Kapitel 3. Approximation av funktioner

Vi skall nu övergå till att beskriva, hur man i praktiken numeriskt beräknar funktioner. I allmänhet kan inte ens elementära funktioner såsom sinus- och cosinusfunktionerna beräknas exakt utom i speciella fall. Några av dem definieras också med hjälp av integraler som inte kan beräknas exakt algebraiskt eller analytiskt. Sålunda brukar man ofta definiera den naturliga logaritmfunktionen som

$$\ln x = \int_1^x \frac{1}{t} dt$$

Metoder att beräkna dylika integraler skall vi studera senare. Andra funktioner (som t.ex. arcus tangenten) definieras som inversa funktioner. De blir då implicit definierade som lösningar till olinjära ekvationer som inte kan lösas algebraiskt. Detta är också något som vi skall studera senare.

Ofta är det mycket enklare att använda serieutvecklingar eller s.k. CORDIC algoritmer för att beräkna funktioner. Vid serieutvecklingen av funktioner är det viktigt att känna till konvergensradien, och att bestämma antalet termer som behövs för att reducera felet under en given toleransgräns. Därefter är det enkelt att summera det erforderliga antalet termer.

Både om man använder serieutvecklingar och CORDIC algoritmer uppstår svårigheter då argumentet hamnar utanför konvergensintervallet. I sådana fall använder man oftast någon intervallreduktionsmetod. Ett exempel på detta är de vanliga trigonometriska funktionerna sinus och cosinus, som är periodiska med perioden 2π. Om man har bra algoritmer för att beräkna funktionerna inom intervallet $[0, 2\pi]$, så kan man reducera större argument till detta intervall genom att subtrahera en multipel av 2π.

3.1. Serieutveckling av funktioner

Det finns två fundamentala potensserier från vilka många andra kan härledas, nämligen den geometriska serien

$$\boxed{\frac{1}{1-x} = \sum_{k=0}^{\infty} x^k = 1 + x + x^2 + \dots \quad (|x| < 1)}$$

och exponentialserien

$$\boxed{\exp(x) = e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots \quad (\forall x).}$$

Med hjälp av identiteten

$$e^{ix} = \cos x + i \sin x$$

får vi serieutvecklingarna för de trigonometriska funktionerna:

$$\cos x = \sum_{k=0}^{\infty} \frac{(-1)^k x^{2k}}{(2k)!} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} \ldots \quad (\forall x)$$

$$\sin x = \sum_{k=0}^{\infty} \frac{(-1)^k x^{2k+1}}{(2k+1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} \ldots \quad (\forall x)$$

samt serieutvecklingarna för de hyperboliska funktionerna:

$$\cosh x = \tfrac{1}{2}(e^x + e^{-x}) = \sum_{k=0}^{\infty} \frac{x^{2k}}{(2k)!} = 1 + \frac{x^2}{2!} + \frac{x^4}{3!} + \ldots \quad (\forall x)$$

$$\sinh x = \tfrac{1}{2}(e^x - e^{-x}) = \sum_{k=0}^{\infty} \frac{x^{2k+1}}{(2k+1)!} = x + \frac{x^3}{3!} + \frac{x^5}{5!} + \ldots \quad (\forall x)$$

Genom att termvis integrera den geometriska serien får vi

$$\ln(1-x) = -\sum_{k=0}^{\infty} \frac{x^{k+1}}{k+1} = -x - \frac{x^2}{2} - \frac{x^3}{3} - \ldots \quad (|x| < 1),$$

och om vi ersätter x med $-x$

$$\ln(1+x) = -\sum_{k=0}^{\infty} \frac{(-1)^{k+1} x^{k+1}}{k+1} = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} \ldots \quad (|x| \leq 1).$$

Denna metod att beräkna den naturliga logaritmen är dock inte särskilt effektiv. Eftersom serien konvergerar för $x = 1$, så har vi $\ln 2 = 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} \ldots$. Om vi beräknar summan av de åtta första termerna i denna serie, får vi $\ln 2 \approx 0.63452381$. Eftersom det exakta värdet är ≈ 0.69314718, så är felet ca 0.06, eller närmare 9%. Serien konvergerar ytterst långsamt.

En snabbare konvergent serie erhålls genom att subtrahera de båda logaritmiska serierna från varandra:

$$\ln \frac{1+x}{1-x} = \ln(1+x) - \ln(1-x) = 2\left(x + \frac{x^3}{3} + \frac{x^5}{5} + \ldots\right)$$

Genom att lösa ekvationen $\frac{1+x}{1-x} = 2$ får vi $x = 1/3$, så att

$$\ln 2 = \frac{2}{3}\left[1 + \frac{(1/3)^2}{3} + \frac{(1/3)^4}{5} + \ldots\right] = \frac{2}{3}\sum_{k=0}^{\infty} \frac{1}{(2k+1)3^{2k}}$$

Om vi vill beräkna logaritmen med ett fel som understiger 10^{-6} så får vi antalet termer N som behövs genom att kräva att olikheten

$$\frac{2}{3} \cdot \frac{1}{2N+1} \cdot \frac{1}{9^N} < 10^{-6}$$

skall gälla. Detta ger $N \geq 6$, varför 6 termer är tillräckligt. Vi får då resultatet

$$\ln 2 \approx \frac{2}{3} \sum_{k=0}^{5} \frac{1}{(2k+1)3^{2k}} = 0.69314707,$$

som stämmer mycket väl överens med det exakta värdet.

Som ett annat exempel skall vi studera hur man kan beräkna ett värde av π utgående från identiteten $\arctan 1 = \pi/4$. Vi börjar med att konstruera en serieutveckling av denna funktion. Genom substitution av $x = -t^2$ i den geometriska serien finner en serieutveckling av arcus tangentens derivata $\frac{d}{dt} \arctan t = \frac{1}{1+t^2}$:

$$\frac{d}{dt} \arctan t = 1 - t^2 + t^4 - t^6 \ldots$$

som konvergerar för $|t| < 1$. Genom termvis integration av denna potensserie får vi

$$\arctan x = \int_0^x (1 - t^2 + t^4 - t^6 \ldots) dt = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} \ldots,$$

som konvergerar för $|x| \leq 1$. Vi finner således, att

$$\frac{\pi}{4} = \arctan 1 = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} \ldots$$

De åtta första termerna ger approximationen

$$\pi \approx 4 \sum_{k=0}^{7} \frac{(-1)^k}{2k+1} = 3.0170718.$$

Om vi tillägger nästa term får vi 3.2523659, vilket fortfarande skiljer sig mycket från det korrekta värdet av π (3.1415926). För dubbel precision enligt IEEE standarden, borde 2^{53} termer beräknas!

En betydligt snabbare konvergent serie får vi med hjälp av identiteten $\arctan \frac{1}{\sqrt{3}} = \pi/6$. Om vi substituerar detta argument i arcustangentens serieutveckling, så får vi

$$\begin{aligned} \pi &= 6 \left[3^{-1/2} - \frac{3^{-3/2}}{3} + \frac{3^{-5/2}}{5} \ldots \right] \\ &= \frac{6}{\sqrt{3}} \left[1 - \frac{1}{3 \cdot 3} + \frac{1}{5 \cdot 3^2} \ldots \right] = 2\sqrt{3} \sum_{k=0}^{\infty} \frac{(-1)^k}{(2k+1)3^k} \end{aligned}$$

Felet i denna serieutveckling får man genom att beräkna storleken av den första bortlämnade termen. Om vi kräver enkel precisions nogrannhet, så måste felet bli mindre än 2^{-23}, och antalet termer N bör vara så stort, att villkoret

$$\frac{2\sqrt{3}}{(2N+1)3^N} < 2^{-23}$$

är uppfyllt. Vi finner alltså $N \leq 13$. Samma noggrannhet kräver minst 17 millioner termer, om man använder den förstnämnda metoden att beräkna π. Med hjälp av den nya serien får vi

$$\pi \approx 2\sqrt{3} \sum_{k=0}^{13} \frac{(-1)^k}{(2k+1)3^k} = 3.1415926,$$

vilket är ett verkligt bra resultat.

Som ett sista exempel på användningen av serieutvecklingar, skall vi undersöka hur många termer som man behöver beräkna i exponentialserien för att uppnå en viss noggrannhet. Låt oss anta, att $|x| \leq 2$ och att felet $< 10^{-5}$. Eftersom felet växer med $|x|$ och serien är alternerande för $x < 0$, så kan vi nöja oss med att behandla $x = 2$.

Om serien avbryts efter N termer, får vi approximationen $\exp x \approx \sum_{k=0}^{N-1} \frac{x^k}{k!}$ och det absoluta felet kan då uttryckas

$$\begin{aligned} e_N(x) &= \sum_{k=N}^{\infty} \frac{x^k}{k!} = \frac{x^N}{N!} + \frac{x^{N+1}}{(N+1)!} + \frac{x^{N+2}}{(N+2)!} + \dots \\ &= \frac{x^N}{N!}\left[1 + \frac{x}{N+1} + \frac{x^2}{(N+1)(N+2)} + \dots\right] \\ &\leq \frac{x^N}{N!}\left[1 + \frac{x}{N+1} + \frac{x^2}{(N+1)^2} + \dots\right] \\ &= \frac{x^N}{N!}\frac{1}{1-\frac{x}{N+1}}, \end{aligned}$$

under förutsättning att $0 < x < N+1$. För $x = 2$ förenklas denna olikhet till

$$e_N(2) \leq \frac{2^N}{N!} \cdot \frac{N+1}{N-1},$$

som alltså skall vara mindre än 10^{-5} enligt vårt antagande.

Efter prövning finner vi att $\frac{2^{11}}{11!} \approx 5.13 \cdot 10^{-5}$, medan $\frac{2^{12}}{12!} = 8.55 \cdot 10^{-6}$. För $N = 12$ blir faktorn $\frac{N+1}{N-1} \approx 1.18$, så att övre gränsen blir $\approx 1.01 \cdot 10^{-5}$, som är något för stort. Med 13 termer blir gränsen säkert mindre än 10^{-5}: $\frac{2^{13}}{13!} \cdot \frac{14}{12} \approx 1.53 \cdot 10^{-6}$.

Om $|x| < \frac{1}{2}$, behöver vi endast medta 7 termer för att uppnå samma noggrannhet. Antalet termer som behövs växer alltså snabbt med x. För att beräkna exponentialfunktionen för stora värden av x kan man använda intervallreduktion. För att beräkna e^2 kan man alltså först använda 7 termer för att beräkna $e^{\frac{1}{2}}$, och sedan beräkna e^2 genom två kvadreringar : $e^2 = [(e^{\frac{1}{2}})^2]^2$ (visserligen kommer felet då att växa, vilket man måste ta i beaktande).

3.2. CORDIC algoritmer

Vi skall nu studera en metod som kan uppfattas som ett modernt sätt att interpolera i funktionstabeller. CORDIC metoden används för att approximativt beräkna funktionsvärden på alla moderna grafiska räknare, som t.ex. TI-85. Algoritmen är inte baserad på serieutveckling eller polynomapproximation, utan istället på ett elementärt system av iterativa ekvationer. CORDIC algoritmen COordinate Rotation DIgital Computer) infördes av Jack Volder år 1959 för att beräkna trigonometriska funktionsvärden, och utvidgades till andra elementära funktioner av John Walther år 1971.

I CORDIC metoden används endast additioner, subtraktioner, skiftoperationer, jämförelser och uppladdning av lagrade konstanter, vilket gör den speciellt lämpad just för räknare och PC-processorer. Med CORDIC metoden kan man t.o.m. utföra multiplikation och division.

CORDIC algoritmen kodas i decimalsystemet, eftersom det tar tid att konvertera från basen 10 till basen 2. Men det är lättare att förstå metoden, om den uttrycks i binärsystemet.

Den trigonometriska CORDIC algoritmen kan härledas ur en allmän rotation i planet:

$$x' = x\cos\phi - y\sin\phi$$
$$y' = y\cos\phi + x\sin\phi$$

som roterar en vektor vinkeln ϕ. Dessa ekvationer kan också skrivas i formen

$$x' = \cos\phi[x - y\tan\phi]$$
$$y' = \cos\phi[y + x\tan\phi]$$

Om rotationsvinklarna begränsas på sådant sätt att $\tan\phi = \pm 2^{-k}$, så kan multiplikationen med tangenten uttryckas som en skiftoperation, och man får godtyckliga rotationsvinklar genom att upprepa ett antal elementära rotationer av detta slag. Emedan $\cos(\arctan x) = (1 + x^2)^{-1/2}$, så kan vi sätta $K_k = \cos(\arctan 2^{-k}) = (1 + 2^{-2k})^{-1/2}$ och uttrycka de successiva rotationerna i formen

$$x_{k+1} = K_k[x_k - d_k y_k 2^{-k}]$$
$$y_{k+1} = K_k[y_k + d_k x_k 2^{-k}],$$

där konstanten $d_k = \pm 1$.

Skalningsfaktorerna K_k behöver inte sättas ut, och kan beaktas senare. Deras produkt kommer att småningom närma sig gränsvärdet 0.6073. Genom ackumulering av vinkelrotationerna tillkommer ytterligare en differensekvation: $z_{k+1} = z_k - d_k \arctan 2^{-k}$.

I allmänhet kan algoritmen uttryckas med hjälp av tre iterativa ekvationer av formen:

$$\boxed{\begin{aligned} x_{k+1} &= x_k - m\delta_k y_k 2^{-k} \\ y_{k+1} &= y_k + \delta_k x_k 2^{-k} \\ z_{k+1} &= z_k - \delta_k \sigma_k, \end{aligned}}$$

där värdet av konstanterna m, δ_k och σ_k beror på vilken typ av beräkning som skall utföras.

m är antingen $0, 1$ eller -1. Värdet $m = 1$ används för trigonometriska och inversa trigonometriska funktioner, medan $m = -1$ används för hyperboliska, inversa hyperboliska och logaritmiska funktioner, liksom även för kvadratrötter. Värdet $m = 0$ används slutligen för multiplikation och division.

Värdet av δ_k bestäms av två signum-funktioner:

$$\delta_k = \mathrm{sgn}(\mathrm{z_k}) = \begin{cases} 1, & \text{om } z_k \geq 0 \\ -1, & \text{om } z_k < 0 \end{cases} \text{ eller } \delta_\mathrm{k} = -\mathrm{sgn}(\mathrm{y_k}) = \begin{cases} 1, & \text{om } y_k < 0 \\ -1, & \text{om } y_k \geq 0. \end{cases}$$

Den förstnämnda funktionen kallas för rotationsmoden, där z–värdena reduceras mot 0, och den andra kallas vektoriseringsmoden, där y–värdena reduceras mot 0. Observera, att beräkningen av δ_k endast kräver en jämförelse.

Talen σ_k är lagrade konstanter, vilkas värden beror av värdet på m. Om $m = 1$, så är $\sigma_k = \arctan 2^{-k}$, om $m = 0$, så är $\sigma_k = 2^{-k}$, och om $m = -1$, så är $\sigma_k = \tanh^{-1} 2^{-k}$.

För att kunna använda dessa ekvationer, måste man ange utgångsvärdena x_0 ,y_0 och z_0. Ett av dessa tal (t.ex. z_0) kan vara den vinkel vars sinus vi önskar beräkna. Eller också kan två av talen (t.ex. x_0 och y_0) vara täljare och nämnare i den kvot vi vill approximera. I varje fall måste utgångsvärdena var begränsade till ett visst intervall kring origo, så att man kan försäkra sig om konvergens. Som vi ser av exemplen, kommer en av variablerna att gå mot noll, medan någon av de andra närmar sig den önskade approximationen.

Vi skall nu tillämpa CORDIC algoritmen på division. Antag att $m = 0$, $\delta_k = -\mathrm{sgn}(y_k)$ och $\sigma_k = 2^{-k}$. Nedanstående iterativa ekvationer kommer då att ge successiva approximationer för kvoten y_0/x_0 då $|y_0/x_0| \leq 2$:

$$x_{k+1} = x_0$$

$$y_{k+1} = y_k + \delta_k x_0 2^{-k}$$

$$z_{k+1} = z_k - \delta_k 2^{-k} \qquad (\sigma_k = 2^{-k})$$

Som vi ser, innehåller ekvationerna endast additioner, subtraktioner, jämförelser och binära skiftoperationer.

Om $z_0 = 0$, så finner vi att $|z_{n+1} - y_0/x_0| < 2^{-n}$, förutsatt att $|y_0/x_0| \leq 2$. Detta inses på följande sätt. Efter ett stort antal iterationer n, så är $y_{k+1} \approx 0$, och vi får $y_0 \approx -\sum_{k=0}^{n} \delta_k x_0 2^{-k}$, eller alltså $y_0/x_0 \approx -\sum_{k=0}^{n} \delta_k 2^{-k}$ med ett fel som understiger 2^{-n}. Å andra sidan följer av den tredje ekvationen ovan att $z_{n+1} = -\sum_{k=0}^{n} \delta_k 2^{-k}$ (eftersom $z_0 = 0$), varav påståendet följer.

Som ett exempel på användningen, kan vi beräkna $1.2/2.3 \approx 0.52714$. Vi får då $x_0 = 2.3$ samt

k	y_k	z_k	δ_k
0	1.2	0	-1
1	-1.1	1	$+1$
2	0.05	0.5	-1
3	-0.525	0.75	$+1$
4	-0.2375	0.625	$+1$
5	-0.09375	0.5625	$+1$
6	-0.021875	0.53125	$+1$
7	0.0140625	0.515625	-1
8	-0.00390625	0.5234375	$+1$
$\vdots$	$\vdots$	$\vdots$	$\vdots$

Som vi ser, är felet i $z_8 \approx 0.0037 < 2^{-8} = 0.0039$.

För att approximera sinus och cosinus av en vinkel $z_0 = \theta$, $-\pi/2 \leq \theta \leq \pi/2$ använder vi CORDIC algoritmen med $m = 1$, $\delta_k = \mathrm{sgn}(z_k)$ och $\sigma_k = \arctan 2^{-k}$:

$$
\begin{aligned}
x_{k+1} &= x_k - \delta_k y_k 2^{-k} \\
y_{k+1} &= y_k + \delta_k x_k 2^{-k} \\
z_{k+1} &= z_k - \delta_k \arctan 2^{-k} \qquad (\sigma_k = \arctan 2^{-k})
\end{aligned}
$$

Begynnelsevärdena är $x_0 = K = \prod_{j=0}^{n} \cos(\sigma_j)$, $y_0 = 0$ och $z_0 = \theta$, som är den givna vinkeln (i radianer). På grund av det sätt varpå z konstrueras, kommer z_k att närma sig 0. Därvid närmar sig x $\cos\theta$ och y $\sin\theta$, som vi ser av exemplet nedan.

Nedanstående MATLAB program gör 47 iterationer av algoritmen för att approximera sinus och cosinus av en vinkel θ inom intervallet $[-\pi/2, \pi/2]$. Vi ger också en utskrift från programmet, som visar hur z–värdena avtar, samtidigt som x–värdena närmar sig $\cos 1$ och y–värdena $\sin 1$.

```
function cordtrig(t)
  n=48; x=zeros(n,1); y=x; z=x;
  s=2.^(0:-1:-46); sig=atan(s);
  K=prod(cos(sig));
  x(1)=K; y(1)=0; z(1)=t;
```

```
for j=1:n-1,
   del=sign(z(j)); if del == 0 del =1; end;
   x(j+1) = x(j) - del*y(j)*s(j);
   y(j+1) = y(j) + del*x(j)*s(j);
   z(j+1) = z(j) - del*sig(j);
end
answer = [x y z]
```

Resultatet av en beräkning med $t = 1$ (endast början och slutet):

```
 0.60725293500888                   0   1.00000000000000
 0.60725293500888   0.60725293500888   0.21460183660255
 0.30362646750444   0.91087940251332  -0.24904577239825
 0.53134631813277   0.83497278563721  -0.00406710927139
 0.63571791633742   0.76855449587062   0.12028788527537
 0.58768326034551   0.80828686564170   0.05786907527941
 0.56242429579421   0.82665196752750   0.02662924184915
 0.54950785880159   0.83543984714929   0.01100551322867
 0.54298098499574   0.83973287729617   0.00319317216857
 0.53970077844380   0.84185389676881  -0.00071305796340
```

```
0.54030230587315   0.84147098480468   0.00000000000595
   ..                 ..                 ..
   ..                 ..                 ..
0.54030230587009   0.84147098480665   0.00000000000231
0.54030230586855   0.84147098480763   0.00000000000049
0.54030230586779   0.84147098480812  -0.00000000000042
0.54030230586817   0.84147098480788   0.00000000000004
0.54030230586798   0.84147098480800  -0.00000000000019
0.54030230586808   0.84147098480794  -0.00000000000008
0.54030230586812   0.84147098480791  -0.00000000000002
0.54030230586815   0.84147098480789   0.00000000000001
0.54030230586814   0.84147098480790   0.00000000000000
```

Varför fungerar algoritmen? Låt oss först analysera geometrin genom att omskriva ekvationerna för x och y i matrisform:

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & -\delta_k 2^{-k} \\ \delta_k 2^{-k} & 1 \end{bmatrix} \begin{bmatrix} x_k \\ y_k \end{bmatrix}$$

Eftersom $\sigma_k = \arctan 2^{-k}$ så är $\tan \sigma_k = 2^{-k}$ och $\sin \sigma_k = \cos \sigma_k 2^{-k}$.

Härav följer, att den tvåradiga matrisen ovan kan skrivas

$$\frac{1}{\cos\sigma_k}\begin{bmatrix}\cos\sigma_k & -\delta_k\cos\sigma_k 2^{-k}\\ \delta_k\cos\sigma_k 2^{-k} & \cos\sigma_k\end{bmatrix} = \frac{1}{\cos\sigma_k}\begin{bmatrix}\cos\sigma_k & -\delta_k\sin\sigma_k\\ \delta_k\sin\sigma_k & \cos\sigma_k\end{bmatrix}$$
$$= \frac{1}{\cos\sigma_k}\begin{bmatrix}\cos(\delta_k\sigma_k) & -\sin(\delta_k\sigma_k)\\ \sin(\delta_k\sigma_k) & \cos(\delta_k\sigma_k)\end{bmatrix}.$$

Detta är en "skalad rotation" i planet, eftersom matrisen

$$\begin{bmatrix}\cos(\delta_k\sigma_k) & -\sin(\delta_k\sigma_k)\\ \sin(\delta_k\sigma_k) & \cos(\delta_k\sigma_k)\end{bmatrix}$$

ger upphov till en motsols rotation med rotationsvinkeln $\delta_k\sigma_k = \pm\sigma_k = \pm\arctan 2^{-k}$, medan faktorn $1/\cos\sigma_k$ skalar resultatet. Geometriskt kommer vektorn (x_k, y_k) att roteras vinkeln $\pm\sigma_k = \pm\arctan 2^{-k}$, varpå den töjs ut med en faktor $1/\cos\sigma_k$.

Härefter är det lätt att se att algoritmen fungerar. Av ekvationen $z_{k+1} = z_k - \delta_k\arctan 2^{-k}$ och $z_0 = \theta$ följer att $z_{n+1} = \theta - \sum_{k=0}^{n}\delta_k\sigma_k$. Man kan visa, att för stora värden av n gäller $z_{n+1} \approx 0$, och således är $\theta \approx \sum_{k=0}^{n}\delta_k\sigma_k$.

Genom att kombinera $n+1$ rotationer, finner vi att x–värdena närmar sig $\cos\theta$ och y–värdena närmar sig $\sin\theta$:

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \frac{1}{\cos\sigma_0} \begin{bmatrix} \cos(\delta_0\sigma_0) & -\sin(\delta_0\sigma_0) \\ \sin(\delta_0\sigma_0) & \cos(\delta_0\sigma_0) \end{bmatrix} \begin{bmatrix} K \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \frac{1}{\cos\sigma_0\cos\sigma_1} \begin{bmatrix} \cos(\delta_0\sigma_0+\delta_1\sigma_1) & -\sin(\delta_0\sigma_0+\delta_1\sigma_1) \\ \sin(\delta_0\sigma_0+\delta_1\sigma_1) & \cos(\delta_0\sigma_0+\delta_1\sigma_1) \end{bmatrix} \begin{bmatrix} K \\ 0 \end{bmatrix}$$

$$\vdots$$

$$\begin{bmatrix} x_{n+1} \\ y_{n+1} \end{bmatrix} = \frac{1}{\cos\sigma_0\cos\sigma_1\ldots\cos\sigma_n} \begin{bmatrix} \cos(\sum_{k=0}^{n}\delta_k\sigma_k) & -\sin(\sum_{k=0}^{n}\delta_k\sigma_k) \\ \sin(\sum_{k=0}^{n}\delta_k\sigma_k) & \cos(\sum_{k=0}^{n}\delta_k\sigma_k) \end{bmatrix} \begin{bmatrix} K \\ 0 \end{bmatrix}$$

$$\approx \frac{1}{K} \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} K \\ 0 \end{bmatrix} = \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix}$$

Av programkoden ovan verkar det som om man borde känna värden både av arctan och cosinus för att kunna använda den. Observera dock, att endast speciella värden av σ_k och $\cos\sigma_k$ behövs då algoritmen kodas i hårdvaran. Dessa värden kan beräknas på förhand och lagras i flyttalsprocessorn.