**File used to document any problems encountered during development**

**This document is used as a template for how to properly document problems that I have come across during development so that I can easily refer to them later.**

**The template is as follows:**

1. **Write a description of the feature I am trying to implement**
2. **State what the error description is I receive**
3. **State the number of unit test if applicable**
4. **Describe what raises the error**
5. **Describe the steps taken to solve the issue**
6. **Write a description of the feature I am trying to implement**

Trying to setup the initial package with licensing.

1. **State what the error description is I receive**
2. checking DESCRIPTION meta-information ... WARNING
3. Non-standard license specification:
4. MIT LICENSE
5. **State the number of unit test if applicable**

N/A

1. **Describe what raises the error**

This error is raised when using devtools::check() to check the validity of the files required by the package.

1. **Describe the steps taken to solve the issue**

A solution to the problem was to change the LICENSE line in the DESCRIPTION file to MIT + file LICENCE, where LICENCE is the name of the file required by CRAN. This file contains the information in the format:

YEAR: 2018

COPYRIGHT HOLDER: Ryan Porteous

Then adding this file to the .gitignore file so that GitHub will still correctly recognise the correct license file and adding the GitHub licensing file to .Rbuildignore.

1. **Write a description of the feature I am trying to implement**
2. **State what the error description is I receive**
3. **State the number of unit test if applicable**
4. **Describe what raises the error**
5. **Describe the steps taken to solve the issue**
6. **Write a description of the feature I am trying to implement**

Trying to create the structure of the chromosome and input/output nodes.

1. **State what the error description is I receive**

Warning messages:

1: In `[<-.data.frame`(`\*tmp\*`, i, 1, value = list(c.1. = 1, c.NA. = NA)) :

provided 2 variables to replace 1 variables

2: In `[<-.data.frame`(`\*tmp\*`, i, 1, value = list(c.1. = 1, c.NA. = NA)) :

provided 2 variables to replace 1 variables

1. **State the number of unit test if applicable**

N/A

1. **Describe what raises the error**

Error popped up when trying to execute the script. The reason was due my misunderstanding of R’s data structures.

1. **Describe the steps taken to solve the issue**

I researched my problem and looked at why I was having the issue. I then redesigned the structure I wanted the chromosome to be. I was previously trying to structure it in a 2D manner to make it easier to translate from the visualisation and actual representation of the chromosome. I am moving towards an approach of representing each section as its own data frame and I will just have to enforce constraints to provide the 2D structure I desired.

1. **Write a description of the feature I am trying to implement**

I am trying to write a function which generates the output nodes required

1. **State what the error description is I receive**

==> devtools::test()

Loading caRtesian

Loading required package: testthat

Testing caRtesian

v | OK F W S | Context

x | 31 1 | Population [0.2 s]

-------------------------------------------------------------------------------------

**test-population.R:65: failure: generateOutpus returns a data frame of the correct structure**

outputNodes1$chromoID inherits from `numeric` not `integer`.

-------------------------------------------------------------------------------------

== **Results** ==========================================================================

Duration: 0.4 s

OK: 31

Failed: 1

Warnings: 0

Skipped: 0

Warning message:

package 'testthat' was built under R version 3.4.3

1. **State the number of unit test if applicable**

test-population.R

test\_that(“generateOutputs returns a data … “, {  
 …

expect\_is(outputNodes1$chromoID, “integer”)

}

1. **Describe what raises the error**

I am trying to ensure that the correct structure is used for the output nodes generated and in this particular case I am checking the chromoID column has the type “integer”. When I run the test, it seems the type is actually “numeric”.

1. **Describe the steps taken to solve the issue**

I looked at the area in my code as to why it was producing numerics for this column instead of integers. I changed the appropriate line to use the seq.int function instead of casting the resulting vector from numeric to integer.

1. **Write a description of the feature I am trying to implement**

Unit testing for the creation of the function sets.

1. **State what the error description is I receive**

==> devtools::test()

Loading caRtesian

Loading required package: testthat

Testing caRtesian

v | OK F W S | Context

x | 45 1 | Population [0.2 s]

-------------------------------------------------------------------------

**test-population.R:123: failure: makeFunctionNode assigns the correct number of valid inputs**

`arity` not equal to `numInputsChosen`.

Types not compatible: character is not integer

-------------------------------------------------------------------------

== **Results** ==============================================================

Duration: 0.4 s

OK: 45

Failed: 1

Warnings: 0

Skipped: 0

Warning message:

package 'testthat' was built under R version 3.4.3

1. **State the number of unit test if applicable**

test-population.R

test\_that(“make FunctionNode returns the correct number … “, {  
 …

expect\_equal(arity, numInputsChosen)

}

1. **Describe what raises the error**

Arity is a character and a character is never equal to an integer which is what numInputsChosen is.

1. **Describe the steps taken to solve the issue**

I didn’t think that the types were the problem and assumed it was the makeFunctionNode function assigning the wrong number of inputs as the unit test suggests. So, I stepped through the function and didn’t find any errors. I then used the class function on the arity column returned from creating a function set and found this to be a character type. This is due to the way I create the data frames, I first convert the function definitions to matrices and matrices are heterogenous, so the integers are converted to strings to match the function names. This error has not appeared until now as the only place that has used the arity so far handles characters as inputs.

To solve this error, I had to do some searching around on StackOverflow for a solution on how to convert a column of a data frame to a different type. This post (<https://stackoverflow.com/questions/2288485/how-to-convert-a-data-frame-column-to-numeric-type>) explained the use of the transform function so this is the approach I took and I used this when creating the function set so that the issue is handled in one location saving potential problems later.

1. **Write a description of the feature I am trying to implement**

Testing that generateFunctionNodes works with different sizes of parameters.

1. **State what the error description is I receive**
2. **State the number of unit test if applicable**
3. **Describe what raises the error**

When the number of columns in the validInputIDs variable was higher than the number of input nodes there would be issues and not all valid nodes would be represented.

For example, if the number of columns was 4 and there were 2 input nodes then 2 valid values would be chopped off.

Another issue was that the validInputIDs was storing duplicates and NA values.

1. **Describe the steps taken to solve the issue**

The method for solving this issue was to set the number of columns accepted by validInputIDs to the largest number out of the number of columns and the number of input nodes. I also had to allow for duplicate values within validInputIDs which will only happen initially when the number of columns is greater than the number of input nodes.

Doing this led to a bias in sampling though so sampleWithoutBiasOrNA had to be implemented to remove this bias.

1. **Write a description of the feature I am trying to implement**

The ability to have multiple random constants through the program.

1. **State what the error description is I receive**

N/A

1. **State the number of unit test if applicable**

N/A

1. **Describe what raises the error**

The issue is that I currently store the return value of the sample function into the inputNodes data frame which worked fine but meant there would only be 1 random constant through the program. Changing it to store the function call instead required me to change the value column into a character type so that I could store “sample” as functions cannot be stored within data frames. This is another reason I should have used lists.

1. **Describe the steps taken to solve the issue**

Steps are described above, need to check it works when I’m decoding. It is going to require a lot of conversion to numerics through the program to compensate for the bad design of using data frames.

1. **Write a description of the feature I am trying to implement**

Allow user to change the selection method used.

1. **State what the error description is I receive**

N/A

1. **State the number of unit test if applicable**

N/A

1. **Describe what raises the error**

There is no error but there is an issue which has to be solved which is how to pass both the function to be called (in this case the selection method, tournamentSelection) and the arguments that the function expects into the program. As different selection methods require different parameters, the way the function is called must be dynamic to the parameters provided.

1. **Describe the steps taken to solve the issue**

I define an additional parameter to cgp which is called selectionMethod. This parameter expects a list which contains the function to call as the first argument and the arguments to the function as the second argument as a list. So that for calling muLambda it would be list(muLambda, c(1, 4)).

These parameters have to be compared to the number of parameters that the function expects and can be done through looking at the formals of the function defined. I wrote the validSelectionInput function to perform this and it also checks that there is a population parameter which is set to NA.

1. **Write a description of the feature I am trying to implement**

Ability for the user to choose the selection method to use and change parameters of it.

1. **State what the error description is I receive**
2. **State the number of unit test if applicable**
3. **Describe what raises the error**

The error lies in that it is not possible to know how many parameters this function will expect. For example, if I define two functions as follows:

Foo <- function(x, y) {}

Bar <- function(x)

Then I store them into the format required:

selectionMethod <- list(func = Foo, c(population = NA, 10, 20))

selectionMethod <- list(func = Bar, c(population = NA, 10))

They are called by doing:

selectionMethod$func() and passing in the parameters but it is impossible to pass the parameters in this way since one of the functions requires one parameter and the other requires two. R will complain if too many parameters are provided to the function and handling a list of parameters can be messy inside of a function which also means the user would have to handle this.

1. **Describe the steps taken to solve the issue**

I have went ahead as if this feature works and created a function validSelectionInput() to check the structure of the input is correct. I have also went ahead as if this works where I call the selection method to use. The user is still able to select a function to use and change the parameters but there must only be 2 parameters.

1. **Write a description of the feature I am trying to implement**

Program being able to have multiple random constants.

1. **State what the error description is I receive**
2. **State the number of unit test if applicable**
3. **Describe what raises the error**

The problem is the way I have designed my program to handle this functionality. I created a node which holds a function call and when this node is used, the function would be called which gives a random number to use as input. The problem is that this approach means the random number would change each time this function is called, thus the behaviour of the model generated would change.

The solution I planned for this was to store the value of the function call in the node that requires it, that way the value would be preserved for use even after evolution has finished. The problem with this is that the value is overwritten once data is propagated through the nodes.

1. **Describe the steps taken to solve the issue**

A method of solving this problem is to create another data frame of nodes but this time the nodes are called randomNodes. These nodes would be created for each time the random function is required and the value from the random function would be stored in these. This would require a good bit of refactoring I think. (NEED TO ACTUALLY CHECK).

So for now my software only supports one random constant and the value remains the same throughout.

1. **Write a description of the feature I am trying to implement**

Decoding of the solution.

1. **State what the error description is I receive**

Error: $ operator is invalid for atomic vectors

1. **State the number of unit test if applicable**
2. **Describe what raises the error**

I am trying to use the $ operator on a vector when this operator is only usable on recursive data structures such as lists or data frames.

1. **Describe the steps taken to solve the issue**

I understood this error and I corrected where the error was but it took a lot longer to fix as I was CTRL+L to load the package and then when calling the function after the changes to test it, it was calling the function I had loaded into my environment instead. This is something else that is quite annoying about R.

1. **Write a description of the feature I am trying to implement**

Decoding of the solution to get a value for a given set of inputs.

1. **State what the error description is I receive**
2. **State the number of unit test if applicable**
3. **Describe what raises the error**

There was no error but I realised this was a mistake.

The recursive version works and it does produce the correct output value but this method of calculating the value meant the value fields of each node was not used. This sounds great but it actually means that I am re-evaluating already used nodes which is what CGP was invented to prevent.

Also doing it this way meant I did not need to find the functionNodes that are actually used first since I could just start at the end and recurse back.

1. **Describe the steps taken to solve the issue**

I redesigned my algorithm into an iterative approach that iterates only through the required functionNodes which are found using the nodesToProcess function.

This new approach also stores the value inside each of the functionNodes so that if they are reused, they do not need to be re-evaluated.

1. **Write a description of the feature I am trying to implement**

Calculating fitness of the population.

1. **State what the error description is I receive**

Warning messages:

1: in sqrt(-79) : NaNs produced

List goes on

1. **State the number of unit test if applicable**
2. **Describe what raises the error**

The warning is raised as there is no square root of a negative number which R handles by setting the value as NaN. The output value for these inputs was set as NaN which then had the knock on effect of causing the fitness value for the solution be NaN.

1. **Describe the steps taken to solve the issue**

I originally thought this would be a problem when trying to sort the fitness values of the population so that NaN was ranked last but the order function in R already handled it. I just added the na.last = TRUE statement to make it clear to anyone using my code that this is how these values are handled.

1. **Write a description of the feature I am trying to implement**

Determining the valid inputs to a functionNode

1. **State what the error description is I receive**
2. **State the number of unit test if applicable**
3. **Describe what raises the error**
4. **Describe the steps taken to solve the issue**

This was an issue solved long ago but I want to write a bit about why the issue was an issue, how I solved it and how I would do it now. The way I would do it now is getValidInputs function but I didn’t know about this functionality at the time.

**The template is as follows:**

1. **Write a description of the feature I am trying to implement**

Calculate fitness of the population

1. **State what the error description is I receive**
2. **State the number of unit test if applicable**
3. **Describe what raises the error**

The issue is that every time the population fitness is calculated, another fitness is added instead of replacing the old one.

1. **Describe the steps taken to solve the issue**

Replace this line:

population[[i]] <- c(population[[i]], fitness = fitness)

with:

population[[i]]$fitness <- fitness

I assumed R would complain when trying to access a “field” (wrong name for this, find correct term) that doesn’t exist but instead it just creates it. The next time this function is run, it finds the $fitness field this time so it overwrites the old value

1. **Write a description of the feature I am trying to implement**

Testing that my program runs evolution

1. **State what the error description is I receive**

Error in \*7 : invalid unary operator

and also

Error in sin(-79, -79) : 2 arguments passed to 'sin' which requires 1

1. **State the number of unit test if applicable**
2. **Describe what raises the error**

The error is raised when running evolution and a solution is being mutated to produce offspring. During this mutation either the input values for a node are changed or the function is changed. In both the cases above the function was changed but the number of inputs were not changed to suit the newly chosen function.

In the first example, the previous function took only one argument and now the new function “\*” takes two so an error is thrown to show that there are not enough arguments to the function.

The previous function for the node in the second example was a function which took 2 arguments and newly generated function only takes one so when trying to call this function with the arguments in the node, an error is thrown as too many arguments were passed to the function.

1. **Describe the steps taken to solve the issue**

It was a mistake which caused both of these errors. The mistake was:

#Get the arity of the function that is currently used

arity <- length(unlist(solution$functionNodes[nodeChanged, ]$inputs))

#Get the inputs currently used

oldInput <- unlist(solution$functionNodes[nodeChanged, ]$inputs)

if (numArguments < length(oldInput)) {

#Need to add another input

…

} else if (numArguments > length(oldInput)) {

#Need to remove an input

…

}

The variables arity and oldInputs were essentially the same thing. oldInputs was correct but arity was supposed to be the arity of the newly chosen function but instead was the arity of the old function.

So when comparing arity and length(oldFunction) they were equal and neither branch of the if block was executed.

1. **Write a description of the feature I am trying to implement**

Testing that my program runs evolution

1. **State what the error description is I receive**

Error in sample.int(length(x), size, replace, prob) :

invalid first argument

1. **State the number of unit test if applicable**
2. **Describe what raises the error**

This error is raised in mutateFunction.

1. **Describe the steps taken to solve the issue**

It was not clear at first why this was an error when it worked for the other columns, so I had to step through this part line by line and check the values being passed around. I realised where the problem was once getValidInputs returned an empty integer vector as sample cannot sample anything from an empty vector and I had set size = 1 is sample’s parameters.

The issue was that the node being mutated was now using a function with a higher arity than before and required a new input to be added. The node being mutated was in the first column of functionNodes so had no other functionNodes it could use as input and getValidInputs worked correctly and returned an empty integer vector. I didn’t account for at this stage of the program that functionNodes can still use inputNodes as input so the solution was to also pass these to the sample function, that way whenever getValidInputs returns an empty integer vector, sample can still pick a random inputNode.

A unit test was added to capture this expected behaviour.