

Newton-Raphson Method

nalib

```
> 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 start with the third root and use 4.8 as an initial approximation. Let's first check the directions for the procedure **newton**.

```
> 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'

We now find the third root.

```
> r3:=newton(f,4.8,.000001,100,r3);
```

i	p	f(p)
0	4.8	-.463384082
1	4.805284444	.1049768e-2
2	4.805272526	.338e-6
3	4.805272522	-.73e-7

The approximate solution is $r3 = 4.805272522$

```
with f(r3) = -.73e-7
```

```
r3 := 4.805272522
```

Note that this only takes three iterations as opposed to 20 for **bisection**. **newton** converges much quicker to a root than does **bisection**. Now let's find the second root using .3 as an initial approximation.

```
> r2:=newton(f, .3, .000001, 100, r2);
```

i	p	f(p)
0	.3	.4027196e-2
1	.3008780224	-.16916e-4
2	.3008743650	.1e-8
3	.3008743652	-.1e-8

The approximate solution is r2 = .3008743652

```
with f(r2) = -.1e-8
```

```
r2 := 0.3008743652
```

Again, three iterations does the trick as opposed to 18. Now, let's try to find the first root with an initial approximation of 0.2081180627. Now I know this is a kind of funny first guess, but let's see what happens.

```
> r1:=newton(f, .2081180627, .000001, 100, r1);
```

i	p	f(p)
0	.2081180627	.224498854
1	37416475.88	.2095311395e24
2	24944317.80	.6208330052e23
3	16629545.76	.1839505202e23
4	11086364.40	.5450385790e22
5	7390910.156	.1614929123e22
6	4927273.993	.4784975180e21
7	3284849.884	.1417770424e21
8	2189900.478	.4200801253e20
9	1459934.208	.1244681854e20
10	973290.0275	.3687946234e19
11	648860.5739	.1092724810e19
12	432574.2715	.3237703141e18
13	288383.4033	.9593194498e17
14	192256.1578	.2842428001e17
15	128171.3274	.8422008886e16
16	85448.10716	.2495410039e16
17	56965.96035	.7393807522e15
18	37977.86249	.2190757783e15
19	25319.13060	.6491134171e14
20	16879.97602	.1923299008e14
21	11253.87300	.5698663685e13
22	7503.137705	.1688492918e13
23	5002.647584	.5002941813e12
24	3335.654282	.1482353017e12
25	2224.325583	.4392156339e11
26	1483.440036	.1301379156e11
27	989.5167177	3855934905.
28	660.2350754	1142497006.
29	440.7148360	338516145.4
30	294.3692950	100300087.1
31	196.8075337	29717880.04
32	131.7692692	8804852.619
33	88.41481519	2608546.074
34	59.51848510	772700.6868
35	40.26435169	228811.2979
36	27.44368495	67702.01308

```

37 18.92037430          19994.59511
38 13.27538026          5878.082480
39 9.571224937          1708.021873
40 7.197507367          480.9259309
41 5.769952971          123.5967202
42 5.050880260          23.96430812
43 4.827539849          1.980123940
44 4.805481425          .18402510e-1
45 4.805272544          .1792e-5
46 4.805272524          -.72e-7

```

The approximate solution is $r1 = 4.805272524$
with $f(r1) = -.72e-7$

$r1 := 4.805272524$

OK, what happened here? We found the third root for the second time, not the first root, and it took quite a while to do it. Why did this happen?

```

> evalf(solve(diff(f,x)=0,x));
3.248477253 - 5. 10-11 I, -0.1232619829 + 2. 10-11 I, 0.2081180623 + 2. 10-11 I

```

The imaginary parts of these numbers are basically 0, so ignore that term. Now do you see what happened? Now let's take .2 as a first approximation.

```

> r1:='r1';
r1 := r1

```

```

> r1:=newton(f, .2, .000001, 100, r1);
i      p      f(p)
--      -      ----
0      .2      .222562088
1      -.2636710167      -1.587838426+3.141592654*I
2      -.1137123127-.2966984268*I      2.128578817-4.117977869*I
3      .778512295e-1-.926695273e-1*I      .166244705-.8650683595*I
4      .1397311354-.4336051349e-1*I      .151993473-.1984541771*I
5      .1425870390+.196697011e-2*I      .85098862e-1+.895618743e-2*I
6      .1239921560-.689382881e-3*I      -.15392297e-1-.4336041648e-2*I
7      .1264468332-.274424889e-4*I      -.281912e-3-.1658199867e-3*I
8      .1264935008-.2108940e-7*I      -.71e-7-.1273338270e-6*I
9      .1264935126-.407e-14*I      0.-.2457388900e-13*I

```

The approximate solution is $r1 = .1264935126-.407e-14*I$
with $f(r1) = 0.-.2457388900e-13*I$

$r1 := 0.1264935126 - 4.07 10^{-15} I$

Again, there is a complex part which is basically 0. Ignore it, and we have the first approximation to the desired degree of accuracy. Now let's start with .12 as a first approximation.

```

> r1:='r1';
r1 := r1

```

```

> r1:=newton(f, .12, .000001, 100, r1);
i      p      f(p)
--      -      ----
0      .12      -.41351536e-1
1      .1261662263      -.1981425e-2
2      .1264926330      -.5311e-5
3      .1264935126      0.

```

The approximate solution is $r1 = .1264935126$
with $f(r1) = 0.$

$r1 := 0.1264935126$

Again, three steps instead of 16 when we use a good first approximation. Now let's find a square root of the complex number $3 + 2I$.

```
> g:=x^2-(3+2*I);
```

$$g := x^2 - 3 - 2I$$

Let's use $1 + 2I$ as a first approximation.

```
> r:=newton(g,1+2*I,.000001,100,r);
```

i	p	f(p)
0	1+2*I	-6.+2.*I
1	1.200000000+.600000000*I	-1.920000000-.560000000*I
2	1.933333333+.4666666667*I	.519999999-.195555556*I
3	1.817790262+.5451310862*I	.7193535e-2-.18132040e-1*I
4	1.817347117+.5502513645*I	-.26020e-4-.4538e-5*I
5	1.817354021+.5502505227*I	0.+0.*I
6	1.817354021+.5502505227*I	0.+0.*I

The approximate solution is $r = 1.817354021 + .5502505227I$
with $f(r) = 0.+0.*I$

$$r := 1.817354021 + 0.5502505227I$$

We check that this is really a square root.

```
> r^2;
```

$$3.000000000 + 2.000000000I$$

So **newton** works for complex functions and roots as well as real functions and roots.

NumericalAnalysis

There is a procedure called [Newton](#) in the **NumericalAnalysis** subpackage of the **Student** package. But this procedure only works for real roots. We use it to find the first 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]
```

```
> Newton(f,x=.12,stoppingcriterion=absolute,tolerance=10^(-6),output=  
information,maxiterations=100);
```

n	p_n	absolute error
0	0.12	-
1	0.1261662263	0.0061662263
2	0.1264926330	0.0003264067
3	0.1264935126	$8.796 \cdot 10^{-7}$

> Newton(f,x=.12, stoppingcriterion=absolute, tolerance=10⁻⁶, output=plot, maxiterations=100);

3 iterations of Newton's method applied to
 $f(x) = 4x^3 - 20x^2 + 3x + 2 + \ln(x)$
 with initial point $p_0 = 0.12$

