

```

> restart;
>
> muller := proc(f::algebraic,x0::complex,x1::complex,x2::complex,
tol::positive,no::posint,root::name)
local P, OK, TOL, X, F, H, DEL1, DEL, II, B, D, E, J, NO;
> X[0]:=evalf(x0);
X[1]:=evalf(x1);
X[2]:=evalf(x2);
NO:=no;
TOL:=evalf(tol);
P := unapply(f,x);
printf(` i      p                f(p)\n`);
printf(` -      -                ----\n`);
> F[0] := P(X[0]);
> F[1] := P(X[1]);
> F[2] := P(X[2]);
printf(`%3d    %-30a    %-30a\n`,0,X[0],F[0]);
printf(`%3d    %-30a    %-30a\n`,1,X[1],F[1]);
printf(`%3d    %-30a    %-30a\n`,2,X[2],F[2]);
OK:=TRUE;
STEP 1
> H[0] := X[1]-X[0];
> H[1] := X[2]-X[1];
> DEL1[0] := (F[1]-F[0])/H[0];
> DEL1[1] := (F[2]-F[1])/H[1];
> DEL := (DEL1[1]-DEL1[0])/(H[1]+H[0]);
> II := 3;
STEP 2
> while II <= NO and OK = TRUE do
STEPS 3
> B := DEL1[1]+H[1]*DEL;
> D := B*B-4*F[2]*DEL;
test to see if straight line
> if abs(DEL) <= 1.0e-20 then
straight line - test if horizontal line
> if abs(DEL1[1]) <= 1.0e-20 then
> printf(`Horizontal Line\n`);
> OK := FALSE;
> else
straight line but not horizontal
> X[3] := (F[2]-DEL1[1]*X[2])/DEL1[1];
> H[2] := X[3]-X[2];
> fi;
> else
not a straight line
> D := sqrt(D);
STEP 4
> E := B+D;
> if abs(B-D) > abs(E) then
> E := B-D;
> fi;
STEP 5
> H[2] := -2*F[2]/E;
> X[3] := X[2]+H[2];
> fi;
> if OK = TRUE then
> F[3] := P(X[3]);
> printf(`%3d    %-30a    %-30a\n`,II,X[3],F[3]);
> fi;
STEP 6

```



```

printf(`%3d  %-30a  %-30a
      `, 1, X[1], F[1]);
printf(`%3d  %-30a  %-30a
      `, 2, X[2], F[2]);
OK := TRUE;
H[0] := X[1] - X[0];
H[1] := X[2] - X[1];
DELI[0] := (F[1] - F[0])/H[0];
DELI[1] := (F[2] - F[1])/H[1];
DEL := (DELI[1] - DELI[0])/(H[1] + H[0]);
II := 3;
while II <= NO and OK = TRUE do
  B := DELI[1] + H[1]*DEL;
  D := B*B - 4*F[2]*DEL;
  if abs(DEL) <= 1.0&sdot;10^-20 then
    if abs(DELI[1]) <= 1.0&sdot;10^-20 then
      printf(`Horizontal Line
            `);
      OK := FALSE
    else
      X[3] := (F[2] - DELI[1]*X[2])/DELI[1];
      H[2] := X[3] - X[2]
    end if
  else
    D := sqrt(D);
    E := B + D;
    if abs(E) < abs(B - D) then
      E := B - D
    end if;
    H[2] := -(2*F[2])/E;
    X[3] := X[2] + H[2]
  end if;
  if OK = TRUE then
    F[3] := P(X[3]);
    printf(`%3d  %-30a  %-30a
          `, II, X[3], F[3])
  end if;

```

```

end if;
if abs( $H[2]$ ) <  $TOL$  then
    printf(`
        The approximate solution is %a`, args[7]);
    printf(` = %-a
        `,  $X[3]$ );
    printf(`with f(%a`, args[7]);
    printf(`) = %-a
        `,  $F[3]$ );
    root :=  $X[3]$ ;
    OK := FALSE
else
    for  $J$  to 2 do
         $H[J - 1]$  :=  $H[J]$ ;
         $X[J - 1]$  :=  $X[J]$ ;
         $F[J - 1]$  :=  $F[J]$ 
    end do;
     $X[2]$  :=  $X[3]$ ;
     $F[2]$  :=  $F[3]$ ;
     $DELI[0]$  :=  $DELI[1]$ ;
     $DELI[1]$  := ( $F[2] - F[1]$ )/ $H[1]$ ;
     $DEL$  := ( $DELI[1] - DELI[0]$ )/( $H[1] + H[0]$ )
end if;
     $II$  :=  $II + 1$ 
end do;
if  $NO < II$  and  $OK = TRUE$  then
    printf(`
        Iteration number %3d`,  $II - 1$ );
    printf(` gave approximation %-a
        `,  $X[3]$ );
    printf(`  $F(P) =$  %-a not within tolerance : %15.8e
        `,  $F[3]$ ,  $TOL$ );
    RETURN()
else
     $X[3]$ 
end if

```

end proc

```
>
> numanal[muller_dir]:=proc()
  printf(`muller returns a root of the given function.\n\n`);
  printf(`The arguments for muller are:\n`);
  printf(`(1)function expression in x (may include complex numbers)
\n`);
  printf(`(2)first initial approximation (real or complex)\n`);
  printf(`(3)second initial approximation (real or complex)\n`);
  printf(`(4)third initial approximation (real or complex)\n`);
  printf(`(5)tolerance\n`);
  printf(`(6)maximum number of iterations\n`);
  printf(`(7)variable for returning root\n\n`);
  printf(`If assigning the result to a variable, have the\n`);
  printf(`variable and the 7th argument the same.\n\n`);
  printf(`If r is the variable for returning the root\n`);
  printf(`and has already been given a value,\n`);
  printf(`the procedure should be preceded by the statement:\n`);
  printf(`r:='r'`);
end;
```

numanal<sub>muller\_dir</sub> := proc()

*printf(`muller returns a root of the given function.*

*`);*

*printf(`The arguments for muller are:*

*`);*

*printf(`(1)function expression in x (may include complex numbers)*

*`);*

*printf(`(2)first initial approximation (real or complex)*

*`);*

*printf(`(3)second initial approximation (real or complex)*

*`);*

*printf(`(4)third initial approximation (real or complex)*

*`);*

*printf(`(5)tolerance*

*`);*

*printf(`(6)maximum number of iterations*

*`);*

*printf(`(7)variable for returning root*

*`);*

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

*`);*

*printf(`variable and the 7th argument the same.*

```
    `);  
    printf(`If r is the variable for returning the root  
    `);  
    printf(`and has already been given a value,  
    `);  
    printf(`the procedure should be preceded by the statement:  
    `);  
    printf(`r:='r'`)
```

**end proc**

[>