Subject : The following demonstrates the integration of a discrete function using different methods.
Authors : Nathan Collier, Autar Kaw, Loubna Guennoun
Date : 9 November 2005

In[1]:=
```
ClearAll;
```

■ 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 2 types of interpolation. For more information on interpolation see the topic. The following outlines this sheet:

1. Trapezoidal Rule
2. Polynomial Interpolation
3. Cubic Spline Interpolation

See the following links for notes and presentations. [click here for textbook notes] [click here for power point presentation].

■ Inputs

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 these data, the program will calculate the average value of the integral using the trapezoidal rule, the polynomial interpolation, and the spline interpolation.

Array of x data

In[1]:=
```
x = {0.0001, 10, 15, 20, 22.5, 30};
```

> Note: If you have a zero in the x matrix, the simulation will not work. Try making the zero a near zero.

Array of y data

In[2]:=
```
y = {320, 120, 620, 720, 320, 620};
```

Lower limit of the integral

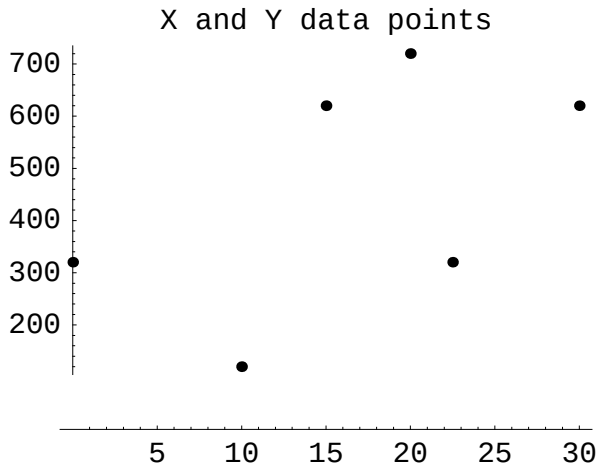
In[3]:=
```
a = 0.1;
```

Upper limit of the integral

In[4]:=
```
b = 30;
```

Setup Plot of Data Points

```
In[5]:= n := Abs[Dimensions[x]][[1]]
        xy = Table[0, {i, n}, {j, 2}];
        Do[xy[[i, 1]] = x[[i]]; xy[[i, 2]] = y[[i]], {i, 1, n}]
        data = ListPlot[xy, PlotStyle → {RGBColor[0, 0, 0], PointSize[0.02]},
           PlotLabel -> "X and Y data points",
           TextStyle → {FontSize → 11}, AxesOrigin → {0, 0}];
```



■ Data Checks

The data set X and Y and limits a and b must have certain characteristics. The following checks that certain criteria are true.

Are the limits a and b in the data range of X? If check0 returns YES, then you may proceed. If it reports NO then you need to redefine either or both a and b so that they are within the data range of X.

```
In[9]:= If[a ≥ Min[x] && b ≤ Max[x], "Yes", "No"]
Out[9]= Yes
```

Are the values of X in increasing order? If check1 returns YES, then you may proceed. If it reports NO, then you need to resort X and Y so that X is in ascending order.

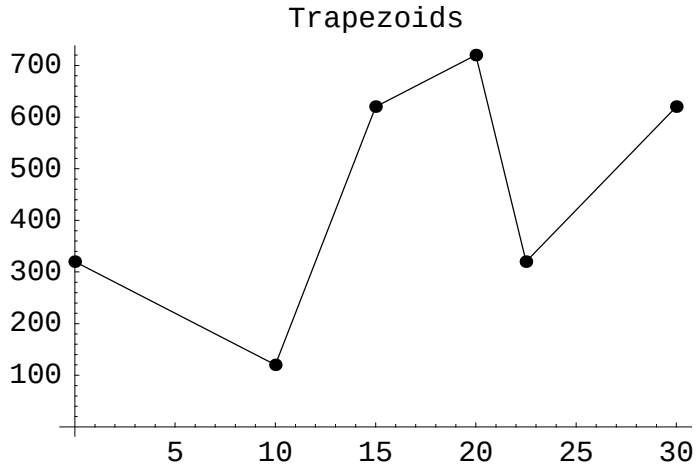
```
In[10]:= xx = Sort[x]; If[x == xx, "Yes", "No"]
Out[10]= Yes
```

■ Trapezoidal Rule

In[11]:=
```
n = Dimensions[x]; n = n[[1]];
AVtrap = 0;
Do[If[a > x[[i]], ain = i]; If[b > x[[i]], bin = i], {i, 1, n}];
ya =
  (a - x[[ain]]) / (x[[ain + 1]] - x[[ain]]) * (y[[ain + 1]] - y[[ain]]) + y[[ain]];
yb = (b - x[[bin]]) / (x[[bin + 1]] - x[[bin]]) *
    (y[[bin + 1]] - y[[bin]]) + y[[bin]];
AVtrap = (x[[ain + 1]] - a) * (ya + (y[[ain + 1]] - ya) / 2);
AVtrap = AVtrap + (-x[[bin]] + b) * (y[[bin]] + (-y[[bin]] + yb) / 2);
i = ain + 1;
While[i ≤ bin - 1,
 AVtrap = AVtrap + (x[[i + 1]] - x[[i]]) * (y[[i]] + (y[[i + 1]] - y[[i]]) / 2);
 i = i + 1]; AVtrap
```

Out[19]=
```
12193.1
```

In[25]:=
```
plot0 = ListPlot[xy, PlotStyle → {RGBColor[0, 0, 0]},
  PlotJoined → True , TextStyle → {FontSize → 11}, AxesOrigin → {0, 0}];
Show[data, plot0, PlotLabel -> "Trapezoids"];
```



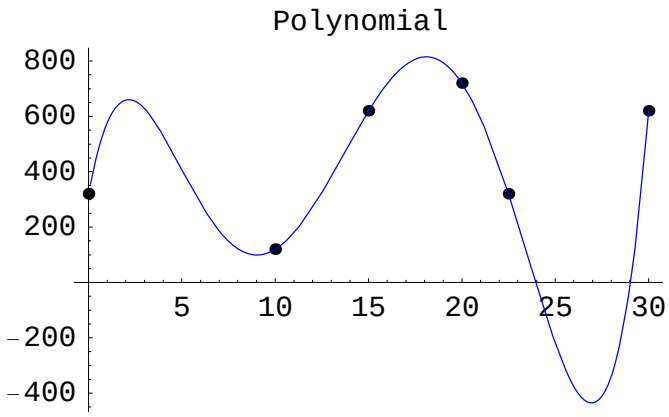
■ Polynomial Interpolation

Using polynomial, find a path that goes through the six data points

In[21]:=
```
M = Table[x[[i + 1]]^j, {i, 0, n - 1}, {j, 0, n - 1}];
A = LinearSolve[M, y];
fp[z_] := Sum[A[[i]] * z^(i - 1), {i, 1, n}]
AVpoly = Integrate[fp[z], {z, a, b}]
```

Out[24]=
```
9641.11
```

In[25]:=
```
plot1 = Plot[fp[z], {z, a, b}, PlotStyle → {RGBColor[0, 0, 1]}];
Show[data, plot1, PlotLabel -> "Polynomial "];
```



■ Cubic Spline Interpolation

Mathematica's Cubic Spline function works in a parametric fashion. The user specifies the relative position in the data set, and the function interpolates based on that. For more information, see the help file. For this reason, we must write our own cubic spline process. The following was adapted from an algorithm found in Burden's *Numerical Analysis*, 4th ed. page 131.

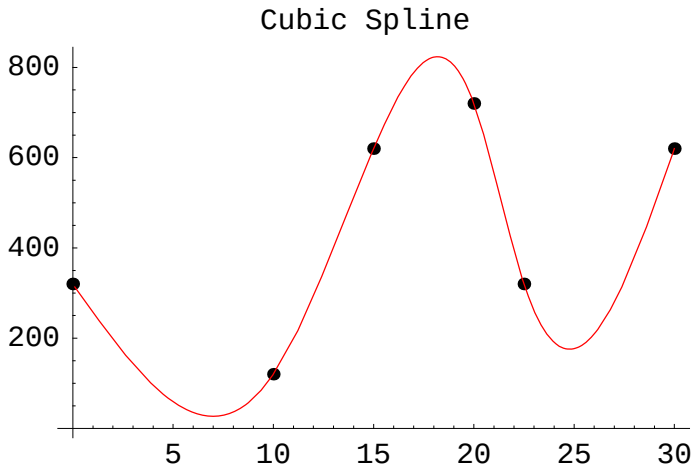
Due to the fact that the module uses a and b as matrices, I will reassign the limits of integration here as a1 and b1 to avoid an error.

In[26]:=
```
a1 = a; b1 = b;
```

In[27]:=
```
CubicSplineCoeff[X0_, Y0_] := Module[{X = X0, Y = Y0},
   GettingCoeff := Module[{},
     n := Length[X] - 1;
     a = b = d = Alpha = h = Table[0, {n}];
     l = u = z = c = Table[0, {n + 1}];
     Do[a[[i]] = Y[[i]], {i, 1, n}];
     Do[h[[i]] = X[[i + 1]] - X[[i]], {i, 1, n}];
     Do[Alpha[[i]] = 3 * (Y[[i + 1]] * h[[i - 1]] - Y[[i]] * (X[[i + 1]] - X[[i - 1]]) +
           Y[[i - 1]] * h[[i]]) / (h[[i - 1]] * h[[i]]), {i, 2, n}];
     l[[1]] := 1;
     u[[1]] := 0;
     z[[1]] := 0;
     Do[
      l[[i]] = 2 * (X[[i + 1]] - X[[i - 1]]) - h[[i - 1]] * u[[i - 1]];
      u[[i]] = h[[i]] / l[[i]];
      z[[i]] = (Alpha[[i]] - h[[i - 1]] * z[[i - 1]]) / l[[i]],
      {i, 2, n}];
     l[[n + 1]] := 1;
     z[[n + 1]] := 0;
     c[[n + 1]] := 0;
     j = n + 1; While[(j = j - 1) ≥ 1,
      c[[j]] = z[[j]] - u[[j]] * c[[j + 1]];
      b[[j]] = (Y[[j + 1]] - Y[[j]]) / h[[j]] - h[[j]] * (c[[j + 1]] + 2 * c[[j]]) / 3;
      d[[j]] = (c[[j + 1]] - c[[j]]) / (3 * h[[j]]);];];
   SplineFunction[t_] := Module[{i},
     For[i = 1, i ≤ n, i++,
      If[X[[i]] ≤ t && t < X[[i + 1]], k = i];];
     If[t < X[[1]], k = 1];
     If[t > X[[n + 1]], k = n];
     q = t - X[[k]];
     Return[((d[[k]] * q + c[[k]]) * q + b[[k]]) * q + a[[k]]];];
   GettingCoeff;];
```

In[28]:=
```
CubicSplineCoeff[x, y]
```

In[42]:=
```
plot2 = Plot[SplineFunction[z], {z, a1, b1}, PlotStyle → {RGBColor[1, 0, 0]}];
Show[data, plot2, PlotLabel -> "Cubic Spline "];
```



In[30]:=
```
Simpson38[x1_, x2_] :=
  (x2 - x1) / 8 * (SplineFunction[x1] + 3 * SplineFunction[x1 + (x2 - x1) / 3] +
     3 * SplineFunction[x1 + 2 * (x2 - x1) / 3] + SplineFunction[x2]);
```

In[31]:=
```
AVspline = 0;
Do[If[a1 > x[[i]], ain = i]; If[b1 > x[[i]], bin = i], {i, 1, n}];
AVspline = Simpson38[a1, x[[ain + 1]]];
AVspline = AVspline + Simpson38[x[[bin]], b1];;
i = ain + 1;
While[i ≤ bin - 1, AVspline = AVspline + Simpson38[x[[i]], x[[i + 1]]]; i = i + 1];
AVspline
```

Out[36]= 10288.8

■ Comparison

Here the values are repeated for comparion. Why might there be such a great difference?

In[37]:=
```
AVtrap
AVpoly
AVspline
```

Out[37]= 12193.1

Out[38]= 9641.11

Out[39]= 10288.8