

Introduction to Function Optimization

Tyler Ransom
Duke University

July 14, 2012

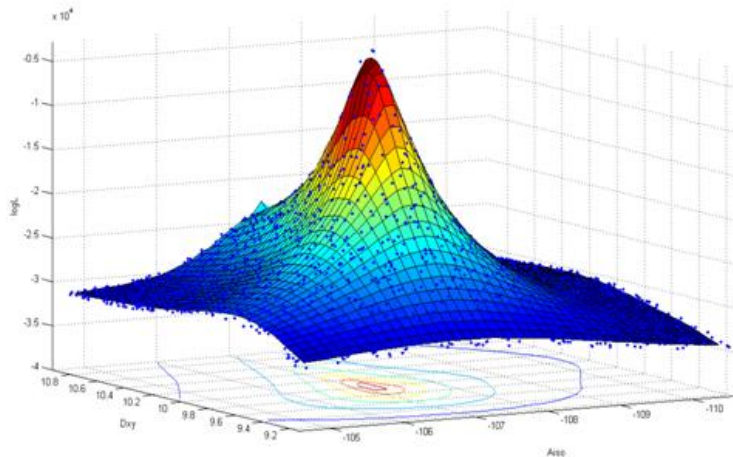
Outline

- Functional Optimization Basics
- Optimizing Functions in Matlab
- Compare/Contrast `fminsearch`, `fminunc`, and `fmincon`
- Discuss `fsolve`
- Determining if your optimizer has given a good solution
- Debugging Programs in Matlab

Basics of functional optimization

- All econometric estimators are either minima of some error function or maxima of some likelihood function
- Some estimators (e.g. OLS, GMM) have **closed form** solutions for the extremum value
- Other estimators (e.g. MLE, some NLLS) require **numerical methods** to find the extremum value
- Matlab has a number of different algorithms to numerically find extrema

Graphical depiction of the optimization problem



Source: <http://earlelab.rit.albany.edu/images.html>

How numerical optimization works

- User provides an objective function and a starting value
- The optimizer then calculates the magnitude and direction of the gradient vector at the starting value
- Given these values of the gradient vector, it decides
 - 1 in which direction to move to get closer to the optimum
 - 2 how far it should move to get closer to the optimum
- Given the magnitude and direction of the gradient vector at the new point, it repeats these steps until convergence is achieved

Determining convergence

There are generally three ways to determine if a numerical optimization algorithm has reached the optimum

- ① The gradient vector (or Jacobian matrix) is numerically close zero
 - Intuition: $f'(x) = 0 \Rightarrow x$ may be an extremum
- ② The location of the previous guess is numerically close to the location of the updated guess
 - Intuition: The optimizer didn't move very far in updating the guess, so we're probably close to the answer
- ③ The value of the objective function at the previous guess is numerically close to the value of the objective function at the updated guess
 - Intuition: The objective function is flat in the area of the previous and updated guesses, which is kind of like having a gradient of zero

Note that “numerically close” generally refers to being within 10^{-6}

Is the extremum global?

- Note that in the previous slide, no mention of second-order conditions was ever made
- Hence, it is unclear if the answer our optimizer gave is a local optimum, a saddle point, or a global optimum
- Luckily, most common econometric estimators have been proven to have globally concave objective functions
- If you are unsure whether or not your objective function is globally concave, you can try different sets of starting values and see how much the answers differ

Optimization functions in Matlab

- All of Matlab's optimizers find *minima*. If a user wants to find the maximum, she must minimize the *negative* of her objective function
- The three most common optimizers are

```
1 fminsearch %Find minimum of unconstrained multivariable function using ...  
   derivative-free method  
2 fminunc    %Find minimum of unconstrained nonlinear multivariable function  
3 fmincon    %Find minimum of constrained nonlinear multivariable function
```

`fminunc` is by far the most popular optimization algorithm in Matlab, but we'll discuss the pros and cons to each

How to call an optimizer in Matlab

The syntax for calling an optimizer to estimate a general function is

```
1 [x,fval,exitflag,output] = fminwhatever('fun',x0,options,varargin)
```

where

`x` is the vector of parameter estimates

`fval` is the value of the objective function at the parameter estimates

`exitflag` is an integer displaying the reason optimization stopped

`output` is a structure containing information about the optimization

`fun` is a function m-file (or function handle) that has the objective function as an output and the parameter vector as an input

`x0` is the starting value for the optimization routine

`options` is a structure of estimation options

`varargin` is whatever other inputs go in to `fun` (e.g. data matrices, rhs variables, other non-estimated parameters)

Example: fminsearch

Let's look at how to estimate OLS on the model $Y = X\beta + \varepsilon$ using `fminsearch`, where β is two-dimensional

- In the script m-file,

```
1 [beta,SSE] = fminsearch('OLS1',.5*ones(2,1),[],X,Y);
```

The function m-file then looks like

```
1 function SSE = OLS1(beta,X,Y)
2 SSE = (Y-X*beta)'*(Y-X*beta);
```

fminsearch optimization options

```
1 Display      %allows user to display iterations
2 FunValCheck  %Check whether objective function values are valid
3 MaxFunEvals  %Maximum number of function evaluations allowed (default is ...
               200*numberOfVariables)
4 MaxIter      %Maximum number of iterations allowed (default value is ...
               200*numberOfVariables)
5 TolFun       %Termination tolerance on the function value (default is 1e-4)
6 TolX        %Termination tolerance on x (default value is 1e-4)
```

Example of how to implement optimization options (optimset command):

```
1 o1 = optimset('Display','iter','MaxFunEvals',1e6,'TolFun',1e-6);
2 [beta,SSE] = fminsearch('OLS1',.5*ones(2,1),o1,X,Y);
```

- Portmanteau of “Functional MINimization UNConstrained”
- `fminunc` is the best optimizer for well-behaved functions
- It has two different optimization algorithms: “medium scale” and “large scale”
- **medium scale:** Solver approximates the gradient and hessian, and uses a quasi-Newton method for searching
- **large scale:** User supplies the analytical gradient and/or hessian, and the solver uses a trust-region algorithm
- The medium scale algorithm is generally OK to use for basic econometric estimators. However, optimization moves much more quickly in the large scale algorithm, so if your objective function is easy enough, you should supply the gradient.

fminunc optimization options

In addition to the options for `fminsearch`, `fminunc` has specific options:

```
1 DerivativeCheck %Compare user-supplied gradient to numerical gradient
2 GradObj        %Use the user-supplied gradient
3 LargeScale     %Use large-scale algorithm
```

- There are many more options, but the ones listed above are the main ones to use
- Use `optimset` to initialize the options in the same way as the `fminsearch` example
- Additionally, `fminunc` has two more outputs than `fminsearch`:

```
1 [x,fval,exitflag,output,gradient,hessian] = ...
   fminunc('fun',x0,options,varargin)
```

The hessian is particularly useful for statistical inference (i.e. std. errors)

fminsearch vs. fminunc

fminsearch

- Simplex search algorithm is more robust than `fminunc`, particularly in the presence of functional abnormalities
- Particularly useful for finding reasonable starting values for `fminunc`
- Runs extremely slowly and inefficiently as the dimension of parameters increases
- Doesn't estimate a hessian, so can't get standard errors on parameter estimates

fminunc

- Uses a line search or trust region algorithm (similar to the 2nd slide of today's lecture) that works very well for well-behaved functions (like those in traditional econometrics)
- Objective function must be continuous
- Starting values need to be reasonable or solution may not be correct
- Can handle any number of parameters
- Estimates a numerical hessian which can be used for inference

fmincon optimization problem

- Portmanteau of “Functional MINimization CONstrained”
- `fmincon` performs optimization of the following form:

$$\min_x f(x) \text{ subject to Constraints}$$

where Constraints is any one or more of

- $c(x) \leq 0$, $c(x)$ is an output of a function m-file which forms the nonlinear constraint function
- $ceq(x) = 0$, $ceq(x)$ is another output of the $c(x)$ function m-file
- $A \cdot x \leq b$, A and b are a matrix and vector, respectively
- $Aeq \cdot x = beq$, Aeq and beq are a matrix and vector, respectively
- $lb \leq x \leq ub$, lb and ub are each vectors the same size as x

fmincon syntax

```
1 [x,fval,exitflag,output,lambda,gradient,hessian] = ...  
    fmincon('fun',x0,A,b,Aeq,beq,lb,ub,'nonlcon',options,varargin of 'fun');
```

- Outputs are same as `fminunc`, with an additional `lambda` which is the Lagrange multipliers from the constraint function
- Note that the ordering of the inputs is quite different from `fminunc`, namely that all of the constraints come after the starting values and before the optimization options (and other inputs to `'fun'`)
- The constraint matrices `A`, `b`, `Aeq`, `beq`, `lb`, `ub` line up with the constraints from the previous slide—but at least one is required
 - Put an empty matrix (`[]`) for any of the `A`, `b`, `Aeq`, `beq`, `lb`, `ub`, `'nonlcon'` you're not using
- `'nonlcon'` refers to $c(x) \leq 0$ or $ceq(x) = 0$ from the previous slide, where c and ceq are outputs of `nonlcon`

fmincon optimization options

- fmincon has four different optimization algorithms (instead of large and medium scale in fminunc)
- Like fminunc, user can provide gradient and hessian to fmincon
- The gradient and hessian are the derivatives of the *Lagrangian*, not just the objective function
- As with fminunc, the more information provided by the user, the faster/better the optimizer solves
- fmincon-specific optimization options:

```
1 GradConstr    %gradient of constraint function; similar to GradObj
2 TolCon        %tolerance on constraint violation (similar to TolX, TolFun)
3 UseParallel   %estimate numerical gradients via parallel computing
```

There are other options specific to each algorithm which we won't go through

fmincon in perspective

- `fmincon` requires at least one type of constraint
- Constrained optimization typically takes longer than unconstrained optimization
- Sometimes it is possible to re-parameterize a constrained problem into becoming an unconstrained one
- Doing so may save computational time, though this savings varies from problem to problem
- You will probably never use `fmincon` (outside of this class) because most standard econometric problems are simple enough for `fminunc` to handle
- At the end of the day, remember that optimization can be more of an art than a science

- Portmanteau of “Function SOLVER”
- `fsolve` is an optimizer for nonlinear systems of equations
- It solves for the vector x such that $f(x) = 0$
- Unlike the other functional optimizers we’ve gone over, `fsolve` requires both inputs and outputs to be vector-valued
- It has three different optimization algorithms: “trust-region-dogleg” (default), “trust-region-reflective”, and “levenberg-marquardt”

fsolve optimization options

`fsolve` has similar options to the other functional optimizers in Matlab:

```
1 DerivativeCheck %Compare user-supplied first derivatives to numerical ...  
   first derivatives  
2 Jacobian       %Use the user-supplied Jacobian (matrix of first ...  
   derivatives)
```

- There are many more options, but the ones listed above are the main ones to use
- Use `optimset` to initialize the options in the same way as the other examples
- `fsolve` has one more output than `fminsearch`:

```
1 [x,fval,exitflag,output,jacobian] = fsolve('fun',x0,options,varargin)
```

The Jacobian can be useful for certain applications

MLE using `fsolve`

- Recall that MLE is just a solution to a system of nonlinear equations (which may not have a closed-form solution)
- We could use `fsolve` to find the solution, where $F(x)$ is the system of first-derivatives
- However, we need the solution to be a maximum, so we would need the `fsolve` Jacobian (the Hessian of the likelihood) to be negative definite
- I can't figure out how to impose this restriction in `fsolve`
- Hence, it's not a good idea in general to use `fsolve` for MLE
- In practice, `fsolve` is commonly used in IO to find FOCs in profit maximization problems, where the FOCs may not have a closed-form solution

Exit Flags

The following table summarizes the exit flags that can be recovered through the third output of a minimizer

Exit flag	Description
-4	Line search can't find a suitable direction
-3	Objective function went below -10^{20}
-2	No feasible point was found
-1	Algorithm was terminated by the output function
0	iterations > MaxIter or funEvals > MaxFunEvals
1	solution found (gradient < TolFun tolerance)
2	Change in x was smaller than the TolX tolerance
3	Change in the Fun value < TolFun tolerance
4	Magnitude of search direction < specified tolerance.
5	Predicted decrease in Fun < TolFun tolerance

Optimization output

The following table summarizes the output that can be recovered through the fourth output of a minimizer

Output	Description
iterations	Number of iterations taken
funcCount	Number of function evaluations
firstorderopt	Measure of first-order optimality
algorithm	Optimization algorithm used
cgiterations	Total number of PCG iterations (large-scale algorithm only)
stepsize	Final displacement in x (only for certain algorithms)
lssteplength	Size of line search step relative to search direction (active-set algorithm only)
constrviolation	Maximum of constraint functions
message	Exit message

Choosing optimization options in a GUI environment

- Matlab has a user-friendly optimization tool (type “`optimtool`” in the command window)
- Users can graphically see all the options available for each optimizer
- Matlab can generate code for the optimization so users don't have to worry about syntax errors

Debugging in Matlab

Recall this diagram illustrating how functions are called in Matlab:

Script m-file (example.m)

```
.  
.   
.   
B=my_function(A)
```

Function m-file (my_function.m)

```
out1=my_function(inp1)  
out1 = (inp1>0);  
end
```

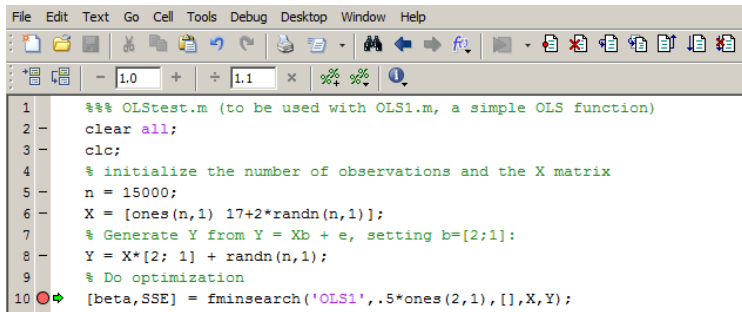
Script m-file (continued)

```
B=my_function(A)  
.   
.   
. 
```

Debugging in Matlab

- Suppose a user wants to try to fix a problem nested deep in a function called by an optimizer
- Recall from the previous slide that the workspace inside a function only includes the inputs passed to the function (and/or global variables)
- Hence, a user needs to get inside the function to identify any problems occurring within the function
- The only problem is telling Matlab to stop at the right spot (inside the function)
- Matlab has a very useful debugging system for accomplishing this task

Debugging in Interactive Mode



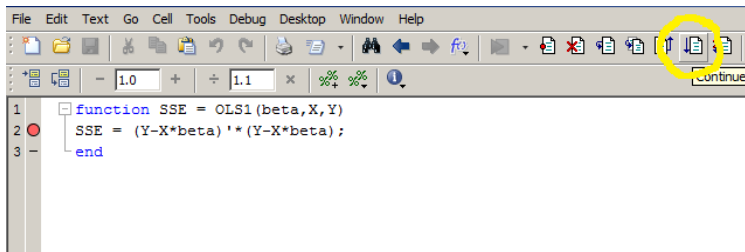
The screenshot shows the MATLAB IDE interface. The menu bar includes File, Edit, Text, Go, Cell, Tools, Debug, Desktop, Window, and Help. The toolbar contains various icons for file operations, editing, and debugging. Below the toolbar is a numeric keypad with fields for values 1.0 and 1.1, and buttons for mathematical operations. The main window displays a script named OLStest.m with the following code:

```
1 %%% OLStest.m (to be used with OLS1.m, a simple OLS function)
2 - clear all;
3 - clc;
4 % initialize the number of observations and the X matrix
5 - n = 15000;
6 - X = [ones(n,1) 17+2*randn(n,1)];
7 % Generate Y from Y = Xb + e, setting b=[2;1]:
8 - Y = X*[2; 1] + randn(n,1);
9 % Do optimization
10 ● ➡ [beta,SSE] = fminsearch('OLS1',.5*ones(2,1),[],X,Y);
```

A red circle with a green arrow (breakpoint icon) is placed on the left margin next to line 10.

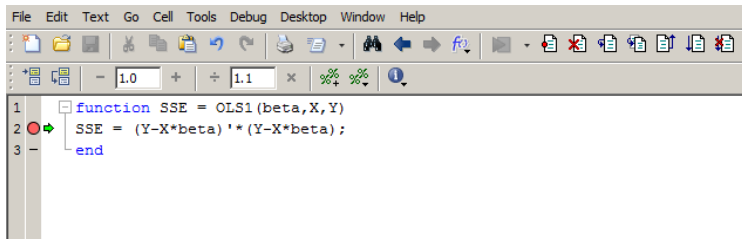
- 1 Begin by placing a breakpoint on the line where the optimization is being called, and press the play button

Debugging in Interactive Mode



- 2 Next, place a breakpoint somewhere in your function m-file, and click “continue”

Debugging in Interactive Mode



- ③ Matlab has stopped on the first line of the function. The user can now examine the workspace to identify inconsistencies
- ④ To quit debug mode, click on the far right icon (to the right of “continue”)
- ⑤ To clear breakpoints, click on the other icon with the red x

Debugging commands

- Debugging is much easier to do in interactive mode, but can also be done from the command line (i.e. in batch mode) using the following commands:

```
1  dbstop in [file] at [line number] %sets a debug break point at the ...  
    specified location  
2  dbcont    %continues execution from the previous breakpoint until the next  
3  dbquit    %quits debug mode  
4  dbclear   %clear breakpoints in certain (or all) files  
5  keyboard  %quick-and-dirty way of debugging; hands control over to the ...  
    keyboard
```

Debugging commands

- The sequence of debugging shown in interactive mode can be accomplished equally from the command line through this sequence of debug commands:

```
1  dbstop in OLStest at 10  %set first breakpoint
2  OLStest                  %execute the script
3  dbstop in OLS1 at 2      %set second breakpoint
4  dbcont                   %continue
5  dbquit                   %exit debug mode
6  dbclear all              %clear all breakpoints
```