

Secant Algorithm

```
> restart;
```

```
> secant:= proc(f::algebraic,p0::numeric,p1::numeric,tol::positive,no::posint,root::name)
  local F, Q, OK, P0, P1, TOL, NO, Q0, Q1, I, P, FP;
  P0:=evalf(p0);
  P1:=evalf(p1);
  F := unapply(f,x);
  Q0 := evalf(F(P0));
  Q1 := evalf(F(P1));
  NO:=no;
  TOL:=evalf(tol);
  printf(` i p          f(p)\n`);
  printf(` - -          ----\n`);
```

STEP 1

```
> I := 2;
```

```
> OK := TRUE;
```

```
> printf(`%3d %14.8e %14.7e\n`,0,P0,Q0);
  printf(`%3d %14.8e %14.7e\n`,1,P1,Q1);
```

STEP 2

```
> while I <= NO and OK = TRUE do
```

STEP 3

compute P(I)

```
> P := P1-Q1*(P1-P0)/(Q1-Q0);
```

```
> Q := evalf(F(P));
```

```
> printf(`%3d %14.8e %14.7e\n`,I,P,Q);
```

STEP 4

```
> if abs(P-P1) < TOL then
```

procedure completed successfully

```
> printf(`\nThe approximate solution is %a`,args[6]);
```

```
  printf(` = %11.8f \n`,P);
```

```
  printf(` with f(%a`,args[6]);
```

```
  printf(`) = %12.8f\n`,Q);
```

```
  root:=P;
```

```
  OK := FALSE;
```

```
> else
```

STEP 5

```
> I := I+1;
```

STEP 6

update P0, Q0, P1, Q1

```
> P0 := P1;  
  Q0 := Q1;  
  P1 := P;  
  
> Q1 := Q;  
  
> fi;  
  
> od;  
  
> if OK = TRUE then
```

procedure completed unsuccessfully

```
> printf(`\nIteration number %3d`,NO);  
  printf(` gave approximation %12.8f\n`,Q);  
  printf(`F(P) = %12.8f not within tolerance : %15.8e\n`,F0,TOL);  
  RETURN();  
else  
  P;  
  fi;  
end;
```

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

secant := **proc**(f::algebraic, p0::numeric, p1::numeric, tol::positive, no::posint, root::name)

local F, Q, OK, P0, P1, TOL, NO, Q0, Q1, I, P, FP;

P0 := evalf(p0);

P1 := evalf(p1);

F := unapply(f, x);

Q0 := evalf(F(P0));

Q1 := evalf(F(P1));

NO := no;

TOL := evalf(tol);

printf(` i p f(p)\n`);

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

I := 2;

OK := TRUE;

printf(`%3d %14.8e %14.7e\n`,0, P0, Q0);

printf(`%3d %14.8e %14.7e\n`,1, P1, Q1);

while I <= NO **and** OK = TRUE **do**

 P := P1 - (Q1*(P1 - P0))/(Q1 - Q0);

 Q := evalf(F(P));

 printf(`%3d %14.8e %14.7e\n`,I, P, Q);

if `abs`(P - P1) < TOL **then**

 printf(`\n□ The approximate solution is %q`args[6]);

 printf(` = %11.8f`□, P);

 printf(` with f(%a` , args[6]);

 printf(`) = %12.8f`□, Q);

 root := P;

 OK := FALSE

else

 I := I + 1;

 P0 := P1;

 Q0 := Q1;

 P1 := P;

 Q1 := Q

end if

```

end do
if OK = TRUE then
    printf( '\nIteration number %3d\n', NO);
    printf( ' gave approximation %12.8f\n', Q);
    printf( ' F(P) = %12.8f not within tolerance : %15.8e\n', F0, TOL);
    RETURN()
else
    P
end if
end proc

```

```

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

```

```

secant_dir := proc()
    printf( 'falseposition returns a root of the given function. \n\n');
    printf( 'The arguments for falseposition are:\n');
    printf( '(1)function expression in x\n');
    printf( '(2)first initial approximation\n');
    printf( '(3)second initial approximation\n');
    printf( '(4)tolerance\n');
    printf( '(5)maximum number of iterations\n');
    printf( '(6)variable for returning root\n\n');
    printf( 'If assigning the result to a variable, have the\n');
    printf( 'variable and the 6th argument the same.\n\n');
    printf(
        'If r is the variable for returning the root and has already been given a value, the procedure should be
        preceded by the statement:\n');
    printf( 'r:='r')
end proc

```