

[6.56] Fourth-order splice joins two functions

A fourth-order splice joins two functions smoothly over an interval. The splice function matches the two functions at the interval boundaries, and the first derivatives of the splice equal those of the two functions. The fourth-order splice uses a point in the interval interior which controls the splice behavior. This splice is useful to join two functions, such as regression models, resulting in a model (with three equations) which is continuous and smooth, in the first derivative sense, over the interval of interest.

This tip is organized in these sections:

- Splice function derivation*
- The function splice4()*
- Example: approximate the sin() function*
- Differentiating the splice*
- Integrating the splice*
- Solving for the splice inverse*
- User interface program for splice4()*
- Scaling the derivatives*
- Scaling the splice integral*

These programs and functions are developed and described:

<i>splice4()</i>	Calculate the splice function coefficients
<i>spli4ui()</i>	User interface for <i>splice4()</i>
<i>spli4de()</i>	Calculate the splice derivative
<i>spli4in()</i>	Calculate a numeric derivative for the splice
<i>spli4x()</i>	Calculate x, given the splice value of y. Uses <i>nsolve()</i>
<i>spli4inv()</i>	Find an approximate polynomial for the inverse of the splice function

Splice function derivation

Define

- x_1 = the left interval bound
- x_3 = the right interval bound
- x_2 = the interval interior control point, where $x_1 < x_2 < x_3$
- $h = x_2 - x_1 = x_3 - x_2$ (the splice half-width; the control point is midway between the interval bounds)
- $f_1(x)$ is the left-hand function to be spliced
- $f_2(x)$ is the right-hand function to be spliced
- $s(x)$ is the splice function

In general, the splice function is $s(x) = a \cdot x^4 + b \cdot x^3 + c \cdot x^2 + d \cdot x + e$ [1]

and the first derivative is $s'(x) = 4 \cdot a \cdot x^3 + 3 \cdot b \cdot x^2 + 2 \cdot c \cdot x + d$ [2]

x_2 is the point between x_1 and x_3 where we specify some desired value for the splice function $s(x)$. There are several choices for $s(x_2)$. For example, if $f_1(x)$ and $f_2(x)$ intersect, we may want to set x_2 as the point of intersection, and set $s(x_2) = f_1(x_2) = f_2(x_2)$. Alternatively, the splice may work better by choosing a value for x_2 slightly away from the intersection. If the functions do not intersect on the splice interval, we may set $s(x_2)$ between the functions so the splice passes smoothly from one to the other.

We solve for the five coefficients of the splice polynomial by imposing five conditions:

1. $f_1(x_1) = s(x_1)$ The splice matches f_1 at x_1 ; there is no discontinuity
2. $f_2(x_3) = s(x_3)$ The splice matches f_2 at x_3 ; there is no discontinuity
3. $s(x_2) = y_2$ We set the splice value at x_2 to get some desired behavior
4. $\frac{d}{dx}f_1(x_1) = \frac{d}{dx}s(x_1)$ Set the first derivatives equal at x_1 for a smooth transition
5. $\frac{d}{dx}f_2(x_3) = \frac{d}{dx}s(x_3)$ Set the first derivatives equal at x_3 for a smooth transition

so we solve these five equations for a, b, c, d and e:

$$a \cdot x_1^4 + b \cdot x_1^3 + c \cdot x_1^2 + d \cdot x_1 + e = f_1(x_1) \quad (\text{condition 1}) \quad [3]$$

$$a \cdot x_3^4 + b \cdot x_3^3 + c \cdot x_3^2 + d \cdot x_3 + e = f_2(x_3) \quad (\text{condition 2}) \quad [4]$$

$$a \cdot x_2^4 + b \cdot x_2^3 + c \cdot x_2^2 + d \cdot x_2 + e = y_2 \quad (\text{condition 3}) \quad [5]$$

$$4 \cdot a \cdot x_1^3 + 3 \cdot b \cdot x_1^2 + 2 \cdot c \cdot x_1 + d = f_1'(x_1) \quad (\text{condition 4}) \quad [6]$$

$$4 \cdot a \cdot x_3^3 + 3 \cdot b \cdot x_3^2 + 2 \cdot c \cdot x_3 + d = f_2'(x_3) \quad (\text{condition 5}) \quad [7]$$

This set of five equations in five unknowns can sometimes be solved, but just as often they cannot. As we typically fit the splice over a narrow range of x , a *singular matrix* error often results from loss of precision in solving for the coefficients. Even if the *singular matrix* error does not occur, the solved coefficients are large and alternating in sign, which means that the polynomial is numerically unstable. This problem is avoided by finding the splice polynomial as a function of a scaled value of x_s , instead of x itself. As usual, proper scaling simplifies the solution. For example, suppose we set $x_{1s} = -1$, $x_{2s} = 0$ and $x_{3s} = 1$, where x_{1s} , x_{2s} and x_{3s} are the scaled values of x_1 , x_0 and x_3 , respectively. The splice polynomial is now $s(x_s)$:

$$s(x_s) = a \cdot x_s^4 + b \cdot x_s^3 + c \cdot x_s^2 + d \cdot x_s + e \quad [9]$$

Above, we defined $h = x_2 - x_1$; now we define a scaled interval half-width h_s :

$$h_s = x_{2s} - x_{1s} = x_{3s} - x_{2s} = 1 \quad [10]$$

We also define a scaling factor k such that

$$k = \frac{hs}{h} = \frac{1}{h} \quad [11]$$

Since we have scaled (transformed) the x - and y -data, we must also scale the derivatives used in [6] and [7]. It turns out that we can easily transform the derivatives as

$$f'_{1s}(x_{1s}) = \frac{1}{k} \cdot f'_1(x_1) \quad [12]$$

$$f'_{2s}(x_{3s}) = \frac{1}{k} \cdot f'_2(x_3) \quad [13]$$

These relations are derived in a section at the end of this tip. We can write [3] to [7] as a matrix equation

$$M \cdot c = v \quad [14]$$

where c is the coefficient vector we need, and

$$M = \begin{bmatrix} x_{1s}^4 & x_{1s}^3 & x_{1s}^2 & x_{1s} & 1 \\ x_{2s}^4 & x_{2s}^3 & x_{2s}^2 & x_{2s} & 1 \\ x_{3s}^4 & x_{3s}^3 & x_{3s}^2 & x_{3s} & 1 \\ 4 \cdot x_{1s}^3 & 3 \cdot x_{2s}^2 & 2 \cdot x_{1s} & 1 & 0 \\ 4 \cdot x_{3s}^3 & 3 \cdot x_{3s}^2 & 2 \cdot x_{3s} & 1 & 0 \end{bmatrix} \quad c = \begin{bmatrix} a \\ b \\ c \\ d \\ e \end{bmatrix} \quad v = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y'_{1s} \\ y'_{3s} \end{bmatrix}$$

We solve [14] for c as $c = M^{-1} \cdot v$ [15]

x_{1s} , x_{2s} and x_{3s} are always -1, 0 and 1, so M^{-1} is a constant matrix:

$$M^{-1} = \begin{bmatrix} -0.5 & 1 & -0.5 & -0.25 & 0.25 \\ 0.25 & 0 & -0.25 & 0.25 & 0.25 \\ 1 & -2 & 1 & 0.25 & -0.25 \\ -0.75 & 0 & 0.75 & -0.25 & -0.25 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \quad [16]$$

Since the scaled matrix inverse is independent of the functions and the splice points, we need not calculate it each time we find a splice: it is pre-computed and appears as a constant in the program. Because of the scaling we chose, the inverse matrix elements are simple and exact as shown. This means no round-off error can result from matrix inverse, which would happen with un-scaled x -values.

The c vector in [15] is the scaled coefficients; but we want the un-scaled result $y = s(x)$. This is found with equation [9] where

$$x_s = \frac{(x-x_2)}{h} \quad [17]$$

so $s(x)$ can be calculated with the TI-89/92+ `polyEval()` function:

```
polyEval({a,b,c,d,e},k*(x-x2))
```

Do not expand the splice polynomial with the relationship shown in [17]! While this may seem to simplify the polynomial, it undoes the benefit of scaling, and the resulting polynomial is again unstable.

Ultimately we use a single function for $f(x)$, which is implemented as a nested `when()` function:

```
when(x<x1,f1(x),when(x<x3,s(x),f2(x)))
```

or, in terms of the splice evaluation

```
when(x<x1,f1(x),when(x<x3,polyEval({a,b,c,d,e},k*(x-x2)),f2(x)))
```

There are several typical operations we want to perform with the splice, such as interpolation, differentiation, integration, and solving for the inverse, that is, solve $y=s(x)$ for x , given y . Interpolation is a simple matter of scaling x to x_s and evaluating the splice polynomial, as just shown. Following sections show how to do integration, differentiation, and solving for the inverse. I also include a user interface routine to find the splice coefficients, which eases entry of the input parameters and calculates the fit errors.

You can use the same basic ideas I have described to develop higher-order splices. For example, you could use a 6th-order splice which forces the second derivatives to match at the interval endpoints, or an 8th-order splice that forces matching second- and third-order derivatives at the endpoints. You could change the 4th-order splice to a fifth-order splice which specifies a first derivative at the interval midpoint x_2 .

The function splice4()

The function shown below, *splice4()*, implements the technique described above. The arguments are defined in the function header comments

splice4() returns the splice polynomial coefficients as a list {a,b,c,d,e}, which can be used directly with *polyEval()* to find $s(x)$. The listing below includes comments which are not in the *splice4.9xf* file in *tlcode.zip*.

```

splice4(y_1,y_2,y_3,yp1,yp3)
Func
@(y1,y2,y3,y1',y3') 4th-order splice
@27jan02/dburkett@infinet.com

© Input arguments

© f1(x) and f2(x) are the two functions to be spliced between x1 and x3.

© y_1  f1(x1)
© y_2  s(x2)
© y_3  f2(x3)
© yp1  f1'(x1), scaled such that f1'(x1) = h*f1'(x1)
© yp3  f2'(x3), scaled such that f2'(x3) = h*f2'(x3)

© Output:
© Returns {a,b,c,d,e} where s(xs) = a*(xs)^4 + b*(xs)^3 + c*(xs)^2 + d*(xs) + e

© Solve for scaled coefficients
© Find list 'c' of scaled coefficients {a, b, c, d, e} by solving
©
©          --  --
©          | y_1 | (y1 = f1(x1))
©          | y_2 | (y2)
© c = M^(-1) * | y_3 | (y3) = f2(x3)
©              | yp1 | (scaled value of f1'(x1))
©              | yp3 | (scaled value of f2'(x3))
©          --  --
©
© M^(-1) has been pre-computed. The following expression is all one line.

mat>list([- .5,1,- .5,- .25,.25;.25,0,- .25,.25,.25;1,-2,1,.25,- .25;- .75,0,.75,- .25,- .25;0,1,
0,0,0]*[y_1;y_2;y_3;yp1;yp3])

EndFunc

```

The following example uses *splice4()* to splice two approximating functions.

Example: approximate the sin() function

Suppose we want to estimate $\sin(x)$ for $0 < x < 0.78$ radians. We have found these estimating functions:

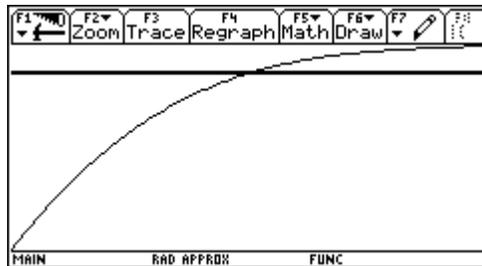
$$f_1(x) = k_1 \cdot x^3 + k_2 \cdot x^2 + k_3 \cdot x + k_4 \quad \text{for } x > 0 \text{ and } x < -0.546 \quad [18]$$

$$\begin{aligned} k_1 &= -0.15972286692682 & k_3 &= 1.0003712707863 \\ k_2 &= -0.00312600795332 & k_4 &= -4.74007298E - 6 \end{aligned}$$

$$f_2(x) = k_5 \cdot x^2 + k_6 \cdot x + k_7 \quad \text{for } x > -0.546 \text{ and } x < 0.78 \quad [19]$$

$$\begin{aligned} k_5 &= -0.3073521499375 & k_7 &= -0.04107647476031 \\ k_6 &= 1.1940610623813 \end{aligned}$$

The first step is to set the center of the splice, x_2 . We want to set the splice center near the boundary between $f_1(x)$ and $f_2(x)$, so as a starting point we plot the difference between the two estimating functions, which is $f_2(x) - f_1(x)$. We plot the difference instead of the functions themselves, because both functions are so close to $\sin(x)$ that we would not be able to visually distinguish any difference. The plot below shows the difference over the range $0.45 < x < 0.65$.



For this example the difference plot crosses the x-axis, so the functions intersect at this point. We choose this point as the splice center, so that the splice will not have to span a large distance between the two functions. Solving for the root gives

$$x_2 = 0.549220479094$$

In some cases, the two functions will not intersect at all, or at least in the desired splice range. In this case, a good choice for the splice center is the value of x at which the function difference is a minimum.

The next step is to set the width of the splice, which in turn sets the half-width h . Assuming the two estimating functions are both reasonably accurate, we want to make the splice rather narrow, but if we make it too narrow, accuracy degrades because of loss of significant digits when solving for the splice

coefficients. For this example, I set $h = 0.001$, and we will verify that this is not too small. This means that the splice will be fit from about $x_1 = 0.54822$ to $x_3 = 0.55022$.

All that remains is to calculate the other `splice4()` arguments, with these steps.

In the Y= editor, define some functions to make the calculations (and plotting) easier. Note that the coefficients k_1 to k_6 have already been stored.

```

y1= polyEval ({k1,k2,k3,k4}, x)
y2= polyEval ({k5,k6,k6}, x)
y3= sin(x)
y4= y1(x)-y3(x)
y5= y2(x)-y3(x)
y6= polyEval (sp14, (x-x2)*k)
y7= y6(x)-y3(x)

```

y_1 and y_2 are $f_1(x)$ and $f_2(x)$. y_3 is the function to be estimated. y_4 and y_5 find the error between the model equations and the function to be estimated. y_6 will calculate the splice function, and y_7 will find the splice function error. Note that $k = 1/h$.

Next, enter these commands in the entry line:

```

.54922048→x2
.001→h
1/h→k
y1(x2-h)→yy1
y1(x2)→yy2
y2(x2+h)→yy3
h*d(y1(x), x) | x=x2-h→yd1
h*d(y2(x), x) | x=x2+h→yd2

```

Finally, call the `splice4()` with the calculated arguments and save the result in `sp14`:

```

math\splice4(yy1,yy2,yy3,yd1,yd2)→sp14

```

Next we will ensure that the splice function meets the requirements. First, check the errors at x_1 , x_2 and x_3 . These expressions should all be near zero:

```

y1(x2-h)-y6(x2-h)    returns 2E-14
y1(x2)-y6(x2)        returns 0
y2(x2+h)-y6(x2+h)    returns 2E-14

```

Next check the derivative errors at the splice endpoints. The derivative errors should be near zero. If we calculate the derivative errors (*incorrectly!*) by directly differentiating the splice function, we get

```

d(y1(x), x)-d(y6(x), x) | x=x2-h    returns 3.7629E-8
d(y2(x), x)-d(y6(x), x) | x=x2+h    returns 7.446E-9

```

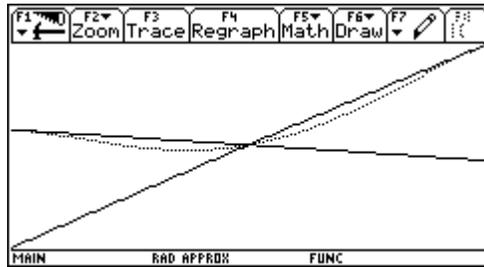
The errors seem small enough, but calculating the derivatives correctly (as shown below) gives the true errors:

```

d(y1(x0), x0)-k*d(polyEval(sp14,x), x) | x=-1 and x0=x2-h    returns -6.087E-11
d(y2(x0), x0)-k*d(polyEval(sp14,x), x) | x=1 and x0=x2+h     returns 3.601E-11

```

So far, the splice seems to work well. As a final check, we graph the splice function over the splice interval. In the Y= editor, use [F4] to select y_4 , y_5 and y_7 . Because $f_1(x)$, $f_2(x)$ and the splice function are all so close together, plotting these *error* functions gives a better picture of the splice fit. To set the x-axis range, press [WINDOW] and set x_{min} to x_2-h , and x_{max} to x_2+h . Press [F2] [A] to plot the splice, which results in this graph:



The two solid traces are $f_1(x)$ and $f_2(x)$, relative to $\sin(x)$, and the dotted trace is the splice function, relative to $\sin(x)$. Note that the splice slope appears to match f_1 and f_2 at the endpoints, and it passes through x_2 , as we specified. Based on the numerical check results, and the plotted result, it appears that we have a splice that we can use. (Note: to get the dotted trace, I set the plot Style for y_7 to Dot, and the plot resolution to 2).

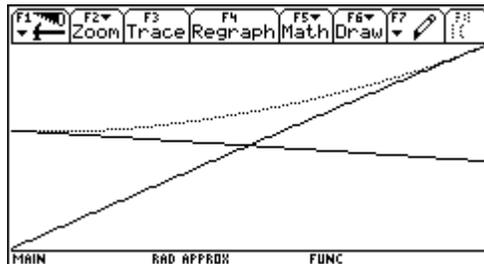
However, the splice is less than f_1 and f_2 over the entire range. Assuming that f_1 and f_2 accurately model the $\sin()$ function, we may prefer that the splice be above the two functions over the splice interval. This is accomplished by changing the value of y_2 . One reasonable value is the average of the values of $f_1(x_1)$, $f_1(x_2)$ and $f_2(x_3)$. This is easily calculated with

$$\text{mean}(\{y_1(x_2-h), y_1(x_2), y_2(x_2+h)\}) \rightarrow yy_2$$

We find and save the splice coefficients as before with

$$\text{math}\backslash\text{splice4}(yy_1, yy_2, yy_3, yd_1, yd_2) \rightarrow \text{sp14}$$

and the splice now looks like this:



Note that the splice slopes still match the functions at the interval endpoints, but now the splice is above both functions.

Differentiating the splice

To differentiate the splice, do not expand the splice polynomial and differentiate it: accuracy will suffer because of round-off error. Instead, first scale x to x_s , find $s'(x_s)$, then scale this result. In other words,

$$\frac{d}{dx} s(x) = \frac{1}{h} \cdot \frac{d}{dx_s} (a \cdot x_s^4 + b \cdot x_s^3 + c \cdot x_s^2 + d \cdot x_s + e) \quad [20]$$

or

$$\frac{d}{dx}s(x) = \frac{1}{h} \cdot (4 \cdot a \cdot x_S^3 + 3 \cdot b \cdot x_S^2 + 2 \cdot c \cdot x_S + d) \quad [21]$$

where $x_S = k \cdot (x - x_2)$

The higher-order derivatives are

$$\frac{d^2}{dx^2}s(x) = \frac{1}{h^2} \cdot (12 \cdot a \cdot x_S^2 + 6 \cdot b \cdot x_S + 2 \cdot c) \quad [22]$$

$$\frac{d^3}{dx^3}s(x) = \frac{1}{h^3} \cdot (24 \cdot a \cdot x_S + 6 \cdot b) \quad [23]$$

$$\frac{d^4}{dx^4}s(x) = \frac{24}{h^4} \cdot a \quad [24]$$

The following function can be used to find the splice derivative of any order.

```
spli4de(x,x2,h,c1,o)
Func
@(x,x2,h,k,{list},order) 4th-order splice derivative
©9apr02/dburkett@infinet.com

© Input arguments
©
© x      The point at which to find the derivative
© x2     The splice interval midpoint
© h      The splice interval half-width
© c1     the list of splice function coefficients
© o      the order of the derivative; o > 0

local xs,a,b,c,d,k

c1[1]→a      © Extract polynomial coefficients
c1[2]→b
c1[3]→c
c1[4]→d
1/h→k        © Invert half-width to simplify later calculations
k*(x-x2)→xs  © Scale x

© Find derivative or return error string. The following when() is all on one line.

when(o≤0,"spli4de err",
when(o=1,k*polyeval({4*a,3*b,2*c,d},xs),
when(o=2,k^2*polyeval({12*a,6*b,2*c},xs),
when(o=3,k^3*(24*a*xs+6*b),when(o=4,k^4*24*a,0))))

EndFunc
```

spli4de() returns the string "spli4de err" if the order *o* is less than 1. No testing is done to ensure the order is an integer. Note that derivatives with order greater than four are zero.

You should not, in general, use the splice function to estimate derivatives of order greater than the first, even though *spli4de()* provides that capability. Since we have not constrained the splice to match the

higher order derivatives, they can vary wildly. In the example above, I fit a splice to two simple polynomials. The table below shows the derivative errors at the splice boundaries x_1 and x_3 .

Derivative order	f1'(x1)	Error at x1	f2'(x3)	Error at x3
1	0.8529	6.09E-11	0.8558	-3.6E-11
2	-0.5316	-6.06	-0.6147	-6.10
3	-.9583	24,258	0	-24,381
4	0	-2.43E7	0	-2.43E7

As expected, the first derivative errors at the splice boundaries are small, but the errors are so large for derivatives of orders two and greater as to make the derivative estimate useless.

Integrating the splice

As with the derivative, we find definite integrals with the scaled polynomial and the scaled x_s values. If

$$I = \int_{x_a}^{x_b} s(x) dx \quad \text{and} \quad I_s = \int_{x_{sa}}^{x_{sb}} s_s(x_s) dx_s$$

then $I = h \cdot I_s$

which is derived in a later section in this tip. We integrate the scaled splice function $s_s(x_s)$ with the scaled limits x_{sa} and x_{sb} found with

$$x_{sa} = \frac{1}{h} \cdot (x_a - x_2) \quad x_{sb} = \frac{1}{h} \cdot (x_b - x_2)$$

The following function can be used to integrate the splice function.

```
spli4in(xa,xb,x2,h,c1)
Func
@(xa,xb,x2,h,{list}) 4th-order splice integral
©9apr02/dburkett@infinet.com

© Input arguments:
© xa lower integration limit
© xb upper integration limit
© x2 splice interval midpoint
© h splice interval half-width
© c1 splice coefficient list

h*f(polyeval(c1,xs),xs,(xa-x2)/h,(xb-x2)/h)

EndFunc
```

spli4in() does not check x_a and x_b to ensure that they are within splice interval. Results may be inaccurate when integrating beyond the interval limits.

Solving for the splice inverse

Some applications require finding x such that $s(x)=y$, given y . This can only be done if the splice is strictly monotonic on the splice interval, that is, the splice $s(x)$ has no minimum or maximum on the interval.

If the splice is monotonic, finding the inverse is a simple matter of using `nSolve()` as shown in the function below. We solve the scaled splice function, then un-scale the returned x_s to find x . Since we are using the scaled splice function, we know that the solution x_s is greater than -1 and less than 1, so we use those as the solution bounds.

```
spli4x(y,x2,h,c1)
Func
@(s(x),x2,k,{list}) 4th-order splice inverse
@9apr02/dburkett@infinet.com

© Input arguments:
© y value of s(x) at which to find x
© x2 splice interval midpoint
© h splice interval half-width
© c1 list of splice coefficients

(nSolve(polyEval(c1,xs)=y,xs)|xs>=-1 and xs<=1)*h+x2

EndFunc
```

This method is satisfactory if you only need a few values of the inverse. There is a faster method to find many values: find an inverse fourth-order splice to calculate x directly, given $f(x)$. This is simple once we have found the original splice for $y = f(x)$. We can use `splice4()` to find the inverse splice; we just exchange the x 's for y 's and correct the end-point derivatives:

```
splice4(xx1,xx2,xx3,xd1,xd2)
```

where $xx1$ and $xx3$ are the interval bounds, as before. For the scaling to work correctly, however, $s(xx2)$ must be halfway between $yy1$ and $yy3$, so we find $yy2$ with

$$yy1 = f_1(xx1) \quad yy3 = f_2(xx3) \quad yy2 = \frac{yy1+yy3}{2}$$

Given $yy2$, we solve for the corresponding $xx2$ with `spli4x()` above:

```
spli4x(yy2,xx2x,hx,sp14x)->xx2
```

but note that $xx2x$, hx , and `sp14x` are the values for the *original* splice, that is, $y = s(x)$. Find they y -axis half-width hy with

```
abs(yy2-yy1)-hy
```

then you can find the inverse derivatives with

```
hy*(d(f1(x),x)|x=xx1)->xd1
hy*(d(f2(x),x)|x=xx3)->xd2
```

We now have all the arguments for `splice4()`. To find x given $s(x)$, use

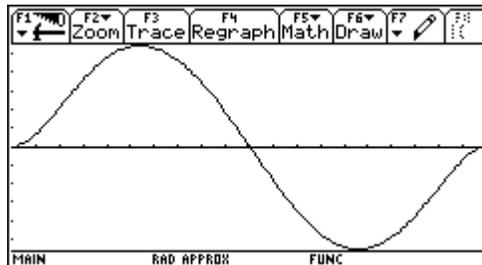
```
polyeval(sp14y,ky*(s(x)-yy2))
```

where $ky = 1/hy$.

This inverse polynomial is not the true inverse of the original splice. It is simply a polynomial which passes through the same three points as the splice, and has the same derivatives at the endpoints. If the splice is used for a narrow interval, the accuracy can be very good. Before using the inverse splice, you should check the accuracy. This is easily accomplished by graphing the error in the estimated x-values, found from

$$\text{error} = \text{si}(s(x)) - x$$

where $s(x)$ is the splice function and $\text{si}(x)$ is the inverse splice function. For the example shown above, this is the error in x :



The error in x is about $\pm 2.4E-10$, so this inverse is accurate to about 9 significant digits. Note that the error is zero at the endpoints and in the center, as it should be.

The function `spli4inv()`, shown below, finds the inverse splice by applying the steps described above.

```
spli4inv(f1,f2,x2,h,c)
Func
© ("f1(x)", "f2(x)", x2, h, coeff_list)
© output: {a,b,c,d,e,y2,hy}
© 4th-order splice inverse polynomial
© 10may02/dburkett@infinet.com
© calls math\spli4x(), math\splice4()

© Input arguments:
©
© f1  f1(x), passed as a string, for example "y1(x)"
© f2  f2(x), passed as a string, for example "y2(x)"
© x2  splice midpoint for s(x)
© h   splice half-width for s(x)
© c   list of splice coefficients for s(x)

© Output: {a,b,c,d,e,y2,hy}
© where si(y) = a*ys^4 + b*ys^3 + c*ys^2 + d*ys + c
© and ys = (y-y2)/hy
© so find x = polyeval({a,b,c,d,e}, (y-y2)/hy)

local yy1,yy2,yy3,yd,x1,x2,x3,yp1,yp3,hy
©
© x1  the original left interval bound; treated as si(y1)
© x2  s(x) such that x2=si(yy2)
© yy1  f1(x1)
© yy2  the mean of yy1 and yy3
© yy3  f2(x3)
© hy   y-interval half-width
©

expr("define f1(x)="&f1)    © Define local functions to be spliced
```

```

expr("define f2(x)=%f2)

x2-h→x1          © Find splice interval bounds
x2+h→x3

f1(x1)→yy1      © Find si(x1), si(x3)
f2(x3)→yy3
(yy1+yy3)/2→yy2 © Find splice interval midpoint
math\splice4x(yy2,x2,h,c)→x2 © Solve for x at interval midpoint
abs(yy2-yy1)→hy © Find y-axis half-width
hy/(d(f1(x),x)|x=x1)→xp1 © Find dx/dy at x1 and x2
hy/(d(f2(x),x)|x=x3)→xp3

augment(math\splice4(x1,x2,x3,xp1,xp3),{yy2,hy}) © Solve for splice coefficients

EndFunc

```

Note that the output list includes $y2$ and hy in addition to the coefficient list, since you will need them to evaluate the inverse polynomial. If you have saved the output list in *list1*, then use

```
left(list1,5)
```

to extract just the polynomial coefficients, and

```
list[6]
```

to get $y2$, and

```
list[7]
```

to get hy .

User interface program for splice4()

The user interface program shown below, *spli4ui()*, automates the process of calculating a splice and checking the results. *spli4ui()* finds the splice coefficients and saves them. It also calculates and displays the errors for $s(x)$ and the first derivatives.

spli4ui() uses a symbolic variable \check{a} . If you have a variable \check{a} in the current folder, it will be deleted.

spli4ui() assumes that the derivatives of the functions to be spliced can be found with the built-in derivative function. For functions which cannot be differentiated by the calculator, you can still calculate a splice, but you cannot use *spli4ui()*. See tip [6.26], *Accurate numerical derivatives with nDeriv() and Ridder's method*, to find the necessary derivatives.

```

sp14ui()
Prgm
©splice4() user interface
©16apr02/dburkett@infinet.com

© Calls math\splice4()

© Local variables
local
l,m,errx1,errx2,errx3,errd1,errd3,x1,äx2,x3,äh,yy1,äyy2,yy3,k,äf1,äf2,yp1,yp2,sp1c,äout

© l          drop-down selection variable
© m          drop-down selection variable

```

```

© äf1          f1(x)
© äf2          f2(x)

© x1          lower interval bound
© äx2         interval midpoint
© x3          upper interval bound

© yy1         f1(x1)
© äyy2        s(x2)
© yy3         f2(x3)

© äh          interval half-width
© yp1         f1'(x1)
© yp2         f2'(x3)
© k           scaling factor
© splc        splice coefficients list
© äout        name of coefficient list output variable

© errx1       splice error at x1
© errx2       splice error at x2
© errx3       splice error at x3
© errd1       derivative error at x1
© errd3       derivative error at x3

© Test for existing user input defaults variable spl4v; create if necessary
if gettype(spl4v)="NONE" then
  {"", "", "0.", ".001", "0.", ""}→spl4v
endif

© Extract user input defaults
spl4v[1]→äf1
spl4v[2]→äf2
spl4v[3]→äx2
spl4v[4]→äh
spl4v[5]→äyy2
spl4v[6]→äout

© Prompt for user input
dialog
  title "SPLI4UI"
  request "f1(x)", äf1
  request "f2(x)", äf2
  request "Coef var name", äout
  request "Splice center x2", äx2
  request "Splice half-width h", äh
  dropdown "s(x2) method:", {"mean", "center", "manual"}, m
  request "manual s(x2)", äyy2
enddlog
if ok=0: return                                © Return if [ESC] pressed

{äf1, äf2, äx2, äh, äyy2, äout}→spl4v          © Save user input defaults

expr(äx2)→äx2                                  © Convert user input strings
expr(äh)→äh
expr(äyy2)→äyy2
expr("define äf1(x)=%&äf1")                    © Create functions f1() and f2()
expr("define äf2(x)=%&äf2")

1/äh→k                                          © Find scaling factor
äx2-äh→x1                                      © ... and splice interval bounds
äx2+äh→x3

äf1(x1)→yy1                                    © Find splice bound y's
äf2(x3)→yy3

if m=1 then                                    © Set yy2 for s(x2) method:
  (yy1+äf1(äx2)+yy3)/3→äyy2                  © ... 'mean' method
elseif m=2 then
  (äf1(äx2)+äf2(äx2))/2→äyy2                © ... 'center' method

```

```

endif

d(af1(a),a)|a=x1→yp1          © Find derivatives at interval bounds
d(af2(a),a)|a=x3→yp3

math\splc4(yy1,äyy2,yy3,äh*yp1,äh*yp3)→splc          © Find splice coefficients
splc→#(spl4v[6])          © Save coefficients to user variable

polyeval(splc,k*(x1-äx2))-yy1→errx1          © Find splice errors at bounds and midpoint
polyeval(splc,0)-äyy2→errx2
polyeval(splc,k*(x3-äx2))-yy3→errx3

k*(d(polyeval(splc,a),a)|a=-1)-yp1→errd1          © Find derivative errors at bounds
k*(d(polyeval(splc,a),a)|a=1)-yp3→errd3

clrio          © Display errors on Program I/O screen
disp "Absolute errors:"
disp "s(x1): "&string(errx1)
disp "s(x2): "&string(errx2)
disp "s(x3): "&string(errx3)
disp "s'(x1): "&string(errd1)
disp "s'(x3): "&string(errd3)

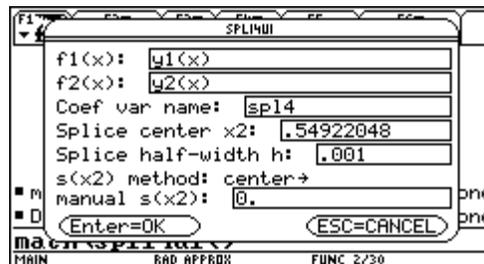
EndPrgm

```

I will use *spli4ui()* to create the splice shown in the example above. On starting *spli4ui()*, the following dialog box is displayed:



This screen shot shows the dialog box with the parameters entered:



For $f_1(x)$ and $f_2(x)$, I can just enter $y_1(x)$ and $y_2(x)$, since I have previously defined these functions in the Y= Editor. I set the splice center and half-width, and chose 'center' for the $s(x_2)$ method. I need not enter a value for 'manual $s(x_2)$ ', since I am not using the manual method. After pressing [ENTER], the errors are shown on the program I/O screen:

```

F1 F2 F3 F4 F5 F6 F7
|-----|-----|-----|-----|-----|-----|
| F1 F2 F3 F4 F5 F6 F7 |
|-----|-----|-----|-----|-----|-----|
Absolute errors:
s(x1): 0.e0
s(x2): 0.e0
s(x3): 0.e0
s'(x1): 2.088E-11
s'(x3): 3.99E-12
MAIN RAD APPROX FUNC 2/30

```

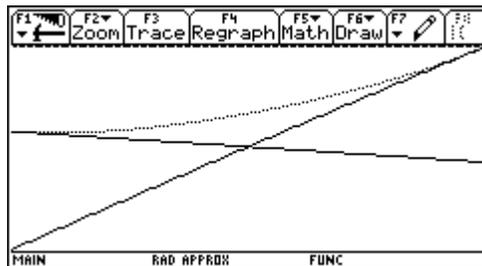
spli4ui() supports three options to set $s(x_2)$: center, mean and manual. With the 'manual' $s(x_2)$ method, you specify $s(x_2)$ in the 'manual $s(x_2)$ ' field. The 'center' method sets

$$s(x_2) = \frac{f_1(x_1) + f_2(x_2)}{2}$$

so use the 'center' method to force the splice midway between two functions which do not intersect, or to force the splice through $f_1(x_2) = f_2(x_2)$. The 'mean' method sets

$$s(x_2) = \frac{f_1(x_1) + f_1(x_2) + f_2(x_3)}{3}$$

so use this method to force the splice through a point away from both functions. For example, this plot shows the splice and function differences with the mean method:



Scaling the derivatives

Since the x - and y -data are scaled to avoid a singular matrix, we must also scale the derivatives of $f_1(x)$ and $f_2(x)$ to be consistent with the scaled x values. Starting with a general function

$$y = f(x)$$

and the general scaling equations

$$x_s = p \cdot x + q$$

$$y_s = r \cdot y + s$$

we solve for x and y ,

$$x = \frac{x_s - q}{p}$$

$$y = \frac{y_s - s}{r}$$

substitute these expressions in the function definition

$$\frac{y_s - s}{r} = f\left(\frac{x_s - q}{p}\right)$$

and solve for y_s

$$y_s = r \cdot f\left(\frac{x_s - q}{p}\right) + s$$

then take the derivative $\frac{dy_s}{dx_s} = r \cdot \frac{d}{dx_s} f\left(\frac{x_s - q}{p}\right)$

or $\frac{dy_s}{dx_s} = \frac{r}{p} \cdot \frac{d}{dx_s} f(x_s - q)$ [1A]

Now since $x_s = p \cdot x + q$

$$x_s + dx_s = p \cdot x + p \cdot dx + q = p \cdot x + q + p \cdot dx$$

$$x_s + dx_s = x_s + p \cdot dx$$

so $dx_s = p \cdot dx$ [2A]

Replace [2A] in [1A] $\frac{dy_s}{dx_s} = \frac{r}{p} \cdot \frac{d}{p \cdot dx} f(x_s - q)$

or $\frac{dy_s}{dx_s} = \frac{r}{p} \cdot \frac{d}{dx} f\left(\frac{x_s - q}{p}\right)$

or simply $\frac{dy_s}{dx_s} = \frac{r}{p} \cdot \frac{d}{dx} f(x)$

which is the relation we wanted. In this splice application, we scale only x , not y , so $r = 1$. In terms of the actual scaling variables x_2 and h , the scaling is

$$x_s = \frac{1}{h} \cdot (x - x_2) = \frac{1}{h}x - \frac{x_2}{h}$$

so by inspection $p = \frac{1}{h}$

then $\frac{dy_s}{dx_s} = h \cdot \frac{d}{dx} f(x)$

Scaling the splice integral

Given the scaled splice polynomial $s_s(x_s)$, we can easily find the definite integral

$$I_s = \int_{x_{sa}}^{x_{sb}} s_s(x_s) dx_s$$

but we really want the integral of the unscaled function:

$$I = \int_{x_a}^{x_b} s(x) dx$$

Using the definition of the definite integral:

$$I_s = \lim_{n \rightarrow \infty} \sum_{m=1}^n s_s(x_s) \cdot \Delta x_{sm} \quad [1B]$$

$$I = \lim_{n \rightarrow \infty} \sum_{m=1}^n s(x) \cdot \Delta x_m \quad [2B]$$

where

$$\Delta x_{sm} = \frac{x_{sb} - x_{sa}}{n} \quad [3B]$$

$$\Delta x_m = \frac{x_b - x_a}{n} \quad [4B]$$

Solve [3B] for n , replace in [4B], and define a new variable c such that

$$c = \frac{\Delta x_m}{\Delta x_{sm}} = \frac{x_b - x_a}{x_{sb} - x_{sa}} \quad [5B]$$

then
$$\Delta x_{sm} = \frac{\Delta x_m}{c} \quad [6B]$$

Replace [6B] in [1B] for
$$I_s = \lim_{n \rightarrow \infty} \sum_{m=1}^{\infty} s_s(x_s) \cdot \frac{\Delta x_m}{c} \quad [7B]$$

Now, since $s_s(x_s) = s(x)$, for any given x and its equivalent scaled x_s , we move the $1/c$ constant outside the sum and limit, and [7B] becomes

$$I_s = \frac{1}{c} \lim_{n \rightarrow \infty} \sum_{m=1}^{\infty} s(x) \cdot \Delta x_m$$

which, on comparison with [2b], is just
$$I_s = \frac{1}{c} \cdot I$$

or
$$I = c \cdot I_s \quad [8B]$$

We can also express [8B] in terms of the original scaling factor k . The scaling is defined by

$$x_{sa} = \frac{1}{h}(x_a - x_2) \quad x_{sb} = \frac{1}{h}(x_b - x_2)$$

which we can solve for h :
$$h = \frac{x_a - x_b}{x_{sa} - x_{sb}}$$

and comparison with [5B] gives
$$h = c$$

so we have the final desired result:
$$I = h \cdot I_s$$