

```

> restart;
>
> clamped_spline :=
  proc(xx::list,y::list,fp0::numeric,fpn::numeric,S::name)
    local N, I, X, A, AA, FP0, FPN, M, H, XA, XL, XU, XZ, C, J, B, D, SS, L,
      P;
    if nops(xx)<>nops(y) then
      ERROR("Both lists must be the same size");
    fi;
    N:=nops(xx)-1;
  > for I from 0 to N do
    X[I]:= xx[I+1];
  > A[I]:=y[I+1];
  > od;
  > FP0 := fp0;
  > FPN := fpn;
STEP 1
  > M := N - 1;
  > for I from 0 to M do
  > H[I] := X[I+1] - X[I];
  > od;
STEP 2
  > XA[0] := 3.0 * (A[1] - A[0]) / H[0] - 3.0 * FP0;
  > XA[N] := 3.0 * FPN - 3.0 * (A[N] - A[N-1]) / H[N-1];
STEP 3
  > for I from 1 to M do
  > XA[I] :=
    3.0*(A[I+1]*H[I-1]-A[I]*(X[I+1]-X[I-1])+A[I-1]*H[I])/(H[I]*H[I-1]);
  > od;
STEP 4
  > XL[0] := 2.0 * H[0];
  > XU[0] := 0.5;
  > XZ[0] := XA[0] / XL[0];
STEP 5
  > for I from 1 to M do
  > XL[I] := 2.0 * (X[I+1] - X[I-1]) - H[I-1] * XU[I-1];
  > XU[I] := H[I] / XL[I];
  > XZ[I] := (XA[I] - H[I-1] * XZ[I-1]) / XL[I];
  > od;
STEP 6
  > XL[N] := H[N-1] * (2.0 - XU[N-1]);

```

```
> XZ[N] := (XA[N] - H[N-1] * XZ[N-1]) / XL[N];
```

```
> C[N] := XZ[N];
```

```
STEP 7
```

```
> for I from 1 to N do
```

```
> J := N - I;
```

```
> C[J] := XZ[J] - XU[J] * C[J+1];
```

```
> B[J] := (A[J+1] - A[J]) / H[J] - H[J] * (C[J+1] + 2.0 * C[J]) / 3.0;
```

```
> D[J] := (C[J+1] - C[J]) / (3.0 * H[J]);
```

```
> od;
```

```
STEP 8
```

```
> printf(`\n\nThe coefficients of the clamped cubic spline S[i]\non the  
subintervals are:\n`);
```

```
> printf(`          a(i)          b(i)          c(i)          d(i)\n`);
```

```
> for I from 0 to M do
```

```
> printf(` %13.8f %13.8f %13.8f %13.8f \n`, A[I], B[I], C[I], D[I]);
```

```
> od;
```

```
printf(`\n\n`);
```

```
> for I from 0 to M do
```

```
SS[I]:=A[I]+B[I]*(x-X[I])+C[I]*(x-X[I])^2+D[I]*(x-X[I])^3;
```

```
> od;
```

```
L:=seq([x<X[I+1],SS[I]],I=0..M);
```

```
P:=seq(op(L[I]),I=1..M+1);
```

```
P := subsop(2*M+1=NULL,P);
```

```
S:=simplify(piecewise(op(P)));
```

```
> end;
```

```
Warning, imaginary unit `I` used as a local variable in procedure clamped_spline
```

```
clamped_spline := proc(xx::list, y::list, fp0::numeric, fpn::numeric, S::name)
```

```
local N, I, X, A, AA, FP0, FPN, M, H, XA, XL, XU, XZ, C, J, B, D, SS, L, P;
```

```
if nops(xx) <> nops(y) then ERROR("Both lists must be the same size") end if
```

```
N := nops(xx) - 1;
```

```
for I from 0 to N do X[I] := xx[I + 1]; A[I] := y[I + 1] end do
```

```
FP0 := fp0;
```

```
FPN := fpn;
```

```
M := N - 1;
```

```
for I from 0 to M do H[I] := X[I + 1] - X[I] end do
```

```
XA[0] := (3.0*(A[1] - A[0]))/(H[0]) - 3.0*FP0;
```

```
XA[N] := 3.0*FPN - (3.0*(A[N] - A[N - 1]))/(H[N - 1]);
```

```
for I to M do
```

```
XA[I] := (3.0*(A[I + 1]*H[I - 1] - A[I]*(X[I + 1] - X[I - 1]) + A[I - 1]*H[I]))/(H[I]*H[I - 1])
```

```
end do
```

```
XL[0] := 2.0*H[0];
```

```
XU[0] := 0.5;
```

```
XZ[0] := (XA[0])/XL[0];
```

```
for I to M do
```

```
XL[I] := 2.0*(X[I + 1] - X[I - 1]) - H[I - 1]*XU[I - 1];
```

```
XU[I] := (H[I])/XL[I];
```

```
XZ[I] := (XA[I] - H[I - 1]*XZ[I - 1])/XL[I]
```

```
end do
```

```

XL[M] := H[N - 1]*(2.0 - XU[N - 1]);
XZ[M] := (XA[M] - H[N - 1]*XZ[N - 1])/(XL[M]);
C[M] := XZ[M];
for I to N do
    J := N - I;
    C[J] := XZ[J] - XU[J]*C[J + 1];
    B[J] := (A[J + 1] - A[J])/(H[J]) - (H[J]*(C[J + 1] + 2.0*C[J]))/(3.0);
    D[J] := (C[J + 1] - C[J])/(3.0*H[J])
end do
printf(` The coefficients of the clamped cubic spline S[i] on the subintervals are);`
printf(`      a(i)      b(i)      c(i)      d(i) );`
for I from 0 to M do printf(` %13.8f %13.8f %13.8f %13.8f `; A[I], B[I], C[I], D[I]) end do
printf(` `);
for I from 0 to M do SS[I] := A[I] + B[I]*(x - X[I]) + C[I]*x - X[I]^2 + D[I]*x - X[I]^3 end do
L := [seq([x < X[I + 1], SS[I]], I = (0 .. M))];
P := [seq(op(L[I]), I = (1 .. M + 1))];
P := subsop(2*M + 1 = NULL, P);
S := simplify(piecewise(op(P)))
end proc

```

>

```

> clamped_spline_dir:=proc()
printf(`clamped_spline returns the piecewise cubic spline.\n\n`);
printf(`The output also includes a table of values\n`);
printf(`for the cubic polynomials\n\n`);
printf(`S[i] (x)=a[i]+b[i] (x-x[i])+c[i] (x-x[i])^2+d[i] (x-x[i])^3\n\n`);
printf(`for i=0..n-1 making up the spline.\n\n`);
printf(`The arguments for clamped_spline are:\n`);
printf(`(1) the list of x-values\n`);
printf(`(2) the list of f(x) or y-values\n`);
printf(`(3) f'(x[0])\n`);
printf(`(4) f'(x[n])\n`);
printf(`(5) the variable for returning the piecewise cubic spline\n\n`);
printf(`If assigning the result to a variable, have the\n`);
printf(`variable and the 5th argument the same.\n\n`);
printf(`If S is the variable for returning the spline\n`);
printf(`and has already been given a value,\n`);
printf(`the procedure should be preceded by the statement:\n`);
printf(`S:='S'`);
end;

```

```

clamped_spline_dir := proc()
printf(`clamped_spline returns the piecewise cubic spline. );`
printf(`The output also includes a table of values );`
printf(`for the cubic polynomials );`
printf(`S[i](x)=a[i]+b[i](x-x[i])+c[i](x-x[i])^2+d[i](x-x[i])^3 );`
printf(`for i=0..n-1 making up the spline. );`
printf(`The arguments for clamped_spline are: );`
printf(`(1) the list of x-values );`
printf(`(2) the list of f(x) or y-values );`
printf(`(3) f'(x[0]) );`
printf(`(4) f'(x[n]) );`
printf(`(5) the variable for returning the piecewise cubic spline );`
printf(`If assigning the result to a variable, have the );`
printf(`variable and the 5th argument the same. );`
printf(`If S is the variable for returning the spline );`
printf(`and has already been given a value, );`

```

```
    printf('the procedure should be preceded by the statement: ');  
    printf('S:='S')  
end proc  
[>
```