$$x_0 = \frac{a+b}{2} = 2.5.$$

Rulat din proiect: - `tema1_rootfinding.exe --json --trace 4 1`

În output-ul JSON, rezultatul Newton apare în `methods[]` la intrarea cu `name="newton"`.

Rezultat Newton (din JSON): $x \approx 2.370686917662517$, cu $f(x) \approx 2.25 \cdot 10^{-12}$.

Primele iterații (valori rotunjite pentru lizibilitate):

iter	$\boldsymbol{x_k}$	$\boldsymbol{f(x_k)}$	$\boldsymbol{f'(x_k)}$	$\boldsymbol{x_{k+1}}$	$\boldsymbol{f(x_{k+1})}$
0	2.5000000	1.16831093	9.15656712	2.3724073	0.0151395834
1	2.3724073	0.0151395834	8.80466623	2.3706878	7.98693159e-06
2	2.3706878	7.98693159e-06	8.79535732	2.3706869	2.24761876e-12

În acest exemplu, Newton converge foarte repede (în câțiva pași) deoarece pornirea $x_0 = 2.5$ este deja în vecinătatea rădăcinii și $f'(x)$ nu este aproape de zero pe parcurs.

Pașii metodei

Metoda aplicată la această ecuație urmează formula

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)},$$

cu $x_0 = 2.5$.

Iterația 0

Se pornește din

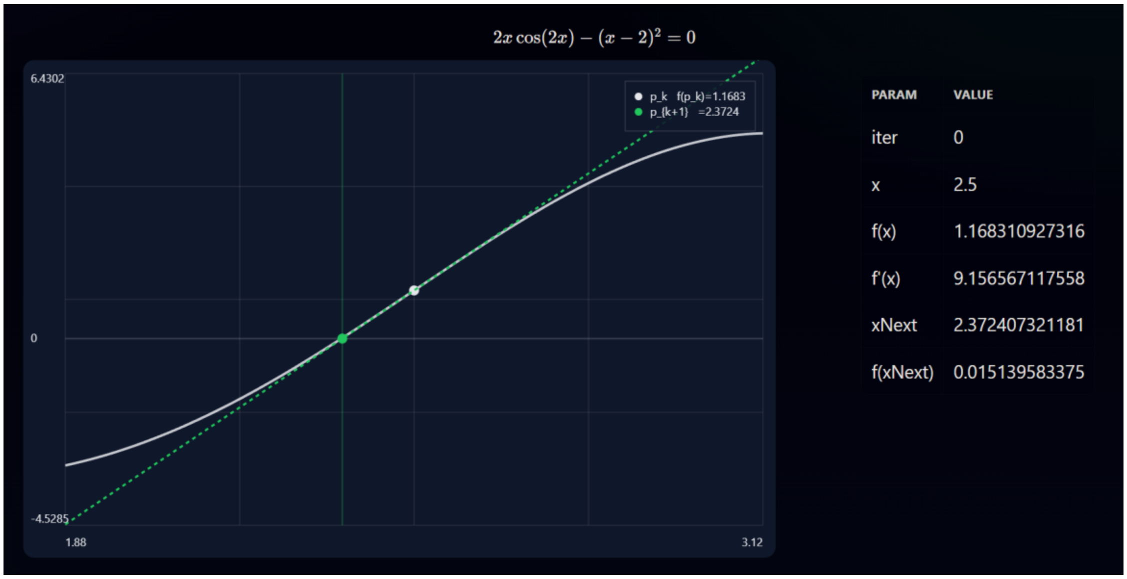
$$x_0 = 2.5, \quad f(x_0) \approx 1.16831093, \quad f'(x_0) \approx 9.15656712.$$

Aplicarea formulei dă

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} \approx 2.3724073,$$

iar

$$f(x_1) \approx 0.01513958.$$



Metoda Newton — iterația 0

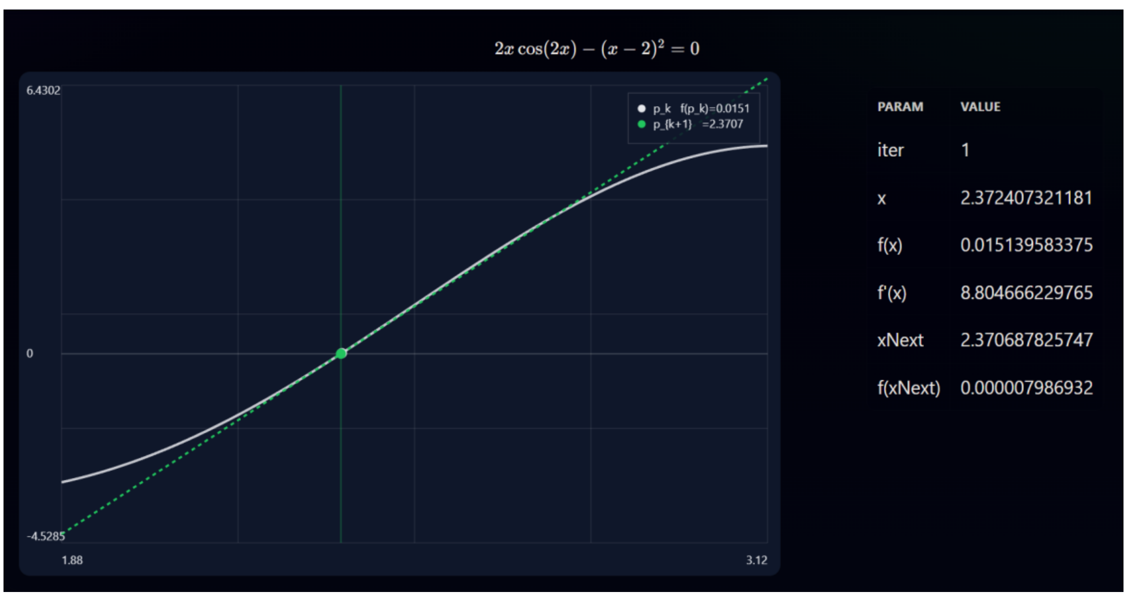
Iterația 1

Se continuă cu

$$x_1 \approx 2.3724073, \quad f(x_1) \approx 0.01513958, \quad f'(x_1) \approx 8.80466623,$$

ceea ce conduce la

$$x_2 \approx 2.3706878, \quad f(x_2) \approx 7.99 \cdot 10^{-6}.$$



Metoda Newton — iterația 1

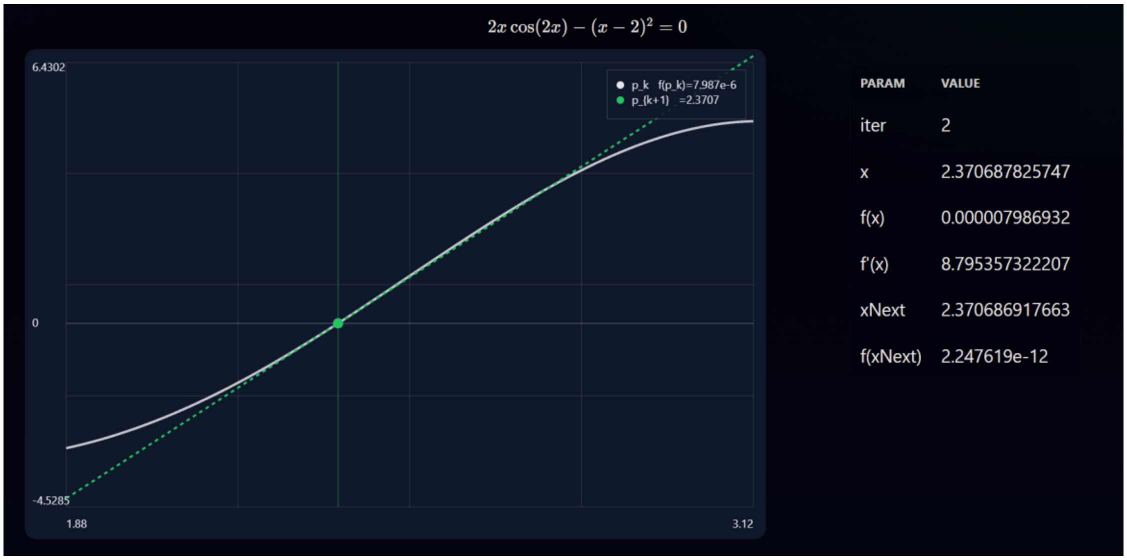
Iterația 2 (pasul final aici)

Încă un pas, cu

$$x_2 \approx 2.3706878, \quad f(x_2) \approx 7.99 \cdot 10^{-6}, \quad f'(x_2) \approx 8.79535732,$$

duce la

$$x_3 \approx 2.3706869, \quad f(x_3) \approx 2.25 \cdot 10^{-12}.$$



Metoda Newton — iterația 2

La acest punct, atât diferența $|x_3 - x_2|$, cât și reziduul $|f(x_3)|$ sunt foarte mici față de toleranța uzuală, astfel că x_3 poate fi considerat o aproximare numerică stabilizată a rădăcinii.

1.3. Metoda regula falsi (false position)

Scop

Metoda regula falsi aproximează o rădăcină p a ecuației neliniare

$$f(x) = 0$$

pe un interval $[a, b]$, cu o precizie cerută $\varepsilon > 0$.

Ipoteze (condiția de bracketing)

Ca metoda să fie aplicabilă, folosim aceleași ipoteze ca la bisecție:

- f este continuă pe $[a, b]$;
- $f(a)$ și $f(b)$ au semne opuse ($f(a) \cdot f(b) < 0$).

Astfel, există cel puțin o rădăcină în interval, iar algoritmul păstrează mereu o **încadrare** a rădăcinii.

Ideea metodei

În loc să luăm mijlocul intervalului, construim secanta prin punctele $\big(a, f(a)\big)$ și $\big(b, f(b)\big)$ și luăm intersecția ei cu axa Ox.

Formula punctului de intersecție (folosită în implementare) este:

$$p = \frac{a\,f(b) - b\,f(a)}{f(b) - f(a)}.$$

Apoi păstrăm jumătatea care rămâne o încadrare:

- dacă $\operatorname{sgn}\big(f(a)\big) = \operatorname{sgn}\big(f(p)\big)$, înlocuim $a \leftarrow p$;
- altfel înlocuim $b \leftarrow p$.

Algoritm (pseudocod)

```
Input: f, a, b, eps
Verifică: a < b și f(a), f(b) finite și sgn(f(a)) != sgn(f(b))

prevP = NaN
Pentru iter = 0..max:
    denom = f(b) - f(a)
    Dacă denom == 0: eroare

    p  = (a*f(b) - b*f(a)) / denom
    fp = f(p)

    Dacă |fp| <= eps:             return p
    Dacă prevP finit și |p-prevP| <= eps: return p
    Dacă |b-a| <= 2*eps:          return p

    Dacă sgn(fp) == sgn(f(a)):
        a = p; f(a) = fp
    altfel:
        b = p; f(b) = fp

    prevP = p

Aruncă eroare NonConvergence dacă depășește numărul maxim de iterații.
```

Criterii de oprire

În proiect apar trei condiții utile:

1. **Criteriu pe reziduu**: $|f(p)| \leq \varepsilon$.
2. **Criteriu pe schimbarea aproximării**: $|p - \text{prevP}| \leq \varepsilon$.
3. **Criteriu pe lățimea intervalului**: $|b - a| \leq 2\varepsilon$.

Observație: regula falsi păstrează intervalul de bracketing, dar nu are o „limită de eroare” la fel de simplă ca bisecția (intervalul nu se înjumătățește la fiecare pas).

Viteza de convergență (intuție)

Regula falsi este adesea mai rapidă decât bisecția pentru funcții „aproape liniare” pe interval, deoarece punctul p este „ghidat” de valori $f(a)$ și $f(b)$ (nu este forțat să fie mijlocul).

În schimb, există situații în care unul dintre capete rămâne mult timp fix (metoda poate stagna), de aceea în practică se folosesc și variante modificate.

Legătura cu implementarea din proiect

- Implementare: `RootFinding::regulaFalsi` în
  - `nm-lib/include/nonlinear/RootFinding.h`
  - `nm-lib/src/nonlinear/RootFinding.cpp`
- Verificări de intrare:
  - `validateEps(eps)` (cere $\varepsilon > 0$)
  - `validateBracket(eq, a, b)` (cere interval valid și semne opuse la capete)
- Pentru a urmări pașii metodei, putem activa trasarea (trace): la fiecare iterație se memorează valorile relevante, astfel încât să putem reconstrui evoluția algoritmului și să construim tabelul de iterații.
  - Structuri folosite: `RegulaFalsiTrace` / `RegulaFalsiTraceStep` (rețin a, b, p și $f(p)$).
- Programul de test `nm-lib/tests/tema1_rootfinding.cpp` poate emite rezultatele în JSON; opțiunea `--trace` include și pașii interni ai metodei.

Exemplu (ecuația 4)

Ecuația:

$$f(x) = 2x\cos(2x) - (x-2)^2$$

pe intervalul [2,3], cu $\varepsilon = 10^{-7}$.

Rulat din proiect: - `tema1_rootfinding.exe --json --trace 4 1`

Rezultat regula falsi (din JSON): $x \approx 2.370686907966889$, cu $f(x) \approx -8.53 \cdot 10^{-8}$.

Primele iterații (valori rotunjite pentru lizibilitate):

iter	a	b	p (secantă)	f(p)
0	2.00000	3.00000	2.3544899	-0.1417167
1	2.3544899	3.00000	2.3731488	0.02166939
2	2.3544899	2.3731488	2.3706741	-0.000112597
3	2.3706741	2.3731488	2.3706869	-8.5274e-08

Observație: metoda păstrează bracketing-ul (semne opuse la capete), dar pasul nu este forțat să înjumătățească intervalul ca la bisecție.

Pașii metodei

Pentru aceeași ecuație și același interval [2,3], se pot detalia câțiva pași, folosind formula secantei pe capetele intervalului.

Iterația 0

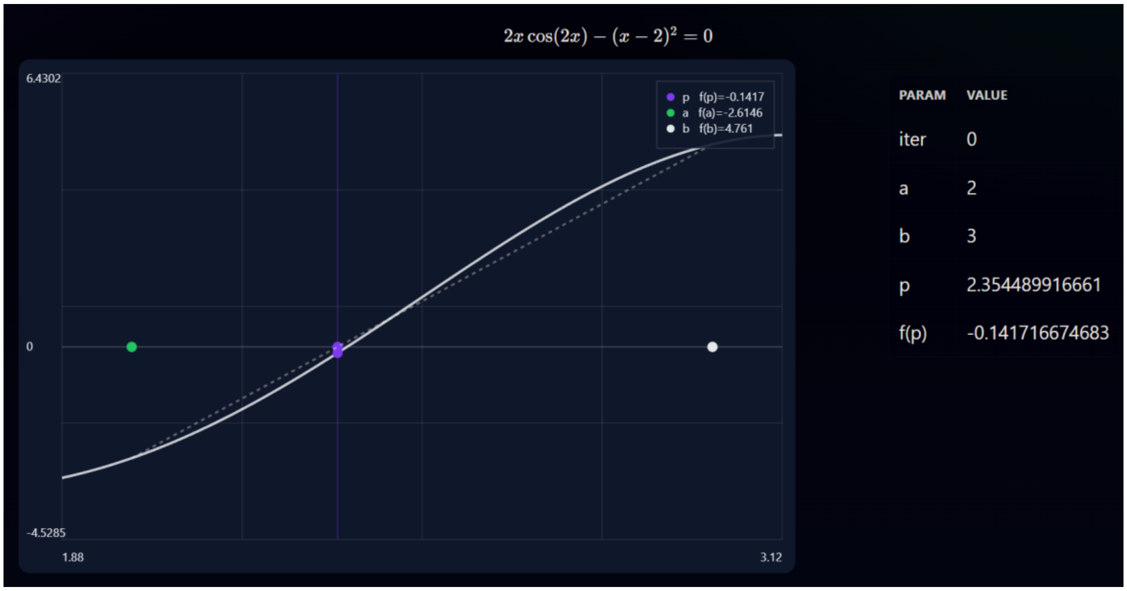
Punctul obținut din secantă este

$$p_0 = \frac{a_0 f(b_0) - b_0 f(a_0)}{f(b_0) - f(a_0)} \approx 2.3544899,$$

unde $a_0 = 2$, $b_0 = 3$ și valorile $f(a_0), f(b_0)$ sunt luate din program.

Semnul lui $f(p_0) \approx -0.1417167$ este același cu al lui $f(a_0)$, astfel că noul interval rămâne o încadrare:

$$[a_1, b_1] = [p_0, b_0] = [2.3544899,3].$$



Metoda regula falsi — iterația 0

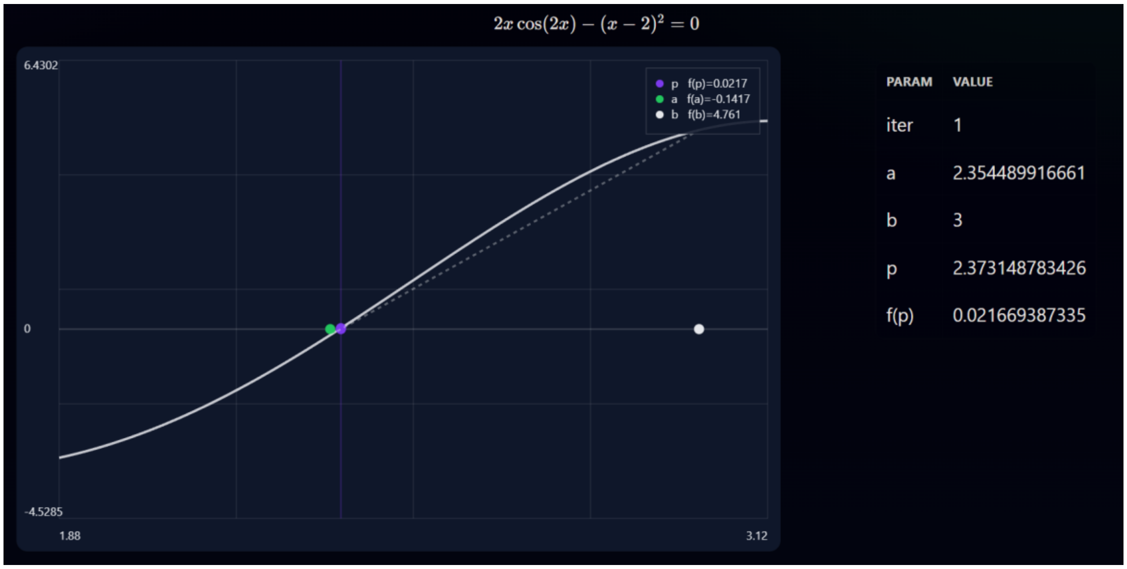
Iterația 1

Aplicând din nou formula secantei pe $[a_1, b_1]$ se obține

$$p_1 = \frac{a_1 f(b_1) - b_1 f(a_1)}{f(b_1) - f(a_1)} \approx 2.3731488.$$

În acest punct, $f(p_1) \approx 0.02166939$ are același semn cu $f(b_1)$, deci noul interval devine

$$[a_2, b_2] = [a_1, p_1] = [2.3544899,2.3731488].$$



Metoda regula falsi — iterația 1

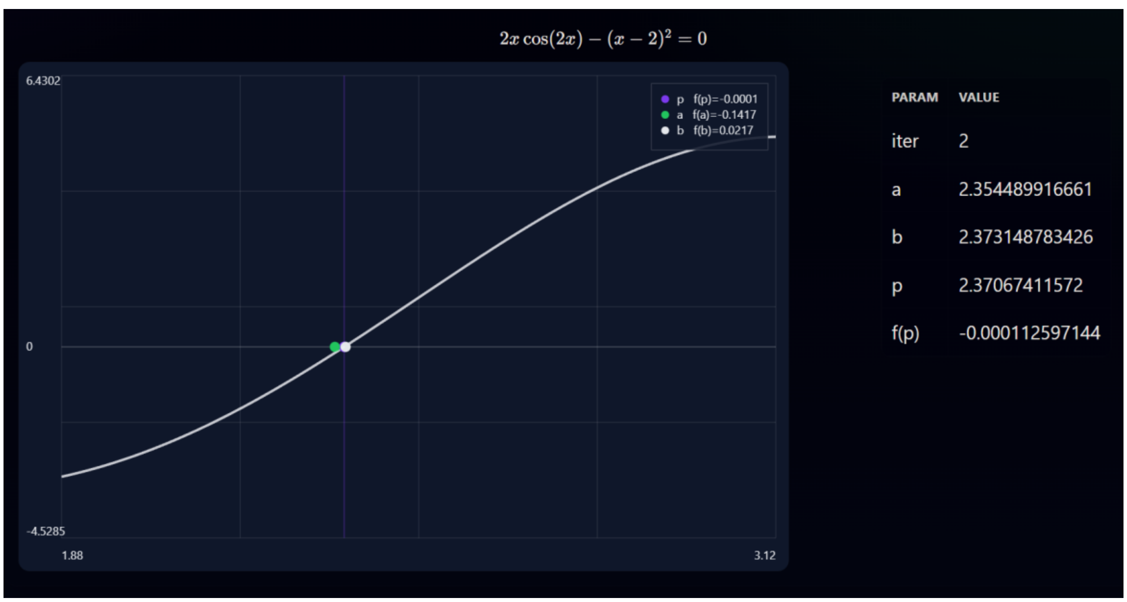
Iterația 2

Se repetă același tip de calcul:

$$p_2 \approx 2.3706741, \quad f(p_2) \approx -0.000112597 < 0.$$

Semnul lui $f(p_2)$ este același cu al lui $f(a_2)$, astfel că intervalul se strânge în jurul rădăcinii:

$$[a_3, b_3] = [p_2, b_2] = [2.3706741, 2.3731488].$$



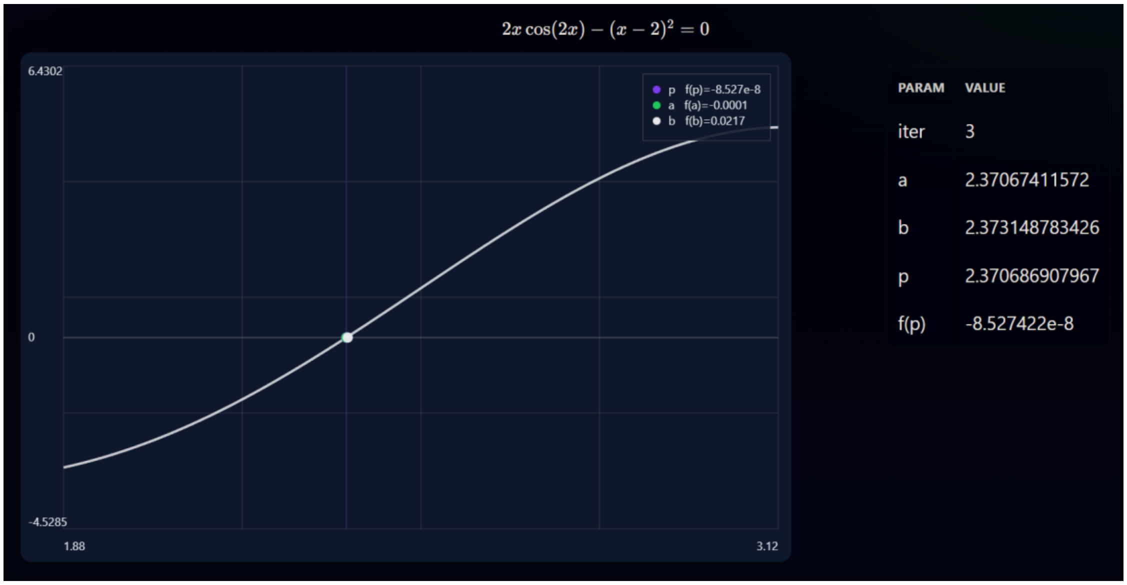
Metoda regula falsi — iterația 2

Iterația 3 (pasul final în acest exemplu)

O nouă aplicare a formulei secantei duce la

$$p_3 \approx 2.3706869, \quad f(p_3) \approx -8.53 \cdot 10^{-8}.$$

Valoarea obținută respectă criteriul pe reziduu $|f(p_3)| \leq \varepsilon$, iar intervalul $[a_3, b_3]$ rămâne în continuare o încadrare pentru rădăcină.



Metoda regula falsi — iterația 3

1.4. Metoda secantei

Scop

Metoda secantei aproximează o rădăcină p a ecuației neliniare

$$f(x) = 0$$

folosind doar evaluări ale funcției (fără derivată), pornind de la două aproximații inițiale x_0 și x_1.

Ipoteze (când are sens să o folosim)

Metoda secantei este o metodă **deschisă**: - nu cere un interval de bracketing $[a, b]$; - cere două puncte inițiale pentru care $f(x_0)$ și $f(x_1)$ sunt finite; - în practică, alegerea bună a lui x_0, x_1 contează mult pentru convergență.

Ideea metodei

În loc de tangenta din Newton, folosim secanta prin punctele $\big(x_0, f(x_0)\big)$ și $\big(x_1, f(x_1)\big)$. Intersecția secantei cu axa Ox dă noul punct:

$$p = x_1 - f(x_1)\,\frac{x_1 - x_0}{f(x_1) - f(x_0)}.$$

Apoi „mutăm fereastra” și repetăm: $(x_0, x_1) \leftarrow (x_1, p)$.

Algoritm (pseudocod)

```
Input: f, x0, x1, eps
Verifică: eps > 0 și f(x0), f(x1) finite

Pentru iter = 0..max:
    denom = f(x1) - f(x0)
    Dacă denom == 0: eroare

    p  = x1 - f(x1) * (x1 - x0) / denom
    fp = f(p)

    Dacă |fp| <= eps:     return p
    Dacă |p - x1| <= eps: return p

    x0 = x1; f0 = f1
    x1 = p;  f1 = fp

Aruncă eroare NonConvergence dacă depășește numărul maxim de iterații.
```

Criterii de oprire

În implementare apar două condiții: 1) **Criteriu pe reziduu**: $|f(p)| \le \varepsilon$. 2) **Criteriu pe pas**: $|p - x_1| \le \varepsilon$.

Viteza de convergență (intuție)

- Secanta este, de obicei, mai rapidă decât bisecția și nu are nevoie de derivată (față de Newton).
- Convergența nu este garantată în general; pot apărea probleme dacă $f(x_1) - f(x_0)$ devine foarte mic sau dacă pornirea este slabă.

Legătura cu implementarea din proiect

- Implementare: `RootFinding::secant` în
  - `nm-lib/include/nonlinear/RootFinding.h`
  - `nm-lib/src/nonlinear/RootFinding.cpp`
- Verificări de intrare:
  - `validateEps(eps)` (cere $\varepsilon > 0$)
  - verificare că $f(x_0)$ și $f(x_1)$ sunt finite
  - verificare `f(x1) - f(x0) != 0` (altfel nu putem aplica formula)
- Pentru a urmări pașii metodei, putem activa trasarea (trace): la fiecare iterație se memorează valorile relevante, astfel încât să putem reconstrui evoluția metodei și să construim tabelul de iterații.
  - Structuri folosite: `SecantTrace` / `SecantTraceStep` (rețin x_0, x_1, p și $f(p)$).
- Programul de test `nm-lib/tests/tema1_rootfinding.cpp` poate emite rezultatele în JSON; opțiunea `--trace` include și pașii interni ai metodei.

Exemplu (ecuația 4)

Ecuația:

$$f(x) = 2x\cos(2x) - (x-2)^2$$

Rulat din proiect: - `tema1_rootfinding.exe --json --trace 4 1`

Rezultat secantă (din JSON): $x \approx 2.370686907966889$, cu $f(x) \approx -8.53 \cdot 10^{-8}$.

Primele iterații (valori rotunjite pentru lizibilitate):

iter	x_0	x_1	p (secantă)	f(p)
0	2.00000	3.00000	2.3544899	-0.1417167
1	3.00000	2.3544899	2.3731488	0.02166939
2	2.3544899	2.3731488	2.3706741	-0.000112597
3	2.3731488	2.3706741	2.3706869	-8.5274e-08

Observație: în acest exemplu, secanta ajunge la aceeași aproximare numerică ca regula falsi, deoarece pornește din aceleași două puncte și folosește aceeași formulă de intersecție; diferența este că secanta nu impune bracketing-ul (nu menține semne opuse la capete).

Pașii metodei

Se pornește din aceleași două aproximații $x_0 = 2$ și $x_1 = 3$ și se aplică formula secantei.

Iterația 0

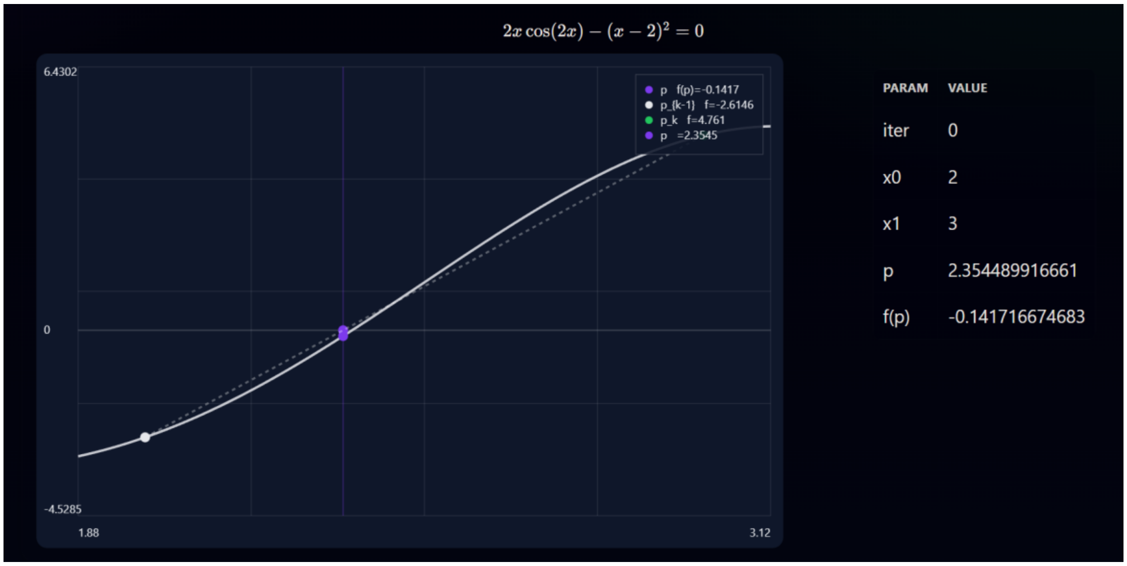
Primul punct nou este

$$p_0 = x_1 - f(x_1)\,\frac{x_1 - x_0}{f(x_1) - f(x_0)} \approx 2.3544899,$$

iar valoarea funcției este $f(p_0) \approx -0.1417167$.

Perechea (x_0, x_1) se actualizează la (x_1, p_0), deci

$$(x_0, x_1) = (3.0,\ 2.3544899).$$



Metoda secantei — iterația 0

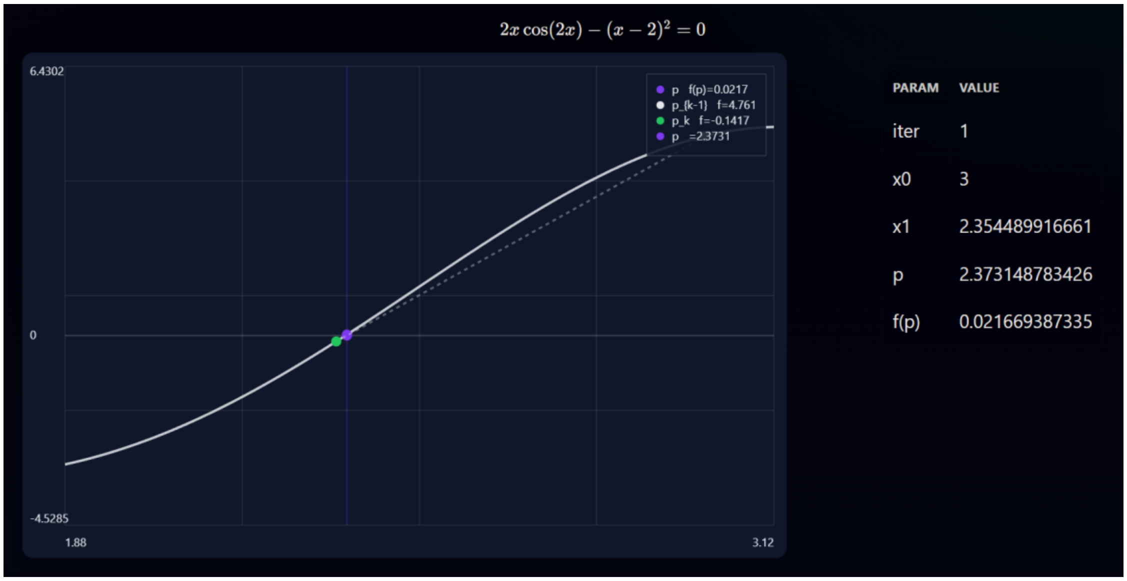
Iterația 1

Aplicarea formulei pe noile valori dă

$$p_1 \approx 2.3731488, \quad f(p_1) \approx 0.02166939.$$

Se trece la

$$(x_0, x_1) = (2.3544899,\ 2.3731488).$$



Metoda secantei — iterația 1

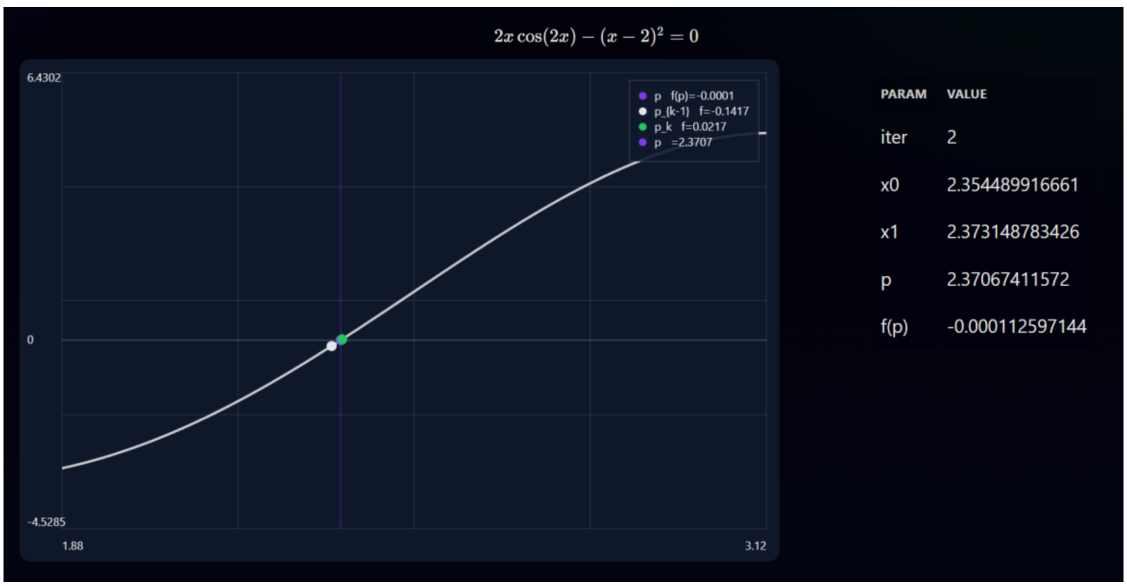
Iterația 2

Un nou pas produce

$$p_2 \approx 2.3706741, \quad f(p_2) \approx -0.000112597,$$

iar perechea devine

$$(x_0, x_1) = (2.3731488,\ 2.3706741).$$



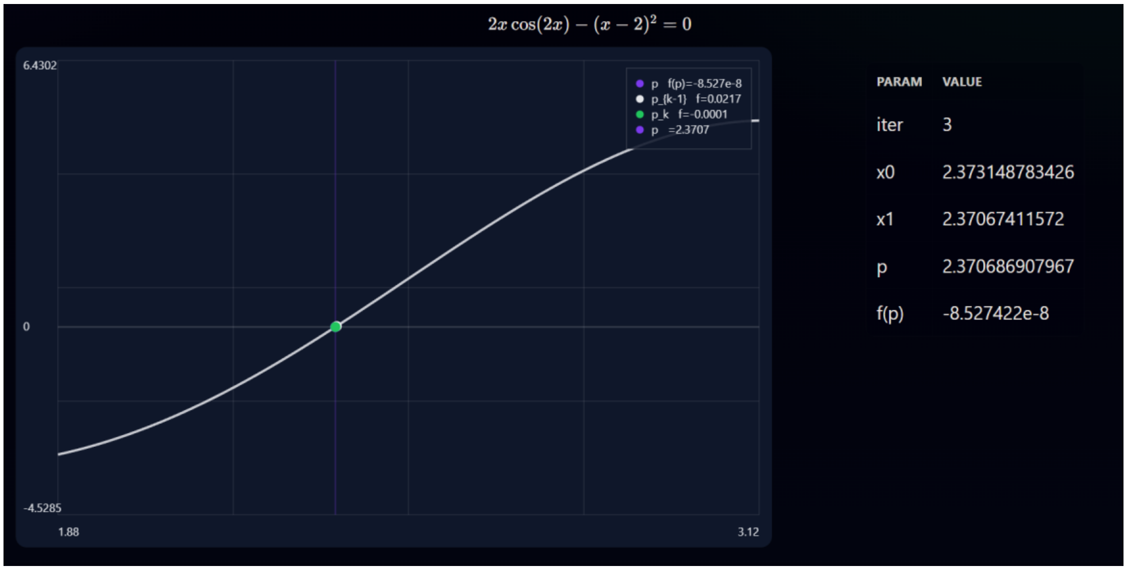
Metoda secantei — iterația 2

Iterația 3 (pasul final aici)

Ultima aplicare a formulei secantei conduce la

$$p_3 \approx 2.3706869, \quad f(p_3) \approx -8.53 \cdot 10^{-8}.$$

Acest punct este folosit ca aproximație finală, deoarece atât pasul $|p_3 - x_1|$, cât și reziduul $|f(p_3)|$ satisfac criteriile de oprire impuse.



Metoda secantei — iterația 3

2. Aproximarea soluțiilor sistemelor liniare de ecuații

2.1. Eliminare Gauss

Scop

Rezolvăm un sistem liniar

$$Ax = b,$$

folosind **eliminare Gauss cu pivotare parțială**, dar modelând „aritmetica cu 3 cifre” ca: - **rotunjire la 3 cifre semnificative** după operațiile relevante; - regula de rotunjire pentru cazul „5 urmat doar de zerouri”: **round-half-to-even**.

În proiect, acest model este implementat explicit și folosit în pașii eliminării.

Ideea metodei

1) **Eliminare înainte (forward elimination)**: transformăm sistemul într-unul echivalent cu o matrice superior triunghiulară U.

2) **Substituție înapoi (back substitution)**: rezolvăm apoi $Ux = \tilde{b}$ începând de la ultima ecuație.

Pivotare parțială (de ce e necesară)

La pasul k alegem ca pivot linia cu cel mai mare modul pe coloana k (pentru $i \geq k$):

$$\text{pivotRow} = \underset{i \geq k}{\operatorname{argmax}} |A_{ik}|.$$

Apoi schimbăm liniile (dacă e nevoie). Pivotarea reduce amplificarea erorilor de rotunjire când un pivot este foarte mic.

Model de „aritmetică cu 3 cifre”

În implementare, se folosește o funcție de rotunjire: - $r(\cdot)$ = rotunjire la `significantDigits` (aici 3).

În timpul eliminării, rotunjim consistent:

- multiplicatorul: $m = r\left(\frac{A_{ik}}{A_{kk}}\right)$

- produsul: $p = r(m \cdot A_{kj})$

- scăderea: $A_{ij} \leftarrow r(A_{ij} - p)$

- și pentru termenul liber: $b_i \leftarrow r(b_i - r(m \cdot b_k))$

Algoritm (pseudocod)

```
Input: A (n-n), b (n), significantDigits = 3
r(v) = roundToSignificantDigits(v, significantDigits)

Pentru k = 0..n-1:
  alege pivotRow = argmax_{i>=k} |A[i,k]|
  dacă |A[pivotRow,k]| este ~0 => matrice singulară
  dacă pivotRow != k => swap linii (A și b)

  pivot = A[k,k]
  pentru i = k+1..n-1:
    m = r(A[i,k] / pivot)
    A[i,k] = 0
    pentru j = k+1..n-1:
      A[i,j] = r(A[i,j] - r(m * A[k,j]))
    b[i] = r(b[i] - r(m * b[k]))

Back-substitution:
  pentru i = n-1..0:
    sum = r( sum_{j=i+1..n-1} r(A[i,j] * x[j]) )
    x[i] = r((b[i] - sum) / A[i,i])

return x
```

Legătura cu implementarea din proiect

- Rotunjire: `roundToSignificantDigits` în `nm-lib/src/utils/Rounding.cpp`.
- Eliminare Gauss: `GaussianElimination::solve` în `nm-lib/src/linear/GaussianElimination.cpp`.
- Program de test: `nm-lib/tests/tema2_gauss.cpp`.

Rulare (cu JSON + trace): - `tema2_gauss.exe --json --trace 1`

Output-ul JSON include: - matricea A, vectorul b, soluția x; - reziduul $\| Ax - b \|_{\infty}$; - opțional (cu `--trace`) pașii de eliminare înainte (matricea și vectorul după fiecare pas k).

2.2. Eliminare Gauss (exemplu)

Problema

Rezolvăm sistemul (4) din programul de test al proiectului (`nm-lib/tests/tema2_gauss.cpp`), folosind: - **pivotare parțială**; - **rotunjire la 3 cifre semnificative** după pașii de calcul (model de „aritmetică finită”).

Sistemul este:

$$\begin{cases} \pi x_1 + \sqrt{2}\, x_2 - x_3 + x_4 = 0 \\ e\, x_1 - x_2 + x_3 + 2x_4 = 1 \\ x_1 + x_2 - \sqrt{3}\, x_3 + x_4 = 2 \\ -x_1 - x_2 + x_3 - \sqrt{5}\, x_4 = 3 \end{cases}$$

Scriem matricea extinsă $[A \mid b]$:

$$\begin{bmatrix} \pi & \sqrt{2} & -1 & 1 & 0 \\ e & -1 & 1 & 2 & 1 \\ 1 & 1 & -\sqrt{3} & 1 & 2 \\ -1 & -1 & 1 & -\sqrt{5} & 3 \end{bmatrix}.$$

Notăm cu $r(\cdot)$ rotunjirea la **3 cifre semnificative**.

Valorile numerice de mai jos sunt cele obținute de implementarea din proiect, rulând: `tema2_gauss.exe --json --trace 4`.

Pasul $k = 0$ (eliminare pe coloana 1)

Pivotul rămâne pe linia 1 (nu se face swap).

După eliminarea elementelor de sub pivot (și rotunjiri la 3 cifre), obținem matricea extinsă:

$$\begin{bmatrix} \pi & \sqrt{2} & -1 & 1 & 0 \\ 0 & -2.22 & 1.86 & 1.14 & 1 \\ 0 & 0.55 & -1.41 & 0.682 & 2 \\ 0 & -0.55 & 0.682 & -1.92 & 3 \end{bmatrix}.$$

Pasul $k = 1$ (eliminare pe coloana 2)

Pivotul rămâne pe linia 2.

După eliminarea elementelor de sub pivot (și rotunjiri), obținem:

$$\begin{bmatrix} \pi & \sqrt{2} & -1 & 1 & 0 \\ 0 & -2.22 & 1.86 & 1.14 & 1 \\ 0 & 0 & -0.949 & 0.965 & 2.25 \\ 0 & 0 & 0.221 & -2.2 & 2.75 \end{bmatrix}.$$

Pasul $k = 2$ (eliminare pe coloana 3)

Pivotul rămâne pe linia 3.

După eliminarea elementului A_{43}, obținem o matrice superior triunghiulară:

$$\begin{bmatrix} \pi & \sqrt{2} & -1 & 1 & 0 \\ 0 & -2.22 & 1.86 & 1.14 & 1 \\ 0 & 0 & -0.949 & 0.965 & 2.25 \\ 0 & 0 & 0 & -1.98 & 3.27 \end{bmatrix}.$$

Substituție înapoi (back-substitution)

Din ultima ecuație:

$$-1.98\, x_4 = 3.27 \Rightarrow x_4 \approx -1.65.$$

Apoi:

$$-0.949\, x_3 + 0.965\, x_4 = 2.25 \Rightarrow x_3 \approx -4.05.$$

Din ecuația 2:

$$-2.22\, x_2 + 1.86\, x_3 + 1.14\, x_4 = 1 \Rightarrow x_2 \approx -4.68.$$

Din ecuația 1:

$$\pi x_1 + \sqrt{2}\, x_2 - x_3 + x_4 = 0 \Rightarrow x_1 \approx 1.34.$$

Rezultat

Soluția raportată de program (cu rotunjire la 3 cifre semnificative în pași) este:

$$x = [1.34,\ -4.68,\ -4.05,\ -1.65].$$