Project 2 – Differential Equations

Time-dependent Ordinary Differential Equations (ODEs)

The equation for logistic population growth from Project 1 is.

$$\frac{dN}{dt} = rN(1 - N / N_{\max}) - vN \qquad (1)$$

We already saw that the exact solution is

$$N(t) = \frac{Ke^{Rt}}{C + e^{Rt}} \qquad (2)$$

where $R = r - v$, and $K = N_{\max}(r - v)/r$.

There are many models for which we can write down ODEs. Often, there is no simple mathematical form of the solution, so we need to be able to find a numerical solution by computer.

Usually the equation to be solved can be written as

$$\frac{dN}{dt} = f(N,t),$$

meaning that the growth rate depends on the current population and the current time.

First-order method (Euler's method)

We want to estimate $N(t+\delta t)$ from the current value $N(t)$.

Let $V = \delta t . f(N,t)$

Let $N(t + \delta t) = N(t) + V$.

This is the simplest method. It assumes that the growth rate throughout the interval δt can be approximated by its value at the beginning of the interval. This is OK if δt is small enough, but the error becomes big if δt is too big.

Use the program **ODE1.nlogo**

Fig 1 shows the results when the program is run with growth rate $r = 5$ and δt = 0.2. The black curve shows the exact solution $N_{exact}(t)$. The green curve shows the approximate solution from the first-order method, $N_{app}(t)$. You can see that there is considerable error for this value of δt.

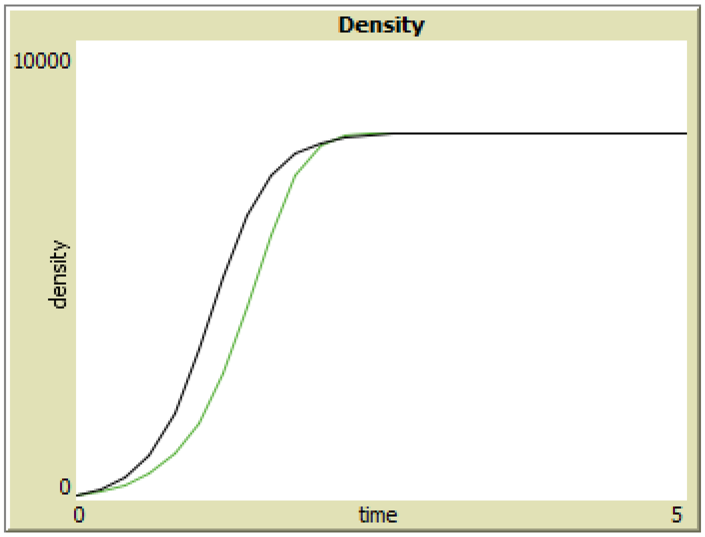


Fig. 1

Read the code in the ODE1.nlogo file and make sure you understand how the $N_{app}(t)$ is calculated. What happens to the green curve if you make δt smaller or larger (using the dt button on the interface)?

A good way to measure the size of the error is to calculate the root mean square error per point between the approximate and exact solutions.

$$E = \sqrt{\frac{1}{n_{pts}} \sum_{time-poins} (N_{app}(t) - N_{exact}(t))^2} \; .$$

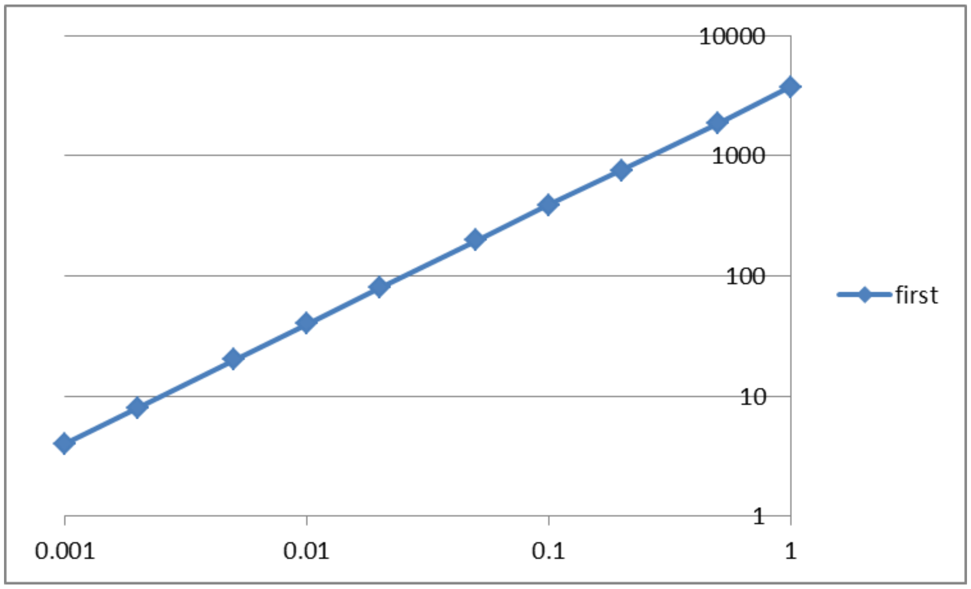
In this example the program stops at time t = 5, so the number of time points is $n_{pts} = 5/\delta t$. The variable rmserror in the program calculates E. When the program stops, it prints δt, N_{pts}, and E in the command centre at the bottom of the screen. In this example we get

δt = 0.2, N_{pts} = 25 and E = 750.

Make sure you understand how E is calculated.

Q1 – Keeping r = 5, and keeping the stop time at t = 5, measure E for each value of δt in the range 1 to 0.001. You should find that $E \sim \delta t$. This is a power law. Power laws are best shown on a log-log plot because they are straight lines.

You should get something like Fig 2.

Fig 2. 

Add a straight line whose slope is exactly 1 onto this graph. It should fit almost exactly through the points at the lower end of the δt range. This means that *E* is proportional to δt to the power 1, when δt is small. For this reason, the method is called a first order method.

Second-order Runge-Kutta method

This method makes a better approximation, whose error is proportional to δt^2. This is a much smaller error than the first-order method when δt is small.

Let $V_1 = \delta t . f(N,t)$

Let $V_2 = \delta t . f(N + V_1 \ , t + \delta t)$

Let $N(t+\delta t) = N(t) + \frac{1}{2}(V_1 + V_2)$.

Fourth-order Runge-Kutta method

This is an even better approximation whose error is proportional to δt^4.

Let $V_1 = \delta t . f(N,t)$

Let $V_2 = \delta t . f(N + \frac{1}{2} V_1 \ , t + \frac{1}{2}\delta t)$

Let $V_3 = \delta t . f(N + \frac{1}{2} V_2 \ , t + \frac{1}{2}\delta t)$

Let $V_4 = \delta t . f(N + V_3 \ , t + \delta t)$

Let $N(t+\delta t) = N(t) + \frac{1}{6}(V_1 + 2V_2 + 2V_3 + V_4)$.

We have assumed here that *f* may be a function of *t*. This is not the case in the logistic equation, but it might sometimes be the case in a more complicated model. For example, suppose the amount of food varies over the year in a sinusoidal manner, so that the growth rate depends on time: $r(t) = a + b \sin \omega t$

Now the right hand side of the ODE is explicitly dependent on *t*.

$$\frac{dN}{dt} = f(N,t) = (a + b \sin \omega t) N \left(1 - \frac{N}{N_{max}}\right) - vN$$

The method sections have been written so that they will work when the function depends on *t*, even though it does not depend on *t* in this example.

Q2 – Modify the ODE1.nlogo to implement the second-order and fourth-order methods. Calculate the error *E* in both methods over the range of δt and plot this on the same graph as Fig 2. You will see that the errors for the higher order methods are much smaller. Plot straight lines on the log-log plot to demonstrate that *E* scales as δt^2 for the second-order method and δt^4 for the fourth-order method.

A Real Example – Trypanosome Infections

Read the paper by Tyler *et al.* on modelling Trypanosome infections. Trypanosomes are single-celled eukaryotic parasites that cause sleeping sickness in mammal hosts. Experimental data was available to measure the number of parasite cells per ml of blood in infected mice. Two types of cells are measurable in the blood samples: slender cells, which multiply exponentially in the host, and stumpy cells, which no longer divide. There is a mechanism by which cells can sense the concentration of cells in the blood. The differentiation of slender to stumpy cells is initiated when the cell concentration becomes high. This stops the exponential growth, and prevents the parasite killing the host immediately.

Here, we will look at the model in section 3(a) of the paper, referred to as Model A. *X* and *Y* are the concentration of slender and stumpy cells. The differential equations for *X* and *Y* are:

$$\frac{dX}{dt} = (r - f)X$$

$$\frac{dY}{dt} = fX - mY$$

where the *f* is proportional to the total population size: $f = f_0(X + Y)$. The results depend on the parameters *r*, f_0, *m* and X_0.

Q3 – Write a brief description of the assumptions of model A, including a description of the meaning of each of the terms in the equations above, and a definition of each of the parameters.

Q4 - By inserting $f = f_0(X + Y)$ into the two equations, we can write

$$\frac{dX}{dt} = \left(r - f_0(X+Y)\right)X$$
$$\frac{dY}{dt} = f_0(X+Y)X - mY$$

Write a program that solves these equations using the fourth-order Runge-Kutta method. Use the parameters from mouse 1 in Table 1 of the paper. Show that your program gives results equivalent to Fig 3 in the paper.

Method: You will need to think carefully about the following things in order to get Q4 to work correctly. The general pair of coupled equations is

$$\frac{dX}{dt} = F_1(X,Y,t)$$
$$\frac{dY}{dt} = F_2(X,Y,t)$$

The first approximation for the change in X and Y is

$$VX_1 = \delta t.F_1(X,Y,t)$$
$$VY_1 = \delta t.F_2(X,Y,t)$$

It is necessary to calculate the first approximation for X and Y before we calculate the second approximation for X. You can't do all four X's before you do the Y's. We use VX_1 and VY_1 to calculate the second approximations

$$VX_2 = \delta t.F_1(X + \frac{1}{2}VX_1, Y + \frac{1}{2}VY_1, t + \frac{1}{2}\delta t)$$
$$VY_2 = \delta t.F_2(X + \frac{1}{2}VX_1, Y + \frac{1}{2}VY_1, t + \frac{1}{2}\delta t)$$

... and the third ....

$$VX_3 = \delta t.F_1(X + \frac{1}{2}VX_2, Y + \frac{1}{2}VY_2, t + \frac{1}{2}\delta t)$$
$$VY_3 = \delta t.F_2(X + \frac{1}{2}VX_2, Y + \frac{1}{2}VY_2, t + \frac{1}{2}\delta t)$$

... and the fourth ....

$$VX_4 = \delta t.F_1(X + VX_3, Y + VY_3, t + \delta t)$$
$$VY_4 = \delta t.F_2(X + VX_3, Y + VY_3, t + \delta t)$$

... and finally ....

$$X(t + \delta t) = X(t) + \frac{1}{6}(VX_1 + 2VX_2 + 2VX_3 + VX_4)$$
$$Y(t + \delta t) = Y(t) + \frac{1}{6}(VY_1 + 2VY_2 + 2VY_3 + VY_4)$$

Fitting a Model to Data

This section uses the program **quickfit.nlogo**

In this example, the data is the population N_{data}(t) measured at n_{pts} time points. Suppose we have a theory that says the population should be N_{th}(t). The root mean square error per point between the theory and the data is

$$E = \sqrt{\frac{1}{n_{pts}} \sum_{points} (N_{th}(t) - N_{data}(t))^2} \qquad (1)$$

We want to make the theory curve go close to the data points. The theory has certain parameters in it. The object of the quickfit program is to find the parameters that minimize E.

In this example, the data is made-up numbers that follow a roughly sigmoidal curve. These are in the array ndata defined in the setup function.

The theory is that the population should obey the logistic equation

$$\frac{dN}{dt} = RN(1 - N/K), \qquad (2)$$

We know the theoretical solution is

$$N_{th}(t) = \frac{K exp(Rt)}{C + \exp(Rt)} \qquad (3)$$

with $C = \frac{K}{N_{init}} - 1.$ (4)

In this case, we have three unknown parameters: N_{init}, K, and R.

The program starts with a guess at what these parameters might be:
N_{init} = 10, K = 300, and R = 2.
When you run setup, the program plots the data (green) and the theory curve (black) for these starting parameters. You can see that the fit is not very good, but it is at least in the right ballpark.

When you hit Go, the program starts to optimize the parameters. On each tick, it changes one parameter. If the new solution is better, we keep the new solution. If the new solution is worse, we go back to the old solution. The algorithm works like this.

- Save the old solution (K_{old}, R_{old}, $N_{initold}$) and the old error E_{old}.
- Change one parameter by a small amount. The program cycles through the three parameters repeatedly, using the remainder function to decide which parameter to change.

For example, $K = K_{old}$ + 10 * (random-float 1.0 – 0.5). This gives a change between -5 and +5.

- Calculate the error E.
- If E is less than E_{old}, keep the new parameters. Otherwise set the parameters back to the old parameters.
- Keep doing this until the error does not get any smaller for many interations.

At each time step, the program plots the current best solution on top of the data. This gradually gets better. This fit looks much better than what we started with, and the error is now much smaller.

Check the program to see how it is implemented in the code. Netlogo has a rather awkward way of dealing with array elements. It is not a great language for numerical calculations like this, but it works.

Q5 - Plot a graph with the following three curves together - (a) the data points, (b) the theory curve with the initial estimates of the parameters, (b) the theory curve with the final estimates of the parameters after 1000 iterations (ticks). It should be obvious that the final fit is much better.

Q6 - Plot a graph with the error as a function of the number of iterations. Is 1000 ticks enough for convergence? Does it make any difference if you run it for twice as long?
Note that the error does not go to zero, because the theory does not exactly fit the data. In reality, there will always be some statistical error or measurement error in the data, so the error will not be zero, even if the theory is a good theory that is essentially 'true'.

Q7 - Plot a graph with parameters K, R, N_{init} as a function of the number of iterations. A log scale would help to fit these on the same graph. You should be able to see that the parameters gradually move up and down as the error is minimized, until no further improvement is possible.

Q8 - use the following set of data instead of the data that is already in the program

```
  set ndata array:from-list [20 24 31 40 59 128 375 402 420 412 418]
```

Comment on the difference between this data set and the first one. Would you say this theory fits the data well?

Several points to be aware of:

- In this case, the theory points came from an exact solution of a differential equation. In other cases the theory might come from a numerical solution of a differential equation using Runge-Kutta (as in the Trypanosomes paper), or it might come from simulation of an agent based model (e.g. you could simulate the patch-growth model and fit the simulated results to the data). The last one would be slow, because we have to run a full simulation for every trial parameter set. Nevertheless, the method of optimizing the parameters would be the same.

- When the data covers a very wide range, it might be better to measure the error in lnN instead of N (as in the Trypanosomes example).
- This method assumes that the error varies smoothly as we change the parameters. The solution gradually slides down to the best fit. In more complex models with many parameters, there may be local minima in the error function, in which case the program will get stuck at parameters that may be far from the best parameters. To check that the program is finding a true minimum, it is best to run it several times starting from different initial parameters. If it converges to the same result each time, you can be reasonably sure that you have a good fit.

FILES TO SUBMIT
Please submit one pdf file with your report, containing graphs of results and answers to the questions. Also please submit one Netlogo file that contains your code for the Trypanosomes model using the 4th order ODE method.