Integrating a Discrete Function

2004 Autar Kaw, Loubna Guennoun, University of South Florida, kaw@eng.usf.edu, http://numericalmethods.eng.usf.edu

NOTE: This worksheet demonstrates the use of Maple to illustrate the integration of a discrete function using different methods.

Section I: Introduction

The following worksheet will illustrate how to integrate a discrete function. This is usually done either by calculating the area of the trapezoids defined by the data points or by integrating an interpolant. So here we show the answer by trapezoidal rule as well as with 4 types of interpolation. For more info on interpolation see the topic. The following outlines this sheet.

1. Trapezoidal Rule
2. Polynomial Interpolation
3. Spline Interpolation
    a. Linear spline
    b. Quadratic spline
    c. Cubic spline

[click here for textbook notes]

```
> restart;
```

Section II: Data

This section is the only section where the user may interact with the program. The user may enter any set of data *X*, and *Y*, and the lower and upper limit for the function to be integrated. By entering this data, the program will calculate the average value of the integral using the trapezoidal rule, polynomial interpolation, and spline interpolation. The X data needs to be in ascending order.

The following is the *y* vs *x* data. The data is presented as *(x,y)* pairs of data.

```
> XY:=[[0,320],[10,120],[15,620],[20,7720],[22.5,320],[30,620]];
```

$$XY := [[0, 320], [10, 120], [15, 620], [20, 7720], [22.5, 320], [30, 620]]$$

The lower limit of the integral *a*. Note that *a* is within the minimum and the maximum of X values.

```
> a:=0.1;
```

$$a := 0.1$$

The upper limit of the integral *b*. Note that *b* is within the minimum and the maximum of X values.

```
> b:=16.0;
```

$$b := 16.0$$

This is the end of the user's section. All information must be entered before proceeding to the next section.

Section III: Checking for Valid Inputs

```
> n:=nops(XY);
  X:=array(1..n):
  Y:=array(1..n):
  for i from 1 to n do
  X[i]:=XY[i,1]:
  Y[i]:=XY[i,2]:
  end do:
```

$$n := 6$$

The first check is to make sure that the lower limit *a* of the integral is equal or greater than the first input of the X array.

```
> lower_limit:=proc(a,xone,xn)
  local iflag:
  if a>=xone and a<=xn then
  iflag:=OK:
  RETURN (iflag);
  else
  iflag:=NotOK;
  printf("a can not be less than X1 or greater then Xn"):
  RETURN (iflag);
  fi;
  end proc:
```

```
> check_1:=lower_limit(a,X[1],X[n]);
```

$$check\_1 := OK$$

The second check is to make sure that the upper limit *b* of the integral is equal or less than the last unput of the X array.

```
> upper_limit:=proc(b,xone,xn)
  local iflag:
```

```
   if b>=xone and b<=xn then
   iflag:=OK:
   RETURN (iflag);
   else
   iflag:=NotOK;
   printf("b can not be less than X1 or greater than Xn"):
   RETURN (iflag);
   fi;
   end proc:
```

```
> check_2:=upper_limit(b,X[1],X[n]);
```

check_2 := *OK*

The third check is to make sure that the user put enough data in both the X and Y arrays.

```
> n_data:=proc(nx,ny)
   local iflag:
   if nx=ny then
   iflag:=OK:
   RETURN (iflag);
   else
   iflag:=NotOK;
   printf("The number of data on the X array has to be equal to
   the number of data on the Y array"):
   RETURN (iflag);
   fi;
   end proc:
```

```
> check_3:=n_data(nops(X),nops(Y));
```

check_3 := *OK*

The last check is to make sure the data in the X array is in ascending order.

```
> x_asc:=proc(x)
   local i,iflag:
   for i from 2 by 1 to n do
   if X[i]>X[i-1] then
   iflag:=OK:
   RETURN (iflag);
   else
   iflag:=NotOK;
   printf("The data in the X array has to be in an ascending
   order"):
   RETURN (iflag);
   fi;
```

```
   end do;
   end proc:
```

```
> check_4:=x_asc(X);
```

$$check\_4 := OK$$

The following spreadsheet displays the results of checking the validity of the user's input. If all the check's results is "OK", then all the input data are valid.

```
> with( Spread ):
> EvaluateSpreadsheet(Input Check):
```

	A	B
1	*Check*	*Results*
2	*Lower limit check*	*OK*
3	*Upper limit check*	*OK*
4	*Number of data check*	*OK*
5	*Ascending order check*	*OK*

Section IV: Integrating a Discrete Function

Trapezoidal Rule

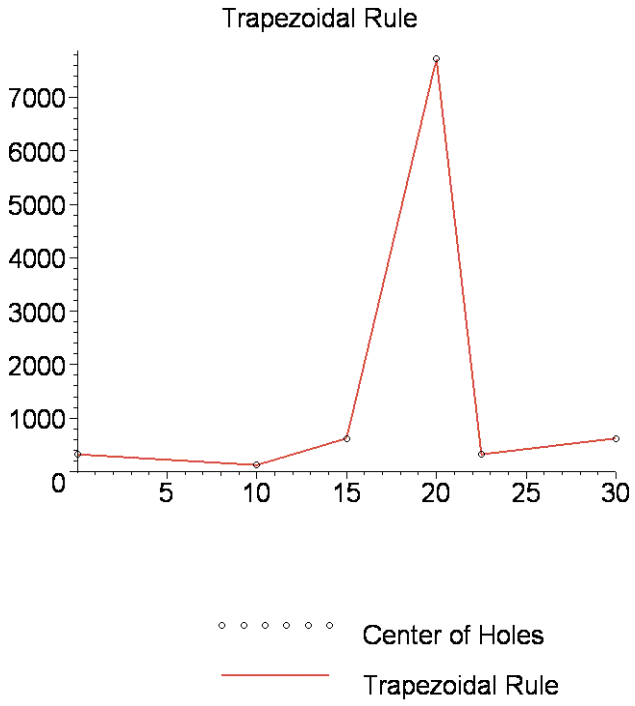
```
> trap:=proc(a,b,X,Y,n)
  local S_trap,S_1,S_2,Y_a,Y_b,l_index,u_index,i:

  l_index:=1:
  u_index:=n:

  S_1:=0:
  S_2:=0:

  for i from 1 by 1 to n do

  if a>=X[i] then
  l_index:=i:
  Y_a:=Y[i]+(Y[i+1]-Y[i])/(X[i+1]-X[i])*(a-X[i]):
  S_1:=(X[l_index+1]-a)*(Y[l_index+1]+Y_a)/2.0:
```

```
    end if:

    if b>X[i] then
    u_index:=i:
    Y_b:=Y[i]+(Y[i+1]-Y[i])/(X[i+1]-X[i])*(b-X[i]):
    S_2:=(b-X[u_index])*(Y_b+Y[u_index])/2.0:
    end if:

    end do:

    S_trap:=S_1+S_2:
    for i from (l_index+1) by 1 to (u_index-1) do
    S_trap:=S_trap+(X[i+1]-X[i])*(Y[i+1]+Y[i])/2.0:
    end do:

    return(S_trap):
    end proc:
```

```
> S_trap:=trap(a,b,X,Y,n);
```

$$S\_trap := 5348.100000$$

```
> trap:=x->spline(X,Y,x,linear):
  plot([XY,trap],X[1]..X[n],style=[POINT,LINE],symbol=CIRCLE,sy
  mbolsize=15,thickness=2,title="Trapezoidal
  Rule",color=[BLACK,ORANGE],legend=["Center of
  Holes","Trapezoidal Rule"]);
```



Polynomial Interpolation

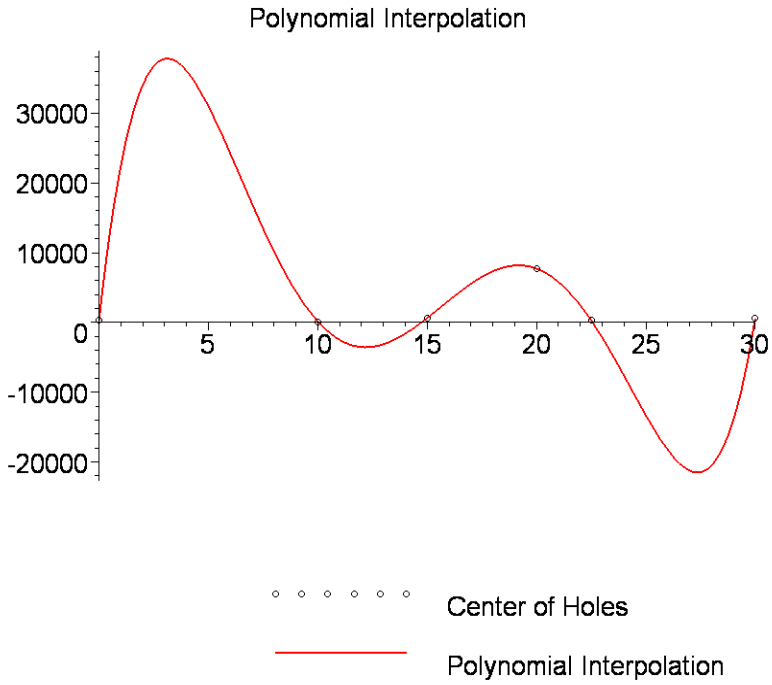
```
> poly_inter:=interp(X,Y,x);
```

$$poly\_inter := 0.2877333332\,x^5 - 22.24333332\,x^4 + 28715.99999\,x + 611.5166663\,x^3 - 7052.166663\,x^2 + 320$$

```
> S_polyinter:=int(poly_inter,x=a..b);
```

$$S\_polyinter := 210921.2154$$

```
> poly_inter:=x->interp(X,Y,x):
  plot([XY,poly_inter],X[1]..X[n],style=[POINT,LINE],symbol=CIR
  CLE,symbolsize=15,thickness=2,title="Polynomial
  Interpolation",color=[BLACK,RED],legend=["Center of
  Holes","Polynomial Interpolation"]);
```



Spline Interpolation

Linear Spline

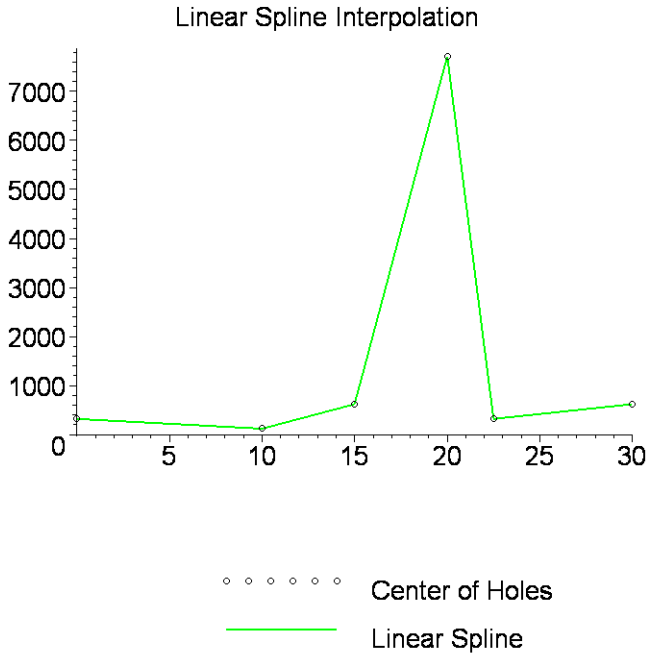
```
> linear_spline:=spline(X,Y,x,linear);
```

$$linear\_spline := \begin{cases} 320 - 20\,x & x < 10 \\ -880 + 100\,x & x < 15 \\ -20680 + 1420\,x & x < 20 \\ 66920.00000 - 2960.000000\,x & x < 22.5 \\ -580.0000000 + 40.00000000\,x & otherwise \end{cases}$$

```
> S_linear:=int(linear_spline,x=a..b);
```

$$S\_linear := 5348.100000$$

```
> linear_spline:=x->spline(X,Y,x,linear):
  plot([XY,linear_spline],X[1]..X[n],style=[POINT,LINE],symb
  ol=CIRCLE,symbolsize=15,thickness=2,title="Linear Spline
  Interpolation",color=[BLACK,GREEN],legend=["Center of
  Holes","Linear Spline"]);
```



Quadratic Spline

```
> quadratic_spline:=spline(X,Y,x,quadratic);
```

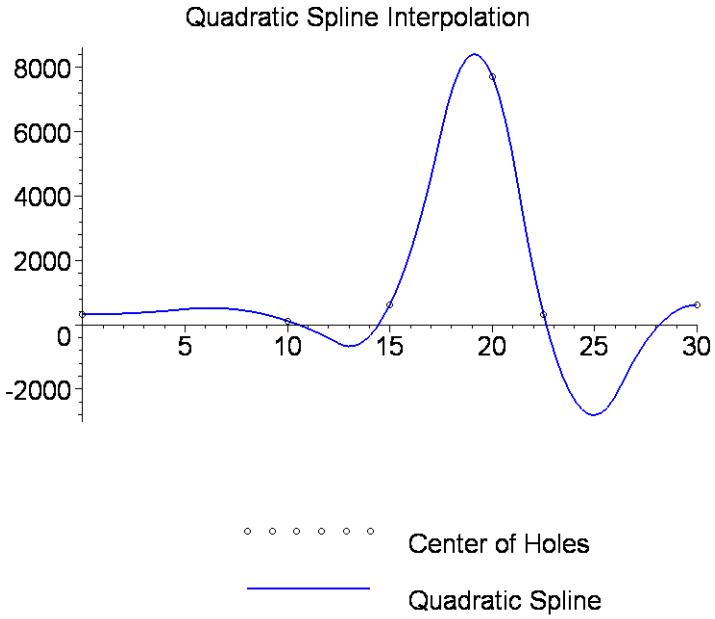
$$quadratic\_spline := \{\ 320. + 6.57428791499999932\ x^2\ ,\ x < 5.$$
$$2234.857582 - 211.485758199999992\ x - 27.7228637300000004\ (x - 10)^2\ ,$$
$$x < 12.5000000000000000$$
$$-19055.28868 + 1311.68591199999992\ x + 332.357197799999995\ (x - 15)^2\ ,$$
$$x < 17.5000000000000000$$
$$39292.59428 - 1578.62971399999992\ x - 910.420322800000008\ (x - 20)^2\ ,$$
$$x < 21.2500000000000000$$
$$57740.20783 - 2552.00923700000022\ x + 521.068514100000016\ (x - 22.5)^2\ ,$$
$$x < 26.2500000000000000$$
$$620. - 180.800615899999997\ (x - 30)^2\ ,\ otherwise$$

```
> S_quadratic:=int(quadratic_spline,x=a..b);
```

$$S\_quadratic := 3993.733503$$

```
> quadratic_spline:=x->spline(X,Y,x,quadratic):
  plot([XY,quadratic_spline],X[1]..X[n],style=[POINT,LINE],s
  ymbol=CIRCLE,symbolsize=15,thickness=2,title="Quadratic
  Spline Interpolation",color=[BLACK,BLUE],legend=["Center
```

```
of Holes","Quadratic Spline"]);
```




```
>
>
```

Cubic Spline

```
> cubic_spline:=spline(X,Y,x,cubic);
```

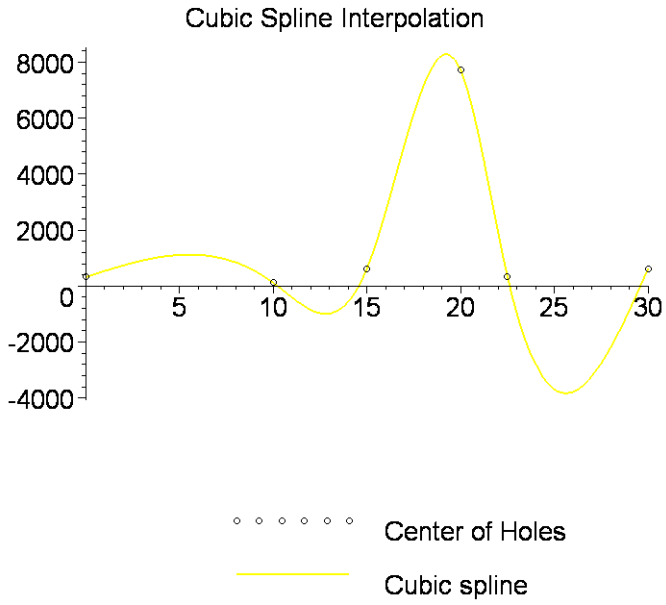
$$cubic\_spline := \{\ 320. + 218.538071099999996\ x + 0.818545231595635434\ 10^{-12}\ x^2 - 2.38538071099999982\ x^3\ ,\ x < 10$$
$$5090.761420 - 497.076142000000004\ x - 71.5614213197972334\ (x-10)^2 + 38.1953299299999998\ (x-10)^3\ ,\ x < 15$$
$$-24159.39085 + 1651.95938999999998\ x + 501.368527918781751\ (x-15)^2 - 109.552081200000002\ (x-15)^3\ ,\ x < 20$$
$$38735.22844 - 1550.76142200000004\ x - 1141.91269035532992\ (x-20)^2 + 231.286903499999994\ (x-20)^3\ ,\ x < 22.5$$
$$66103.14724 - 2923.69543299999987\ x + 592.739086294416211\ (x-22.5)^2 - 26.3439593800000012\ (x-22.5)^3\ ,\ otherwise$$

```
> S_cubic:=int(cubic_spline,x=a..b);
```

$$S\_cubic := 7088.916303$$

```
> cubic_spline:=x->spline(X,Y,x,cubic):
  plot([XY,cubic_spline],X[1]..X[n],style=[POINT,LINE],symbo
```

```
l=CIRCLE,symbolsize=15,thickness=2,title="Cubic Spline
Interpolation",color=[BLACK,YELLOW],legend=["Center of
Holes","Cubic spline"]);
```



Section V: Conclusion

```
> with( Spread ):
> EvaluateSpreadsheet(Discrete Function):
```

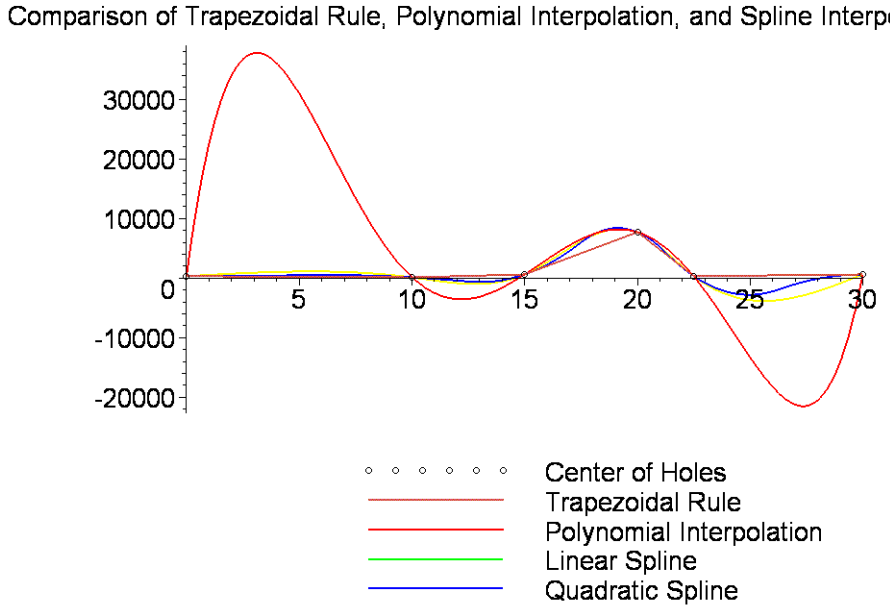
	A	B	C
1	*Method of Integration*	*Trapezoidal Rule*	*Polynomial Inte*
2	*Average Value*	38443.1000	151251.7

```
> with (plots);
```

```
Warning, the name changecoords has been redefined
```

[*Interactive, animate, animate3d, animatecurve, arrow, changecoords, complexplot, complexplot3d, conformal, conformal3d, contourplot, contourplot3d, coordplot, coordplot3d, cylinderplot, densityplot, display, display3d, fieldplot, fieldplot3d, gradplot, gradplot3d, graphplot3d, implicitplot, implicitplot3d, inequal, interactive, interactiveparams, listcontplot, listcontplot3d, listdensityplot, listplot, listplot3d, loglogplot, logplot, matrixplot, multiple, odeplot, pareto, plotcompare, pointplot, pointplot3d, polarplot, polygonplot, polygonplot3d, polyhedra_supported, polyhedraplot, replot, rootlocus, semilogplot, setoptions, setoptions3d, spacecurve, sparsematrixplot, sphereplot, surfdata, textplot, textplot3d, tubeplot*]

```
> trap:=x->spline(X,Y,x,linear):
  poly_inter:=x->interp(X,Y,x):
  linear_spline:=x->spline(X,Y,x,linear):
  quadratic_spline:=x->spline(X,Y,x,quadratic):
  cubic_spline:=x->spline(X,Y,x,cubic):
  plot([XY,trap,poly_inter,linear_spline,quadratic_spline,cubic_s
  pline],X[1]..X[n],style=[POINT,LINE,LINE,LINE,LINE,LINE],symbol
  =CIRCLE,symbolsize=15,thickness=2,title="Comparison of
  Trapezoidal Rule, Polynomial Interpolation, and Spline
  Interpolation",color=[BLACK,ORANGE,RED,GREEN,BLUE,YELLOW],legen
  d=["Center of Holes","Trapezoidal Rule","Polynomial
  Interpolation","Linear Spline","Quadratic Spline","Cubic
  spline"]);
```




```
> 
> 
```