User guide for Concordia

A MATLAB code for gene flow simulations

Developed by:
Hein van Heck and Maarten van Strien

Document updated: May 14, 2014

Cover image: ”The Dutch VOC ship de Concordia in a calm sea, a flagship and a small ship in the distance” by Cornelis Bouwmeester.

Contents

Part I

Concordia

Chapter 1

Background and introduction

1.1 Idea and purpose of Concordia

Concordia is a code developed to model the spread of DNA in grasshopper demes. The code is written in MATLAB and works on any system provided MATLAB is installed. Concordia could be classified as an agent based model projected on a static cellular landscape. The agents/individuals are grouped in demes/colonies where the ease/probability of migration from one deme to an other is dictated by the cellular landscape. The information that the agents cary with them is their DNA, their present location and the locations of their birth. The information stored on the cellular landscape is the location of suitable habitat.

For big problems calculation time increases but the code is stable till fairly large problem sizes. Millions of individuals per generation are possible to model for hundreds of generations, on a single laptop.

1.2 Running Concordia

A calculation can be performed by putting all the .m-files in one directory and opening MATLAB in that directory. The code is controlled via the main MATLAB-file ”Concordia.m”. Performing a calculation can be done by typing ”Concordia” in the MATLAB command line, or by opening the file Concordia.m in MATLAB and pressing ”RUN” (Green play button).

The setup of the calculation is controlled by the input parameters which are save at the top of Concordia.m The input parameter values can be changed as much as the user likes without changing the main code.

1.3 Getting started with MATLAB

A new Matlab user should get familiar with the main commands by typing them in the Matlab command line and see what they do. Typing "help ***" in Matlab gives explanations and help on whatever is typed in stead of ***. For example typing "help fprintf" gives information on how to use "fprintf". Free MATLAB-tutorials can be found at many locations on the internet. Some examples are:

- The mathworks website: www.mathworks.com
- YouTube has many videos: MATLAB basics - intro - course for beginners etc..
- http://www.math.ufl.edu/help/matlab-tutorial/

A quick overview of the most commonly used Matlab-commands is given to get a quick idea of how the code is working. Concordia is saved into several files which use these commands.

- Commands via %. All text after a % symbol are not seen by the code.
- ; at the end of a line to suppress printing to the screen.
- parameters via: x=10.
- vectors via x(1:10)=1.
- matrices as: x(1:3,1:4)=2.
- vector/matrix multiplications as: z=x(1,:)*y(:) or z=times(x(1,:),y(:))
- do-loops.
- while-loops.
- if-elseif-else-end-statements.
- print statements via: fprintf.
- random numbers via rand(2) or random integers via randi(10).
- tic-toc timings, which work as a stopwatch to track the time spend on certain parts of the code.

1.4 Structure of Concordia: Naming conventions

1.4.1 Concordia

Concordia is named after the sailing ship 'Concordia' that was built and owned by the Dutch VOC (East Indian Compagny). The ship Concordia was built in 1696 in Delft and last seen in 1708 after which it was registered as being lost.

1.4.2 Physical naming

DNA: a substance that carries genetic information in the cells of plants and animals (http://www.merriam-webster.com)
Allele: any of the alternative forms of a gene that may occur at a given position in a chromosome (http://www.merriam-webster.com)
Deme: a group / colony of interbreeding individuals.

1.4.3 Numerical naming: Parameters and matrixes

The names of parameters and matrixes in Concordia are quite long in order to keep them as self explanatory as possible. All names consist of letters only. Parameters and individual numbers start with lower case letter while all matrixes start with a upper case letter. When names consist of several words all words after the first are written with a capital. Such as: ’numberOfMigrators’ (which is an integer) or ’DemeLocations’ (which is a matrix).

1.5 Structure of Concordia: Files and functions

To call/execute a function or subfunction the syntax is as follows. To use it we first need to define the arguments ”aNumber” and ”anotherNumber”

```
aNumber=3;    anotherNumber=5;
[sum]=add(aNumber,anotherNumber);
fprintf('Sum = %2.1f' ,sum)
```

would print ”Sum = 8.0” to the screen. Provided a function ”add” is a in file ”add.m” in the same directory, which looks like:

```
function [result] = add (i1,i2)
result=i1+i2;
end
```

The complete code is distributed over several files. The main file is Concordia.m. When running the code, the program in this file is executed which will call routines/functions in other files from here.

In Concordia, different functions are saved in different files. Some functions may have sub-functions which are saved at the end of that file and are used only within that file. Concordia.m is the main file which holds calls to other files but none calls to Concordia.m or any other file. Flow control is always from the main file to one other and directly back to the main file. This structure makes Concordia extremely modular. Each file can be studied/expanded/substituted independently without affecting the rest of the code. The contents and purpose of each file of Concordia are discussed one by one in chapter 2.

Chapter 2

The four steps of a calculation

An overview of the code is given here, for more details see the commands added inside the files of interest. After reading the input parameters some initialisation is done, such as defining the appropriated sizes of some matrixes and setting the time counters to zero.

2.1 Reading parameters

Parameters are read from the main file Concordia.m. They can be saved in .mat files from which they can be loaded again for future calculations.

2.1.1 Input parameters

The input parameters are set at the top of the file Concordia.m. An overview is given in the tables below.

2.2 Initialisation

Further initialisation is done in 5 steps. The main purpose of these steps is to create the matrix MigrationProb which holds the probabilities to migrate from one deme to all others. In the current version of the code (version 1.0, 14 October 2013) several migration probability functions are implemented. A normal Gaussian and various forms of leptokurtic migration functions can be set following Ibrahim et al. 1996. Next to that, these steps create the matrices "Grasshoppers, Landscape, DemeLocations, MigrationDistance, DemeAngles and Shadow". Most of these matrices are mainly used to create/update MigrationProb and are useful in plotting routines. Apart from Grasshoppers, which is used extensively in the rest of the Concordia.

Parameter	Typical value	Options and description
Parameters for the landscape, habitat and DNA setup		
landscapeNx	100	The size of the landscape in X-direction. Also used as length scale for the rest of the code.
landscapeNy	100	Same in Y-direction.
demeGeometry	1	Flag for what setup to use for the deme-locations. Demes are only placed at ”integer-coordinate numbers”. (Centre of cell for later treatment.) • 0; Random placement of demes inside the habitat. • 1; (Multiple-) Line set up. In this case, demeTotal and demeXDirection have to match demeYDirection. When demeTotal and demeXDirection are the same just one line of evenly spaced demes is produced. When $demeTotal = 4 * demeXDirection$, 4 lines of demes, etc. • 2; Equal ’cell-skipping-placement’ inside the habitat.
habitatForm	6	Shape of the habitat. Demes can only be placed inside the habitat. • 0; All domain is habitat. • 0-40; Neutral landscapes. • 101; Quadrants. • 102; 4 corners (islands). • 103; Big, connected rivers. • 104; Several big, not connected rivers. • 105; Connected rivers. • 106; Circle. • 107; Horizontal cross. • 108; Double cross. (amplifier for mutation spread). • 109; Imitation of Maarten’s area in Switzerland. • 110; Lines.
demeXDirection	10	Number of demes in X-direction (used when demeGeometry = 1).
demeTotal	100	Total number of demes.
demeSize	100	Number of individuals in a deme.
InitDNA	2	Flag for how to initialise allele distribution. • 1; All start as homozygous (2 of the same alleles) Where the allele-number equals the deme-number. So all individuals starting in deme 5 have DNA 5-5. • 2; Random. Each individual get two alleles allocated at random, varying from 1 to ”maxAllel” • 3; All start homozygous (2 of the same allele). Where only 1 allele-number is allowed in each deem and the allele-number for each deme is allocated at random (from 1:maxAllel)
maxAllel	10	Total different values for alleles. Not used when initDNA=1.
nLoci		Number of loci to track.

Parameter	Typical value	Options and description
Parameters for reproduction		
noChildren	0	When 1, just 1 generation of grasshoppers is moved around. When 1, the calls to reproduce and die are skipped.
numberOfOffspring	20	Number of eggs that two parents produce.

2.2.1 The landscape / habitat / demes

The function "habitat", at the end of the file "initialise.m", builds the habitat. In the future the habitat could be used for various things but at the moment it is only used to dictate the possible locations of demes. The habitat is then imposed on the landscape which is used further on in the code. Which geometry the habitat takes is determined by the input parameter "habitatForm". Any geometry could be added to mimic a certain landscape.

(Initialise.m) Setting up the landscape, including the locations of the demes and individual grasshoppers. This function produces the matrixes "Grasshoppers, Landscape and DemeLocations".

- GrassHoppers holds location and DNA-info for each individual, this matrix is not changed until time stepping has begun.
- Landscape holds the meaning of each cell on the landscape. Each cell gets either the value 0, 1 or 2. (where 0=habitat, 1=deme and 2=not-habitat). Once created, this matrix is never changed.
- DemeLocations holds the x-y-coordinate of each cell that holds a deem, plus the number of the deme. Strictly, this matrix is not needed since the information is also stored in Landscape. Loops over all demes (later in the code) are easier to write when this matrix exists.

2.2.2 The migration function

(MigrationFunction.m) Setting up the migration function and determining the distances and directions (angles) between all demes. This function produces the matrixes "MigrationProb, MigrationDistance and DemeAngles".

MigrationProb Holds the probabilities to migrate from each deme to each other deme. The probability to move from deme 2 to 3 is stored at (2,3), later translated to an accumulated sense. MigrationProb is build over different files-functions where it is updated to include the effects of blocking, failing, accumulate etc. Once time stepping has begun, this

Parameter	Typical value	Options and description
Parameters for migration		
migrationCode	1	Code to define migration mode. • 1; Equal probability for all except the one where the individual departs from (which is 0). • 2; Probability depends linearly on distances between demes. • 3; Step function. • 4; Leptokurtic (or normal).
normalMigrationVar1	5	Variance used for normal-pdf (migrationCode = 4). Ibrahim et al. (1996) used 5.
normalMigrationVar2	40	Variance used for second normal-pdf of the leptokurtic law.
alphaLeptoKurtic	0.05	Ratio of the two normal-pdf for leptoKurtic law.
resNormMigrationF	0.1	Resolution of normal migration functions. An exact solution is calculated at each distance per cell. Set to 0.1 for a solution of the normal pdf at 10 locations per cell in X, and the same amount in Y-direction. Keep small, much lower than 1.
coeffLinearMigration	1	Used for linear relation to distance (migrationCode = 2). Multiplied with the distances to set the probabilities. (1 is neutral, 10 makes it less likely to migrate far).
maxMigrationDistance	0.2	Used to set the limit of the step function (migrationCode=4). Scaled to the length of the landscape (landscapeNx). 0.2 means that all demes within 20% of the size of the landscape have equal probability to be reached, all other 0.
migrationPropability	0.01	Chance that each individual has to migrate (each time step).
scaleMigrProb	0	1=yes, 0=no. Normalise the probabilities to reach a deme such that theyadd up to 1.
useMigrationFail	1	(1=yes, 0=no) For distance dependent failing. (Die during migration.)
failExponent	2	For failed-distance law. Used to calculate the probability of dying during migration as function of distance to this exponent.
failCoefficient	1	For failed-distance law. Coefficient to multiply with the probability that an individual will die during migration.
matrixEffect	0	Incorporate matrix effect in migration probability. 0 = none, 1 = inhibiting matrix, 2 = facilitating matrix.
weakness	1	If matrixEffect = 1 or 2, the higher this effect is, the weaker is the inhibiting or facilitative effect of the matrix. Do not set to 0, 0.1 is very strong effect.
blockingMode	1	Code to set the blocking mode. • 0; Demes do not block migration paths, i.e. all demes can be reached. (provided the migrationCode allows it). • 1; Demes block migration paths for demes located behind them.”captureRadius” controls the width of the ”shadow-zone” in which demes can not be reached.
captureRadius	1.5	Used for blockingMode=1. length scale: LandscapeNx. When set very small (10^{-10} or so) only demes in a straight line behind the blocking deme will be blocked.

Parameter	Typical value	Options and description
Parameters for output		
calculateLCPaths	0	(1=yes) Switch for calling dijkstaHek to get least-costs-paths
onlyOneLeastCost	1	(1=yes) Calculate only least-cost-paths from one starting deme.
weightHabitat	1	weight to step to a habitatcell (dijkstra).
weightNonHabitat	10	weight to step to a non-habitatcell (dijkstra).
calcHabnonHab	1	(1=yes) Switch to skip timeconsuming part.
propHabOnlyDirect	0	(1=yes, 0=no) Used for proportion of habitat i n paths.
straightPathCalc	2	Version to calculate cells in a straight-line path between demes. 0=exact distance-in-cell version. 1=loops, 2=matrix-version.
calculateTimeDiagnostics	0	Calculate statistics every time step (1=yes).
calculateRawFst	1	1 = Save the raw pairwise Fst values together with the distance and proportion of habitat to a table.
plotResults	11	Best to leave 0 or 11 and plot things afterwards.
maxTimeSteps	100	Length of the calculation.
plotIntermFst	0	Switch for creating Fst graphs during calculation.
saveSteps	[25,50,75]	Time steps at which intermediate plots will be created.
distanceCut	0.5	non-Dim distance clustering for output.
outputFileStem	'Test'	name for the output files.
outputFileDir	'+Output'	Directory to save output.
saveMatFiles	0	save .mat files with all parameters for each run in the output directory (several MB per file).
copyMatFiles	0	save a copy to the current dir (1=yes)
aveLastSteps	5	Save average values over the last X steps (results.txt).

Parameter	Typical value	Options and description
Parameters for multiple calculations		
numberOfRuns1	3	Repeated calculations including initialisation.
allAlphaRun	0	(1=yes) Repeat calc. for multiple values of alphaLeptoKurtic
multipleAlpha	(0.1:0.1:1)	Values for alphaLeptoKurtic.
testCase	0	Use for testing simple setups via testCaseSetup.m (0 means not use it).

matrix is not changed anymore. It is used every time step, in migration.m.

MigrationDistance holds the distance between all combinations of 2 demes. The distance is made non-dimensional by scaling to the lengthscale "LandscapeNx". Once created, this matrix is not changed anymore. The matrix is used in later initialisation steps. migrationFail.m uses it to determine the probability to fail (die) during migration. blocking.m uses it to find which demes are blocking others. During time stepping, this matrix is not used apart from for plotting intermediate results.

DemeAngles holds the angles (directions) of the migration paths in degrees. 0 is in the positive x-direction, the angle increases anti-clockwise till 360 degree. The angle of the path from deme 2 to deme 3 is stored in DemeAngles(2,3). Once created, this matrix is not changed anymore. During time stepping, this matrix is never used. It is used for determining which paths are blocked by other demes (in blocking.m) and for determining the proportion of habitat that each path covers (in higherOrderPaths.m).

2.2.3 Failed migrations

(migrationFail.m) Decrease the migration probabilities as function of deme-distances. This function changes the values in the matrix MigrationProb. It reduces the probabilities (values in MigrationProb) based on the distance between the demes. It also creates the matrix MigrationFail which is not used further in the code in any way, it is just send back to the main file so that it can be inspected after each calculation.

Migration can fail for two reasons:
1. Not enough demes are close enough to be reached.
2. The individuals have a certain probability to die halfway migrating, typically increasing with the distance between the two demes.

In Concordia we needed to include these two rules, since we want to deal with different landscape geometries and include local deme density / direction to the landscape. It is implemented like this: The input parameter migrationProbability sets the probability that each individual has to start migrating (i.e.leaving it's deme). Then, the migration-function determines the probabilities to reach the other demes. In earlier versions of Concordia (before June 18th) the probabilities were scaled/normalised so that the total of these probabilities was set to 1. (in accumulateProb.m) meaning that all individuals that start a migration reach a deme. Now (June 18th) an option is added not to use this scaling. When not used the probabilities come directly from the migration function. I believe this is what we should use. Having this option is equivalent to a simple implementation of a directional spread of migrators. For example, if a deme is so remote that migrators leaving this deme can only reach 1 other deme, the old version (with the scaling in accumulateProb.m) would dictate that all individuals that start migrating would reach this one deme. In the new version the probability that this deme is successfully reached depends on the migration function. In all other cases the individual reaches no deme and dies while migrating. This might be the simplest way of including a directional walk/spread provided input parameters for the shape of the migration function are chosen properly.

When individuals fail to migrate in this way they are moved to "deme 0" (in accumulateProb.m). reproduce.m is updated to ignore (/delete) the individuals that are placed in "deme 0".

A switch is added to the input parameters (scaleMigrProb) to switch to scaling on and off. It should be off (= 0) in standard cases.

- 2: useMigrationFail is one of the input parameters/switches that dictates whether or not this is used. If it is, individuals that migrate have some probability to die half way migrating. When used, the input parameters failExponent and failCoefficient determine the probabilities. In the shape of $probability - to - die = failCoefficient * (distance - demes^{failExponent})$.

2.2.4 Blocking

(blocking.m) Setting up which demes block access to others, and updating the migration probabilities. This function creates the matrix Shadow and changes the matrix MigrationProb.
In this file, the probabilities in the matrix MigrationProb are set to 0 for blocked paths, or, if demeTransparacy is not equal to 0, the probabilities for blocked paths are reduced by (1-demeTransparacy) for each deme that blocks the path. Once created, Shadow is not changed anymore. Shadow is used later in the code for splitting results for block-not-blocked paths and for finding what the shortest step-wise path between all demes is (in higherOrderPath.m)

2.2.5 Organise accumulated migration probabilities

(accumulate.m) Finalising the migration probabilities so that they can be used for time stepping. MigrationProb is updated one final time, after this routine MigrationProb is not changed anymore. The probabilities are accumulated so that the probability to move from deme 1 to deme 3 is given by the difference in MigrationProb(1,3) and MigrationProb(1,2). -so, summed row-wise towards the right.- If "scaleMigrProb==1" the probabilities are scaled so that they always add up to 1. So an individual that migrates will always reach a destination in this case. During time stepping, MigrationProb is (only) used in migrate.m.

2.2.6 Determine the portion of habitat on all paths

(habvsNonHab.m) Creates the matrix HabitatvsNonHabitat. Which holds the fraction of habitat on all straight line paths. Also StraightLinePath is created here which holds full paths (the landscape cell numbers).

Three version are implemented, separated by straightPathCalc. All versions assume that demes are placed at the centre of a landscape cell. If propHabOnlyDirect=1, only the non-blocked paths are included in this routine.
straightPathCalc=0 uses the oldest version in which exact distances in each landscape cell on the paths are calculated. straightPathCalc=1 uses a much faster version in which the landscape cells toughed by the paths are counted. The fraction of habitat cells is then saved as the number to indicate HabitatvsNonHabitat. straightPathCalc=2 is very similar to straightPathCalc=1 but uses faster matrix multiplications.
straightPathCalc=1 could be removed since either 0 or 2 should be used. Typically 2 is preferred since this methods is also used for the least cost paths, calculated in leastCost-Path.m.

2.2.7 Determine the shortest stepwise distance between demes

(higherOrderPaths.m) Calculates the shortest step-wise distance between all demes. The routine uses the matrix Shadow and creates the matrix ShortestPath. By multiplying Shadow with itself a matrix is obtained that holds all ”two-step-paths”. Multiplying that again with Shadow the three-step paths etc..

This routine is only based on blocked paths, and does not deal with demes so far apart that the migration function sets the migration probability to 0 (which is never with a leptokurtic law but will happen with other migration laws).

2.2.8 Find the least cost paths between demes

(leastCostPath.m) Works with a Dijkstra algorithm. It finds the least cost path between all demes, where a habitat-cell is cheap, and a non-habitat-cell is expensive to cross. (exact prices are controlled via input parameters). The location of other demes are threated as habitat-cells. Especially on large landscapes this routine takes up a lot of calculation time. An option is implemented to skip this routine completely or to calculate the least coast paths for only the middle-deme to all other demes to save calculation time.

2.3 Time stepping

Each time step is build up from 5 actions. Each action is performed in a separate file. ”tic-toc” commands are placed around each step so that the time spent in each routine can be tracked.

2.3.1 Migration

Migrate the grasshoppers: (Done in migrate.m)

Overview Each time step all individuals have some probability to migrate to an other deme.

Detailed description:
Loops over all the individuals (maxGH) and executes 3 steps:
A: See if this individual migrates.
This is based on migrationProbability which is the same for all individuals.
B: Check if he/she makes it.
Based on the highest value in MigrationProb, which represents the accumulated probability to succesfully migrate to all other demes. If this individual does not make it to a destination, it is placed in "deme 0" by setting it's present location to 0 (in GrassHoppers). Later routines (reproduce.m) will recognise that this individual died during migration.
C: If migration will be succesfull, determine to which deme it goes.
Based on MigrationProb. The individual's present location is changed to the deme it migrates to.

2.3.2 Reproduction

Create a new generation: (Done in reproduce.m)

Overview Done in three steps;
A: Find couples of parents (mom and dad).
B: Create eggs (from parents, number comes from numberOfOffspring).
C: Kill eggs and transform the survivors to new children.
Note that creating eggs and killing them (step 2b and 2c) are by far the most time consuming actions of the whole code. It could be replaced by a statistical method and/or producing only surviving eggs.

2.3.3 Generation replacement

Replace the old generation by the new one: (Done in die.m)

Overview Very simple routine. The old generation is removed and replaced by the new one. Later versions of the code might need to save information here so it is left separate.

2.3.4 Time dependent statistics

Calculate statistics: (Done in statistics.m)

Overview Saves some statistics that need to be saved every time step.
A: Saves "StaredEnd" which keeps track of the demes that moved from one deme to another in this time step.
B: Calculated the average standard deviation of the number of individuals per deme started in each deme. Mainly usefull when no children are born but might have it uses otherwise as well.
C: Calculates the number of homozygous individuals (per deme when combined with Grasshoppers).
D: Calculate Hs, Ht and Fst.
E: Calculate Spearman's rho (rank coefficient) for various diagnostics versus Fst.

2.3.5 Intermediate plots

Save data to output files: (Done in saveData.m)

Overview The routine is called every pre-set time step. (controlled by saveSteps) Typically allele frequencies and Fst vs distance is included here.

2.4 Finalising

2.4.1 Output: time.txt files

Each run (within each calculation) produces one time.txt file. In these files, one line is added every time step that statistics are calculated. Typical information is: the timestep, the number of migrators, the average Fst and the Spearman's rho for various diagnostics versus Fst.

If a time.txt file with the same name already exists it is overwritten by later calculations/runs so always change the name of the output files (via outputFileStem, or directory via outputFileDir) when previous results are supposed to be saved.

2.4.2 Output: fdh.txt files

Optionally (through calculateRawFst = 1) each run will produce a fdh.txt file. In these files pariwise FST values are given between each deme pair. Also the distance and the proportion of habitat is given between each deme pair.

2.4.3 Output: results.txt files

Each calculation performed produces one results.txt file, in which 1 line is added for each run performed within the full calculation. The results printed here are time averaged values over the last time steps, controlled via the input parameter aveLastSteps. Typical information is: the number of the run, the number of migrators, the value of alphaLeptoKurtic, and values for Spearman's rho for various diagnostics.

If a results.txt file with the same name allready exists it is overwritten by later calculations so always change the name of the output files (via outputFileStem) when previous results are supposed to be saved.

2.4.4 Output: .mat files

.mat files hold all information that was in the MATLAB memory at the time the file is created. For Concordia this means all input parameters plus all data as it was in the final time step of the calculation that created the .mat file.

The information in a .mat file can be loaded in MATLAB with the command:
load 'Test.mat'. This brings all information back into the memory and can be accessed and examined as usual.

2.4.5 Plotting

Concordia produces several forms of output automatically. "plotResults" controls the automatically generated output after each run. Note that latest versions of the matrixes/values used in the main file (Concordia.m) are saved in the memory after the calculation and can be used for plotting results after the calculation is finished. By running the code in plotting.m (by typing "plotting") the user can easily produce plots of pre-set format. Reset the parameter "plotResults" (by typing plotResults=5) before rerunning plotting gives different options. See plotting.m for the options.

Alternatively the user can create their own plot. For example, after the calculation is finished, the user could type: "scatter(MigrationDistance(1,:),Fst(1,:))" in the Matlab command line to get a scatter plot of Fst vs distance for deme 1. Or; "plot(steps,DNAHist(10,1:istep))" to get a plot of the number of allele type 10 versus time in the whole domain.

2.4.6 Checks and warnings

Throughout the code checks are implemented to prevent mistakes. The user should however realise that it is close to impossible to prevent/predict all possible mistakes that could be made. So when using Concordia always check if the combination of input parameters are sensible to use.

A big part of the checks/warnings is captured by checkInput.m, checkSetup.m and warnings.m. These routines are not needed for the calculation but just help the user/developer to run/expand the code without making mistakes.

After reading/defining the input parameters the function checkInput is called. In there some basic checks are performed mainly to prevent not-allowed values for input parameters. (such as negative values/inconsistent values etc.) When something impossible is asked by the input parameters the code stops the calculations, when something strange/unrealistic is asked a warning is given which is printed to the screen at the end of the calculation. (by calling warnings.m)

warnings.m is called at the very end of the calculation. It should help the user to direct him/her to points of improvement in the input parameters or bugs in the code.

Chapter 3

Main Matrixes

Here is an overview and explanation of the most important matrixes used in Concordia, listed in alphabetical order.

- "Children" is created in migration.m and passed to die.m. Children copies the information from "Eggs" for the eggs that make it to the new generation. Most eggs don't make it (depending on the input parameter "survivalChance"). So "Children" also has 4 columns (as Grasshoppers and Eggs), 2 for locations and 2 for DNA.

- "DemeAngles" holds the direction of the migration paths between all demes (in degrees). The direction of increasing Nx is taken as 0/360. The size of DemeAngles is NumberOfDemes x NumberOfDemes. The diagonal of the matrix is zero and the matrix is anti-symmetric (180 degrees difference between the direction of 3 to 1 from 1 to 3).

- "DemeLocations" is a matrix used to initialize the field and determine migration probabilities. It has 3 rows and as many columns as there are demes. The first two columns hold the X and Y coordinate of the deme and the third a number to identify the deme ("Deme number" or Deme-ID so to say).

- "Eggs" is created and used in reproduce.m. It is build from the information in "Parents" and "Grasshoppers". Each couple (row) in "Parents" produce "numberOfOffspring" eggs. The matrix Eggs has 4 rows (as Grasshoppers) to hold informations about their current deme, deme of birth, DNA1 and DNA2. The number of columns is equal to the number of eggs which is the number of couples (parents/2) times the number of offspring.

- "Grasshoppers" is an important matrix which holds information about the individuals and is used extensively in almost all functions. It has 4 rows (plus 2 rows for each extra locus) and as many columns as there are individuals at the time of calculation (constant

for now since deme size is constant). Grasshoppers looks something like this:
— 2 1 5 ... Deme number where the individual is now.
— 2 3 5 ... Deme number where the individual started or was born.
— 3 3 5 ... DNA-information on 1st allele.
— 2 1 5 ... DNA-information on 2nd allele.

- "Habitat" has the size of the landscape and hold a 1 for habitat and a 0 for non-habitat. It is used to restrict the possible locations of the demes. (in initialise.m) It can be used to guide/direct migration-paths and/or plot results on migration probability. The matrix HabitatSetup is used in initialise.m to build the matrix Habitat.

- "Landscape" is used for finding the ratio of habitat-nonhabitat on straight and least cost paths. And for finding the least cost paths. Later in the code it is only used for visualization. It is a LandsacpeNx by LandscapeNy matrix which has a 0 for "land", a 1 for a deme, and 2 for "non-habitat".

-"LandscapeList" Holds the same information as Landscape but is put in a list where the coordinates of the landscape cells are translated to node numbers.

- "MigrationDistance" holds information about the distances between each deme. This matrix is always symmetric regardless of the distance law. It has the size of NumberOfDemes x NumberOfDemes. For now (18 March) the distance can only be calculated as the shortest distance between 2 demes. It might be used in the future to hold "relative distances" based on terrain (longer over non-habitat). With different options to define the distance. Distances are normalised to the lengthscale (LandscapeNx) for more physical meaning.

- "MigrationProb" is a matrix first created in initialize.m (based on the migration law/code) and modified in migrationFail.m and blocking.m (based on the blocking code) and acumulate.m. Later it is used in migrate.m. It has rows and columns equal to the number of demes. Each row holds the (cumulated; scaled) probability to migrate from deme (row) to deme (column). Note that going from 1 to 3 doesn't need to have the same probability as going from 3 to 1.

- "Parents" is creates and used in reproduce.m. It is build from the matrix Grasshoppers to link two parents together. It has 2 rows, one row for "mothers" and one for "fathers", holding the location/column-number of the mothers/fathers in the matrix Grasshoppers. So each column holds the "ID/location in Grasshoppers" of a couple. The code looks for singles that live in the same deme to make couples of parents. Note that an odd number of grasshoppers in one deme leads to not every individual finds a "parent/couple". The code looks for couples stochastically, which is nice but takes a lot of calculation time.

-”PlotMigrationLaw” is a 2-row matrix to show the migration law after calculation. One row holds the distance, the other the probability given by the migration-law. The distance as saved here is non-Dimensional so 1 means LandscapeNx. (Type: ”plot(PlotMigrationLaw(2,:) , PlotMigrationLaw(1,:))” in the comandline.) The resulting plot shows the migration function before dealing with blocking/failing/scaling.

- ”Shadow” has the size of NumberOfDemes x NumberOfDemes. It is constructed in blocking.m. It holds a ”0” for demes that are blocked by other demes and a 1 for demes that can be reached. Once constructed it is multiplied by MigrationProp to set the probabilities of migrating to blocked demes to 0.

- ”SecondPath, ThirdPath, ..” have the size NumberOfDemes x NumberOfDemes. These matrixes are constructed in the file ”higherOrderPath.m”. They are similar to Shadow. The hold a 1 for the paths that need two steps (or 3 or 4..) to complete. They hold a 0 in all other cases.

Chapter 4

Assumptions and simplifications

As any model, Concordia makes use of several assumptions and simplifications. The most important ones are listed here.

4.1 Simplifications

- All individuals can mate with any other individual. The individuals are not separated in male/female.
- All couples produce the same amount of eggs. (But not all have to survive to become adults.)
- All demes hold the same number of individuals (demeSize=constant).
- Migration probabilities depend on the straight line distances between demes.
- The landscape consists only of 'suitable habitat' and 'not suitable habitat'.
- The location of suitable habitat is rasterised to the resolution of the landscape cells.

4.2 Assumptions

- Generations do not overlap. Individuals produce eggs, die, and after that, the eggs form the new generation.
- Individuals mate only once.
- All individuals mate to produce eggs unless there is an uneven number of individuals in a deme.
- All individuals have the same probabilities to: migrate, die during migration, be left out of the mating process.
- All eggs within one deme have the same probability to survive to adulthood.
- Allele type does not influence the probability to be passed to eggs.

Chapter 5

Performance

5.1 The speed of Concordia

In theory, Concordia can perform calculations with any number of demes and individuals on a landscape of any size. It has been tested for landscapes up to 10.000x10.000 grid cells on which no problems where reported. Tests on smaller landscapes with 1000 individuals per deme and 10.000 demes were also successful. For many runs or runs with a big setup however, calculation time becomes a severe limitation. (See figure 5.1)

The speed of a calculation is determined mainly by the size of the landscape and the number of demes. Some effort has been made to make the code efficient, but at several location in the code we chose clarity over speed and deliberately implemented time inefficient routines. A considerable speed-up can be expected if we would rewrite these routines. Several switches are included to switch off certain parts of the code that might give a speedup.

When analysing the speed of calculations, split thinking in initialisation and time-stepping. These things are very different and should be dealt with separately. The size of the landscape (number of landscape cells) greatly influences the calculation time needed for the initialisation phase. The number of demes influences both the initialisation phase and the time stepping phase. The speed of initialisation is limited by leastCostPath.m and habvsNonHab.m. The speed of time-stepping is limited by reproduce.m and statistics.m.

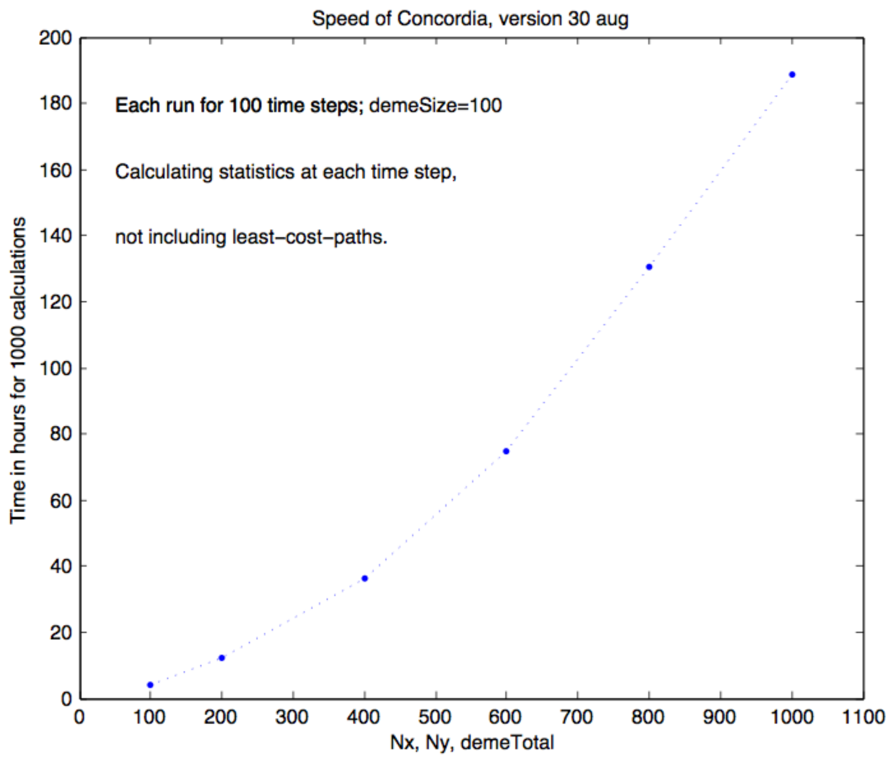

Figure 5.1: Indication of the speed of Concordia. Tests were performed on a laptop (standard MacBook, 2009) for different sizes of the landscape and number of demes. Each calculation was restarted from scratch, recreating the landscape and random placement of demes. The calculation time is extrapolated from 10 to 1000 calculations.

Part II

Benchmark and analytical solutions

Chapter 1

Intro

For any numerical code, comparison to analytical solutions are necessary. So far, Concordia has past several basic tests to analytical solutions which gives reasonable confidence that calculations performed produce correct results. Concordia should be able to pass all tests so if we think of another one we should implement and perform it.

A second reason to develop / find analytical solutions is to deepen our understanding of the systems. An analytical prediction of the evolution of Fst-values over time, under different conditions, would be ideal. The sections below describe simplified scenarios where analytical solutions are developed and tested which could eventually lead to predictions of Fst in a complex system where migration and fixation work simultaneously.

The analytical solutions described here have as goal to predict the evolution of the value of Fst over time in a system where individuals are both migrating and reproducing. In order to reach that goal we first developed solutions for systems in which either migration or reproduction was not allowed. In that way, we can isolate and study the effect of these two things independently.

At some point, we found it necessary to build solutions to scenario's where migration is only possible in one-way. This is not realistic (and probably eventually not needed) but was very useful in working towards solutions for two-way migration. It is also helpful in the sense that we now have at least a starting point for a-symmetric migration. All other scenario's considered assume equal migration probability to all demes where in simulations, we typically use a distance-dependent migration law.

Chapter 2

Fixation speed: Reproduction without migration

demeTotal can be set to 1 or migration can be stopped to create this scenario. When migrationProb=0, migration is not allowed so the system evolves only due to fixation.

The average number of timesteps it take before only 1 allele-type is present in a deme (fixation time) follow the analytical solution (see for example Wikipedia):

$$\text{Fixation Time} = \frac{-4 * demeSize * (1-p) * ln(1-p)}{p} \tag{2.1}$$

where p is the allele frequency which equals (on average) 1/maxAllele when DNA is allocated ad random.

A fit between this equation and several calculations performed by Concordia is shown in figure 2.1.

Note that demeSize in this equation (2.1) is the effective population size which is the same in our setup. This equation assumes DNA passing to the next generation ”drawn ad random from the allele-frequency table”. In Concordia this can be approximated by chosing a high value for ”numberOfOffspring”, in the same order as ”demeSize”.

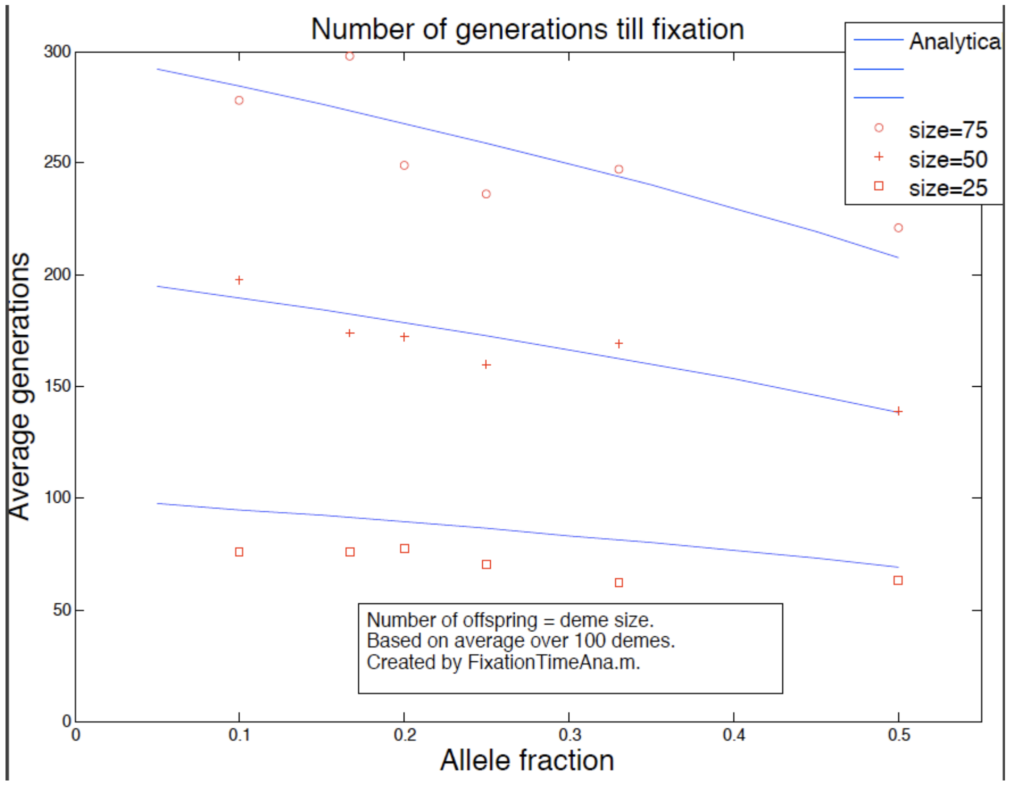

Figure 2.1: Fixation without migration. Varying both demeSize and Allele fraction (maxAllele). The results shows a good fit to the analytical solution. (Image produced with version 22-nov-2011)

Chapter 3

Pure mixing: Migration without reproduction

Via the input parameter noChildren the user can switch off reproduction (and dying). In this way, the effect of migration can be isolated and studied.

3.1 Pure mixing: Asymptotic value

Without reproduction, we can track where grasshoppers migrate to, and via initDNA=1 (all individuals get the DNA x,x where x is the deme number they start in) we can track how the DNA spreads over the domain. After a state of perfect mixing is reached the distribution of individuals with the same DNA over all demes is predictable and should follow a poisson distribution. So the number of individuals with DNA x,x in each deme should have a predictable distribution and standard deviation.

The standard deviation is calculated as:

$$\text{Standard Dev.} = \sqrt{demeTotal^{-1} * \Sigma_{i=1}^{i=demeTotal}(X_i - mean)^2}, \tag{3.1}$$

where X is the number of individuals with DNA 1,1 in deme i, and mean is the average individuals with DNA 1,1 in a deme (which equals demeSize/demeTotal).

The standard deviation used is calculated for all starting demes (first DNA 1,1 then 2,2, then 3,3 etc.) and averaged.

For a perfectly mixed system this standdard deviation is:

$$\text{Standard Dev.} = \sqrt{mean} \tag{3.2}$$

or:

$$\text{Standard Dev.} = \sqrt{\frac{demeSize}{demeTotal}} \tag{3.3}$$

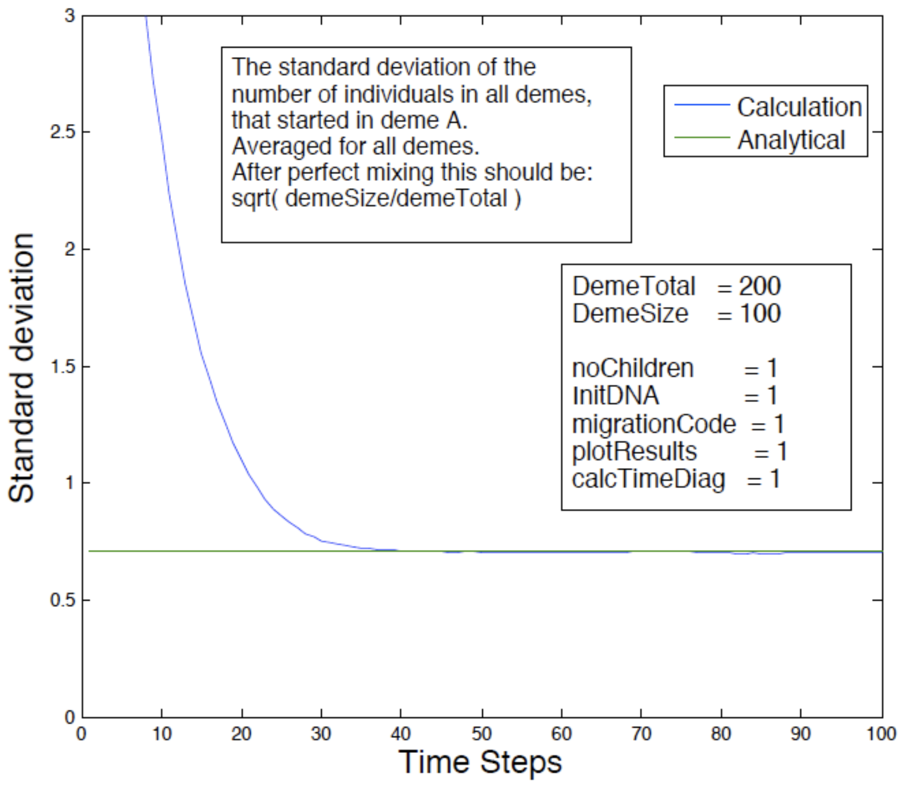

Figure 3.1: Pure mixing. This calculation was performed in a way that no reproduction (or dying) was possible, only migration. The results shows a good fit to the analytical solution. (Image produced with version 22-nov-2011)

The parameter migrationProbability (partly) controls the time it takes (number of timesteps) before the analytical solution is met. Results are shown in figure 3.1.

3.2 Pure mixing: Predicting evolution and time till perfectly mixed

The previous section described an analytical solution for the perfectly mixed state. This section describes the evolution of the system before that state is reached. In other words, we seek an analytical solution for the standard deviation as function of time.

3.2.1 The standard deviation for Time=0

At time=0, all indiciduals with DNA 1,1 are in deme 1, and none are in the other demes. So at time=0, the standard deviation of the number of individuals with DNA 1,1 in all demes is:

$$\begin{aligned}
\text{Standard Dev.} &= \sqrt{\frac{(demeSize - mean)^2 + (demeTotal - 1) * (mean)^2}{demeTotal}} && (3.4)\\
mean &= \frac{demeSize}{demeTotal} && (3.5)\\
\text{Standard Dev.} &= \sqrt{\frac{(demeSize - \frac{demeSize}{demeTotal})^2 + (demeTotal - 1) * (\frac{demeSize}{demeTotal})^2}{demeTotal}} && (3.6)\\
\text{Standard Dev.} &= \sqrt{\left(dS^2 - 2\frac{dS^2}{dT} + \frac{dS^2}{dT^2} + \frac{dS^2}{dT} - \frac{dS^2}{dT^2}\right) * \frac{1}{demeTotal}} && (3.7)\\
\text{Standard Dev.} &= \sqrt{\frac{demeSize^2}{demeTotal} - \frac{demeSize^2}{demeTotal^2}} && (3.8)\\
\text{Standard Dev.} &= \sqrt{mean * (demeSize - mean)} && (3.9)
\end{aligned}$$

For understanding, compare the first equation with equation 3.3. The rest is just rewriting the first equation to arrive at a simpler, more general form, written in the last equation which we can now use to make further predictions about the evolution over time.

3.2.2 Prediction for Time=Infinite

From the previous section we know that at time=infinity, the standard deviation will be:

$$\text{Standard Dev.} = \sqrt{\frac{demeSize}{demeTotal}} \qquad (3.10)$$

3.2.3 Form of the equation to find

Combining equations 3.9 and 3.10 to one equation gives:

$$\text{Standard Dev.(Time)} = \sqrt{\frac{demeSize}{demeTotal} * X} \qquad (3.11)$$

where X has to run from $\left(demeSize - \frac{demeSize}{demeTotal}\right)$ (at time=0 to match equation 3.9) to 1 (at time=inf to match equation 3.10).

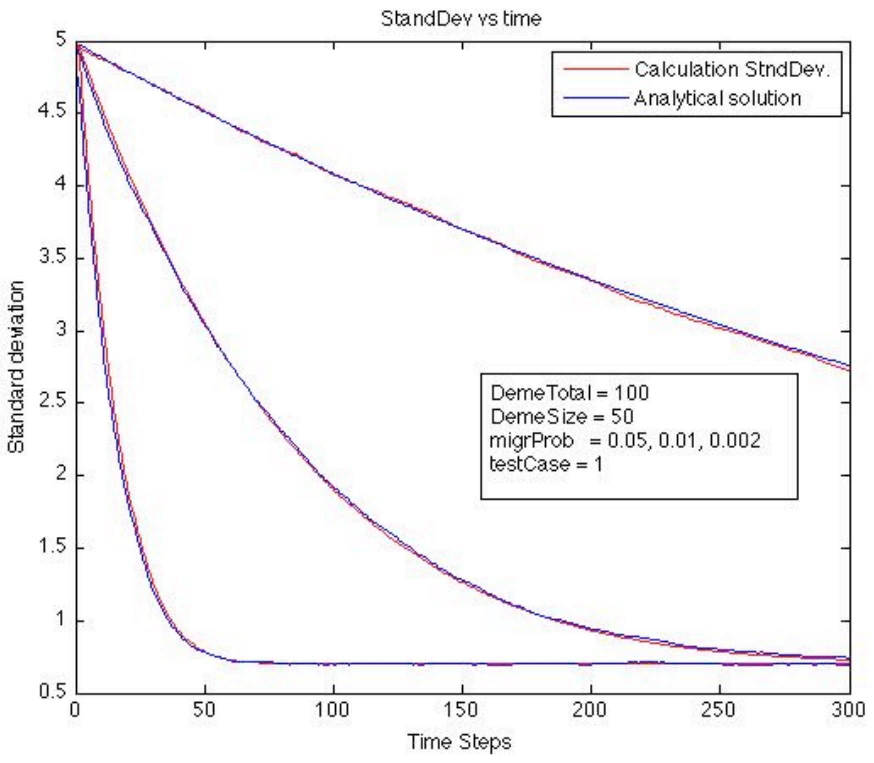

Figure 3.2: Pure mixing. Same setup as before, now matching evolution for 3 different values of the migrationProb. The blue lines follow equation 3.12, and the red lines come from the calculation. The same equation fits different values for demeSize and demeTotal but a plot in the same figure would require some more programming. (Image produced with version 6-dec-2011)

3.2.4 The eventual equation

After running a few testcases and trying to get a fit this equation seems to work very well (plotted in figure 3.2) :

$$\text{StandDev.(Time)} = \sqrt{\frac{(c(Time) - mean)^2 + mean * (c(Time) - mean)}{demeTotal} + \frac{demeSize - c(Time)}{demeTotal - 1}} \tag{3.12}$$

where c(Time) is the number of individuals with DNA 1,1 in deme 1 (on average). I found this equation by thinking about the effect of the number of individuals that are still in their starting deme, and the individuals that moved to an other deme separately. In equation 3.12 the first term under the square-root describes the effect of the own deme (compare to the equation defining standard deviation for understanding), and the second the effect of the other demes (which had canceling terms in it so is a bit harder to understand).

At $Time = 0$: c(Time)=demeSize (since all are in the starting deme), which means that the second term equals 0. At $Time = infinity$: c(Time)=mean, which means that the first term equals 0.

3.2.5 Show that the equation fits T=Infinity

To match equation 3.2 at $Time = Infinity$, our equation has to equal $\sqrt{mean}$. Which can be shown as follows:

$$\text{Standard Dev.(Time=Inf)} = \sqrt{0 + \frac{demeSize - c(Time = Inf)}{demeTotal - 1}} \quad (3.13)$$

$$\text{Standard Dev.(Time=Inf)} = \sqrt{\frac{demeSize - \frac{demeSize}{demeTotal}}{demeTotal - 1}} \quad (3.14)$$

$$\text{Standard Dev.(Time=Inf)} = \sqrt{\frac{demeSize(1 - \frac{1}{demeTotal})}{demeTotal(1 - \frac{1}{demeTotal})}} \quad (3.15)$$

$$\text{Standard Dev.(Time=Inf)} = \sqrt{\frac{demeSize}{demeTotal}} = \sqrt{mean} \quad (3.16)$$

3.2.6 Show that the equation fits T=0

To match equation 3.9 at $Time = 0$, our equation has to equal $\sqrt{mean * (demeSize - mean)}$. Which can be shown as follows (using "dS" for demeSize and "dT" for demeTotal), note that c(T=0)=demeSize:

$$\text{StandDev.(T=0)} = \sqrt{\frac{(c(T=0) - mean)^2 + mean * (c(T=0) - mean)}{demeTotal} + 0} \quad (3.17)$$

$$\text{StandDev.(T=0)} = \sqrt{\left((dS - \frac{dS}{dT})^2 + \frac{dS}{dT} * (dS - \frac{dS}{dT})\right) * \frac{1}{dT}} \quad (3.18)$$

$$\text{StandDev.(T=0)} = \sqrt{\left(dS^2 - 2 * \frac{dS^2}{dT} + \frac{dS^2}{dT^2} + \frac{dS^2}{dT} - \frac{dS^2}{dT^2}\right) * \frac{1}{dT}} \quad (3.19)$$

$$\text{StandDev.(T=0)} = \sqrt{\left(\frac{dS^2}{dT} - \frac{dS^2}{dT^2}\right)} \quad (3.20)$$

$$\text{StandDev.(T=0)} = \sqrt{mean * (demeSize - mean)} \quad (3.21)$$

3.2.7 Find an equation for c(Time)

In plotting.m, c(Time) is calculated by calculating b and d, where b is the number of indivuduals with DNA 1,1, leaving deme 1; and d is the number returning to deme 1.

$$\begin{aligned} b &= c(time) * migrProb \qquad (3.22)\\ d &= migrProb * (demeSize - c(time))/(demeTotal - 1) \qquad (3.23)\\ c(time+1) &= c(time) - b + d \qquad (3.24)\end{aligned}$$

It should be not to difficult to find an equation which predicts this decay. See that c starts as "demeSize", and ends as "mean".

The individuals that stay behind in the starting deme (so not including the ones that migrate and come back) follows a basic exponential decay as:

$$\text{IndividualsLeftInDeme(Time)} = demeSize * (1 - migrProb)^{Time} \qquad (3.25)$$

where Time simply is the number of time steps.

The individuals that return will be:

$$\text{ReturningIndividuals(Time)} = migrProb * \frac{demeSize - \text{IndividualsLeftInDeme}}{demeTotal - 1} \qquad (3.26)$$

These two equations have to "update each other" each time step. So they can be solved iteratively as is done in Matlab.

$$\text{c(Time)} = demeSize * (1 - migrProb)^{Time} + \frac{demeSize - demeSize * (1 - migrProb)^{Time}}{demeTotal} \qquad (3.27)$$

3.2.8 Write the equation to a final form

Our final equation (3.12) can be written in a slightly more practical form.

$$\text{StDev.(Time)} = \sqrt{\frac{(c(Time)}{demeTotal} * (c(Time) - mean)) + \frac{demeSize - c(Time)}{demeTotal - 1}} \qquad (3.28)$$

3.3 Pure mixing: Predicting Fst-global

By careful inspection of how the Fst-values and the standard deviations are calculated it becomes clear that the two are a measure of the same quantity (how well the system is mixed), and so one can be transformed into the other.

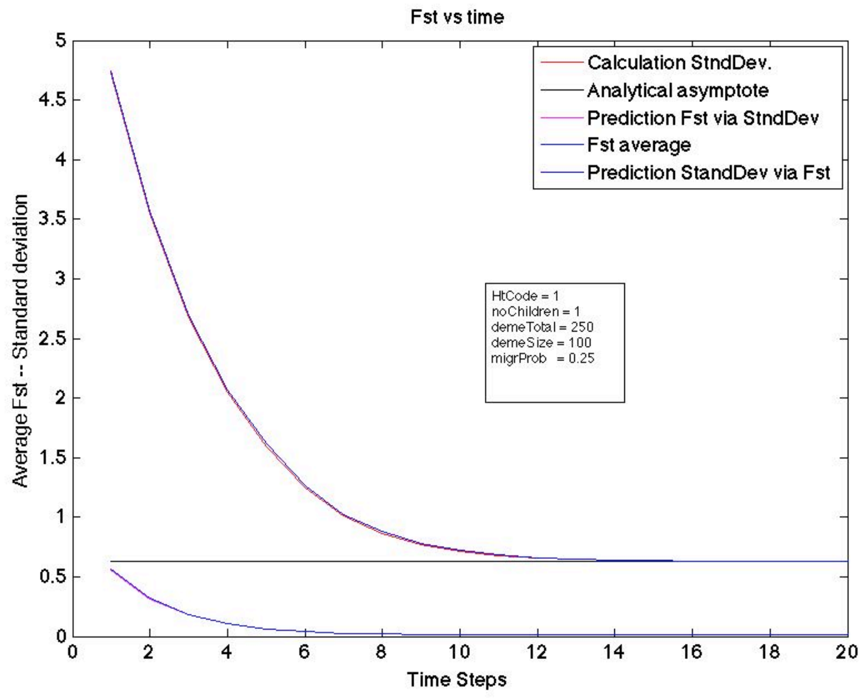

Figure 3.3: Migration, linking Fst and stand deviation. The figure shows the measured average Fst (global, via HtCode=1) and the measured standard deviation (as before). Plus the Fst calculation from the standard deviation and the standard deviation calculated from the observed Fst. Conversion is perfect since the lines are virtually the same. (Image produced with version 22-nov-2011)

$$\text{Fst-global} = \left(\frac{\text{equation 3.11}}{\text{equation 3.9}}\right)^2 \tag{3.29}$$

$$\text{Fst-global (Time)} = \left(\frac{\text{Standard Dev.(Time)}}{\sqrt{\frac{demeSize^2}{demeTotal}}}\right)^2 \tag{3.30}$$

Equation 3.30 can be understood by realizing that the time dependent standard deviation is scaled by its starting value. Then squared to translate the standard deviation to the variance which is the measure used in calculating the Fst parameter. Equation 3.30 can easily be inverted to calculate the standard deviation if the Fst-value is known.

Since we found an analytical solution to the evolution of the standard deviation (section

3.2, equation 3.12) equation 3.30 gives us an analytical solution of the evolution of Fst over time in a system where only migration is acting and fixation is not possible.

Figure 3.3 shows the conversion from Fst to standard deviation and standard deviation to Fst.

Chapter 4

Reproduction plus migration: 1 way migration

Now we have some idea how reproduction and migration individually effect the evolution of the system the next step is to try to understand and predict their effect in a system where both migration and reproduction takes place. Lets start with 1 way migration, so their is a separation between sending and receiving demes.

4.1 Reproduction plus migration: 2 demes, 1 sending, 1 receiving

The setup chosen is a landscape with only 2 demes, starting with all individuals in deme 1 having DNA 1,1 and all in deme 2 having 2,2. Migration is only possible from deme 1 to deme 2 so eventually both demes will consist only of individuals with DNA 1,1. An analytical solution is derived here to predict how many time steps that will take. Migration is set such that all individuals that leave deme 1 will arrive at deme 2 (migrationlaw - migrationfailing etc. are not included.).

The analytical solution for the fixation time given in equation 2.1 predicts the time till fixation starting from the moment that the individual that will eventually pass its DNA to the whole deme is present in that deme. So to develop an analytical solution to our scenario here, we can use that prediction and add the time it takes for an individual that will take over the deme to reach the deme (migration speed).

The speed with which migrators arrive at deme 2 = migrationProbability*demeSize.

The probability that this individual will eventually pass his DNA to the whole deme = 1/demeSize (all individuals present in deme 2 have equal chance).

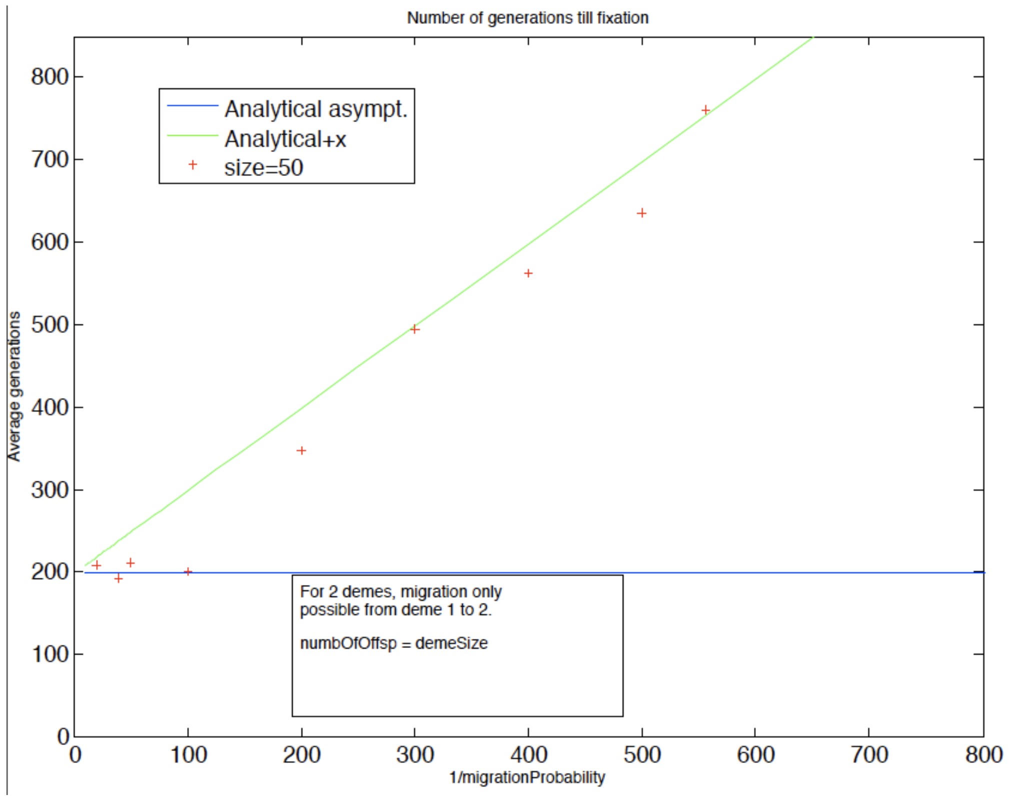

Figure 4.1: Fixation plus migration. Average result over 50 calculations per data point. (Image produced with version 22-nov-2011)

So the speed at which individuals that will pass their DNA to the whole deme arrive at deme 2 is given by the SuccesSpeed:

$$MigrationSpeed = migrationProbability * demeSize \tag{4.1}$$
$$SuccesChance = 1/demeSize \tag{4.2}$$
$$SuccesSpeed = MigrationSpeed * SuccesChance = migrationProb. \tag{4.3}$$

where SuccesSpeed has the unit [per timeStep] so 1/SuccesSpeed is of unit [timeSteps]. So 1/SuccesSpeed is number of timeSteps it takes, on average, before a migrater arrives at deme 2 that will eventually pass its DNA to the whole deme. Last step is to add this to the fixation time we had before which gives:

$$TotalFixationTime = 1/migrationProbability + FixationTime \tag{4.4}$$

As shown in figure 4.1 this solution fits the data well if the FixationTime is smaller than the succes speed. Which means that (on average) fixation is completed before the new "successful migrator" will arrive. If the fixation time is longer than the succes speed, total fixation is faster than predicted by our equation because during the fixation process new successful migrators arrive that will speed up the fixation process (left side of figure 4.1). I think it is possible to capture this effect in the equations to make a "perfect prediction" for total fixation time for this setup.

Note that in equation 2.1 we have to set the allele-frequence (p) to 1/demeSize since this will be the allele frequency of allele 1 in deme 2 at the start of the fixation phase.

Combining the equations give the prediction for this scenario as:

$$\text{Total Fixation Time} = \frac{1}{\text{migrProbability}} + \frac{-4 * demeSize * (1-p) * ln(1-p)}{p}, \qquad (4.5)$$

where $p = \frac{1}{demeSize}$

4.2 Reproduction plus migration: several demes receiving, 1 sending

As will become clear in later sections, in order to understand more complex systems, we first have to extend equation 4.5 to a more general case where any number of receiving demes can be present.

In this section we investigate the total fixation time for the setup of several demes receiving migrators and only deme1 can send migrators to all other demes (with equal probability using testCase=3).

The *FixationTime* for each receiving deme is the same as before, what changes is the *MigrationSpeed* (and so the *SuccesSpeed*) since now the migrators have to invade more demes.

In the case of 3 demes (2 receiving), 2 demes have to be invaded. The first deme to be invaded will take the *SuccesSpeed* (plus Fixation Time), the second one will take longer since the now successful migrators are send to both demes, which means that half of them will arrive at the deme that is already going to be overtaken. So, on average, after $1 * SuccesSpeed$ it will take $2 * SuccesSpeed$ to send a succesful migrator to the second deme. Then it will take *FixationTime* before this deme will reach fixation as well.

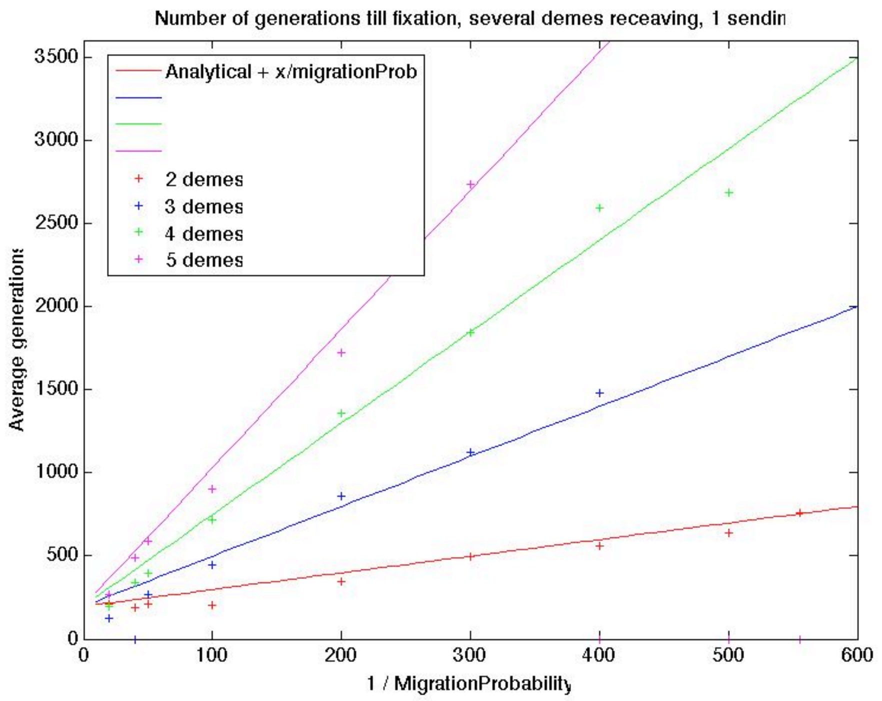

Figure 4.2: Fixation plus migration, 1 way migration. Same setup as before but now added '3-demes' (2 receiving) and '4-demes' and '5 demes'. Analytical solutions according to equation 4.12, where 'x' is explained below and p = 1/demeSize. Average result over 100 calculations per data point. (Image produced with version 28-nov-2011)

The 'new' $TotalFixationTime$, the time it takes before all demes reached fixation is thus:

$$\text{Total Fixation Time} = 1 * \text{Succes Speed} + 2 * \text{Succes Speed} + \text{Fixation Time}, \tag{4.6}$$

where the 2 terms with $SuccesSpeed$ combined represent the time it takes before both demes have received a migrator that will eventually spread its DNA to the whole deme.

To expand this equation to work for any number of demes we need to add a "$SuccesSpeedTerm$" for each deme that needs to be invaded (so $demeTotal - 1$ in total).

Each of these terms consists of 1/"the chance of reaching a deme that has not been reached yet" * $SuccesSpeed$. For 2, 3, 4 and 5 demes that equals:

$$(1) * \text{Succes Speed} \quad = \quad 1 * \text{Succes Speed}, \tag{4.7}$$

$$\begin{aligned}(1+2)*\text{Succes Speed} &= 3*\text{Succes Speed}, & (4.8)\\ (1+1.5+3)*\text{Succes Speed} &= 5.5*\text{Succes Speed}, & (4.9)\\ (1+1.33+2+4)*\text{Succes Speed} &= 8.33*\text{Succes Speed}. & (4.10)\end{aligned}$$

which is what is plotted in figure 4.2.

These equations can be put in general form as:

$$\text{Succes Speed Term} = \Sigma_{i=1}^{i=demeTotal-1}\left(\frac{\text{(demeTotal-1)}}{i}\right)*\text{Succes Speed}, \qquad (4.11)$$

So the final equation which predicts the average number of timesteps it takes before the whole domain reaches fixation in this scenario becomes:

$$\text{TFT} = \frac{\Sigma_{i=1}^{i=demeTotal-1}\left(\frac{\text{(demeTotal-1)}}{i}\right)}{\text{migrProbability}} + \frac{-4*demeSize*(1-p)*ln(1-p)}{p}, \qquad (4.12)$$

where "TFT" is the total fixation time and $p = 1/demeSize$.

This equation is in good agreement with the results from Concordia plotted in figure 4.2.

Chapter 5

Reproduction plus migration: 2 way migration

The next step is to consider 2 way migration, so their is no longer a separation between sending and receiving demes. All demes can both send and receive migrators.

5.1 Reproduction plus migration: 2 demes, 2 sending, 2 receiving

Repeat the previous experiment with 2-way migration.

The results of some test-calculations are plotted in figure 5.1. The slope of the line fits 1/(2*migrationProbability). Which makes sense since now twice as many individuals are migrating.
The starting value can be explained as follows.

When migrationPorbability is very high (so 1/migrationProbability approaches 0), the 2 demes can be seen as 1 deme (since transport-interaction is so high) for which we can immediately apply the equation for fixation time within 1 deme (equation 2.1).

Where now the demeSize is the size of both demes together, so demeSize*demeTotal, and alleleFrequency (p) is now $1/demeTotal$. So the equation now looks like:

$$\text{Fixation Time} = \frac{-4 * demeSize * demeTotal * (1-p) * ln(1-p)}{p} \tag{5.1}$$

The equations from the previous sectiion have to change to:

$$\begin{aligned} MigrationSpeed &= 2 * migrationProbability * demeSize & (5.2) \\ SuccesChance &= 1/demeSize & (5.3) \end{aligned}$$

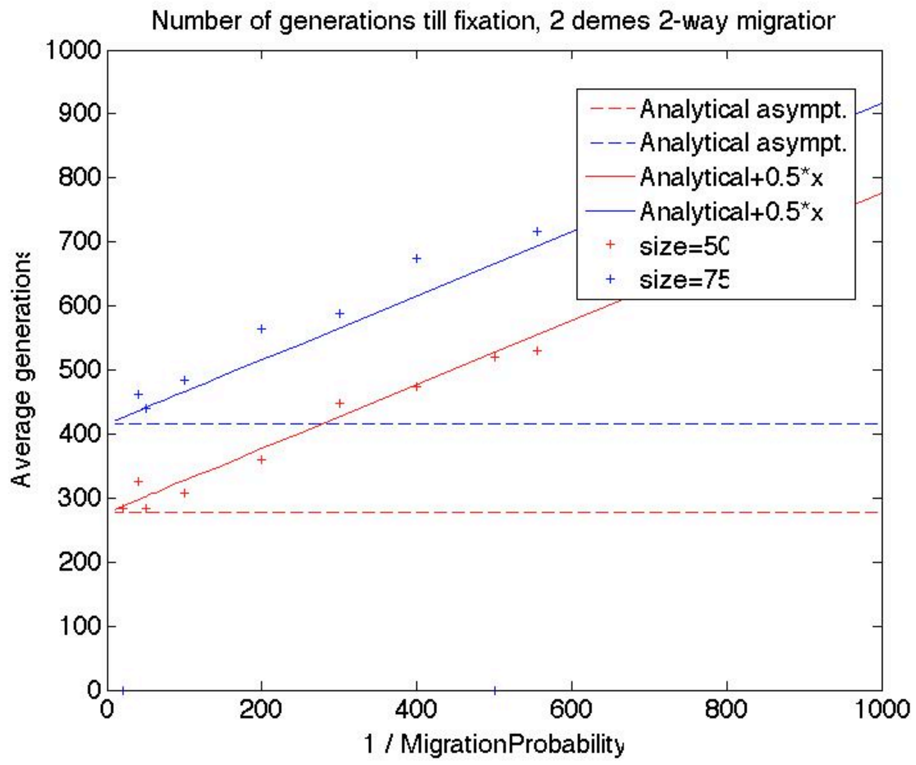

Figure 5.1: Fixation plus migration. Average results over 100 calculations per data point. Holding 2 demes from which individuals can migrate to each other. Calculations are performed for demeSize=50 and demSize=75. numberOfOffspring=demeSize. Clearly dynamics is different from the "1-way migration setup". Results seem to fit the equations plotted very well. (Image produced with version 28-nov-2011)

$$SuccesSpeed \quad = \quad MigrationSpeed * SuccesChance = 2 * migrationProb. \tag{5.4}$$

Total fixation time obays:

$$TotalFixationTime = 1/(2 * migrationProbability) + FixationTime \tag{5.5}$$

where the FixationTime now has a multiplication of demeTotal in it (equation 5.1).

Part III

Miscellanious

Chapter 1

Contact and reference

1.1 Contact

Concordia is made freely available so if you consider using it please do so. It would be appreciated if you inform either Hein van Heck or Maarten van Strien.

1.2 Reference

Ibrahim KM, Nichols RA, Hewitt GM (1996). Spatial patterns of genetic variation generated by different forms of dispersal during range expansion. Heredity 77: 282-291.