Nonlinear Programming on TI-89

Yanchao Liu
University of Wisconsin-Madison
yliu67@wisc.edu

August 21, 2010

nfmin() implements line search methods to solve $\min f(x)$, $x \in \mathrm{R}^n$ on TI-89 calculator, provided f is a convex function. Iff is concave, the solution will be the maximum point of f. If f exhibit neither convex nor concave on R^n, then a stationary point (where the gradient of f equals 0) will be returned if found.

The following five files must be copied and placed in the same folder on the calculator in order for the program to work properly.

Table 1: List of files

File	Type and Size on TI-89
eval.89f	FUNC 290
grad.89f	FUNC 46
hessian.89f	FUNC 182
nfmin.89p	PRGM 2436
stepsize.89f	FUNC 1404

nfmin() is the main program. Upon its execution, a dialog box shows up (see Figure 1 (a)(b)). The first field is a dropdown list with three method options, namely, *FastNewton*, *Newton* and *SteepDecent*. *SteepDecent* stands for the steepest decent method in which the search direction is always the negative of the gradient. *Newton* represents the Newton's method with automatic Hessian corrections (to make sure the search direction is a decent direction). *FastNewton* is the same as *Newton* except that it circumvents the stepsize routine by always using the unit step size, therefore saving search time. If the objective function is quadratic and convex (i.e. the highest order of variables is 2 and the Hessian matrix is positive definite), such like $f(x, y) = 3x^2 + xy + y^2$, *FastNewton* is guaranteed to find the solution in one iteration. In many other cases, *FastNewton* may also be the best choice.

The second field specifies the objective function (the function to be minimized). Users can either type in the expression directly in the text field (as shown in Figure 1 (c)), or type the name of the expression (or function) which the user pre-defines prior to running the program. For example, if the user has defined an expression with **x^2-3x+y^2→f** (Note:

user inputs are presented in boldface, same below) and wants to minimize this expression with respect to x,y, she could just type in **f** in the Function field. Or if the user has defined the function using **Define f(x,y) = x^2-3x+y^2**, then she has to type in **f(x,y)** in the Function field.

The third field specifies the variable list, for example, **{x,y}**, or **{x1,x2,x3}**, or **{x}** if the function involves only one variable x.

The fourth field specifies the initial point in a list, such as **{-1,1.2}** or **{3,0,5}** or **{0.5}**, etc. Note that the values in this list corresponds to the elements in the variable list in the same order.

The last field gives parameters. There are five elements in this list, i.e. {*maxit,toler,beta,c1, c2,step*}. *maxit* is the maximum number of iterations and the default value is 20. *toler* is the precision level, the program terminates (optimality) when the norm of the gradient of the objective function is less than *toler*. The default value of *toler* is 1E-5. *beta* is the minimum (positive) value to be added to the diagonal entries of the Hessian matrix when the Hessian is not positive definite, with 0.5 as the default value. *c1* and *c2*, with $0 < c1 < c2 < 1$, are the parameters to be used in the Wolfe conditions for step size determination. *step* is the trial step size of the line search, with a default value of 1. Note that if *FastNewton* method is used, *step* must be 1.

Once done with this dialog box, the user should press ENTER and the program will start to run. Since minimizing a nonlinear function (regardless of the method used) essentially involves seeking a stationary point (where the gradient of the function converges to 0), the norm of the gradient should ideally keep decreasing iteration by iteration until it reaches *toler*. The norm of gradient will be updated on the screen during the run. If the user finds the norm goes up along the iterations, she might want to break the run by pressing ON, revise the parameters and try again. Parameters that could make an impact include the starting point and the *step*. If *SteepDecent* diverges, it is especially useful to shrink the *step*, to 0.1 for example.

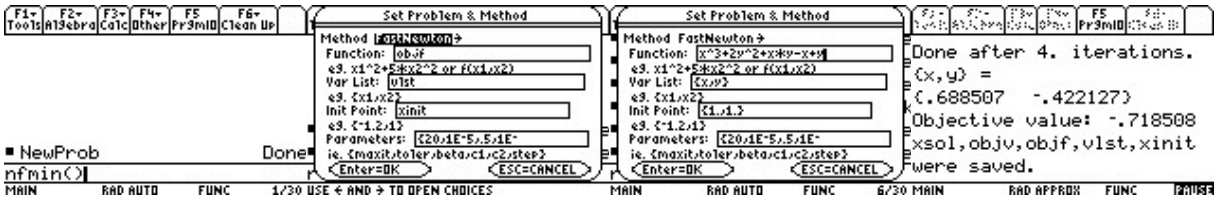
If a solution is found, the program will terminate and display the results. Several variables that contain the problem information will be saved to the same folder as nfmin() is in (See Figure 1 (d)). Specifically, *xsol* stores the solution point in a list format; *objv* is the objective value; *objf* stores the objective expression; *vlst* is the variable list; and *xinit* stores the starting point of the search which the user specifies upfront. These values will not be saved (updated) if the run is not successful. Note also that these variables are auxiliary and the users are free to delete them as they please.

The use of these saved values is not only in the ease of review, but when the next time nfmin() is run, they will be retrieved and set as default values in the dialog text fields, intending to save the user some labor.

The other files in the package, as can be seen from Table 1, are functions written in TI-Basic. Apart from facilitating nfmin(), they are also quite useful on their own.

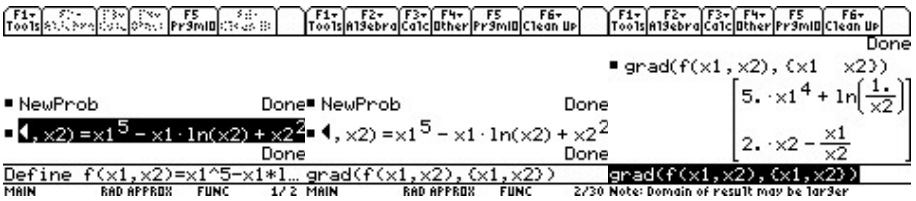
grad(*f,vars*) returns the gradient of a function (or expression) f with respect to the variables in the *vars* list. Note that *vars* must be a list (enclosed by {}) even if there is only one variable. See Figure 2 for an example.

eval(*expr,syms,vals*) evaluates the expression in *expr* with the symbols in *syms* taking the values in *vals*. *syms* and *vals* must be lists with the same dimension. See the example in Figure 3 (a)(b) (continued from Figure 2).



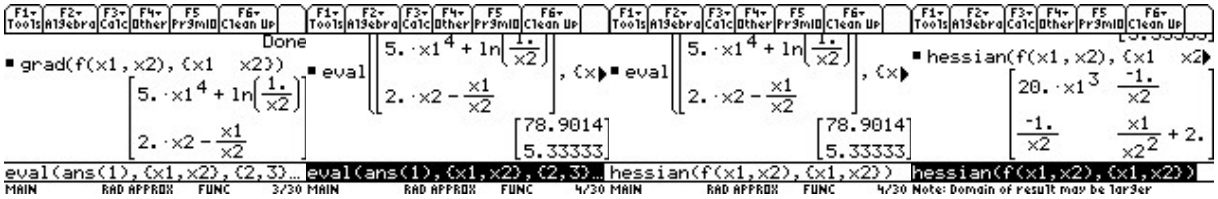
(a) (b) (c) (d)

Figure 1: Demonstration of nfmin()



(a) (b) (c)

Figure 2: Demonstration of grad(f,vars)



(a) (b) (c) (d)

Figure 3: Demonstration of eval(expr,syms,vals) and hessian(f,vars)

hessian(*f*,*vars*) returns the Hessian matrix (second order derivatives) of the function f with respect to *vars*. See Figure 3 (c)(d) for an example.

stepsize(many parameters here) implements the Wolfe conditions to find a suitable step size for line search methods. There is not much use of it as a stand-alone function thus the explanation is omitted.

I hope this package could be a useful addition to your TI-89's software collection. Enjoy!