Como funciona a sua calculadora?

Tiago J. Fonseca

ICMC - USP

Seminários de Coisas Legais - Abril, 2011

Como aproximar funções trigonométricas?

- Responda rápido: qual é o algoritmo usado, nas calculadoras, para aproximar funções trigonométricas (e.g. $\sin\theta$, $\cos\theta$, etc.)?

Como aproximar funções trigonométricas?

- Responda rápido: qual é o algoritmo usado, nas calculadoras, para aproximar funções trigonométricas (e.g. $\sin\theta$, $\cos\theta$, etc.)?
- Não é expansão em Taylor!

Como aproximar funções trigonométricas?

- Responda rápido: qual é o algoritmo usado, nas calculadoras, para aproximar funções trigonométricas (e.g. $\sin\theta$, $\cos\theta$, etc.)?
- Não é expansão em Taylor!

$$\sin\theta = \theta - \frac{\theta^3}{3!} + \frac{\theta^5}{5!} - \frac{\theta^7}{7!} + \cdots$$

Como aproximar funções trigonométricas?

- Responda rápido: qual é o algoritmo usado, nas calculadoras, para aproximar funções trigonométricas (e.g. $\sin\theta$, $\cos\theta$, etc.)?
- Não é expansão em Taylor!

$$\sin\theta = \theta - \frac{\theta^3}{3!} + \frac{\theta^5}{5!} - \frac{\theta^7}{7!} + \cdots$$

- Por que não?

Como aproximar funções trigonométricas?

- Responda rápido: qual é o algoritmo usado, nas calculadoras, para aproximar funções trigonométricas (e.g. $\sin\theta$, $\cos\theta$, etc.)?
- Não é expansão em Taylor!

$$\sin\theta = \theta - \frac{\theta^3}{3!} + \frac{\theta^5}{5!} - \frac{\theta^7}{7!} + \cdots$$

- Por que não? O problema são as multiplicações.

Custo computacional

Lembremos que computadores normalmente trabalham com números binários.

Custo computacional

Lembremos que computadores normalmente trabalham com números binários. As operações mais rápidas que podem ser feitas num computador são:

1. Adicionar e subtrair números.

Custo computacional

Lembremos que computadores normalmente trabalham com números binários. As operações mais rápidas que podem ser feitas num computador são:

1. Adicionar e subtrair números.
2. Comparação de números (i.e. decidir se é maior ou menor).

Custo computacional

Lembremos que computadores normalmente trabalham com números binários. As operações mais rápidas que podem ser feitas num computador são:

1. Adicionar e subtrair números.
2. Comparação de números (i.e. decidir se é maior ou menor).
3. Armazenamento e leitura de números na memória.

Custo computacional

Lembremos que computadores normalmente trabalham com números binários. As operações mais rápidas que podem ser feitas num computador são:

1. Adicionar e subtrair números.
2. Comparação de números (i.e. decidir se é maior ou menor).
3. Armazenamento e leitura de números na memória.
4. Multiplicação por 2^n, $n \in \mathbb{Z}$.

Custo computacional

Lembremos que computadores normalmente trabalham com números binários. As operações mais rápidas que podem ser feitas num computador são:

1. Adicionar e subtrair números.
2. Comparação de números (i.e. decidir se é maior ou menor).
3. Armazenamento e leitura de números na memória.
4. Multiplicação por 2^n, $n \in \mathbb{Z}$. Ex.:
   $0.0001011 \times 2^2 = 0.0101100$.

Custo computacional

Lembremos que computadores normalmente trabalham com números binários. As operações mais rápidas que podem ser feitas num computador são:

1. Adicionar e subtrair números.
2. Comparação de números (i.e. decidir se é maior ou menor).
3. Armazenamento e leitura de números na memória.
4. Multiplicação por 2^n, $n \in \mathbb{Z}$. Ex.:
   $0.0001011 \times 2^2 = 0.0101100$.

Num computador com pouca capacidade de processamento, o ideal seria obter um algoritmo que utilizasse apenas estas operações.

CORDIC

- Em 1959, Jack Volder inventou o algoritmo CORDIC.

CORDIC

- Em 1959, Jack Volder inventou o algoritmo CORDIC.

COordinate **R**otation **DI**gital **C**omputer

CORDIC

- Em 1959, Jack Volder inventou o algoritmo CORDIC.

  COordinate **R**otation **DI**gital **C**omputer

- O CORDIC é muito rápido! Consiste, majoritariamente, em utilizar as 4 operações anteriores.

CORDIC

- Em 1959, Jack Volder inventou o algoritmo CORDIC.

COordinate **R**otation **DI**gital **C**omputer

- O CORDIC é muito rápido! Consiste, majoritariamente, em utilizar as 4 operações anteriores.
- Assim como boa parte dos algoritmos numéricos, o CORDIC foi desenvolvido por uma razão muito nobre: fins militares.

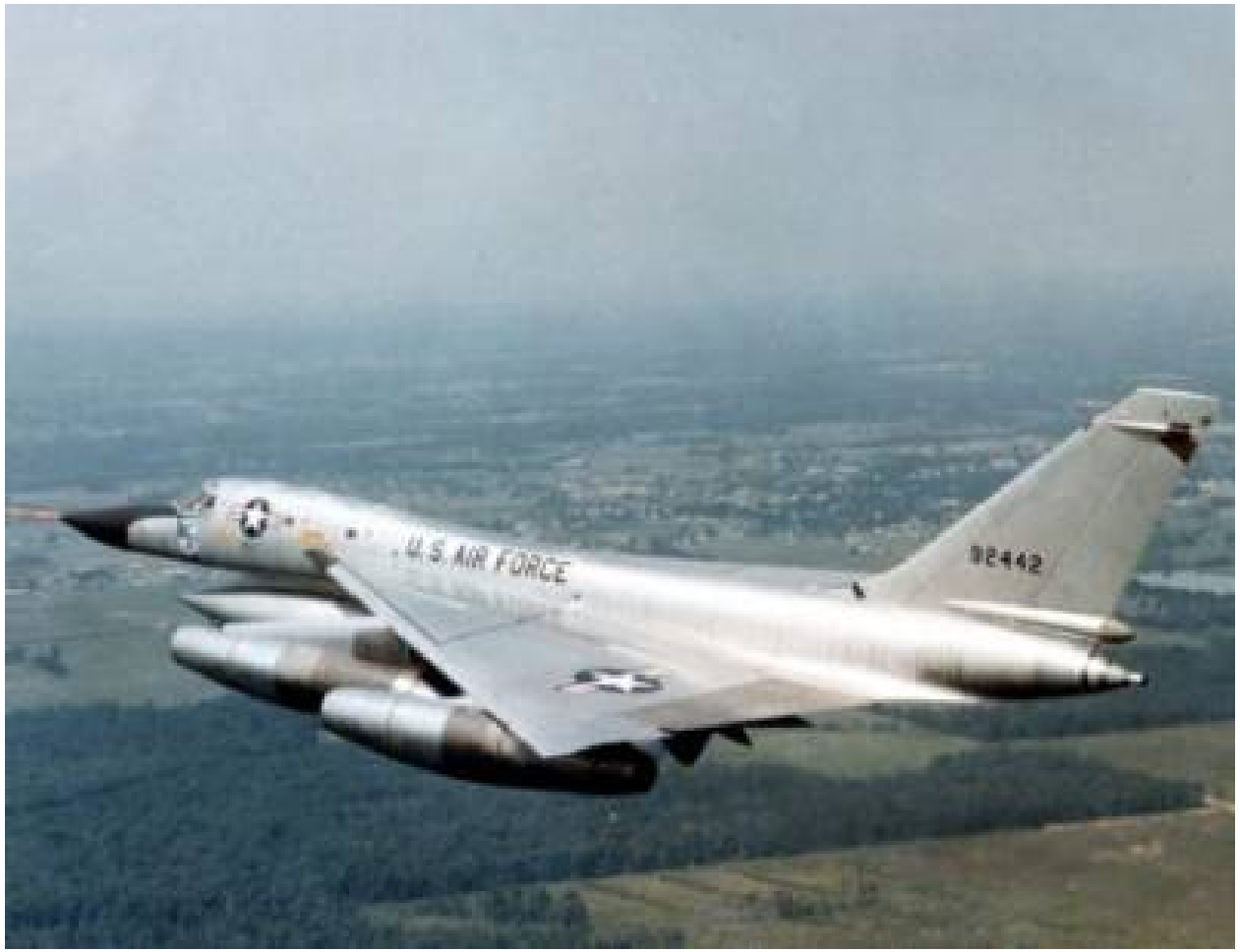
CORDIC

- Em 1959, Jack Volder inventou o algoritmo CORDIC.

COordinate **R**otation **DI**gital **C**omputer

- O CORDIC é muito rápido! Consiste, majoritariamente, em utilizar as 4 operações anteriores.
- Assim como boa parte dos algoritmos numéricos, o CORDIC foi desenvolvido por uma razão muito nobre: fins militares. O objetivo era utilizá-lo no sistema de navegação do bombardeiro B-58.

B58-Hustler



Ideia

- Vamos aproximar $\cos\theta$ e $\sin\theta$ ao mesmo tempo.

Ideia

- Vamos aproximar $\cos\theta$ e $\sin\theta$ ao mesmo tempo.
- Podemos considerar o ponto $(\cos\theta, \sin\theta)$ no círculo unitário.

Ideia

- Vamos aproximar $\cos\theta$ e $\sin\theta$ ao mesmo tempo.
- Podemos considerar o ponto $(\cos\theta, \sin\theta)$ no círculo unitário.
- Objetivo: obter uma sequência de pontos em $\mathbb{R}^2$ convergindo para $(\cos\theta, \sin\theta)$.

Ideia

- Vamos aproximar $\cos\theta$ e $\sin\theta$ ao mesmo tempo.
- Podemos considerar o ponto $(\cos\theta, \sin\theta)$ no círculo unitário.
- Objetivo: obter uma sequência de pontos em $\mathbb{R}^2$ convergindo para $(\cos\theta, \sin\theta)$.
- Vamos assumir que todos os pontos desta sequência estão no círculo unitário. Porquê?

Ideia

- Vamos aproximar $\cos\theta$ e $\sin\theta$ ao mesmo tempo.
- Podemos considerar o ponto $(\cos\theta, \sin\theta)$ no círculo unitário.
- Objetivo: obter uma sequência de pontos em $\mathbb{R}^2$ convergindo para $(\cos\theta, \sin\theta)$.
- Vamos assumir que todos os pontos desta sequência estão no círculo unitário. Porquê?
- Pois é fácil "mover" pontos no círculo unitário.

Rotações no $\mathbb{R}^2$

$$R_\theta = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

Rotações no $\mathbb{R}^2$

$$R_\theta = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

Dado um vetor (ponto) em $\mathbb{R}^2$ com coordenadas (x, y). O vetor

$$\begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta \end{pmatrix}$$

é o vetor (x, y) rotacionado de um ângulo θ no sentido anti-horário.

Rotações no $\mathbb{R}^2$

$$R_\theta = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

Dado um vetor (ponto) em $\mathbb{R}^2$ com coordenadas (x, y). O vetor

$$\begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta \end{pmatrix}$$

é o vetor (x, y) rotacionado de um ângulo θ no sentido anti-horário. Obs.: Se θ é negativo, giramos no sentido horário.

- Vamos fixar que o ponto de partida é sempre $(1, 0)$.

- Vamos fixar que o ponto de partida é sempre $(1, 0)$.
- Dado um ponto $(\cos x, \sin x)$, queremos agora obter ângulos $\theta_1, \theta_2, \ldots, \theta_n$ (positivos ou negativos), de maneira esperta, tais que

$$\begin{pmatrix} \cos x \\ \sin x \end{pmatrix} \approx R_{\theta_n} \cdots R_{\theta_2} R_{\theta_1} \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

e o produto da direita seja "rápido" de calcular.

- Vamos fixar que o ponto de partida é sempre $(1, 0)$.
- Dado um ponto $(\cos x, \sin x)$, queremos agora obter ângulos $\theta_1, \theta_2, \ldots, \theta_n$ (positivos ou negativos), de maneira esperta, tais que

$$\begin{pmatrix} \cos x \\ \sin x \end{pmatrix} \approx R_{\theta_n} \cdots R_{\theta_2} R_{\theta_1} \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

  e o produto da direita seja "rápido" de calcular.
- Note que

- Vamos fixar que o ponto de partida é sempre $(1, 0)$.
- Dado um ponto $(\cos x, \sin x)$, queremos agora obter ângulos $\theta_1, \theta_2, \ldots, \theta_n$ (positivos ou negativos), de maneira esperta, tais que

$$\begin{pmatrix} \cos x \\ \sin x \end{pmatrix} \approx R_{\theta_n} \cdots R_{\theta_2} R_{\theta_1} \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

  e o produto da direita seja "rápido" de calcular.
- Note que (aqui começa a sacanagem)

- Vamos fixar que o ponto de partida é sempre $(1, 0)$.
- Dado um ponto $(\cos x, \sin x)$, queremos agora obter ângulos $\theta_1, \theta_2, \ldots, \theta_n$ (positivos ou negativos), de maneira esperta, tais que

$$\begin{pmatrix} \cos x \\ \sin x \end{pmatrix} \approx R_{\theta_n} \cdots R_{\theta_2} R_{\theta_1} \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

  e o produto da direita seja "rápido" de calcular.
- Note que (aqui começa a sacanagem)

$$R_\theta = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} = \cos\theta \begin{pmatrix} 1 & -\tan\theta \\ \tan\theta & 1 \end{pmatrix}$$

- Vamos fixar que o ponto de partida é sempre $(1, 0)$.
- Dado um ponto $(\cos x, \sin x)$, queremos agora obter ângulos $\theta_1, \theta_2, \ldots, \theta_n$ (positivos ou negativos), de maneira esperta, tais que

$$\begin{pmatrix} \cos x \\ \sin x \end{pmatrix} \approx R_{\theta_n} \cdots R_{\theta_2} R_{\theta_1} \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

  e o produto da direita seja "rápido" de calcular.
- Note que (aqui começa a sacanagem)

$$R_\theta = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} = \cos\theta \begin{pmatrix} 1 & -\tan\theta \\ \tan\theta & 1 \end{pmatrix}$$

- E se os ângulos θ_n fossem tais que $\tan\theta_n = \frac{1}{2^n}$?

Se $\theta_n = \arctan \frac{1}{2^n}$,

Se $\theta_n = \arctan \frac{1}{2^n}$,então

$$
\begin{aligned}
R_{\theta_n} \begin{pmatrix} a \\ b \end{pmatrix} &= \cos\theta_n \begin{pmatrix} 1 & -\tan\theta_n \\ \tan\theta_n & 1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} \\
&= \cos\theta_n \begin{pmatrix} 1 & -1/2^n \\ 1/2^n & 1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} \\
&= \cos\theta_n \begin{pmatrix} a - b/2^n \\ a/2^n + b \end{pmatrix}
\end{aligned}
$$

Se $\theta_n = \arctan \frac{1}{2^n}$,então

$$
\begin{aligned}
R_{\theta_n} \begin{pmatrix} a \\ b \end{pmatrix} &= \cos\theta_n \begin{pmatrix} 1 & -\tan\theta_n \\ \tan\theta_n & 1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} \\
&= \cos\theta_n \begin{pmatrix} 1 & -1/2^n \\ 1/2^n & 1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} \\
&= \cos\theta_n \begin{pmatrix} a - b/2^n \\ a/2^n + b \end{pmatrix}
\end{aligned}
$$

Esquecendo o $\cos\theta_n$ por um momento, observe que estamos realizando apenas operações rápidas para multiplicar R_θ por um vetor.

Basicamente, o algoritmo funciona assim: dado um ângulo x, devemos aproximar x por uma sequência de somas e subtrações de ângulos $\theta_n = \arctan 1/2^n$.

Basicamente, o algoritmo funciona assim: dado um ângulo x, devemos aproximar x por uma sequência de somas e subtrações de ângulos $\theta_n = \arctan 1/2^n$. É melhor explicar com um exemplo.

Basicamente, o algoritmo funciona assim: dado um ângulo x, devemos aproximar x por uma sequência de somas e subtrações de ângulos $\theta_n = \arctan 1/2^n$. É melhor explicar com um exemplo.

n	Aproximação de θ_n
0	0.785398 rad
1	0.463648 rad
2	0.244979 rad
3	0.124355 rad
4	0.062419 rad

- Eventualmente, podemos parar de calcular estes valores pois $\tan\theta \approx \theta$, para θ pequeno.

Basicamente, o algoritmo funciona assim: dado um ângulo x, devemos aproximar x por uma sequência de somas e subtrações de ângulos $\theta_n = \arctan 1/2^n$. É melhor explicar com um exemplo.

n	Aproximação de θ_n
0	0.785398 rad
1	0.463648 rad
2	0.244979 rad
3	0.124355 rad
4	0.062419 rad

- Eventualmente, podemos parar de calcular estes valores pois $\tan\theta \approx \theta$, para θ pequeno.
- Por exemplo, para $n = 9$ ($\theta_n = \arctan 1/512$), esta aproximação já tem 8 casas decimais de precisão.

Tome $x = 1$ rad. Vamos aproximar este ângulo utilizando os θ_n.

Tome $x = 1$ rad. Vamos aproximar este ângulo utilizando os θ_n.

$$x_0 = \theta_0 \approx 0.785398$$

Tome $x = 1$ rad. Vamos aproximar este ângulo utilizando os θ_n.

$$x_0 = \theta_0 \approx 0.785398$$

Muito pequeno, então somamos:

$$x_1 = \theta_0 + \theta_1 \approx 1.249046$$

Tome $x = 1$ rad. Vamos aproximar este ângulo utilizando os θ_n.

$$x_0 = \theta_0 \approx 0.785398$$

Muito pequeno, então somamos:

$$x_1 = \theta_0 + \theta_1 \approx 1.249046$$

Passou, então subtraímos:

$$x_2 = \theta_0 + \theta_1 - \theta_2 \approx 1.004067$$

Etc.

Tome $x = 1$ rad. Vamos aproximar este ângulo utilizando os θ_n.

$$x_0 = \theta_0 \approx 0.785398$$

Muito pequeno, então somamos:

$$x_1 = \theta_0 + \theta_1 \approx 1.249046$$

Passou, então subtraímos:

$$x_2 = \theta_0 + \theta_1 - \theta_2 \approx 1.004067$$

Etc. Em cada passo, o computador deve verificar se a aproximação obtida é maior ou menor (operação rápida) que o número desejado.

Enquanto aproximamos o ângulo x, vamos aproximando $\cos x$ e $\sin x$, basta ir multiplicando as matrizes correspondentes. Por exemplo, no terceiro passo, vamos ter

$$\begin{pmatrix} \cos x_2 \\ \sin x_2 \end{pmatrix} = R_{-\theta_2} R_{\theta_1} R_{\theta_0} \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$
$$= c \begin{pmatrix} 1 & \frac{1}{4} \\ -\frac{1}{4} & 1 \end{pmatrix} \begin{pmatrix} 1 & -\frac{1}{2} \\ \frac{1}{2} & 1 \end{pmatrix} \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

onde $c = \cos(-\theta_2)\cos\theta_1\cos\theta_0$.

Enquanto aproximamos o ângulo x, vamos aproximando $\cos x$ e $\sin x$, basta ir multiplicando as matrizes correspondentes. Por exemplo, no terceiro passo, vamos ter

$$\begin{pmatrix} \cos x_2 \\ \sin x_2 \end{pmatrix} = R_{-\theta_2} R_{\theta_1} R_{\theta_0} \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$= c \begin{pmatrix} 1 & \frac{1}{4} \\ -\frac{1}{4} & 1 \end{pmatrix} \begin{pmatrix} 1 & -\frac{1}{2} \\ \frac{1}{2} & 1 \end{pmatrix} \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

onde $c = \cos(-\theta_2)\cos\theta_1\cos\theta_0$. Fazendo apenas 3 passos, temos $\cos x_2 \approx 0.5368$, $\sin x_2 \approx 0.8436$. Enquanto que $\cos 1 \approx 0.5403$ e $\sin 1 \approx 0.8414$.

Então, se o algoritmo rodar $n+1$ passos, teríamos algo do tipo

$$\begin{pmatrix} \cos x \\ \sin x \end{pmatrix} \approx \begin{pmatrix} \cos x_n \\ \sin x_n \end{pmatrix} = cR_{\pm\theta_n} \cdots R_{\pm\theta_1} R_{\pm\theta_0} \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

onde $c_n = \cos\theta_n \cdots \cos\theta_1 \cos\theta_0$.

Então, se o algoritmo rodar $n+1$ passos, teríamos algo do tipo

$$\begin{pmatrix} \cos x \\ \sin x \end{pmatrix} \approx \begin{pmatrix} \cos x_n \\ \sin x_n \end{pmatrix} = cR_{\pm\theta_n} \cdots R_{\pm\theta_1} R_{\pm\theta_0} \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

onde $c_n = \cos\theta_n \cdots \cos\theta_1 \cos\theta_0$. Problema: ainda temos que lidar com o c_n.

O pulo do gato

O pulo do gato

O pulo do gato

Quem garante que o nosso método realmente converge?

O pulo do gato

Quem garante que o nosso método realmente converge?

Theorem

CORDIC Utilizando o método descrito anteriormente, temos que

$$\begin{cases} |\cos x - \cos x_{n+1}| < 1/2^n \\ |\sin x - \sin x_{n+1}| < 1/2^n \end{cases}$$

se $-\pi/2 \leq x \leq \pi/2$.

O pulo do gato

Quem garante que o nosso método realmente converge?

Theorem

CORDIC Utilizando o método descrito anteriormente, temos que

$$\begin{cases} |\cos x - \cos x_{n+1}| < 1/2^n \\ |\sin x - \sin x_{n+1}| < 1/2^n \end{cases}$$

se $-\pi/2 \leq x \leq \pi/2$.

O teorema acima nos diz de maneira precisa como estimar o erro das aproximações. Por exemplo, se quisermos uma aproximação com 10 casas decimais, poderíamos tomar $n+1 = 40$ e parar de aproximar sempre em $n = 39$, pois

$$\frac{1}{2^{39}} \approx 1.8189 \times 10^{-12}.$$

O pulo do gato - continuação

Mas isso acaba nosso problema! Pois se fixarmos que vamos parar sempre em $n = 39$, $c_{39} = c$ será uma constante (pois cosseno independe do sinal).

O pulo do gato - continuação

Mas isso acaba nosso problema! Pois se fixarmos que vamos parar sempre em $n = 39$, $c_{39} = c$ será uma constante (pois cosseno independe do sinal). Utilizando qualquer outro método, temos

$$c = \cos\theta_{39} \cdots \cos\theta_1 \cos\theta_0 \approx 0.607252935$$

O pulo do gato - continuação

Mas isso acaba nosso problema! Pois se fixarmos que vamos parar sempre em $n = 39$, $c_{39} = c$ será uma constante (pois cosseno independe do sinal). Utilizando qualquer outro método, temos

$$c = \cos\theta_{39} \cdots \cos\theta_1 \cos\theta_0 \approx 0.607252935$$

Logo, a única operação não-rápida que estamos fazendo é uma multiplicação por uma constante (que já fica implementada no hardware) no final. Observe que, como os θ_n também já estão pré-definidos, então podemos também implementá-los direto no hardware.

Não acabou!

O mais impressionante, é que este mesmo algoritmo pode ser adapatado para calcular qualquer função de uma calculadora científica comum:

Não acabou!

O mais impressionante, é que este mesmo algoritmo pode ser adapatado para calcular qualquer função de uma calculadora científica comum:

- Funções hiperbólicas;
- exponencial;
- logaritmo;
- raízes.

Não acabou!

O mais impressionante, é que este mesmo algoritmo pode ser adapatado para calcular qualquer função de uma calculadora científica comum:

- Funções hiperbólicas;
- exponencial;
- logaritmo;
- raízes.

Porém, os detalhes são mais técnicos e algumas adaptações são até proprietárias.

Não acabou!

O mais impressionante, é que este mesmo algoritmo pode ser adapatado para calcular qualquer função de uma calculadora científica comum:

- Funções hiperbólicas;
- exponencial;
- logaritmo;
- raízes.

Porém, os detalhes são mais técnicos e algumas adaptações são até proprietárias.

Copiado descaradamente de:

1. Alan Sultan, *CORDIC: How Hand Calculators Calculate*, The College Mathematics Journal (MAA), Vol. 40, nº 2, 2009, pg. 87-92.
2. J. Underwood and B. Edwards, How do calculators calculate trigonometric functions? Educational Resources Information Center (ERIC) document ED461519.
3. `http://en.wikipedia.org/wiki/CORDIC`