OPEN SOURCE ENGINEERING

openeering

A SCILAB PROFESSIONAL PARTNER

powered by ENGIN SOFT

SOLVING NONLINEAR SYSTEMS IN SCILAB

Everyday engineers encounter steady-state nonlinear problems in their real-case applications.
In this tutorial we show how nonlinear systems can be easily solved using Scilab.

Level

www.openeering.com

Step 1: Purpose of this tutorial

It is very common in the engineering area to solve steady state nonlinear problems.

Typically, two kinds of nonlinear systems arise:

- Systems with n nonlinear equations in n unknowns.
- Systems with n nonlinear equations in $m < n$ unknown

If the reader is interested in determining the zeros of polynomials, please refer to the help of the main Scilab commands for managing polynomials (e.g. ***roots***, ***poly***, and ***horner***).

$$\begin{cases} f_1(x_1, x_2, \cdots, x_n) = 0 \\ f_3(x_1, x_2, \cdots, x_n) = 0 \\ \vdots \\ f_n(x_1, x_2, \cdots, x_n) = 0 \end{cases}$$

(Systems with n nonlinear equations in n unknowns)

$$\begin{cases} f_1(x_1, x_2, \cdots, x_m) = 0 \\ f_3(x_1, x_2, \cdots, x_m) = 0 \\ \vdots \\ f_n(x_1, x_2, \cdots, x_m) = 0 \end{cases}$$

(Systems with n nonlinear equations in m < n unknowns)

Step 2: Roadmap

In the first part of this tutorial we show how to use the command ***fsolve*** for equations and systems of equations. The command is used for solving systems with exactly the same number of equations and unknowns.

The second part focuses on the use of the command ***lsqrsolve***. In this last part the reader can see how to solve systems with fewer unknowns than equations.

Descriptions	Steps
fsolve	3-7
lsqrsolve	8-10
Exercise	11
Final remarks and references	12-13

Step 3: The fsolve function

The "***fsolve***" function solves systems of n nonlinear equations and n unknowns.

The algorithm characteristics are:

- The ***fsolve*** function is based on the idea of the Newton method;
- It is an iterative method, i.e. it starts from an initial approximation value x_0 and then it performs an iteration, obtaining x_1, and so on;
- Only one solution is found by the command and this solution depends on the initial approximation x_0 (basin of attraction).

Scilab syntax for ***fsolve***:

```
[x [,v [,info]]]=fsolve(x0,fct [,fjac] [,tol])
```

Arguments:

- ***x0***: real vector (initial value of function arguments);
- ***fct***: external (i.e. function or list or string);
- ***fjac***: external (i.e. function or list or string);
- ***tol***: real scalar, precision tolerance: termination occurs when the algorithm estimates that the relative error between x and the solution is at most tol. (`tol=1.d-10` is the default value);
- ***x***: real vector (final value of function argument, estimated zero);
- ***v***: real vector (value of function at x);
- ***info***: termination indicator:
  - 0: improper input parameters;
  - 1: algorithm estimates that the relative error between x and the solution is at most tol;
  - 2: number of calls to fct reached;
  - 3: tol is too small. No further improvement in the approximate solution x is possible;
  - 4: iteration is not making good progress.

For examples and more details see:
http://help.scilab.org/docs/5.3.0/en_US/fsolve.html

Step 4: fsolve example (scalar case)

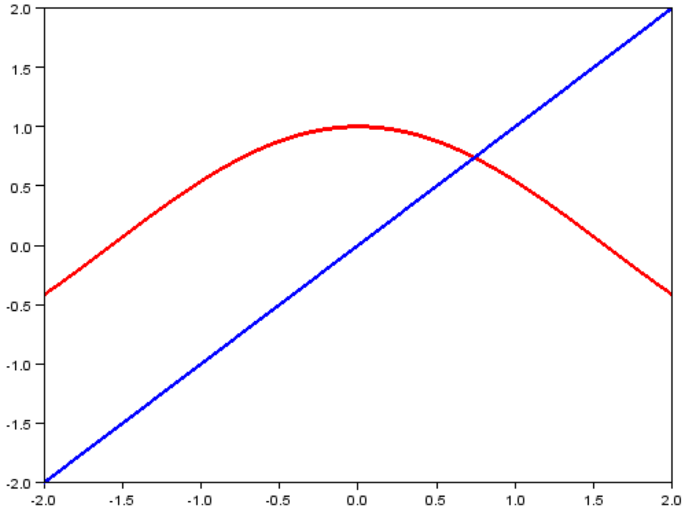
In this example we want to solve the following function;

$$cos(x) = x$$

or, equivalently, to find the zero of

$$f(x) = cos(x) - x$$

The graphical representation is given in the following figure.




```
// Example 1
deff('res=fct_1(x)','res=cos(x)-x')
x0 = 0.
xsol =fsolve(x0,fct_1)

x = linspace(-2,2,51)
fcos = cos(x)
fx = x
scf(1)
clf(1)
plot(x,fcos,'r-');
p = get("hdl"); p.children.thickness = 3;
plot(x,fx,'b-');
p = get("hdl"); p.children.thickness = 3;
```

(Coding of the example)

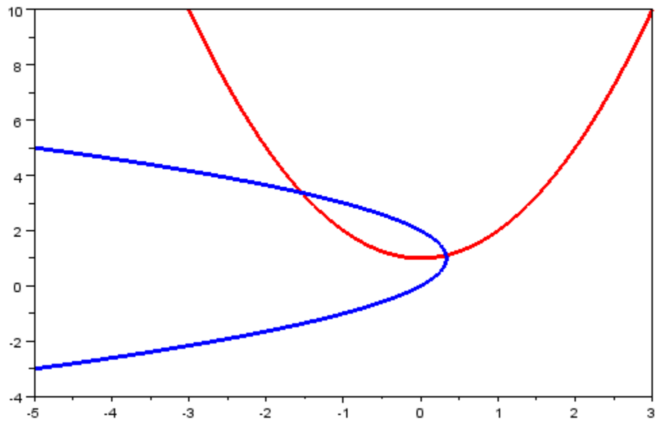
The obtained solution is: xsol = 0.7390851

Step 5: fsolve example (2-dimensional case)

In this example we solve the following 2-dimensional problem

$$\begin{cases} y - x^2 + 1 = 0 \\ x - \dfrac{(2y - y^2)}{3} = 0 \end{cases}$$

The graphical representation is given in the following figure. The figure reveals the presence of two solutions. Depending on the initial point, the function ***fsolve*** can reach the first or the second solution as reported by the example.




```
// Example 2
deff('res=fct_2(x)',['res(1)=x(2)-
(x(1).^2+1)';'res(2)=x(1)-(2*x(2)-x(2).^2)/3'])
scf(2)
clf(2)
x1 = linspace(-3,3,101)
y1 = x1.^2+1

y2 = linspace(-3,5,51)
x2=(2*y2-y2.^2)/3
plot(x1,y1,'r-');
p = get("hdl"); p.children.thickness = 3;
plot(x2,y2,'b-');
p = get("hdl"); p.children.thickness = 3;

x0 = [1;0]
xsol1 =fsolve(x0,fct_2)
res1 = fct_2(xsol1)

x0 = [-3;8]
xsol2 =fsolve(x0,fct_2)
res2 = fct_2(xsol2)
```

(Coding of the example)

The obtained solutions are

- xsol1 = [0.3294085; 1.10851]
- xsol2 = [- 1.5396133; 3.3704093]

with

- res1 = 10^(-14) * [0.1332268; 0.0610623]
- res2 = 10^(-13) * [- 0.3375078; - 0.1310063]

Step 6: fsolve example with embedded solver

In this example we combine the use of the ***fsolve*** function to solve a boundary value problem using the shooting method.

The idea is to embed the Ordinary Differential Equation (ODE) solver (*shooting method*) inside the ***fsolve*** function creating an appropriate function to be solved. This approach is quite general since the ODE solver can be seen as a black box function

The problem under consideration is the following:

$$w''(t) = \frac{3}{2} w^2(t) \text{ with } w(0) = 4 \text{ and } w(1) = 1.$$

This boundary value problem can be reduced to the following initial differential equation problem

$$\begin{cases} \dot{w}_1(t) = w_2(t) \\ \dot{w}_2(t) = \frac{3}{2} w_1^2(t) \end{cases}$$

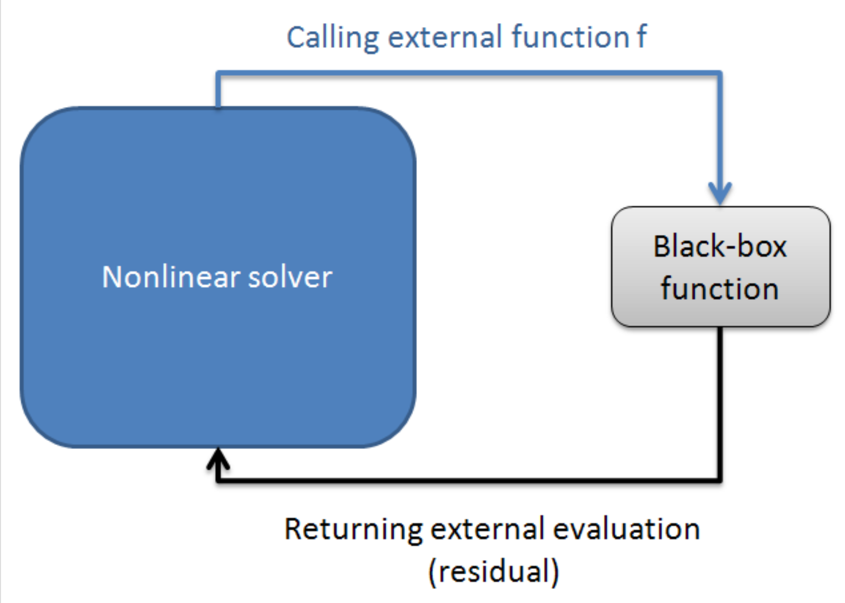
where one of the initial condition in unknown, i.e.

$$\begin{cases} w_1(0) = 4 \\ w_2(0) = s \end{cases}$$

In order to use the ***fsolve*** function we need to introduce the function for which we want to find the zero. In this case, we define the function as

$$f(s) = w_1(1; s) - 1$$

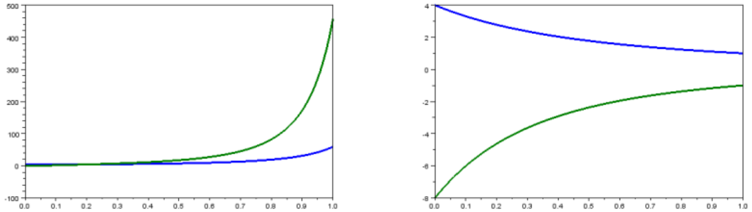
where $w_1(1; s)$ is the solution of the boundary value problem subject to the initial condition that depends on s.



(Example of black-box)

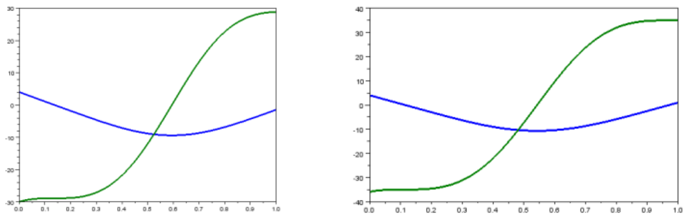
Step 7: Coding and solving

The Scilab code is reported on the right. In the following figures we report the initial solution and the optimal solution.



(Initial solution with $s = 1$ and optimal solution with $s = -7.9999997$)

Changing the initial point s to $s = -30$ it is possible to find a different solution.



(Initial solution with $s = -30$ and optimal solution with $s = -35.858547$)

```
// Example 3
function wdot=wsystem(t, w)
    wdot(1) = w(2)
    wdot(2) = 3/2*w(1).^2
endfunction
s = -1
w0 = [4;s]
t0 = 0
t = linspace(0,1)
// winit = ode(w0,t0,t,wsystem);
// plot(t,winit)

deff('res=fct_3(s)',['w0 = [4;s];','w =
ode(w0,t0,t,wsystem);','res=w(1,$)-1'])
s = -1
ssol =fsolve(s,fct_3)

// compute solution
w0 = [4;ssol]
t0 = 0
t = linspace(0,1)
wsol = ode(w0,t0,t,wsystem);
plot(t,wsol)
p = get("hdl"); p.children.thickness = 3
```

(Example's code)

Step 8: Nonlinear least square fitting

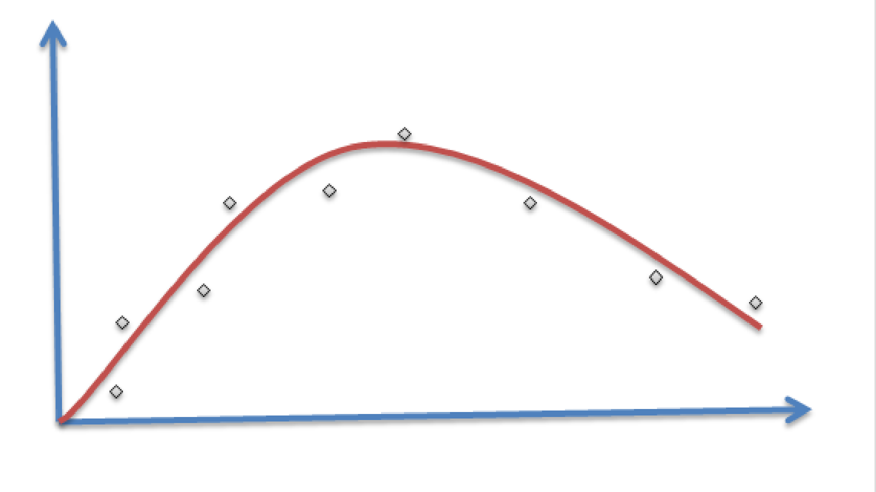
Nonlinear least square is a numerical technique used when we have n nonlinear equations in $m < n$ unknowns. This means that, in these cases, we have more equations than unknowns. This case is very common and interesting and it arises, for example, when we want to fit data with nonlinear (and non polynomial) equations.

The mathematical problem corresponds to find a local minimizer x^* of the following equation

$$F(x) = \frac{1}{2}\sum_{i=1}^{n}(f_i(x))^2$$

where n is the number of points, and f_i represents the residual between the i^{th} given measure data point and the interpolation model (estimated data). Reducing the total sum of residuals corresponds to find the optimal parameters for our fitting model with our given data.

For example, this is particularly useful for experimental data interpolations.



(Example of nonlinear fitting)

Step 9: lsqrsolve

In Scilab the solution of problems in which we have more equations than unknowns is obtained using the ***lsqrsolve*** function.

The algorithm features are:

- The ***lsqrsolve*** function is based on the Levenberg-Marquardt algorithm;
- If the Jacobian is not provided it is calculated by a forward-difference approximation;
- It is an iterative method, i.e. it starts from an initial approximation value x_0 and then it performs an iteration obtaining x_1, and so on;
- Only one solution is found and this solution depends on the initial approximation x_0 (basin of attraction).

Scilab syntax for lsqrsolve:

```
[x [,v [,info]]]=lsqrsolve(x0,fct,m [,stop [,diag]])
[x [,v [,info]]]=lsqrsolve(x0,fct,m ,fjac [,stop [,diag]])
```

Arguments:

- ***x0***: real vector of length `n` (initial value of function argument);
- ***fct***: external (i.e. function or list or string);
- ***m***: integer, the number of functions. `m` must be greater than or equal to `n`;
- ***fjac***: external (i.e. function or list or string);
- ***stop***: optional vector `[ftol,xtol,gtol,maxfev,epsfcn,factor]` the default value is `[1.d-8,1.d-8,1.d-5,1000,0,100]`
- ***diag***: is an array of length `n`. `diag` must contain positive entries that serve as multiplicative scale factors for the variables;
- ***x***: real vector (final estimate of the solution vector);
- ***v***: real vector (value of `fct(x)`);
- ***info***: termination indicator

For examples and more details see:
http://help.scilab.org/docs/5.3.3/en_US/lsqrsolve.html

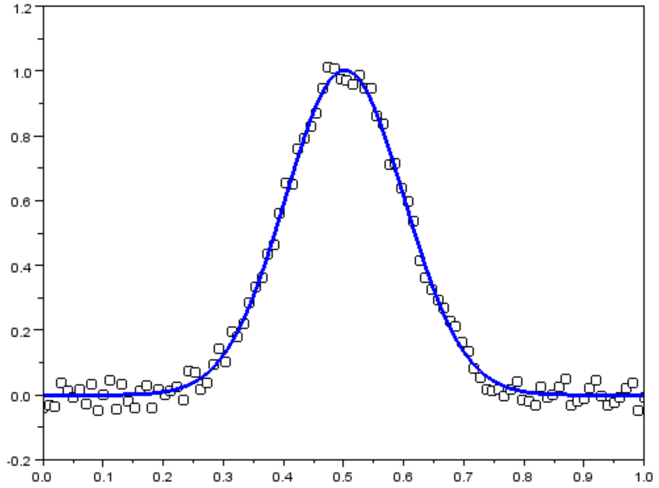
Step 10: lsqrsolve example

In this example we want to estimate the parameter of the following Gaussian function (`fgauss`)

$$y = Ae^{-\frac{(x-x_0)^2}{2\sigma^2}}$$

over a set of 100 data points. The data points are generated starting from a Gaussian distribution, adding a uniform noise. The obtained result is reported in the following figure.

We will have one equation for any point, so the total number of equations is equal to 100. In our code the error function is named `fgausseq`.



(Optimal solution and given data)

```
// Example 4
function y=fgauss(x, A, x0, sigma)
    y = A*exp(-(x-x0).^2/(2*sigma^2))
endfunction

xdata = linspace(0,1)';
A = 1.0;
x0 = 0.5;
sigma = 0.1;
ydata = fgauss(xdata, A, x0, sigma) +
(rand(xdata)-0.5)*0.1;
// plot(xdata,ydata)

function err=fgausseq(param, m)
    A = param(1);
    x0 = param(2);
    sigma = param(3);
    err = ydata -fgauss(xdata, A, x0, sigma);
endfunction

pinit = [0.25; 0.25; 0.25]
[psol,v, info] =
lsqrsolve(pinit,fgausseq,size(xdata,1))
disp(psol)

plot(xdata,ydata,'ko')
p = get("hdl"); p.children.thickness = 3
plot(xdata,fgauss(xdata, psol(1), psol(2),
psol(3)),'b-')
p = get("hdl"); p.children.thickness = 3
```

(Example's code)

Step 11: Exercise #1

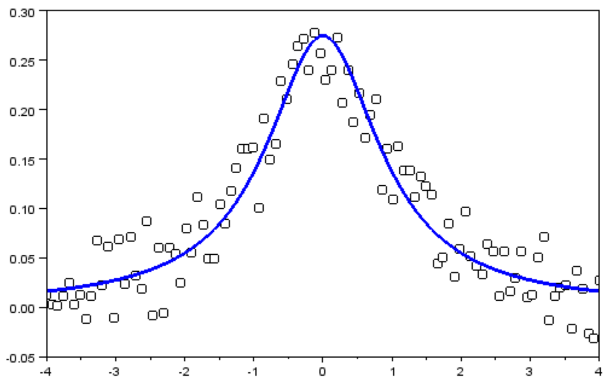
Modify the previous example considering the following distribution (Student's t-distribution)

$$f(x) = \frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\sqrt{\nu\pi}\,\Gamma\left(\frac{\nu}{2}\right)}\left(1+\frac{t^2}{\nu}\right)^{-\frac{\nu+1}{2}}$$

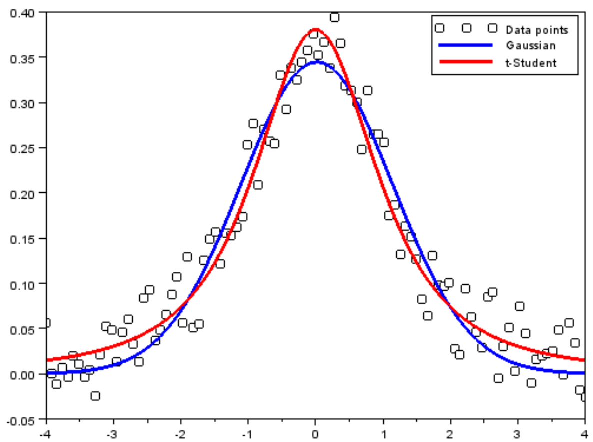
Here, we are interested in estimating ν from measured data.

Compare t-student and Gaussian distributions.

Hints: The gamma function in Scilab is ***gamma***.



(Initial distribution)



(Comparisons between two different distributions)

Step 12: Concluding remarks and References

In this tutorial we have shown how to solve nonlinear problems in Scilab. We described the use of both the ***fsolve*** and ***lsqrsolve*** functions.

On the right-hand column you may find a list of interesting references for further studies.

1. Scilab Web Page: www.scilab.org
2. Openeering: www.openeering.com
3. http://help.scilab.org/docs/5.3.0/en_US/fsolve.html
4. http://en.wikipedia.org/wiki/Shooting_method
5. http://help.scilab.org/docs/5.3.3/en_US/lsqrsolve.html

Step 13: Software content

To report bugs or suggest improvements please contact the Openeering team at www.openeering.com.

Thank you for reading,

Manolo Venturin and Silvia Poles

```
---------------------------
Nonlinear systems in Scilab
---------------------------

--------------
Main directory
--------------
ex_nonlinear.sce             : All the examples
ex1.sce                      : Solution of exercise 1
license.txt                  : The license file
```