# 关于DSP的一些反正切查表算法的编写

一直在拿TMS20F28335做电机控制，也从早前的简单寄存器IO口配置时代过渡到了真正的算法时代,很不适应，起初去查各种书籍,网上搜索各种网站，发现书本上讲的很多都是快速傅里叶啊啥的,找不到很原始的像啥低通滤波器,正弦,正切这些函数的编写。

起初写程序发现，C2000的库里支持啥正弦函数，浮点预算，反正弦函数，开根运算等算法，好兴奋。用了才知道，太慢了，根本没法用在工程上。

关于三角函数的编写，其实都是利用查表发，查表法大家应该都懂。这个就不再赘述，怎样去查一个正弦表呢？这里做一个基本的介绍。

**首先**，你可以自己写一个查表数组，也可以在程序初始化生成一个。

Eg：

**void** SinCOS\_TAB(**void**)

{

**unsigned** **char** i;

**float** Theta;

**for**(i=0;i<TAB\_Max;i++) // **#define** TAB\_Max 180

{

Theta = i \* Step\_Angle; // Step\_Angle=1.74532925E-02

Sin\_tab[i] =sin(Theta); // 生成正弦表数组，利用了库里现成的SIN函数

Cos\_tab[i] =cos(Theta); // 生成余弦表数组

}

}

为啥我的数组里面只有180个元素呢？一：我的精度要求不高，我是一度一个步进角的，如果你要更高的精度，你可以选择容量更大的数组。二：我不想因为庞大的浮点数组占据太大的存储空间，所以我利用了奇函数，偶函数的原理

**再次**，就进入正题写查表程序，就我们初中数学所知，正弦函数是奇函数，余弦函数是偶函数，所以我只用了360角度中的一半。

下面为正弦查表程序：

/\*Refer to the truth that DSP use some SinCos operation will use six or more command

\* periods , So I look up the pre\_generate SINCOS\_TAB to speed up the Execution speed\*/

**float** SinF(**float** single)

{

**int** i=(**int**)(single\*180/pi);

**float** value=0;

**if**(i<0)

value = -Sin\_tab[i];

**else**

value = Sin\_tab[i];

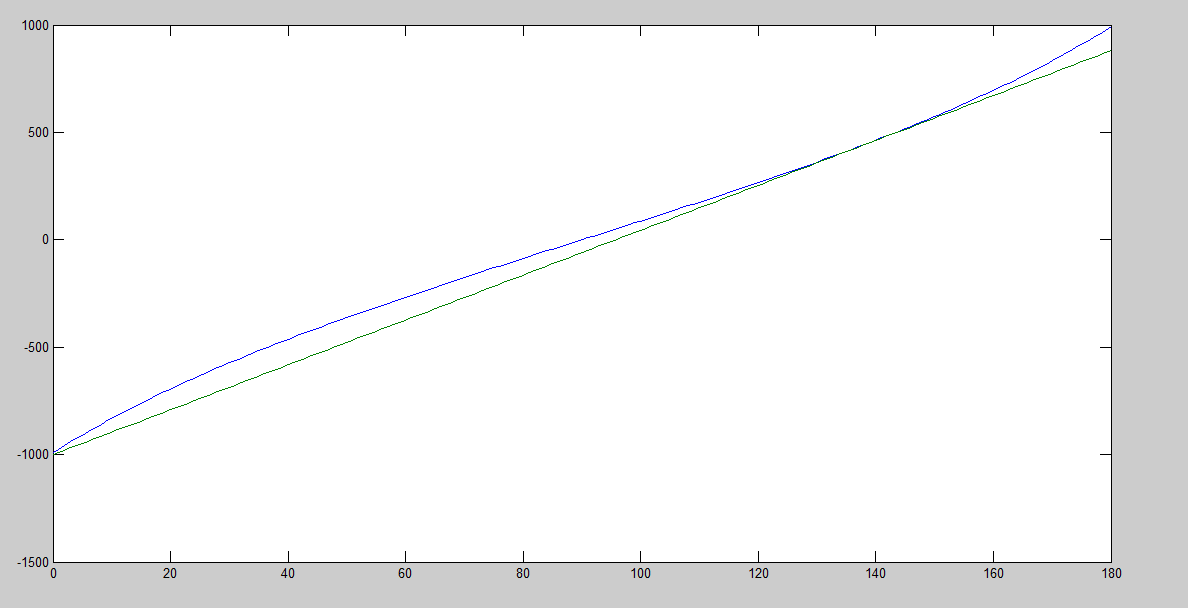
**return** value;

}

看起来应该很简单吧，余弦，正切，余切的编写和上面基本相似。

下面来探讨一下**反正弦，反正切**函数的编写

我在网上查了一下，运用函数库的反正弦，正切指令计算一个至少需要400以上各时钟周期，网上也有很多方法，看了有些能看懂，但大多看了头疼。自己呢，还是从正切函数的曲线出手，利用单调递增写了一个查表函数，已在MATLAB上实验过。



一次函数线性逼近正弦函数

这是我算法在MATLAB上面实验的程序：

function single = tanflook( a )

%UNTITLED Summary of this function goes here

% Detailed explanation goes here

global theta;

%global single;

global Actan\_tab;

global x;

global res;

theta=-90:90;

theta=theta\*3.141596/360;

Actan\_tab=1000\*tan(theta);

pi

z=input('the Z scale is :');

y=input('the Y scale is :');

a=(z+y>=0);

a

b=(z-y<=0);

b

if((a==b))

single = z/y;

else

single = y/z;

end

tab =fix(single\*99-8)+90;

tab

comp=fix(single\*1000);

for i=tab:181

if(comp>=Actan\_tab(i))&&(comp<Actan\_tab(i+1))

break;

% else

% i=i+1;

end

end

i

c = (i>90);

c

single= (i-90)/360\*pi;

x=2\*b+a;

% x=2;

switch x

case 0

single=-pi/2-single;%break; %case 0:single=single;break;

case 1

;%break;

case 2

single=pi+single-2\*pi\*c;%break;

case 3

single=pi/2-single;%break;

end

x

disp('the single value is ');

disp(180\*single/3.141596);

End

讲讲原理吧，感觉还是有点东西，还是从那个一次函数线性逼近正弦函数的地方说起，为什么要这样了，因为我们从图上可见发觉，每一个正弦值其实都在某段线性函数包围之间，我们已知了正切值，利用正切函数的线性递增原理，可以大致推断出他所处的大概位置，然后再查建立好的表，这里就不要再用处理器去生成了，查表时应尽量避免浮点数运算来节约时间，我们就可以精确的算法所对应的角度。试验了一下，最差的情况下差不多要消耗40个时钟周期。

关于上面几个CASE的推导，可以仿造SVPWM里面推导CASE的方法，感兴趣的可以推导一下，还是蛮有意思的，千万不要糊涂。

东西感觉不是很多，主要自己会的比较少，希望对大家有帮助，如果那个地方讲的不对，还请大神能及时指出，与我进行交流，我很喜欢和别人探讨