Week 5 Question Sheet: Loops

Question 1

A common model of the growth of a population of phytoplankton (the microscopic forests of the oceans) is

$x_{n+1} = (1 + \mu)x_n - g\ x_n^2$

where x_n is the population size on day n, and μ and g are constants that describe the population's growth and losses. Over time the population size approaches an theoretical equilibrium value μ/g. (To see why this is so, look for the value of x_n that makes $x_n = x_{n+1}$.)

Here is repetitive code that implements this model for 5 days, using a starting population of 2 and $\mu = 0.6, g = 0.05$.

```
x0 = 2
x1 = (1 + 0.6) * x0 - 0.05 * x0^2
x2 = (1 + 0.6) * x1 - 0.05 * x1^2
x3 = (1 + 0.6) * x2 - 0.05 * x2^2
x4 = (1 + 0.6) * x3 - 0.05 * x3^2
x5 = (1 + 0.6) * x4 - 0.05 * x4^2
```

This code can be much improved. Answer *(a)–(f)* either all in one go or step by step as you prefer.

(a) Rewrite the code above to save population size as a vector `x`, rather than individual variables `x0`, `x1`, , ...

```
x = c()
x[1] = 2
x[2] = (1 + 0.6) * x[1] - 0.05 * x[1]^2
x[3] = (1 + 0.6) * x[2] - 0.05 * x[2]^2
x[4] = (1 + 0.6) * x[3] - 0.05 * x[3]^2
x[5] = (1 + 0.6) * x[4] - 0.05 * x[4]^2
x[6] = (1 + 0.6) * x[5] - 0.05 * x[5]^2
```

(b) Rewrite the code in a less repetitive way using a `for` loop.

```
x = c()
x[1] = 2
for (n in 1:5) {
  x[n+1] = (1 + 0.6) * x[n] - 0.05 * x[n]^2
}
```

(c) Rewrite the code as a function that takes x_0, μ, and g as input and returns the vector `x`.

```
phyto_growth <- function(x0, mu, g) {
   x = c()
   x[1] = x0
   for (n in 1:5) {
     x[n+1] = (1 + mu) * x[n] - g * x[n]^2
   }
   return(x)
}
```

(d) Modify this function to take a fourth input, number of time steps N, and calculate `x` for that many steps.

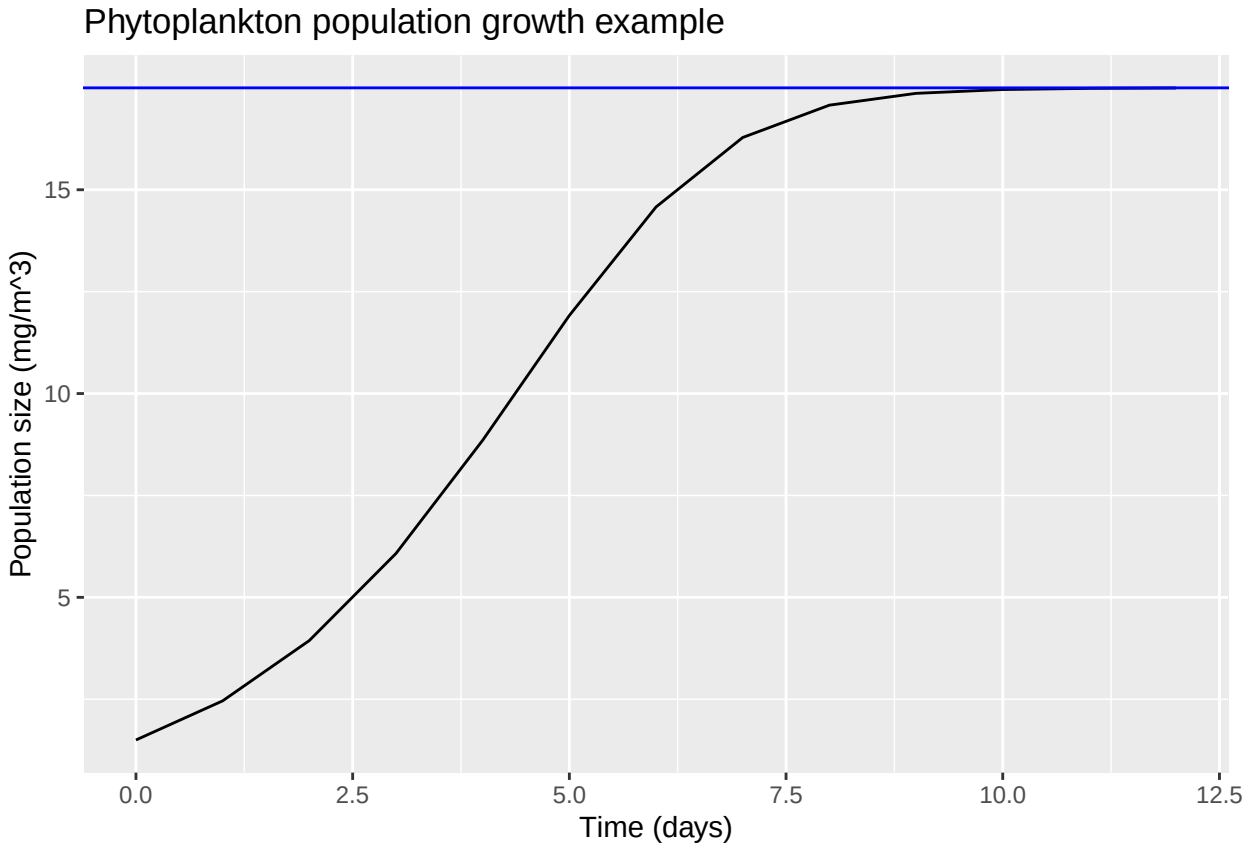
```
phyto_growth <- function(x0, mu, g, N) {
   x = c()
   x[1] = x0
   for (n in 1:N) {
     x[n+1] = (1 + mu) * x[n] - g * x[n]^2
   }
   return(x)
}
```

(e) Finally, modify your function so that it returns a named list with three elements: the vector `x`, the corresponding vector of time points $0, 1, 2, ...N$, and the theoretical equilibrium value.

```
phyto_growth <- function(x0, mu, g, N) {
   x = c()
   x[1] = x0
   for (n in 1:N) {
     x[n+1] = (1 + mu) * x[n] - g * x[n]^2
   }
   xeq = mu/g
   list(x=x, time=0:N, xeq=xeq)
}
```

(f) Use your function to make a plot of population size vs. time for 12 days, with initial population 1.5, $\mu = 0.7$, and $g = 0.04$. Add a horizontal line that indicates the theoretical equilibrium. Label the axes and give the plot a title.

```
   mylist = phyto_growth(1.5, 0.7, 0.04, 12)
   ggplot() + geom_line(aes(mylist$time, mylist$x)) +
     geom_abline(slope=0, intercept=mylist$xeq, color = "blue") +
     labs(x="Time (days)", y="Population size (mg/m^3)",
          title="Phytoplankton population growth example")
```



Question 2

In lecture we make a correlation matrix for the 4 continuous variables in the Palmer Penguins dataset. The matrix comes out with 4×4 elements, but because the main diagonal of a correlation matrix is always 1, and because a correlation matrix is always symmetrical around the main diagonal, only 9 of the 16 elements are actually informative. Use a nested loop to replace the non-informative elements with NA.

```
require(palmerpenguins)
```

```
## Loading required package: palmerpenguins
```

```
data(penguins)
selectedVars = penguins |> select(bill_length_mm : body_mass_g) # select 4 columns
palmer_r = cor(selectedVars, use = "complete.obs")
print("before:")
```

```
## [1] "before:"
```

```
palmer_r
```

```
##                   bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
## bill_length_mm         1.0000000    -0.2350529         0.6561813   0.5951098
## bill_depth_mm         -0.2350529     1.0000000        -0.5838512  -0.4719156
## flipper_length_mm      0.6561813    -0.5838512         1.0000000   0.8712018
## body_mass_g            0.5951098    -0.4719156         0.8712018   1.0000000
```

```
for (i in 1:4) {
  for (j in 1:i) {        # these are the columns to blank out with NA on row i
    palmer_r[i,j] = NA
```

```
  }
}
print("after:")
```

```
## [1] "after:"
```

```
palmer_r
```

```
##                    bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
## bill_length_mm                 NA    -0.2350529         0.6561813   0.5951098
## bill_depth_mm                  NA            NA        -0.5838512  -0.4719156
## flipper_length_mm              NA            NA                NA   0.8712018
## body_mass_g                    NA            NA                NA          NA
```

Question 3

(a) Write a function that returns the first n Fibonacci numbers. (See the Wikipedia page if you are unfamiliar with the Fibonacci sequence.)

```
fibo <- function(n) {
  x = c()
  x[1] = 1
  x[2] = 1
  for (i in 3:n) {
    x[i] = x[i-1] + x[i-2]
  }
  return(x)
}

# test it
fibo(10)
```

```
##  [1]  1  1  2  3  5  8 13 21 34 55
```

(b) (optional) Write an alternate function that uses a `while` loop to return all the Fibonacci numbers that are smaller than a maximum value. For example, the function should return the first 30 (up through 832,040) if the user specifies 1,000,000 as the maximum.

```
fibo2 <- function(maxval) {
  x <- c()
  x[1] <- 1
  x[2] <- 1
  i <- 3
  while(x[i-1] + x[i-2] < maxval) {
    x[i] <- x[i-1] + x[i-2]
    i <- i + 1
  }
  return(x)
}

# test it
fibo2(1e6)
```

```
##  [1]      1      1      2      3      5      8     13     21     34     55
## [11]     89    144    233    377    610    987   1597   2584   4181   6765
## [21]  10946  17711  28657  46368  75025 121393 196418 317811 514229 832040
```

Question 4

A simple method for smoothing a time series is what's known as a "three-point smoother": you replace each point x_i with the average of x_{i-1}, x_i, and x_{i+1}. This is related to the idea of a running mean.

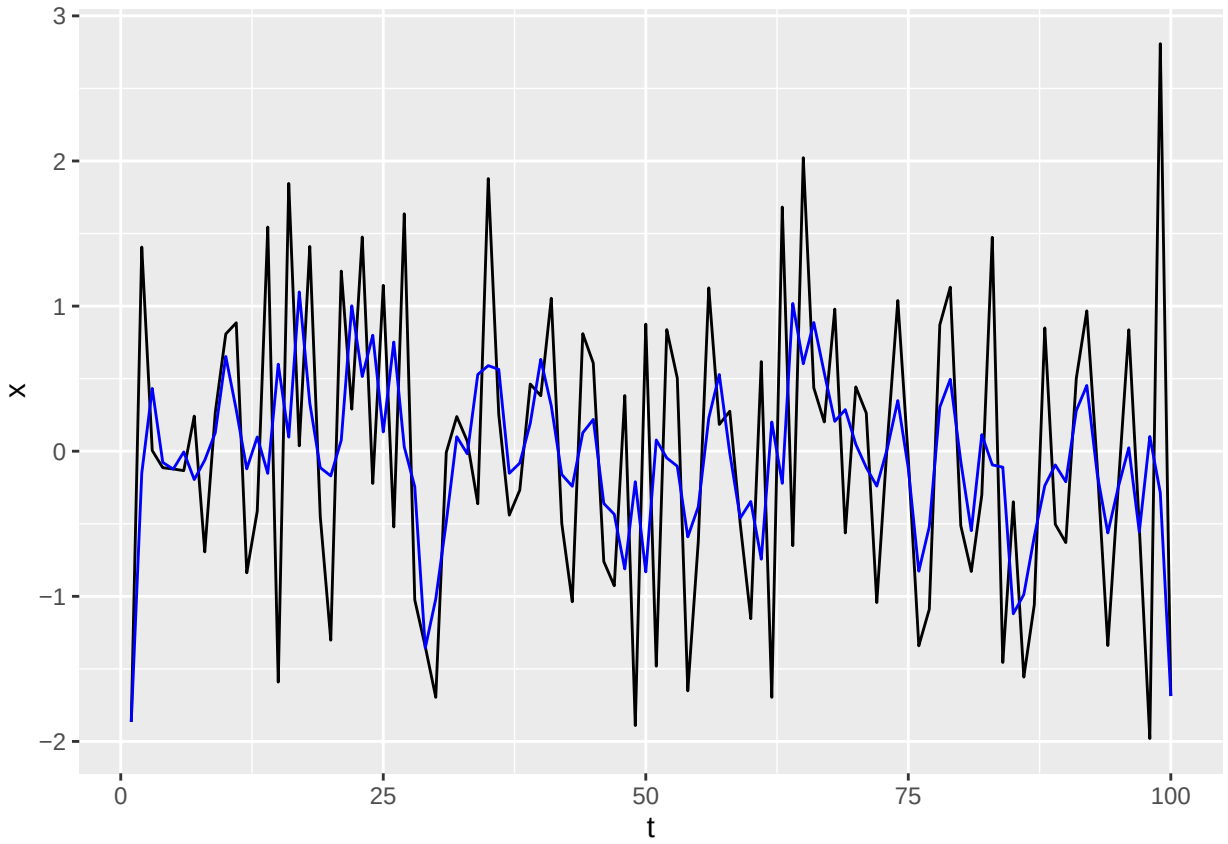
(a) Use the `rnorm` function to generate a 100-point-long random dataset to work with as an example.

```
x = rnorm(100)
```

(b) Write a function that performs a three-point smoothing on an input vector. For simplicity, leave the first and last points unchanged. (There are two ways to do this: with a `for` loop, or with no loop and clever indexing.)

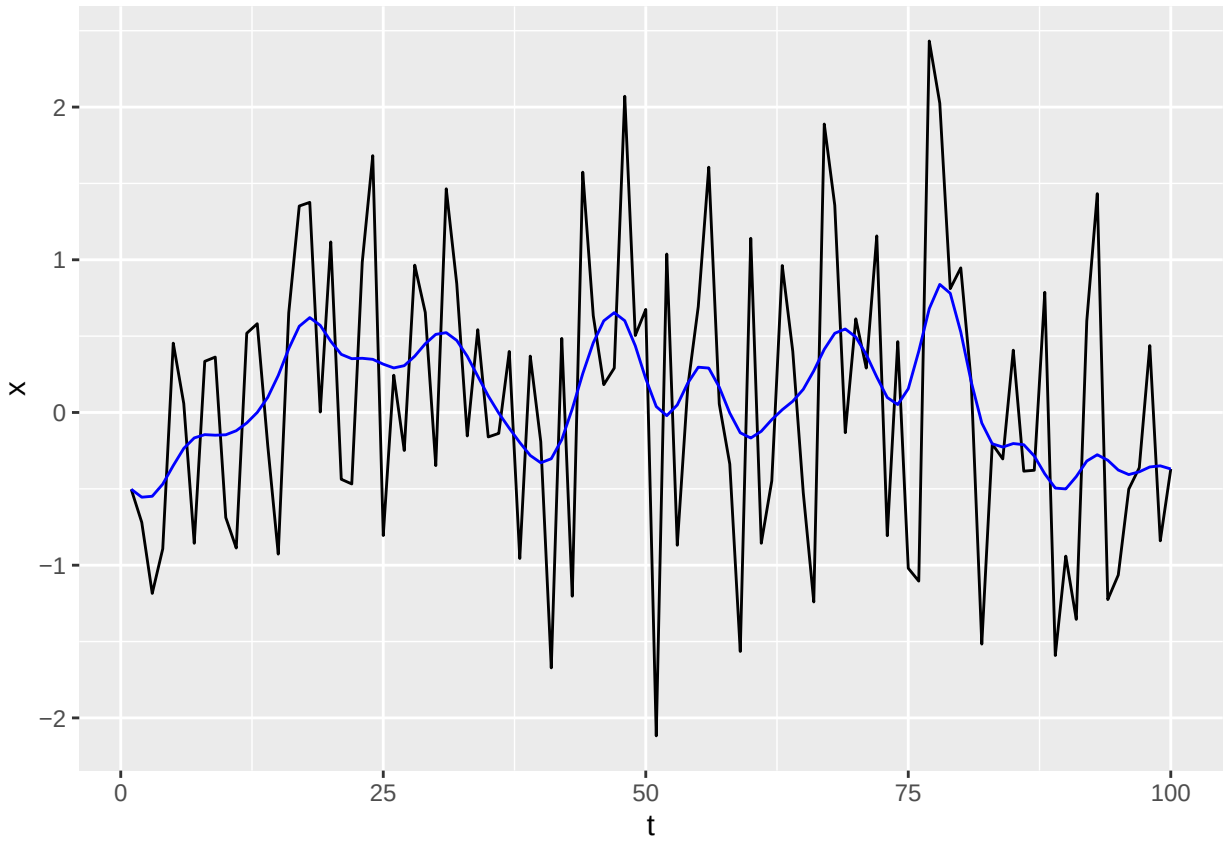
Test it using the random data you generated in (a), and plot both the original and smoothed versions of the data on one set of axes.

```
# for-loop version
smoother <- function(x) {
  n = length(x)
  xs = rep(NA, n) # blank vector of the right length to hold the results
  xs[1] = x[1]
  xs[n] = x[n]
  for (i in 2:(length(x)-1)) {
    xs[i] = (x[i-1] + x[i] + x[i+1]) / 3
  }
  return(xs)
}

# no-loop version
smoother <- function(x) {
  n = length(x)
  xs = rep(NA, n) # blank vector of the right length to hold the results
  xs[1] = x[1]
  xs[n] = x[n]
  xs[2:(n-1)] =
    ( x[1:(n-2)] + x[2:(n-1)] + x[3:n] ) / 3
  return(xs)
}

# test
x = rnorm(100)
xs = smoother(x)
t = 1:length(x)
ggplot() + geom_line(aes(t, x), color="black") +
  geom_line(aes(t, xs), color="blue")
```

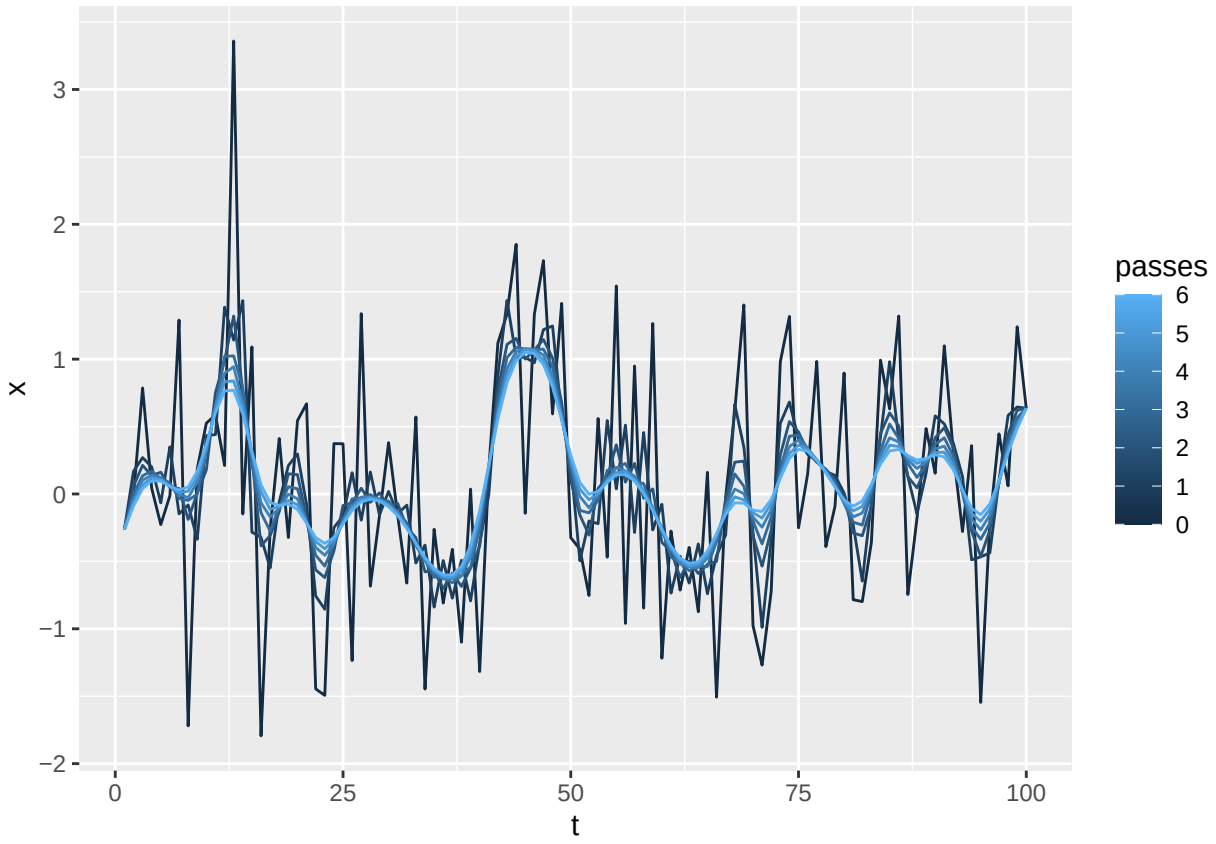


(c) To get extra smoothness, you can pass the three-point smoother over the same data more than once. Modify your function so that it takes N as an additional input and runs the smoother N times. *Optional*: make a plot that shows not just the initial and final time series, but all the steps in between, using a continuous color scale to represent the nubmer of smoother-passes.

```
smoother <- function(x,Npasses) {
  n = length(x)
  xs = rep(NA, n) # blank vector of the right length to hold the results
  xs[1] = x[1]
  xs[n] = x[n]
  xprevious = x
  for (j in 1:Npasses) {
    xs[2:(n-1)] =
    ( xprevious[1:(n-2)] + xprevious[2:(n-1)] + xprevious[3:n] ) / 3
    xprevious = xs
  }
  return(xs)
}

# test
x = rnorm(100)
xs = smoother(x,6)
t = 1:length(x)
ggplot() + geom_line(aes(t, x), color="black") +
  geom_line(aes(t, xs), color="blue")
```




```
# optional version, that overlays 0,1,2,3,4,5,6 passes of the smoother.
x = rnorm(100)
t = 1:100
# we will save all the results in a tidy data frame: time step t, actual data x,
# and the number of passes that were used
df = data.frame(t = t, x = x, passes = rep(0,length(x)))
for (Npasses in 1:6) {
  xs = smoother(x,Npasses)
  df = rbind(df,
             data.frame(t = t, x = xs, passes = rep(Npasses,length(x))))
}
# now have a 700-point-long data frame (passes = 0...7, a version of x for each one)
# we want to plot 7 lines on top of each other, but because there is a single named
# column (passes) that distinguishes the data that goes into each line, we can
# make the whole plot in one line:
ggplot(df) + geom_line(aes(t,x,group=passes,color=passes))
```




```
# ggplot is so cool (if your data frame is tidy).
```