

Secant Method

nalib

The **secant** method is used rather than **Newton-Raphson** if the derivative is hard or inconvenient to evaluate. This method approximates the derivative by using a secant line joining the two previous approximations. As such, we need to give this algorithm two initial starting points, p_0 and p_1 . We try it with our original problem.

```
> restart;
> libname:="c:/nalib",libname;
libname := "/nalib", "/Library/Frameworks/Maple.framework/Versions/15/lib",
           "/Library/Frameworks/Maple.framework/Versions/15/toolbox/NAG/lib"
> with(numanal);
[SOR, SOR_dir, adaptq, adaptq_dir, bezier, bezier_dir, bisection, bisection_dir, chop, chop_dir,
 clamped_spline, clamped_spline_dir, divided_diff, divided_diff_dir, extrap, extrap_dir,
 falseposition, falseposition_dir, fixedpoint, fixedpoint_dir, gaussseidel, gaussseidel_dir, hermite,
 hermite_dd, hermite_dd_dir, hermite_dir, horner, horner_dir, jacobi, jacobi_dir, muller,
 muller_dir, natural_spline, natural_spline_dir, newton, newton_dir, romberg, romberg_dir,
 secant, secant_dir, steffensen, steffensen_dir]
```

We continue to work with the following function, which has roots near .12, .3, and 4.8. Suppose we wish to find all the roots of the following function accurate to within 10^{-6} .

```
> f:=4*x^3-20*x^2+3*x+2+ln(x);
           f:= 4 x3 - 20 x2 + 3 x + 2 + ln(x)
```

Let's look at the directions for **secant**.

```
> secant_dir();
falseposition returns a root of the given function.
```

The arguments for falseposition are:

- (1)function expression in x
- (2)first initial approximation
- (3)second initial approximation
- (4)tolerance
- (5)maximum number of iterations
- (6)variable for returning root

If assigning the result to a variable, have the variable and the 6th 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'
```

We know the first root is between .1 and .14, so let's use these as our two initial approximations.

```
> r1:=secant(f, .1, .14, .000001, 100, r1);
  i   p                               f(p)
  -   -                               -
  0   1.00000000e-01                   -1.9858509e-01
  1   1.40000000e-01                   7.2863144e-02
```

```

2  1.29263051e-01    1.6344024e-02
3  1.26158174e-01   -2.0303100e-03
4  1.26501254e-01    4.6737000e-05
5  1.26493534e-01    1.2800000e-07
6  1.26493513e-01    1.0000000e-09

```

```

The approximate solution is r1 = 0.12649351
with f(r1) = 0.00000000
                    r1 := 0.1264935127

```

We see that **secant** takes more steps than **newton**, but the tradeoff is that the derivative does not need to be computed at each step. Now let's go for the second root and take .2 and .23 as initial approximations.

```
> r2:=secant(f, .23, .2, .000001, 100, r2);
```

i	p	f(p)
0	2.30000000e-01	2.1099203e-01
1	2.00000000e-01	2.2256209e-01
2	7.77081173e-01	-6.1210921e+00
3	2.20246436e-01	2.2029655e-01
4	2.39590586e-01	1.9688857e-01
5	4.02297585e-01	-6.8010064e-01
6	2.76119141e-01	1.0080624e-01
7	2.92407351e-01	3.7579184e-02
8	3.02088295e-01	-5.6467170e-03
9	3.00823647e-01	2.3451600e-04
10	3.00874075e-01	1.3400000e-06
11	3.00874365e-01	1.0000000e-09

```

The approximate solution is r2 = 0.30087436
with f(r2) = 0.00000000
                    r2 := 0.3008743650

```

Notice that our initial approximations did not bracket the root. Neither did some other pairs of successive approximations, such as the third and fourth. Also notice that the initial approximations can be entered out of order. Now let's find the third root, and bracket the root with initial approximations of 4 and 5.

```
> r3:=secant(f, 4, 5, .000001, 100, r3);
```

i	p	f(p)
0	4.00000000e+00	-4.8613706e+01
1	5.00000000e+00	1.8609438e+01
2	4.72316918e+00	-6.9805422e+00
3	4.79868425e+00	-5.7869447e-01
4	4.80551043e+00	2.0958366e-02
5	4.80527185e+00	-5.9232000e-05
6	4.80527252e+00	-2.7300000e-07

```

The approximate solution is r3 = 4.80527252
with f(r3) = -0.00000027
                    r3 := 4.805272523

```

NumericalAnalysis

There is a procedure called [Secant](#) in the **NumericalAnalysis** subpackage of the **Student** package. We use it to find the third root above.

```
> with(Student[NumericalAnalysis]);
```

```

[AbsoluteError, AdamsBashforth, AdamsBashforthMoulton, AdamsMoulton, AdaptiveQuadrature,
AddPoint, ApproximateExactUpperBound, ApproximateValue, BackSubstitution, BasisFunctions,
Bisection, CubicSpline, DataPoints, Distance, DividedDifferenceTable, Draw, Euler, EulerTutor,

```

ExactValue, FalsePosition, FixedPointIteration, ForwardSubstitution, Function, InitialValueProblem, InitialValueProblemTutor, Interpolant, InterpolantRemainderTerm, IsConvergent, IsMatrixShape, IterativeApproximate, IterativeFormula, IterativeFormulaTutor, LeadingPrincipalSubmatrix, LinearSolve, LinearSystem, MatrixConvergence, MatrixDecomposition, MatrixDecompositionTutor, ModifiedNewton, NevilleTable, Newton, NumberOfSignificantDigits, PolynomialInterpolation, Quadrature, RateOfConvergence, RelativeError, RemainderTerm, Roots, RungeKutta, Secant, SpectralRadius, Steffensen, Taylor, TaylorPolynomial, UpperBoundOfRemainderTerm, VectorLimit]

> Secant(f, x=[4,5], stoppingcriterion=absolute, tolerance=10⁽⁻⁶⁾, output=information, maxiterations=100);

<i>n</i>	<i>p_n</i>	<i>absolute error</i>
0	4.	-
1	5.	-
2	4.723169181	0.276830819
3	4.798684254	0.075515073
4	4.805510432	0.006826178
5	4.805271851	0.000238581
6	4.805272523	6.72 10 ⁻⁷

> Secant(f, x=[4,5], stoppingcriterion=absolute, tolerance=10⁽⁻⁶⁾, output=plot, maxiterations=100);

5 iteration(s) of the secant method applied to
 $f(x) = 4x^3 - 20x^2 + 3x + 2 + \ln(x)$
with initial points $a = 4.$ and $b = 5.$

