One-Dimensional Dynamic Systems

Andre Bourbonnais

2023-11-18

One-Dimensional Dynamic Systems

Code chunk to load library

```
library(tidyverse)
library(deSolve)
library(ggpubr)
```

Exercise 1

Introduction to one-dimensional dyamic systems.

- how to find *equilibrium points*.
- how to determine tipping points for extinction.
- how to determine if an equilibrium point is stable or not.

Question 1

Consider the logistic equation that we discussed in the lecture:

$$\frac{dn}{dt} = r_0 n(1 - \frac{n}{K})$$

Describes the growth of a single population with some density dependence, i.e. there is an upper bound to how large the population can become. Assume there a constant harvest on the population at a rate h which means the number of harvested individuals per time unit is proportional to population size. In other words, the number of harvested individuals per time unit can be written as hn.

a) Write a modified growth equation that takes harvesting into account

$$\frac{dn}{dt} = r_0 n(1 - \frac{n}{K}) - hn$$

b) Find the equilibrium points of the system:
Expanding the equation from 1a, we get:

$$n(r_0(1-\frac{n}{K})-h)$$

Through the zero product law we see that

$$n_1 = 0, \qquad n_2 = K(1-\frac{h}{r_0})$$

As by setting $r_0(1-\frac{n}{K})-h=0$ and solving for n.
As n_1 is an trivial equlibrium, we are left with n_2 which we will call n^*

c) At what values of h does the population go extinct?
We understand that:

$$r_0 n(1-\frac{h}{K}) < hn$$

To determine which values of h breaks this rule we need to solve for h in $n^*=0$.

$$n^*=0 \quad \iff \quad K(1-\frac{h}{r_0})=0 \quad \iff \quad \frac{h}{r_0}=1 \quad \iff \quad h=r_0$$

Thus, the population goes extinct when:

$$r_0 \leq h$$

d) Confirm that the equilibrium point is stable
An equilibrium point is stable if:

$$f'(n^*) < 0$$

Therefore, we need to derivate $f(n)$ and evaluate if using $r_0, \quad r_0 > h$ is less than zero at $f'(n^*)$.

$$f(n) = \frac{dn}{dt} = r_0 n(1-\frac{n}{K}) - hn$$

$$f'(n) = [\text{product rule}] = r_0(1-\frac{n}{K}) + r_0 n(-\frac{1}{K}) - h$$

$$f'(n^*) = r_0(1-\frac{K(1-\frac{h}{r_0})}{K}) + r_0 K(1-\frac{h}{r_0})(-\frac{1}{K}) - h$$

$$f'(n^*) = h + h - h - r_0 = h - r_0$$

Thus, if $r_0 > h$ then $f'(n^*) < 0$ proving that the equilibrium point is stable.

e) The *yield* per time unit is simply hn. What is the *yield* at equilibrium?
We calculate hn at the equilibrium which is hn^*:

$$hn^* = hK(1-\frac{h}{r_0})$$

f) At what value of h is the yield maximized?

The *yield* can be seen as the function $g(h) = hn$. Therefore, the max value for h can be found by taking the derivative of $g(h)$ at n^* and setting it to zero.

$$g(h) = hK(1 - \frac{h}{r_0})$$

$$g'(h) = K(1 - \frac{h}{r_0}) + hK(-\frac{1}{r_0})$$

$$g'(h) = K - \frac{2hK}{r_0}$$

$$g'(h) = 0 \iff K - \frac{2hK}{r_0} = 0 \iff h = \frac{r_0}{2}$$

When $h = \frac{r_0}{2}$, the yield is maximized. Thus we can calculate the maximum yield by plugging in $\frac{r_0}{2}$ into $g(h)$:

$$g(\frac{r_0}{2}) = \frac{r_0}{2}K(1 - \frac{\frac{r_0}{2}}{r_0}) = \frac{r_0}{2}K(1 - \frac{1}{2}) = \frac{r_0}{2}K(\frac{1}{2}) = \frac{r_0 K}{4}$$

g) How is the return to equilibrium affected by h?

The return time is calculated by:

$$T_R = \frac{1}{f'(n^*)}$$

Which gives us:

$$T_R = \frac{1}{h - r_0}$$

Thus, the closer $h \to r_0$ the longer the return time. This makes sense as h is the harvesting rate, so the more we harvest the longer it will take for the population to return to equilibrium.

Question 2

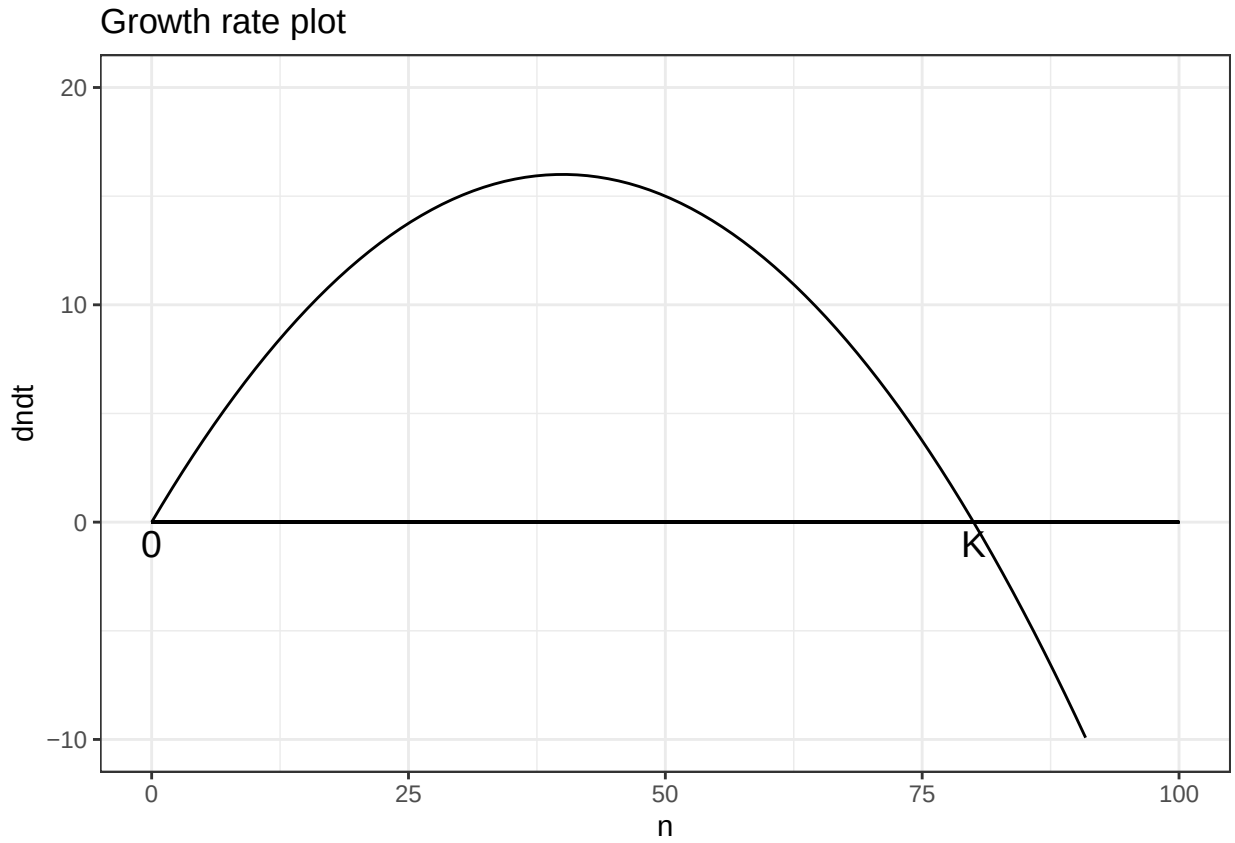
As Question 1 but done in `R`.

a) Plot the growth function:

$$\frac{dn}{dt} = r_0 n(1 - \frac{n}{K}) - hn$$

```r
# Params
r0 <- 1
K <- 100
n <- seq(0, 100, length.out = 100)
h <- 0.2

# dndt function
dndt <- r0 * n * (1 - n/K) - h * n

# Make it to df
df <- data.frame(n, dndt)
```

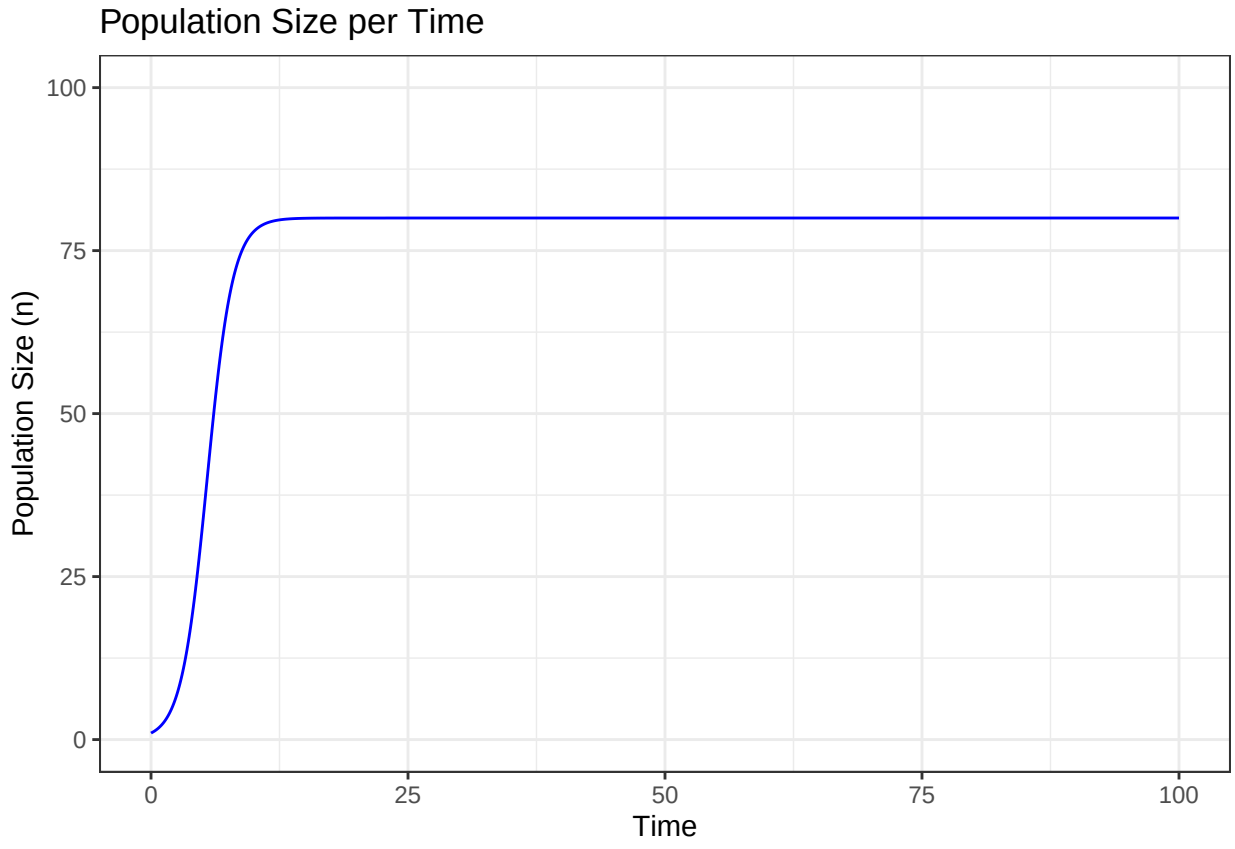
```r
# Plot
df %>%
  ggplot(aes(x = n, y = dndt)) +
  geom_line() +
  geom_segment(aes(x = 0, y = 0, xend = 100, yend = 0)) +
  annotate("text", x = 80, y=-1, label = "K", size = 5) +
  annotate("text", x = 0, y=-1, label = "0", size = 5) +
  theme_bw() +
  ylim(-10, 20) +
  labs(title = "Growth rate plot")
```



b) Solve the differential equation from 2a using the `ode()` function of the `deSolve` package. Plot the result.

```r
# Function
pop_growth <- function(t, n, P) {

  dndt <- P$r0 * n * (1 - n/P$K) - P$h * n
  return(list(dndt))
}

# Params
P <- list(r0 = 1, K = 100, h = 0.2)
```

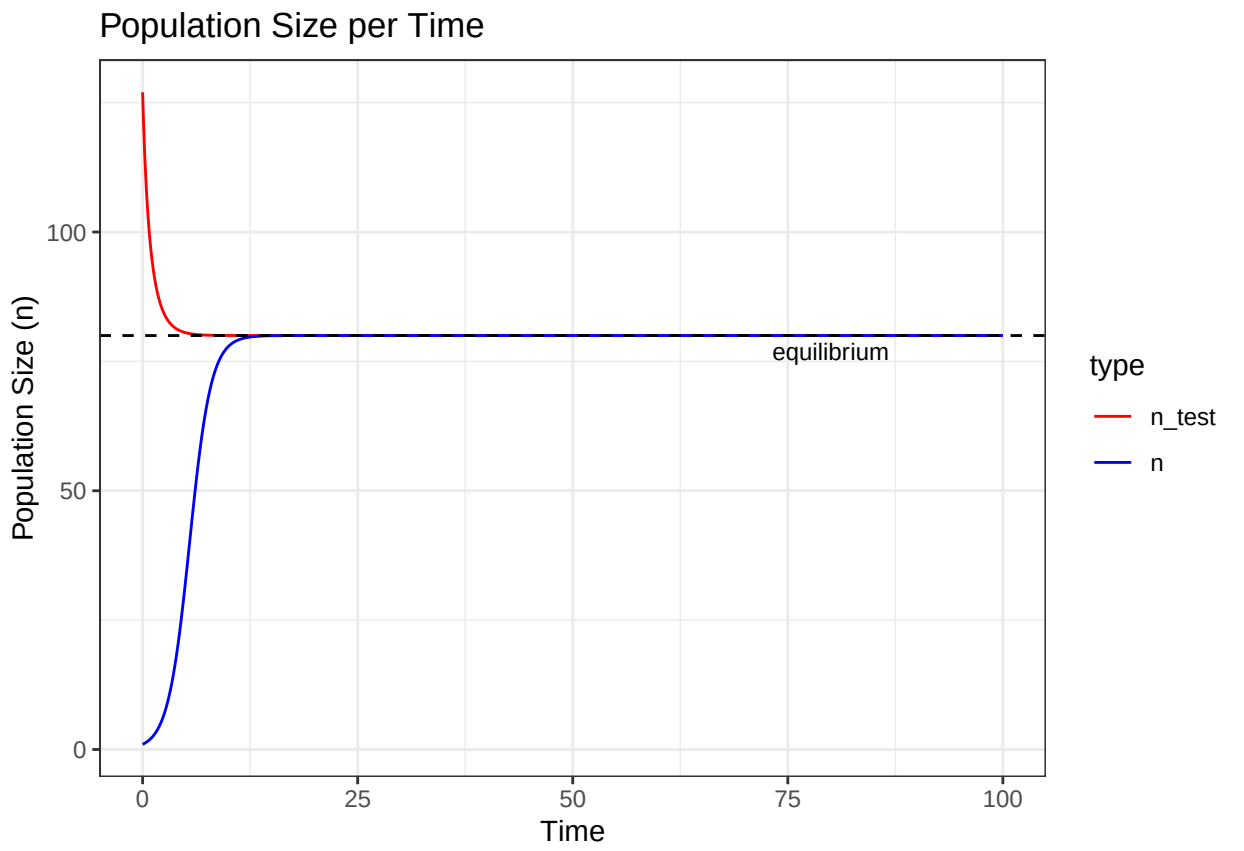
```r
# Vector of time points
times <- seq(0, 100, 0.01)

# Initial population size is 1, thus we set y = 1.
res <- ode(y = 1, times = times, func = pop_growth, parms = P)

# Change column names to avoid "1"
colnames(res) <- c("time", "n")

# Plot using ggplot
res %>%
  as.data.frame() %>%
  ggplot(aes(x = time, y = n)) +
  ylim(0, 100) +
  geom_line(color = "blue") +
  theme_bw() +
  labs(title = "Population Size per Time", x = "Time", y = "Population Size (n)")
```



c) Confirm that the equilibrium point calculated in 1b) is a stable equilibrium point.

We calculated the equilibrium point to be $n^* = K(1 - \frac{h}{r_0})$. If we calculate this and plot it it should line up with the *Population Size per Time* plot from 2b). Furthermore, if we calculate a starting population size which is bigger than n^*, the population size should decrease to the equilibrium point.

```r
# The calculated equilibrium point
equi <- P$K * (1 - P$h/P$r0)

# Larger  point to see if it moves to equi
test_point <- equi+47

# Calculate a new line with the new starting point to see if it moves towards equilibrium
test_stab <- ode(y=test_point, times = times, func = pop_growth, parms = P)

# Rename column
colnames(test_stab) <- c("time", "n_test")

# Merge test_stab and res, and add a column with the equilibrium point
df_sim <- merge(test_stab, res, by = "time")


# Reshape dataframe to long format
df_sim_long <- df_sim %>%
  pivot_longer(cols = c("n_test", "n"), names_to = "type", values_to = "value")

# I set the factor in this order so that "n" is drawn after "n_test".
df_sim_long$type <- factor(df_sim_long$type, levels = c("n_test", "n"))


# Plot in ggplot
df_sim_long %>%
  ggplot(aes(x = time, y = value, color = type)) +
  geom_line() +
  geom_hline(yintercept = equi, linetype = "dashed") +
  scale_color_manual(values = c("n" = "blue", "n_test" = "red")) +
  annotate("text", x = 80, y = equi-3, label = "equilibrium", size = 3) +
  theme_bw() +
  labs(
    title = "Population Size per Time",
    x = "Time",
    y = "Population Size (n)"
    )
```



d) Confirm with simulations that a higher h gives a slower return to equilibrium after a disturbance.

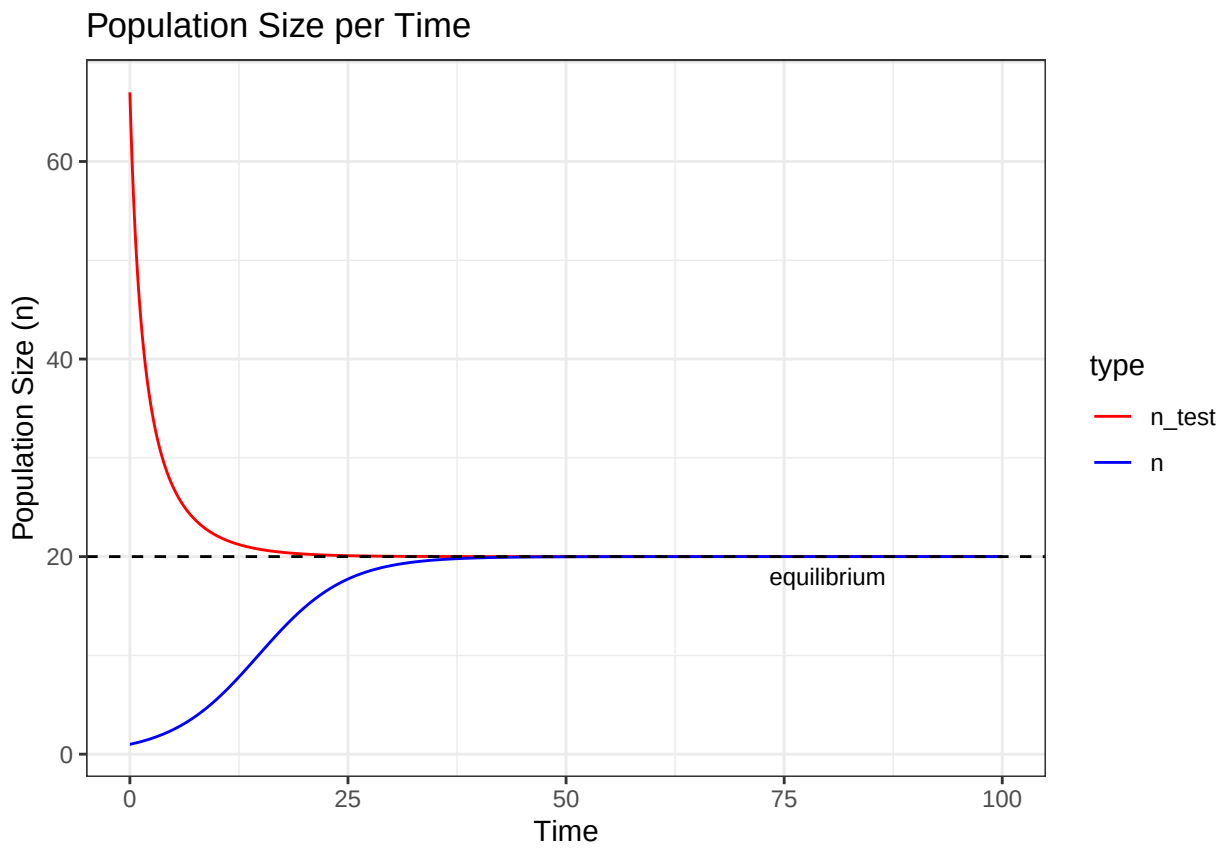
Lets increase h to 0.8 and see how fast it reaches equilibrium. What we see from the plot is that it takes longer to reach equilibrium. This is also what we would expect from the equation for the return time:

$$T_R = \frac{1}{h - r_0}$$

As a smaller denominator gives a larger number. In the first plot we see that n reaches equilibrium around 16 time units, while in the second plot it takes around 45 time units.

```
# Change the h parameter
P$h <- 0.8

# The calculated equilibrium point
equi <- P$K * (1 - P$h/P$r0)

# Larger  point to see if it moves to equi
test_point <- equi+47

# Calculate a new line with the new starting point to see if it moves towards equilibrium
test_stab <- ode(y=test_point, times = times, func = pop_growth, parms = P)

# Initial population size is 1, thus we set y = 1.
res <- ode(y = 1, times = times, func = pop_growth, parms = P)
```

```r
# Change column names to avoid "1"
colnames(res) <- c("time", "n")

# Rename column
colnames(test_stab) <- c("time", "n_test")

# Merge test_stab and res, and add a column with the equilibrium point
df_sim <- merge(test_stab, res, by = "time")


# Reshape dataframe to long format
df_sim_long <- df_sim %>%
  pivot_longer(cols = c("n_test", "n"), names_to = "type", values_to = "value")

# I set the factor in this order so that "n" is drawn after "n_test".
df_sim_long$type <- factor(df_sim_long$type, levels = c("n_test", "n"))


# Plot in ggplot
df_sim_long %>%
  ggplot(aes(x = time, y = value, color = type)) +
  geom_line() +
  geom_hline(yintercept = equi, linetype = "dashed") +
  scale_color_manual(values = c("n" = "blue", "n_test" = "red")) +
  annotate("text", x = 80, y = equi-2, label = "equilibrium", size = 3) +
  theme_bw() +
  labs(
    title = "Population Size per Time",
    x = "Time",
    y = "Population Size (n)"
    )
```



Question 3 Nutrient uptake of a cell

Cells consume nutrients, such as sugars, but the nutrients first have to pass through the cell membrane from the surrounding fluid. The transport of nutrients through the cell membrane depends on the surrounding nutrient concentration, but also on the number of receptors (transport proteins) in the membrane. It can be shown that the nutrient concentration c in the surrounding fluid follows (approximately):

$$\frac{dc}{dt} = -\frac{K_{max}c}{k_n + c}n$$

Where n is the density of cells (per unit volume), K_{max} is the maximal uptake rate for a single cell and k_n is a half-saturation constant. Both K_{max} and k_n are parameters that depend on the number of receptors per cell and their efficiency.

a) Can you see that the maximal uptake per cell approaches K_{max} as c approaches infinity?

Yes, as c approaches infinity, the fraction $\frac{c}{k_n+c} \approx \frac{c}{c} = 1$ as k_n becomes minuscule. Thus, the equation becomes:

$$\frac{dc}{dt} = -K_{max}n$$

b) Confirm that $c = 0$ is an equilibrium and that it is stable.

From the equation we can see that if $c = 0$, the equation becomes:

$$\frac{dc}{dt} = 0$$

Which means that the concentration of nutrients is constant. Thus, $c = 0$ is an equilibrium and to determine if its stable this must hold true:

$$h'(c^*) < 0$$

i) Derivative $h(c)$

$$h'(c) = \frac{K_{\max} n c}{(c + k_n)^2} - \frac{K_{\max} n}{c + k_n} = -\frac{K_{\max} k_n n}{(c + k_n)^2}$$

Thus, c^* is a stable equilibrium as the derivative becomes negative:

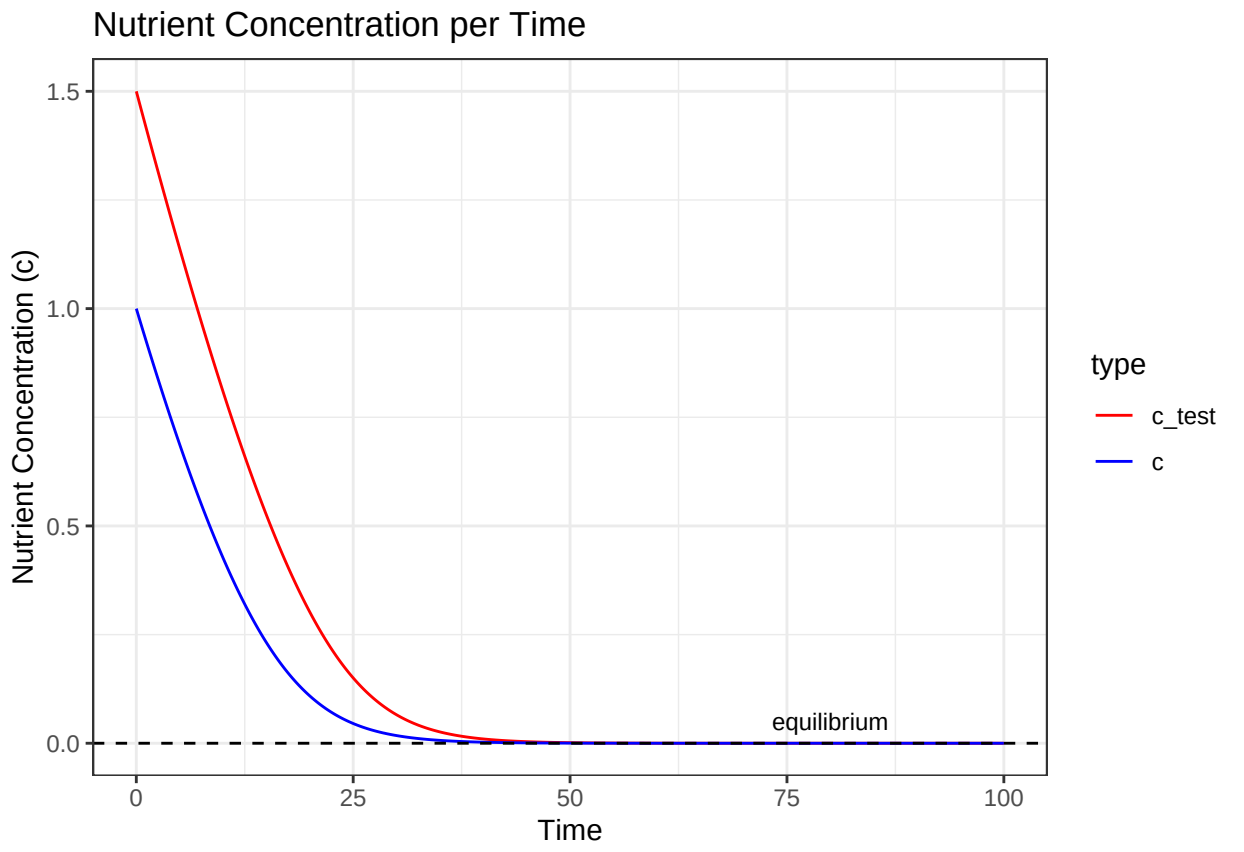
$$h'(c^*) = -\frac{K_{\max} k_n n}{(0 + k_n)^2} = -K_{\max} n < 0$$

c) Solve the differential equation and plot the results

To see if it is stable, we can plot the equation and see if it returns to the equilibrium point. We can see that it is stable as it returns to the equilibrium point.

```r
# Define function
nutrient_uptake <- function(t, c, P) {
  dcdt <- -P2$Kmax*c/(P2$kn+c)*P2$n
  return(list(dcdt))
}

# Define parameters
P2 <- list(kn = 0.5, Kmax = 0.1, n = 1)

# Time
times <- seq(0, 100, 0.01)

# Run the function
res2 <- ode(y = 1, times = times, func = nutrient_uptake, parms = P2)
res2_test <- ode(y=1+0.5, times = times, func = nutrient_uptake, parms = P2)

# Fix colnames
colnames(res2) <- c("time", "c")
colnames(res2_test) <- c("time", "c_test")

# Merge res2_test and res2, and add a column with the equilibrium point
df2_sim <- merge(res2_test, res2, by = "time")


# Reshape dataframe to long format
df2_sim_long <- df2_sim %>%
  pivot_longer(cols = c("c_test", "c"), names_to = "type", values_to = "value")

# I set the factor in this order so that "n" is drawn after "n_test".
df2_sim_long$type <- factor(df2_sim_long$type, levels = c("c_test", "c"))

# Plot in ggplot
df2_sim_long %>%
  ggplot(aes(x = time, y = value, color = type)) +
  geom_line() +
  geom_hline(yintercept = 0, linetype = "dashed") +
```

```
  scale_color_manual(values = c("c" = "blue", "c_test" = "red")) +
  annotate("text", x = 80, y = 0.05, label = "equilibrium", size = 3) +
  theme_bw() +
  labs(
    title = "Nutrient Concentration per Time",
    x = "Time",
    y = "Nutrient Concentration (c)"
    )
```



d) Write the new equation for the nutrient dynamics.

Now assume we have chemostat conditions, i.e. that there is a constant inflow of l of nutrients and an outflow μc proportional to the current concentration. The equation for the nutrient concentration now becomes:

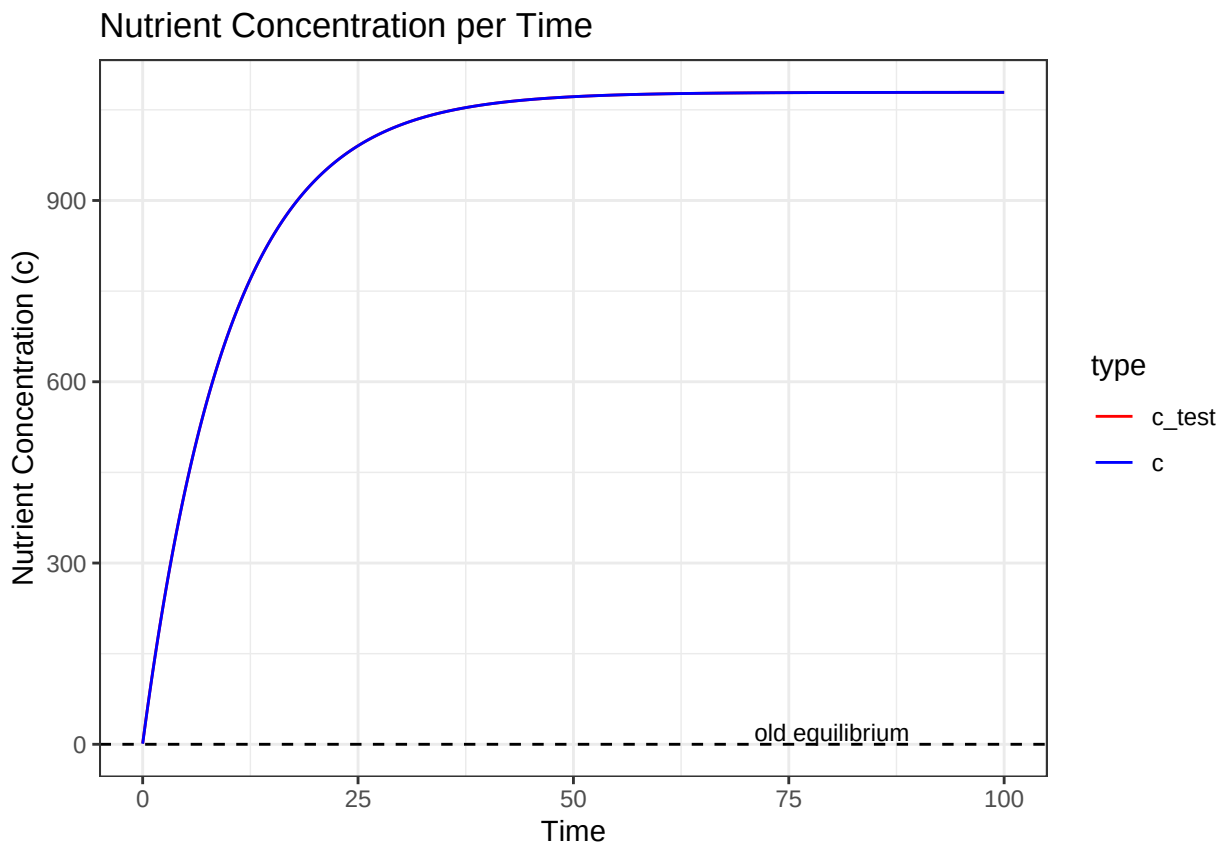
$$\frac{dc}{dt} = l - \mu c - \frac{K_{max} c}{k_n + c} n$$

d,e) Plot the new equation for the nutrient dynamics. Is there a new equilibrium and is it stable?

There does not seem to be a new equilibrium point, by looking at the plot. Further, setting $\frac{dc}{dt} = 0$ and solving for c does not yield a solution. Thus, there is no new equilibrium point.

```
# Define function
nutrient_uptake_chemostat <- function(t, c, P) {
  dcdt <- P3$l - P3$mu*c - P3$Kmax*c/(P3$kn+c)*P3$n
```

```r
  return(list(dcdt))
}

# Define parameters
P3 <- list(kn = 0.5, Kmax = 0.1, n = 1, l = 108, mu = 0.1)

# Time
times <- seq(0, 100, 0.01)

# Run the function
res3 <- ode(y = 1, times = times, func = nutrient_uptake_chemostat, parms = P3)
res3_test <- ode(y = 1 + 0.5, times = times, func = nutrient_uptake_chemostat,
                 parms = P3)
# Fix colnames
colnames(res3) <- c("time", "c")
colnames(res3_test) <- c("time", "c_test")

# Merge res3_test and res3, and add a column with the equilibrium point
df3_sim <- merge(res3_test, res3, by = "time")

# Reshape dataframe to long format
df3_sim_long <- df3_sim %>%
  pivot_longer(cols = c("c_test", "c"), names_to = "type", values_to = "value")

# I set the factor in this order so that "n" is drawn after "n_test".
df3_sim_long$type <- factor(df3_sim_long$type, levels = c("c_test", "c"))

# Plot in ggplot
df3_sim_long %>%
  ggplot(aes(x = time, y = value, color = type)) +
  geom_line() +
  geom_hline(yintercept = 0, linetype = "dashed") +
  scale_color_manual(values = c("c" = "blue", "c_test" = "red")) +
  annotate("text", x = 80, y = 20, label = "old equilibrium", size = 3) +
  theme_bw() +
  labs(
    title = "Nutrient Concentration per Time",
    x = "Time",
    y = "Nutrient Concentration (c)"
    )
```



Exercise 2 Programming dynamic systems

In thius exercise we will use the logistic equation, that was introduced in the lecture:

$$\frac{dn}{dt} = r_0 n(1 - \frac{n}{K})$$

The idea is to write your own numerical solver of differential equations. The `deSolve` package is forbidden in this exercise.

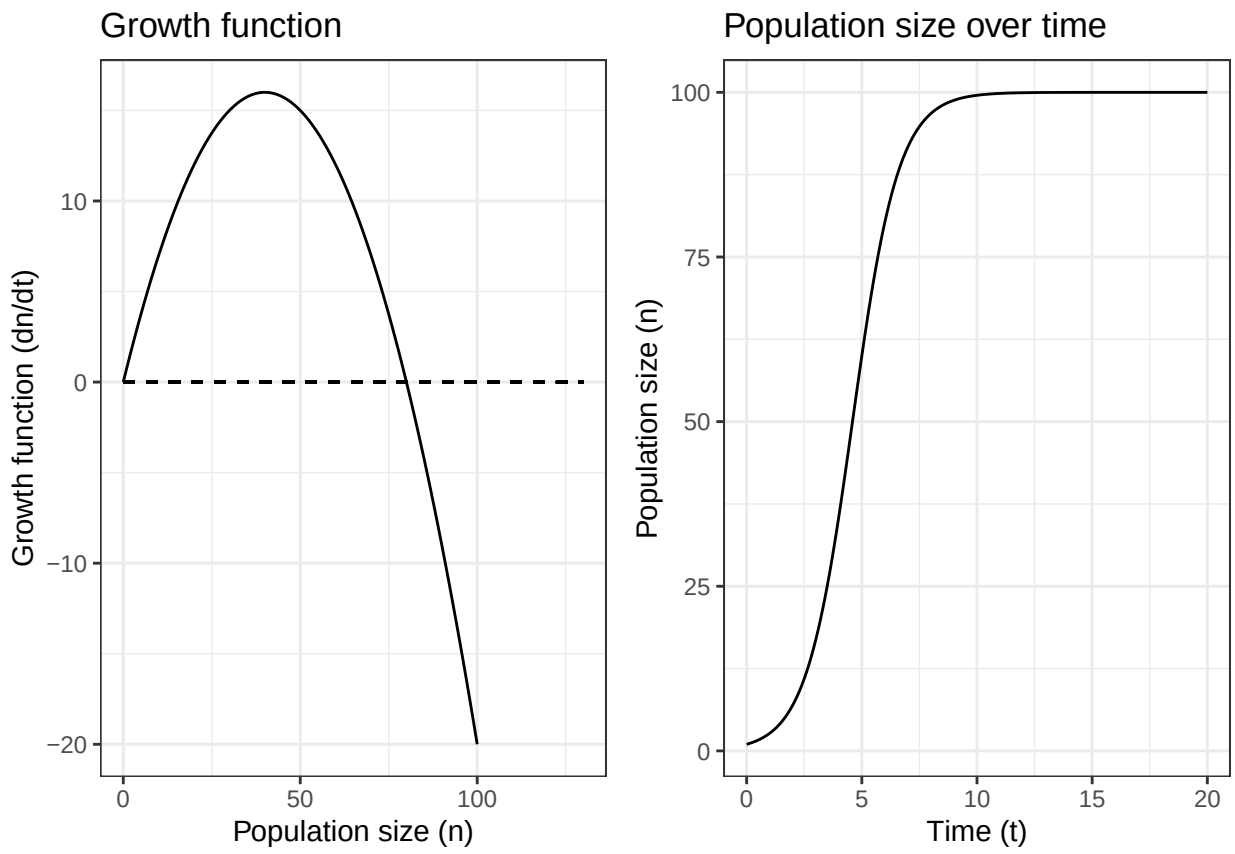
1 and 2. Plot the growth function against population size $n. As well as $n(t)$

Define the value of r_0 and K and plot the growth function against population size n. The logistic equation has an exact mathematical solution given by:

$$n(t) = \frac{K}{1 + (\frac{K}{n(0)} - 1)e^{-r_0 t}}$$

```
# Define parameters
P4 <- list(r0 = 1, K = 100, n0 = 1)

# Population values
```

```r
n_vec <- seq(0, 120)

# Time values
t_vec <- seq(0, 20, 0.1)

# Run the functions
dndt <- P4$r0*n_vec*(1-n_vec/P4$K)
nt <- P4$K/(1+(P4$K/P4$n0-1)*exp(-P4$r0*t_vec))

# Set up dataframes
df_dndt <- data.frame(n = n_vec, dndt = dndt)
df_nt <- data.frame(t = t_vec, nt = nt)

# Plot growth function in ggplot
dndt_plot <- df %>%
  ggplot(aes(x = n, y = dndt)) +
  geom_line() +
  geom_segment(aes(x = 0, y = 0, xend = 130, yend = 0),linetype = "dashed") +
  theme_bw() +
  labs(
    title = "Growth function",
    x = "Population size (n)",
    y = "Growth function (dn/dt)"
    )

# Plot n(t) in ggplot
nt_plot <- df_nt %>%
  ggplot(aes(x = t, y = nt)) +
  geom_line() +
  theme_bw() +
  labs(
    title = "Population size over time",
    x = "Time (t)",
    y = "Population size (n)"
    )

# Plot both plots
ggarrange(dndt_plot, nt_plot, nrow = 1, ncol = 2)
```



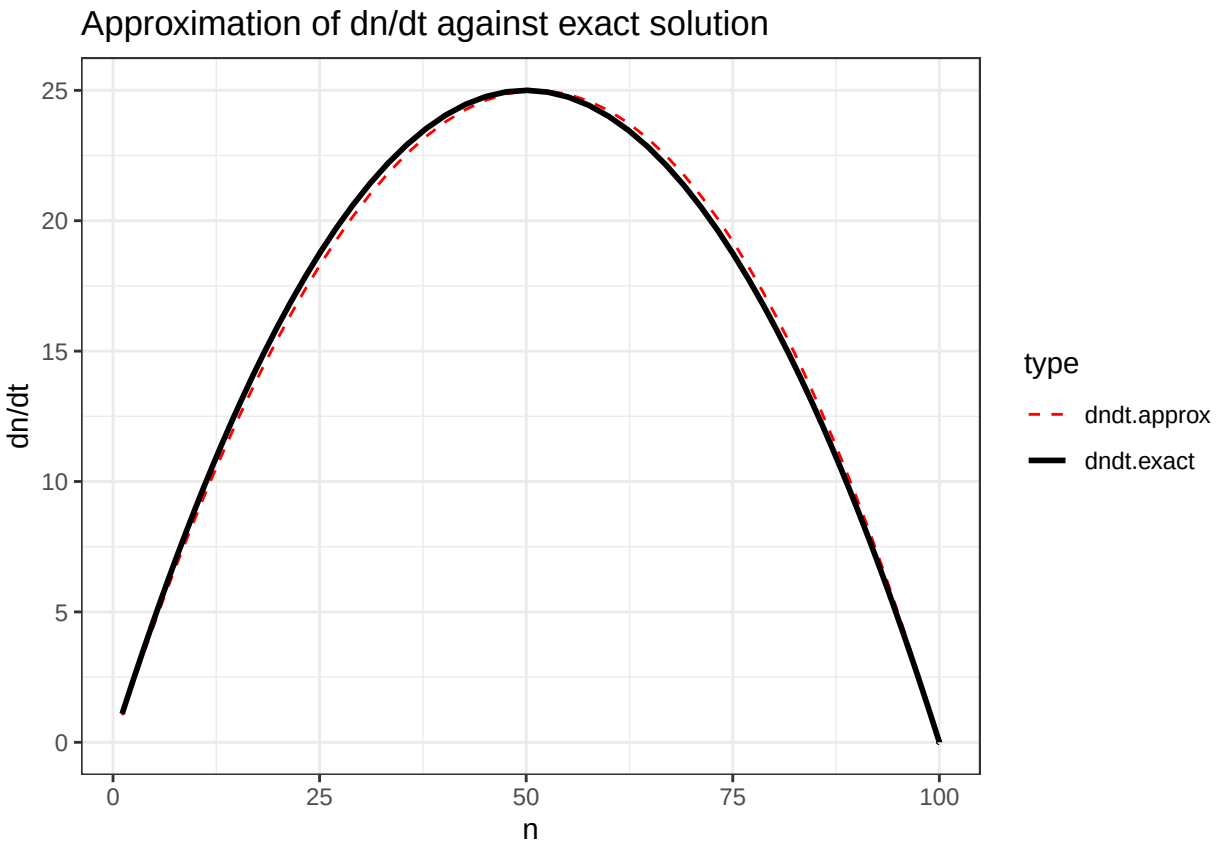
3. Make a numerical test of the solution.

In other words, calculate $\frac{dn}{dt}$ for different points on the curve in question 2. Make a suitable plot to compare the result to the correct values given by the model:

$$\frac{dn}{dt} = r_0 n(1 - \frac{n}{K})$$

```r
# Parameters
P4 <- list(r0 = 1, K = 100, n0 = 1)

# Time vector
t_vec <- seq(0, 20, 0.1)

# Calculate the n-values from n(t)
nt <- P4$K/(1+(P4$K/P4$n0-1)*exp(-P4$r0*t_vec))

# Calculate an approximation of dn/dt
dndt_approx <- diff(nt)/diff(t_vec)

# Calculate the exact dn/dt
dndt <- P4$r0*nt[-1]*(1-nt[-1]/P4$K)

# Make it as dataframe
df_dndt_approx <- data.frame(
```

```r
  n = nt[-1],
  dndt.approx = dndt_approx,
  dndt.exact = dndt
  )

# Make it to long format instead of wide
df_dndt_approx_long <- df_dndt_approx %>%
  pivot_longer(
    cols = c("dndt.approx", "dndt.exact"),
    names_to = "type",
    values_to = "value"
    )


# Plot the approximation against the exact solution
df_dndt_approx_long %>%
  ggplot(aes(x = n, y = value, color = type)) +
  geom_line(aes(linetype = type, size = type)) +
  scale_linetype_manual(
    values = c("dndt.approx" = "dashed", "dndt.exact" = "solid")) +
  scale_size_manual(values = c("dndt.approx" = 0.5, "dndt.exact" = 1)) +
  scale_color_manual(values = c("dndt.approx" = "red", "dndt.exact" = "black")) +
  theme_bw() +
  labs(
    title = "Approximation of dn/dt against exact solution",
    x = "n",
    y = "dn/dt"
    )
```



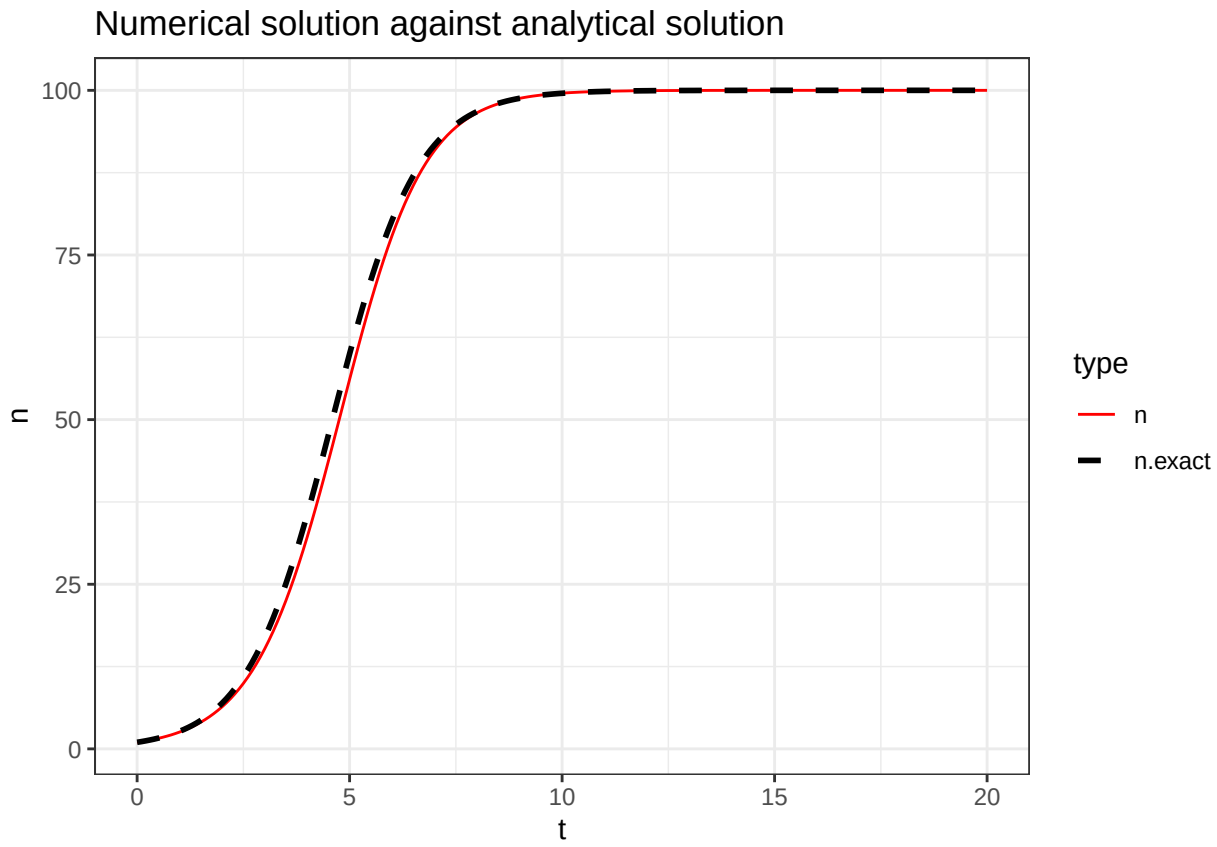
4. Finally, it is time to solve the logistic equation numerically. Use the procedure outlined in the lecture:

1. Start by setting $x = x_0, t = 0$.
2. Choose a small time step Δt.
3. Calculate $f(t, x)$.
4. Calculate $\Delta x = f(t, x)\Delta t$.
5. Update $x \leftarrow x + \Delta x$ and $t \leftarrow t + \Delta t$.
6. Repeat steps 3-5 until t reaches the desired value.

Does the solution match the analytical solution plotted in question 2?
Answer: Yes, it does.

```
# Parameters
P5 <- list(r0 = 1, K = 100, n0 = 1, dt = 0.1, t_init = 0, t_end = 20)

# Time values and vector for n values
t_vec <- seq(P5$t_init, P5$t_end, P5$dt)
n_vec <- rep(NA, length(t_vec))


# The steps to repeat as described in the question
for (i in 1:length(t_vec)) {
  if (i == 1) {
```

```
    n_vec[i] <- P5$n0
  } else {
    # Calculate dndt using the logistic growth model formula
    dndt <- P5$r0 * n_vec[i-1] * (1 - n_vec[i-1] / P5$K)

    # Calculate dndt times delta t
    delta_n <- dndt * P5$dt

    # Update n with the previous n plus delta n
    n_vec[i] <- n_vec[i-1] + delta_n
  }
}

# Get the analytical solution (aka the exact solution)
n_ana <- P5$K/(1+(P5$K/P5$n0-1)*exp(-P5$r0*t_vec))

# Make it to a dataframe
df_n <- data.frame(
  t = t_vec,
  n = n_vec,
  n.exact = n_ana
  )

# Make it to long format instead of wide
df_n_long <- df_n %>%
  pivot_longer(
    cols = c("n", "n.exact"),
    names_to = "type",
    values_to = "value"
    )

# Plot with ggplot
df_n_long %>%
  ggplot(aes(x = t, y = value, color = type)) +
  geom_line(aes(linetype = type, size = type)) +
  scale_linetype_manual(
    values = c("n" = "solid", "n.exact" = "dashed")) +
  scale_size_manual(values = c("n" = 0.5, "n.exact" = 1)) +
  scale_color_manual(values = c("n" = "red", "n.exact" = "black")) +
  theme_bw() +
  labs(
    title = "Numerical solution against analytical solution",
    x = "t",
    y = "n"
    )
```


Numerical solution against analytical solution
100
75
50
25
0
n
0
5
10
15
20
t
type
n
n.exact