UNIVERZITA PALACKÉHO V OLOMOUCI
PŘÍRODOVĚDECKÁ FAKULTA
KATEDRA MATEMATICKÉ ANALÝZY A APLIKACÍ MATEMATIKY

BAKALÁŘSKÁ PRÁCE

Elementární funkce a praktický výpočet jejich hodnot

Vedoucí bakalářské práce:
RNDr. Jan Tomeček, Ph.D.
Rok odevzdání: 2013

Vypracoval:
Jan Musil
M–E poj, 3. ročník

Prohlášení

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně pod vedením pana RNDr. Jana Tomečka, Ph.D. s použitím uvedené literatury.

V Olomouci dne 20. dubna 2013

Poděkování

Na tomto místě bych chtěl poděkovat především svému vedoucímu bakalářské práce panu RNDr. Janu Tomečkovi, Ph.D., že měl se mnou dostatek trpělivosti, aby mi pomohl dovézt tuto práci ke zdárnému konci. Také bych rád poděkoval své rodině a přátelům, že mne po celou dobu studia podporovali.

Obsah

Úvod

V běžném životě se můžeme setkat s řadou problémů, které lze velmi jednoduše vyřešit tak, že je reprezentujeme konkrétními matematickými modely, jenž jsou popsány tzv. elementárními funkcemi. Elementární funkce je každá funkce, která vznikne jako výsledek konečného počtu operací sčítání, odčítání, násobení, dělení a skládání základních elementárních funkcí. Tyto základní elementární funkce se dělí na konstantní, mocninné, exponenciální, logaritmické, goniometrické, cyklometrické, hyperbolické a hyperbolometrické.

Výpočty funkčních hodnot některých těchto funkcí jsou snadné a stačí nám k tomu jednoduché binární operace. Pro výpočet funkčních hodnot některých složitějších funkcí, jako např. $\sin(\alpha)$, či $\log(x)$, bychom však museli umět z hlavy určit hodnotu úhlu sinus, či logaritmovat, což ne každý sám zvládne. Jak ale jednoduše zjistit hodnoty funkcí $\sin(\alpha)$, či $\log(x)$, aniž bychom toto uměli?

V této bakalářské práci se dozvíme, že to ani není zapotřebí. Stačí, když budeme tyto funkce aproximovat podobné funkci s nějakou blíže specifikovanou chybou, a tím dostaneme výsledek s předem zvolenou přesností. Jedním z algoritmů pro aproximaci funkce je CORDIC (COordinate Rotation DIgital Computer), který blíže popíšu v druhém bloku pro výpočet hodnot $\sin(\alpha)$ a $\cos(\alpha)$. K jejich výpočtu však potřebujeme hodnoty $\operatorname{arctg}(\alpha)$, které si spočítáme v prvním bloku pomocí Taylorových polynomů.

1. Taylorovy polynomy

V matematice se pro výpočet hodnot funkcí může využít Taylorova řada. Je to zvláštní mocninná řada, která byla pojmenována po anglickém matematikovi Brooku Taylorovi. Ten ji publikoval roku 1712, i když metoda aproximace funkce mocninou řadou je známa již od roku 1671. Informace pro tuto kapitolu jsem čerpal z [1], [2], [3], [4], [5], [6] a [8].

Za jistých předpokladů lze funkci $f(x)$ v okolí bodu a vyjádřit (neboli rozvinout) jako mocninnou řadu. Takto vyjádřená funkce pomocí Taylorovy řady se značí jako Taylorův rozvoj.

Chceme-li však vyjádřit pouze přibližné hodnoty funkce (například s přesností na předem stanovený počet desetinných míst), nemusíme vyjadřovat všechny členy této nově vzniklé Taylorovy řady, avšak můžeme je omezit pouze na členy s nižšími derivacemi, až do příslušného řádu (např. n). Tím získáme tzv. Taylorův polynom řádu n. V dalším textu předpokládejme, že $n, k \in \mathbb{N}_{\mathbf{0}}$.

Taylorův polynom tedy aproximuje hodnoty funkce, která má v daném bodě derivaci, pomocí polynomu, jehož koeficienty závisí na derivacích funkce v tomto bodě. V případě existence všech derivací funkce f v bodě a lze Taylorovu řadu zapsat jako:

$$f(x) = f(a)+\frac{f'(a)}{1!}(x-a)+\frac{f''(a)}{2!}(x-a)^2+\frac{f'''(a)}{3!}(x-a)^3+\ldots = \sum_{k=0}^{\infty} \frac{f^{(k)}(a)}{k!}(x-a)^k.$$

Má-li funkce f v bodě a derivace až do řádu n, pak Taylorův polynom řádu n funkce f v bodě a je polynom:

$$T_n^{f,a}(x) = f(a)+\frac{f'(a)}{1!}(x-a)+\frac{f''(a)}{2!}(x-a)^2+\ldots+\frac{f^{(n)}(a)}{n!}(x-a)^n = \sum_{k=0}^{n} \frac{f^{(k)}(a)}{k!}(x-a)^k,$$

kde nultou derivací je myšlena samotná funkce, tzn. $f^{(0)} = f$.

1.1. Maclaurinova řada

Pro výpočet hodnot Taylorovy řady v okolí bodu a, kdy $a = 0$, se bavíme o Maclaurinově řadě, která má tvar:

$$f(x) = f(0) + \sum_{k=1}^{\infty} \frac{f^{(k)}(0)}{k!} x^k.$$

O samotném Taylorově polynomu můžeme obecně říct, že dvě funkce si jsou v okolí bodu x tím více podobné, čím více se podobají součty jejich derivací v tomto bodě až do vyšších řádů, tedy $T_{n+1}^{f,a}(x)$ je vždy lepší aproximací funkce $f(x)$, než $T_n^{f,a}(x)$.

1.2. Arkustangens

Nyní použijeme poznatků Taylorova polynomu pro výpočet hodnot funkce arkustangens, abychom je mohli následně použít při konstrukci algoritmu CORDIC. Jelikož víme, že:

$$f(t) = \operatorname{arctg}(t) \qquad \Longrightarrow \qquad f'(t) = \frac{1}{1+t^2},$$

pak platí i obráceně

$$\int \frac{1}{1+t^2} dt \qquad \Longrightarrow \qquad f(t) = \operatorname{arctg}(t).$$

Využijeme-li vlastností, že $D(\operatorname{arctg}(t)) = \langle -1, 1 \rangle$ a že

$$\frac{1}{1-r} \quad = \quad 1 + r + r^2 + \ldots \quad = \quad \sum_{k=0}^{\infty} r^k \qquad\qquad \forall r \in (-1, 1)\,,$$

substitucí $r = -t^2$ můžeme integrál $\int \frac{1}{1+t^2} dt$ vyjádřit jako $\int \frac{1}{1-r} dr$:

$$\int_0^x \sum_{k=0}^{\infty} r^k dr = \int_0^x \sum_{k=0}^{\infty} (-t^2)^k dt = \int_0^x \sum_{k=0}^{\infty} (-1)^k \cdot t^{2k} dt = \sum_{k=0}^{\infty} \frac{(-1)^k \cdot x^{2k+1}}{2k+1}.$$

Tedy Taylorův polynom řádu $n = 2k + 1$ funkce arkustangens je tvořen hodnotami:

$$x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \ldots + \frac{(-1)^k \cdot x^{2k+1}}{2k+1}.$$

Nyní zkontrolujeme, zda-li tyto hodnoty vyhovují hodnotám Taylorova polynomu, tedy vztahu:

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + ... + \frac{f^{(n)}(a)}{n!}(x-a)^n = \sum_{k=0}^{n} \frac{f^{(k)}(a)}{k!}(x-a)^k.$$

1.2.1. Postupná aproximace

Jelikož funkce $\mathrm{arctg}(x)$ je lichá funkce symetrická kolem počátku, budeme počítat aproximaci Taylorova polynomu pro $a = 0$, tzv. Maclaurinovu řadu. Nejprve si spočítáme funkční hodnoty derivací funkce $\mathrm{arctg}(x)$ až do řádu n, tedy $\mathrm{arctg}^{(n)}(0)$:

$$f(x) = \mathrm{arctg}(0) + \frac{\mathrm{arctg}'(0)}{1!}(x) + \frac{\mathrm{arctg}''(0)}{2!}(x)^2 + \frac{\mathrm{arctg}'''(0)}{3!}(x)^3 + \ldots + \frac{\mathrm{arctg}^{(n)}(0)}{n!}(x)^n =$$

$$= 0 + \frac{\frac{1}{1+0^2}}{1!}(x) + \frac{\frac{-(2\cdot 0)}{(1+0^2)^2}}{2!}(x)^2 + \frac{\frac{-2+6\cdot 0^2}{(1+0^2)^3}}{3!}(x)^3 + \frac{\frac{-(24\cdot 0\cdot(-1+0^2))}{(1+0^2)^4}}{4!}(x)^4 + \ldots + \frac{\mathrm{arctg}^{(n)}(0)}{n!}(x)^n =$$

$$= x + \frac{0}{2\cdot 1}x^2 + \frac{-2}{3\cdot 2\cdot 1}x^3 + \frac{0}{4\cdot 3\cdot 2\cdot 1}x^4 + \frac{24}{5\cdot 4\cdot 3\cdot 2\cdot 1}x^5 + \ldots + \frac{\mathrm{arctg}^{(n)}(0)}{n!}(x)^n =$$

$$= \sum_{k=0}^{n} \frac{(-1)^k \cdot 2k! \cdot x^{2k+1}}{(2k+1)!} = \sum_{k=0}^{n} \frac{(-1)^k \cdot x^{2k+1}}{(2k+1)} = x - \frac{x^3}{3} + \frac{x^5}{5} + \ldots + \frac{(-1)^n \cdot x^{2n+1}}{(2n+1)}.$$

Funkce arkustangens tvoří alternující řadu. Sudé derivace Maclaurinovy řady jsou rovny nule ($f^{(2k)}(0) = 0$). Pro liché derivace platí vztah: $f^{(2k+1)}(0) = (2k)!$.

1.2.2. Obor konvergence

Nyní vyšetříme obor konvergence Taylorovy řady funkce arkustangens, tedy $\sum_{k=0}^{\infty} \frac{(-1)^k \cdot x^{2k+1}}{(2k+1)}$. Nejprve určíme jeho střed:

$$\sum_{k=0}^{\infty} \frac{(-1)^k \cdot x^{2k+1}}{(2k+1)} = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2^k} \cdot x}{(2k+1)} = x \cdot \sum_{k=0}^{\infty} (-1)^k \frac{x^{2^k}}{(2k+1)} \Rightarrow a = 0.$$

Následně určíme k-tý člen posloupnosti $b_k = \frac{(-1)^k}{2k+1}$, pomocí něhož spočítáme poloměr konvergence $r = \frac{1}{\lambda}$, kde λ odpovídá limitě:

$$\lambda = \lim_{k\to\infty} \left| \frac{b_{k+1}}{b_k} \right| = \lim_{k\to\infty} \left| \frac{\frac{(-1)^{k+1}}{(2k+3)}}{\frac{(-1)^k}{(2k+1)}} \right| = \lim_{k\to\infty} \frac{2k+1}{2k+3} = 1 \quad \Rightarrow \quad r = \frac{1}{\lambda} = 1.$$

Tento poloměr však platí pro x^2. Jeho odmocněním dostaneme poloměr pro x. Jelikož nemůže mít zápornou hodnotu, získáme:

$$r(x^2) = 1 \quad \Rightarrow \quad r(x) = 1,$$

z čehož vyplývá interval absolutní konvergence $(a - r, a + r) \Rightarrow (-1, 1)$. Nyní musíme ještě vyšetřit konvergenci v krajních bodech $\{-1, 1\}$:

$$x = -1 : \qquad \sum_{k=0}^{\infty} \frac{(-1)^k \cdot (-1)^{2k+1}}{(2k+1)} = -\sum_{k=0}^{\infty} \frac{(-1)^k}{(2k+1)},$$

$$x = 1 : \qquad \sum_{k=0}^{\infty} \frac{(-1)^k \cdot 1^{2k+1}}{(2k+1)} = \sum_{k=0}^{\infty} \frac{(-1)^k}{(2k+1)}.$$

Obě tyto výsledné alternující řady prověříme pomocí Leibnizova kritéria. Zkontrolujeme, zda vyhovují základním podmínkám:

- $a_n > 0, \forall n \in \mathbb{N}$,
- $\lim_{n\to\infty} a_n = 0$,
- a_n musí být nerostoucí.

Jelikož obě řady splňují všechna kritéria, konvergují relativně. Z tohoto důvodu je můžeme zahrnout do oboru konvergence, který je $\langle -1, 1 \rangle$.

1.2.3. Výpočet funkčních hodnot

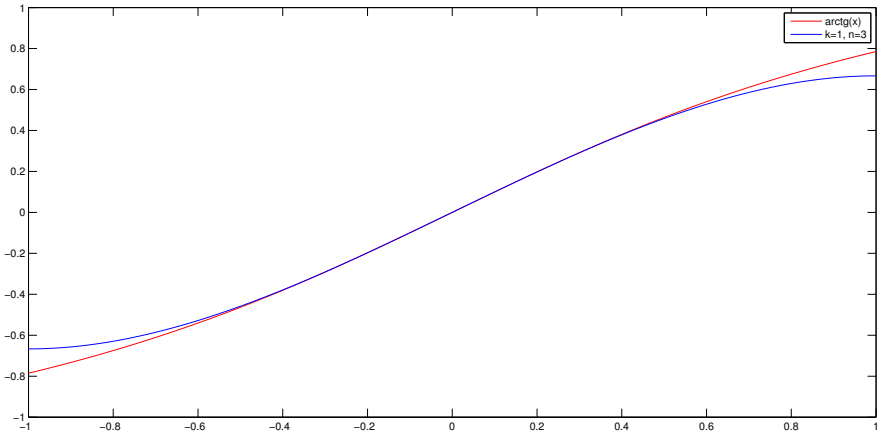
Pro výpočet hodnot Maclaurinova polynomu řádu n můžeme využít následujícího předpisu, jenž sečte všechny jeho členy až do n-tého řádu:

```
function v= tay(x,n)
for i=0:n
    c(i+1)=((-1)^i*x^(2*i+1)/(2*i+1));
    v=sum(c);
end;
end;
```

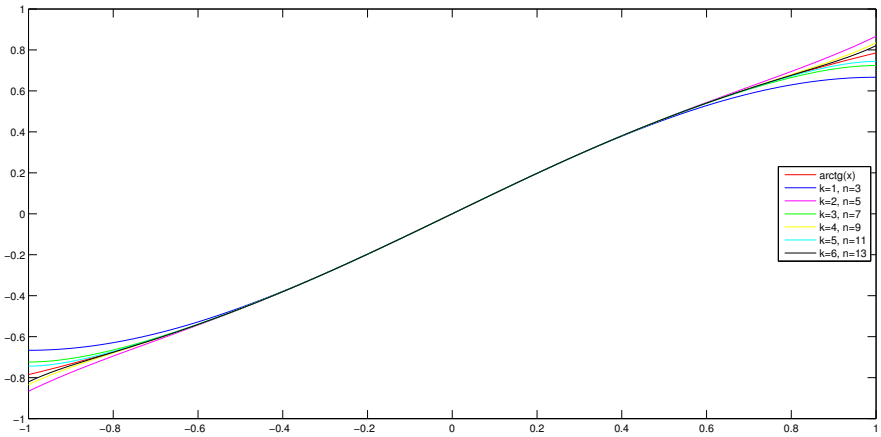
Tento předpis nám však spočítá pouze hodnotu v jednom bodě (pravděpodobně s velmi značnou nepřesností, závislou na parametru n). Abychom si však dokázali představit, jak výsledná aproximovaná funkce vypadá, použijeme níže uvedený předpis, jenž nám zobrazí červeně graf funkce arkustangens (hodnoty spočítané pomocí MATLABu) a také námi spočítané hodnoty Maclaurinova polynomu řádu 1 na intervalu $\langle -1, 1\rangle$ s krokem $0,001$:

```
function v= tayll(n)
t=[-1:0.001:1];
v= zeros(size(t));
for i=1:length(t);
    result = tay(t(i),n);
    v(i)=result;
end;
plot(t,atan(t),'r')
hold on
plot(t,v,'b');
```

Na Obrázku 1. vidíme, že hodnoty blízko nuly jsou velmi dobře aproximované, což se nedá říct o hodnotách blízko krajních bodů $\{-1, 1\}$. Jak si však můžeme všimnout na Obrázku 2., derivacemi vyšších řádů získáme přesnější hodnoty:



Obrázek 1: Arkustangens Taylorova polynomu řádu 3



Obrázek 2: Arkustangens Taylorova polynomu řádu 13

Tyto aproximované hodnoty se budou rovnat hodnotám funkce $\operatorname{arctg}(x)$ pouze pro $\lim_{n\to\infty} T_n^{f,a}(x)$. Pro všechna $n < \infty$ budou hodnoty Taylorova polynomu vycházet vždy s nějakou chybou vyjádřenou jako $R_n^{f,a}(x)$, pro kterou platí:

$$\lim_{n\to\infty} R_n^{f,a}(x) = 0.$$

1.2.4. Taylorova věta

Pro určení velikosti chyby, které jsme se dopustili předčasným ukončením Taylorova rozvoje, použijeme Taylorovu větu:

Věta 1. *Nechť má funkce f v okolí bodu a vlastní derivace až do řádu $(n+1)$. Pak pro všechna $x \neq a$ z tohoto okolí platí:*

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + ... + \frac{f^{(n)}(a)}{n!}(x-a)^n + R_{n+1}^{f,a}(x).$$

Chyba $R_{n+1}^{f,a}(x)$ se nazývá zbytek a funkce $f(x)$ se nazývá Taylorův vzorec. Nechť je funkce $\varphi(t)$ spojitá na intervalu (x,a) a má v každém jeho vnitřním bodě derivaci různou od nuly, potom existuje uvnitř tohoto intervalu bod ξ závislý na a, x, n a na tvaru funkcí f, φ takový, že:

$$R_{n+1}^{f,a}(x) = \frac{(x-\xi)^n}{n!} \cdot \frac{\varphi(x) - \varphi(a)}{\varphi'(\xi)} \cdot f^{(n+1)}(\xi).$$

Volba $\varphi(t) = (x-t)^n + 1$ dává speciálně

$$R_{n+1}^{f,a}(x) = \frac{(x-a)^{n+1}}{(n+1)!} \cdot f^{(n+1)}(\xi)$$

takzvaný Lagrangeův tvar. Volba $\varphi(t) = t$ dává

$$R_{n+1}^{f,a}(x) = \frac{(x-\xi)^n \cdot (x-a)}{n!} \cdot f^{(n+1)}(\xi)$$

takzvaný Cauchyův tvar.

Poznámka 1. *Číslo x je od počátku dáno, vystupuje zde tedy jako konstanta, proto jsme proměnnou ve funkci φ značili t.*

Důkaz: viz [1].

Přesnou hodnotu chyby nejsme schopni jednoznačně spočítat. Můžeme však pomoci Taylorova vzorce určit maximální hranici, kterou chyba nemůže překročit, tedy zjistit, o kolik se maximálně může naše aproximace lišit od funkční hodnoty. To je pro nás dostačující pro určení přesnosti aproximace na námi předem stanovený počet desetinných míst.

Prozkoumáme bod $x = 1$, jelikož je nejvzdálenější od bodu $a = 0$, a tedy tvoří největší chybu. Proměnnou x tedy uvažujme jako konstantu. Jako bod ξ zvolíme ten bod z intervalu $(0, 1)$, pro který má Lagrangův tvar zbytku nejvyšší hodnotu. Uvažujme jej proto jako proměnnou. Pro následující výpočet uvažujme $R_{n+1}(\xi)$. Jelikož víme, že sudé derivace jsou rovny nule a že $T_n(x) = T_{2k+1}(x) = T_{2k+2}(x)$, prověříme pouze liché derivace, pro něž platí $f^{(2k+1)}(0) = (2k)!$.

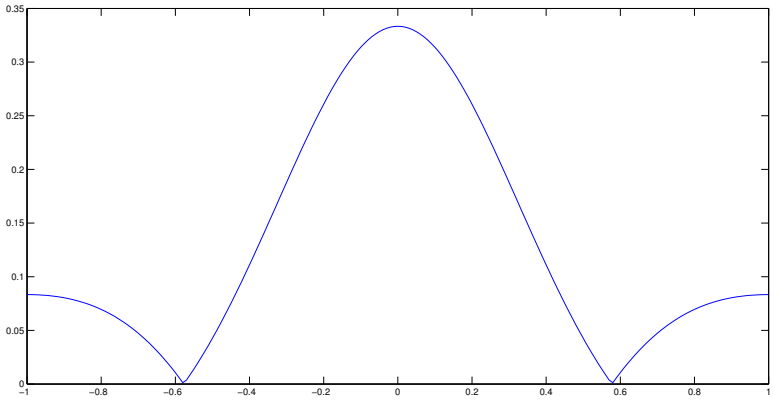
Tuto nejvyšší hodnotu zjistíme tak, že zderivujeme funkci $R_{n+1}(\xi)$ podle proměnné ξ. Protože hledáme absolutní hodnotu této funkce, stačí nám najít lokální extrémy a nemusíme zjišťovat, zda-li jde o lokální maximum, nebo lokální minimum. Položením derivace rovno nule získáme:

$$\frac{\partial R_{n+1}(\xi)}{\partial \xi} = \left(\frac{f^{(n+1)}(\xi)}{(n+1)!}(x-a)^{n+1} \right)' = (n+1) \cdot f^{(n+2)}(\xi) \frac{(1-0)^{n+1}}{(n+1)!} = 0.$$

Jelikož se jedná o Maclaurinovu řadu a víme, že její sudé derivace funkce $\operatorname{arctg}(x)$ v bodě $x = 0$ jsou nulové, získáme jediný lokální extrém v bodě $\xi = 0$. Protože bod 0 do intervalu nepatří, výsledná chyba bude pouze menší než, nikoliv však rovna zjištěné hodnotě.

Pro ověření správnosti výpočtu můžeme využít matematického programu MATLAB, který nám tuto hodnotu vykreslí graficky pro $k = 1$, tedy $n = 3$ pomocí následujícího předpisu:

```
x=[-1:0.01:1];
v = zeros(size(x));
for i=1:length(x);              %třetí derivace/(2k+1)!
    result=((-2+6*x(i)^2)/(1+x(i)^2)^3)/FACTORIAL(3);
    c(i)=result;
    v=plot(x,abs(c));           %absolutní hodnota chyby v bodě c
end;
```



Obrázek 3: Absolutní hodnota chyby třetí derivace funkce $\mathrm{arctg}(x)$

1.2.5. Odhad velikosti chyby

Využijeme Taylorova vzorce a Lagrangeova tvaru zbytku pro stupeň Taylorova polynomu $2k+2$:

$$\mathrm{arctg}(x) = \sum_{k=0}^{n} \frac{(-1)^k \cdot x^{2k+1}}{2k+1} + \frac{f^{(2k+3)}(\xi) \cdot (x)^{2k+3}}{(2k+3)!} \Rightarrow R_{2k+2}(x) = \frac{f^{(2k+3)}(\xi)}{(2k+3)!}.$$

Jelikož jsme zjistili, že velikost chyby pro $\xi \in (0,1)$ je největší pro $\lim_{\xi \to 0} f^{(2k+3)}(\xi)$, můžeme chybu spočítat:

$$\mathrm{arctg}(x) = \sum_{k=0}^{n} \frac{(-1)^k \cdot x^{2k+1}}{2k+1} + \frac{f^{(2k+3)}(0) \cdot (x)^{2k+3}}{(2k+3)!} \Rightarrow R_{2k+2}(x) < \frac{f^{(2k+3)}(0)}{(2k+3)!}.$$

Tyto hodnoty jsou uvedeny v následující tabulce.

k	n	Lagrangův tvar chyby
0	3	0,333333333333333
1	5	0,200000000000000
2	7	0,142857142857143
3	9	0,111111111111111
4	11	0,090909090909091
5	13	0,076923076923077
6	15	0,066666666666667
7	17	0,058823529411765
8	19	0,052631578947368
9	21	0,047619047619048
⋮	⋮	⋮

Hodnota k představuje pořadí členu Maclaurinova polynomu a n představuje stupeň derivace. Tedy $n = 2k + 3$.

Jak vidíme, tak řada konverguje velmi pomalu. Abychom dosáhli přesného výpočtu arctg(1), např. na šest desetinných míst, museli bychom spočítat obrovské množství členů v řádech stovek tisíc, až milionů derivací, což je výpočetně velmi náročné. To však není vůbec nutné, jak si ukážeme v následující části.

1.2.6. Van Wijngaardenova transformace

Jelikož víme, že jde o alternující řadu, můžeme využít Van Wijngaardenovi transformace. Nejprve si nadefinujeme posloupnost částečných součtů:

$$s_{0,0} = 1, s_{0,1} = 1 - \frac{1}{3}, s_{0,2} = 1 - \frac{1}{3} + \frac{1}{5}, s_{0,3} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7}, \ldots$$

a následně posloupnost $s_{j+1,k} = \frac{s_{j,k}+s_{j,k+1}}{2}$ zobrazenou v tabulce:

$s_{j,k}$	0	1	2	3	4	5	6
0	1,0000	0,6667	0,8667	0,7238	0,8349	0,7440	0,8209
1	0,8333	0,7667	0,7952	0,7794	0,7895	0,7825	
2	0,8000	0,7810	0,7873	0,7844	0,7860		
3	0,7905	0,7841	0,7859	0,7852			
4	0,7873	0,7850	0,7855				
5	0,7861	0,7853					
6	0,7857						

První sloupec $s_{j,0}$ obsahuje částečné součty Eulerovi transformace. Adriaan Van Wijngaarden však dokázal, že není potřeba provést tento postup až do samotného konce, ale mnohem výhodnější je ukončit jej ve dvou třetinách. Tedy jsou-li definovány $s_{0,0}, s_{0,1}, \ldots, s_{0,6}$, pak $s_{4,2}$ vyjadřuje téměř vždy lepší aproximaci součtu než $s_{6,0}$. Použijeme-li tuto myšlenku pro hodnoty $j, k \in \{0, 1, \ldots, 15\}$, dostaneme hodnotu $s_{10,5} = 0,7853981590250$, která nám určuje hodnotu arctg(1) s přesností na 8 desetinných míst.

Pro výpočet hodnot funkcí $\sin(\alpha)$ a $\cos(\alpha)$ pomocí algoritmu CORDIC potřebujeme spočítat hodnoty $\text{arctg}(2^{-i})$ pro $i = \{0, 1, ..., 59\}$. Výpočet hodnoty $\text{arctg}(2^{-1})$, která je druhá nejvyšší (všechny ostatní jsou blíže bodu $a = 0$, a jsou lépe aproximovány, tedy přesnější) s přesností na 8 desetinných míst, je však už snazší a stačí nám sečíst pouze několik členů Maclaurinova polynomu. Kolik jich bude, zjistíme pomocí obecného tvaru vzorce:

$$|R_{2k+2}(x)| < \left|\frac{f^{(2k+3)}(0) \cdot x^{2k+3}}{(2k+3)!}\right| = \left|\frac{(2k+2)! \cdot x^{2k+3}}{(2k+3)!}\right| = \left|\frac{x^{2k+3}}{2k+3}\right|,$$

Jelikož $n = 2k + 3$, získáme po dosazení $x = 0,5$, $x = 0,25$ a $x = 0,125$ nejen velikost chyby, ale také počet členů:

$$|R_{2k+2}(0,5)| < \left|\frac{f^{(2k+3)}(0) \cdot x^{2k+3}}{(2k+3)!}\right| = \left|\frac{0,5^{23}}{23}\right| := 5.183012589164402 \cdot 10^{-9} \Rightarrow k = 10$$

$$|R_{2k+2}(0,25)| < \left|\frac{f^{(2k+3)}(0) \cdot x^{2k+3}}{(2k+3)!}\right| = \left|\frac{0,25^{13}}{13}\right| := 1.146243168757512 \cdot 10^{-9} \Rightarrow k = 5$$

$$|R_{2k+2}(0,125)| < \left|\frac{f^{(2k+3)}(0) \cdot x^{2k+3}}{(2k+3)!}\right| = \left|\frac{0,125^{9}}{9}\right| := 8.278422885470920 \cdot 10^{-10} \Rightarrow k = 3$$

Pro kontrolu použijeme námi vytvořený předpis pro výpočet funkčních hodnot arkustangens a porovnáme je s hodnotami spočítanými pomocí MATLABu.

x	Taylorův polynom	MATLAB	Rozdíl
0,5	0,463647613215610	0,463647609000806	0,000000004214804
0,25	0,244978662039466	0,244978663126864	0,000000001087398
0,125	0,124354993729364	0,124354994546761	0,000000000817397

Hodnoty arctg(x), pro účel algoritmu CORDIC, lze také spočítat pomocí MATLABu následujícím předpisem doplněným o tabulku s výslednými hodnotami:

```
function v= tayl(n) %vstupním parametrem je počet členů polynomu
j=1:31; t=[2.^-j]; v= zeros(size(t));
for i=1:length(t);
    result = tay(t(i),n);
    v(i)=result;
end;
```

i	arctg(2^{-i})	i	arctg(2^{-i})	i	arctg(2^{-i})	i	arctg(2^{-i})
0	0,78539816	8	0,00390623	16	0,00001526	24	0,00000006
1	0,46364761	9	0,00195312	17	0,00000763	25	0,00000003
2	0,24497866	10	0,00097656	18	0,00000381	26	0,00000001
3	0,12435499	11	0,00048828	19	0,00000191	27	0,00000001
4	0,06241881	12	0,00024414	20	0,00000095	28	0,00000000
5	0,03123983	13	0,00012207	21	0,00000048	29	0,00000000
6	0,01562373	14	0,00006104	22	0,00000024	30	0,00000000
7	0,00781234	15	0,00003052	23	0,00000012	31	0,00000000

Hodnoty arctg(2^{-i}) $\leq 10^{-9}$ pro $i \geq 28$, tedy pro naše potřeby zanedbatelné.

1.3. Exponencionální funkce

Nyní použijeme všech poznatků z první kapitoly a aplikujeme je pro výpočet funkčních hodnot exponencionální funkce e^x pomocí Maclaurinovy řady. Hodnota e se nazývá Eulerovo číslo a je nejpřirozenějším základem exponencionální funkce. Platí pro ni vztah $\left(1+\frac{1}{n}\right)^n < e < \left(1+\frac{1}{n}\right)^{n+1}$, pro $\forall n \in \mathbb{N}$. Speciálně pro $n=1$ získáme $2 < e < 4$. Této vlastnosti využijeme v kapitole 1.3.3. Dále o této funkci víme, že jejím definičním oborem jsou všechna reálná čísla, tedy $D(e^x) = \mathbb{R}$, a že má derivace všech řádů pro všechna x, proto můžeme využít obecného tvaru Taylorovy řady:

$$f(x) = f(a)+\frac{f'(a)}{1!}(x-a)+\frac{f''(a)}{2!}(x-a)^2+\frac{f'''(a)}{3!}(x-a)^3+\ldots = \sum_{k=0}^{\infty}\frac{f^{(k)}(a)}{k!}(x-a)^k.$$

Jelikož všechny derivace funkce e^x jsou opět rovny e^x a prověřujeme derivace v bodě $a=0$, tedy $f^{(k)}(a) \to (e^a)' = e^a = e^0 = 1$, získáme následující tvar Maclaurinovy řady:

$$f(x) = f(0)+\frac{f'(0)}{1!}(x-0)+\frac{f''(0)}{2!}(x-0)^2+\frac{f'''(0)}{3!}(x-0)^3+\ldots = \sum_{k=0}^{\infty}\frac{f^{(k)}(0)}{k!}(x-0)^k,$$

$$f(x) = 1+\frac{1}{1!}x+\frac{1}{2!}x^2+\frac{1}{3!}x^3+\ldots = \sum_{k=0}^{\infty}\frac{x^k}{k!},$$

případně Maclaurinův polynom řádu n:

$$f(x) = 1+\frac{1}{1!}x+\frac{1}{2!}x^2+\frac{1}{3!}x^3+\ldots+\frac{1}{n!}x^n = \sum_{k=0}^{n}\frac{x^k}{k!},$$

a také Taylorovu větu vyjádřenou pomocí Lagrangova tvaru zbytku jako:

$$f(x) = T_n^{f,a}(x) + R_{n+1}^{f,a}(x),$$

$$f(x) = \sum_{k=0}^{n}\frac{f^{(k)}(a)}{k!}(x-a)^k + \frac{f^{(n+1)}(\xi)}{(n+1)!}(x-a)^{n+1},$$

$$f(x) = \sum_{k=0}^{n}\frac{x^k}{k!} + \frac{e^{\xi}}{(n+1)!}x^{n+1}.$$

1.3.1. Obor konvergence

Nyní určíme obor konvergence nově vzniklé řady $\sum_{k=0}^{\infty} \frac{x^k}{k!}$ se středem v bodě nula. Nejprve určíme n-tý člen posloupnosti $b_k = \frac{1}{k!}$, pomocí něhož spočítáme poloměr konvergence $r = \frac{1}{\lambda}$, kde λ odpovídá limitě:

$$\lambda = \lim_{k\to\infty} \left| \frac{b_{k+1}}{b_k} \right| = \lim_{k\to\infty} \left| \frac{\frac{1}{(k+1)!}}{\frac{1}{k!}} \right| = \lim_{k\to\infty} \left| \frac{1}{k+1} \right| \quad \Rightarrow \quad r = \frac{1}{\lambda} = \lim_{k\to\infty} \left| \frac{k+1}{1} \right| = \infty,$$

což znamená, že funkce e^x konverguje pro $\forall x \in \mathbb{R}$.

1.3.2. Hornerovo schéma

V této kapitole si ukážeme, jak jednoduše upravit výpočet hodnoty polynomu, aby byl přehlednější a pohodlnější, užitím Hornerova schémata. Mějme např. polynom:

$$P_5(x) = 1 + \frac{1}{1!}x + \frac{1}{2!}x^2 + \frac{1}{3!}x^3 + \frac{1}{4!}x^4 + \frac{1}{5!}x^5,$$

ve kterém chceme zjistit jeho hodnotu $x = 2$. Nejprve seřadíme jeho členy sestupně od nejvyšší mocniny po nejnižší:

$$P_5(x) = \frac{1}{5!}x^5 + \frac{1}{4!}x^4 + \frac{1}{3!}x^3 + \frac{1}{2!}x^2 + \frac{1}{1!}x + 1.$$

Výpočet můžeme provést buď přímým dosazením $x = 2$ do pravé strany rovnice, tedy:

$$P_5(2) = \frac{1}{5!}2^5 + \frac{1}{4!}2^4 + \frac{1}{3!}2^3 + \frac{1}{2!}2^2 + \frac{1}{1!}2 + 1,$$

$$P_5(2) = \frac{32}{120} + \frac{16}{24} + \frac{8}{6} + \frac{4}{2} + \frac{2}{1} + 1,$$

$$P_5(2) = \frac{4}{15} + \frac{2}{3} + \frac{4}{3} + 2 + 2 + 1 \cong 7,2667.$$

Další možností je upravit polynom $P_5(x)$ na tvar:

$$\left\langle \left\{ \left[\left(\frac{1}{5!}x + \frac{1}{4!} \right) x + \frac{1}{3!} \right] x + \frac{1}{2!} \right\} x + \frac{1}{1!} \right\rangle x + 1$$

a počítat postupně výrazy v jednotlivých závorkách (výsledky zaokrouhlené na 4 desetinná místa):

$$\frac{2}{5!} + \frac{1}{4!} = 0,0583; \quad 0,0583 \cdot 2 + \frac{1}{3!} = 0,2833; \quad 0,2833 \cdot 2 + \frac{1}{2!} = 1,0667;$$

$$1,0667 \cdot 2 + 1 = 3,1333; \quad 3,1333 \cdot 2 + 1 = 7,2667.$$

Celkový výpočet můžeme zapsat přehledněji do tabulky,

$P_5(x)$	$0,0083$	$0,0417$	$0,1667$	$0,5$	1	1
$x = 2$	$0,0083$	$0,0583$	$0,2833$	$1,0667$	$3,1333$	$7,2667$

kde v prvním řádku jsou koeficienty sestupně uspořádaného polynomu. Ve druhém řádku je v první buňce hodnota argumentu a ve druhé buňce hodnota rovna druhé buňce prvního řádku (tj. $\frac{1}{5!} = 0,0083$). Hodnotu každé další buňky druhého řádku (např.: $0,2833$) získáme z předchozí buňky (tj. $0,0583$), vynásobíme-li ji hodnotou argumentu x (tj. $2 \cdot 0,0583 = 0,1166$) a následně k ní přičteme hodnotu prvního řádku nejblíže vpravo (tj. $0,1166 + 0,1667 = 0,2833$). Poslední buňka druhého řádku nám dává hodnotu polynomu $P_5(x)$.

V obecném případu polynomu

$$P_n(x) = a_0 \cdot x^n + a_1 \cdot x^{n-1} + a_2 \cdot x^{n-2} + \ldots + a_{n-1} \cdot x + a_n, \qquad a_0 \neq 0,$$

bude Hornerovo schéma pro výpočet hodnoty $P_n(x)$ vyjádřeno ve tvaru,

$P_n(x)$	a_0	a_1	a_2	$\ldots$	$a_n + 1$	a_n
x	b_0	b_1	b_2	$\ldots$	$b_n + 1$	b_n

kde

$$b_0 = a_0, \quad b_1 = b_0 \cdot x + a_1, \quad b_2 = b_1 \cdot x + a_2, \quad \ldots, \quad b_n = b_{n-1} \cdot x + a_n = P_n(x).$$

Námi spočítaná hodnota $P_5(2)$ je rovna hodnotě $T_n^{f,a}(x) = T_5^{e^x,0}(2)$.

Pro výpočet můžeme také využít následujícího předpisu:

```
function v = expo(x,n)        %x = argument; n=řád polynomu
for i=0:n
    c(i+1)=(x^i)/(factorial(i));
    v=sum(c);
end
end
```

Vyvoláním funkce `expo(2,5)`; získáme opět výsledek $7,266666666667$.

1.3.3. Odhad velikosti chyby

Pro výpočet hodnoty e^2 pomocí Maclaurinova polynomu však musíme spočítat ještě chybu $R_{n+1}^{f,a}(x) = R_6^{e^x,0}(2)$, které jsme se dopustili předčasným ukončením Maclaurinova rozvoje. K tomu využijeme Lagrangeova tvaru zbytku:

$$R_{n+1}^{f,a}(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}(x-a)^{n+1}, \qquad \forall n \in \mathbb{N}, \forall x \in \mathbb{R}, \quad kde\, |\xi| < |x|\,.$$

Navíc využijeme vlastnosti, že

$$e^x = e^{b^k} = e^{\overbrace{b+\ldots+b}^{k}} = \overbrace{e^b \cdot \ldots \cdot e^b}^{k},$$

tj. speciálně pro $x = 2$ získáme:

$$e^2 = e^{2\cdot 1} = e^{1+1} = e^1 \cdot e^1,$$

z čehož plyne, že nám stačí spočítat pouze hodnoty e^b pro $b \in \langle -1, 0) \cup (0, 1\rangle$. Pomocí nich můžeme následně spočítat hodnoty e^x pro $\forall x \in \mathbb{R}$.

Abychom nezapomněli na závislost ξ na x a n, budeme používat značení $\xi \equiv \xi_{x,n}$. Je-li $0 < b \leq 1$, je též $\xi_{b,n} < 1$, tedy $e^{\xi_{b,n}} < e^1$. Využitím vlastnosti $e < 4$ z kapitoly 1.3. spočítáme velikost chyby:

$$0 < \frac{e^{\xi_{b,n}} \cdot b^{n+1}}{(n+1)!} < \frac{e^1 \cdot b^{n+1}}{(n+1)!} < \frac{4 \cdot b^{n+1}}{(n+1)!} \leq \frac{4}{(n+1)!}.$$

Z čehož plyne, že nahrazením čísla e^b číslem $1+\frac{b}{1!}+\frac{b^2}{2!}+\ldots+\frac{b^n}{n!}$ jsme se dopustili chyby menší než $\frac{4\cdot b^{n+1}}{(n+1)!}$, což nikdy nebude větší než $\frac{4}{(n+1)!}$. Je-li $-1 \leq b < 0$, je výpočet ještě přiznivější, protože $\xi_{b,n} < 0$, tedy $e^{\xi_{b,n}} < e^0 = 1$ a proto:

$$0 < \left| \frac{e^{\xi_{b,n}} \cdot b^{n+1}}{(n+1)!} \right| < \frac{e^0 \cdot |b|^{n+1}}{(n+1)!} < \frac{|b|^{n+1}}{(n+1)!} \leq \frac{1}{(n+1)!}.$$

Počet kroků (n) potřebný k dosažení námi zvolené maximální chyby $\frac{4}{(n+1)!}$ (resp. $\frac{1}{(n+1)!}$) pro $b \in (0,1\rangle$ (resp. $\langle -1,0)$) zjistíme z následující tabulky:

n	$(n+1)!$	$\frac{4}{(n+1)!}$	$\frac{1}{(n+1)!}$
0	1	4,00000000000000000000	1,00000000000000000000
1	2	2,00000000000000000000	0,50000000000000000000
2	6	0,66666666666666700000	0,16666666666666700000
3	24	0,16666666666666700000	0,04166666666666670000
4	120	0,03333333333333330000	0,00833333333333333000
5	720	0,00555555555555556000	0,00138888888888889000
6	5040	0,00079365079365079400	0,00019841269841269800
7	40320	0,00009920634920634920	0,00002480158730158730
8	362880	0,00001102292768959440	0,00000275573192239859
9	3628800	0,00000110229276895944	0,00000027557319223986
10	39916800	0,00000010020843354177	0,00000002505210838544
11	479001600	0,00000000835070279515	0,00000000208767569879
12	6227020800	0,00000000064236175347	0,00000000016059043837
13	87178291200	0,00000000004588298239	0,00000000001147074560
14	1307674368000	0,00000000000305886549	0,00000000000076471637
15	20922789888000	0,00000000000019117909	0,00000000000004779477
16	355687428096000	0,00000000000001124583	0,00000000000000281146
17	6402373705728000	0,00000000000000062477	0,00000000000000015619
18	121645100408832000	0,00000000000000003288	0,00000000000000000822
19	2432902008176640000	0,00000000000000000164	0,00000000000000000041
20	51090942171709400000	0,00000000000000000008	0,00000000000000000002

Pro výpočet hodnot e^b pro $b \in \langle -1,0) \cup (0,1\rangle$ si nadefinujme tabulku pro kladné a záporné hodnoty 10^{-r} a $e^{10^{-r}}$ s chybou menší než $10^{-9} \Leftrightarrow (n = 11)$. Obě tyto hodnoty se pro $\lim_{r\to\infty}$ blíží nule:

r	10^{-r}	$e^{10^{-r}}$	$-(10^{-r})$	$e^{-(10^{-r})}$
0	1,0000000000	2,7182818262	-1,0000000000	0,3678794392
1	0,1000000000	1,1051709181	-0,1000000000	0,9048374180
2	0,0100000000	1,0100501671	-0,0100000000	0,9900498337
3	0,0010000000	1,0010005002	-0,0010000000	0,9990004998
4	0,0001000000	1,0001000050	-0,0001000000	0,9999000050
5	0,0000100000	1,0000100001	-0,0000100000	0,9999900001
6	0,0000010000	1,0000010000	-0,0000010000	0,9999990000
7	0,0000001000	1,0000001000	-0,0000001000	0,9999999000
8	0,0000000100	1,0000000100	-0,0000000100	0,9999999900
9	0,0000000010	1,0000000010	-0,0000000010	0,9999999990
10	0,0000000001	1,0000000001	-0,0000000001	0,9999999999

Např.: hodnotu $e^{2,13024}$ tedy můžeme spočítat jako:

$$e^2 \cdot e^{0,1} \cdot e^{0,03} \cdot e^{0,0002} \cdot e^{0,00004} =$$

$$= e^1 \cdot e^1 \cdot e^{0,1} \cdot e^{0,01} \cdot e^{0,01} \cdot e^{0,01} \cdot e^{0,0001} \cdot e^{0,0001} \cdot e^{0,00001} \cdot e^{0,00001} \cdot e^{0,00001} \cdot e^{0,00001} =$$

$$= 2,7182818262 \cdot 2,7182818262 \cdot 1,1051709181 \cdot \ldots \cdot 1,0000100001 \cong$$

$$\cong 8,4168866078,$$

čímž jsme se dopustili chyby menší než:

$$0 < \frac{e^{2,13024 \cdot \xi_{b,n}} \cdot b^{n+1}}{(n+1)!} < \frac{e^{2,13024} \cdot b^{n+1}}{(n+1)!} < \frac{4^{2,13024} \cdot b^{n+1}}{(n+1)!} \leq \frac{4^{2,13024}}{12!} \leq 0,0000000400124.$$

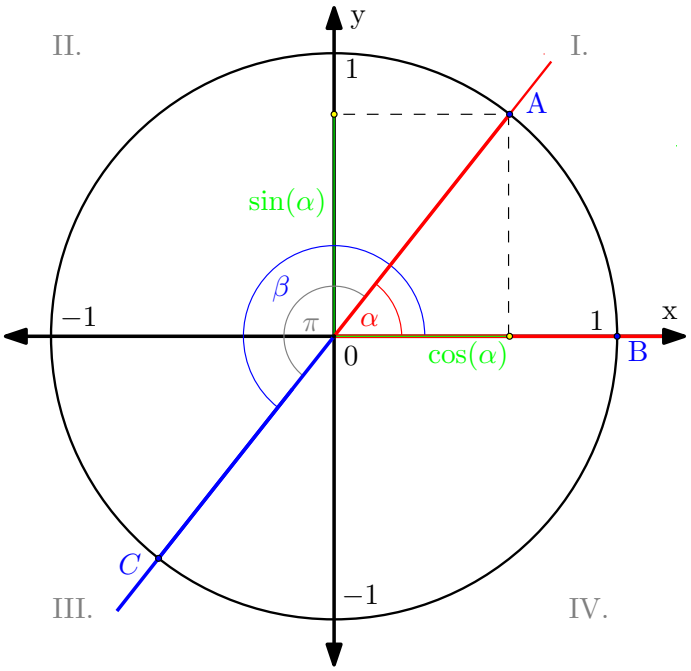
2. Cordic

Algoritmus CORDIC se používá k výpočtu funkčních hodnot elementárních funkcí sinus a kosinus. Informace pro tuto kapitolu jsem čerpal z [7], [8], [9], [10]. Je několik způsobů, jak se dá na tento algoritmus nahlížet, pracovat s ním a využívat jeho vlastností. Ve stručnosti CORDIC funguje tak, že použijeme několik zjednodušení pro snadnější výpočty. V praxi následně využíváme rotaci vektoru po jednotkové kružnici okolo počátku o předem stanovené úhly nabývající hodnot 2^{-i} pro $i \in \mathbb{N}_0$. Tyto hodnoty můžeme pro usnadnění zjistit velmi rychlým a jednoduchým bitovým posunem.

Velikou výhodou je, že tento algoritmus je variabilní vzhledem k náročnosti výpočtu, ale tím bohužel také k přesnosti výsledku. Můžeme použít malého počtu iterací (opakování určitého procesu), což bude velmi nenáročné a rychlé pro výpočet, ale na druhou stranu dostaneme nepřesný výsledek. Čím více iterací, tím přesnější výsledek, ale zároveň pomalejší průběh. V dnešní době, kdy 3GHz čtyřjádrové procesory, ani $4GB$ RAM operačních pamětí, nejsou žádnou zásadní výjimkou či přepychem, není problém počítat s velmi vysokým počtem iterací a zároveň dosáhnout výsledku během zlomku vteřiny, avšak při použití elementárních zařízení, jako jsou kalkulačky, které mají omezené relativně malé možnosti operační paměti, může být CORDIC vhodnou volbou.

Jak ale určit nejvhodnější počet iterací? Tento problém budu řešit následně v kapitole 2.5. Abychom však mohli CORDIC správně implementovat, je zapotřebí předem znát několik důležitých informací. Např.: Musíme znát a rozumět jednotkové kružnici a také umět v ní číst.

2.1. Jednotková kružnice



Obrázek 4: Jednotková kružnice

Jednotková kružnice má poloměr 1, střed v bodě S se souřadnicemi $[0, 0]$ a počátek v bodě B se souřadnicemi $[1, 0]$. Její rovnice je $x^2 + y^2 = 1$. Bod A je rotován o úhel α od počátku (B). Vzdálenost bodu A od osy x je rovna hodnotě $\sin(\alpha)$ a vzdálenost bodu A od osy y je rovna hodnotě $\cos(\alpha)$. Hodnoty v druhém (resp. třetím) kvadrantu jsou rovny záporným hodnotám ve čtvrtém (resp. prvním) kvadrantu. Např.:

$$\sin(\beta) = -\sin(\alpha).$$

2.2. Rotace vektoru okolo počátku

Nejprve použijeme vzorce pro výpočet součtu úhlů, čímž dostaneme:

$$\cos(\varphi + \delta) = \cos(\varphi) \cdot \cos(\delta) - \sin(\varphi) \cdot \sin(\delta),$$

$$\sin(\varphi + \delta) = \sin(\varphi) \cdot \cos(\delta) + \cos(\varphi) \cdot \sin(\delta),$$

kde φ je původní úhel a δ je velikost úhlu o který rotujeme.

Jednotlivé úhly si převedeme na polární souřadnice, jejichž hodnotu získáme pomocí:

$$x = r \cdot \cos(\varphi),$$

$$y = r \cdot \sin(\varphi),$$

kde r je poloměr kružnice, neboli vzdálenost bodu od středu kružnice, a φ je úhel, o který rotujeme.

V rámci zjednodušení budeme používat pouze jednotkovou kružnici ($r = 1$). Následně si můžeme určit souřadnice počátku, od kterého budeme provádět jednotlivé rotace, tedy:

$$x_0 = 1 \cdot \cos(0) = 1,$$

$$y_0 = 1 \cdot \sin(0) = 0.$$

Pro účel našich výpočtů v MATLABu budeme používat vektor počátku:

$$v = \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

2.3. Souřadnice rotovaného úhlu

Následně nám stačí pouze zjistit souřadnice počátečního úhlu φ rotovaného o úhel δ:

$$x_r = r \cdot \cos(\varphi + \delta),$$

$$y_r = r \cdot \sin(\varphi + \delta).$$

Použijeme-li vzorce pro součet úhlů, získáme:

$$x_r = r \cdot (\cos(\varphi) \cdot \cos(\delta) - \sin(\varphi) \cdot \sin(\delta)) = x_0 \cdot \cos(\delta) - y_0 \cdot \sin(\delta),$$

$$y_r = r \cdot (\sin(\varphi) \cdot \cos(\delta) - \cos(\varphi) \cdot \sin(\delta)) = x_0 \cdot \sin(\delta) - y_0 \cdot \cos(\delta).$$

Ve výsledných vztazích se nevyskytuje ani hodnota r, ani úhel původního vektoru φ. Z toho plyne, že převod na polární souřadnice byl pro nás pouze matematickou pomůckou při odvozování vzorce pro rotaci.

Pro účely algoritmu CORDIC se tento vztah dále upravuje. První úprava spočívá v tom, že obě rovnice vydělíme hodnotou $cos(\delta)$, takže dostaneme vztahy:

$$\frac{x_r}{\cos(\delta)} = \frac{x_0 \cdot \cos(\delta)}{\cos(\delta)} - \frac{y_0 \cdot \sin(\delta)}{\cos(\delta)},$$

$$\frac{y_r}{\cos(\delta)} = \frac{x_0 \cdot \sin(\delta)}{\cos(\delta)} - \frac{y_0 \cdot \cos(\delta)}{\cos(\delta)}.$$

Využitím vlastnosti, že $\frac{\sin(\delta)}{\cos(\delta)} = \mathrm{tg}(\delta)$, můžeme pokračovat v následujících úpravách:

$$\frac{x_r}{\cos(\delta)} = x_0 - y_0 \cdot \mathrm{tg}(\delta),$$

$$\frac{y_r}{\cos(\delta)} = x_0 \cdot \mathrm{tg}(\delta) - y_0,$$

což můžeme vynásobením $\cos(\delta)$ upravit na:

$$x_r = \cos(\delta) \cdot (x_0 - y_0 \cdot \mathrm{tg}(\delta)),$$

$$y_r = \cos(\delta) \cdot (y_0 + x_0 \cdot \mathrm{tg}(\delta)).$$

Nyní přichází hlavní myšlenka, na které je CORDIC postaven. Pokud budeme volit úhel δ tak, aby jeho tangenta nabývala hodnot 2^{-i} pro $i \in N_0$, je možné tangentu ve vzorci nahradit násobením zvolenou hodnotou 2^{-i}, navíc můžeme násobení nahradit jednoduchým a rychlým bitovým posunem.

Toto omezení však znamená, že vektor nemůžeme rotovat o libovolný úhel, ale pouze o úhel odpovídající tangentě z dané sady. Pro naše účely bude stačit pouze 28 iterací, tedy $i = \{0, 1, \ldots, 27\}$, protože hodnota $2^{-27} \leq 10^{-9}$.

Vzhledem k tomu, že

$$\mathrm{tg}(\delta) = 2^{-i} \quad \Rightarrow \quad \delta = \mathrm{arctg}(2^{-i}),$$

využijeme námi vypočítáné hodnoty funkce $\mathrm{arctg}(x)$ pomocí Taylorova polynomu ze strany 16. Tyto hodnoty jsou však přesné pouze na 8 desetinných míst a proto i výsledky algoritmu CORDIC budou přesné pouze na osm desetinných míst. Abychom dosáhli větší přesnosti, museli bychom spočítat více členů Taylorova polynomu! Nahrazením hodnot $\mathrm{tg}(\delta)$ hodnotami 2^{-i} ve vzorcích z předchozí strany získáme:

$$x_r = \cos(\delta) \cdot (x_0 - y_0 \cdot 2^{-i}),$$

$$y_r = \cos(\delta) \cdot (y_0 + x_0 \cdot 2^{-i}).$$

Jelikož však potřebujeme získat přesné hodnoty úhlu δ, a k tomu budeme využívat pouze rotace o předem stanovené úhly δ_i pro $i = \{0, 1, \ldots, 27\}$, musíme přidat i možnost, že úhly budeme nejen přičítat, ale také odečítat. Požadovaný úhel δ tak získáme sčítáním jednotlivých úhlu δ_i, tedy např.:

$$\delta = \delta_1 - \delta_2 + \delta_3 + \ldots \ .$$

K tomu je zapotřebí přidat nový parametr. Ten si můžeme označit jako S, který bude nabývat pouze hodnot $\{-1, 1\}$, proto:

$$x_r = \cos(\delta) \cdot (x_0 - y_0 \cdot S \cdot 2^{-i}),$$

$$y_r = \cos(\delta) \cdot (y_0 + x_0 \cdot S \cdot 2^{-i}).$$

Nyní je naším jediným problémem $\cos(\delta)$, jelikož jeho hodnoty neznáme. Využijeme však toho, že funkce kosinus je symetrická a platí:

$$\cos(\delta) = \cos(-\delta),$$

a navíc nepotřebujeme rotovat o všechny úhly, ale pouze o námi předem stanovené úhly δ_i. Na základě těchto vlastností můžeme nahradit $\cos(\delta)$ konstantou Z_i:

$$x_r = Z_i \cdot (x_0 - y_0 \cdot S \cdot 2^{-i}),$$

$$y_r = Z_i \cdot (y_0 + x_0 \cdot S \cdot 2^{-i}).$$

Hodnoty, kterých tato konstanta nabývá, zjistíme pomocí následujících rovnic:

$$Z_i = \cos(\delta),$$

$$\mathrm{tg}(\delta) = 2^{-i} \quad \Longrightarrow \quad \delta = \mathrm{arctg}(2^{-i}) \quad \Longrightarrow \quad Z_i = \cos(\mathrm{arctg}(2^{-i})).$$

Následně můžeme ze vztahu:

$$\cos(\delta) = \frac{1}{\sqrt{1 + \mathrm{tg}^2(\delta)}}$$

vyjádřit hodnotu konstanty Z_i pomocí:

$$Z_i = \cos(\mathrm{arctg}(2^{-i})) = \frac{1}{\sqrt{1 + \mathrm{tg}^2(\mathrm{arctg}(2^{-i})}} = \frac{1}{\sqrt{1 + 2^{(-2i)}}}.$$

Tímto jsme upravili původní vzorce na:

$$x_r = \frac{1}{\sqrt{1 + 2^{(-2i)}}} \cdot (x_0 - y_0 \cdot S \cdot 2^{-i}),$$

$$y_r = \frac{1}{\sqrt{1 + 2^{(-2i)}}} \cdot (y_0 + x_0 \cdot S \cdot 2^{-i}),$$

což jsou naše výsledné rovnice, s nimiž budeme pracovat. Pro potřeby MATLABu si je však můžeme převést na matice:

$$\begin{pmatrix} x_r \\ y_r \end{pmatrix} = Z_i \cdot \begin{pmatrix} 1 & -S_i \cdot 2^{-i} \\ S_i \cdot 2^{-i} & 1 \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ y_0 \end{pmatrix},$$

$$\begin{pmatrix} x_r \\ y_r \end{pmatrix} = \frac{1}{\sqrt{1 + 2^{(-2i)}}} \cdot \begin{pmatrix} 1 & -S_i \cdot 2^{-i} \\ S_i \cdot 2^{-i} & 1 \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ y_0 \end{pmatrix}.$$

Nyní však potřebujeme zjistit hodnoty Z_i a také součiny těchto hodnot označených Z_n. Pro jejich výpočet jsem využil následujícího předpisu:

```
format long              %zobrazíme výsledky na 15 desetinných míst
n=26;                    %počet iterací
i=0:n-1;
t=1;                     %rozsah součinu konstanty Zn
Zi=1./sqrt(1+2.^(-2*i));
for y=1:n;
  Zn(y)=prod(Zi(1:t));  %funkce prod vynásobí všechny členy od 1 do n
   if t < n;            %v každé smyčce zvýší rozsah součinu Zn o 1,
    t=t+1;              %dokud nespočítá konstantu Zn rozsahu n
   end
end
disp([i',Zi',Zn']);
```

i	hodnoty Zi	hodnoty Zn
0	0,707106781186547	0,707106781186547
1	0,894427190999916	0,632455532033676
2	0,970142500145332	0,613571991077896
3	0,992277876713668	0,608833912517752
4	0,998052578482889	0,607648256256168
⋮	⋮	⋮
20	0,999999999999545	0,607252935008973
21	0,999999999999886	0,607252935008904
22	0,999999999999972	0,607252935008887
23	0,999999999999993	0,607252935008883
24	0,999999999999998	0,607252935008882
25	1,000000000000000	0,607252935008881

Nyní již máme veškeré informace k tomu, abychom mohli sestavit algoritmus CORDIC a vypočítat pomocí něj funkční hodnoty funkcí $\cos(\varphi)$ a $\sin(\varphi)$.

2.4. Sestavení předpisu pro algoritmus CORDIC

Nejprve převedeme úhly z druhého a třetího kvadrantu do prvního a čtvrtého s příslušnými zápornými znaménky:

```
function vektor = cordic(delta,n) %vstupní argumenty jsou úhel a počet
                                  %iterací, které chceme provést
if delta < -pi/2 || delta > pi/2
    if delta < 0
        vektor = cordic(delta + pi, n);
    else
        vektor = cordic(delta - pi, n);
    end
    vektor = -vektor;
    return
end
```

Následně si nadefinujeme jednotlivé parametry potřebné k výpočtu, jako jsou úhly, o které budeme rotovat, hodnoty konstanty Z_i, počáteční vektor v a konstantu pro dělení dvěma, neboli náš zmiňovaný bitový posun.

```
uhly = [ ...
0.78539816 0.46364761 0.24497866 0.12435499
0.06241881 0.03123983 0.01562373 0.00781234
0.00390623 0.00195312 0.00097656 0.00048828
0.00024414 0.00012207 0.00006104 0.00003052
0.00001526 0.00000763 0.00000381 0.00000191
0.00000095 0.00000048 0.00000024 0.00000012
0.00000006 0.00000003 0.00000001 0.00000001];
i=0:n-1;
t=1;
Zi=1./sqrt(1+2.^(-2*i));
for y=1:n;
```

```
    Zn(y)=prod(Zi(1:t));
if t < n; t=t+1;
end;
end;
Zk = Zn(min(n, length(Zn)));
vektor = [1;0];
hodnoty = 1;
uhel = uhly(1);
```

Nyní už přidáme celý zbytek, který obsahuje výše zmiňované parametry včetně parametru S označujícího směr rotace úhlu a námi vypočítanou matici A.

```
for i = 0:n-1;
    if delta < 0
        S = -1;
    else
        S = 1;
    end
    faktor = S * hodnoty;
    R = [1, -faktor; faktor, 1];
    vektor = R * vektor;
    delta = delta - S * uhel;
    hodnoty = hodnoty / 2;
    if i+2 > length(uhly)
        uhel = uhel / 2;
    else
        uhel = uhly(i+2);
    end
end;
vektor = vektor * Zk;
return
```

2.5. Určení počtu iterací

Jak si však můžeme všimnout, pro výpočet úhlu pomocí tohoto algoritmu je zapotřebí dvou argumentů. Prvním je úhel, který chceme spočítat, a druhým je počet iterací, které chceme provést, abychom se s nějakou určitou námahou dostali k nějakému určitému výsledku.

Tato možnost však pro nás může být v praxi nevýhodná a zbytečně komplikovaná. Pokud s tímto algoritmem začínáme, nemusíme být schopni jednoznačně určit vhodný počet iterací. Použijeme-li jich málo, dostaneme nepřesný výsledek. Použijeme-li jich naopak mnoho, budeme zbytečně zatěžovat hardware, který může mít pouze omezenou kapacitu.

Tomu můžeme jednoduše předejít tím, že si vytvoříme funkci, která nám určí, kolik musíme použít iterací, abychom dosáhli přesnosti na námi předem zvolený počet desetinných míst, maximálně však 8, jelikož máme hodnoty $\mathrm{arctg}(x)$ přesné pouze na 8 desetinných míst. Nejprve si do funkce (ještě před for cyklus) nadefinujeme další parametry:

```
n=100;     %maximální počet iterací
vn = 0;
```

a přímo do for cyklu:

```
Kn = Zn(min(i+1, length(Zn)));
vp= vn;
vn = vektor * Kn;
if j>0 && abs(vp(1)-vn(1))<10^(-r) && (vp(2)-vn(2) < 10^(-r))
    break;
end;
```

Tento předpis funguje tak, že nejprve načte první konstantu $vn = 0$ a uloží si ji jako první výsledek vp. Následně proběhne CORDIC s jednou iterací a výsledek uloží jako vn. Pokud bude splněna podmínka, že proběhne alespoň jedna iterace a zároveň absolutní rozdíl těchto hodnot je menší, než 10^{-r}, kde r představuje

počet desetinných míst, CORDIC se zastaví a vypíše výsledné hodnoty sinu i kosinu a počet iterací, které jsou nutné zadat, abychom dosáhli daného výsledku. Pokud však jedna z těchto podmínek splněna nebude, uloží si nově spočítanou hodnotu *vn* jako nové *vp* a znovu spočítá hodnotu *vn* a to tak dlouho, dokud podmínky splněny nebudou. Pro zobrazení výsledku použijeme:

```
disp(sprintf('Počet iterací = %d',i+1));
```

Celý předpis pak bude vypadat následovně:

```
function vektor = cordicp(delta,r) %vstupní argumenty jsou úhel a počet
n=100;                             %iterací, které chceme provést
if delta < -pi/2 || delta > pi/2
    if delta < 0
        vektor = cordicp(delta + pi, r);
    else
        vektor = cordicp(delta - pi, r);
    end
    vektor = -vektor;
    return
end
uhly = [ ...
0.78539816 0.46364761 0.24497866 0.12435499 ...
0.06241881 0.03123983 0.01562373 0.00781234 ...
0.00390623 0.00195312 0.00097656 0.00048828 ...
0.00024414 0.00012207 0.00006104 0.00003052 ...
0.00001526 0.00000763 0.00000381 0.00000191 ...
0.00000095 0.00000048 0.00000024 0.00000012 ...
0.00000006 0.00000003 0.00000001 0.00000001];
i=0:n-1;
t=1;
Zi=1./sqrt(1+2.^(-2*i));
```

```
for y=1:n;
    Zn(y)=prod(Zi(1:t));
if t < n; t=t+1;
end;
end;
vektor = [1;0];
hodnoty = 1;
uhel = uhly(1);
vn = 0;
for i = 0:n-1;
    if delta < 0
        S = -1;
    else
        S = 1;
    end
    faktor = S * hodnoty;
    R = [1, -faktor; faktor, 1];
    vektor = R * vektor;
    delta = delta - S * uhel;
    hodnoty = hodnoty / 2;
    if i+2 > length(uhly)
        uhel = uhel / 2;
    else
        uhel = uhly(i+2);
    end
    Kn = Zn(min(i+1, length(Zn)));
    vp= vn;
    vn = vektor * Kn;
    if i>0 && abs(vp(1)-vn(1))<10^(-r) && (vp(2)-vn(2) < 10^(-r))
        break;
```

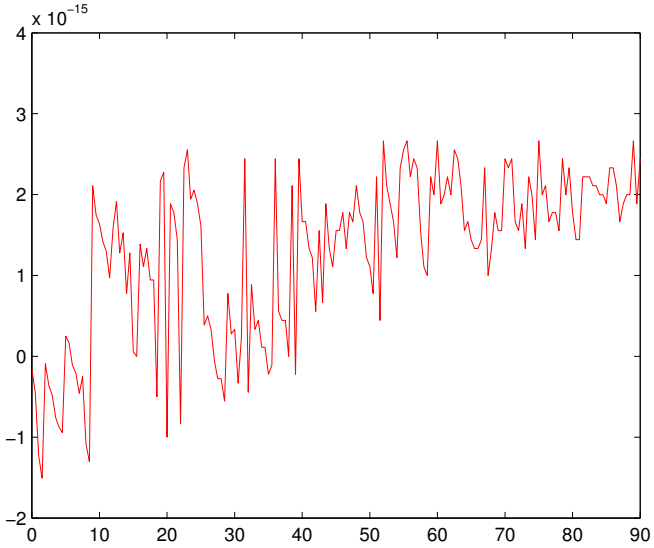
```
    end;
end
disp(sprintf('Počet iterací = %d',i+1));
vektor = vn;
return
```

2.6. Absolutní a relativní chyba

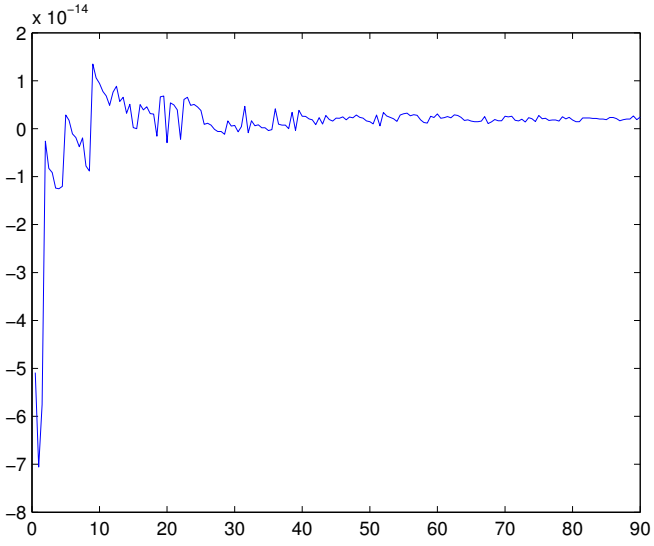
Nyní nás však může ještě napadnout, s jakou přesností počítá námi zvolený MATLAB. Pro většinu použití stačí krátký formát, který má čtyři desetinná místa, ale při náročnějších výpočtech můžeme chtít větší přesnost. Za tímto účelem však musíme spočítat více členů Taylorova polynomu, nebo použít hodnoty arkustangens spočítané pomocí MATLABU, tedy nahradit 12. řádek předpisu za `uhly = atan(2.^-(0:27))`. MATLAB může počítat až na 16 desetinných míst, což je cca 60 iterací. Můžeme porovnat jeho hodnoty s hodnotami CORDICu a graficky vyjádřit jejich absolutní a relativní chyby, například pro všechny úhly od nuly do $\frac{\pi}{2}$ s krokem 0,5° následujícím předpisem:

```
function v= test(n)
beta=[0:0.5:90];
t=beta*pi/180;
v= zeros(size(t));
for i=1:length(t);
    result = cordic(t(i),n);
    v(i) = result(2);
end;
figure(1); plot(beta,sin(t)-v,'r');%absolutní chyba
figure(2); plot(beta,((sin(t)-v)./sin(t)));%relativní chyba
```

Výsledné grafy znázorňují červeně absolutní chybu (odchylku naměřené hodnoty od skutečné hodnoty) a modře relativní chybu (poměr skutečnosti a absolutní chyby, tedy jak je chyba velká). Po vynásobení stem je vyjádřena v procentech.



Obrázek 5: absolutní chyba



Obrázek 6: relativní chyba

2.7. Závěr

Problematika výpočtu funkčních hodnot elementárních funkcí je velmi obsáhlá, přesto jsem rád, že jsem si toto téma vybral. Nejen, že jsem si oprášil svoje znalosti, ale získal i mnoho nových. Zjistil jsem, že to není věda pouze teoretická, ale má široké praktické využití. Aplikací vhodného matematického modelu v reálném životě nám může pomoci vyřešit nejeden problém a usnadnit jinak velmi náročnou práci.

Přitom řešení problémů pomocí elementárních funkcí, ač se může zdáti být obtížným, lze velmi snadno řešit pomocí základních operací (sčítání, odčítání, násobení a dělení). Je však potřebné mít správné informace, které jsou v dnešní době velmi často podceňovány. Největší výhodou je možnost určení přesnosti výsledků pomocí jednoduchých výpočtů pouze pomocí tužky a papíru, což nám může ušetřit zbytečnou práci, nebo naopak opakovaným používáním jednoduchého principu získat velmi přesné výsledky za použití výpočetní techniky, která je v dnešní době nedílnou součástí každé domácnosti.

V práci jsem se snažil využít jednoduchých výpočtů k získání přibližných výsledků, stejně jako jejich implementaci do matematického softwaru, konkrétně MATLABu, pomoci něhož získáme přesnější výsledky, včetně grafického znázornění.

Tato práce obohatila moje znalosti v oblasti elementárních funkcí, Taylorových polynomů, mocniných řad a dalších matematických výpočtů. Téma bylo pro mne velmi přínosné a potvrdil jsem si, že moje volba byla správná.

Literatura

[1] Jarník, V.: Diferenciální počet I, 4. vydání, Nakladatelství Československé akademie věd, Praha, 1955.

[2] Gál, T. - Růžička, J.: Elementární funkce v teorii a praxi, 1. vydání, Praha: SNTL, 1967.

[3] Taylorova řada [online], dostupné z: http://cs.wikipedia.org/wiki/Taylorova_řada

[4] A Taylor series for the function arctan [online], dostupné z: http://www.mathstat.concordia.ca/faculty/rhall/mc/arctan.pdf [citováno 14.3.2013]

[5] Aproximace funkcí polynomy [online], dostupné z: http://tjn.fjfi.cvut.cz/~edita/taylorN.pdf [citováno 18.3.2013]

[6] Van Wijngaarden transformation [online], dostupné z: http://en.wikipedia.org/wiki/Van_Wijngaarden_transformation#cite_note-1 [citováno 24.3.2013]

[7] Ivánkův blog [online], dostupné z: http://ivankuckir.blogspot.cz/2010/09/tayloruv-polynom-srozumitelne.html [citováno 5.3.2013]

[8] Matematické fórum [online], dostupné z: http://forum.matweb.cz/ [citováno 4.4.2013]

[9] CORDIC [online], dostupné z: http://en.wikipedia.org/wiki/CORDIC [citováno 15.12.2012]

[10] Výpočet goniometrických funkcí algoritmem CORDIC [online], dostupné z: http://www.root.cz/clanky/vypocet-goniometrickych-funkci-algoritmem-cordic/ [citováno 10.12.2012]