An Introduction to Mathematical Modeling in Ecology and Evolution
Sally Otto (2020)

Sponsors:
BIOS2 (NSERC - CREATE Training program)

Resources:
* *An Introduction to Mathematical Modeling in Ecology and Evolution* (2007) by Otto and Day
* Biomathematical modeling lecture notes (http://www.zoology.ubc.ca/~bio301/Bio301/Lectures.html)
* *Mathematica* labs (http://www.zoology.ubc.ca/biomath/labs.htm)

Part 1: Classic one-variable models in ecology and evolution

Exponential growth

Assumes that each individual replicates at a constant rate over time:

$$n[t+1] = R\, n[t]$$ Discrete-time model ("Recursion equation")

$$\frac{dn}{dt} = n'[t] = r\, n[t]$$ Continuous-time model ("Differential equation")

n[t]: The population size at time t
R: The number of offspring per parent in a time unit (from t to t+1).
r: The growth rate per parent per time unit.

Logistic growth

Assumes that each individual replicates at a rate that declines as a linear function of the current population size:

$$n[t+1] = \left(1 + r\left(1 - \frac{n[t]}{K}\right)\right) n[t]$$ Discrete-time model

$$\frac{dn}{dt} = n'[t] = r\left(1 - \frac{n[t]}{K}\right) n[t]$$ Continuous-time model

r: The "intrinsic" growth rate per parent per unit time, realized when competition is weak (population size small)

K: The "carrying capacity" defined as the population size at which competition causes the population to neither grow nor shrink

Haploid model of selection

Assumes that each individual carrying A has a fitness of 1+s relative to individuals carrying a, where p is the frequency of A:

$$p[t+1] = \frac{(1+s)\, p[t]}{(1+s)\, p[t]+(1-p[t])} \qquad \text{Discrete-time model}$$

$$\frac{dp}{dt} = p'[t] = s\, p[t]\,(1-p[t]) \qquad \text{Continuous-time model}$$

p: Frequency of allele *A* at a locus with two alleles (*a* and *A*). *p* must lie between 0 and 1.
s: Selection coefficient favoring allele *A* over *a*.

Getting used to *Mathematica*

Mathematica can be used as a sophisticated calculator.

For example, if r = 0.3, K = 100, and n = 90 at some time, what population size do we expect in the next generation given logistic growth?

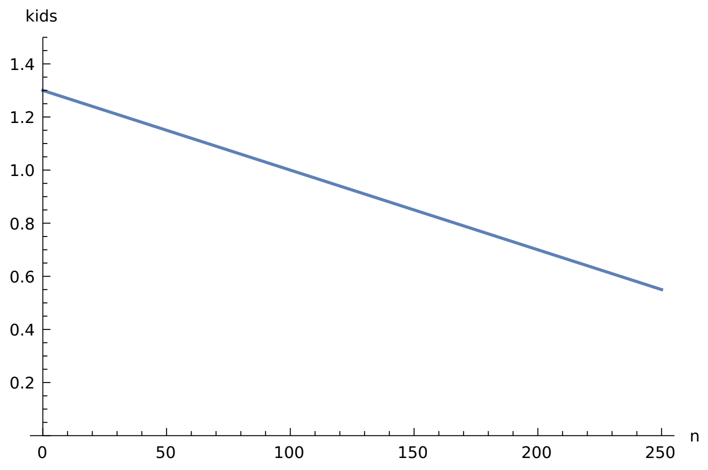
```
In[1]:=  (1 + r (1 - n/K)) n /. n → 90 /. r → 0.3 /. K → 100

Out[1]=  92.7
```

Mathematica can also plot functions.

For example, the per capita number of offspring per parent in the logistic model:

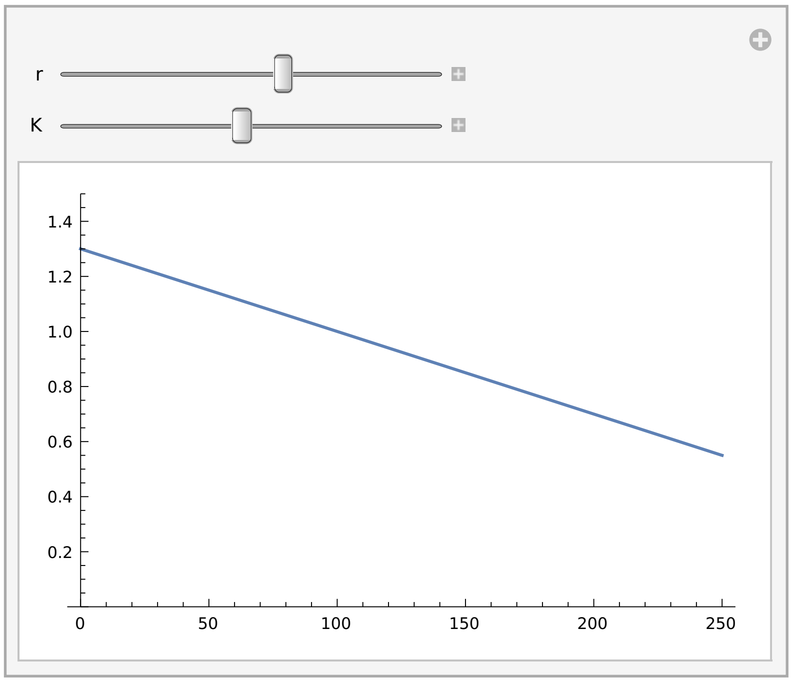
```
In[2]:= Plot[(1 + r (1 - n/K)) /. r → 0.3 /. K → 100, {n, 0, 250},
          PlotRange → {Automatic , {0, 1.5}}, AxesLabel → {"n", "kids"}]
```

Out[2]=



```
In[3]:= Manipulate[Plot[(1 + r (1 - n/K)), {n, 0, 250}, PlotRange → {Automatic , {0, 1.5}}],
          {{r, 0.3}, 0.01 , 0.5}, {{K, 100}, 10, 200}]
```

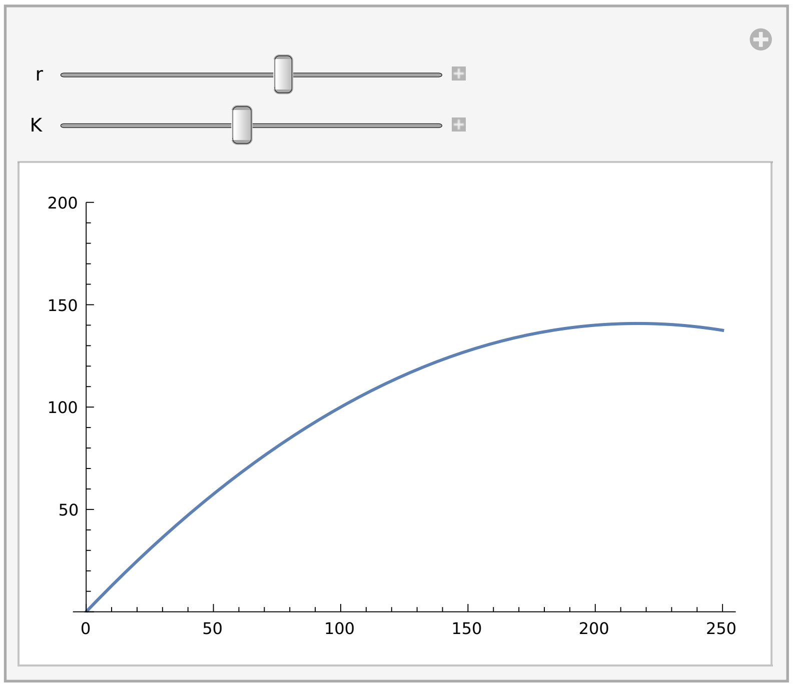
Out[3]=



Or the number of offspring produced by the entire population in the logistic model:

In[4]:=
```
Manipulate[Plot[(1 + r (1 - n/K)) * n, {n, 0, 250}, PlotRange → {Automatic , {0, 200}}],
 {{r, 0.3}, 0.01 , 0.5}, {{K, 100}, 10 , 200}]
```

Out[4]=



The real power of *Mathematica,* however, is that it can handle commands involving variables and parameters without having to specify their values.

For example, at what population size do we see the largest total number of offspring produced in the logistic model?

In[5]:=
```
D[(1 + r (1 - n/K)) n, n]
```

Out[5]=

$$1 - \frac{n\,r}{K} + \left(1 - \frac{n}{K}\right) r$$

In[6]:=
```
Solve[% == 0, n]
```

Out[6]=

$$\left\{\left\{n \to \frac{K\,(1 + r)}{2\,r}\right\}\right\}$$

Question 1: At what speed does an allele rise in frequency?

Using the continuous-time model, plot the rate of change $s\,p\,(1 - p)$ as a function of p (between 0 and 1). Choose whatever numerical value of s you wish.

Question 2: When is evolutionary change fastest?

Again, using the continuous-time model, $\frac{dp}{dt} = s\, p[t]\,(1 - p[t])$, what value of p maximizes the rate of allele frequency change? Use D[ ] to find this result analytically, for any possible value of s.

Part 2: Equilibria and their stability

Equilibria

DEFINITION: An equilibrium is a special value of a variable such that if a system starts at that value, it stays there.

RECIPE: We find equilibria by:

* Setting the variable at one time and the next (n[t + 1] and n[t]) to the same equilibrium value, $\hat{n}$, (Discrete-time model)

* Setting the change in a variable ($\frac{dn}{dt}$) to zero when started at $\hat{n}$, (Continuous-time model)

and then solving for the value(s) of $\hat{n}$ that satisfy the resulting equations.

E.g., in the logistic model, if the population size at time t is $n[t] = \hat{n}$, it will remain at this value only if $n[t + 1] = \hat{n}$.
This gives us an equation that we can solve for $\hat{n}$, by replacing all instances of n with $\hat{n}$ in
$n[t + 1] = \left(1 + r\left(1 - \frac{n[t]}{K}\right)\right) n[t]$.

In[7]:= $\text{Solve}\left[\hat{n} == \left(1 + r\left(1 - \frac{\hat{n}}{K}\right)\right)\hat{n},\ \hat{n}\right]$

Out[7]= $\{\{\hat{n} \to 0\}, \{\hat{n} \to K\}\}$

```
In[8]:= Solve[n == (1 + r (1 - n / K)) n, n]
Out[8]= {{n → 0}, {n → K}}
```

NOTE: We use == instead of = here because we don't want to set $\hat{n}$ equal to the right-hand side, we want TO TEST when the left and right will be equal.

Similarly, in continuous-time, we determine what value of $\hat{n}$ causes $\frac{dn}{dt} = r\left(1 - \frac{n[t]}{K}\right) n[t]$ to equal zero:

In[9]:= $\text{Solve}\left[0 == r\left(1 - \frac{\hat{n}}{K}\right)\hat{n},\ \hat{n}\right]$

Out[9]= $\{\{\hat{n} \to 0\}, \{\hat{n} \to K\}\}$

Question 3: What are the equilibria for the haploid model of selection?

Find $\hat{p}$ for both the discrete-time and continuous-time models:

$$p[t+1] = \frac{(1+s)\,p[t]}{(1+s)\,p[t]+(1-p[t])} \qquad \text{Discrete-time model}$$

$$\frac{dp}{dt} = p'[t] = s\,p[t]\,(1-p[t]) \qquad \text{Continuous-time model}$$

First, use *Mathematica* to solve for $\hat{p}$ analytically, then carry out the calculations by hand to confirm your answer.

Stability

A system may or may not move toward a particular equilibrium.

DEFINITION: An equilibrium is locally stable if a system near the equilibrium approaches it (attracting). An equilibrium is unstable if a system near the equilibrium moves away from it (repelling).

A local stability analysis determines whether a system that starts near an equilibrium moves toward (stable) or away from it (unstable).

The idea: We approximate the dynamics of a model near an equilibrium and then determine how it moves. To do so, we use an incredibly useful approximation tool: the Taylor Series.

Taylor series: For any nicely behaved function, f, of a variable, x[t], we can write the function around $\hat{x}$ as a series of terms, $(x[t]-\hat{x})^i$:

```
In[10]:= Series[f[x[t]], {x[t], x̂, 10 }]
```

Out[10]=

$$f[\hat{x}] + f'[\hat{x}]\,(x[t]-\hat{x}) + \frac{1}{2}\,f''[\hat{x}]\,(x[t]-\hat{x})^2 + \frac{1}{6}\,f^{(3)}[\hat{x}]\,(x[t]-\hat{x})^3 + \frac{1}{24}\,f^{(4)}[\hat{x}]\,(x[t]-\hat{x})^4 + \frac{1}{120}\,f^{(5)}[\hat{x}]\,(x[t]-\hat{x})^5 + \frac{1}{720}\,f^{(6)}[\hat{x}]\,(x[t]-\hat{x})^6 + \frac{f^{(7)}[\hat{x}]\,(x[t]-\hat{x})^7}{5040} + \frac{f^{(8)}[\hat{x}]\,(x[t]-\hat{x})^8}{40\,320} + \frac{f^{(9)}[\hat{x}]\,(x[t]-\hat{x})^9}{362\,880} + \frac{f^{(10)}[\hat{x}]\,(x[t]-\hat{x})^{10}}{3\,628\,800} + O[x[t]-\hat{x}]^{11}$$

For example, Sin[x] can be rewritten as the following function around the point 1/2:

```
In[11]:= Series[Sin[x], {x, 1/2, 4}]
```

Out[11]=

$$\mathrm{Sin}\left[\frac{1}{2}\right] + \mathrm{Cos}\left[\frac{1}{2}\right]\left(x-\frac{1}{2}\right) - \frac{1}{2}\,\mathrm{Sin}\left[\frac{1}{2}\right]\left(x-\frac{1}{2}\right)^2 - \frac{1}{6}\,\mathrm{Cos}\left[\frac{1}{2}\right]\left(x-\frac{1}{2}\right)^3 + \frac{1}{24}\,\mathrm{Sin}\left[\frac{1}{2}\right]\left(x-\frac{1}{2}\right)^4 + O\left[x-\frac{1}{2}\right]^5$$

If we are close enough to the point of interest, then $(x[t]-\hat{x})^i$ will be small, and we can drop terms that we consider to be negligible.

Constant approximation (drops $(x[t]-\hat{x})^i$ for i > 0):

In[12]:=
```
const = Normal[Series[Sin[x], {x, 1/2, 0}]]
```

Out[12]= $\text{Sin}\left[\frac{1}{2}\right]$

Linear approximation (drops $(x[t]-\hat{x})^i$ for i > 1):

In[13]:=
```
lin = Normal[Series[Sin[x], {x, 1/2, 1}]]
```

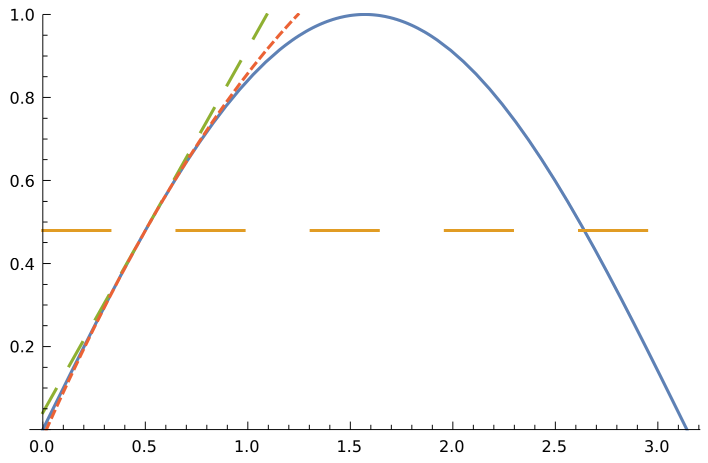
Out[13]= $\left(-\frac{1}{2}+x\right)\text{Cos}\left[\frac{1}{2}\right]+\text{Sin}\left[\frac{1}{2}\right]$

Quadratic approximation (drops $(x[t]-\hat{x})^i$ for i > 2):

In[14]:=
```
quad = Normal[Series[Sin[x], {x, 1/2, 2}]]
```

Out[14]= $\left(-\frac{1}{2}+x\right)\text{Cos}\left[\frac{1}{2}\right]+\text{Sin}\left[\frac{1}{2}\right]-\frac{1}{2}\left(-\frac{1}{2}+x\right)^2\text{Sin}\left[\frac{1}{2}\right]$

In[15]:=
```
Plot[{Sin[x], const, lin, quad}, {x, 0, Pi}, PlotRange → {Automatic, {0, 1}},
 PlotStyle → {Automatic, Dashing[0.1], Dashing[0.04], Dashing[0.01]}]
```

Out[15]=


In a local stability analysis, we assume that we are near an equilibrium, and use the Taylor series to approximate the recursion or differential equation, f[n], to linear order:

In[16]:=
```
Normal[Series[f[n[t]], {n[t], n̂, 1}]]
```

Out[16]= $f[\hat{n}]+(n[t]-\hat{n})\,f'[\hat{n}]$

Describing the recursion equation as a function (f), where $n[t+1] = f[n[t]]$, the Taylor series tells us that:

$n[t+1] \approx f[\hat{n}] + (n[t] - \hat{n})\, f'[\hat{n}]$ (note that $f'[\hat{n}]$ is the derivative of f with respect to n, evaluated at $\hat{n}$: $\left(\frac{df}{dn}\right)_{n=\hat{n}}$)

$n[t+1] \approx \hat{n} + (n[t] - \hat{n})\, f'[\hat{n}]$ ($f[\hat{n}] = \hat{n}$ because the recursion started at an equilibrium returns the equilibrium)

$(n[t+1] - \hat{n}) \approx (n[t] - \hat{n})\, f'[\hat{n}]$ (moving $\hat{n}$ to the left)

so the distance to the equilibrium $(n[t] - \hat{n})$ changes by a factor $\lambda = f'[\hat{n}]$ each generation.

Describing the differential equation as a function (f), where $\frac{dn}{dt} = f[n[t]]$, the Taylor series tells us that:

$\frac{dn}{dt} \approx f[\hat{n}] + (n[t] - \hat{n})\, f'[\hat{n}]$

$\frac{dn}{dt} \approx 0 + (n[t] - \hat{n})\, f'[\hat{n}]$ ($f[\hat{n}]=0$ because there is no change at an equilibrium)

$\frac{d\,(n[t]-\hat{n})}{dt} \approx (n[t] - \hat{n})\, f'[\hat{n}]$ (because $\frac{d\,(n[t]-\hat{n})}{dt} = \frac{d\,(n[t])}{dt} - \frac{d\,(\hat{n})}{dt} = \frac{d\,(n[t])}{dt} - 0$)

so the distance to the equilibrium changes at a rate $r = f'[\hat{n}]$.

RECIPE: To determine the stability of an equilibrium:

* Take the derivative of the recursion equation with respect to the variable, $\frac{df}{dn}$, and evaluate at an equilibrium $\lambda = \left(\frac{df}{dn}\right)_{n=\hat{n}}$.

→ The equilibrium is stable only if $-1<\lambda<1$ (Discrete-time model)

If $1<\lambda$, equilibrium is unstable with exponential growth away from it

If $0<\lambda<1$, equilibrium is stable with exponential growth toward it

If $-1<\lambda<0$, equilibrium is stable with damped oscillations toward it

If $\lambda<-1$, equilibrium is unstable with growing oscillations away from it

* Take the derivative of the differential equation with respect to the variable, $\frac{df}{dn}$, and evaluate at an equilibrium $r = \left(\frac{df}{dn}\right)_{n=\hat{n}}$.

→ The equilibrium is stable only if $r < 0$ (Continuous-time model)

If $0<r$, equilibrium is unstable with exponential growth away from it

If $r<0$, equilibrium is stable with exponential growth toward it

Repeat for each equilibrium of interest.

For example, consider the logistic model in discrete time. When will the system approach $\hat{n} = 0$, causing the system to go extinct?

```
In[17]:= derivative = D[(1 + r (1 - n[t]/K)) n[t], n[t]]
```

Out[17]= $1 - \frac{r\, n[t]}{K} + r\left(1 - \frac{n[t]}{K}\right)$

```
In[18]:= λ = derivative /. n[t] → 0
```

Out[18]= $1 + r$

For λ to lie between -1 and +1, r must lie between -2 and 0. [Technically, because 1+r measures the number of offspring per parent when competition is weak, r cannot fall below -1 or it becomes biologi - cally meaningless.]

When will the system approach $\hat{n} = K$, causing the system to remain stably at carrying capacity?

```
In[19]:= λ = derivative /. n[t] → K
```

Out[19]= $1 - r$

This will be greater than one if:	$r<0$	$\hat{n}$=K is an unstable equilibrium
This will be between 0 and 1 if:	$0<r<1$	$\hat{n}$=K is a stable equilibrium
This will be between -1 and 0 if:	$1<r<2$	$\hat{n}$=K is a stable equilibrium with damped oscillations
This will be less than -1 if:	$r>2$	$\hat{n}$=K is an unstable equilibrium with growing oscillations

Question 4: What determines stability of the equilibria for the haploid model of selection?

Under what conditions is $\hat{p} = 0$ stable?

Under what conditions is $\hat{p} = 1$ stable?

You can use either the discrete-time or the continuous-time model:

$$p[t+1] = \frac{(1+s)\, p[t]}{(1+s)\, p[t] + (1 - p[t])} \qquad \text{Discrete-time model}$$

$$\frac{dp}{dt} = p'[t] = s\, p[t]\,(1 - p[t]) \qquad \text{Continuous-time model}$$

Note that, in the discrete-time model, the fitness of *A* individuals is 1+s times greater than the fitness of *a* individuals, where 1+s must be positive (fitness can't be negative).

Part 3: Beyond equilibria

General solutions

Some models can be solved generally, allowing us to predict the future state at any time in the future and how this depends on the parameters.

DEFINITION: A general solution describes the state of a system at any future point in time.

There are many methods for finding general solutions (see Chapters 6 & 9), including iterating recursions to deduce a general rule and using recipes to solve differential equations (e.g., separation of variables).

Mathematica makes it easy to find general solutions for many of the simpler models.

Recursion equations:

```
In[20]:= RSolve[{n[t + 1] == R * n[t], n[0] == n0}, n[t], t]
```

Out[20]= $\{\{n[t] \to n0\, R^t\}\}$

NOTE: Again, we use == instead of = here because we don't want to set n[t + 1] equal to the right-hand side, we want TO TEST when the left and right will be equal.

In[21]:= $\text{RSolve}\left[\left\{p[t+1] == \dfrac{(1+s)\, p[t]}{(1+s)\, p[t] + (1 - p[t])},\ p[0] == p0\right\},\ p[t],\ t\right]$

Out[21]= $\left\{\left\{p[t] \to -\dfrac{p0}{-p0 - \left(\frac{1}{1+s}\right)^t + p0\left(\frac{1}{1+s}\right)^t}\right\}\right\}$

Differential equations:

```
In[22]:= DSolve[{D[n[t], t] == R * n[t], n[0] == n0}, n[t], t]
```

Out[22]= $\{\{n[t] \to e^{R\, t}\, n0\}\}$

```
In[23]:= DSolve[{D[p[t], t] == s p[t] (1 - p[t]), p[0] == p0}, p[t], t]
```

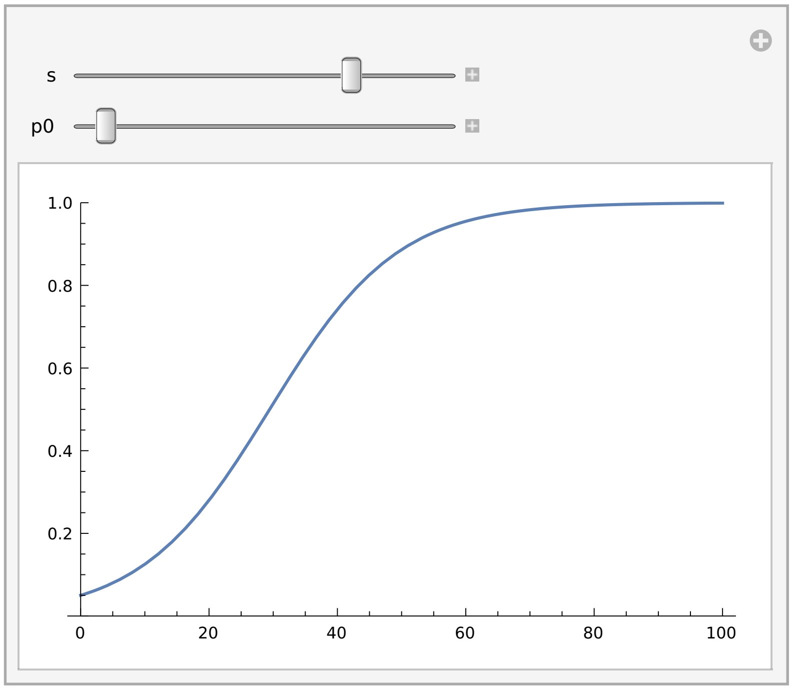
Solve : Inverse functions are being used by Solve , so some solutions may not be found ; use Reduce for complete solution information .

Out[23]= $\left\{\left\{p[t] \to \dfrac{e^{s\, t}\, p0}{1 - p0 + e^{s\, t}\, p0}\right\}\right\}$

These can then be used to predict where the system will be:

In[24]:=
```
Manipulate[Plot[ℯ^(s t) p0/(1 - p0 + ℯ^(s t) p0), {t, 0, 100}, PlotRange → {Automatic, {0, 1}}],
 {{s, 0.1}, -0.2, 0.2}, {{p0, 0.05}, 0.01, 0.99}]
```

Out[24]=



Question 5: Show that the logistic model in discrete time does not have a solution, but the model in continuous time does and then plot this solution.

Use:

$$n[t+1] = \left(1 + r\left(1 - \frac{n[t]}{K}\right)\right) n[t] \qquad \text{Discrete-time model}$$

$$\frac{dn}{dt} = n'[t] = r\left(1 - \frac{n[t]}{K}\right) n[t] \qquad \text{Continuous-time model}$$

Simulations

Some models, however, are too complicated to solve. Some cannot be solved (e.g., the logistic in discrete time), while others may not be worth the time needed to obtain a general solution when all that is needed are solutions to specific cases.

DEFINITION: A simulation explores a model using specific parameter values and initial states by applying the model's equations repeatedly.

```
In[25]:= Clear[logistic]
         logistic[r_, K_, n0_, t_] := logistic[r, K, n0, t] =
           Block[{n = logistic[r, K, n0, t - 1]},
                n + r n (1 - n / K)
                ]

         logistic[r_, K_, n0_, 0] = n0
Out[27]= n0
```

Some notes:

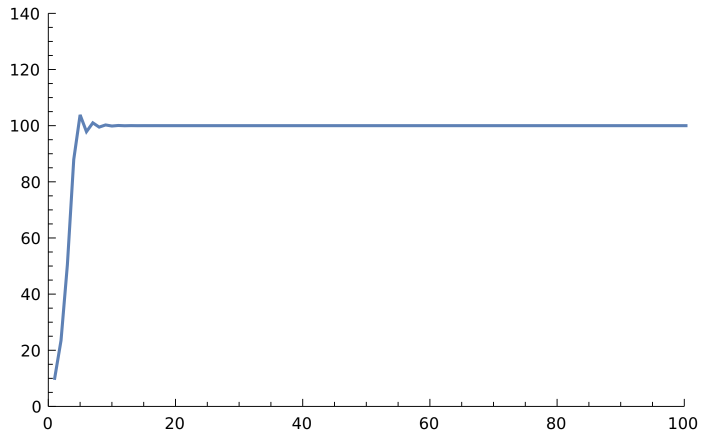
* Block keeps a set of calculations together, defining local variables in the first {}. The output will be the last entry of the Block.
* := sets the left to the right-hand side only after being called and remembers this value.
* We have to set a starting point or else the system will end up in an infinite loop going backwards in time.

We can then make a table of population sizes:

```
In[28]:= Table[logistic[1.5, 100, 10, t], {t, 0, 100}]
Out[28]= {10, 23.5, 50.4663, 87.963, 103.845, 97.8556, 101.003, 99.4833, 100.254, 99.8719,
          100.064, 99.968, 100.016, 99.992, 100.004, 99.998, 100.001, 99.9995, 100.,
          99.9999, 100., 100., 100., 100., 100., 100., 100., 100., 100., 100., 100., 100.,
          100., 100., 100., 100., 100., 100., 100., 100., 100., 100., 100., 100., 100., 100.,
          100., 100., 100., 100., 100., 100., 100., 100., 100., 100., 100., 100., 100., 100.,
          100., 100., 100., 100., 100., 100., 100., 100., 100., 100., 100., 100., 100., 100.,
          100., 100., 100., 100., 100., 100., 100., 100., 100., 100., 100., 100., 100.,
          100., 100., 100., 100., 100., 100., 100., 100., 100., 100., 100., 100., 100., 100.}
```

and plot this list:

```
In[29]:= ListPlot[Table[logistic[1.5, 100, 10, t], {t, 0, 100}],
          PlotRange → {{0, 100}, {0, 140}}, Joined → True]
Out[29]=
```



For r values between 0 and 2, we see a stable equilibrium, as we would expect from the local stability analysis of $\hat{n} = k$. For 2<r, we see oscillations, for 2.58<r, chaotic dynamics are observed, while for 3<r the system is driven extinct.

Question 6: Simulating extinction

Add an If[ ] statement to the above simulation to return zero if ever the population size is predicted to be negative.

Use this simulation to explore what happens with r = 3.01 (generate a table as above, and ListPlot it).

Numerical solutions

Differential equations can also be solved numerically using *Mathematica*'s NDSolve routines, which can work even for models that cannot be solved analytically.

```
In[30]:=  logisticD[r_, K_, n0_] := NDSolve[{D[n[t], t] == r (1 - n[t]/K) n[t], n[0] == n0}, n[t], {t, 0, 100}]

In[31]:=  logisticD[0.5, 100, 10]

Out[31]=  {{n[t] → InterpolatingFunction[ Domain : {{0., 100.}}
                                           Output : scalar ][t]}}
```

Some notes:

* We must use := here because we don't want the routine to start calculating until we call it with specific values of the parameters.

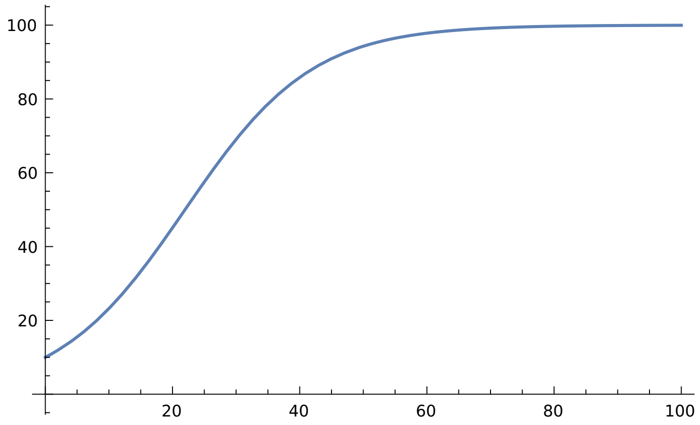
* The code is very similar to DSolve, except that we must specify the time frame over which we want a solution (here 0-100)

* When we do call the function, *Mathematica* tells us that it has represented the solution as a function, which we can interrogate.

We can then evaluate this function at specific values or in a plot:

```
In[32]:=  Evaluate[logisticD[0.1, 100, 10] /. t → 20]

Out[32]=  {{n[20] → 45.0853}}
```

In[33]:=
```
Plot[Evaluate[(n[t] /. logisticD[0.1, 100, 10])], {t, 0, 100}]
```

Out[33]=


Part 4: Example of building a model from scratch

Scenario: Let's model the number of species on a large landmass over long periods of evolutionary time, where the number of species is primarily influenced by speciation and extinction events within the landmass. If extinction risk, d, is constant per species but speciation rate declines from an initial value, b, exponentially with the number of species already present (because of competition for overlapping resources/niches), then develop a model that might describe the number of species, n, on the landmass.

Part 5: Extending to models with more than one variable

The above models were all in one variable (e.g., the population size or the allele frequency). Fortunately, the ideas are the same (but the methods more cumbersome) with models involving more than one variable. As an example, let's work with a model of an alternation of generations between haploids and diploids.

Alternation of generations is a life history common to many algae, in which both the haploid and the diploid phases exist as independently growing and reproducing organisms.

Haploid individuals (gametophytes) produce diploid individuals by producing haploid gametes that then unite.

Diploid individuals (sporophytes) produce haploid individuals by meiosis.

During a year, we assume that parents reproduce and then die, where:

a = the number of diploid offspring produced per haploid parent b = the number of haploid offspring produced per diploid parent

Then, if h[t] represents the number of haploid individuals and d[t] the number of diploid individuals, we can track the size of both populations over time using discrete-time recursions:

Haploids: h[t+1] = b d[t]
Diploids: d[t+1] = a h[t]

We can describe analogous procedures for continuous-time models, but to keep matters simple, we'll focus here only on discrete-time models

Equilibria

An equilibrium is defined in the same way, as a special point at which the system stays if started there. The only trick is to make sure that all of the variables remain constant.

For the model of alternating generations:

```
In[34]:= Solve[{ĥ == b d̂, d̂ == a ĥ}, {ĥ, d̂}]

Out[34]= {{ĥ → 0, d̂ → 0}}
```

Here we've asked *Mathematica* for all of the sets of solutions for the number of haploids and diploids $\{\hat{h}, \hat{d}\}$ that cause both of these values to remain constant in the next generation. The answer is that there's only one equilibrium point, with neither haploids or diploids.

Stability

To determine stability, we need to have a multi-dimensional analogue of λ that describes the factor by which the distance from the equilibrium grows each generation.

Idea: We again approximate the recursions near an equilibrium point using the Taylor series, but we do this for each recursion equation and each variable. For example, if we have two recursion equa - tions, f and g, describing changes in two variables, n and m, we could apply the Taylor series as before and rearrange the results as:

$$(n[t+1]-\hat{n}) = (n[t]-\hat{n})\left(\frac{df}{dn}\right)_{\hat{n},\hat{m}} + (m[t]-\hat{m})\left(\frac{df}{dm}\right)_{\hat{n},\hat{m}}$$
$$(m[t+1]-\hat{m}) = (n[t]-\hat{n})\left(\frac{dg}{dn}\right)_{\hat{n},\hat{m}} + (m[t]-\hat{m})\left(\frac{dg}{dm}\right)_{\hat{n},\hat{m}}$$

Book-keeping is easier, however, and we can use the power of matrix algebra, if we write this in matrix form:

distance to equilibrium in next generation = stability matrix times distance to equilibrium in previous generation

$$\begin{pmatrix} n[t+1]-\hat{n} \\ m[t+1]-\hat{m} \end{pmatrix} = \begin{pmatrix} \frac{df}{dn} & \frac{df}{dm} \\ \frac{dg}{dn} & \frac{dg}{dm} \end{pmatrix}_{\hat{n},\hat{m}} \begin{pmatrix} n[t]-\hat{n} \\ m[t]-\hat{m} \end{pmatrix}$$

But how do we tell if this matrix shrinks the distance or expands the distance?

DEFINITION: A matrix involving *d* variables has *d* eigenvalues that describe the factor by which the system grows along different directions (called eigenvectors). The leading eigenvalue, λ, of a matrix is the largest in magnitude of these eigenvalues, and it predicts whether the matrix stretches ($|\lambda| > 1$) or shrinks ($|\lambda| < 1$) a vector that it multiplies. If λ is a complex number, the system will cycle.

CAUTION: When calculating the stability matrix, decide on an order for the variables and then take the derivative of the recursion for the first variable in the first row with respect to the first variable in the first column. Keep the order of the variables the same for all subsequent rows and columns.

For example, take the two recursion equations for the haploid-diploid model:

```
In[35]:= eqnh = b d[t];
         eqnd = a h[t];
```

The first equation gives us h[t+1], so we must take the derivatives first with respect to h[t] and then d[t], and then evaluate at the equilibrium of interest (here 0,0):

```
In[37]:= matrix = {{D[eqnh, h[t]], D[eqnh, d[t]]},
            {D[eqnd, h[t]], D[eqnd, d[t]]}} /. {h[t] → 0, d[t] → 0}

Out[37]= {{0, b}, {a, 0}}
```

The eigenvalues of which are:

```
In[38]:= Eigenvalues[%]
```

Out[38]= $\{-\sqrt{a}\,\sqrt{b},\ \sqrt{a}\,\sqrt{b}\}$

Because a and b describe the number of offspring, these eigenvalues must be positive. The magnitude of these two eigenvalues is the same, $|\lambda| = \sqrt{a\,b}$, which tells us that the system will move away from the equilibrium at {0,0} only if $\sqrt{a\,b} > 1$, which implies that the product of a and b must be greater than one.

→ A haploid-diploid species can grow in size even if one phase produces less than a replacement number of offspring (e.g., a < 1), as long as the other phase compensates (e.g., b > 1).

Simulation

We can use the same basic code as with the logistic model to track the size of the haploid and diploid populations. The only difference is that the result is now a vector {h,d}, whose first part is the number of haploids and whose second part is the number of diploids.

In[39]:=
```
Clear[hapdip]
hapdip[a_, b_, h0_, d0_, t_] := hapdip[a, b, h0, d0, t] =
  Block[{h = Part[hapdip[a, b, h0, d0, t - 1], 1], d = Part[hapdip[a, b, h0, d0, t - 1], 2]},
       {b d, a h}
       ]

hapdip[a_, b_, h0_, d0_, 0] := {h0, d0}
```

Some notes:

* Block keeps a set of calculations together, defining local variables in the first {}. The output will be the last entry of the Block.

* := sets the left to the right-hand side only after being called and remembers this value.

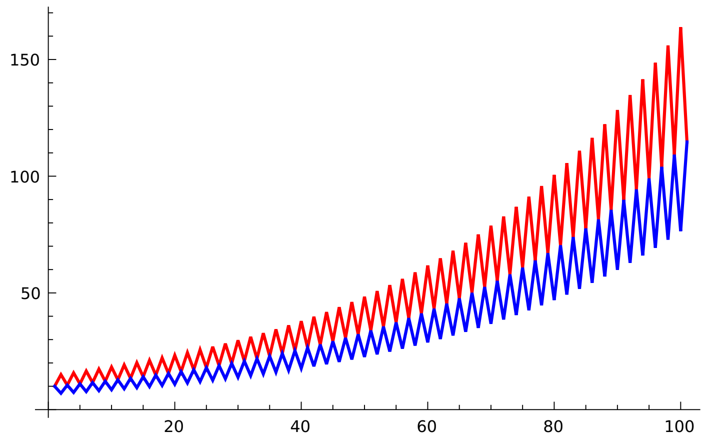
* We have to set a starting point or else the system will end up in an infinite loop going backwards in time.

We can then make a table of population sizes for {haploids, diploids}:

```
In[42]:=  Table[hapdip[0.7, 1.5, 10, 10, t], {t, 0, 100}]

Out[42]=  {{10, 10}, {15., 7.}, {10.5, 10.5}, {15.75, 7.35}, {11.025, 11.025},
           {16.5375, 7.7175}, {11.5762, 11.5762}, {17.3644, 8.10337}, {12.1551, 12.1551},
           {18.2326, 8.50854}, {12.7628, 12.7628}, {19.1442, 8.93397}, {13.401, 13.401},
           {20.1014, 9.38067}, {14.071, 14.071}, {21.1065, 9.8497}, {14.7746, 14.7746},
           {22.1618, 10.3422}, {15.5133, 15.5133}, {23.2699, 10.8593}, {16.2889, 16.2889},
           {24.4334, 11.4023}, {17.1034, 17.1034}, {25.6551, 11.9724}, {17.9586, 17.9586},
           {26.9378, 12.571}, {18.8565, 18.8565}, {28.2847, 13.1995}, {19.7993, 19.7993},
           {29.699, 13.8595}, {20.7893, 20.7893}, {31.1839, 14.5525}, {21.8287, 21.8287},
           {32.7431, 15.2801}, {22.9202, 22.9202}, {34.3803, 16.0441}, {24.0662, 24.0662},
           {36.0993, 16.8463}, {25.2695, 25.2695}, {37.9043, 17.6887}, {26.533, 26.533},
           {39.7995, 18.5731}, {27.8596, 27.8596}, {41.7894, 19.5017}, {29.2526, 29.2526},
           {43.8789, 20.4768}, {30.7152, 30.7152}, {46.0729, 21.5007}, {32.251, 32.251},
           {48.3765, 22.5757}, {33.8635, 33.8635}, {50.7953, 23.7045}, {35.5567, 35.5567},
           {53.3351, 24.8897}, {37.3346, 37.3346}, {56.0018, 26.1342}, {39.2013, 39.2013},
           {58.8019, 27.4409}, {41.1614, 41.1614}, {61.742, 28.8129}, {43.2194, 43.2194},
           {64.8291, 30.2536}, {45.3804, 45.3804}, {68.0706, 31.7663}, {47.6494, 47.6494},
           {71.4741, 33.3546}, {50.0319, 50.0319}, {75.0478, 35.0223}, {52.5335, 52.5335},
           {78.8002, 36.7734}, {55.1602, 55.1602}, {82.7402, 38.6121}, {57.9182, 57.9182},
           {86.8772, 40.5427}, {60.8141, 60.8141}, {91.2211, 42.5698}, {63.8548, 63.8548},
           {95.7822, 44.6983}, {67.0475, 67.0475}, {100.571, 46.9333}, {70.3999, 70.3999},
           {105.6, 49.2799}, {73.9199, 73.9199}, {110.88, 51.7439}, {77.6159, 77.6159},
           {116.424, 54.3311}, {81.4967, 81.4967}, {122.245, 57.0477}, {85.5715, 85.5715},
           {128.357, 59.9001}, {89.8501, 89.8501}, {134.775, 62.8951}, {94.3426, 94.3426},
           {141.514, 66.0398}, {99.0597, 99.0597}, {148.59, 69.3418}, {104.013, 104.013},
           {156.019, 72.8089}, {109.213, 109.213}, {163.82, 76.4493}, {114.674, 114.674}}
```

or plotting haploids in red and diploids in blue:

```
In[43]:=  ListPlot[{Table[Part[hapdip[0.7, 1.5, 10, 10, t], 1], {t, 0, 100}],
            Table[Part[hapdip[0.7, 1.5, 10, 10, t], 2], {t, 0, 100}]},
           Joined → True, PlotStyle → {Red, Blue}]

Out[43]=
```



Notice that in this case the diploids have higher reproductive capacity (1.5 offspring on average), yet it is the haploid population size that grows to a larger size. This is, of course, because diploids beget haploids.

A case with complex eigenvalues [EXTRA]

Some systems cycle over time. As an example, let's consider the classic predator-prey model.

Let H[t] represent the numbers of prey and P[t] represent the numbers of predators. The prey are assumed to grow exponentially, but they are
captured by predators at a rate that depends on both the availability of prey and predators.

The discrete-time recursions are thus:

Prey: H[t+1] = H[t] + r H[t] - b H[t] P[t]
Predator: P[t+1] = P[t] + c H[t] P[t] - d P[t]

which uses the following parameters:
the growth rate of the prey in the absence of the predator (r),
the capture rate at which predators contact and kill prey (b),
the rate at which eaten prey are turned into predator babies (c),
and the death rate of predators (d).

Question 7: What are the equilibria of this model?

```
In[44]:= Solve[{...}, {Ĥ, P̂}]
```

```
Syntax : "Solve [" cannot be followed by "{...}, {Ĥ, P̂}]".
```

Here we've asked *Mathematica* for all of the sets of solutions for the number of prey and predators $\{\hat{H}, \hat{P}\}$that cause both of these values to remain constant in the next generation.

Question 8: Under what conditions is the equilibrium with both species present stable?

```
In[44]:= eqn1 = H[t] + r H[t] - b H[t] × P[t];
         eqn2 = P[t] + c H[t] × P[t] - d P[t];
```

```
In[46]:= matrix = {...}
```

```
Syntax : "matrix =" cannot be followed by "{...}".
```

Let's consider the stability of the equilibrium with both prey and predators present:

```
In[46]:= matrix /. {H[t] → ..., P[t] → ...}
```

```
Syntax : "{" cannot be followed by "H[t] → ..., P[t] → ...}".
```

```
In[46]:= Eigenvalues [%]
Out[46]= Eigenvalues [P[t] - d P[t] + c H[t] × P[t]]
```

The answer will involve a complex number, where $i = \sqrt{-1}$. The above eigenvalues can be written more simply as $\{1 + i \sqrt{d\, r},\ 1 - i \sqrt{d\, r}\}$. Note that the death rate of predators (d) and the intrinsic growth rate of prey (r) can be assumed to be positive in this model, so these eigenvalues are complex numbers.

The recipe for discrete-time models remains the same: we find the absolute magnitude of the eigenvalue, $|\lambda|$, and if it is greater than one then the system is unstable. (The rule is slightly different for continuous-time models. See Chapter 8.)

DEFINITION: The absolute magnitude of a complex number is
$| \lambda | = \sqrt{(\text{real part})^2 + (\text{complex part})^2}$.

Here, the real part is "1" and the complex part is the part "$\sqrt{d\, r}$" multiplying $i = \sqrt{-1}$. So the magnitude of both eigenvalues is $\sqrt{(1)^2 + \left(\sqrt{d\, r}\right)^2}$, or just $\sqrt{1 + d\, r}$, which must be a number greater than one.

→ The predator-prey model will cycle (because λ is complex) and expand away from the equilibrium at $\left\{H[t] \to \frac{d}{c},\ P[t] \to \frac{r}{b}\right\}$ (because $|\lambda| > 1$).

We can use the same basic code again to track the prey and predator population sizes.

```
In[47]:= predprey[r_, b_, c_, d_, H0_, P0_, t_] := predprey[r, b, c, d, H0, P0, t] =
           Block[{H = Part[predprey[r, b, c, d, H0, P0, t - 1], 1],
             P = Part[predprey[r, b, c, d, H0, P0, t - 1], 2]},
               {H + r H - b H P, P + c H P - d P}
               ]

         predprey[r_, b_, c_, d_, H0_, P0_, 0] := {H0, P0}
```

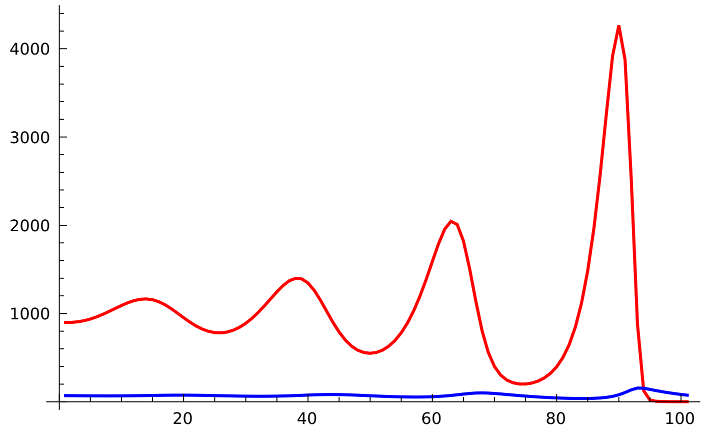
We can then make a table of population sizes for {prey, predators}:

```
In[49]:=  Table[predprey[0.7, 0.01, 0.0001, 0.1, 900, 70, t], {t, 0, 100}]

Out[49]=  {{900, 70}, {900., 69.3}, {906.3, 68.607}, {918.925, 67.9642}, {937.633, 67.4131},
           {961.888, 66.9927}, {990.815, 66.7374}, {1023.14, 66.6761}, {1057.15, 66.8304},
           {1090.66, 67.2123}, {1121.06, 67.8216}, {1145.48, 68.6427}, {1161.03, 69.6413},
           {1165.19, 70.7628}, {1156.31, 71.9317}, {1133.97, 73.0561}, {1099.32, 74.0348},
           {1054.96, 74.7701}, {1004.64, 75.181}, {952.587, 75.2159}, {902.901, 74.8593},
           {859.027, 74.1324}, {823.528, 73.0873}, {798.103, 71.7975}, {783.757, 70.348},
           {781.03, 68.8267}, {790.194, 67.3196}, {811.373, 65.9072}, {844.581, 64.6641},
           {889.648, 63.659}, {946.06, 62.9566}, {1012.69, 62.617}, {1087.46, 62.6965},
           {1166.89, 63.2448}, {1245.71, 64.3003}, {1316.71, 65.8802}, {1370.96, 67.9667},
           {1398.83, 70.488}, {1392.01, 73.2993}, {1346.08, 76.1727}, {1262.99, 78.8089},
           {1151.74, 80.8815}, {1026.41, 82.1088}, {902.124, 82.3256}, {790.932, 81.5198},
           {699.818, 79.8155}, {631.127, 77.4196}, {584.3, 74.5638}, {557.634, 71.4642},
           {549.469, 68.3029}, {558.794, 65.2256}, {585.473, 62.3478}, {630.275, 59.7633},
           {694.794, 57.5537}, {781.27, 55.7971}, {892.233, 54.5767}, {1029.84, 53.9885},
           {1194.74, 54.1497}, {1384.11, 55.2042}, {1588.9, 57.3246}, {1790.3, 60.7004},
           {1956.79, 65.4976}, {2044.89, 71.7643}, {2008.81, 79.2629}, {1822.74, 87.259},
           {1508.15, 94.4381}, {1139.59, 99.237}, {806.406, 100.622}, {559.466, 98.6743},
           {399.043, 94.3274}, {301.967, 88.6587}, {245.624, 82.47}, {214.994, 76.2487},
           {201.56, 70.2631}, {201.03, 64.653}, {211.779, 59.4874}, {234.042, 54.7985},
           {269.62, 50.6012}, {321.923, 46.9054}, {396.27, 43.7248}, {500.391, 41.085},
           {645.078, 39.0324}, {844.844, 37.647}, {1118.18, 37.0629}, {1486.47, 37.5009},
           {1969.56, 39.3252}, {2573.72, 43.138}, {3265.07, 49.9267}, {3920.48, 61.2355},
           {4264.09, 79.1192}, {3875.24, 104.944}, {2521.06, 135.118}, {879.388, 155.671},
           {126.011, 153.793}, {20.4225, 140.352}, {6.05492, 126.603}, {2.62765, 114.02},
           {1.47097, 102.648}, {0.990734, 92.3979}, {0.768831, 83.1672}, {0.667597, 74.8569}}
```

or plotting prey in red and predators in blue:

```
In[50]:=  ListPlot[{Table[Part[predprey[0.7, 0.01, 0.0001, 0.1, 900, 70, t], 1], {t, 0, 100}],
            Table[Part[predprey[0.7, 0.01, 0.0001, 0.1, 900, 70, t], 2], {t, 0, 100}]},
           Joined → True, PlotStyle → {Red, Blue}, PlotRange → All]
```

Out[50]=

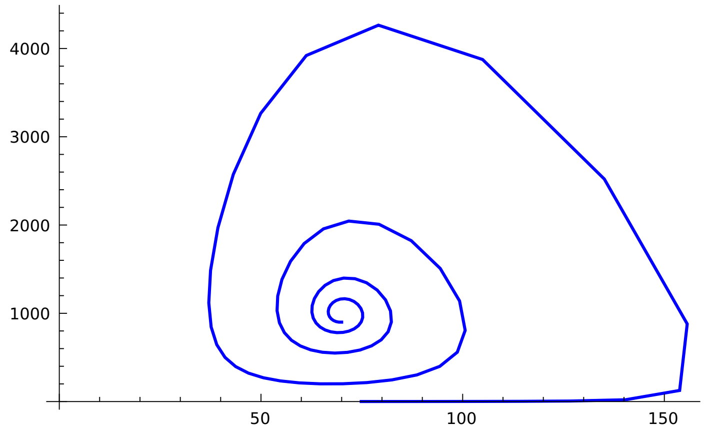


I purposely started this near the equilibrium (1000,70), otherwise it cycles out so fast that it crashes and burns very quickly. Even here, the prey go extinct by the end of this 100-generation simulation.

We can also present this graph as a phase-diagram, where the number of predators (y-axis) is plotted against the number of prey (x-axis):

```
In[51]:= ListPlot[Table[{Part[predprey[0.7, 0.01, 0.0001, 0.1, 900, 70, t], 2],
             Part[predprey[0.7, 0.01, 0.0001, 0.1, 900, 70, t], 1]}, {t, 0, 100}],
          Joined → True, PlotStyle → {Red, Blue}, PlotRange → All, AxesOrigin → {0, 0}]
```

Out[51]=



Accounting for randomness [EXTRA]

In all of the above models, we've assumed that the state of the system can be precisely predicted in the next time step. Such models are called "deterministic". In natural systems, there is always a degree of randomness or stochasticity to life. For example, even if the average number of offspring per parent were 2.3, the actual number will be an integer (0,1,2,3...).

The offspring number might, for instance, represent a random draw of the Poisson distribution with a certain mean (here 2.3):

```
In[52]:= RandomInteger[PoissonDistribution[2.3]]

Out[52]= 1
```

We can incorporate such randomness (called "demographic stochasticity") into the above ecological models by having the number of offspring be drawn from a Poisson:

```
In[53]:= Clear[hapdip]
         hapdip[a_, b_, h0_, d0_, t_] := hapdip[a, b, h0, d0, t] =
           Block[{h = Part[hapdip[a, b, h0, d0, t - 1], 1], d = Part[hapdip[a, b, h0, d0, t - 1], 2]},
             numhap = If[b d > 0, RandomInteger [PoissonDistribution [b d]], 0];
             numdip = If[a h > 0, RandomInteger [PoissonDistribution [a h]], 0];
             {numhap , numdip}
                 ]

         hapdip[a_, b_, h0_, d0_, 0] := {h0, d0}
```

Some notes:

* Block keeps a set of calculations together, defining local variables in the first {}. The output will be the last entry of the Block.

* := sets the left to the right-hand side only after being called and remembers this value.

* We have to set a starting point or else the system will end up in an infinite loop going backwards in time.

* The Poisson distribution can be evaluated by *Mathematica* only if the expected number is positive. This will cause problems if the population ever goes extinct (expectation is zero). The If statements says to draw a random number from the Poisson only if the expected number is positive.

We can then make a table of population sizes for {haploids, diploids}:

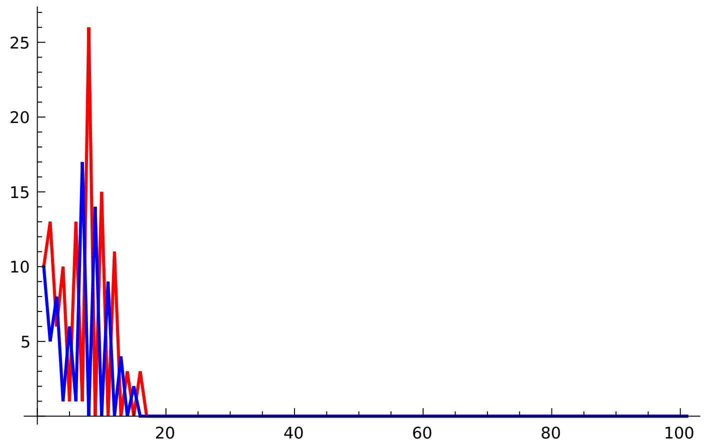
```
In[56]:= Table[hapdip[0.7, 1.5, 10, 10, t], {t, 0, 100}]

Out[56]= {{10, 10}, {13, 5}, {6, 8}, {10, 1}, {1, 6}, {13, 1}, {1, 17}, {26, 0}, {0, 14}, {15, 0},
          {0, 9}, {11, 0}, {0, 4}, {3, 0}, {0, 2}, {3, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0},
          {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0},
          {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0},
          {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0},
          {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0},
          {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0},
          {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0},
          {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}}
```

or plotting haploids in red and diploids in blue:

In[57]:=

```
ListPlot[{Table[Part[hapdip[0.7, 1.5, 10, 10, t], 1], {t, 0, 100}],
   Table[Part[hapdip[0.7, 1.5, 10, 10, t], 2], {t, 0, 100}]},
 Joined → True, PlotStyle → {Red, Blue}]
```

Out[57]=



Reenter this sub-directory a few times to see how the plots change due to demographic stochasticity. Occassionally, you'll see the whole population go extinct, despite the fact that we expect it to grow with a = 0.7 and b = 1.5.

To learn more about probability theory and stochastic models (both analytical and simulation-based methods), read Primer 3 and Chapters 13-15.

Part 6: Another example of building a model from scratch

Scenario: Consider two types of individuals that can help one another to raise a brood (e.g., one type might be better at detecting predators and the other type at collecting food). Assume that neither of the two types would be able to grow on their own, with the number of offspring per parent, 1-r1 and 1-r2, respectively, being less than one. Through cooperation, however, the per capita growth of each type rises linearly with the number of the other type, with slopes $\rho 1$ and $\rho 2$.

Appendix: Quick reference to *Mathematica*

Getting started

In[58]:= **Help -> Documentation Center** – Can be used to search for commands of interest

?Command – gives a fairly detailed description of a command, e.g., **?Plot** tells you all about this command.
You can use * as a wildcard, for instance **?*Plot*** gives a list of all commands with Plot in their name.
More help can be found in the menu under "Help", in the Function Navigator or Documentation Cente

***** – Times command (2*3 gives 6). Spaces can also be used but be careful (e.g, **a=2 3** gives six
but **a=23** gives twenty–three)

^ – Power command (2^3 gives 8)

n! – factorial (3! gives 6)

{} – denotes a list, e.g., {2,3,4}

() – Places variables together, e.g., (1+x)/(1–x) takes 1+x over 1–x

[] – Generally used to denote that something is a function of something else

%**#** – grabs previous output number #.

% – grabs the previous line of output regardless of the number

%% – grabs output two lines back. Note: naming outputs is safer (see next section).

f /. object1 -> object2
– tells Mathematica to make replace object1 with object2 in the function
e.g., **3*x^2 /. x -> 2*y+z** gives **3*(2*y + z)^2**

Syntax : Incomplete expression ; more input is needed .

Avoiding conflict with *Mathematica*

Mathematica tends to use capital letters for its functions, so its often a good idea to use
lower case names for your functions and variables.

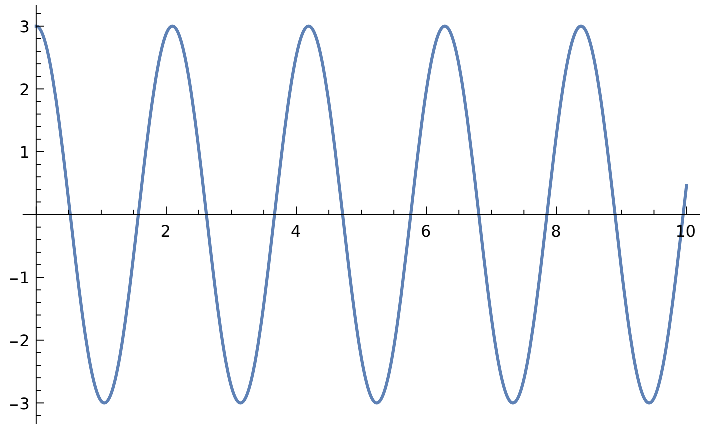
If you refer to previous entries using %, it can be difficult to know exactly what your
previous entry was. It is safer to assign a name to the output and then refer to this name later.

For example,

```
In[58]:= myderivative = D[a Sin[b x], x]
Out[58]= a b Cos[b x]
```

```
In[59]:= Plot[myderivative /. a → 1 /. b → 3, {x, 0, 10}]
```

Out[59]=



Functions and constants in *Mathematica* (A small fraction!)

`Abs[x]` - Takes the absolute value of x

`E` - The exponential constant 2.71838. **`E^(x)`** can be invoked using **`Exp[x]`**

`I` - The square root of negative 1.

`Infinity` - Self-explanatory.

`Log[x]` -Takes the natural log of x

`Log[b,x]` -Takes the log of x in base b

`Pi` - 3.14159...

`Sin[x], Cos[x], Tan[x]` - trigonometric functions
`ArcSin[x], ArcCos[x], ArcTan[x]` - inverse trigonometric functions

`Sqrt[x]` - Square root

Writing equations in *Mathematica*

In[60]:= **x=y** – Sets x to y immediately and from then on (use Clear[x] to unassign x), e.g., **plot1=Plot[x^2,{x,0,10}]**

x:=y – Does nothing until x is called, at which point x is assigned the value y

x==y – Tests whether x equals y BUT makes no assignment

f[x_]:= – This is how you define a function (called "f") of x, e.g., **f[x_] := x^2**

f[x] – This gives the function evaluated at x, e.g., **f[3]** gives **9** in the above example

f[x_,y_,...]= – This is how you define a function of several variables

Syntax : Incomplete expression ; more input is needed .

A list of helpful commands

Clear[symbol1,symbol2,...] - clears variable or function definitions,
e.g., **Clear[x, y, pop1]**

Clear["Global`*"] - clears all variable or function definitions from memory

Collect[eqn,{terms},Factor] - collects parts of an equation involving "terms" and factors them separately (if only one "term", the braces aren't needed)
e.g. Collect[$a - b + a\, x - 2\, b\, x + a^2\, x^2 + 2\, a\, b\, x^2 + b^2\, x^2$, x, Factor]

D[f,x] - takes the partial derivative of f with respect to x - e.g. D[x^2+y Log[x], x]

D[f, {x, n}] - takes the nth derivative with respect to x - e.g. D[x^2+y Log[x],{x,2}]

DSolve[eqn, y[x],x] - solves differential equation for y as a function of x (SYMBOLICALLY) e.g. DSolve[{y'[x] == k y[x], y[0]==y0, y[x],x]

DSolve[eqns, {y1,y2,y3,..}, x] same as above but for a system of eqns
e.g. pred-prey equations DSolve[{y'[x] == k y-x, z'[x]==x+z}, {y[x],z[x]}, x]

NDSolve[eqns, y, {x, xmin,xmax}] - same as DSolve but seeks solution NUMERICALLY - e.g. NDSolve[{y'[x] ==4 y[x], y[3]==62}, y[x], {x, 0,20}]

Expand[expr] - expands an expr e.g. Expand[(1+x)^2] gives 1+2x+x^2

Evaluate[object] - evaluates a symbolic object like interpolating functions

`Factor[polynomial]` - self explanatory - e.g. Factor[x^2 + 2 x + 1]

`FindRoot[eqn1==eqn2, {x, x0}]` - searches for numerical root of eqn1==eqn2 starting at x0 e.g. FindRoot[Log[x] + x + Arctan[x] == 0, {x, 4}] tries to find x that satisfies this very ugly - impossible to solve by hand equation, starting at x=4.

`For[start,test,increment,body]` - repeats procedure "body", starting from "start" until the "test" condition is met, adding "increment" each time, e.g., **`For[i=1, i≤10, i=i+1, Print[i]]`** prints out integers 1 through 10.

`Integrate[f,x]` - finds indefinite integral of f with respect to x
e.g., Integrate[Log[x], x]

`Integrate[f, {x, xmin, xmax}]` - computes definite integral from xmin to xmax
e.g., Integrate[Log[x], {x, 1,6}]

`ListPlot[list]` - plots a list of integers, e.g., **`ListPlot[{2,4,3,5,4}]`**

`ListPlot[{{x1, y1},{x2, y2},...}]` - plots a series of {x, y} values,
e.g., **`ListPlot[{{1,2},{2,1},{5,7}}]`** To join the points with a line use:
`ListPlot[{{1,2},{2,1},{5,7}},PlotJoined->True]` (in *Mathematica* 5)
`ListPlot[{{1,2},{2,1},{5,7}},Joined->True]` (in more recent versions)

`N[f]` - gives a numerical value for an expression - e.g. N[Pi] gives 3.14159

`Part[eqn,i]` - grabs the ith part of eqn, e.g., Part[3x^2+x^3,2] gives x^3

`Plot[f,{x,xmin,xmax}]` - plots f versus x on the interval [xmin,xmax]
e.g., Plot[x^2, {x,0,2}] NOTE: Plot has lots of options e.g. AxesLabel, Grid, AxesOrigin, etc. See the manual for a complete list and usage
e.g., Plot[x^2, {x,0,2}, PlotStyle->Dashed] makes a dashed curve.

`Plot3D[f, {x, xmin, xmax}, {y, ymin, ymax}]` - makes a 3D plot of f

`Show[graphics, options]` - displays graphic objects using options e.g.
Show[popplot1, PlotJoined->True]

`Simplify[expr]` - does its best to simplify an expression, expr

`Solve[eqns, vars]` - tries to solve one or a system equations for the vars specified
(SYMBOLICALLY)- e.g. Solve[{x+y ==1, x-y ==4}, {x,y}]

`NSolve[eqns, vars]` - does the same thing as Solve, but does it NUMERICALLY
(See also FindRoot)

`Sum[f, {i, imin, imax}]` - sums f from i to imax i.e. f[1] + f[2] + f[3] + ...
(only really interesting if f depends on i) - e.g. Sum[i, {i, 1,4}] gives 10.

`Reduce[{eqns}]` - can be used to determine if a statement is true or false
e.g., Reduce[{a + b > 1, a < 0, b < 0}]

`RSolve[eqns, vars]` - solves a discrete-time equation for y as a function of x
(SYMBOLICALLY) e.g. RSolve[{n[t+1] == R n[t], n[0]==n0}, n[t],t]

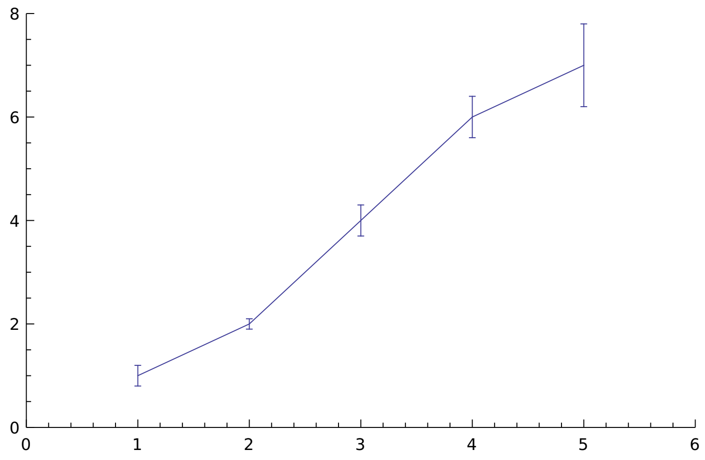
`Table[f, {i, imin, imax}]` - makes a table in list format of the function f
with i values that run from imin to imax - e.g. Table[i, {i, 1,4}] gives {1,2,3,4}.

Libraries

Mathematica has some libraries or packages that it does not load automatically.
The Documentation Center will tell you if a function needs a library.
For example, to plot error bars on a list plot, you will need:

```
In[60]:= Needs["ErrorBarPlots`"]

General : ErrorBarPlots` is now obsolete . The legacy version being loaded may conflict with current functionality .
     See the Compatibility Guide for updating information .
```

```
ErrorListPlot[{{{1, 1}, ErrorBar[0.2]}, {{2, 2}, ErrorBar[0.1]},
  {{3, 4}, ErrorBar[0.3]}, {{4, 6}, ErrorBar[0.4]}, {{5, 7}, ErrorBar[0.8]}},
 Joined → True, PlotRange → {{0, 6}, {0, 8}}]
```


0
2
4
6
8
0
1
2
3
4
5
6