

```

○ restart;
○
○ hermite_dd := proc(xx::list,y::list,z::list,r::name)
  local N, I, X, Q, J, K, P, PP, C, L, M;
○ 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 I from 0 to 4*N+2 do
○ for J from 0 to 2*N+1 do
  Q[I,J]:=infinity;
  od;
  od;
○ for I from 0 to N do
  X[4*I] := xx[I+1];
○ X[4*I+2] := xx[I+1];
  Q[4*I,0] := round((10.^7)*y[I+1])/(10.^7);
○ Q[4*I+2,0] := round((10.^7)*y[I+1])/(10.^7);
  od;
STEP 1
○ for I from 3 to 4*N-1 by 4 do
  Q[I,1]:=round((10.^7)*(Q[I+1,0] - Q[I-1,0]) / (X[I+1] - X[I-1]))/
  (10.^7);
  od;
  for I from 0 to N do
  Q[4*I+1,1]:=round((10.^7)*z[I+1])/(10.^7);
  od;
  for J from 2 to 2*N+1 do
○ for I from J to 4*N-J+2 by 2 do
○ Q[I,J] := round((10.^7)*(Q[I+1,J-1] - Q[I-1,J-1]) / (X[I+J] - X[I-
  J]))/(10.^7);
○ od;
○ od;
STEP 2
○ printf(` i      z[i]      f(z[i])\n`);
  printf(` -      ----      -\n`);
○ for I from 0 to 4*N+2 do
○ if Q[I,0]<>infinity then
  printf(`%2d %7.3f %11.7f`, iquo(I,2), X[I], Q[I,0]);
  else
  printf(`          `);
  fi;
  for J from 1 to min(2*N+1,4) do
  if Q[I,J]<>infinity then
  printf(` %11.7f`,Q[I,J]);
  else
  printf(`          `);
  fi
  od;
  printf(`\n`);
○ od;
  if 2*N+1>4 then
  K:=iquo(2*N+1,4)-1;
  L:=irem(2*N+1,4);
  if K>0 then
  for M from 1 to K do
○ for I from 4*M to 4*N-4*M+2 do
  for J from 4*M+1 to 4*(M+1) do
  if Q[I,J]<>infinity then
  printf(` %11.7f`,Q[I,J]);

```

```

else
printf(`          `);
fi
od;
printf(`\n`);
○ od;
od;
fi;
if L>0 then
K:=K+1;
○ for I from 4*K to 4*N-4*K+2 do
for J from 4*K+1 to 4*K+L do
if Q[I,J]<>infinity then
printf(` %11.7f`,Q[I,J]);
else
printf(`          `);
fi
od;
printf(`\n`);
od;
○ fi;
fi;
○ P:=Q[0,0];
for J from 1 to 2*N+1 do
PP:=1;
for K from 0 to J-1 do
PP:=PP*(x-X[2*K])
od;
P:=P+Q[J,J]*PP;
od;
printf(`\nThe interpolating polynomial is %a`,P);
PP:=0;
for J from 0 to 2*N+1 do
C[J]:=round((10.^7)*coeff(P,x,J))/(10.^7);
PP:=PP+C[J]*x^J;
od;
r:=PP;
end;

```

Warning, imaginary unit `I` used as a local variable in procedure hermite\_dd  
hermite\_dd := **proc** (xx:list, y:list, z:list, r:name)

**local** N, I, X, Q, J, K, P, PP, C, L, M;

**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** I **from** 0 **to** 4 \* N - 2 **do**

**for** J **from** 0 **to** 2 \* N - 1 **do**

        Q[I, J] := infinity

**end do**

**end do**;

**for** I **from** 0 **to** N **do**

    X[4 \* I] := xx[I - 1];

```

X[4 * IC 2] := xx[IC 1];
Q[4 * I, 0] := round(10.^7 * y[IC 1]) / 10.^7;
Q[4 * IC 2, 0] := round(10.^7 * y[IC 1]) / 10.^7
end do;
for I from 3 by 4 to 4 * NK 1 do
    Q[I, 1] := round(10.^7 * (Q[IC 1, 0] K Q[IK 1, 0]) / (X[IC 1] K X[IK 1])) / 10.^7
end do;
for I from 0 to N do
    Q[4 * IC 1, 1] := round(10.^7 * z[IC 1]) / 10.^7
end do;
for J from 2 to 2 * NC 1 do
    for I from J by 2 to 4 * NK JC 2 do
        Q[I, J] :=
            round(10.^7 * (Q[IC 1, JK 1] K Q[IK 1, JK 1]) / (X[IC J] K X[IK J]))
            / 10.^7 end do
    end do;
    printf(` i z[i] f(z[i])
           `);
    printf(` - ---- -
           `);
    for I from 0 to 4 * NC 2 do
        if Q[I, 0]! O infinity then
            printf(` %2d %7.3f %11.7f, iquo(I,
            2), X[I], Q[I, 0])
        else
            printf(
        end if;
        for J to min(2 * NC 1, 4) do
            if Q[I, J]! O infinity then
                printf(` %11.7f, Q[I, J])
            else
                printf(
            end if
        end do;
        printf(
    )

```

```

end do;
if 0! 2 * N K 3 then
    K := iquo(2 * N C 1, 4) K 1;
    L := irem(2 * N C 1, 4);
    if 0! K then
        for M to K do
            for I from 4 * M to 4 * N K 4 * M C 2 do
                for J from 4 * M C 1 to 4 * M C 4 do
                    if Q[I, J]! 0 infinity then
                        printf(`%11.7f, Q[I, J])
                    else
                        printf(
                    )
                    end if
                end do;
                printf(
            )
            end do
        end do
    end if;
    if 0! L then
        K := K C 1;
        for I from 4 * K to 4 * N K 4 * K C 2 do
            for J from 4 * K C 1 to 4 * K C L do
                if Q[I, J]! 0 infinity then
                    printf(`%11.7f, Q[I, J])
                else
                    printf(
                )
                end if
            end do;
            printf(
        )
        end do
    end if
end if;
P := Q[0, 0];
for J to 2 * N C 1 do

```

```

    PP := 1;
    for K from 0 to J - 1 do
        PP := PP * (x - X[2 * K])
    end do;
    P := P * Q[J, J] * PP
end do;
printf(`
    The interpolating polynomial is %a`, P);
PP := 0;
for J from 0 to 2 * N - 1 do
    C[J] := round(10.^7 * coeff(P, x, J)) / 10.^7;
    PP := PP * C[J] * x^J
end do;
r := PP
end proc

```

○

```

○ hermite_dd_dir := proc()
    printf(`hermite_dd builds a divided-difference table and\n`);
    printf(`returns the interpolating polynomial\n`);
    printf(`with all coefficients rounded to 7 decimal places.\n\n`);
    printf(`The arguments for hermite_dd 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\_dd\_dir := proc()*

*printf(`hermite\_dd builds a divided-difference table and`);*

*printf(`returns the interpolating polynomial`);*

*printf(`with all coefficients rounded to 7 decimal places.`);*

*);*

*printf(`The arguments for hermite\_dd 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**

