**Newton's Forward Difference Formula**

Newton's forward difference formula is a [finite difference](http://mathworld.wolfram.com/FiniteDifference.html) identity giving an interpolated value between tabulated points {f_p}in terms of the first value f_0and the [powers](http://mathworld.wolfram.com/Power.html) of the [forward difference](http://mathworld.wolfram.com/ForwardDifference.html) Delta. For a in [0,1], the formula states

|  |  |
| --- | --- |
| f_a=f_0+aDelta+1/(2!)a(a-1)Delta^2+1/(3!)a(a-1)(a-2)Delta^3+.... | (1) |

When written in the form

|  |  |
| --- | --- |
| f(x+a)=sum_(n=0)^infty((a)_nDelta^nf(x))/(n!) | (2) |

with (a)_nthe [falling factorial](http://mathworld.wolfram.com/FallingFactorial.html), the formula looks suspiciously like a finite analog of a [Taylor series](http://mathworld.wolfram.com/TaylorSeries.html) expansion. This correspondence was one of the motivating forces for the development of [umbral calculus](http://mathworld.wolfram.com/UmbralCalculus.html).

An alternate form of this equation using binomial coefficients is

|  |  |
| --- | --- |
| f(x+a)=sum_(n=0)^infty(a; n)Delta^nf(x), | (3) |

where the [binomial coefficient](http://mathworld.wolfram.com/BinomialCoefficient.html) (a; n)represents a polynomial of degree nin a.

# Code :

#include<bits/stdc++.h>

using namespace std;

int fact(int n)

{

int i=0;

int sum=1;

for(i=2;i<=n;i++)

sum\*=i;

return sum;

}

int main()

{

freopen("input.txt","r",stdin);

double x[20]={0},y[20][20]={0},p,a,sum=0,arr[10],h,y0;

int i,j,n;

scanf("%d",&n);

for(i=0;i<n;i++){

scanf("%lf %lf",&x[i],&y[i][0]);

}

for(i=0;i<n;i++){

for(j=1;j<n-i; j++){

y[j-1][i+1]=y[j][i]-y[j-1][i];

}

}

printf("x- - - - - - - -y0- - - - - - -y1- - - - - - -y2- - - - - -y3- - - - - -y4\n");

for(i=0;i<n;i++){

printf("%.4lf\t",x[i]);

for(j=0;j<n-i; j++){

if(i==0)

{

arr[j]=y[i][j+1];

}

printf("%.4lf\t",y[i][j]);

}

printf("\n");

}

for(i=0;i<n-1;i++)

{

printf("%.4lf ",arr[i]);

}

h=x[1]-x[0];

y0=y[0][0];

printf("\n%.4lf ",h);

printf("\n%.4lf ",y0);

for(i=0;i<n-1;i++)

{

if(i%2==0)

sum=sum+(arr[i]/(i+1));

else

sum=sum+(arr[i]/(i+1)\*(-1));

}

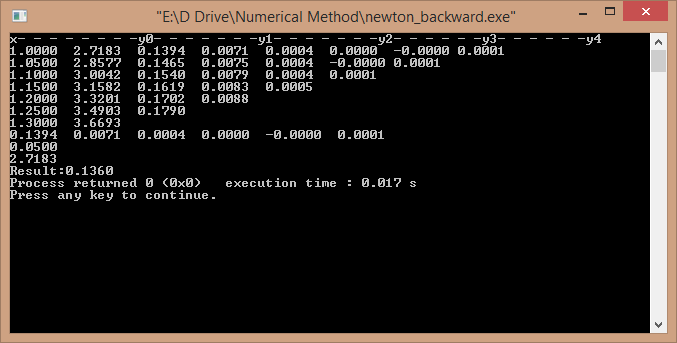
printf("\nResult:%.4lf ",sum);

/\*printf("Enter the number\n");

scanf("%lf",&a);\*/

return 0;

}



**Newton's Backward Difference Formula**

|  |
| --- |
| f_p=f_0+pdel _0+1/(2!)p(p+1)del _0^2+1/(3!)p(p+1)(p+2)del _0^3+..., |

where del is the [backward difference](http://mathworld.wolfram.com/BackwardDifference.html).

# Code :

#include <stdio.h>

main()

{

float a[10][10],x[10];

int i,j,fact=1,m;

float xm,p,psum,inter;

printf ("enter number of data points: ");

scanf ("%d", &m);

for (i=0; i<m; i++)

{

for (j=0; j<=m; j++)

{

a[i][j] = 0;

}

}

printf ("\nenter the data (x <space> y)\n");

for (i=0; i<m; i++)

scanf ("%f%f", &x[i],&a[i][0]);

for (j=1; j<m; j++)

{

for (i=1; i<m; i++)

{

if (j<=i)

a[i][j]=a[i][j-1]-a[i-1][j-1];

}

}

printf ("\nbackward difference table\n");

for (i=0; i<m; i++)

{

printf("%.3f", x[i]);

for (j=0; j<=i; j++)

{

printf ("\t%.3f", a[i][j]);

}

printf ("\n");

}

printf ("\nenter x: ");

scanf ("%f", &xm);

p=(xm-x[m-1])/(x[1]-x[0]);

inter=a[m-1][0]+(p\*a[m-1][1]);

printf ("\n");

psum=p;

for(i=2; i<m; i++)

{

psum=psum\*(p+i-1);

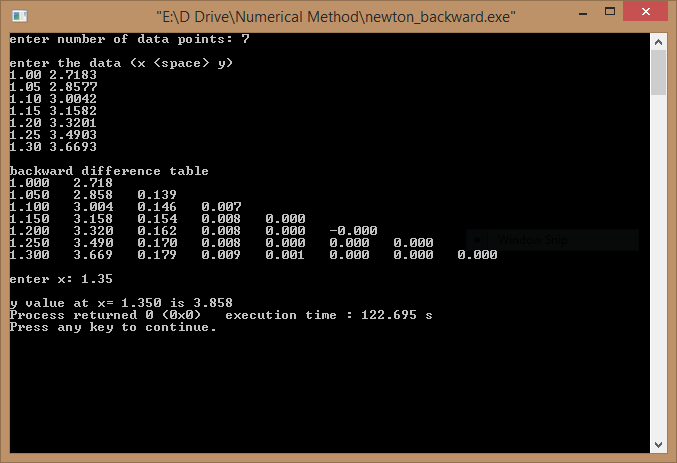
fact=fact\*i;

inter=inter+((psum\*a[m-1][i])/fact);

}

printf("y value at x= %.3f is %.3f", xm,inter);

}



# Observation :

We use Forward difference formula for the point 1.03 as the lower boundary 1.00 is near to the point 1.03 and backward difference formula for 1.35 as the upper boundary 1.30 is near to the point 1.35.

# Ramanujan's Theorem

Suppose that in some [neighborhood](http://mathworld.wolfram.com/Neighborhood.html) of x=0,

|  |  |
| --- | --- |
| F(x)=sum_(k=0)^infty(phi(k)(-x)^k)/(k!) | (1) |

for some function (say analytic or integrable) phi(k). Then

|  |  |
| --- | --- |
| int_0^inftyx^(n-1)F(x)dx=Gamma(n)phi(-n). | (2) |

These functions form a forward/inverse transform pair. For example, taking phi(k)=1for all kgives

|  |  |
| --- | --- |
| F(x)=sum_(k=0)^infty((-x)^k)/(k!)=e^(-x), | (3) |

and

|  |  |
| --- | --- |
| int_0^inftyx^(n-1)e^(-x)dx=Gamma(n), | (4) |

which is simply the usual integral formula for the [gamma function](http://mathworld.wolfram.com/GammaFunction.html).

Ramanujan used this theorem to generate amazing identities by substituting particular values for phi(n).

# Code :

#include<stdio.h>

#include<math.h>

#include<stdlib.h>

double a[100]={0},b[100]={0};

double bn( double a[],int n)

{

int i;

double sum=0;

for(i=1;i<n;i++){

sum=sum+a[i]\*b[n-i];

}

return sum;

}

int main()

{

int i;

double sum,pre;

printf("f(x)= x3 - 9x2 + 26x - 24\n");

a[1]=(26.0/24.0); a[2]= -3.0/8.0; a[3]= 1.0/24.0; b[1]=1;

for(i=2; ;i++){

b[i]=bn(a,i);

sum=b[i-1]/b[i];

printf("%d\t b[%d] / b[%d] = %lf\n",i-1,i-1,i,sum);

if(fabs(pre-sum)<=0.01)

break;

pre=sum;

}

printf("The smallest Root is: %lf\n",sum);

}

