ELECTRONOTES APPLICATION NOTE NO. 372
1016 Hanshaw Rd
Ithaca, NY 14850 January 2008

INTERPOLATION ERROR IN SINEWAVE TABLES

INTRODUCTION:

The basic notion of interpolation is fundamental to many things we do with digital signal processing. Indeed it is the mechanism (in either continuous form, or as a combination of discrete and continuous forms) of reconstruction from samples. It also appears in rate changing and smoothing procedures, and so on. Many times in the past we have introduced interpolation by reminding students that an example of interpolation is the linear interpolation they used when they needed values for trig functions (or other functions) that were not those actually found in their printed tables. That is - read between the lines. Of course, in recent years, with calculators and computers being nearly universal, no one really needs or uses math table.

Still the interpolation process can be studies from the assumption that one has tabulated values and that values in between are needed. In our example here, we assume that we have a table of values of a sine function for each degree from 0 to 90. [Of course, this "quadrant" is all we need for sines and cosines through the use of symmetries.] Two simple methods suggest themselves when we need values that are not exactly integer values of degrees: we could just use the closest value, or we could use linear interpolation.

{In fact, many other methods are possible. Higher order polynomials could be used (the straight line being a first-order polynomial). Another popular method is to start with integer angle just below the desired angle, and then employ the trig identity (also seen below) for the sine of the sum of two angles, the second angle being the additional fraction of a degree. In fact, the small angle approximation to a sine and cosine [sin(x) ~ x and cos(x)~1] meant that you probably did not need a table of sine and cosine values for the range 0 to 1 degree. And so on.}

INTERPOLATION ERROR:

Here our interest is not the method, but rather the computation of the error. Intuitively, we might expect the error to get smaller as the angle approaches 90 degrees. This is because the difference between values of the sine is greatest at 0 degrees (where the slope is maximum) and least near 90 degrees where the slope is leveling off. This is exactly what we do see when we use rounding to the nearest integer (See Fig. 1a through Fig. 1c). Further, the error is sawtooth-like locally as we expect.

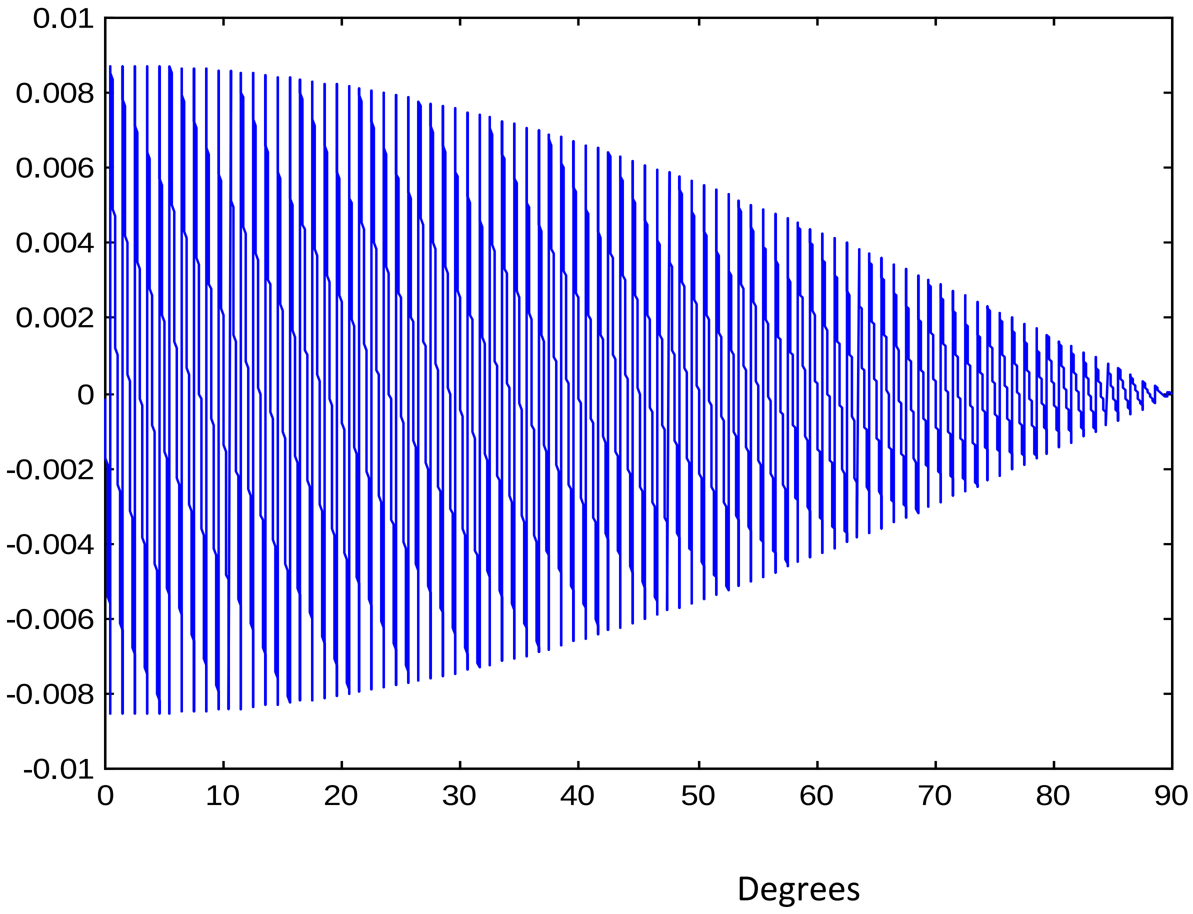

Fig. 1a Error with Rounding (see detail for 10 to 15 degree range shown in Fig. 1b below). The error is largest near zero.

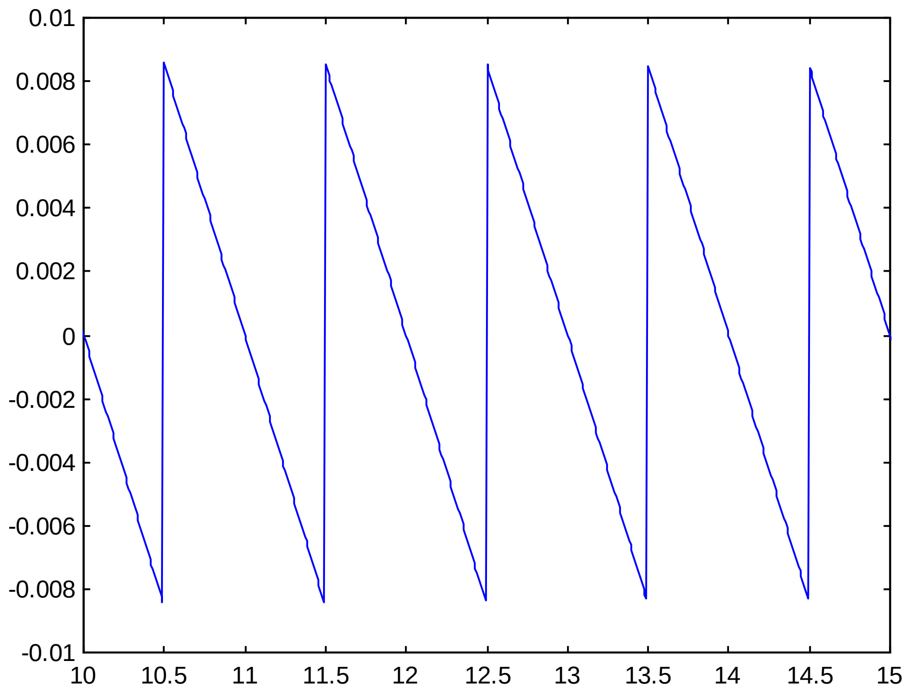

Fig. 1b Error with Rounding (Detail 10 to 15 degrees). Detail of Fig. 1a above. Error is linear about integer degrees.

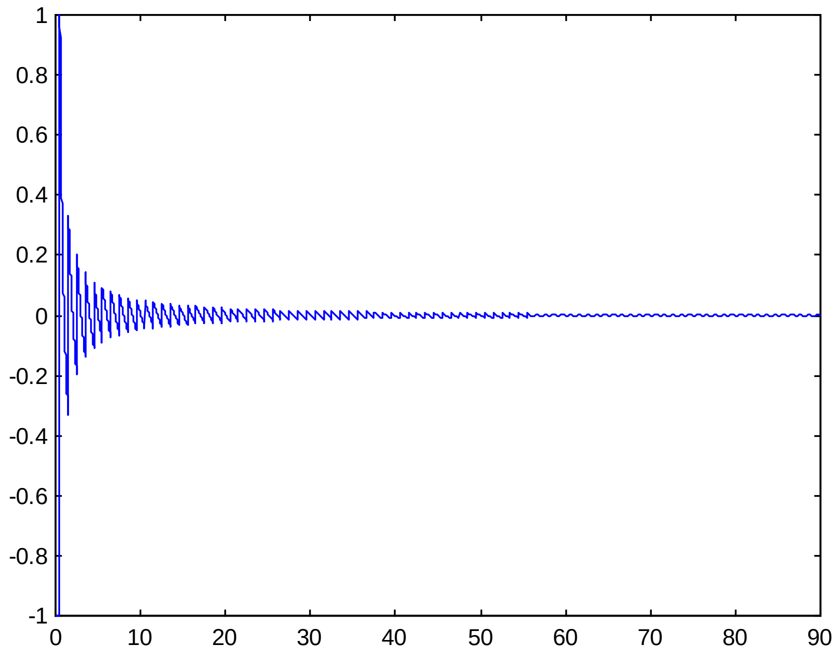

Fig 1c Relative Error (see programs) with Rounding

The case of linear interpolation (Fig. 2a through Fig. 2c) the error is not totally what we might expect, however. The error is always negative as we expect: the straight line is under the curve (see Fig. 3). Further, the error is lobe-like, maximizing near the center of the interval, as we expect. Also, the error is far smaller that we had with rounding, as we expect.   But - it gets larger as we approach 90 degrees.

What is evidently going on is that there is a "contest" here. While the difference between sine values is greatest around 0 degrees, the slope there is very much like the straight line of the linear interpolator. Around 90 degrees, the difference between sine values is much smaller, but the curvature away from the straight line segment is greater.   Who wins?   Evidently the curvature wins - on evidence of Fig. 2a.

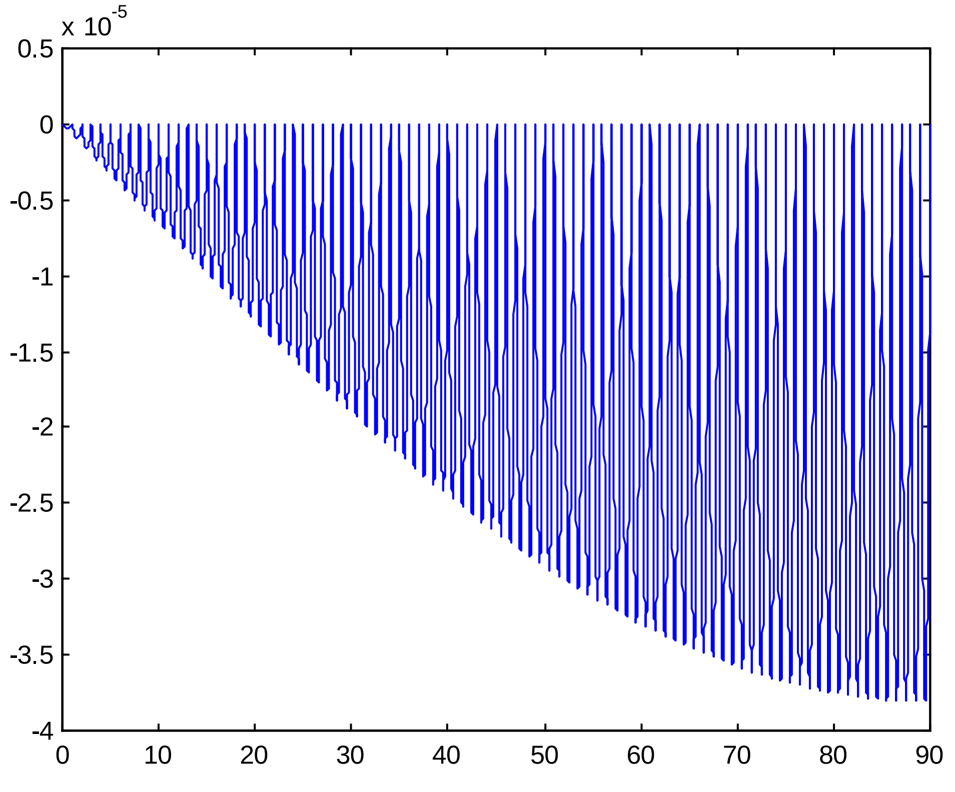

Fig. 2a   Error with Linear Interpolation
(see detail in Fig. 2b)

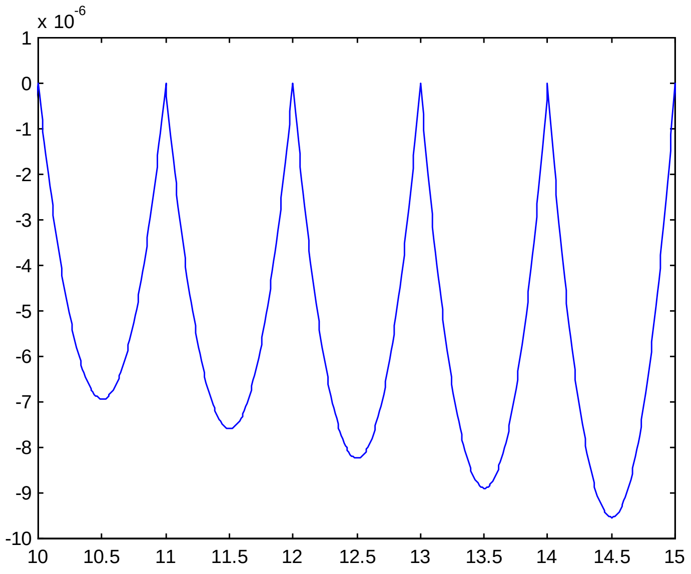

Fig. 2b Detail of Fig 2a, 10-15 degrees

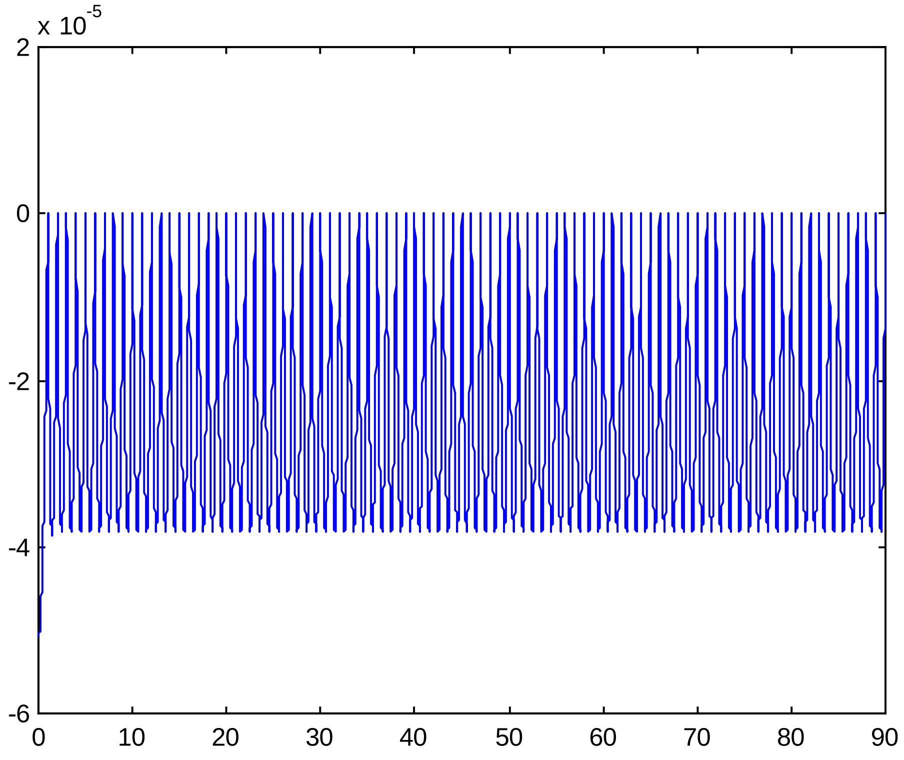

Fig. 2c Relative Error with Linear Interpolation

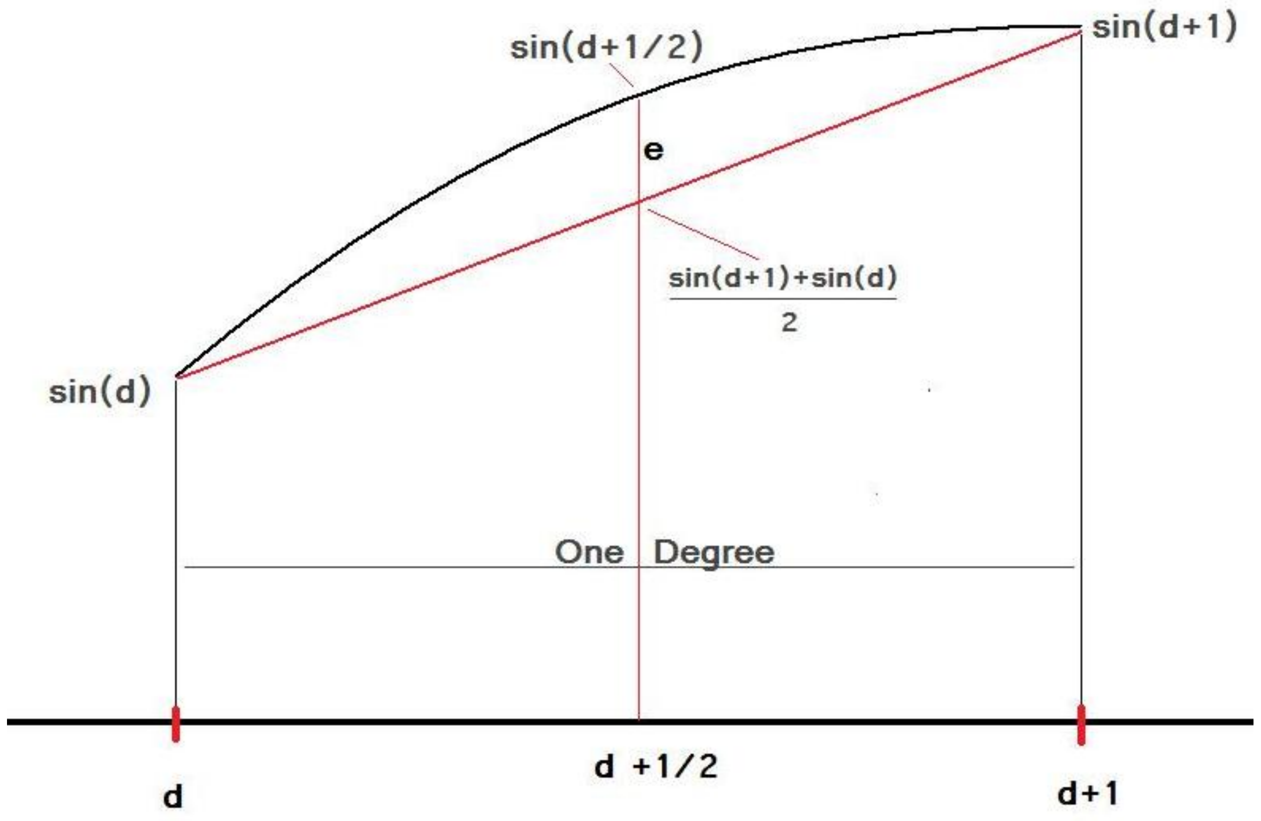

Fig. 3 Error with linear interpretation

We can demonstrate this finding theoretically by a simple procedure. Fig 3 shows an (exaggerated) segment from d degrees to d+1 degrees. We will find it convincing to just compute the error e at d+1/2 degrees, essentially looking for the "envelope" of the error in Fig. 2a. The value of the linear interpolation at d+1/2 is of course:

$$[\sin(d+1)+ \sin(d)]/2$$

where the actual sine is sin(d+1/2), so the error is:

$$e= [\sin(d+1)+ \sin(d)] /2 - \sin(d+1/2) \qquad (1)$$

We can apply the trig identity for the sine of a sum (used twice here):

$$\sin(A+B) = \sin(A)\cos(B) + \cos(A)\sin(B) \qquad (2)$$

and arrive at:

$$e = \sin(d)\ [\ \cos(1)/2 + 1/2 - \cos(1/2)\ ] + \cos(d)\ [\ \sin(1)/2 - \sin(1/2)] \qquad (3)$$

Computing the terms in the brackets we find the constant multiplier of sin(d) to be:

$$C_s = -3.8075 \times 10^{-5} \qquad (4a)$$

while the constant multiplier of cos(d) is:

$$C_c = -3.3228 \times 10^{-7} \qquad (4b)$$

So the error is clearly dominated by the term increasing as sin(d). Note that while we have chosen an interval of one degree between "tabulated" values, we could have chosen other values, with the same general results.

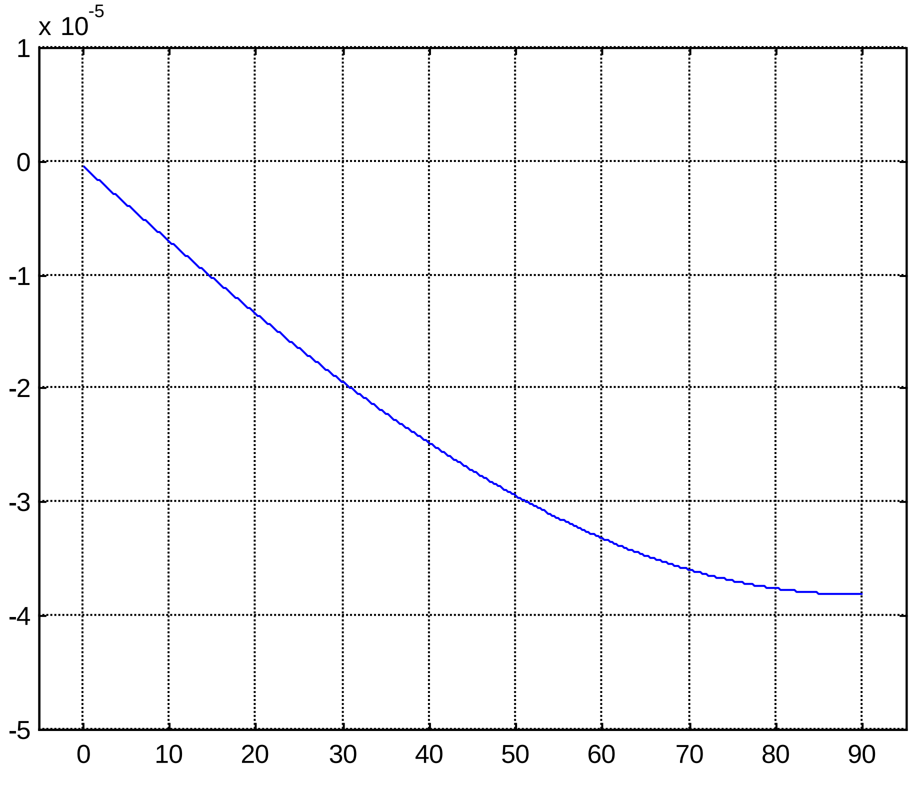

Fig. 4 Here we have plotted the theoretical "envelope of the error from Fig. 2a.

Fig. 2c is a bit of a curiosity. We see that the relative error with linear interpolation is essentially just enveloped by a constant slightly less than 4×10^{-5}. This we identify with the C_s constant above, and note that since for equation (3) we normalized e by dividing by sin(d), we understand that C_s dominates over C_c . The anomlyous exception near d=0 is understood as the C_c term being divided by a very small value of sin(d).

<u>SUMMARY:</u>

While it is absolutely no surprise that linear interpolation is a much better approximation than rounding (by as much as a factor of 200), and that the error with rounding is greatest about zero degrees, it was likely not obvious that the error with linear interpolation was largest not about zero, but about 90 degrees. Likely it is not obvious, either, that the relative error with linear interpolation has no trend (Fig. 2c).

PROGRAMS:

The figures for this note were created using Matlab. Nothing beats actual computer code (even if not in your preferred language) for showing exactly what was done.

For Fig. 1, Two Programs

```
function [sa,err,rerr]=p14core(x)
%  This function rounds to the nearest degree
%   and computes the errors
%     and does not prints them on the screen
format long
sa=sin(2*pi*(floor(x+1/2)/360)); % sine of rounded
smat=sin(2*pi*x/360);   % correct sine
err=sa-smat;   % error
rerr=err/smat;                     % relative error
```

```
% p14global.m
%  error on 0 to 90 degrees with rounding
k=1;
e=zeros(1,9000);
er=zeros(1,9000);
for x=0.01:.01:90
   [sa,err,rerr]=p14core(x);
   e(k)=err;
   er(k)=rerr;
k=k+1; end
figure(1)
plot([0.01:.01:90],e)
figure (2)
plot([0.01:.01:90],er)
figure (1)
```

For Fig.2, two programs

```
function [sa,err,rerr]=p15core(x)
%  This function does linear interpolation
%   and computes the errors
%     and does not print them on the screen
format long
xlow=floor(x) ;
xhigh=ceil(x);
offset=x-xlow;
change=sin(2*pi*xhigh/360)-sin(2*pi*xlow/360);
sa=sin(2*pi*xlow/360)+change*offset;
smat=sin(2*pi*x/360);
err=sa-smat;
rerr=err/smat;
```

```
% pl5global.m
%  error on 0 to 90 degrees with rounding
k=1;
e=zeros(1,9000);
er=zeros(1,9000);
for x=0.01:.01:90
   [sa, err, rerr]=p15core(x);
   e(k)=err;
   er(k)=rerr;
   k=k+1;
end
figure(1)
plot ( [0.01:.01:90],e)
figure(2)
plot ( [0.01:.01:90],er)
figure(1)
```

Note: The “c” parts of Fig. 1 and Fig. 2 were obtained by changing the axis

For Fig. 4

```
%  an372fig4.m

d=0:.01:90;
e=sin(d*2*pi/360)*(-3.8075E-5)+cos(d*2*pi/360)*(-3.3228E-7);
figure(1)
plot(d,e)
axis([-5 95 -5E-5 1E-5])
grid
```