Tesis Documentation

Release 1.0

Carlos M.

March 18, 2015

CONTENTS

Contents:

CHAPTER

ONE

INTRODUCTION

This file contains all the documentation for the code written for my undergraduate thesis, which explores the relationship that exist between predator body mass and resource-consumer body size ratios over the assembly mechanism of the *intraguild predation* module. More formally I explore mathematical models of the form :

$$\begin{aligned}
\frac{dR}{dt} &= F(R) - G_c(R)C - G_p(R,C)P \\
\frac{dC}{dt} &= \epsilon_1 G_c(R)C - H_p(R,C)P - q_1 C \\
\frac{dP}{dt} &= P(\epsilon_2 G_p(R,C) + \epsilon_3 H_p(R,C) - q_2)
\end{aligned}$$

And find invasibility conditions for each of the subcases assuming point attractors in the *recipient* community.

More information will come ...

CHAPTER

TWO

ANALISIS

2.1 Specifications

- This set of functions and Modules make possible the computation of the invasibility boundaries and the associated zones.
- It also has code to explore a range of different parameter combinations and save the results to files.

2.2 Contents

2.2.1 AuxiliarClases module

class AuxiliarClases.**MyFloat**
Bases: float

differs from float in just the printing method

class AuxiliarClases.**MyInnerTuple**
Bases: tuple

class AuxiliarClases.**MyTuple**
Bases: tuple

differs from tuple in just the printing method

2.2.2 Auxiliaryfunctions module

Auxiliaryfunctions.**ConstructArray**(*List_Arrays*)
Reads in a list of arrays, calculates the one with the largest number of elements an storage it in the value Max_length completes all the other arrays to that number by filling the missing elements with NaN, after that appends all of them in the list Array_handler, convert it to a numpy array and transpose it. @param List_Arrays : a List of lists

Auxiliaryfunctions.**Convert**(*X*, *Y*)
Fuse the elements of the sublists of X and Y in a tuple and append it to the array 'new_array' record the number of elements in each sublist in X(Y) in the list dist @param X a List of lists contanining floating point elements @param Y a List of lists contanining floating point elements

Auxiliaryfunctions.**F**(*D*, *F1*, *D1*, *D2*, *EfD*, **args*)

Auxiliaryfunctions.**FindIntersection**(*BoundaryPoints1*, *BoundaryPoints2*)
find the intersection of two sets of intervals A and B whose limit points are stored in BoundaryPoints1 and BoundaryPoints2 respectively. It is based on the fact that :

$$(\cup_{I\in A} I) \cap (\cup_{J\in B} J) = \cup_{J\in B}(\cup_{I\in A}(J \cap I))$$

Auxiliaryfunctions.**FindUnion**(*BP1*, *BP2*)

Auxiliaryfunctions.**FormatStart**(*Points*)
Format the list of Points to account that [(0,)] elements could have been added to by the first iteration, that is the first interval(s) could have a null intersection. But the overall intersection is not, in that cases it deletes the first element.

Auxiliaryfunctions.**FormatUnionSet**(*L*)
Glue together the intervals of the sorted List L,sorted in increasing order, if its limits coincide . Each interval is represented by a Tuple . It uses a recurse algorithm in which we start at the leftmost position L[0], proceed to the right gluing together all the intervals to which it coincide ,i.e. L[0][1] = L[i][0], it keeps updating L[0] until it don't found more, and repeat the same procees starting from that position. It terminates when there are no more intervals to add.

Auxiliaryfunctions.**FormatWidths**(*Keys*, *WidthDict*)
Format Widths using a custom tupple class to get 20 digits printing

Auxiliaryfunctions.**FormatZones**(*Header*, *SZBounds*)
Convert each element of the SZBounds into the format used for the InvFunctions dict

Auxiliaryfunctions.**GetComplement**(*Boundaries*, *Min*, *Max*)
From a set of Boundary points of a set, return the Boundary Points of its complement. since each one dimensional slide will be a set of intervals it pieces together the complements of each contiguous pair of intervals in each list $((a, b) \cup (c, d))^c = [b, c]$

Auxiliaryfunctions.**GetIntersection**(*Bound1*, *Bound2*)
Get the boundary of the intersection of two sets A and B, by having as input the set of boundary points at each $x \in xRange$

Auxiliaryfunctions.**GetPositiveBoundaries**(*PosPoints*, *xRange*)
Input a range of Points and for each x in Points a set of tuples containing the points of the boundary for the set in which the function has positive values. Output a dictionary in the format of the invasibility boundary dicts

Auxiliaryfunctions.**GetPositiveRegions**(*Bounds*, *xRange*, *yRange*)

Auxiliaryfunctions.**GetUnion**(*Bound1*, *Bound2*)
get the boundary points of the union of two sets A and B, having as input the set of boundary points at each $x \in xRange$

Auxiliaryfunctions.**Intersection**(*bp1*, *bp2*)
For intervals I and J whose boundary points are given by bp1 and bp2 respectively , returns the boundary points for $I \cap J$

Auxiliaryfunctions.**IntervalInt**(*bpL*, *bpH*)
Find the boundary points of $(a, b) \cap (c, d)$ for which $a \leq c$

Auxiliaryfunctions.**SumBound**(*B*)
B is a list of tuples [(a1,a2),...], that may contain null elements specified by (0,), get the sum of the difference between the elements of each tuple.

Auxiliaryfunctions.**addPoints**(*IntPoints*, *NewBPoints*)
Add NewBPoints **N** to the List IntPoints **I**, only add $x \in N$ if $|x[0]| > 1 \vee |\mathbf{I}| = 0$

Auxiliaryfunctions.**findIntPoints**(*BoundaryPoints1*, *BoundaryPoint2*)
returns the boundary points of $(\cup_{I \in A}(J \cap I))$ for a particular interval J whose boundary points are given by BoundaryPoint2 and the boundary points of the intervals I are given by the list BoundaryPoints1

Auxiliaryfunctions.**getCompPoints**(*boundPoints*, *Min*, *Max*)

Auxiliaryfunctions.**getWidth**(*Boundary*, **args*)
Given the Boundary points of a set D at each location x* in xRange, get the width of the set at the Intersection of (x*,y) and D, it is assumed that this intersection is a Finite Union of Intervals.

Auxiliaryfunctions.**inList**(*yPoint*, *ListofTuples*)
Returns True if $y \in T, T \in LoT$, where *T* is a tuple and *LoT* is a list of tuples

2.2.3 Baseclass module

class Baseclass.**BSR**(*params*, *mode*, *xLims*, *ksim=True*)
Bases: object

Class that stores the basic information for the analysis such as: * The mode of the ODE system(RM or LV) * The approach(Top down or bottom up) * Generate the dictionary with all the functions and the x and y ranges that will be explored. * ksim is a boolean variable that set the assumption about similar or not body size ratios across trophic levels. * In the case of the Active-Grazing-Grazing strategy, it exists a singular point in the x axis when ksim is false, in this case we set a finer xrange in points near to it, this is stored in the xFocus_sep variable.

getParams()

getandSetxFocus(*nPoints*, *dist*)
nPoints : the number of points in which the interval [x0-dist,x0] is going to be divided, x0 is where the singularity occurres (if it exists)

getfDict()
return the dictionary of functions

getmode()
return the mode of the analysis, possible answers at the moment are LV and RM

getxLims()
return the boundary points of the xRange list(assumed as an Interval)

setInvFunctions()

setfDict()
Constructs the dictionary containing all the elementary functions used in the analysis Separates two different approachs: if bottom is true m_C and m_P are expressed in terms of m_R, else m_C and m_R are expressed in terms of m_P, in reality the scenarios will give similar pictures, we are just rotating the plane if we interchange them

setxFocus(*focusLims*)

setxFocusSep(*Focus_sep*)

setxRange()
Sets the range of X points which are going to be explored

setyRange()

2.2.4 ExploreParams module

ExploreParams.**AddToCombinations**(*param*, *ParamsToExplore*, *AuxiliarParams*, *SpecialParams*, *Combinations*)

From a base set List of Combinations, update it using the values stored in ParamsToExplore[param], AuxiliarParams[param]

ExploreParams.**AddValtoComb**(*Comb*, *val*, *param*, *AuxiliarParams*, *SpecialParams*)

Find all the associate parameter combinations given a particular value val of the parameter 'param'. This is donde in a recursive way, the depth of the recursion is finite since the number of keys for AuxiliarParams is finite. If param is a SpecialParam, it adds the low level representation of it to Comb. Finally it updates each of the combinations found with Comb and returns it.

ExploreParams.**ConstructDir**(*ParamDict*, *mode*, *ParamsDirCoder*, *Type*)

ExploreParams.**CreateDirCoder**(*ParamsToExplore*, *dimDict*)

ParamsToExplore is a dictionary with parameter names as keys and a list of values as values dimDict

ExploreParams.**CreateTxtCoder**(*ParamsDirCoder*, *direction*)

ExploreParams.**EvaluateParams**(*paramdict*, *combination*, *mode*, *xlims*, *HeaderInv*, *ParamsDirCoder*, *Direction*, *ksim*, *mass*)

For a given dict of parameters: * calculate the invasibility boundaries, the Equilibrium, eigenvalues ,MTP, Zones and width of the zones for (x,y) values between $xRange \times [-10, 5]$ * Write the results to csv files

ExploreParams.**ExploreParamSpace**(*InitDict, ParamsToExplore, TotalParams, xlims, mode, HeaderInv, AuxiliarParams, SpecialParams, dimDict, initDirection, ksim=True, massVals=[0]*)

Explore the parameter space, by first constructing the total params combinations to explore , creating a Dict that contains a name for each combination and then Evaluating and Saving each of the parameter combinations

ExploreParams.**MakeTotalParamsCombination**(*ParamsToExplore*, *AuxiliarParams*, *SpecialParams*)

- ParamsToExplore is a dictionary with parameter names as keys and a list of values as Values
- AuxiliarParams is a dictionary that stores the information about correlated parameters, that is we set K to a value K0 we must also change the values of the params specified in this dict to the respective values.
- SpecialParams refer to a superClass of Params which are defined as combinations of the elementary params , e.g Comb1:= {'fmC':Grazing,'fmPC':Active,'fmPR':Grazing}

returns a list of Dictionaries containing all possible combinantions for the given params

ExploreParams.**UpdateComb**(*Comb*, *combinations*)

Update each dictiondary in *combinations* with the dictionary *Comb*.

ExploreParams.**setMassTag**(*mass*)

2.2.5 ExploreParamsProgram module

2.2.6 InvasionAnalysis module

class InvasionAnalysis.**InvBoundaries**(*workingData*, *currentMass=0.0*)

Bases: Baseclass.BSR

Class that stores and computes the values for the invasibility boundaries of the distinct scenarios, it receives as initial input the values of the parameters and the invasibility functions

InvBoundary(*Invfunc*, *searchRange*, *xRange*)

Find the inv boundaries(zeros of the invasibility function) using the brentq method from the SciPy package,

input arguments, the invasibility function, the limits for the interval to look for zeros(depending on the x values)

UpdateMass(*newMass*)

WriteWidths(*Direction*)
Write the Width dictionary to a file specified in Direction.

WriteZonesBounds(*Direction*)
Write the SZBounds into a file whose pointer is specified in Direction.

findWidths()
Find the widths of each of the Zones

findZones()
From each of the Boundaries, create a 2D array using xRange and yRange and return a 1 0 array , each 1 located at a position in which the point is interior to the Positive Region delimited by the Boundary

findZonesBoundaries()
Find the boundary of the Zones and format them to the same data structure used in the computation of the Invasibility boundaries

getBounds()
return the dictionary containing all the invasibility boundaries

setAndWriteInvBoundaries(*Header*, *Direction*)

- Find the zero boundaries for each of the functions specified in self.InvFunctions
- Write them to a csv file specified in Direction.

Header refers to the first row of the csv file.

setAndWriteWidthsZones(*DirectionW*, *DirectionZB*, *DirectionZones*)

- For each x in xRange and any Invasibility function f calculates the sum of the length of all the intervals which are in the Positive region of f
- Get the Positive Boundaries for all the target zones in the analysis
- Write both of the above results to a file whose direction is specified in the arguments DirectionW and DirectionZ.

setIntersections()
Find the boundary points of the Intersections of the Positive Regions of some invasibility functions, which delineate the zones expressed in the analysis. For example $Z(I_{C4}) := Z(I_{C2}) \cap PR_4$ where PR_4 is the set for which the Invasibility function I_P_s4 is positive. We assume that one dimensional subset of the set are always a union of emph{connected} sets.

setInvBoundaries()
For each of the functions present in the InvFunctions list, computes the invasibility boundaries by means of the Scipy.Brentq numerical method. For each of the invasibility functions it returns a dict object with x and y keys indicating the zeros(size ratio values delimiting the boundaries) for the function at each x (mass), the values of x and y are a list of lists. the total number of sublists denotes the maximum number of zeros found at any location x. if e1e3 - e2 = EfDif <0 it also computes the boundaries for the D function

setPositiveBoundaries()
Creates a dict storying the boundary points of the set in which each of the criterions is satisfied

setUpGuess(*Guess*)

setWidthsAndZones()

•Calculates the Boundaries of the Zones described in the Study, which are intersection of the Positive Regions of a subset of the Invasibility functions

•Calculates the widths of each of the Zones

•Convert them to the format used for the Invasibility functions and save them in the ZBounds dict

•if EfDif<0 , calculates the boundary for the stable coexistence zone.

writeInvasibilityValues(*Header*, *direction*, *delimiter=', '*)
Write data into a csv file specified in direction @param Boundaries a list of 2-elements lists which each of them stores the X and Y coordinates of the invasibility boundaries computed in the analysis @param the list of params used to compute the boundaries which will be used in the footer of the csv file @direction the system direction where the file is going to be stored @Header the first row of the csv file

2.2.7 OutputClasses module

class OutputClasses.**Data**(*data*)
Bases: object

Class that stores a dataset in the format of a n x n matrix

TransformtoFloats()

getrow(*index*)

ncols()

nrows()

reshape()

setData(*Data*)

class OutputClasses.**OutputInvData**(*data*, *header*, *footer*, *xFocusAndSep*, *distribution*)
Bases: OutputClasses.Data

Creates an abstraction of a csv table, with a header and footer. data stores all the body of the table distribution is a tuple whose length referes to the number of parts which has been put together in the column, and each component of it referees to the number of items in each part

WriteEquilibrium(*direction*, *delimiter*)

WriteInvasibility(*direction*, *delimiter*)
Make use of the csv module to produce an csv file with the data contained in self.data

WriteWidths(*direction*, *delimiter*)

setFooter(*footer*)

setHeader(*header*)

OutputClasses.**constructFooter**(*params*)

2.2.8 RMEquibrium module

RMEquibrium.**AddPoints**(*BreakPoints*, *yList*, *n*, *maxY*, *minY*, *Type*)
From a list of zeros determine the ones that delimit a pass from positive to negative and store them in a list, i.e. find the boundary points of the positive region

RMEquibrium.**AddToBoundaryPoints**(*BreakPoints*, *xPoint*, *xList*, *yList*, *f*, *maxY*, *minY*, *ksim*, *arg1*)

RMEquibrium.**ExtractPoints**(*xRanges*, *BoundaryPoints*)

RMEquibrium.**FindCoexistenceRegion**(*self*)

RMEquibrium.**FindRPositiveRegion**(*self*)

RMEquibrium.**FindpositiveRegion**(*self*, *Pred*)

RMEquibrium.**GetGuessBounds**(*Arr*, *Arr2*, *f*, *defaultUpGuess*, *defaultLowGuess*, *ksim*, *arg1*)
Code for delimiting the boundaries of the positive region of a given function

RMEquibrium.**isNegative**(*f*, *xPoint*, *yPoint*, *ksim*, *arg1*)

RMEquibrium.**toDict**(*xRanges*, *BPoints*)

2.2.9 SimulationDynamics module

class SimulationDynamics.**Dynamics**(*workingData*, *initCondition*, *finalTime*, *separation*, *K_CP*, *K_RC*, *m_P*)
Bases: Baseclass.BSR

DynamicFunction(*X*, *t*)

Simulate()
simulation routine using the Odeint solver from the scipy optimize package , Odeint is a python implementation of the ODEPACK package in FORTRAN which uses a multi-step solver in the non stiff case

makeinitCondition(*case*)
se the init condition , depending on the scenario , in the Invasibility by P to C-R the initial condition is the equilibrium of the latter two in isolation , a similar situation is for the invasibility of C to P-R(labeled scenario 5)

runSimulationSimK(*case*, *massLims*, *lowKLims*, *upKLims*, *initDirection*)

setDynamicFunction()
input the corresponding values for the paramters K_CP, K_RC and m_P and convert the dynamical functions dR, dC and dP in three-argument functions, just depending on the value of the biomass densities R, C and P

setParamVals(*K_CP*, *K_RC*, *m_P*)
Set the values for the three key paramaters of the model , the size ratios and the body mass

setfinalTime(*finalTime*)
set the time until when to stop the simulation

setinitCondition(*initCondition*)
Specifices the given initial condition from which to start the simulation

setseparation(*separation*)

2.2.10 TestProgram module

TestProgram.**Test**(*params*, *ksim*, *xRange*, *DirEq*, *DirStab*, *DirMTP*, *DirInv*, *type='LV'*, *mass=1e-10*)

2.2.11 bounds module

class bounds.**Cubic**(*a*, *b*, *c*, *d*)
Bases: object

Evaluate(*x*)

derivativeEvaluate(*x*)

getGeometricDiscriminant()

getRealRoots()

setYn()

setdelta2()

seth2()

bounds.**Get_bounds**(*f*, *get_roots*, *m_range*, *upper_guess*, *lower_guess*, *k_2_range=[]*, *k_sim=True*, *debug=False*, *sep=0.05*)

bounds.**Get_bounds2**(*f*, *get_roots*, *x_range*, *search_range*, *additionalPar=0*, *k_sim=True*, *debug=False*)
Search for the zeros of the function f using the brentq optimization algorithm, separates two cases:

- k_sim is true : f is a bivariate function and the additional argument is given by x_range[i]
- k_sim is false: f is a trivariate function and the additinal arguments are given by (x_range[i],additionalPar)

bounds.**Roots**(*mass*, *lower_guess*, *upper_guess*, *sep*, *f*, *get_roots*)

bounds.**plot_**(*x*, *y*, *ax*, *label*, *color='b'*, *log=True*, *marker='o'*, *linestyle='-'*)

bounds.**procce_**(*Arr*, *Arr2*)

2.2.12 roots module

roots.**Get_roots**(*f*, *arg*, *array*, *j=0*, *debug=False*, *method=<function brentq at 0x000000000A0AB748>*)
Search for the all the zeros of a continuos and smooth function f, using the brentq optimization algorithm * Assumes that there is at most one zero in any of the subintervals (x[i],x[i+1]), where d(x[i+1],x[i]) <=0.03

The way it performs the search is the following: 1) Computes F_array using the function f, array and any other arguments that the functions needs(This means that f could be a multivariate function, but we are just fixing a subset of a line in which to search), this arguments are formated by the createArray function and we are making use of the convenient way python functions handle numpy arrays. 2) set a inital index j to 0 3) Starts at F_array[j] computes the sign of it, proceed through the elements of F_array until it founds i such that sign(F_array[j]) != sign(F_array[i]) 4) apply the brentq method to the interval array[j],array[i] and give all the necessary arguments. 5) changes j to i 6) Repeats steps 3-4-5 until there are no more elements in F_array

roots.**createArray**(*args*, *baseArray*)
Create a list of np.arrays whose values are all the same, using the values stored in args

CHAPTER

THREE

MODEL-CONSTRUCTION.PARAMETRIZATION

3.1 Specifications

- This modules define the functions for a set of models(atm *lotka volterra* and *rozenweigh macArthur*) and store it in a dictionary format for future use.

3.2 Contents

3.2.1 Dynamics module

Dynamics.**set_dCLV**(*R*, *C*, *P*, *a1*, *a3*, *e1*, *q1*, *m_C*, *m_P*)

Dynamics.**set_dCRM**(*R*, *C*, *P*, *a1*, *a3*, *e1*, *t_hc*, *t_hp*, *q1*, *m_C*, *m_P*)

Dynamics.**set_dPLV**(*R*, *C*, *P*, *a2*, *a3*, *e2*, *e3*, *q2*, *m_P*)

Dynamics.**set_dPRM**(*R*, *C*, *P*, *a2*, *a3*, *e2*, *e3*, *t_hp*, *q2*, *m_P*)

Dynamics.**set_dPredLV**(*R*, *Pred*, *a*, *e*, *m*, *q*)

Dynamics.**set_dRLV**(*R*, *C*, *P*, *r*, *K*, *a1*, *a2*, *m_C*, *m_P*)

Dynamics.**set_dRLVPart**(*R*, *Pred*, *r*, *K*, *a*, *m*)

Dynamics.**set_dRRM**(*R*, *C*, *P*, *r*, *K*, *a1*, *a2*, *t_hp*, *t_hc*, *m_C*, *m_P*)

Dynamics.**set_dRalone**(*R*, *r*, *K*)

3.2.2 Eq module

Eq.**setD**(*K*, *a_1*, *a_2*, *a_3*, *e_1*, *e_2*, *e_3*, *m_C*, *r*)

Eq.**setDBound**(*K*, *a_1*, *a_2*, *a_3*, *e_1*, *e_2*, *e_3*, *m_C*, *r*)

Eq.**setDis**(*K*, *q_1*, *q_2*, *r*, *a1*, *a2*, *a3*, *e_1*, *e_2*, *e_3*, *thc*, *thp*, *mP*, *mC*, *q2_0*, *q1_0*, *hC_0*, *hP_0*)

Eq.**setEqCDen_RM**(*e_3*, *q2_0*, *hP_0*)

Eq.**setEqCNum_RM**(*q_2*, *mP*, *a2*, *R*, *e_2*, *q2_0*, *hP_0*)

Eq.**setEqPDen_RM**(*K*, *q_1*, *q_2*, *r*, *a1*, *a2*, *a3*, *e_1*, *e_2*, *e_3*, *thc*, *thp*, *R*, *C*, *P*, *mP*, *mC*, *q2_0*, *q1_0*, *hC_0*, *hP_0*)

Eq.**setEqPNum_RM**(*K*, *q_1*, *q_2*, *r*, *a1*, *a2*, *a3*, *e_1*, *e_2*, *e_3*, *thc*, *thp*, *R*, *C*, *P*, *mP*, *mC*, *q2_0*, *q1_0*, *hC_0*, *hP_0*)

Eq.**setRoot1**(*K*, *q_1*, *q_2*, *r*, *a1*, *a2*, *a3*, *e_1*, *e_2*, *e_3*, *thc*, *thp*, *mP*, *mC*, *q2_0*, *q1_0*, *hC_0*, *hP_0*)

Eq.**set_C_eq**(*m_P*, *m_C*, *K*, *q_1*, *q_2*, *r*, *a_1*, *a_2*, *a_3*, *e_1*, *e_2*, *e_3*)

Eq.**set_P_eq**(*m_P*, *m_C*, *K*, *q_1*, *q_2*, *r*, *a_1*, *a_2*, *a_3*, *e_1*, *e_2*, *e_3*)

Eq.**set_R_eq**(*m_P*, *m_C*, *K*, *q_1*, *q_2*, *r*, *a_1*, *a_2*, *a_3*, *e_1*, *e_2*, *e_3*)

Eq.**setb_R**(*K*, *q_1*, *q_2*, *r*, *a1*, *a2*, *a3*, *e_1*, *e_2*, *e_3*, *thc*, *thp*, *mP*, *mC*, *q2_0*, *q1_0*, *hC_0*, *hP_0*)

Eq.**setden_R**(*K*, *q_1*, *q_2*, *r*, *a1*, *a2*, *a3*, *e_1*, *e_2*, *e_3*, *thc*, *thp*, *mP*, *mC*, *q2_0*, *q1_0*, *hC_0*, *hP_0*)

3.2.3 Equilibrium module

Equilibrium.**Cramers_L_solver**(*E*, *B*, *j*)

3.2.4 MTP module

MTP.**MTP_O**(*P_eq*, *C_eq*, *R_eq*, *MTP_f*)

MTP.**MTP_f**(*R_eq*)

MTP.**set_MTP**(*K_CP*, *K_RC*, *m_P*, *f_R*, *f_C*, *f_P*, *MTP_C*, *MTP_O*, *MTP_f*)

3.2.5 coarsegrainparams module

coarsegrainparams.**alfa**(*m2*, *alfa0*, *pv*, *pd*, *D*, *g*, *f*)

coarsegrainparams.**b_k = 8.617332400000001e-05**
metabolic and biomechanic parameterization of the search rate taking into consideration foraging strategies

coarsegrainparams.**f**(*k_*, *form*, *a*, *b*, *c*)

coarsegrainparams.**g**(*k_*, *pv*, *pd*, *T1*, *T2*, *E1*, *E2*, *D*, *v01*, *v02*, *fm*, *thermy1*, *thermy2*, *k*)

coarsegrainparams.**set_K**(*k0*, *mr*, *w*, *Ek*, *Tr*, *k*)

coarsegrainparams.**set_alfa**(*m2*, *alfa0*, *k_*, *pv*, *pd*, *T1*, *T2*, *E1*, *E2*, *D*, *v01*, *v02*, *g*, *alfa*, *fm*, *thermy1*, *thermy2*, *k*, *a*, *b*, *c*, *form*)

coarsegrainparams.**set_alfa0**(*d0*, *D*)

coarsegrainparams.**set_q1**(*q10*, *mc*, *w*, *Eq1*, *Tc*, *k*)

coarsegrainparams.**set_q2**(*q20*, *mp*, *w*, *Eq2*, *Tp*, *k*)

coarsegrainparams.**set_r**(*r0*, *mr*, *w*, *Er*, *Tr*, *k*)

coarsegrainparams.**set_th**(*t_h0*, *m*, *w*, *E*, *k*, *T*)

3.2.6 fingrainparams module

fingrainparams.**ConstructDynamicalFunctions**(*params*, *par_dict*, *K_RC*, *K_CP*, *m_P*, *R*, *C*, *P*)

fingrainparams.**Jacobian**(*dR*, *dC*, *dP*, *R*, *C*, *P*)

fingrainparams.**Jacobian2**(*dX*, *dY*, *X*, *Y*)

fingrainparams.**Stability**(*params*, *par_dict*, *eq_dict*, *K_RC*, *K_CP*, *m_P*)

fingrainparams.**Trophic_position**(*params*, *par_dict*, *eq_dict*, *m_P*)

fingrainparams.**construct_equilibrium**(*params*, *par_dict*, *K_RC*, *K_CP*, *m_P*)

fingrainparams.**construct_inv_boundaries**(*params*, *par_dict*, *eq_dict*, *K_RC*, *K_CP*, *m_P*)

fingrainparams.**construct_param_dict**(*params*, *K_RC*, *K_CP*, *m_P*)

fingrainparams.**setJacobianDict**(*DynamicsDict*, *R*, *C*, *P*)

3.2.7 functions module

functions.**func_transform**(*params*, *K_CP*, *K_RC*, *m_P*, *R*, *C*, *P*, *sim=True*, *bottom=False*)

functions.**setArgs**(*args*)

3.2.8 interpopparams module

interpopparams.**alfa**(*m2*, *alfa0*, *pv*, *pd*, *D*, *g*)

interpopparams.**g**(*k_*, *pv*, *pd*, *T1*, *T2*, *E1*, *E2*, *D*, *v01*, *v02*, *fm*, *thermy1*, *thermy2*, *k*)

interpopparams.**set_K**(*k0*, *mr*, *w*, *Ek*, *Tr*, *k*)

interpopparams.**set_alfa**(*m2*, *alfa0*, *k_*, *pv*, *pd*, *T1*, *T2*, *E1*, *E2*, *D*, *v01*, *v02*, *g*, *alfa*, *fm*, *thermy1*, *thermy2*, *k*)

interpopparams.**set_alfa0**(*d0*, *D*)

interpopparams.**set_q1**(*q10*, *mc*, *w*, *Eq1*, *Tc*, *k*)

interpopparams.**set_q2**(*q20*, *mp*, *w*, *Eq2*, *Tp*, *k*)

interpopparams.**set_r**(*r0*, *mr*, *w*, *Er*, *Tr*, *k*)

interpopparams.**set_th**(*t_h0*, *m*, *w*, *E*, *k*, *T*)

3.2.9 inva_fcl_stab module

inva_fcl_stab.**setI_C_LV**(*C*, *alfa3*, *e3*, *m_P*)

inva_fcl_stab.**setI_C_RM**(*C*, *e3*, *alfa3*, *m_P*, *t_hp*)

inva_fcl_stab.**setI_R_LV**(*R*, *alfa2*, *e2*, *m_P*)

inva_fcl_stab.**setI_R_RM**(*R*, *e2*, *alfa2*, *m_P*, *t_hp*)

inva_fcl_stab.**setMPTEqRM**(*I_R*, *q_2*)

inva_fcl_stab.**setMTP**(*I_R*, *I_C*)

inva_fcl_stab.**set_C_eq_s**(*m_C*, *r*, *K*, *q_1*, *a_1*, *e_1*)

inva_fcl_stab.**set_D**(*K*, *a_1*, *a_2*, *a_3*, *e_1*, *e_2*, *e_3*, *m_C*, *r*)

inva_fcl_stab.**set_I_C_s2**(*e_1*, *alfa_1*, *m_C*, *K*, *q_1*)

inva_fcl_stab.**set_I_C_s2RM**(*e_1*, *alfa_1*, *m_C*, *K*, *q_1*, *hC_0*, *q1_0*)

inva_fcl_stab.**set_I_C_s5**(*e_1*, *alfa_1*, *alfa_3*, *m_C*, *m_P*, *R*, *P*, *q_1*)

inva_fcl_stab.**set_I_C_s5RM**(*e_1*, *e_2*, *alfa_1*, *alfa_3*, *m_C*, *m_P*, *R*, *P*, *q_1*, *t_hc*, *q1_0*, *q2_0*, *hP_0*, *hC_0*)

inva_fcl_stab.**set_I_P_s3**(*e_2*, *alfa_2*, *m_P*, *K*, *q_2*)

inva_fcl_stab.**set_I_P_s3RM**(*e_2*, *alfa_2*, *m_P*, *K*, *q_2*, *hP_0*, *q2_0*)

inva_fcl_stab.**set_I_P_s4**(*e_2*, *e_3*, *alfa_2*, *alfa_3*, *m_P*, *q_2*, *R*, *C*)

inva_fcl_stab.**set_I_P_s4RM**(*e_2*, *e_3*, *alfa_2*, *alfa_3*, *m_P*, *q_2*, *R*, *C*, *hP_0*, *q2_0*)

inva_fcl_stab.**set_MTP_C**(*Req*, *alfa2*, *m_P*, *q_2*, *e2*)

inva_fcl_stab.**set_R_C_eq_sRM**(*m_C*, *r*, *K*, *q1*, *q1_0*, *alfa_1*, *e_1*, *t_hc*, *hC_0*)

inva_fcl_stab.**set_R_eq_s**(*m_C*, *q_1*, *a_1*, *e_1*)

inva_fcl_stab.**set_a1**(*r*, *Req*, *K*)

inva_fcl_stab.**set_a2**(*e_1*, *e_2*, *e_3*, *a_1*, *a_2*, *a_3*, *m_C*, *m_P*, *Ceq*, *Req*, *Peq*)

inva_fcl_stab.**set_a3**(*D*, *a_3*, *Ceq*, *Req*, *Peq*, *K*, *m_C*, *m_P*)

inva_fcl_stab.**set_hdet2**(*a1*, *a2*, *a3*)

3.2.10 ipopparams module

ipopparams.**set_K**(*k0*, *mr*, *w*, *Ek*, *Tr*, *k*)

ipopparams.**set_q1**(*q10*, *mc*, *w*, *Eq1*, *Tc*, *k*)

ipopparams.**set_q2**(*q20*, *mp*, *w*, *Eq2*, *Tp*, *k*)

ipopparams.**set_r**(*r0*, *mr*, *w*, *Er*, *Tr*, *k*)

CHAPTER

FOUR

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

A

B

C

D

E

F

G

I

J

M

N

O

P

R

S

T

U

W