One-Species Logistic Growth Parameter Estimation with Tikhonov Regularization

Zane Billings

08 May, 2020

Abstract

Existing competition models for the human gut microbiome typically use several species at once, raising an interesting question: if we consider pairwise combinations of these species, will we get the same results as in the larger model? Previous work by the HMBG at Warren Wilson College and Western Carolina University have attempted to solve this problem, but have encountered substantial error in trying to fit parameters to time-series data. In order to hone in on the source of the error, we examine the single species modeling case using the same least squares machinery as previous results, and additionally examine the application of Tikhonov regularization to the problem. We find that small amounts of error deviating from the expected trend in time-series data leads to serious errors in estimating both the intrinsic growth rate and carrying capacity, but when the model is converted to its dimensionless form, we can recover the intrinsic growth rate from noisy data by applying Tikhonov regularization.

Introduction

The main goal of my research was to investigate errors in the use of an ordinary least squares regression method for fitting parameters to the one-species logistic growth mode.

One-Species Logistic Growth

The one-species logistic growth model describes the change in the population size of a species with a given intrinsic growth rate, r, and a carrying-capacity, K. Mathematically, the model is given as

$$\frac{dP}{dt} = rP(1 - \frac{P}{K}),$$

and its analytic solution is

$$P(t) = \frac{KP_0}{P_0 + (K - P_0)e^{-rt}},$$

where P_0 is the initial population size at time $t = 0$ (Lipkin and Smith 2001).

The classical example of the validity of the logistic growth model is P.F. Verhulst's logistic fit of U.S. census data. In 1840, he used census data from the previous five censuses (1790 - 1840) to predict the population of the United States in 1940, with a surprisingly accurate result. Verhulst's results are reproduced in Lipkin and Smith (2001), but as I do not read French, I could not repeat them myself from the source material. Pearl and Reed also provided a similar model of census data (Pearl and Reed 1920), although both of these logistic models became inaccurate over time. An ecological application of the model occurs in Alliende and Harper (1989), where the growth of willow trees was modeled after rabbits were removed from a specific area in Australia.

The model assumes that carrying capacity is constant, the population is not age-stratified (i.e., each individual has the same probability of dying or reproducing) and furthermore, all members of the population experience the same effects of density, birth and rate rates are linear functions of population size–also referred to as linear density dependence of the population, no stochastic effects act on the population, and finally, density

has an instantaneous effect on the population (Rockwood and Witt 2015, 46). Rockwood and Witt (2015 cp. 2) provides several more examples of populations which seem to fit these criteria.

Noting that such microorganisms as *Paramecium sp.*, yeasts, and diatoms (again, from (Rockwood and Witt 2015)) tend to follow logistic growth curves when grown in the lab, the idea that bacteria follow this model is not unreasonable (although the careful biologist will note that all three provided examples are eukaryotes and are much larger in size than bacteria).

Two-Species Logistic Growth

The overarching goal of the research group is to analyze the model posed by (Stein et al. 2013), which is an n-dimensional Lotka-Volterra competition model between several species. We wish to analyze whether a subset of the larger model will produce the same results. That is, if we take a pair of species analyzed in the model using all species, will a model created with just that pair of species give the same results?

In order to analyze competition between two species of gut bacteria, we apply the Lotka-Volterra model of interspecies competition:

$$\frac{dP_1}{dt} = r_1 P_1 \left(1 - \left(\frac{P_1 + \alpha_{12} P_2}{K_1}\right)\right)$$
$$\frac{dP_2}{dt} = r_2 P_2 \left(1 - \left(\frac{P_2 + \alpha_{21} P_1}{K_2}\right)\right)$$

where P_i is the population size, r_i is the intrinsic growth rate, and K_i is the carrying capacity, all of species i. The α_{ij} term refers to the competitive effect of species j on species i (Rockwood and Witt 2015). It is typically convenient to write the Lotka-Volterra equations in a simplified form:

$$\frac{dx}{dt} = ax + bxy$$
$$\frac{dy}{dt} = cy + dxy$$

where a and c represent intrinsic rates of growth and b and d represent the overall effect of competition on a species.

The Error Problem

Given exact data, previous results (from HMBG) have shown that parameters can be fitted to the two-dimensional model using an ordinary least-squares method given by (Stein et al. 2013). However, when Gaussian noise (with a mean equal to 0 and a standard deviation of 4% of the equilibrium value of the system) is introduced, predictions become much less reliable.

In order to examine this error further, I have analyzed this method in the one-dimensional case.

One-Dimensional Least Squares Method

Given the logistic equation described above, we wish to construct a least squares problem which will predict the values of r and K given time series data representing the population size of a population following logistic growth over time. Expanding the logistic growth equation as given earlier, we have that

$$\frac{dP}{dt} = rP(1 - \frac{P}{K}) = rP - \frac{r}{k}P^2,$$

and thus

$$\frac{dP}{dt}\frac{1}{P} = \frac{d\ln P}{dt} = r - \frac{r}{k}P,$$

assuming that $P \neq 0$, since our problem would not be very interesting if this were the case. This method is used by Stein et al. (2013) in this way, and is also used by Kloppers and Greeff (2013). Finally, we discretize the system in order to obtain a discrete-time dynamical system rather than a continuous differential equation model, and we get

$$\frac{d \ln P}{dt} \approx \frac{\ln(P(t_{i+1})) - \ln(P(t_i))}{t_{i+1} - t_i} = \frac{\ln\left(\frac{(P(t_{i+1}))}{P(t_i)}\right)}{\Delta t_i} = \frac{1}{\Delta t} \ln\left(\frac{(P(t_{i+1}))}{P(t_i)}\right).$$

Then, returning to our model, we have

$$\frac{1}{\Delta t} \ln\left(\frac{(P(t_{i+1}))}{P(t_i)}\right) = r - \frac{r}{K} P(t_i)$$

Now, denote $P(t_I)$ as P_i for all i. We then convert to a matrix equation to get

$$\frac{1}{\Delta t} \ln\left(\frac{P_{i+1}}{P_i}\right) = \begin{bmatrix} 1 & -P_i \end{bmatrix} \begin{bmatrix} r \\ \frac{r}{k} \end{bmatrix},$$

which is an equation of the form

$$\vec{b} = A\vec{x}.$$

So, applying the ordinary least squares method, we get that the optimal parameters are given by

$$\vec{x} = (A^T A)^{-1} A^T \vec{b}.$$

And this is the least squares method we have explored over the semester. After developing this model, we also considered a model with smoothing (a.k.a. Tikhonov regularization) applied, which can help correct parameter estimations by forcing the parameter values to be smaller. The smoothed least squares method we used has a solution given by

$$\vec{x} = (A^T A - \lambda I)^{-1} A^T \vec{b},$$

where $\lambda \in \mathbb{R}$ (Gockenbach 2016).

My Goals

Over the course of the year, my goals were to:

- Develop `R` code to generate time-series logistic growth data for one species, and to fit the ordinary least squares model (as described above) to this data;
- Using these codes, examine the error in the one species logistic growth model, and determining if smoothing (Tihkonov regularization) was effective in obtaining better parameter estimates; and
- Examine the dimensionless one species logistic growth model, which only has one parameter, determine if this model suffers from the same error problem, and examine the effect of smoothing.
- Implement *k*-fold cross validation using the method of Stein et al, and apply cross validation to determine the optimal smoothing value for a model.
- Determine the best methods for applying the OLS model to data.

Results

R Codes for Data Generation and Fitting

All scripts which I created in Fall 2019 can be found on the Github page.. All of the scripts mentioned below likely rely on the functions in the `Helpers.R` script file. I am working on an improved pipeline which is simpler to implement, but I do not know how long it will be before this is finished; ideally I will be able to streamline both the generation/fitting process and the usability and portability of the functions.

I have successfully created several methods for generating time-series one species logistic growth data. These functions are provided in the `Sample_data_generation.R` file, which can be sourced into any R or Rmd file fairly easily. Data can be generated with the following methods:

- The analytic solution to the logistic growth ODE.
- The Euler discretization to the logistic growth ODE.
- A numerical ODE solver, specifically `deSolve::ode()`.
- All of these methods can be adjusted to include Gaussian noise.

Additionally, I've generated a suite of functions for fitting the least squares model to the generated data. These methods are provided in the `Least_squares_methods.R` script.

- There is a function which takes in prepped data (prepared via `prep_data()` in the `Helpers.R` script) and fits the model to all of the data, as you would do normally.
- The data can be fitted using only the data either before or after the inflection point (which we used to experiment with fitting).
- There is a function for fitting with smoothing that also accepts the lambda parameter for implementing Tikhonov regularization.
- Eventually there will be a function for fitting a model to a random sample of the data points, but this is not currently implemented.

The `Dimensionless_exploration.R` file provides several functions which do the same thing as the above listed functions, but for the dimensionless one-parameter model rather than the standard two-parameter model.

The HMBGR Package

In Spring 2020, I updated the code repository to a more user-friendly R-package design. The Lawson-380 repository still contains all of the function scripts, the R-markdown document for this manuscript, and all of the journals and tests I performed over the course of the year. The `HMBGR` package can be installed and loaded with the following code.

```
if (require('devtools')==FALSE) {install.packages('devtools', repos="https://cran.rstudio.com")} else {
devtools::install_github("wz-billings/HMBGR")
library(HMBGR)
```

After being installed once, the HMBGR package can be loaded like any other `R` package. I wrote some documentation for all of the included functions, but the second half of this document should show more examples of how the functions can be used. (I did not update the earlier portions of the manuscript to use `HMBGR` code, but I did start using it in the latter portion.)

Two-Parameter Fitting and Smoothing

The results of the two-parameter model fitting tests are summarized in the `Tests/Noisy_fitting_test.R` file.

In the plot below, the left column shows the logistic time series data, and the right column shows $1/b_i$ vs. P_i, which is actually the line we are obtaining the equation of using our least squares model. (Note that for future work, the plots should be cleaned up to have proper titles and annotations!)

The top two plots are for data generated using the analytic solution with no noise, the middle two plots have 0.04% noise, and the bottom two plots have 4% noise. I also generated the last case with 10% noise just to really show the problem: the noise messes the computation up so badly that we get an NaN warning! (If you are interested in seeing this, this result is in the mentioned `R` script, but I commented it out due to errors being produced!) The parameters used to generate these data are $r = 0.2; k = 1000$ with time steps of 0.1 and an initial population size of 100.

```
source(here::here("Tests","Noisy_fitting_test.R"))
```

```
## Loading required package: deSolve
```

```
## Loading required package: MASS
```

```
## No noise
```

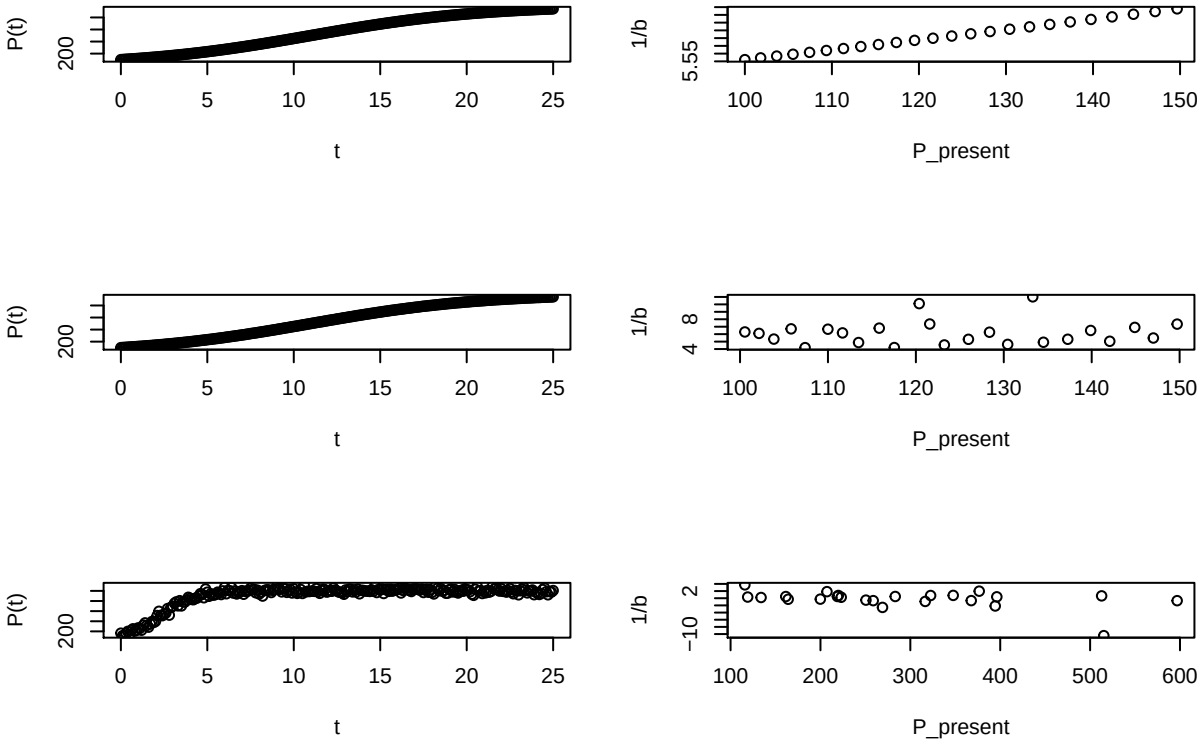
```
## The estimated growth rate is: 0.199969
## The estimated carrying capacity is: 992.3672
```

```
##
## 0.04% noise
```

```
## The estimated growth rate is: 0.2032465
## The estimated carrying capacity is: 792.7341
```

```
##
## 4% noise
```

```
## The estimated growth rate is: 1.329709
## The estimated carrying capacity is: 450.7519
```



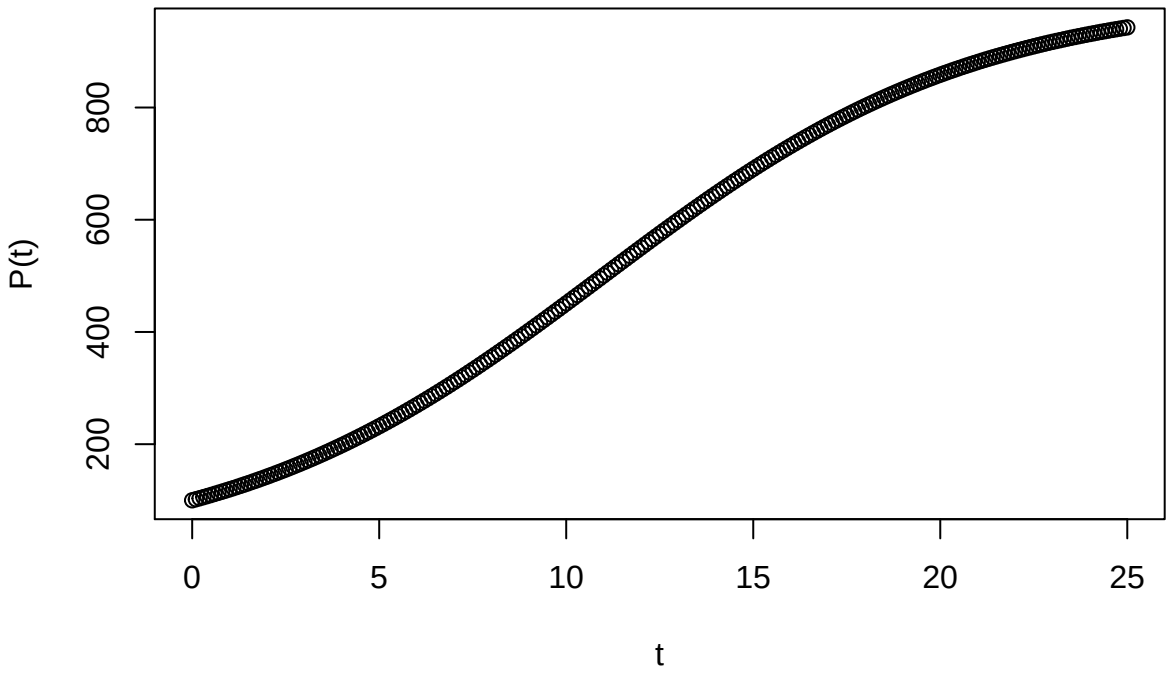
From the outputted results and the plots, we can make a few observations.

1. In the case with no noise, we get the population parameters back almost exactly. They aren't *exact* fits, but they are really very close.
2. In the case with low noise, our results are very close to the results with no noise, and are thus very close to the population parameters. However, notice that while r is very close to correct, K got worse faster.
3. In the case with high noise, the fits are not very good at all! Notably, the fit for r is super overestimated, and the fit for K is super underestimated.
4. In the linear plots (the right column), we can see that the points move further away from the line as we get to higher values of i; error is increasing over time. We currently believe that this is due to the nature of the inverse problem, and the error is magnified by the log discretization and the reciprocals in the parameter calculations, which have an effect of magnifying the error.
5. If we go to even 10% noise, our results stop making sense because we end up taking the log of negatives, and having negatives here doesn't really make a lot of sense anyways.
6. However, all of our results are still in the range that give us the correct stability condition, which is good.

Now let's examine the case with smoothing. (A lot of this code will be copied from the file `Tests\Smoothing_test.Rmd`; I ported it into another `R` script for the sake of conciseness here.) Here, I generated the data with no noise, and with 4% Gaussian noise, since this amount was problematic enough to give poor estimates, but not problematic enough to break the discretization machinery.

First, let's see what happens when we apply smoothing to data without added noise. I used the same parameters as before ($r = 0.2; k = 1000$) to generate these data.

```
source(here::here("Tests","Smoothing_demo_no_noise.R"))
```

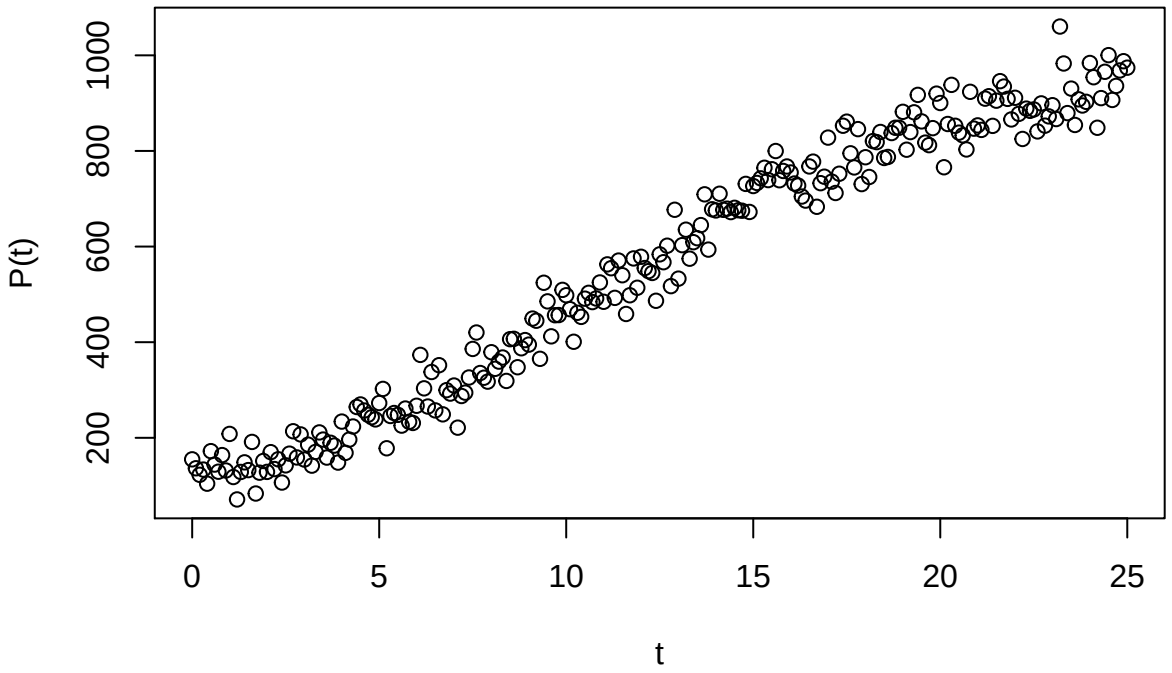



```
## Regular; no noise.
## The estimated growth rate is: 0.199969
## The estimated carrying capacity is: 992.3672
## Smoothing = 0.001; no noise
## Smoothing = 0.1; no noise
## Smoothing = 10; no noise
```

Just as we would expect, if we apply smoothing to a case where the fit is already accurate, we get worse results. Smoothing forces the parameters to be smaller than the ordinary least squares estimate, so if we force the parameters to be smaller than a correct fit, of course we get a mess.

Now, given that the data with 4% Gaussian noise was messy enough to give a bad fit, but not messy enough to break the discretization machinery, let's apply smoothing to this case.

```
source(here::here("Tests","Smoothing_demo_with_noise.R"))
```




```
## Regular; 4% noise.
## The estimated growth rate is: 12.87121
## The estimated carrying capacity is: 137.5314
## Smoothing = 0.001; 4% noise
## The estimated growth rate is: 12.85893
## The estimated carrying capacity is: 137.5241
## Smoothing = 0.1; 4% noise
## The estimated growth rate is: 11.74947
## The estimated carrying capacity is: 136.8113
## Smoothing = 1; 4% noise
## The estimated growth rate is: 6.584683
## The estimated carrying capacity is: 130.6546
## Smoothing = 10; 4% noise
## The estimated growth rate is: 1.220318
## The estimated carrying capacity is: 90.10552
## Smoothing = 15; 4% noise
## The estimated growth rate is: 0.8400849
## The estimated carrying capacity is: 76.85432
```

This is an interesting result! We see that as we increase the value of the smoothing parameter we get closer and closer to the true value of `r`. However, we make a sacrifice here! Since the value of K is already underestimated by the model, smoothing does not help us!

Smoothing forces parameters to be smaller, so if we are biased towards underestimating a parameter, smoothing does not solve our problem. We attempted to deal with this by moving to the dimensionless version of the one-species logistic growth model.

One-Parameter Reduction

A common technique for dealing with data that have a maximum value (in this case, the carrying capacity), or when the relative values matter more than the absolute values, in the sciences is to normalize the data. Since the carrying capacity is the only attracting fixed point of the one-species logistic growth model, so long as the data go above zero, we assume that the data will eventually arrive at the carrying capacity. Later on, I will examine the numerical derivative of some data to try and understand if the data can approximate the steady-state well. In either case, a visual examination of either the data or the numerical derivative should be sufficient to determine if the value of the carrying capacity can be approximation from the data or not. Methods for the estimation of K will be discussed later.

We can apply this concept to our model. Say we normalize our model by dividing both sides of our ODE by the maximal value, K. We get

$$\frac{dP}{dt}\frac{1}{k} = \frac{rP}{K}\left(1 - \frac{P}{K}\right).$$

Now, we make the substitution $x = \frac{P}{k}$. We get

$$\frac{dx}{dt} = rx(1 - x),$$

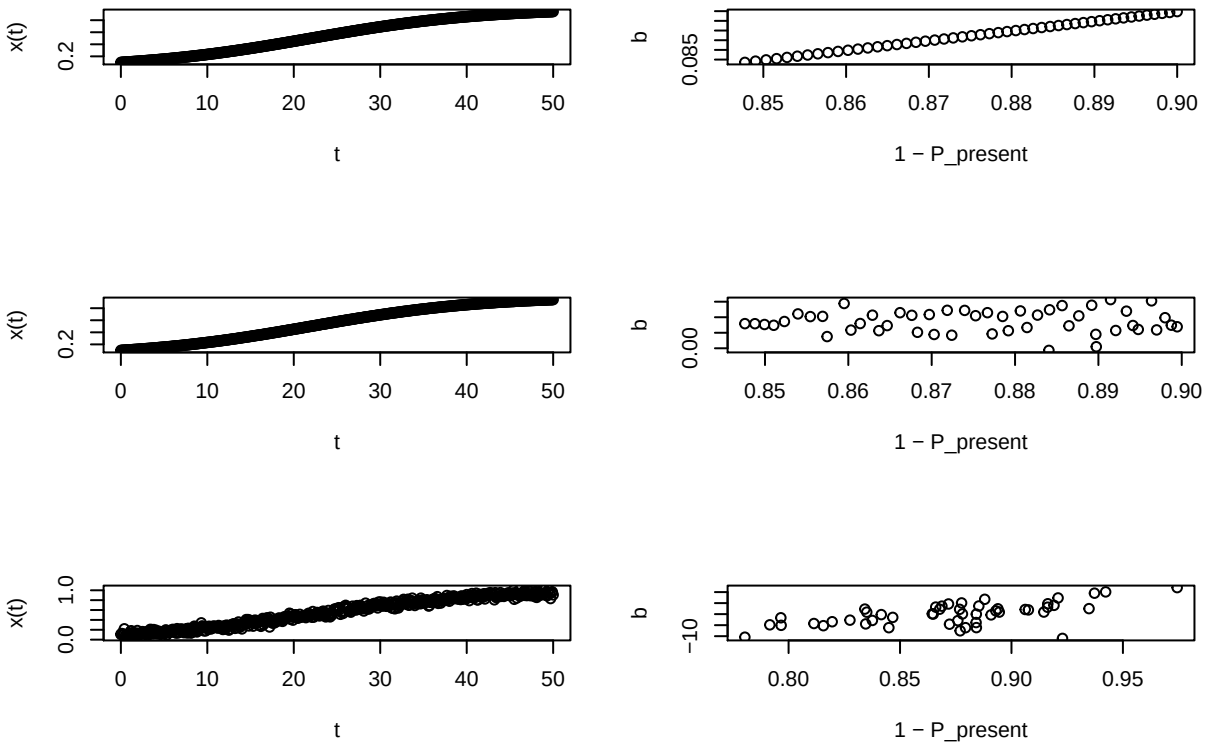
which notably only has one parameter, r, instead of the two parameters we were dealing with before. The easiest way to visualize this model is to think of the model describing the population size as a proportion of the carrying capacity, rather than describing the actual population size as a number of individuals. In effect, our carrying capacity has been normalized to 1.

I have rewritten many of the functions used in the two-parameter case to fit this case, and they can be found in the `Dimensionless_exploration.R` script.

Now that we have established our dimensionless model, let's look at some of the same examples. First, let's look at the regular comparison between non-noisy and noisy data. For these models, I had to adjust which parameters to use, so I will be using $r = 0.1, P_0 = 0.1$, with a time step of 0.1.

```
source(here::here("Tests", "Noisy_fitting_demo_1D.R"))
```

```
## No noise

## The estimated growth rate is: 0.09993782

##
## 0.04% noise

## The estimated growth rate is: 0.09880704

##
## 4% noise

## The estimated growth rate is: 0.2278902
```

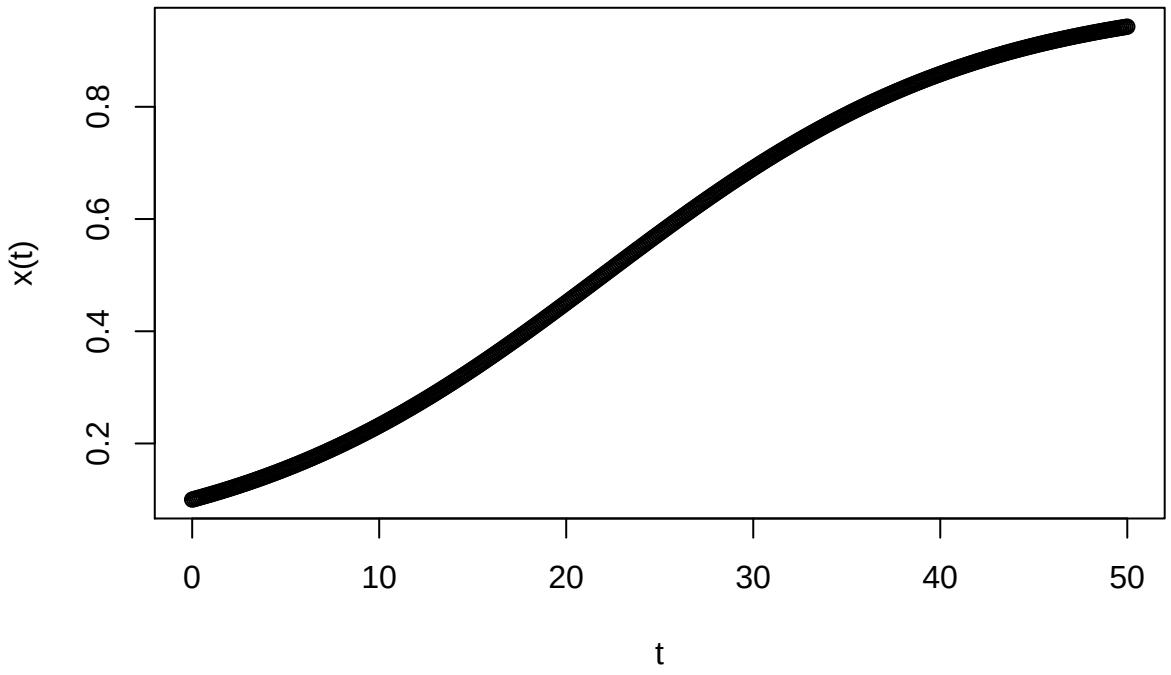


Wow, this time our predictions are actually a lot better and less affected by noise! However, we still get the same issue where if we increase noise too much we have to deal with NaN values–using anything around 0.1 or higher I ended up with NaNs; that makes sense, sense this is the initial condition. So, our r prediction in the 4% Gaussian noise case still isn't great, but it's honestly better than I was expecting when I started messing with the dimensionless model.

Notably, the problem here lies in the noise generation. All values have to be greater than zero for the model to work since the discretization relies on a log transform. All real data should be fine.

Now, let's look at the case with smoothing when there is no noise. I used the same parameters as above (i.e. $r = 0.1$).

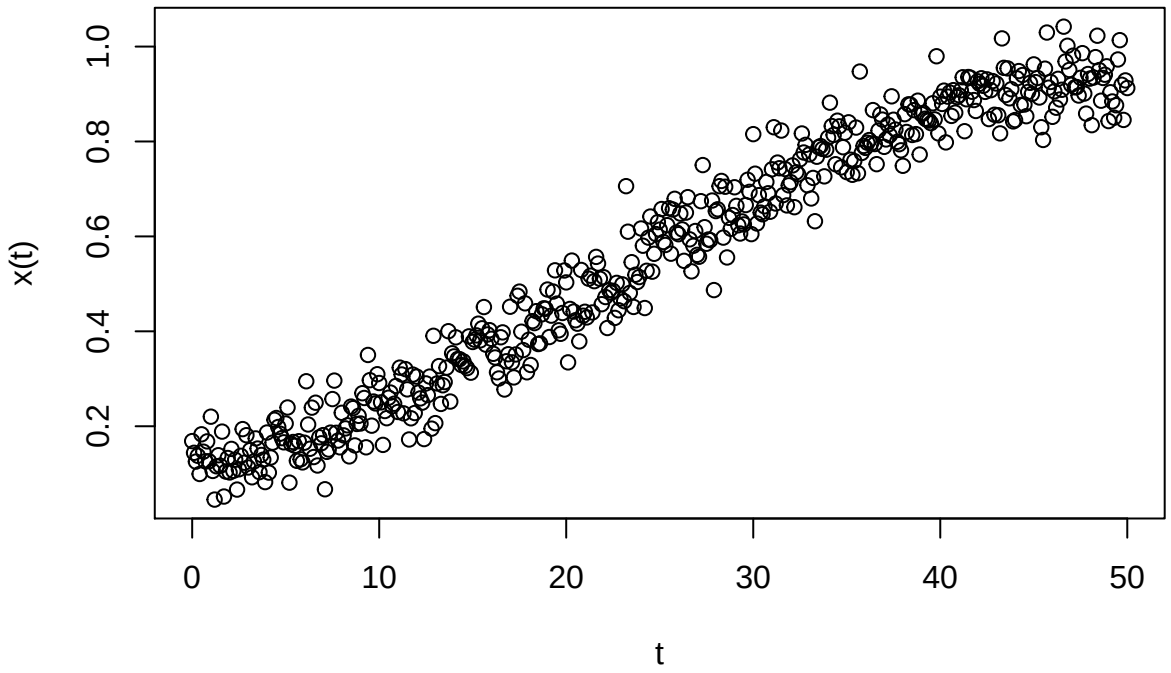
```
source(here::here("Tests", "1p_Smoothing_NN_demo.R"))
```




```
## Regular; no noise.
## The estimated growth rate is: 0.09993782
## Smoothing = 0.001; no noise
## Smoothing = 0.01; no noise
## Smoothing = 0.1; no noise
## Smoothing = 1; no noise
## Smoothing = 10 ; no noise
```

Foremost, we can see the same result as last time: if we apply low smoothing, not much changes. Once we get to high values of smoothing, our value of `r` begins to be depressed by smoothing and our fit becomes less accurate, as we would expect. Now let's look at the interesting case: smoothing noisy data.

```
source(here::here("Tests", "1p_Smoothing_WN_demo.R"))
```




```
## Regular; no noise.
## The estimated growth rate is: 0.1557859
## Smoothing = 0.001; 4% noise
## The estimated growth rate is: 0.1557816
## Smoothing = 0.01; 4% noise
## The estimated growth rate is: 0.1557432
## Smoothing = 0.1; 4% noise
## The estimated growth rate is: 0.1553596
## Smoothing = 1; 4% noise
## The estimated growth rate is: 0.1516253
## Smoothing = 10 ; 4% noise
## The estimated growth rate is: 0.1222424
## Smoothing = 20 ; 4% noise
## The estimated growth rate is: 0.1005848
## Smoothing = 100 ; 4% noise
## The estimated growth rate is: 0.04160936
```

Now that's great! When we get to sufficiently high values of the smoothing parameter, we get closer to the correct estimate! However, if we overshoot the smoothing parameter, our estimate gets worse again. This is what we should expect: since we're only looking at one parameter, if we overestimate that parameter, as the model tends to do with r, we should just be able to apply the correct amount of smoothing to lower the value of r. Now our main concern is figuring out what the correct value of λ is.

Smoothing Parameter Optimization by Cross Validation

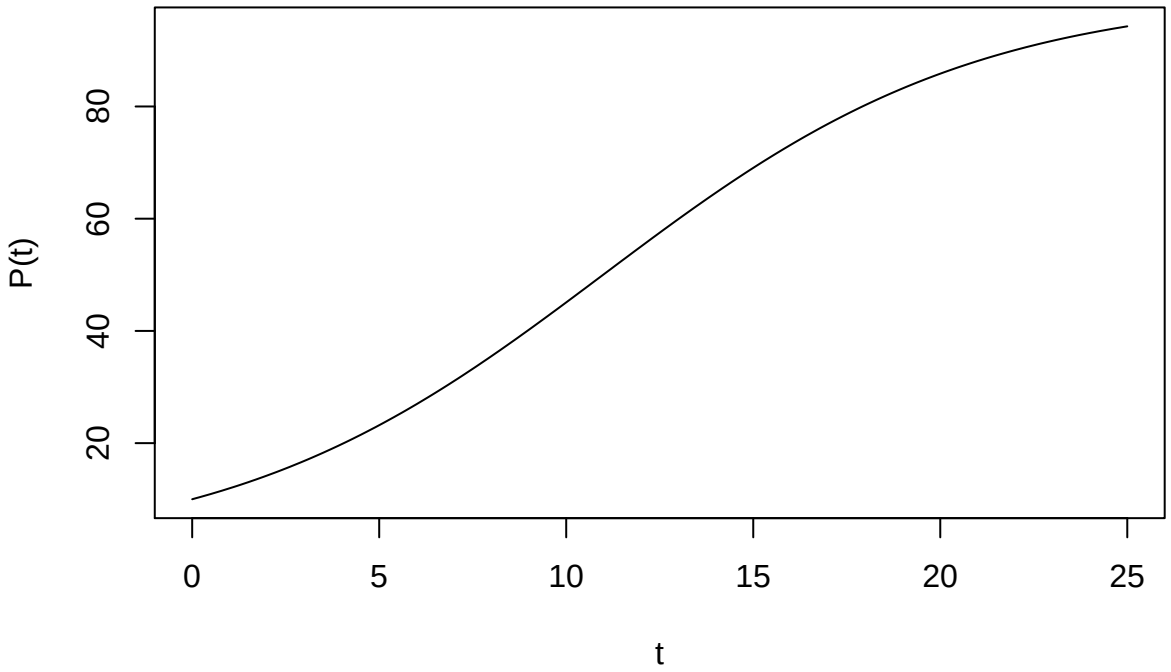
In order to figure out the correct value for λ, we choose to apply k-fold cross validation to the model in the method of Stein et al. The method works as follows.

1. Generate a list of "candidate" values for lambda.
2. For a given dataset, fit the model with each value of lambda. 3-fold cross validation was used to fit each model–the data is randomly partitioned into three datasets, and the model is fit three times. Each dataset is used as the "testing" data once, and in each fitting, the other two datasets are combined to use as the "training" data. For each of the three fitted models, compute the SSE. The average of the SSEs is called the CVE (cross validation error).

3. For each value of lambda, repeat the cross validation procedure ten times, and take the average of all the CVEs. This helps to remove the effect of random fluctuations caused by the random partitioning.
4. Choose the optimal value of lambda as the lambda that minimizes the mean CVE.
5. Fit a final model using the optimal value of lambda. This model should be checked for diagnostics and goodness-of-fit like a regular regression model.

The methods for cross validation were implemented manually in many of the journals, but an automated process is available via `HMBGR::cross_validate()` and the `replicate()` function can be used as a wrapper to call the cross validation function any number of times. If the `replicate` argument `simplify = TRUE` is specified, the function will output a vector of CVEs which can be averaged. The following simple example shows this process.

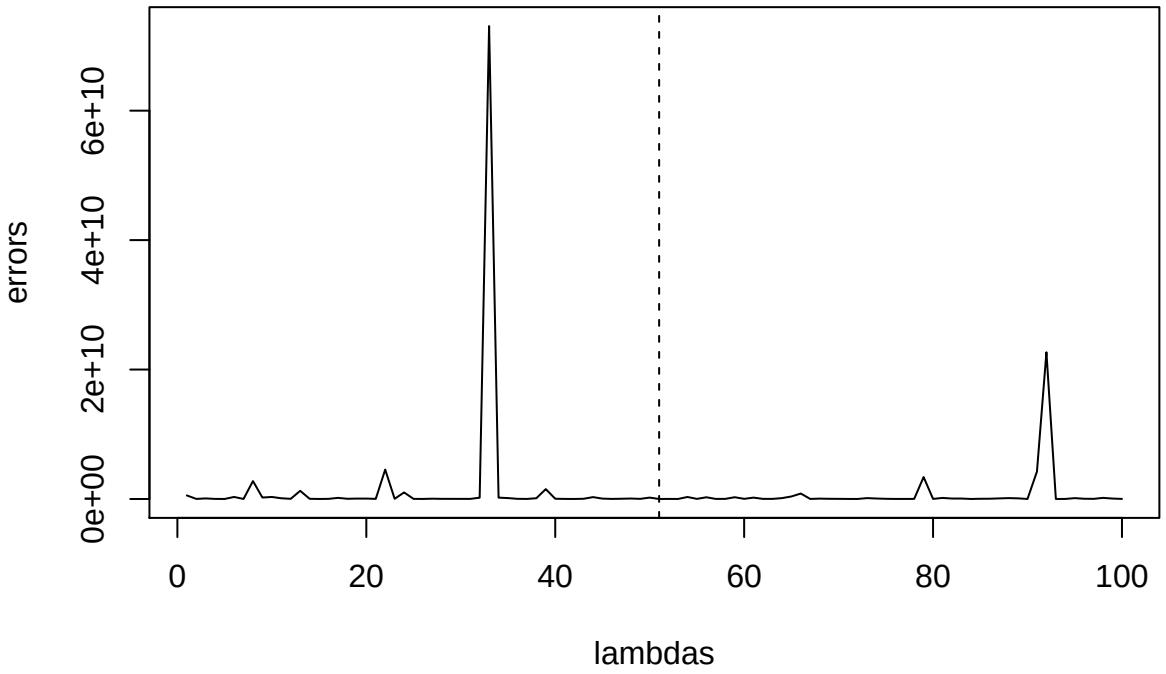
```
cve_ex <- HMBGR::generate_logistic_data(P0 = 10,
                                        K = 100,
                                        r = 0.2,
                                        max_t = 25,
                                        time_step = 0.1,
                                        make_plot = TRUE)
```




```
lambdas <- seq(1, 100, 1)
errors <- numeric(length(lambdas))
for (i in 1:length(lambdas)) {
  cves <- replicate(10,
                    HMBGR::cross_validate(cve_ex,
                                          k = 3,
                                          reduced = FALSE,
                                          lambda = lambdas[[i]]),
                    simplify = TRUE)
  errors[i] <- mean(cves)
}
```

The optimal value of lambda can then be visualized.

```
plot(lambdas, errors, typ = "l")
abline(v = lambdas[which.min(errors)], lty = 2)
```




```
cat("The optimal lambda is", lambdas[which.min(errors)], "with an error of", errors[which.min(errors)])
```

```
## The optimal lambda is 51 with an error of 1962299
```

While this is fairly easy to do manually, one improvement I would like to make to the `HMBGR` package is to implement an `optimal_lambda` function which takes a dataset, a list of lambdas, and a list of optional arguments for cross validation, and returns the optimal lambda from the list and optionally a plot like the one shown. Some fine-tuning would need to be done so that obvious outliers are not shown on the plot, but honestly just automating the replicate and lambda process would be a good step.

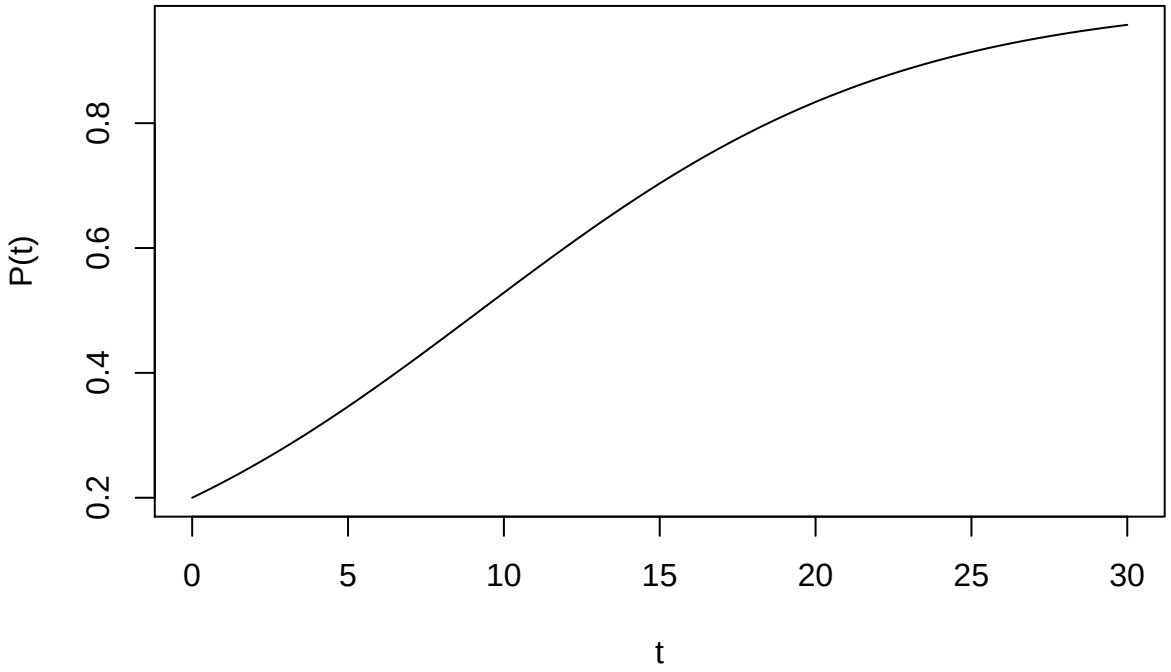
Optimal Smoothing Performance

The estimation of the model works better in some cases than in others. Over the last few weeks of my research (April 2020), I worked on developing a protocol for starting with data and estimating parameters.

In the two parameter case, we run into some problems as before. As seen in the example above, the errors in the two-parameter case can be extremely large. When a finer scale for lambda is used, this problem is much more noticeable (but also more computationally expensive, so while it is included in my Journal from April 21, I have not replicated it here). Part of the issue is that smoothing typically makes the value of $\hat{K}$ worse, so the SSEs tend to blow up.

Also in the two-parameter case, it remains impossible to recover the value of the carrying capacity in most cases, since the model underestimates the carrying capacity in the first place. Since we already expected just throwing smoothing at the two-parameter case to fail regardless of the value of lambda used, a more interesting question is about the performance of the smoothing model with the optimal lambda. So, for an example, we can generate some data (the `HMBGR` package assumes $K = 1$ unless otherwise specified).

```
test <- HMBGR::generate_logistic_data(P0 = 0.2,
                                      r = 0.15,
                                      max_t = 30,
                                      time_step = 0.1,
                                      make_plot = TRUE)
```

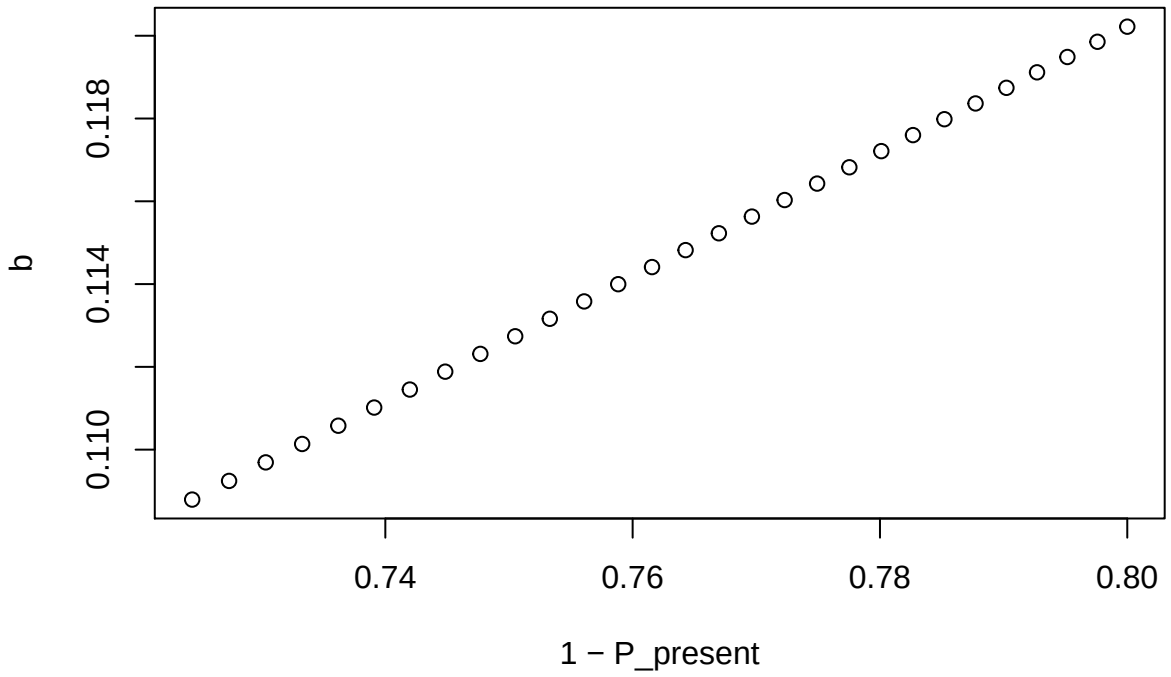


And apply cross-validation.

```
lambdas <- seq(0.1, 100, 0.1)
errors <- numeric(length(lambdas))
for (i in 1:length(lambdas)) {
  cves <- replicate(10,
                    HMBGR::cross_validate(test,
                                          k = 3,
                                          reduced = FALSE,
                                          lambda = lambdas[[i]]),
                    simplify = TRUE)
  errors[i] <- mean(cves)
}
```

Then, we can extract the optimal lambda and use it to fit the model.

```
best_lambda <- lambdas[which.min(errors)]
CVE <- errors[which.min(errors)]
mod <- HMBGR::model_logistic_data(df = test,
                                  smoothing = best_lambda,
                                  make_plot = TRUE,
                                  print_res = TRUE)
```




```
## The estimated growth rate is: 0.02336083
```

So, given that the true value was $r = 0.2$, our estimate was, $\hat{r} \approx 0.0212$, which is pretty close–and we could get an even better estimate by allowing more candidate values for λ–again, the more candidate values are allowed (using a smaller step size), the more computational power is required but the better the result should be.

Data Normalization and Smoothing Performance

This raises an interesting question. Since the single parameter from the dimensionless model can be recovered accurately, can the parameters be estimated in stages? If the carrying capacity can be estimated from the raw data, the data can be normalized to have a carrying capacity of 1, which allows the dimensionless model to be fit in order to estimate the growth rate.

I only briefly had time to examine the separate estimation of K, and some details are described later on. However, before concluding I want to record one example of all the steps in the method just so that my process is recorded.

An Example of the Entire Procedure so Far

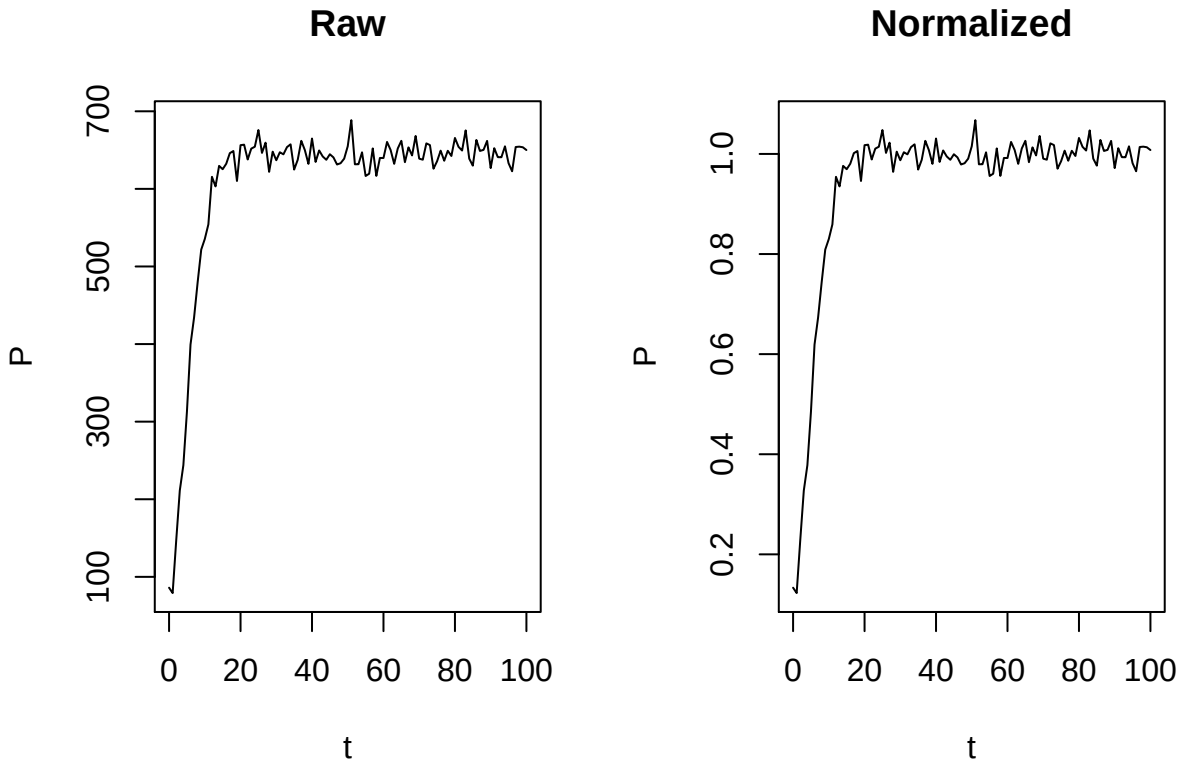
Let's start with some random data. I will randomly generate the parameters and assume a time step of 1 unit.

```
set.seed(300)
init <- sample(1:100, 1)
ex <- HMBGR::generate_logistic_data(P0 = init,
                                    K = round(init * runif(1, 1, 10),0),
                                    r = runif(1, 0, 0.5),
                                    max_t = 100,
                                    time_step = 1,
                                    noise = 0.02)
```

So now I don't know the parameters that generated this time series. So, we will start by estimating K from the data. For this example, I will use the mean of time points 90 to 100. Then, I will normalize the data by

this value and assume the normalized carrying capacity is 1.

```
khat <- mean(ex$P[90:100])
ex2 <- data.frame(t = ex$t, P = ex$P / khat)

par(mfrow = c(1,2))
plot(ex$t, ex$P, type = "l", main = "Raw", xlab = "t", ylab = "P")
plot(ex2$t, ex2$P, type = "l", main = "Normalized", xlab = "t", ylab = "P")
```

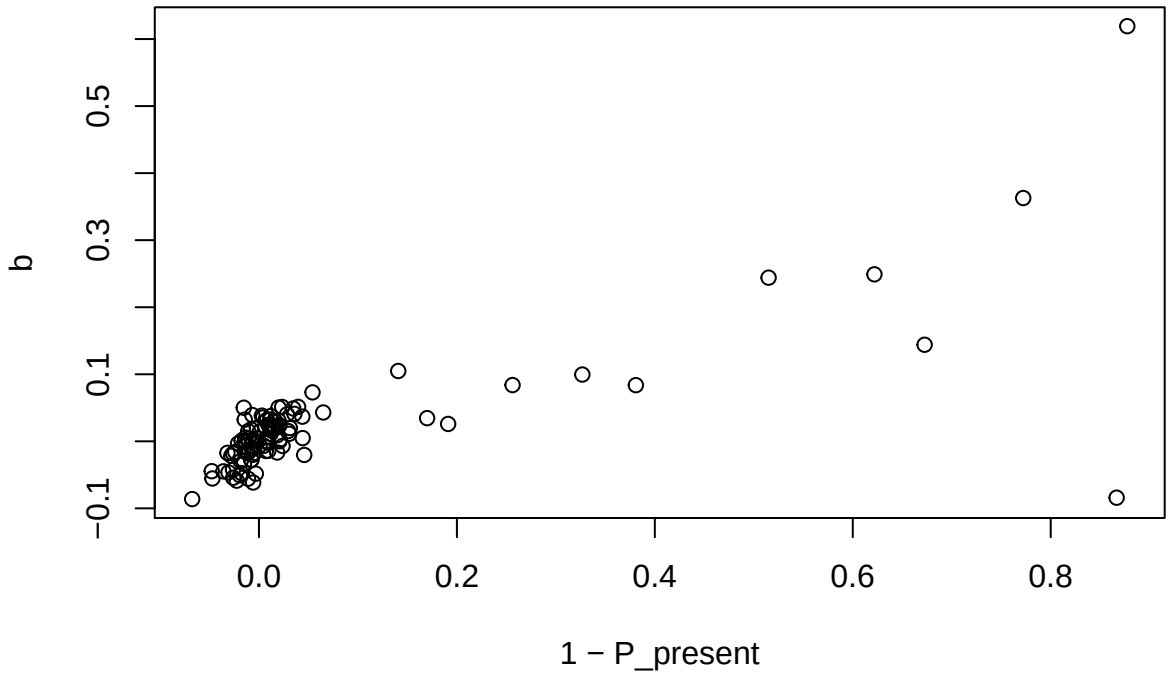


Now, given this estimate of K, we optimize λ.

```
lambdas <- seq(0.01, 1000, 0.01)
errors <- numeric(length(lambdas))
for (i in 1:length(lambdas)) {
  cves <- replicate(10,
                    HMBGR::cross_validate(ex2,
                                          k = 3,
                                          reduced = FALSE,
                                          lambda = lambdas[[i]]),
                    simplify = TRUE)
  errors[i] <- mean(cves)
}
best_lambda <- lambdas[which.min(errors)]
CVE <- errors[which.min(errors)]
```

Now, given the optimal value of lambda we fit the model. We don't technically need to specify the time step here, but I think it is good practice.

```
ex_mod <- HMBGR::model_logistic_data(ex2, smoothing = best_lambda, make_plot = TRUE, print_res = TRUE,
```

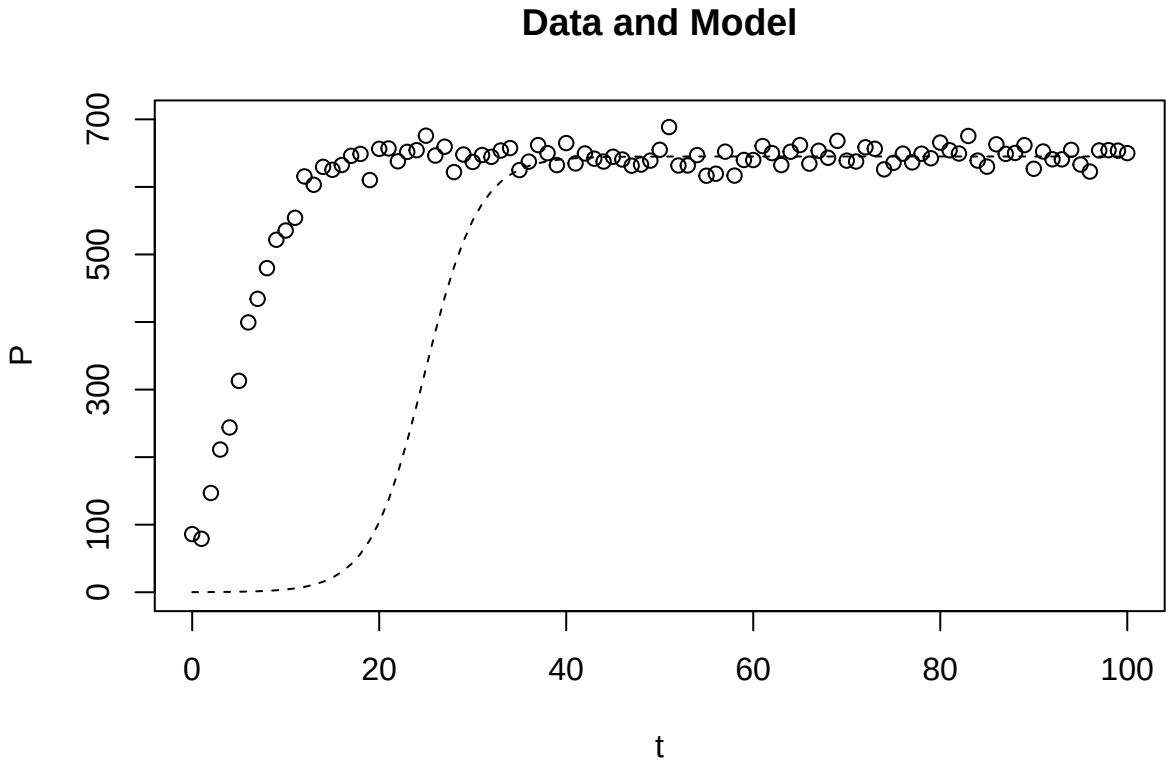



```
## The estimated growth rate is: 0.3415017
```

```
rhat <- ex_mod[[1]]
```

So now we can see how well our model fits the data visually. We could use a metric such as RMSE or AIC to quantify the goodness-of-fit as well. It would be ideal for `HMBGR` to return a linear model object, but at the moment I am not sure that this is possible. Any statistical tests may need to be computed by hand.

```
# Calculate the model.
pred <- HMBGR::generate_logistic_data(P0 = ex2$P[[1]], K = khat, r = rhat,
                                      max_t = max(ex2$t), time_step = 1)
plot(ex$t, ex$P, xlab = "t", ylab = "P", main = "Data and Model",
     xlim = c(0,100), ylim = c(0, 700)) +
lines(pred$t, y = pred$P, lty = 2)
```




```
## integer(0)
```

While this model doesn't appear to be perfect, it hits several data points and actually brings up an important point: the model fitting works much better if an equal number of data points are distributed at the top and bottom of the logistic curve. My guess is that there just isn't enough data at the bottom of the curve to get a better fit.

Conclusions and Further Work

The main conclusion of my work is that the one-species problem is prone to the same error as the two-species problem (and higher-dimensional problems as well). So, rather than the error being from computational issues or issues with the least squares method, we hypothesize that the main source of error is the inverse problem itself. We saw that even when our A matrix in the single parameter case was 1×1, and thus has condition number $\kappa_A = 1$ (i.e. A is well-conditioned), we had issues with error.

Smoothing didn't really help too much in the two parameter case: it's kind of like a double-edged sword. Applying smoothing makes the estimate of r better, but it makes the estimation of K worse, since our model tends to underestimate K anyway and smoothing forces smaller parameter values. However, we saw that smoothing can help find the correct parameter estimates in the dimensionless case.

In order to apply Tikhonov regularization to the data, we implemented a cross validation method in order to estimate the optimal value of lambda to recover the parameters of the model. However, even when using the optimal value of lambda, the smoothing model still can only ever successfully recover the growth rate, as the carrying capacity is underestimated (and thus cannot be improved by smoothing, which controls overestimation). However, in the dimensionless case, the model parameters can be successfully recovered. More testing should be conducted with noise in order to make sure this is typically the case.

One approach which I found to be effective is to estimate the parameters in stages. The carrying capacity can be estimated from the data first. Some strategies for estimating the carrying capacity include just using the maximum of the data (which is likely an efficient estimator, since the theoretical definition of the carrying capacity is the maximum population size), or some metric on part of the data points. For example, I used

the mean of the last five values to estimate the carrying capacity, and some metric like this may be more appropriate for noisy data. The best method for estimating the carrying capacity should be investigated further: another idea would be to normalize the data by the mean; for data that perfectly follows the logistic distribution, it should be true that

$$K \approx 2\bar{x},$$

so this could be another possibility.

However, in each case we observed that all parameters were predicted to be positive real numbers, within a few orders of magnitude of the true population parameter. This suggests that the stability conditions in higher-dimensional problems and the relationships between species might be preserved, although it is difficult to extrapolate these conclusions definitively from the one-species problem. In each test problem we fitted parameters to, we still obtained parameters in the correct range to predict the system would tend towards the stable equilibrium.

One preliminary step before moving on from this project may be to implement QR or SVD methods rather than the Moore-Penrose Generalized Inverse in the least-squares step, which is currently implemented. However, we do not expect this change to make a significant impact. Future research will likely concern ways to further mitigate the error, and applying these methods to real one-dimensional data. A further step in dealing with real one-dimensional data may be choosing an appropriate normalization method so that the one-parameter least squares model can be applied.

References

Alliende, M.C., and J.L. Harper. 1989. "Demographic Studies of a Dioecious Tree. 1. Colonization, Sex, and Age Structure of a Population of *Salix Cinerea*." *Journal of Ecology* 77: 1029–47.

Gockenbach, Mark S. 2016. *Linear Inverse Problems and Tikhonov Regularization*. MAA.

Kloppers, P.H., and Johanna C. Greeff. 2013. "Lotka-Volterra Model Parameter Estimation Using Experiential Data." *Applied Mathematics and Computation* 224: 817–25.

Lipkin, Leonard, and David Smith. 2001. *Logistic Growth Model*. JOMA.

Pearl, Raymond, and Lowell J. Reed. 1920. "On the Rate of Growth of the Population of the United States Since 1790 and Its Mathematical Representation." *PNAS* 6 (6).

Rockwood, Larry L., and Jonathan W. Witt. 2015. *Introduction to Population Ecology*. Second. John Wiley; Sons Ltd.

Stein, Richard R., Vanni Bucci, Nora C. Toussaint, Charlie G. Buffie, Gunner Rätsch, Eric G. Pamer, Chris Sander, and João B. Xavier. 2013. "Ecological Modeling from Time-Series Inference: Insight into Dynamics and Stability of Intestinal Microbiota." *PLOS Computational Biology* 9 (12).