9 Root finding

Functions:

scrsho: graph a function roughly
zbrac: search for brackets on a root of a function
zbrak: search for brackets on a root of a function
rtbis: find a root by bisection
rtlsp: find a root by false position
rtsec: find a root by the secant method
zbrent: find a root by Brent's method
rtnewt: find a root by Newton-Raphson
rtsafe: find a root by Newton-Raphson and bisection
laguer: root of a polynomial by Laguerre's method
zroots: roots of a polynomial by Laguerre's method with deflation
qroot: root of a real or complex polynomial by Bairstow's method
mnewt: nonlinear system of equations by Newton-Raphson

scrsho

Description:

This function plots a given function roughly on the ascii screen to assist with a search for roots. The user is prompted for the endpoints of the interval on which the function is to be plotted.

scscho(fx) ⟶ done

Where:

fx is the name of a Senac function of one real variable,
done is a symbol representing that a simple plot of fx on the interval *int* has been completed.

Pre-variables:

scrsho_iscr [60] the number of horizontal positions in the display,
scrsho_jscr [21] the number of vertical positions in the display.

Example:

The first root of the bessel function `bessj0` is located approximately in the interval [0,4].

```
-> scrsho(bessj0)?
Enter x1: 0.0
Enter x2 with x2 > x1 or x1 = x2 to stop: 4.0
   1.00     x-----------------------------------------------------------l
            x-----------------------------------------------------------l
            lxxxxxxx                                                    l
            l       xxxx                                                l
            l           xx                                              l
            l             xxx                                           l
            l                xx                                         l
            l                  xx                                       l
            l                    xx                                     l
            l                      xx                                   l
            l                        x                                  l
            l                         xx                                l
            l                           xx                              l
            l                             xx                            l
            l                               xx                          l
            l                                 x                         l
            -----------------------------------xx------------------------
            l                                    xxx                    l
            l                                       xx                  l
            l                                         xxx               l
            l                                            xxx            l
 -0.403     x-----------------------------------------------------------l
             0.00                                              4.00
```

zbrac

Description:

This function searches for an interval to bracket a root of a function of one variable.

$$\textbf{zbrac(func, int)} \longrightarrow \textbf{[success, int]}$$

Where:

func is the name of a Senac function of one real variable,
int is a real interval of the form $[x1, x2]$ which on input is the interval in which a root is sought and on output an interval in which a root may have been found,
success is the symbol *true* if a root has been found and *false* otherwize.

Pre-variables:

zbrac_factor [1.6d0] a floating point number being the relative increase in length of *int* at each iteration,
zbrac_ntry [50] the number of attempts which are made to bracket a root before the algorithm gives up.

Example:

A zero of the bessel function `bessj0` is located in [1,3.6] and another in [4,6] by `zbrac`.

```
-> zbrac(bessj0,[1.0,2.0])?
[true, [1.000000000, 3.600000000]]
-> zbrac(bessj0,[4.0,6.0])?
[true, [4.000000000, 6.000000000]]
```

zbrak

Description:

This function searches for intervals in which lie the roots of a function of one variable.

zbrak(fx, int, n, nb) ⟶ [xb1, xb2, nb]

Where:

fx is the name of a Senac function of one real variable,
int is a real interval of the form $[x1, x2]$ in which the algorithm searches for roots,
n is the number of subintervals of equal length making up the subdivision of *int*,
nb is on input the maximum number of bracketing subintervals and on output the number of pairs found,
xb1 is a list of real numbers representing the lefthand endpoints of the bracketing subintervals,
xb2 is a list of real numbers representing the righthand endpoints of the bracketing subintervals.

Example:

The bessel function `bessj0` has 16 zeros in the interval [1,50]. This is checked by `zbrak` and the endpoints of bracketing subintervals computed. These are tested by evaluating `bessj0` at each end point and seeing that the signs are different.

```
-> braks : zbrak(bessj0,[1.0,50.0],100,20)?
[[1.980000000, 5.410000000, 8.350000000, 11.290000000, 14.720000000,
17.660000000, 21.090000000, 24.030000000, 27.460000000, 30.400000000,
33.340000000, 36.770000000, 39.710000000, 43.140000000, 46.080000000,
49.020000000, 0.000000000, 0.000000000, 0.000000000, 0.000000000],
[2.470000000, 5.900000000, 8.840000000, 11.780000000, 15.210000000,
18.150000000, 21.580000000, 24.520000000, 27.950000000, 30.890000000,
33.830000000, 37.260000000, 40.200000000, 43.630000000, 46.570000000,
49.510000000, 0.000000000, 0.000000000, 0.000000000, 0.000000000], 16]
-> Test brackets: #
-> braks1 : first(braks); braks2 : proj(braks,2);
-> map(bessj0,braks1)?
[0.235437638, -0.037757845, 0.074721432, -0.109569203, 0.040090554,
-0.073408654, 0.019038299, -0.049978596, 0.003711393, -0.032483245,
0.057396713, -0.018401078, 0.042480661, -0.006556474, 0.029736406,
-0.050339117, 1.000000003, 1.000000003, 1.000000003, 1.000000003]
-> map(bessj0,braks2)?
[-0.033361145, 0.122033357, -0.057175806, 0.002251069, -0.059597627,
0.017341156, -0.063707674, 0.028489215, -0.067743834, 0.037392123,
-0.008441191, 0.044775325, -0.018531717, 0.051008353, -0.027131391,
0.003677938, 1.000000003, 1.000000003, 1.000000003, 1.000000003]
```

rtbis

Description:

This function searches for a root of a function of one variable using the method of bisection.

rtbis(func, int, xacc) $\longrightarrow$ value

Where:

func is the name of a Senac function of one real variable,
int is a real interval of the form $[x1, x2]$ in which the function is known to have a root,
xacc is a real number defining the accuracy to which the root is computed,
value is a floating point number being the value of the computed root.

Pre-variables:

rtbis_jmax [40] is the maximum allowed number of bisections.

Example:

The first root of `bessj0` is determined by first bracketing it with `zbrak` and then using `rtbis` applied to that interval. The interval is used in the following 6 documented functions.

```
-> braks : zbrak(bessj0,[1.0,50.0],100,20);
-> int : [proj(proj(braks,1),1), proj(proj(braks,2),1)]?
[1.980000000, 2.470000000]
-> fund(x) := -1*bessj1(x);
-> Find first root of bessj0: #
-> root : rtbis(bessj0,int,1.0d-6)?
2.404825611
-> Test root: #
-> bessj0(root)?
-0.000000026
```

rtflsp

Description:

This function searches for a root of a function of one variable using the method of false-position.

rtflsp(func, int, xacc) $\longrightarrow$ value

Where:

func is the name of a Senac function of one real variable,
int is a real interval of the form $[x1, x2]$ in which the function is known to have a root,
xacc is a real number defining the accuracy to which the root is computed,
value is a floating point number being the value of the computed root.

Pre-variables:

rtflsp_maxit [30] is the maximum allowed number of iterations

Example:

The first root of `bessj0` is determined using `rtflsp`.

```
-> int:[1.98, 2.47];
-> root : rtflsp(bessj0,int,1.0d-6)?
2.404825565
-> Test root: #
-> bessj0(root)?
-0.000000002
```

rtsec

Description:

This function searches for a root of a function of one variable using the secant method.

rtsec(func, int, xacc) $\longrightarrow$ value

Where:

func is the name of a Senac function of one real variable,
int is a real interval of the form $[x1, x2]$ in which the function is known to have a root,
xacc is a real number defining the accuracy to which the root is computed,
value is a floating point number being the value of the computed root.

Pre-variables:

maxit [30] is the maximum allowed number of iterations.

Example:

The first root of `bessj0` is determined using `rtsec`.

```
-> int: [1.98, 2.47];
-> root : rtsec(bessj0,int,1.0d-6)?
2.404825560
-> Test root: #
-> bessj0(root)?
-0.000000000
```

zbrent

Description:

This function searches for a root of a function of one variable using Brent's method.

zbrent(func, int, tol) $\longrightarrow$ value

Where:

func is the name of a Senac function of one real variable,
int is a real interval of the form $[x1, x2]$ in which the function is known to have a root,
tol is a real number defining the accuracy to which the root is computed,
value is a floating point number being the value of the computed root.

Pre-variables:

zbrent_itmax [100] is the maximum allowed number of iterations,
zbrent_eps [3.0d-16] a floating point number representing the machine precision.

Example:

The first root of `bessj0` is determined using `zbrent`.

```
-> int: [1.98, 2.47];
-> root : zbrent(bessj0,int,1.0d-6)?
2.404825579
-> Test root: #
-> bessj0(root)?
-0.000000009
```

rtnewt

Description:

This function searches for a root of a function of one variable using the Newton-Raphson method.

rtnewt(func, fund, int, xacc) $\longrightarrow$ value

Where:

func is the name of a Senac function of one real variable,
fund is the name of a Senac function of one real variable representing the derivative of $func$,
int is a real interval of the form $[x1, x2]$ in which there is a root of $func$,
xacc is a real number being the accuracy with which the root is to be determined,
value is a floating point number being the computed root.

Pre-variables:

rtnewt_maxit [20] the maximum number of iterations.

Example:

The first zero of `bessj0` is determined using `rtnewt`.

```
-> int:[1.98, 2.47];
-> fund(x):=-bessj1(x);
-> root : rtnewt(bessj0,fund,int,1.0d-6)?
2.404825560
-> Test root: #
-> bessj0(root)?
-0.000000000
```

rtsafe

Description:

This function searches for a root of a function of one variable using the Newton-Raphson method together with the bisection method.

rtsafe(func, fund, int, xacc) $\longrightarrow$ value

Where:

func is the name of a Senac function of one real variable,
fund is the name of a Senac function of one real variable representing the derivative of $func$,
int is a real interval of the form $[x1, x2]$ in which there is a root of $func$,
xacc is a real number being the accuracy with which the root is to be determined,
value is a floating point number being the computed root.

Pre-variables:

rtsafe_maxit [20] the maximum number of iterations.

Example:

The first zero of `bessj0` is determined using `rtsafe`.

```
-> int:[1.98, 2.47];
-> fund(x):=-bessj1(x);
-> root : rtsafe(bessj0,fund,int,1.0d-6)?
2.404825560
-> Test root: #
-> bessj0(root)?
-0.000000000
```

laguer

Description:

This function finds a roots of a polynomial in one variable with complex coefficients using Laguer's method.

laguer(ar, ai, x) $\longrightarrow$ x

Where:

ar is a real list representing the real parts of the complex coefficients of a polynomial in increasing order,
ai is a real list having the same length as *ar* representing the imaginary parts of the complex coefficients of a polynomial in increasing order,
x is a real list of the form $[xr, xi]$ representing on input an approximation to the root of the polynomial and on output a computed root.

Pre-variables:

laguer_eps [1.0d-3] a floating point number being the desired fractional accuracy of the root,
laguer_epss [1.0d-16] a floating point number representing the machine accuracy,
laguer_polish [false] a symbol being *true* if root polishing is requested or *false* otherwise,
laguer_maxit [100] the maximum allowed number of iterations.

Example:

A complex root of the polynomial $z^4 - (1+2i)z^2 + 2i$ is found using `laguer`. The root is then checked.

```
-> %i : (-1)^(1/2);
-> Find roots of #
-> f(x) := (first(x) + %i*last(x))^4 - (1+2*%i)*(first(x) + %i*last(x))^2
          + 2*%i;
-> which has roots :
x = 1.0, -1.0, 1+%i, -1-%i #
-> ar : [0,0,-1,0,1];
-> ai : [2,0,-2,0,0];
-> xtrial : [1.0,-1.0];
-> root : laguer(ar,ai,xtrial)?
[1.000000000, 0.000000000]
-> Test root: #
-> eval(expand(f(root)))?
0.000000000
```

zroots

Description:

This function finds all of the roots of a polynomial in one variable with complex coefficients using Laguer's method.

zroots(ar, ai) $\longrightarrow$ **[rootsr, rootsi]**

Where:

ar is a real list representing the real parts of the complex coefficients of a polynomial in increasing order,
ai is a real list having the same length as *ar* representing the imaginary parts of the complex coefficients of a polynomial in increasing order,
rootsr is a floating point list representing the real parts of the computed roots,
rootsi is a floating point list representing the imaginary parts of the computed roots.

Pre-variables:

zroots_eps [1.0d-3] a floating point number being the desired fractional accuracy of the root,
zroots_epss [1.0d-16] a floating point number representing the machine accuracy,
zroots_polish [false] a symbol being *true* if root polishing is requested or *false* otherwise,
zroots_maxit [100] the maximum allowed number of iterations.

Example:

A complex roots of the polynomial $z^4 - (1 + 2i)z^2 + 2i$ are found using `zroots`.

```
-> Take same poly as above for laguer: #
-> roots : zroots(ar,ai)?
[[-1.000000000, -1.000000000, 1.000000000, 1.000000000], [-1.000000000,
0.000000000, 1.000000000, 0.000000000]]
```

qroot

Description:

This function finds a complex or double root of a polynomial with real coefficients in one variable.

$$\mathbf{qroot(p,\ b,\ c,\ eps)} \longrightarrow \mathbf{[b,\ c]}$$

Where:

p is a real list representing the coefficients of a polynomial in increasing order,
b is a real number which on input represents an initial guess for the coefficient of the linear term in a quadratic factor of p and on output the computed coefficient,
c is a real number which on input represents an initial guess for the coefficient of the constant term in a quadratic factor of p and on output the computed coefficient,
eps is a real number specifying the accuracy to which the coefficients are to be computed.

Pre-variables:

qroot_nmax [20] the maximum number of terms in p,
qroot_itmax [20] the maximum number of iterations,
qroot_tiny [1.0d-6] a floating point number representing a small quantity.

Example:

A complex root of a sixth degree polynomial is found and the result tested by division. The remainder is found to be zero.

```
-> f(x) := 10 - 18*x + 25*x^2 - 24*x^3 + 16*x^4 - 6*x^5 + x^6;
-> p : [10.0,-18.0, 25.0, -24.0, 16.0, -6.0, 1.0];
-> btrial : 1.0;
-> ctrial : 1.0;
-> eps : 1.0d-6;
-> cofs : qroot(p,btrial,ctrial,eps)?
[-0.000000000, 1.000000000]
-> Test the root: #
-> b : first(cofs); c : last(cofs);
-> poldiv(p,[c,b,1])?
[[10.000000000, -18.000000000, 15.000000000, -6.000000000, 1.000000000],
[0.000000000, 0.000000000]]
```

mnewt

Description:

This function finds a zero of a system of non-linear equations using the Newton-Raphson algorithm.

mnewt(x, usrfun, jacobian) $\longrightarrow$ **x**

Where:

x is a real list representing an initial guess for a root,
usrfun is a list of the names of Senac functions of many variables of the form $[f_1, f_2, ..., f_n]$ such that the equations to be solved have the form

$$f_j(x_1, ..., x_n) = 0, \quad 1 \leq j \leq n$$

jacobian is an n by n matrix of the names of Senac functions of n variables representing the Jacobian of $usrfun$.

Pre-variables:

mnewt_ntrial [20] the maximum number of Newton-Raphson steps to be taken,
mnewt_tolx [1.0d-6] the sought for maximum for the summed absolute values of the variable increments,
mnewt_tolf [1.0d-6] the sought for maximum for the summed absolute values of the functions value increments.

Example:

A system of 4 non-linear equations in 4 unknowns is solved using `mnewt`. The solution is then tested.

```
-> Equations to be solved :
       -x1^2 - x2^2 - x3^2 + x4       = 0
        x1^2 + x2^2 + x3^2 + x4^2 - 1 = 0
        x1   - x2                     = 0
               x2   - x3              = 0
So the functions for mnewt are : #
-> f1(x1,x2,x3,x4) := -x1^2 - x2^2 - x3^2 + x4;
-> f2(x1,x2,x3,x4) :=  x1^2 + x2^2 + x3^2 + x4^2 - 1.0;
-> f3(x1,x2,x3,x4) :=  x1   - x2;
-> f4(x1,x2,x3,x4) :=         x2   - x3;
-> usrfun : [f1,f2,f3,f4];
-> And the partial derivatives are: #
-> f1x1(x1,x2,x3,x4) := -2.0*x1;
-> f1x2(x1,x2,x3,x4) := -2.0*x2;
-> f1x3(x1,x2,x3,x4) := -2.0*x3;
-> f1x4(x1,x2,x3,x4) :=  1.0;
-> f2x1(x1,x2,x3,x4) := 2.0*x1;
-> f2x2(x1,x2,x3,x4) := 2.0*x2;
-> f2x3(x1,x2,x3,x4) := 2.0*x3;
-> f2x4(x1,x2,x3,x4) := 2.0*x4;
-> f3x1(x1,x2,x3,x4) :=  1.0;
-> f3x2(x1,x2,x3,x4) := -1.0;
-> f3x3(x1,x2,x3,x4) :=  0.0;
-> f3x4(x1,x2,x3,x4) :=  0.0;
-> f4x1(x1,x2,x3,x4) :=  0.0;
-> f4x2(x1,x2,x3,x4) :=  1.0;
-> f4x3(x1,x2,x3,x4) := -1.0;
-> f4x4(x1,x2,x3,x4) :=  0.0;
-> jacobian : matrix([f1x1, f1x2, f1x3, f1x4],
                     [f2x1, f2x2, f2x3, f2x4],
                     [f3x1, f3x2, f3x3, f3x4],
                     [f4x1, f4x2, f4x3, f4x4]);
-> xtrial : [1,1,1,1/3];
-> x : mnewt(xtrial,usrfun,jacobian)?
[0.453884710, 0.453884710, 0.453884710, 0.618033989]
-> Test solution: #
-> x1 : first(x); x2 : proj(x,2); x3 : proj(x,3); x4 : last(x);
-> [f1(x1,x2,x3,x4), f2(x1,x2,x3,x4), f3(x1,x2,x3,x4), f4(x1,x2,x3,x4)]?
[-0.000000000, 0.000000000, 0.000000000, 0.000000000]
```