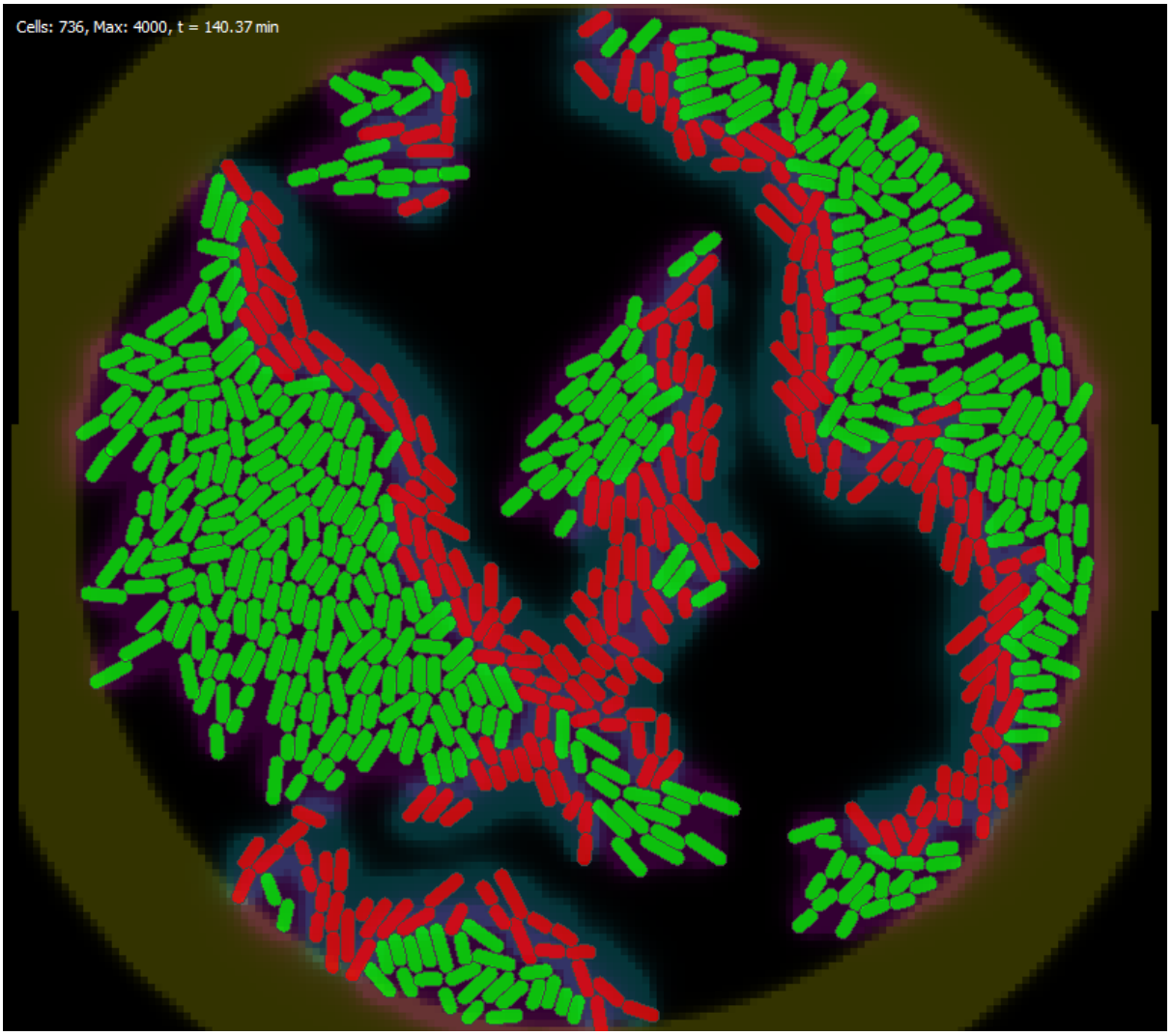
Alejandro Astudillo Aguiar
12447368K
190168

Introduction to Biocomputing Final Assignment

A visually engaging predator-prey bacterial simulation with GRO



Introduction

Gro [1] is a language used to program, simulate and manage artificial colonies of bacteria. One can program a great variety of behaviors ranging from simple to complex, modeling the exact transcriptions of genes into proteins and signals as one would encounter in nature (in a great level of detail thanks to the LIA version of the language [2]), or just simulating more simple interactions between the bacteria with a lesser degree of complexity behind the scenes.

For this specific simulation, I chose to loosely emulate the prey-predator system for E. Coli engineered by researchers at Stanford [3], where they genetically engineered two types of E. Coli, a predator and a prey. The prey would produce a Quorum Sensing signal that kept the predator bacteria alive, while in turn the predator bacteria would produce a killing signal that would cause the death of the prey. This is the behavior I have chosen to reproduce, without delving into the real complexities of the problem, which involves a more complex genetically based approach. It is important to note that this more complex approach **could** have been tackled with gro, but I decided not to in order to keep the time and extension dedicated to this assignment reasonable.

It is because of this that results should not be expected to be realistic, but instead aim to achieve visually impactful results, as well as an oscillating system that will not die out.

The results Ihave achieved are very visually striking, where the two populations of bacteria dance all around the screen, chasing each other in a way that likens to a piece of paper or fabric catching on fire, as can be seen on **Figure 1.**

Code

The full code file can be found alongside this document, as well as all the graphs and shown.

Initial adjustments

```
set ( "dt", 0.01 );
set ( "population_max", 4000  );
set_theme ( dark_theme );

//Bacteria count for graphs
total_prey := 1;
total_pred := 1;
```

First we set the time step to be 0.01, a value I found acceptable as enough to make the bacteria able to have meaningful interactions and not be too slow. Then we set the max population to 4000, although the simulation never reaches any more than 1000 cells. We set

the dark theme for added aesthetic value. Then we initialize global variables to count the number of bacteria of each type in order to graph them later.

Signal definition

```
//Signals definition
ahl6 := signal ( 2, 1.6 );
ahl12 := signal ( 1.3, 2.3 );
bounds := signal (0.01, 0);
```

Next, we define each signal. These are ahl6 and ahl12, named roughly after the QS signals used in the real-life paper [3]. Ahl6 is the antidote signal, released by prey. Ahl12 is the killer signal, released by the predators. The diffusion and degradation rates have been carefully tweaked to obtain the best possible results, achieving a balance between the antidote degrading and spreading fast enough to keep the predators alive just the right amount of time, and the killer signal not spreading too much as to not kill too many prey, only those in close proximity.

The third signal is the “bounds” signal, which I use to make a circle around the colonies, killing them if they touch it. This is in order to contain the colonies, confining the action to a restricted space.

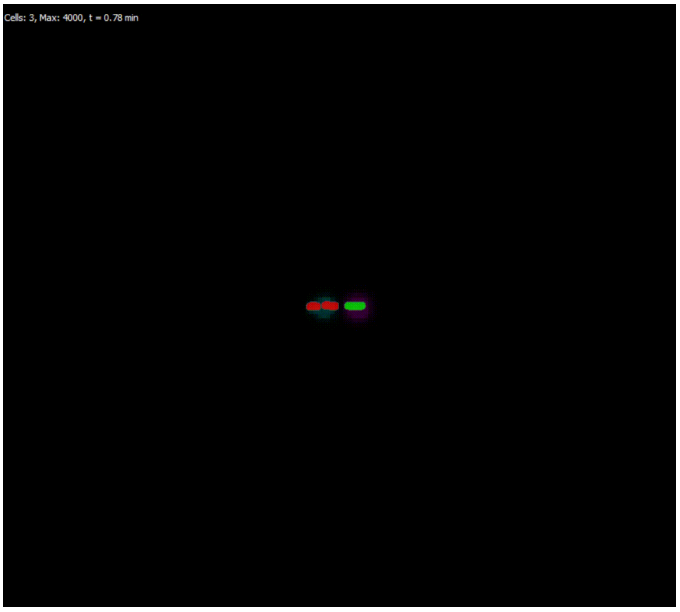


Figure 1: Prey in green, predators in red. 150 in-simulation minutes show the two colonies interacting with each other in a visually significant way. Usual prey-predator patterns can be observed, such as the oscillation between population sizes: when too many prey bacteria reproduce, more predator bacteria reproduce in turn and this dwindles the prey population, decreasing again the predator population. As we can see, some very interesting patterns emerge, akin to a fire burning smoke or paper.

Prey program

```
program prey () := {
```

```
    set ( "ecoli_growth_rate", 0.75  );
    gfp := 50;
    //Constantly produce gfp to distinguish the prey
    true : {gfp := gfp + 1};

    //Emits antidote signal
    true: {emit_signal ( ahl6, 1.7 )};

    //Dies if recieves killer signal
    get_signal ( ahl12 ) > 0.5 :
    {total_prey := total_prey - 1, die()};

    //Stops growing if there are too many prey
    get_signal ( ahl6 ) > 18 :
    {set ( "ecoli_growth_rate", 0  ), gfp := 40 };

    //Dies in contact with "bounds" signal:
    (get_signal ( bound ) > 0.3) : {total_prey := total_prey - 1, die()};
    daughter : {total_prey := total_prey + 1};
};
```

This is the program the prey will execute whenever it divides or is spawned. First we set its growth rate. I set it to a pretty big value, so that prey have a chance to “escape” the predators and move across the screen. Then we set the gfp to max value and constantly increase it so as to keep the prey cells green.

Next is emitting the antidote signal. It constantly emits the signal, at the concentration that I determined, after extensive testing, would yield the best results. Then we check for presence of the killer signal, and if it is above a certain threshold, the cell dies.

The next line is incredibly important to the behavior of this system, and crucial to the way the bacteria move around each other. I made it so that if the prey detects too much ahl6 molecule, that is, if there are too many prey bacteria around, it will stop growing altogether. This makes it so that only prey in the edges of the colony will keep growing, which gives that really engaging effect of the prey “running away”.

When in contact with the bounds signal, the prey will die. When it dies, we subtract one to the prey tally. When a new cell is born, we add one to the prey tally.

Predator program

```
program predator () := {
    p := [ t := 0 ];
    p.t := p.t + dt;
    set ( "ecoli_growth_rate", 0.6 );

    rfp := 50;
    //Constantly produce rfp to distinguish the predator
    true: {rfp := rfp + 1};
    //Emits killer signal
```

```
    true: {emit_signal ( ahl12, 1.6 )};

    //Dies without antidote signal
    (get_signal ( ahl6 ) < 0.2  ) & (p.t > 4) : {total_pred := total_pred - 1,
die()};

    //Dies in contact with "bounds" signal:
    (get_signal ( bounds ) > 0.5) : {total_pred := total_pred - 1, die()};
    true : { p.t := p.t + dt }

    daughter : {total_pred := total_pred + 1};
};
```

The predator program is similar to the prey program. The predator produces rfp to distinguish itself from the green prey, and has a slightly lower growth rate. It emits the killer signal, and dies if it doesn't receive a fixed amount of the antidote molecule. I had to add a time check to prevent predators from dying too early, before they have had a chance to encounter any prey.

Main program

```
rt := 0; // real time tracker

program main() := {

  a := 0;
  t := 0;
  t := t + dt;


  t < 3 : {

    a := a + 0.25*dt,

    set_signal ( bounds,
      400*sin((3)*a+6.28/5.0),
      400*cos((3)*a+6.28/5.0),
      1000 )

  }

  t := 0; // framerate time tracker
    s := 0; // total time tracker
    n := 0;
    true : {
        rt := rt + dt;
        t := t + dt,
        s := s + dt
    }
    t > 1 : {
```

```
        print ( rt, ", ", total_prey, ", ", total_pred, "\n" );
        snapshot ( ".\movie\preydator" <> if n <10 then "0" else "" end <>
tostring(n) <> ".tif" ),
        n := n + 1,
        t := 0
    }
    s > 500 : {
        stop()
    }

};
```

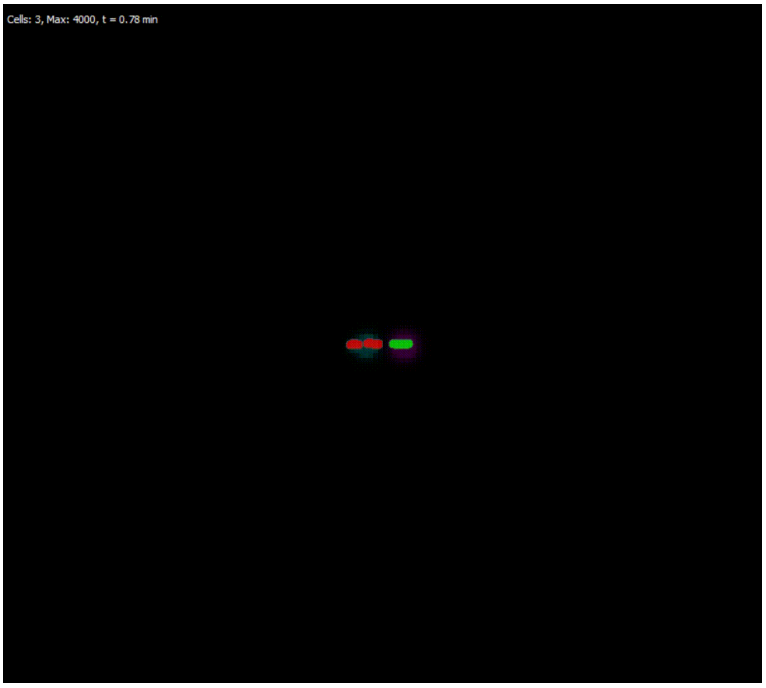
In this program we do three things:

1. We set the bounds. We do this by defining an angle *a* and increasing it every timestep, as well as placing the signal every 3 timesteps along a circle. We use basic trigonometry [4] to do this.
2. We take a snapshot every minute, in order to make the gifs and videos included in this document
3. We print each type of cell's current population size in order to create the graphs seen in the next section.

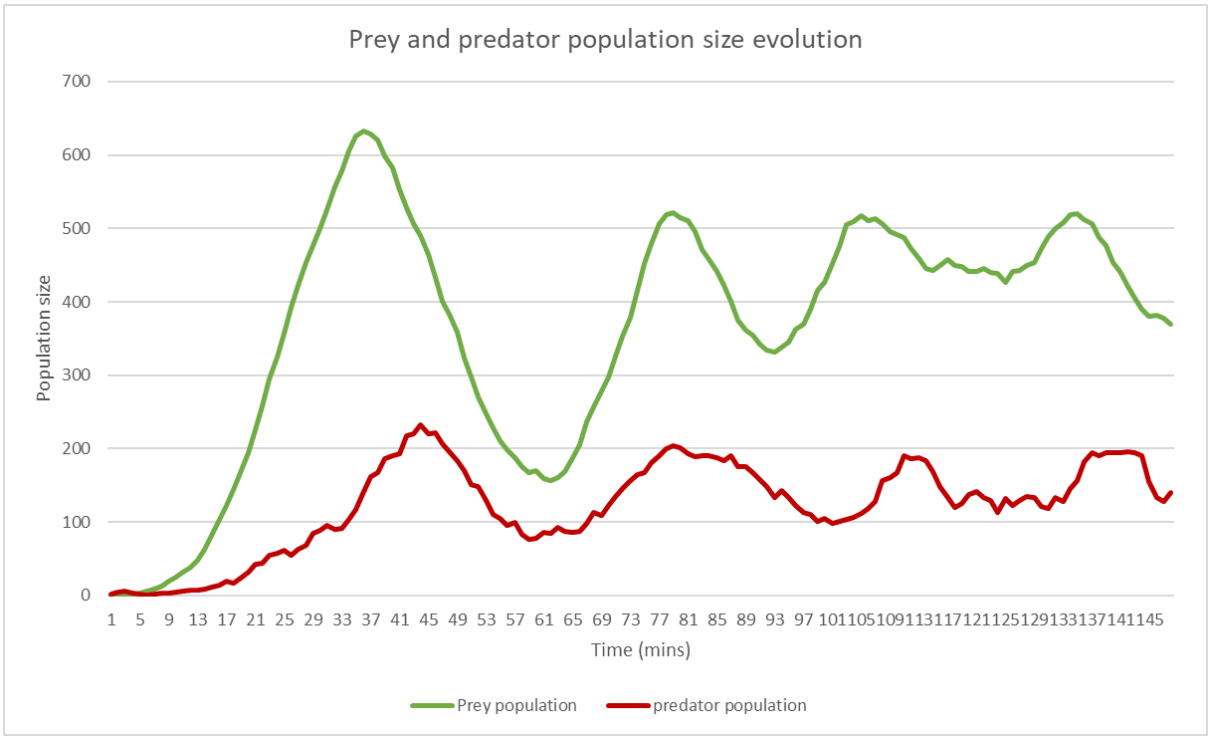
Results

A shorter simulation

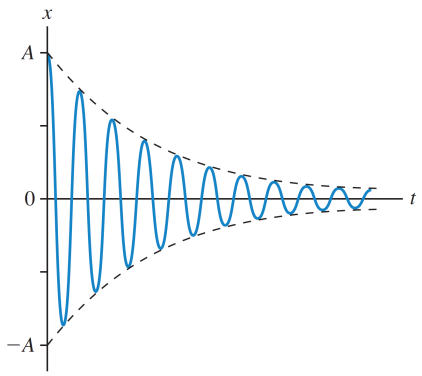
This simulation is 150 minutes of simulated time:



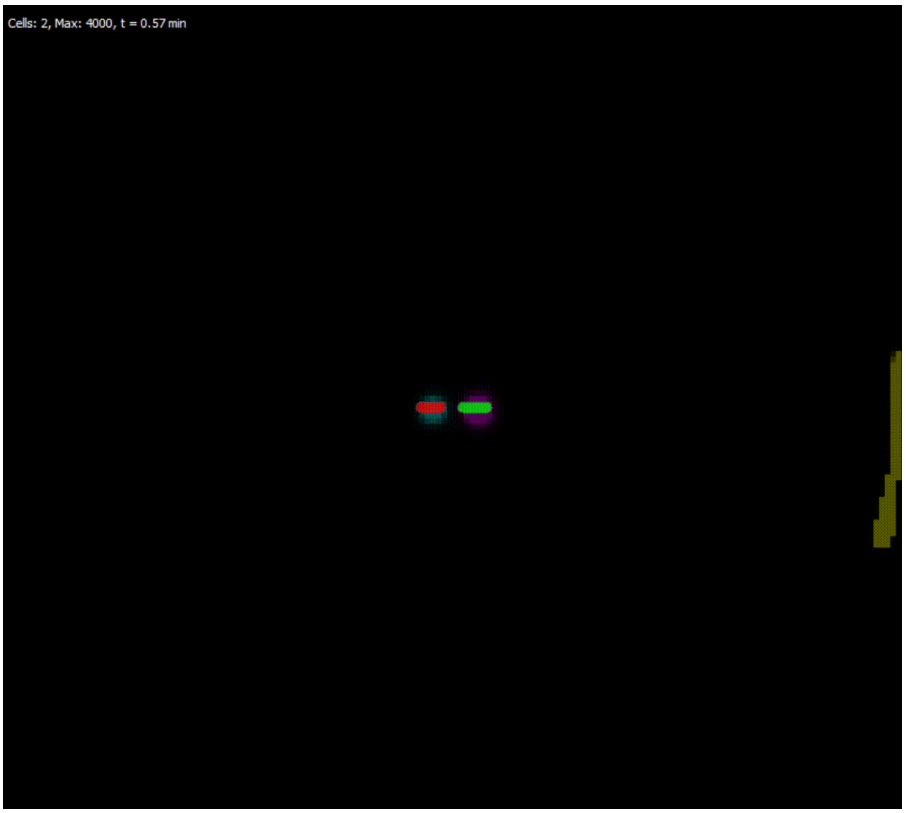
As you can see, the prey narrowly escape multiple times, creating some really interesting spiral movement patterns. This particular simulation's population size data graphed is:



We can very clearly observe the usual oscillations characteristic of prey-predator models. I also find it interesting that the graph loosely resembles a damped sine wave:

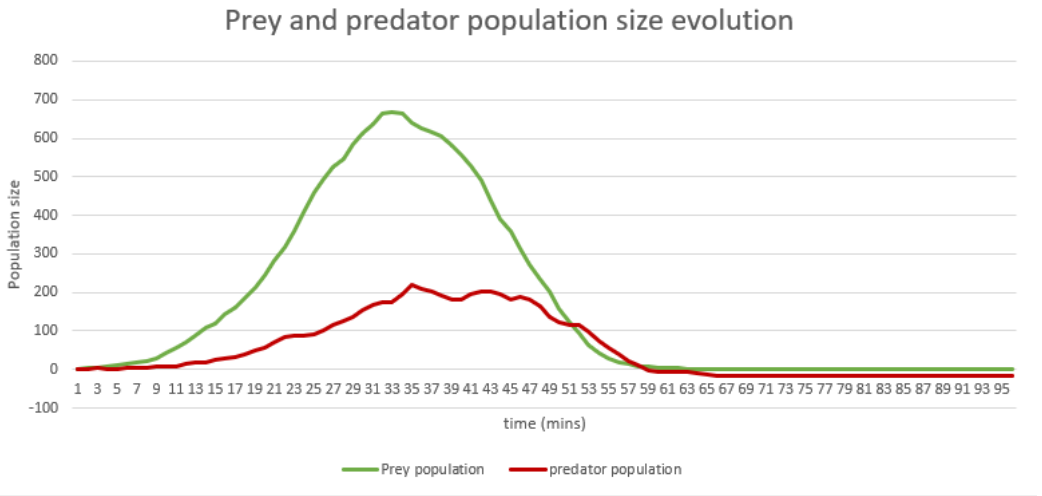


Another scenario:



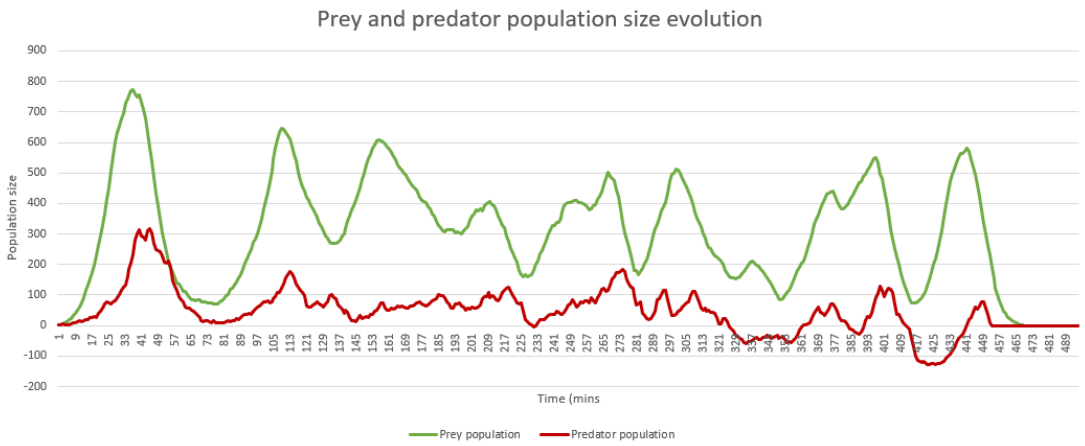
As we can observe, in some scenarios the predators will form a circle against the prey, and push them against the barrier. This eventually spells out their own doom too, and this is the problem with having a killing barrier. Physical barriers would have been more appropriate for this problem.

When we graph the data from this simulation, it looks like this:



Longer simulation:

This simulation runs for 500 simulated minutes. As we can see, the predators eventually enclose the prey and kill them off. I think this is inevitable in the long run, and perhaps some adjustments could be made to the model to ensure the prey always survive, to make the cycle infinite.



Final thoughts

This was a very engaging project and I'm glad I chose to do a gro simulation, even though I had no experience at all with the language. Gro is a great tool, especially once glaring performance issues are fixed by the LIA version. I am left wanting to try some of the more advanced features with gene expression, protein synthesis, etc. and make a more realistic simulation. However, I am happy with how this turned out, and I think I achieved some great results. If anything, I would have liked to be a little more original with the concept of my simulation.

I made the graphs using Excel and made the videos and GIFs using Blender free software.

Bibliography

1. *Gro* programming language: https://depts.washington.edu/soslab/gro/
2. Gutiérrez, M. *et al.* (2016) "A new improved and extended version of the multicell bacterial simulator gro." Available at: https://doi.org/10.1101/097444.
3. Balagaddé, F.K. *et al.* (2008) "A synthetic *escherichia coli* predator–prey ecosystem," *Molecular Systems Biology*, 4(1), p. 187. Available at: https://doi.org/10.1038/msb.2008.24.
4. Sterling, M.J. (2014) *Trigonometry for dummies*. Hoboken, NJ: John Wiley & Sons.