Problem.1

Part a)

We could write out the tree as:
$(\mathrm{x}+((\mathrm{x}+(((\mathrm{x}+(\cos(\mathrm{x}-\mathrm{x})-(\mathrm{x}-\mathrm{x})))*\mathrm{x})*\mathrm{x}))*\mathrm{x}))$

Which simplifies to:

$(\mathrm{x}+((\mathrm{x}+(((\mathrm{x}+1)*\mathrm{x})*\mathrm{x}))*\mathrm{x}))$

$=\mathrm{x}+(\mathrm{x}^3+\mathrm{x}^2+\mathrm{x})*\mathrm{x}$

$=\mathrm{x}^4+\mathrm{x}^3+\mathrm{x}^2+\mathrm{x}$

This confirms with the $f(x)$ required.

Part b)

Yes there are unnecessary terms. For example, the $\cos(\mathrm{x}-\mathrm{x})-(\mathrm{x}-\mathrm{x})$ term could be simplified into $\cos(\mathrm{x}-\mathrm{x})$ directly, since $\mathrm{x}-\mathrm{x}=0$. If we allow constants(namely, 1 in this case) in the terminal set, we could further simplify this $\cos(x-x)$ into constant terminal 1.

Problem.2

Part a)

Attached is the table:

Objective:	Find $pred(x)$ that minimizes fitness value.
Terminal set	{X1, 1}
Function set:	{plus, minus, times, mydivide}
Fitness cases:	21
Raw fitness:	$\sum_{j=1}^{21} \lvert pred(x_j)-f(x_j)\rvert$, where $f(x_j)=\frac{x_j^2}{2}+2x_j+2$
Standardized fitness:	We can use the same as Raw fitness, as it satisfies the standardization requirements already.
Hits:	[100 1]
Wrapper:	

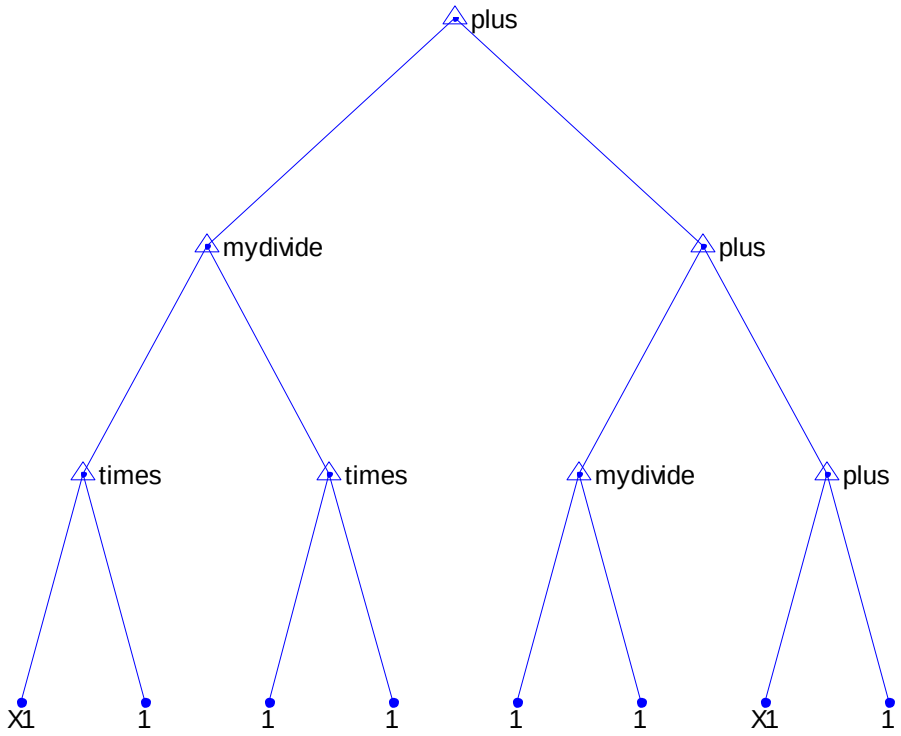
Parameters:	`p = resetparams;`
	`p = setfunctions(p, 'plus', 2, 'minus', 2, 'times', 2, 'mydivide', 2);`
	`p = setoperators(p,'crossover',2,2,'mutation',1,1);`
	`p.initialfixedprobs = [0.9, 0.0];`
	`p.operatorprobstype = 'fixed';`
	`p = addterminals(p, '1');`

	p.hits = [100 1];
	p.savetofile = 'every10';
	p.savename = 'result.txt';
	p.savedir = 'c:\\temp';
	
	p.reproduction = 0.1;
	p.minprob = 0.0;
	
	p.datafilex='x.txt';
	p.datafiley='y.txt';

Success predicate:	Given by Hits, we specify the success predicate as 100% of the fitness cases falls within 1% of the desired value. This automatically indicates we have found the exact solution to the problem. (the 1% is to get rid of precision)

Part b)

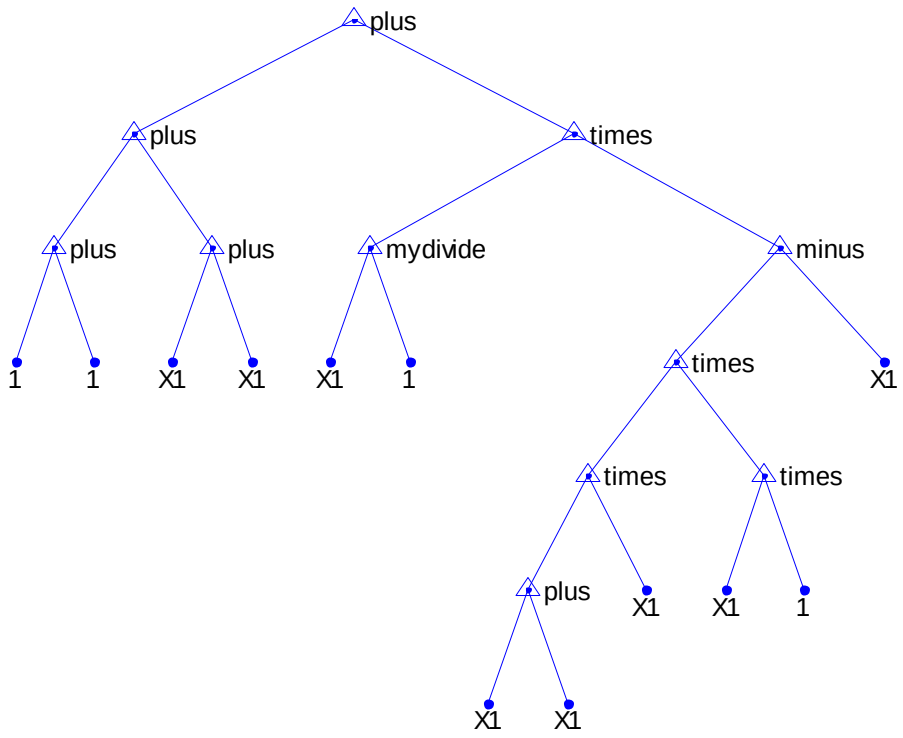
For generation 0, we have the fittest member as:



It has fitness value of 3.85, depth of 4, and 15 nodes. We also notice that this tree corresponds to the function $f(x) = 2x + 2$

Part c)

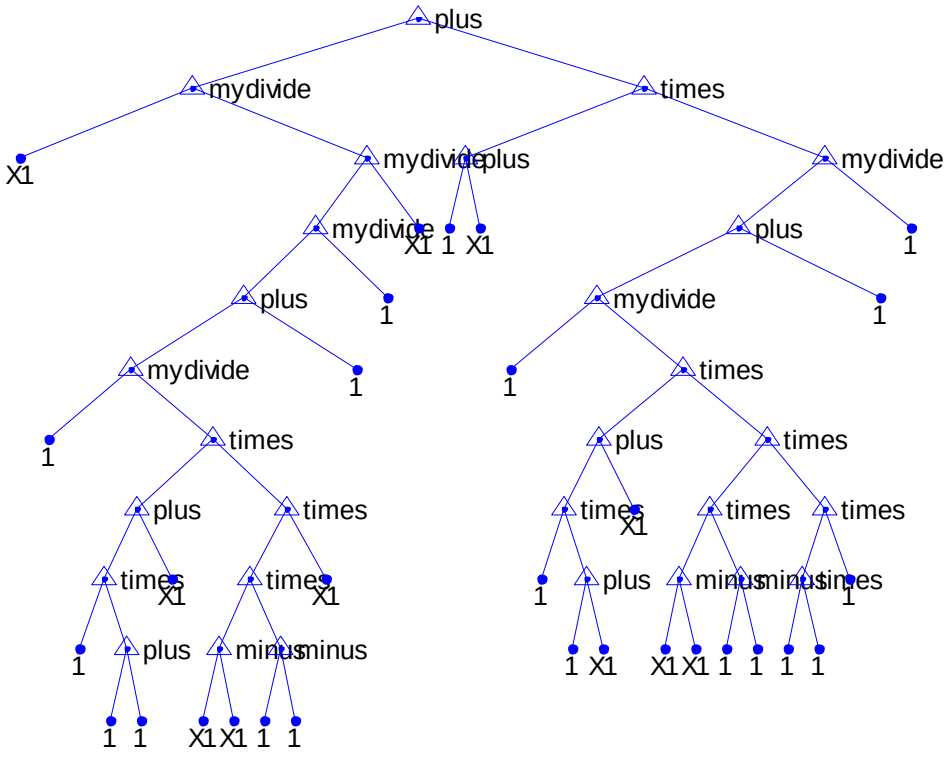
For example, we could take a look at generation 5. It has fitness value of 3.805, 23 nodes, and 7 levels.



We notice that $(x) = 2 + 2x + 2x^4 - x^2$. Although the fitness value does not decrease too much, we started to see some complicated structures in the tree, which is more aligned with the optimal solution. We also see the constant 2 there, which is a good part towards the optimal solution(that sub-tree could then be crossovered and produce the solution).

Part d)

We have the following tree:



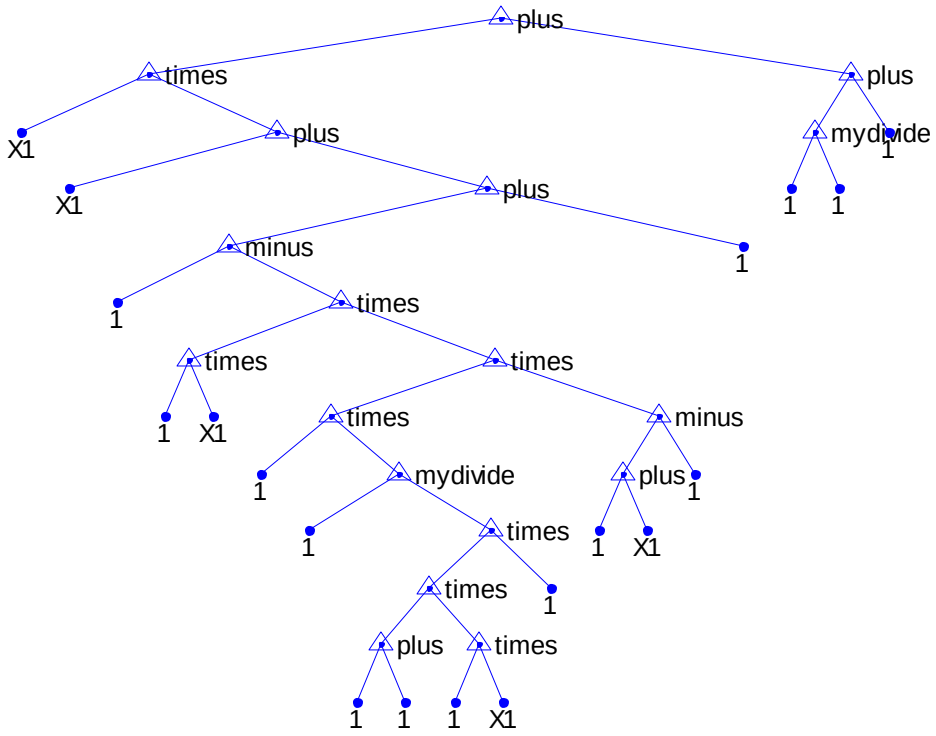
This tree has 59 nodes and 11 levels, and a fitness value of 0, indicating that this might be the algebraically correct solution. After simplifications, we confirmed that this is indeed the exact solution to the problem. This tree happens on generation 10, and the program stopped at our hits conditions.

Part e)

This is the algebraically correction solution to the problem.

Part f)

We have re-run the algorithm with population 1000 and the solution generated looks like:



Which has the fitness value of 0, level of 13, and node count as 37. This individual actually comes from generation 18, and it hits our hit condition, which means that it is indeed an algebraically correct solution.

Problem.3

Part a)

In the question statement it mentioned that situations at location 116 and 136 that present in Los Altos trail that do not occur in Sante Fe trail. That is, the Sante Fe Trail, if the ant does not see a food in front of it, or to its side, it will move forward and check again. At location 116 and 136, the ant sees no food in front or to its side. In this case, the ant will choose to move forward. However, the optimal choice is to actually move to the left. Once the ant moves forward at position 116 and 136, it will never go on the trail again.

Part b)

Now the ant needs to take care of the problem seen at position 116 and 136.

The new algorithm will be as follows:

START) At each step, the ant first

1) LEFT and checks for food
   a. If yes: MOVE and program goes to **START**
   b. If no: MOVE
      i. if left, right or forward has food, (possibly turn and) MOVE there and program goes to **START**
      ii. if not, MOVE
         1. if left, right or forward has food, (possibly turn and) MOVE there and program goes to **START**
         2. MOVE back to original position at this step (**step 1**). This can be done by turning LEFT twice and MOVE twice.

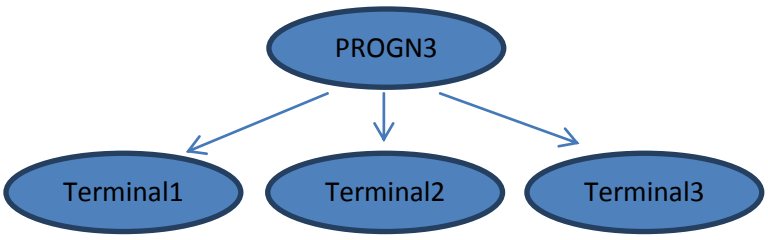
2) RIGHT and checks for food
   a. If yes: MOVE and program goes to **START**
   b. If no: MOVE
      i. if left, right or forward has food, (possibly turn and) MOVE there and program goes to **START**
      ii. if not, MOVE
         1. if left, right or forward has food, (possibly turn and) MOVE there and program goes to **START**
         2. MOVE back to original position at this step (**step 1**). This can be done by turning LEFT twice and MOVE twice.

3) MOVE and checks for food
   a. If yes: MOVE and program goes to **START**
   b. If no: MOVE
      i. if left, right or forward has food, (possibly turn and) MOVE there and program goes to **START**
      ii. if not, MOVE and program goes to **START**

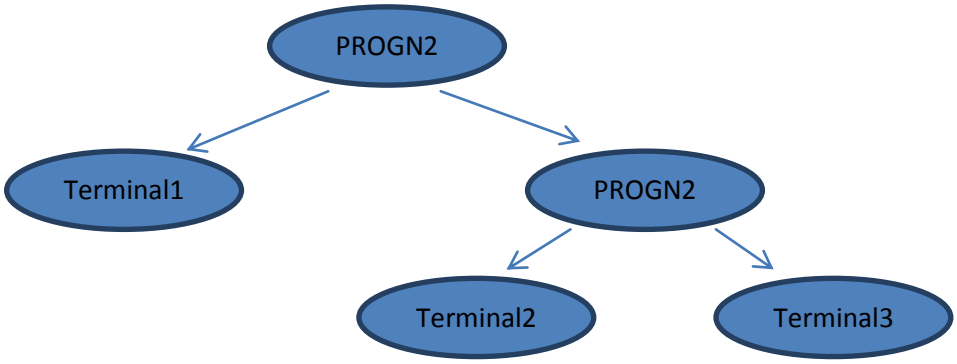
The terminals are all still the same: MOVE, LEFT, RIGHT, since the ant only needs these terminals to execute the algorithm described above.

The functions include: IF-FOOD-AHEAD and PROGNm, since the ant only needs these terminals to execute the algorithm described above. However, since the tree is significantly deeper and more complex now, there will be many more functions.

With regards to which PROGNm we need, we note that PROGN2 is all we need for any types of PROGNm functions (for m >=2), because we can get any PROGNm function by chaining together PROGN2. For example:



is equivalent to:



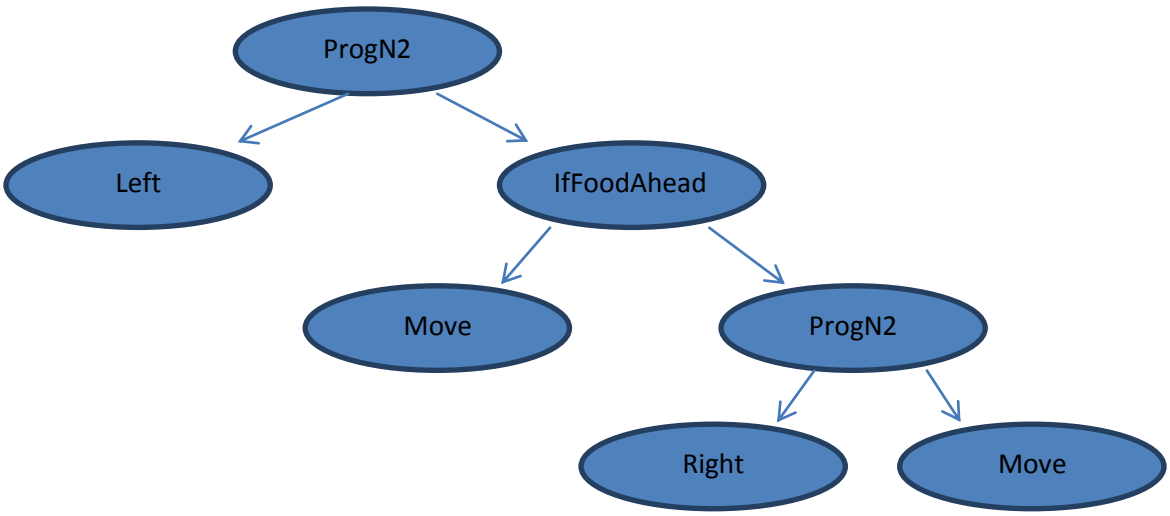
Thus, we have a more complicated program with same set of terminals but with higher depth.

Problem.4

Now our program will be:

```
ProgN2
   Turn left
   IF food ahead THEN
      Forward
   ELSE
      ProgN2
         Turn right
         Forward
```

The graphical representation looks like:



Problem.5

Note: For this question, we assume that the ant starts on a tile with food. Since it's hard to tell if the first box is grey or not, please subtract one from our answers if the first tile is not food.

Also, the coordinates used are in form of (row, column).

Part a)

The ant first moves to (1,2), sensing there's no food, moves to (1,3), etc. During this whole time, there will never be food ahead of it, so it will just keep moving forwards (towards the right). The ant will eat exactly 1 food(starting tile).

Part b)

There is no food ahead, so it will turn right. There is still no food ahead, so it will turn left (facing right again). Then it will move to (1,2)

There is no food ahead, so it will turn right. There is food ahead, so it will move twice. Now it is at (3,2)

There is no food ahead, so it will turn right. There is still no food ahead, so it will turn left (facing down again). Then it will move to (4,2).

There is no food ahead, so it will turn right. There is still no food ahead, so it will turn left (facing down again). Then it will move to (5,2).

Since there is no more food to the right and straight of the ant's path, it will just go down until the time is up.

The ant will eat 3 food in the path(starting tile, and then (2,2) and (3,2)).

Problem.6

The answers to part a, b, and c are embedded in this table:

	a) Number of children generated	b) Number of children for which expensive function evaluation is done	c) Whether a response surface is used
(32,200)-ES	200	200	No
(91,32,200)-ES-SLHD	200	200	No
(91,32,200,80)-ESQR	200	80	Yes
(91,32,200,80)-ESRBF	200	80	Yes
(91,32,80)-ES-SLHD	80	80	No
(32,80)-ES	80	80	No

From the table above we see it is indeed computationally advantageous to use surrogate response surfaces coupled to a heuristic.

Out of the 6 algorithms, only 2 of them (ESQR and ESRBF) use surrogate response surfaces. We note that those 2 algorithms do better than the other algorithms pretty consistently (after 100 evaluations). Although, the other algorithms catch up in performance to ESQR at 1000 evaluations, they are not even close to catching up to ESRBF.

ES, in particular, does the worst after 1000 evaluations. This suggests that SLHD does help. In other words, part of ESQR and ESRBF's success is from the surrogate response surfaces, and part of their success is from the SLHD.

However, we also note that this is only one specific problem with some specific parameters. We cannot conclusively say that surrogate response surfaces are helpful, but it certainly suggests it!