Aproximace (elementárních) funkcí

Vyhodnocení fcí f(x)

- Redukce argumentu – omezení rozsahu x: x‘=g(x) x’ v intervalu <a;b>

- Aproximace (v intervalu <a;b>, tj. výpočet f(x‘) =f(g(x))

  (polynomiální, racionální, tabulková, shift-add (coordic))

- Rekonstrukce f(x)=h(f(g(x)))

Příklady (ilustrační)

```
float sin(float x)
  float z=x mod (pi/2) //redukce na interval <0;pi/2>
  float r= (c0*z*z+c1*z+c2) / (c3*z*z+c4*z+c5); //racionální  aproximace (cx jsou koef.aprox.)
  return(r);  //zde není třeba zpětná rekonstrukce, fce je periodicka
  //pozor: redukce na interval pro větší rozsah
  //pozor: symetrie (sin(x) vs sin(-x)), rozsah (-1 az 1)\
  //pozor: chování sin() pro velká x, operace mod

float exp(float x)
  int N = x/(ln(2));  //N = celociselny koeficient (div)
  float m = x- N*(ln(2));   //m=zbytek  - redukce na interval <0;ln2>  (mod)
  //tj. vychazime z rovnice e^x = 2^(x/ln(2)) = 2^(N+m) = 2^N * 2^m
  float r= …  // r = aproximace 2^m v rozsahu <0;ln2>
  r = (2^N)*r  //rekonstrukce (2^N = bitovy posun)
  return(r)
  //pozor: redukce na interval pro větší rozsah
  //pozor: přechod na rozhraní ln(2)
```

Pozn. k názvosloví:

interpolace = nalezení přibl.hodnoty fce v jistém intervalu, hodnota je v některých bodech známa
extrapolace = nalezení přibl.hodnoty fce (interpolace) ale mimo interval známých hodnot
aproximace = nalezení přibl.hodnoty fce, avšak fce nemusí procházet danými body

Iterativní metody

Typicky jsou založeny na spojité a diverencovatelné fci ve formě f(X)=0

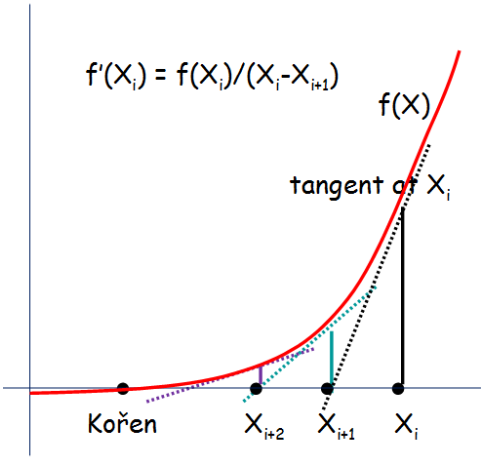
Řešení se hledá pomocí iteračního vzorce (metoda tečen):

(1) $X_{i+1} = X_i - f(X_i)/f'(X_i)$

Lze užít pro širokou třídu úloh – dělení, mocniny/odmocniny, log, exp

Iterace Newton(-Raphson) – dělení

hledáme kořen Q=P/D (nebo jen Q=1/D)



Metoda Newton-Raphson upravená pro výpočet převrácené hodnoty vychází ze vztahu
f(X) = 1/X – D = 0
Kořenem rovnice je X (= 1/D) (tj. D je fix a hledáme X, které bude konvergovat k 1/D)
Protože f'(X) = -(1/X)^2, dává po dosazení do (1) po úpravách rekurentní formuli:

Xi+1 = Xi.(2 – Xi.D)

Vybíráme X0 takové, aby platilo 0 < X0 < 2/D
Pro jednu iteraci jsou třeba dvě násobení, tedy pro n-bitové operandy potřebueme 2log(n) násobení

Příklad: Nalezněte 1/D kde D = 0.75 (dec) = 0.1100 (bin) :

Krok	Dekadicky	Chyba	Binárně	Chyba
X0=	1	0.333..	1.0000	<2^-1
X1=	1(2 – 0.75) = 1.25	0.0833..	1.0100	<2^-2
X2=	1.25(2 – 1.25*0.75) = 1.328125	0.005208	1.01010100	<2^-4
X3=	. . . = 1.333313	0.000021	1.0101010101010100	<2^-8

Iterace Newton-Raphson – dělení - rychlost konvergence (důkaz)

Metoda konverguje kvadraticky ($\varepsilon_{i+1} \le |\varepsilon_i|^2$) pro D<1

$X_{i+1} = X_i.(2 - X_i.D)$ a $\varepsilon_i = 1/D - X_i$ ...Potom...

$X_i = 1/D - \varepsilon_i = (1 - D.\varepsilon_i)/D$ a $\varepsilon_{i+1} = 1/D - X_{i+1}$ ...Z toho vyplývá...

$\varepsilon_{i+1} = 1/D - [X_i.(2-X_i.D)] = [1-2.D.X_i+(D.X_i)^2]/D$ ...Substitucí pro X_i …

$\varepsilon_{i+1} = [1 - 2.D.((1-D.\varepsilon_i)/D) + (1-D.\varepsilon_i)^2]/D = D.\varepsilon_i^2$ ...takze pak pro D<1 platí že

$\varepsilon_{i+1} \le |\varepsilon_i|^2$

Startovací hodnoty

Pro D z intervalu [½, 1) může být dobrá výchozí hodnota X0 = 1.5 protože omezuje počáteční chybu na maximálně 0.5

Lepší aproximací může být výraz X0 = 4(√3 – 1) - 2D = 2.9282 – 2D (max. chyba cca 0.1)

V obecnosti: dobrým počátečním odhadem můžeme rychlost konvergence značně urychlit (viz konvergence) → použití lepšího odhadu nebo look-up table.

(např. dělení 32 bitů pořebuje 5 kroků, pokud poč.odhad bude přesný jen na 4 bity potřebujeme jen 3 kroky)

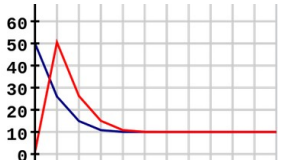
Další aproximace: X0=48/17-32/17*D (max.chyba cca 0.06) – pro IEEE čísla v jednoducé přesnosti pak stačí 3 následné iterace, pro dvojnásobnou přesnost 4 iterace (přesněji 10 násobení, 9sčítání, 2 posuvy)

Další zrychlení

Zejména pro operandy s vyšším počtem bitů lze v počátečních fázích mat.operace dělat s nižší přesností.

Např. Dělení 64/64bitů: 256*8 bitů lookup-table + 9bit násobička + 17bit násobička + 33bit násobička

Iterace Newton – odmocnina



Babylónská metoda (spec.případ Newtonovi metody): x=sqrt(a)

počítáme odmocninu (a), kde (x) je odhad odmocniny a x' nový odhad.

x' =1/2 * (a/x + x) ..tj. nový odhad je hodnota mezi a/x a x

(myšlenka: je li odhad (x) přehnaný pak a/x by byl urcitě „podehnaný" a naopak)

obrázek: konvergence pro startovací hodnoty x=1 a x=50 (počítáme sqrt(100))

Newtonova metoda

x=sqrt(a) → f(x) = x^2-a (kde hledáme bod kdy f(x)=0)

f'(x) = 2x

dosadíme do obecného vzorce (viz Newton(-Rapson) - dělení):

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} = x_k - \frac{x_k^2 - a}{2x_k} \qquad x_{k+1} = \frac{1}{2}\left(x_k + \frac{a}{x_k}\right)$$

Příklad, pro výpočet sqrt(9) kde jako startovací hdonotu zvolíme x0=a(=9)

x0 = 9
x1 = 5
x2 = 3.4
x3 = 3.02352941176471
x4 = 3.00009155413138
x5 = 3.00000000139698
x6 = 3.00000000000000

Newtonova metoda – poznámky

- Neznáme li derivaci fce f(x) můžeme se ji pokusit nahradit numerickou derivací
- zaokrouhlovací chyby při výpočtu typicky snižují rychlost konvergence
- konverguje kvadraticky (v každé iteraci se zdvojnásobí počet platných číslic) → volba startovací hodnoty je důležitá
- Halleyho metoda je taktéž metodou hledání kořenu, k hledání využívá i druhou (případně vyšší) derivaci a má kubickou(^3) (případně vyšší) konvergenci. Nemá zcela zaručenou stabilitu a je výpočetně náročnejší (zejména pro vyšší stupně)

Výpočet odmocniny – po číslicích (digit by digit)

myšlenka – chceme spočítat sqrt(a):

hledáme číslici (e) takovou že (x+e)(x+e)<=a

(tedy k řešení se přibližujeme zdola, číslo x+e pak bude novým základem pro další krok)

rozepíšeme: x*x+2*x*e+e*e <=a

Pro binární soustavu je e [0/1] *2^n čímž možno násobení nahradit posuvy

Příklad: sqrt(81) = sqrt(1010001_{bin})

x0=0000, x0^2=0000000, e=1000: 0000000+0000000+1000000 = 01000000 <= 01010001 (e is valid)

x1=1000, x1^2=1000000, e=0100: 1000000+1000000+0010000 = 10010000 > 01010001 (e invalid)

x2=1000, x2^2=1000000, e=0010: 1000000+0100000+0000100 = 01110000 > 01010001 (e invalid)

x3=1000, x3^2=1000000, e=0001: 1000000+0010000+0000001 = 01010001 > 01010001 (e is valid)

x4=1001

lze aplikovat i pro jiné soustavy – dekadická – vhodné pro výpočet „na papíře“

konvergence pomalejší než Newtonova metoda, ale výpočetně jednodušší

Aproximace na okolí bodu (Taylorovým polynomem)

Předpoklad: aproximovaná fce f() má v bodě x0 n-derivací

Za aproximaci fci f() v bodě x0 prohlásíme fci T() takovou která splňuje že prvních n-derivací je shodných s derivacemi f(x0).

Tuto podmínku splňuje Taylorův polynom:

$$T_n(x) = f(x_0) + \frac{f'(x_0)}{1!}(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \ldots + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n$$

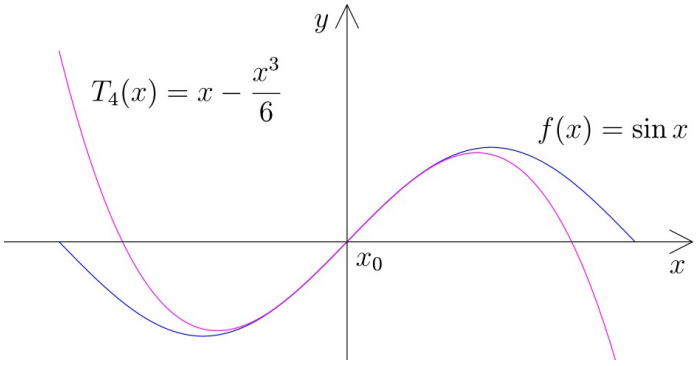
známe-li i n+1 derivaci fce f a platí li $|f^{(n+1)}(x)| \leq M \quad \forall x \in \mathcal{U}(x_0)$

lze odhadnout chybu aproximace jako $|e(x)| \leq \frac{M}{(n+1)!}|x - x_0|^{n+1}$

Příklad: Stanovte Taylorův polynom 4.stupně, který aproximuje funkci f(x) = sin x v bodě x0 = 0

$$T_4(x) = \sin 0 + x.\cos 0 - \frac{x^2}{2}\sin 0 - \frac{x^3}{6}\cos 0 + \frac{x^4}{24}\sin 0 \;=\; x - \frac{x^3}{6}$$

(pozn. sin(0)=0 (takže všechny sudé členy jsou nulové), sin' = cos, cos' = -sin)



Odhadněme pro jaký rozsah x platí že chyba aproximace e(x) je menší než 10^-6:
(odhad chyby = další člen Taylorovy řady)

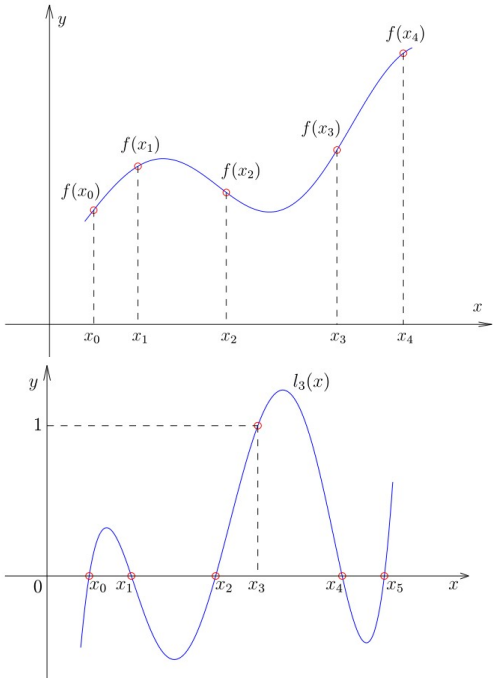
Výraz pro chybu je $e(x) = \frac{x^5}{5!}(-\cos\xi)$, kde $|\cos\xi| \leq 1$

vycházím z předpokladu že e(x) je 10^6 a hledám x:
a tedy 10^-6 = x^5 /120 → x = 0.164375 (rad) (= cca 9.4°)

T.Mainzer, ACS2

Aproximace iterpolačním polynomem (Interpolace)

Chceme aproximovat(interpolovat) fci která je dána svými hodnotami v několika bodech a to tak že fce má těmito danými body procházet. Aproximace pak slouží k odhadu hodnot v ostatních bodech.



Máme li zadány hodnoty fce f v n+1 různých bodech tak existuje právě jeden polynom n-tého stupně který je interpolačním polynomem pro zadanou fci.

Existuje několik mat.postupů určení ale všechny vedou na stejný výsledek – např. Langrangeův inter.polynom, Newtonův interpolační polynom a další.

Langrangeovy polynomy jsou takové které v jednom z uzlových bodů (xi) interpolace mají hodnotu f(xi)=1 a v ostatních 0.

Lze je zapsat ve tvaru:

$$l_i(x) = \frac{(x-x_0)(x-x_1)\dots(x-x_{i-1})(x-x_{i+1})\dots(x-x_n)}{(x_i-x_0)(x_i-x_1)\dots(x_i-x_{i-1})(x_i-x_{i+1})\dots(x_i-x_n)}$$

Výsledný aproximační polynom je pak ve tvaru

$$L_n(x) = \sum_{i=0}^{n} f(x_i) \cdot l_i(x)$$

pozn. Ln(x) = lagrangeův polynom řádu n, f(xi) = hodnota fce v bodě xi (z tabulky), li = lagrangeův polynom (mající hodnotu 1 v bodě xi)

Příklad (Langrangeův interpolační polynom):

i	0	1	2
x_i	0	1	3
$f(x_i)$	1	2	0

Mějme fci danou následující tabulkou a určeme její Langrangerův interpolační polynom a hodntu fce v bodu x=2.

$$L_2(x) = f(x_0) \cdot l_0(x) + f(x_1) \cdot l_1(x) + f(x_2) \cdot l_2(x)$$

Jednotlivé lagrangeovy polynomy:

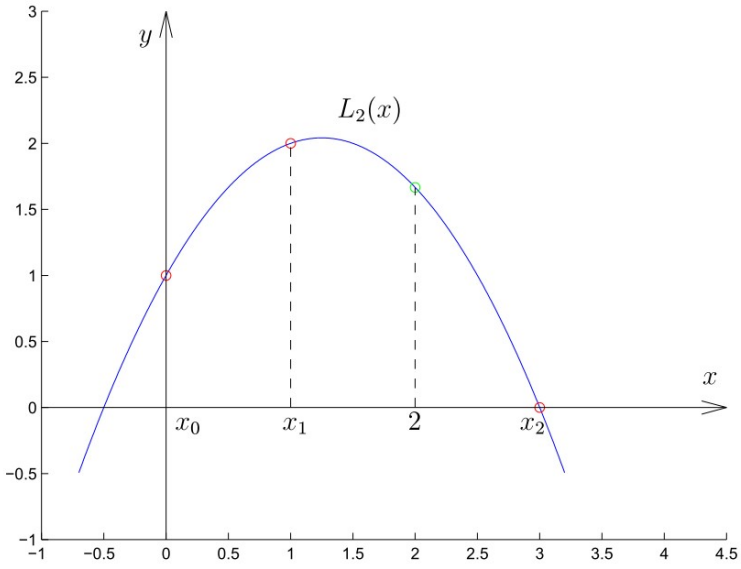
$$l_0(x) = \frac{(x-1)(x-3)}{(0-1)(0-3)} = \frac{1}{3}(x-1)(x-3)$$
$$l_1(x) = \frac{(x-0)(x-3)}{(1-0)(1-3)} = -\frac{1}{2}x(x-3)$$
$$l_2(x) = \frac{(x-0)(x-1)}{(3-0)(3-1)} = \frac{1}{6}x(x-1)$$

Po dosazení:

$$L_2(x) = 1 \cdot \left(\frac{1}{3}(x-1)(x-3)\right) + 2 \cdot \left(-\frac{1}{2}x(x-3)\right) + 0 \cdot \left(\frac{1}{6}x(x-1)\right) =$$
$$= \frac{1}{3}(x^2 - 4x + 3) - (x^2 - 3x) = -\frac{2}{3}x^2 + \frac{5}{3}x + 1$$

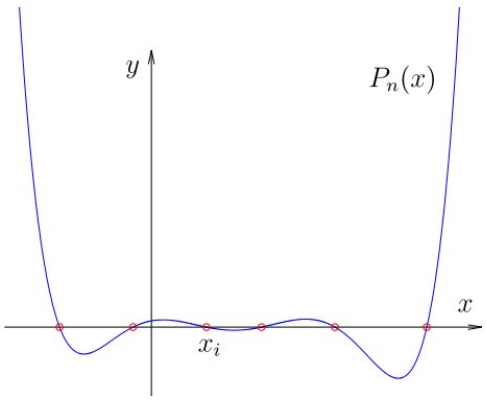
Hodnota fce v bodě x=2:

$$L_2(2) = -\frac{2}{3} \cdot 2^2 + \frac{5}{3} \cdot 2 + 1 = \frac{-8 + 10 + 3}{3} = \frac{5}{3}$$



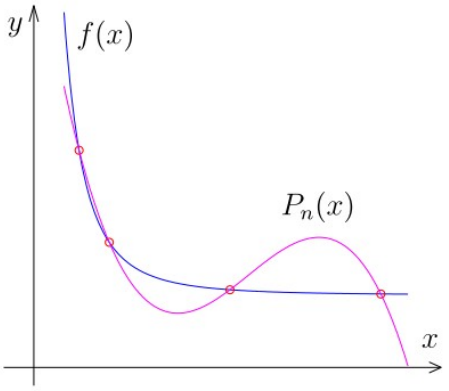
Aproximace iterpolačním polynomem – poznámky k použitelnosti

- Aproximace polynomy není obecně vhodné pro extrapolaci mimo rozsah zadaných hodnot (zejména pro vyšší řády polynomu)



- Obecně je třeba být obezřetný při použití vysokého stupně polynomu, kde ani interpolace uvnitř rozsahu nemusí dávat očekáváné výsledku z důvodů „oscilace“ (má tendenci nastávat zejména pro ekvidistantní uzly + vlivem nepřesnosti vstupních dat) – tyto oscilace lze ovlivnit vhodnou volbou uzlů.

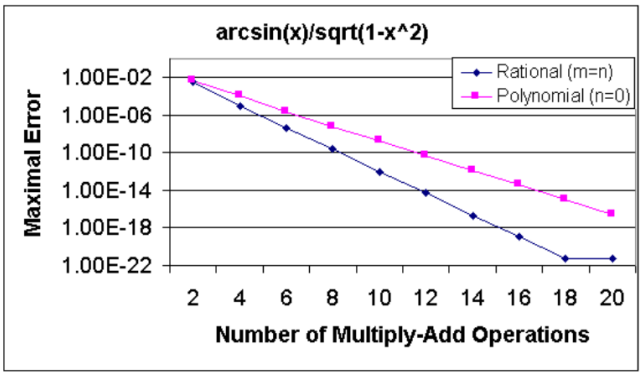
- Interpolace polynomy není vhodná pro asymptotické fce (blížící se konstantě)

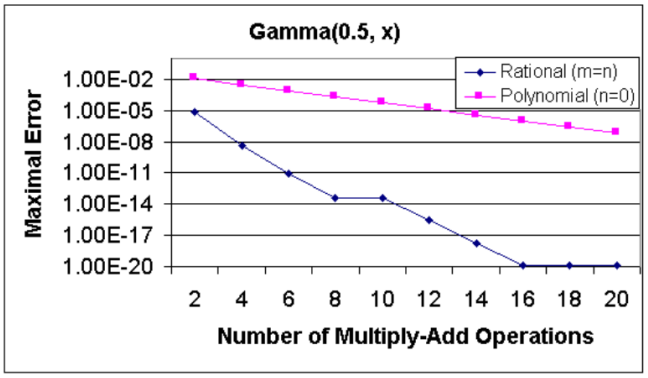


Polynomiální vs Racionální aproximace

$$f(x) \approx \frac{\cdots a_3x^3 + a_2x^2 + a_1x + a_0}{\cdots b_3x^3 + b_2x^2 + b_1x + b_0}$$ or $$\cdots c_3x^3 + c_2x^2 + c_1x + c_0$$

“Rational Approximation” “Polynomial Approximation”





Výpočet polynomu – poznámky

Vyhodnocení polynomu a+b*x+c*x^2+d*x^3, pro x:<0;1>

(obecně, pro spec.případy nemusí být vždy pravda)

nikdy: a + b*x+ c*POW(x,2) + d*POW(x,3)

trochu lépe: a + b*x + c*x*x + d*x*x*x

lépe: d*x*x*x+ c*x*x + b*x + a (sčítáme od menších čísel – predpoklad ze x=<0;1>)

správně: ((d*x+c)*x+b)*x+a

=Hornerovo schema, výpočetně efektivnější (zejména s FMA), a přesnější

Paralelizace (Estrinův algorimus)

vyhodnocení polynomu: a+b*x+c*x^2+d*x^3+...+h *x^7

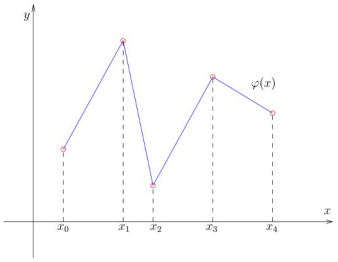
krok1 (vše paralelně): X2=x*x, W=hx+g, E= fx+e, R=dx+c, T=bx+a

krok2: X4=Q^2(=x^4), U=W*X2+E (=hx^3+gx^2+fx+e), I=R*X2+T (=dx^3+cx^2+bx+a)

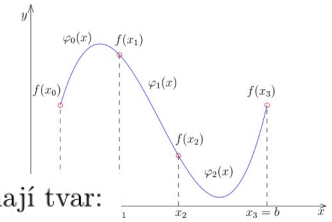
krok3: result = U*X4+I

pozn. tedy složitost log(n)   (n=řád polynomu)

Interpolace spline funkcemi



Je interpolace jednoduchými fukcemi mezi zadanými body. Nejjednodušší spline funkcí je tzv. lineární spline funkce; jde vlastně o lomenou čáru spojující zadané interpolované body.



Nejvíce používanou je tzv. kubická spline interpolace - fce f je po částech aproximáva polynomy 3.stupně (volba vyššího stupně již nepřináší lepší výsledky). Pro uzové body pak platí že v daných bodech mají obě splineové fce stejnou hodnotu a spojité první a druhé derivace.

Jednotlivé funkce $\varphi_i(x)$ (na každém intervalu $\langle x_i, x_{i+1}\rangle$ jde o jinou funkci) mají tvar:

$$\varphi_i(x) = a_i + b_i(x - x_i) + \frac{c_i}{2}(x - x_i)^2 + \frac{d_i}{6}(x - x_i)^3$$

Musí platit: $(\varphi \in \mathbb{C}^2(\langle a, b\rangle)$ a podmínky interpolace)

1) $\left.\begin{array}{l}\varphi_0(x_1) = \varphi_1(x_1)\\ \varphi_1(x_2) = \varphi_2(x_2)\end{array}\right\}$ spojitost funkce φ

2) $\left.\begin{array}{l}\varphi_0'(x_1) = \varphi_1'(x_1)\\ \varphi_1'(x_2) = \varphi_2'(x_2)\end{array}\right\}$ spojitost 1. derivace funkce φ

3) $\left.\begin{array}{l}\varphi_0''(x_1) = \varphi_1''(x_1)\\ \varphi_1''(x_2) = \varphi_2''(x_2)\end{array}\right\}$ spojitost 2. derivace funkce φ

4) $\left.\begin{array}{l}\varphi_0(x_0) = f(x_0)\\ \varphi_1(x_1) = f(x_1)\\ \varphi_2(x_2) = f(x_2)\\ \varphi_2(x_3) = f(x_3)\end{array}\right\}$ interpolační podmínky

Aby bylo řešení jednoznačné je třeba k výše uvednému (10 rovnic) nutno ještě doplnit okrajové podmínky (+2 rovnice). Typicky se doplňuje jedna z níže uvedených:

5) a) $\varphi'(a) = f'(a),\ \varphi'(b) = f'(b)$ … podmínky tečen

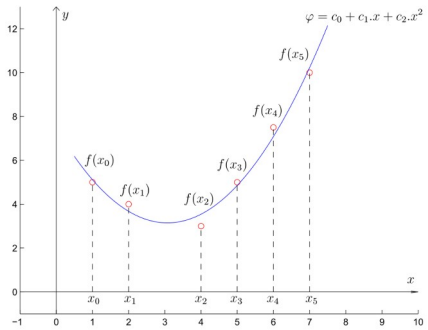
b) $\varphi'(a) = \varphi'(b),\ \varphi''(a) = \varphi''(b)$ … podmínky periodicity

c) $\varphi''(a) = 0,\quad \varphi''(b) = 0$ … tzv. přirozené podmínky

Podmínky 1-5 představují soustavu lineárních algebraických rovnic (s řídkou maticí). Jejím vyřešením určíme koeficienty ai,bi,ci.di. tj. Aproximační splineové fce.

L2 aproximace

Chceme aproximovat fci která je dána svými hodnotami v několika bodech a to tak že chceme minimalizovat odchylku fce od naměřených dat (nejčastěji se používá kvadratická odchylka → met.nejmenších čtverců). Vhodné např. pro případy kdy jsou vstupní hodnoty zatížené chybou (typicky výsledky měření). Řád aproximační fce je často << než počet dat.



(dále používananou je minimalizace maximální odchyky - minimax)

Příklad

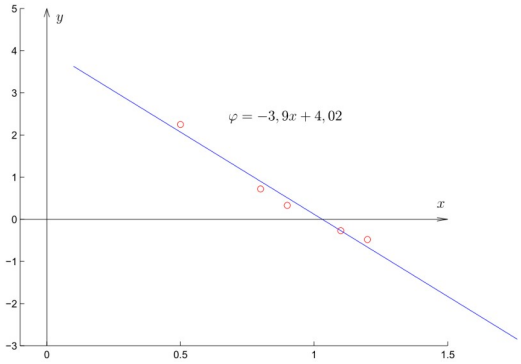
Tabulkou jsou dány naměření hodnoty fce f. O fci f přepokládáme že je lineární, tj. Budeme ji aproximovat lineární funkcí metodou nejmenších čtverců.

x_i	$0,5$	$0,8$	$0,9$	$1,1$	$1,2$
$f(x_i)$	$2,25$	$0,72$	$0,33$	$-0,27$	$-0,48$

Tedy:

$$c_1 x_i + c_0 = f(x_i)$$

$$\begin{bmatrix} 0,5 & 1 \\ 0,8 & 1 \\ 0,9 & 1 \\ 1,1 & 1 \\ 1,2 & 1 \end{bmatrix} . \begin{bmatrix} c_1 \\ c_0 \end{bmatrix} = \begin{bmatrix} 2,25 \\ 0,72 \\ 0,33 \\ -0,27 \\ -0,48 \end{bmatrix} \quad \Leftrightarrow \quad Q.c = f$$



Získanou soustavu řešíme ve smyslu metody nejm.čtverců:

$$Q^T Q c = Q^T f$$

poznámky:

- Tuto metodu lze zobecnit i pro spojité fce, kde místo součtu (kvadratických) odchylek figuruje integrál (kvadrátu) rozdílu fcí (dané fce a hledané aproximace).

- Soustavu bázových fcí je vhodné volit dle přepokládaného tvaru řešení – bázové polynomy by měli být ortogonální (jinými používanými jsou např. Čebyševovy polynomy)

- přesnost aproximace silně závisí na aproximované fci. Viz tabulka počtu významných bitů pro různé fce a řády aproximačního polynomu. (aproximováno metodou minimalizace max.odchylky)

Function\Degree	2	3	4	5	6	7	8	9
$\sin(x)$	7.8	12.7	16.1	21.6	25.5	31.3	35.7	41.9
e^x	6.8	10.8	15.1	19.8	24.6	29.6	34.7	40.1
$\ln(1+x)$	8.2	11.1	14.0	16.8	19.6	22.3	25.0	27.7
$(x+1)^x$	6.3	8.5	11.9	14.4	18.1	20.0	22.7	25.1
$\arctan(x)$	8.7	9.8	13.2	15.5	17.2	21.2	22.3	24.5
$\tan(x)$	4.8	6.9	8.9	10.9	12.9	14.9	16.9	19.0
$\sqrt{x}$	3.9	4.4	4.8	5.2	5.4	5.6	5.8	6.0
$\arcsin(x)$	3.4	4.0	4.4	4.7	4.9	5.1	5.3	5.5

Tabulkové metody

Tabulka s koeficienty

Požadavek na vysokou přenost aproximace vede na vysoké řády aproximačních polynomů – což má za následek složitější (déletrvající) výpočet a obtížnější vypořádání se s numerickými chybami. Metoda jak se s tímto vypořádat je rozdělit interval aproximace na menší části – a pro každou část mít (v tabulce) jiné koeficienty aproximace platné pro daný interval.

Interval	Degree	Error
$[0, \pi/4]$	5	0.609×10^{-7}
	6	0.410×10^{-8}
	7	0.418×10^{-10}

Interval	Degree	Error
$[0, \pi/16]$	3	0.478×10^{-7}
	4	0.472×10^{-8}
	5	0.382×10^{-11}
$[\pi/16, \pi/8]$	3	0.140×10^{-6}
	4	0.454×10^{-8}
	5	0.113×10^{-10}
$[\pi/8, 3\pi/16]$	3	0.228×10^{-6}
	4	0.418×10^{-8}
	5	0.183×10^{-10}
$[3\pi/16, \pi/4]$	3	0.307×10^{-6}
	4	0.367×10^{-8}
	5	0.246×10^{-10}

Příklad: aproximujeme sin(x), x =<0;pi/4> s přesností 10^-8.

- pro celý rozsah potřebujeme polynom 6 řádu

- pro 2 subintervaly potřebujeme polynom 5 řádu

- pro 4 subintervaly potřebujeme polynom 4 řádu

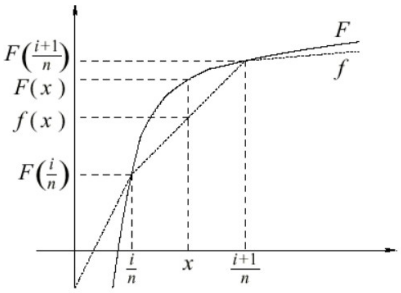
Lookup table - úplná

Úplná tabulka (tab), kde (x) je přímo indexem do tabulky a hodnota f(x) je uložena v tabulce. Vhodné jen pro nízké rozlišení x, jinak je tab.obrovská - velikost tabulky M*2^N bitů pro N-bitové a M-bitovou hodnotu f(x).

pozn. Velikost tabulky nemusí implikovat jen obsazení paměti, ale i rychlost přístupu (stránkování, cachování)

Lookup table s lin.aproximací (+mul+add)

Tabulka s lineární aproximací (tab+mul+add) – vyšší bity x jsou indexem do tabulky a hodnota f(x) je počítána lineární aproximací mezi dvěmi sousedními body v tabulce. Někdy se do tabulky místo hodnot f(x) ukládají koeficienty a,b (aproximačního výrazu ax+b).



Lookup table s funkcí

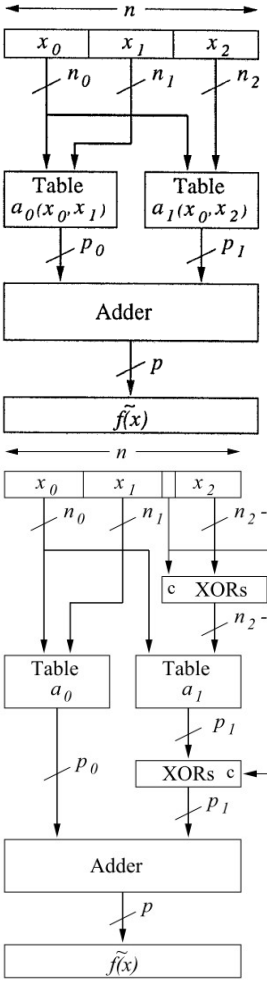
Vstup x je rozdělen na dvě(tři,..) bitové množiny (např. vyšší byte, nižší byte) a je využito mat.vlastností dané fce (tj. nelze obecně vždy, je vázáno na funkci)

Např. x je 16b, a = horních 8b, b=dolních 8b

sin(x) = sin(a+b) = sin(a) cos (b) + cos(a) sin(b)

použijeme lookup tabulky pro sin(a),sin(b), případně cos(a),cos(b) – tyto tabulky mají „adresu“ jen 8b (oproti vstupímu 16bitovému rozsahu x) + příslušné matematické operace

Bipartitní(multipartitní) tabulka (+add)



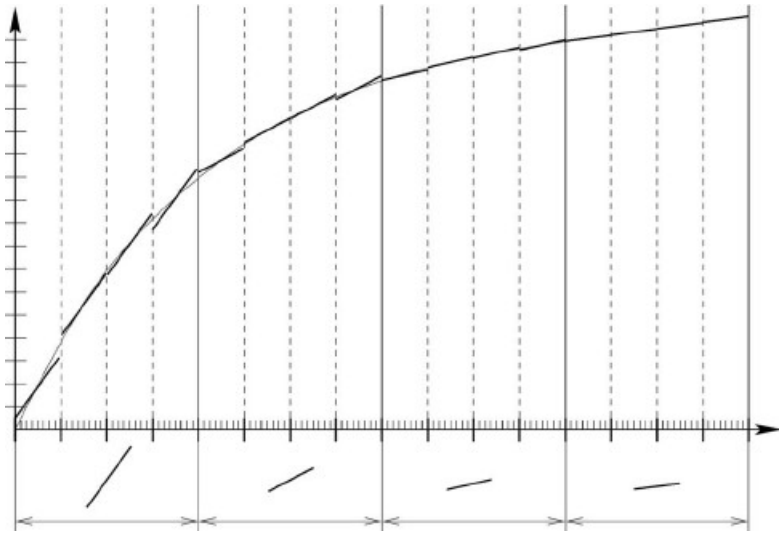
vstup (x) je v rozdělěn na 3(či více) bitových množin, které slouží jako ukazatele to tabulek. Hodnoty z tabulek jsou po té sečteny a použity jako aproximace f(x).

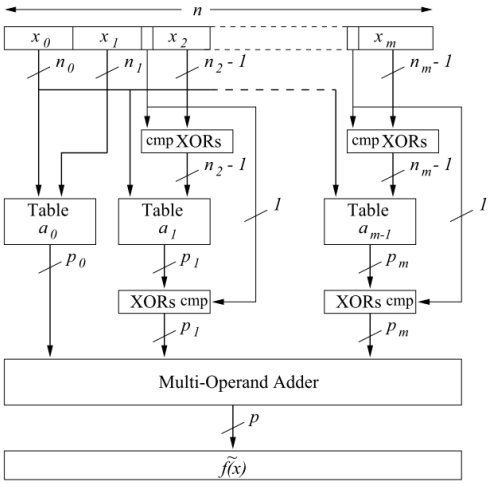
Použitím jen části (x) jako ukazatele je paměť významně redukována.

Pro každou část x (vyjma x0,x1) lze vyhradit jeden (nejvyšší) bit na znaménko a tabulky konstruovat jako symetrickou – o tento bit může být dále tabulka menší – symetrická bipartitní tabulka.

Příklad redukce bipartitní tabulky pro 16bitový vstupní operand.

$f(x)$	Symmetric Bipartite Table n_0, n_1, n_2	Symmetric Bipartite Table Sizes	Standard Table Sizes	Compression
$1/x$	6, 4, 5	$2^{10} \times 17 + 2^{11} \times 7$	$2^{15} \times 15$	15.5
$\sqrt{x}$	5, 5, 6	$2^{10} \times 17 + 2^{10} \times 6$	$2^{16} \times 15$	41.7
$\sin(x)$	6, 4, 6	$2^{10} \times 18 + 2^{11} \times 7$	$2^{16} \times 16$	32.0
$\cos(x)$	6, 4, 6	$2^{10} \times 18 + 2^{11} \times 7$	$2^{16} \times 16$	32.0
$\tan^{-1}(x)$	6, 4, 6	$2^{10} \times 18 + 2^{11} \times 7$	$2^{16} \times 16$	32.0
$\log_2(x)$	7, 3, 5	$2^{10} \times 18 + 2^{11} \times 8$	$2^{15} \times 16$	15.0
2^x	6, 4, 6	$2^{10} \times 17 + 2^{11} \times 8$	$2^{16} \times 15$	29.1



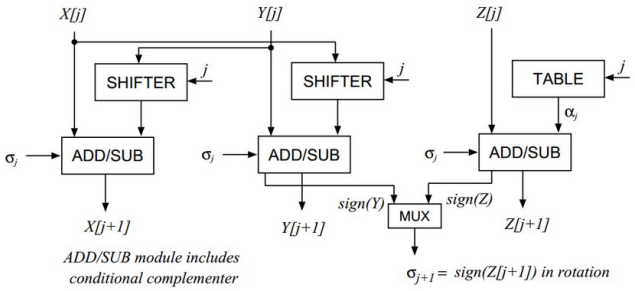


Příklad redukce bipartitní a multipatitní tabulky pro 24bitový vstupní operand:

$f(x)$	Tables	Bit Partitions	Table Sizes	Total Memory
$1/x$	1	23	$2^{23} \cdot 23$	192,937,984
$1/x$	2	9, 7, 7	$2^{16} \cdot 25 + 2^{15} \cdot 9$	1,933,312
$1/x$	3	11, 3, 4, 5	$2^{14}(26 + 12) + 2^{15} \cdot 8$	884,736
$1/x$	4	11, 3, 3, 3, 3	$2^{14} \cdot 27 + 2^{13}(13 + 10 + 7)$	688,128
$1/x$	5	11, 3, 2, 2, 2, 3	$2^{14} \cdot 27 + 2^{12}(13 + 11 + 9) + 2^{13} \cdot 7$	634,880
$1/x$	6	11, 3, 1, 2, 2, 2, 2	$2^{14} \cdot 28 + 2^{11} \cdot 14 + 2^{12}(13 + 11 + 9 + 7)$	651,264
$\sqrt{x}$	1	24	$2^{24} \cdot 23$	385,875,968
$\sqrt{x}$	2	8, 7, 9	$2^{15} \cdot 25 + 2^{16} \cdot 9$	1,409,024
$\sqrt{x}$	3	9, 5, 5, 5	$2^{14} \cdot 26 + 2^{13}(12 + 7)$	581,632
$\sqrt{x}$	4	10, 3, 3, 4, 4	$2^{13} \cdot 27 + 2^{12} \cdot 14 + 2^{13}(11 + 7)$	425,984
$\sqrt{x}$	5	10, 3, 2, 3, 3, 3	$2^{13} \cdot 27 + 2^{11} \cdot 14 + 2^{12}(12 + 9 + 6)$	360,448
$\sqrt{x}$	6	10, 3, 2, 2, 2, 2, 3	$2^{13} \cdot 28 + 2^{11}(15 + 13 + 11 + 9) + 2^{12} \cdot 7$	356,352
$\sin(x)$	1	24	$2^{24} \cdot 24$	402,653,184
$\sin(x)$	2	8, 8, 8	$2^{16} \cdot 26 + 2^{15} \cdot 9$	1,998,848
$\sin(x)$	3	10, 4, 5, 5	$2^{14} \cdot (27 + 12 + 7)$	753,664
$\sin(x)$	4	10, 4, 3, 3, 4	$2^{14} \cdot 28 + 2^{12}(13 + 10) + 2^{13} \cdot 7$	610,304
$\sin(x)$	5	11, 2, 2, 3, 3, 3	$2^{13} \cdot 28 + 2^{12} \cdot 14 + 2^{13}(12 + 9 + 6)$	507,904
$\sin(x)$	6	11, 2, 2, 2, 2, 2, 3	$2^{13} \cdot 29 + 2^{12}(15 + 13 + 11 + 9) + 2^{13} \cdot 7$	491,520
$\ln(x)$	1	23	$2^{23} \cdot 24$	201,326,592
$\ln(x)$	2	9, 6, 8	$2^{15} \cdot 26 + 2^{16} \cdot 10$	1,507,328
$\ln(x)$	3	10, 4, 4, 5	$2^{14} \cdot 27 + 2^{13} \cdot 12 + 2^{14} \cdot 8$	671,744
$\ln(x)$	4	10, 4, 3, 3, 3	$2^{14} \cdot 28 + 2^{12}(13 + 10 + 7)$	581,632
$\ln(x)$	5	11, 2, 2, 2, 3, 3	$2^{13} \cdot 28 + 2^{12}(14 + 12) + 2^{13}(10 + 7)$	475,136
$\ln(x)$	6	11, 2, 2, 2, 2, 2, 2	$2^{13} \cdot 29 + 2^{12}(15 + 13 + 11 + 9 + 7)$	462,848

Cordic (lookup tabulka +shift+add, iterativní)

Cordic (COordinate ROtation DIgital Computer) je metoda využívající iterační metody pro jednoduchý a rychlý výpočetu goniometrických funkcí za použití sčítání a posuvů.



Pro výpočet sin(x) a cos(x) algoritmus CORDIC v několika krocích rotuje bod na jednotkové kružnici o předem vypočítané úhly δ a jakmile je úhel φ ten, který hledáme, máme i hodnoty souřadnic `x` a `y`, které reprezentují `cos` a `sin`.

Pro rotaci bodu platí že [x',y']= $\begin{bmatrix} \cos(\delta) & -\sin(\delta) \\ \sin(\delta) & \cos(\delta) \end{bmatrix}$.[x,y]

$$\cos\alpha = \frac{1}{\sqrt{1+\tan^2\alpha}}$$

$$\sin\alpha = \frac{\tan\alpha}{\sqrt{1+\tan^2\alpha}}$$

upravíme: [x',y']= $\frac{1}{\sqrt{1+\tan(\delta)^2}} \begin{bmatrix} 1 & -\tan(\delta) \\ \tan(\delta) & 1 \end{bmatrix}$.[x,y]

hlavní trik pak spočívá v tom že se budeme rotovat pouze o úhly jejichž tangens je (záporná) mocnina 2 – a tedy násobení se stane jen binárním posuvem. úhly odpovídající mocněne dvou jsou uloženy v tabulce. (hodnota konstanty před maticí se limitně blíží 0,60725293501 a je možné ji vynásobit až nakonec). V každém kroku pak zjišťujeme jestli jsme se se k žádoucímu úhlu (vstupu x) a podle toho v následujícím kroku rotujeme po a nebo proti směru.

Myšlenku lze pak upravit a metodu cordic použít pro výpočet ln,exp,sqrt,atanh,..

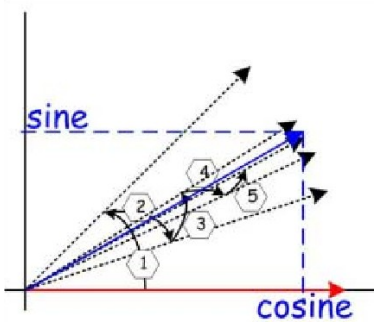
Příklad: Cos(30°):

startovací bod je 45st.

30 = (zaokrouhleně)

45 – 26.6 + 14.0 – 7.1 + 3.6 – 1.8 – 0.9 + 0.4 – 0.2 + 0.1 … 9

pozn. tg(26.6)= 0.5 = 2^-1 = >>1 , tg(14.0) = 0.25 = 2^-2 = >>2, … atd



T.Mainzer, ACS2

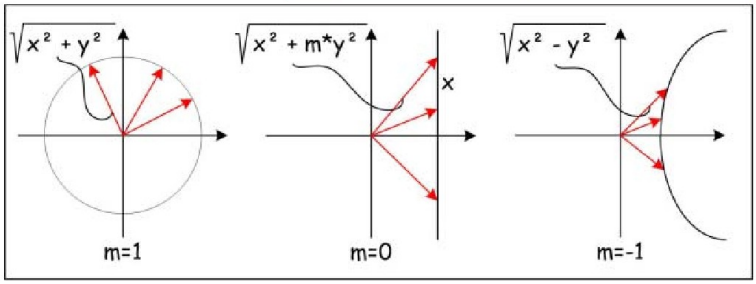
- CORDIC equations

$x^{(i+1)} = x^{(i)} - m\, d_i\, y^{(i)} 2^{-i}$

$y^{(i+1)} = y^{(i)} + d_i\, x^{(i)} 2^{-i}$

$z^{(i+1)} = z^{(i)} - d_i\, e^{(i)}$

m	Coordinate System	Domain of Convergence
1	circular	1.74 radian
0	linear	2
-1	hyperbolic	1.11 radian



CORDIC		Rotation	Vectoring
Linear MUL/DIV/MAC K = 1	IN	x_n, z_n, y_n	x_n, y_n, z_n
	OUT	$y_{n+1} = x_n * z_n + y_n$	$z_{n+1} = y_n / x_n + z_n$
Circular SINE/COSINE angle/ magnitude K = 1.64676	IN	$z_n = angle,\ x_n = K, y_n = 0$	$x_n, y_n,\ z_n = 0$
	OUT	$x_{n+1} = \cos(angle)$ $y_{n+1} = \sin(angle)$	$z_{n+1} = \arctan(y_n/x_n)$ $x_{n+1} = 1/K*(x_n^2 + y_n^2)^{1/2}$
Hyperbolic K = 0.828	IN	$z_1 = angle,\ x_1 = K',\ y_1 = 0$	$y_1 < x_1,\ z_1 = 0$
	OUT	$x_{n+1} = \cosh(angle)$ $y_{n+1} = \sinh(angle)$	$z_{n+1} = \operatorname{arctanh}(y_1/x_1)$ $x_{n+1} = 1/K'*(x_1^2 - y_1^2)^{1/2}$

Odkazy a Literarura:

Elementary Functions, Algorithms and Implementation, Jean-Michel Muller (ISBN 0-8176-4372-9, 2006 Birkhauser Boston, 2nd edition)