Mašinska reprezentacija brojeva i greške

Numerički algoritmi su skup tehnika pomoću kojih se matematički problemi formulišu kao problemi koji se mogu rešiti aritmetičkim i logičkim operacijama.

Numeričke metode koristimo kada nam je cilj da modelujemo realan svet na računaru. Tada moramo da koristimo matematički model. Relan svet je kompleksan pa je i model kompleksan. Kod kompleksnih modela (polinomi velikog stepena, ogromni sistemi jednačina, diferencijalne jednačine) obično je teško naći analitičko rešenje. Tada na scenu stupaju numeričke metode.

Matematički modeli nisu savršena replika realnog sveta (mnogi faktori se zanemaruju). Numeričke metode ne pružaju savršena rešenja. Računari na kojima se metode izvršavaju imaju ograničen kapacitet za reprezentaciju brojeva.

Kompjuterska reprezentacija brojeva

Decimalni sistem:

$$257.76 = 2\times10^{2} + 5\times10^{1} + 7\times10^{0} + 7\times10^{-1} + 6\times10^{-2}$$

Binarni sistem:

$$(1011.0011)_2 = \left((1\times2^{3} + 0\times2^{2} + 1\times2^{1} + 1\times2^{0}) + (0\times2^{-1} + 0\times2^{-2} + 1\times2^{-3} + 1\times2^{-4})\right)_{10} = 11.1875$$

Konverzija celog broja u binaran:

	Količnik	Ostatak
11/2	5	$1=a_0$
5/2	2	$1=a_1$
2/2	1	$0=a_2$
1/2	0	$1=a_3$

$$(11)_{10} = (a_3a_2a_1a_0)_2 = (1011)_2$$

Konverzija razlomačkog dela broja u binaran:

	Proizvod	Iza dec. t.	Ispred dec. t
0.1875*2	0.375	0.375	$0=a_{-1}$
0.375*2	0.75	0.75	$0=a_{-2}$
0.75*2	1.5	0.5	$1=a_{-3}$
0.5*2	1.0	0.0	$1=a_{-4}$

$$(0.1875)_{10} = (a_{-1}a_{-2}a_{-3}a_{-4})_2 = (0.0011)_2$$

Konverzija decimalnog broja u binaran:

Decimalan broj = ceo deo + razlomački deo

$$(11)_{10} = (1011)_2 \qquad (0.1875)_{10} = (0.0011)_2 \qquad \Rightarrow \qquad (11.1875)_{10} = (1011.0011)_2$$

Nije moguće tačno predstaviti svaki decimalan broj na računaru.

	Proizvod	Iza dec. t.	Ispred dec. t
0.3*2	0.6	0.6	$0=a_{-1}$
0.6*2	1.2	0.2	$1=a_{-2}$
0.2*2	0.4	0.4	$0=a_{-3}$
0.4*2	0.8	0.8	$0=a_{-4}$
0.8*2	1.6	0.6	$1=a_{-5}$
0.6*2	1.2	0.2	$1=a_{-6}$
.....	.....	vrtimo se u krug	staćemo kod petog člana

$$(0.3)_{10} \approx (a_{-1}a_{-2}a_{-3}a_{-4}a_{-5})_2 = (0.01001)_2 = 0.28125$$

Pokretni zarez

Savremeni računari realne brojeve reprezentuju u obliku pokretnog zareza. Opšti oblik broja u pokretnom zarezu je:

$$znak \times mantisa \times 10^{eksponent}$$

$$\sigma \times m \times 10^{e}$$

Na primer: -2.5678×10^{2} $\quad \sigma = -1$, $m = 2.5678$, $e = 2$

Kod normalizovanog oblika prva cifra mantise je uvek $\neq 0$.

$$\pm \underline{d.\ f_1\ f_2\ f_3\ f_4} \times 10^{\pm e}$$

znak mantisa eksponent

$$d \neq 0$$

Na primer: -2.5678×10^{2}

Kod normalizovanog binarnog oblika prva cifra mantise je uvek = 1.

$$\pm \underline{1.\ f_1\ f_2\ f_3\ f_4} \times 2^{\pm e}$$

znak mantisa eksponent

Prednost ovog oblika je što kada znamo da je prvi bit mantise 1, ne moramo da ga čuvamo na računaru. Dakle, dobijamo još jedan bit za decimalni deo mantise - povećavamo preciznost.

Na primer: $(54.75)_{10} = (110110.11)_2 = (1.1011011)_2 \times 2^5 \cong (1.1011)_2 \times (101)_2$

IEE 754 standard za reprezentaciju realnih brojeva

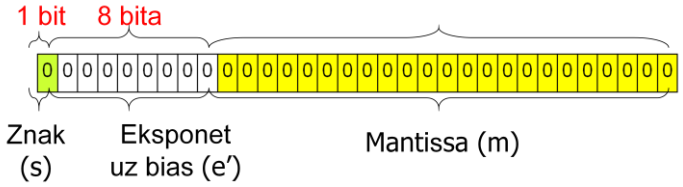
Savremeni računari koriste ovaj standard. On definiše nekoliko formata koji propisuju koliko bitova se koristi za reprezentaciju. Što više bitova, veća je preciznost. Ovaj standard je bio potreban da bi uskladio prozvodnju hardvera i softvera.

	Veličina u bitovima	Znak (0 = +, 1 = -)	Eksponent	Bias eksponenta	Mantisa
jednostruka preciznost (float)	32 bita	1 bit	8 bita (-126 do +127)	127	23 bita
dvostruka preciznost (double)	64 bita	1 bit	11 bita (-1022 do +1023)	1023	52 bita

Veći eksponent - veći raspon brojeva

Veća mantisa - veća preciznost

Jednostruka preciznost (float):



$$Vrednost = (-1)^{s} \times (1.m)_2 \times 2^{e'-127}$$

Nemamo bit za znak eksponenta.

8 bita za ekponent nam daje raspon $0 \le e' \le 255$. Oduzmemo 127 i imamo $-127 \le e \le 128$.
$e' = 0$ i $e' = 255$ su rezervisani za specijalne vrednosti.
$e' = 0$ - sve 0 binarno
$e' = 255$ - sve 1 binarno

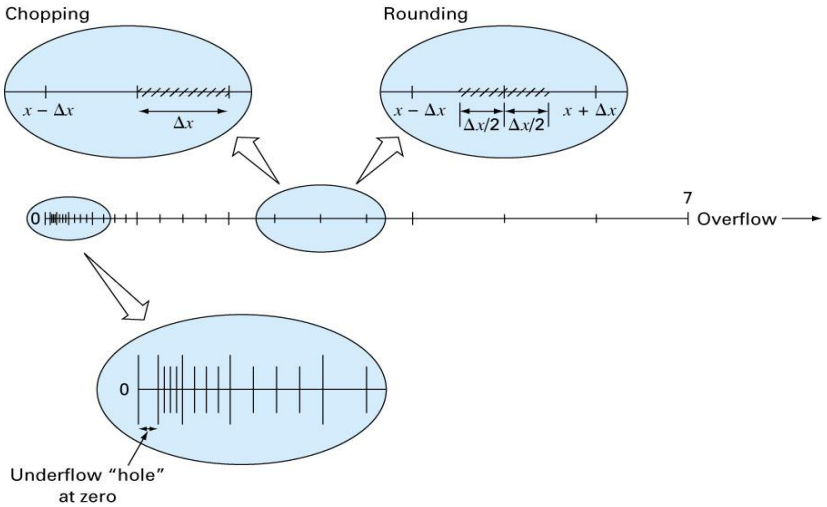
znak		m	Specijalna vrednost
0	sve nule	sve nule	0
1	sve nule	sve nule	-0
0	sve jedinice	sve nule	∞
1	sve jedinice	sve nule	$-\infty$
0 ili 1	sve jedinice	bilo šta $\neq 0$	NaN

Zaokruživanje

Ako imamo broj koji ima više cifara nego što računar može da podrži, moramo da ga zaokružimo.

Postoji dve vrste zaokruživanja:

- odsecanje (*choping*) - postupak kojim se odsecaju cifre od neke cifre nadesno, bez korekcije vrednosti preostalog broja.
- zaokruživanje na najbliži (*rounding*) - gledamo poslednju cifru d. Ako je d > 5 zaokružimo na najbliži veći mašinski broj, ako je d < 5 zaokružimo na najbliži manji mašinski broj, a ako je d = 5 IEE 745 koristi zaokruživanje na najbliži parni.



Najveća greška za *choping* je razmak između dva binarna broja. Npr. 1.999.. zaokružimo na 1.
Najveća greška za *rounding* je pola razmaka između dva binarna broja. Npr. 1.49 zaokružimo na 1.
Razmak između brojeva nije ravnomeran. Isti je samo za isti stepen eskponent.

Mašinska tačnost

Mašinska tačnost je razlika između 1.0 i najbližeg broja koji može da se reprezentuje u brojevnom sistemu.

Moguće je reprezentovati samo ograničen raspon brojeva.
Preveliki broj - owerflow, premali broj - underflow

Moguće je reprezentovati samo ograničenu količinu brojeva u rasponu.

Značajne cifre

Značajne cifre su one koje mogu da se koriste sa pouzdanošću. Saopštava ih onaj koji vrši merenje.

Pravila za prepoznavanje značajnih cifara:

1. Sve cifre različite od 0 su značajne:
   - 1.234 ima četiri značajne cifre, 1.2 ima dve.
2. Nule između cifara različitih od 0 su značajne:
   - 1002 ima četiri značajne cifre, 3.07 ima tri.
3. Nule pre prve cifre različite od 0 nisu značajne (služe samo samo kao indikator položaja decimalne tačke):
   - 0.001 ima samo jednu značajnu cifru, 0.012 ima dve.
4. Sve nule posle poslednje cifre različite od 0 su značajne
   - 0.0230 ima tri značajne cifre, 0.20 ima dve.
5. Ako broj nije decimalan, a ima nule na kraju za njih se ne može tvrditi da li su značajne ili ne:
   - 150 može imati dve ili tri značajne cifre.
   - 60500 može imati tri, četiri ili pet značajnih cifara.

- Da bi se izbegli ovakvi slučajevi preporučuje se upotreba tzv. naučne notacije (*scientifc notation*):
  - $5.06 * 10^4$ (3 značajne cifre)
    $5.060 * 10^4$ (4 značajne cifara),
    $5.0600 * 10^4$ (5 značajnih cifara).
  - (primetite da smo do sada na predavanju na taj način reprezentovali *floating point* brojeve...)

Greške

Kada primenjujemo numeričke metode imamo posla sa dva tipa grešaka:

- ne-numeričke
- numeričke

Ne-numeričke greške:

- greške u modelovanju – zanemarujemo neke faktore iz realnog sveta koji nam nisu od značaja (ako u kompjuterskoj igri simulirate sudar dva automobila vrlo su male šanse da ćete uzeti sve moguće faktore fizike iz realnog sveta, a da to radi na "običnom" računaru...)
- ljudske greške (tehnika radi dobro ali onaj koji pritiska dugmiće baš i ne...)
- nepreciznost u informacijama i merenjima (greške mernih instrumenata...)

Numeričke greške:

1. greške zaokružvianja (*rounding error*)
   - zbog ograničenog broja značajnih cifara (ograničenog prostora za reprezentaciju na računaru).
2. greške zanemarivanja (*truncation error*)
   - nastaje kada zanemarimo (ne koristimo) delove matematičkih izraza (npr. Tejlorov red) ili kad zaustavimo numerički algoritam posle određenog broja koraka. – više u nastavku predavanja.
3. greške koja su posledica nagomilavanja prethodnih grešaka (*propagation errors*):
   - nastaju kad imamo niz računskih operacija kod kojih se javljaju greške tipa 1. ili 2. Greške se onda prostiru (gomilaju) od početka do kraja računanja. – više na sledećem predavanju.
4. greške matematičke aproksimacije (*mathematical-approximation errors*)
   - nastaju zbog pojednostavljivanja modela npr. aproksimiramo komplikovanu funkciju polinomom.

Greške merimo da odredimo tačnost numeričkih metoda i da odredimo kada ćemo da zaustavimo numerički algoritam.

Prava greška

Prava greška je razlika između tačnog (pravog) rešenja i rezultata numeričke metode:

$$E_T = x_T - x_A$$

Obično se koristi apsolutna vrednost i onda imamo apsolutnu pravu grešku:

$$E_T = |x_T - x_A|$$

Prava greška se može koristiti samo kada znamo tačno rešenje, što je retko, jer što bismo onda uopšte koristili numeričke metode.

Relativna prava greška

Problem sa pravom greškom je što ne možemo uvek da kažemo da li je vrednost koju smo dobili mala ili velika. Bolje bi bilo kada bismo imali grešku u procentima.

Relativna prava greška se definiše na sledeći način:

$$E_R = \frac{|x_T - x_A|}{x_T}$$

Približna greška

Približna greška je razlika trenutne i prethodne procene. Obično se koristi apsolutna vrednost, tj. apsolutna približna greška:

$$E_A = |x_i - x_{i-1}|$$

x_i - trenutna procena rešenja (i -ta iteracija)
x_{i-1} - prethodna procena rešenja (i-1 -a iteracija)

Približna greška se često koristi kao kriterijum za zaustavljanje numeričkih algoritama. Zaustavljamo algoritam ako razlika prethodnog i trenutnog rešenja padne ispod neke tolerancije npr. 10^{-5}.

$$E_A = |x_i - x_{i-1}| < tolerancija$$

Toleracijom možemo da zadamo posle koje decimale ne želimo više da poboljšavamo rešenje.

Relativna približna greška

Relativnu približnu grešku koristimo da možemo da procenimo koliko je velika približna greška.

$$E_{RA} = \frac{|x_i - x_{i-1}|}{x_i}$$

Greška odsecanja

Greška odsecanja je greška koja nastaje pojednostavljivanjem matematičkog izraza.

Ako koristimo prva tri člana Maklorenovog reda da izračunamo e^x:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + ...$$

greška odsecanja je

$$Truncation\ \ Error = e^x - \left(1 + x + \frac{x^2}{2!}\right)$$

Gausova eliminacija

Gausova eliminacija je direktan metod. Ima dva koraka:

1. eliminacija unapred - kolonu po kolonu eliminišemo elemente ispod glavne dijagonale. Rezultat je gornja trougaona matrica
2. zamena unazad - izračunavamo x_n direktno. Zamenom unazad računamo ostale promenljive, od x_{n-1} do x_1.

Prvo eliminišemo x_1 iz svih jednačina osim prve. Pomnožimo prvu jednačinu sa -a_{21}/a_{11} i saberemo je sa drugom. Sličan postupak ponavljamo za ostale jednačine, dok se ne dobije:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &= b_1 \\ a'_{22}x_2 + \ldots + a'_{2n}x_n &= b'_2 \\ &\vdots \\ a'_{n2}x_2 + \ldots + a'_{nn}x_n &= b'_n \end{aligned}$$

Postupak koji smo koirstili za promenljive x_1 i x_2 koristimo i za preostale promenljive (osim x_n), čime dobijamo trougaoni oblik:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n &= b_n \end{aligned}$$

Zatim treba da eliminišemo x_2 iz svih jednačina osim prve i druge. Množimo drugu jednačinu sa -a'_{32}/a'_{22} i saberemo je sa trećom. Na sličan način eliminišemo iz preostalih jednačina.

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n &= b_1 \\ a'_{22}x_2 + a'_{23}x_3 + \ldots + a'_{2n}x_n &= b'_2 \\ a''_{33}x_3 + \ldots + a''_{3n}x_n &= b''_3 \\ &\vdots \\ a_{nn}^{(n-1)}x_n &= b_n^{(n-1)} \end{aligned}$$

Sada zamenom unazad dobijamo rešenje (x_1,x_2,...,x_n).

x_n dobijamo običnim deljenjem: $x_n = \dfrac{b_n^{(n-1)}}{a_{nn}^{(n-1)}}$

Zamenimo x_n u (n-1)-u jednačinu i izračunamo x_{n-1}: $a_{n-1,n-1}^{(n-2)}x_{n-1} + a_{n-1,n}^{(n-2)}x_n = b_{n-1}^{(n-2)}$

Ponovimo proces kako bismo dobili preostale promenljive.

Opšta formula za zamenu unazad: $x_i = \dfrac{b_i^{(i-1)} - \sum_{j=i+1}^{n} a_{ij}^{(i-1)}x_j}{a_{ii}^{(i-1)}} \qquad a_{ii}^{(i-1)} \neq 0$

<u>Algoritam</u> za Gausovu elminiaciju:

1. Elminacija u napred

   za svaku jednačinu k, k = 1 do n-1
   za svaku jednačinu i, i = (k+1) do n
   (a) pomnoži jednačinu k sa -a_{ik}/a_{kk}
   (b) saberi jednačinu k sa jednačinom i

   Ovde dobijamo gornju trougaonu matricu
2. Zamena u nazad
   (a) izračunaj x_n kao b_{nn}/a_{nn}
   (b) zameni x_n u (n-1)-u jednačinu, izračunaj x_{n-1}
   (c) ponavljaj (b), za n-2, n-3, itd. dok sve nepoznate nisu određene.

Problemi sa Gausovom eliminacijom

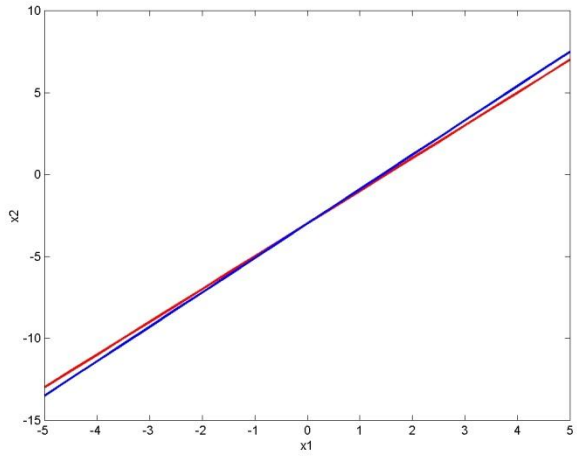
- deljenje nulom
- deljenje malim brojevima (greške u zaokruživanju)
- loše uslovljeni sistemi

Greške u zaokruživanju

Ima jako mnogo zaokruživanja u $n^3/3$ operacija. Još važnije – greška se propagira (nagomilava). Za velike sisteme (sa više od 100 jednačina), greške u zaokruživanju su veoma značajne.

Greške u zaokruživanju su naročito ozbiljne kod loše uslovljenih sistema.

Loše uslovljeni sistemi su sistemi kod kojih male promene u koeficijentima dovode do velikih promena u rešenju.



Parcijalni pivoting

Jednačina, koju koristimo da elimišemo promenljivu, naziva se pivot jednačina. Koeficijent u pivot jednačini koji je uz promenljivu koju eliminšemo naziva se pivot element. Pivot elementom delimo koeficijente u procesu eliminacije. Iz tog razloga on ne bi trebalo da bude nula, a ni jako mali broj. Deljenje malim brojevima unosi velike greške u postupak zbog ograničenosti računara za njihovu reprezentaciju.

Parcijalni pivoting predstavlja zamenu pivot jednačine sa jednačinom koja sadrži maksimalni element po apsolutnoj vrednosti u koloni u kojoj se pivot nalazi. Za zamenu se koriste samo jednačine koje su ispod pivot jednačine. Na taj način deljenje vršimo sa što je moguće većim brojem. .

Ako pivot tražimo u celoj matrici, a ne u trenutnoj koloni onda menjamo i vrstu i kolonu i imamo kompletan pivoting. Kompletan pivoting se retko koristi jer je zahtevan, a ne pomaže mnogo.

Algoritam za eliminaciju unapred sa parcijalnim pivotingom:

za svaku jednačinu k, $k = 1$ do n-1
    pretražiti sve jednačine $i \geq k$ za maksimalan element po apsolutnoj vrednosti
    zameniti jedančinu k sa tom koja ima max.
    Izvršiti eliminaciju
        (a) pomnožiti jednačinu k sa $-a_{ik}/a_{kk}$
        (b) sabrati sa jednačinom i

Procena greške Gausove eliminacije

Intuitivno, ako rešimo sistem $Ax=b$, da bi izračunali grešku trebalo bi samo da zamenimo x u Ax i uporedimo sa b:

$r = Ax-b$ - ostatak ili rezidual.

Prilikom računanja r, može se javiti problem deljenja jako malih brojeva.

Problem deljenja jako malih brojeva na računaru zove se potiranje (*cancellation*) i treba ga izbegavati u numerici, ako je to moguće.

Ne moramo znati tačnu grešku ako smo uspeli da je procenimo na malu vrednost tj. nije nam važno koliko je tačno greška mala, već da je mala. Za procenu nam je od velikog značaja kondicioni broj matrice.

Kondicioni broj funkcije $f(x)$ meri koliko promene ulaza x utiču na promene izlaza y. Veliki kondicioni broj znači da male promene ulaza daju velike na promene izlaza. Tada kažemo da je funkcija loše uslovljena.

Kondicioni broj nesingularne kvadratne matrice: $cond(A) = \|A\| \cdot \|A^{-1}\|$

Po konvenciji za singularnu matricu: $cond(A) = \infty$

Veliki kondicioni broj znači da je matrica skoro singularna, a to je loše za rešavanje sistema.

Iterativne metode

Kod iterativnih metoda krećemo od odabrane početne vrednosti x^0. Bira je korisnik metode (na slučajan ili neki drugi način). Koristimo iterativnu formulu koja daje vezu između x_k i x_{k-1}. Na taj način izračunavamo niz $x^0,x^1,x^2,....,x^{k-1},x^k,.....$

Niz $x^0,x^1,x^2,....,x^{k-1},x^k,.....$ konvergira ka tačnom rešenju *x*, za beskonačno mnogo iteracija. U praksi nam ne treba ∞ iteracija. Koristimo apsolutnu približnu grešku da zaustavimo iterativni metod:

$$|x_k - x_{k-1}| < tolerancija$$

Tolerancijom kontrolišemo odnos brzine i tačnosti.

Sistem Ax=b transformišemo u oblik x=Tx+c. Uzimamo početno rešenje x^0 i pomoću iterativne formule $x^k=Tx^{k-1}+c$, kreiramo niz $x^0,x^1,x^2,....,x^{k-1},x^k,.....$

Matricu T i vektor c određujemo pomoću A i b. U zavisnosti od toga kako ih određujemo imamo dva poznata iterativna metoda:

- Jakobijev (Jacobi)
- Gaus-Zajdelov (Gauss-Seidel)

Rastavljanje na T i c (za sistem 3x3):

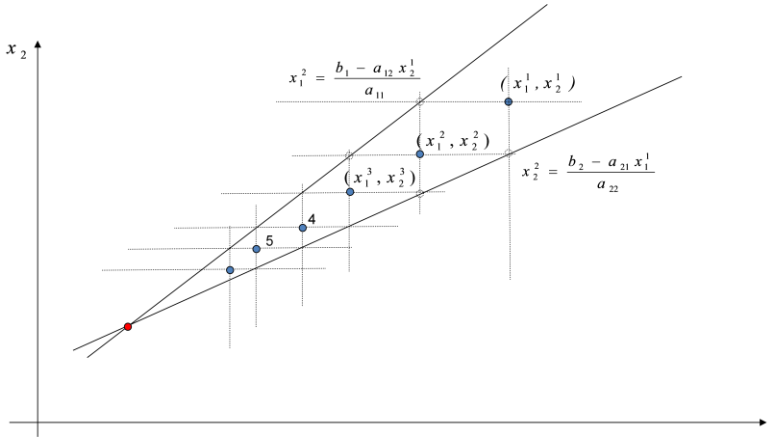
$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &= b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 &= b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 &= b_3 \end{aligned}$$

$$\begin{aligned} x_1 &= \underbrace{-\frac{a_{12}}{a_{11}}x_2 - \frac{a_{13}}{a_{11}}x_3}_{T} + \underbrace{\frac{b_1}{a_{11}}}_{c} \\ x_2 &= -\frac{a_{21}}{a_{22}}x_1 - \frac{a_{23}}{a_{22}}x_3 + \frac{b_2}{a_{22}} \\ x_3 &= -\frac{a_{31}}{a_{33}}x_1 - \frac{a_{32}}{a_{33}}x_2 + \frac{b_3}{a_{33}} \end{aligned}$$

Jakobijev metod

Jakobijev metod je iterativni metod za rešavanje SLAJ. Njegova mana je što za izračunavnje trenutne komponete rešenja x_i^{k+1} ne koristimo najnovije informacije tj. sve do tada izračunate x_j^{k+1}, gde je *j*=1,...*i*-1. Formula za određivanje komponenti rešenja:

$$x_i^{k+1} = \frac{1}{a_{ii}}\left[b_i - \sum_{j=1}^{i-1} a_{ij}x_j^k - \sum_{j=i+1}^{n} a_{ij}x_j^k\right] \qquad \text{tj.} \qquad x_i^{novo} = \frac{b_i - (\sum_{j=1}^{i-1} a_{ij}x_j^{staro} + \sum_{j=i+1}^{n} a_{ij}x_j^{staro})}{a_{ii}}$$

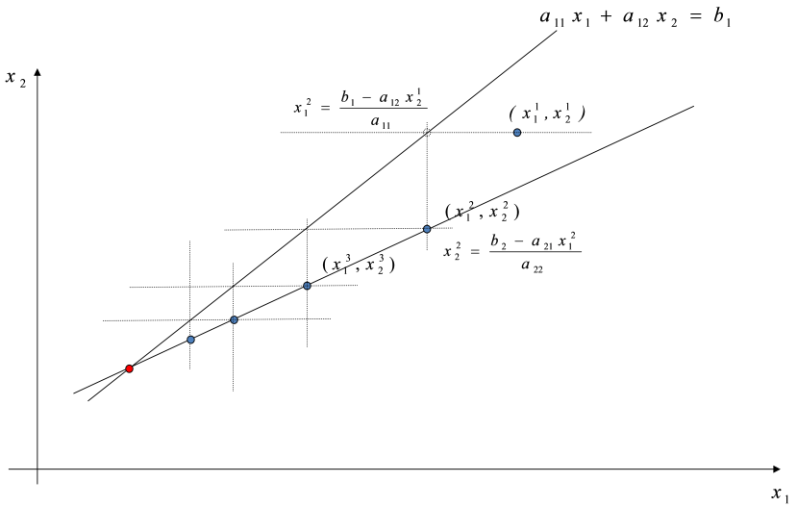


Geometrijska interpretacija

Gaus-Zajdelov metod

Gauss-Seidelov metod je iterativni metod za rešavanje SLAJ. Ideja metoda je da za izračunavnje trenutne komponete rešenja x_i^{k+1} koristimo najnovije informacije tj. sve do tada izračunate x_j^{k+1}, gde je j=1,...i-1. Formula za određivanje komponenti rešenja:

$$x_i^{k+1} = \frac{1}{a_{ii}}\left[b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij} x_j^k \right] \quad \text{tj.} \quad x_i^{novo} = \frac{b_i - (\sum_{j=1}^{i-1} a_{ij} x_j^{novo} + \sum_{j=i+1}^{n} a_{ij} x_j^{staro})}{a_{ii}}$$



Geometrijska interpretacija

SOR metoda

Cilj SOR postupka je ubrzanje konvergencije Gaus-Seidelov-og postupka. Ideja je da pomeranje od x^k do x^{k+1} možemo ubrzati množenjem konstantom ω>1.

$$x_i^{k+1} = x_i^k + \frac{1}{a_{ii}}\left[b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{k+1} - \sum_{j=i}^{n} a_{ij} x_j^k \right]$$

$$x_i^{k+1} = x_i^k + \delta_i^k$$

$$x_i^{k+1} = x_i^k + \omega\delta_i^k$$

$$x_i^{k+1} = x_i^k + \omega\frac{1}{a_{ii}}\left[b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{k+1} - \sum_{j=i}^{n} a_{ij} x_j^k \right]$$

$$x_i^{k+1} = (1-\omega)x_i^k + \omega\frac{1}{a_{ii}}\left[b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij} x_j^k \right]$$

1<ω<2 *over relaxation* (brža konvergencija), 0<ω<1 *under relaxation* (sporija konvergencija). Optimalna vrednost za ω se pronalazi ručno iz više pokušaja i grešaka (obično je oko 1.6). Ako ne važi 1<ω<2 SOR metoda divergira.

Konvergencija Jakobijevog metoda

Ako je matrica *A* dijagonalno dominantna, Jakobijev metod konvergira za bilo koje početno rešenje. Matrica A je dijagonalno dominantna ako je: $|a_{ii}| > \sum_{\substack{j=1 \\ i \neq j}}^{n} |a_{ij}|$,

tj. ako je svaki element glavne dijagonale veći od zbira ostalih elemenata u istoj vrsti.

Konvergencija Gaus-Zajdelovog metoda

Gaus-Zajdelov metod konvergira za bilo koje početno rešenje ako je matrica A dijagonalno dominantna ili ako je matrica A simetrična i pozitivno definitna. Matrica A je pozitivno definitna ako važi $x^T Ax>0$ za svaki vektor $x \neq 0$, tj. ako su svi elementi na glavnoj dijagonali pozitivni i ako se element koji ima najveću vrednost u celoj matrici nalazi na glavnoj dijagonali.

Performanse iterativnih metoda

Podsetimo se da je broj operacija za Gaussovu eliminaciju $O(n^3)$ tj. reda n^3. Za iterativne metode broj množenja u svakoj iteraciji je $O(n^2)$. Ako je broj iteracija potrebnih za konvergenciju mnogo manji od *n*, tada su iterativne metode efikasnije od direktnih.

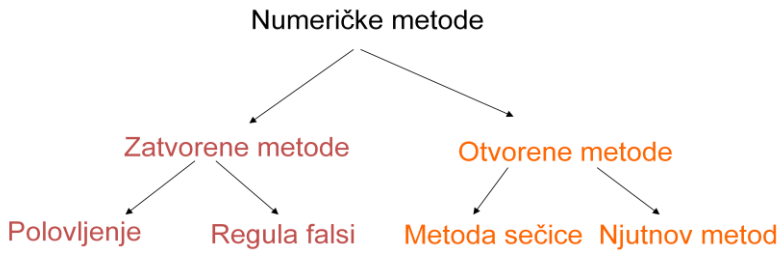
Iterativne metode su takođe efikasne kad je A matrica koja sadrži puno nula elemenata (što je čest slučaj u praksi – npr. PageRank, sistemi za preporuku itd).

Rešavanje nelinearnih jednačina

Formulacija opšteg iterativnog postupka za određivanje nula funkcije:

Data nam je funkcija f(x). Cilj nam je da odredimo tačku x, za koju važi f(x)=0. Za određivanje tačke x, koristimo iterativni postupak koji se sastoji od iterativne formule koja daje način za određivanje trenutne procene rešenja x^k pomoću prethodne procene x^{k-1}. Krećemo od početne procene rešenja x^0 (koja je data ili je biramo) i pomoću iterativne formule kreiramo niz procena $x^0,x^1,x^2,....,x^{k-1},x^k,....$ Iterativni prostupak možemo zaustaviti posle zadatog broja iteracija ili ako razlika između trenutne i prethodne procene padne ispod zadate tačnosti.

Numeričke metode za rešavanje nelinearnih jednačina



Zatvorene metode

Kod zatvorenih metoda algoritam kreće od zatvorenog intervala koji sadrži rešenje. Svakim sledećim korakom taj interval se smanjuje:

- dok se interval ne svede na jednu tačku tj. rešenje ili
- dok veličina intervala ne padne ispod zadate tolerancije (kriterijum koji se koristi u praksi).

Otvorene metode

Kod otvorenih metoda algoritam kreće od početnog (inicijalnog) rešenja x^0. Svakim sledećim korakom dobija se nova procena rešenja $x^0,x^1,x^2,....,x^{k-1},x^k,.....$ Algoritam se zaustavlja kad:

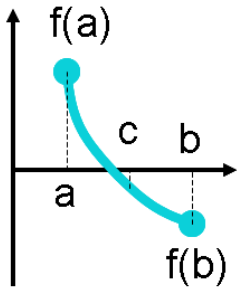
- pronađemo rešenje (tačku za koju važi $f(x)=0$) ili
- $|x_k - x_{k-1}| < tolerancija$ ili
- $|f(x_k)| < tolerancija$

Metoda polovljenja

Metoda polovljenja je iterativna metoda za određivanje nula funkcije f(x). Kao uslov za korišćenje zahteva zatvoreni interval za koji je poznato da sadrži rešenje. Metoda polovljenja sistematski smanjuje (polovi) zatvoreni interval. Pre svakog polovljenja izvršava se jednostavna provera na osnovu koje se donosi odluka koja polovina se dalje polovi. Polovljenje prestaje kad je pronađeno tačno rešenje ili je trenutni interval dovoljno mali.

Pretpostavke:

- f(x) je neprekidna na [a,b]
- f(a) f(b) < 0



Algoritam:
Loop

1. Izračunati polovinu [a,b] c = (a+b)/2
2. Izračunati f(c)
3. Ako f(a) f(c) < 0 novi interval je [a, c]
   Ako f(a) f(c) > 0 novi interval je [c, b]
4. Ako $|b-a| < tolerancija$ vrati c = (a+b)/2 kao rešenje

End loop

Metoda sečice

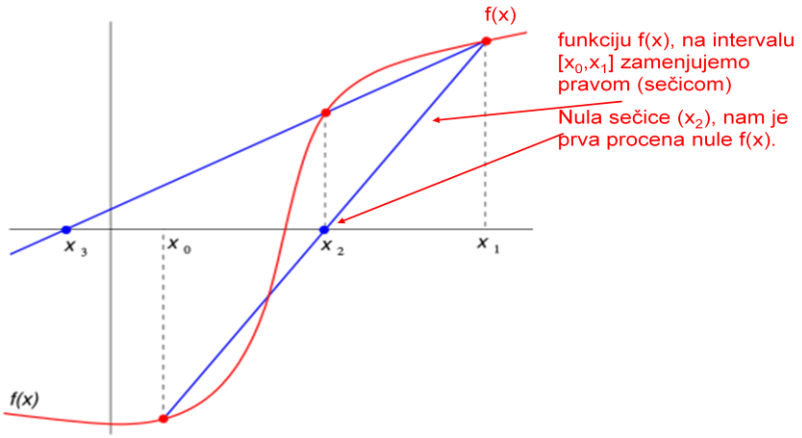
Metoda sečice je iterativna metoda za određivanje nula funkcije f(x). Ideja je da prilikom traženja nule, f(x) zamenimo pravom na malom intervalu (linearna interpolacija). Umesto da tražimo nulu f(x) koja može biti komplikovanog oblika, tražimo nulu prave koja je jednostavna.

Pretpostavka:

- Dve početne tačke x_i i x_{i-1} takve da važi $f(x_i) \neq f(x_{i-1})$

Sledeća procena, tačka x_{i+1} dobija se pomoću formule:

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



Metoda regula falsi

Metoda regula falsi je iterativna metoda za određivanje nula funkcije f(x). Ona uvodi u metodu sečice zahtev da se rešenje nalazi u zatvorenom intervalu, tj. da na početku važi f(a) f(b) < 0. TIme dobijamo metodu koja malo sporije, ali garantovano konvergira.

Pretpostavka:

- Dve početne tačke x_i i x_{i-1} takve da važi $f(x_i) \neq f(x_{i-1})$ i $f(x_i)\, f(x_{i-1}) < 0$.

Sledeća procena, tačka x_{i+1} dobija se pomoću formule:

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

Metoda tangente (Njutnova metoda)

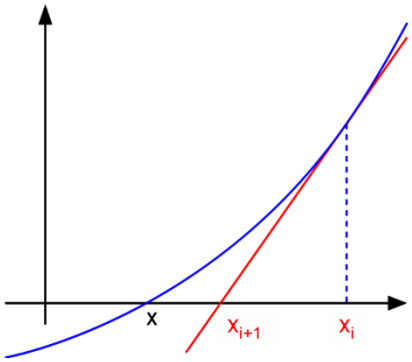
Metoda tangente je iterativna metoda za određivanje nula funkcije f(x). Ideja metode je da se funkcija između svake dve procene rešenja aproksimira tangentom. Tačka u kojoj tangenta seče x-osu je sledeća procena rešenja.

Pretpostavke:

- f(x) je neprekidna i ima prvi izvod
- x_0 je početna tačka takva da je $f'(x_0) \neq 0$

Formula za računanje sledeće tačke:

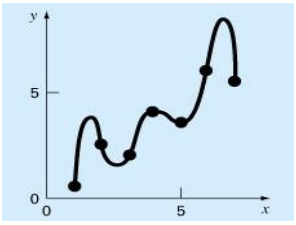
$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

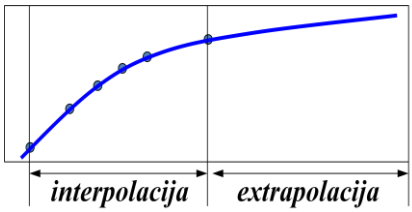


Metod	Prednosti	Mane
Polovljenje	- Garantovana konvergencija - Broj iteracija za toleranciju se može unapred odrediti - Laka za implementaciju - Ne zahteva prvi izvod	- Spora konvergencija - Zahteva da se rešenje nalazi u [a,b] tj. da važi f(a)f(b)<0
Sečica	- Brza konvergencija (sporija od Njutnove metode) - Ne zahteva prvi izvod	- Nema garantovanu konvergenciju - Zahteva dve početne tačke takve da važi f(x0)≠f(x1)
Regula falsi	- Garantovana konvergencija (brža od polovljenja, sporija od sečice) Ne zahteva prvi izvod	- Zahteva dve početne tačke takve da važi f(x0)≠f(x1)
Njutnov	- Veoma brza konvergencija (ako je početno rešenje blizu nule)	- Nema garantovanu konvergenciju - Zahteva postojanje prvog izvoda i f'(x0)≠0

Interpolacija

Interpolacija je metod za određivanje nepoznatih vrednosti koje se nalaze između vrednosti koje su nam poznate. Poznat nam je skup podataka (x_1,y_1), (x_2,y_2), (x_3,y_3),..., (x_n,y_n). Želimo da odredimo y za x koje nije ni jedno od x_i. Da bi procenili y za novo x, vršimo interpolaciju poznatih x, tako što kreiramo funkciju koja mora da prođe kroz svaku od tačaka (x_i,y_i). Najuobičajeniji tip je interpolacija polinomom.





Interpolacija: podaci koje želimo da odredimo nalaze se u rasponu podataka koje imamo.

Ekstrapolacija: podaci koje želimo da odredimo se ne nalaze u rasponu podataka koje imamo. (nije pouzdano)

Interpolacija polinomom

Dat nam je niz parova tačaka (x_i, y_i) koji predstavljaju vrednosti funkcije koju aproksimiramo. Ideja postupka interpolacije je određivanje polinoma takvog da prolazi kroz sve date tačke tj. mora da važi $p(x_i)=y_i$, za sve date (x_i, y_i). p(x) je onda interpolacioni polinom.

Njutnova interpolacija

Njutnove "podeljene (konačne) razlike" se koriste da bi se odredili koeficijenti polinoma koji ima oblik: $f_{n-1}(x) = b_1 + b_2(x-x_1) + b_3(x-x_1)(x-x_2) + \cdots + b_n(x-x_1)(x-x_2)\cdots(x-x_{n-1})$

Koeficijenti nižeg reda (stepena) ostaju isti kad se poveća stepen polinoma. Dakle, lako je dodati nove podatke (tačke) i uraditi interpolaciju polinomom većeg stepena.

Opšta formula za Njutnov polinom:

$$f_{n-1}(x) = b_1 + b_2(x-x_1) + b_3(x-x_1)(x-x_2) + \cdots + b_n(x-x_1)\cdots(x-x_{n-1})$$

$$b_1 = f(x_1) = f[x_1]$$

$$b_2 = f[x_2,\ x_1] = \frac{f[x_2] - f[x_1]}{x_2 - x_1}$$

$$b_3 = f[x_3, x_2,\ x_1] = \frac{f[x_3,\ x_2] - f[x_2,\ x_1]}{x_3 - x_1}$$

$$\vdots$$

$$b_n = f[x_n, x_{n-1}, \cdots, x_2, x_1] = \frac{f[x_n, x_{n-1}, \ldots, x_3, x_2] - f[x_{n-1}, x_{n-2}, \ldots, x_2, x_1]}{x_n - x_1}$$

$f[x_i] = f(x_i)$ konačna razlika nultog reda

$$f\left[x_i, x_j\right] = \frac{f\left(x_i\right) - f\left(x_j\right)}{x_i - x_j}$$ konačna razlika prvog reda

$$f\left[x_i, x_j, x_k\right] = \frac{f\left[x_i, x_j\right] - f\left[x_j, x_k\right]}{x_i - x_k}$$ konačna razlika drugog reda

$$f\left[x_n, x_{n-1}, \ldots, x_2, x_1\right] = \frac{f\left[x_n, x_{n-1}, \ldots, x_3, x_2\right] - f\left[x_{n-1}, x_{n-2}, \ldots, x_2, x_1\right]}{x_n - x_1}$$ konačna razlika n-tog reda

Iterativni algoritam:

1. Izračunati sve konačne razlike prvog reda pomoću vrednosti funkcije $f(x_i)$.
2. Izračunati sve konačne razlike drugog reda koristeći razlike prvog reda.
3. Nastaviti proces do razlika n-tog reda.

Prednost Njutnovog polinoma je u tome što pri dodavanju novih tačaka ne moramo ponovo da izračunavamo sve koefcijente. Moramo samo da sračunamo onaj uz najveći stepen (jer je sad stepen za jedan veći, zato što smo dodali novu tačku).

Lagranžova interpolacija

Podsetimo se da za n tačaka postoji jednistveni polinon stepena n-1 koji prolazi kroz njih. Tako da faktički govorimo o Lagranžovom i Njutnovom obliku istog polinoma.

Lagranžov polinom ima sledeći oblik:

$$f_{n-1}(x) = L_1(x) f(x_1) + L_2(x) f(x_2) + \cdots + L_n(x) f(x_n) = \sum_{i=1}^{n} L_i(x) f(x_i)$$

$$L_i(x) = \prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j} = \frac{P_i(x)}{P_i(x_i)} = \frac{(x - x_1)(x - x_2) \cdots\cdots (x - x_{i-1})(x - x_{i+1}) \cdots\cdots (x - x_n)}{(x_i - x_1)(x_i - x_2) \cdots (x_i - x_{i-1})(x_i - x_{i+1}) \cdots (x_k - x_n)}$$

$$\left.\begin{array}{ll} j = i\ ; & L_i(x_i) = \dfrac{P_i(x_i)}{P_i(x_i)} = 1 \\ j \neq i\ ; & L_i(x_j) = 0 \end{array}\right\} \Rightarrow L_i(x_j) = \delta_{ij}$$

Lagranžov polinom prvog reda:

$$f_1(x) = L_1 f(x_1) + L_2(x) f(x_2) = \frac{x - x_2}{x_1 - x_2} f(x_1) + \frac{x - x_1}{x_2 - x_1} f(x_2)$$

Lagranžov polinom drugog reda:

$$f_2(x) = \frac{(x - x_2)(x - x_3)}{(x_1 - x_2)(x_1 - x_3)} f(x_1) + \frac{(x - x_1)(x - x_3)}{(x_2 - x_1)(x_2 - x_3)} f(x_2) + \frac{(x - x_1)(x - x_2)}{(x_3 - x_1)(x_3 - x_2)} f(x_3)$$

Lagranžov polinom je pogodan za slučajeve kada imamo više različitih skupova y_i za iste x_i (npr. veličina koja se meri uvek u istim trenutcim x_i), jer $L_k(x)$ zavise samo od x, treba da se izračunaju samo jednom. Nije toliko pogodan za dodavanje novih tačaka kao Njutnov polinom.

Inverzna interpolacija

Do sada, za date *x* i *f(x)* interpolacija nam daje mogućnost da izračunamo *f(x)* za novo *x*. Šta bi bilo ako želimo da odredimo u kojoj tački x funkcija f(x) ima određenu vrednost?

1. Zameniti *x* i *f*(*x*) i uraditi interpolaciju. Međutim, razmak između y je obično veoma ne-uniforman (za razliku od x) što rezultuje u oscilacijama u interpolacionom polinomu.
2. Interpolirati f(x) u tačkama x i upotrebiti metode za određivanje nula funkcija da bi pronašli x za dato f(x).

Interpolacija Splajnom (deo-po-deo)

Ideja deo-po-deo interpolacije je određivanje zasebnih interpolacionih polinoma za svaki interval $[x_i,x_{i+1}]$. Takvi polinomi su nižeg stepena u odnosu na jedan polinom koji prolazi kroz sve tačke. Razlozi za deo-po-deo interpolaciju su to što: za veliki broj datih tačaka interpolacioni polinom koji prolazi kroz sve tačke ima jako veliki stepen; takav polinom ima velike oscilacije u prostoru u kome radimo; osteljiv je na šum u podacima; operacije kao što su evaluacija u tački, određivanje izvoda itd. su računski zahtevne. Deo-po-deo interpolacioni polinomi prevazilaze sve navedene nedostatke.

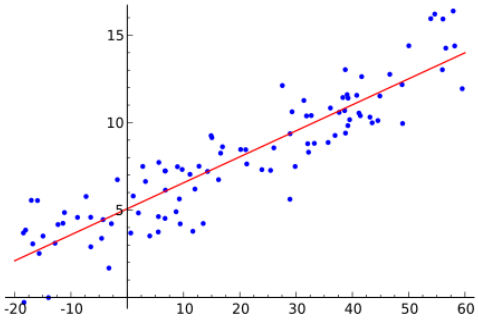
Aproksimacija

Regresija

Regresija pruža alternativni način za rešavanje problema sa velikim skupovima tačaka. Regresija ima za cilj da pronađe funkciju koja se "najbolje uklapa" u podatke uz sledeće dve pretpostavke:

- rezultujuća funkcija ne mora da prođe kroz svaku tačku.
- ako koristimo polinom, njegov stepen ne zavisi od broja tačaka, već ga zadaje korisnik.

Pomoću regresije pokušavamo da pronađemo trend u podacima (ako je prisutan) i da taj trend opišemo pomoću funkcije. Na slici je dat primer linearnog trenda



Opšti problem:

Dat nam je skup podataka (x_1,y_1), (x_2,y_2), (x_3,y_3),..., (x_n,y_n). Cilj je pronaći funkciju koja se "najbolje uklapa" u podatke. Konkretno, ako imamo:

- skup podataka
- oblik funkcije
- kriterijum za meru "najboljeg uklapanja",

potrebno je da odredimo nepoznate parametre funkcije.

Postoji više mogućnosti za oblik funkcije koju koristimo:

Linearni (prava) $f(x) = a + bx$

Kvadratni polinom $f(x) = a + bx + cx^2$

Opsti polinom $f(x) = \sum_{k=0}^{n} a_k \; x^k$

Opsti oblik $f(x) = \sum_{k=0}^{m} a_k \; g_k(x)$

$g_k(x)$ *su proizvoljne funkcije date unapred*

Dva načina za definisanje kriterijuma "najbolje uklapanje" (y_i -date tačke, $f(x_i)$ -naša procena):

1. Tačno uklapanje (Interpolacija)
   - $y_i=f(x_i)$
2. Regresija najmanjih kvadrata (Least squares regression)
   - Određujemo minimum kvadrata grešaka: $\sum_{i=1}^{n} (y_i - f(x_i))^2$

Regresija najmanjih kvadrata

Dat nam je niz parova tačaka (x_i, y_i) koji predstavljaju vrednosti funkcije koju aproksimiramo. Funkciju aproksimiramo polinomom (označenim sa f(x)), čiji se stepen zadaje unapred i mora biti bar za jedan manje od broja tačaka.

$$f(x) = \sum_{k=0}^{n} a_k \ x^k$$

Polinom uklapamo u date tačke tako da minimizujemo zbir kvadrata grešaka. Greška predstavlja razliku između vrednosti y_i za dato x_i i vrednosti polinoma u x_i tj. $f(x_i)$.

$$E = \sum_{i=1}^{n} (y_i - f(x_i))^2$$

Minimum zbira kvadrata grešaka određujemo izjedačavanjem parcijalnih izvoda E po svakom koeficjentu polinoma sa nulom. Na taj način dobijamo formulu za određivanje polinoma:

$$(A^T * A) * a = A^T * y$$

$$A = \begin{bmatrix} x_1^0 & x_1^1 & x_1^2 & \dots & x_1^m \\ x_2^0 & x_2^1 & x_2^2 & \cdots & x_2^m \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ x_n^0 & x_n^1 & x_n^2 & \cdots & x_n^m \end{bmatrix}_{n x (m+1)} \quad a = \begin{bmatrix} a_0 \\ \vdots \\ a_m \end{bmatrix}_{(m+1) x 1} \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}_{n x 1}$$