Cubic Hermite Curves

$$P(t) = \begin{pmatrix} t^3 & t^2 & t & 1 \end{pmatrix} \begin{pmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} P_0 \\ P_1 \\ \mathbf{u}_0 \\ \mathbf{u}_1 \end{pmatrix}$$

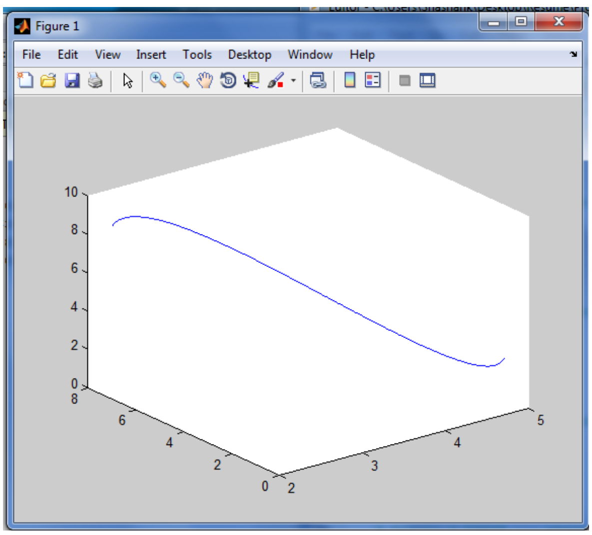
Where P_o and P_1- End Points

and u_0 and u_1- End Tangents

```
% Hermite Cubic Curve
% M= square Hermite matrix
M=[2 -2 1 1
    -3 3 -2 -1
    0 0 1 0
    1 0 0 0];

% Result = U*M*B ; where U-Paremetric matrix, B-Geometric Coff. Matrix
% Relation b/w Algebric and Geometric Coff.
% A = M*B  ; where A-Algebric Coff. Matrix

% Generates U(Parameter matrix) b/t parameter 0 to 1
U=[];
for u=0:.001:1
    U =[U
        u^3 u^2 u 1];
end

% Input B matrix for x,y,z coordinate.
xl=xlsread('Geometric Coff', -1);
B=xl;

R=U*M*B;

line(R(:,1),R(:,2),R(:,3));
view(3);

% Conversion to 4 point form P=Param*M*B where Param-default parameters
Param=[0 0 0 1
   .33^3 .33^2 .33 1
   .66^3 .66^2 .66 1
   1 1 1 1];
P=Param*M*B
```

	X COORD	Y COORD	Z COORD
PT1	2	7	9
PT2	5	1	2
TANGENT1	3	5	3
TANGENT2	2	3	4

INPUT



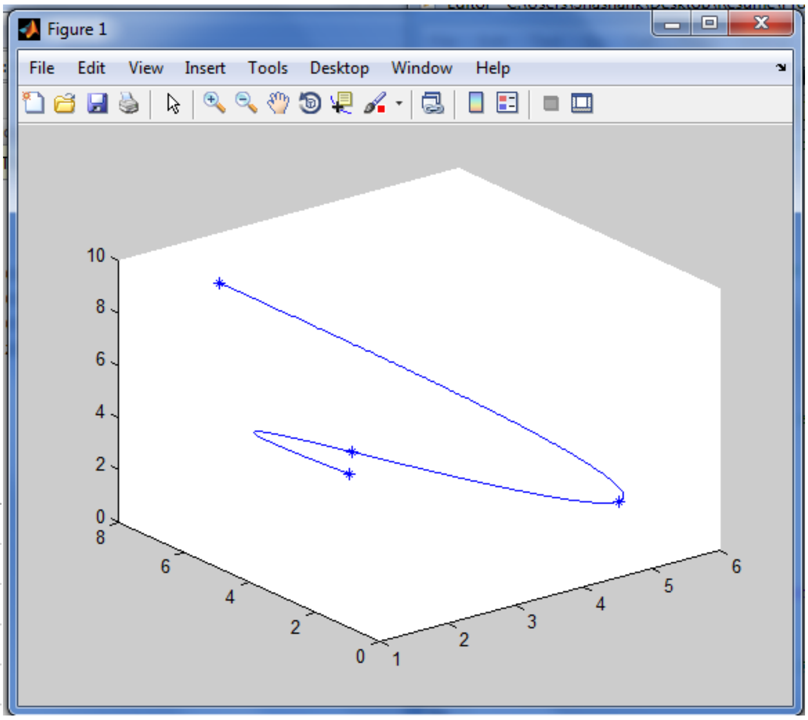
OUTPUT

Cubic Curve 4 Point Form

We obtain Geometric Coefficient Matrix by satisfying the 4 Points a different parameters (Default- 0, 0.33, 0.66, 1) and inverting the equation.

```
%Cubic Segments 4 Point Form
% M= square Hermite matrix
M=[2 -2 1 1
    -3 3 -2 -1
    0 0 1 0
    1 0 0 0];

% Param= Based on Default equidistant parameters
Param=[0 0 0 1
   .33^3 .33^2 .33 1
   .66^3 .66^2 .66 1
   1 1 1 1];

% Input of Points Matrix
xl=xlsread('Four Point Form', -1);
P=xl;

% Conversion to Geometric Form--- B=inv(M)*inv(Param)*P
B=inv(M)*inv(Param)*P

% Displaying the Curve
U=[];
for u=0:.001:1
    U =[U
        u^3 u^2 u 1];
end
R=U*M*B;
line(R(:,1),R(:,2),R(:,3));
view(3);
hold on;
plot3(P(:,1),P(:,2),P(:,3),'LineStyle','none','Marker','*');
```

	X-Coord	Y-Coord	Z-Coord
PT 1	2	7	9
PT 2	5	1	2
PT 3	3	5	3
PT 4	2	3	4



INPUT OUTPUT

Bezier Curves

$$P(t) = \sum_{i=0}^{n} B_i \; J_{n,i}(t) \qquad 0 \le t \le 1$$

Where Bezier or Bernstein Basis or Blending Function is

$$J_{n,i}(t) = \binom{n}{i} t^i (1-t)^{n-i}$$

$$\binom{n}{i} = \frac{n!}{i!\,(n-i)!}$$

Generalised Matrix Representation

$$P(t) = [\,T\,][\,N\,][\,G\,]$$

where here

$$[\,T\,] = [\,t^n \quad t^{n-1} \quad \cdots \quad t \quad 1\,]$$

$$[\,N\,] = \begin{bmatrix} \binom{n}{0}\binom{n}{n}(-1)^n & \binom{n}{1}\binom{n-1}{n-1}(-1)^{n-1} & \cdots & \binom{n}{n}\binom{n-n}{n-n}(-1)^0 \\ \binom{n}{0}\binom{n}{n-1}(-1)^{n-1} & \binom{n}{1}\binom{n-1}{n-2}(-1)^{n-2} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ \binom{n}{0}\binom{n}{1}(-1)^1 & \binom{n}{1}\binom{n-1}{0}(-1)^0 & \cdots & 0 \\ \binom{n}{0}\binom{n}{0}(-1)^0 & 0 & \cdots & 0 \end{bmatrix}$$

$$(5-70)$$

$[\,G\,]^T$ is again $[\,B_0 \quad B_1 \quad \cdots \quad B_n\,]$. The individual terms in $[\,N\,]$ are given by

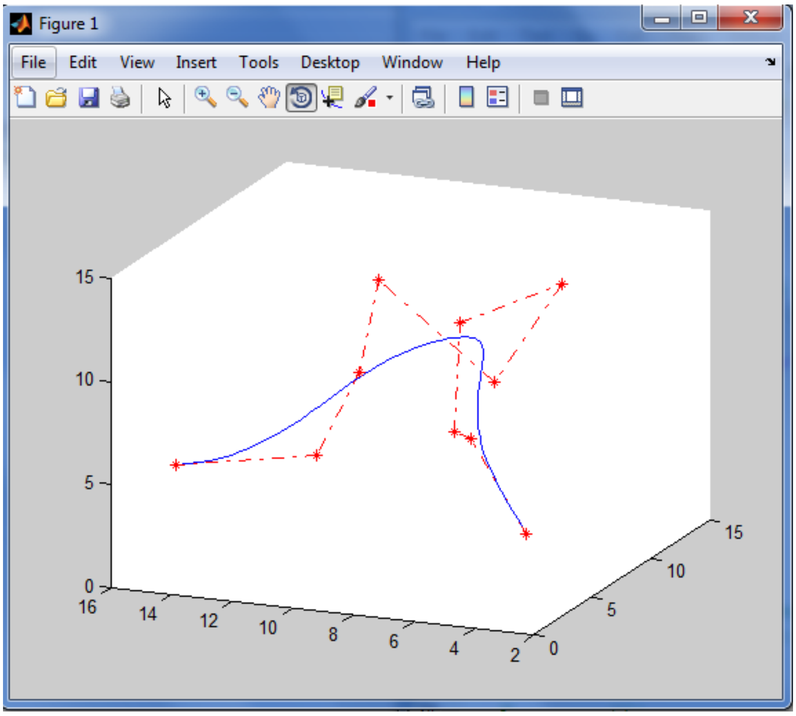
$$(N_{i+1,j+1})_{i,j=0}^{n} = \begin{cases} \binom{n}{j}\binom{n-j}{n-i-j}(-1)^{n-i-j} & 0 \le i+j \le n \\ 0 & \text{otherwise} \end{cases}$$

```
%Bezier Curve
% P-Control Point Matrix
% B_i-Bernstein Polonomial Function Value for i_th term
% BEZ= SUMISSION 0 to Deg (B_i*P_i)

xl=xlsread('Control Points', -1);
G=xl;
[Deg,dump]=size(xl);
Deg=Deg-1;
P=[];

% ONE WAY TO DO IT.
% for u=0:.001:1
%     t=[0 0 0];
%     for i=0:1:Deg
%         B_i=factorial(Deg)/(factorial(Deg-i)*factorial(i))*u^i*(1-u)^(Deg-i);
%         t=t+B_i*P(i+1,:);
%     end
%     Bez=[Bez
%         t];
% end
```

```
% BETTER WAY
% P= T*N*G where T= parameter matrix (t^3, t^2, t, 1)
% G= Points Matrix
% N= General Bezier Basis matrix

T=[];
for i=0:.01:1
    xyz=[];
    for j=Deg:-1:0
        xyz=cat(2,xyz,i^j);
    end
    T=cat(1,T,xyz);
end

N=[];
for i=0:1:Deg
    for j=0:1:Deg
        if (i+j)>=0 && (i+j)<=Deg
            N(i+1,j+1)=nchoosek(Deg,j)*nchoosek(Deg-j,Deg-i-j)*(-1)^(Deg-i-j);
        else
            N(i+1,j+1)=0;
        end
    end
end

P= T*N*G;

line(P(:,1),P(:,2),P(:,3));
view(3);
hold on;
plot3(G(:,1),G(:,2),G(:,3),'LineStyle','-.','Marker','*','color','r');
```

	x	y	z
PT1	2	3	4
PT2	5	6	7
PT3	1	5	9
PT4	4	6	13
PT5	5	3	15
PT6	7	6	9
PT7	10	11	12
PT8	11	12	7
PT9	15	15	1
PT10	3	15	5

INPUT



OUTPUT

B-Splines

A b-spline curve is mathematically defined as:

$$\mathbf{Q}(u) = \sum_{j=0}^{n} \mathbf{P}_j B_{j,d}(u)$$
$$t_{d-1} \leq u \leq t_{n+1}$$

where P_j is a control point. j is the index of the control points. $n+1$ is the number of the control points. d is the number of the control points that control a segment. Implying, $d-1$ is the degree of the polynomial, for example: $d=2$ is linear, $d=3$ is quadratic, $d=4$ is cubic, etc. The value of n must be larger or equal to d. t_j is a knot value. A few knot values form a knot vector $\mathbf{t}$.

1. **when** $d=1$

$$B_{j,1}(u) = \begin{cases} 1 & \textbf{if } t_j \leq u < t_{j+1} \\ 0 & \textbf{otherwise} \end{cases}$$

2. **when** $d>1$

$$B_{j,d}(u) = \left(\frac{u-t_j}{t_{j+d-1}-t_j}\right) B_{j,d-1}(u) + \left(\frac{t_{j+d}-u}{t_{j+d}-t_{j+1}}\right) B_{j+1,d-1}(u)$$

The knot vector, as can be observed in the last two equations, affects the values of the basis functions. Moreover, different ways to build the vector create two different types of b-spline: **uniform** and **non-uniform**. The number of the knot values (or the size of the knot vector) is determine by:

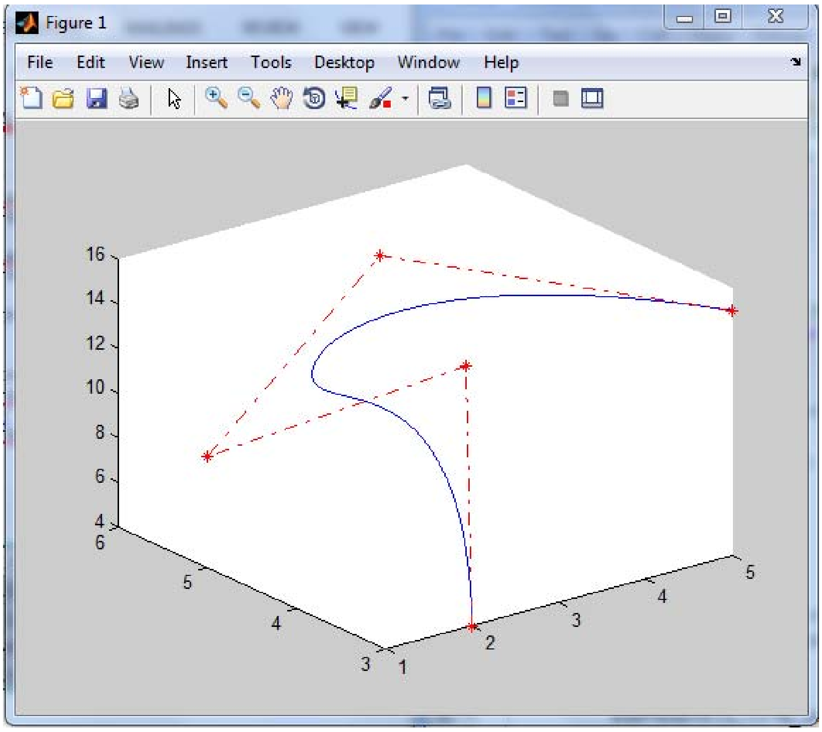
$$m = n + d + 1$$

```
%B-Spline Curves
% P-Control Point Matrix
% N_i,k= normalized B-spline basis function
% B_spline= SUMISSION 0 to Deg (P_i*N_i,k)
% order=k
% no. of pts.= n+1
% no. of knots= n+k+1

% Inputs- P(Control point matrix), Knot(Knot vector), k(order)
xl=xlsread('Control Points', -1);
P=xl;
[n,dump]=size(P);
xl=xlsread('Control Points', -1);
Knot=xl;
k = str2num(input('Enter the order of the B-Spline (k):','s'));
% Knot=[0 0 0 0 1 2 2 2 2];
% k=4;
 B_spl=[];

% Calculating the sumission for each parameter
for t=Knot(k-1):.001:Knot(n+2)-.001
    sub=[0 0 0];
    for i=1:1:n
        sub=sub+P(i,:)*N_ik(t,i,k,Knot);
    end
    B_spl=[B_spl
        sub];
end

% Plotting of B-Spline
plot3(B_spl(:,1),B_spl(:,2),B_spl(:,3));
view(3);
hold on;
plot3(P(:,1),P(:,2),P(:,3),'LineStyle','-.','Marker','*','color','r');
```

```
function [ val ] = N_ik( t,i,k,Knot )
if k~=1       % Use of Recursion
    val1=(t-Knot(i))*N_ik(t,i,k-1,Knot)/(Knot(i+k-1)-Knot(i));
    val2=(Knot(i+k)-t)*N_ik(t,i+1,k-1,Knot)/(Knot(i+k)-Knot(i+1));

    if isnan(val1)  % For 0/0 Cases
        val1=0;
    end
    if isnan(val2)
        val2=0;
    end
    val=val1+val2;

else
    if t>=Knot(i) && t<Knot(i+1)
        val=1;
    else
        val=0;
    end
end
end
```

	x	y	z
P1	2	3	4
P2	5	6	7
P3	1	5	9
P4	4	6	13
P5	5	3	15

INPUT Points



OUTPUT