APPROXIMATING SMOOTH STEP FUNCTIONS USING PARTIAL FOURIER SERIES SUMS

Disclaimer: Reference herein to any specific commercial company, product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the Department of the Army (DoA). The opinions of the authors expressed herein do not necessarily state or reflect those of the United States Government or the DoA, and shall not be used for advertising or product endorsement purposes.

UNCLASSIFIED: Distribution Statement A.
Approved for public release; distribution is unlimited.
APPROXIMATING SMOOTH STEP FUNCTIONS USING PARTIAL FOURIER SERIES SUMS

W. Bylsma
Dynamics and Structures
U.S. Army Research, Development and Engineering Command (RDECOM)
U.S. Army Tank-automotive and Armaments Research, Development and Engineering Center (TARDEC)
Detroit Arsenal
ATTN: RDTA-RS/MS157
6501 East 11 Mile Road
Warren, Michigan 48397-5000

*****

Disclaimer: Reference herein to any specific commercial company, product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the Department of the Army (DoA). The opinions of the authors expressed herein do not necessarily state or reflect those of the United States Government or the DoA, and shall not be used for advertising or product endorsement purposes.

*****

UNCLASSIFIED: Distribution Statement A. Approved for public release; distribution is unlimited.
This report documents several continuous functions that can approximate a discrete step function. The functions considered include the cosine, cubic polynomial, bezier polynomial, and hyperbolic tangent. Based on the error condition used, the Hyperbolic Tangent (Tanh) and Cosine functions provide the “best” approximations depending on the degree of smoothness desired.
1.0 INTRODUCTION

This report documents several continuous functions that can approximate the transition between different constant levels of a signal or step function. The desire to avoid abrupt changes in signal levels will be demonstrated. Figure 1 illustrates the problem where a continuous transition is sought from level $L_0$ to $L_1$ between $-\varepsilon$ and $+\varepsilon$. While this has been done in [1] the affects on the signal have never been clearly demonstrated.

![Figure 1 – Signal Levels and Transition Space](image)

The continuous functions that will be considered are the cosine, cubic polynomial, Bezier polynomial, and hyperbolic tangent.

2.0 FOURIER REPRESENTATION OF SIGNALS

Before analyzing the individual functions mentioned in Section 1.0, a brief review of Fourier theory is appropriate. For periodic signals the Fourier representation is

$$x(t) = \sum_{n=0}^{\infty} (a_n \cos n \omega_0 t + b_n \sin n \omega_0 t)$$

where ($T$ is the length of one period)
and noting that the summation (to infinity) will be relaxed to a partial sum (a large number of terms) that closely approximates the original signal. As outlined in [2], with the identities

$$\sin \theta = \frac{-j}{2}(e^{j\theta} - e^{-j\theta})$$

$$\cos \theta = \frac{1}{2}(e^{j\theta} + e^{-j\theta})$$

a complex number representation of the Fourier sum is defined as

$$x(t) = \sum_{n=-\infty}^{\infty} c_n e^{jnw_0t}$$

with

$$c_n = \frac{a_n - jb_n}{2}$$

$$c_n^* = c_{-n} = \frac{a_n + jb_n}{2}$$

$$c_0 = a_0.$$  

For even functions

$$a_n = 2c_n, b_n = 0.$$  

From (5) ([2], pg. 21) the Fourier coefficients can be obtained from the original signal as

$$c_m = \frac{1}{T} \int_{-T/2}^{T/2} x(t)e^{-jmw_0t} dt$$

For the periodic square pulse shown in Figure 2 the Fourier coefficients are

$$c_n = \frac{1}{T} \int_{-\pi/2}^{\pi/2} e^{-j2\pi t} dt = \frac{1}{-j2\pi n} (e^{-j2\pi nt})^{T/2} = \frac{\tau}{T} \text{sinc} \frac{\tau}{T} = d \frac{\text{sinc} \, nd}$$

where \(d = \tau/T\). This will be used as the test function to determine the affects the continuous step functions have on the Fourier representation of the signal and its approximation in the interval of interest.
3.0 DIRECT VERSUS NUMERICAL INTEGRATION

The four functions mentioned in Section 1.0 increase the complexity of the integration in (10) needed to obtain a closed form solution for the Fourier coefficients. Here we pause to investigate the feasibility of obtaining the coefficients by numerical integration. Using MATLAB (and its notation, see [3]) to define the resulting Fourier coefficients in (10) as

\[\text{Closed Form} \quad c_n = (2) d \cdot \text{sinc}(n d)\]

\[\text{Direct} \quad d_i = (2) \cdot \text{real} \left( \frac{\exp(-j\cdot2\pi\cdot n \cdot (\tau/2)/T)}{(-j\cdot2\pi\cdot n)} - \frac{\exp(-j\cdot2\pi\cdot n \cdot (-\tau/2)/T)}{(-j\cdot2\pi\cdot n)} \right)\]

\[\text{Numerical} \quad n_i = (2) \cdot \frac{1}{T} \cdot \text{real} \left( \text{trapz} \left( \exp(-j\cdot2\pi\cdot n \cdot t/T) \right) \right) \cdot dt\]

the maximum error for direct and numerical integration (using trapezoidal integration (\text{trapz})) is shown in Figure 3. The maximum error is defined as the maximum difference between the closed form solution for a 50 term partial Fourier series sum at each time step size. The initial maximum error was set to the machine precision epsilon (\(\text{eps} = 2.2204e-16\)). The results indicate that with small time steps, less than 0.001 (1e-3), numerical integration is an acceptable method to obtain the Fourier coefficients. Time steps less than 1-e6 increase the computational time and memory requirements significantly. For details of the computation see Appendix A.1 – Integration Differences.
In considering the “best” continuous function approximation to the discrete step, two points need to be considered: 1) Smoothness and 2) Accuracy. The trade-off between the two is reflected in a fewer number of Fourier series terms needed to reach a “good” approximation for smooth signals versus the reduced accuracy the smooth signal represents the non-smooth signal (a discrete step in this case) that is being approximated. In order to capture this three error terms are defined

\[ e_1 = \text{discrete step partial fourier series sum (N)} - \text{discrete step} \]

\[ e_2 = \text{smooth step partial fourier series sum (N)} - \text{smooth step (epsilon)} \]

\[ e_3 = \text{smooth step (epsilon)} - \text{discrete step} \]

(11)

Note the dependencies in the error on---N, the number of Fourier series terms, and epsilon, the distance allowed for smoothing the discrete transition. In short, e1 and e2 represent the error between what is trying to be approximated with a partial Fourier series sum and e3 represents the difference between the discrete and smoothed representations before each is approximated with a partial Fourier series sum.

4.0 ERROR CRITERIA

Figure 3 – Fourier Coefficient Integration Error Comparison - Numerical and Direct
With these definitions, the criteria for selecting the “best” smooth approximation is defined when

\[ e^2(N, \varepsilon) + e^3(\varepsilon) < e^1(N) \]

expressed as a percentage of the pulse area. This criteria gives an upper bound, dependent on N and epsilon.

Figures 4 and 5 show examples of the smooth step functions for epsilon at 10% and 50% of tau respectively. For the examples in this report the parameters (period length, pulse width, number of terms) for Figure 2 are

\[ T = 25, \tau = 12.5, N = 150. \]

Figure 4 – Smooth Step Functions with epsilon equal to 10% of the pulse width (tau).

Figure 6 shows a zoomed in right side of Figure 5.
Figure 5 – Smooth Step Functions with epsilon equal to 50% of the pulse width (tau).
Table 1 shows the relative difference between the smooth step functions for the two cases of epsilon.

<table>
<thead>
<tr>
<th>Epsilon, % of tau</th>
<th>Cosine</th>
<th>Cubic</th>
<th>Bezier</th>
<th>Tanh</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>7.2676</td>
<td>7.5000</td>
<td>4.0000</td>
<td>2.5205</td>
</tr>
<tr>
<td>50</td>
<td>36.3380</td>
<td>37.5000</td>
<td>20.000</td>
<td>12.6024</td>
</tr>
</tbody>
</table>

The values in Table 1 correlate with Figure 6 and suggest the “best” smooth step approximation rank order to be tanh, Bezier, cosine, and cubic in terms of e3 error.

5.0 COSINE

The cosine function is plotted in Figure 7 for a domain of 0 to $2\pi$ (1 period). Looking at the first half of the interval there is a smooth transition between 1 and -1. With translation and scaling the function can be modified to make a smooth step between defined levels as in Figure 1.
Each side of the center pulse in Figure 2 will use the first half (right side) or second half (left side) of the cosine period. Scaling (or normalizing) the amplitude to obtain an output value between zero and one is accomplished by adding 1 and multiplying by 0.5 as

$$\frac{1}{2}(1 + \cos \omega_0 t)$$

(13)

Translation is used to transform the time scale for the square pulse into the desired half of the cosine function time scale. With $t_c$ being the cosine function time scale, for the right side of the center square pulse the desired shift is

$$0 < t_c < T/2$$
$$0 < t_c < 2\varepsilon$$
$$0 < t - \left(\frac{T}{2} - \varepsilon\right) < 2\varepsilon$$

(14)

while for the left side it is

$$T/2 < t_c < T$$
$$2\varepsilon < t_c < 4\varepsilon$$
$$2\varepsilon < t - \left(-\frac{T}{2} - 3\varepsilon\right) < 4\varepsilon$$

(15)

A more straightforward approach is to use the shift in (14) for both sides but change the sign after the one in (13).

Implementation specifics are denoted using MATLAB notation. From (14)

$$\frac{T}{2} = 2\varepsilon \rightarrow T = 4\varepsilon$$

(16)

the cosine period is
% set period
   tc = 4*epi;

For the specific example in this report with the following parameters

%%% parameters
   T = 25;                         % period length
   tau = 12.5;                     % square pulse width
   d = tau/T;                      % duty ratio
   dt = 0.001;                     % time step
   t = [-50:dt:50];                % time vector
   ierr = find( abs(t) < T/2 );

%%% loop for each epsilon
   for epi = [ 0.01:.01:.1,.5]*tau, % percentage of tau

the cosine step function is defined as

   ii = find{ abs(t + tau/2) <= epi };  
   stepm(ii) = 0.5 * (1 - cos( (2*pi/tc)*(t(ii) - (-tau/2-epi)) ) );
   ii = find{ abs(t - tau/2) <= epi };  
   stepm(ii) = 0.5 * (1 + cos( (2*pi/tc)*(t(ii) - (tau/2-epi)) ) );

Calculation of the Fourier coefficients is based on (9) and (10)

   trange = find ( (t >= -tau/2-epi) & (t <= tau/2+epi) );
   coef = trapz( stepm(trange).*exp(-j*2*pi*n.*t(trange)/T) );
   coef=1/T*real(coef)*dt;
   if (n == 0),
      y(cnt,:) = coef + 0*t;
   else
      y(cnt,:) = y(cnt-1,:) + 2 * coef*cos(n*2*pi/T.*t);
   end
   cnt = cnt + 1;
   % compute discrete step coefficients directly
   if ( n == 0 ),
      yo(cnto,:) = d * sinc(n*d) * cos(n*2*pi/T.*t);
   else
      yo(cnto,:) = yo(cnto-1,:) + 2 * d*sinc(n*d)*cos(n*2*pi/T.*t);
   end
   cnto = cnto + 1;

The error computation follows from (11)

   % continuous step
   % cosine
   err2 = trapz( abs( y(nnum+1,ierr) - stepm(ierr) ) ) * dt;
   % difference
   % cosine
   err3 = trapz( abs( stepm(ierr) - step(ierr) ) ) * dt;

Figure 8 shows the cosine function partial Fourier series sum error results for e1 (red, center), e2 (green, lower left), and e2+e3 (blue, upper right) for epsilon as a percentage of tau. Note that e1 is a function of N only, and e3 is a function of epsilon only. As epsilon increases, the e2 error is reduced quickly for a small number of terms. For e2+e3, as epsilon increases, e3 begins to dominate and reduces the number of terms at which the condition in (12) is satisfied. The "squiggles" in the curves are attributed to the odd coefficients being zero which results in a constant sum for two consecutive terms. This principle is described in [4] for the normalized sinc function.
6.0 CUBIC POLYNOMIAL

The cubic polynomial function

\[ y = ax^3 + bx^2 + cx + d \]  

(17)

is plotted in Figure 9 for a domain of 0 to 1 that provides a smooth transition between 0 and 1 assuming the derivative at each end point is equal to zero. With these conditions

\[ y(0) = 0, y(1) = 1, y'(0) = 0, y'(1) = 0. \]

we arrive at

\[ y = -2x^3 + 3x^2 = xx(3 - 2x). \]  

(18)

As described in [5] and [6], the domain is normalized using a “clamping” function defined below

```matlab
% clamp
function y = clamp(x,min,max)
    y = x;
    % limit x values to min/max
    imax = find(y > max);
    y(imax) = max;
    imin = find(y < min);
    y(imin) = min;
```

Figure 8 – Cosine Function Partial Fourier Series Sum Error
which is used in defining the cubic polynomial or “smooth step” as

```matlab
function y = smoothstep(xmin,xmax)
% normalize to 0 to 1 using xmin/xmax in domain of x
xx = (x-xmin)/(xmax-xmin);
s_min = 0.0;
s_max = 1.0;
% set domain limits
y = clamp(xx, s_min, s_max);
% return value
y = y .* y .* (3 - 2*y);
```

The left side of the center pulse in Figure 2 will use the normalized domain between +/- epsilon and the right side will use a reversed normalized domain (or subtract the left side result from one) with appropriate amplitude scaling (in this case just one).

![Cubic Polynomial Function](image)

**Figure 9 – Cubic Polynomial Function**

With the same parameters and definitions as outlined in Section 5.0, with exceptions called out, the cubic polynomial step function is defined as

```matlab
% cubic
ii = find( abs(t+tau/2) <= eps );
stepm(ii) = smoothstep( t(ii), min(t(ii)), max(t(ii)) );
ii = find( abs(t-tau/2) <= eps ) ;
iii = t(ii);
rt = iii(end:-1:1);
stepm(ii) = smoothstep( rt, min(rt), max(rt) );
% stepm(ii) = 1 - smoothstep( t(ii), min(t(ii)), max(t(ii)) );
```

Figure 10 shows the cubic polynomial partial Fourier series sum error results. Similar results are obtained as in Section 5.0.
7.0 BEZIER POLYNOMIAL

The Bezier polynomial function is similar to the cubic polynomial in Section 6.0 except that both x and y values are defined as a function of parameter t with a domain of 0 to 1

\[ x = a_xt^3 + b_xt^2 + c_xt + d_x \]
\[ y = a_yt^3 + b_yt^2 + c_yt + d_y \]

Four points, P0, P1, P2, and P3 of (x,y), are needed to solve for each variable as discussed in [7]. The procedure is the same as for the cubic polynomial in Section 6.0 except that the slopes at the end points are defined in terms of the parameter t as

\[ x(0) = d_x = P0x \]
\[ x(1) = a_x + b_x + c_x + d_x = P3x \]
\[ x'(0) = c_x = \text{slope} = \frac{\text{rise}}{\text{run}} = \frac{P1x - P0x}{\frac{1}{3} - 0} = 3(P1x - P0x) \]
\[ x'(1) = 3a_x + 2b_x + c_x = \frac{P3x - P2x}{1 - \frac{2}{3}} = 3(P3x - P2x) \]

(20).
The same steps are followed to determine $y$. In matrix form, the coefficients defined in terms of the points are

\[
\begin{bmatrix}
  a_x \\
  b_x \\
  c_x \\
  d_x
\end{bmatrix} = \begin{bmatrix}
  -1 & 3 & -3 & 1 \\
  3 & -6 & 3 & 0 \\
  -3 & 3 & 0 & 0 \\
  1 & 0 & 0 & 0
\end{bmatrix}\begin{bmatrix}
  P0x \\
  P1x \\
  P2x \\
  P3x
\end{bmatrix}
\]

(21).

With the points

\[
P0 = (0,0), P1 = (0.8,0), P2 = (0.2,1.0), P3 = (1.0,1.0)
\]

(22)

the coefficients for $x$ and $y$ are (2.8, -4.2, 2.4,0) and (-2, 3, 0, 0), respectively. Figure 11 shows the difference between the Bezier and cubic polynomials. Adjustments to the point definitions allow for manipulation of the curve shape. For example, redefining the points in (22) to

\[
P1 = (0.33,0), P2 = (0.66,1.0)
\]

the Bezier becomes the cubic polynomial with $x$ coefficients of (0.01, 0, 0.99, 0). Changes to the "smooth step" function in Section 6.0, to allow for more flexibility with the Bezier polynomial, are included below.

```matlab
% smoothstepbez
function y = smoothstepbez(x,min,max,ind)
    if (ind == 'x')
        p=[0 0.8 0.2 1]';
    elseif (ind == 'y')
        p=[0 0 1 1]';
    else
        p=[0 .33 .66 1]'; % cubic
    end
```

Figure 11 – Bezier Polynomial Function

![Bezier Polynomial Function](image)
\[ T = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}; \]
\[ c = T*p; \]

% normalize (0 to 1) using min/max in domain of x
\[ xx = (x-min)/(max-min); \]
\[ s_min = 0.0; \]
\[ s_max = 1.0; \]

% set domain limits
\[ y = \text{clamp}(xx, s_min, s_max); \]

% return value
\[ y = c(1)*y^3 + c(2)*y^2 + c(3)*y + c(4); \]

As for the cubic polynomial in Section 6.0, the left side of the center pulse in Figure 2 will use the normalized domain between +/- epsilon and the right side will use a reversed normalized domain (or subtract the left side result from one) with appropriate amplitude scaling.

With the same parameters and definitions as outlined in Section 5.0, with exceptions called out, the Bezier polynomial step function is defined as

\% Bezier
\[ xt = t; \]
\[ ii = \text{find}(\text{abs}(t + tau/2) <= epi); \]
\[ xt_start = t(ii(1)); \]
\[ xt_len = t(ii(end)) - t(ii(1)); \]

% define x domain
\[ xt_i = xt_start + xt_len * \text{smoothstepbez}(t(ii), \text{min}(t(ii)), \text{max}(t(ii)), 'x'); \]
\[ xt = [xt(1:ii(1)-1), xt_i, xt(ii(end)+1:end)]; \]

% interpolate t, using x, to get y
\[ \text{stepm}(ii) = \text{interp1}(xt(ii), \text{smoothstepbez}(t(ii), \text{min}(t(ii)), \text{max}(t(ii)), 'y'), t(ii), 'linear', 'extrap'); \]

\% define reverse x domain
\[ xrt = xrt_start + xrt_len * \text{smoothstepbez}(rt, \text{min}(rt), \text{max}(rt), 'x'); \]
\[ xrt = [xt(1:ii(1)-1), xrt, xt(ii(end)+1:end)]; \]

% interpolate t, using reverse x, to get y
\[ \text{stepm}(ii) = 1 - \text{interp1}(xt(ii), \text{smoothstepbez}(t(ii), \text{min}(t(ii)), \text{max}(t(ii)), 'y'), t(ii), 'linear', 'extrap'); \]

In this case, because x is also defined as a function of the independent parameter t, the interpolation function is used to obtain the associated x value for the time grid which is used as the independent variable for the other step functions.

Figure 12 shows the Bezier polynomial partial Fourier series sum error results. Differences are seen between the cosine and cubic functions in that while the e3 error is smaller, more terms are needed to reduce the e2 error for larger values of epsilon as a percentage of tau. However, all the e2 errors are less than e1.
8.0 HYPERBOLIC TANGENT

The hyperbolic tangent (tanh) function is plotted in Figure 13. Looking at the center there is a smooth transition between 1 and -1—although the function approaches 1 (-1) as the domain approaches infinity (-infinity). As with the cosine function in Section 5.0, with translation and scaling the tanh function can be modified to make a smooth step between defined levels as in Figure 1.
An output value between zero and one is accomplished by adding 1 and multiplying by 0.5 as

\[ \frac{1}{2} (1 + \tanh bx) \]  

(23)

The sign after the one in (23) is used to change the direction of the level transition.

Due to machine precision the tanh function will achieve the value of 1 in some interval, even though it approaches infinity in theory. Matching the transition space in Figure 1 to that of the tanh function requires a scaling factor, b, which can be found from the test case below

```matlab
% b.m
t = [-10:.1:10];                % time scale
y = find( abs( round( tanh(t)*10000 )/10000) >= 1.0 );
min( abs(t(y)) )
```

which gives 5.3. With the ranges

\[-\varepsilon < x < \varepsilon \]
\[-5.3 < bx < 5.3 \]

then

\[ b = \frac{5.5}{\varepsilon} \]

where the value of 5.3 was increased to provide some margin of safety.

With the same parameters and definitions as outlined in Section 5.0, with exceptions called out, the hyperbolic tangent step function with the scaling factor

```matlab
% tanh
b = 5.5/epi;
```

is defined as

```matlab
% tanh
ii = find( abs(t + tau/2) <= epi );
stepm(ii) = 0.5 * (1 + tanh( b*(t(ii) - (-tau/2)) ));
ii =find( abs(t - tau/2) <= epi ) ;
stepm(ii) = 0.5 * (1 - tanh( b*(t(ii) - (tau/2)) ));
```

Figure 14 shows the hyperbolic tangent partial Fourier series sum error results. The e3 error is smaller and the e2+e3 curves are converging to the e1 error as tanh more closely approximates the discrete step as indicated by Table 1 under the condition in (12). As compared to the cosine and cubic functions, more terms are required to reduce the e2 error.

As a side note, another use for the tanh function, which is not covered in detail here, is for blending functions to provide a smooth transition between them as described in [8]. With \( s(x) \) defined in (23) the result is

\[ h(x) = s(x)f_1(x) + (1 - s(x))f_2(x). \]
9.0 RESULTS AND SUMMARY

The partial Fourier series sums error results (Figures 8, 10, 12, and 14) for the cosine, cubic polynomial, Bezier polynomial, and hyperbolic tangent step functions, respectively, demonstrate that the smoother the function (in terms of e3, larger e3 indicates smoother) the more quickly the error is reduced for a smaller number of Fourier series terms. Computation of the coefficient terms is outlined in Appendix A.2. The error data was processed to determine the maximum number of terms at which condition (12) was met. This procedure is outlined in Appendix A.3. The summarized data is presented in Figure 15. Note that since the amplitude of the square pulse in Figure 2 is one, the percentage of tau is equivalent to the percentage of the pulse area. As epsilon increases the size of e3 does also and is independent of the number of terms N (constant as a function of N). This pushes the intersection point (between the e2+e3 and e1) to a lower (leftward) number of terms (N) as Figures 10 and 14 demonstrate. The continuous step functions that have the largest e3 error (see Table 1) are restricted to the lowest number of partial sum terms, although reducing the error more quickly.

Figures 16, 17, and 18 show the e1, e2, and e3 errors, respectively, at the error limit condition in Figure 15 (N is not shown on the plots). Details for each function are included in Table 2.

The e1 error, in Figure 16, is greater than 1% in all cases and follows the same order as in Figure 15 where for fewer number of terms the error is larger (epsilon increases => e3 increases => N decreases => e1 increases).

The e2 error, in Figure 17, is less than 1% for all epsilon except for the Bezier Polynomial (which is greater than 1% for epsilon greater than 8%), but still less than the associated e1 error. This indicates that at the error limit condition the continuous step functions have smaller approximation error than the discrete step partial Fourier series sums.
The order change in the e2 error of Figure 17 is explained in Figures 18 and 19 by the cross-over of the Bezier and tanh functions for some intermediate value of epsilon between 1% and 10%. At this value of epsilon, the e2 error decreases quicker than the Bezier function for larger values of N. This illustrates one advantage of the tanh function for an epsilon greater than 4%.

![Graph showing error condition limits for different functions]

Figure 15 –Partial Fourier Series Sums Error Results

The e3 error, in Figure 18, has constant slope consistent with Table 1 as it is independent of N.

The e2+e3 error in Figure 21 shows that the e3 error order of continuous step functions is preserved, except for small epsilon (see Figure 17).

From this analysis and the error criteria in (12), we can conclude that the overall “best” continuous step function approximation to a discrete step is the Hyperbolic Tangent (tanh) function for epsilon greater than approximately 1.5% and the cosine function for epsilon less than approximately 1.25%.

Future work in this area would be to analyze the errors for epsilon less than 1% and the transition area between epsilon of 1% to 1.5%.
Figure 16 – e1 Error (for N in Fig. 15)

Figure 17 – e2 Error (for N in Fig. 15)
Figure 18 – e3 Error (for N in Fig. 15)
<table>
<thead>
<tr>
<th>Function</th>
<th>epsilon</th>
<th>N</th>
<th>e1</th>
<th>e2</th>
<th>e3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cosine</td>
<td>1</td>
<td>150</td>
<td>1.209537</td>
<td>0.039066</td>
<td>0.726777</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>118</td>
<td>1.485054</td>
<td>0.029608</td>
<td>1.453529</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>72</td>
<td>2.256793</td>
<td>0.071739</td>
<td>2.180287</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>50</td>
<td>3.061496</td>
<td>0.132045</td>
<td>2.907046</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>38</td>
<td>3.841763</td>
<td>0.187851</td>
<td>3.633806</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>30</td>
<td>4.662628</td>
<td>0.243676</td>
<td>4.360566</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>24</td>
<td>5.587874</td>
<td>0.289478</td>
<td>5.087326</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>20</td>
<td>6.469539</td>
<td>0.312949</td>
<td>5.814086</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>18</td>
<td>7.036793</td>
<td>0.349583</td>
<td>6.540846</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>16</td>
<td>7.725602</td>
<td>0.377597</td>
<td>7.267606</td>
</tr>
<tr>
<td>Cubic</td>
<td>1</td>
<td>150</td>
<td>1.209537</td>
<td>0.059059</td>
<td>0.750016</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>112</td>
<td>1.552625</td>
<td>0.0463</td>
<td>1.500008</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>68</td>
<td>2.367846</td>
<td>0.107298</td>
<td>2.250005</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>46</td>
<td>3.280907</td>
<td>0.201364</td>
<td>3.000004</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>36</td>
<td>4.016389</td>
<td>0.251514</td>
<td>3.750003</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>28</td>
<td>4.931978</td>
<td>0.319425</td>
<td>4.500003</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>22</td>
<td>5.993539</td>
<td>0.353315</td>
<td>5.250002</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>20</td>
<td>6.469539</td>
<td>0.405416</td>
<td>6.000002</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>16</td>
<td>7.725602</td>
<td>0.396347</td>
<td>6.750002</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>14</td>
<td>8.581817</td>
<td>0.426217</td>
<td>7.500002</td>
</tr>
<tr>
<td>Bezier</td>
<td>1</td>
<td>150</td>
<td>1.209537</td>
<td>0.509525</td>
<td>0.400092</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>150</td>
<td>1.209537</td>
<td>0.255185</td>
<td>0.800046</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>124</td>
<td>1.423539</td>
<td>0.218305</td>
<td>1.200003</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>86</td>
<td>1.942723</td>
<td>0.330454</td>
<td>1.600023</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>64</td>
<td>2.491377</td>
<td>0.465372</td>
<td>2.000018</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>50</td>
<td>3.061496</td>
<td>0.617477</td>
<td>2.400015</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>42</td>
<td>3.537433</td>
<td>0.72593</td>
<td>2.800013</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>34</td>
<td>4.209219</td>
<td>0.938606</td>
<td>3.200012</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>30</td>
<td>4.662628</td>
<td>1.041265</td>
<td>3.60001</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>26</td>
<td>5.237663</td>
<td>1.209506</td>
<td>4.000009</td>
</tr>
<tr>
<td>Tanh</td>
<td>1</td>
<td>150</td>
<td>1.209537</td>
<td>0.777596</td>
<td>0.252106</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>150</td>
<td>1.209537</td>
<td>0.340412</td>
<td>0.504124</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>150</td>
<td>1.209537</td>
<td>0.122548</td>
<td>0.756162</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>150</td>
<td>1.209537</td>
<td>0.040091</td>
<td>1.008205</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>140</td>
<td>1.283229</td>
<td>0.019434</td>
<td>1.260249</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>112</td>
<td>1.552625</td>
<td>0.028767</td>
<td>1.512295</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>94</td>
<td>1.801826</td>
<td>0.036689</td>
<td>1.764341</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>80</td>
<td>2.065102</td>
<td>0.047711</td>
<td>2.016387</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>68</td>
<td>2.367846</td>
<td>0.066982</td>
<td>2.268434</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>60</td>
<td>2.629701</td>
<td>0.080618</td>
<td>2.520481</td>
</tr>
</tbody>
</table>
Figure 19 – $e^2$ Error (for $N$ in Fig. 15, $\epsilon = 1\%$)

Figure 20 – $e^2$ Error (for $N$ in Fig. 15, $\epsilon = 10\%$)
Figure 21 – $e_2+e_3$ Error (for N in Fig. 15)
REFERENCES


APPENDIX A.1 – INTEGRATION DIFFERENCES

%intdiff.m

clear all
close all

%%% parameters
T = 5;            % period length
tau = 1;          % square pulse width
d = tau/T;        % duty ratio
err = [];         % error results
nnuml = 50;       % number of terms

%%% loop for time step
for dt = [1e-1 1e-2 1e-3 1e-4 1e-5 1e-6 1e-7 1e-8 1e-9 1e-10],

    t = [-tau/2:dt:tau/2];
    med = eps;  % max error (direct)
    mei = eps;  % max error (numerical)

% loop coefficients
for n = 1:nnuml,

    % closed form
    cn = 2 * d*sinc(n*d);

    % direct
    di = 2 * (( exp(-j*2*pi*n*(tau/2)/T)/(-j*2*pi*n) ) -
               ( exp(-j*2*pi*n*(-tau/2)/T)/(-j*2*pi*n) ));

    % numerical
    ni = trapz( exp(-j*2*pi*n.*t/T) );
    ni = 2 * 1/T*real(ni)*dt;

    % maximum error
    med = max( abs(cn-di), med );
    mei = max( abs(cn-ni), mei );

% output 'term #, closed form, direct(di), err w/di, numerical(ni), error w/ni'
',dt,n,cn,di,cn-di,ni,cn-ni);
end

fprintf('max err:  %20.14f %20.14f
',-log10(med),-log10(mei));

% results
err = [err;[-log10(dt),-log10(med),-log10(mei)]];

end

% output results
fid = fopen('intout.txt','wt');
fprintf(fid,'%20.14f %20.14f %20.14f
',err');
fclose(fid)
%pfss.m (partial Fourier series sum)

clear all

close all

%=== parameters
T = 25;                         % period length
tau = 12.5;                     % square pulse width
d = tau/T;                      % duty ratio
dt = 0.001;                     % time step
t = (-50:dt:50);                % time vector

iierr = find( abs(t) < T/2 );
errout=[];                      % error results
plotf = 0;                      % 0=no plots, 1=include plots

%=== discrete step (square pulse)
step = 0*t;
ii = find( abs(t) < tau/2 );
step(ii) = 1;

%=== loop for each epsilon
for epi = [ 0.01:.01:.1,.5]*tau, % percentage of tau
    % cosine
    %
    tc = 4*epi;             % set period
    %
    % define continuous step
    clear ii;
    stepm = 0*t;
    ii = find( abs(t) < (tau/2-epi) );
    stepm(ii) = 1;

    % cosine
    % ii = find( abs(t+tau/2) <= epi );
    % stepm(ii) = 0.5 * ( 1 - cos( (2*pi/tc)*(t(ii) - (-tau/2-epi)) ) );
    % ii = find( abs(t-tau/2) <= epi );
    % stepm(ii) = 0.5 * ( 1 + cos( (2*pi/tc)*(t(ii) - (tau/2-epi)) ) );

    % cubic
    % ii = find( abs(t+tau/2) <= epi );
    % stepm(ii) = smoothstep( t(ii), min(t(ii)), max(t(ii)) );
    % ii = find( abs(t-tau/2) <= epi );
    % iii = t(ii);
    % rt = iii(end:-1:1);
    % stepm(ii) = smoothstep( rt, min(rt), max(rt) );

    % Bezier
    % xt = t;
    % ii = find( abs(t+tau/2) <= epi );
    % xt_start = t(ii(1));
    % xt_len = t(ii(end)) - t(ii(1));
    % xt_i = xt_start + xt_len * smoothstepbez( t(ii), min(t(ii)), max(t(ii)), 'x' );
    % xt = [xt[1:ii(1)-1], xt_i, xt[ii(end)+1:end]];
    % stepm(ii) = interp1(xt(ii), smoothstepbez( t(ii), min(t(ii)), max(t(ii)), 'y' ), t(ii), 'linear', 'extrap');

    % ii = find( abs(t+tau/2) <= epi );
    % iii = t(ii);
    % rt = iii(end:-1:1);
    % xrt_start = rt(end) - rt(1);
    % xrt_len = rt(end) - rt(1);
    % xrt = xrt_start + xrt_len * smoothstepbez( rt, min(rt), max(rt), 'x' );
    % xt = [xt[1:ii(1)-1], xrt, xt[ii(end)+1:end]];
    % stepm(ii) = interp1( xt(ii), smoothstepbez( rt, min(rt), max(rt), 'y' ), t(ii), 'linear', 'extrap');

end
\% \text{tanh}
\begin{align*}
  i_i &= \text{find}(\text{abs}(t+\tau/2) \leq \varepsilon) ; \\
  \text{stepm}(i_i) &= 0.5 \ast (1 + \tanh(b \ast (t(i_i) - (-\tau/2))) ) ; \\
  i_i &= \text{find}(\text{abs}(t-\tau/2) \leq \varepsilon) ; \\
  \text{stepm}(i_i) &= 0.5 \ast (1 - \tanh(b \ast (t(i_i) - (\tau/2))) ) ; \\
\end{align*}
\% 
\begin{verbatim}
clear y yo; 
cnt = 1; \quad \% \text{partial sum counter (continuous step)}
cnto = 1; \quad \% \text{partial sum counter (discrete step)}
nnuml = 150 \quad \% \text{number of terms}
trange = \text{find} \{ (t \geq -\tau/2-\varepsilon) \& (t \leq \tau/2+\varepsilon) \};
\% \text{compute partial sums}
\text{for } n=(0,1:nnuml),
\begin{align*}
  \% \text{compute continuous step coefficients} \\
  \% \text{cosine/cubic/Bezier/tanh} \\
  \text{coef} &= \text{trapz}(\text{stepm}(\text{trange}).\ast \text{exp}(-j\ast 2\pi \ast n \ast t(\text{trange})/T) ) ; \\
  \text{coef} &= 1/T \ast \text{real} (\text{coef}) \ast dt ; \\
\end{align*}
\% \text{handle constant term separate} \\
\text{if } ( n == 0 ),
\begin{align*}
  y(cnt,:) &= \text{coef} + 0 \ast t ; \\
  \text{else} \\
  y(cnt,:) &= y(cnt-1,:) + 2 \ast \text{coef} \ast \text{cos} (n \ast 2 \pi / T \ast t) ;
\end{align*}
\text{end}
\text{cnt} = \text{cnt} + 1 ; \\
\% \text{compute discrete step coefficients directly} \\
\text{if } ( n == 0 ),
\begin{align*}
  yo(cnto,:) &= d \ast \text{sinc} (n \ast d) \ast \text{cos} (n \ast 2 \pi / T \ast t) ; \\
  \text{else} \\
  yo(cnto,:) &= yo(cnto-1,:) + 2 \ast d \ast \text{sinc} (n \ast d) \ast \text{cos} (n \ast 2 \pi / T \ast t) ;
\end{align*}
\text{end}
\text{cnto} = \text{cnto} + 1 ;
\end{verbatim}
\% \text{compute error results for partial sums}
\text{for } nnum=10:1:nnuml,
\begin{verbatim}
\% \text{discrete step} \\
\text{err1} &= \text{trapz} ( \text{abs} ( yo(nnum+1,iierr) - \text{step}(iierr) ) ) \ast dt ; \\
\% \text{continuous step} \\
\% \text{cosine/cubic/Bezier/tanh} \\
\text{err2} &= \text{trapz} ( \text{abs} ( y(nnum+1,iierr) - \text{stepm}(iierr) ) ) \ast dt ; \\
\% \text{difference} \\
\% \text{cosine/cubic/Bezier/tanh} \\
\text{err3} &= \text{trapz} ( \text{abs} ( \text{stepm}(iierr) - \text{step}(iierr) ) ) \ast dt ; \\
\% \text{indicate when sum of error (difference) between steps (discrete vs. continuous)} \\
\% \text{and approximation to continuous step is less than approximation to discrete} \\
\% \text{step} \\
\text{if } ( \text{err2} + \text{err3} < \text{err1} ),
\begin{align*}
  \text{err4} &= 0 ; \\
  \text{else} \\
  \text{err4} &= 1 ;
\end{align*}
\text{end}
\% \text{results normalized as percentage of pulse area} \\
\text{errout} &= [ \text{errout} ; (\varepsilon/\tau) \ast 100, \text{nnum} , (\text{err1}/\tau) \ast 100, (\text{err2}/\tau) \ast 100, \\
(\text{err3}/\tau) \ast 100, \text{err4}) ] ; \\
\text{fprintf} ( '\%6.2f \%3d \%20.14f \%20.14f \%20.14f \%20.14f \%d\n', \text{errout} ) ;
\% \text{plot Fourier approximation results for} \\
\text{if } (\text{plotf} == 1),
\begin{verbatim}
\% \text{discrete step}
\end{verbatim}
\end{verbatim}
figure(1)
cif
plot( t(iierr), yo(nnum+1,iierr), 'g');
hold on
plot( t(iierr), step(iierr), 'r');

% continuous step
figure(2)
cif
plot( t(iierr), y(nnum+1,iierr), 'g');
hold on
% cosine/cubic/Bezier/tanh
plot( t(iierr), stepm(iierr), 'b');

% differences
figure(3)
cif
plot( t(iierr), yo(nnum+1,iierr) - step(iierr), 'r');
hold on
% cosine/cubic/Bezier/tanh
plot( t(iierr), y(nnum+1,iierr) - stepm(iierr), 'b');

% step profiles
figure(4)
cif
plot( t(iierr), step(iierr), 'g');
hold on
% cosine/cubic/Bezier/tanh
plot( t(iierr), stepm(iierr), 'b')
% axis([-T/2,T/2,0,1.3])

end
end
end

% output results
fid = fopen('[cos/cub/bez/tanh]_errout','wt');
fprintf(fid,'%6.2f %3d %20.14f %20.14f %20.14f %d
',errout);
fclose(fid);
APPENDIX A.3 – ERROR RESULTS COMPUTATION

With the error output defined as

\[
\text{errout} = \left[ \text{errout}; \left(\frac{e_{pi}}{\tau}\right) \times 100, n_{num}, \left(\frac{e_{r1}}{\tau}\right) \times 100, \left(\frac{e_{r2}}{\tau}\right) \times 100, \left(\frac{e_{r3}}{\tau}\right) \times 100, \text{err4} \right];
\]

\% fprintf('%6.2f %3d %20.14f %20.14f %20.14f %d\n',errout);

the results are computed by

\% results.m
\clear all;
\% parameters
\m_{e_{pi}} = 1;
\m_{num} = 2;
\m_{e_{1}} = 3;
\m_{e_{2}} = 4;
\m_{e_{3}} = 5;
\m_{cnt} = 10;
\fn = {'\text{../cos/cos\_errout}' ;
\text{../cub/cub\_errout}' ;
\text{../bez/bez\_errout}' ;
\text{ ../tanh/tanh\_errout}'};
\err_{1} = []; \err_{2} = [];
\text{for f} = 1: \text{length} (\text{fn}(:,1)),
% read in data
\yn = \text{ sprintf('\%s', char(\text{fn}(f,1)))}
\y = \text{dlmread(yn)};
\text{figure(1)}
\text{clf}
% loop on epsilon (% of \tau)
\text{for i} = 1: \m_{cnt},
% get entries for epsilon
\ii_{1} = \text{find}(\y(:,1)==i);
% get entries for condition e_{2} + e_{3} < e_{1} within given epsilon
\ii_{1} = \text{find}(\y(\ii_{1},\m_{e_{2}}) + \y(\ii_{1},\m_{e_{3}}) \text{ < } \y(\ii_{1},\m_{e_{1}}));
% save min/max N that meets condition for each epsilon and associated e_{1}, e_{2}, e_{3} error
\text{imin} = \text{min}(\y(\ii_{1}(\ii_{1}),\m_{num}))
\text{imax} = \text{max}(\y(\ii_{1}(\ii_{1}),\m_{num}))
\ii_{2} = \text{find}(\y(\ii_{1}(\ii_{1}),\m_{num}) == \text{imax});
\err_{1} = [\err_{1};[f, i, \text{im}_{1}, \text{im}_{2}, y(\ii_{1}(\ii_{1}(\ii_{2})), \m_{e_{1}}), y(\ii_{1}(\ii_{1}(\ii_{2})), \m_{e_{2}}),
\text{y}(\ii_{1}(\ii_{1}(\ii_{2})), \m_{e_{3}})]];
% plot error data
\text{% plot}(\y(\ii_{1},\m_{num}),y(\ii_{1},\m_{e_{1}}),'r');
\text{% hold on}
\text{% plot}(\y(\ii_{1},\m_{num}),y(\ii_{1},\m_{e_{2}}),'g');
\text{% plot}(\y(\ii_{1},\m_{num}),y(\ii_{1},\m_{e_{2}})+y(\ii_{1},\m_{e_{3}}),'b');
% save data for e_{1}, e_{2}, and e_{2} + e_{3}
\err_{2} = [\err_{2};[f+0*y(\ii_{1},\m_{num}),i+0*y(\ii_{1},\m_{num}), y(\ii_{1},\m_{num}), y(\ii_{1},\m_{e_{1}}), y(\ii_{1},\m_{e_{2}}),
\text{y}(\ii_{1},\m_{e_{2}}) + \text{y}(\ii_{1},\m_{e_{3}})]];
end
\text{figure(2)}
\text{clf}
\text{for i} = 1: \m_{cnt},
% get entries for file number
j = find(err1(:,1)==f)

% plot maximum N that meets condition e2+e3 < e1
plot(err1(j,2),err1(j,3),'r')

end

end

% write error summary to file
fid = fopen('res_errout','wt');
fprintf(fid,'%d %d %d %d %20.14f %20.14f %20.14f
',err1');
fclose(fid)

% write error data to file
save('res_errout_all','err2','-ascii','-tabs');
APPENDIX A.4 – MATLAB FUNCTION EXAMPLES

x=[0 1 2 3 4 5 6 7 8]'
y=[0 2 -5.5 3.1 0.1 5 4 4 0]'
i=find(y>3)
y(i)
trapz(y)
interp1(x,y,1.5)
interp1(x,y,7.5)

---results:

x -

0
1
2
3
4
5
6
7
8

y -

0
2.0000
-5.5000
3.1000
0.1000
5.0000
4.0000
4.0000
0

i -

4
6
7
8

ans -

3.1000
5.0000
4.0000
4.0000

ans -

12.7000

ans -

-1.7500

ans -

2

>>
APPENDIX A.5 – APPROXIMATION EXAMPLES - COSINE

Cosine (epsilon = 1% and 50%, N=20)

Cosine (epsilon = 1% and 50%, N=75)
Cosine (epsilon = 1%, N=20)

Continuous Step & Partial Fourier Series Sum

Time [seconds]

Cosine (epsilon = 1%, N=75)

Continuous Step & Partial Fourier Series Sum

Time [seconds]
Cosine (epsilon = 1%, N=20)

Cosine (epsilon = 1%, N=75)
Cosine (epsilon = 50%, N=20)

Cosine (epsilon = 50%, N=75)