Week 3

Simulating Dynamics

1 Birds and differential equations

On a certain deserted island in the Pacific Ocean, a certain species of bird initially has a population of 5271. There are no predators on this island, and we make the simplifying assumption that the only way the population can change is through births or deaths. It is known that the rate of births at any given time is proportional to the population at that time and is equal to 17 births per year per 1000 birds. It is also known that the rate of deaths at any given time is proportional to the current population and is equal to 32 deaths per year per 1000 birds. We want to know what the population of birds will be as a function of time. We model this as the following differential equation initial value problem:

$$\dot{P}(t) = (B - D)P(t), \qquad P_0 = P(t_0) \tag{1}$$

where

$$\begin{aligned} P(t) &= \text{The population of birds at time } t \\ t_0 &= \text{The initial time.} \\ P_0 &= 5{,}271{,}000 \text{ birds} \\ B &= 17 \text{ births per year per 1000 birds} \\ D &= 32 \text{ deaths per year per 1000 birds} \end{aligned} \tag{2}$$

It is possible to solve this initial value problem for $P(t)$ with pencil and paper using the method of separation of variables, but imagine that you have forgotten how to do this, but you're a really good coder with a fast computer. You're in luck – you can use your coding skills and computing power to very quickly generate an approximate solution that is quite close to the exact solution.

The basic idea is that computers are good at quickly performing a large number of discrete steps, like arithmetic operations, so we want to find a way to generate an approximate, discrete solution by repeatedly telling the computer to do arithmetic. A good step in the right direction is to notice that if Δt is a small amount of time, then we can make the following discrete approximation to the derivative:

$$\dot{P}(t) \approx \frac{P(t+\Delta t) - P(t)}{\Delta t}. \tag{3}$$

It we plug this into the differential equation we want to solve, then we find that

$$\underbrace{P(t+\Delta t)}_{\substack{\text{population a short} \\ \text{time after } t}} \approx \underbrace{P(t)}_{\substack{\text{population at} \\ \text{time } t}} + \underbrace{(B-D)P(t)\Delta t}_{\substack{\text{change in} \\ \text{population during} \\ \text{that short time}}} \tag{4}$$

Motivated by this observation, we choose some small Δt and tell the computer to compute the number P_1 defined by

$$P_1 = P_0 + (B-D)P_0\Delta t. \tag{5}$$

According to our observations above, the number P_1 is the approximate population at time $t_1 = t_0 + \Delta t$. Since we used a discrete approximation to the derivative, we incur some error in computing the population at time t_1 in this way, but that error is second order in Δt, so can be made as small as we wish by choosing Δt sufficiently small. The error we make in performing this one step, called a **time step** is called the **local truncation error** of our procedure. This term is motivated by the fact that we can think of our approximation as having arisen from using a Taylor series to approximate $P(t+\Delta t)$, but we truncated the Taylor series just after the term that is first-order in Δt. Now that we have this number stored in computer memory we instruct the computer to calculate a new number P_2 defined by

$$P_2 = P_1 + (B-D)P_1\Delta t. \tag{6}$$

The number P_2 is the approximate population at time $t_0 + 2\Delta t$. We can imagine repeating this procedure to obtain a sequence

$$P_1, P_2, \ldots, P_N \tag{7}$$

of approximate populations at the times $t_n = t_0 + n\Delta t$, where N is a positive integer giving the total number of time steps we take. The entire algorithm can be compactly described by how it generates P_{n+1} given P_n where $n = 0, 1, \ldots, N-1$;

$$P_{n+1} = P_n + (B - D)P_n\Delta t. \tag{8}$$

By the time the algorithm is run for the full N time steps, the local truncation error builds up over all of the steps and results in a total error called the **global truncation error**. It's instructive to run this algorithm by hand for a few time steps to get one's hands dirty and see how this all works out, but before we do this, note that the exact solution to the initial value problem we started with is as follows:

$$P_e(t) = P_0 e^{(B-D)t} \tag{9}$$

Knowing this allows us to compare the approximate solution we generate to the exact solution to investigate how accurate it is. Let us choose a time step Δt of 1 month = 1/12 years, then the following table gives the approximate population P_a, the exact population P_e, and global truncation error $P_e - P_a$ at each time step for $N = 12$ steps (1 year). All values are rounded to the nearest integer number of birds.

step	P_a	P_e	$P_e - P_a$
0	5,271,000	5,271,000	0
1	5,264,411	5,264,415	4
2	5,257,831	5,257,839	8
3	5,251,258	5,251,271	12
4	5,244,694	5,244,711	16
5	5,238,139	5,238,159	20
6	5,231,591	5,231,615	25
7	5,225,051	5,225,080	29
8	5,218,520	5,218,553	33
9	5,211,997	5,212,034	37
10	5,205,482	5,205,523	41
11	5,198,975	5,199,020	45
12	5,192,476	5,192,525	49

After one year, our approximation algorithm predicts a population of 5,192,476 birds, while the exact number is 5,192,525. For a step size of 1 month, the method has underestimated the resulting population at the end of a year by 49 birds. This is actually quite good

considering what a small error this is compared to the overall population of birds which is on the order of 5 million! What would have happened if had chosen a step size that was smaller while keeping the total time of the simulation fixed at one year?

step size	$P_e(t_N) - P_{a,N}$
1/12	49
1/24	24
1/48	12
1/96	6
1/192	3

Notice that the global truncation error goes down by roughly a factor of $1/2$ each time the step size is decreased by a factor of $1/2$! In other words, the global truncation error scales linearly with the step size. For this reason, this method is called a **first-order** method. In general, if the global truncation error scales as the step size to a power k, then the method is called k^{th}**-order**.

2 What happens if the problem is more complex?

Using the numerical method of approximating the population's time evolution might seem silly for the population model govering by such a simple differential equation as eq. (1), but what if the population of birds were instead governed by the following equation:

$$\dot{P}(t) = R\left(1 - \frac{P(t)}{K}\right) - \frac{CP(t)^2}{P_c^2 + P(t)^2}. \tag{10}$$

This equation tells us that the rate of change of the population at any time is some complicated, nonlinear function of the population at that time. The parameters R, K, C and P_c are constants. You might be tempted to solve this equation analytically, but good luck! In general, one could have extremely complicated functions on the right hand side of the equation. One could even have functions that depend not only on P, but also explicitly on time. In general, one could have

$$\dot{P}(t) = f(t, P(t)) \tag{11}$$

In cases when f is a complicated function, exact solution is impractical and numerical solution becomes necessary. In such cases, the

method we have been using thus far would be written as follows:

$$P_{n+1} = P_n + f(t_n, P_n)\Delta t. \tag{12}$$

By choosing Δt sufficiently small for a given amount of simulation time, one can in principle control the global truncation error of the method and obtain a numerical solution that accurately approximates the dynamics of the system.

2.1 What happens for higher-dimensional systems?

Another complication can arise when simulating dynamics. What if, for example, there were more than just birds on the island. Suppose, for example, that there were also cats that liked to hunt birds, and we wanted to keep track of both the bird population P_b and the cat population P_c. Then in general, we might have a population model governed by a system of differential equations of the following form:

$$\dot{P}_b = f_b(t, P_b, P_c) \tag{13}$$
$$\dot{P}_c = f_c(t, P_b, P_c). \tag{14}$$

We have suppressed the time arguments of all functions in the notation for compactness. Could we use the same method to simulate the dynamics of this system? Yes, just generalize the same algorithm as follow:

$$P_{b,n+1} = P_{b,n} + f(t_n, P_{b,n}, P_{c,n})\Delta t. \tag{15}$$
$$P_{c,n+1} = P_{c,n} + f(t_n, P_{b,n}, P_{c,n})\Delta t \tag{16}$$

Even more generally, one could imagine a system where we want to keep track of D different numbers as a function of time. Let's call the vector of all such numbers

$$s = (s_1, s_2, \ldots, s_D) \tag{17}$$

I used the variable s here because in general, this list of numbers can be thought of as the **state** of the system, and I have used the variable D to denote the number of components the state has because it can be thought of as the **dimension** of the system, where we are imagining that at any given time t, knowing the values of the D numbers allows us to locate the system in a D-dimensional space.

This space of all possible states is typically called **phase space**. The general dynamics of the system would be governed by a system of equations

$$\dot{s}_1(t) = f_1(t, s_1(t), s_2(t), \ldots, s_D(t)) \tag{18}$$
$$\dot{s}_2(t) = f_2(t, s_1(t), s_2(t), \ldots, s_D(t)) \tag{19}$$
$$\vdots = \vdots \tag{20}$$
$$\dot{s}_D(t) = f_D(t, s_1(t), s_2(t), \ldots, s_D(t)) \tag{21}$$

which we can write in compact form as follows:

$$\dot{s}(t) = f(t, s(t)). \tag{22}$$

Even for this case, we can use the same approximation algorithm we used above, which as a vector equation is

$$s_{n+1} = s_n + f(t_n, s_n)\Delta t, \tag{23}$$

and in terms of components is

$$s_{1,n+1} = s_{1,n} + f_1(t_n, s_{1,n})\Delta t \tag{24}$$
$$s_{2,n+1} = s_{2,n} + f_2(t_n, s_{2,n})\Delta t \tag{25}$$
$$\vdots = \vdots \tag{26}$$
$$s_{D,n+1} = s_{D,n} + f_D(t_n, s_{D,n})\Delta t \tag{27}$$

This method, which holds for a first-order system of ordinary differential equations of any dimension, is called **Euler's Method**. It is the simplest method for numerically determining the dynamics of a system governed by ordinary differential equations.

3 Are there better methods than Euler's method?

Yes! Once you've converted your system to a system of first-order equations of the form (22), you have lots of methods to choose from. For example, there are so-called **Runge-Kutta** methods. A second-order Runge-Kutta method, abbreviated **RK2**, is

$$k_{1,n} = h \cdot f(t_n, s_n) \tag{28}$$
$$k_{2,n} = h \cdot f(t_n + \tfrac{1}{2}h, s_n + \tfrac{1}{2}k_{1,n}) \tag{29}$$
$$s_{n+1} = s_n + k_{2,n}. \tag{30}$$

We are now using the notation h for the step size instead of Δt because it's more common in the numerical analysis literature. This method is second-order in the sense that its global truncation order is second order in the time step, as defined earlier. A fourth-order Runge-Kutta method, abbreviated **RK4**, is

$$k_{1,n} = h \cdot f(t_n, s_n) \tag{31}$$

$$k_{2,n} = h \cdot f(t_n + \tfrac{1}{2}h, s_n + \tfrac{1}{2}k_1) \tag{32}$$

$$k_{3,n} = h \cdot f(t_n + \tfrac{1}{2}h, s_n + \tfrac{1}{2}k_2) \tag{33}$$

$$k_{4,n} = h \cdot f(t_n + h, s_n + k_3) \tag{34}$$

$$s_{n+1} = s_n + \frac{1}{6}(k_{1,n} + 2k_{2,n} + 2k_{3,n} + k_{4,n}). \tag{35}$$