

```

○ restart;
○
○ hermite := proc(xx::list,y::list,z::list,r::name)
local N, K, X, Lprime, J, L, H, Hhat, Y, Z, HH;
○ if (nops(xx)<>nops(y)) or (nops(xx)<>nops(z)) or
(nops(z)<>nops(y)) then
ERROR("All three lists must be the same size");
fi;
N:=nops(xx)-1:
for J from 0 to N do X[J]:=xx[J+1]:od:
for J from 0 to N do Y[J]:=y[J+1]:od:
for J from 0 to N do Z[J]:=z[J+1]:od:
L:=array(N..N,0..N):
for J from 0 to N do L[N,J]:=1:od:
for J from 0 to N do
for K from 0 to N do
if K<>J then L[N,J]:=L[N,J]*(x-X[K])/(X[J]-X[K]) fi:
od:
od:
for J from 0 to N do
L[N,J]:=unapply(collect(L[N,J],x),x):
od:
Lprime:=array(N..N,0..N):
H:=array(N..N,0..N):
for J from 0 to N do
Lprime[N,J]:=D(L[N,J]):
H[N,J]:=unapply(collect((1-2*(x-X[J])*Lprime[N,J](X[J]))*(L[N,J]
(x))^2,x),x):
od:
Hhat:=array(N..N,0..N):
HH:=0:
for J from 0 to N do
Hhat[N,J]:=unapply(collect((x-X[J])*(L[N,J](x))^2,x),x);
HH:=HH+Y[J]*H[N,J](x)+Z[J]*Hhat[N,J](x):
od:
HH:=collect(HH,x);
printf(`\nThe interpolating polynomial is %a`,HH);
HH;
end;

```

hermite := proc(xx:list, y:list, z:list, r:name)

local *N, K, X, Lprime, J, L, H, Hhat, Y, Z, HH;*

if *nops(xx)!* *○nops(y)* **or** *nops(xx)!* *○nops(z)* **or** *nops(z)!* *○nops(y)* **then**

ERROR("All three lists must be the same size")

end if;

N := nops(xx) - 1;

for *J* **from** 0 **to** *N* **do**

X[J] := xx[J+1]

end do;

for *J* **from** 0 **to** *N* **do**

Y[J] := y[J+1]

end do;

for *J* **from** 0 **to** *N* **do**

```

    Z[J] := z[J C 1]
end do;
L := array(N..N, 0..N);
for J from 0 to N do
    L[N, J] := 1
end do;
for J from 0 to N do
    for K from 0 to N do
        if K! = J then
            L[N, J] := L[N, J] * (x - X[K]) / (X[J] - X[K])
        end if
    end do
end do;
for J from 0 to N do
    L[N, J] := unapply(collect(L[N, J], x), x)
end do;
Lprime := array(N..N, 0..N);
H := array(N..N, 0..N);
for J from 0 to N do
    Lprime[N, J] := D(L[N, J]);
    H[N, J] := unapply(
        collect((1 - x^2) * (x - X[J]) * Lprime[N, J] * L[N, J](x)^2, x), x)
end do;
Hhat := array(N..N, 0..N);
HH := 0;
for J from 0 to N do
    Hhat[N, J] := unapply(collect((x - X[J]) * L[N, J](x)^2, x), x);
    HH := HH + Y[J] * H[N, J](x) + Z[J] * Hhat[N, J](x)
end do;
HH := collect(HH, x);
printf(`
    The interpolating polynomial is %a`, HH);
HH
end proc

```

○

○ hermite_dir:=proc()
 printf(`hermite returns the interpolating polynomial.\n\n`);

```

printf(`The arguments for hermite are:\n`);
printf(`(1)the list of x-values\n`);
printf(`(2)the list of f(x) or y-values\n`);
printf(`(3)the list of f'(x)-values\n`);
printf(`(4)the variable for returning the polynomial\n\n`);
printf(`If assigning the result to a variable, have the\n`);
printf(`variable and the 4th argument the same.\n\n`);
printf(`If p is the variable for returning the polynomial\n`);
printf(`and has already been given a value,\n`);
printf(`the procedure should be preceded by the statement:\n`);
printf(`p:='p'`);
end;

```

```
hermite_dir := proc()
```

```

    printf(`hermite returns the interpolating polynomial.

```

```

    `);

```

```

    printf(`The arguments for hermite are:

```

```

    `);

```

```

    printf(`(1)the list of x-values

```

```

    `);

```

```

    printf(`(2)the list of f(x) or y-values

```

```

    `);

```

```

    printf(`(3)the list of f'(x)-values

```

```

    `);

```

```

    printf(`(4)the variable for returning the polynomial

```

```

    `);

```

```

    printf(`If assigning the result to a variable, have the

```

```

    `);

```

```

    printf(`variable and the 4th argument the same.

```

```

    `);

```

```

    printf(`If p is the variable for returning the polynomial

```

```

    `);

```

```

    printf(`and has already been given a value,

```

```

    `);

```

```

    printf(`the procedure should be preceded by the statement:

```

```

    `);

```

```

    printf(`p:='p'`)

```

```
end proc
```



