
How to use the EBT

Contents

1 This practical

The goals of this practical are to learn:

- how to use the ebttool
- how the EBT program works
- how to program the EBT (including some basic C code)
- to analyse output generated by the EBT
- to translate biological problems into (EBT) models

2 The EBT and its code

The general idea of the EBT method (for solving PSPMs) has been explained in the lectures and in the course handbook. One step further is to actually implement a PSPM as a computer program and to study it numerically. To make this step easy, we use the EBT program. It is a program, written in the programming language 'C', which allows you to study PSPMs without worrying too much about the technical problems (i.e., computer and programming related problems). Instead, you can focus on the modelling aspects, which are a lot more interesting (from a biological point of view anyway).

However, in order to do so, you will have to understand some basics of the programming of the EBT method, and of programming in C. That is the first goal of this practical. The second (and ultimate) is to know how to translate a biological problem (concerning a structured population) into an EBT model from scratch.

We start with the first goal, and we use an example. In section B you can find the files 'coho.h' and 'coho.c' which, together, make up the EBT implementation of the coho salmon model as listed in table 1.1 of the course handbook.

The EBT software consists of a set of routines written in C, which take care of:

- reading input necessary for a computation, usually from a file written by you
- doing all the computation of life history and population dynamics
- writing output to file.

The ebttool is graphic tool which gives you easy control over all of these aspects, and which allows you to see the output visually in the form of graphs. Most of the routines of the EBT software are written down in a collection of files hidden away from you (you can find them in places such as /usr/local/escbox). Together with two so-called 'problem files' (i.e., one .c file and one .h file such as coho.c and coho.h) these files can be compiled into an executable computer program. The problem .c file contains routines which specify the biological model, and the .h file (the 'header file') contains some global definitions of the model. These two files are where you implement your models; the goal of this practical. Once you have written the model, you can compile the whole program using the command 'ebt', or using the ebttool. Then in order to run the program, you need two more files. One to set the so-called 'control variables', which are general settings for the ebt program, such as the total time for your computation and the time interval between output. The other contains the initial conditions of the run.

Summarising, there are four files of interest:

- the .c file (e.g. coho.c) with the model
- the .h file (e.g. coho.h) with some global definitions
- the .cvf file (e.g. run.cvf) with settings and parameter values
- the .isf file (e.g. run.isf) with initial conditions

As mentioned above, we start by studying an existing implementation of the coho salmon model. Some of the code is explained in detail in section A and the code itself is given in section B.

3 PROBLEM: coho salmon

- Consider the files coho.h, coho.c, run.cvf and run.isf of the coho salmon model (see sections A and B)
- Find parameter values of D0, D1, D2 and E such that the dynamics correspond to the dynamics of the matrix model with $s_j = 0.2$ and $s_a f = 10$ (see the Excel worksheet). Note that in coho.c the time unit is assumed to be days and that a year is assumed to last YEAR = 100 days.
- Question: why is there no correspondence at the level of the total population, but only (partially) at cohort-level?

4 Output files

The EBT produces different types of output. First of all, the output as you define it yourself in DefineOutput() can be found in the file **run.out**. The first column is the time, followed by column which contain output[0], output[1], etc. The interval between output is determined in the file run.cvf.

Second, the complete state of the system at the end of the run is saved in the file **run.esf**. This file contains information on all environmental variables, and all population variables, at the last moment of the run. (So you can rename this file into run2.isf, for example, to start a new run from this point). The first line lists env[0], env[1], etc. The second line is blank. The third line lists the data for the first cohort. The first column has pop[0][i][number], followed by the i-states, in turn followed by the i-consts (popIDcards). The next lines contain the data of the other cohorts.

Throughout the run, at intervals specified in run.cvf, the program writes such 'complete state' output to the file **run.csb**. This file is binary but can be translated into normal text with the command **csb2txt run.csb** at the command line, or via the ebttool directly. If you write **csb2txt run.csb > run.cso** then the complete state output is saved in the file run.cso.

The file **run.cso** now contains, for each moment of output as defined in run.cvf, information on all environmental variables, and all population variables. The format is the same as for the run.esf file.

Finally, the file **run.rep** contains a report of the run which can be used to reproduce the same run at a later time, for example.

5 PROBLEM: coho salmon with nursery competition

- Extend the coho model to include nursery competition.
- Note that the coho.c model uses days as time unit. The equations for correspondence are therefore:

$$\begin{aligned} s_j &= e^{-\mu_0\tau} \\ s_a &= e^{-\mu_0\tau} \\ f &= E\,e^{-\mu_0\tau} \\ c &= \frac{\mu_1(1-e^{-\mu_0\tau})\,e^{-\mu_0\tau}}{\mu_0}\,E \end{aligned}$$

  where $\tau =$ YEAR.

6 PROBLEM: chinook salmon (age-structured)

- Convert the coho model into a model for chinook salmon, including nursery competition. Parameterise the model such that it has the same population growth rate (without nursery competition) as the matrix $\mathbf{B}$ (equation 1.7 of the course book) with $s_a = s_j = 0.2$, $f = 50$, and the matrix elements $b_{21} = 0.8$ and $b_{32} = 0.25$.
- Check that with the appropriate choice of spawning probabilities $p(a)$, the model gives the same results as your model of coho salmon.

7 PROBLEM: chinook salmon (age + size-structured)

- Implement the age and size-structured model of chinook salmon as described in chapter 2.

- What is the maximum size reached by the chinook? (Do you think these parameter values are very realistic?)
- Does food-dependent growth rate limit the population size?
- An ad hoc (and not very pretty) way to include food-dependent fecundity is to assume that egg production depends on the food level at the moment of reproduction. For the sake of the practical, implement the assumption that
$$E(s, F) = \beta s^3 \frac{F}{F_h + F}$$
- Perhaps a more mechanistic assumption would be that fecundity depends on the amount of food consumed during the entire season. Again, for the sake of education, modify the model to include such accumulation of reproductive potential and implement it in the EBT model.

8 PROBLEM: Daphnia

- Implement the size-structured model of Daphnia as described in chapter 3.
- The function Gradient for the Daphnia model has been printed in section B, which can be used as a help for implementing the model.
- See if you can recreate the dynamics in figure 4.7 of the course book.

A Explanation of the EBT code

Most of the text here is copied from the ebttmpl.h and ebttmpl.c files.

A.1 The header file coho.h

This file contains the settings of the various constants that are necessary to tailor the Escalator Boxcar Train program to the problem under study. The more important entries in the file are explained below.

```
#define POPULATION_NR  1
```

...defines the number of structured populations in the problem.

```
#define I_STATE_DIM  1
#define I_CONST_DIM  1
```

...define the dimension of the i-state and the number of constant variables that characterize a cohort. These dimensions should be the same for all the populations.

```
#define ENVIRON_DIM  1
```

...defines the number of variables characterizing the environment. Here it is 1, because ''time'' is defined as an environmental variable (unnecessarily).

```
#define OUTPUT_VAR_NR  5
```

...defines the number of quantities that have to be written to the output file each time that output should be generated.

```
#define PARAMETER_NR  4
```

...defines the number of free parameters in the problem. These parameters can be changed between various runs without compilation of the program. Parameters that are fixed in the problem can be defined as constants in the problem-specific program file written by the user. Changing of these constants requires a new compilation of the program before use.

A.2 The problem file coho.c

This file contains the user-defined routines of the EBT integration program. So this is where you specify your model.

LABELING ENVIRONMENT AND I-STATE VARIABLES

For convenience it is possible to label the environment and i-state variables with a more meaningful name by defining, for instance, see:

```
#define time  env[0]
#define age   i-state(0)
```

...which define the variables t (E-state) and A_i (i-state). Note the zero-based array indexing in the C-language.

DEFINING AND LABELING CONSTANTS AND PARAMETERS

Define the constant names and values used in the user-specified routines. Most parameters in the problem can be treated as constants for this file. This seems the most easy way to specify the parameters. The parameters that are free to change will be in a vector called "parameter[]", defined elsewhere. As with the i-state variables it is possible to label these parameters with a more meaningful name, e.g.:

```
#define D0     parameter[0]
#define D1     parameter[1]
#define D2     parameter[2]
#define E      parameter[3]

#define YEAR   100.0
#define SMALL  0.01
```

SPECIFICATION OF THE NUMBER AND VALUES OF BOUNDARY POINTS

SetBpointNo and SetBpoints

Newborn individuals enter the population at the so-called ‘‘boundary points’’. Specify here the number of boundary cohorts that should be created at the start of the next cohort integration cycle. Fill the array "bpoint_no[]" with the appropriate integer values. The length of the array is "POPULATION_NR". The state of the environment and the population can be used to adapt the number of the fixed points on the boundary to the current state.

In the function SetBpoints you specify the state of newborn individuals. In the example coho.c:

```
bpoint_no[0]=1;
```

...so there is only one boundary point (all individuals are born with the same state), and

```
bpoints[0][0][age]=0.0;
```

...which means that all individuals are born with age 0.

SPECIFICATION OF DERIVATIVES

Gradient.

Here the derivatives of the i-states, E-states and cohort abundances are specified. Define the derivatives of the various environment variables, which have to be returned to the main program in the array "envgrad[]". Define also the derivatives of the various population variables, which have to be returned to the main program in the matrix of cohort variables "popgrad[][][]" for each population. Finally define the derivatives of the various offspring variables, which have to be returned to the main program in the matrix of cohort variables "ofsgrad[][][]" for each population. NB :

- The integer array "cohort_no[]" is globally available and denotes the number of internal cohorts present in each structured population.
- The integer array "bpoint_no[]" is globally available and denotes the number of boundary cohorts present in each structured population during the current cohort cycle.
- The offspring number can be zero. If used in a division, check for this equality to zero!!

So, to conclude, the goal of this function is to define the derivatives of the state variables of the model. In the coho.c model, these derivatives are:

```
envgrad[0] = 1.0;
popgrad[0][i][number] = -popIDcard[0][i][deathrate]*pop[0][i][number];
popgrad[0][i][age] = 1.0;
```

...and the last two for each cohort i={1,2,3}, ofcourse.

Note the use of the ID cards, which also referred to as i-constants. They are not state variables themselves, but they usually are functions of the state variables. They can come in handy to remember something, such as the age-specific death rate in this case.

SPECIFICATION OF BETWEEN COHORT CYCLE DYNAMICS

InstantDynamics

This routine is called at the end of each cohort cycle. The routine can be used to implement any type of instantaneous dynamics, occurring between two subsequent cohort cycles. It can, for instance, be used for a pulsed, instantaneous reproduction process or to set the number of individuals in cohorts that have reached their maximum lifespan to 0.

(Note that the transformation of boundary cohorts into internal cohorts has already been performed and that the boundary cohorts are hence characterized by the number of individuals and their transformed moments (usually denoted by the symbol μ). In an instantaneous reproduction process the i_state of the offspring can therefore be simply specified in terms of the mean i_state!)

This routine can be used to do discrete reproduction. In the program coho.c reproduction takes place only once per year (= once in YEAR days). The eggs are put into the ofs[0][0] cohort:

```
ofs[0][0][number] = eggs;
ofs[0][0][age] = 0.0;
ofsIDcard[0][0][deathrate] = D0;
```

...and note that in this model adults are killed after reproduction...

SPECIFICATION OF OUTPUT VARIABLES

DefineOutput

Define here the values of the output variables in terms of the population and environment statistics. These values have to be returned to the main program in the array "output[]".

B Example files

```
/***
   NAME
     coho.h
   PURPOSE
     This file contains the settings of the various constants that are
     necessary to tailor the Escalator Boxcar Train program to the problem
     under study.
   NOTES
***/

#define POPULATION_NR  1
#define I_STATE_DIM    1
#define I_CONST_DIM    1
#define ENVIRON_DIM    1
#define OUTPUT_VAR_NR  5
#define PARAMETER_NR   4
#define TIME_METHOD    RKCK




/*==========================================================================*/
```

The file run.cvf (note that the parameter values D0, D1 etc are chosen arbitratily here):

```
"Fixed step size or integration accuracy when adaptive"     1.0E-8
"Cohort cycle time interval"                                1.0
"Tolerance value, determining identity with zero"           1.0E-5
"Maximum integration time"                                         1000.0
"Output time interval"                                      100.0
"Complete state output interval, 0 for none"                100.0
"Minimum allowable number of individuals in cohort"         0.0
"Relative tolerances for i-state variable #0"                      0.0
"Absolute tolerances for i-state variable #0"                      0.0

"D0"                                                               0.1
"D1"                                                               0.2
"D2"                                                               0.3
"E"                                                                1.0
```

The file run.isf:

```
0.0

1.0   100.0    0.0
```

```
/***
   NAME
     coho.c
***/

/*==========================================================================
 *                      INCLUDING THE HEADER FILE
 *==========================================================================
 */
#include  "escbox.h"

/*
 *==========================================================================
 *              LABELLING ENVIRONMENT AND I-STATE VARIABLES
 *==========================================================================
 */

#define time           env[0]
#define age            i_state(0)
#define deathrate      i_const(0)      /* use an ID card for death rate */


/*
 *==========================================================================
 *              DEFINING AND LABELLING CONSTANTS AND PARAMETERS
 *==========================================================================
 */

#define D0             parameter[0]   /* death rates D0, D1 and D2 */
#define D1             parameter[1]
#define D2             parameter[2]
#define E                parameter[3]   /* fecundity */

#define YEAR             100.0          /* assume a year is 100 days */
#define SMALL            0.01

/*
 *==========================================================================
 * USER INITIALIZATION ROUTINE ALLOWS OPERATIONS ON INITIAL POPULATIONS
 *==========================================================================
 */

void    UserInit( int argc, char **argv, double *env,  population *pop)
{

  return;
}

/*
 *==========================================================================
 *      SPECIFICATION OF THE NUMBER AND VALUES OF BOUNDARY POINTS
 *==========================================================================
 */

void    SetBpointNo(double *env, population *pop, int *bpoint_no)
{
  bpoint_no[0]=1;                      /* all newborns have same age! */

  return;
}

/*==========================================================================*/

void    SetBpoints(double *env, population *pop, population *bpoints)
{
  bpoints[0][0][age]=0.0;              /* all newborns have same age! */

  return;
}


/*
 *==========================================================================
 *                      SPECIFICATION OF DERIVATIVES
 *==========================================================================
 */

void    Gradient(double *env,     population *pop,     population *ofs,
                 double *envgrad, population *popgrad, population *ofsgrad,
                 population *bpoints)
{
  register int        i;

  for(i=0; i<cohort_no[0]; i++)                     /* Determine death rates  */
    {
      if(pop[0][i][age]<1.0*YEAR)               /* age 0+ */
        popIDcard[0][i][deathrate] = D0;
      else if(pop[0][i][age]<2.0*YEAR)          /* age 1+ */
        popIDcard[0][i][deathrate] = D1;
      else if(pop[0][i][age]<3.0*YEAR)          /* age 2+ */
        popIDcard[0][i][deathrate] = D2;
    }

  for(i=0; i<cohort_no[0]; i++)                      /* The derivatives for all  */
    {                                          /* internal cohorts       */
      popgrad[0][i][number] = -popIDcard[0][i][deathrate]*pop[0][i][number];
      popgrad[0][i][age]    = 1.0;
    }
  envgrad[0] = 1.0;

  return;
}



/*
 *===========================================================================
 *             SPECIFICATION OF BETWEEN COHORT CYCLE DYNAMICS
 *===========================================================================
 */

void    InstantDynamics(double *env, population *pop, population *ofs)
{
  double                date, eggs;
  register int        i;


  date = fmod(time, YEAR);                /* What is the date? */
  if (date != 0.0)
    return;  /* Return if within season, else opportunity to reproduce */


  /* semelparity: 3 year old individuals reproduce and die */
  for (i=0, eggs=0.0; i<cohort_no[0]; i++)
    if (pop[0][i][age]>(3.0*YEAR-SMALL))
      {
        printf("reproduction at time %f\n",time);
        eggs += E*pop[0][i][number];                /* add all eggs   */
        pop[0][i][number] = 0.0;                    /* all adults die */
      }

  ofs[0][0][number]             = eggs;        /* put eggs into ofs cohort */
  ofs[0][0][age]                = 0.0;         /* specify age and ID card  */
  ofsIDcard[0][0][deathrate]    = D0;

  return;
}



/*
 *===========================================================================
 *                     SPECIFICATION OF OUTPUT VARIABLES
 *===========================================================================
 */

void    DefineOutput(double *env, population *pop, double *output)
{

  double                totpop, age0, age1, age2;
  register int        i;

  for(i=0, totpop=0.0; i<cohort_no[0]; i++)
    {
      totpop += pop[0][i][number];
    }

  /* count individuals in three age classes */
  age0=0.0;
  age1=0.0;
  age2=0.0;
  for(i=0; i<cohort_no[0]; i++)
```

```
    {
      if(pop[0][i][age] < 1.0*YEAR-SMALL)
        age0=pop[0][i][number];
      else if(pop[0][i][age] < 2.0*YEAR-SMALL)
        age1=pop[0][i][number];
      else if(pop[0][i][age] < 3.0*YEAR-SMALL)
        age2=pop[0][i][number];
    }

  output[0] = totpop;
  output[1] = age0;
  output[2] = age1;
  output[3] = age2;
  output[4] = cohort_no[0];

  return;
}
/*==========================================================================*/
```

The function Gradient which may be used as a help for the problem with continuous time reproduction.

```
/*
 *===========================================================================
 *
 *                    SPECIFICATION OF DERIVATIVES
 *
 *===========================================================================
 */

void   Gradient(double *env,     population *pop,     population *ofs,
                double *envgrad, population *popgrad, population *ofsgrad,
                population *bpoints)

{
  double              functional_response;
  double              total_area, mean_offspring_length;
  register int        i;

  functional_response = env[1]/(1+env[1]);   /* Determine S/(1+S)        */

                                             /* Determine the integral   */
  for(i=0, total_area=0; i<cohort_no[0]; i++)        /* of the total area        */
      total_area += pow(pop[0][i][length], 2.0)*pop[0][i][number];

                                             /* Include the boundary     */
  if(ofs[0][0][number] > 1.0E-10)            /* cohort if non-zero       */
    {                                        /* Convert length moment    */
      mean_offspring_length = ofs[0][0][length]/ofs[0][0][number];
      total_area += pow(mean_offspring_length, 2.0)*ofs[0][0][number];
    }
                                             /* The time derivative      */
  envgrad[0] = 1;                            /* The food derivative      */
  envgrad[1] = RM*env[1]*(1-env[1]/K) - functional_response*total_area;

  for(i=0; i<cohort_no[0]; i++)                      /* The derivatives for all  */
    {                                        /* internal cohorts       */
      popgrad[0][i][number] = -DELTA*pop[0][i][number];
      popgrad[0][i][age]    = 1;
      popgrad[0][i][length] = functional_response - pop[0][i][length];
    }
                                             /* The derivatives for the  */
                                             /* boundary cohort        */
  ofsgrad[0][0][number] = -DELTA*ofs[0][0][number]
      + ALPHA*functional_response*total_area;
  ofsgrad[0][0][age]    = -DELTA*ofs[0][0][age]
      + ofs[0][0][number];
  ofsgrad[0][0][length] = -DELTA*ofs[0][0][length]
      + functional_response*ofs[0][0][number]
      - ofs[0][0][length];

  return;
}
```

C Some hints about C code

Here is a very limited list of some useful things to know about C in case you are not familiar with this programming language. (Only things that are relevant to this practical!).

- In general, lines should end with a semicolon (;)
- Everything in between /* and */ is ignored: here you can write your comments about your code for later reference, etc.
- There are local and global variables. Local variables are only valid inside the function where they are defined (see below), but global variables are always valid. In the EBT, cohort_no[0] is an example of a global variable.
- C is based on 'functions', which are routines or procedures in which you can make the computer do something (compute, read, write, sing, etc.). Functions can be called from within functions.
- Functions come in different types and have different types of arguments. For example the function Gradient:

  ```
  void    Gradient(argument1, argument1, etc.)
  {
   double        a, b;  /* declaration of doubles */
   int           c, d;  /* declaration of integers */
   register int  i      /* declaration of frequently used integers */

  (... lots of code goes in here ...)
  return;
  }
  ```

  - The actual contents of the function is in between the { and }
  - The names and types of variables (except global variables) that will be used is the function have to be declared first (see 'double a, b' etc).
  - At the statement **return;** we return to the place from which the function was called.

-
  ```
  for(i=0; i<cohort_no[0]; i++)
  {
     popgrad[0][i][age] = 1.0;
  }
  ```

  - This is a loop starting from i=0
  - For each value of i, the code in between { and } is executed,

  - after which i is increased by 1.
  - This is repeated as long as i<cohort_no[0]

- i++ is the same as i=i+1,

  see also:

  ```
  eggs += pop[0][i][number]*E*pr;
  ```

  which means

  ```
  eggs = eggs + pop[0][i][number]*E*pr;
  ```

- a = floor((pop[0][i][age]+SMALL)/YEAR);

  The function floor() rounds of to the nearest integer **below** the number in between parenthesis.

- if (a==1.0)

  ```
    pr = P1;
  else if (a==2.0)
    pr = P2;
  ```

  - If the condition inside (...) is true, then the command is executed, otherwise, we skip to the next command (the 'else' in this case). Everything behind 'else' is only executed if the condition inside (...) was false.
  - note the difference between a == 1.0 (which checks the equality and is therefore true or false) and pr = P1 (assigns P1 to pr)

- exp(x) means e^x
- pow(a,b) means a^b
- If you're interested in more, see the book 'Practical C' which should be present at the practical.