Gene Expression Programming

with the FRANKENSTEIN package

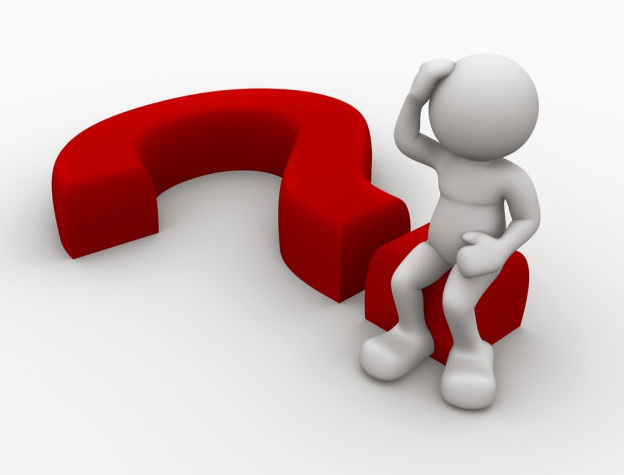
Roger Barlow

The University of Huddersfield

November 7th, 2019

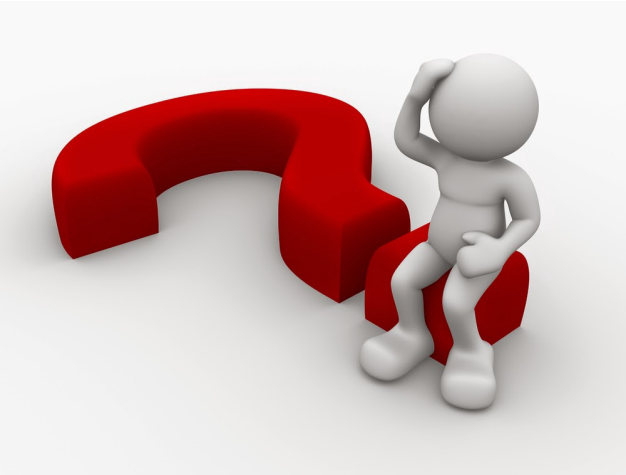
Solving a problem(1)

The obvious traditional approach



Solving a problem(1)

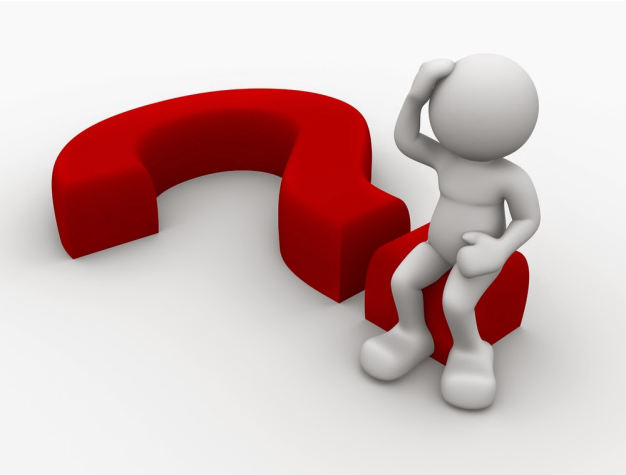
The obvious traditional approach



Write a program

Solving a problem(1)

The obvious traditional approach

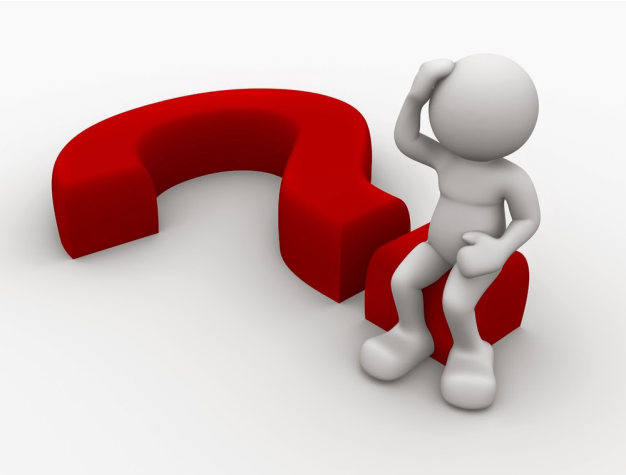


Write a program

Carefully consider goals, use cases, exceptions...
Using expertise and hard work

Solving a problem(1)

The obvious traditional approach

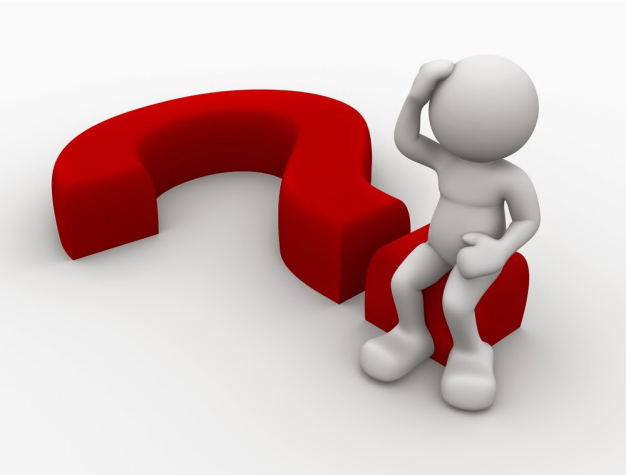


Write a program

Carefully consider goals, use cases, exceptions...
Using expertise and hard work
Intelligent design

Solving a problem (2)

Nature's way



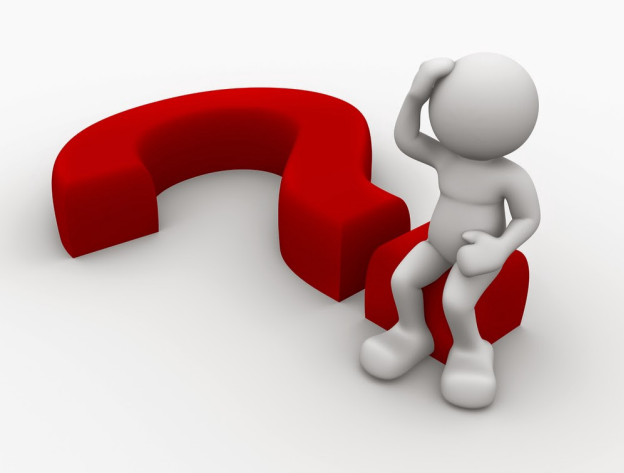
Many small feeble programs

Evolution – Natural selection – survival of the fittest

heredity – random mutations

Solving a problem (2)

Nature's way

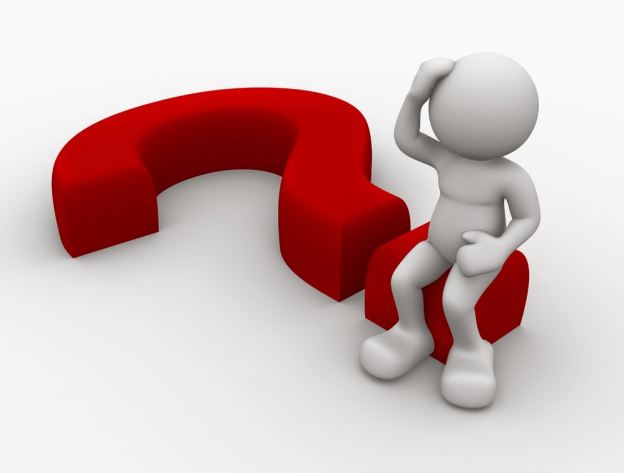


Evolve to bigger and better programs

Evolution – Natural selection – survival of the fittest
heredity – random mutations

Solving a problem (2)

Nature's way

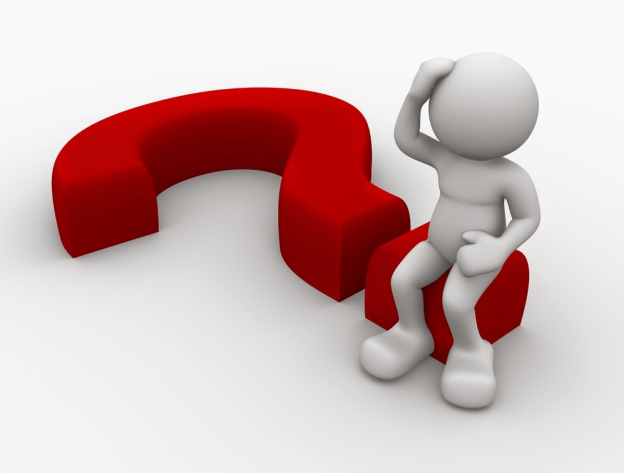


And a final super-program

Evolution – Natural selection – survival of the fittest
heredity – random mutations

Solving a problem (2)

Nature's way



And a final super-program

Evolution – Natural selection – survival of the fittest

heredity – random mutations

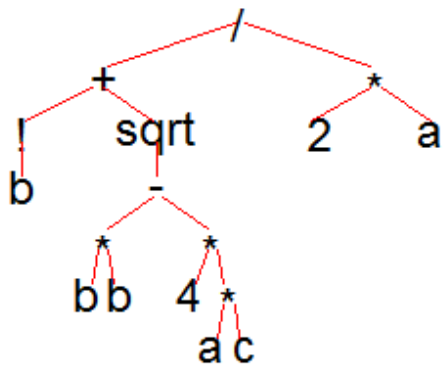
But how can you randomly modify a program without breaking it?

K language

C Ferreira "Gene Expression programming: Mathematical Modelling by an Artificial Intelligence", Springer (2006)

A program to evaluate any formula can be drawn as a tree and then uniquely written down as a string

$$\frac{-b+\sqrt{b^2-4ac}}{2a}$$



"/+*! $\sqrt{}$ 2ab-**bb4*ac"

K language syntax

Two sorts of string symbols:

- Functions
  - Monadic, with 1 argument (sqrt, sin, cos, log, exp, !...)
  - Dyadic, with 2 arguments (+, *, /, - ...)
- Terminators
  - Constants (2,4...)
  - Variables (a, b, c)

Bases for genetics, equivalent of ACGT in DNA (so larger alphabet)
Exact choice depends on problem - up to the user

Gene (fixed length) contains program(variable, up to length of gene).
Program starts with 1st gene element and finishes when complete.
Anything over is 'junk DNA'.
Gene divides into head and tail. $N_H + N_T = N_G$ and $N_T > N_H$

Rule#1: There can be no functions in the tail

Any gene obeying Rule#1 contains a valid program

Genetics

- Create population of genes, fill at random (subject to Rule #1)
- Cycle through many generations (size of population is constant)
  - Each gene/program is evaluated ('score')
  - The highest-scoring program survives and is not modified (elitism)
  - A random number of copies is made of each program, proportional to its score (Roulette wheel)
  - Point mutations
  - Transpositions (root and non-root)
  - Information Exchange (1-point and 2-point)
- It is good to do this several times (parallel universes)

Genes may undergo several mutations from one generation to the next

The scoring function

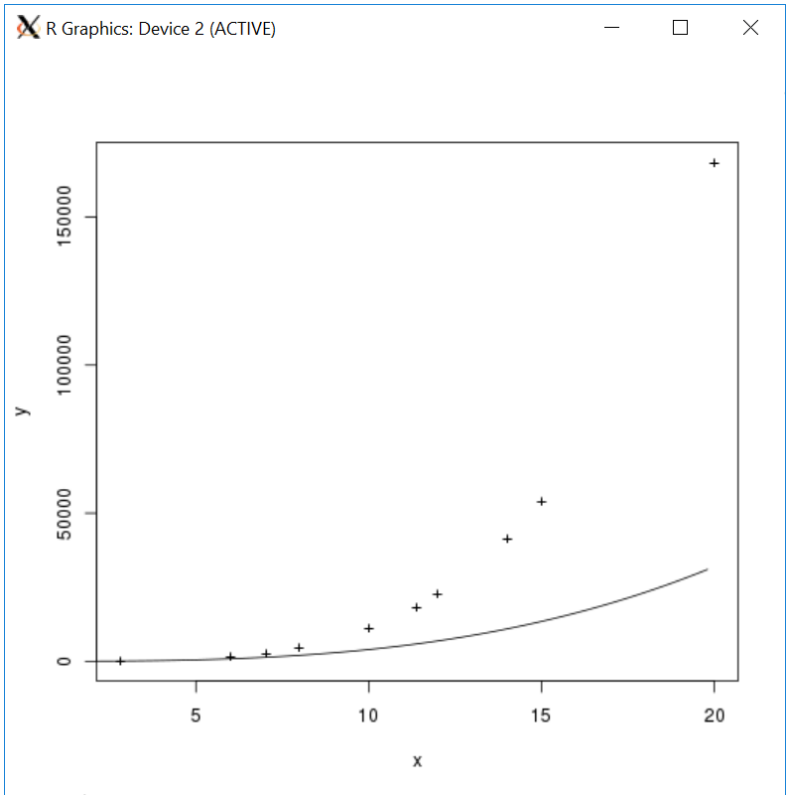
Turns out to be surprisingly important
Larger score is better program - for fitting this means smaller
$\sum(y_i - f(x_i))^2$
Need wide range of scores for roulette to get leverage. Weaklings must die!

But too much differentiation can flood the gene pool with good-but-not-excellent genes
Ferreira recommends $\sum(R - |y_i - f(x_i)|)$, where R is max possible range.
(If infinite, need to choose R and set any negative contributions to zero)

This does better than the 'more elegant' $\sum \frac{1}{1+(y_i - f(x_i))^2}$

An example

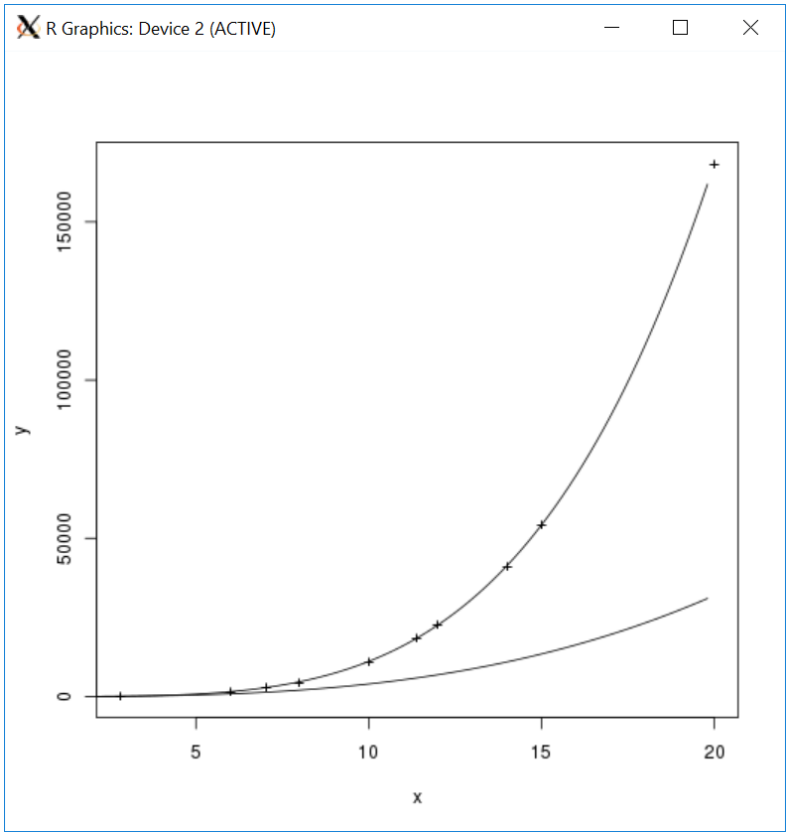
Suppose $y = x + x^2 + x^3 + x^4$
Evaluated for $x = \{2.81, 6, 7.043, 8, 10, 11.38, 12, 14, 15, 20\}$
Fit with population of 100, gene length 41 (=20+21)
Operations +-*/, variable x, no constants
Maximum distance for scoring $R = 100$



First generation: Score 65.1 with
$*--*+x-*/x+//++x/+-*$
$xxxxxxxxxxxxxxx$ which is $4x^3 - 3x$

An example

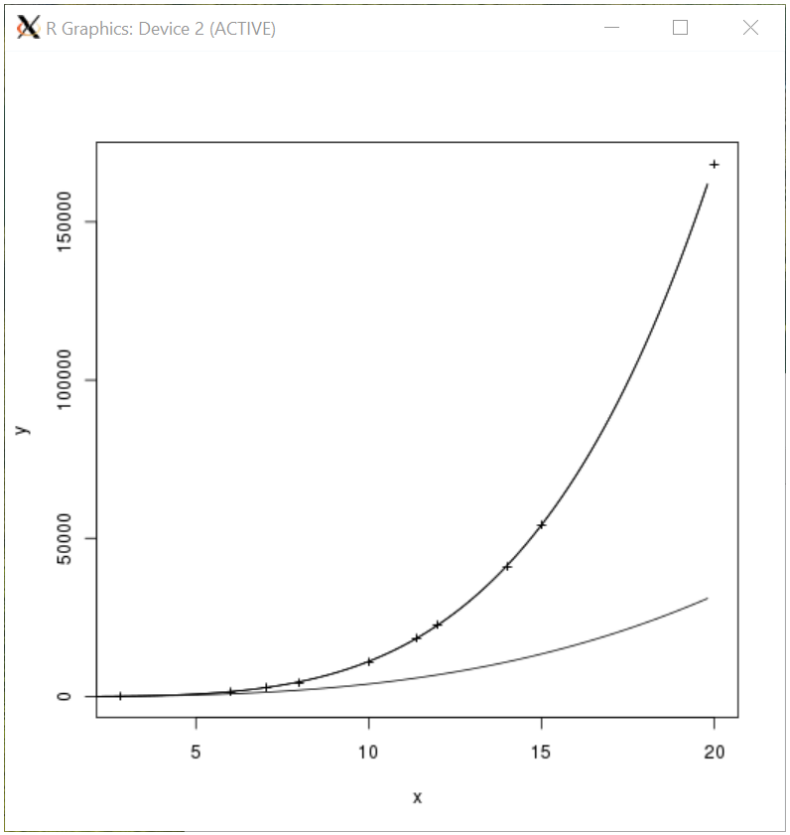
Suppose $y = x + x^2 + x^3 + x^4$
Evaluated for $x = \{2.81, 6, 7.043, 8, 10, 11.38, 12, 14, 15, 20\}$
Fit with population of 100, gene length 41 (=20+21)
Operations +-*/, variable x, no constants
Maximum distance for scoring $R = 100$



First generation: Score 65.1 with
$*--*+x-*/x+//++x/+-*$
$xxxxxxxxxxxxxxx$ which is $4x^3 - 3x$
Second generation score 903.8 with
$+-**x/+/++/*x/*-*$
$*/xxxxxxxxxxxxxxx$ which is
$x^4 + x^3 + x^2 + 1$

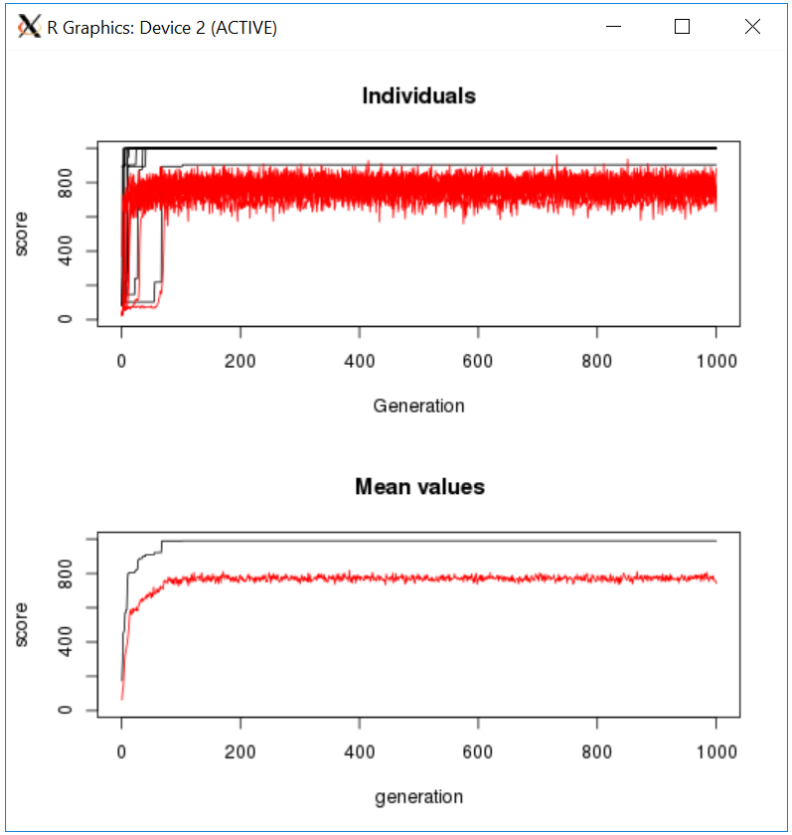
An example

Suppose $y = x + x^2 + x^3 + x^4$
Evaluated for $x = \{2.81, 6, 7.043, 8, 10, 11.38, 12, 14, 15, 20\}$
Fit with population of 100, gene length 41 (=20+21)
Operations +-*/, variable x, no constants
Maximum distance for scoring $R = 100$



First generation: Score 65.1 with
$*--*+x-*/x+//++x/+-*$
$xxxxxxxxxxxxxxx$ which is $4x^3 - 3x$
Second generation score 903.8 with
$+-**x/+/++/*x/*-*$
$*/xxxxxxxxxxxxxxx$ which is
$x^4 + x^3 + x^2 + 1$
Ninth generation score 1000 with
$+-**x/+/++/*x/x-**$
$/xxxxxxxxxxxxx$ which is
$x^4 + x^3 + x^2 + x$

Plotting progress



Plot shows improvement with time of best score(black) and mean score(red) for 10 different universes (above) and their average (below)
Period of improvement, then stable
Some universes get stuck in evolutionary dead ends.
Just because it's converged doesn't mean it's found the answer

The parameters

Design parameters for a particular problem:

- The alphabet of variables (prescribed), functions (guesswork) and constants (more guesswork)
- The number of generations - the larger the better, up to a point
- The number of universes - be generous
- The score function

Tunable parameters

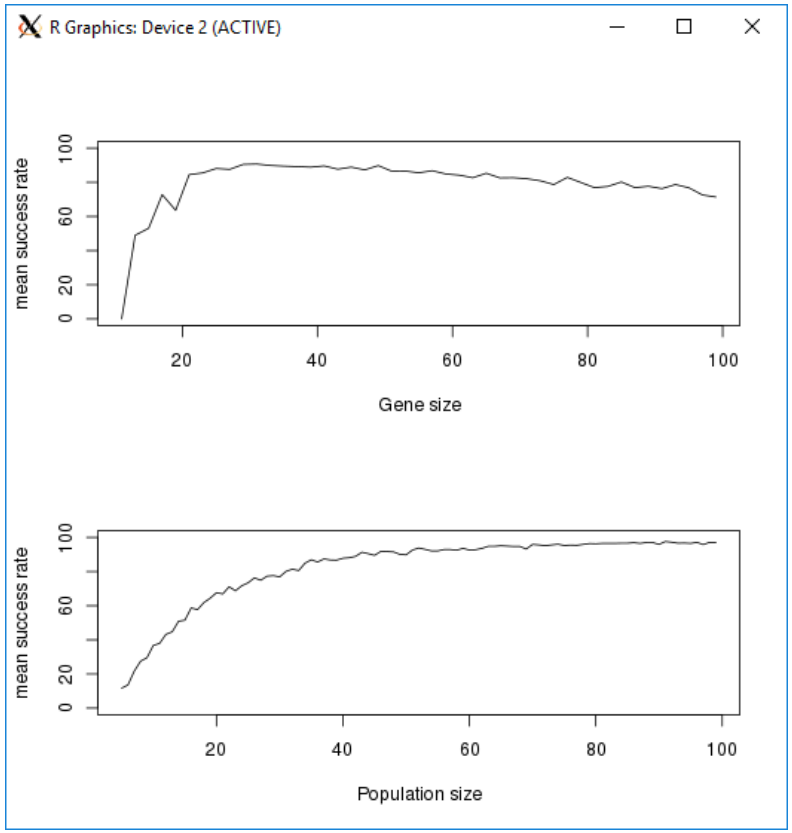
- L, the Gene length
- Npop, the population size - the larger the better, up to a point
- Pmu - number of random mutations
- Pti - number of non-root transpostiions
- Ptr - number of root transpositions
- Pr1 - number of single-switch reproductions
- Pr2 - number of double-switch reproductions

Dependence on L and Npop

Percentage succeeding - success defined as solution found within 100 iterations

Averaged over 1000 universes

Mutation numbers 2.0,0.7,0.7,0.7,0.7



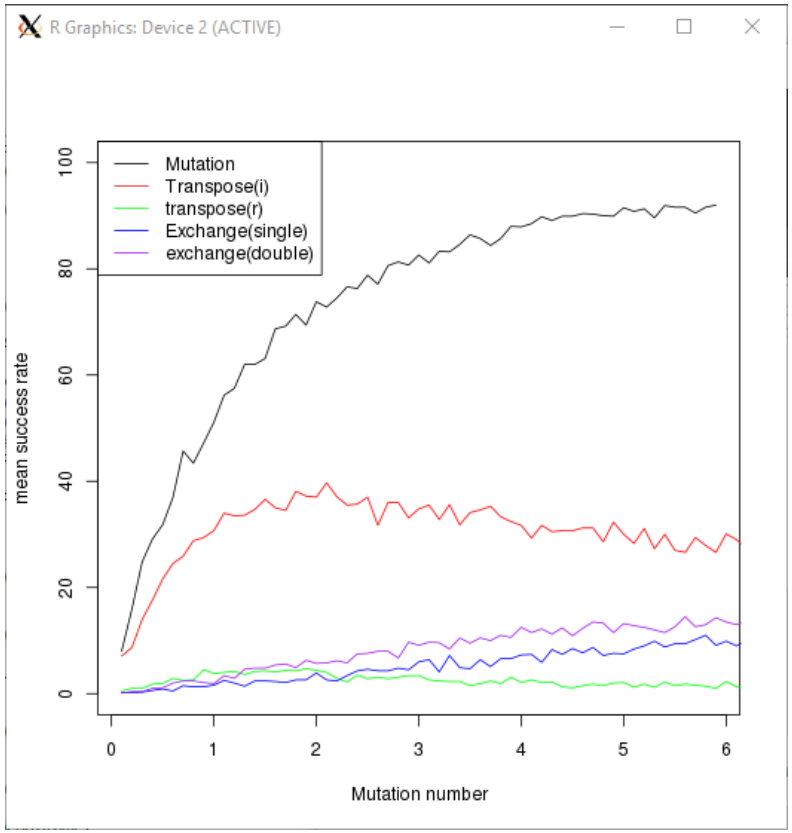
Rate rises and then slowly falls with L

Rises monotonically with Npop

Dependence on evolution parameters

L set to 81 and Npop=30

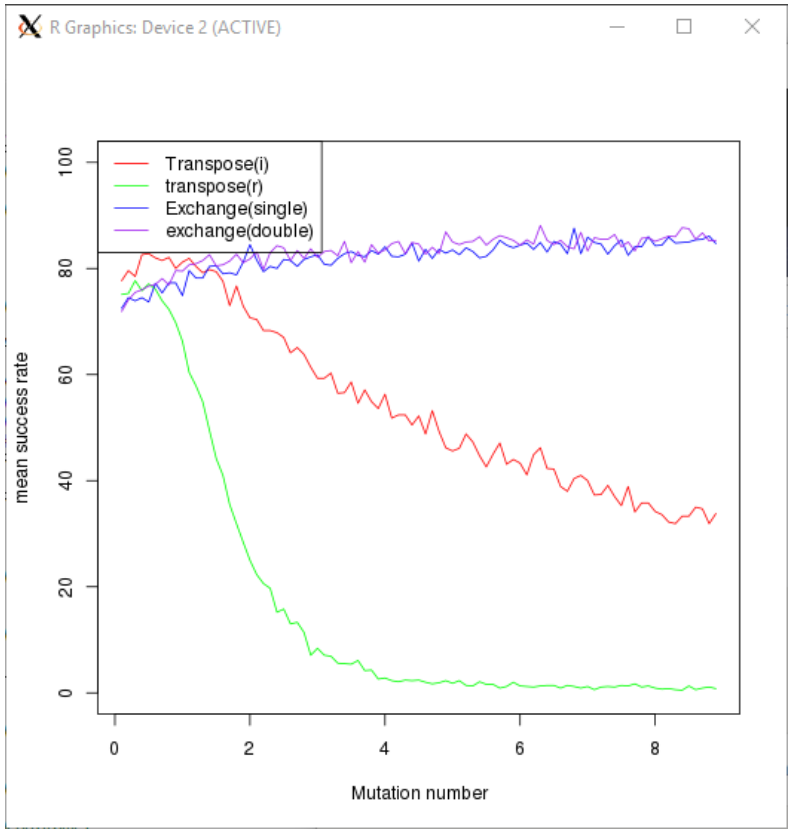
All other parameters zero



Pmu is far more effective than the others

Dependence on Evolution parameters

Set Pmu to 2.0: vary the others



Ptr does more harm than good.

Exchange brings slow benefits.

This is a very simple case. Blind chance can work well. More interesting cases may have different behaviour – but saw same behaviour in more complicated case

Parallelisation

Readily parallelisable using OMP at the universe level
(One core handling a complete evolutionary system so nothing shared)
Increase from single core to 24 cores increases speed by only factor ~ 2
Though was occupying $\sim 2400\%$ of the CPU
Using `time` showed very large system time

Reason

Program needs to generate lots of random numbers
Early version used C++ `rand()` function.
This is thread-safe but achieves this by imposing bottleneck
Instead use `gsl_rng` with separate allocation (and deletion) for each universe/thread.
Factor goes up to ~ 20

Extending the model

Constant constants and variable constants

Genetic optimisation good for choosing between models
Ordinary optimization good at determining parameters within models

Idea!

Let's combine the two

Have two sorts of constants in the K language

1. Constant constants, like 1,2,π...
2. Variable constants, like a, b, c...

To score a program, first optimize the variable constants

Variable constants: numerical minimisation

Use gnu scientific library `gsl_multimin.h` to find best values before returning score
Two methods:

Without-Dervatives

Simplex (Nelder-Mead) - evaluate function at mesh of random points, use results to update mesh.

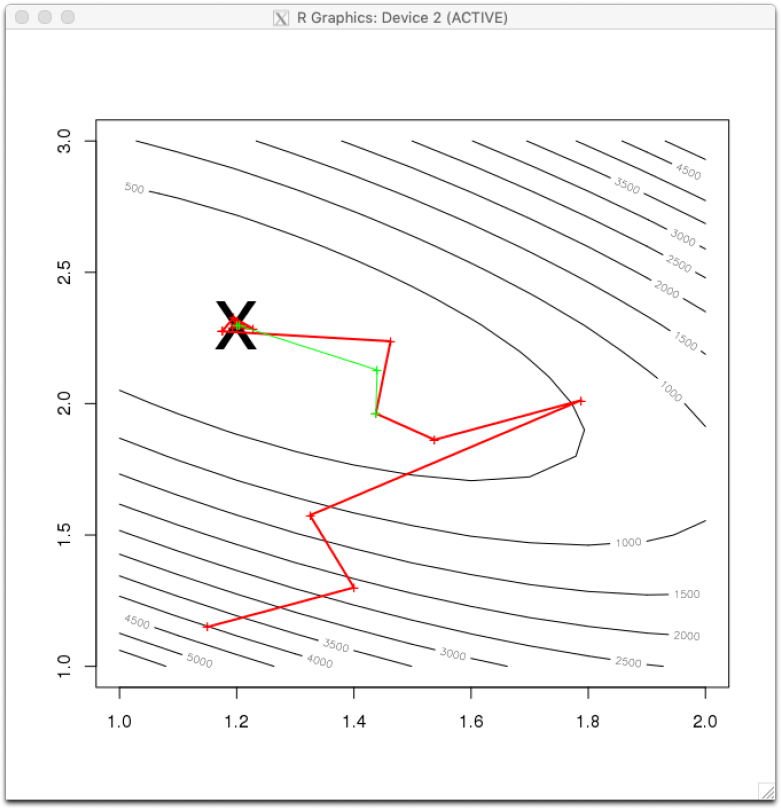
With-Derivatives

Quasi-Newton (Fletcher BFGS2) - use derivatives to get best direction

With-derivatives generally gives much better performance on final approach - but user has to supply derivative function.
Usual strategy: simplex, followed by quasi-Newton

Minimisation: simplex and gradient-based

Function $1.2x + 2.3y$ sampled at 100 points, $x = 0, 1...9, y = 0, 1, ...9$



Start at $a = 1.15, b = 1.15$

Red line is Simplex: reaches $a = 1.19967, b = 2.30016$ after 40 iterations

Green line is bfgs2 starting after 10th simplex iteration: reaches $a = 1.2, b = 2.3$ in just 3 more iterations

Differentiation with K language

We are minimising $S = \sum(y_i - f(x_i, \vec{a}))^2$

Differentials are $\frac{dS}{da_j} = -2\sum(y_i - f(x_i, \vec{a}))\frac{df(x_i,\vec{a})}{da_j}$

Need to differentiate, with respect to a and b, programs like

$$sin(a + x) + 2a - by \equiv \texttt{+ sin - + * * x a 2 a b y}$$

For each variable the differential is the sum of the differential for each instance (here 2 instances of a and one of b)

For each instance use chain rule up the tree: if $p(x) = q(r(x)), p' = q'r'$

Here: $\frac{df}{da} = cos(a + x) + 2, \frac{df}{db} = -y$

Have to supply functions that give differentials

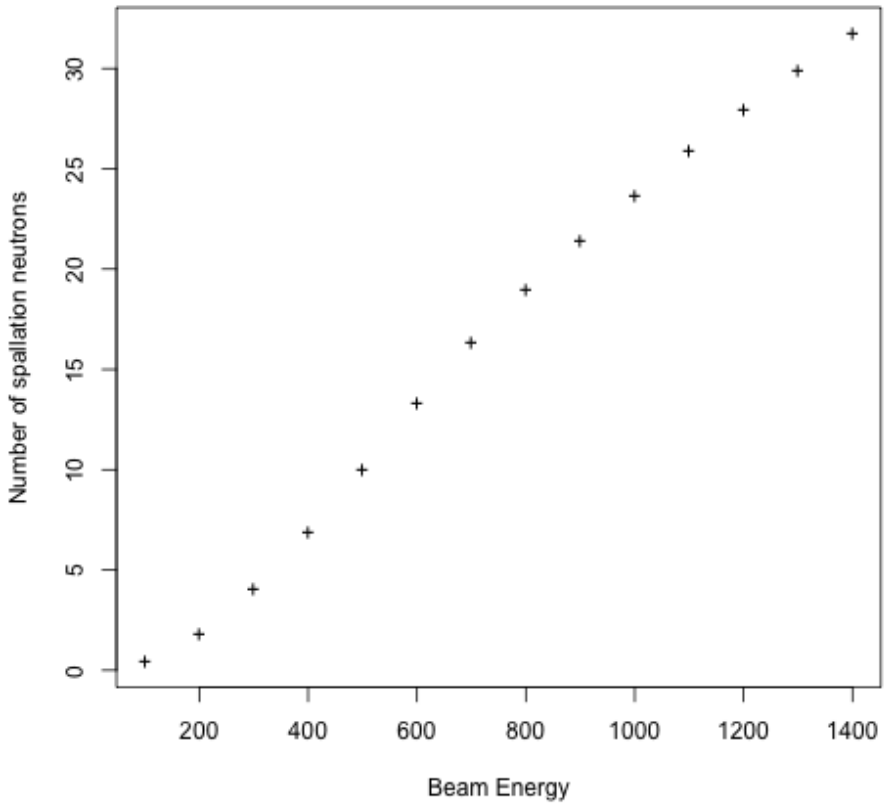
For `sin` can use `cos`. For `cos` define new function `Negsin` returning $-sin$

For dyadic operators have to distinguish between first and second argument. For simplicity and technical reasons this is done with extra argument which is + or - for 1st and 2nd, e.g.

```
double subD(double x,double y, int i) {return( i>0 ?  1 :-1);}
```

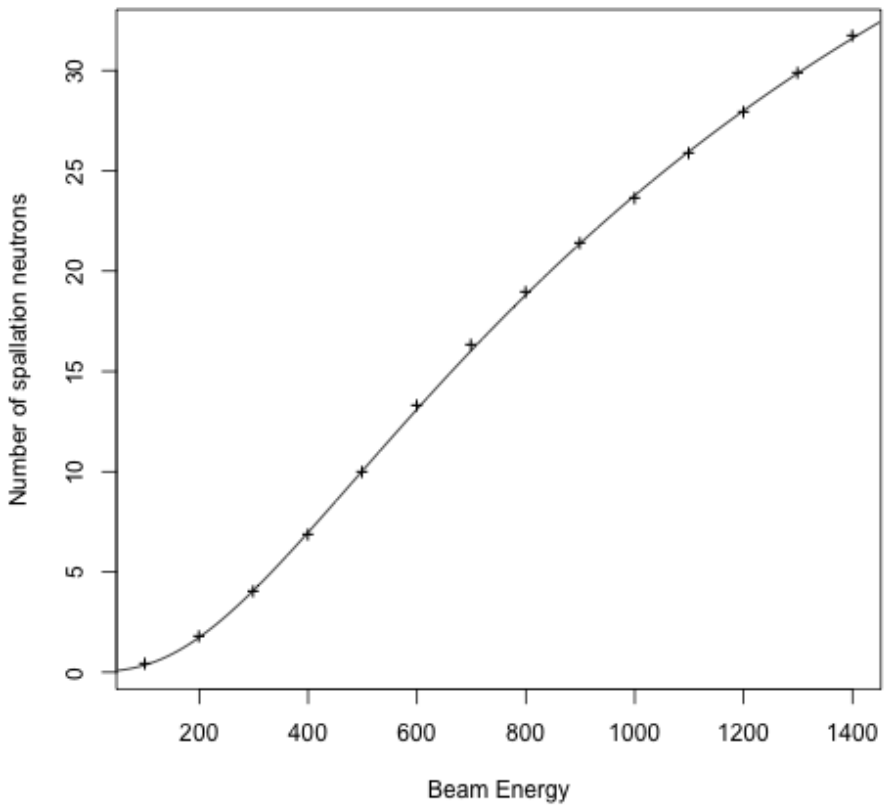
Example 1

Number of spallation neutrons from a lead target (60 cm long, 30 cm radius), as a function of beam energy. (MCNPX simulation: Geant4 similar)



Example 1

Number of spallation neutrons from a lead target (60 cm long, 30 cm radius), as a function of beam energy. (MCNPX simulation: Geant4 similar)

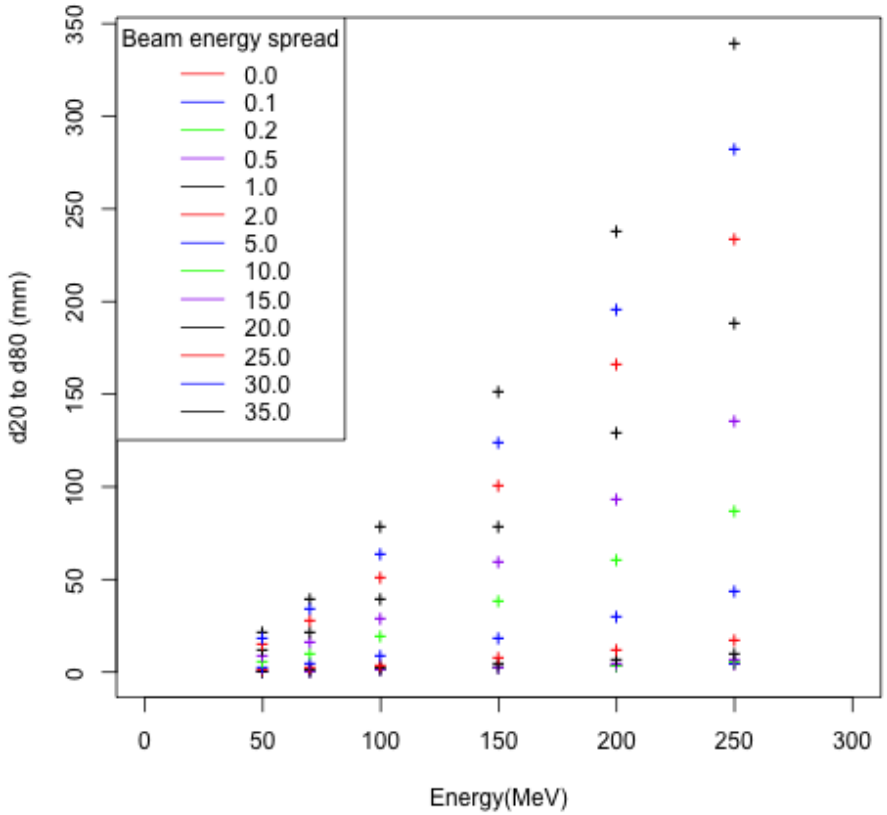


Function is $\frac{b}{2} e^{\frac{a}{E+b-e^2}}$ with $a = -1238.91, b = 141.713$

Example 2

Data from simulations by Tim Fulcher

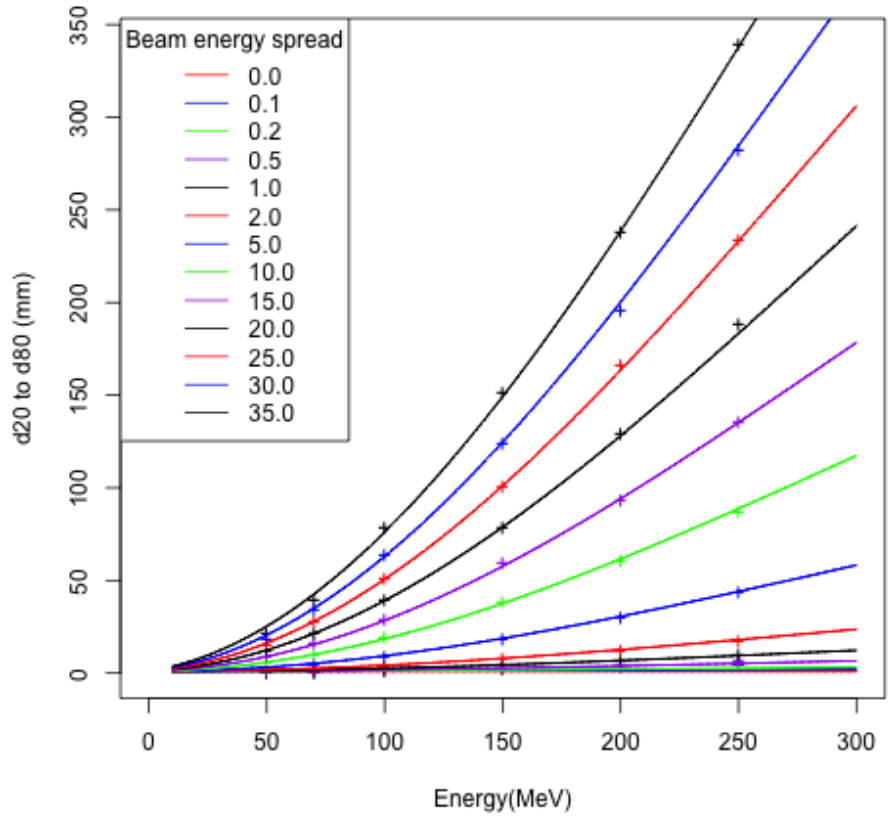
Distal shape of Bragg peak ($Y_{80} - Y_{20}$) as function of beam energy E and initial energy spread P



Example 2

Data from simulations by Tim Fulcher

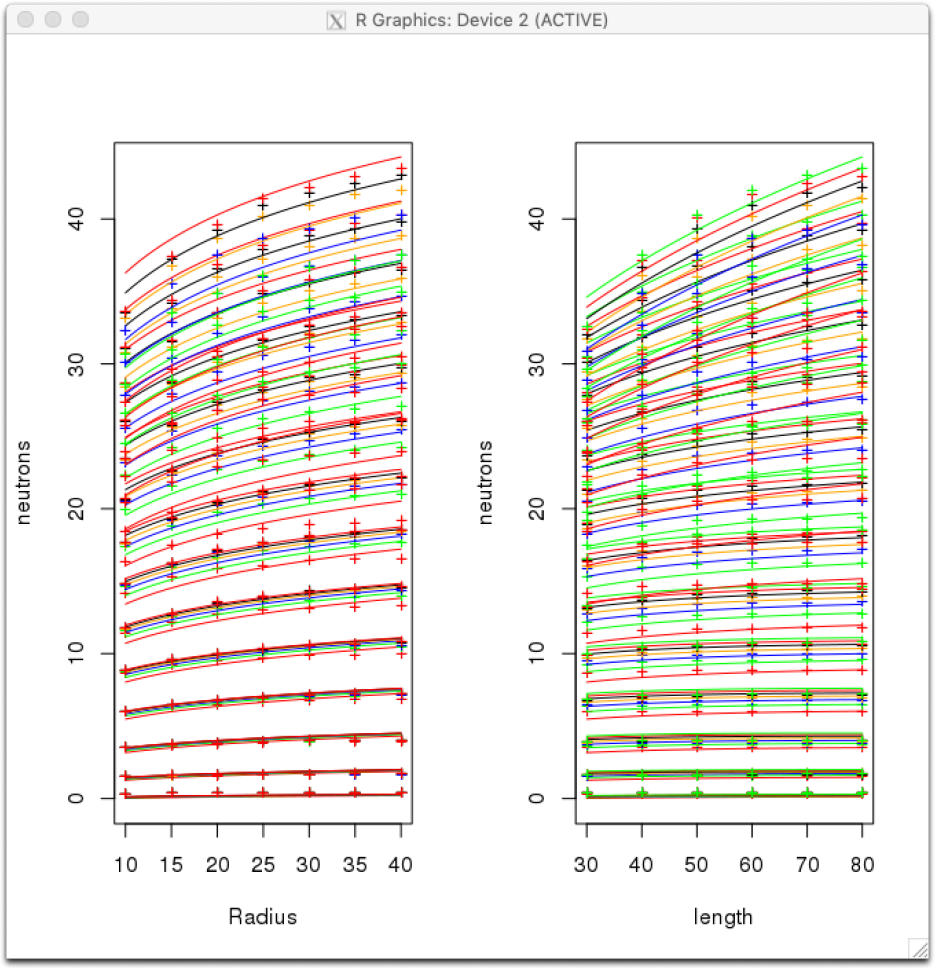
Distal shape of Bragg peak ($Y_{80} - Y_{20}$) as function of beam energy E and initial energy spread P



Function is $a^2 \frac{E(E-1)+e^{2b}}{E+P}$ with $a = 0.145241, b = 6.28564$

Example 3

Number of spallation neutrons as function of beam energy, target radius and length

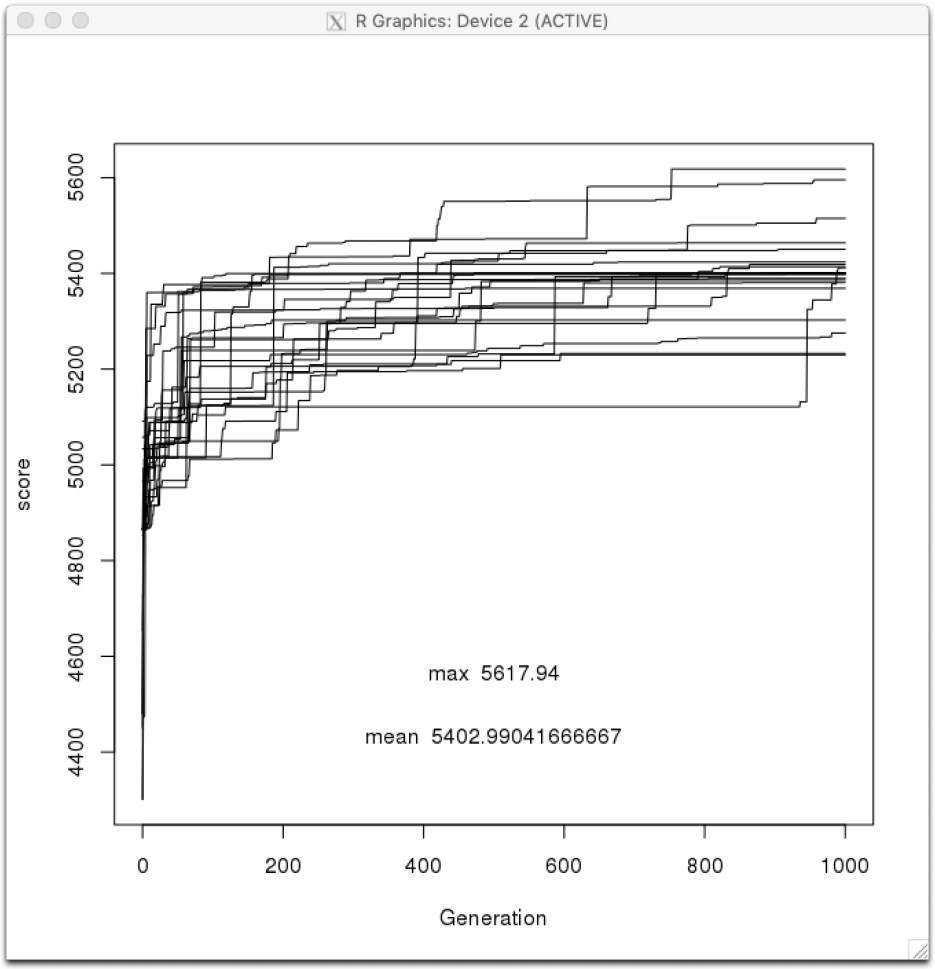


Function is $Ec(a + Elog(LR) * (c - b\frac{c}{\frac{1}{\log(E)}(d-(2+(b-aE+L+b)))})$ with

$a = -0.56, b = 192.4, c = -2.017.d = 156.9$

Example 3

That's for 1000 generations, took about a day

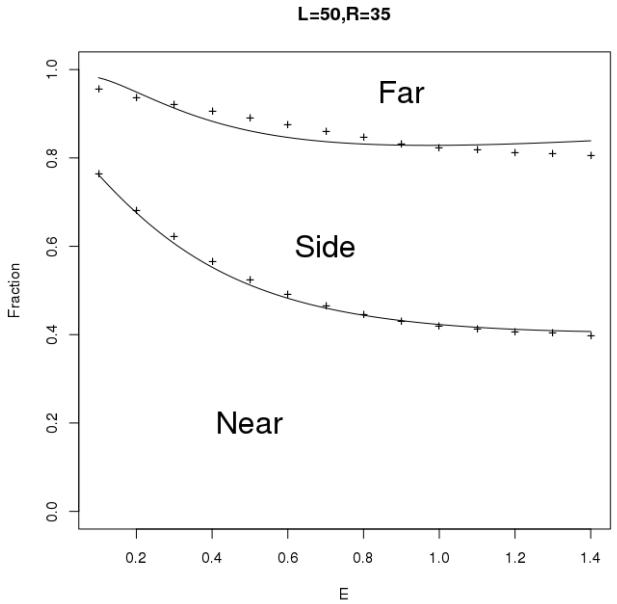


Achieves score 5618 - ideal would be 5880 - so more improvement possible

Example 4

Where do they come from?

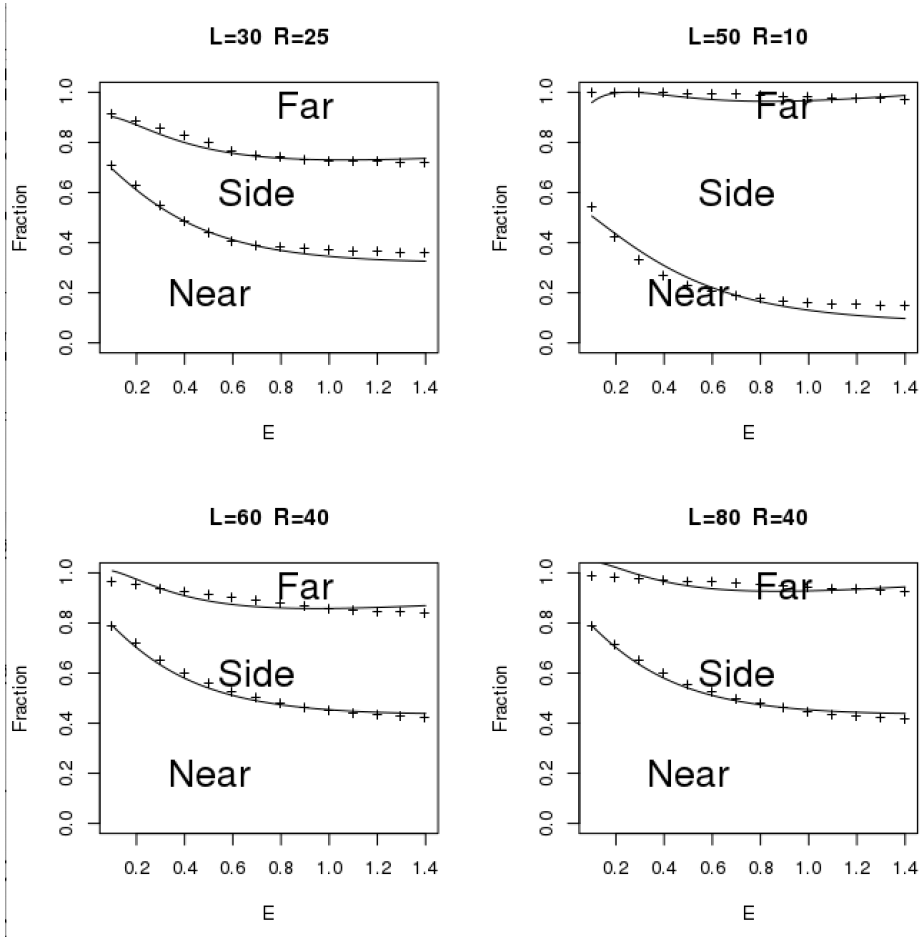
Spallation neutrons come from near end of target, side, or far end
Fit fractions. near/total and side/total as functions of energy, target radius, target length
Resulting functions too horrible to transcribe...



(Not perfect, but this is just 1 of 42 plots)

Example 4

Here are 4 more



Frankenstein - A hybrid of numeric & biological techniques

https://github.com/RogerJBarlow/Frankenstein

```
#include "Frankenstein.h"
void main(){
Frankenstein f(20,1,{1,2},{&Exp,&sin},{&add,&sub,&mul,&div},2);
f.setProbabilities(2.0,0.1,0.1,.1,.1);
f.givederivatives({&Exp,&cos},{&addD,&subD,&mulD,&divD});
f.forPrinting({"x","1","2","a","b","exp","sin","+","-","*","/"});
double X[] = {1,2,3,4,5,6,7,8,9,10};
double N[]= { .4,1.7,4.0,6.8,10.0,13.2,16.3,18.,21.,23.};
for(int j=0;j<10;j++) f.addtarget(N[j],vector<double> {X[j]});
#pragma omp parallel for
for (int itry=0; itry<24;itry++){
 biology G(f,itry);
 G.populate(1000);
 for (int i=0;i<=500;i++) G.mutate();
 cout<<"CORE "<<itry<<" best " << *(G.pop[G.ibest])
   <<" score "<<G.bestscore<<" parameters ";
 for(int iii=0;iii<G.Nmin;iii++)
   cout<<G.pop[G.ibest]->pmin[iii]<<" ";
cout<<endl;
} // end of cores loop
} // end of main program
```

Differences

Between FRANKENSTEIN and GeneXProTools, the industry standard

Plus points

1. Incorporates numerical minimisation
2. Open Source and free to download

Minus points

1. No triadic functions (if A then B else C...)
2. Single genes as opposed to multi-gene chromosomes

More to explore

Adaptive strategy - change tuning parameters as evolution progresses?

Recognise lack of biodiversity and take action?

Recognise lack of progress and take action? (Asteroid strike)

Restrictive strategy - insist programs contain all variables and adjustable parameters?

Should we try and shorten programs?

Gain experience

Bring me your data and we'll fit it!