Lecture 8 SimpleDynamicModels

Return to big picture - model types

Types of Models: WHAT's in the BOX

Conceptual.....Mathematical

Static.....Dynamic :*TIME*

Lumped.....Spatially Distributed: *SPACE*

Stochastic.....Deterministic

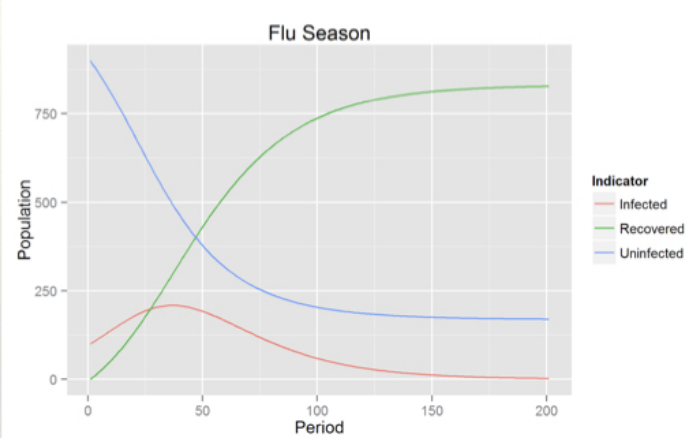
Abstract.....Physically / Process Based

but biggest differences may often be the degree specific processes / parameters are accounted for

Dynamic Models

Static- Dynamic Time Varying

- Static - Processes or Variables modeled do not evolve with time
- Dynamic - model elements evolve through time - and variables/results at one time step typically depends on previous time step



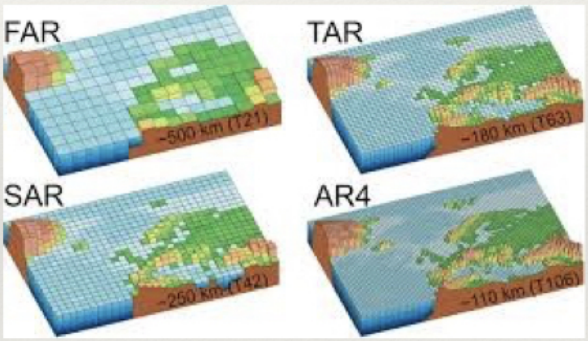
http://www.econometricsbysimulation.com/2013/05/sir-model-flue-season-dynamic.html

Spatial models that interact - similar to dynamic

- Key idea is dependency from one point in time/space to the next!
- This implies a differential/difference equation to describe the process/transfer function

Lumped ...Spatially distributed

- Lumped - single point in space, or space doesn't matter
- Spatially distributed - model is applied to different "patches" in space
  - spatial units are independent
  - **spatial units interact with each other**



http://eo.ucar.edu/staff/rrussell/climate/modeling/climate_model_resolution.html

Dynamic modeling

Dynamics - connection in space and time

- Dynamic modeling is common in environmental problems solving
- Similar issues: what happens at one place, depends on neighbors; what happens at one time; depends on previous time
- Space - two way; Time is usually one-way
- Dynamic system modeling - quickly becomes complex (Engineering degrees spend a lot of time on this; there are books, entire journals etc on this topic)

Why you should care

Dynamics models

- Many environmental problems and questions can be related to
  - Diffusion
  - Population
- Both often require dynamics models; and both often require thinking about dynamics in space and in time

Vocab

Dynamic Systems

Some useful terminology

- *stocks* - variables that evolve over time
- *flows* - transfers between variables or from the system
- *parameters* - values that controls the relationship between stocks and flows
- *sink* - something that absorbs flows
- *source* - something that generates flows

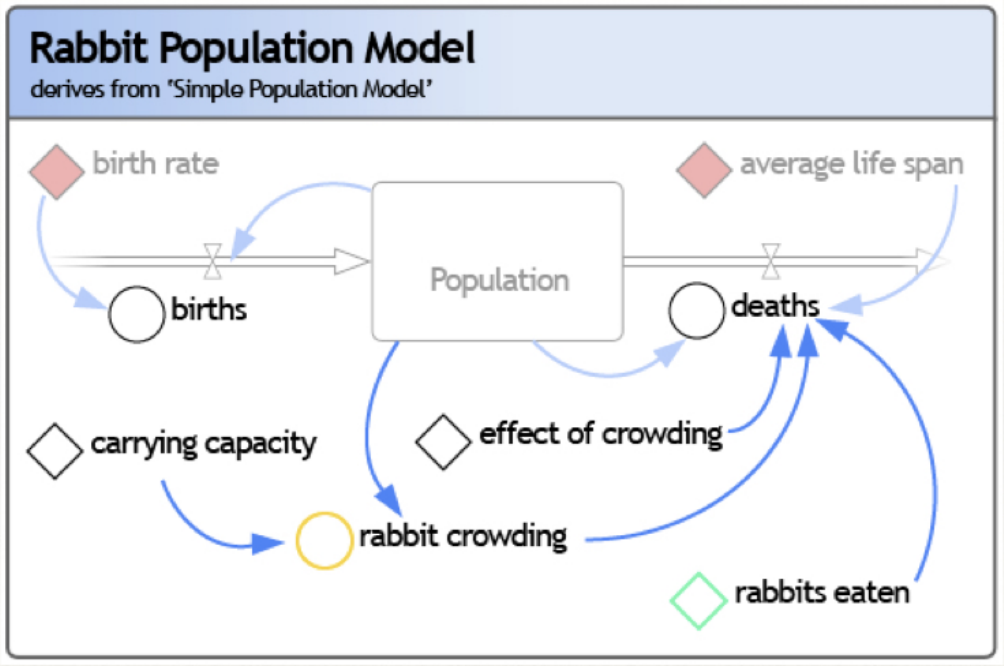
Dynamic Systems

- *System state*: value of all variables need to describe the "entity that evolves through time" at a particular point in time
  - usually think of these as stores (soil moisture, bank account balance, number of individuals in a particular age class)
- *State-space*: description of the entity may require multiple variables - for a watershed this could be soil moisture, water currently in dam and water stored in trees, and for each "grid" in a watershed)
- *State-space trajectories*: how the system state evolves through time
- *Initial conditions*: values to describe the system state at the beginning

Examples - note the feedback loops (time dependency)

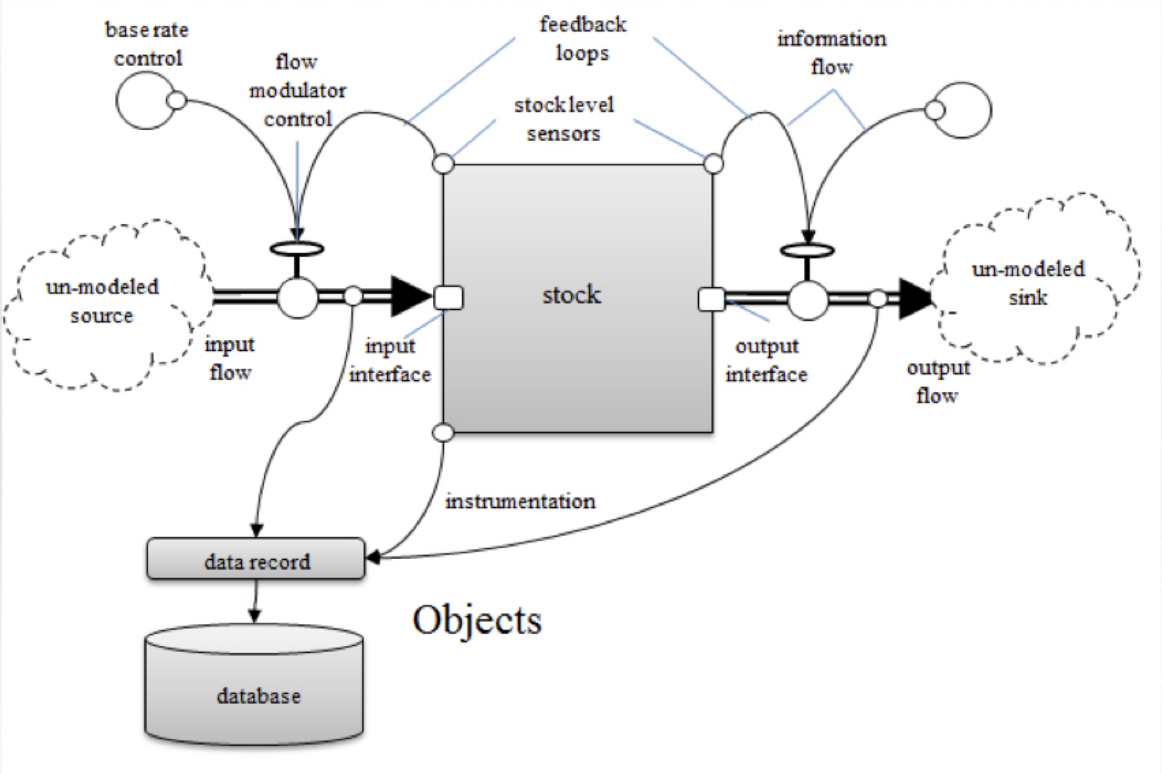
Dynamic Systems

Nature



Dynamic Systems

Human Engineered



Feekback is a key feature of dynamic models

- dynamic systems often have *feedbak* loops
  - *positive*
  - *negative*
- feedbacks often lead to non-linear responses
  - *think "runaway" growth*

Discrete vs Continuous

- do we break time/space in to "chunks" *discrete*
- or think of time/space as a stream. *continuous*

Discrete might be modeling something that occurs once every period; water withdrawls, tax revenues

Continuous would be things like pollution, water that are always changing in time

Discrete models for dynamic processes use *difference* equations Continuous models for dynamic processes use *differential* equations

Often it depends on how you look at the system -

Dynamic System

Initial conditions

- to model how something evolves over space and time
    - *you need to know where it starts from*
    - *setting initial conditions is usually part of dynamic modellinng*

Boundary Value Problems

- especially for spatial dynamic models sometimes initial conditions are provided values at "end points"

Stability

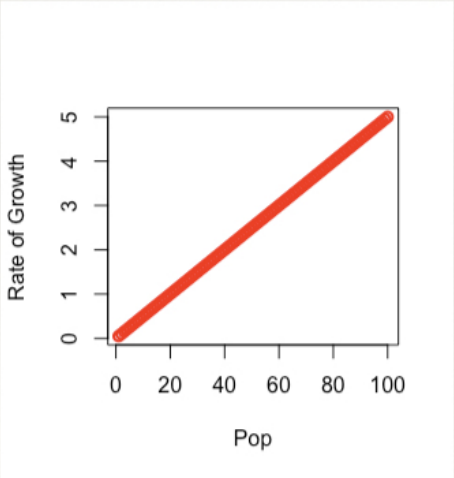
Dynamic systems may lead to *unstable* or *stable* or *cyclic* states over time

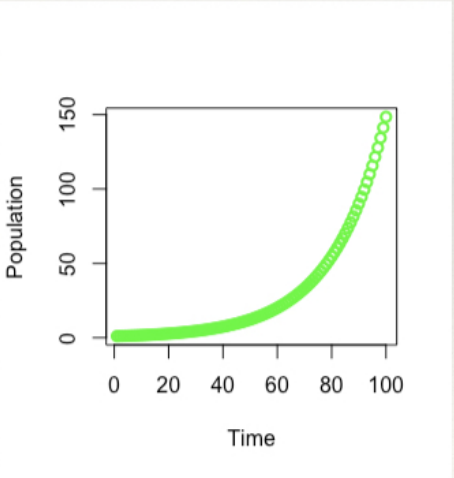
- stable…output converges to a particular over time, or for *cyclic* a repeated patterns of values
- unstable …grows to infinity
- *chaotic* ..highly sensitive to initial conditions
- for the same dynamic systems whether you are stable can change with parameters and initial conditions

Simple Example

Exponential Growth - Simple Dynamic System

- rate of growth(change) = r * population(density)
- differential equation
- $dP/dt = rP$



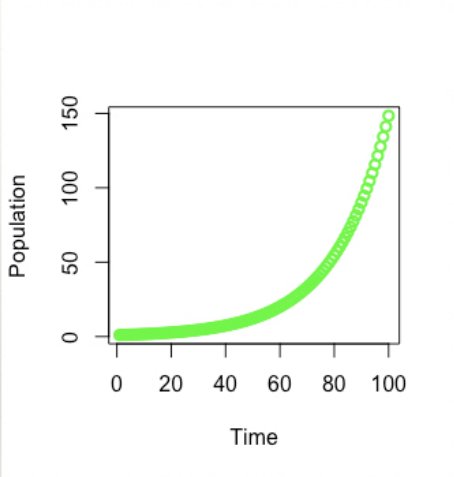


Exponential Growth - Simple Dynamic System

- differential equation
- $dP/dt = rP$
- an analytic solution exists so we can write Population as a function of time (integrating both sides)
  - $P = P0 * \exp(rt)$
  - This is a regular input-output function - that gives population after some time t

Exponential Growth - Simple Dynamic System

```
#'  Simple population growth
#' @param T  period of growth
#' @param P initial population
#' @param r intrinsic growth rate
#' @return population at time T
#'
exppop = function(T,P0,r) {
   P = P0 * exp(r*T)
   return(P)
}
```



Exponential Growth - Simple Dynamic System

But what if we couldn't 'solve' it analytically ????

Integrate the differential equation step by step

Also called numerical integration!

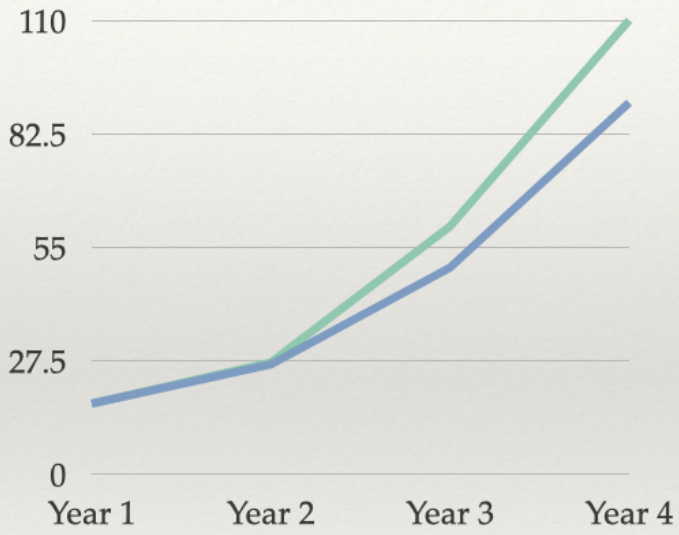
R has tools to help you do this!

First you need to code your differential equation as a function

Integration, or Solving Differential Equations

- We want the value of the dependent variation (population) over a range of values for independent variable (time)
- We know how dependent variable is changing (that's the differential equation) $dP/dt = rP$
- For each P we can approximate the next P after a small time period
  - $Pt+1 = P + dP/dt$. Timestep
  - But as P changes dP/dt changes so we have to keep time step small (really small if possible)

Integration, or Solving Differential Equations



Implementing Dynamic Models in R

Dynamic models always involves derivatives (equations that express how things change from time step to time step or place to place)

Implement population growth as a derivative - a model of population change

```
# note that we include time here but we don't use it; we will need this later
source("../R/dexppop.R")

dexppop
```

```
## function (time, P, r)
## {
##     dexpop = r * P
##     return(list(dexpop))
## }
```

Text

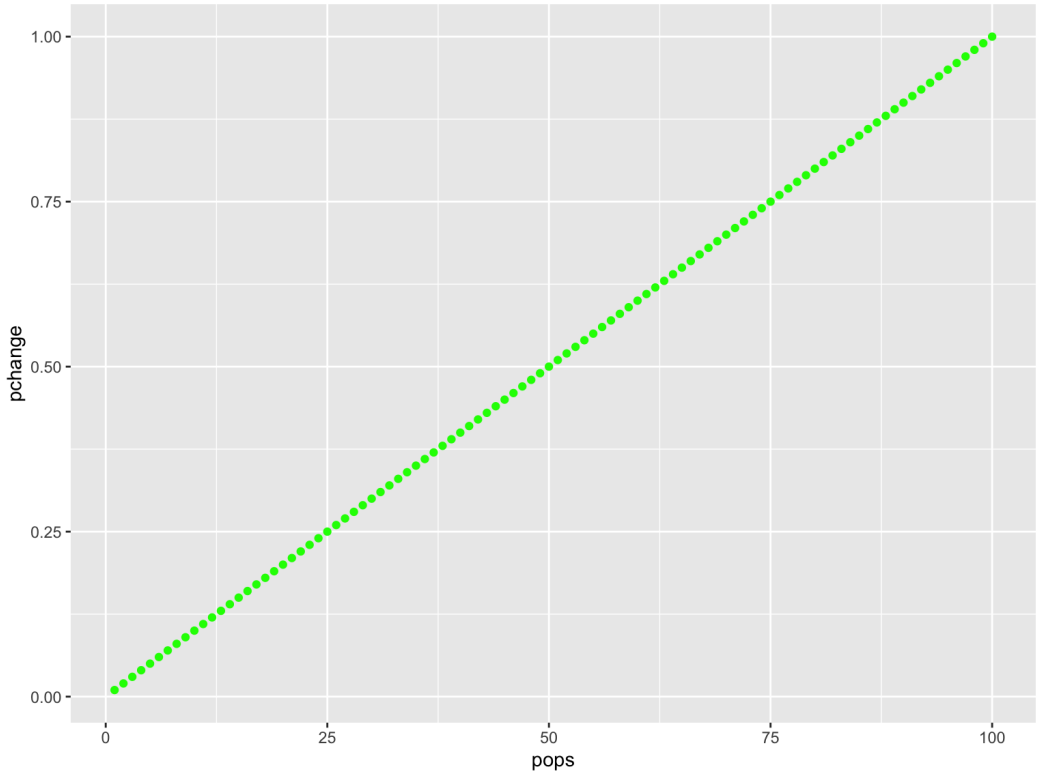
```
# see how it works
dexppop(P=20, r=0.01)
```

```
## [[1]]
## [1] 0.2
```

```
#what is this?

# notices this is the same as
dexppop(t=100,P=20, r=0.01)
```

```
## [[1]]
## [1] 0.2
```

```
# lets look at this for a range of initial populations
pops = seq(from=1, to=100)
tmp = pops %>% map(~dexppop( time=0,r=0.01, P=.x))
pchange = unlist(tmp)


pdyn = data.frame(pops, pchange)
ggplot(pdyn, aes(pops, pchange))+geom_point(col="green", size=1.5)
```




```
# why is this a straight line?
# how many new individuals are born at each population level

# try this - add a carrying capacity (dP/dt = 0 if P > carryingcapacity)
```

Integration

What if we wanted to look at population in 20 years given an initial condition

Two options

- explicit solution to differential equation is known; e.g. you can integrate both sides of the equation! Not always possible but lets look at a case where it is possible
- must be solved by iteration; this is what we do when we can't integrate both sides

Explicit Solution is available

```
source("../R/exppop.R")

exppop
```

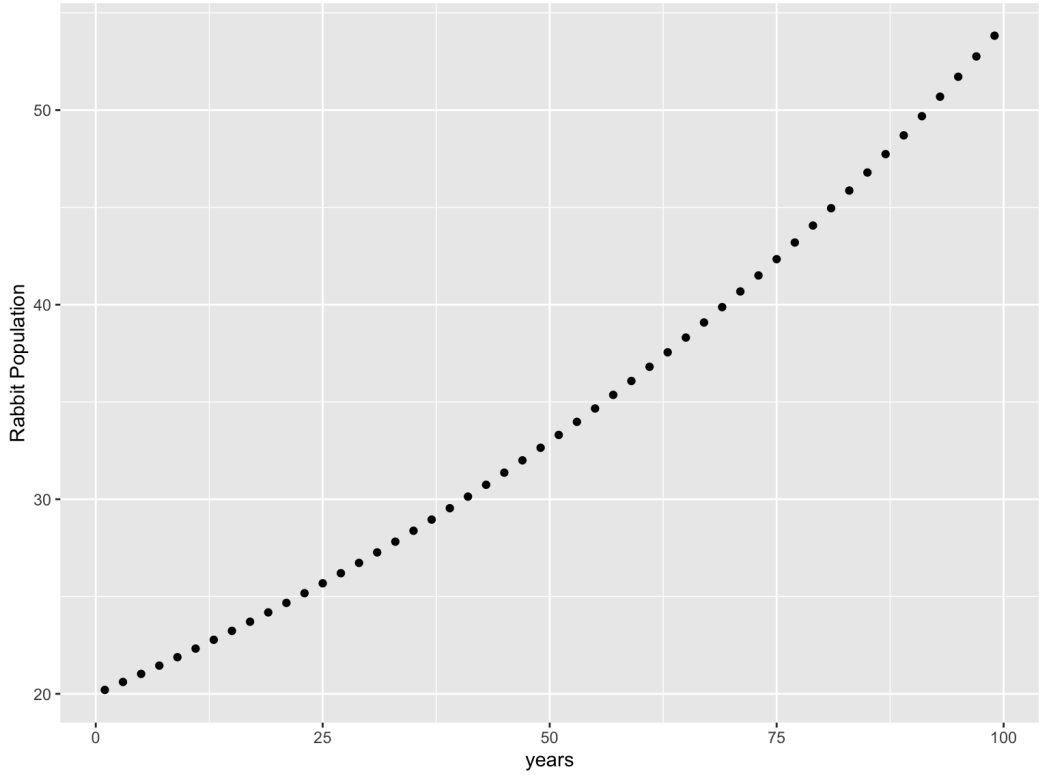
```
## function (T, P0, r, K)
## {
##     P = P0 * exp(r * T)
##     if (P > K) {
##         P = K
##     }
##     return(P)
## }
```

```
# gives population after any time given an initial population

# 20 rabbits, growth rate of 0.01 how many in 30 years
exppop(T=30, P0=20, r=0.01, K=1000)
```

carrying capacity 1k

```
## [1] 26.99718
```

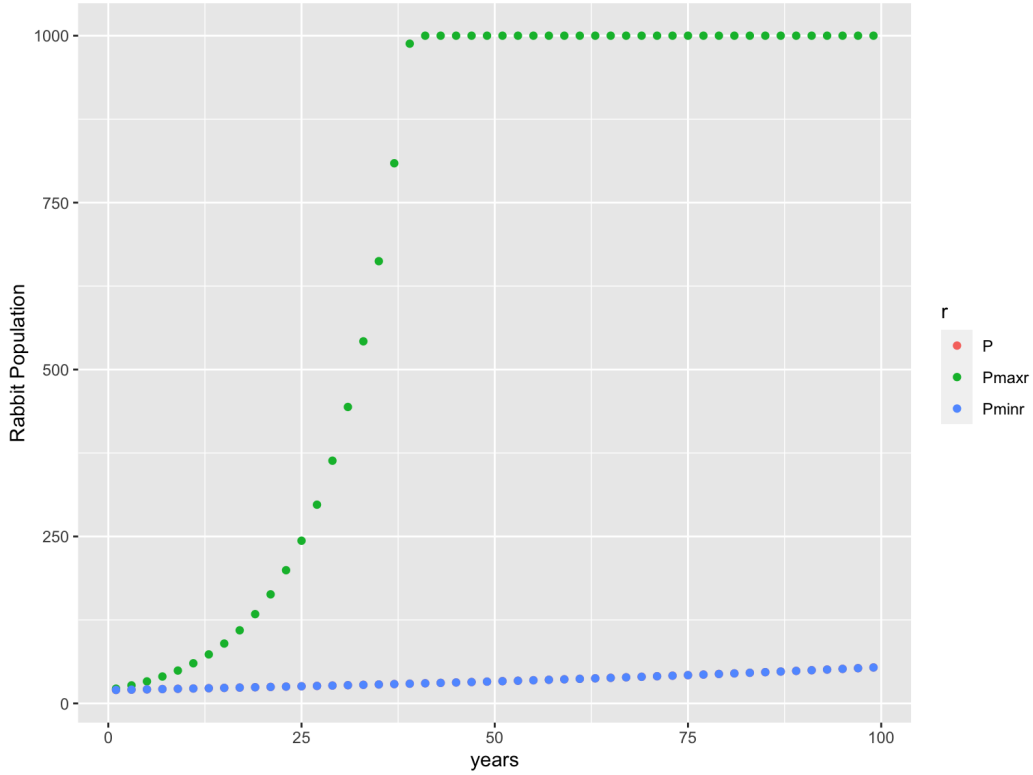
```
# if we want to see how population evolves over time - generate a time series by running our model for each point in
        time

initialrabbits = 20
years = seq(from=1, to=100, by=2)
Ptime = years %>% map_dbl(~exppop( P0=initialrabbits, r=0.01, K=1000, T=.x))

# keep track of what times we ran
Ptime = data.frame(P=Ptime, years=years)

ggplot(Ptime, aes(years,P))+geom_point()+labs(x="years",y="Rabbit Population")
```




```
# try generating results for maximum and minimum possible r values to compare (guess at what you think)


max_r = 0.1
min_r = 0.01
K = 1000

tmp = years  %>% map_dbl(~exppop(r=max_r, P0=initialrabbits, K=K, T=.x))
Ptime$Pmaxr = tmp
tmp = years %>% map_dbl(~exppop(r=min_r, P0=initialrabbits, K=K, T=.x))
Ptime$Pminr = tmp

head(Ptime)
```

```
##          P years    Pmaxr    Pminr
## 1 20.20100     1 22.10342 20.20100
## 2 20.60909     3 26.99718 20.60909
## 3 21.02542     5 32.97443 21.02542
## 4 21.45016     7 40.27505 21.45016
## 5 21.88349     9 49.19206 21.88349
## 6 22.32556    11 60.08332 22.32556
```

```
Ptimep = Ptime %>% gather(key="r",value="P",-years)
ggplot(Ptimep, aes(years,P, col=r))+geom_point()+labs(x="years",y="Rabbit Population")
```




```
# notice how population becomes unstable for high growth rates!
```

Integration

What if we wanted to look at population in 20 years given an initial condition

Two options

- explicit solution to differential equation is known; e.g. you can integrate both sides of the equation! Not always possible but lets look at a case where it is possible
- must be solved by iteration; this is what we do when we can't integrate both sides # What if no analytical solutions is available

Example

Continue looking at our rabbit population but we can't solve it..

- iterate through time
- calculate rate of change at initial conditions (e.g value of differential equation for initial conditions)
- add that value to initial conditions, to create state at time t + 1
- re-calculate rate of change (e.g value of differential equation for state at time t+1)
- keep doing this

Key questions is how much of a "jump" in t do we do

If you do this you are "kind of" turning the differential equation into a difference equation

Solving by thinking of problem as a difference equations

Population models can be discrete (rather than continuous)

So we could implement them as difference equations and iterate

```
source("../R/discrete_logistic_popK.R")
# notice how a for loop is used to iterate

# how many rabbits after 50 years given a growth of 0.1
# starting with 1 rabbit - but a carrying capcity of 500

discrete_logistic_pop
```

for loop discrete logistic population, the "disrete solution"

```
## function (P0, r, K, T = 10)
## {
##     pop = P0
##     for (i in 1:T) {
##         pop = pop + r * pop
##         pop = ifelse(pop > K, K, pop)
##     }
##     return(pop)
## }
```

```
discrete_logistic_pop(P0=1, r=0.05,K=200, T=50)
```

```
## [1] 11.4674
```

```
# save results
discrete_result = discrete_logistic_pop(P0=1, r=0.05, K=200, T=50)

# lets also keep the parameters for use later
P0=1
r=0.05
K=200
T=50
```

Compare discrete and analytic results

Save the results from both to compare

```
source("../R/exppop.R")

exppop(P0=P0, r=r, K=K, T=T)
```

```
## [1] 12.18249
```

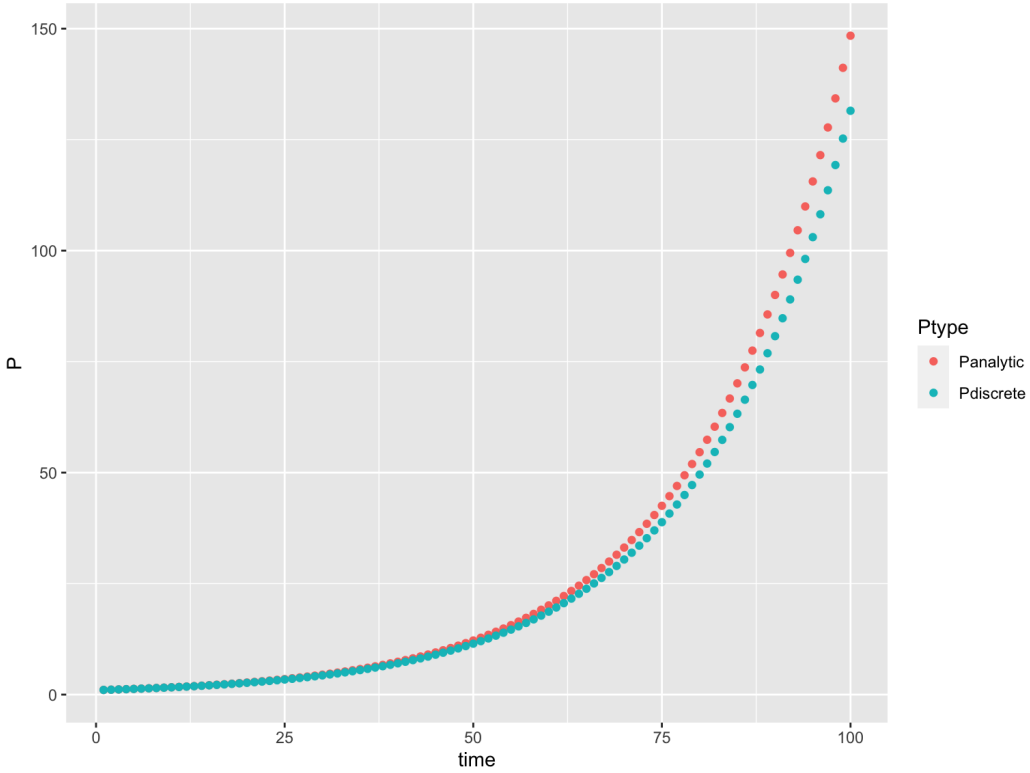
```
analytic_result = exppop(P0=P0, r=r, K=K, T=T)

analytic_result
```

```
## [1] 12.18249
```

```
discrete_result
```

```
## [1] 11.4674
```

```
# why are they different
# look at trajectories

growth_result = data.frame(time=seq(from=1,to=100))

growth_result$Panalytic = growth_result$time %>% map_dbl(~exppop( P0=1,r=0.05, K=200,T=.x ))

growth_result$Pdiscrete = growth_result$time %>% map_dbl(~discrete_logistic_pop( P0=1,r=0.05, K=200,T=.x ))

tmp = growth_result %>% gather(key="Ptype",value="P",-time)
ggplot(tmp, aes(time,P, col=Ptype))+geom_point()
```




```
# try running them for longer time periods to see what happens
# change the value of r, K , P0 - see how it effects the results
```

Solving using numeric integration

Using a solver….when you can't do the integration by hand :)

Solvers integrate by iteration but doing so in a way that more closely *approximates* analytic integration

Use mathematical tricks to deal with the fact that the rate of change keeps changing :)

There are different types of *solvers*, some work better than others depending on the form of the derivative

Numerical integration with ODE

Implement the differential equation as a function that

- returns the derivative (as a list)
- inputs time, the variable(s) and a parameter list

(it needs time even though you don't use it)

My convention: name derivative functions starting with *d* to remind myself that they are computing a derivative

ODE

Only works for Ordinary Differential Equations - single independent variable (in our case time)

Partial differential equations - more than 1 independent variable (e.g x and y if changing in space)

R has a solver called *ODE* for solving ordinary differential equations from package **desolve**

ODE requires

- initial conditions
- values of independent where we want values of dependent variable (e.g times where we want population)
- the derivative as a function
- a list that contains all parameter values (or if you have only one parameter then you can use a single value)

```
source("../R/dexppop.R")

dexppop
```

```
## function (time, P, r)
## {
##     dexpop = r * P
##     return(list(dexpop))
## }
```

```
library(deSolve)
initialrabbits = 20
years = seq(from=1, to=100, by=2)

# run the solver
Ptime = ode(y=initialrabbits, times=years, func=dexppop,parms=c(0.01))
head(Ptime)
```

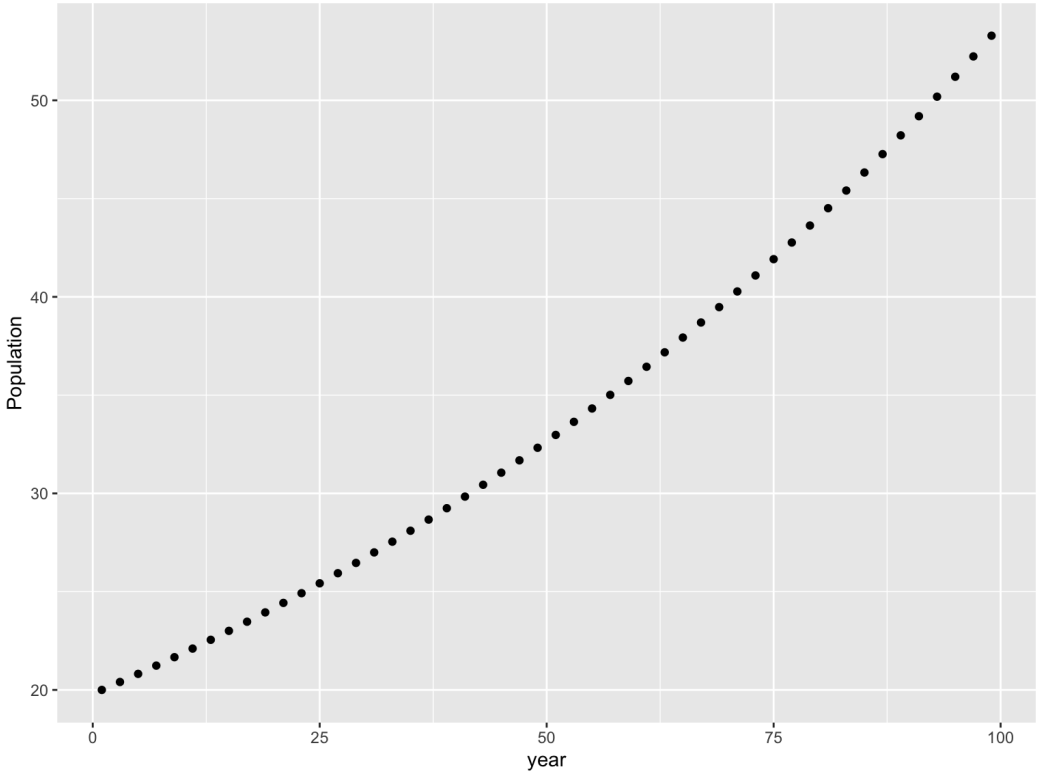
```
##      time        1
## [1,]    1 20.00000
## [2,]    3 20.40404
## [3,]    5 20.81623
## [4,]    7 21.23675
## [5,]    9 21.66576
## [6,]   11 22.10344
```

```
colnames(Ptime)=c("year","P")

# notice that there are additional pieces of information year, including the method used for integration
attributes(Ptime)
```

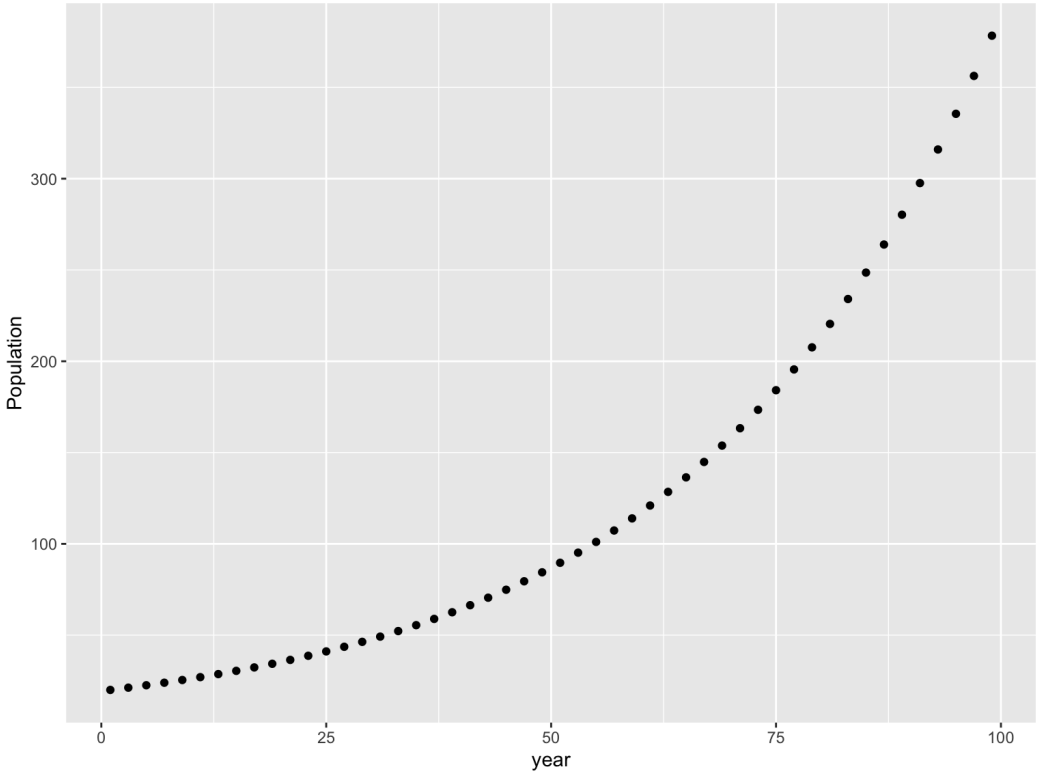
```
## $dim
## [1] 50  2
##
## $dimnames
## $dimnames[[1]]
## NULL
##
## $dimnames[[2]]
## [1] "year" "P"
##
##
## $istate
```

```
##  [1]   2  52 105  NA   6   6   0  36  21  NA  NA  NA  NA   0   1   1  NA  NA  NA
## [20]  NA  NA
##
## $rstate
## [1]  2.00000  2.00000 99.98839  0.00000  1.00000
##
## $lengthvar
## [1] 1
##
## $class
## [1] "deSolve" "matrix"
##
## $type
## [1] "lsoda"
```

```
# this also means you need to extract just the data frame for plotting
ggplot(as.data.frame(Ptime), aes(year,P))+geom_point()+labs(y="Population", "years")
```




```
# this also works (of course function can be by order)
Ptime=ode(initialrabbits, years, dexppop,0.03)
colnames(Ptime)=c("year","P")
ggplot(as.data.frame(Ptime), aes(year,P))+geom_point()+labs(y="Population", "years")
```


Population
300
200
100
0
25
50
75
100
year

Homework

Try to modify *dexppop* so that it includes carrying capacity and compare with what we did for our analytic and then discrete ways of doing this