CODERS' CORNER

CSAE CENTRE FOR THE STUDY OF AFRICAN ECONOMIES

UNIVERSITY OF OXFORD

HOW TO SOLVE CONSTRAINED OPTIMISATION PROBLEMS USING MATLAB

In this post, I'll explain how to set up a constrained optimisation problem using MATLAB. We'll be working with three files: the main file (**fmincon_main_file.m**) and two function files defining the objective function and the constraint (**mycost.m** and **myprod.m**).

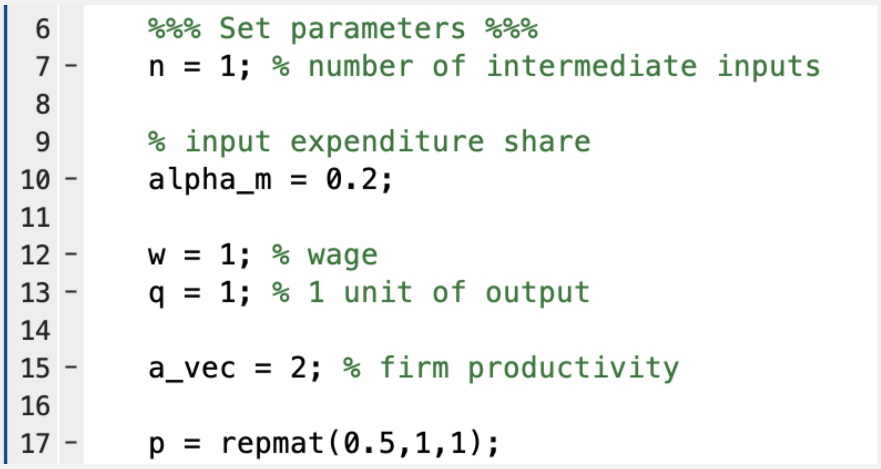
Setting up the main file

[see fmincon_main_file.m]

Take a simple cost minimisation problem example. Consider a firm that chooses labour, L, and intermediate input, X, to minimise its costs of production subject to some given level of output. We assume the firm has a Cobb-Douglas production function, operates under constant-returns-to-scale and Hicks-neutral productivity. The minimisation problem is:

min $C = wL + pX$, subject to $Y = AL^{\beta}X^{\alpha},\ \alpha + \beta = 1$

In the fmincon_main_file.m, we specify the parameters of our optimisation problem. We can vary the number of inputs in addition to labour in the model and code. At the moment, the number of inputs in addition to labour is 1.



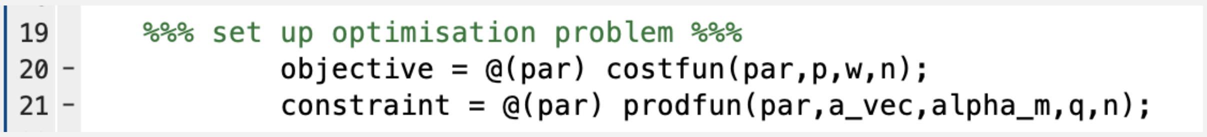
```
 6        %%% Set parameters %%%
 7 -      n = 1; % number of intermediate inputs
 8
 9        % input expenditure share
10 -      alpha_m = 0.2;
11
12 -      w = 1; % wage
13 -      q = 1; % 1 unit of output
14
15 -      a_vec = 2; % firm productivity
16
17 -      p = repmat(0.5,1,1);
```

We set the input expenditure share to be 0.2, and take the wage as the numeraire, setting it equal to 1. We also set the firm's Hicks-neutral productivity scalar equal to 2 here.

We also specify the price of the intermediate input used as 0.5. I use the repmat command on line 20 to illustrate that in cases where we may want to specify multiple prices for multiple inputs, the last two numeric inputs correspond to the number of rows or columns we want the vector/matrix to have.

Once we have set our parameters, we can specify the cost minimisation problem, where our objective function is the cost function, and the constraint, is the given level of output we produce using the (Cobb-Douglas) production function.

I have created the objective function and constraint in separate .m files. Before we proceed to solve this cost minimisation using the fmincon solver, we can set up the optimisation, as below, and specify the parameter inputs each function will take, with the @(par) taking the vector of variables over which we minimise this cost minimisation problem.

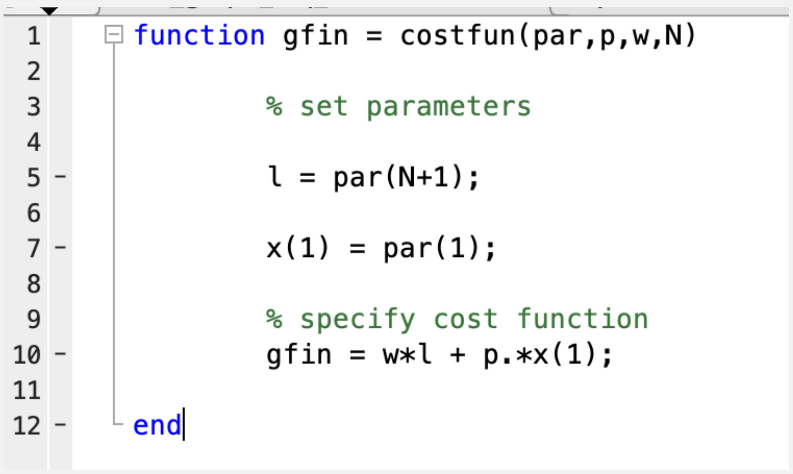


```
    %%% set up optimisation problem %%%
            objective = @(par) costfun(par,p,w,n);
            constraint = @(par) prodfun(par,a_vec,alpha_m,q,n);
```

Defining the objective function

[see mycost.m]

We now define a function setting up the cost function in mycost.m. It is important to note, you must always save your function file with the same name as your function, otherwise MATLAB will give you an error when you run the main file.



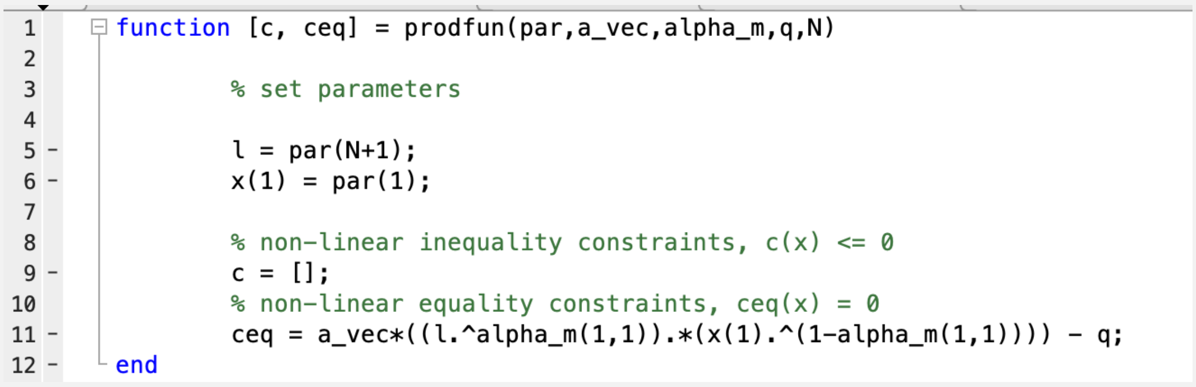
```
function gfin = costfun(par,p,w,N)

        % set parameters

        l = par(N+1);

        x(1) = par(1);

        % specify cost function
        gfin = w*l + p.*x(1);

end
```

The parameters over which MATLAB minimises costs are labour and our only intermediate input X. We need to tell MATLAB these are the two inputs we are minimising over by equating each to par(), the vector of parameters. The last line of the .m file defines the linear cost function the firm will minimise.

Defining the production function

[see mycost.m]

We specify the production function in the myprod.m file. We will first set the parameters in the same way we defined them in the mycost.m file.



```
function [c, ceq] = prodfun(par,a_vec,alpha_m,q,N)

        % set parameters

        l = par(N+1);
        x(1) = par(1);

        % non-linear inequality constraints, c(x) <= 0
        c = [];
        % non-linear equality constraints, ceq(x) = 0
        ceq = a_vec*((l.^alpha_m(1,1)).*(x(1).^(1-alpha_m(1,1)))) - q;
end
```

Then, we need to specify either a non-linear inequality or equality constraint depending on the nature of the problem. In this case, we have a non-linear equality constraint where the firm wants to produce exactly q units of output. We need to rearrange our constraint such that the right-hand side is equal to zero.

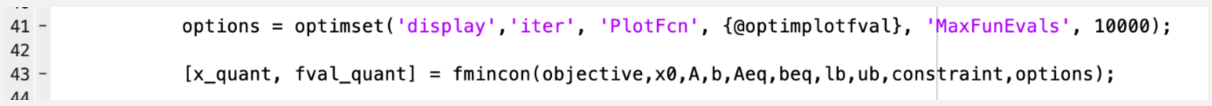
Using fmincon to minimise the constrained optimisation problem

[see fmincon_main_file.m]

fmincon is a useful optimiser when we want to find the minimum of a constrained non-linear function. If your problem does not have a constraint, alternative optimisers such as fminsearch may be better suited to your purposes.

fmincon requires us to set lower and upper bounds for each input, as well as to specify initial values from which the solver will start its search for the combination of labour and intermediate input, x, that will minimise costs.

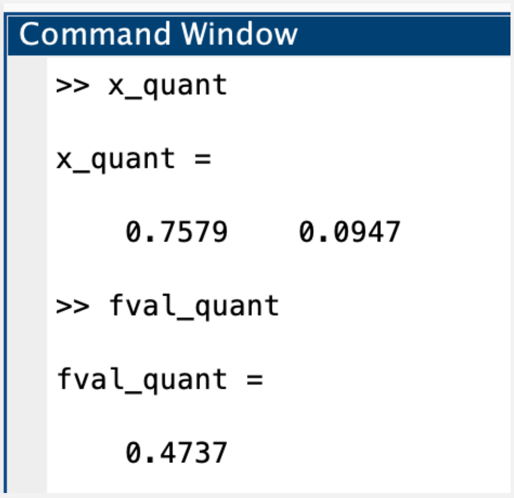
You can change the fmincon options using optimset. With the current options, MATLAB plots the function value against the number of iterations. We can use optimset to specify the maximum number of function evaluations to be 10,000.



```
            options = optimset('display','iter', 'PlotFcn', {@optimplotfval}, 'MaxFunEvals', 10000);

            [x_quant, fval_quant] = fmincon(objective,x0,A,b,Aeq,beq,lb,ub,constraint,options);
```

We specify the objective and constraint, the initial starting values, lower/upper bounds, and constraints. In square brackets, we tell MATLAB to save the optimal quantity of labour and intermediate inputs to the x_quant vector (where the first element will correspond to parameter one we defined in the .m files. In our case, par(1) was the intermediate input, and so the second element, par(2), will be the quantity of labour. The value of the objective function, i.e. the cost of producing one unit of output for this firm, will be saved in the fval_quant vector.

We can inspect the results of interest by running the main file – namely the quantities of labour and the intermediate input the firm purchases, as well as the total associated cost to produce one unit of output. We can see the firm purchases 0.7579 units of the intermediate input and 0.0947 units of labour, at a total cost of 0.4737 units.



```
>> x_quant

x_quant =

    0.7579    0.0947

>> fval_quant

fval_quant =

    0.4737
```

While this is a simple example, MATLAB also allows us to write and solve more complicated models. For example, in my work, I use MATLAB to solve cost minimisation problems for firms who buy and sell inputs on a production network. You can consult this useful resource by Adams, Clarke, and Quinn (2015) in case you're interested in reading more about optimisation problems in MATLAB.

References:

Adams, A., Clarke, D., & Quinn, S. (2015). *Microeconometrics and MATLAB: An Introduction.* Oxford University Press

Diana Beltekian, DPhil candidate in the Economics, University of Nottingham
21 June 2021