Tutorial I Report

Shagun Agarwal
NA12B028

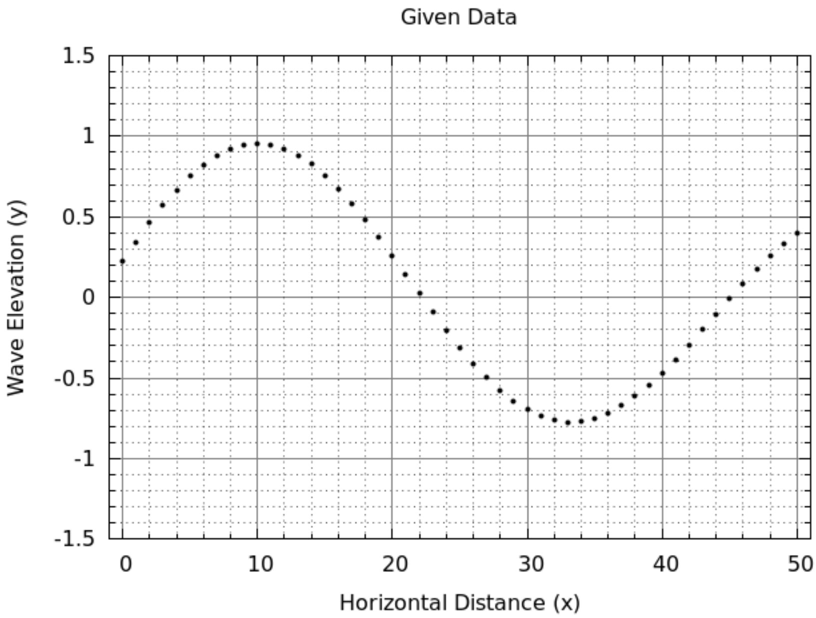
12 February 2015

Question

Free surface levels of a wave is given in the attached file with respect to horizontal distance. Obtain:

1. Water surface plot using, Lagrange's polynomial and Cubic spline.
2. How do you numerically estimate the free surface slope at any x? Compare the values using above methods at x=26.5m.

Given Data



Lagrange's Polynomial Interpolation

The Lagrange's polynomial method provides an interpolation solution for the given N points (x_1, y_1),(x_2, y_2)...(x_N, y_N) through a polynomial of degree $N-1$ which is given by:

$$P(x) = \frac{(x-x_2)(x-x_3)...(x-x_N)}{(x_1-x_2)(x_1-x_3)....(x_1-x_N)}y_1 + \frac{(x-x_1)(x-x_3)...(x-x_N)}{(x_2-x_1)(x_2-x_3)...(x_2-x_N)}y_2 + ...$$

$$... + \frac{(x-x_1)(x-x_2)...(x-x_{N-1})}{(x_N-x_1)(x_N-x_2)...(x_N-x_{N-1})}y_N$$

Slope at a given point in the given range can be obtained by differentiating this expression.

$$P'(x) = \sum_{i=1}^{N} \frac{Numer(i)}{Denom(i)} y_i$$

Where

$$Denom(i) = \prod_{\substack{j=1 \\ j \neq i}}^{N} (x_i - x_j)$$

And

$$Numer(i) = \sum_{\substack{j=1 \\ j \neq i}}^{N} \Big(\prod_{\substack{k=1 \\ k \neq i \\ k \neq j}}^{N} (x - x_k) \Big)$$

The solution will be continuous at all points but may not be differentiable at all points. For obtaining a solution with continuous first and second derivative we use the Cubic Interpolation method.

Cubic Spline Interpolation

This method provides us with a solution for interpolation which has continuous first and second derivative. The solution between any intervals x_k and x_{k+1} is given by

$$y(x) = Ay_k + By_{k+1} + Cy''_k + Dy''_{k+1}$$

Where

$$A = \frac{x_2 - x}{x_2 - x_1}$$

$$B = 1 - A$$

$$C = \frac{(A^3 - A)(x_{k+1} - x_k)^2}{6}$$

$$D = \frac{(B^3 - B)(x_{k+1} - x_k)^2}{6}$$

The values of y''_i are obtained by equating the first derivative at each point. The first derivative at any point is given by:

$$y'(x) = \frac{y_{k+1} - y_k}{x_{k+1} - x_k} - \frac{(3A^2 - 1)(x_{k+1} - x_k)y''_k}{6} + \frac{(3B^2 - 1)(x_{k+1} - x_k)y''_{k+1}}{6}$$

This gives us $N-2$ equations of form:

$$\frac{(x_k - x_{k-1})y''_{k-1}}{6} + \frac{(x_{k+1} - x_{k-1})y''_k}{3} + \frac{(x_{k+1} - x_k)y''_{k+1}}{6} = \frac{y_{k+1} - y_k}{x_{k+1} - x_k} - \frac{y_k - y_{k-1}}{x_k - x_{k-1}}$$

Hence if we know the values of y''_1 and y''_N, we can determine the value for all y''_k; $k = 2...(N-1)$ thus forming a 2 parameter problem. We can also observe that the second term in each equation is having higher magnitude than the first and second terms. Hence we can solve this set of equations using the method of solving a Tri-diagonal Matrix using Forward Sweep and Backward Substitution.

We set the values $y''_1 = y''_N = 0$, which will give us the natural spline result.

Code

The following code executes the theory mentioned above for both Largrange's Interpolation method and Cubic Spline (Tridiagonal) Method.

It provides the value of wave elevation at an interval of 0.1 units. The final results are plotted on a graph for each case.

Also the values of second derivative are calculated for x=26.5 and have been mentioned in the output.

```

#define use_stdout 1
#include <iostream>
#include <fstream>
#include <curses.h>
#include <string>
#include <stdlib.h>
#include <math.h>
#include <time.h>

using namespace std;

int input_data(double [][2], int);
int lagrangian_method(double [][2], int, double [][2], const int
    ,const float);
int tridiagonal_method(double [][2], int, double [][2], const
    int,const float);
```

```
void plotting(string, string);

int main()
{
  system("clear");

  //To write screen output to a file
  if(!use_stdout)
    freopen("Screen Output.txt","w+",stdout);

  const int num_ele=51;
  const float step=0.1;
  const int size=50/step+1;
  double data[num_ele][2];
  double result[size][2];

  printf("Start!\n");

  //Inputting data from the given file.
  int err=input_data(data,num_ele);
  if(!err)
  {
    printf("Data Input Successful\n");

    //Function to provide result for Lagrangian Method
    err=lagrangian_method(data,num_ele,result,size,step);
    if(!err)
    {
      printf("Lagrangian Method Implementation Successful\n");
      //Function to plot the result of Lagrangian Method
      plotting("Lagrangian_Output","Lagrangian Interpolation Result");
      printf("Lagrangian Output Plotted\n");
    }

    //Refeshing the result matrix
    for(int i=0;i<size;i++)
    {
      result[i][0]=0;
      result[i][1]=0;
    }

    //Fuction to provide the result for Tridiagonal Method
    err=tridiagonal_method(data,num_ele,result,size,step);
    if(!err)
    {
      printf("Tridiagonal Method Implementation Successful\n");
      //Function to plot the result of Tridiagonal Method
      plotting("Tridiagonal_Output","Tridiagonal Interpolation
```

```
          Result");
      printf("Tridiagonal Output Plotted\n");
    }
  }

  printf("End!\n");

  if(!use_stdout)
    fclose(stdout);

  return 0;
}

int input_data(double data[][2], int num_ele)
{
  fstream infile("data.dat",ios::in);
  if (infile.is_open())
  {
    infile.seekg(0);
    char a[50];
    char b[50];
    int i=0;
    while(!infile.eof())
    {
      infile.getline(a,30,'\t');
      infile.getline(b,30,'\n');
      data[i][0]=atoi(a);
      data[i][1]=atof(b);
      //printf("%d\t%f\t%.10f\n",i,data[i][1],data[i][2]);

      i++;
      if(i==num_ele)
        break;
    }
    infile.close();
    return 0;
  }
  else
  {
    printf("Unable to open the file!\n");
    return 1;
  }

}

int lagrangian_method(double data[][2], int num_ele, double result[][2], const int size, const float step)
{
  double denom[num_ele];
```

```

//Calculating Denominator Terms First
for(int i=0;i<num_ele;i++)
  denom[i]=1;

for(int i=0;i<num_ele;i++)
{
  for(int j=0;j<i;j++)
    denom[i]=denom[i]*(data[i][0]-data[j][0]);
  for(int j=i+1;j<num_ele;j++)
    denom[i]=denom[i]*(data[i][0]-data[j][0]);
}

FILE* otfile1=fopen("Denom.temp","w");
for(int i=0;i<num_ele;i++)
  fprintf(otfile1,"%f\n",denom[i]);
fclose(otfile1);

//Calculating results
for(int k=0;k<size;k++)
{
  double y=0;
  double x=data[num_ele-1][0]*k/(size-1);
  for(int i=0;i<num_ele;i++)
  {
    double numer=1;
    for(int j=0;j<i;j++)
      numer=numer*(x-data[j][0]);
    for(int j=i+1;j<num_ele;j++)
      numer=numer*(x-data[j][0]);

    y=y+(numer*data[i][1]/denom[i]);
  }
  result[k][0]=x;
  result[k][1]=y;
}

FILE* otfile2=fopen("Lagrangian_Output.temp","w");
for(int i=0;i<size;i++)
  fprintf(otfile2,"%f\t%.10f\n",result[i][0],result[i][1]);
fclose(otfile2);

//Calculating Slope
double x=26.5;
double yp=0;
for(int i=0;i<num_ele;i++)
{
  double numer=0;
  for(int j=0;j<num_ele;j++)
```

```
    {
      double num_term=1;
      if(j!=i)
        for(int k=0;k<num_ele;k++)
          if((k!=i) && (k!=j))
            num_term*=x-data[k][0];
      numer+=num_term;
    }
    yp+=data[i][1]*numer/denom[i];
  }
  printf("As per Lagrangian Method:\nSlope at x=26.5 is %.10f\n",yp);
  return 0;

}

int tridiagonal_method(double data[][2], int num_ele, double result[][2], const int size, const float step)
{
  double coef[num_ele][4], ypp[num_ele];

  //Calculating the coefficient matrix for Ax=B
  for(int i=1;i<num_ele-1;i++)
  {
    coef[i][0]=(data[i][0]-data[i-1][0])/6;
    coef[i][1]=(data[i+1][0]-data[i-1][0])/3;
    coef[i][2]=(data[i+1][0]-data[i][0])/6;
    coef[i][3]=((data[i+1][1]-data[i][1])/(data[i+1][0]-data[i][0]))-((data[i][1]-data[i-1][1])/(data[i][0]-data[i-1][0]));
  }
  coef[1][0]=coef[num_ele-2][2]=0;

  //Updating the coefficients using Forward Sweep
  coef[1][2]=coef[1][2]/coef[1][1];
  coef[1][3]=coef[1][3]/coef[1][1];
  for(int i=2;i<num_ele-2;i++)
  {
    coef[i][2]=(coef[i][2])/(coef[i][1]-(coef[i][0]*coef[i-1][2]));
    coef[i][3]=(coef[i][3]-(coef[i][0]*coef[i-1][3]))/(coef[i][1]-(coef[i][0]*coef[i-1][2]));
  }
  int i=num_ele-2;
  coef[i][3]=(coef[i][3]-(coef[i][0]*coef[i-1][3]))/(coef[i][1]-(coef[i][0]*coef[i-1][2]));

  //Calculating ypp by Backward Substitution
  ypp[0]=ypp[num_ele-1]=0;
```

```
ypp[num_ele-2]=coef[num_ele-2][3];
for(int i=num_ele-3;i>0;i--)
  ypp[i]=coef[i][3]-(coef[i][2]*ypp[i+1]);

//Calculating Results
int srch_st=0,index=0;
for(int k=0;k<size;k++)
{
  //Search the interval in which the current x lies
  double x=data[num_ele-1][0]*k/(size-1);
  for(i=srch_st;i<num_ele;i++)
    if(data[i][0]>x)
      break;
  index=i-1;
  if(index>srch_st)
    srch_st=index;

  //Calculating coefficient terms
  double a=(x-data[index+1][0])/(data[index][0]-data[index
      +1][0]);
  double b=1-a;
  double c=(pow(a,3.00)-a)*(pow((data[index+1][0]-data[index
      ][0]),2))/6;
  double d=(pow(b,3.00)-b)*(pow((data[index+1][0]-data[index
      ][0]),2))/6;

  //Calculating results
  double y=(a*data[index][1])+(b*data[index+1][1])+(c*ypp[
      index])+(d*ypp[index+1]);
  result[k][0]=x;
  result[k][1]=y;
}

FILE* otfile3=fopen("ypp.temp","w");
for(int i=0;i<num_ele;i++)
  fprintf(otfile3, "%f\n",ypp[i]);
fclose(otfile3);

otfile3=fopen("Tridiagonal_Output.temp","w");
for(int i=0;i<size;i++)
  fprintf(otfile3, "%f\t%.10f\n",result[i][0],result[i][1]);
fclose(otfile3);

//Calculating Slope at x=26.5
double x=26.5;
for(i=0;i<num_ele;i++)
  if(data[i][0]>x)
    break;
index=i-1;
```

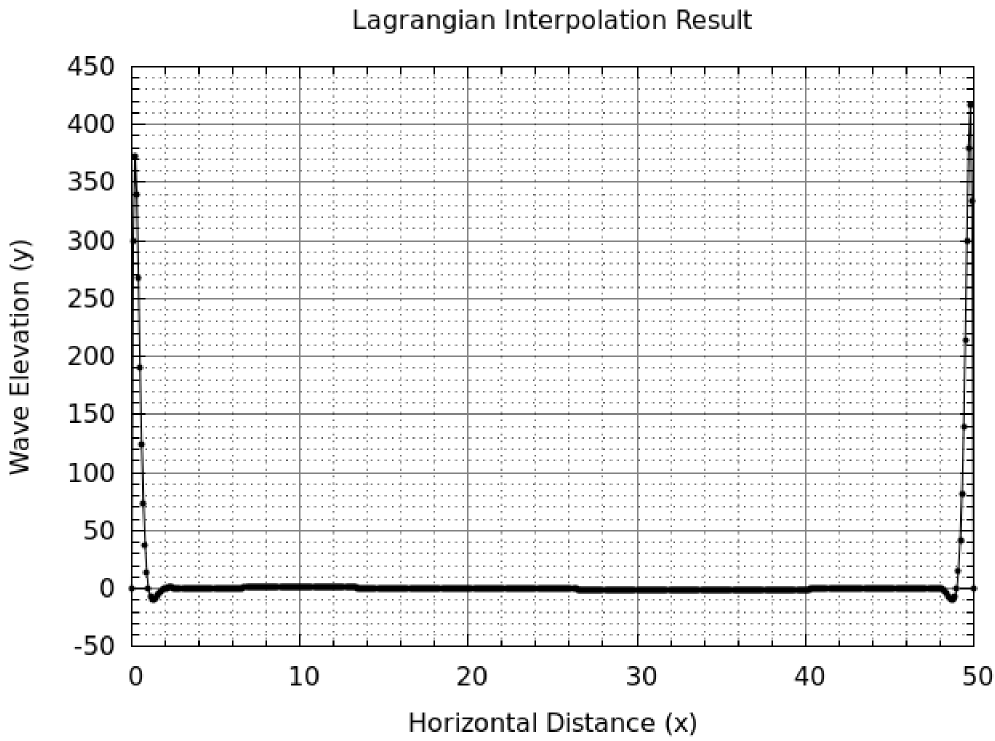
```
  double a=(x-data[index+1][0])/(data[index][0]-data[index+1][0]);
  double b=1-a;
  double c=(pow(a,3.00)-a)*(pow((data[index+1][0]-data[index][0]),2))/6;
  double d=(pow(b,3.00)-b)*(pow((data[index+1][0]-data[index][0]),2))/6;
  double yp=(data[index+1][1]-data[index][1])/(data[index+1][0]-data[index][0]);
  yp-=(3*pow(a,2)-1)*(data[index+1][0]-data[index][0])*ypp[index]/6;
  yp+=(3*pow(b,2)-1)*(data[index+1][0]-data[index][0])*ypp[index+1]/6;
  printf("As per Tridiagonal Method:\nSlope at x=26.5 is %.10f\n",yp);

  return 0;
}

void plotting(string file_name, string title)
{
  //This function plots the results using gnuplot
  FILE * gnuplot = popen ("gnuplot -persistent", "w");
  fprintf(gnuplot, "set terminal pngcairo\n");
  fprintf(gnuplot, "set title '%s'\n",title.c_str());
  fprintf(gnuplot, "set xlabel 'Horizontal Distance (x)'\n");
  fprintf(gnuplot, "set ylabel 'Wave Elevation (y)'\n");
  fprintf(gnuplot, "set key off\n");
  fprintf(gnuplot, "set xrange [-1:51]\nset yrange [-1.5:1.5]\n");
  fprintf(gnuplot, "set mytics 5\nset mxtics 5\n");
  fprintf(gnuplot, "set grid ytics xtics mytics mxtics ls 9, ls 0\n");
  fprintf(gnuplot, "set output '%s.png'\n",file_name.c_str());
  fprintf(gnuplot, "plot '%s.temp' w linespoints ls -1 pt 7 ps 0.5\n",file_name.c_str());
}
```

Output

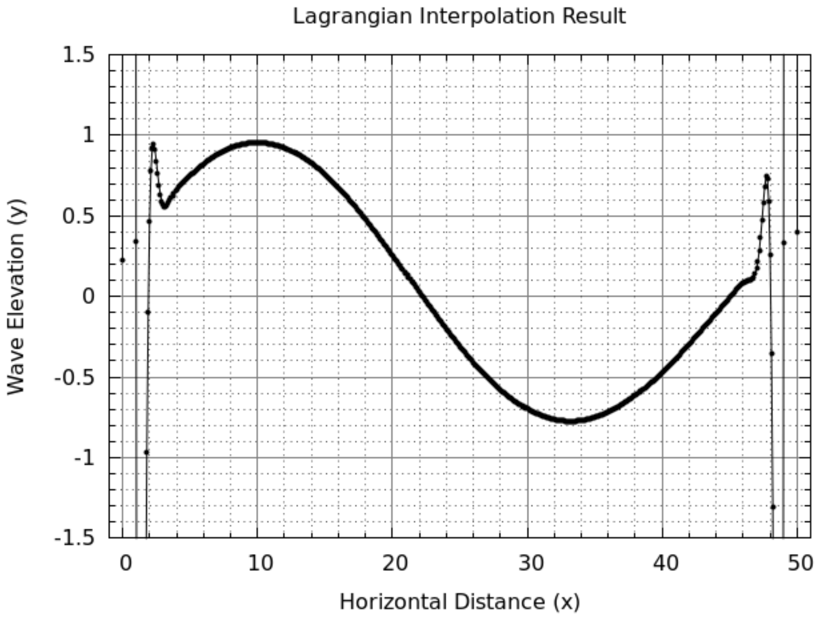
Screen Output

Start!
Data Input Successful
As per Lagrangian Method:
Slope at x=26.5 is -0.0898740734
Lagrangian Method Implementation Successful
Lagrangian Output Plotted
As per Tridiagonal Method:
Slope at x=26.5 is -0.0898741372
Tridiagonal Method Implementation Successful
Tridiagonal Output Plotted
End!

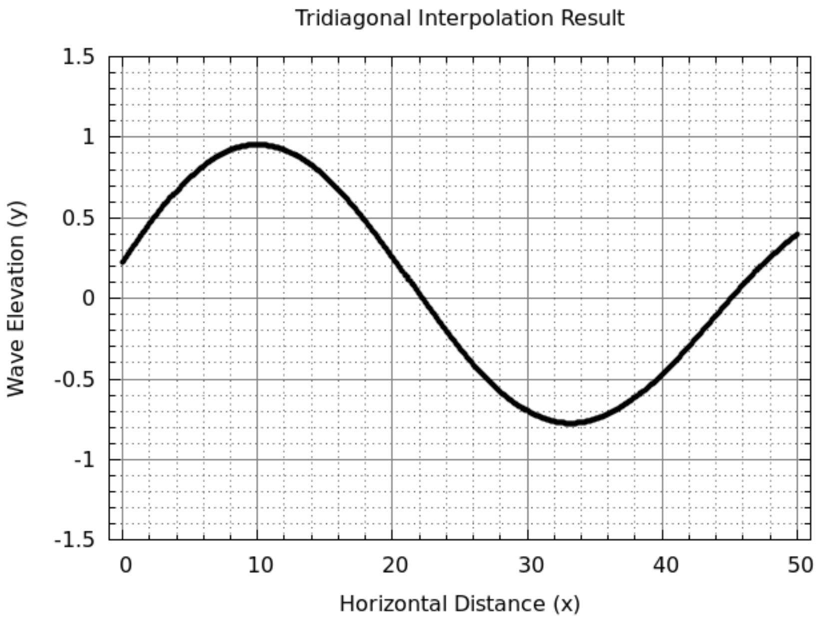
Lagrange's Interpolation Output



On zooming in by limiting the y range between (-1.5 and 1.5) we get



Cubic Spline Output



Observations

Screen Output

The screen output shows the values of first derivative calculated at x=26.5 using the two different methods.

As per Lagrangian Method:
Slope at x=26.5 is -0.0898740734
As per Cubic Spline Method:
Slope at x=26.5 is -0.0898741372

It is observed that the values obtained by both the methods are very close showing that in this region of the range both the methods provide a similar interpolation result.

Lagrange's Interpolation Output

It can be observed that the result oscillates wildly about the given data-points near the two corners and then stabalises to a resonable output away from the corners.
This divergence of the output from the true function is called Runge's Phenomenon and occurs because the function is a very high degree polynomial (50 in this case) and can get worse with increasing number of points.

Cubic Spline Output

The output of cubic spline method has a smooth curvature with no oscillation about the data points.
This has been ensured by making the first and second derivative of the objective function continuous thus giving a very smooth output even near the corner points.

Conclusion

It can be concluded that both Lagrange's Methos and Cubic Spline method of interpolation can provide a desirable inerpolation result but both have their pros and cons.

The Lagrange's method is easier to implement but one has to be careful with its use given the number of data-points. As the result is obtained using a high degree polynomial, large number of data-points will give a high degree polynomial which can cause the result to oscillate about the data-points atleast near the corners.

Cubic spline interpolation method provides a reasonably accurate solution in most case, with increasing accuracy as the number of points are increased; but it is a relatively tougher method to implement and will require use of computational tools for large numer of data-points.