

```

[> restart;

> newton := proc(f::algebraic,p0::complex,tol::positive,no::posint,
root::name)
local F, FP, OK, P0, TOL, NO, F0, II, FP0, D, COMP;
NO:=no;
P0:=p0;
TOL:=evalf(tol);
FP := unapply(diff(f,x),x);
F := unapply(f,x);
printf(` i      p          f(p)\n`);
printf(` -      -          ----\n`);
F0 := evalf(F(P0));
STEP 1
II := 0;
printf(`%3d    %-30a    %-30a\n`,II,P0,F0);
OK := TRUE;
STEP 2
while II <= NO and OK = TRUE do
STEP 3
compute P(I)
FP0 := evalf(FP(P0));
D := F0/FP0;
STEP 6
P0 := P0 - D;
F0 := evalf(F(P0));
printf(`%3d    %-30a    %-30a\n`,II+1,P0,F0);
STEP 4
if abs(D) < TOL then
procedure completed sucessfully
printf(`\nThe approximate solution is %a`,args[5]);
printf(` = %-a \n`,P0);
printf(`with f(%a`,args[5]);
printf(`) = %-a\n`,F0); OK := FALSE;
STEP 5
else
II := II+1;
fi;
od;
if OK = TRUE then
STEP 7
procedure completed unsuccessfully
printf(`\nIteration number %3d`,NO);
printf(` gave approximation %-a\n`,P0);
printf(`F(P) = %-a not within tolerance : %-a\n`,F0,TOL);
RETURN();
else
P0;
fi;
end;
newton := proc( f:algebraic, p0:complex, tol:positive, no:posint, root:name)
local F, FP, OK, P0, TOL, NO, F0, II, FP0, D, COMP,
NO := no;

```

```

P0 := p0;
TOL := evalf(tol);
FP := unapply(diff(f,x),x);
F := unapply(f,x);
printf(" i  p                f(p)
          ); printf(" - - - - -
          ); F0 := evalf(F(P0)); II := 0; printf("%3d %-30a %-30a
, II, P0, F0);
OK := TRUE;
while II <= NO and OK = TRUE do
  FP0 := evalf(FP(P0));
  D := F0/FP0;
  P0 := P0 - D;
  F0 := evalf(F(P0));
  printf("%3d %-30a %-30a
          , II + 1, P0, F0); if abs(D) < TOL then printf(
The approximate solution is %a, args[5]); printf(" = %a
          , P0); printf("with f(%a, args[5]); printf(") = %a
          , F0); OK := FALSE else II := II + 1 end if end do; if OK = TRUE then printf(
Iteration number %3d, NO); printf(" gave approximation %a
          , P0); printf("F(P) = %a not within tolerance : %a
          , F0, TOL); RETURN( ) else P0 end if end proc

```

```
> r:=newton(cos(x)-x, .7853981635, 10^(-8), 100, r);
```

i	p	f(p)
0	.7853981635	-.782913824e-1
1	.7395361335	-.7548747e-3
2	.7390851781	-.751e-7
3	.7390851332	0.
4	.7390851332	0.

The approximate solution is r = .7390851332  
with f(r) = 0.

```
r := 0.7390851332
```

```
> r:='r';
```

```
r := r
```

```
> r:=newton(cos(x)-x, .7853981635, .00000001, 1, r);
```

i	p	f(p)
0	.7853981635	-.782913824e-1
1	.7395361335	-.7548747e-3
2	.7390851781	-.751e-7

```
Iteration number 1 gave approximation .7390851781
F(P) = -.751e-7 not within tolerance : .1e-7
r :=
```

```
> r:='r';
```

```
r := r
```

```
> newton(exp(x)+2^(-x)+2*cos(x)-6,1.5,.000001,100,r);
```

i	p	f(p)
0	1.5	-1.023283136
1	1.956489721	.579701372
2	1.841533061	.50340952e-1
3	1.829506013	.502120e-3
4	1.829383615	.53e-7
5	1.829383602	0.

```
The approximate solution is r = 1.829383602
with f(r) = 0.
```

```
1.829383602
```

```
> r:='r';
```

```
r := r
```

```
> newton((3+5*I)*x+cos(I*x),1.5*3*I,.0001,100,r);
```

i	p	f(p)
0	4.5*I	-22.71079580+13.5*I
1	.5492017089-.736382444*I	6.184993766+.1491868010*I
2	-.1884208808+.1291969995*I	-.201926558-.5789330481*I
3	-.850677397e-1+.1464849608*I	.52438949e-2+.168457553e-2*I
4	-.8575253323e-1+.1471159178*I	.296e-7-.42934e-6*I
5	-.8575247254e-1+.1471159579*I	.2e-9-.6e-10*I

```
The approximate solution is r = -.8575247254e-1+.1471159579*I
with f(r) = .2e-9-.6e-10*I
```

```
-0.08575247254 + 0.1471159579 I
```

```
> r:='r';
```

```
r := r
```

```
> newton((3+5*I)*x+cos(I*x),1.5*3*I,.0000001,3,r);
```

i	p	f(p)
0	4.5*I	-22.71079580+13.5*I
1	.5492017089-.736382444*I	6.184993766+.1491868010*I
2	-.1884208808+.1291969995*I	-.201926558-.5789330481*I
3	-.850677397e-1+.1464849608*I	.52438949e-2+.168457553e-2*I
4	-.8575253323e-1+.1471159178*I	.296e-7-.42934e-6*I

```
Iteration number 3 gave approximation -.8575253323e-1+.1471159178*I
F(P) = .296e-7-.42934e-6*I not within tolerance : .1e-6
```

```
> newton_dir:=proc()
  printf(`newton returns a root of the given function.\n\n`);
  printf(`The arguments for newton are:\n`);
  printf(`(1)function expression in x (may include complex numbers)
\n`);
  printf(`(2)initial approximation (real or complex)\n`);
  printf(`(3)tolerance\n`);
  printf(`(4)maximum number of iterations\n`);
  printf(`(5)variable for returning root\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 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;**

*newton\_dir := proc( ) printf (newton returns a root of the given function.*

*); printf (The arguments for newton are:*

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

*); printf ((2)initial approximation (real or complex)*

*); printf ((3)tolerance*

*); printf ((4)maximum number of iterations*

*); printf ((5)variable for returning root*

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

*); printf (variable and the 5th 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*

**> newton\_dir();**

newton returns a root of the given function.

The arguments for newton are:

(1)function expression in x (may include complex numbers)

(2)initial approximation (real or complex)

(3)tolerance

(4)maximum number of iterations

(5)variable for returning root

If assigning the result to a variable, have the variable and the 5th argument the same.

If r is the variable for returning the root and has already been given a value, the procedure should be preceded by the statement:

r:='r'