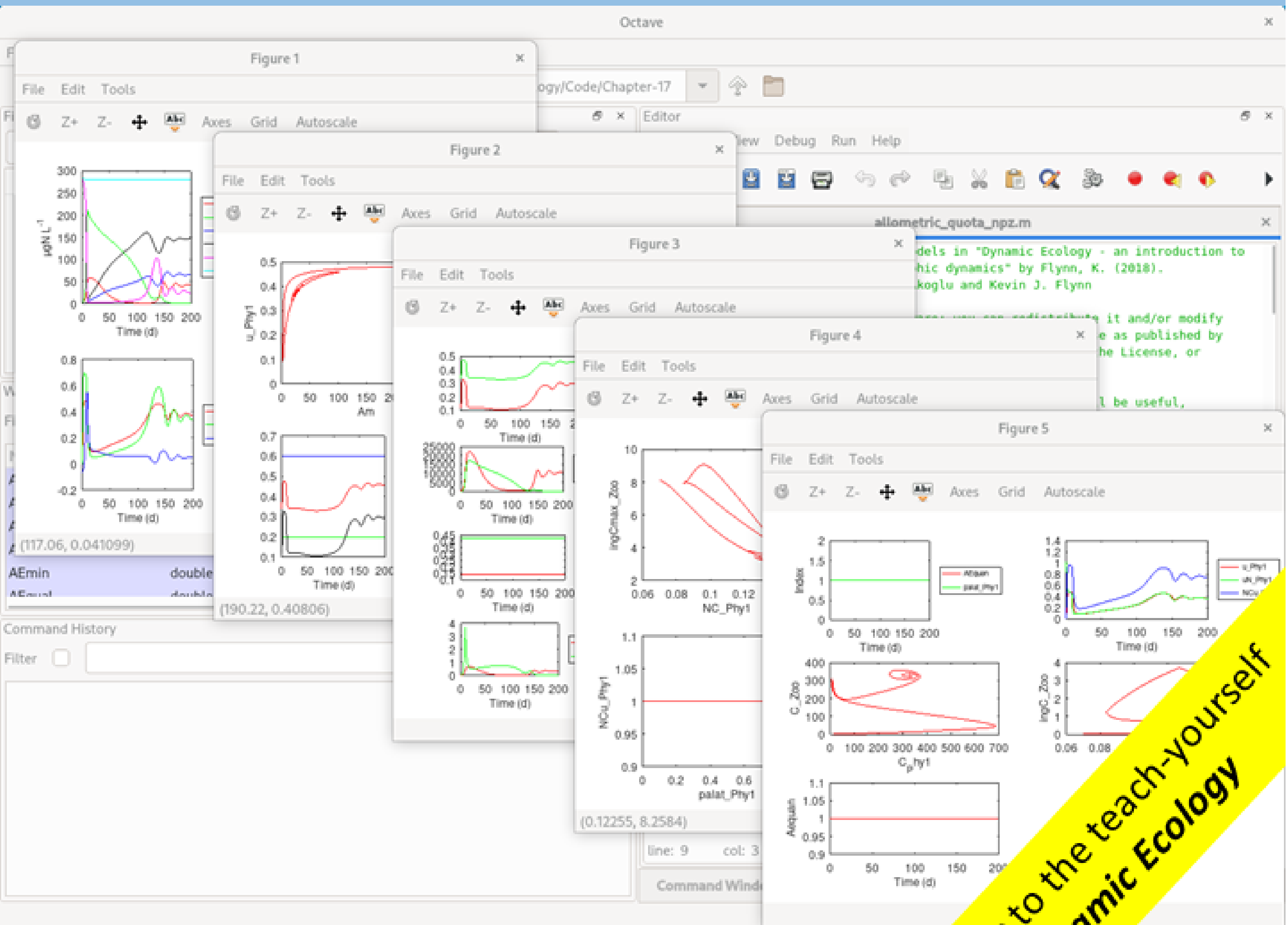
Dynamic Ecology

in GNU Octave

Ekin Akoglu

Kevin J Flynn



a companion to the teach-yourself book ***Dynamic Ecology***

How to reference this work:

Akoglu, E., & Flynn, K. J. (2020). Dynamic Ecology in GNU Octave. Zenodo. http://doi.org/10.5281/zenodo.4279642

Disclaimer

The model code presented in this work is strictly intended for educational and academic research use only. No guarantee, or other assurances, are given that the code is fit for any commercial or similar purpose.

Copyright

About the authors

Ekin Akoglu is a marine biologist and has expertise in ecological modelling with emphasis on trophodynamic and end-to-end ecosystem models. He carries out research on the effects of climate change, trophic competition and fisheries on fish stocks and marine ecosystems. He is currently employed as an assistant professor in the Institute of Marine Sciences at Middle East Technical University, Turkey.

Kevin J Flynn is a plankton physiologist who has combined laboratory and modelling studies in his teaching and research work over 4 decades. He has a particular interest in developing simulation models to guide experiment design and to enthuse the next generation of marine scientists in plankton dynamics and ecophysiology. He has authored, or co-authored, over 175 papers, and also authored the book *Dynamic Ecology* upon which this work was developed. He currently works at the Plymouth Marine Laboratory, UK.

Contents

Please use the PDF “find” facility.

Acknowledgements

The authors wish to thank all those who have contributed directly or indirectly to the development of this work.

EA was employed by Middle East Technical University during the development of this work. EA wishes to thank Ozgul Akoglu for all her support and encouragement.

KJF was employed by Plymouth Marine Laboratory during the development of this work, and in part this work was supported via research grant funds from the Natural Environment Research Council (NERC, UK) through programme NE/R011087/1, and by EC MSCA-ITN 2019 funding to the project MixITiN (grant number 766327). KJF wishes to thank Aditee Mitra for her help.

Preface

The aim of this book is to provide the reader with a text to enable them to explore the models and simulations provided in the textbook, *Dynamic Ecology* (Flynn, 2018) using a free-to-end-user software platform, namely GNU Octave.

The aim of the work is ultimately, as described in the Preface to *Dynamic Ecology*, to …

… *provide the biologist, and indeed the non-biologist (mathematician), with an introduction to dynamic ecology. The construction and operation of simulation platforms (models) provides an excellent test of understanding while also generating insight into how real complex processes in ecology operate over time. … This text is intended to provide a platform for even the least maths-orientated biologist to engage with dynamic simulations. The emphasis is on building models with at least a nod to mechanistic (trait-based) functionality.*

Like the book upon which it was based, this work provides example outputs, so the reader can check that their own creations are operating correctly before they start to modify and otherwise develop their own models. Ideas for further exploration are provided in *Dynamic Ecology*, which would be best read in parallel to this text. Together these two books provide a step-by-step introduction to systems dynamic modelling, leading the reader progressively through levels of increasing complexity.

If you identify any errors, or wish to provide feedback for future editions, please contact the authors via email to eakoglu@metu.edu.tr or KJF@PML.ac.uk

FINALLY: If you did not download this e-book yourself, please do so, via www.mixotroph.org/models It will cost you nothing to do so but it will ensure you have the latest version, and it helps us to keep track of the level of interest.

Ekin Akoglu and Kevin J Flynn

September 2020

PLEASE READ ME FIRST!

You can use this book in two ways:

1. You can work your way through it all, together with the allied chapters in the book *Dynamic Ecology* (Flynn 2018), train yourself in some different approaches to systems dynamics, build and develop models upon your chosen software platform, and hence become a modeller rather than a model user.

2. You could (largely) ignore the information justifying details of the construction that may otherwise train you to build your own models, and run the simulations to explore the suggestions made at the end of each modelling chapter in *Dynamic Ecology*. Becoming a model user is an important part of being an ecologist in the 21st century. And it is likely that you will in time start to tinker with the model itself and evolve to become a modeller!

Through the use of simulation models you can learn things very quickly, and you can experiment without fear of killing the system, or indeed without having to fill out ethics and health & safety forms.

However, it is easy to become totally immersed in modelling and not take the breaks that you need for your own health. **You are strongly advised to take a break every hour or so.** Go and walk outside and observe real ecology at work – it will stimulate your mind as well.

1. Introduction

The purpose of this book is to provide the reader with a route to building and using the models described in the volume *Dynamic Ecology* (Flynn 2018) using non-proprietary software.

From here-on, that book will simply be referred to as *Dynamic Ecology.*

The original book, *Dynamic Ecology*, while providing documentation that enabled the building of models using different platforms by someone who was confident with computer programming using Fortran, R, Python etc., was specifically designed to enable the novice to avoid the learning of such a language by use of the GUI system-dynamics software provided by the MS Windows-based Powersim Studio (www.powersim.com).

Through this new work, the models in *Dynamic Ecology* are now available as GNU Octave scripts via a GitLab repository so that readers can run the models using open-source software.

The scripts have been tested to work with GNU Octave versions 5.1.0 and 5.2.0. The authors can give no assurance that the models will work on other versions of Octave.

GNU Octave (Eaton et al. 2020) is a high-level programming language for scientific computing and has a similar syntax to MATLAB by MathWorks (www.mathworks.com/products/matlab.html). Critically, however, these scripts can be operated on any of the standard OS systems (Microsoft Windows, macOS, GNU/Linux).

This book contains chapters that guide you through the process of installing GNU Octave, and then installing and using the models described in each of the modelling chapters in *Dynamic Ecology*. Not all the chapters in *Dynamic Ecology* contain models; some explore other facets of the subject. In this book, such non-model chapters are described briefly in order to remind the reader of those facets. This has the additional advantage that the modelling chapters in the two books align numerically.

As with all such enterprises, while all due efforts have been taken to eliminate errors, the authors can accept no liability for errors however those errors may arise and neither for whatever the consequences may be.

If you identify an error or otherwise encounter any challenges, please alert the authors so that corrections can be made to the GNU Octave scripts, and/or to the next edition of this book.

Please note that the screenshots used in the following chapters are for guidance only. Depending on the characteristics of your computer's operating system, and upon updates in the files you are accessing, the exact image that you will see may differ.

This book, together with *Dynamic Ecology,* aims to provide an introduction to the modelling of ecology as it relates to the flows of material between biological and abiotic components of the ecosystem over time. To provide a point of reference, the books are based upon plankton ecology; for justifications, see Section 1.3 of *Dynamic Ecology*. The books work through from very simple (often technically highly questionable) descriptions of biology and physiology through to more complex creations. These biological entities are operated within a simple framework describing the physical and chemical environment. The abiotic components are also described in simple terms, but with sufficient complexity and variety to demonstrate that the environment can easily have an overwhelming influence on the dynamics of ecology.

By the time you have worked your way to the end of the two books you will be well equipped to either develop and run your own physiologically detailed creations within simple abiotic frameworks, or you will have also expanded the physical description (perhaps to planetary scales).

There is a big difference between building a model using a graphic user interface (GUI) platform, such as Powersim Studio, rather than developing one directly in a non-GUI platform such as GNU Octave. Different people have different preferences, but ultimately the goals are the same; to produce a robust meaningful and useful description of the processes being simulated. These matters are considered in more detail in the first chapters in *Dynamic Ecology*.

2. Installing GNU Octave and downloading GNU octave model scripts

This chapter guides you through the installation process required before you can use the model code, "scripts" provided with this book.

Chapter 2 in *Dynamic Ecology* (Flynn 2018) considers terms and concepts in system dynamics modelling. It is recommended that you read both that chapter 2, and also chapter 3 (on the subject of variable names and rebuilding models in different software platforms), of *Dynamic Ecology* in their entirety before continuing.

2.1 Installing GNU Octave

Before downloading the models scripts, it is necessary to download and install GNU Octave for your computer's operating system (e.g. Microsoft© Windows©, macOS©, GNU/Linux). This is achieved from https://www.gnu.org/software/octave/#install.

Once you download and install GNU Octave, you can download the models' scripts as a zip file at https://gitlab.com/dynamic-ecology/dynamic-ecology-models-in-gnu-octave as shown in Fig. 2.1.

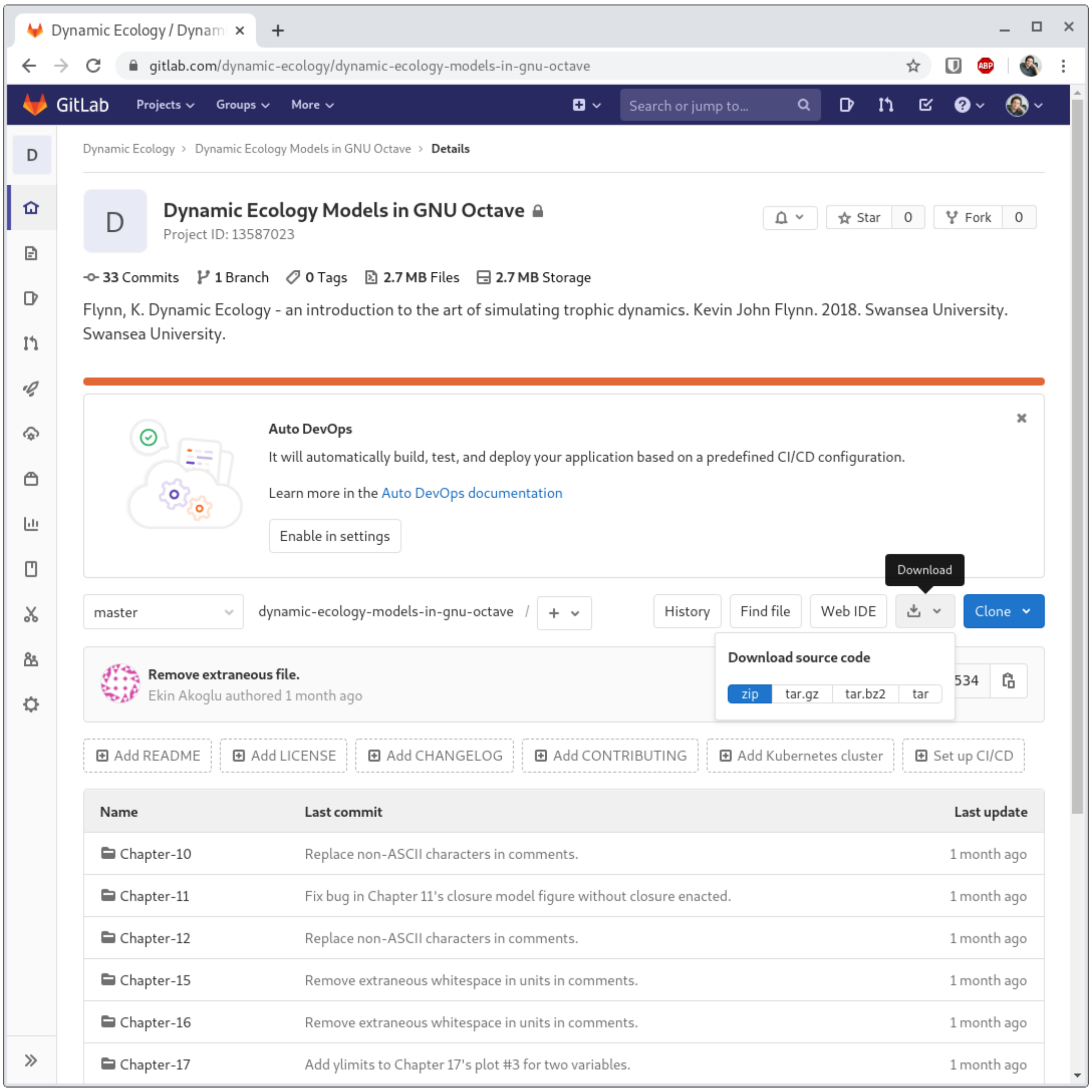


Fig. 2.1 Downloading GNU Octave scripts from GitLab repository.

A tutorial for the basics of using GNU Octave is available at https://wiki.octave.org/Using_Octave.

2.2 The model directory

By default you will download the zipped file containing scripts to your computer's "Downloads" folder.

When the download is complete, you will have a file named "dynamic-ecology-models-in-gnu-octave-master.zip" in your "Downloads" folder. Unzip the file and navigate to the directory named "dynamic-ecology-model-in-gnu-octave-master".

In the directory, you will see a folder hierarchy structure aligned with the modelling chapters in *Dynamic Ecology*. For instance, the scripts for the models pertaining to chapter 4 of *Dynamic Ecology* are located in the directory with the same name. The script files' extensions are ".m".

In each directory, depending on the number of the models in a given chapter you will notice at least three files per model;

i) the main model file with the model's name (e.g. "<model_name>.m",

ii) a file in the format "func_<model_name>.m" for the derivative function, and

iii) solver files named "solver.m" and/or "rk4.m" that iterate the model through time.

In addition, you will see figure files in PNG format automatically plotted by the model scripts.

2.3 Running your first model

To start working with the GNU Octave models, first, run GNU Octave.

A GNU Octave window will appear as in Fig.2.2. In the main window of GNU Octave, on the left part of the window, from top to bottom, you will see "File Browser", "Workspace" and "Command History" panes.

On the right part of the main window, there lies the command window where you can type in commands.

Now using the "File Browser" on the upper left panel of the GNU Octave window, navigate to the directory where your scripts reside (Fig. 2.2).

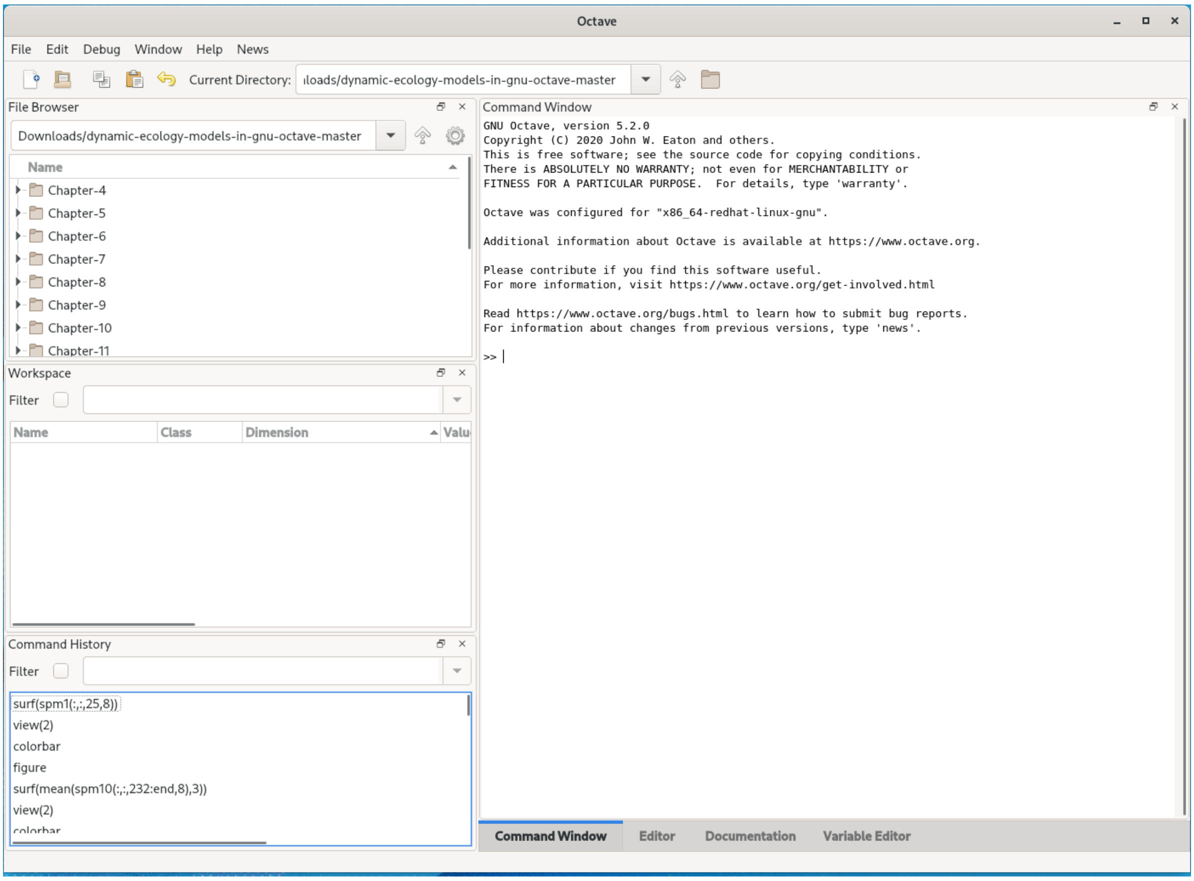


Fig. 2.2 Overview of GNU Octave's main window.

Now you can see the directories containing the model scripts per chapter in *Dynamic Ecology* (Fig. 2.2). For this tutorial, we will use Chapter 4's model as an example; however, all the models in the folders have a similar structure, and the details outlined hereinafter applies to all models.

Navigate to the folder "Chapter-4" in the "File Browser" by double-clicking it. You will now see the script files and related figures of the model in the "File Browser" (Fig. 2.3).

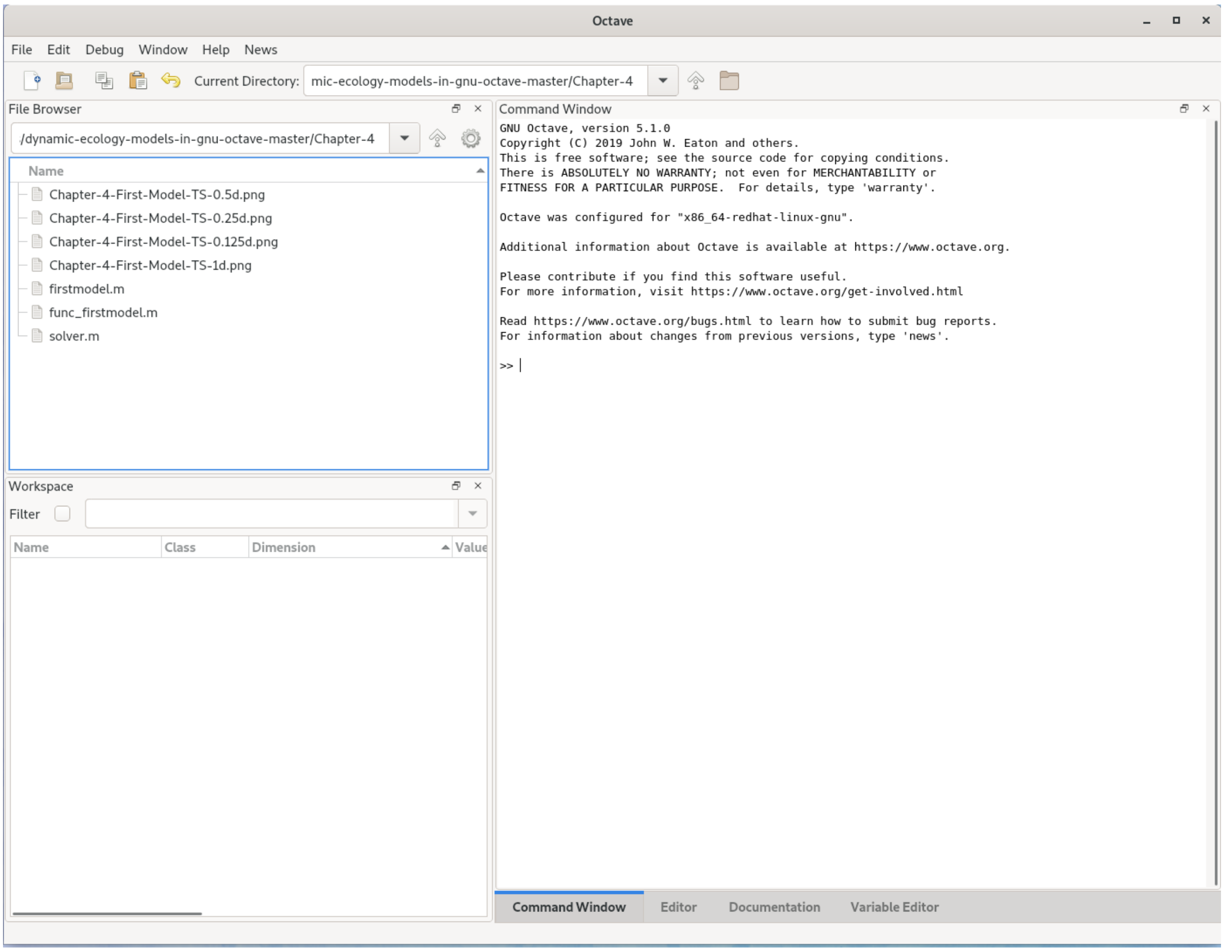


Fig. 2.3 Model of Chapter 4 in *Dynamic Ecology* and its related script files and figures.

Double-click the "firstmodel.m" file to open it (Fig. 2.4).

The script file will be opened in the editor window of GNU Octave. As you will notice, the file includes a series of GNU Octave commands and comments (lines prepended by a hashtag and coloured in green) that explain what each line of the code corresponds to.

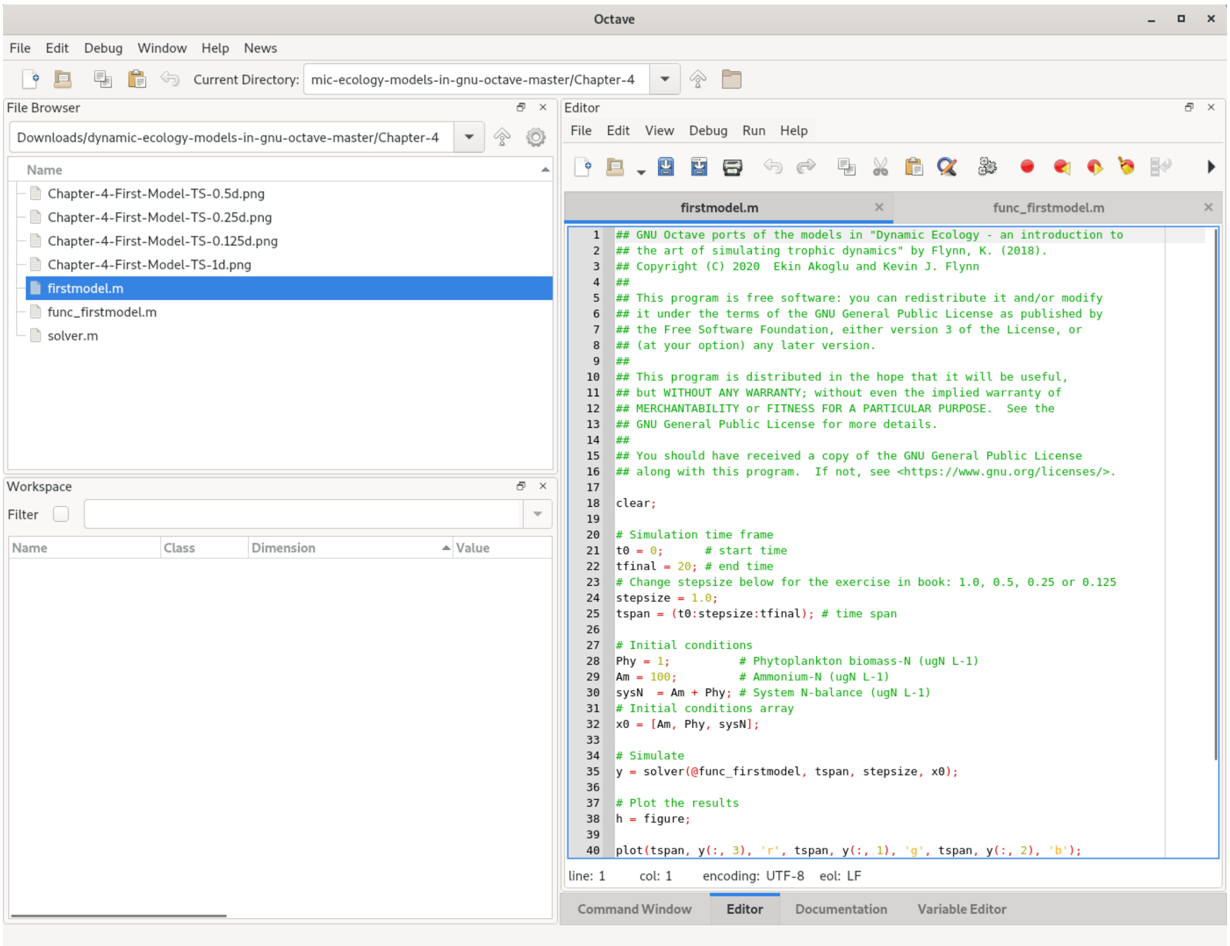


Fig. 2.4 Overview of *firstmodel.m*

To run the "firstmodel.m", hit F5 on your keyboard. Alternatively, you may switch back to the "Command Window" by using the tabs at the bottom of the right part of the main window and then enter the command "firstmodel" and press "Enter" key while you are in the "Command Window".

The model will now run and produce a plot from simulation results once the simulation is completed (Fig. 2.5).

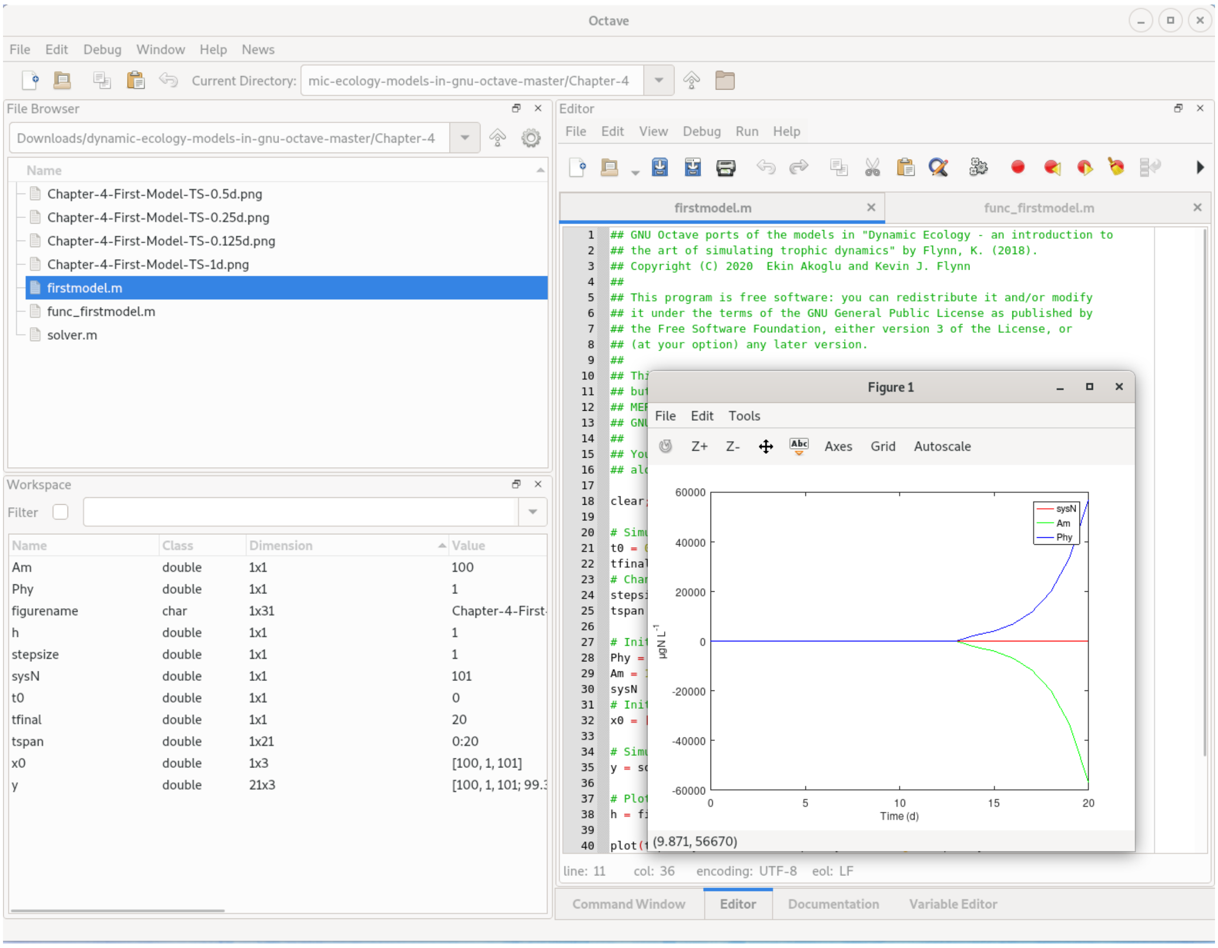


Fig. 2.5 The plot of *Dynamic Ecology* Chapter 4's model as produced by *firstmodel.m*

The plot shown in Fig. 2.5 is also printed to a PNG file in the same directory where your script file resides.

2.4 The model code and integration routine

To see the model code, the GNU Octave version of the equations given in *Dynamic Ecology*, double-click and open the derivative function (func_firstmodel.m) of Chapter 4's model for inspection (Fig. 2.6).

This file contains the main model equations detailed in chapter 4 of *Dynamic Ecology*. The file is also extensively commented and there are explanations about what each line of the code corresponds to. See also the original description in *Dynamic Ecology* for further commentary.

There is another file named "solver.m" in the model directory. This script file iterates the model through time. This employs the Euler integration method for most of the models in the book. **Unless you understand the implications of doing so, you are advised not to modify the contents of this directory or the integration routine.**

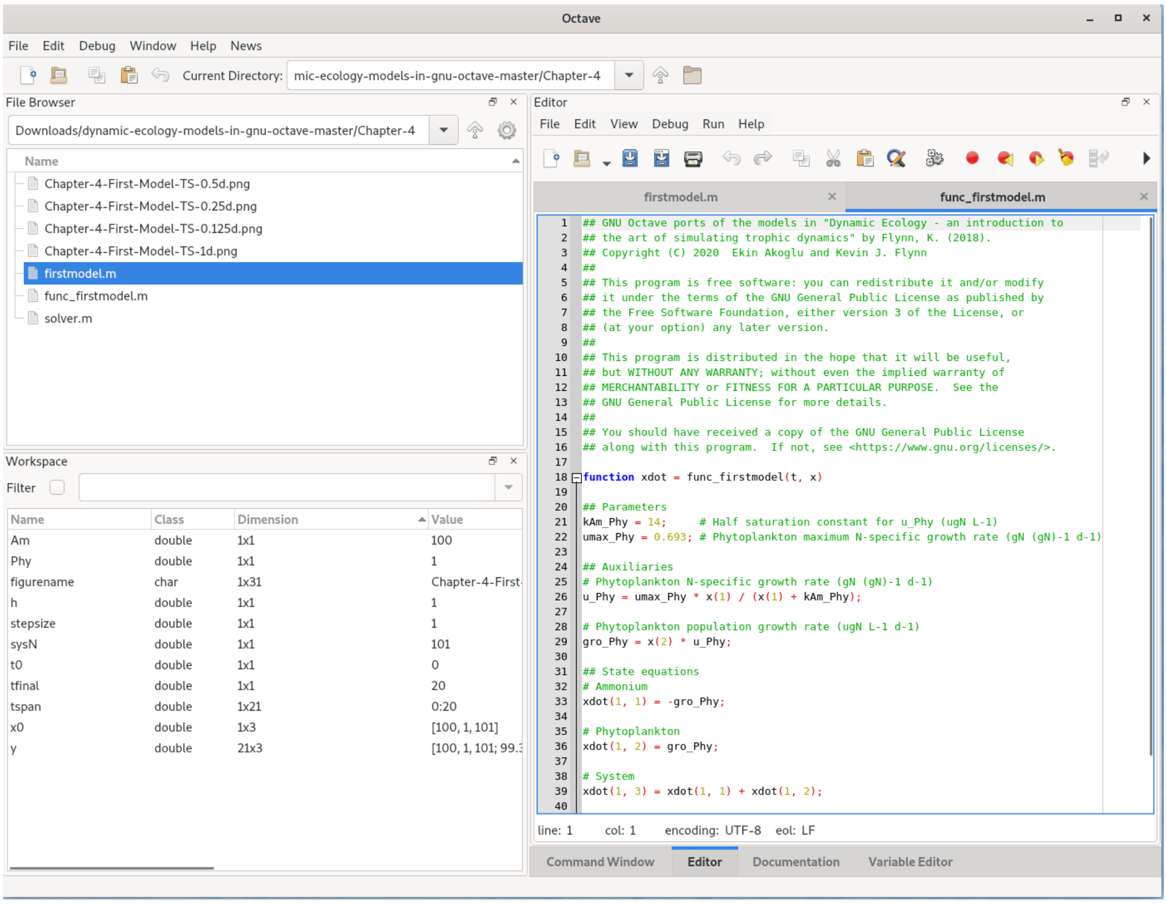


Fig. 2.6 The derivative function of *Dynamic Ecology* Chapter 4's model.

Remember, if for whatever reason the files become corrupted or non-operational, you just need to return to the source files (as per Section 2.1) and reload them.

The remaining chapters in this book provide some brief commentary on implementing the other models in *Dynamic Ecology* via GNU Octave. Non-model chapters in *Dynamic Ecology* also have their counterparts in this volume, so alerting the reader to important additional information and also preserving the chapter-chapter referencing for models between the two books.

3. Naming Variables and Building Third Party Models

Chapter 3 in *Dynamic Ecology* (Flynn 2018) discusses the issue of the naming of variables and changes that may be required to re-code models in different platforms. Please refer to *Dynamic Ecology* for more information on these topics.

This volume, of course, provides a worked example of transferring the original equations into another platform, namely GNU Octave. As can be seen by comparing the equation syntax, subtle but critical differences are often required.

Computer languages are often unforgiving in even the slightest errors in syntax. Matters such as changing parameter names may be automatically propagated throughout the rest of the code, or the platform may require manual intervention (find-replace) to update changes throughout the code script.

If you further develop the models provided here written in GNU Octave be sure to resave the file with a new file name or use a version control system (e.g. Git, Subversion) to track modifications whenever you make substantial changes; it is easy to make changes that corrupt the model in some way, and to subsequently make a bad situation worse through attempting to correct errors, so having a fallback file is very useful. Make full use of in-file documentation opportunities as well; this is critical as it is very easy to forget why you made changes. Tracking units throughout the equations is also essential; remember that the units will follow the mathematical operations, so while you can multiply and divide variables with different units, only variables with the same unit can be added or subtracted.

Some software platforms provide for automatic unit checking (e.g. Powersim Studio). However, GNU Octave does not provide such support.

Be sure to read the other chapters in *Dynamic Ecology*, especially chapters 2 and 4, which provide hints on building models. Also, remember that you can write computer code that works (in that it does not crash) but that is a mathematical and/or biological nonsense!

4. Nutrient-limited Growth model in GNU Octave

This Chapter provides information on running the model in chapter 4 of *Dynamic Ecology* (Flynn 2018), running through each of the steps. It is assumed that you have installed the Octave interface (Chapter 2).

In nature, very many factors change simultaneously, thus confounding interpretation of interactions between abiotic and biotic processes. A common driver in experimental biology is the notion of changing one factor at a time and seeing what happens. In experimental physiology, responses to resource (nutrient or food) limitation represents a popular arena for research, there being 10000's of publications on the topic ranging from very specific detailed empirical investigations to generalised theoretical studies of competition for different resources. The model described here considers a single nutrient-limitation of phytoplankton growth. The accompanying chapter in *Dynamic Ecology* provides an in-depth consideration of the model itself.

The final sections of chapter 4 in *Dynamic Ecology* provide you with ideas for experimenting and developing your models.

4.1 Running the model

Navigate to the folder "Chapter-4" in the "File Browser" by double-clicking it. You will now see the script files and related figures of the model in the "File Browser" (Fig. 4.1).

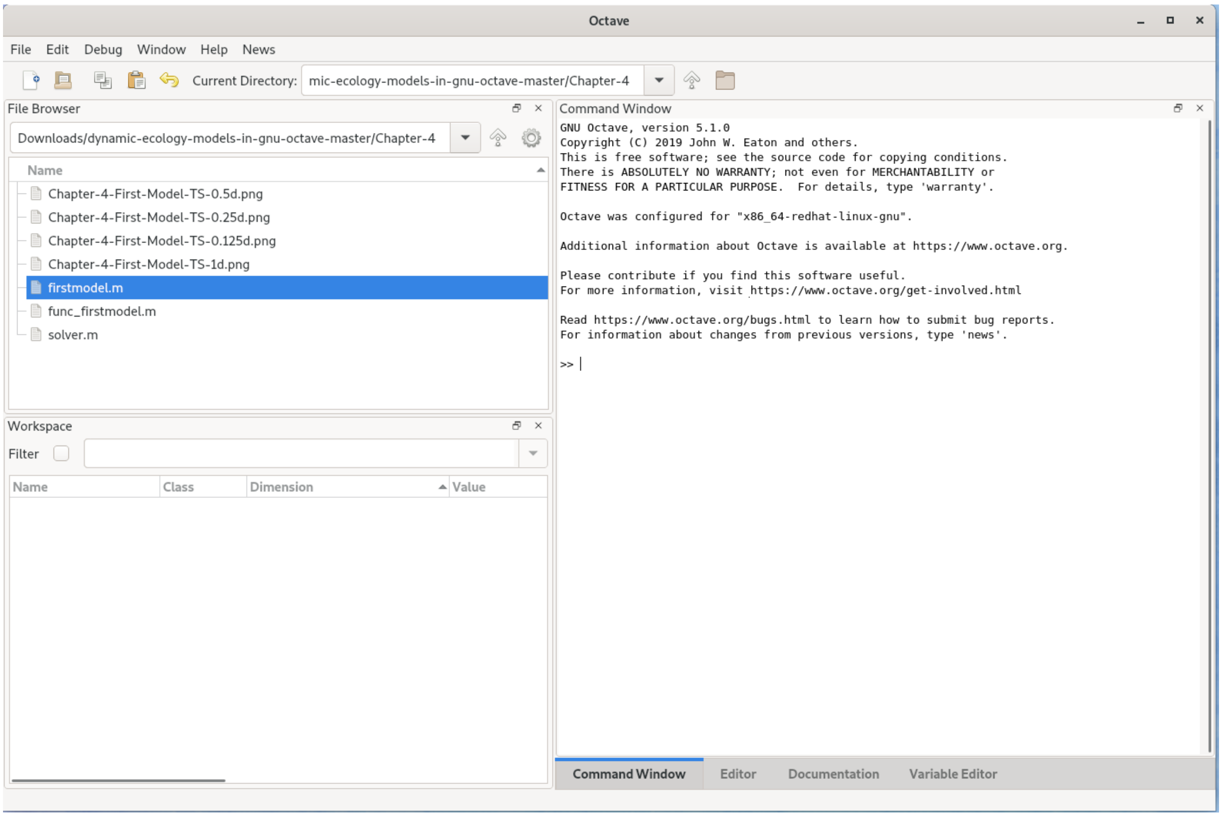


Fig. 4.1 Model of Chapter 4 in *Dynamic Ecology* and its related script files and figures.

Double-click the "firstmodel.m" file to open it (Fig. 4.2).

The script file will be opened in the editor window of GNU Octave. As you will notice, the file includes a series of GNU Octave commands and comments (lines prepended by a hashtag and coloured in green) that explain what each line of the code corresponds to.

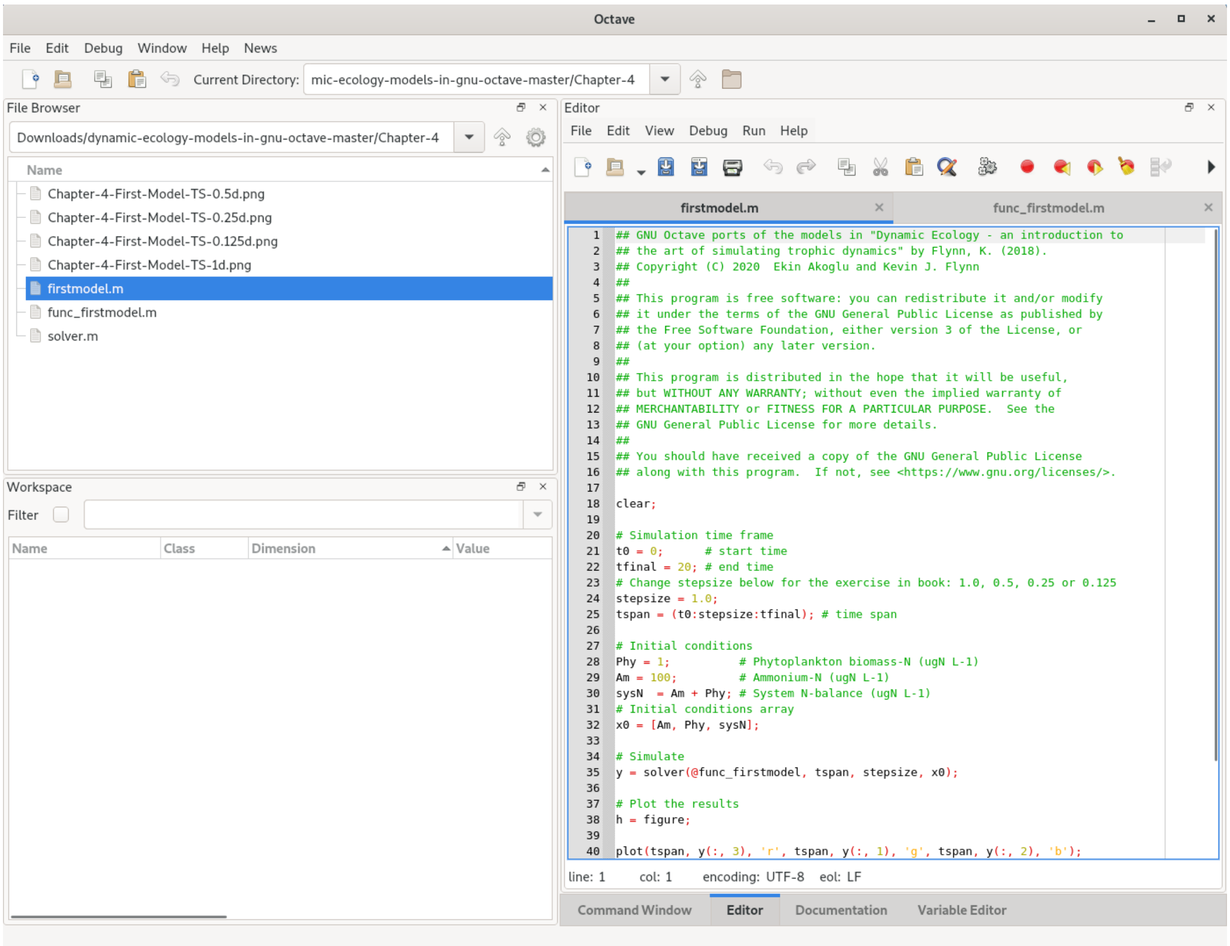


Fig. 4.2 Overview of *firstmodel.m*

To run the "firstmodel.m", hit F5 on your keyboard. Alternatively, you may switch back to the "Command Window" by using the tabs at the bottom of the right part of the main window and then enter the command "firstmodel" and press "Enter" key while you are in the "Command Window".

The model will now run and produce a plot from simulation results once the simulation is completed (Fig. 4.3).

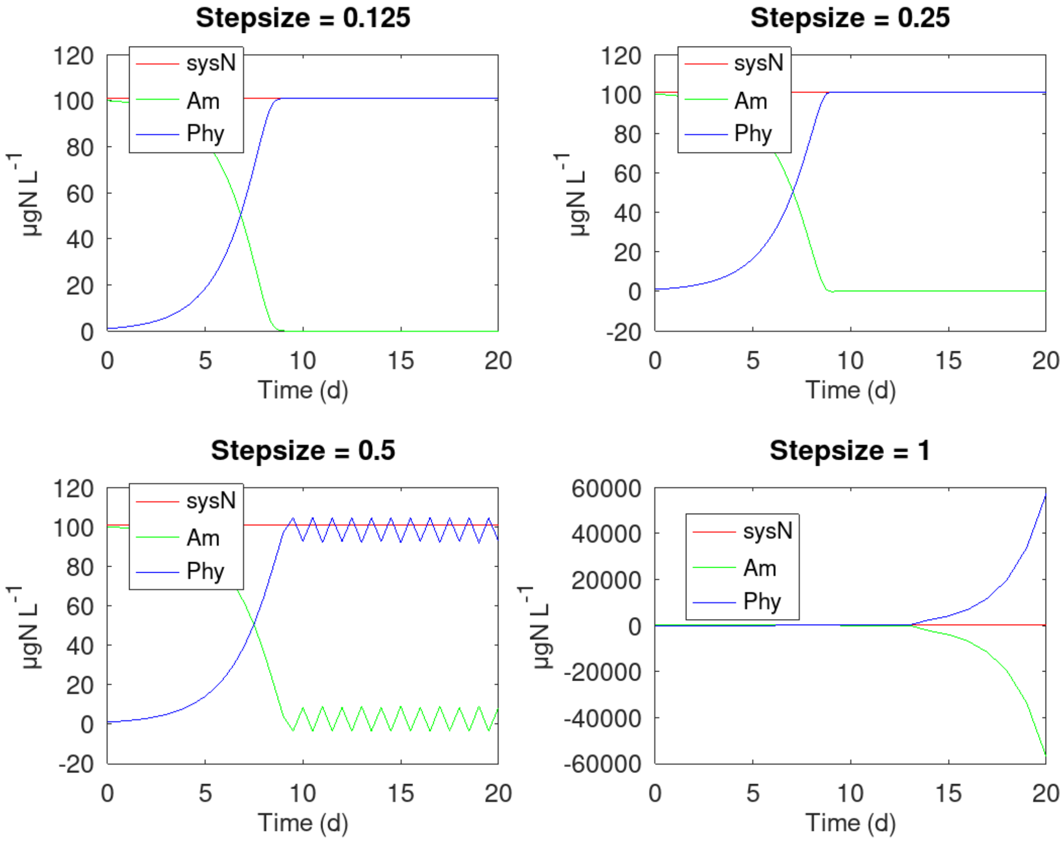


Fig. 4.3 The plot of *Dynamic Ecology* Chapter 4's model as produced by *firstmodel.m*

4.2 Changing the time step of the model

To observe the impact of changing the time step of the model as detailed in section “4.9 Operating the Model” of *Dynamic Ecology*, change the value of the “stepsize” variable in the “firstmodel.m” file on line 24. Then save and run the model.

To experiment with the model as detailed in section “4.10 Things to explore” of *Dynamic Ecology* you need to change values of the model constants in file “func_firstmodel.m” (Fig. 4.4). The model constants are listed under the comment “## Parameters” on line 20 in the file. You can refer to the comments (lines prepended with a hashtag and coloured in green) next to the variable names to understand what each variable corresponds to.

If you make a mistake, you can always undo/redo using the arrow buttons just above the Octave’s Editor Window, and if you cannot resolve the problem, just download a new copy of the original GNU Octave model, and start over again.

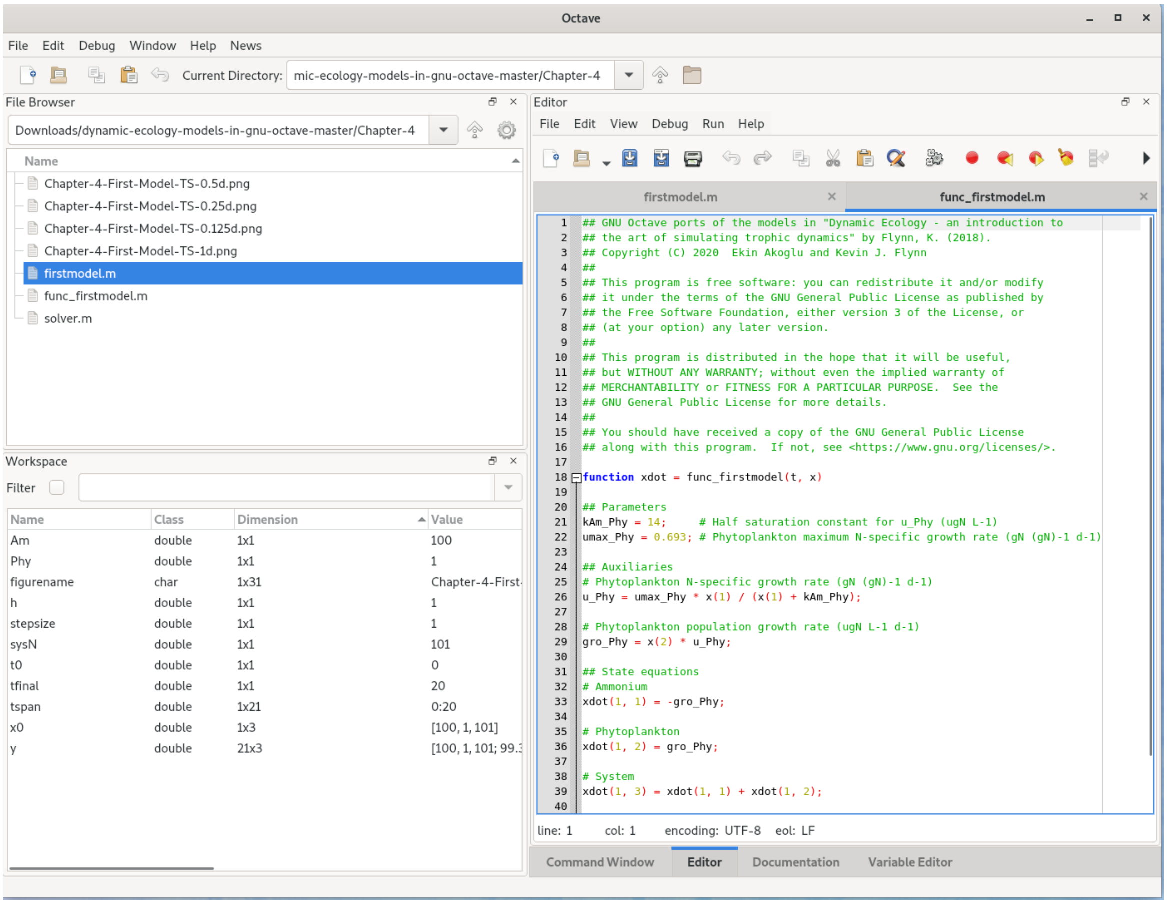


Fig. 4.4 The derivative function of *Dynamic Ecology* Chapter 4's model.

4.3 GNU Octave code

This section, running over the following pages, provides a complete dump of the GNU Octave code as it appears in the download.

4.3.1 firstmodel.m

```
## GNU Octave ports of the models in "Dynamic Ecology - an introduction to
## the art of simulating trophic dynamics" by Flynn, K. (2018).
## Copyright (C) 2020 Ekin Akoglu and Kevin J. Flynn

## This program is free software: you can redistribute it and/or modify
## it under the terms of the GNU General Public License as published by
## the Free Software Foundation, either version 3 of the License, or
## (at your option) any later version.

## This program is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
## GNU General Public License for more details.

## You should have received a copy of the GNU General Public License
## along with this program. If not, see https://www.gnu.org/licenses/.

clear;

# Simulation time frame
t0 = 0;      # start time
tfinal = 20; # end time
# Change stepsize below for the exercise in book: 1.0, 0.5, 0.25 or 0.125
stepsize = 1.0;
tspan = (t0:stepsize:tfinal); # time span

# Initial conditions
Phy = 1;           # Phytoplankton biomass-N (ugN L-1)
Am = 100;          # Ammonium-N (ugN L-1)
sysN = Am + Phy;   # System N-balance (ugN L-1)
# Initial conditions array
x0 = [Am, Phy, sysN];

# Simulate
y = solver(@func_firstmodel, tspan, stepsize, x0);

# Plot the results
h = figure;

plot(tspan, y(:, 3), 'r', tspan, y(:, 1), 'g', tspan, y(:, 2), 'b');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
ylabel('\mugN L^{-1}', 'FontSize', 12);
legend('sysN', 'Am', 'Phy');
figurename = ['Chapter-4-First-Model-TS-' num2str(stepsize), 'd.png'];
print(h, figurename, '-dpng', '-color');
```

4.3.2 func_firstmodel.m

```
## GNU Octave ports of the models in "Dynamic Ecology - an introduction to
## the art of simulating trophic dynamics" by Flynn, K. (2018).
## Copyright (C) 2020 Ekin Akoglu and Kevin J. Flynn

## This program is free software: you can redistribute it and/or modify
## it under the terms of the GNU General Public License as published by
## the Free Software Foundation, either version 3 of the License, or
## (at your option) any later version.

## This program is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
## GNU General Public License for more details.

## You should have received a copy of the GNU General Public License
## along with this program. If not, see https://www.gnu.org/licenses/.

function xdot = func_firstmodel(t, x)

## Parameters

kAm_Phy = 14;      # Half saturation constant for u_Phy (ugN L-1)
umax_Phy = 0.693; # Phytoplankton maximum N-specific growth rate (gN (gN)-1 d-1)

## Auxiliaries
## Phytoplankton N-specific growth rate (gN (gN)-1 d-1)
u_Phy = umax_Phy * x(1) / (x(1) + kAm_Phy);

# Phytoplankton population growth rate (ugN L-1 d-1)
gro_Phy = x(2) * u_Phy;

##State equations
# Ammonium
xdot(1, 1) = -gro_Phy;

# Phytoplankton
xdot(1, 2) = gro_Phy;

# System
xdot(1, 3) = xdot(1, 1) + xdot(1, 2);


endfunction
```

5. A Simple Predator-Prey Model in GNU Octave

This Chapter provides information on running the model in chapter 5 of *Dynamic Ecology* (Flynn 2018), running through each of the steps. It is assumed that you have installed the Octave interface (Chapter 2).

The simplest, and most enduring, of biological models involve predator-prey interactions. We will come to the classic model, which describe the interactions in crude terms, in Chapter 6. Here we develop something which is actually significantly more realistic, which shows the flows of nutrients around the ecosystem. Such flows, the accounting of material between components of a system, are fundamental features defining the dynamics of ecology and the system dynamics approach. To build this model we will extend the description of the phytoplankton model built in Chapter 4 to include a predator to feed upon the phytoplankton prey, with the consequential nutrient recycling.

Please see chapter 5 in *Dynamic Ecology* for more contextual information, explanations for model construction, and (in the final sections of that chapter) ideas for experimenting and developing your models.

5.1 Running the model

Navigate to the folder “Chapter-5” in the “File Browser” by double-clicking it. You will now see the script files and related figures of the model in the “File Browser” (Fig. 5.1).

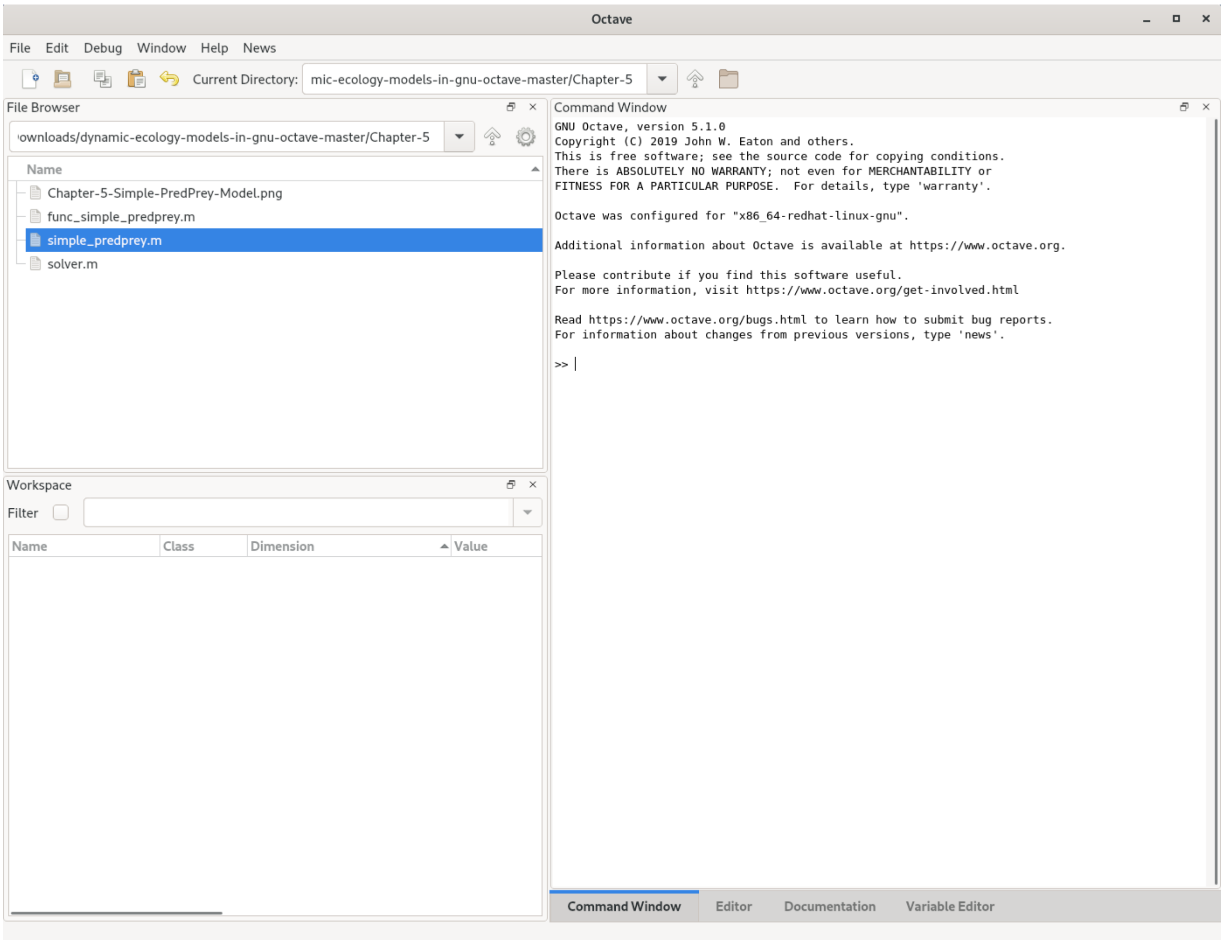


Fig. 5.1 Model of Chapter 5 in *Dynamic Ecology* and its related script files and figures.

Double-click the "simple_predprey.m" file to open it (Fig. 5.2).

The script file will be opened in the editor window of GNU Octave. As you will notice, the file includes a series of GNU Octave commands and comments (lines prepended by a hashtag and coloured in green) that explain what each line of the code corresponds to.

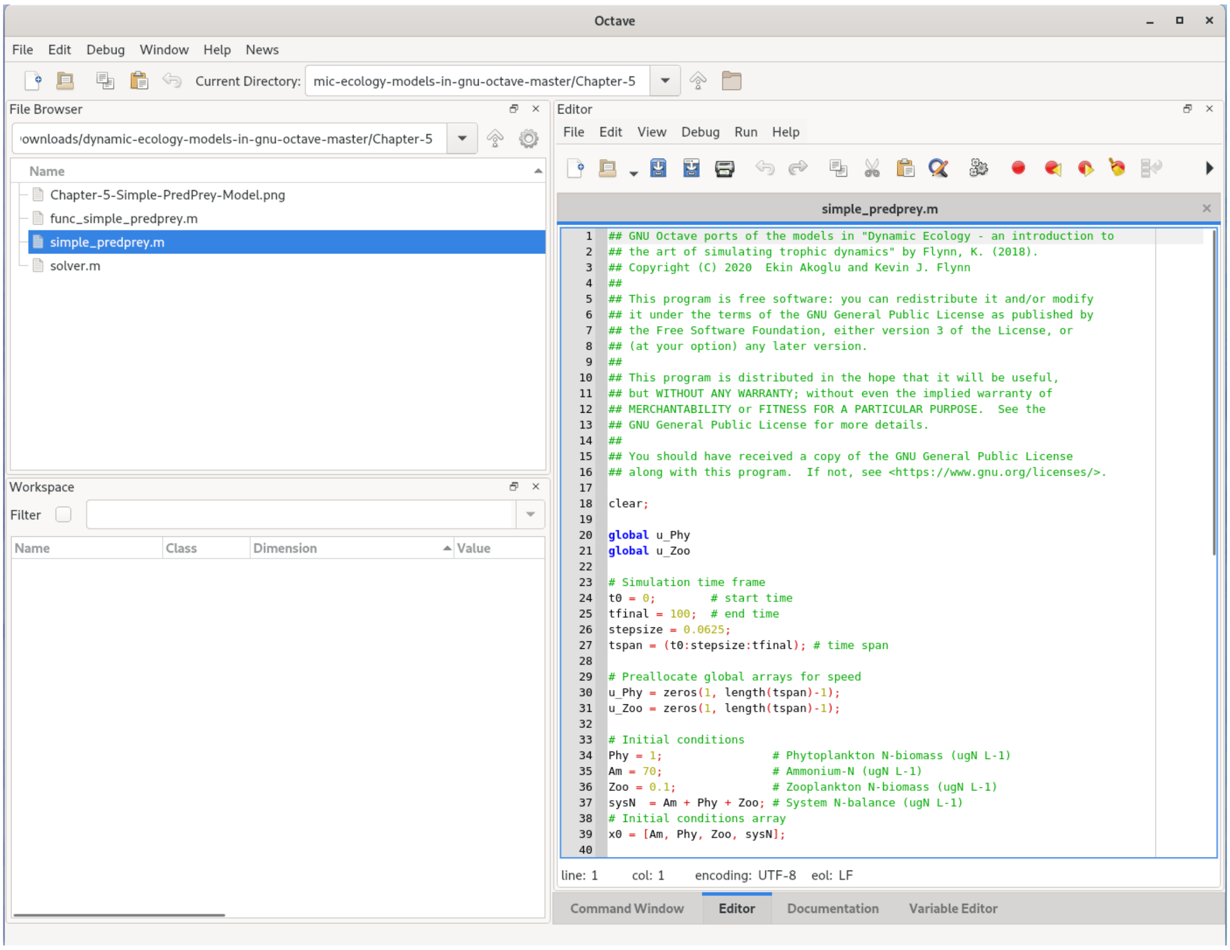


Fig. 5.2 Overview of *simple_predprey.m*

To run the "simple_predprey.m", hit F5 on your keyboard. Alternatively, you may switch back to the "Command Window" by using the tabs at the bottom of the right part of the main window and then enter the command "simple_predprey" and press "Enter" key while you are in the "Command Window".

The model will now run and produce a plot from simulation results once the simulation is completed (Fig. 5.3).

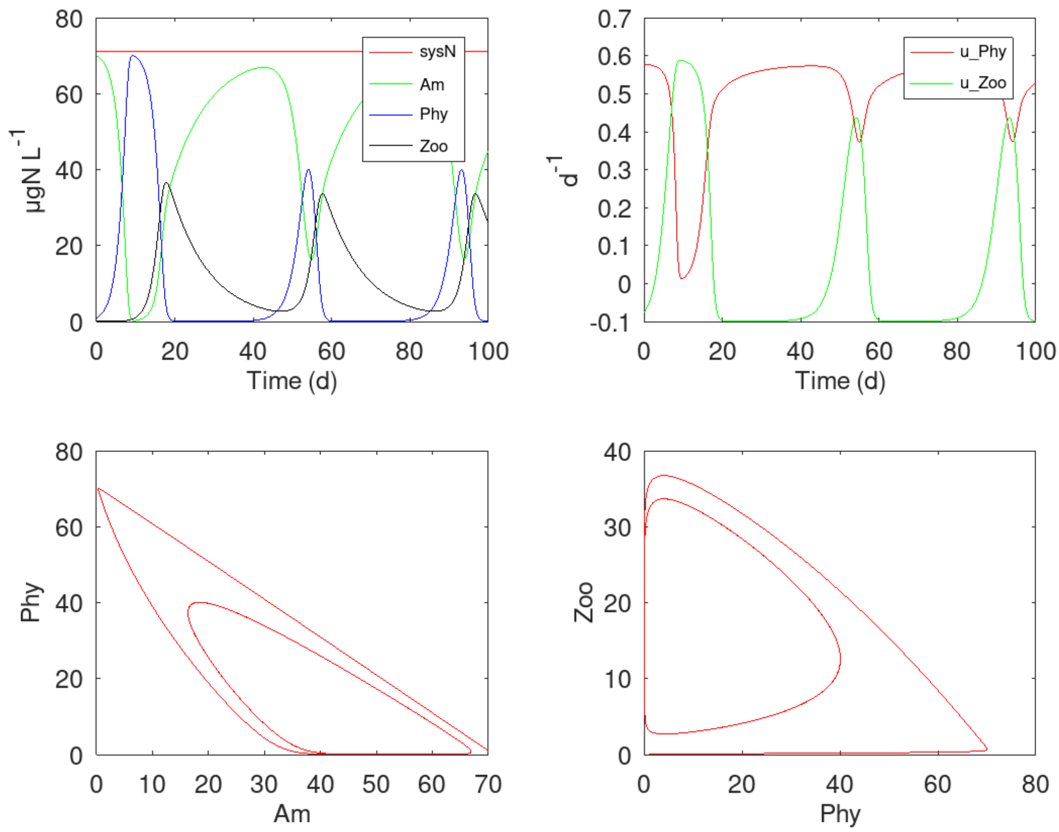


Fig. 5.3 The plot of *Dynamic Ecology* Chapter 5's model as produced by *simple_predprey.m*

5.2 Experimenting with the model

To experiment with the model as detailed in section "5.9 Things to explore" of *Dynamic Ecology* you need to change values of the model constants in file "func_simple_predprey.m" (Fig. 5.4). You can refer to the comments (lines prepended with a hashtag and coloured in green) next to the variable names to understand what each variable corresponds to.

If you make a mistake, you can always undo/redo using the arrow buttons just above the Octave's Editor Window, and if you cannot resolve the problem, just download a new copy of the original GNU Octave model, and start over again.

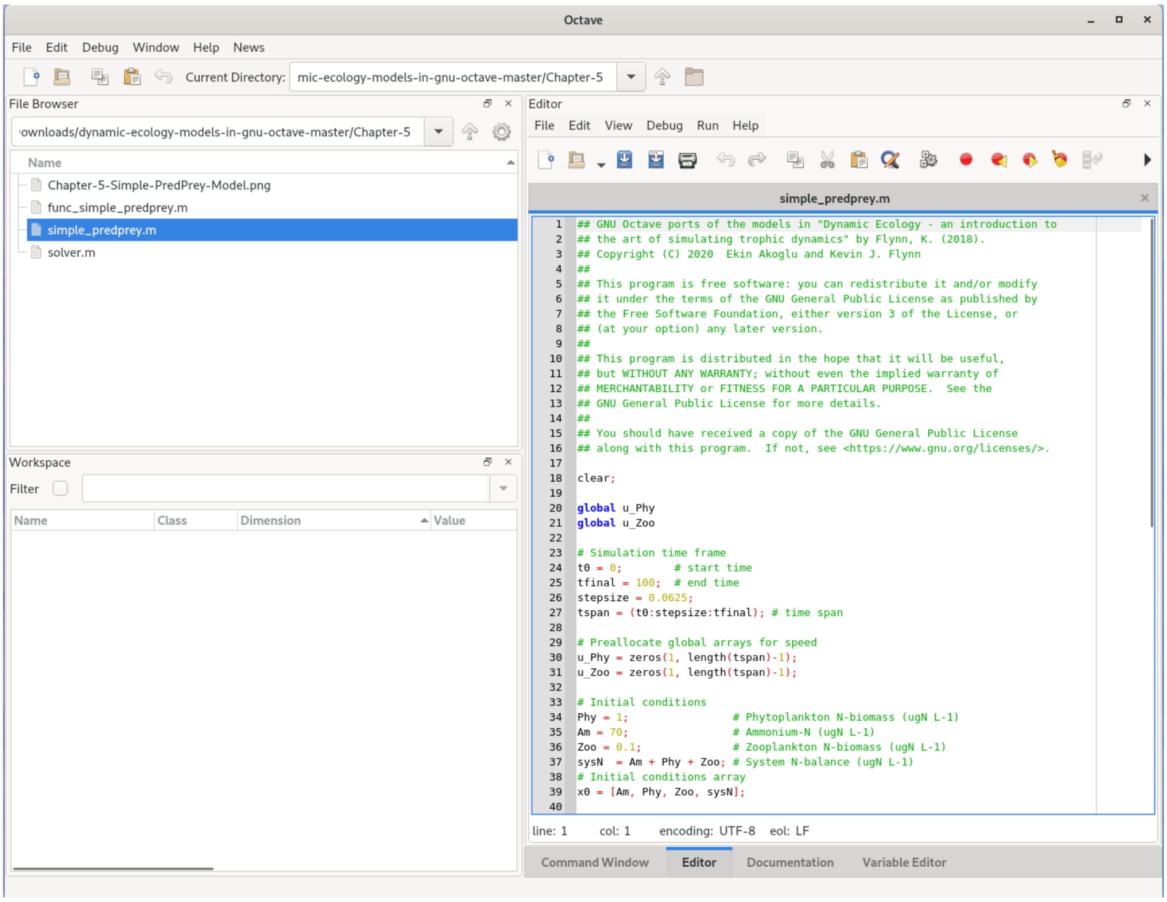


Fig. 5.4 The derivative function of *Dynamic Ecology* Chapter 5's model.

5.3 GNU Octave code

This section, running over the following pages, provides a complete dump of the GNU Octave code as it appears in the download.

5.3.1 simple_predprey.m

```
## GNU Octave ports of the models in "Dynamic Ecology - an introduction to
## the art of simulating trophic dynamics" by Flynn, K. (2018).
## Copyright (C) 2020 Ekin Akoglu and Kevin J. Flynn

## This program is free software: you can redistribute it and/or modify
## it under the terms of the GNU General Public License as published by
## the Free Software Foundation, either version 3 of the License, or
## (at your option) any later version.

## This program is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
## GNU General Public License for more details.

## You should have received a copy of the GNU General Public License
## along with this program. If not, see https://www.gnu.org/licenses/.

clear;

global u_Phy
global u_Zoo

# Simulation time frame
t0 = 0;         # start time
tfinal = 100;   # end time
stepsize = 0.0625;
tspan = (t0:stepsize:tfinal); # time span

# Preallocate global arrays for speed
u_Phy = zeros(1, length(tspan)-1);
u_Zoo = zeros(1, length(tspan)-1);

# Initial conditions
Phy = 1;                  # Phytoplankton N-biomass (ugN L-1)
Am = 70;                  # Ammonium-N (ugN L-1)
Zoo = 0.1;                # Zooplankton N-biomass (ugN L-1)
sysN = Am + Phy + Zoo;    # System N-balance (ugN L-1)
# Initial conditions array
x0 = [Am, Phy, Zoo, sysN];

# Simulate
y = solver(@func_simple_predprey, tspan, stepsize, x0);

# Plot the results
h = figure;

subplot(2, 2, 1);
plot(tspan, y(:, 4), 'r', tspan, y(:, 1), 'g', tspan, y(:, 2), 'b', tspan, y(:,
3), 'k');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
ylabel('\mugN L^{-1}', 'FontSize', 12);
```

```
hleg = legend('sysN', 'Am', 'Phy', 'Zoo');
set(hleg, 'FontSize', 8);

subplot(2, 2, 2);
plot(tspan(2:end), u_Phy', 'r', tspan(2:end), u_Zoo', 'g');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
ylabel('d^{-1}', 'FontSize', 12);
hleg = legend('u\_Phy', 'u\_Zoo');
set(hleg, 'FontSize', 8);

subplot(2, 2, 3);
plot(y(:, 1), y(:, 2), 'r');
set(gca,'FontSize',12);
xlabel('Am', 'FontSize', 12);
ylabel('Phy', 'FontSize', 12);

subplot(2, 2, 4);
plot(y(:, 2), y(:, 3), 'r');
set(gca,'FontSize',12);
xlabel('Phy', 'FontSize', 12);
ylabel('Zoo', 'FontSize', 12);

print(h, 'Chapter-5-Simple-PredPrey-Model.png', '-dpng', '-color');
```

5.3.2 func_simple_predprey.m

```
## GNU Octave ports of the models in "Dynamic Ecology - an introduction to
## the art of simulating trophic dynamics" by Flynn, K. (2018).
## Copyright (C) 2020 Ekin Akoglu and Kevin J. Flynn

## This program is free software: you can redistribute it and/or modify
## it under the terms of the GNU General Public License as published by
## the Free Software Foundation, either version 3 of the License, or
## (at your option) any later version.

## This program is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
## GNU General Public License for more details.

## You should have received a copy of the GNU General Public License
## along with this program. If not, see https://www.gnu.org/licenses/.

function xdot = func_simple_predprey(t, x)

global u_Phy
global u_Zoo

# Phytoplankton parameters
kAm_Phy = 14;      # Half saturation constant for u_Phy (ugN L-1)
umax_Phy = 0.693; # Phytoplankton maximum N-specific growth rate (gN (gN)-1 d-1)

# Zooplankton parameters
umax_Zoo = 1;  # Maximum specific growth rate of the zooplankton (gN (gN)-1 d-1)
kPhy_Zoo = 42;     # Half saturation constant for ingN_Zoo (ugN L-1)
thresPhy = 0.014; # Threshold for predation (ugN L-1)
BR_Zoo = 0.1;      # Index of basal (catabolic) respiration (dl)
AEN_Zoo = 0.6;     # Assimilation efficiency for N (dl)
SDA = 0.3;  # Specific dynamic action (anabolic respiration cost for
assimilating N, gN/gN)

## Auxiliaries
# Phytoplankton N-specific growth rate (gN (gN)-1 d-1)
u_Phy(t - 1)=  umax_Phy * x(1) / (x(1) + kAm_Phy);

# Phytoplankton population growth rate (ugN L-1 d-1)
gro_Phy = x(2) * u_Phy(t - 1);

# Ingestion rate with inclusion of threshold control (gN (gN)-1 d-1)
ingNmax_Zoo = (umax_Zoo * ( 1 + BR_Zoo)) / (AEN_Zoo * (1 - SDA));

# Maximum ingestion rate (gN (gN)-1 d-1)
if x(2) > thresPhy
  ingPhy_Zoo = ingNmax_Zoo * (x(2) - thresPhy) / (x(2) - thresPhy + kPhy_Zoo);
else
  ingPhy_Zoo = 0;
endif

# Zooplankton N-specific growth rate (gN (gN)-1 d-1)
u_Zoo(t - 1) = ingPhy_Zoo * AEN_Zoo * (1 - SDA) - (umax_Zoo * BR_Zoo);

# Zooplankton assimilation rate (gN (gN)-1 d-1)
assN_Zoo = ingPhy_Zoo * AEN_Zoo;

# Zooplankton N-specific regeneration rate (gN (gN)-1 d-1)
regN_Zoo = (umax_Zoo * BR_Zoo) + assN_Zoo * SDA;
```

```
# Zooplankton population ingestion rate (ugN L-1 d-1)
ing_Zoo = x(3) * ingPhy_Zoo;

# Zooplankton population N-regeneration rate (ugN L-1 d-1)
reg_Zoo = x(3) * regN_Zoo;

# Zooplankton population N-voiding rate (ugN L-1 d-1)
void_Zoo = x(3) * ingPhy_Zoo * (1 - AEN_Zoo);

## State equations
# Ammonium
xdot(1, 1) = -gro_Phy + reg_Zoo + void_Zoo;

# Phytoplankton
xdot(1, 2) = gro_Phy - ing_Zoo;

# Zooplankton
xdot(1, 3) = ing_Zoo - reg_Zoo - void_Zoo;

# System
xdot(1, 4) = xdot(1, 1) + xdot(1, 2) + xdot(1, 3);

endfunction
```

6. Logistic and Lotka -Volterra Models in GNU Octave

This Chapter provides information on running the model in chapter 6 of *Dynamic Ecology* (Flynn 2018), running through each of the steps. It is assumed that you have installed the Octave interface (Chapter 2).

A feature common in all real biological systems is the non-linear density dependence of rate processes. In other words, as the abundance of resources and of biomass of different organisms changes so the rates of growth and so on do not change *pro rata*, in a simple fashion. So far we have considered such dynamics using an explicit link to resource availability (as nutrient or food) through the use of rectangular hyperbolic functions (see chapters 4 & 5 in *Dynamic Ecology*). However, this is not how density-dependence has been described in models in classic theoretical ecology.

Classically, and with an eye to the pragmatic reality of lacking conceptual and numeric information to do otherwise, such relationships have been described using wholly empirical approaches that simply describe the fact that growth does not continue for ever (something must restrict it, but we do not know what) and that predator-prey interactions also involve process that display cyclic density dependence (again, relating to some factors about which we are not quite sure). These classic descriptions are the **Logistic equation** and **Lotka-Volterra** (L-V) models.

Traditionally, a text on dynamic ecology would have started with these two models. Scientists now have a much firmer grasp of how real systems work, and our computational abilities are also much improved, such that we can now explicitly involve controlling factors that we were formally ignorant of and/or could not readily model. It is nonetheless useful to see how these tradition approaches operate in comparison with systems dynamic approaches.

There are two models in this chapter:

i) logistic, and
ii) Lotka-Volterra.

Please see chapter 6 in *Dynamic Ecology* for more contextual information, explanations for model construction, and (in the final sections of that chapter) ideas for experimenting and developing your models.

6.1 Running the logistic model

Navigate to the folder "Chapter-6" in the "File Browser" by double-clicking it. You will now see the script files and related figures of the model in the "File Browser" (Fig. 6.1).

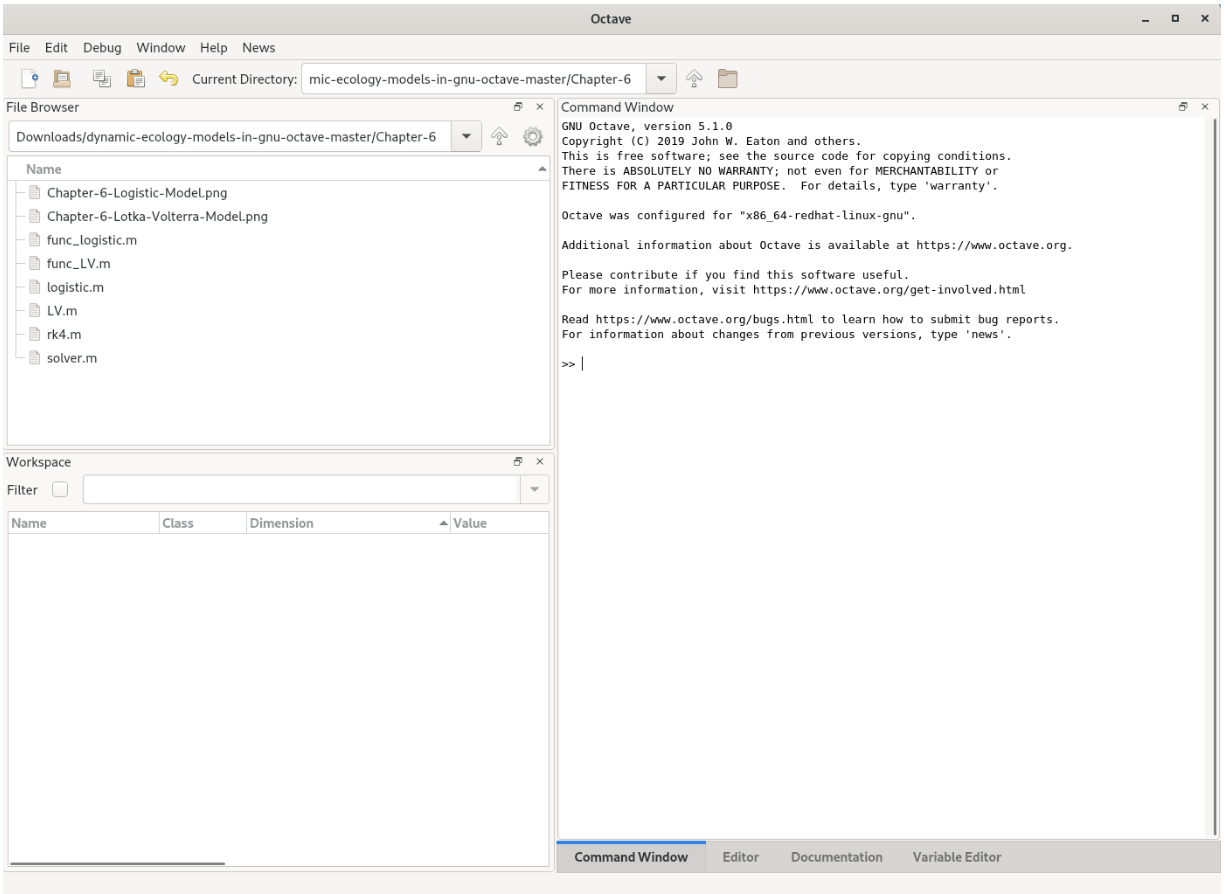


Fig. 6.1 Models of Chapter 6 in *Dynamic Ecology* and their related script files and figures.

Double-click the “logistic.m” file to open the logistic model script (Fig. 6.2).

The script file will be opened in the editor window of GNU Octave. As you will notice, the file includes a series of GNU Octave commands and comments (lines prepended by a hashtag and coloured in green) that explain what each line of the code corresponds to.

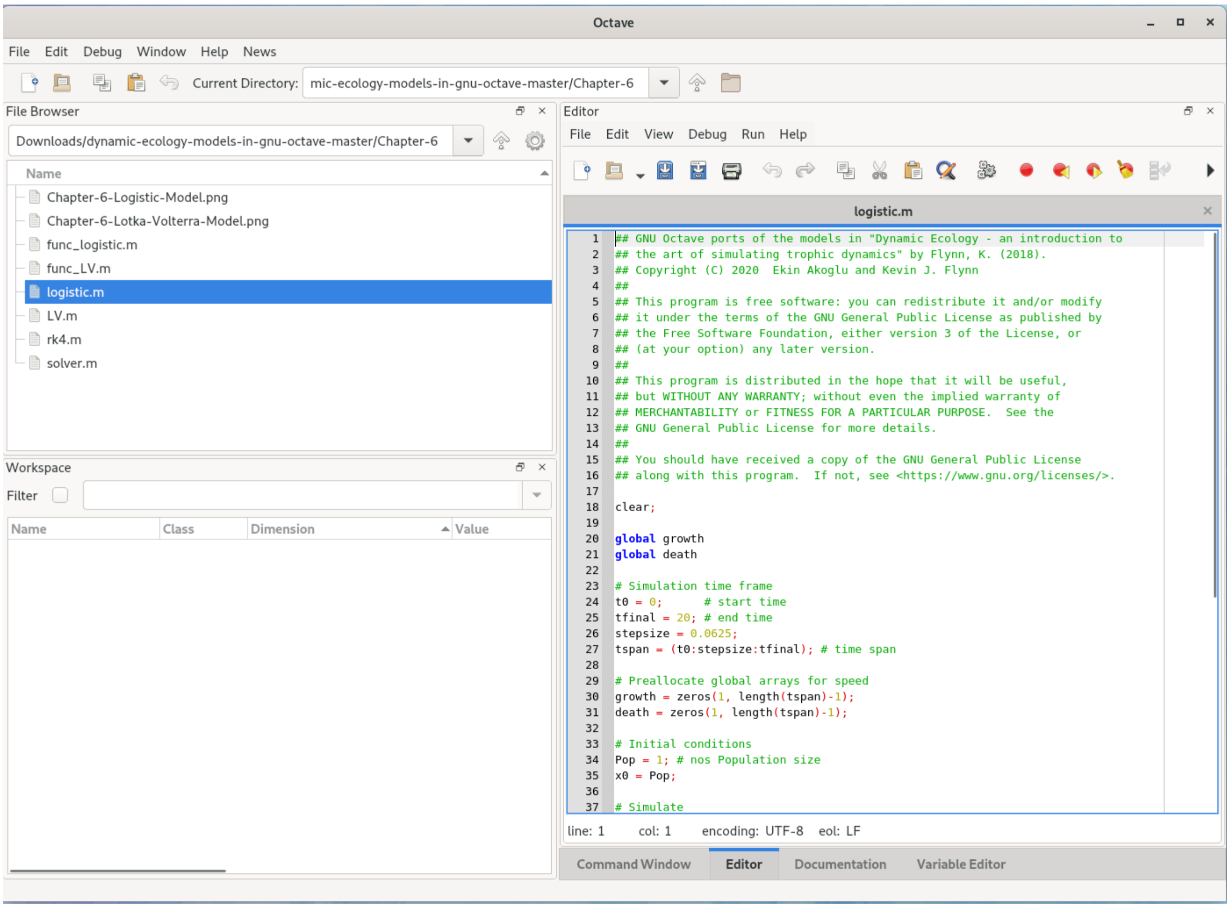


Fig. 6.2 Overview of *logistic.m*

To run the "logistic.m", hit F5 on your keyboard. Alternatively, you may switch back to the "Command Window" by using the tabs at the bottom of the right part of the main window and then enter the command "logistic" and press "Enter" key while you are in the "Command Window".

The model will now run and produce a plot from simulation results once the simulation is completed (Fig. 6.3).

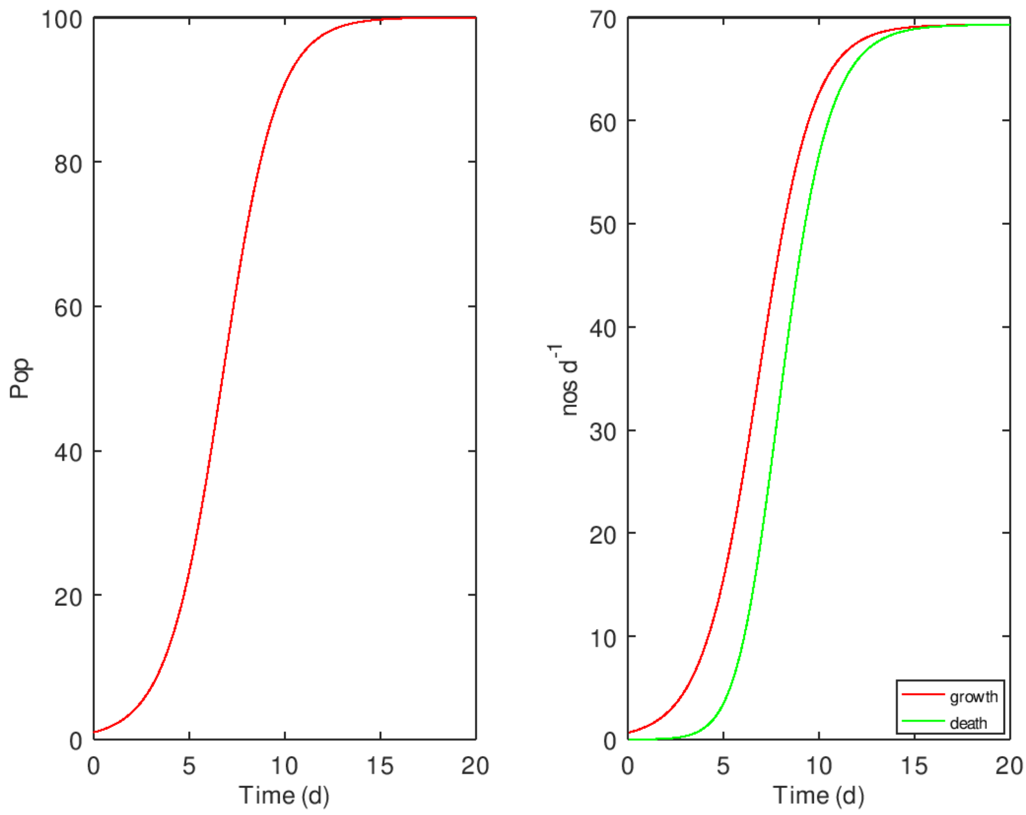


Fig. 6.3 The plot of *Dynamic Ecology* Chapter 6's logistic model as produced by *logistic.m*

6.2 Experimenting with the logistic model

To experiment with the model as detailed in section “6.3 Things to explore with the logistic equations” of *Dynamic Ecology* you need to change values of the model constants in file “func_logistic.m” (Fig. 6.4). You can refer to the comments (lines prepended with a hashtag and coloured in green) next to the variable names to understand what each variable corresponds to. Specifically, you need to play with the *K* and *r* parameters on lines 24 and 25 respectively.

If you make a mistake, you can always undo/redo using the arrow buttons just above the Octave’s Editor Window, and if you cannot resolve the problem, just download a new copy of the original GNU Octave model, and start over again.

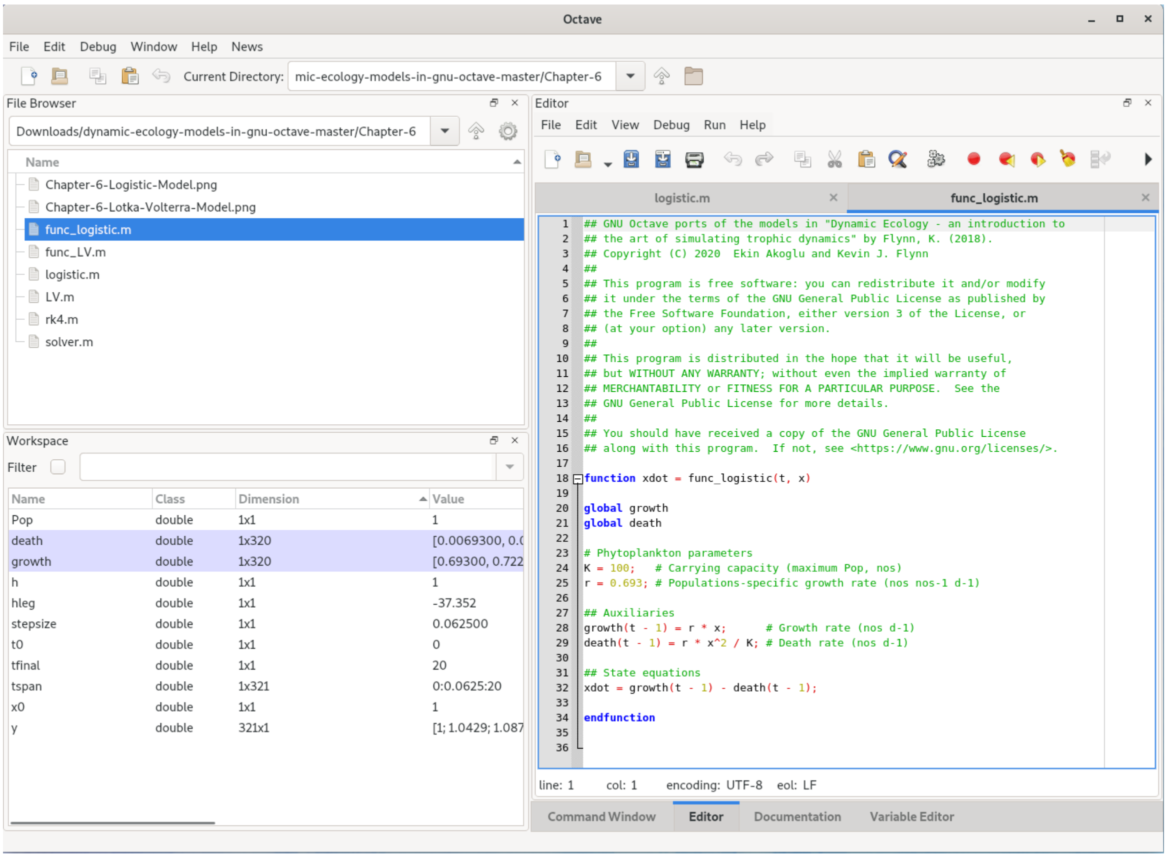


Fig. 6.4 The derivative function of *Dynamic Ecology* Chapter 6's logistic model.

6.3 Running the Lotka-Volterra model

Double-click the "LV.m" file to open the Lotka-Volterra model script (Fig. 6.5).

The script file will be opened in the editor window of GNU Octave. As you will notice, the file includes a series of GNU Octave commands and comments (lines prepended by a hashtag and coloured in green) that explain what each line of the code corresponds to.

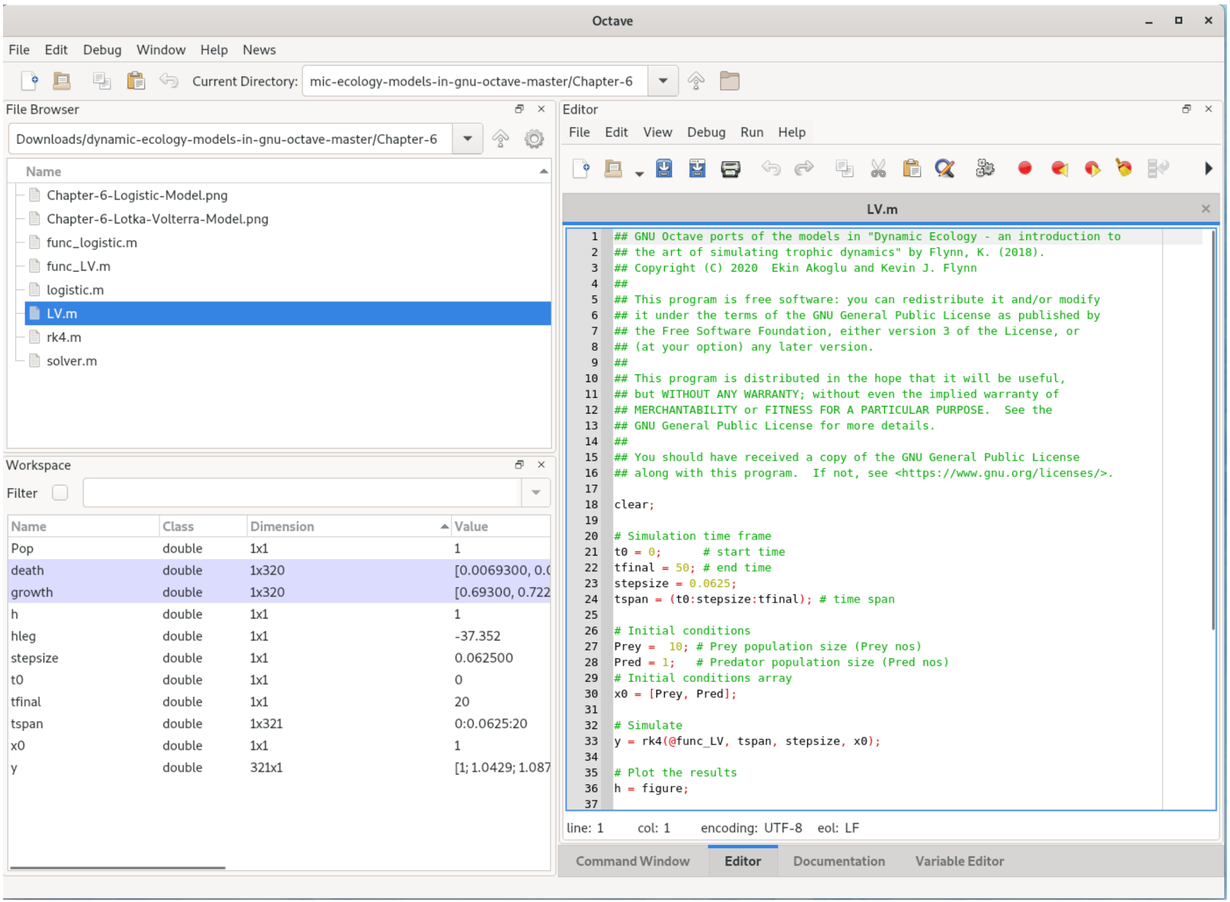


Fig. 6.5 Overview of *LV.m*

To run the "LV.m", hit F5 on your keyboard. Alternatively, you may switch back to the "Command Window" by using the tabs at the bottom of the right part of the main window and then enter the command "LV" and press "Enter" key while you are in the "Command Window".

The model will now run and produce a plot from simulation results once the simulation is completed (Fig. 6.6).

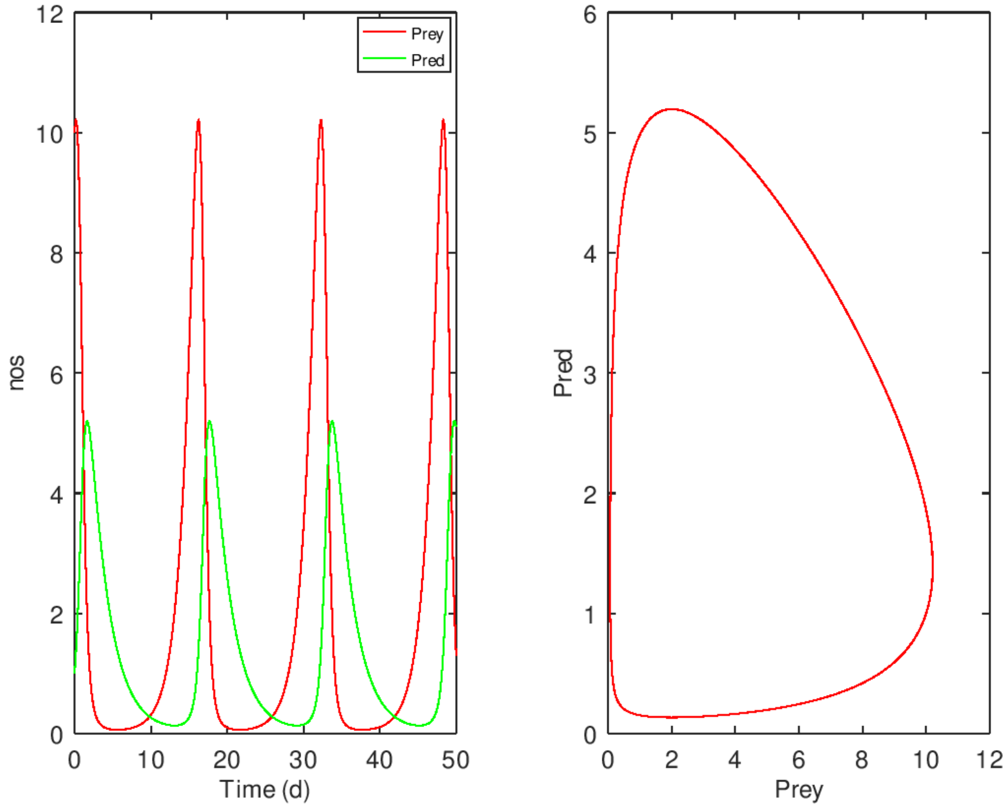


Fig. 6.6 The plot of *Dynamic Ecology*Chapter 6's Lotka-Volterra model as produced by *LV.m*

6.4 Experimenting with the Lotka-Volterra model

To experiment with the model as detailed in section “6.7 Things to explore with L-V models” of *Dynamic Ecology* you need to first change the integration scheme from 4th-order Runge-Kutta to Euler. For this purpose, edit the line 33 of the file “LV.m” from:

```
y = rk4(@func_LV, tspan, stepsize, x0);
```

to

```
y = solver(@func_LV, tspan, stepsize, x0);
```

Further, change the value of the “stepsize” variable on line 23 to experiment with the time step.

In addition, you need to change the values of the model constants in file “func_LV.m” (Fig. 6.7). You can refer to the comments (lines prepended with a hashtag and coloured in green) next to the variable names to understand what each variable corresponds to.

If you make a mistake, you can always undo/redo using the arrow buttons just above the Octave’s Editor Window, and if you cannot resolve the problem, just download a new copy of the original GNU Octave model, and start over again.

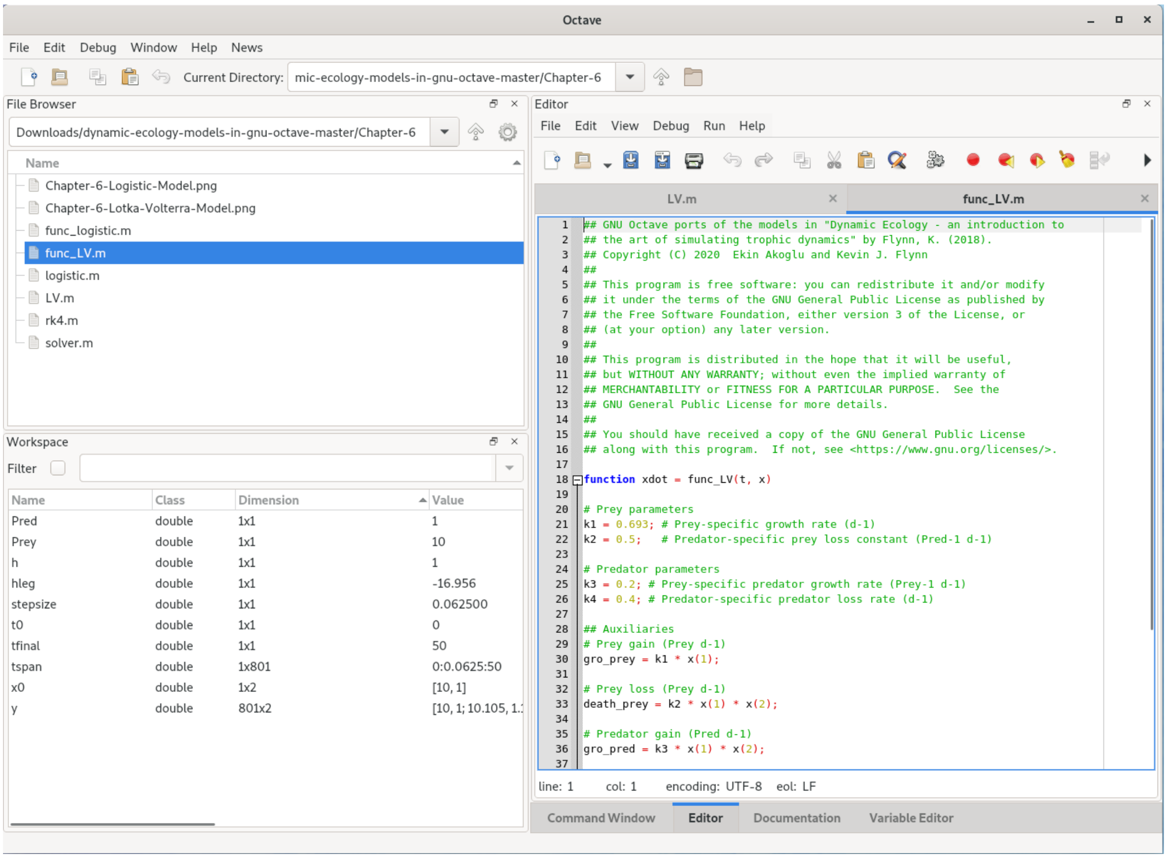


Fig. 6.7 The derivative function of *Dynamic Ecology* Chapter 6's Lotka-Volterra model.

6.5 GNU Octave code

This section, running over the following pages, provides a complete dump of the GNU Octave code as it appears in the download.

6.5.1 logistic.m

```
## GNU Octave ports of the models in "Dynamic Ecology - an introduction to
## the art of simulating trophic dynamics" by Flynn, K. (2018).
## Copyright (C) 2020 Ekin Akoglu and Kevin J. Flynn

## This program is free software: you can redistribute it and/or modify
## it under the terms of the GNU General Public License as published by
## the Free Software Foundation, either version 3 of the License, or
## (at your option) any later version.

## This program is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
## GNU General Public License for more details.

## You should have received a copy of the GNU General Public License
## along with this program. If not, see https://www.gnu.org/licenses/.

clear;

global growth
global death

# Simulation time frame
t0 = 0;      # start time
tfinal = 20; # end time
stepsize = 0.0625;
tspan = (t0:stepsize:tfinal); # time span

# Preallocate global arrays for speed
growth = zeros(1, length(tspan)-1);
death = zeros(1, length(tspan)-1);

# Initial conditions
Pop = 1; # nos Population size
x0 = Pop;

# Simulate
y = solver(@func_logistic, tspan, stepsize, x0);

# Plot the results
h = figure;

subplot(1, 2, 1);
plot(tspan, y, 'r');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
ylabel('Pop', 'FontSize', 12);

subplot(1, 2, 2);
plot(tspan(2:end), growth', 'r', tspan(2:end), death', 'g');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
```

```
ylabel('nos d^{-1}', 'FontSize', 12);
hleg = legend('growth', 'death', 'location', 'southeast');
set(hleg, 'FontSize', 8);

print(h, 'Chapter-6-Logistic-Model.png', '-dpng', '-color');
```

6.5.2 func_logistic.m

```
## GNU Octave ports of the models in "Dynamic Ecology - an introduction to
## the art of simulating trophic dynamics" by Flynn, K. (2018).
## Copyright (C) 2020 Ekin Akoglu and Kevin J. Flynn

## This program is free software: you can redistribute it and/or modify
## it under the terms of the GNU General Public License as published by
## the Free Software Foundation, either version 3 of the License, or
## (at your option) any later version.

## This program is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
## GNU General Public License for more details.

## You should have received a copy of the GNU General Public License
## along with this program. If not, see https://www.gnu.org/licenses/.

function xdot = func_logistic(t, x)

global growth
global death

# Phytoplankton parameters
K       =       100;    # Carrying capacity (maximum Pop, nos)
r = 0.693; # Populations-specific growth rate (nos nos-1 d-1)

## Auxiliaries
growth(t - 1) =r * x;       # Growth rate (nos d-1)
death(t - 1) = r * x^2 / K; # Death rate (nos d-1)

## State equations
xdot = growth(t - 1) - death(t - 1);

endfunction
```

6.5.3 LV.m

```
## GNU Octave ports of the models in "Dynamic Ecology - an introduction to
## the art of simulating trophic dynamics" by Flynn, K. (2018).
## Copyright (C) 2020 Ekin Akoglu and Kevin J. Flynn

## This program is free software: you can redistribute it and/or modify
## it under the terms of the GNU General Public License as published by
## the Free Software Foundation, either version 3 of the License, or
## (at your option) any later version.

## This program is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
## GNU General Public License for more details.

## You should have received a copy of the GNU General Public License
## along with this program. If not, see https://www.gnu.org/licenses/.

clear;

# Simulation time frame
t0 = 0;      # start time
tfinal = 50; # end time
stepsize = 0.0625;
tspan = (t0:stepsize:tfinal); # time span

# Initial conditions
Prey =  10; # Prey population size (Prey nos)
Pred = 1;   # Predator population size (Pred nos)
# Initial conditions array
x0 = [Prey, Pred];

# Simulate
y = rk4(@func_LV, tspan, stepsize, x0);

# Plot the results
h = figure;

subplot(1, 2, 1);
plot(tspan, y(:, 1), 'r', tspan, y(:, 2), 'g');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
ylabel('nos', 'FontSize', 12);
hleg = legend('Prey', 'Pred');
set(hleg, 'FontSize', 8);

subplot(1, 2, 2);
plot(y(:, 1), y(:, 2), 'r');
set(gca,'FontSize',12);
xlabel('Prey', 'FontSize', 12);
ylabel('Pred', 'FontSize', 12);

print(h, 'Chapter-6-Lotka-Volterra-Model.png', '-dpng', '-color');
```

6.5.4 func_LV.m

```
## GNU Octave ports of the models in "Dynamic Ecology - an introduction to
## the art of simulating trophic dynamics" by Flynn, K. (2018).
## Copyright (C) 2020 Ekin Akoglu and Kevin J. Flynn

## This program is free software: you can redistribute it and/or modify
## it under the terms of the GNU General Public License as published by
## the Free Software Foundation, either version 3 of the License, or
## (at your option) any later version.

## This program is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
## GNU General Public License for more details.

## You should have received a copy of the GNU General Public License
## along with this program. If not, see https://www.gnu.org/licenses/.

function xdot = func_LV(t, x)

# Prey parameters
k1 = 0.693; # Prey-specific growth rate (d-1)
k2 = 0.5;   # Predator-specific prey loss constant (Pred-1 d-1)

# Predator parameters
k3 = 0.2; # Prey-specific predator growth rate (Prey-1 d-1)
k4 = 0.4; # Predator-specific predator loss rate (d-1)

## Auxiliaries
# Prey gain (Prey d-1)
gro_prey = k1 * x(1);

# Prey loss (Prey d-1)
death_prey = k2 * x(1) * x(2);

# Predator gain (Pred d-1)
gro_pred = k3 * x(1) * x(2);

# Predator loss (Pred d-1)
death_pred = k4 * x(2);

## State equations
# Prey
xdot(1, 1) = gro_prey - death_prey;

# Predator
xdot(1, 2) = gro_pred - death_pred;

endfunction
```

7. Dilutions Models in GNU Octave

This Chapter provides information on running the model in chapter 6 of *Dynamic Ecology* (Flynn 2018), running through each of the steps. It is assumed that you have installed the Octave interface (Chapter 2).

Very few real systems operate in a closed environment, akin to a culture system in a sealed flask. The most obvious feature of real environments is that they are not closed, that they have an exchange in and out of the system, transferring material with adjoining environments. Think of a lake ecosystem, for example, with inputs from rainwater, from rivers and off the land, and outputs to evaporation, leakage into the sediment, and outflowing rivers.

In laboratories, experiments are most easily conducted using a flask operating essentially as a sealed, closed, system. Alternatively, experiments may be conducted in a system called a chemostat. A chemostat is a vessel in which the liquid volume stays constant, with the flows of material in and out occurring at the same rate. Operation of a chemostat could be likened to a pond with streams entering and leaving but with the pond remaining of constant volume. In a commercial setting, the crop is harvested; this is a form of dilution.

There are two models in this chapter:

i) dilution, and
ii) harvest.

Please see chapter 7 in *Dynamic Ecology* for more contextual information, explanations for model construction, and (in the final sections of that chapter) ideas for experimenting and developing your models.

7.1 Running the dilution model

Navigate to the folder “Chapter-7” in the “File Browser” by double-clicking it. You will now see the script files and related figures of the model in the “File Browser” (Fig. 7.1).

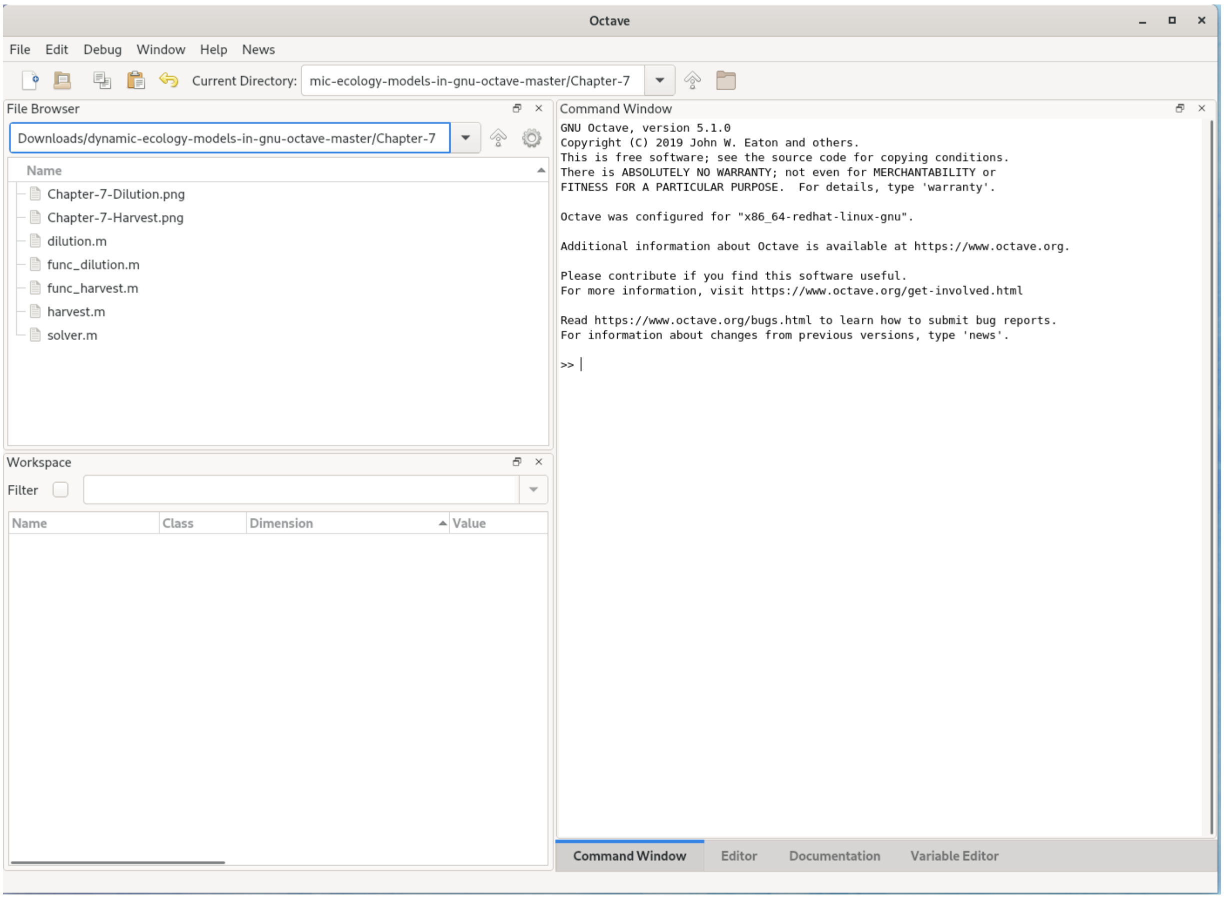


Fig. 7.1 Models of Chapter 7 in *Dynamic Ecology* and their related script files and figures.

Double-click the “dilution.m” file to open the dilution model script (Fig. 7.2).

The script file will be opened in the editor window of GNU Octave. As you will notice, the file includes a series of GNU Octave commands and comments (lines prepended by a hashtag and coloured in green) that explain what each line of the code corresponds to.

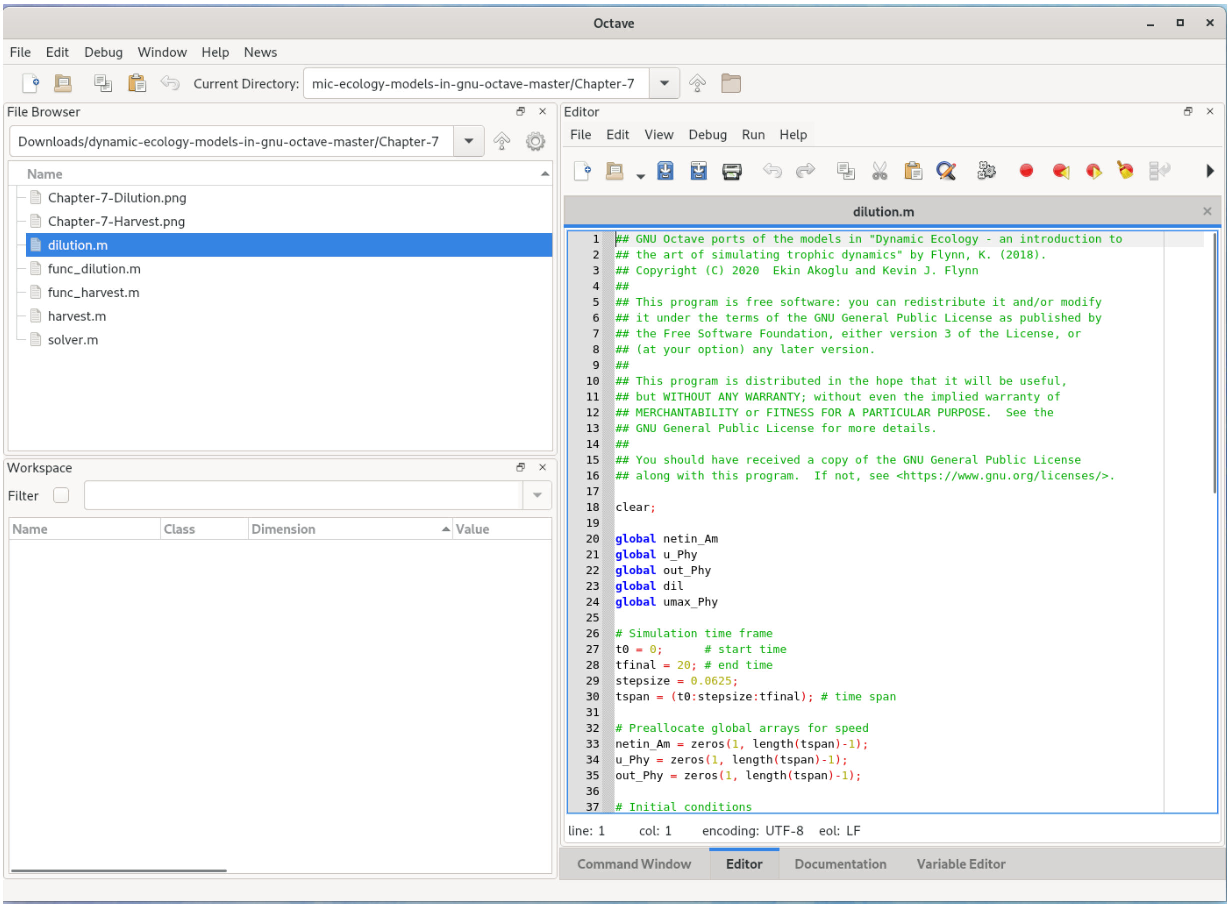


Fig. 7.2 Overview of *dilution.m*

To run the “dilution.m”, hit F5 on your keyboard. Alternatively, you may switch back to the “Command Window” by using the tabs at the bottom of the right part of the main window and then enter the command “dilution” and press “Enter” key while you are in the “Command Window”.

The model will now run and produce a plot from simulation results once the simulation is completed (Fig. 7.3).

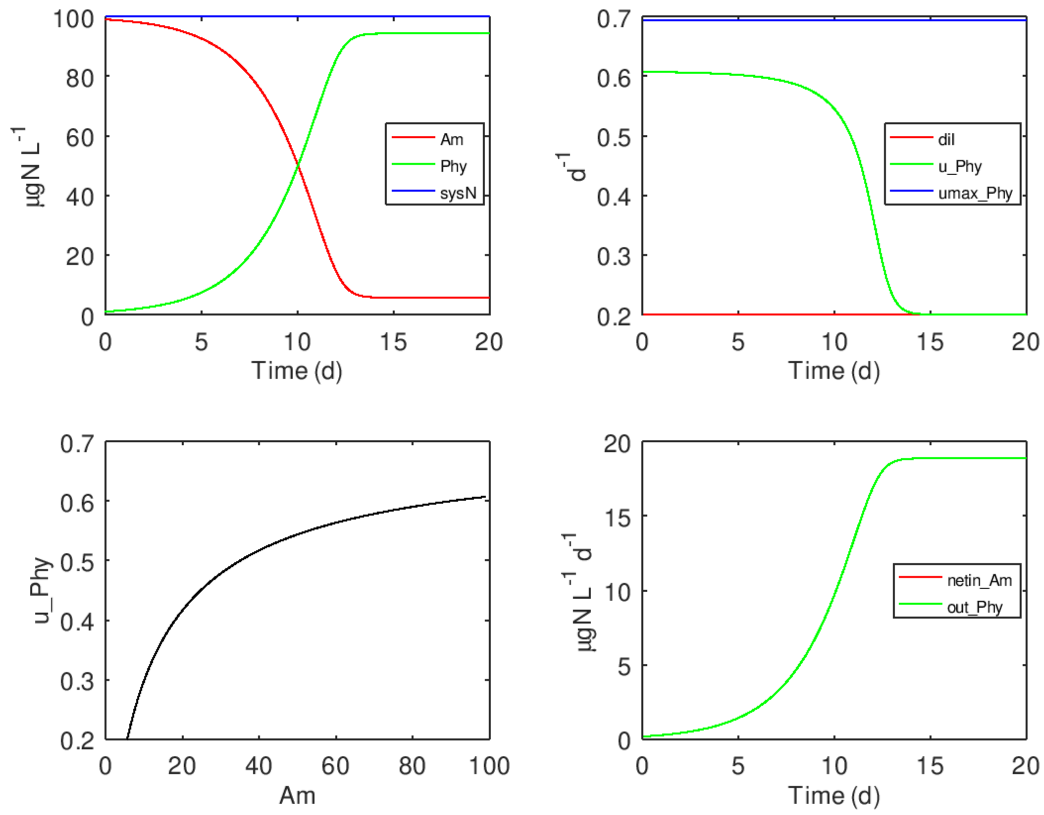


Fig. 7.3 The plot of *Dynamic Ecology* Chapter 7's dilution model as produced by *dilution.m*

7.2 Running the harvest model

Double-click the "harvest.m" file to open the harvest model script (Fig. 7.4).

The script file will be opened in the editor window of GNU Octave. As you will notice, the file includes a series of GNU Octave commands and comments (lines prepended by a hashtag and coloured in green) that explain what each line of the code corresponds to.

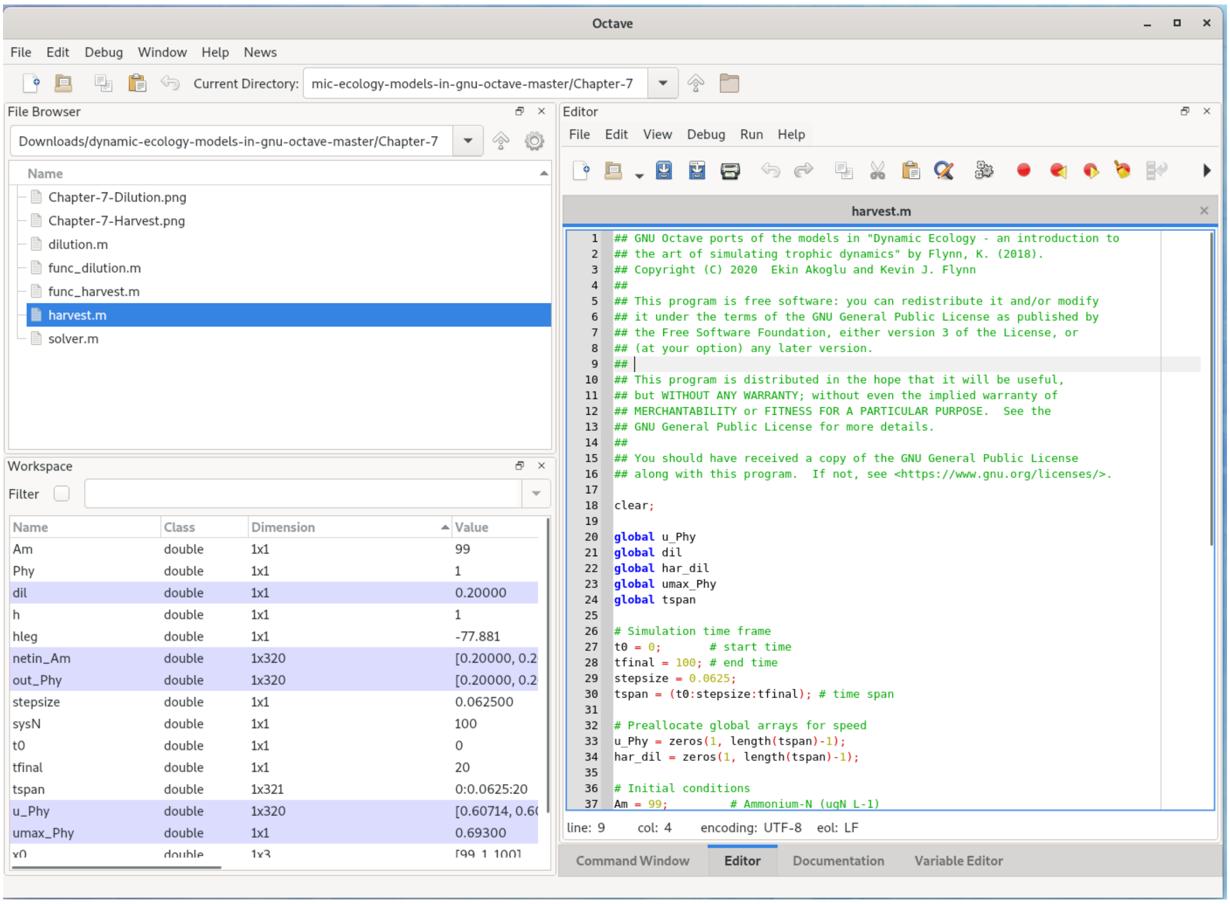


Fig. 7.4 Overview of harvest.*m*

To run the “harvest.m”, hit F5 on your keyboard. Alternatively, you may switch back to the “Command Window” by using the tabs at the bottom of the right part of the main window and then enter the command “harvest” and press “Enter” key while you are in the “Command Window”.

The model will now run and produce a plot from simulation results once the simulation is completed (Fig. 7.5).

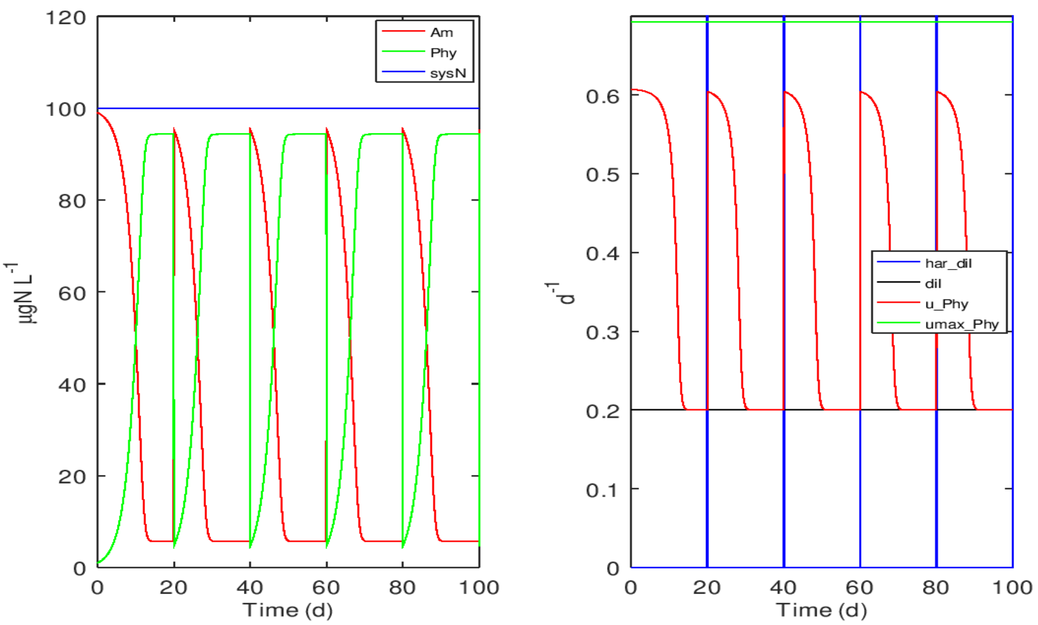


Fig. 7.5 The plot of *Dynamic Ecology* Chapter 7's harvest model as produced by harvest.*m.*

7.3 Experimenting with the models

To experiment with the models as detailed in section "7.6 Things to explore" of *Dynamic Ecology* please follow the instructions. You may need to refer back to previous chapters' models in both this volume and in *Dynamic Ecology*.

7.4 GNU Octave code

This section, running over the following pages, provides a complete dump of the GNU Octave code as it appears in the download.

7.4.1 dilution.m

```
## GNU Octave ports of the models in "Dynamic Ecology - an introduction to
## the art of simulating trophic dynamics" by Flynn, K. (2018).
## Copyright (C) 2020 Ekin Akoglu and Kevin J. Flynn

## This program is free software: you can redistribute it and/or modify
## it under the terms of the GNU General Public License as published by
## the Free Software Foundation, either version 3 of the License, or
## (at your option) any later version.

## This program is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
## GNU General Public License for more details.

## You should have received a copy of the GNU General Public License
## along with this program. If not, see https://www.gnu.org/licenses/.

clear;

global netin_Am
global u_Phy
global out_Phy
global dil
global umax_Phy

# Simulation time frame
t0 = 0;       # start time
tfinal = 20; # end time
stepsize = 0.0625;
tspan = (t0:stepsize:tfinal); # time span

# Preallocate global arrays for speed
netin_Am = zeros(1, length(tspan)-1);
u_Phy = zeros(1, length(tspan)-1);
out_Phy = zeros(1, length(tspan)-1);

# Initial conditions
Am = 99;           # Ammonium-N (ugN L-1)
Phy = 1;           # Phytoplankton biomass-N (ugN L-1)
sysN = Am + Phy; # System N-balance (ugN L-1)
# Initial conditions array
x0 = [Am Phy sysN];

# Simulate
y = solver(@func_dilution, tspan, stepsize, x0);

# Plot the results
h = figure;

subplot(2, 2, 1);
plot(tspan, y(:,1), 'r', tspan, y(:,2), 'g', tspan, y(:,3), 'b');
set(gca,'FontSize',12);
```

```
xlabel('Time (d)', 'FontSize', 12);
ylabel('\mugN L^{-1}', 'FontSize', 12);
hleg = legend('Am', 'Phy', 'sysN', 'location', 'east');
set(hleg, 'FontSize', 8);

subplot(2, 2, 2);
plot(tspan(2:end), repmat(dil, 1, length(tspan)-1), 'r', tspan(2:end), u_Phy',
'g', tspan(2:end), repmat(umax_Phy, 1, length(tspan)-1), 'b');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
ylabel('d^{-1}', 'FontSize', 12);
hleg = legend('dil', 'u\_Phy', 'umax\_Phy', 'location', 'east');
set(hleg, 'FontSize', 8);

subplot(2, 2, 3);
plot(y(2:end,1), u_Phy', 'k');
set(gca,'FontSize',12);
xlabel('Am', 'FontSize', 12);
ylabel('u\_Phy', 'FontSize', 12);

subplot(2, 2, 4);
plot(tspan(2:end), netin_Am, 'r', tspan(2:end), out_Phy, 'g');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
ylabel('\mugN L^{-1} d^{-1}', 'FontSize', 12);
hleg = legend('netin\_Am', 'out\_Phy', 'location', 'east');
set(hleg, 'FontSize', 8);

print(h, 'Chapter-7-Dilution.png', '-dpng', '-color');
```

7.4.2 func_dilution.m

```
## GNU Octave ports of the models in "Dynamic Ecology - an introduction to
## the art of simulating trophic dynamics" by Flynn, K. (2018).
## Copyright (C) 2020 Ekin Akoglu and Kevin J. Flynn

## This program is free software: you can redistribute it and/or modify
## it under the terms of the GNU General Public License as published by
## the Free Software Foundation, either version 3 of the License, or
## (at your option) any later version.

## This program is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
## GNU General Public License for more details.

## You should have received a copy of the GNU General Public License
## along with this program. If not, see https://www.gnu.org/licenses/.

function xdot = func_dilution(t, x)

global netin_Am
global u_Phy
global out_Phy
global dil
global umax_Phy

# Ammonium parameters
dil = 0.2;    # Dilution rate (L L-1 d-1)
ext_Am = 100; # Concentration of Am in external reservoir (ugN L-1)
Pause_t = 10; # Time between pauses (d)

# Phytoplankton parameters
umax_Phy = 0.693; # Phytoplankton maximum N-specific growth rate (gN (gN)-1 d-1)
kAm_Phy = 14;     # Half saturation constant for u_Phy (ugN L-1)

## Auxiliaries
# Inflow of ext_AM from reservoir (ugN L-1 d-1)
in_Am = ext_Am * dil;

# Outflow of AM from culture vessel (ugN L-1 d-1)
out_Am = x(1) * dil;

# Net input of AM (ugN L-1 d-1)
netin_Am(t - 1) = dil * (ext_Am - x(1));

# Phytoplankton N-specific growth rate (gN (gN)-1 d-1)
u_Phy(t - 1)=  umax_Phy * x(1) / (x(1) + kAm_Phy);

# Phytoplankton population growth rate (ugN L-1 d-1)
gro_Phy = u_Phy(t - 1) * x(2);

# Outflow of Phy from culture vessel (ugN L-1 d-1)
out_Phy(t - 1) = x(2) * dil;

## State equations
# Ammonium
xdot(1, 1) = in_Am - out_Am - gro_Phy;

# Phytoplankton
xdot(1, 2) = gro_Phy - out_Phy(t - 1);

# System
```

```
xdot(1, 3) = xdot(1, 1) + xdot(1, 2);

endfunction
```

7.4.3 harvest.m

```
## GNU Octave ports of the models in "Dynamic Ecology - an introduction to
## the art of simulating trophic dynamics" by Flynn, K. (2018).
## Copyright (C) 2020 Ekin Akoglu and Kevin J. Flynn

## This program is free software: you can redistribute it and/or modify
## it under the terms of the GNU General Public License as published by
## the Free Software Foundation, either version 3 of the License, or
## (at your option) any later version.

## This program is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
## GNU General Public License for more details.

## You should have received a copy of the GNU General Public License
## along with this program. If not, see https://www.gnu.org/licenses/.

clear;

global u_Phy
global dil
global har_dil
global umax_Phy
global tspan

# Simulation time frame
t0 = 0;       # start time
tfinal = 100; # end time
stepsize = 0.0625;
tspan = (t0:stepsize:tfinal); # time span

# Preallocate global arrays for speed
u_Phy = zeros(1, length(tspan)-1);
har_dil = zeros(1, length(tspan)-1);

# Initial conditions
Am = 99;          # Ammonium-N (ugN L-1)
Phy = 1;          # Phytoplankton biomass-N (ugN L-1)
sysN = Am + Phy; # System N-balance (ugN L-1)
# Initial conditions array
x0 = [Am Phy sysN];

# Simulate
y = solver(@func_harvest, tspan, stepsize, x0);

# Plot the results
h = figure;

subplot(1, 2, 1);
plot(tspan, y(:,1), 'r', tspan, y(:,2), 'g', tspan, y(:,3), 'b');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
ylabel('\mugN L^{-1}', 'FontSize', 12);
hleg = legend('Am', 'Phy', 'sysN');
set(hleg, 'FontSize', 8);

subplot(1, 2, 2);
plot(tspan(2:end), har_dil, 'b', tspan(2:end), repmat(dil, 1, length(tspan)-1),
'k', tspan(2:end), u_Phy', 'r', tspan(2:end), repmat(umax_Phy, 1, length(tspan)-
1), 'g');
ylim([0 0.7]);
```

```
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
ylabel('d^{-1}', 'FontSize', 12);
hleg = legend('har\_dil', 'dil', 'u\_Phy', 'umax\_Phy', 'location', 'east');
set(hleg, 'FontSize', 8);

print(h, 'Chapter-7-Harvest.png', '-dpng', '-color');
```

7.4.4 func_harvest.m

```
## GNU Octave ports of the models in "Dynamic Ecology - an introduction to
## the art of simulating trophic dynamics" by Flynn, K. (2018).
## Copyright (C) 2020 Ekin Akoglu and Kevin J. Flynn

## This program is free software: you can redistribute it and/or modify
## it under the terms of the GNU General Public License as published by
## the Free Software Foundation, either version 3 of the License, or
## (at your option) any later version.

## This program is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
## GNU General Public License for more details.

## You should have received a copy of the GNU General Public License
## along with this program. If not, see https://www.gnu.org/licenses/.

function xdot = func_harvest(t, x)

global u_Phy
global dil
global har_dil
global umax_Phy
global tspan

# Ammonium parameters
dil = 0.2;        # Dilution rate (L L-1 d-1)
har_f = 20;       # Frequency of harvesting (d)
har_pc = 0.95;    # Proportion harvested at frequency of har_f (dl)
ext_Am = 100;     # Concentration of Am in external reservoir (ugN L-1)
Pause_time = 20; # Time between pauses (d)

# Phytoplankton parameters
umax_Phy = 0.693; # Phytoplankton maximum N-specific growth rate (gN (gN)-1 d-1)
kAm_Phy = 14;     # Half saturation constant for u_Phy (ugN L-1)

## Auxiliaries
if tspan(t) > 0
  mult1 = 1;
else
  mult1 = 0;
endif
if isequal(mod(tspan(t), har_f), 0)
  mult2 = 1;
else
  mult2 = 0;
endif
# Harvesting dilution rate (d-1)
har_dil(t - 1) = mult1 * mult2 * har_pc / 0.0625;

# Total dilution rate (d-1)
time_dil = dil + har_dil(t - 1);

# Inflow of ext_AM from reservoir (ugN L-1 d-1)
in_Am = ext_Am * time_dil;

# Outflow of AM from culture vessel (ugN L-1 d-1)
out_Am = x(1) * time_dil;

# Phytoplankton N-specific growth rate (gN (gN)-1 d-1)
u_Phy(t - 1)=  umax_Phy * x(1) / (x(1) + kAm_Phy);
```

```
# Phytoplankton population growth rate (ugN L-1 d-1)
gro_Phy = u_Phy(t - 1) * x(2);

# Outflow of Phy from culture vessel (ugN L-1 d-1)
out_Phy = x(2) * time_dil;

## State equations
# Ammonium
xdot(1, 1) = in_Am - out_Am - gro_Phy;

# Phytoplankton
xdot(1, 2) = gro_Phy - out_Phy;

# System
xdot(1, 3) = xdot(1, 1) + xdot(1, 2);

endfunction
```

8. Light Limitation Model in GNU Octave

This Chapter provides information on running the model in chapter 8 of *Dynamic Ecology* (Flynn 2018), running through each of the steps. It is assumed that you have installed the Octave interface (Chapter 2).

In previous models we have described the control of phytoplankton growth through single nutrient limitation. In reality, most often light at least co-limits phototrophic growth. Sometimes that is due to too much light, which then causes photodamage and/or photoinhibition, and perhaps even kills cells. Typically, though, the limitation is due to a lack of light. Furthermore, there is a positive feedback interaction involved with low-light limitation because through the process of photoacclimation (commonly, though incorrectly, referred to as “shade-adaptation”, for example in reference to house plants) the individual organism becomes more heavily pigmented.

In crude terms each photoautotrophic organism becomes greener as it acclimates to capture the decreasing number of photons available, and so the ratio of chlorophyll to biomass, which we may describe as Chl:C, increases towards a maximum. The consequence of each member of the phytoplankton population becoming more densely pigmented, plus the increase in the population size, rapidly leads to a decrease in the amount of energy being available to the individual, and hence to a decrease in specific growth rate.

The process of light limitation of growth can be seen to have several facets. The surface irradiance may itself, even for cells at the water surface, be too low to permit high rates of photosynthesis. And, of course, light varies over the course of the day, with cloud cover and with the day-night cycle. Then, as the population grows, the light available to the individual declines as the sum total of pigment increases. And then we add in the aforementioned photoacclimation, where the ratio of chlorophyll pigment to biomass changes.

Here, although we shall consider the simplest scenario in which we ignore photoacclimation (assuming a set fixed Chl:C), we will explore how nutrient loading, mixing depth of the environment and irradiance all interact to affect the emergent growth rates of the individual and of the population.

Please see chapter 8 in *Dynamic Ecology* for more contextual information, explanations for model construction, and (in the final sections of that chapter) ideas for experimenting and developing your models.

8.1 Running the model

Navigate to the folder “Chapter-8” in the “File Browser” by double-clicking it. You will now see the script files and related figures of the model in the “File Browser” (Fig. 8.1).

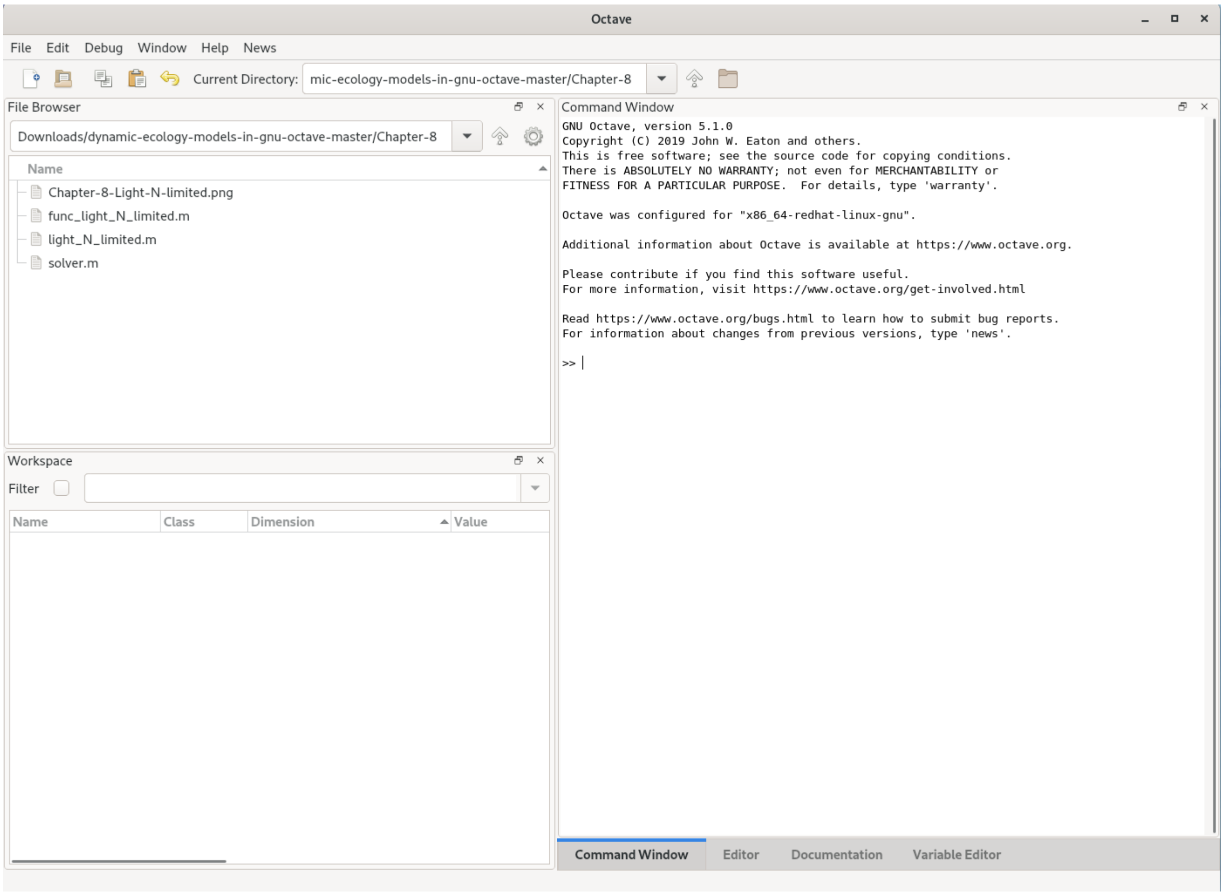


Fig. 8.1 Model of Chapter 8 in *Dynamic Ecology* and its related script files and figures.

Double-click the “light_N_limited.m” file to open it (Fig. 8.2).

The script file will be opened in the editor window of GNU Octave. As you will notice, the file includes a series of GNU Octave commands and comments (lines prepended by a hashtag and coloured in green) that explain what each line of the code corresponds to.

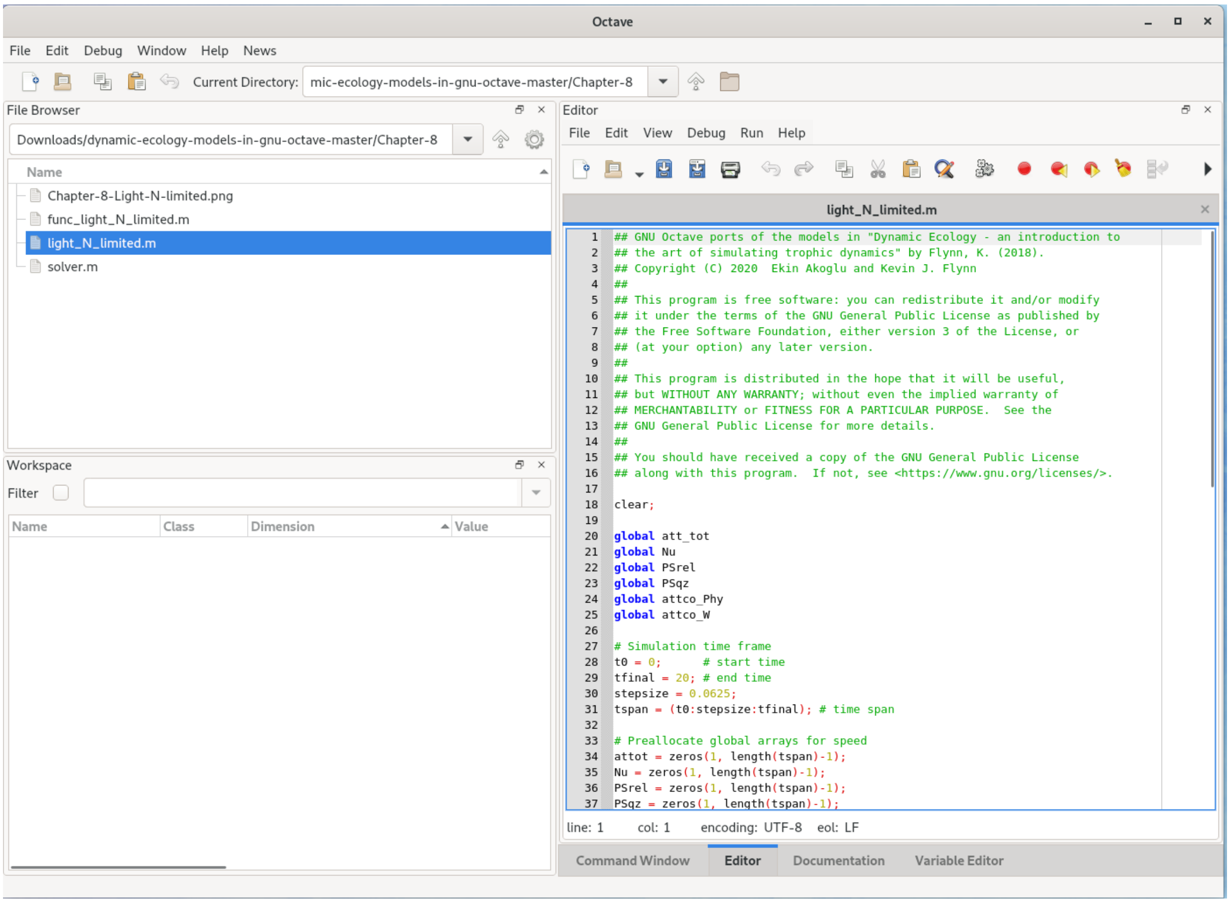


Fig. 8.2 Overview of *light_N_limited.m*

To run the "light_N_limited.m", hit F5 on your keyboard. Alternatively, you may switch back to the "Command Window" by using the tabs at the bottom of the right part of the main window and then enter the command "light_N_limited" and press "Enter" key while you are in the "Command Window".

The model will now run and produce a plot from simulation results once the simulation is completed (Fig. 8.3).

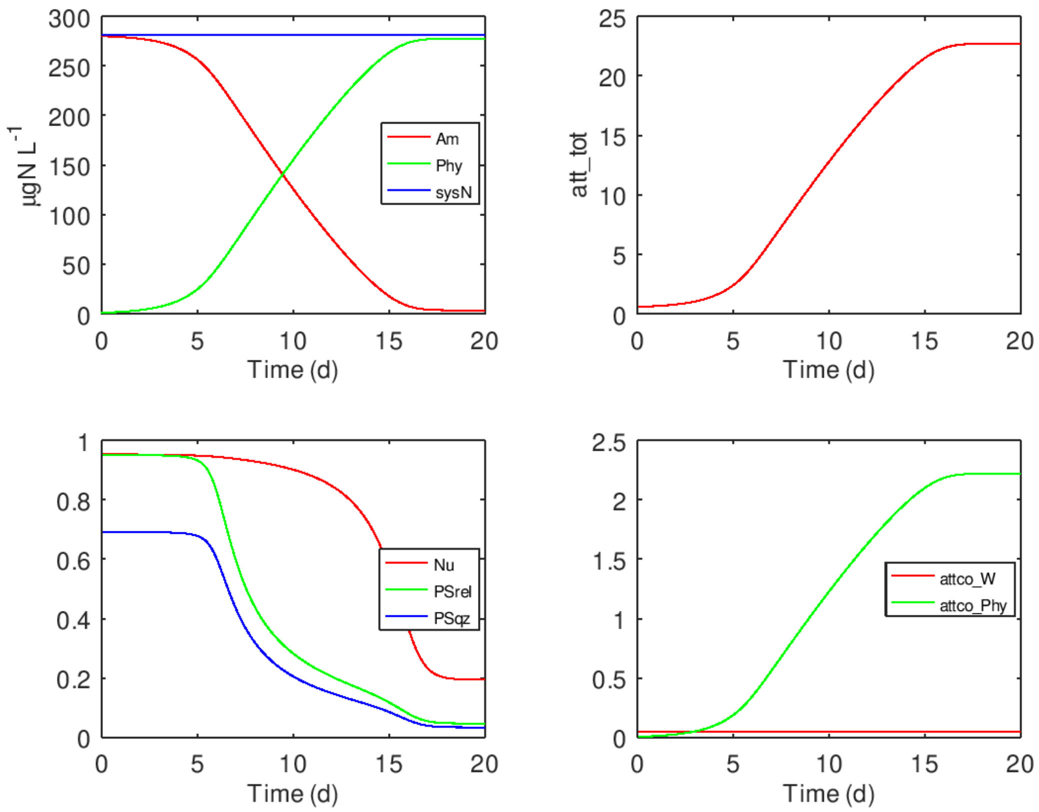


Fig. 8.3 The plot of *Dynamic Ecology* Chapter 8's model as produced by *light_N_limited.m*

8.2 Experimenting with the model

To experiment with the model as detailed in section "8.8 Things to explore" of *Dynamic Ecology* you need to change values of the model constants in file "func_light_N_limited.m" (Fig. 8.4). You can refer to the comments (lines prepended with a hashtag and coloured in green) next to the variable names to understand what each variable corresponds to. Further, you may need to refer back to previous chapters' models.

If you make a mistake, you can always undo/redo using the arrow buttons just above the Octave's Editor Window, and if you cannot resolve the problem, just download a new copy of the original GNU Octave model, and start over again.

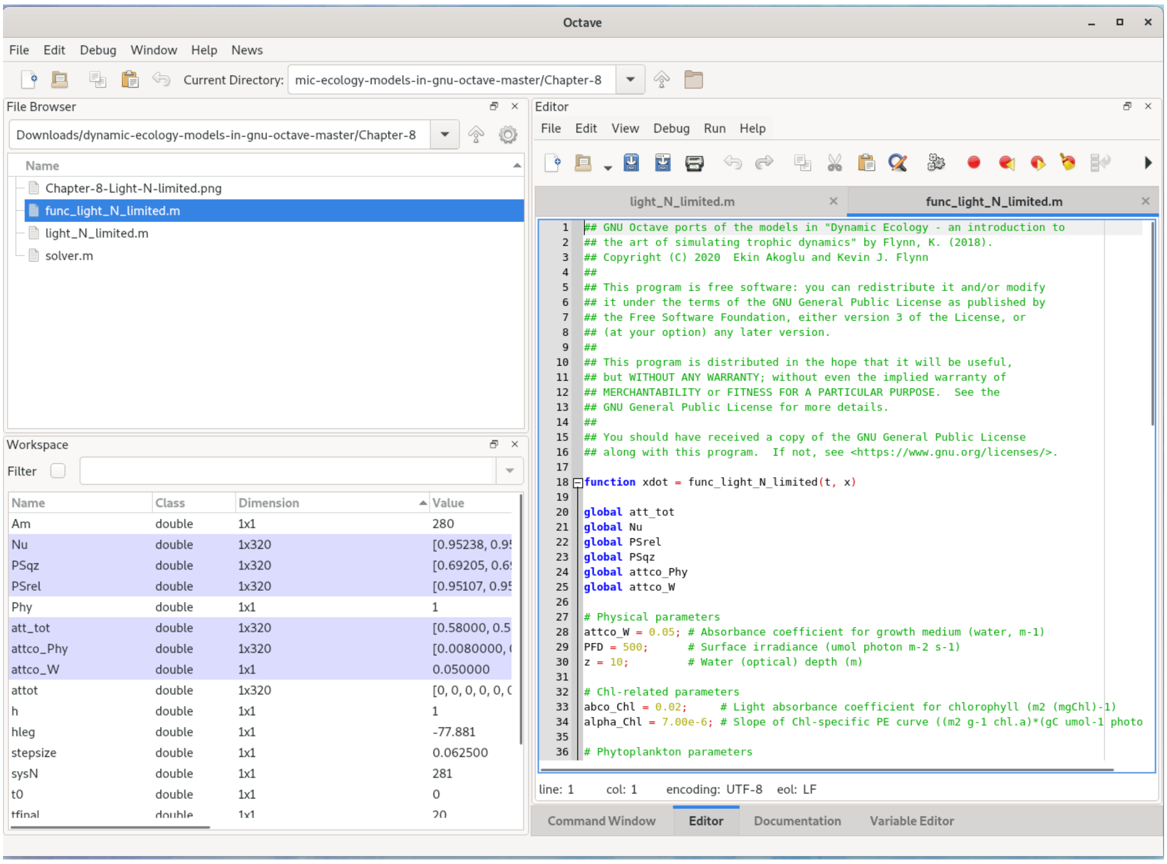


Fig. 8.4 The derivative function of *Dynamic Ecology* Chapter 8's model.

8.3 GNU Octave code

This section, running over the following pages, provides a complete dump of the GNU Octave code as it appears in the download.

8.3.1 light_N_limited.m

```
## GNU Octave ports of the models in "Dynamic Ecology - an introduction to
## the art of simulating trophic dynamics" by Flynn, K. (2018).
## Copyright (C) 2020 Ekin Akoglu and Kevin J. Flynn

## This program is free software: you can redistribute it and/or modify
## it under the terms of the GNU General Public License as published by
## the Free Software Foundation, either version 3 of the License, or
## (at your option) any later version.

## This program is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
## GNU General Public License for more details.

## You should have received a copy of the GNU General Public License
## along with this program. If not, see https://www.gnu.org/licenses/.

clear;

global att_tot
global Nu
global PSrel
global PSqz
global attco_Phy
global attco_W

# Simulation time frame
t0 = 0;       # start time
tfinal = 20; # end time
stepsize = 0.0625;
tspan = (t0:stepsize:tfinal); # time span

# Preallocate global arrays for speed
attot = zeros(1, length(tspan)-1);
Nu = zeros(1, length(tspan)-1);
PSrel = zeros(1, length(tspan)-1);
PSqz = zeros(1, length(tspan)-1);
attco_Phy = zeros(1, length(tspan)-1);

# Initial conditions
Am = 14 * 20;     # Ammonium-N concentration (ugN L-1)
Phy = 1;          # Phytoplankton biomass-N concentration (ugN L-1)
sysN = Am + Phy; # System N-balance (ugN L-1)
# Initial conditions array
x0 = [Am Phy sysN];

# Simulate
y = solver(@func_light_N_limited, tspan, stepsize, x0);

# Plot the results
h = figure;
```

```
subplot(2, 2, 1);
plot(tspan, y(:,1), 'r', tspan, y(:,2), 'g', tspan, y(:,3), 'b');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
ylabel('\mugN L^{-1}', 'FontSize', 12);
hleg = legend('Am', 'Phy', 'sysN', 'location', 'east');
set(hleg, 'FontSize', 8);

subplot(2, 2, 2);
plot(tspan(2:end), att_tot', 'r');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
ylabel('att\_tot', 'FontSize', 12);

subplot(2, 2, 3);
plot(tspan(2:end), Nu, 'r', tspan(2:end), PSrel', 'g', tspan(2:end), PSqz, 'b');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
hleg = legend('Nu', 'PSrel', 'PSqz', 'location', 'east');
set(hleg, 'FontSize', 8);

subplot(2, 2, 4);
plot(tspan(2:end), repmat(attco_W, 1, length(tspan)-1), 'r', tspan(2:end),
attco_Phy, 'g');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
hleg = legend('attco\_W', 'attco\_Phy', 'location', 'east');
set(hleg, 'FontSize', 8);

print(h, 'Chapter-8-Light-N-limited.png', '-dpng', '-color');
```

8.3.2 func_light_N_limited.m

```
## GNU Octave ports of the models in "Dynamic Ecology - an introduction to
## the art of simulating trophic dynamics" by Flynn, K. (2018).
## Copyright (C) 2020 Ekin Akoglu and Kevin J. Flynn

## This program is free software: you can redistribute it and/or modify
## it under the terms of the GNU General Public License as published by
## the Free Software Foundation, either version 3 of the License, or
## (at your option) any later version.

## This program is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
## GNU General Public License for more details.

## You should have received a copy of the GNU General Public License
## along with this program. If not, see https://www.gnu.org/licenses/.

function xdot = func_light_N_limited(t, x)

global att_tot
global Nu
global PSrel
global PSqz
global attco_Phy
global attco_W

# Physical parameters
attco_W = 0.05; # Absorbance coefficient for growth medium (water, m-1)
PFD = 500;      # Surface irradiance (umol photon m-2 s-1)
z = 10;         # Water (optical) depth (m)

# Chl-related parameters
abco_Chl = 0.02;     # Light absorbance coefficient for chlorophyll (m2 (mgChl)-
1)
alpha_Chl = 7.00e-6; # Slope of Chl-specific PE curve ((m2 g-1 chl.a)*(gC umol-1
photon))

# Phytoplankton parameters
umax_Phy = 0.693; # Phytoplankton maximum N-specific growth rate (gN (gN)-1 d-1)
kAm_Phy = 14;     # Half saturation constant for Am limitation (ugN L-1)
BR_Phy = 0.05;    # Scaler for basal respiration rate (dl)
ChlC_Phy = 0.06;  # Mass ratio content of chlorophyll:C in the phytoplankton
(gChl (gC)-1)
NC_Phy = 0.15;    # Mass ratio content of N-biomass:C in the phytoplankton (gN
(gC)-1)

## Auxiliaries
# Phytoplankton-N specific coefficient for light absorbance (m2 (mgN)-1)
abco_PhyN = abco_Chl * ChlC_Phy / NC_Phy;

# Attenuation coefficient to phytoplankton N-biomass (m-1)
attco_Phy(t - 1) = abco_PhyN * x(2);

# Total attenuation (dl)
att_tot(t - 1) = z * (attco_W + attco_Phy(t - 1));

# Negative exponent of total attenuation (dl)
exatt = exp(-att_tot(t - 1));

# Maximum gross photosynthetic rate required to enable u_Phy=umax_Phy (d-1)
PSmax = umax_Phy * (1 + BR_Phy);
```

```
# Quotient for N-status (dl)
Nu(t - 1) = x(1) / (x(1) + kAm_Phy);

# Maximum photosynthetic rate down-regulated by nutrient stress (d-1)
PSqmax = PSmax * Nu(t - 1);

# Specific slope of PE curve ((m2)*(umol-1 photon))
alpha_u = alpha_Chl * ChlC_Phy;

# Intermediate in depth-integrated photosynthesis rate (dl)
pytq = (alpha_u * PFD * 24 * 60 * 60) / PSqmax;

# Phytoplankton N-specific growth rate (d-1)
PSqz(t - 1) = PSqmax * (log(pytq + sqrt(1 + pytq^2)) - log(pytq * exatt + sqrt(1
+ (pytq * exatt)^2))) / att_tot(t - 1);

# Quotient for relative rate of PS (dl)
PSrel(t - 1) = PSqz(t - 1) / PSmax;

# Net growth rate (d-1)
u_Phy = PSqz(t - 1) - umax_Phy * BR_Phy;

# Phytoplankton population growth rate (ugN L-1 d-1)
gro_Phy = u_Phy * x(2);

## State equations
# Ammonium
xdot(1, 1) = -gro_Phy;

# Phytoplankton
xdot(1, 2) = gro_Phy;

# System
xdot(1, 3) = xdot(1, 1) + xdot(1, 2);

endfunction
```

9. Describing Light Model in GNU Octave

This Chapter provides information on running the model in chapter 9 of *Dynamic Ecology* (Flynn 2018), running through each of the steps. It is assumed that you have installed the Octave interface (Chapter 2).

In most considerations of ecology, the subjects of primary production, photosynthesis, and thence light, soon come to the fore. This is absolutely true of plankton ecology in the sunlit photic zone, but it is also true when considering the commercial exploitation of microalgae (and indeed of macroalgae). In Chapter 8 we explored the issue of light limitation for production, and how this was exacerbated by self-shading of and by the growing phytoplankton population within the water column. The surface irradiance in that instance was set by a constant, *PFD*. Here we explore describing *PFD* as a variable. What follows are not models as such; they are bolt-ons to models that allow light to be described in different ways.

Please see chapter 9 in *Dynamic Ecology* for more contextual information, explanations for model construction, and (in the final sections of that chapter) ideas for experimenting and developing your models.

9.1 Running the model

Navigate to the folder “Chapter-9” in the “File Browser” by double-clicking it. You will now see the script files and related figures of the model in the “File Browser” (Fig. 9.1).

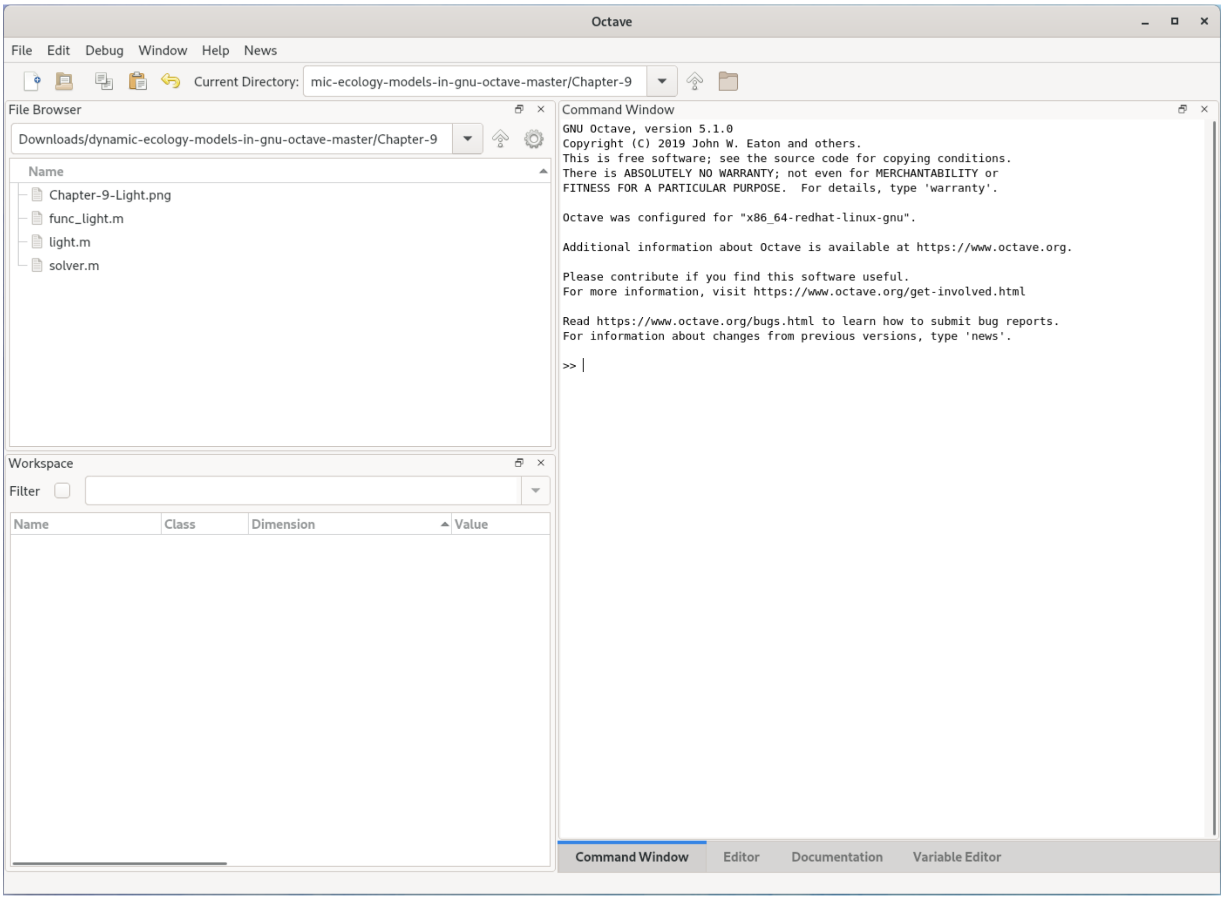


Fig. 9.1 Model of Chapter 9 in *Dynamic Ecology* and its related script files and figures.

Double-click the "light.m" file to open it (Fig. 9.2).

The script file will be opened in the editor window of GNU Octave. As you will notice, the file includes a series of GNU Octave commands and comments (lines prepended by a hashtag and coloured in green) that explain what each line of the code corresponds to.

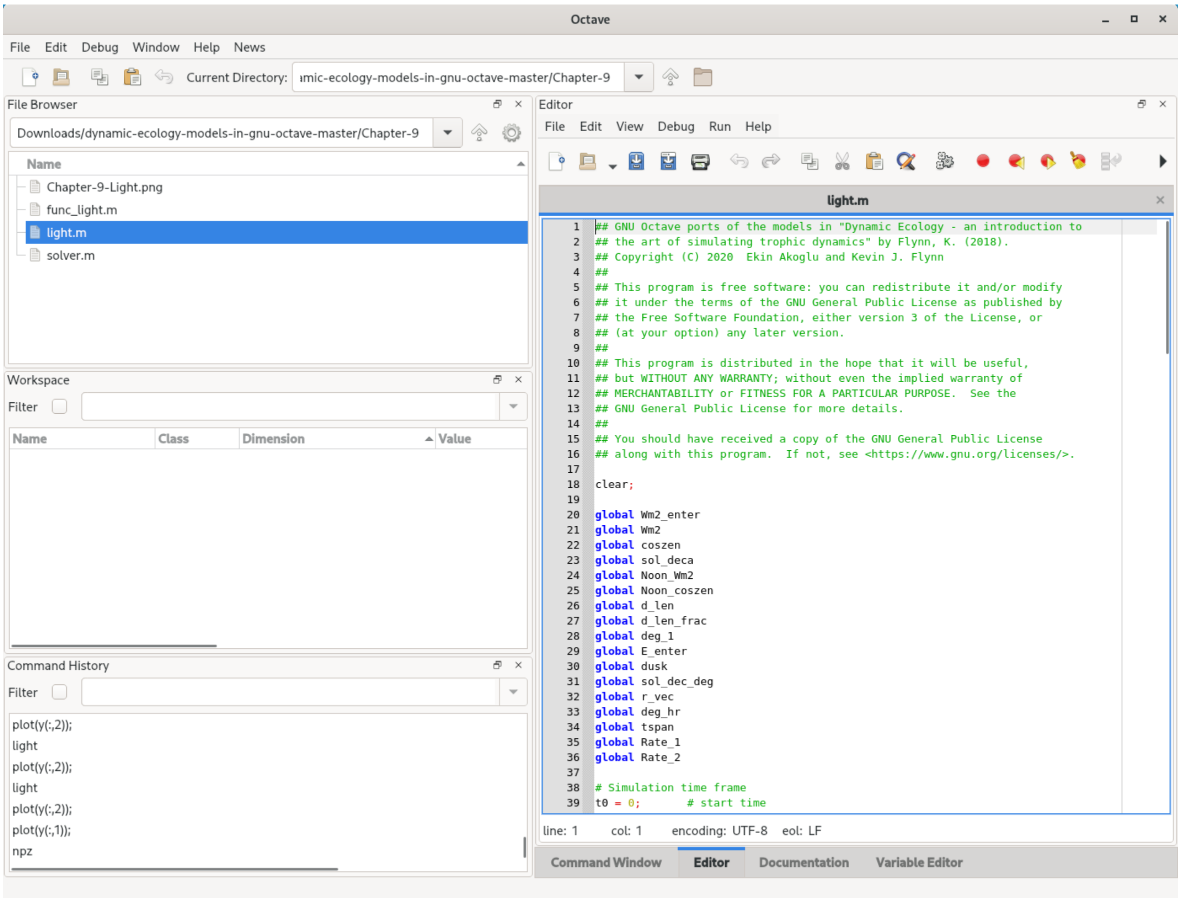


Fig. 9.2 Overview of *light.m*

To run the “light.m”, hit F5 on your keyboard. Alternatively, you may switch back to the “Command Window” by using the tabs at the bottom of the right part of the main window and then enter the command “light” and press “Enter” key while you are in the “Command Window”.

The model will now run and produce a plot from simulation results once the simulation is completed (Fig. 9.3).

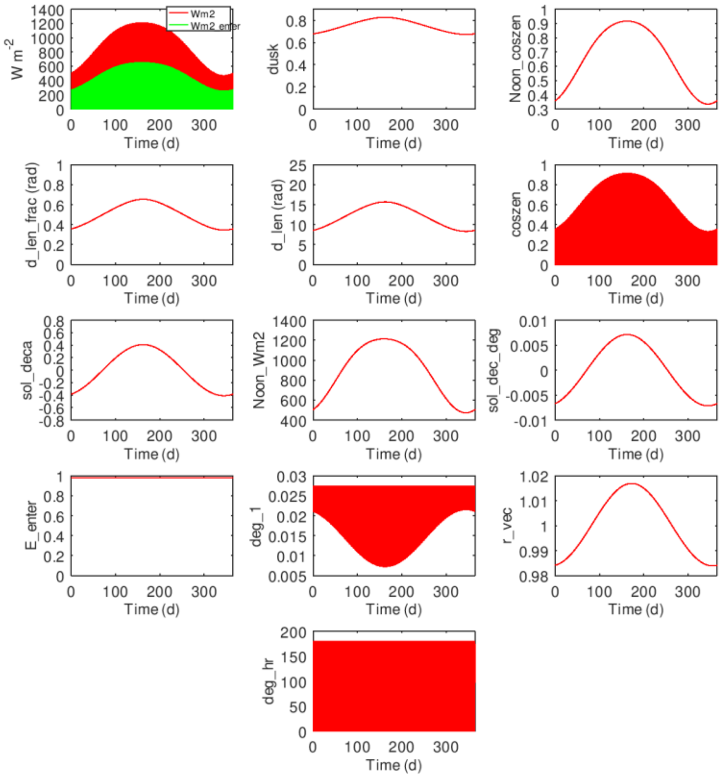


Fig. 9.3 The plot of *Dynamic Ecology* Chapter 9's model as produced by *light.m*

9.2 Experimenting with the model

To experiment with the model as detailed in section "9.5 Things to explore" of *Dynamic Ecology* you may need to refer back to previous chapters' models.

If you make a mistake, you can always undo/redo using the arrow buttons just above the Octave's Editor Window, and if you cannot resolve the problem, just download a new copy of the original GNU Octave model, and start over again.

9.3 GNU Octave code

This section, running over the following pages, provides a complete dump of the GNU Octave code as it appears in the download.

9.3.1 light.m

```
## GNU Octave ports of the models in "Dynamic Ecology - an introduction to
## the art of simulating trophic dynamics" by Flynn, K. (2018).
## Copyright (C) 2020 Ekin Akoglu and Kevin J. Flynn

## This program is free software: you can redistribute it and/or modify
## it under the terms of the GNU General Public License as published by
## the Free Software Foundation, either version 3 of the License, or
## (at your option) any later version.

## This program is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
## GNU General Public License for more details.

## You should have received a copy of the GNU General Public License
## along with this program. If not, see https://www.gnu.org/licenses/.

clear;

global Wm2_enter
global Wm2
global coszen
global sol_deca
global Noon_Wm2
global Noon_coszen
global d_len
global d_len_frac
global deg_1
global E_enter
global dusk
global sol_dec_deg
global r_vec
global deg_hr
global tspan
global Rate_1
global Rate_2

# Simulation time frame
t0 = 0;        # start time
tfinal = 365; # end time
stepsize = 0.015625;
tspan = (t0:stepsize:tfinal); # time span

# Preallocate global arrays for speed
Wm2_enter = zeros(1, length(tspan)-1);
Wm2 = zeros(1, length(tspan)-1);
coszen = zeros(1, length(tspan)-1);
sol_deca = zeros(1, length(tspan)-1);
Noon_Wm2 = zeros(1, length(tspan)-1);
Noon_coszen = zeros(1, length(tspan)-1);
d_len = zeros(1, length(tspan)-1);
d_len_frac = zeros(1, length(tspan)-1);
```

```
deg_1 = zeros(1, length(tspan)-1);
E_enter = zeros(1, length(tspan)-1);
dusk = zeros(1, length(tspan)-1);
sol_dec_deg = zeros(1, length(tspan)-1);
r_vec = zeros(1, length(tspan)-1);
deg_hr = zeros(1, length(tspan)-1);
Rate_1 = zeros(1, length(tspan)-1);
Rate_2 = zeros(1, length(tspan)-1);

# Initial conditions
cum_MJ_m2 = 0;     # Cummulative dose (MJ m-2)
DAY_avg_W_m2 = 0; # Average daily irradiance (Wm-2)
# Initial conditions array
x0 = [cum_MJ_m2 DAY_avg_W_m2];

# Simulate
y = solver(@func_light, tspan, stepsize, x0);

# Plot the results
h = figure;
set(h, 'Position', [0    50    900    950]);
set(h, 'PaperPositionMode', 'auto');

subplot(5, 3, 1);
plot(tspan(2:end), Wm2, 'r', tspan(2:end), Wm2_enter, 'g');
set(gca,'FontSize',12);
xlim([0 365]);
xlabel('Time (d)', 'FontSize', 12);
ylabel('W m^{-2}', 'FontSize', 12);
hleg = legend('Wm2', 'Wm2\_enter');
set(hleg, 'FontSize', 8);

subplot(5, 3, 2);
plot(tspan(2:end), dusk', 'r');
set(gca,'FontSize',12);
xlim([0 365]);
ylim([0 0.9]);
xlabel('Time (d)', 'FontSize', 12);
ylabel('dusk', 'FontSize', 12);

subplot(5, 3, 3);
plot(tspan(2:end), Noon_coszen, 'r');
set(gca,'FontSize',12);
xlim([0 365]);
xlabel('Time (d)', 'FontSize', 12);
ylabel('Noon\_coszen', 'FontSize', 12);

subplot(5, 3, 4);
plot(tspan(2:end), d_len_frac, 'r');
set(gca,'FontSize',12);
xlim([0 365]);
ylim([0 1.0]);
xlabel('Time (d)', 'FontSize', 12);
ylabel('d\_len\_frac (rad)', 'FontSize', 12);

subplot(5, 3, 5);
plot(tspan(2:end), d_len, 'r');
set(gca,'FontSize',12);
xlim([0 365]);
ylim([0 25]);
xlabel('Time (d)', 'FontSize', 12);
ylabel('d\_len (rad)', 'FontSize', 12);
```

```
subplot(5, 3, 6);
plot(tspan(2:end), coszen, 'r');
set(gca,'FontSize',12);
xlim([0 365]);
xlabel('Time (d)', 'FontSize', 12);
ylabel('coszen', 'FontSize', 12);

subplot(5, 3, 7);
plot(tspan(2:end), sol_deca, 'r');
set(gca,'FontSize',12);
xlim([0 365]);
xlabel('Time (d)', 'FontSize', 12);
ylabel('sol\_deca', 'FontSize', 12);

subplot(5, 3, 8);
plot(tspan(2:end), Noon_Wm2, 'r');
set(gca,'FontSize',12);
xlim([0 365]);
xlabel('Time (d)', 'FontSize', 12);
ylabel('Noon\_Wm2', 'FontSize', 12);

subplot(5, 3, 9);
plot(tspan(2:end), sol_dec_deg, 'r');
set(gca,'FontSize',12);
xlim([0 365]);
xlabel('Time (d)', 'FontSize', 12);
ylabel('sol\_dec\_deg', 'FontSize', 12);

subplot(5, 3, 10);
plot(tspan(2:end), E_enter, 'r');
set(gca,'FontSize',12);
xlim([0 365]);
ylim([0 1.0]);
xlabel('Time (d)', 'FontSize', 12);
ylabel('E\_enter', 'FontSize', 12);

subplot(5, 3, 11);
plot(tspan(2:end), deg_1, 'r');
set(gca,'FontSize',12);
xlim([0 365]);
xlabel('Time (d)', 'FontSize', 12);
ylabel('deg\_1', 'FontSize', 12);

subplot(5, 3, 12);
plot(tspan(2:end), r_vec, 'r');
set(gca,'FontSize',12);
xlim([0 365]);
xlabel('Time (d)', 'FontSize', 12);
ylabel('r\_vec', 'FontSize', 12);

subplot(5, 3, 14);
plot(tspan(2:end), deg_hr, 'r');
set(gca,'FontSize',12);
xlim([0 365]);
xlabel('Time (d)', 'FontSize', 12);
ylabel('deg\_hr', 'FontSize', 12);

print(h, 'Chapter-9-Light.png', '-dpng', '-color');
```

9.3.2 func_light.m

```
## GNU Octave ports of the models in "Dynamic Ecology - an introduction to
## the art of simulating trophic dynamics" by Flynn, K. (2018).
## Copyright (C) 2020 Ekin Akoglu and Kevin J. Flynn

## This program is free software: you can redistribute it and/or modify
## it under the terms of the GNU General Public License as published by
## the Free Software Foundation, either version 3 of the License, or
## (at your option) any later version.

## This program is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
## GNU General Public License for more details.

## You should have received a copy of the GNU General Public License
## along with this program. If not, see https://www.gnu.org/licenses/.

function xdot = func_light(t, x)

global Wm2_enter
global Wm2
global coszen
global sol_deca
global Noon_Wm2
global Noon_coszen
global d_len
global d_len_frac
global deg_1
global E_enter
global dusk
global sol_dec_deg
global r_vec
global deg_hr
global tspan
global Rate_1
global Rate_2

# Parameters
solar_const = 1368; # Solar constant irradiance (W m-2 = J/m2/s); maximum
irradiance to Earth from the sun (W m-2)
sw_JD = 1;          # Switch; if 0 then date is fixed to set_JD; if 1 then
increment with TIME (dl)
atmos_clar = 0.55;  # Corrects for atmospheric clarity (varies with lat, long  &
JD) (dl)
con_fact = 4.57;    # Converts W m-2 to PAR umol m-2 s-1 for cloud-less sky with
sun (dl)
lat = 47;           # Latitude (degrees-north)
set_JD = 0;         # Required fixed date (see sw_JD) (d)

## Auxiliaries
# Current time as fraction of day (dl)
frac_day = tspan(t - 1) - floor(tspan(t - 1));

# Current time as fraction of day in hours (hrs)
t24 = 24 * frac_day;

# Degrees of hour angle away from noon (default 12:00) (dl)
deg_hr(t - 1) =abs(12 - t24) * 15;

# Hour angle radians (rad)
r_hr =  deg_hr(t - 1) * pi / 180;
```

```
if sw_JD == 1
  mult1 = 1;
else
  mult1 = 0;
endif
if sw_JD == 0
  mult2 = 1;
else
  mult2 = 0;
endif
# Julian day; note the 10d offset (starting the year on 22nd December) (d)
JD = mult1 * 365 * (((tspan(t - 1) + 10) / 365) - floor(((tspan(t - 1) + 10) /
365))) + mult2 * set_JD;

# Solar declination angle (rad)
sol_deca(t - 1)= 23.45 * sin(2 * pi * (284 + JD) * 0.00274) * pi / 180;

# Latitude in radians (rad)
r_lat = lat * pi / 180;

# Cosine of zenith angle; positive values only accepted (dl)
coszen(t - 1) = max(sin(r_lat) * sin(sol_deca(t - 1)) + cos(r_lat) *
cos(sol_deca(t - 1)) * cos(r_hr), 0);

# Angle the sun makes with the vertical (solar zenith angle) (rad)
theta1 = acos(coszen(t - 1));

# Intermediate #1 in day length calculator (dl)
d_cal1 = -1 * tan(r_lat) * tan(sol_deca(t - 1));

if d_cal1 > -1
  mult1 = 1;
else
  mult1 = 0;
endif
if d_cal1 <= 1
  mult2 = 1;
else
  mult2 = 0;
endif
if d_cal1 <= -1
  mult3 = 1;
else
  mult3 = 0;
endif
if d_cal1 > 1
  mult4 = 1;
else
  mult4 = 0;
endif
# Intermediate #2 in day length calculator (dl)
d_cal2 = d_cal1 * mult1 * mult2 + -1 * mult3 + 1 * mult4;

# Day length at current Julian date (hr)
d_len(t - 1) = (2 * acos(d_cal2) * 12 / pi);

# Day length at current Julian date (d)
d_len_frac(t - 1) = d_len(t - 1) / 24;

# Time of dusk (d)
dusk(t - 1) = (0.5 + d_len_frac(t - 1) / 2);
```

```
# Angle the sun makes with the vertical (solar zenith angle)
deg_1(t - 1) = theta1 * deg2rad(1.0);

# Proportion of light incident with the water surface that is just under the
surface, accounting for reflectance (dl)
E_enter(t - 1) = 1 - (1.15e-06 * deg_1(t - 1)^3 - 69.1340e-06 * deg_1(t - 1)^2 +
0.001 * deg_1(t - 1) + 0.0187);

# Photon m-2 s-1 PFD just under surface (umol)
nat_PFD = x(1) * con_fact;

# Daily irradiance (kJ m-2 d-1)
kJ_m2_day = x(2) * 86400 / 1000;

# Value of coszen at noon (hence COS(0) at end of definition) (dl)
Noon_coszen(t - 1) = max(sin(r_lat) * sin(sol_deca(t - 1)) + cos(r_lat) *
cos(sol_deca(t - 1)) * cos(0), 0);

# Earth radius vector
r_vec(t - 1) = 1 / (1 + 0.033 * cos(2 * pi * JD * 0.00274))^0.5;

# Maximum irradiance (at noon) on this Julian date (W m-2)
Noon_Wm2(t - 1) = solar_const / r_vec(t - 1) / r_vec(t - 1) * Noon_coszen(t -
1);

if coszen(t - 1) > 0
  mult1 = 1;
else
  mult1 = 0;
endif
# Irradiance at given hour and day; W m-2 [W = J s-1; i.e. J/m2/s] (W m-2)
Wm2(t - 1) = solar_const / r_vec(t - 1) / r_vec(t - 1) * coszen(t - 1) * mult1;

# Light actually entering water (just under surface), accounting for reflectance
(W m-2)
Wm2_enter(t - 1) = Wm2(t - 1) * E_enter(t - 1) * atmos_clar;

# Intermediate calc
Rate_1(t - 1) = Wm2_enter(t - 1);

# Intermediate calc; to average over 1 time unit (day)
if t < 10
  Rate_2(t - 1) = 0;
else
  Rate_2(t - 1) = Rate_1(t - 9);
endif

# Sets time for 2nd year to zero at t = 365
simtime = tspan(t - 1) - 365;

# Solar declination angle (degrees)
sol_dec_deg(t - 1) = sol_deca(t - 1) * deg2rad(1.0);

## State equations
# Cumulative dose
xdot(1, 1) = Wm2_enter(t - 1);

# Average daily irradiance
xdot(1, 2) = Rate_1(t - 1) - Rate_2(t - 1);

endfunction
```

10. Pond Life Model in GNU Octave

This Chapter provides information on running the model in chapter 10 of *Dynamic Ecology* (Flynn 2018), running through each of the steps. It is assumed that you have installed the Octave interface (Chapter 2).

One of the simplest operational structures for a real planktonic ecosystem is the humble pond. Ponds exist widely in terrestrial ecosystems as small, often transient, pools of water. They become inoculated with phytoplankton and zooplankton from cysts in the soil or sediment, blown in by the wind, or carried in by animals, such as waterfowl. Ponds also exist as artificial structures in support of aquaculture or for the commercial production of microalgal biomass. And, of course, many contain fish, be they wild, ornamental or for food. Ponds are also important features of polar regions, developing as the ice melts.

In this chapter we will consider the pond as a habitat for growing plankton, subjected to inflows of water carrying nutrients, leakage, water evaporation and overflows. In some ways this chapter could be viewed as an extension to Chapter 7 on Dilutions, but there is a distinct difference in the core of the model structure. While hitherto we have considered state variables describing nutrient and biomass concentrations (e.g., as μgN L^{-1}), in the model described here the state variables describe absolute amounts; the concentrations themselves are thus auxiliaries. That is to say, we have state variables for the volume of water, the total mass of N in the pond as nutrient, and as plankton biomass.

Please see chapter 10 in *Dynamic Ecology* for more contextual information, explanations for model construction, and (in the final sections of that chapter) ideas for experimenting and developing your models.

10.1 Running the model

Navigate to the folder “Chapter-10” in the “File Browser” by double-clicking it. You will now see the script files and related figures of the model in the “File Browser” (Fig. 10.1).

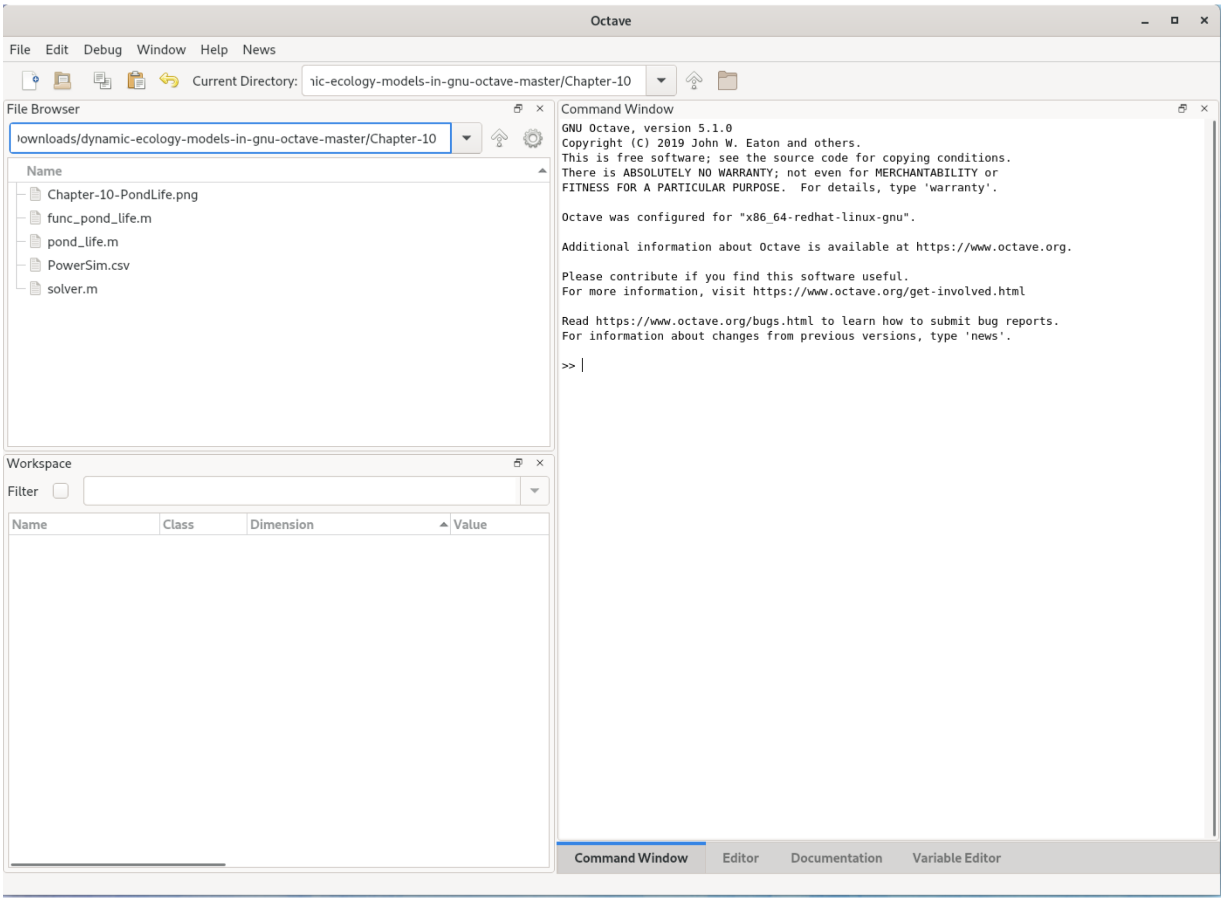


Fig. 10.1 Model of Chapter 10 in *Dynamic Ecology* and its related script files and figures.

Double-click the “pond_life.m” file to open it (Fig. 10.2).

The script file will be opened in the editor window of GNU Octave. As you will notice, the file includes a series of GNU Octave commands and comments (lines prepended by a hashtag and coloured in green) that explain what each line of the code corresponds to.

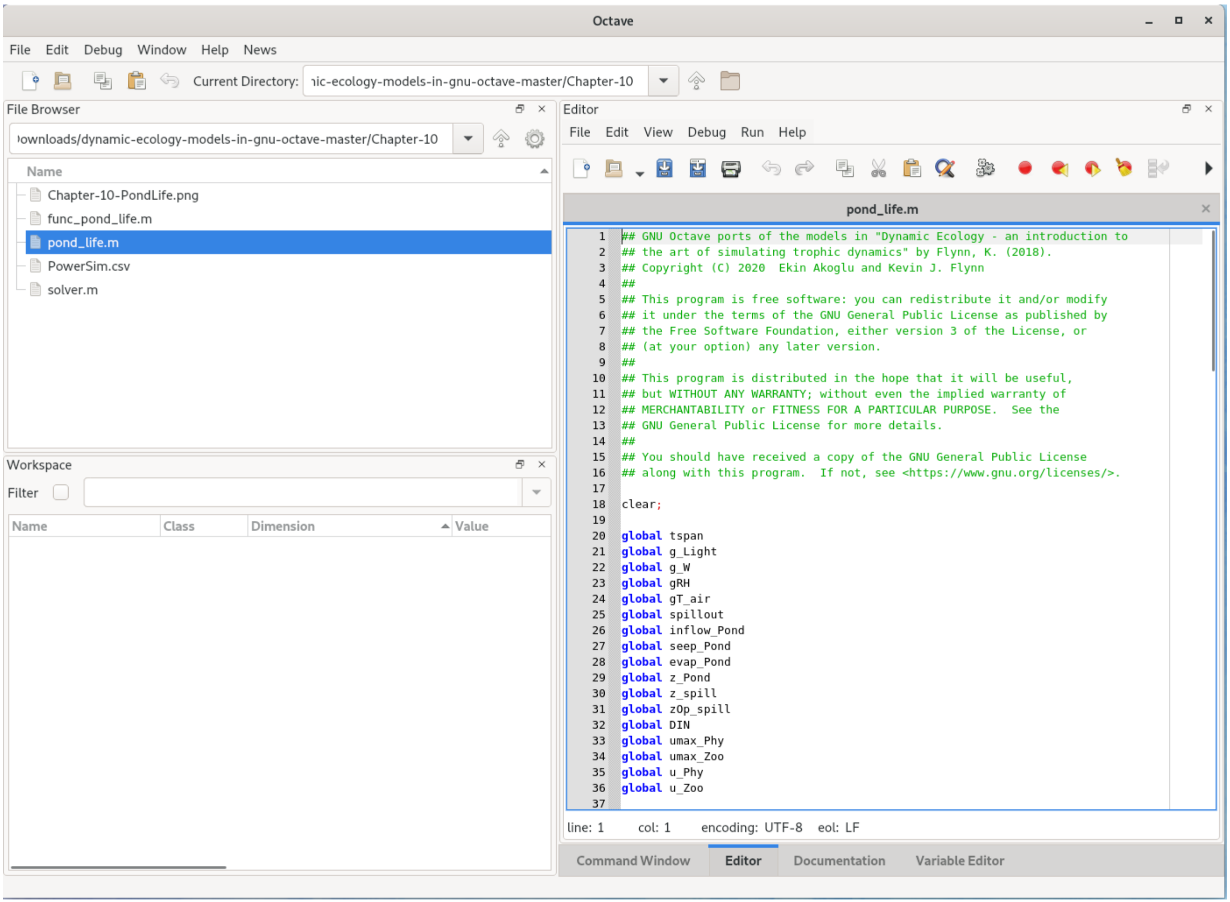


Fig. 10.2 Overview of *pond_life.m*

To run the “pond_life.m”, hit F5 on your keyboard. Alternatively, you may switch back to the “Command Window” by using the tabs at the bottom of the right part of the main window and then enter the command “pond_life” and press “Enter” key while you are in the “Command Window”.

The model will now run and produce a plot from simulation results once the simulation is completed (Fig. 10.3).

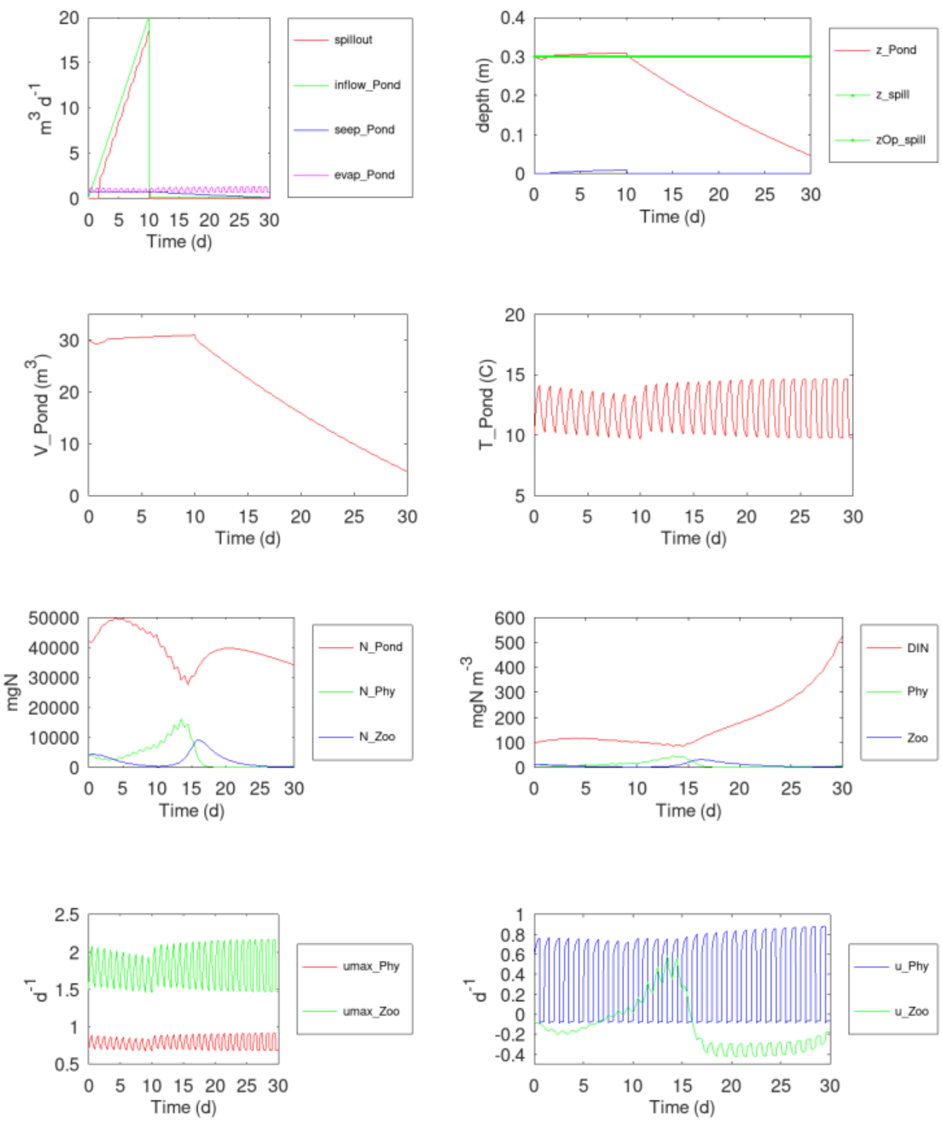


Fig. 10.3 The plot of *Dynamic Ecology* Chapter 10's model as produced by *pond_life.m*

10.2 Experimenting with the model

To experiment with the model as detailed in section "10.11 Things to explore" of *Dynamic Ecology* you need to change values of the model constants in file "func_pond_life.m" (Fig. 10.4). You can refer to the comments (lines prepended with a hashtag and coloured in green) next to the variable names to understand what each variable corresponds to. Further, you may need to refer back to previous chapters' models.

If you make a mistake, you can always undo/redo using the arrow buttons just above the Octave's Editor Window, and if you cannot resolve the problem, just download a new copy of the original GNU Octave model, and start over again.

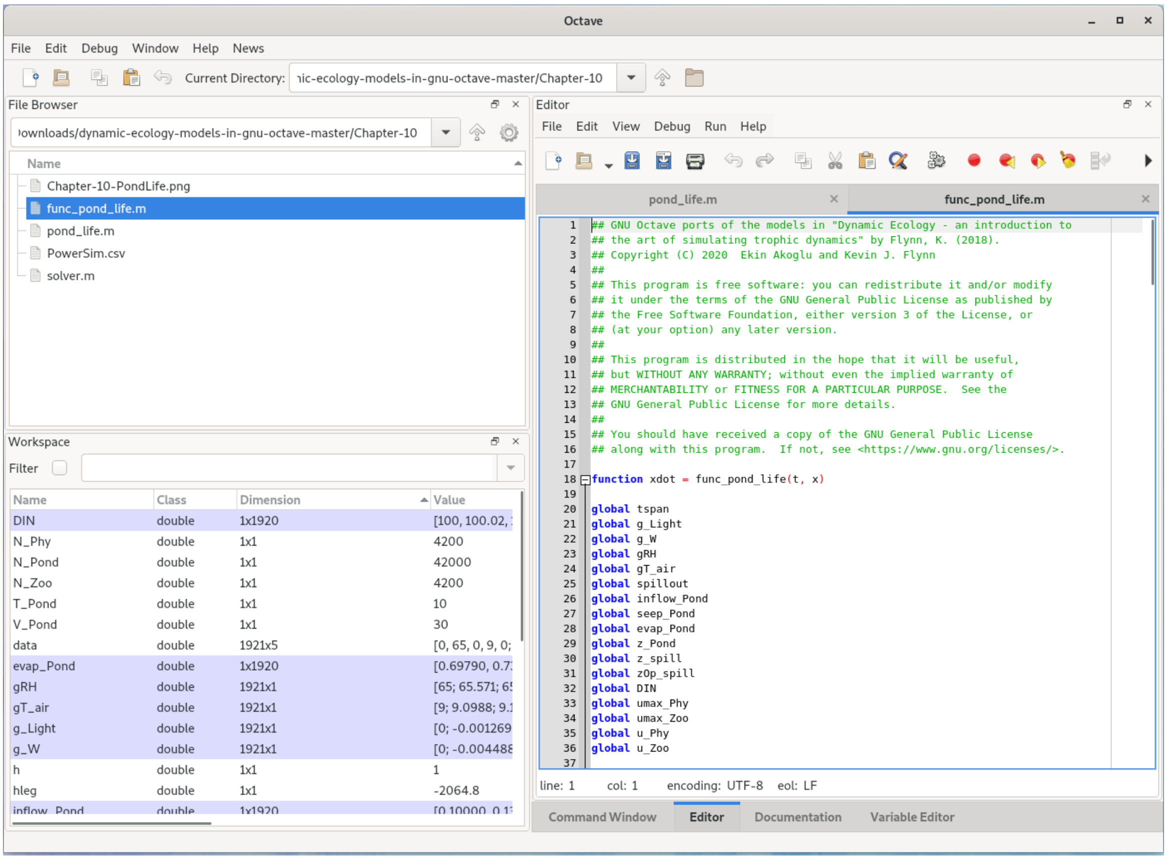


Fig. 10.4 The derivative function of Chapter 10's model.

10.3 GNU Octave code

This section, running over the following pages, provides a complete dump of the GNU Octave code as it appears in the download.

10.3.1 pond_life.m

```
## GNU Octave ports of the models in "Dynamic Ecology - an introduction to
## the art of simulating trophic dynamics" by Flynn, K. (2018).
## Copyright (C) 2020 Ekin Akoglu and Kevin J. Flynn

## This program is free software: you can redistribute it and/or modify
## it under the terms of the GNU General Public License as published by
## the Free Software Foundation, either version 3 of the License, or
## (at your option) any later version.

## This program is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
## GNU General Public License for more details.

## You should have received a copy of the GNU General Public License
## along with this program. If not, see https://www.gnu.org/licenses/.

clear;

global tspan
global g_Light
global g_W
global gRH
global gT_air
global spillout
global inflow_Pond
global seep_Pond
global evap_Pond
global z_Pond
global z_spill
global zOp_spill
global DIN
global umax_Phy
global umax_Zoo
global u_Phy
global u_Zoo

data = data = dlmread('PowerSim.csv', ",", [1, 0, 1921, 4]);

gRH = data(:, 2);       # Relative humidity data (%)
g_W = data(:, 3);       # Wind input data (m s-1)
gT_air = data(:, 4);    # Air temperature data (Celsius)
g_Light = data(:, 5);   # Light input data (W m-2)

# Simulation time frame
t0 = 0;      # start time
tfinal = 30; # end time
stepsize = 0.015625;
tspan = (t0:stepsize:tfinal); # time span

# Preallocate global arrays for speed
spillout = zeros(1, length(tspan)-1);
```

```
inflow_Pond = zeros(1, length(tspan)-1);
seep_Pond = zeros(1, length(tspan)-1);
evap_Pond = zeros(1, length(tspan)-1);
z_Pond = zeros(1, length(tspan)-1);
z_spill = zeros(1, length(tspan)-1);
zOp_spill = zeros(1, length(tspan)-1);
DIN = zeros(1, length(tspan)-1);
umax_Phy = zeros(1, length(tspan)-1);
umax_Zoo = zeros(1, length(tspan)-1);
u_Phy = zeros(1, length(tspan)-1);
u_Zoo = zeros(1, length(tspan)-1);

# Initial conditions
N_Phy = 4200;   # Phytoplankton biomass (mgN)
N_Pond = 42000; # Pond nutrient-N content (mgN)
N_Zoo = 4200;   # Zooplankton N-biomass (mgN)
T_Pond = 10;    # Temperature of pond water (Celsius)
V_Pond = 30;    # Pond volume (m3)
# Initial conditions array
x0 = [N_Phy N_Pond N_Zoo T_Pond V_Pond];

# Simulate
y = solver(@func_pond_life, tspan, stepsize, x0);

# Plot the results
h = figure;

subplot(4, 2, 1);
plot(tspan(2:end), spillout, 'r', tspan(2:end), inflow_Pond, 'g', tspan(2:end),
seep_Pond, 'b', tspan(2:end), evap_Pond, 'm');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
ylabel('m^{3} d^{-1}', 'FontSize', 12);
hleg = legend('spillout', 'inflow\_Pond', 'seep\_Pond', 'evap\_Pond',
'location', 'eastoutside');
set(hleg, 'FontSize', 8);
ylim([0 20]);

subplot(4, 2, 2);
plot(tspan(2:end), z_Pond, 'r', tspan(2:end), z_spill, 'g', tspan(2:end),
zOp_spill, 'b');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
ylabel('depth (m)', 'FontSize', 12);
hleg = legend('z\_Pond', 'z\_spill', 'zOp\_spill', 'location', 'eastoutside');
set(hleg, 'FontSize', 8);
ylim([0 0.4]);

subplot(4, 2, 3);
plot(tspan, y(:, 5), 'r');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
ylabel('V\_Pond (m^{3})', 'FontSize', 12);
ylim([0 35]);
set(gca, 'YTick', 0:10:30);

subplot(4, 2, 4);
plot(tspan, y(:, 4), 'r');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
ylabel('T\_Pond (C)', 'FontSize', 12);
ylim([5 20]);
set(gca, 'YTick', 5:5:20);
```

```
subplot(4, 2, 5);
plot(tspan, y(:, 2), 'r', tspan, y(:, 1), 'g', tspan, y(:, 3), 'b');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
ylabel('mgN', 'FontSize', 12);
hleg = legend('N\_Pond', 'N\_Phy', 'N\_Zoo', 'location', 'eastoutside');
set(hleg, 'FontSize', 8);

subplot(4, 2, 6);
plot(tspan, (y(:, 2) ./ y(:, 5)) ./ 14, 'r', tspan, (y(:, 1) ./ y(:, 5)) ./ 14,
'g', tspan, (y(:, 3) ./ y(:, 5)) ./ 14, 'b');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
ylabel('mgN m^{-3}', 'FontSize', 12);
hleg = legend('DIN', 'Phy', 'Zoo', 'location', 'eastoutside');
set(hleg, 'FontSize', 8);

subplot(4, 2, 7);
plot(tspan(2:end), umax_Phy, 'r', tspan(2:end), umax_Zoo, 'g');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
ylabel('d^{-1}', 'FontSize', 12);
hleg = legend('umax\_Phy', 'umax\_Zoo', 'location', 'eastoutside');
set(hleg, 'FontSize', 8);

subplot(4, 2, 8);
plot(tspan(2:end), u_Phy, 'b', tspan(2:end), u_Zoo, 'g');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
ylabel('d^{-1}', 'FontSize', 12);
hleg = legend('u\_Phy', 'u\_Zoo', 'location', 'eastoutside');
set(hleg, 'FontSize', 8);
ylim([-0.5 1]);

set(gcf, 'PaperPosition', [0.25000 2.50000 9.00000 12.00000]);
print(h, 'Chapter-10-PondLife.png', '-dpng', '-color');
```

10.3.2 func_pond_life.m

```
## GNU Octave ports of the models in "Dynamic Ecology - an introduction to
## the art of simulating trophic dynamics" by Flynn, K. (2018).
## Copyright (C) 2020 Ekin Akoglu and Kevin J. Flynn

## This program is free software: you can redistribute it and/or modify
## it under the terms of the GNU General Public License as published by
## the Free Software Foundation, either version 3 of the License, or
## (at your option) any later version.

## This program is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
## GNU General Public License for more details.

## You should have received a copy of the GNU General Public License
## along with this program. If not, see https://www.gnu.org/licenses/.

function xdot = func_pond_life(t, x)

global tspan
global g_Light
global g_W
global gRH
global gT_air
global spillout
global inflow_Pond
global seep_Pond
global evap_Pond
global z_Pond
global z_spill
global zOp_spill
global DIN
global umax_Phy
global umax_Zoo
global u_Phy
global u_Zoo

# Physical parameters
T_air = 20;           # Air temperature (Celsius)
T_inflow = 6;         # Inflow water temperature (Celsius)
Tini = 10;            # Initial pond temperature (Celsius)
salinity = 0;         # Salinity (dl)
RH = 30;              # Relative humidity
SA_Pond = 100;        # Surface area of pond (m2)
wid_spill = 0.05;     # Width of spillway (m)
z_spill = 0.3;        # Height of spill way lip above lowest point of pond (m)
SeepR = 0.025;        # Seepage rate of water from the pond related to SA and per
m of depth (d-1)
SBconst = 5.73E-08; # W/m2/ K4 Stefan-Boltzmann constant
SDA = 0.3;            # Specific Dynamic Action (dl)
bkRad1 = 0.05;        # Back radiation constant (1 / mb^0.5)
bkRad2 = 0.35;        # Back radiation constant (dl)
cloud = 2;            # Cloud cover (0 to 8) (oktas)
con_fact = 4.57;      # Correction factor to converts light as W / m2 to PAR (umol
s-1 W-1)
cp = 4186;            # Specific heat of water (J/kg * C)
Emissivity = 0.985; #  Emissivity of thermal radiation (dl)
g = 9.81;             # Acceleration due to gravity (m s-2)
W = 10;               # Wind speed (m s-1)

# Nutrient parameters
```

```
N_inflow = 100 * 14; # Inflow concentration of N into pond (mgN m-3)

# Phytoplankton parameters
attco_W = 0.1;        # Absorbance coefficient for growth medium (water) (m-1)
abco_Chl = 0.02;      # Light absorbance coefficient for chlorophyll (m2
(mgChl)-1)
BR_Phy = 0.1;         # Phytoplankton basal respiration as proportion of
umax_Phy (dl)
alpha_Chl = 7.00E-06; # Slope of Chl-specific PE curve       (m2 g-1 chl.a)*(gC
umol-1 photon)
ChlC_Phy = 0.06;      # Mass ratio content of chlorophyll:C in the phytoplankton
(gChl (gC)-1)
inflow_Phy = 10 * 14; # Concentration of incoming phytoplankton biomass (mgN m-
3)
kN_Phy = 1 * 14;      # Half saturation constant for u_Phy (mgN m-3)
NC_Phy = 0.15;        # Mass ratio content of N-biomass:C in the phytoplankton
(gN (gC)-1)
Q10_Phy = 1.8;        # Phytoplankton Q10 (dl)
Tref_Phy = 10;        # Reference temperature for phytoplankton growth (Celsius)
Uref_Phy = 0.693;     # Phytoplankton maximum growth rate at reference T (d-1)

# Zooplankton parameters
AEN_Zoo = 0.6;     # Assimilation efficiency (dl)
BR_Zoo = 0.2;      # Zooplankton basal respiration rate proportioned to umax_Zoo
(dl)
kPhy_Zoo = 14 * 5; # Half saturation constant for zooplankton predation on
phytoplankton (mgN m-3)
Q10_Zoo = 2.2;     # Zooplankton Q10 (dl)
Tref_Zoo = 10;     # Reference temperature for zooplankton growth (Celsius)
Uref_Zoo = 1.5;    # Zooplankton maximum growth rate at reference T (d-1)

flag = 0; # 0 = fixed; 1 = data input flag between fixed or data input values

# Auxiliaries
## Inflow of water (m3 d-1)
if tspan(t - 1) < 10
  In = 2 * tspan(t - 1) + 0.1;
else
  In = 0.1;
endif
# Inflow of water (m3 d-1)
inflow_Pond(t - 1) = In;
# Effective dilution rate of pond (d-1)
dil = inflow_Pond(t - 1) / x(5);

if flag == 0
  mult1 = 1;
  mult2 = 0;
else
  mult1 = 0;
  mult2 = 1;
endif
# Operational air temperature (Celsius)
op_Tair = mult1 * T_air + mult2 * gT_air(t - 1);

if flag == 0
  mult1 = 1;
  mult2 = 0;
else
  mult1 = 0;
  mult2 = 1;
endif
```

```
# Operational relative humidity
op_RH = mult1 * RH + mult2 * gRH(t - 1);
# Water vapour pressure in the atmosphere (mb)
ea = (op_RH / 100) * 6.11 * 10^(7.5  *  op_Tair / (op_Tair + 237));
# Back radiation (W m-2)
Q_br = (1 - 0.1 * cloud) * Emissivity * SBconst * (bkRad2 - bkRad1 * sqrt(ea)) *
(x(4) + 273)^4;
# Saturated vapour pressure (mb)
es = 6.11  *  10^(7.5  *  x(4) / (x(4) + 237));

if flag == 0
  mult1 = 1;
  mult2 = 0;
else
  mult1 = 0;
  mult2 = 1;
endif
# Operational wind speed (m s-1)
Wind = mult1 * W + mult2 * g_W(t - 1);

# Cooling evaporative heat flux (W m-2)
Qe = (3.8 * (es - ea) * Wind);
# Sensible heat flux from pond (W m-2)
Qh = 2.5 * (x(4) - op_Tair) * Wind;

if tspan(t - 1) - floor(tspan(t - 1)) < 0.5
  Light = 300;
else
  Light = 0;
endif

if flag == 0
  mult1 = 1;
else
  mult1 = 0;
endif
if flag == 1 && g_Light(t - 1) > 0
  mult2 = 1;
else
  mult2 = 0;
endif
# Light at the pond surface (W m-2)
Wm2 = mult1 * Light + mult2 * g_Light(t - 1);
# Net heat flux (W m-2)
Qn = Wm2 - (Q_br + Qe + Qh);
 # Water density (kg m-3)
rho = 1000 + salinity;

# Depth of pond water (m)
if x(5) > 0
  z_Pond(t - 1) = x(5) / SA_Pond;
else
  z_Pond(t - 1) = 0;
endif

# Rate of change of temperature due to heating and cooling (Celsius d-1)
dTwIn = (Qn / (cp  *  rho  *  z_Pond(t - 1))) * 60  *  60  *  24;
# Latent heat of water evaporation (J kg-1)
LH = 1000 * (2500.8 - 2.36 * x(4) + 0.0016 * x(4)^2 - 0.00006 * x(4)^3);
# Evaporation rate (m s-1)
er = Qe / (LH  *  rho);
# Loss of water through evaporation (m3 d-1)
if x(5) > 0.1
```

```
  evap_Pond(t - 1) = (er * 60 * 60 * 24) * SA_Pond;
else
  evap_Pond(t - 1) = 0;
endif

# Phytoplankton biomass concentration (umolN L-1)
Phy = (x(1) / x(5)) / 14;
# Phytoplankton-N specific coefficient for light absorbance (m2 (mgN)-1)
abco_PhyN = abco_Chl * ChlC_Phy / NC_Phy;
# Specific slope of PE curve ((m2)*(umol-1 photon))
alpha_u = alpha_Chl * ChlC_Phy;
# Attenuation coefficient to phytoplankton N-biomass (m-1)
attco_Phy = abco_PhyN * Phy;
# Total attenuation (dl)
att_tot = z_Pond(t - 1) * (attco_W + attco_Phy);

# Negative exponent of total attenuation (dl)
exatt = exp(-att_tot);


# Concentration of nutrient-N (uM)
DIN(t - 1) = (x(2) / x(5)) / 14;

# Inflow of N into pond (mgN d-1)
inflow_N = inflow_Pond(t - 1) * N_inflow;
# Incoming phytoplankton; this also serves to inoculate the system (mgN d-1)
inflow_Phyto = inflow_Phy * inflow_Pond(t - 1);
# Temperature adjusted zooplankton maximum growth rate (d-1)
umax_Zoo(t - 1) = Uref_Zoo * Q10_Zoo^((x(4) - Tref_Zoo) / 10);
# Maximum ingestion rate, allowing u_Zoo=umax_Zoo under optimal conditions (d-1)
ingNmax_Zoo = (umax_Zoo(t - 1) * (1 + BR_Zoo)) / (AEN_Zoo * (1 - SDA));
# Ingestion rate of phytoplankton (d-1)
ingN_Zoo = ingNmax_Zoo * Phy / (Phy + kPhy_Zoo);
# Zooplankton growth rate (d-1)
u_Zoo(t - 1) = ingN_Zoo * AEN_Zoo * (1 - SDA)-(umax_Zoo(t - 1) * BR_Zoo);
# Assimilation rate (d-1)
assN_Zoo = ingN_Zoo * AEN_Zoo;

# Loss of phytoplankton through ingestion by zooplankton (mgN d-1)
ingN = x(3) * ingN_Zoo;
# PFD at surface (umoles m-2 s-1)
nat_PFD = Wm2 * con_fact;
# Regeneration of N by zooplankton as a consequence of grazing and respiration
(mgN d-1)
Nregen = x(3) * (umax_Zoo(t - 1) * BR_Zoo) + assN_Zoo * SDA + ingN * (1 -
AEN_Zoo);
# Index of N-limitation for phytoplankton growth (dl)
Nu = DIN(t - 1) / (DIN(t - 1) + kN_Phy);

# Temperature adjusted phytoplankton maximum growth rate (d-1)
umax_Phy(t - 1) = Uref_Phy * Q10_Phy^((x(4) - Tref_Phy) / 10);
# Maximum photosynthetic rate to balance BR_Phy to give u_Phy=umax_Phy (d-1)
PSmax = umax_Phy(t - 1) * (1 + BR_Phy);
# Maximum photosynthetic rate down-regulated in consequence of nutrient stress
(d-1)
PSqmax = PSmax * Nu;
# Intermediate in depth-integrated photosynthesis rate (d)
pytq = (alpha_u * nat_PFD * 24 * 60 * 60) / PSqmax;
# Phytoplankton N-specific growth rate (d-1)
PSqz = PSqmax * (log(pytq + sqrt(1 + pytq^2)) - log(pytq * exatt + sqrt(1 +
(pytq * exatt)^2))) / att_tot;
# Phytoplankton growth rate (d-1)
u_Phy(t - 1) = PSqz - (umax_Phy(t - 1) * BR_Phy);
```

```
# Phytoplankton biomass growth (mgN d-1)
gro_Phy = x(1)  *  u_Phy(t - 1);

# Depth of water over spillway (m)
if z_Pond(t - 1) > z_spill;
  zOp_spill(t - 1) = z_Pond(t - 1) - z_spill;
else
  zOp_spill(t - 1) = 0;
endif
# Area of the mouth of the spillway (m2)
XSA_spill = zOp_spill(t - 1) * wid_spill;

# Loss of water through overflow through a spillway (m3 d-1)
spillout(t - 1) = 60 * 60 * 24 * XSA_spill * (2 * g * zOp_spill(t - 1))^0.5;

# Seepage loss of water (m3 d-1)
if z_Pond(t - 1) > 0
  seep_Pond(t - 1) = SeepR * SA_Pond * z_Pond(t - 1);
else
  seep_Pond(t - 1) = 0;
endif
# Loss of nutrient-N through seepage (mgN d-1)
seep_N = x(2) * seep_Pond(t - 1) / x(5);

# Loss of nutrient-N over the spillway (mgN d-1)
spill_N = x(2) * spillout(t - 1) / x(5);

# Loss of phytoplankton biomass over the spillway (mgN d-1)
spill_Phy = x(1) * spillout(t - 1) / x(5);

# Loss of zooplankton biomass over the spillway (mgN d-1)
spill_Zoo = x(3) * spillout(t - 1) / x(5);

# Specific dilution rate as would apply to phytoplankton (d-1)
spilld = spillout(t - 1) / x(5);

# Stop command to halt simulation when water attains a minimum depth (dl)
if z_Pond(t - 1)< 0.01
  Stop_z = 1;
  error("Minimum depth is attained! Simulation stopped!");
endif

# Change in water temperature with incoming water (Celsius d-1)
T_dil = (T_inflow - x(4)) * dil;

# Zooplankton biomass concentration (umolN L-1)
Zoo(t - 1) = (x(3) / x(5)) / 14;

## State equations
# Phytoplankton
xdot(1, 1) = gro_Phy + inflow_Phyto - ingN - spill_Phy;

# Pond nutrient-N
xdot(1, 2) = inflow_N + Nregen - gro_Phy - seep_N - spill_N;

# Zooplankton
xdot(1, 3) = ingN - Nregen - spill_Zoo;

# Temperature of pond water
xdot(1, 4) = dTwIn + T_dil;

# Pond volume
```

```
xdot(1, 5) = inflow_Pond(t - 1) - evap_Pond(t - 1) - seep_Pond(t - 1) -
spillout(t - 1);

endfunction
```

11. Closure Model in GNU Octave

This Chapter provides information on running the model in chapter 11 of *Dynamic Ecology* (Flynn 2018), running through each of the steps. It is assumed that you have installed the Octave interface (Chapter 2).

No model can ever describe everything; there have to be boundaries of in terms of physics, chemistry, biology, and of course time. So how do you handle these boundaries, and specifically here, how do you handle the upper most trophic level in an ecosystem model?

If you are modelling the changes in the volume of a lake then you do not need to simulate, in a consequence of the lake filling through rainfall, that the amount of moisture in the air must decrease. Neither will you likely need to simulate changes in the volume of the oceans as the lake water drains into the sea.

In models of food webs it is likewise often necessary to limit the detail at the lowest and uppermost reaches of the food web. It is rare that microbial communities are described in any detail, so that nutrient regeneration is treated rather as a black-box of organics entering and inorganics flowing out; this is the route we used in Chapters 5 and 10. The upper extremes of food webs contain top predators; these organisms (for aquatic systems, larger fish, whales, sharks, etc.) are often enigmatic and feature strongly in perceptions of importance. However, in reality they are often responsible for very little of the biomass and energy flows through food webs, while their activity is often also only occasional, being linked to movement of these animals between feeding areas. This is not to say that the activity of these higher trophic levels is not of importance in structuring the system, so somehow we need to include their activity. However, rather than describe their activity explicitly, we can describe it implicitly using a function called a closure term.

Please see chapter 11 in *Dynamic Ecology* for more contextual information, explanations for model construction, and (in the final sections of that chapter) ideas for experimenting and developing your models.

11.1 Running the model

Navigate to the folder “Chapter-11” in the “File Browser” by double-clicking it. You will now see the script files and related figures of the model in the “File Browser” (Fig. 11.1).

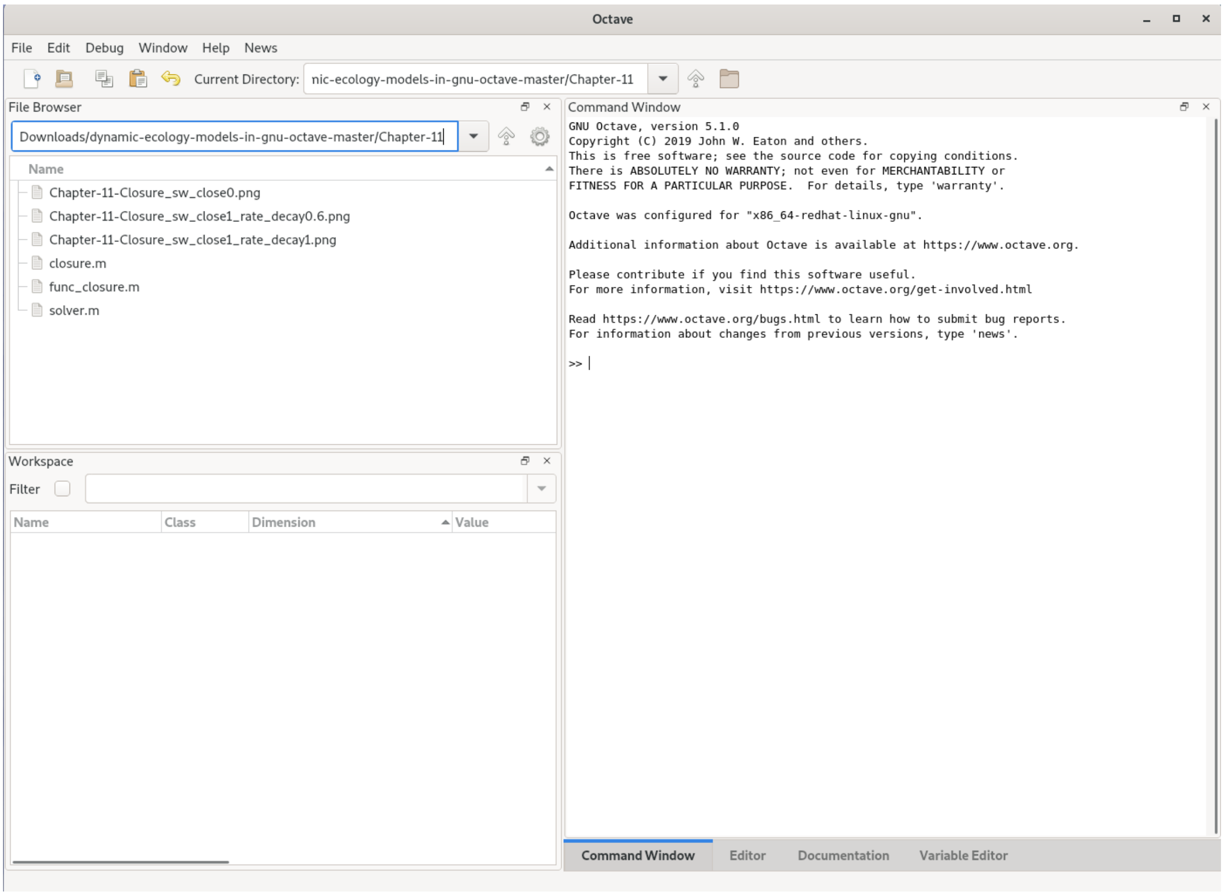


Fig. 11.1 Model of Chapter 11 in *Dynamic Ecology* and its related script files and figures.

Double-click the "closure.m" file to open it (Fig. 11.2).

The script file will be opened in the editor window of GNU Octave. As you will notice, the file includes a series of GNU Octave commands and comments (lines prepended by a hashtag and coloured in green) that explain what each line of the code corresponds to.

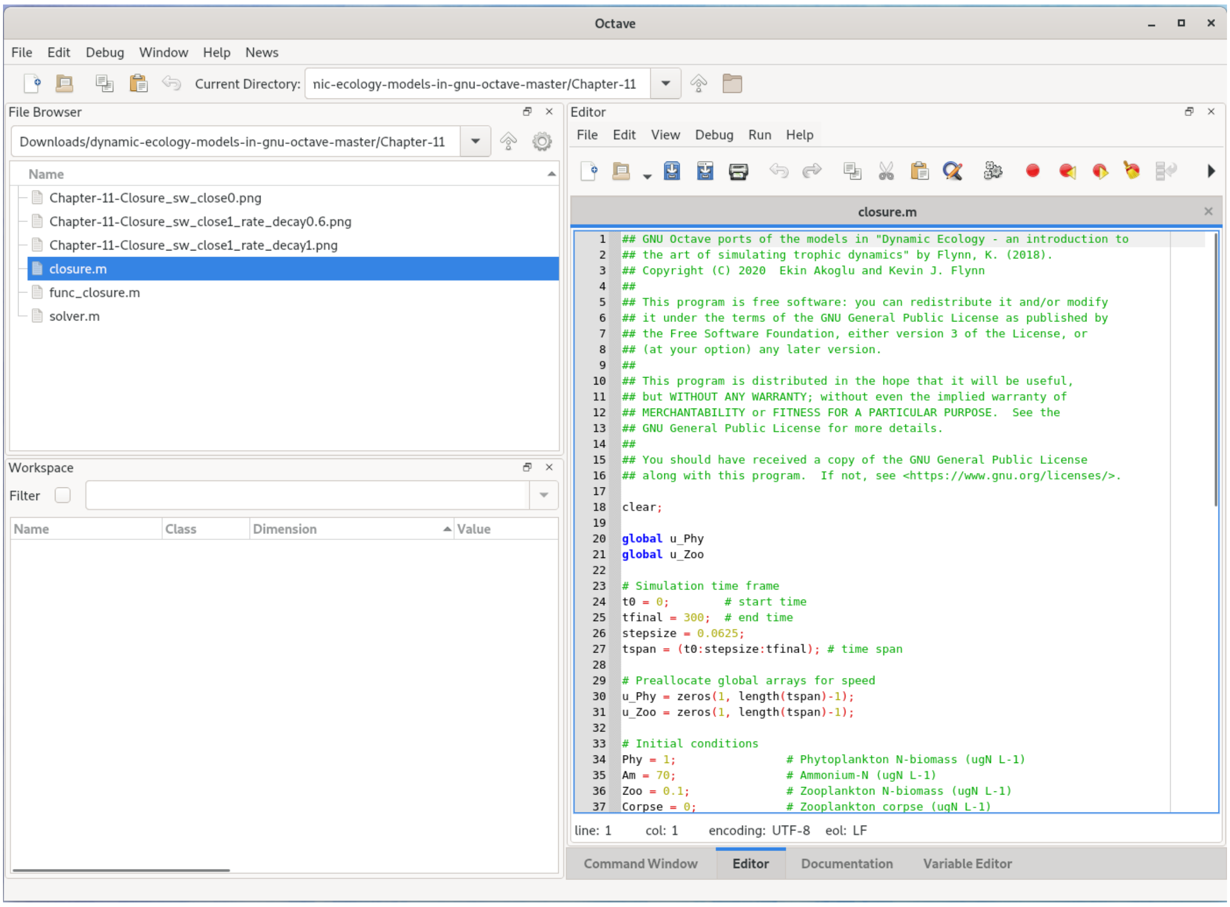


Fig. 11.2 Overview of *closure.m*

To run the “closure.m”, hit F5 on your keyboard. Alternatively, you may switch back to the “Command Window” by using the tabs at the bottom of the right part of the main window and then enter the command “closure” and press “Enter” key while you are in the “Command Window”.

The model will now run and produce a plot from simulation results once the simulation is completed (Fig. 11.3).

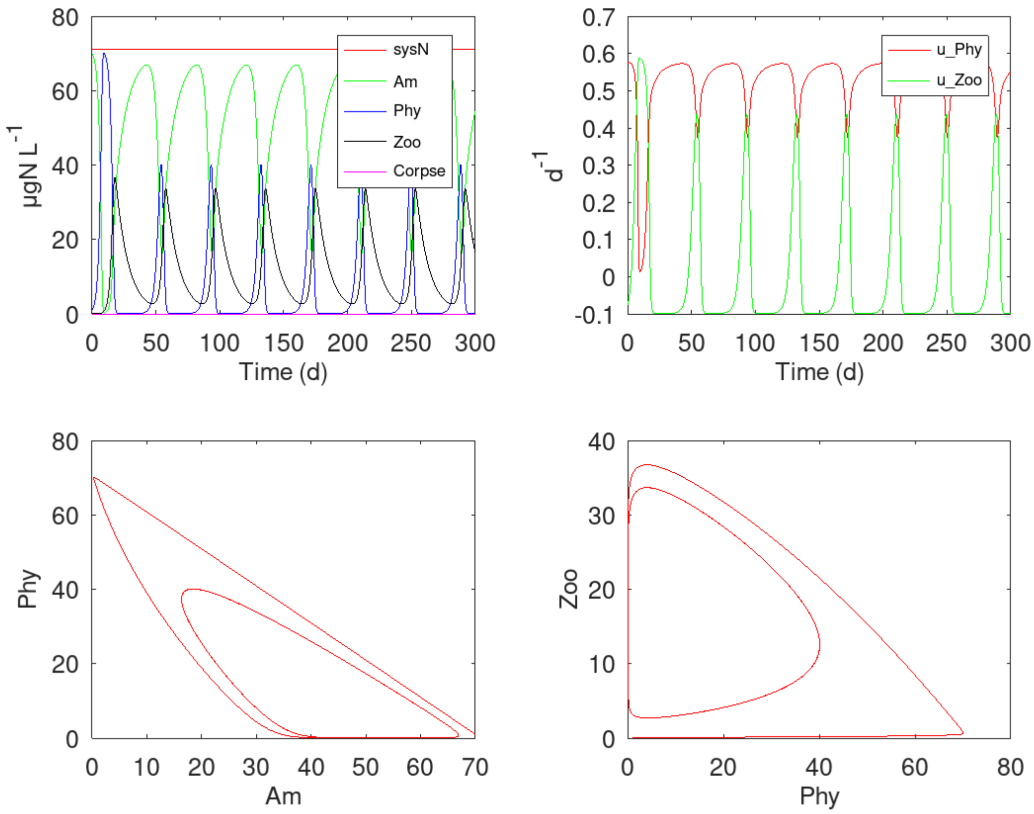
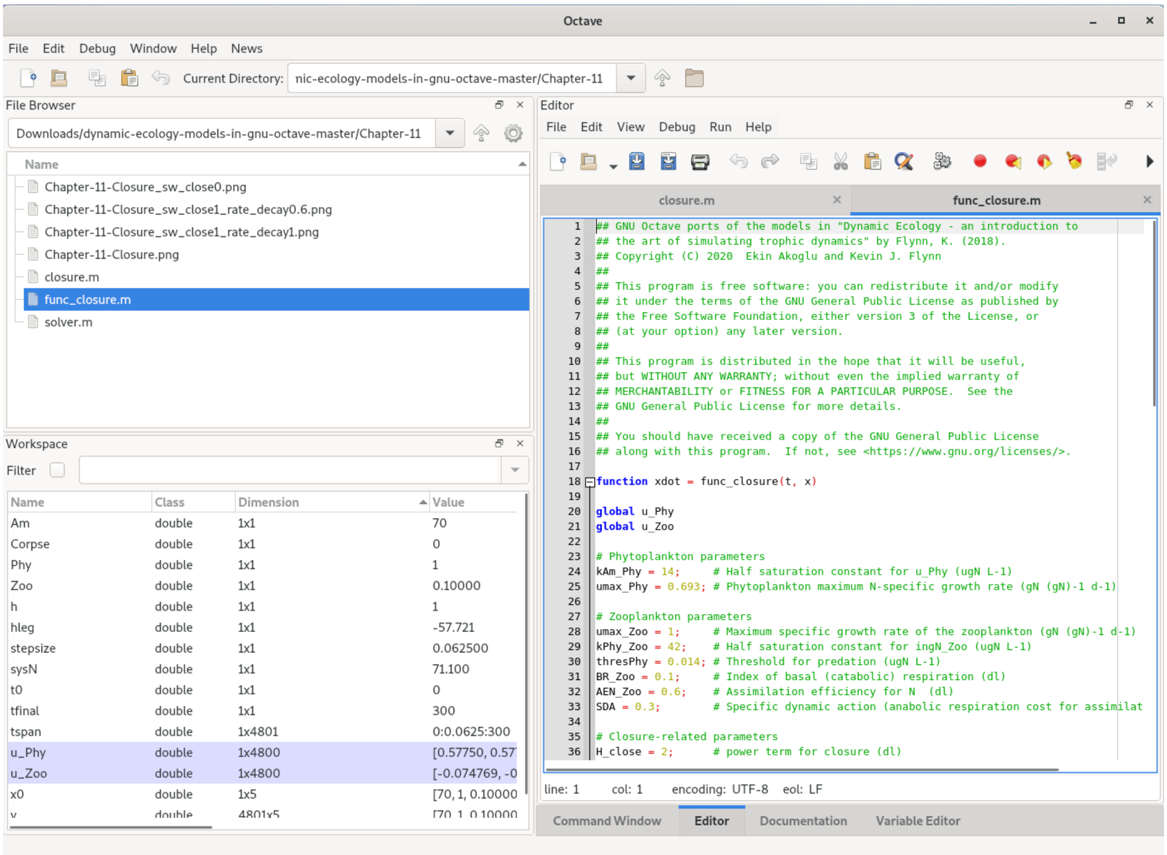


Fig. 11.3 The plot of *Dynamic Ecology* Chapter 11's model as produced by *closure.m*

11.2 Experimenting with the model

To experiment with the model as detailed in section "11.7 Things to explore" of *Dynamic Ecology* you need to change values of the model constants in file "func_closure.m" (Fig. 11.4). You can refer to the comments (lines prepended with a hashtag and coloured in green) next to the variable names to understand what each variable corresponds to. Specifically, you need to change parameters below the comment on line 35 reading "# Closure-related parameters".

If you make a mistake, you can always undo/redo using the arrow buttons just above the Octave's Editor Window, and if you cannot resolve the problem, just download a new copy of the original GNU Octave model, and start over again.



Name	Class	Dimension	Value
Am	double	1x1	70
Corpse	double	1x1	0
Phy	double	1x1	1
Zoo	double	1x1	0.10000
h	double	1x1	1
hleg	double	1x1	-57.721
stepsize	double	1x1	0.062500
sysN	double	1x1	71.100
t0	double	1x1	0
tfinal	double	1x1	300
tspan	double	1x4801	0:0.0625:300
u_Phy	double	1x4800	[0.57750, 0.57
u_Zoo	double	1x4800	[-0.074769, -0
x0	double	1x5	[70, 1, 0.10000

Fig. 11.4 The derivative function of *Dynamic Ecology* Chapter 11's model.

11.3 GNU Octave code

This section, running over the following pages, provides a complete dump of the GNU Octave code as it appears in the download.

11.3.1 closure.m

```
## GNU Octave ports of the models in "Dynamic Ecology - an introduction to
## the art of simulating trophic dynamics" by Flynn, K. (2018).
## Copyright (C) 2020 Ekin Akoglu and Kevin J. Flynn

## This program is free software: you can redistribute it and/or modify
## it under the terms of the GNU General Public License as published by
## the Free Software Foundation, either version 3 of the License, or
## (at your option) any later version.

## This program is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
## GNU General Public License for more details.

## You should have received a copy of the GNU General Public License
## along with this program. If not, see https://www.gnu.org/licenses/.

clear;

global u_Phy
global u_Zoo

# Simulation time frame
t0 = 0;         # start time
tfinal = 300;   # end time
stepsize = 0.0625;
tspan = (t0:stepsize:tfinal); # time span

# Preallocate global arrays for speed
u_Phy = zeros(1, length(tspan)-1);
u_Zoo = zeros(1, length(tspan)-1);

# Initial conditions
Phy =   1;               # Phytoplankton N-biomass (ugN L-1)
Am = 70;                 # Ammonium-N (ugN L-1)
Zoo = 0.1;               # Zooplankton N-biomass (ugN L-1)
Corpse = 0;              # Zooplankton corpse (ugN L-1)
sysN    = Am + Phy + Zoo + Corpse; # System N-balance (ugN L-1)
# Initial conditions array
x0 = [Am, Phy, Zoo, sysN, Corpse];

# Simulate
y = solver(@func_closure, tspan, stepsize, x0);

# Plot the results
h = figure;

subplot(2, 2, 1);
plot(tspan, y(:, 4), 'r', tspan, y(:, 1), 'g', tspan, y(:, 2), 'b', tspan, y(:,
3), 'k', tspan, y(:, 5), 'm');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
```

```
ylabel('\mugN L^{-1}', 'FontSize', 12);
hleg = legend('sysN', 'Am', 'Phy', 'Zoo', 'Corpse');
set(hleg, 'FontSize', 8);

subplot(2, 2, 2);
plot(tspan(2:end), u_Phy', 'r', tspan(2:end), u_Zoo', 'g');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
ylabel('d^{-1}', 'FontSize', 12);
hleg = legend('u\_Phy', 'u\_Zoo');
set(hleg, 'FontSize', 8);

subplot(2, 2, 3);
plot(y(:, 1), y(:, 2), 'r');
set(gca,'FontSize',12);
xlabel('Am', 'FontSize', 12);
ylabel('Phy', 'FontSize', 12);

subplot(2, 2, 4);
plot(y(:, 2), y(:, 3), 'r');
set(gca,'FontSize',12);
xlabel('Phy', 'FontSize', 12);
ylabel('Zoo', 'FontSize', 12);

print(h, 'Chapter-11-Closure.png', '-dpng', '-color');
```

11.3.2 func_closure.m

```
## GNU Octave ports of the models in "Dynamic Ecology - an introduction to
## the art of simulating trophic dynamics" by Flynn, K. (2018).
## Copyright (C) 2020 Ekin Akoglu and Kevin J. Flynn

## This program is free software: you can redistribute it and/or modify
## it under the terms of the GNU General Public License as published by
## the Free Software Foundation, either version 3 of the License, or
## (at your option) any later version.

## This program is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
## GNU General Public License for more details.

## You should have received a copy of the GNU General Public License
## along with this program. If not, see https://www.gnu.org/licenses/.

function xdot = func_closure(t, x)

global u_Phy
global u_Zoo

# Phytoplankton parameters
kAm_Phy = 14;      # Half saturation constant for u_Phy (ugN L-1)
umax_Phy = 0.693; # Phytoplankton maximum N-specific growth rate (gN (gN)-1 d-1)

# Zooplankton parameters
umax_Zoo = 1;      # Maximum specific growth rate of the zooplankton (gN (gN)-1
d-1)
kPhy_Zoo = 42;     # Half saturation constant for ingN_Zoo (ugN L-1)
thresPhy = 0.014; # Threshold for predation (ugN L-1)
BR_Zoo = 0.1;      # Index of basal (catabolic) respiration (dl)
AEN_Zoo = 0.6;     # Assimilation efficiency for N  (dl)
SDA = 0.3;         # Specific dynamic action (anabolic respiration cost for
assimilating N, gN/gN)

# Closure-related parameters
H_close = 2;       # power term for closure (dl)
K_close = 0.01;    # constant term for closure (d-1)
rate_decay = 0.6; # proportion of zoo_death decaying to Ammonium (dl)
sw_close = 0;      # switch to enact closure (dl)

# Auxiliaries
## Phytoplankton N-specific growth rate (gN (gN)-1 d-1)
u_Phy(t - 1)=  umax_Phy * x(1) / (x(1) + kAm_Phy);

# Phytoplankton population growth rate (ugN L-1 d-1)
gro_Phy = x(2) * u_Phy(t - 1);

# Ingestion rate with inclusion of threshold control (gN (gN)-1 d-1)
ingNmax_Zoo = (umax_Zoo * (1 + BR_Zoo)) / (AEN_Zoo * (1 - SDA));

# Maximum ingestion rate (gN (gN)-1 d-1)
if x(2) > thresPhy
  ingPhy_Zoo = ingNmax_Zoo * (x(2) - thresPhy) / (x(2) - thresPhy + kPhy_Zoo);
else
  ingPhy_Zoo = 0;
endif

# Zooplankton N-specific growth rate (gN (gN)-1 d-1)
u_Zoo(t - 1) = ingPhy_Zoo * AEN_Zoo * (1 - SDA) - (umax_Zoo * BR_Zoo);
```

```
# Zooplankton assimilation rate (gN (gN)-1 d-1)
assN_Zoo = ingPhy_Zoo * AEN_Zoo;

# Zooplankton N-specific regeneration rate (gN (gN)-1 d-1)
regN_Zoo = (umax_Zoo * BR_Zoo) + assN_Zoo * SDA;

# Zooplankton population ingestion rate (ugN L-1 d-1)
ing_Zoo = x(3) * ingPhy_Zoo;

# Zooplankton population N-regeneration rate (ugN L-1 d-1)
reg_Zoo = x(3) * regN_Zoo;

# Zooplankton population N-voiding rate (ugN L-1 d-1)
void_Zoo = x(3) * ingPhy_Zoo * (1 - AEN_Zoo);

# Closure term for death  (ugN L-1 d-1)
if sw_close == 1
  death_Zoo = 1 * K_close * (x(3)^H_close);
else
  death_Zoo = 0 * K_close * (x(3)^H_close);
endif

# Decay rate of corpse (ugN L-1 d-1)
decay = death_Zoo * rate_decay;

## State equations
# Ammonium
xdot(1, 1) = -gro_Phy + reg_Zoo + void_Zoo + decay;

# Phytoplankton
xdot(1, 2) = gro_Phy - ing_Zoo;

# Zooplankton
xdot(1, 3) = ing_Zoo - reg_Zoo - void_Zoo - death_Zoo;

# Corpse
xdot(1, 5) = death_Zoo - decay;

# System
xdot(1, 4) = xdot(1, 1) + xdot(1, 2) + xdot(1, 3) + xdot(1, 5);

endfunction
```

12. Classic NPZ Model in GNU Octave

This Chapter provides information on running the model in chapter 12 of *Dynamic Ecology* (Flynn 2018), running through each of the steps. It is assumed that you have installed the Octave interface (Chapter 2).

Two thirds of Earth is covered by the oceans. The bulk of the biological activity in the oceans, and indeed 50% of all planetary primary production, is mediated by the marine phytoplankton, controlled by a combination of nutrients, light and predation by the zooplankton, and other losses. Accepting that this is now recognised as a flawed simplification (as ca. 50% of the microplankton are mixotrophic – Flynn et al. 2013, Mitra et al. 2014, 2016) the oceans represent arguably the most important single, continuously linked, and well researched ecosystem on the planet.

In this chapter we will build and explore a classic description of oceanographic nutrient-phytoplankton-zooplankton ("NPZ") interactions. The model described here was written by the late Prof Michael JR Fasham FRS, a father figure for the "NPZ" genera of marine models as applied to oceanography (the classic paper is Fasham et al. 1990). The naming of the variables is largely consistent with those used in the original description, though the structure has been modified slightly to conform to approaches developed in this book.

Please see chapter 12 in *Dynamic Ecology* for more contextual information, explanations for model construction, and (in the final sections of that chapter) ideas for experimenting and developing your models.

12.1 Running the model

Navigate to the folder "Chapter-12" in the "File Browser" by double-clicking it. You will now see the script files and related figures of the model in the "File Browser" (Fig. 12.1).

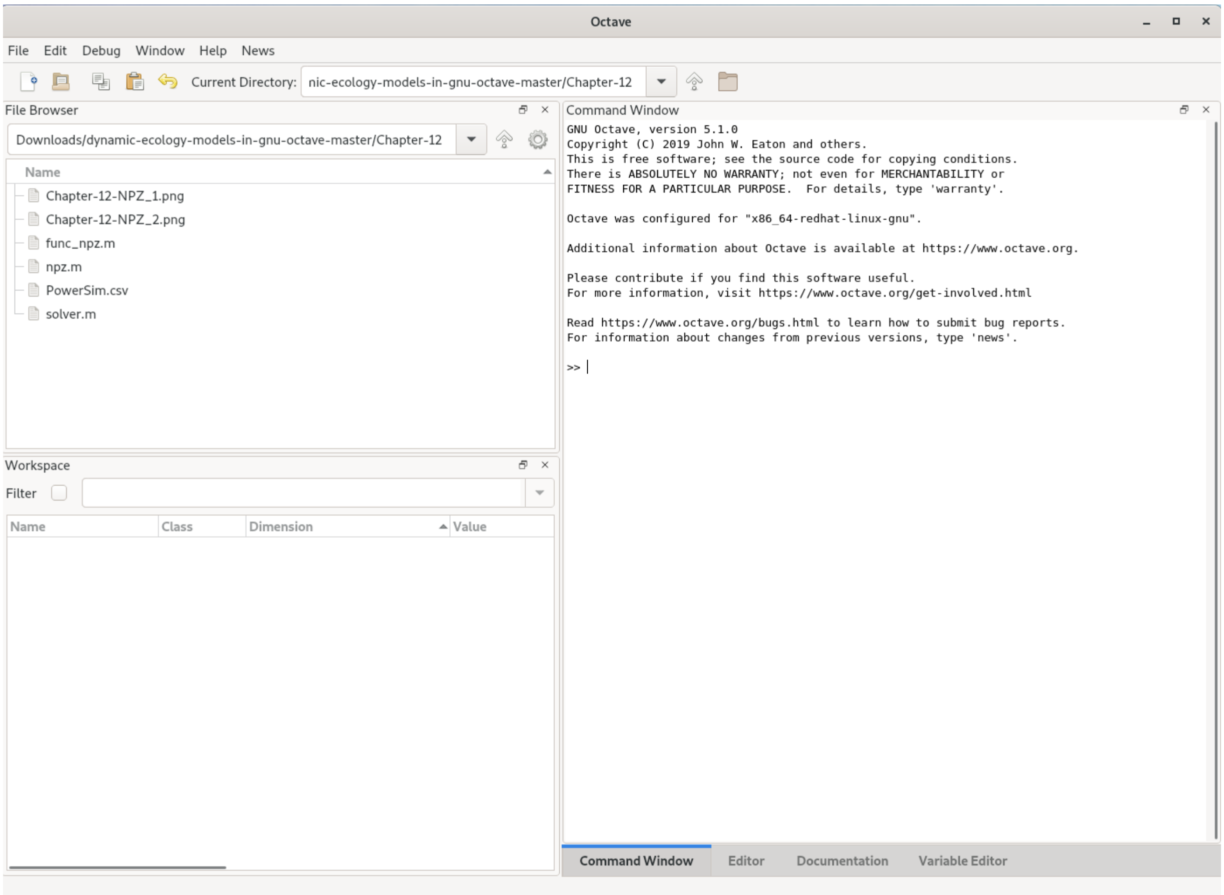


Fig. 12.1 Model of Chapter 12 in *Dynamic Ecology* and its related script files and figures.

Double-click the “npz.m” file to open it (Fig. 12.2).

The script file will be opened in the editor window of GNU Octave. As you will notice, the file includes a series of GNU Octave commands and comments (lines prepended by a hashtag and coloured in green) that explain what each line of the code corresponds to.

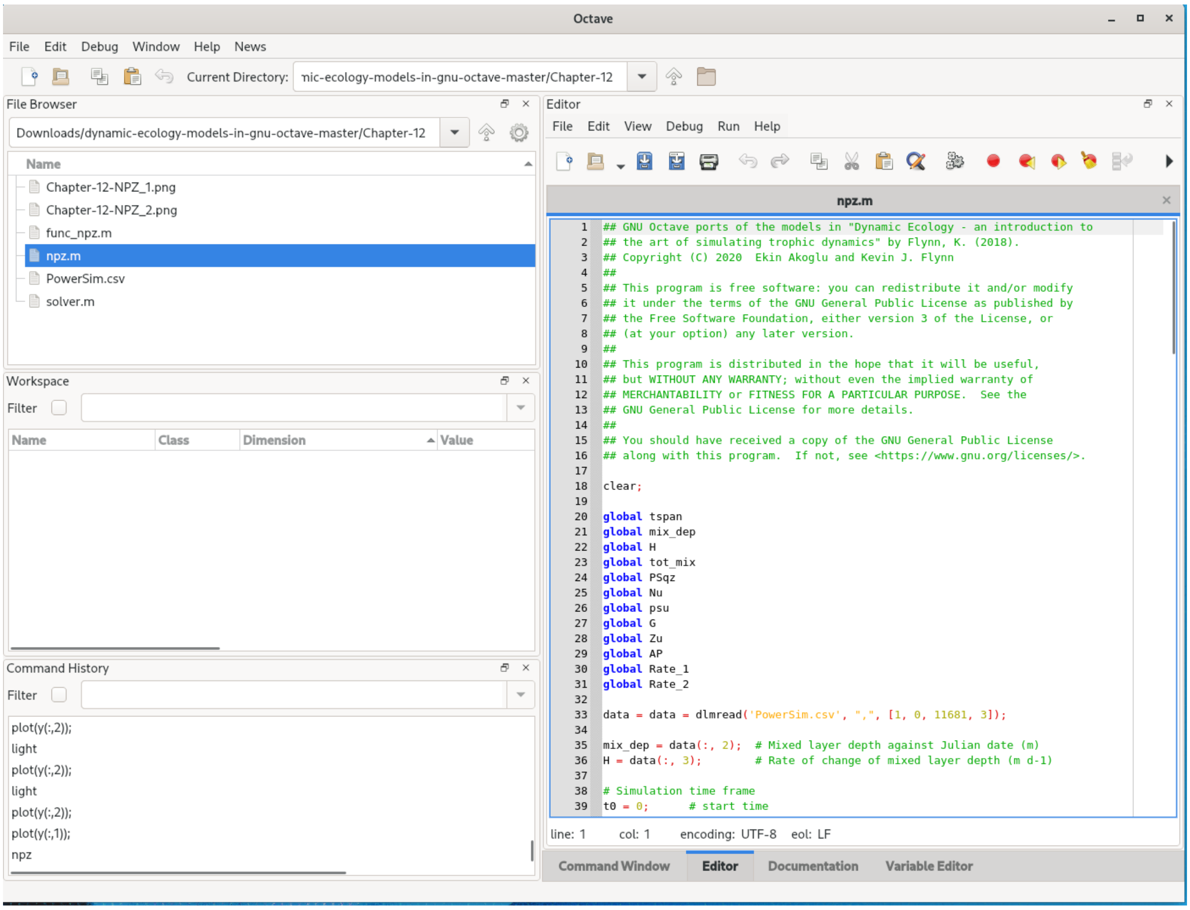


Fig. 12.2 Overview of *npz.m*

To run the “npz.m”, hit F5 on your keyboard. Alternatively, you may switch back to the “Command Window” by using the tabs at the bottom of the right part of the main window and then enter the command “npz” and press “Enter” key while you are in the “Command Window”.

The model will now run and produce a plot from simulation results once the simulation is completed (Fig. 12.3).

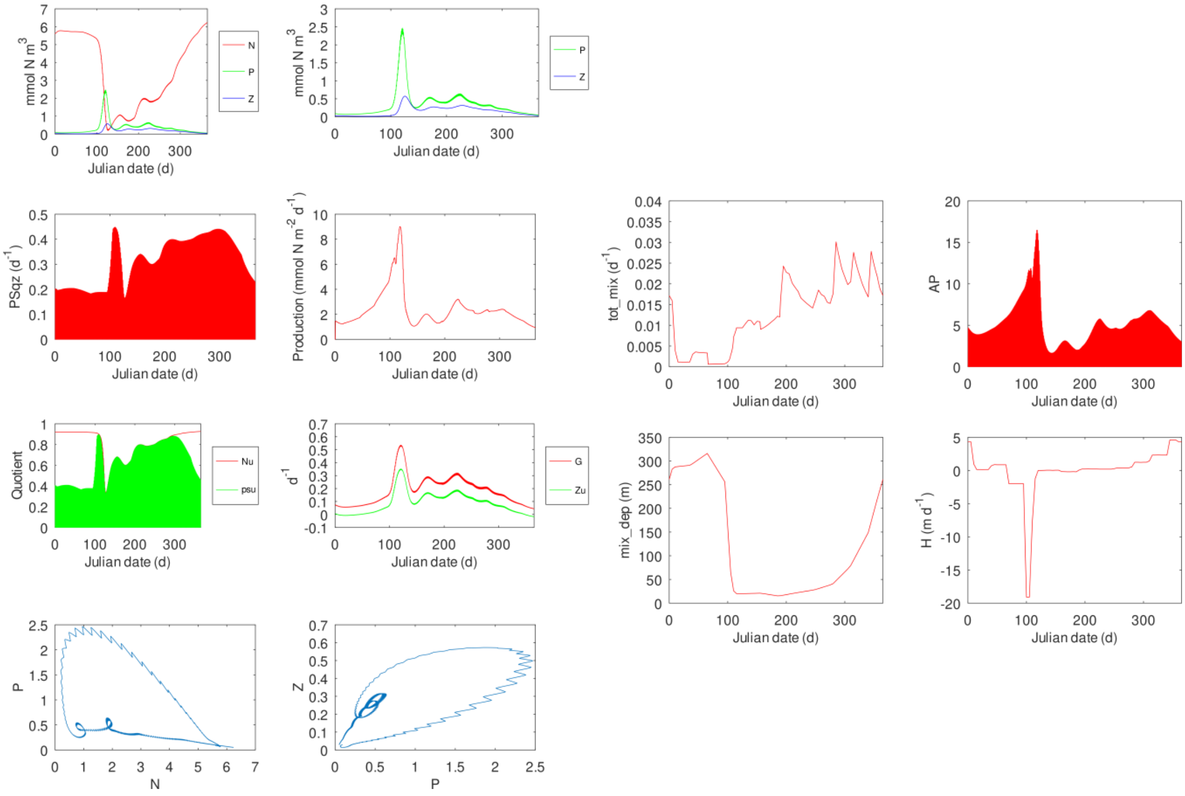


Fig. 12.3 The plots of *Dynamic Ecology* Chapter 12's model as produced by *npz.m*

12.2 Experimenting with the model

To experiment with the model as detailed in section "12.6 Things to explore" of *Dynamic Ecology* above all, you need to run the model for three years by modifying the variable "tfinal" value in "npz.m" on line 38. Further, you need to change values of the model constants in file "func_npz.m" (Fig. 12.4). You can refer to the comments (lines prepended with a hashtag and coloured in green) next to the variable names to understand what each variable corresponds to.

If you make a mistake, you can always undo/redo using the arrow buttons just above the Octave's Editor Window, and if you cannot resolve the problem, just download a new copy of the original GNU Octave model, and start over again.

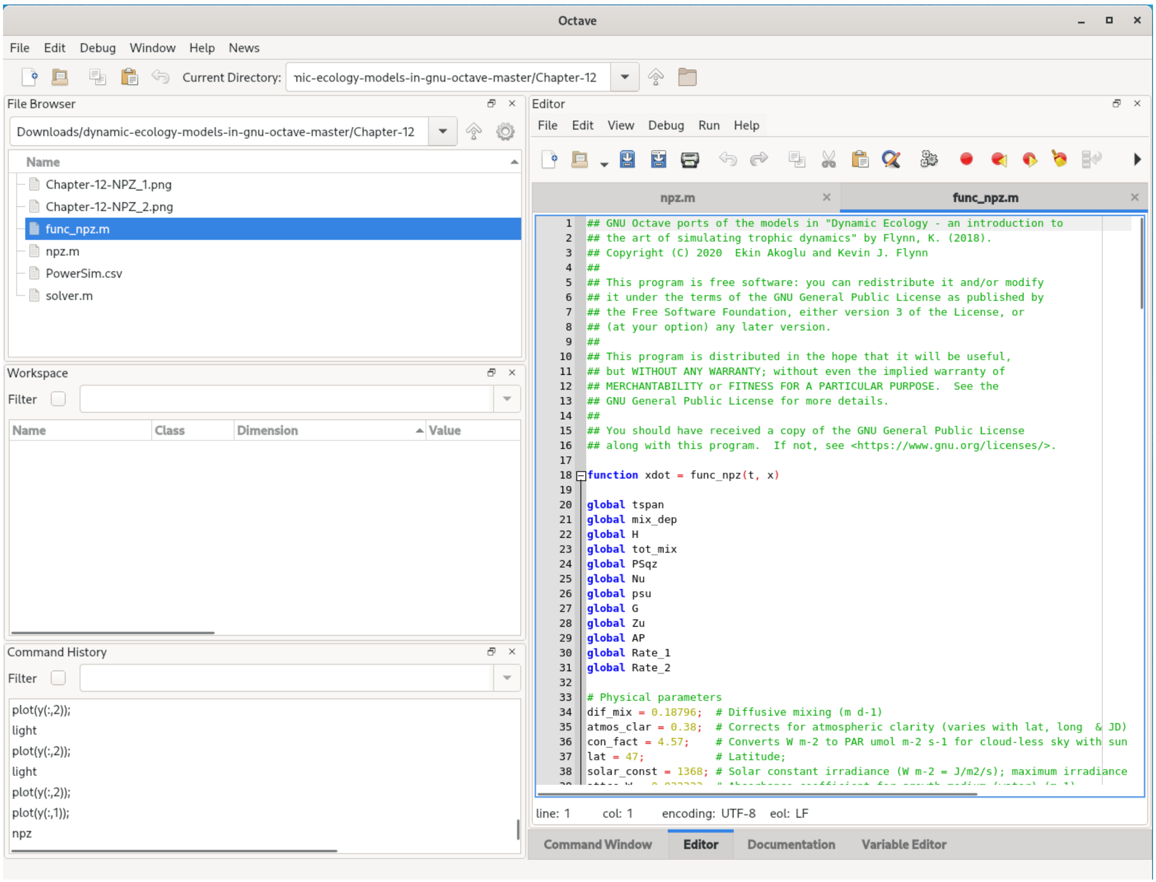


Fig. 12.4 The derivative function of *Dynamic Ecology* Chapter 12's model.

12.3 GNU Octave code

This section, running over the following pages, provides a complete dump of the GNU Octave code as it appears in the download.

12.3.1 npz.m

```
## GNU Octave ports of the models in "Dynamic Ecology - an introduction to
## the art of simulating trophic dynamics" by Flynn, K. (2018).
## Copyright (C) 2020 Ekin Akoglu and Kevin J. Flynn

## This program is free software: you can redistribute it and/or modify
## it under the terms of the GNU General Public License as published by
## the Free Software Foundation, either version 3 of the License, or
## (at your option) any later version.

## This program is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
## GNU General Public License for more details.

## You should have received a copy of the GNU General Public License
## along with this program. If not, see https://www.gnu.org/licenses/.

clear;

global tspan
global mix_dep
global H
global tot_mix
global PSqz
global Nu
global psu
global G
global Zu
global AP
global Rate_1
global Rate_2

data = data = dlmread('PowerSim.csv', ",", [1, 0, 11681, 3]);

mix_dep = data(:, 2);  # Mixed layer depth against Julian date (m)
H = data(:, 3);        # Rate of change of mixed layer depth (m d-1)

# Simulation time frame
t0 = 0;      # start time
tfinal = 365; # end time
stepsize = 0.03125;
tspan = (t0:stepsize:tfinal); # time span

# Preallocate global arrays for speed
tot_mix = zeros(1, length(tspan)-1);
PSqz = zeros(1, length(tspan)-1);
Nu = zeros(1, length(tspan)-1);
psu = zeros(1, length(tspan)-1);
G = zeros(1, length(tspan)-1);
Zu = zeros(1, length(tspan)-1);
AP = zeros(1, length(tspan)-1);
Rate_1 = zeros(1, length(tspan)-1);
```

```
Rate_2 = zeros(1, length(tspan)-1);

# Initial conditions
N = 5.6;         # Dissolved inorganic-N (nitrate and ammonium) (mmolN m-3)
P = 0.09;        # Phytoplankton biomass (mmolN m-3)
Z = 0.029;       # Zooplankton biomass (mmolN m-3)
Corpse = 0;      # Corpse N-biomass lost from system; records cumulative loss
(mmolN m-3)
Pellets = 0;     # Zooplankton faecal pellets; records cumulative loss (mmolN m-
3)
cum_prod = 0;    # Cummulative primary production (mmolN m-2 d-1)
DAY_avg_AP = 0; # Day-averaged areal primary production (mmolN m-2 d-1)
# Initial conditions array
x0 = [N P Z Corpse Pellets cum_prod DAY_avg_AP];

# Simulate
y = solver(@func_npz, tspan, stepsize, x0);

# Plot the results
h = figure;

subplot(4, 2, 1);
plot(tspan, y(:, 1), 'r', tspan, y(:, 2), 'g', tspan, y(:, 3), 'b');
set(gca,'FontSize',12);
xlabel('Julian date (d)', 'FontSize', 12);
ylabel('mmol N m^{3}', 'FontSize', 12);
hleg = legend('N', 'P', 'Z', 'location', 'eastoutside');
set(hleg, 'FontSize', 8);
ylim([0 7]);
xlim([0 365]);

subplot(4, 2, 2);
plot(tspan, y(:, 2), 'g', tspan, y(:, 3), 'b');
set(gca,'FontSize',12);
xlabel('Julian date (d)', 'FontSize', 12);
ylabel('mmol N m^{3}', 'FontSize', 12);
hleg = legend('P', 'Z', 'location', 'eastoutside');
set(hleg, 'FontSize', 8);
ylim([0 3]);
xlim([0 365]);

subplot(4, 2, 3);
plot(tspan(2:end), PSqz, 'r');
set(gca,'FontSize',12);
xlabel('Julian date (d)', 'FontSize', 12);
ylabel('PSqz (d^{-1})', 'FontSize', 12);
ylim([0 0.5]);
xlim([0 365]);

subplot(4, 2, 4);
plot(tspan, y(:, 7), 'r');
set(gca,'FontSize',12);
xlabel('Julian date (d)', 'FontSize', 12);
ylabel('Production (mmol N m^{-2} d^{-1})', 'FontSize', 12);
xlim([0 365]);

subplot(4, 2, 5);
plot(tspan(2:end), Nu, 'r', tspan(2:end), psu, 'g');
set(gca,'FontSize',12);
xlabel('Julian date (d)', 'FontSize', 12);
ylabel('Quotient', 'FontSize', 12);
hleg = legend('Nu', 'psu', 'location', 'eastoutside');
set(hleg, 'FontSize', 8);
```

```
xlim([0 365]);

subplot(4, 2, 6);
plot(tspan(2:end), G, 'r', tspan(2:end), Zu, 'g');
set(gca,'FontSize',12);
xlabel('Julian date (d)', 'FontSize', 12);
ylabel('d^{-1}', 'FontSize', 12);
hleg = legend('G', 'Zu', 'location', 'eastoutside');
set(hleg, 'FontSize', 8);
xlim([0 365]);

subplot(4, 2, 7);
plot(y(:, 1), y(:, 2));
set(gca,'FontSize',12);
xlabel('N', 'FontSize', 12);
ylabel('P', 'FontSize', 12);

subplot(4, 2, 8);
plot(y(:, 2), y(:, 3));
set(gca,'FontSize',12);
xlabel('P', 'FontSize', 12);
ylabel('Z', 'FontSize', 12);

set(gcf, 'PaperPosition', [0.25000 2.50000 9.00000 12.00000]);
print(h, 'Chapter-12-NPZ_1.png', '-dpng', '-color');

h2 = figure;

subplot(2, 2, 1);
plot(tspan(2:end), tot_mix, 'r');
set(gca,'FontSize',12);
xlabel('Julian date (d)', 'FontSize', 12);
ylabel('tot\_mix (d^{-1})', 'FontSize', 12);
set(hleg, 'FontSize', 8);
xlim([0 365]);

subplot(2, 2, 2);
plot(tspan(2:end), AP, 'r');
set(gca,'FontSize',12);
xlabel('Julian date (d)', 'FontSize', 12);
ylabel('AP', 'FontSize', 12);
set(hleg, 'FontSize', 8);
xlim([0 365]);

subplot(2, 2, 3);
plot(tspan, mix_dep, 'r');
set(gca,'FontSize',12);
xlabel('Julian date (d)', 'FontSize', 12);
ylabel('mix\_dep (m)', 'FontSize', 12);
xlim([0 365]);

subplot(2, 2, 4);
plot(tspan, H, 'r');
set(gca,'FontSize',12);
xlabel('Julian date (d)', 'FontSize', 12);
ylabel('H (m d^{-1})', 'FontSize', 12);
xlim([0 365]);

print(h2, 'Chapter-12-NPZ_2.png', '-dpng', '-color');
```

12.3.2 func_npz.m

```
## GNU Octave ports of the models in "Dynamic Ecology - an introduction to
## the art of simulating trophic dynamics" by Flynn, K. (2018).
## Copyright (C) 2020 Ekin Akoglu and Kevin J. Flynn

## This program is free software: you can redistribute it and/or modify
## it under the terms of the GNU General Public License as published by
## the Free Software Foundation, either version 3 of the License, or
## (at your option) any later version.

## This program is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
## GNU General Public License for more details.

## You should have received a copy of the GNU General Public License
## along with this program. If not, see https://www.gnu.org/licenses/.

function xdot = func_npz(t, x)

global tspan
global mix_dep
global H
global tot_mix
global PSqz
global Nu
global psu
global G
global Zu
global AP
global Rate_1
global Rate_2

# Physical parameters
dif_mix = 0.18796;  # Diffusive mixing (m d-1)
atmos_clar = 0.38;  # Corrects for atmospheric clarity (varies with lat, long  &
JD) (dl)
con_fact = 4.57;    # Converts W m-2 to PAR umol m-2 s-1 for cloud-less sky with
sun (dl)
lat = 47;           # Latitude;
solar_const = 1368; # Solar constant irradiance (W m-2 = J/m2/s); maximum
irradiance to Earth from the sun (W m-2)
attco_W = 0.032323; # Absorbance coefficient for growth medium (water) (m-1)
abco_Chl = 0.02;    # Light absorbance coefficient for chlorophyll (m2 (mgChl)-
1)

# Nutrient parameters
ext_NO3 = 7.25;     # Nitrate concentration below mixed layer (mmolN m-3)
remin_frac = 0.167; # Fraction remineralised (dl)

# Phytoplankton parameters
alpha = 7.00E-06;     # Slope of Chl-specific PE curve       (m2 g-1 chl.a)*(gC
umol-1 photon)
ChlC = 0.06;          # Mass ratio content of chlorophyll:C in the phytoplankton
(gChl (gC)-1)
inflow_Phy = 10 * 14; # Concentration of incoming phytoplankton biomass (mgN m-
3)
phy_k = 0.5;          # Half saturation constant for Nu (mmolN m-3)
Pmax = 0.5;           # Phytoplankton maximum N-specific growth rate (gN (gN)-1
d-1)
NC = 0.15;            # Mass ratio content of N-biomass:C in the phytoplankton
(gN (gC)-1)
```

```
P_mort = 0.05;        # Mortality rate for phytoplankton (d-1)

# Zooplankton parameters
AE = 0.75;          # Assimilation efficiency (dl)
ex_rate = 0.05;     # Excretion rate (d-1)
G_max = 0.7;        # Maximum N-specific grazing rate of zooplankton on
phytoplankton (d-1)
K_pred = 0.761354; # Half saturation constant for predat,on (mmolN m-3)
Z_mort = 0.564669; # Closure constant (dl);

## Auxiliaries
# Selection of only positive values of H
if H(t - 1) > 0
  H_plus = H(t - 1);
else
  H_plus = 0;
endif

# Total mixing across the ergocline (d-1)
tot_mix(t - 1) = (dif_mix + H_plus) / mix_dep(t - 1);

# Current time as fraction of day (dl)
frac_day = tspan(t - 1) - floor(tspan(t - 1));

# Julian day; note the 10d offset (starting the year on 22nd of December) (d)
JD = 365 * (((tspan(t - 1) + 10) / 365) - floor((tspan(t - 1) + 10) / 365));

# Current time as fraction of day in hours (hrs)
t_24 = 24 * frac_day;

# Degree of hour angle away from noon (default 12:00) (dl)
deg_hr = abs(12 - t_24) * 15;

# Hour angle radians (rad)
r_hr = deg_hr * pi / 180;

# Latitude in radians (rad)
r_lat = lat * pi / 180;

# Solar declination angle (rad)
sol_deca = 23.45 * sin(2 * pi * (284 + JD) * 0.00274) * pi / 180;

# Cosine of zenith angle (dl)
coszen = max(sin(r_lat) * sin(sol_deca) + cos(r_lat) * cos(sol_deca) *
cos(r_hr), 0);

# Angle the sun makes with the vertical (solar zenith angle) (rad)
theta1 = acos(coszen);

# Angle the sun makes with the vertical (solar zenith angle) (degrees)
deg_1 = theta1 * deg2rad(1.0);

# Proportion of light incident with the water surface that is just under the
surface, accounting for reflectance (dl)
E_enter = 1 - (1.15e-06 * deg_1^3 - 69.1340e-06 * deg_1^2 + 0.001 * deg_1 +
0.0187);

# Earth radius vector
r_vec  = 1 / (1 + 0.033 * cos(2 * pi * JD * 0.00274))^0.5;

# Value of coszen at noon (hence COS(0) at end of definition) (dl)
Noon_coszen = max(sin(r_lat) * sin(sol_deca) + cos(r_lat) * cos(sol_deca) *
cos(0), 0);
```

```
# Maximum irradiance (at noon) on this Julian date (W m-2)
Noon_Wm2 = solar_const / r_vec / r_vec * Noon_coszen;

if coszen > 0
  mult1 = 1;
else
  mult1 = 0;
endif
# Irradiance at given hour and day; W m-2 [W = J s-1; i.e. J/m2/s] (W m-2)
Wm2 = solar_const / r_vec / r_vec * coszen * mult1;

# Light actually entering water (just under surface), accounting for reflectance
(W m-2)
Wm2_enter = Wm2 * E_enter * atmos_clar;

# Photon m-2 s-1 PFD just under surface (umol)
nat_PFD = Wm2_enter * con_fact;

# Nitrate input and nutrient-N output (mmolN m-3 d-1)
N_mix = (ext_NO3 - x(1)) * tot_mix(t - 1);

# Phytoplankton-N specific coefficient for light absorbance (m2 (mgN)-1)
abco_PhyN = abco_Chl * ChlC / NC;

# Specific slope of PE curve ((m2)*(umol-1 photon))
alpha_u = alpha * ChlC;

# Attenuation coefficient to phytoplankton N-biomass (m-1)
attco_Phy = abco_PhyN * x(2) * 14;

# Total attenuation (dl)
att_tot = mix_dep(t - 1) * (attco_W + attco_Phy);

# Negative exponent of total attenuation (dl)
exatt = exp(-att_tot);

# Index of N-limitation (dl)
Nu(t - 1) = x(1) / (x(1) + phy_k);

# Maximum photosynthetic rate down-regulated in consequence of nutrient stress
(d-1)
PSqmax = Pmax * Nu(t - 1);

# Intermediate in depth-integrated photosynthesis rate (d)
pytq = (alpha_u * nat_PFD * 24 * 60 * 60) / PSqmax;

# Phytoplankton N-specific growth rate (d-1)
PSqz(t - 1) = PSqmax * (log(pytq + sqrt(1 + pytq^2)) - log(pytq * exatt + sqrt(1
+ (pytq * exatt)^2))) / att_tot;

# Relative photosynthetic rate (dl)
psu(t - 1) = PSqz(t - 1) / Pmax;

# N-assimilation by phytoplankton (mmolN m-3 d-1)
N_ass = x(2) * PSqz(t - 1);

# Loss of phytoplankton by death (mmolN m-3 d-1)
P_death = x(2) * P_mort;

# Removal of phytoplankton by mixing (mmolN m-3 d-1)
P_mix = x(2) * tot_mix(t - 1);
```

```
# Closure term on zooplankton (mmolN m-3 d-1)
Z_death = Z_mort * x(3)^2;

# Remineralisation of zooplankton corpses to nutrient-N within mixed layer
(mmolN m-3 d-1)
corpse_remin = Z_death * remin_frac;

# Grazing rate by zooplankton on phytoplankton (mmolN m-3 d-1)
pred = x(3) * G_max * x(2) / (x(2) + K_pred);

# N-specific grazing rate (d-1)
G(t - 1) = G_max * (x(2) / (x(2) + K_pred));

# N-specific zooplankton growth rate
Zu(t - 1) = (G_max * (x(2) / (x(2) + K_pred)) * AE) - ex_rate;

# Defecation by zooplankton (mmolN m-3 d-1)
defec = (1 - AE) * pred;

# Regeneration of N by zooplankton (mmolN m-3 d-1)
excret = x(3) * ex_rate;

# Remineralisation of faecal pellets (mmolN m-3 d-1)
pellet_remin = defec * remin_frac;

# Removal of zooplankton by mixing (mmolN m-3 d-1)
Z_mix = x(3) * tot_mix(t - 1);

# Depth integrated areal primary production (mmolN m-2 d-1)
AP(t - 1) = N_ass * mix_dep(t - 1);

# Intermediate calc
Rate_1(t - 1) =AP(t - 1);

# Intermediate calc; to average over 1 time unit (day)
if t < 34
  Rate_2(t - 1) = 0;
else
  Rate_2(t - 1) =      Rate_1(t - 33);
endif

## State equations
# Inorganic-N
xdot(1, 1) = excret + pellet_remin + corpse_remin + N_mix - N_ass;

# Phytoplankton
xdot(1, 2) = N_ass - pred - P_mix - P_death;

# Zooplankton
xdot(1, 3) = pred - excret - defec - Z_death;

# Corpse
xdot(1, 4) = Z_death - corpse_remin;

# Pellets
xdot(1, 5) = defec - pellet_remin;

# cum_prod
xdot(1, 6) = Rate_1(t - 1);

# DAY_avg_AP
xdot(1, 7) = Rate_1(t - 1) - Rate_2(t - 1);
```

```
endfunction
```

13. Sensitivity (Risk) Analyses

Chapter 13 in *Dynamic Ecology* (Flynn 2018) discusses the important role of sensitivity analyses in both model development (to ensure the model is robust and also reacts to changes in parameter values in the appropriate direction and magnitude) and in simulations.

No model should be deployed "in anger" unless it has successfully passed one or ideally both of a steady-state and a dynamic assessment. Tools to undertake such assessments vary between modelling platforms. A proprietary platform such as Powersim Studio may be equipped with built-in tools to readily undertake such studies. In other platforms, you will have to develop a subroutine to undertake and report such analyses.

There is a toolbox called SAFE (Sensitivity Analysis For Everybody) that was developed by Pianosi et al. (2015) for the application of Global Sensitivity Analysis (GSA) in GNU Octave. The toolbox can be downloaded at https://www.safetoolbox.info/info-and-documentation/ .

In deployment of the model, sensitivity analyses have a different role, to determine how safe is the output of the model given uncertainties in the input values. Hence the use of the term "risk" rather than "sensitivity". In a financial model, that risk may be in whether a company turns a profit or goes bankrupt, but in an ecological setting it could consider the likelihood of a harmful algal bloom developing under different weather conditions or nutrient loading.

Please read chapter 13 in *Dynamic Ecology* if only to make yourself aware of the topic.

14. Tuning (Optimising the Fit) to Data and Validation

Chapter 14 in *Dynamic Ecology* (Flynn 2018) discusses the topic of tuning your model to data in order to optimise the behaviour of the model against that of the system that you are trying to simulate. This step should be undertaken only with a model that has already been subjected to (and passed) a sensitivity analysis. Validation refers to a comparison of your model output, with its parameters tuned to accord to a different input data set, with a different data set, for different conditions.

Tools to undertake such a tuning process vary between modelling platforms. A proprietary platform such as Powersim Studio may be equipped with built-in tools to undertake such a process. In other platforms, you will have to develop a subroutine to tune your model. The default is, of course, to run your model with a manual changing of parameter values until you get the type of response that you are satisfied with. This may be relatively painless with a simple model, but rapidly becomes very time-consuming, if not overwhelming, with a large complex model.

To the best of our knowledge, GNU Octave does not provide a toolbox for optimising the models' fit to data; therefore, you will have to develop a subroutine to tune your model. Tuning to data is an iterative process and such a subroutine may include, but not limited to, the steps outlined below:

1. Load reference time series data
2. Run the model
3. Assess model skill against reference time series data by using commonly employed statistical metrics (e.g. Stow *et al*. 2003). If satisfied then stop; otherwise, proceed to step 4.
4. Tune model parameters
5. Go to step 2

Please read chapter 14 in *Dynamic Ecology* if only to make yourself aware of the topic.

15. Variable Stoichiometry - A Simple C:N-based Phytoplankton Model in GNU Octave

This Chapter provides information on running the model in chapter 15 of *Dynamic Ecology* (Flynn 2018), running through each of the steps. It is assumed that you have installed the Octave interface (Chapter 2).

Hitherto we have considered simple, single-currency, models. In all instances we have used N as the currency, and hence described all ecological interactions with respect to the transfer of that element. We could have used P instead of N, but of course in real systems many elements, and indeed many biochemicals (notably so-called essential amino and fatty acids) are transferred and that transference could be rate limiting for growth. Most obviously C (for both structure and energy) is transferred.

The ratio of different elements and of biochemicals to each other differs between organisms, and also within organisms of different physiological status. Such ratios are termed stoichiometric ratios. In consequence of differences in stoichiometry, during trophic interactions there is scope for interactions developing because of an excess in one component (element or biochemical) in the food versus that in the consumer. This excess needs to be removed. The flip side is a shortage in one or other components that causes an inadequacy in the nutritional value of the diet.

Models that describe the resultant interactions of differences in chemical composition are multi-currency, exhibit differential stoichiometry, and usually (in reflection of changes in stoichiometry in the individual organism depending on their nutrient history) they are variable stoichiometric. Thus, for example, they describe variations in C:N:P in each organism functional type during trophic interactions. In much of ecological research, while it becomes very obvious (as we shall see in this chapter and in Chapter 16) when operating variable stoichiometric models that such variability has profound impacts on the dynamics of ecology, it took the advent of the now classic work of Sterner & Elser (2002) on “Ecological Stoichiometry” to bring this matter to the attention of mainstream ecology.

In this chapter, and the next, we commence a consideration of the challenge in modelling variable stoichiometry. Typically, in our context, this refers to elemental stoichiometry such as C:N:P within organic material (organisms, faecal material, dissolved organics); for simplicity we shall restrict our considerations here to C:N. We could also apply the concept to biochemical stoichiometric ratios such as protein:carbohydrate, down to ratios of specific amino acids to protein, or PUFA to total fatty acids. In organisms these ratios vary within bounds depending on the physiology. The ratios are also bound by biochemistry; for example, the C:N in protein is constrained by the C:N in the constituent amino acids, and ultimately by the fundamentals of chemical valency.

Please see chapter 15 in *Dynamic Ecology* for more contextual information, explanations for model construction, and (in the final sections of that chapter) ideas for experimenting and developing your models.

15.1 Running the model

Navigate to the folder "Chapter-15" in the "File Browser" by double-clicking it. You will now see the script files and related figures of the model in the "File Browser" (Fig. 15.1).

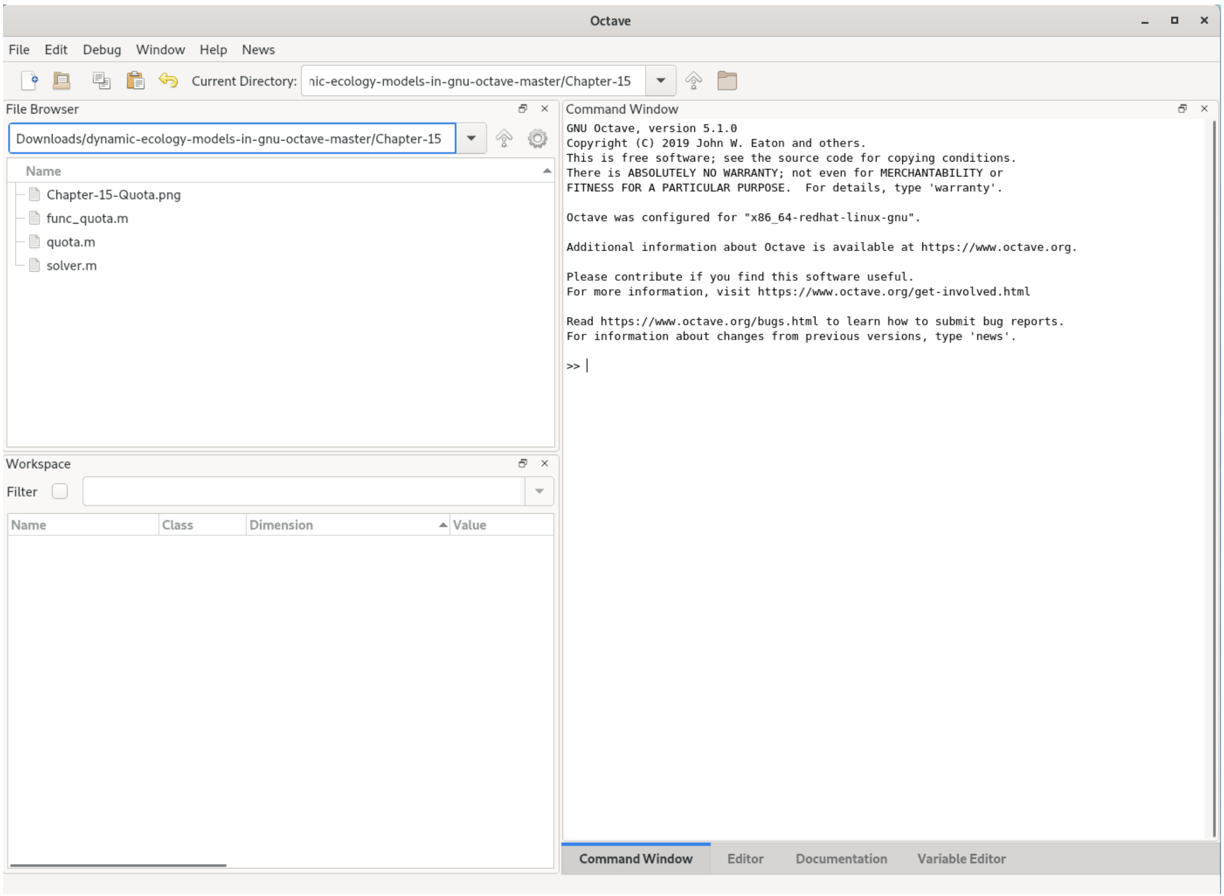


Fig. 15.1 Model of Chapter 15 in *Dynamic Ecology* and its related script files and figures.

Double-click the "quota.m" file to open it (Fig. 15.2).

The script file will be opened in the editor window of GNU Octave. As you will notice, the file includes a series of GNU Octave commands and comments (lines prepended by a hashtag and coloured in green) that explain what each line of the code corresponds to.

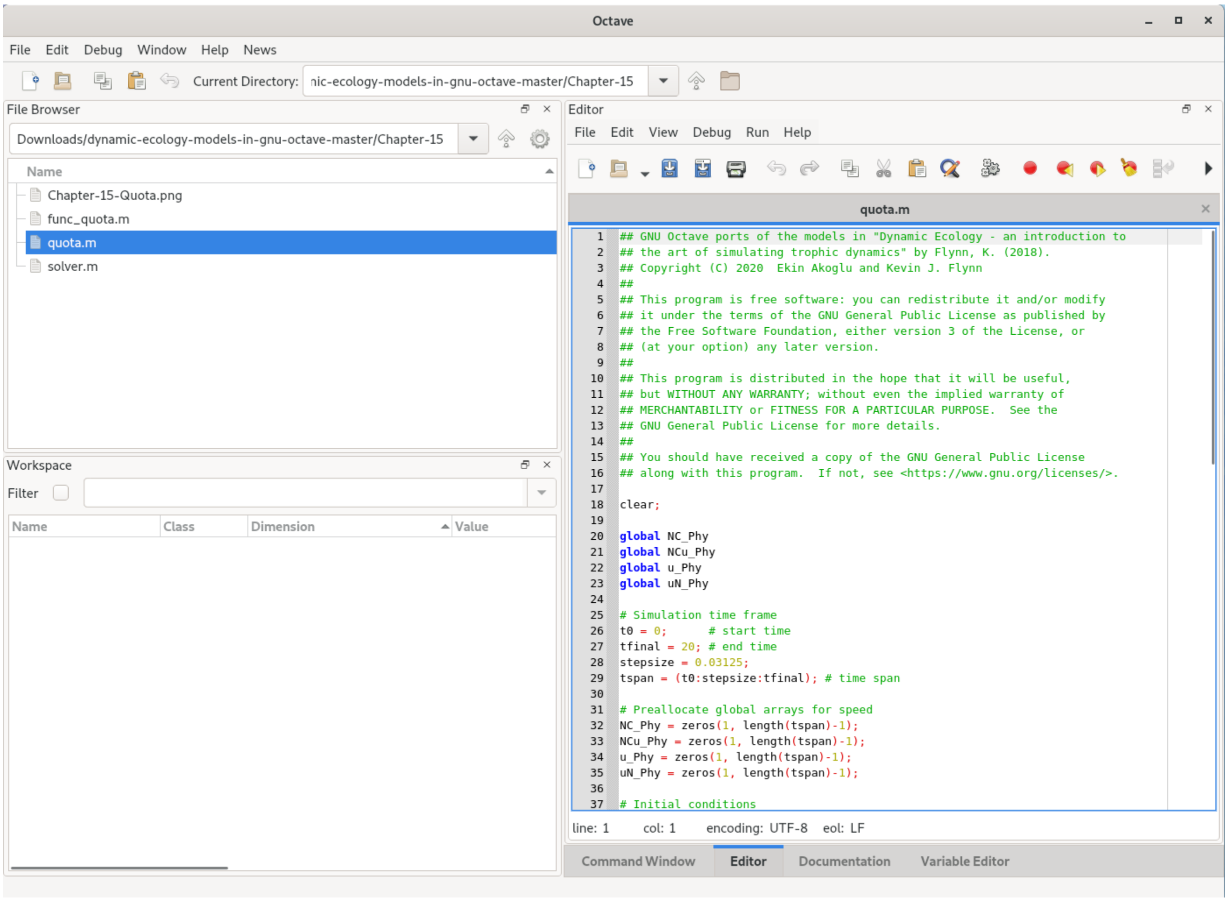


Fig. 15.2 Overview of *quota.m*

To run the "quota.m", hit F5 on your keyboard. Alternatively, you may switch back to the "Command Window" by using the tabs at the bottom of the right part of the main window and then enter the command "quota" and press "Enter" key while you are in the "Command Window".

The model will now run and produce a plot from simulation results once the simulation is completed (Fig. 15.3).

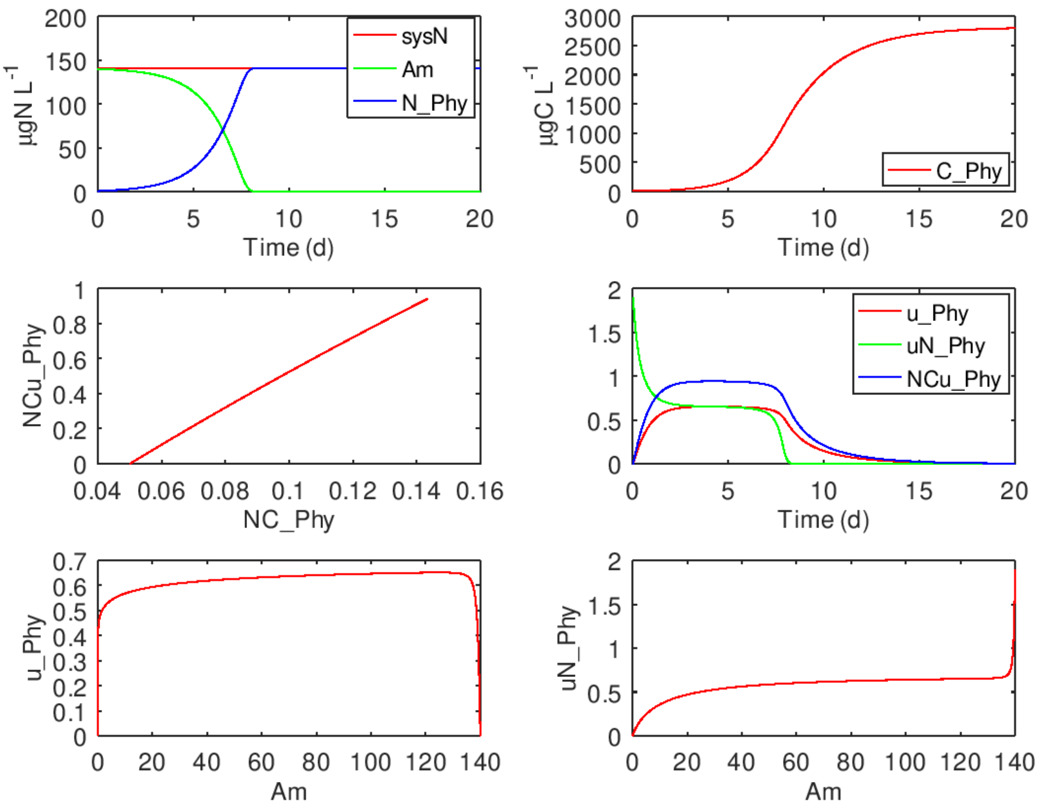


Fig. 15.3 The plots of *Dynamic Ecology* Chapter 15's model as produced by *quota.m*

15.2 Experimenting with the model

To experiment with the model as detailed in section "15.7 Things to explore" of *Dynamic Ecology* You need to change values of the model constants in file "func_quota.m" (Fig. 15.4). You can refer to the comments (lines prepended with a hashtag and coloured in green) next to the variable names to understand what each variable corresponds to.

If you make a mistake, you can always undo/redo using the arrow buttons just above the Octave's Editor Window, and if you cannot resolve the problem, just download a new copy of the original GNU Octave model, and start over again.

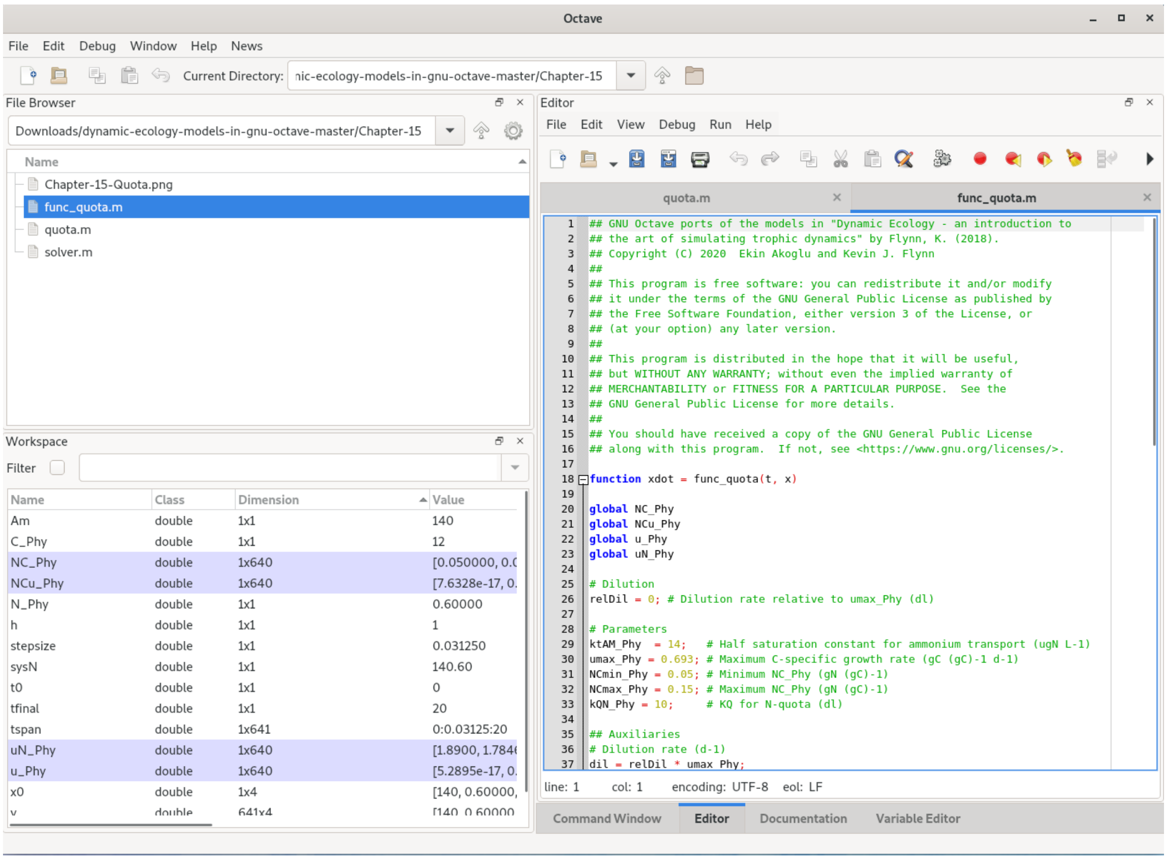


Fig. 15.4 The derivative function of *Dynamic Ecology* Chapter 15's model.

15.3 GNU Octave code

This section, running over the following pages, provides a complete dump of the GNU Octave code as it appears in the download.

15.3.1 quota.m

```
## GNU Octave ports of the models in "Dynamic Ecology - an introduction to
## the art of simulating trophic dynamics" by Flynn, K. (2018).
## Copyright (C) 2020 Ekin Akoglu and Kevin J. Flynn

## This program is free software: you can redistribute it and/or modify
## it under the terms of the GNU General Public License as published by
## the Free Software Foundation, either version 3 of the License, or
## (at your option) any later version.

## This program is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
## GNU General Public License for more details.

## You should have received a copy of the GNU General Public License
## along with this program. If not, see https://www.gnu.org/licenses/.

clear;

global NC_Phy
global NCu_Phy
global u_Phy
global uN_Phy

# Simulation time frame
t0 = 0;       # start time
tfinal = 20; # end time
stepsize = 0.03125;
tspan = (t0:stepsize:tfinal); # time span

# Preallocate global arrays for speed
NC_Phy = zeros(1, length(tspan)-1);
NCu_Phy = zeros(1, length(tspan)-1);
u_Phy = zeros(1, length(tspan)-1);
uN_Phy = zeros(1, length(tspan)-1);

# Initial conditions
Am = 14 * 10;          # Ammonium-N (ugN -L-1)
C_Phy = 12;            # Phytoplankton-C (ugC -L-1)
N_Phy = C_Phy * 0.05; # Phytoplankton-N (ugN -L-1)
sysN    = Am + N_Phy;    # System N (ugN L-1)
# Initial conditions arrayfun
x0 = [Am, N_Phy, C_Phy, sysN];

# Simulate
y = solver(@func_quota, tspan, stepsize, x0);

# Plot the results
h = figure;

subplot(3, 2, 1)
plot(tspan, y(:, 4), 'r', tspan, y(:, 1), 'g', tspan, y(:, 2), 'b');
```

```
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
ylabel('\mugN L^{-1}', 'FontSize', 12);
legend('sysN', 'Am', 'N\_Phy');

subplot(3, 2, 2)
plot(tspan, y(:, 3), 'r');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
ylabel('\mugC L^{-1}', 'FontSize', 12);
legend('C\_Phy', 'location', 'southeast');

subplot(3, 2, 3)
plot(NC_Phy, NCu_Phy, 'r');
set(gca,'FontSize',12);
xlabel('NC\_Phy', 'FontSize', 12);
ylabel('NCu\_Phy', 'FontSize', 12);

subplot(3, 2, 4)
plot(tspan(2:end), u_Phy, 'r', tspan(2:end), uN_Phy, 'g', tspan(2:end), NCu_Phy,
'b');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
legend('u\_Phy', 'uN\_Phy', 'NCu\_Phy');

subplot(3, 2, 5)
plot(y(2:end, 1), u_Phy, 'r');
set(gca,'FontSize',12);
xlabel('Am', 'FontSize', 12);
ylabel('u\_Phy', 'FontSize', 12);

subplot(3, 2, 6)
plot(y(2:end, 1), uN_Phy, 'r');
set(gca,'FontSize',12);
xlabel('Am', 'FontSize', 12);
ylabel('uN\_Phy', 'FontSize', 12);

print(h, 'Chapter-15-Quota.png', '-dpng', '-color');
```

15.3.2 func_quota.m

```
## GNU Octave ports of the models in "Dynamic Ecology - an introduction to
## the art of simulating trophic dynamics" by Flynn, K. (2018).
## Copyright (C) 2020 Ekin Akoglu and Kevin J. Flynn

## This program is free software: you can redistribute it and/or modify
## it under the terms of the GNU General Public License as published by
## the Free Software Foundation, either version 3 of the License, or
## (at your option) any later version.

## This program is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
## GNU General Public License for more details.

## You should have received a copy of the GNU General Public License
## along with this program. If not, see https://www.gnu.org/licenses/.

function xdot = func_quota(t, x)

global NC_Phy
global NCu_Phy
global u_Phy
global uN_Phy

# Dilution
relDil = 0; # Dilution rate relative to umax_Phy (dl)

# Parameters
ktAM_Phy = 14;     # Half saturation constant for ammonium transport (ugN L-1)
umax_Phy = 0.693; # Maximum C-specific growth rate (gC (gC)-1 d-1)
NCmin_Phy = 0.05; # Minimum NC_Phy (gN (gC)-1)
NCmax_Phy = 0.15; # Maximum NC_Phy (gN (gC)-1)
kQN_Phy = 10;      # KQ for N-quota (dl)

## Auxiliaries
# Dilution rate (d-1)
dil = relDil * umax_Phy;

# Nutrient exchange (ugN L-1 d-1)
in_out_Am = dil * (140 - x(1));

# Washout of N_Phy (ugN L-1 d-1)
outN_Phy = x(2) * dil;

# Washout of C_Phy (ugC L-1 d-1)
outC_Phy = x(3) * dil;

# Phytoplankton N:C quota (gN (gC)-1)
NC_Phy(t - 1) = x(2) / x(3);

# Quotient for N status (dl)
NCu_Phy(t - 1) = ((1 + kQN_Phy) * (NC_Phy(t - 1) - NCmin_Phy)) / ((NC_Phy(t - 1)
- NCmin_Phy) + kQN_Phy * (NCmax_Phy - NCmin_Phy));

# C-specific growth rate controlled by N:C quota (gC (gC)-1 d-1)
u_Phy(t - 1) = umax_Phy * NCu_Phy(t - 1);

# Maximum C-specific N transport rate (gN (gC)-1 d-1)
TNmax_Phy = umax_Phy * NCmax_Phy;

# Phytoplankton C-specific N transport rate (gN (gC)-1 d-1)
```

```
NCt_Phy = TNmax_Phy * x(1) / (x(1) + ktAM_Phy);

# N-specific growth rate (gN (gN)-1 d-1)
uN_Phy(t - 1) = NCt_Phy / NC_Phy(t -1);

# Phytoplankton population uptake of ammonium-N (ugN L-1 d-1)
Nup_Phy = x(3) * NCt_Phy;

# Growth rate in phytoplankton-C (ugC L-1 d-1)
groC_Phy = x(3) * u_Phy(t - 1);

## State equations
# Ammonium
xdot(1, 1) = in_out_Am - Nup_Phy;

# Phytoplankton-N
xdot(1, 2) = Nup_Phy - outN_Phy;

# Phytoplankton-C
xdot(1, 3) = groC_Phy - outC_Phy;

# System
xdot(1, 4) = xdot(1, 1) + xdot(1, 2);

endfunction
```

16. Variable Stoichiometric Predator –Prey Model in GNU Octave

This Chapter provides information on running the model in chapter 16 of *Dynamic Ecology* (Flynn 2018), running through each of the steps. It is assumed that you have installed the Octave interface (Chapter 2).

As introduced in the previous chapter, differential stoichiometry between members of a trophic web has the ready potential to significantly affect the dynamics of ecology. Having built a description of phytoplankton growth describing variable stoichiometry (Chapter 15), and hence variable nutritional value for a consumer, here we build a consumer model to feed upon it.

Even though we make the assumption that the elemental stoichiometry of the consumer is fixed (here, as C:N), as you will see there is plenty of scope for considering interactions linking both the quantity and quality of the phytoplankton prey to consumer feeding and growth. Throughout the following the text couple *predator-prey* will be used, though in most instances *consumer-food* would apply equally.

Please see chapter 16 in *Dynamic Ecology* for more contextual information, explanations for model construction, and (in the final sections of that chapter) ideas for experimenting and developing your models.

16.1 Running the model

Navigate to the folder “Chapter-16” in the “File Browser” by double-clicking it. You will now see the script files and related figures of the model in the “File Browser” (Fig. 16.1).

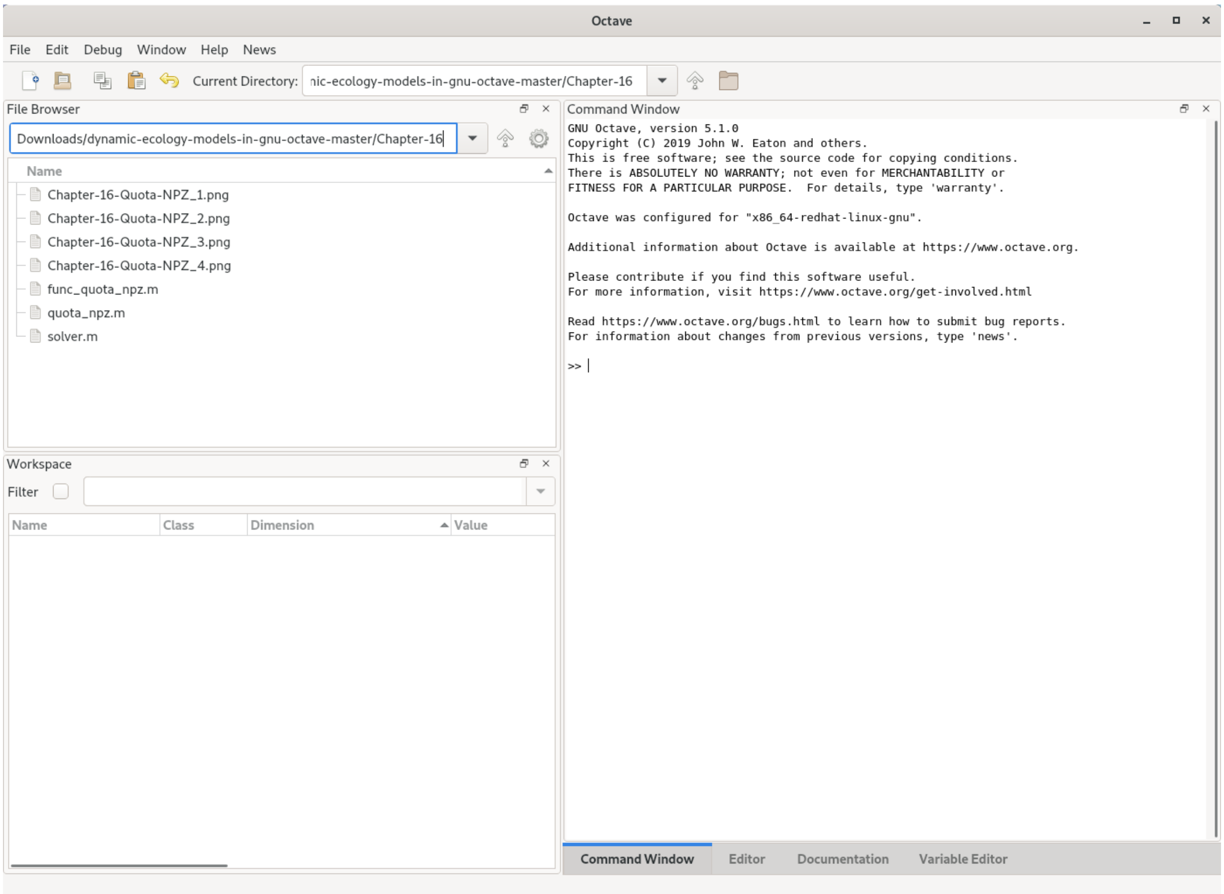


Fig. 16.1 Model of Chapter 16 in *Dynamic Ecology* and its related script files and figures.

Double-click the “quota_npz.m” file to open it (Fig. 16.2).

The script file will be opened in the editor window of GNU Octave. As you will notice, the file includes a series of GNU Octave commands and comments (lines prepended by a hashtag and coloured in green) that explain what each line of the code corresponds to.

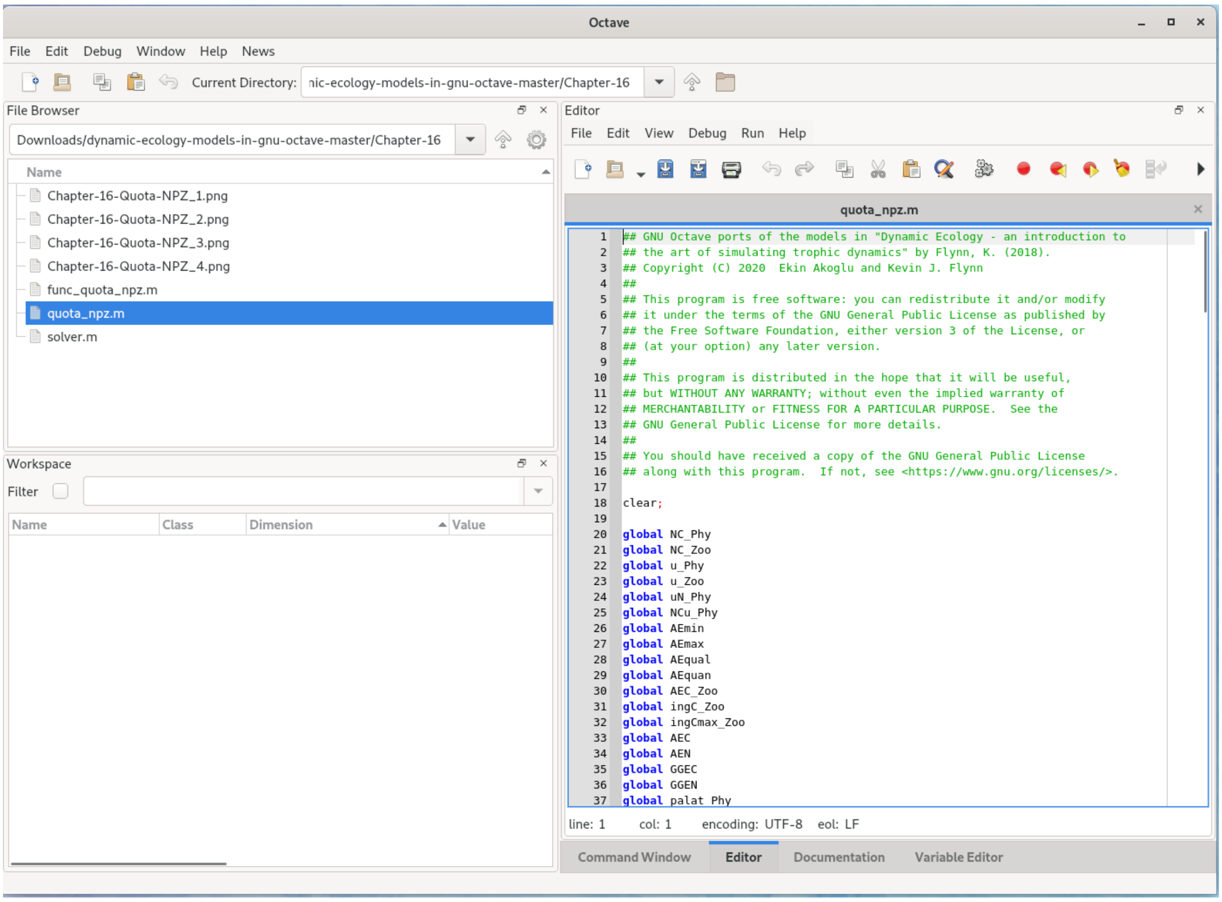


Fig. 16.2 Overview of *quota_npz.m*

To run the "quota_npz.m", hit F5 on your keyboard. Alternatively, you may switch back to the "Command Window" by using the tabs at the bottom of the right part of the main window and then enter the command "quota_npz" and press "Enter" key while you are in the "Command Window".

The model will now run and produce a plot from simulation results once the simulation is completed (Fig. 16.3).

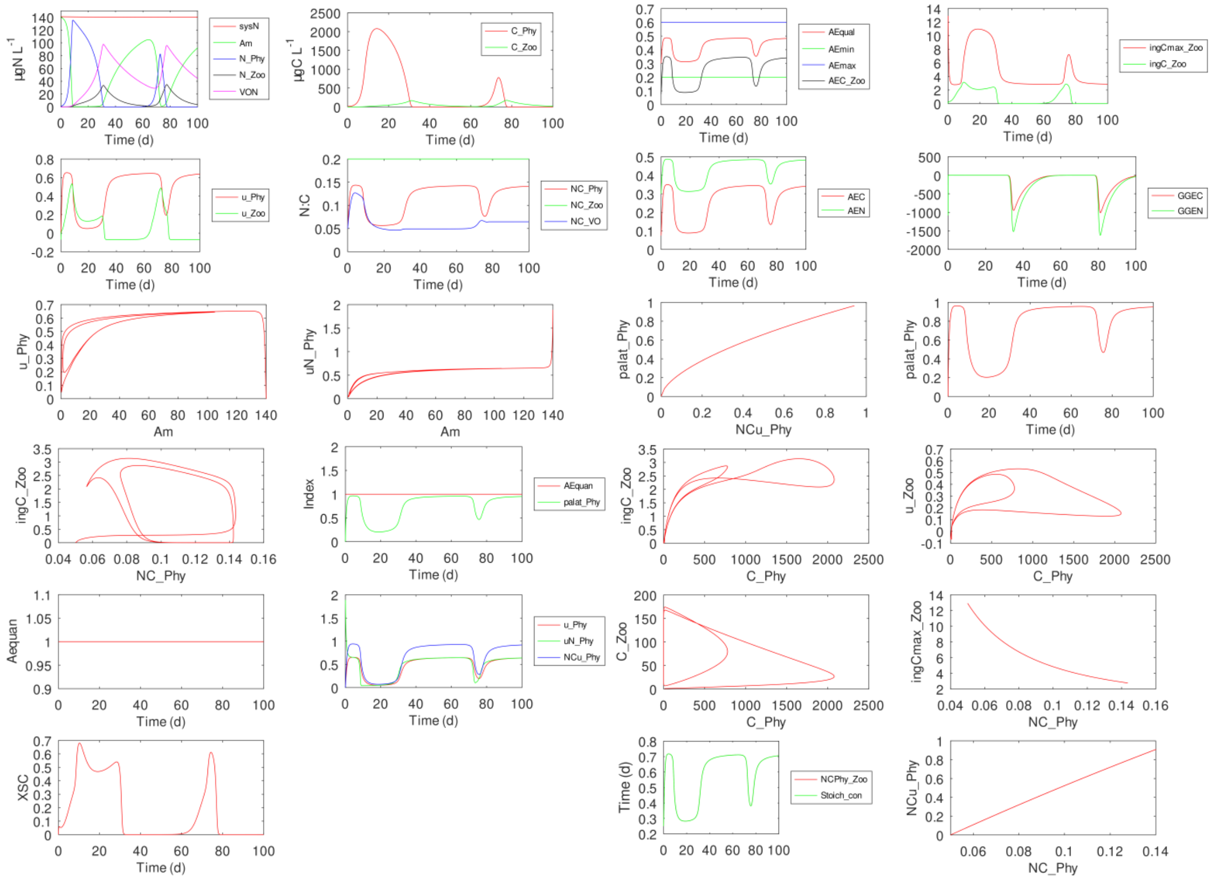


Fig. 16.3 The plots of *Dynamic Ecology* Chapter 16's model as produced by quota_*npz.m*

16.2 Experimenting with the model

To experiment with the model as detailed in section "16.12 Things to explore" of *Dynamic Ecology* You need to change values of the model constants in file "func_quota_npz.m" (Fig. 16.4). You can refer to the comments (lines prepended with a hashtag and coloured in green) next to the variable names to understand what each variable corresponds to.

If you make a mistake, you can always undo/redo using the arrow buttons just above the Octave's Editor Window, and if you cannot resolve the problem, just download a new copy of the original GNU Octave model, and start over again.

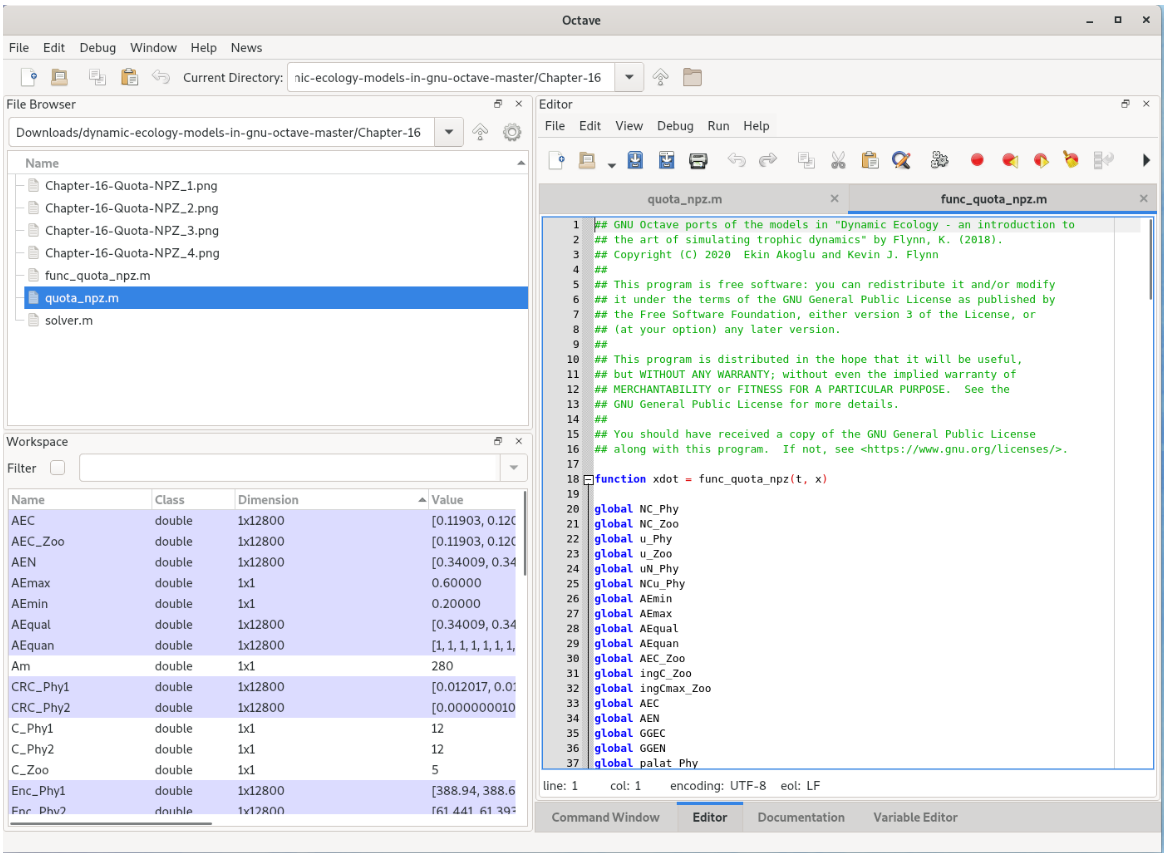


Fig. 16.4 The derivative function of *Dynamic Ecology* Chapter 16's model.

16.3 GNU Octave code

This section, running over the following pages, provides a complete dump of the GNU Octave code as it appears in the download.

16.3.1 quota_npz.m

```
## GNU Octave ports of the models in "Dynamic Ecology - an introduction to
## the art of simulating trophic dynamics" by Flynn, K. (2018).
## Copyright (C) 2020 Ekin Akoglu and Kevin J. Flynn

## This program is free software: you can redistribute it and/or modify
## it under the terms of the GNU General Public License as published by
## the Free Software Foundation, either version 3 of the License, or
## (at your option) any later version.

## This program is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
## GNU General Public License for more details.

## You should have received a copy of the GNU General Public License
## along with this program. If not, see https://www.gnu.org/licenses/.

clear;

global NC_Phy
global NC_Zoo
global u_Phy
global u_Zoo
global uN_Phy
global NCu_Phy
global AEmin
global AEmax
global AEqual
global AEquan
global AEC_Zoo
global ingC_Zoo
global ingCmax_Zoo
global AEC
global AEN
global GGEC
global GGEN
global palat_Phy
global NCPhy_Zoo
global Stoich_con
global XSC

# Simulation time frame
t0 = 0;       # start time
tfinal = 100; # end time
stepsize = 0.015625;
tspan = (t0:stepsize:tfinal); # time span

# Preallocate global arrays for speed
NC_Phy = zeros(1, length(tspan)-1);
NC_VO = zeros(1, length(tspan)-1);
u_Phy = zeros(1, length(tspan)-1);
u_Zoo = zeros(1, length(tspan)-1);
```

```
uN_Phy = zeros(1, length(tspan)-1);
NCu_Phy = zeros(1, length(tspan)-1);
AEC_Zoo = zeros(1, length(tspan)-1);
ingC_Zoo = zeros(1, length(tspan)-1);
ingCmax_Zoo = zeros(1, length(tspan)-1);
AEC = zeros(1, length(tspan)-1);
AEN = zeros(1, length(tspan)-1);
GGEC = zeros(1, length(tspan)-1);
GGEN = zeros(1, length(tspan)-1);
palat_Phy = zeros(1, length(tspan)-1);
NCPhy_Zoo= zeros(1, length(tspan)-1);
Stoich_con = zeros(1, length(tspan)-1);

# Initial conditions
Am = 14 * 10;           # Ammonium-N (ugN L-1)
C_Phy = 12;             # Phytoplankton-C (ugC L-1)
N_Phy = C_Phy * 0.05; # Phytoplankton-N (ugN L-1)
C_Zoo = 1;              # Zooplankton C-biomass (ugC L-1)
VON = 0;                # Faecal material-C (ugC L-1)
VOC = 0;                # Faecal material-N (ugN L-1)
sysN    = Am + N_Phy + C_Zoo * 0.2 + VON;    # System N (ugN L-1)
# Initial conditions arrayfun
x0 = [Am, N_Phy, C_Phy, C_Zoo, VON, VOC, sysN];

# Simulate
y = solver(@func_quota_npz, tspan, stepsize, x0);

# Plot the results
h = figure;

subplot(3, 2, 1)
plot(tspan, y(:, 7), 'r', tspan, y(:, 1), 'g', tspan, y(:, 2), 'b', tspan, y(:,
4) * 0.2, 'k', tspan, y(:, 5), 'm');
set(gca,'FontSize',12);
ylim([0 150]);
xlabel('Time (d)', 'FontSize', 12);
ylabel('\mugN L^{-1}', 'FontSize', 12);
hleg = legend('sysN', 'Am', 'N\_Phy', 'N\_Zoo', 'VON', 'location',
'eastoutside');
set(hleg, 'FontSize', 8);

subplot(3, 2, 2)
plot(tspan, y(:, 3), 'r', tspan, y(:, 4), 'g');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
ylabel('\mugC L^{-1}', 'FontSize', 12);
hleg = legend('C\_Phy', 'C\_Zoo');
set(hleg, 'FontSize', 8);

subplot(3, 2, 3)
plot(tspan(2:end), u_Phy, 'r', tspan(2:end), u_Zoo, 'g');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
hleg = legend('u\_Phy', 'u\_Zoo', 'location', 'eastoutside');
set(hleg, 'FontSize', 8);

subplot(3, 2, 4)
plot(tspan(2:end), NC_Phy, 'r', tspan(2:end), repmat(NC_Zoo, 1, length(tspan)-
1), 'g', tspan(2:end), (y(2:end, 5) ./ y(2:end, 6)), 'b');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
ylabel('N:C', 'FontSize', 12);
hleg = legend('NC\_Phy', 'NC\_Zoo', 'NC\_VO', 'location', 'eastoutside');
```

```
set(hleg, 'FontSize', 8);

subplot(3, 2, 5)
plot(y(2:end, 1), u_Phy, 'r');
set(gca,'FontSize',12);
xlabel('Am', 'FontSize', 12);
ylabel('u\_Phy', 'FontSize', 12);

subplot(3, 2, 6)
plot(y(2:end, 1), uN_Phy, 'r');
set(gca,'FontSize',12);
xlabel('Am', 'FontSize', 12);
ylabel('uN\_Phy', 'FontSize', 12);

print(h, 'Chapter-16-Quota-NPZ_1.png', '-dpng', '-color');

h2 = figure;

subplot(3, 2, 1);
plot(tspan(2:end), AEqual, 'r', tspan(2:end), repmat(AEmin, 1, length(tspan)-1),
'g', tspan(2:end), repmat(AEmax, 1, length(tspan)-1), 'b', tspan(2:end),
AEC_Zoo, 'k');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
hleg = legend('AEqual', 'AEmin', 'AEmax', 'AEC\_Zoo', 'location',
'eastoutside');
set(hleg, 'FontSize', 8);

subplot(3, 2, 2);
plot(tspan(2:end), ingCmax_Zoo, 'r', tspan(2:end), ingC_Zoo, 'g');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
hleg = legend('ingCmax\_Zoo', 'ingC\_Zoo', 'location', 'eastoutside');
set(hleg, 'FontSize', 8);

subplot(3, 2, 3);
plot(tspan(2:end), AEC, 'r', tspan(2:end), AEN, 'g');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
hleg = legend('AEC', 'AEN', 'location', 'eastoutside');
set(hleg, 'FontSize', 8);

subplot(3, 2, 4);
plot(tspan(2:end), GGEC, 'r', tspan(2:end), GGEN, 'g');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
hleg = legend('GGEC', 'GGEN', 'location', 'eastoutside');
set(hleg, 'FontSize', 8);

subplot(3, 2, 5);
plot(NCu_Phy, palat_Phy, 'r');
set(gca,'FontSize',12);
xlabel('NCu\_Phy', 'FontSize', 12);
ylabel('palat\_Phy', 'FontSize', 12);

subplot(3, 2, 6);
plot(tspan(2:end), palat_Phy, 'r');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
ylabel('palat\_Phy', 'FontSize', 12);

set(gcf, 'PaperPosition', [0.25000 2.50000 9.00000 12.00000]);
print(h2, 'Chapter-16-Quota-NPZ_2.png', '-dpng', '-color');
```

```
h3 = figure;

subplot(3, 2, 1);
plot(y(2:end, 3), ingC_Zoo, 'r');
set(gca,'FontSize',12);
xlabel('C\_Phy', 'FontSize', 12);
ylabel('ingC\_Zoo', 'FontSize', 12);

subplot(3, 2, 2);
plot(y(2:end, 3), u_Zoo, 'r');
set(gca,'FontSize',12);
xlabel('C\_Phy', 'FontSize', 12);
ylabel('u\_Zoo', 'FontSize', 12);

subplot(3, 2, 3);
plot(y(:, 3), y(:, 4), 'r');
set(gca,'FontSize',12);
xlabel('C\_Phy', 'FontSize', 12);
ylabel('C\_Zoo', 'FontSize', 12);

subplot(3, 2, 4);
plot(NC_Phy, ingCmax_Zoo, 'r');
set(gca,'FontSize',12);
xlabel('NC\_Phy', 'FontSize', 12);
ylabel('ingCmax\_Zoo', 'FontSize', 12);

subplot(3, 2, 5);
plot(tspan(2:end), NCPhy_Zoo, 'r', tspan(2:end), Stoich_con, 'g');
set(gca,'FontSize',12);
ylabel('Time (d)');
hleg = legend('NCPhy\_Zoo', 'Stoich\_con', 'location', 'eastoutside');
set(hleg, 'FontSize', 8);

subplot(3, 2, 6);
plot(NC_Phy, NCu_Phy, 'r');
set(gca,'FontSize',12);
xlim([0.05 0.14]);
xlabel('NC\_Phy', 'FontSize', 12);
ylabel('NCu\_Phy', 'FontSize', 12);

set(gcf, 'PaperPosition', [0.25000 2.50000 9.00000 12.00000]);
print(h3, 'Chapter-16-Quota-NPZ_3.png', '-dpng', '-color');

h4 = figure;

subplot(3, 2, 1);
plot(NC_Phy, ingC_Zoo, 'r');
set(gca,'FontSize',12);
xlabel('NC\_Phy', 'FontSize', 12);
ylabel('ingC\_Zoo', 'FontSize', 12);

subplot(3, 2, 2);
plot(tspan(2:end), AEquan, 'r', tspan(2:end), palat_Phy, 'g');
set(gca,'FontSize',12);
ylim([0 2]);
xlabel('Time (d)', 'FontSize', 12);
ylabel('Index', 'FontSize', 12);
hleg = legend('AEquan', 'palat\_Phy', 'location', 'eastoutside');
set(hleg, 'FontSize', 8);

subplot(3, 2, 3);
plot(tspan(2:end), AEquan, 'r');
```

```
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
ylabel('Aequan', 'FontSize', 12);

subplot(3, 2, 4);
plot(tspan(2:end), u_Phy, 'r', tspan(2:end), uN_Phy, 'g', tspan(2:end), NCu_Phy,
'b');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
hleg = legend('u\_Phy', 'uN\_Phy', 'NCu\_Phy', 'location', 'eastoutside');
set(hleg, 'FontSize', 8);

subplot(3, 2, 5);
plot(tspan(2:end), XSC, 'r');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
ylabel('XSC', 'FontSize', 12);

set(gcf, 'PaperPosition', [0.25000 2.50000 9.00000 12.00000]);
print(h4, 'Chapter-16-Quota-NPZ_4.png', '-dpng', '-color');
```

16.3.2 func_quota_npz.m

```
## GNU Octave ports of the models in "Dynamic Ecology - an introduction to
## the art of simulating trophic dynamics" by Flynn, K. (2018).
## Copyright (C) 2020 Ekin Akoglu and Kevin J. Flynn

## This program is free software: you can redistribute it and/or modify
## it under the terms of the GNU General Public License as published by
## the Free Software Foundation, either version 3 of the License, or
## (at your option) any later version.

## This program is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
## GNU General Public License for more details.

## You should have received a copy of the GNU General Public License
## along with this program. If not, see https://www.gnu.org/licenses/.

function xdot = func_quota_npz(t, x)

global NC_Phy
global NC_Zoo
global u_Phy
global u_Zoo
global uN_Phy
global NCu_Phy
global AEmin
global AEmax
global AEqual
global AEquan
global AEC_Zoo
global ingC_Zoo
global ingCmax_Zoo
global AEC
global AEN
global GGEC
global GGEN
global palat_Phy
global NCPhy_Zoo
global Stoich_con
global XSC

# Dilution
relDil = 0.05; # Dilution rate relative to umax_Phy (dl)

# Phytoplankton parameters
ktAM_Phy = 14;  # Half saturation constant for ammonium transport (ugN L-1)
umax_Phy = 0.693; # Maximum C-specific growth rate (gC (gC)-1 d-1)
NCmin_Phy = 0.05; # Minimum NC_Phy (gN (gC)-1)
NCmax_Phy = 0.15; # Maximum NC_Phy (gN (gC)-1)
kQN_Phy = 10;     # KQ for N-quota (dl)

# Zooplankton parameters
AEmax = 0.6;      # Maximum AE for N (dl)
AEmin = 0.2;      # Maximum AE for N (dl)
BR_Zoo = 0.1;     # Basal respiration rate as a proportion of umax_Zoo (dl)
kAE = 1.00e+03;   # Constant for control of AE in response to prey quality (dl)
kCPhy_Zoo = 140;  # Half saturation of zooplankton predation on phytoplankton
(ugC L-1)
kGTT = 100;       # Curve control for density dependant inefficiency (ug C L-1)
minAE_mult = 1;   # Minimum AEC scalar for density dependant inefficiency (dl)
NC_Zoo = 0.2;     # N:C of zooplankton (gN gC-1)
```

```
SDA = 0.3;         # Specific dynamic action (dl)
thresC_Phy = 0.014; # Threshold for predation upon phytoplankton (ugC L-1)
tox_Phy = 0.6;     # Toxicity scalar (dl)
umax_Zoo = 0.693; # Zooplankton maximum growth rate (d-1)

# Voided materials parameters
NCmax = 0.3;       # Maximum mass ratio of N:C which could be attained in the
organic form (gN (gC)-1)

## Auxiliaries
# Dilution rate (d-1)
dil = relDil * umax_Phy;

# Nutrient exchange (ugN L-1 d-1)
in_out_Am = dil * (140 - x(1));

# Washout of N_Phy (ugN L-1 d-1)
outN_Phy = x(2) * dil;

# Washout of C_Phy (ugC L-1 d-1)
outC_Phy = x(3) * dil;

# Phytoplankton N:C quota (gN (g C)-1)
NC_Phy(t - 1) = x(2) / x(3);

# Quotient for N status (dl)
NCu_Phy(t - 1) = ((1 + kQN_Phy) * (NC_Phy(t - 1) - NCmin_Phy)) / ((NC_Phy(t - 1)
- NCmin_Phy) + kQN_Phy * (NCmax_Phy - NCmin_Phy));

# C-specific growth rate controlled by N:C quota (gC (gC)-1 d-1)
u_Phy(t - 1) = umax_Phy * NCu_Phy(t - 1);

# Maximum C-specific N transport rate (gN (gC)-1 d-1)
TNmax_Phy = umax_Phy * NCmax_Phy;

# Phytoplankton C-specific N transport rate (gN (gC)-1 d-1)
NCt_Phy = TNmax_Phy * x(1) / (x(1) + ktAM_Phy);

# N-specific growth rate (gN (gN)-1 d-1)
uN_Phy(t - 1) = NCt_Phy / NC_Phy(t -1);

# Phytoplankton population uptake of ammonium-N (ugN L-1 d-1)
Nup_Phy = x(3) * NCt_Phy;

# Growth rate in phytoplankton-C (ugC L-1 d-1)
groC_Phy = x(3) * u_Phy(t - 1);

# Ratio of NC in prey compared to predator (dl)
NCPhy_Zoo(t - 1) = NC_Phy(t -1) / NC_Zoo;

# Selection of release of N related to difference in food to consumer N:C (dl)
Stoich_con(t - 1) = min(NCPhy_Zoo(t - 1), 1);

# AEC scalar for density dependant inefficiency (dl)
AEquan(t - 1) = (1 - minAE_mult) * (1 - x(3) / (x(3) + kGTT)) + minAE_mult;

# Efficiency parameter for assimilation (dl)
AEqual(t - 1)  = AEmin + (AEmax - AEmin) * Stoich_con(t - 1) / (Stoich_con(t -
1) + kAE) * (1 + kAE);

# Operational AE for C (dl)
AEC_Zoo(t - 1) = Stoich_con(t - 1) * AEqual(t - 1)  * AEquan(t - 1);
```

```
# Zooplankton basal respiration rate (d-1)
BR = umax_Zoo * BR_Zoo;

# Maximum ingestion rate by zooplankton (gC (gC)-1 d-1)
ingCmax_Zoo(t -1) = (umax_Zoo * (1 + SDA) + BR) / AEC_Zoo(t - 1) ;

# Palatability index (0 not palatable) (dl)
palat_Phy(t - 1) = (NCu_Phy(t - 1) + 1.0e-6)^tox_Phy;

# Ingestion rate pf prey into zooplankton (gC (gC)-1 d-1)
if x(3) > thresC_Phy
  ingC_Zoo(t -1) = palat_Phy(t - 1) * ingCmax_Zoo(t -1) * (x(3) - thresC_Phy) /
(x(3) - thresC_Phy + kCPhy_Zoo);
else
  ingC_Zoo(t -1) = 0;
endif

# C available for support of respiration (gC (gC)-1 d-1)
if Stoich_con(t - 1) < 1
  XSC(t - 1) = AEqual(t - 1)  * ingC_Zoo(t -1) * (1 - Stoich_con(t - 1));
else
  XSC(t - 1) = 0;
endif

# Basal respiration that is met bu respiration of excess C in diet (gC (gC)-1 d-
1)
if BR <= XSC(t - 1)
  BRi = BR;
else
  BRi = XSC(t - 1);
endif

# Balance of basal respiration that cannot be met from dietary excees C (gC
(gC)-1 d-1)
BRb = BR - BRi;

# Grazing upon phytoplankton population (ugC L-1 d-1)
grazC_Phy = x(4) * ingC_Zoo(t -1);

# Grazing upon phytoplankton population in terms of N (ugN L-1 d-1)
grazN_Phy = x(4) * ingC_Zoo(t -1) * NC_Phy(t -1);

# Assimilation rate into zooplankton (gC (gC)-1 d-1)
assC_Zoo = AEC_Zoo(t - 1)  * ingC_Zoo(t -1);

# Assimilation of C into zooplankton population biomass (ugC L-1)
assC = x(4) * assC_Zoo;

# Zooplankton respiration rate (gC (gC)-1 d-1)
resC_Zoo = BRb + assC_Zoo * SDA;

# Zooplankton population respiration (ugC L-1 d-1)
respC = x(4) * resC_Zoo;

# Zooplankton growth rate (gC (gC)-1 d-1)
u_Zoo(t - 1) = assC_Zoo - resC_Zoo;

# Amount of N initially in the organic form to be voided to maintain constant
predator N:C (gN (gC)-1 d-1)
XSassN = ingC_Zoo(t -1) * NC_Phy(t -1) - assC_Zoo * NC_Zoo;

# AE in terms of C (dl)
AEC(t - 1) = assC_Zoo / ingC_Zoo(t -1);
```

```
# AR in terms of N (dl)
AEN(t - 1) = (assC_Zoo * NC_Zoo) / (ingC_Zoo(t -1) * NC_Phy(t -1));

# Voiding of C by zooplankton (gC (gC)-1 d-1)
voidC_Zoo = ingC_Zoo(t -1) - assC_Zoo - BRi;

# Population rate of C voiding (ugC L-1 d-1)
voidC = x(4) * voidC_Zoo;

# Voiding of N by zooplankton (gN (gC)-1 d-1)
if (XSassN / voidC_Zoo) > NCmax
  voidN_Zoo = voidC_Zoo * NCmax;
else
  voidN_Zoo = XSassN;
endif

# Population rate of N voiding (ugN L-1 d-1)
voidN = x(4) * voidN_Zoo;

# Zooplankton ammonium regeneration (gN (gC)-1 d-1)
DINr = resC_Zoo * NC_Zoo + XSassN - voidN_Zoo;

# GGE in terms of C (dl)
GGEC(t - 1) = (ingC_Zoo(t -1) - voidC_Zoo - resC_Zoo - BRi) / ingC_Zoo(t -1);

# GGE in terms of N (dl)
GGEN(t - 1) = (ingC_Zoo(t -1) * NC_Phy(t -1) - voidN_Zoo - DINr) / (ingC_Zoo(t -
1) * NC_Phy(t -1));

# Zooplankton population regeneration of ammonium (ugN L-1 d-1)
reg_Am = x(4) * DINr;

# Washout of voided C (ugC L-1 d-1)
out_VOC = x(6) * dil;

# Washout of voided N (ugN L-1 d-1)
out_VON = x(5) * dil;

# Washout of zooplankton biomass (ugC L-1 d-1)
outC_Zoo = x(4) * dil;

## State equations
# Ammonium
xdot(1, 1) = in_out_Am + reg_Am - Nup_Phy;

# Phytoplankton-N
xdot(1, 2) = Nup_Phy - grazN_Phy - outN_Phy;

# Phytoplankton-C
xdot(1, 3) = groC_Phy - grazC_Phy - outC_Phy;

# Zooplankton-C
xdot(1, 4) = assC - respC - outC_Zoo;

# VON
xdot(1, 5) = voidN - out_VON;

# VOC
xdot(1, 6) = voidC - out_VOC;

# System
xdot(1, 7) = xdot(1, 1) + xdot(1, 2) + xdot(1, 4) * NC_Zoo + xdot(1, 5);
```

```
endfunction
```

17. Allometry & Prey Selection Model in GNU Octave

This Chapter provides information on running the model in chapter 17 of *Dynamic Ecology* (Flynn 2018), running through each of the steps. It is assumed that you have installed the Octave interface (Chapter 2).

So far we have considered very simple food chain interactions. In the real world, all organisms face choices between resources and, through a combination of biochemical, physiological and behavioural responses, they make their selections. But there is more to this than meets the eye and simulating different facets of the events is non-trivial. Behaviour of predators is also perhaps the factor that most attracts humans to observe organisms playing the survival game in the wild, or indeed in a test tube.

So, you are hungry - what will you do about it? Will you move around in search of food, and risk meeting your predator and being eaten up yourself? Will you try and eat anything you come across, that you bump into, or will you be picky? Will that pickiness vary depending on how hungry you are? What happens if your favourite food deteriorates in quality, perhaps even becoming noxious? How efficiently do you process your food? And what happens when you become satiated; do you sit there digesting your meal, or do you still race around chasing the next meal?

Please see chapter 17 in *Dynamic Ecology* for more contextual information, explanations for model construction, and (in the final sections of that chapter) ideas for experimenting and developing your models.

17.1 Running the model

Navigate to the folder "Chapter-17" in the "File Browser" by double-clicking it. You will now see the script files and related figures of the model in the "File Browser" (Fig. 17.1).

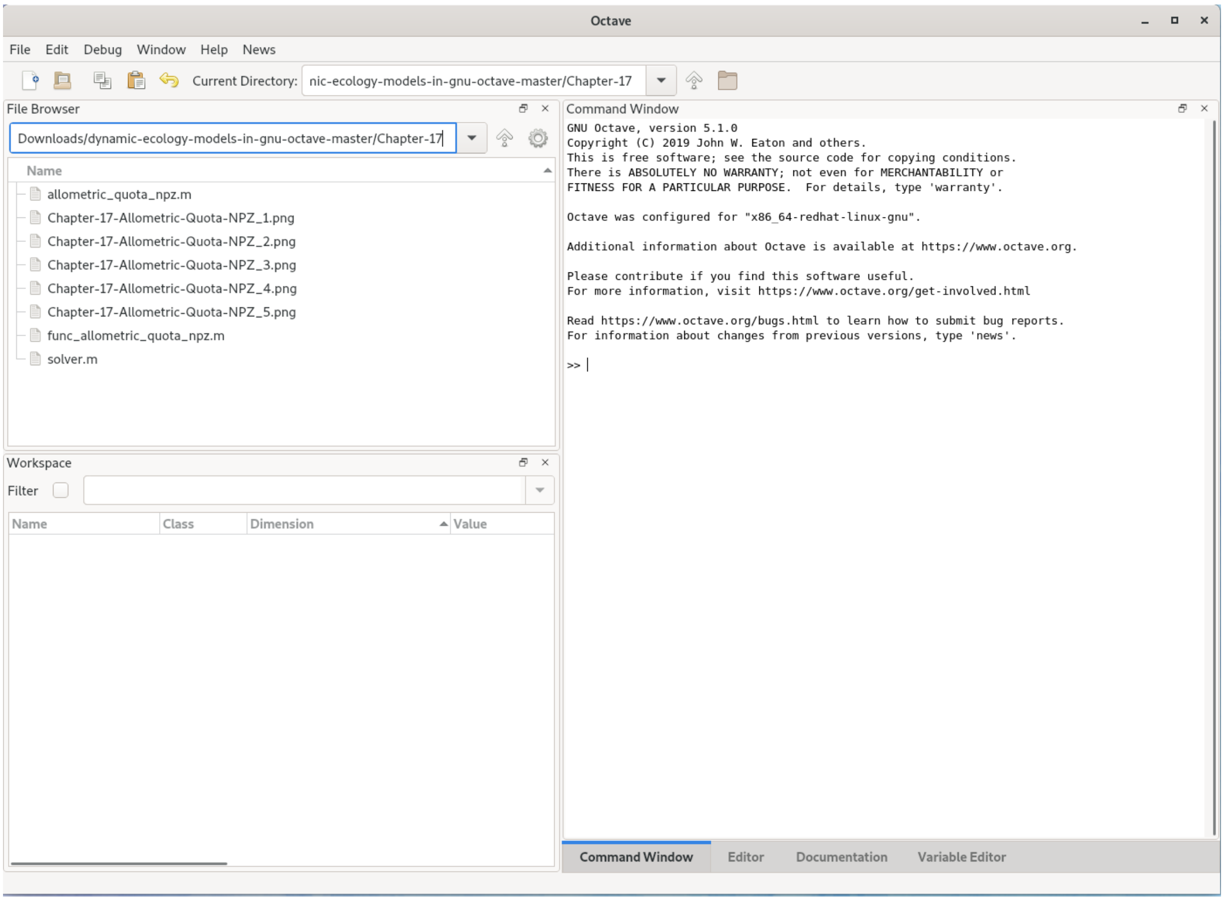


Fig. 17.1 Model of Chapter 17 in *Dynamic Ecology* and its related script files and figures.

Double-click the "allometric_quota_npz.m" file to open it (Fig. 17.2).

The script file will be opened in the editor window of GNU Octave. As you will notice, the file includes a series of GNU Octave commands and comments (lines prepended by a hashtag and coloured in green) that explain what each line of the code corresponds to.

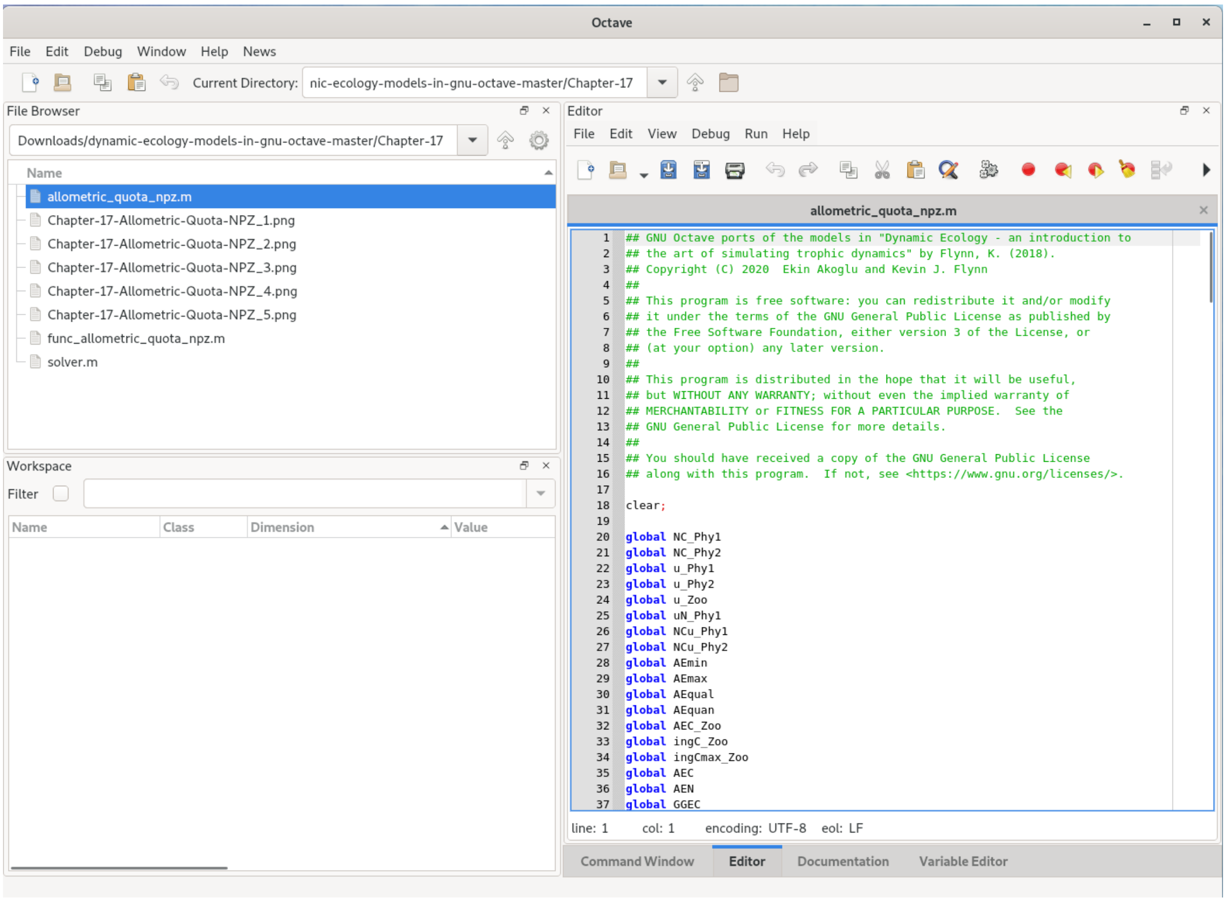


Fig. 17.2 Overview of allometric_*quota_npz.m*

To run the "allometric_quota_npz.m", hit F5 on your keyboard. Alternatively, you may switch back to the "Command Window" by using the tabs at the bottom of the right part of the main window and then enter the command "allometric_quota_npz" and press "Enter" key while you are in the "Command Window".

The model will now run and produce a plot from simulation results once the simulation is completed (Fig. 17.3).

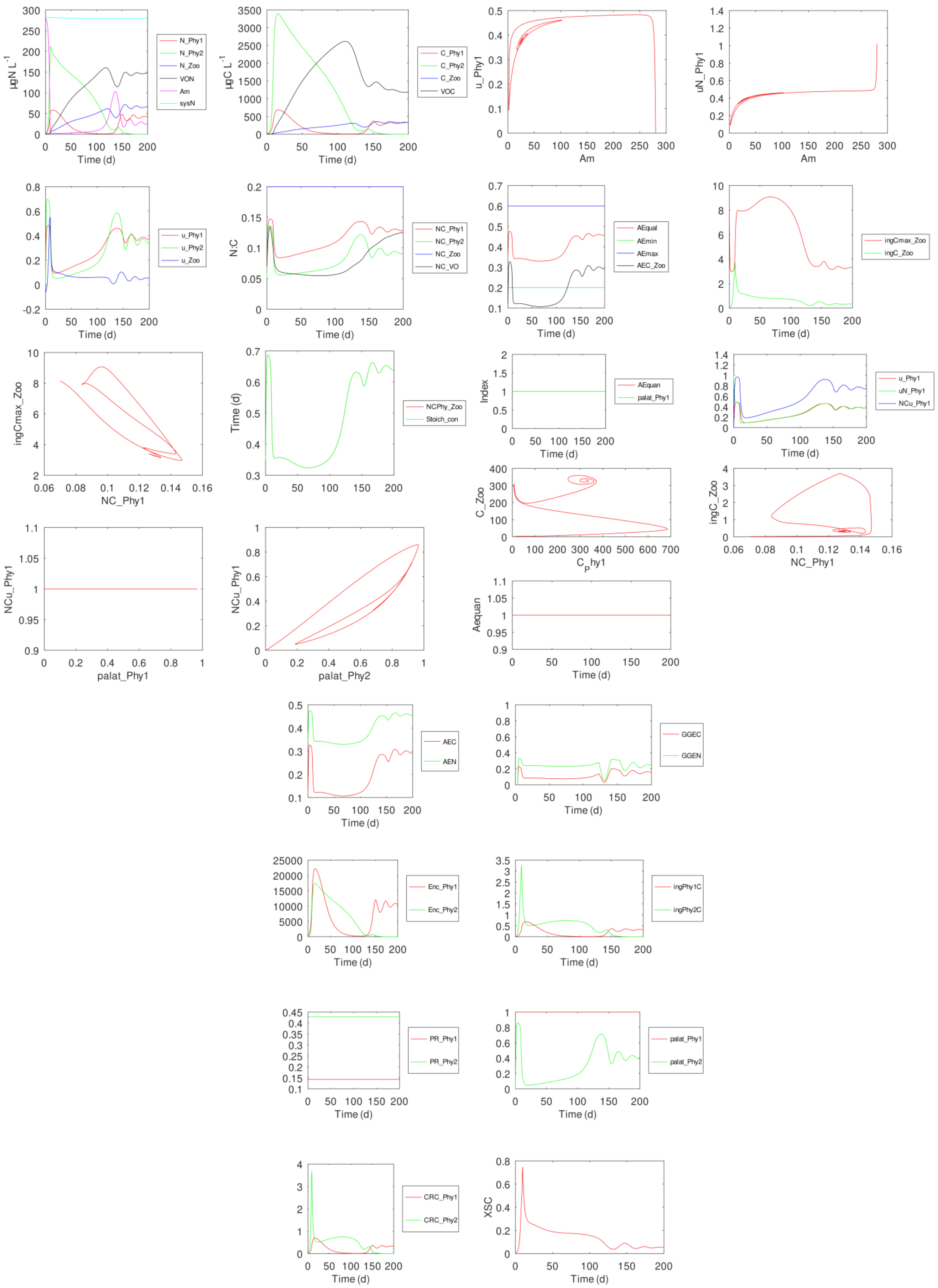


Fig. 17.3 The plots of *Dynamic Ecology* Chapter 17's model as produced by allometric_quota_*npz.m*

17.2 Experimenting with the model

To experiment with the model as detailed in section "17.10 Things to explore" of *Dynamic Ecology* you need to change values of the model constants in file "func_allometric_quota_npz.m" (Fig. 17.4). You can refer to the comments (lines prepended with a hashtag and coloured in green) next to the variable names to understand what each variable corresponds to.

If you make a mistake, you can always undo/redo using the arrow buttons just above the Octave's Editor Window, and if you cannot resolve the problem, just download a new copy of the original GNU Octave model, and start over again.

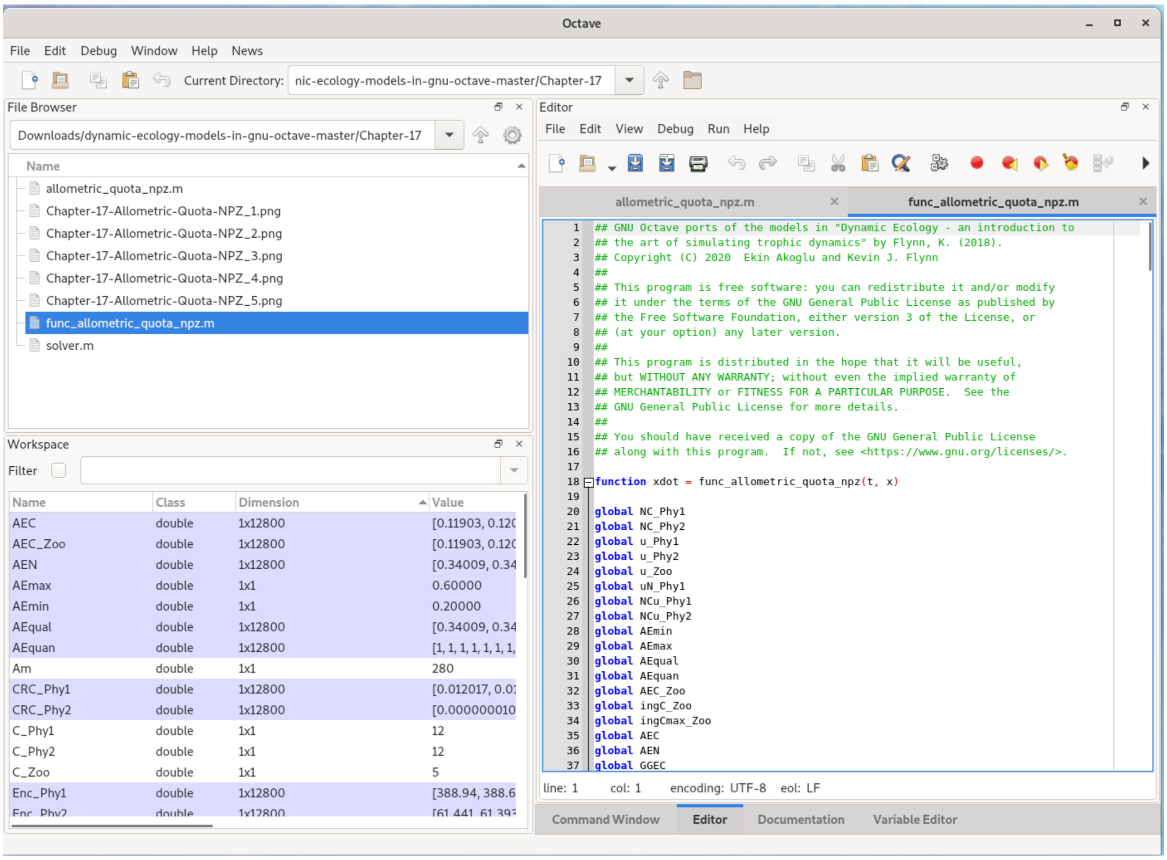


Fig. 17.4 The derivative function of *Dynamic Ecology* Chapter 17's model.

17.3 GNU Octave code

This section, running over the following pages, provides a complete dump of the GNU Octave code as it appears in the download.

17.3.1 allometric_quota_npz.m

```
## GNU Octave ports of the models in "Dynamic Ecology - an introduction to
## the art of simulating trophic dynamics" by Flynn, K. (2018).
## Copyright (C) 2020 Ekin Akoglu and Kevin J. Flynn

## This program is free software: you can redistribute it and/or modify
## it under the terms of the GNU General Public License as published by
## the Free Software Foundation, either version 3 of the License, or
## (at your option) any later version.

## This program is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
## GNU General Public License for more details.

## You should have received a copy of the GNU General Public License
## along with this program. If not, see https://www.gnu.org/licenses/.

clear;

global NC_Phy1
global NC_Phy2
global u_Phy1
global u_Phy2
global u_Zoo
global uN_Phy1
global NCu_Phy1
global NCu_Phy2
global AEmin
global AEmax
global AEqual
global AEquan
global AEC_Zoo
global ingC_Zoo
global ingCmax_Zoo
global AEC
global AEN
global GGEC
global GGEN
global palat_Phy1
global palat_Phy2
global Enc_Phy1
global Enc_Phy2
global ingPhy1C
global ingPhy2C
global PR_Phy1
global PR_Phy2
global CRC_Phy1
global CRC_Phy2
global NCPhy_Zoo
global Stoich_con
global XSC
```

```
# Simulation time frame
t0 = 0;       # start time
tfinal = 200; # end time
stepsize = 0.015625;
tspan = (t0:stepsize:tfinal); # time span

# Preallocate global arrays for speed
NC_Phy1 = zeros(1, length(tspan)-1);
NC_Phy2 = zeros(1, length(tspan)-1);
u_Phy1 = zeros(1, length(tspan)-1);
u_Phy2 = zeros(1, length(tspan)-1);
u_Zoo = zeros(1, length(tspan)-1);
uN_Phy1 = zeros(1, length(tspan)-1);
NCu_Phy1 = zeros(1, length(tspan)-1);
NCu_Phy2 = zeros(1, length(tspan)-1);
AEC_Zoo = zeros(1, length(tspan)-1);
ingC_Zoo = zeros(1, length(tspan)-1);
ingCmax_Zoo = zeros(1, length(tspan)-1);
AEC = zeros(1, length(tspan)-1);
AEN = zeros(1, length(tspan)-1);
GGEC = zeros(1, length(tspan)-1);
GGEN = zeros(1, length(tspan)-1);
palat_Phy1 = zeros(1, length(tspan)-1);
palat_Phy2 = zeros(1, length(tspan)-1);
Enc_Phy1 = zeros(1, length(tspan)-1);
Enc_Phy2 = zeros(1, length(tspan)-1);
ingPhy1C = zeros(1, length(tspan)-1);
ingPhy2C = zeros(1, length(tspan)-1);
PR_Phy1 = zeros(1, length(tspan)-1);
PR_Phy2 = zeros(1, length(tspan)-1);
CRC_Phy1 = zeros(1, length(tspan)-1);
CRC_Phy2 = zeros(1, length(tspan)-1);
NCPhy_Zoo = zeros(1, length(tspan)-1);
Stoich_con = zeros(1, length(tspan)-1);

# Initial conditions
Am = 280;                  # Ammonium-N (ugN L-1)
C_Phy1 = 12;               # Phytoplankton1-C (ugC L-1)
N_Phy1 = C_Phy1 * 0.07; # Phytoplankton1-N (ugN L-1)
C_Phy2 = 12;               # Phytoplankton2-C (ugC L-1)
N_Phy2 = C_Phy2 * 0.05; # Phytoplankton2-N (ugN L-1)
C_Zoo = 5;                 # Zooplankton C-biomass (ugC L-1)
VON = 0;                   # Faecal material-N (ugC L-1)
VOC = 0;                   # Faecal material-C (ugN L-1)
sysN    = Am + N_Phy1 + N_Phy2 + C_Zoo * 0.2 + VON; # System N (ugN L-1)
# Initial conditions arrayfun
x0 = [Am, N_Phy1, C_Phy1, N_Phy2, C_Phy2, C_Zoo, VON, VOC, sysN];

# Simulate
y = solver(@func_allometric_quota_npz, tspan, stepsize, x0);

### Plot the results
h = figure;

subplot(2, 2, 1)
plot(tspan, y(:, 2), 'r', tspan, y(:, 4), 'g', tspan, y(:, 6) * 0.2, 'b', tspan,
y(:, 7), 'k', tspan, y(:, 1), 'm',  tspan, y(:, 9), 'c');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
ylabel('\mugN L^{-1}', 'FontSize', 12);
hleg = legend({'N\_Phy1'; 'N\_Phy2'; 'N\_Zoo'; 'VON'; 'Am'; 'sysN'}, 'location',
'eastoutside');
set(hleg, 'FontSize', 8);
```

```
subplot(2, 2, 2)
plot(tspan, y(:, 3), 'r', tspan, y(:, 5), 'g', tspan, y(:, 6), 'b', tspan, y(:,
8), 'k');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
ylabel('\mugC L^{-1}', 'FontSize', 12);
hleg = legend('C\_Phy1', 'C\_Phy2', 'C\_Zoo', 'VOC', 'location', 'eastoutside');
set(hleg, 'FontSize', 8);

subplot(2, 2, 3)
plot(tspan(2:end), u_Phy1, 'r', tspan(2:end), u_Phy2, 'g', tspan(2:end), u_Zoo,
'b');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
hleg = legend('u\_Phy1', 'u\_Phy2', 'u\_Zoo', 'location', 'eastoutside');
set(hleg, 'FontSize', 8);

subplot(2, 2, 4)
plot(tspan(2:end), NC_Phy1, 'r', tspan(2:end), NC_Phy2, 'g', tspan(2:end),
repmat(0.2, 1, length(tspan)-1), 'b', tspan(2:end), (y(2:end, 7) ./ y(2:end,
8)), 'k');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
ylabel('N:C', 'FontSize', 12);
hleg = legend('NC\_Phy1', 'NC\_Phy2', 'NC\_Zoo', 'NC\_VO', 'location',
'eastoutside');
set(hleg, 'FontSize', 8);

print(h, 'Chapter-17-Allometric-Quota-NPZ_1.png', '-dpng', '-color');

h2 = figure;

subplot(2, 2, 1)
plot(y(2:end, 1), u_Phy1, 'r');
set(gca,'FontSize',12);
xlabel('Am', 'FontSize', 12);
ylabel('u\_Phy1', 'FontSize', 12);

subplot(2, 2, 2)
plot(y(2:end, 1), uN_Phy1, 'r');
set(gca,'FontSize',12);
xlabel('Am', 'FontSize', 12);
ylabel('uN\_Phy1', 'FontSize', 12);

subplot(2, 2, 3);
plot(tspan(2:end), AEqual, 'r', tspan(2:end), repmat(AEmin, 1, length(tspan)-1),
'g', tspan(2:end), repmat(AEmax, 1, length(tspan)-1), 'b', tspan(2:end),
AEC_Zoo, 'k');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
hleg = legend('AEqual', 'AEmin', 'AEmax', 'AEC\_Zoo', 'location',
'eastoutside');
set(hleg, 'FontSize', 8);

subplot(2, 2, 4);
plot(tspan(2:end), ingCmax_Zoo, 'r', tspan(2:end), ingC_Zoo, 'g');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
hleg = legend('ingCmax\_Zoo', 'ingC\_Zoo', 'location', 'eastoutside');
set(hleg, 'FontSize', 8);

print(h2, 'Chapter-17-Allometric-Quota-NPZ_2.png', '-dpng', '-color');
```

```
h3 = figure;

subplot(4, 2, 1);
plot(tspan(2:end), AEC, 'r', tspan(2:end), AEN, 'g');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
hleg = legend('AEC', 'AEN', 'location', 'eastoutside');
set(hleg, 'FontSize', 8);

subplot(4, 2, 2);
plot(tspan(2:end), GGEC, 'r', tspan(2:end), GGEN, 'g');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
ylim([0 1]);
hleg = legend('GGEC', 'GGEN', 'location', 'eastoutside');
set(hleg, 'FontSize', 8);

subplot(4, 2, 3);
plot(tspan(2:end), Enc_Phy1, 'r', tspan(2:end), Enc_Phy2, 'g');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
hleg = legend('Enc\_Phy1', 'Enc\_Phy2', 'location', 'eastoutside');
set(hleg, 'FontSize', 8);

subplot(4, 2, 4);
plot(tspan(2:end), ingPhy1C, 'r', tspan(2:end), ingPhy2C, 'g');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
hleg = legend('ingPhy1C', 'ingPhy2C', 'location', 'eastoutside');
set(hleg, 'FontSize', 8);

subplot(4, 2, 5);
plot(tspan(2:end), PR_Phy1, 'r', tspan(2:end), PR_Phy2, 'g');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
hleg = legend('PR\_Phy1', 'PR\_Phy2', 'location', 'eastoutside');
set(hleg, 'FontSize', 8);

subplot(4, 2, 6);
plot(tspan(2:end), palat_Phy1, 'r', tspan(2:end), palat_Phy2, 'g');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
hleg = legend('palat\_Phy1', 'palat\_Phy2', 'location', 'eastoutside');
set(hleg, 'FontSize', 8);

subplot(4, 2, 7);
plot(tspan(2:end), CRC_Phy1, 'r', tspan(2:end), CRC_Phy2, 'g');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
hleg = legend('CRC\_Phy1', 'CRC\_Phy2', 'location', 'eastoutside');
set(hleg, 'FontSize', 8);

subplot(4, 2, 8);
plot(tspan(2:end), XSC, 'r');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
ylabel('XSC', 'FontSize', 12);

set(gcf, 'PaperPosition', [0.25000 2.50000 9.00000 12.00000]);
print(h3, 'Chapter-17-Allometric-Quota-NPZ_3.png', '-dpng', '-color');

h4 = figure;
```

```
subplot(2, 2, 1);
plot(NC_Phy1, ingCmax_Zoo, 'r');
set(gca,'FontSize',12);
xlabel('NC\_Phy1', 'FontSize', 12);
ylabel('ingCmax\_Zoo', 'FontSize', 12);

subplot(2, 2, 2);
plot(tspan(2:end), NCPhy_Zoo, 'r', tspan(2:end), Stoich_con, 'g');
set(gca,'FontSize',12);
ylabel('Time (d)');
hleg = legend('NCPhy\_Zoo', 'Stoich\_con', 'location', 'eastoutside');
set(hleg, 'FontSize', 8);

subplot(2, 2, 3);
plot(NCu_Phy1, palat_Phy1, 'r');
set(gca,'FontSize',12);
xlabel('palat\_Phy1', 'FontSize', 12);
ylabel('NCu\_Phy1', 'FontSize', 12);

subplot(2, 2, 4);
plot(NCu_Phy1, palat_Phy2, 'r');
set(gca,'FontSize',12);
xlabel('palat\_Phy2', 'FontSize', 12);
ylabel('NCu\_Phy1', 'FontSize', 12);

print(h4, 'Chapter-17-Allometric-Quota-NPZ_4.png', '-dpng', '-color');

h5 = figure;

subplot(3, 2, 1);
plot(tspan(2:end), AEquan, 'r', tspan(2:end), palat_Phy1, 'g');
set(gca,'FontSize',12);
ylim([0 2]);
xlabel('Time (d)', 'FontSize', 12);
ylabel('Index', 'FontSize', 12);
hleg = legend('AEquan', 'palat\_Phy1', 'location', 'eastoutside');
set(hleg, 'FontSize', 8);

subplot(3, 2, 2);
plot(tspan(2:end), u_Phy1, 'r', tspan(2:end), uN_Phy1, 'g', tspan(2:end),
NCu_Phy1, 'b');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
hleg = legend('u\_Phy1', 'uN\_Phy1', 'NCu\_Phy1', 'location', 'eastoutside');
set(hleg, 'FontSize', 8);

subplot(3, 2, 3);
plot(y(:, 3), y(:, 6), 'r');
set(gca,'FontSize',12);
xlabel('C_Phy1', 'FontSize', 12);
ylabel('C\_Zoo', 'FontSize', 12);

subplot(3, 2, 4);
plot(NC_Phy1, ingC_Zoo, 'r');
set(gca,'FontSize',12);
xlabel('NC\_Phy1', 'FontSize', 12);
ylabel('ingC\_Zoo', 'FontSize', 12);

subplot(3, 2, 5);
plot(tspan(2:end), AEquan, 'r');
set(gca,'FontSize',12);
xlabel('Time (d)', 'FontSize', 12);
```

```
ylabel('Aequan', 'FontSize', 12);

print(h5, 'Chapter-17-Allometric-Quota-NPZ_5.png', '-dpng', '-color');
```

17.3.2 func_allometric_quota_npz.m

```
## GNU Octave ports of the models in "Dynamic Ecology - an introduction to
## the art of simulating trophic dynamics" by Flynn, K. (2018).
## Copyright (C) 2020 Ekin Akoglu and Kevin J. Flynn

## This program is free software: you can redistribute it and/or modify
## it under the terms of the GNU General Public License as published by
## the Free Software Foundation, either version 3 of the License, or
## (at your option) any later version.

## This program is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
## GNU General Public License for more details.

## You should have received a copy of the GNU General Public License
## along with this program. If not, see https://www.gnu.org/licenses/.

function xdot = func_allometric_quota_npz(t, x)

global NC_Phy1
global NC_Phy2
global u_Phy1
global u_Phy2
global u_Zoo
global uN_Phy1
global NCu_Phy1
global NCu_Phy2
global AEmin
global AEmax
global AEqual
global AEquan
global AEC_Zoo
global ingC_Zoo
global ingCmax_Zoo
global AEC
global AEN
global GGEC
global GGEN
global palat_Phy1
global palat_Phy2
global Enc_Phy1
global Enc_Phy2
global ingPhy1C
global ingPhy2C
global PR_Phy1
global PR_Phy2
global CRC_Phy1
global CRC_Phy2
global NCPhy_Zoo
global Stoich_con
global XSC

# Dilution
dil = 0.05;        # Dilution rate (d-1)

# Phytoplankton 1 parameters
ktAM_Phy1 = 14;    # Half saturation constant for ammonium transport (ugN L-1)
umax_Phy1 = 0.5;   # Maximum C-specific growth rate (gC (gC)-1 d-1)
NCmin_Phy1 = 0.07;# Minimum NC_Phy (gN (gC)-1)
NCmax_Phy1 = 0.15;# Maximum NC_Phy (gN (gC)-1)
kQN_Phy1 = 10;     # KQ for N-quota (dl)
```

```
# Phytoplankton 2 parameters
ktAM_Phy2 = 70;    # Half saturation constant for ammonium transport (ugN L-1)
umax_Phy2 = 0.8;   # Maximum C-specific growth rate (gC (gC)-1 d-1)
NCmin_Phy2 = 0.05;# Minimum NC_Phy (gN (gC)-1)
NCmax_Phy2 = 0.15;# Maximum NC_Phy (gN (gC)-1)
kQN_Phy2 = 10;     # KQ for N-quota (dl)

# Zooplankton parameters
AEmax = 0.6;       # Maximum AE for N (dl)
AEmin = 0.2;       # Maximum AE for N (dl)
BR_Zoo = 0.1;      # Basal respiration rate as a proportion of umax_Zoo (dl)
kAE = 1.00e+03;    # Constant for control of AE in response to prey quality (dl)
kGTT = 100;        # Curve control for density dependant inefficiency (ugC L-1)
minAE_mult = 1;    # Minimum AEC scalar for density dependant inefficiency (dl)
NC_Zoo = 0.2;      # N:C of zooplankton (gN (gC)-1)
SDA = 0.3;         # Specific dynamic action (dl)
tox_Phy1 = 0;      # Toxicity factor for Phy1; 0 not toxic (dl)
tox_Phy2 = 1.1;    # Toxicity factor for Phy2 (dl)
Optimal_CR = 1;    # Proportion of prey of optimal characteristics captured by
starved Zoo (dl)
umax_Zoo = 0.693; # Zooplankton maximum growth rate (d-1)

# Encounter sub-model variables
a = 0.216;         # Parameter for derivation of C-cell content for protist of a
given volume (dl)
b = 0.939;         # Parameter for derivation of C-cell content for protist of a
given volume (dl)
r_Phy1 = 2.5;      # Radius of Phy1 cell (um)
r_Phy2 = 5;        # Radius of Phy2 cell (um)
r_Zoo = 50;        # Radius of Zoo cell (um)
w = 0;             # Root-mean-squared turbulence (m s-1)

# Prey optimality sub-model variables
relMaxPrey = 0.3; # Maximum prey:pred (dl)
relMinPrey = 0.025; # Minimum prey:pred (dl)
relOpPrey = 0.2;  # Optimal prey:pred (dl)

# Voided materials parameters
NCmax = 0.3;       # Maximum mass ratio of N:C which could be attained in the
organic form (gN (gC)-1)

## Auxiliaries
# C content of Phy1 (pgC cell-1)
Ccell_Phy1 = a * (4 / 3 * pi * (r_Phy1)^3)^b;

# C content of Phy2 (pgC cell-1)
Ccell_Phy2 = a * (4 / 3 * pi * (r_Phy2)^3)^b;

# C content of Zoo (pgC cell-1)
Ccell_Zoo = a * (4 / 3 * pi * (r_Zoo)^3)^b;

# Cell abundance of Phy1 (Phy1 cells m-3)
nos_Phy1 = 10^9 * x(3) / Ccell_Phy1;

# Cell abundance of Phy2 (Phy2 cells m-3)
nos_Phy2 = 10^9 * x(5) / Ccell_Phy2;

# Speed of motility of Phy1 (m s-1)
v_Phy1 = (10^-6) * (38.542 * (r_Phy1 * 2)^0.5424);

# Speed of motility of Phy2 (m s-1)
v_Phy2 = (10^-6) * (38.542 * (r_Phy2 * 2)^0.5424);
```

```
# Speed of motility of Zoo (m s-1)
v_Zoo = (10^-6) * (38.542 * (r_Zoo * 2)^0.5424);

# Encounter rate of a cell of Phy1 by a cell of Zoo (Phy1 Zoo-1 d-1)
Enc_Phy1(t - 1) = (24 * 60 * 60) * pi * (r_Phy1 / 1E6 + r_Zoo / 1E6)^2 *
nos_Phy1 * (v_Phy1^2 + 3 * v_Zoo^2 + 4 * w^2) * ((v_Zoo^2 + w^2)^-0.5) * 3^-1;

# Encounter rate of a cell of Phy2 by a cell of Zoo (Phy2 Zoo-1 d-1)
Enc_Phy2(t - 1) = (24 * 60 * 60) * pi * (r_Phy2 / 1E6 + r_Zoo / 1E6)^2 *
nos_Phy2 * (v_Phy2^2 + 3 * v_Zoo^2 + 4 * w^2) * (v_Zoo^2 + w^2)^-0.5 * 3^-1;

# prey:pred for Phy1 (dl)
rel_Phy1 = r_Phy1 / r_Zoo;

# prey:pred for Phy2 (dl)
rel_Phy2 = r_Phy2 / r_Zoo;

# Prey handling index for Phy1, taking into account the prey:pred relative size
(dl)
if relMaxPrey > rel_Phy1 && rel_Phy1 > relMinPrey
  if rel_Phy1 < relOpPrey
    PR_Phy1(t - 1) = (rel_Phy1 - relMinPrey) / (relOpPrey - relMinPrey);
  else
    PR_Phy1(t - 1) = (relMaxPrey - rel_Phy1) / (relMaxPrey - relOpPrey);
  endif
else
  PR_Phy1(t - 1) = 0;
endif

# Prey handling index for Phy2, taking into account the prey:pred relative size
(dl)
if relMaxPrey > rel_Phy2 && rel_Phy2 > relMinPrey
  if rel_Phy2 < relOpPrey
    PR_Phy2(t - 1) = (rel_Phy2 - relMinPrey) / (relOpPrey - relMinPrey);
  else
    PR_Phy2(t - 1) = (relMaxPrey - rel_Phy2) / (relMaxPrey - relOpPrey);
  endif
else
  PR_Phy2(t - 1) = 0;
endif

# Nutrient exchange (ugN L-1 d-1)
in_out_Am = dil * (280 - x(1));

# Washout of N_Phy1 (ugN L-1 d-1)
outN_Phy1 = x(2) * dil;

# Washout of C_Phy1 (ugC L-1 d-1)
outC_Phy1 = x(3) * dil;

# Washout of N_Phy2 (ugN L-1 d-1)
outN_Phy2 = x(4) * dil;

# Washout of C_Phy2 (ugC L-1 d-1)
outC_Phy2 = x(5) * dil;

# Phytoplankton 1 N:C quota (gN (gC)-1)
NC_Phy1(t - 1) = x(2) / x(3);

# Phytoplankton 2 N:C quota (gN (gC)-1)
NC_Phy2(t - 1) = x(4) / x(5);
```

```
# Quotient for N status (dl)
NCu_Phy1(t - 1) = ((1 + kQN_Phy1) * (NC_Phy1(t - 1) - NCmin_Phy1)) / ((NC_Phy1(t
- 1) - NCmin_Phy1) + kQN_Phy1 * (NCmax_Phy1 - NCmin_Phy1));

# Quotient for N status (dl)
NCu_Phy2(t - 1) = ((1 + kQN_Phy2) * (NC_Phy2(t - 1) - NCmin_Phy2)) / ((NC_Phy2(t
- 1) - NCmin_Phy2) + kQN_Phy2 * (NCmax_Phy2 - NCmin_Phy2));

# C-specific growth rate controlled by N:C quota for Phy1 (gC (gC)-1 d-1)
u_Phy1(t - 1) = umax_Phy1 * NCu_Phy1(t - 1);

# C-specific growth rate controlled by N:C quota for Phy2 (gC (gC)-1 d-1)
u_Phy2(t - 1) = umax_Phy2 * NCu_Phy2(t - 1);

# Maximum C-specific N transport rate for Phy1 (gN (gC)-1 d-1)
TNmax_Phy1 = umax_Phy1 * NCmax_Phy1;

# Maximum C-specific N transport rate for Phy2 (gN (gC)-1 d-1)
TNmax_Phy2 = umax_Phy2 * NCmax_Phy2;

# Phytoplankton C-specific N transport rate for Phy1 (gN (gC)-1 d-1)
NCt_Phy1 = TNmax_Phy1 * x(1) / (x(1) + ktAM_Phy1);

# Phytoplankton C-specific N transport rate fpr Phy 2(gN (gC)-1 d-1)
NCt_Phy2 = TNmax_Phy2 * x(1) / (x(1) + ktAM_Phy2);

# N-specific growth rate for Phy1 (gN (gN)-1 d-1)
uN_Phy1(t - 1) = NCt_Phy1 / NC_Phy1(t -1);

# N-specific growth rate for Phy2 (gN (gN)-1 d-1)
uN_Phy2(t - 1) = NCt_Phy2 / NC_Phy2(t -1);

# Phytoplankton-1 population uptake of ammonium-N (ugN L-1 d-1)
Nup_Phy1 = x(3) * NCt_Phy1;

# Phytoplankton-2 population uptake of ammonium-N (ugN L-1 d-1)
Nup_Phy2 = x(5) * NCt_Phy2;

# Total consumption of ammonium (ugN L-1 d-1)
Am_up = Nup_Phy1 + Nup_Phy2;

# Growth rate in phytoplankton1-C (ugC L-1 d-1)
groC_Phy1 = x(3) * u_Phy1(t - 1);

# Growth rate in phytoplankton2-C (ugC L-1 d-1)
groC_Phy2 = x(5) * u_Phy2(t - 1);

# Palatability index (0 not palatable) (dl)
palat_Phy1(t - 1) = (NCu_Phy1(t - 1) + 1.0e-6)^tox_Phy1;

# Palatability index (0 not palatable) (dl)
palat_Phy2(t - 1) = (NCu_Phy2(t - 1) + 1.0e-6)^tox_Phy2;

# Potential capture of Phy1 taking into account all factors (Phy1 Zoo-1 d-1)
CR_Phy1 = Enc_Phy1(t - 1) * PR_Phy1(t - 1) * palat_Phy1(t - 1) * Optimal_CR;

# Potential capture of Phy2 taking into account all factors (Phy2 Zoo-1 d-1)
CR_Phy2 = Enc_Phy2(t - 1) * PR_Phy2(t - 1) * palat_Phy2(t - 1) * Optimal_CR;

# Potential C-specific ingestion Phy1 (gC (gC)-1 d-1)
CRC_Phy1(t - 1) = CR_Phy1 * Ccell_Phy1 / Ccell_Zoo;

# Potential C-specific ingestion Phy2 (gC (gC)-1 d-1)
```

```
CRC_Phy2(t - 1) = CR_Phy2 * Ccell_Phy2 / Ccell_Zoo;

# Sum of potential prey capture rate (gC (gC)-1 d-1)
SCRC = CRC_Phy1(t - 1) + CRC_Phy2(t - 1);

# N:C of incoming food (gN (gC)-1)
ingNC(t - 1) = (CRC_Phy1(t - 1) * NC_Phy1(t - 1) + CRC_Phy2(t - 1) * NC_Phy2(t -
1)) / SCRC;

# Ratio of NC in prey compared to predator (dl)
NCPhy_Zoo(t - 1) = ingNC(t -1) / NC_Zoo;

# Selection of release of N related to difference in food to consumer N:C (dl)
Stoich_con(t - 1) = min(NCPhy_Zoo(t - 1), 1);

# AEC scalar for density dependant inefficiency (dl)
AEquan(t - 1) = (1 - minAE_mult) * (1 - (x(3) + x(5)) / (x(3) + x(5) + kGTT)) +
minAE_mult;

# Efficiency parameter for assimilation (dl)
AEqual(t - 1)  = AEmin + (AEmax - AEmin) * Stoich_con(t - 1) / (Stoich_con(t -
1) + kAE) * (1 + kAE);

# Operational AE for C (dl)
AEC_Zoo(t - 1) = Stoich_con(t - 1) * AEqual(t - 1)  * AEquan(t - 1);

# Zooplankton basal respiration rate (d-1)
BR = umax_Zoo * BR_Zoo;

# Maximum ingestion rate by zooplankton (gC (gC)-1 d-1)
ingCmax_Zoo(t -1) = (umax_Zoo * (1 + SDA) + BR) / AEC_Zoo(t - 1);

# Satiation control constant (gC (gC)-1 d-1)
KI = ingCmax_Zoo(t -1) / 4;

# Ingestion rate of prey into zooplankton (gC (gC)-1 d-1)
ingC_Zoo(t - 1) = min(ingCmax_Zoo(t - 1) * SCRC / (SCRC + KI), SCRC);

# Ingestion rate of Phy1 by Zoo (gC (gC)-1 d-1)
ingPhy1C(t - 1) = ingC_Zoo(t - 1) * CRC_Phy1(t - 1) / SCRC;

# Ingestion rate of Phy2 by Zoo (gC (gC)-1 d-1)
ingPhy2C(t - 1) = ingC_Zoo(t - 1) * CRC_Phy2(t - 1) / SCRC;

# C available for support of respiration (gC (gC)-1 d-1)
if Stoich_con(t - 1) < 1
  XSC(t - 1) = AEqual(t - 1)  * ingC_Zoo(t - 1) * (1 - Stoich_con(t - 1));
else
  XSC(t - 1) = 0;
endif

# Basal respiration that is met bu respiration of excess C in diet (gC (gC)-1 d-
1)
if BR <= XSC(t - 1)
  BRi = BR;
else
  BRi = XSC(t - 1);
endif

# Balance of basal respiration that cannot be met from dietary excees C (gC
(gC)-1 d-1)
BRb = BR - BRi;
```

```
# Grazing upon phytoplankton-1 population (ugC L-1 d-1)
grazC_Phy1 = x(6) * ingPhy1C(t - 1);

# Grazing upon phytoplankton-2 population (ugC L-1 d-1)
grazC_Phy2 = x(6) * ingPhy2C(t - 1);

# Grazing upon phytoplankton-1 population in terms of N (ugN L-1 d-1)
grazN_Phy1 = x(6) * ingPhy1C(t - 1) * NC_Phy1(t -1);

# Grazing upon phytoplankton-2 population in terms of N (ugN L-1 d-1)
grazN_Phy2 = x(6) * ingPhy2C(t - 1) * NC_Phy2(t -1);

# Assimilation rate into zooplankton (gC (gC)-1 d-1)
assC_Zoo = AEC_Zoo(t - 1)  * ingC_Zoo(t -1);

# Assimilation of C into zooplankton population biomass (ugC L-1)
assC = x(6) * assC_Zoo;

# Zooplankton respiration rate (gC (gC)-1 d-1)
resC_Zoo = BRb + assC_Zoo * SDA;

# Zooplankton population respiration (ugC L-1 d-1)
respC = x(6) * resC_Zoo;

# Zooplankton growth rate (gC (gC)-1 d-1)
u_Zoo(t - 1) = assC_Zoo - resC_Zoo;

# Amount of N initially in the organic form to be voided to maintain constant
predator N:C (gN (gC)-1 d-1)
XSassN = ingC_Zoo(t - 1) * ingNC(t - 1) - assC_Zoo * NC_Zoo;

# AE in terms of C (dl)
AEC(t - 1) = assC_Zoo / ingC_Zoo(t - 1);

# AR in terms of N (dl)
AEN(t - 1) = (assC_Zoo * NC_Zoo) / (ingC_Zoo(t - 1) * ingNC(t - 1));

# Voiding of C by zooplankton (gC (gC)-1 d-1)
voidC_Zoo = ingC_Zoo(t - 1) - assC_Zoo - BRi;

# Population rate of C voiding (ugC L-1 d-1)
voidC = x(6) * voidC_Zoo;

# Voiding of N by zooplankton (gN (gC)-1 d-1)
if (XSassN / voidC_Zoo) > NCmax
  voidN_Zoo = voidC_Zoo * NCmax;
else
  voidN_Zoo = XSassN;
endif

# Population rate of N voiding (ugN L-1 d-1)
voidN = x(6) * voidN_Zoo;

# Zooplankton ammonium regeneration (gN (gC)-1 d-1)
DINr = resC_Zoo * NC_Zoo + XSassN - voidN_Zoo;

# GGE in terms of C (dl)
GGEC(t - 1) = (ingC_Zoo(t -1) - voidC_Zoo - resC_Zoo - BRi) / ingC_Zoo(t -1);

# GGE in terms of N (dl)
GGEN(t - 1) = (ingC_Zoo(t -1) * ingNC(t -1) - voidN_Zoo - DINr) / (ingC_Zoo(t -
1) * ingNC(t -1));
```

```
# Zooplankton population regeneration of ammonium (ugN L-1 d-1)
reg_Am = x(6) * DINr;

# Washout of voided C (ugC L-1 d-1)
out_VOC = x(8) * dil;

# Washout of voided N (ugN L-1 d-1)
out_VON = x(7) * dil;

# Washout of zooplankton biomass (ugC L-1 d-1)
outC_Zoo = x(6) * dil;

## State equations
# Ammonium
xdot(1, 1) = in_out_Am + reg_Am - Am_up;

# Phytoplankton1-N
xdot(1, 2) = Nup_Phy1 - grazN_Phy1 - outN_Phy1;

# Phytoplankton1-C
xdot(1, 3) = groC_Phy1 - grazC_Phy1 - outC_Phy1;

# Phytoplankton2-N
xdot(1, 4) = Nup_Phy2 - grazN_Phy2 - outN_Phy2;

# Phytoplankton2-C
xdot(1, 5) = groC_Phy2 - grazC_Phy2 - outC_Phy2;

# Zooplankton-C
xdot(1, 6) = assC - respC - outC_Zoo;

# VON
xdot(1, 7) = voidN - out_VON;

# VOC
xdot(1, 8) = voidC - out_VOC;

# System
xdot(1, 9) = xdot(1, 1) + xdot(1, 2) + xdot(1, 4) + xdot(1, 6) * NC_Zoo +
xdot(1, 7);

endfunction
```

18. Concluding comments

We hope that the models described in this work, together with the descriptions in *Dynamic Ecology* (Flynn 2018) have helped to provide an insight into the dynamics under which ecosystems function. Not only will you have learnt from playing with the models, but you will at the least have also come away with an appreciation of the need for particular types of data to inform model construction and testing. Organism identifications (be it achieved by traditional or molecular biological means) and estimates of abundance are simply insufficient to guide an understanding of ecology.

As you will have seen, the subject of the construction and deployment of dynamic (simulation) models is at once intriguing, often raising more questions than answers, and also troubling. Troubling in that, given the multitude of organisms growing on Earth (in this context in the oceans that cover 2/3^{rds} of the planet), and the complexity of a model describing just a few organism types in a simple physical system, we have to ask how we can ever usefully simulate large scale ecological processes.

In truth of course we cannot extent this level of detail to whole ecosystems. However, ultimately, we do indeed need to understand the fluxes of materials between the abiotic and biotic systems. And that requires ecosystem scientists at all levels to better appreciate the importance of feedback processes etc., and how we need to distil the massive variety of life identified by molecular biology down to a few functional type descriptions.

Making, what many may view as, simple models like those described in this work is an important step along that journey. In future editions of this and *Dynamic Ecology* we will explore additional facets of the systems to further aid this important work, to enhance the models that, at the grandest level, form the basis of global models used to guide global-scale political discussions that affect us all. Please see chapter 18 in *Dynamic Ecology* for further commentary of these matters.

19. REFERENCES

Eaton JW, Bateman D, Hauberg S, Wehbring R (2020). GNU Octave version 5.2.0 manual: a high-level interactive language for numerical computations. https://www.gnu.org/software/octave/doc/v5.2.0/.

Fasham MJR, Ducklow HW, McKelvie SM (1990). A nitrogen-based model of plankton dynamics in the oceanic mixed layer. *Journal of Marine Research* 48; 591–639.

Flynn KJ (2018). Dynamic Ecology - an introduction to the art of simulating trophic dynamics. Swansea University, ISBN: 978-0-9567462-9-0

Flynn KJ, Stoecker DK, Mitra A, Raven JA, Glibert PM, Hansen PJ, Granéli E, Burkholder JM (2013). Misuse of the phytoplankton-zooplankton dichotomy: the need to assign organisms as mixotrophs within plankton functional types. *Journal of Plankton Research* 35; 3-11.

Mitra A, Flynn KJ, Burkholder JM, Berge T, Calbet A, Raven JA, Granéli E, Hansen PJ, Stoecker DK, Thingstad F, Tillmann U, Våge S, Wilken S, Glibert PM, Zubkov MV (2014). The role of mixotrophic protists in the biological carbon pump. *Biogeosciences* 11; 995-1005 DOI: 10.5194/bg-11-995-2014.

Mitra A, Flynn KJ, Tillmann U, Raven JA, Caron D, Stoecker DK, Not F, Hansen PJ, Hallegraeff G, Sanders R, Wilken S, McManus G, Johnson M, Pitta P, Vågen S, Berge T, Calbet A, Thingstad F, Jeong HJ, Burkholder J-A, Glibert PM, Granéli E, Lundgren V (2016). Defining planktonic protist functional groups on mechanisms for energy and nutrient acquisition; incorporation of diverse mixotrophic strategies. *Protist* 167; 106–120.

Pianosi F, Sarrazin F, Wagener T (2015). A Matlab toolbox for global sensitivity analysis. *Environmental Modelling & Software* 70; 80-85.

Sterner RW, Elser JJ (2002). *Ecological Stoichiometry: the Biology of Elements from Molecules to the Biosphere.* Princeton University Press, Princeton, NJ.

Stow CA, Roessler C, Borsuk ME, Bowen JD, Reckhow KH (2003). Comparison of estuarine water quality models for total maximum daily load development in Neuse River Estuary. *Journal of Water Resources Planning and Management* 129; 307-314.