Lecture 5

Numerical Optimization

Dr. Ido Filin
`ifilin@univ.haifa.ac.il`

22 November 2012

Outline

Numerical optimization

- We have already seen how to find optimal values using analytic and graphical methods.
- However, a more general approach is to use a set of many different algorithms that seek the optimum **numerically**.
- Numerically means using repeated iterations, usually done by the computer, where in each iteration some approximation of the fitness function is obtained and used to calculate an estimate of the optimal value, which is then inputed as the "initial guess" for the next iteration.
- We repeat that process until **convergence** is obtained (hopefully) – i.e., the estimate of the optimal value no longer changes significantly form one iteration to the next.

Numerical optimization

- In R, there are several packages for doing optimization.
- We will use the package `optimx`.
- We can install the package either through the `install.packages(...)` function, or using the Packages menu of the console window.
- Once the package is installed, we can load it into our current R session, using the function `library(...)`, or again through the Packages menu.
- Note that functions and help on the package and its functions are not available until you have loaded the package into the Current R session.
- Of course, you can use `library(...)` inside your R-script, so the script would do the loading for you.

Defining our own functions

- A function declaration has the following structure

<function name> `<- function(`*<list of argument names>*`)`

```
    {
        <list of R commands>
        ...
        return(...)
    }
```

- Note that this is similar to assignment into variables.
- Example: `funfun <- function( numarg, textarg ) { print(textarg); val <- numarg^2 + numarg; return(val) }`
  Test it by typing
  `print( 1.5 * funfun( 4, "Learning R is fun!" ) )`

Default values of arguments

- We can define default values for arguments.
- We do it using the **=** sign within the function declaration.
- Example:

```
funfun <- function( numarg = 5, textarg = "**
Default text **" )
{ print(textarg); val <- numarg^2 + numarg;
return(val) }
```

- Test it by typing
  1. `funfun( 4, "Learning R is fun!" )`
  2. `funfun( 4 )`
  3. `funfun()`
- If we want to change only the second argument
  1. `funfun( , "R is fun!" )`
  2. `funfun( textarg = "Good morning" )`

Default values of arguments

- Or input them in a different order
  1. `funfun( textarg = "fun fun fun!", numarg = 3)`
- We can use the explicit names of the arguments, as defined in the function declaration, when setting values of arguments during a function call.
- In that case, we don't need to observe the original order of the arguments.
- Example: `foo <- function(x, y, z) {...}`
  The function call `foo( z = 3, x = 1, y = 2 )`
  is identical to the call `foo( 1, 2, 3 )`
  but different than `foo( 3, 1, 2 ).`
- We have already seen this syntax with the `plot` function.

  `plot( x, y, xlab = ..., type = ..., ylab = ...)`

Numerical solution of the annual plant problem

The first step is to define the fitness function.

1. Create a new script: `AnnualPlant.r`
2. Add the following command lines to the script file:

   ```
   1 Fitness <- function( tau, T, b ) {
      fitnessVal <- b * tau * ( T - tau )
      return( fitnessVal ) }
   ```

3. Save and run the script.
4. Now the function `Fitness` has been defined. In the console window, you can call the function with different values of `tau`, `T` and `b`, and observe the results.
5. For example:
   - `Fitness( 3, 10, 0.5 )` $\boxed{\text{Enter}} \longrightarrow$ `10.5`
   - `Fitness( 100, 100, 3 )` $\boxed{\text{Enter}} \longrightarrow$ `0`

Numerical solution of the annual plant problem

1. Add the following command lines to the script file:
    1. `library(optimx)`

    2.
       ```
       Fitness <- function( tau, T, b ) {
        fitnessVal <- b * tau * ( T - tau )
        return( fitnessVal ) }
       ```
    3. `tauInitial <- 1`
    4. `seasonLength <- 10`
    5. `productionParam <- 2`
    6. `controlList <- list( maximize = TRUE )`

    7.
       ```
       sol <- optimx( par = tauInitial, fn = Fitness, T =
       seasonLength, b = productionParam, control =
       controlList, method = "nlm", lower = 0, upper =
       seasonLength )
       ```
    8. `cat( "The optimal value is:\n" )`
    9. `print(sol$par)`
2. Save and run the script.

Numerical solution of the annual plant problem

- The call to `optimx` has the following structure:

<result> `<- optimx( par =` *<initial guess>*,
`fn =` *<function to minimize/maximize>*,
<additional parameters that the function requires>,
`control =` *<optional parameters of algorithms>*,
<additional optional arguments of `optimx`*>* `)`

- Providing the initial guess `par`, and the function to minimize/maximize `fn`, including any additional parameters of that function are the minimum requirements.
- Other arguments of `optimx` (e.g., `control`) are optional, and used depending on what we want `optimx` to do.
- The *<result>* variable is an object of type `list`, and contains all kind of information on the performance of `optimx`
- We can access the estimated optimal value through *<result>*`$par`

Numerical solution of the annual plant problem

- The function to be maximized / minimized must follow a strict format.
- First argument must be the variable to be optimized.
- If there are more than one variable to be optimized, the first argument would be a vector.
- Following arguments are the parameters of the problem.
- For example:

```
fitness <- function( tau, T, b ) { ... }

fitness <- function( V, T, b ) { tau <- V[1]; ... }

fitness <- function( V, T, b )
   { tau <- V[1]; rho <- V[2]; ... }
```

Outline

More advanced plotting

- `plot` has additional arguments to control and modify graphics.

- Modify color with `col`. E.g., `col="black"`, `col = "white"`, `col = "red"`, `col = ""gold4"` etc.

- We will use `"black"`, `"white"` and shades of gray.

  e.g., `"gray0"` (= black), `"gray20"` (= dark gray), `"gray57"` (= light gray), `"gray78"` (= very light gray), `"gray100"` (= white).

- All shades of gray from 0 to 100 exist.

- Color table available in HighLearn.

More advanced plotting

- Modify the symbols with `pch`. This argument gets either a number or a text character within double quotation marks.
  E.g., `pch = 0` (squares), `pch = 1` (circles), `pch = 23` (filled diamond), `pch = "c"` (letter c as symbol), `pch = "7"` (the digit 7 as symbol), `pch = "*"` (asterisk as symbol).

- Explore the different numerical values of pch from 0 to 25 to see the different symbols that are available.

- Help on these optional arguments and great many more is through the help for function `par` (a utility function for manipulating the graphics parameters).

- E.g., `lwd` to change line width (e.g., 1, 2 ,3.7, 4.2), `lty` to change line-type (solid, dashed, dotted etc.).

More advanced plotting

Several curves in one graph.

- `plot` always clears the graphics window before drawing the curve and axes.
- Use the function `lines` to add a second curve or as many additional curves as you need.
- The call to `lines` is very similar to to that of `plot`: e.g., `lines( x, y, pch = ..., col = ..., ...)`
- Some arguments of `plot` (such as `type`) cannot be used.
- As a rule, begin your graph with a call to `plot` that draws the first curve (with its formatting: `pch, col, lwd`, etc.) and defines axis labels, curve type, limits of axes (with `xlim` and `ylim`), etc.
- Then use `lines` repeatedly to add additional curves. Again use `pch, col, lwd` and so on to manipulate the appearance of each additional curve.

Outline

The annual plant problem with mortality

$$R_0 = \int_0^\infty l(a)b(a)da$$

In our original annual plant problem, there is no mortality, i.e.,$l(a) = 1$ for all a within the season, and so:

$$R_0 = \int_0^\tau (1 \cdot 0)\ da + \int_\tau^T (1 \cdot b\tau)\ da + \int_T^\infty 0\ da\ ,$$

the sum of contributions from growth period, reproductive period, and off-season.
It is easy to see that this expression simply gives us

$$R_0 = b\tau(T - \tau)\ ,$$

which is the fitness function that we have used for this problem.

The annual plant problem with mortality

$$R_0 = \int_0^\infty l(a)b(a)da$$

However, if there is a fixed rate of mortality within the season, survival is no longer fixed to 1. In this case $l(a) = e^{-\mu a}$, where μ is the mortality rate parameter. And so the expression for R_0 becomes

$$R_0 = \int_0^\tau (e^{-\mu a} \cdot 0)\ da + \int_\tau^T (e^{-\mu a} \cdot b\tau)\ da + \int_T^\infty 0\ da\ ,$$

We obtain in this case a different expression for fitness,

$$R_0 = e^{-\mu\tau}\left[b\tau \cdot \frac{1}{\mu}\left(1 - e^{-\mu(T-\tau)}\right)\right]\ ,$$

The annual plant problem with mortality

$$R_0(\tau) = e^{-\mu\tau}\left[b\tau \cdot \frac{1}{\mu}\left(1 - e^{-\mu(T-\tau)}\right)\right] \ ,$$

This expression has an intuitive interpretation:

Fitness is given by survival probability to first reproduction multiplied by the the expected mean reproductive output of a reproducing individual (once it survived to first reproduction).
Expected mean reproductive output takes into account that mortality may also occur at any time during the reproductive phase of the life-cycle.

1. Create a different version of the R-script with this fitness function.
2. What would be the effect of mortality on optimal age at first reproduction, τ^* ?