Lecture 4

Trade-offs continued and Numerical Optimization

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

15 November 2012

Outline

Acquisition and allocation

- However, often when measuring trade-offs in the fields or in the lab we encounter positive correlations between life-history traits, rather than negative, as expected from the existence of a trade-off.
- Females that lay more eggs on day 1, also lay more eggs on day 2, 3, . . .
- This is usually a result of variation among individuals in the total amount of resources that they acquired.
- In order to observe the trade-off we first must correct for variation in body size, quality, and total amount of acquired resources among individuals.
- e.g., we must only compare females of the same body size, or remove the effect of body size by statistical analysis.

Consequences of trade-offs to optimization of life-history traits

1. **Constrained optimization**
   Not all combinations of trait-values are possible or achievable.
   e.g., both high current reproduction and high survival may not be possible.

2. Optimization depends on the **shape of a trade-off curve**.
   Whether a trade-off curve is linear, concave-up or concave-down may affect the outcome of optimization – i.e., the optimal trait values.
   In particular, whether the optimum would have intermediate values or extreme values.

Formulating optimization problems

τ = duration of growth period = age at first reproduction.
T = season length.
$T - \tau$ = duration of reproductive investment.
b = rate of increase in biomass production.
fitness = cumulative biomass investment in reproduction.

$$\boxed{f(\tau) = b\tau(T - \tau)}$$

The optimization problem is formulated as:

$$\max_{\tau} f(\tau) \ , \quad \tau \in [0,\ T]$$

Maximize the function $f(\tau)$ over all possible values of τ between 0 and T.

Formulating optimization problems

τ = duration of growth period = age at first reproduction.
T = season length.
ρ = duration of reproductive investment.
b = rate of increase in biomass production.
fitness = cumulative biomass investment in reproduction.

$$\boxed{f(\tau, \rho) = b\tau\rho}$$

The **constrained** optimization problem is formulated as:

$$\max_{\tau,\rho} f(\tau, \rho) \ , \quad \tau, \rho \in [0,\ T]$$

Maximize the function $f(\tau, \rho)$ over all possible values of τ and ρ, subject to the constraint $\tau + \rho - T = 0$.

Consequences of trade-offs to optimization of life-history traits

1. **Constrained optimization**
   Not all combinations of trait-values are possible or achievable.
   e.g., both high current reproduction and high survival may not be possible.

2. Optimization depends on the **shape of a trade-off curve**.
   Whether a trade-off curve is linear, concave-up or concave-down may affect the outcome of optimization – i.e., the optimal trait values.
   In particular, whether the optimum would have intermediate values or extreme values.

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 )` Enter ⟶ `10.5`
   - `Fitness( 100, 100, 3 )` Enter ⟶ `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 )
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`