三角函数逼近快速算法(正余弦)

原文出自：

http://lab.polygonal.de/2007/07/18/fast-and-accurate-sinecosine-approximation/

里面提到了查表，采用查表并配合插值；以及泰勒级数

看过第一篇的文章后，大呼过瘾！原文作者的思路非常简捷，有趣，偶英语比较差，欢迎指正，废话不多说看文章

原文出处：

http://www.devmaster.net/forums/showthread.php?t=5784

http://lab.polygonal.de/2007/07/18/fast-and-accurate-sinecosine-approximation/

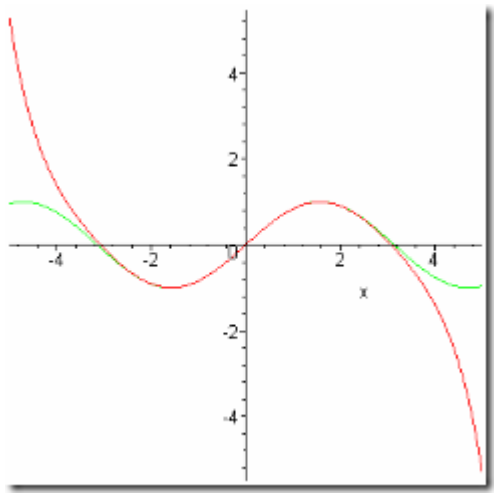
在某些情况下我们需要一些更高效的且近似于标准值的 sin 和 cos 函数。

有 时候我们并需要过高的精度，这时 C 语言中自带的三角函数（sinf() 和 cosf() f）计算的精度超出了我们所需要的精度要求，所以其效率很低。我们真正需要的是间于精度和效率的一个折中的方案。众所周知的取近似值的方法是：泰勒级数 （和著名的马克劳林级数）

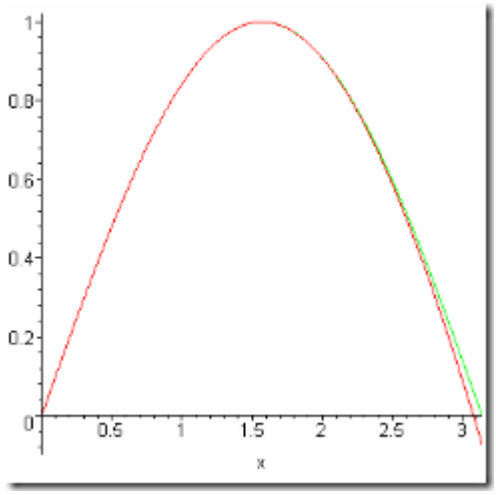
代码是：

x - 1/6 x^3 + 1/120 x^5 - 1/5040 x^7 + ...

我们绘制了下图:



绿线是标准的 sin 函数，红线是 4 项泰勒级数展开式。这个近似值的效果看起来还不错，但是如果你仔细观察后会发现



它在 pi/2 之前的效果还是很好的，但是超过了 pi/2 后就开始快速偏离标准 sin。它在 pi 处的值比原来的 0 多了 0.075.用这个方法来模拟波动忽动忽停，看起来很呆板，这个效果当然不是我们想要的。

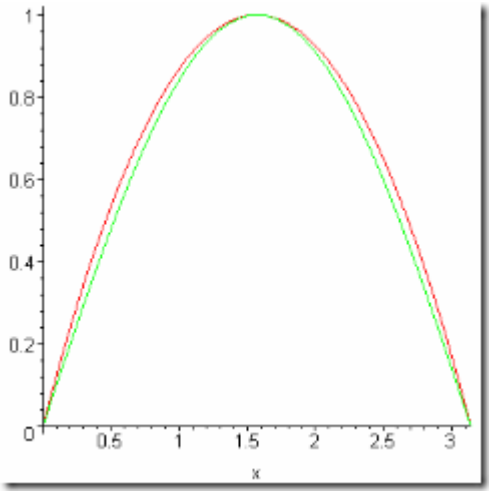
我们可以继续增加泰勒级数项的个数来减小误差，但是这将导致我们的公式非常的冗长。用 4 项的泰勒级数展开式需要我们进行 7 次乘法和 3 次加法来完成。泰勒级数不能同时满足我对精度和效率的要求。

刚 刚近似如果能满足 sin（pi）=0 就好了。从上图我还可以发现另一件事：这个曲线看起来很象抛物线！所以我们来寻找一个尽可能和 sin 接近的抛物线（公式）。抛物线的范式方程是：A + B x + C x^2.这个公式可以让们控制三个自由度。显然我们需要其满足 sine(0) = 0, sine(pi/2) = 1 and sine(pi) = 0. 这样我们就得到了 3 个等式。A + B 0 + C 0^2 = 0

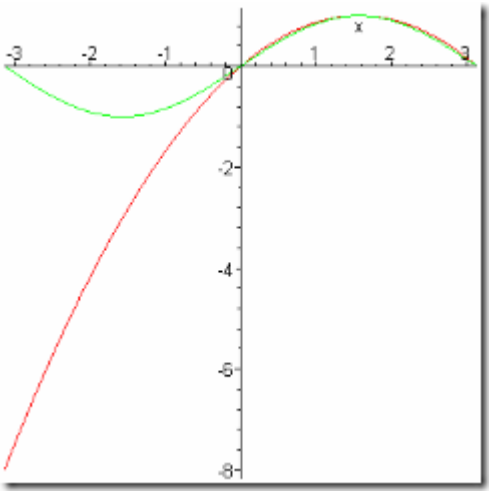
A + B pi/2 + C (pi/2)^2 = 1

A + B pi + C pi^2 = 0

解得：A = 0, B = 4/pi, C = -4/pi^2.我们的抛物线诞生啦！



貌似这个的误差看起来比泰勒级数还要遭。其实不是的！这种方法的最大误差是0.056.（译者：而且这种近似值没有误差积累）而且这个近似值的绘制出的波动是光滑的，而且只需要 3 次乘法和一次加法。不过它还不够完美。下图是[-pi, pi]之间的图像：

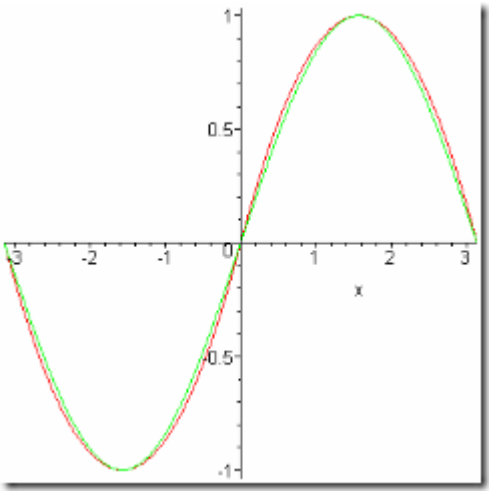


显然我们至少需要它在 1 个完整的周期内都符合我们要求。但是我们可以看出，我们缺少的另一半是原抛物线的一个映射。它的公式是：4/pi x + 4/pi^2 x^2。所以我们可以直接这样写：

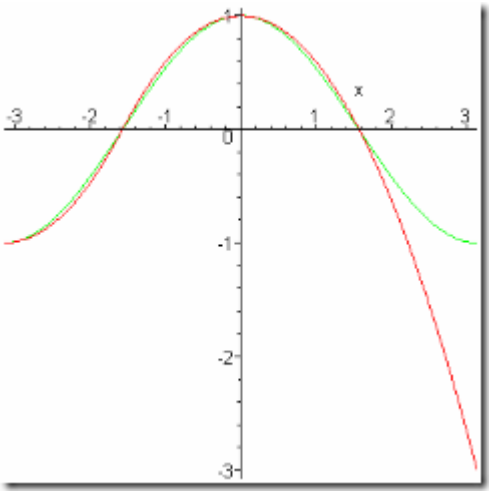
Code:

```
if(x > 0) { y = 4/pi x - 4/pi^2 x^2; } else { y = 4/pi x + 4/pi^2 x^2; }
```

添 加一个条件分支不是一个好的方法。它会让程序渐渐的变慢。但是观察一下我们模拟的和标准的图像是多么的接近啊！观察上面两式子，只是中间的一项正负号不 同，我的第一个单纯的想法是可以提取 x 的正负号来消除分支，即使用：x / abs(x)。除法的代价是非常大的，我们来观察一下现在的公式： 4/pi x - x / abs(x) 4/pi^2 x^2。将除法化简后我们得到：4/pi x - 4/pi^2 x abs(x). 所以只需要多一步运算我们就得到了我们需要的 sin 逼近值！ 下图是结果



接下来我们要考虑 cos。有基础的三角公理可以知道：cos(x) = sin(pi/2 + x). 把 x 多加一个 pi/2 就可以搞定了？事实上它的某一部分不是我们期望得到的。



我们需要做的就是当 x > pi/2 时“跳回”。这个可以由减去 2 pi 来实现。

Code:

```
x += pi/2;

if(x > pi) // Original x > pi/2 { x -= 2 * pi; // Wrap: cos(x) = cos(x - 2 pi)}

y = sine(x);
```

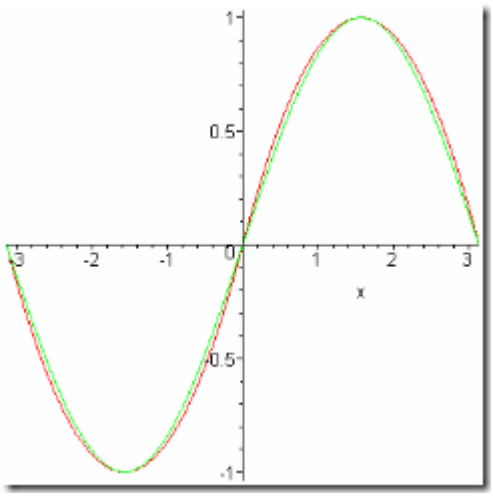
又出现了一个分支，我们可以用逻辑“与”来消除它，像是这样：

```
x -= (x > pi) & (2 * pi);
```

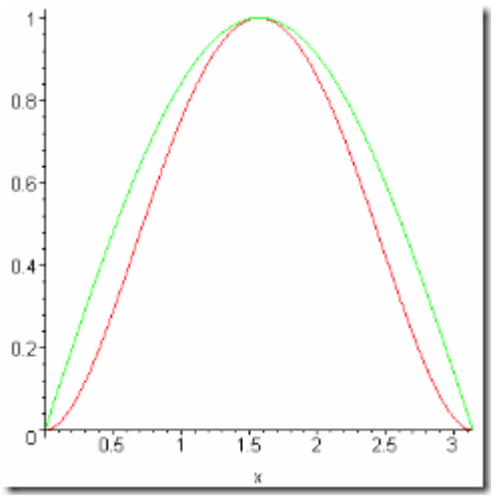
Code:

```
x -= (x > pi) & (2 * pi);
```

注 意这并不是 c 的源代码。但是这个应该可以说明它是怎么样运行的。当 x > pi 是 false 时，逻辑“与”（&）运算后得到的是 0，也就是（x-=0）大小没有改变，哈哈完美的等价!我会给读这篇文章的读者留一些关于这个练习。虽然 cos 比 sin 需要多一些运算，但是相比之下貌似也没有更好方法可以让程序更快了。现在我们的最大误差是 0.056 ，四项泰勒级数展开式每一次都会有一点点误差。再来看看我们 sin 函数：



现在是不是不能继续提升精准度了呢？当前的版本已经可以满足大多度 sin 函数的应用了。但是对一些要求更高一些的程序现在做的还够。
仔细观察图像，你会注意到我们的近似值总是比真实值大，当然除了 0，pi/2 和 pi。所以我们要做的就是在不改变这些点（0，pi/2 和 pi）的情况下，将函数再“按下去”一些。解决方法是利用抛物线的平方。看起来就像这样：

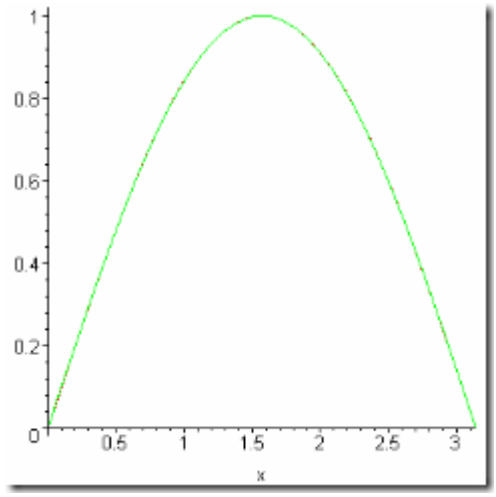


注意它保持着原来那些关键点，不同的是它比真实的 sin 函数值更低了。所以我们可以用一个加权的平均值来使两个函数更接近。

Code:

Q (4/pi x - 4/pi^2 x^2) + P (4/pi x - 4/pi^2 x^2)^2

利 用 Q + P = 1. 你可以灵活的控制绝对误差或相对误差。别急我来告诉你取不同的极限结果时 Q，P 的值。绝对误差的最佳权值是：Q = 0.775, P = 0.225 ；相对误差的最佳权值是：Q = 0.782，P = 0.218 。让我们来看一下结果的图像。



红 线呢？它几乎被绿线完全覆盖了，这足以证明我们的近似十分完美。最大误差是 0.001，50 倍的提升！这个公式看起来很长，但是括号里面的公式最终得到的 值是相同的，也就是说括号里的只需要被计算一次。事实上在原来的基础上只是增加了额外的 2 次乘法和 2 次加法就可以得到现在的结果。

先别高兴的太早，我们还要“制造”一个负号出来。我们需要增加一个 abs()运算。最终的 c 代码是：

Code:

```
float sine(float x)

{

const float B = 4/pi;

const float C = -4/(pi*pi);

float y = B * x + C * x * abs(x);

#ifdef EXTRA_PRECISION // const float Q = 0.775;

const float P = 0.225;

y = P * (y * abs(y) - y) + y;

// Q * y + P * y * abs(y)
```

#endif }
所以我们仅仅是需要多加 5 次乘法和 3 次加法就可以完成了。如果我们忽略 abs()这个仍然是比 4 项泰勒级数展开式快，更精准！Cos 只需要相应的变换一下 x 就可以了。

(译者注：后面是汇编程序，不翻译了)

part2

我 选取了最小误差的情况，用 as3 运行后发现提升了 14 倍，而且仍然是非常精准。不过你必须直接使用它，不能把它放到一个函数中，因为每调用一次额外的函数 调用会削减执行效率，最终你会得到一个比 Math.sin() 和 Math.cos()效率更差的结果。 还有这里会用到的三角定理：

cos(x) = sin(x + pi/2)

cos(x - pi/2) = sin(x)

下载: fastTrig.as.

可以清楚到对比结果，现在你可以用这个替换 Math.sin() 和 Math.cos()了

哇哦！！！几乎是相同的精准度（14 倍速度提升）

```
//always wrap input angle to -PI..PI
if (x< -3.14159265)
	x+= 6.28318531;
else
if (x> 3.14159265)
	x-= 6.28318531;
//compute sine
if (x< 0)
	sin= 1.27323954 * x+ .405284735 * x* x;
else
	sin= 1.27323954 * x- 0.405284735 * x* x;
//compute cosine: sin(x + PI/2) = cos(x)
x+= 1.57079632;
if (x> 3.14159265)
	x-= 6.28318531;
if (x< 0)
	cos= 1.27323954 * x+ 0.405284735 * x* x
else
	cos= 1.27323954 * x- 0.405284735 * x* x;
}
```

High precision sine/cosine (~8x faster)

```
//always wrap input angle to -PI..PI
if (x< -3.14159265)
       x+= 6.28318531;
else
if (x> 3.14159265)
       x-= 6.28318531;
//compute sine
if (x< 0)
{
       sin= 1.27323954 * x+ .405284735 * x* x;
if (sin< 0)
               sin= .225 * (sin*-sin- sin) + sin;
else
               sin= .225 * (sin* sin- sin) + sin;
}
else
{
       sin= 1.27323954 * x- 0.405284735 * x* x;
if (sin< 0)
               sin= .225 * (sin*-sin- sin) + sin;
else
               sin= .225 * (sin* sin- sin) + sin;
}
//compute cosine: sin(x + PI/2) = cos(x)
x+= 1.57079632;
if (x> 3.14159265)
       x-= 6.28318531;
if (x< 0)
{
       cos= 1.27323954 * x+ 0.405284735 * x* x;
if (cos< 0)
               cos= .225 * (cos*-cos- cos) + cos;
else
               cos= .225 * (cos* cos- cos) + cos;
}
else
{
       cos= 1.27323954 * x- 0.405284735 * x* x;
if (cos< 0)
               cos= .225 * (cos*-cos- cos) + cos;
else
```

```
cos= .225 * (cos* cos- cos) + cos;
}
```

Tag 标签: as3, 近似值, sin, cos, 优化

+++

:

Fast and accurate sine/cosine approximation

July 18, 2007 on 2:31 pm | In Actionscript | 50 Comments

Trigonometric functions are costly operations and can slow down your application if they are extensively used. There are two reasons why: First, Math.sin() is a function, and thus needs a function call which simple eats up some time. Second, the result is computed with much more precision than you would ever need in most situations.

Most often you just want the periodic wave-like characteristics of the sine or cosine, which can be approximated in various ways. One common way of making it faster is to create a *lookup-table* by computing the sine at discrete steps and storing the result in an array. For example:

```
var sineTable:Array = [];
for (var i:int = 0; i < 90; i++)
{
       sineTable[i] = Math.sin(Math.PI/180 * i)
}
```

Due to the symmetry of the sine wave, it's sufficient to compute one quadrant only (0..pi/2), and the other 3/4's of the circle can be computed by shifting and wrapping the input value. The biggest drawback is that the values are stored at a fixed resolution and so the result is not very accurate. This can be enhanced with linear interpolation:

```
x = 22.5;
y = sineTable[int(x)] + (sineTable[int(x + .5)] - sineTable[int(x)]) / 2;
```

Much better, but yet the error exists. It also involves accessing array elements which makes the code rather slow. Another technique uses *taylor series approximation*:

sin(x) = x - (x^3)/3! + (x^5)/5! - (x^7)/7! + ...

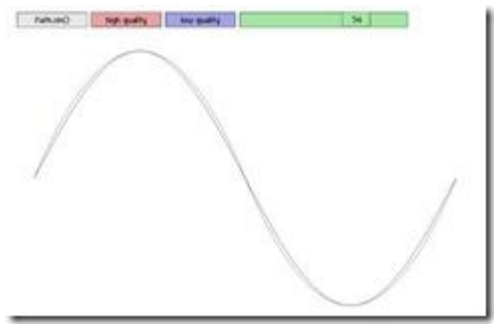
Like with the lookup-table, evaluating this term is costly.

After searching for alternatives, I finally found a fantastic solution using a quadratic curve which blows everything away in terms of performance *and* accuracy. For a detailed derivation, please follow the link because I won't go into it.

I did minor optimizations to figure out what AS3 likes most, and arrived at some code that can be up to 14x faster, while still being very accurate. However, you have to use it directly - do not place the code inside a function, because the additional function call sweeps out the performance gain, and you are left with an approximation that is actually *slower* compared to a native Math.sin() or Math.cos() call. Also note that cos(x) = sin(x + pi/2) or cos(x - pi/2) = sin(x), so computing the cosine is just of matter adding pi/2 to the input value.

Download source: fastTrig.as.

Below is a simple visualization to show you the quality of the approximation. The high precision version can replace the Math.sin() and Math.cos() calls in nearly all situations.



Low precision sine/cosine (~14x faster)

```
//always wrap input angle to -PI..PI
if (x < -3.14159265)
	x += 6.28318531;
else
if (x > 3.14159265)
	x -= 6.28318531;
//compute sine
if (x < 0)
	sin = 1.27323954 * x + .405284735 * x * x;
else
```

```
        sin = 1.27323954 * x - 0.405284735 * x * x;
//compute cosine: sin(x + PI/2) = cos(x)
x += 1.57079632;
if (x > 3.14159265)
        x -= 6.28318531;
if (x < 0)
        cos = 1.27323954 * x + 0.405284735 * x * x
else
        cos = 1.27323954 * x - 0.405284735 * x * x;
}
```

High precision sine/cosine (~8x faster)

```
//always wrap input angle to -PI..PI
if (x < -3.14159265)
        x += 6.28318531;
else
if (x > 3.14159265)
        x -= 6.28318531;
//compute sine
if (x < 0)
{
        sin = 1.27323954 * x + .405284735 * x * x;
if (sin < 0)
                sin = .225 * (sin *-sin - sin) + sin;
else
                sin = .225 * (sin * sin - sin) + sin;
}
else
{
        sin = 1.27323954 * x - 0.405284735 * x * x;
if (sin < 0)
                sin = .225 * (sin *-sin - sin) + sin;
else
                sin = .225 * (sin * sin - sin) + sin;
}
//compute cosine: sin(x + PI/2) = cos(x)
x += 1.57079632;
if (x > 3.14159265)
        x -= 6.28318531;
if (x < 0)
{
        cos = 1.27323954 * x + 0.405284735 * x * x;
if (cos < 0)
```

```
                cos = .225 * (cos *-cos - cos) + cos;
else
                cos = .225 * (cos * cos - cos) + cos;
}
else
{
        cos = 1.27323954 * x - 0.405284735 * x * x;
if (cos < 0)
                cos = .225 * (cos *-cos - cos) + cos;
else
                cos = .225 * (cos * cos - cos) + cos;
}
```

50 COMMENTS ?

RSS feed for comments on this post. TrackBack URI

1. Nice find, though I think I'd have to be in a real optimization bind before I replaced all my sin/cos calls with all that stuff. :)

Comment by Keith Peters — July, 18 2007 #

2. Once again, incredible post.

There is an error in the link. The good link is : http://lab.polygonal.de/wp-content/articles/fast_trig/fastTrig.as :

```
//1.27323954   = 4/pi
//0.405284735 =-4/(pi^2)
/*************************************************************
 * low precision sine/cosine
 *************************************************************/

//always wrap input angle to -PI..PI
if (x < -3.14159265)
        x += 6.28318531;
else
if (x >  3.14159265)
        x -= 6.28318531;

//compute sine
if (x < 0)
        sin = 1.27323954 * x + .405284735 * x * x;
else
        sin = 1.27323954 * x - 0.405284735 * x * x;
```

```
//compute cosine: sin(x + PI/2) = cos(x)
x += 1.57079632;
if (x >  3.14159265)
	x -= 6.28318531;

if (x < 0)
	cos = 1.27323954 * x + 0.405284735 * x * x
else
	cos = 1.27323954 * x - 0.405284735 * x * x;
}

/**************************************************************
 * high precision sine/cosine
 *************************************************************/

//always wrap input angle to -PI..PI
if (x < -3.14159265)
	x += 6.28318531;
else
if (x >  3.14159265)
	x -= 6.28318531;

//compute sine
if (x < 0)
{
	sin = 1.27323954 * x + .405284735 * x * x;

	if (sin < 0)
		sin = .225 * (sin *-sin - sin) + sin;
	else
		sin = .225 * (sin * sin - sin) + sin;
}
else
{
	sin = 1.27323954 * x - 0.405284735 * x * x;

	if (sin < 0)
		sin = .225 * (sin *-sin - sin) + sin;
	else
		sin = .225 * (sin * sin - sin) + sin;
}

//compute cosine: sin(x + PI/2) = cos(x)
```

```
x += 1.57079632;
if (x >    3.14159265)
        x -= 6.28318531;

if (x < 0)
{
        cos = 1.27323954 * x + 0.405284735 * x * x

        if (cos < 0)
                cos = .225 * (cos *-cos - cos) + cos;
        else
                cos = .225 * (cos * cos - cos) + cos;
}
else
{
        cos = 1.27323954 * x - 0.405284735 * x * x;

        if (cos < 0)
                cos = .225 * (cos *-cos - cos) + cos;
        else
                cos = .225 * (cos * cos - cos) + cos;
}
```

Comment by Jerome — July, 18 2007 #

3. thank you, fixed now.

Comment by Michael — July, 18 2007 #

4. Woah, that’s sick! Thanks for sharing, I always stopped at the first step :/

Comment by Mr.doob — July, 19 2007 #

5. Awesome! Let’s see how PV3D can benefit from that! ;-)

Comment by Ralph Hauwert — July, 19 2007 #

6. The article on DevMaster mentions that you should avoid branching, especially in loops. I wonder if the same problem applies to AS3? Is there a speed improvement if you replace the branches by binary logic twists?

Comment by Patrick — July, 20 2007 #

7. Hi, it didn’t work for me (the author also says that this is not valid C code at all) Binary & converts the result to an integer. Only this works: x -= int(x > y) && z;

But the code above is a lot slower than the pure if..else statements.

Comment by Michael — July, 20 2007 #

8. Nice work Michael – this one goes a long way back. I think it may have been in Hart’s ‘Computer Approximations.’

One thing I’ve found helpful when working with compilers that don’t support function inlining is to use the C preprocessor or awk or something similar to do all my debugging with library calls and then have the tool replace them with inline approximations going into production.

Another benefit of having inline code is that you can combine the sin or cos computation with whatever equation it’s used in and realize additional gains by having the trig function result immediately available in a register.

regards,

– jim

Comment by Jim Armstrong — July, 20 2007 #

9. hey cool I didn’t know there is even a whole book about approximations only ;-) talking about preprocessors, what IDE do you use ? I was trying to integrate the C preprocessor into FlashDevelop, but failed at configuring make or ant to use it.

Comment by Michael — July, 21 2007 #

10. Michael:

Another classic is Cody’s ‘A Software Manual for the Elementary Functions’. There is a relatively new book by Muller – ‘Elementary Functions: Algorithms and Implementation’, but I haven’t read it yet.

I’ve given up on direct integration of other tools to any IDE – don’t have the bandwidth to work out all the issues, so I run awk or another text processor directly on the .AS files — very old school.

Best of luck in your continued (and superb) efforts!

regards,

- jim

Comment by Jim Armstrong — July, 22 2007 #

11. Hey thanks for the reference!
Doing University and making a blog isn’t easy but shortly some content will popout =D

Comment by Eduardo — August, 9 2007 #

12. [...] Fast and accurate sine/cosine approximation More code optimization to do sine/cosine operations without using the Math class. [...]

Pingback by ? Blog Archive ? Links: Mathematical themes with Actionscript polyGeek.com — August, 26 2007 #

13. [...] Fast and accurate sine/cosine approximation [...]

Pingback by ?????¨?????? BLOG ? Blog Archive ? as3????????§??????□???????’??“ — September, 19 2007 #

14. Here’s a modified fixed-point version that performs better : http://blog.haxe.org/entry/26

Comment by Nicolas — November, 12 2007 #

15. I’ev tested this in AS2, and it’s not faster. So this optimisation trick is ONLY for AS3. Just to let you know.

Comment by JP — January, 8 2008 #

16. Thanks. This post was useful and more accurate than using an MS-Excell trendline function (2nd degree). Made minor changes that should speed things up since addition is cheaper than mult usually. This is the c code:
（注：我本人已经亲自测试过 OK,性能很好---tang）

```
double sin(double rad){
double sine;
if (rad < 0)
sine = rad*(1.27323954 + .405284735 * rad);
else
sine = rad*(1.27323954  -  0.405284735 * rad);
```

```
if (sine < 0)
sine = sine*(-0.225 * (sine + 1) + 1);
else
sine = sine*(0.225 * (sine  -  1) + 1);
return sine;
}
```

Comment by DYessick — January, 28 2008 #

17. @DYessick: I tried your version in AS3 and there is no noticeable difference unfortunately. Probably due to the weirdness of AS3.

Overall the high precision approximation gives a nice boost though, although it is closer to ~6.7 faster on my tests. This could depend on a variety of factors I imagine, not least of all your processor architecture.

Comment by brainclog — February, 4 2008 #

18. [...] polygonal labs ??????? Fast and accurate sine/cosine approximation (tags: optimization math) [...]

Pingback by fcicq's del.icio.us ? Blog Archive ? links for 2008-02-09 — February, 9 2008 #

19. You wrote:

"[...] it????????s sufficient to compute one quadrant only (0..pi), and the other 3/4????????s [...]"

But … 1/4 of 2pi is pi/2 and not pi, so a quadrant is (0..pi/2), i'm wrong ?

Greg

Comment by Gregoire — March, 4 2008 #

20. thanks greg, you are right. i have fixed the article.

Comment by Michael — March, 5 2008 #

21. More over, you can use a Quadrant but an Octant is sufficient due to the symetry (x y, -x y, -x -y, x -y, y x, -y x, -y -x, y -x)

Comment by Passault Gr????goire — March, 18 2008 #

22. This is about 50% slower than using Math.cos(); ive found…I don’t get it? I’m not using it in a function either.

Comment by ColbyCheeze — April, 12 2008 #

23. always compile in release mode and use the release player (not the debug version) for doing benchmarks.

Comment by Michael — April, 12 2008 #

24. That makes a difference? Damn I didn’t know that…I’ll have to go find my release player, haven’t even used that thing in forever.

Comment by ColbyCheeze — April, 14 2008 #

25. [...] Fast and accurate sine/cosine approximation [...]

Pingback by Rozengain.com - New Media Development Blog ? Blog Archive ? Some ActionScript 3.0 Optimizations — August, 11 2008 #

26. [...] Fast and accurate sine/cosine approximation [...]

Pingback by Optimizations links | JADBOX: Web Application Musings — August, 11 2008 #

27. This may be a bit outdated - I think Adobe took pretty harsh to your approximation genius and generated a super fast trig table…

Compiling using the Flex 3.2 SDK, I get these results…

iterations: 1,000,000
algorithm: FastMath [high-quality approximation]
angle: 1.302888286613767

FastMath.sin(1.302888286613767) = 0.9643253756148902
- Time: 475 ms

…

iterations: 100,000
algorithm: Math [flash]
angle: 1.302888286613767

Math.sin(1.302888286613767) = 0.9643267785343497
- Time: 167 ms

…

Can anyone else confirm this? Thanks!

Comment by Matt Bolt — January, 2 2009 #

28. I also made a mistake when typing the last post…. The iterations for the Math.sin() test were also = 1,000,000 - *not* 100,000.

Comment by Matt Bolt — January, 2 2009 #

29. hi matt,

I cannot confirm this. make sure you are using the mxmlc compiler with debug=false and run the swf inside a release player. otherwise you get misleading results. with the Flex3.2 SDK the high quality approximation is now almost exactly 10x faster on my machine.

Comment by Michael — January, 3 2009 #

30. [...] Optimización de operaciones con seno y cosenohttp://lab.polygonal.de/2007/07/18/fast-and-accurate-sinecosine-approximation/ [...]

Pingback by Rendimiento y Optimización de AS3 | ?y por qué no? — January, 23 2009 #

31. Thanks alot, this really helps. But I was wondering if anyone could tell me where the numbers 0.405284735 and 0.225 came from?

Comment by Chris — January, 28 2009 #

32. Hi,

I know this is outdated, but these results may help somebody - these tests were done in Flash Lite 2.0 on a Nokia 6121 (I will test on more phones)

Calculating both sin and cos of 100 angles using Math.sin and Math.cos
= 6216 ms

Calculating the same thing using FastMath inline
= 4685 ms

In Flash Lite FastMath is around 25-30% faster

Just a note (it says this in the post too) that wrapping FastMath in a function call negates the speed increase, and you get the same results as just using Math.sin/cos.

Comment by Peter — February, 7 2009 #

33. And sorry ‘of 100 angles’ should be ‘of 1000 angles’…

Comment by Peter — February, 7 2009 #

34. [...] Fast and accurate sine and cosine approximation [...]

Pingback by Fast and accurate sine and cosine approximation — Some Random Dude — February, 21 2009 #

35. for sin, radians 0 <= x <= pi
temp = x * 2 / pi = x * 0.636619…
sin = temp * (2 - temp)
sin = sin * (0.225 * (sin - 1) + 1)

without refinement step, maximum error is 0.056 at +- 0.47
with refinement, maximum error is 0.001 at +- 1.38

for degrees, swap pi with 90

Comment by Eric Cole — February, 22 2009 #

36. sorry, swap pi with 180, so temp = x / 90

Comment by Eric Cole — February, 22 2009 #

37. [...] in January I saw this blog post by Michael Baczynski over at http://lab.polygonal.de/. The blog post describes a technique for fast [...]

Pingback by Low Precision Sine and Cosine — March, 21 2009 #

38. [...] a big fan of Polygonal Labs, and Michael has numerous posts such as this one that deal with fast approximation algorithms. This is another good one from his Actionscript [...]

Pingback by Actionscript Optimization ? The Algorithmist — March, 27 2009 #

39. [...] polygonal labs ? Fast and accurate sine/cosine approximation [...]

Pingback by polygonal labs ? Fast and accurate sine/cosine approximation — Some Random Dude — March, 27 2009 #

40. [...] Fast sin/cosine approximation [...]

Pingback by Flash and the art of optimization ? whitenoise — June, 19 2009 #

41. [...] polygonal labs - Fast and accurate sine/cosine approximation [...]

Pingback by sine approximation with fixed point math ? console-dev.de — July, 6 2009 #

42. Hi

i want draw sine curve same as above but i need text insted of pixel

Comment by Sweety — December, 1 2009 #

43. [...] os usamos intensamente como por exemplo em sistemas de partículas. Por isso uma leitura a este post do Michael Baczynski é [...]

Pingback by Comunidade Portuguesa de Rich Internet Applications? Blog Archive ? Introdu??o ao seno e coseno com Actionscript — January, 26 2010 #

44. [...] os usamos intensamente como por exemplo em sistemas de partículas. Por isso uma leitura a este post do Michael Baczynski é [...]

Pingback by Introdu??o ao seno e coseno com Actionscript - redeRIA | Agregador de noticias, artigos, tutoriais Flex, Flash, JavaFX, AJAX e Rich internet applications em geral! — January, 26 2010 #

45. Thanks for a wonderful post, l ve been looking for such information, I will join jour rss feed now.

Comment by Mike — March, 22 2010 #

46. Thanks for this. Excellent post.

Does anyone have a similar technique for Atan? I’ m scratching my head a bit on this one.

Thanks.

Comment by MikeBee — April, 27 2010 #

47. For help with atan/atan2 check out this thread for starters. Hope it helps.

Comment by Peter — April, 28 2010 #

48. And this fast atan2 with very low error also seems sound.

Comment by Peter — April, 28 2010 #

49. Man, thanks for taking the time to post this. I'm sure you've been getting a few of these generic thank-you's, but this blog is basically a godsend for us as3 game devs!

Comment by Victor Reano — May, 7 2010 #

50. AS3…

Code Practices Use constants when appropriate Only use variables if the value is going to change otherwise use constants. \\ References vs. lookups In general,……

Trackback by Confluence: Social City — July, 9 2010 #