ST542

Chien-Lan Hsueh

4/9/23

Table of contents

Packages and Helper Functions . . . 2
Introduction . . . 3
    Background . . . 3
    Perturbed SSE Curve Fitting (PSCF) Method . . . 3
    Research Questions . . . 3
Methods and Data . . . 4
    Methods . . . 4
    Data . . . 4
Study Case: Attenuation Decay . . . 5
    Physical Model . . . 5
    Data Simulation and Visualization . . . 5
Best Fit Model using Nonlinear Regression Model . . . 10
Perturbed SSE Curve Fitting (PSCF) Method . . . 12
    Curve Flipping and Range Selection . . . 14
    Fitting Strategy I: Fit Without Specifying Initial Fitting Range . . . 18
    Fitting Strategy II: Fit With Specifying Initial Fitting Range . . . 21
    Fitting Strategy III: Grid Search . . . 24
Bootstrapping CI . . . 34
    Parametric Bootstrap CI . . . 34
    Non-Parametric Bootstrap CI . . . 34
    Bootstrap CI Results . . . 35
Delta Method Normality . . . 37
    Univariate Delta Method . . . 37
    Bivariate Delta Method . . . 37
    Delta Method Results . . . 37
Comparison . . . 38
Conclusion . . . 39
References . . . 41

Packages and Helper Functions

```r
# packages
if (!require("pacman")) utils::install.packages("pacman", dependencies = TRUE)

pacman::p_load(
  conflicted, here,
  scales, skimr, glue, gt, GGally, ggpubr,
  broom, nlshelper, invgamma,
  tidyverse, zeallot
)

# resolve conflicts by setting preference
conflict_prefer("select", "dplyr", quiet = T)
conflict_prefer("filter", "dplyr", quiet = T)

# infix operator: string concatenation (ex: "act" %&% "5")
'%&%' <- function(x, y) paste0(x, y)

# infix operator: not %in%
'%notin%' <- Negate('%in%')
```

Introduction

A new algorithm has been developed by Prof. Hayes in order to estimate variances of a model parameter based on perturbation of the best fitted parameter. This algorithm has been claimed to give a good assessment of the standard error of the model parameter. The obtained results have been published in several research articles.

This work is to reproduce the algorithm and compare it with the other common statistics methods including nonlinear regression models, bootstrap confidence intervals and the delta normality method.

Background

Prof. Hayes' is an active faculty in the department of nuclear engineering at NCSU. His research of retrospective dosimetry (O'Mara and Hayes 2018) involves developing a new method to measure radiation dosages and analyze them to infer the actual radiation exposures. Although the traditional technique, which is a direct radiation measurement on the subjects, can provide an accurate depth profile measurement, it is costly and time consuming. Using the new method he developed (Hayes and O'Mara 2019), he can use forensic luminescence data that is high correlated to the actual radiation dosage and an industry standard Monte Carlo n-particle method (MCNP) (Brown et al. 2004) to fit the dose deposition profile and estimate the physical model parameter. Furthermore, with the obtained best fitted model parameter as a reference, a series of "perturbed" model fittings is then used in a normal curve fitting to estimate the variance of the parameter. A detailed description of this algorithm can be found in Dr. O'Mara's Ph.D. dissertation (O'Mara 2020).

Perturbed SSE Curve Fitting (PSCF) Method

For convenience, we call this algorithm Perturbed SSE Curve Fitting (PSCF) method in this work.

1. The parameter is estimated by minimizing the sum square of errors (SSE) of the model prediction and measurement data.
2. By deviating from the best fitted value, the sum square of errors is expressed as a function of the parameter and has a convex "hyperbolic" curve.
3. After this curve is "flipped" upside down, a Gaussian curve fitting is performed to obtain the spread as the estimated variance of the model parameter.

Research Questions

The goal of this work is to answer the following questions:

1. how good the algorithm is in term of determining the variances of physical model parameters?
2. if there is a solid statistics ground to support and backup the validity of this method?
3. if yes in (2), can it be improved and further generalized to any physical models?
4. if no in (2), how well does it estimate as an approximation approach?

The main challenge of this project is to verify this algorithm and use statistics theory to explain when it works. If we can find the connection, then we are able to put this novel method on a solid statistics ground with high confidence for its reliability. Brake and West, former statistics graduate students at NCSU, have made some attempts (Brake and West 2021). Unfortunately, the work is not quite complete and many unanswered questions remain.

Methods and Data

Methods

In addition to the proposed algorithm Perturbed SSE Curve Fitting (PSCF) Method, we are going to use the following statistical methods to estimate the variance of an unknown parameter of interest. These include:

1. Non-linear regression model
2. Bootstrap confidence intervals
    i. Parametric bootstrap
    ii. Non-parametric bootstrap
3. Delta method normality

The description of these method including their analysis techniques and procedures as well as the reason why we choose them for this study are given in each corresponding section below. At the end of this report, we compare the obtained results for a side-by-side comparison to justify if these method are sufficient and powerful to estimate a model parameter compared to Perturbed SSE Curve Fitting (PSCF) Method.

Data

In this study, we will look into a physical process. A physical model is used to generate simulated measurement data with a predetermined randomness in the model parameter. The detailed procedure is described in the next section.

Study Case: Attenuation Decay

In this work, we will use a simulated data set (details given later) to study a physical model **attenuation decay**.

Physical Model

Physical Model: Radiation strength measured at distance x

$$y = e^{-\lambda x} \tag{1}$$

- Response variable: y = radiation strength
- Explanatory variable (numeric): x = distance
- Parameter of interest: λ = attenuation coefficient.
- Estimator: $\hat{\lambda} = -\frac{1}{x}\ln y$

Goal: Estimate the parameter λ and its standard error

We treat the parameter λ as a random variable and follows a normal distribution. By central limit theorem (CLT), the sample mean of a random sample with sample size n is also distributed normally:

$$\begin{aligned} \lambda_i &\overset{iid}{\sim} N(\mu, \sigma^2) \\ \Rightarrow \bar{\lambda} &= \frac{1}{n}\sum_{i=1}^{n} \lambda_i \sim N(\mu, \sigma^2/n) \end{aligned} \tag{2}$$

Data Simulation and Visualization

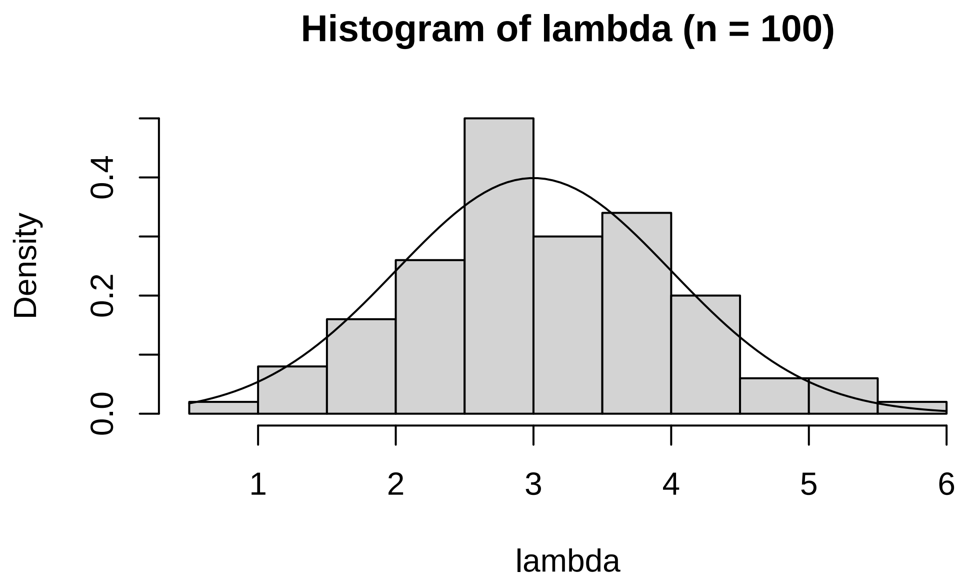
To make the analysis reproducible, we set the seed of the random generator:

```r
# seed for random generator
seed <- 2023

# sample size
n <- 100

# model function and its inverse
fmod <- function(x, lambda){ exp(-lambda*x) }
fmod_inv <- function(y, x){ -log(y)/x }

# parameter - truth
lambda_mean <- 3
lambda_sd <- 1
```

We also assume $\lambda_i \sim N(3, 1)$. With sample size of 100, the standard error of the sample mean is 0.1.

Next, we generate a simulated measurement data. For each measurement, y represents the recorded radiation strength measured at distance x.

```r
# make this analysis reproducible
set.seed(seed)

# parameter - random samples
lambdas <- rnorm(n, lambda_mean, lambda_sd)

# histogram of lambda RS
hist(lambdas, breaks = 10, freq = F, main = "", xlab = "")
curve(dnorm(x, lambda_mean, lambda_sd), add = T)
title(main = "Histogram of lambda (n = " %&% n %&% ")", xlab = "lambda")
```



```r
# create simulated data
df <- tibble(
  lambda = lambdas,
  x = rep_len( (1:30)/10, n) %>% sort(),
  ) %>%
  mutate(y = fmod(x, lambda)) %>%
  relocate(x, y, lambda)
```

The data set contains 100 observations of the radiation strength and measurement distance:

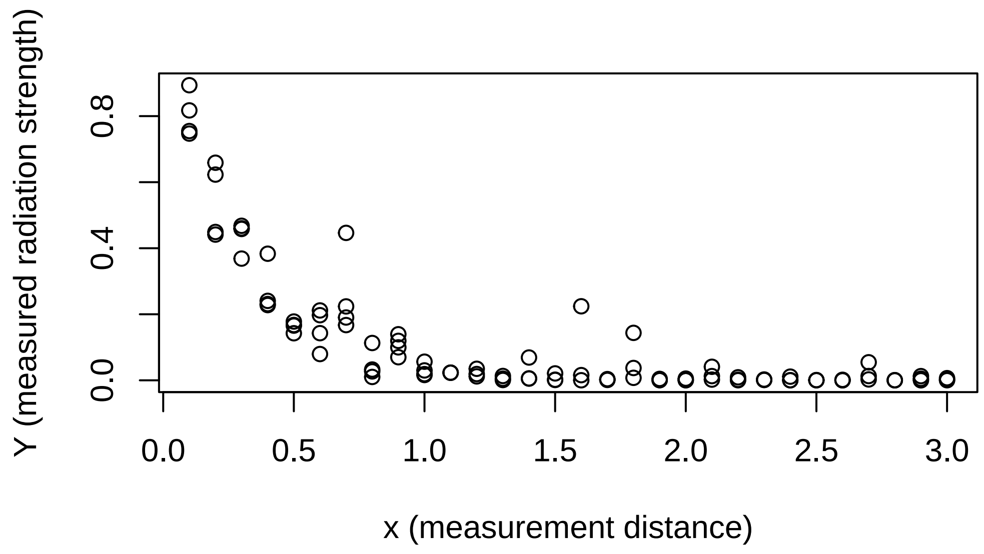
```r
# exam data set
df %>% select(-lambda)
```

x	y
0.1	0.7470512
0.1	0.8173355
0.1	0.8936034

x	y
0.1	0.7547373
0.2	0.6229414
0.2	0.4412430
0.2	0.6588532
0.2	0.4491816
0.3	0.4583052
0.3	0.4678708
0.3	0.3685833
0.3	0.4601623
0.4	0.2405531
0.4	0.2309987
0.4	0.3833369
0.4	0.2277855
0.5	0.1656425
0.5	0.1779868
0.5	0.1425059
0.5	0.1676112
0.6	0.2116105
0.6	0.1972263
0.6	0.0795679
0.6	0.1427776
0.7	0.1672286
0.7	0.4464011
0.7	0.1901728
0.7	0.2237461
0.8	0.0269995
0.8	0.0101712
0.8	0.1130304
0.8	0.0326692
0.9	0.1394404
0.9	0.0699783
0.9	0.1194893
0.9	0.0997110
1.0	0.0299807
1.0	0.0160954
1.0	0.0187673
1.0	0.0563938
1.1	0.0230251
1.1	0.0232698
1.1	0.0228452
1.2	0.0349134
1.2	0.0190094
1.2	0.0118761
1.3	0.0009257
1.3	0.0049978
1.3	0.0130888
1.4	0.0052224
1.4	0.0059157
1.4	0.0690611

x	y
1.5	0.0209452
1.5	0.0018826
1.5	0.0010310
1.6	0.0002140
1.6	0.2241884
1.6	0.0158115
1.7	0.0036548
1.7	0.0018655
1.7	0.0011364
1.8	0.0075313
1.8	0.1437157
1.8	0.0373544
1.9	0.0004601
1.9	0.0043073
1.9	0.0006720
2.0	0.0052119
2.0	0.0003050
2.0	0.0003894
2.1	0.0409694
2.1	0.0127469
2.1	0.0021947
2.2	0.0091530
2.2	0.0003081
2.2	0.0016247
2.3	0.0013453
2.3	0.0024619
2.3	0.0009695
2.4	0.0111439
2.4	0.0004421
2.4	0.0000114
2.5	0.0007458
2.5	0.0001640
2.5	0.0003884
2.6	0.0001840
2.6	0.0000309
2.6	0.0015747
2.7	0.0026832
2.7	0.0546166
2.7	0.0129155
2.8	0.0000677
2.8	0.0005119
2.8	0.0000004
2.9	0.0048785
2.9	0.0000046
2.9	0.0127241
3.0	0.0002473
3.0	0.0014657
3.0	0.0065553

```
# scatter plot between measured strength y and distance x
plot(df$x, df$y, xlab = "x (measurement distance)", ylab = "Y (measured radiation strength)")
```



It's clearly to see the decay trend described in the physical model (the decay equation) and the randomness due to the uncertainty of the model parameter.

```
# Save session variables and function definition
# summary table of truth (for later comparison)
tbl_truth <- tibble(
  method = "Truth (All sample sizes = " %&% n %&% ")",
  lambda = lambda_mean,
  std.error = lambda_sd/sqrt(n),
  lower_0.95 = lambda_mean + qt(0.05/2, n-1)*std.error,
  upper_0.95 = lambda_mean + qt(1 - 0.05/2, n-1)*std.error,
  width = upper_0.95 - lower_0.95)

# save session image
save.image(here("data", "sim_decay.RData"))
```

Best Fit Model using Nonlinear Regression Model

To estimate the model parameter, the best fitted value of the model parameter is determined by minimizing the the residual sum-of-squares (sum squares of errors, SSE). The least squares estimation for the parameter is therefore determined by:

$$\hat{\theta} = \arg\min_{\theta} \sum_{i=1}^{n} \left[y_i - f(x_i, \hat{\theta})\right]^2 \tag{3}$$

where $f : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}$ is an unknown function with the independent variables x_i for $i = 1, \ldots, n$ and the unknown parameter $\hat{\theta}$. We are assuming the randomness of the measurement data y_i are independent and identically distributed (i.i.d.) and can be estimated by:

$$\begin{gathered} y_i = f(x_i, \theta) + \epsilon_i \quad i = 1, 2, \ldots, n \\ \\ E[\epsilon_i] = 0 \\ Cov(\epsilon_i, \epsilon_j) = \delta_{ij}\sigma^2 \quad \forall i, j \end{gathered} \tag{4}$$

There are many possible way to perform the east squares estimation and obtain the parameter estimate numerically. In Prof. Hayes's research, an industrial standard method, Monte Carlo N-particle (MCNP), is used.

For this work, since we use nonlinear regression instead to achieve the same minimization of SSE. There are two major reasons we choose this method. First, using nonlinear regression model doesn't rely on the some strong assumptions including normality and linearity. With sufficiently large sample size, we can use nonlinear function f and its asymptotic normality of the least squares estimate(Seber and Wild 1989) to estimate the parameter:

$$\begin{aligned} \hat{\theta} \dot{\sim} N\left(\theta, Var\left(\left[\mathbf{F}(\theta)' \cdot \mathbf{F}(\theta)\right]^{-1}\right)\right) \\ Var(y_i - f(x_i, \theta)) &\approx Var(y_i) + Var(f(x_i, \theta)) \\ &\approx \sigma^2 + \sigma^2 \nabla f(x_i, \theta)' \left[\mathbf{F}(\theta)' \cdot \mathbf{F}(\theta)\right]^{-1} \nabla f(x_i, \theta) \end{aligned} \tag{5}$$

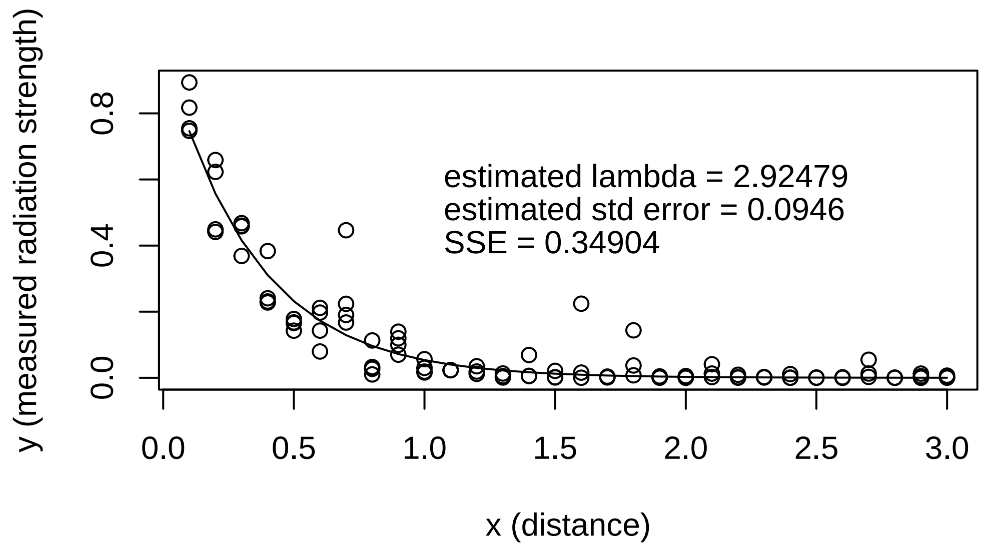
The second reason is the computational algorithm of this method is well developed and the base R provides `nsl()`:

```r
# fit nlr model
mod_fit <- nls(y ~ fmod(x, lambda), df, start = c(lambda = lambda_mean))
mod_fit
```

```
Nonlinear regression model
  model: y ~ fmod(x, lambda)
   data: df
lambda
 2.925
 residual sum-of-squares: 0.349
```

```
Number of iterations to convergence: 2
Achieved convergence tolerance: 7.308e-06
```

```
# save fit result
bestfit <- list(
  lambda = tidy(mod_fit)$estimate,
  se = tidy(mod_fit)$std.error,
  sd = tidy(mod_fit)$std.error*sqrt(n),
  sse = glance(mod_fit)$deviance)

# overlay plot with fitted values
plot(df$x, df$y, xlab = "x (distance)", ylab = "y (measured radiation strength)")
lines(x = df$x,
      y = predict(mod_fit, tibble(x = df$x, lambda = bestfit$lambda)),
     type="l")
text(1, 0.6, "estimated lambda = " %&% round(bestfit$lambda, 5), pos = 4)
text(1, 0.5, "estimated std error = " %&% round(bestfit$se, 5), pos = 4)
text(1, 0.4, "SSE = " %&% round(bestfit$sse, 5), pos = 4)
```



```
# save as a comparison table
tbl_nlr <- tibble(
  method = "Nonlinear Regression",
  lambda = bestfit$lambda,
  std.error = bestfit$se,
  lower_0.95 = lambda_mean + qt(0.05/2, n-1)*std.error,
  upper_0.95 = lambda_mean + qt(1 - 0.05/2, n-1)*std.error,
  width = upper_0.95 - lower_0.95)
```

Note that the residual sum-of-squares is 0.3490421.

Perturbed SSE Curve Fitting (PSCF) Method

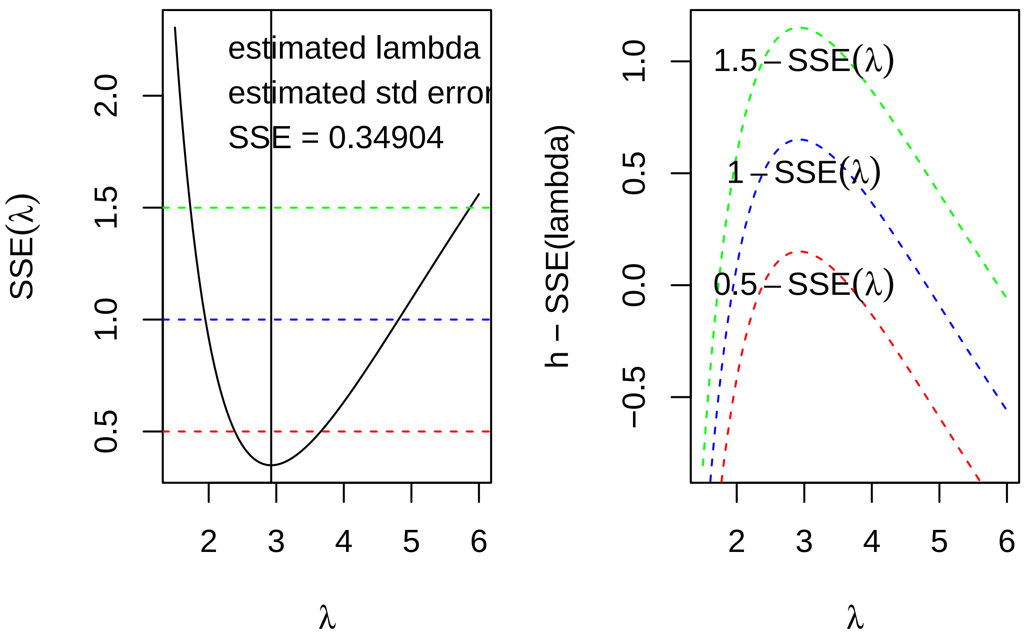
This proposed method fits a Gaussian curve on the sum square of errors (SSE) as a function of the model parameter λ. Since the estimate is obtained by minimizing SSE, the SSR curve is a convex curve with the minimum at $\hat{\lambda}$. To further investigate this visually, we first define functions to compute the SSE as a function of the mode parameter.

```
# compute SSE (a vectorized function)
SSE <- function(lambda, df, fmod){
  Vectorize(\(lambda){sum((df$y-fmod(df$x, lambda))^2)})(lambda)
}

# compute range ("reflection points")
get_range <- function(h, df, fmod, bestfit){
  # upper and lower bound
  lb <- NA
  try(
    lb <- uniroot(
    \(x){h - SSE(x, df, fmod)},
    c(bestfit$lambda, bestfit$lambda - 3*bestfit$sd),
    extendInt = "yes")$root
  )
  if(is.na(lb)) lb <- bestfit$lambda - 3*bestfit$sd

  ub <- NA
  try(
    ub <- uniroot(
      \(x){h - SSE(x, df, fmod)},
      c(bestfit$lambda, bestfit$lambda + 3*bestfit$sd),
      extendInt = "yes")$root
  )
  if(is.na(lb)) ub <- bestfit$lambda + 3*bestfit$sd

  # standardized upper and lower bound
  c(lb_std, ub_std) %<-% (c(lb, ub) - bestfit$lambda)/bestfit$sd

  list(
    ub = ub, lb = lb, range = ub - lb,
    ub_std = ub_std, lb_std = lb_std, range_std = ub_std - lb_std)
}
```

To flip the convex curve: $h - SSE(\lambda)$ with some constant h:

```
# define plotting parameters
h_values <- c(0.5, 1, 1.5)
from_to <- lambda_mean + c(-1.5, 3)*lambda_sd
color <- c("red", "blue", "green")

# save and restore option
op <- par(pty="m", mfrow=c(1, 2), mar=c(4.2, 4.2, 1, 1))
```

```
# plot SSE vs. lambda
curve(SSE(x, df, fmod),
      from = from_to[1], to = from_to[2],
      xlab = bquote(lambda), ylab = bquote(SSE(lambda)))
abline(v = tidy(mod_fit)$estimate)
text(2, 2.2, "estimated lambda = " %&% round(bestfit$lambda, 5), pos = 4)
text(2, 2, "estimated std error = " %&% round(bestfit$se, 5), pos = 4)
text(2, 1.8, "SSE = " %&% round(bestfit$sse, 5), pos = 4)
abline(h = h_values, lty = 2, col = color)

# plot inverse SSE vs. lambda
walk(3:1, \(h){
  curve(h_values[h] - SSE(x, df, fmod),
        add = (h!=3), col = color[h], lty = 2,
        from = from_to[1], to = from_to[2],
        xlab = bquote(lambda), ylab = "h - SSE(lambda)")
  text(3, h_values[h] - 0.5, bquote(.(h_values[h]) - SSE(lambda)))
})
```

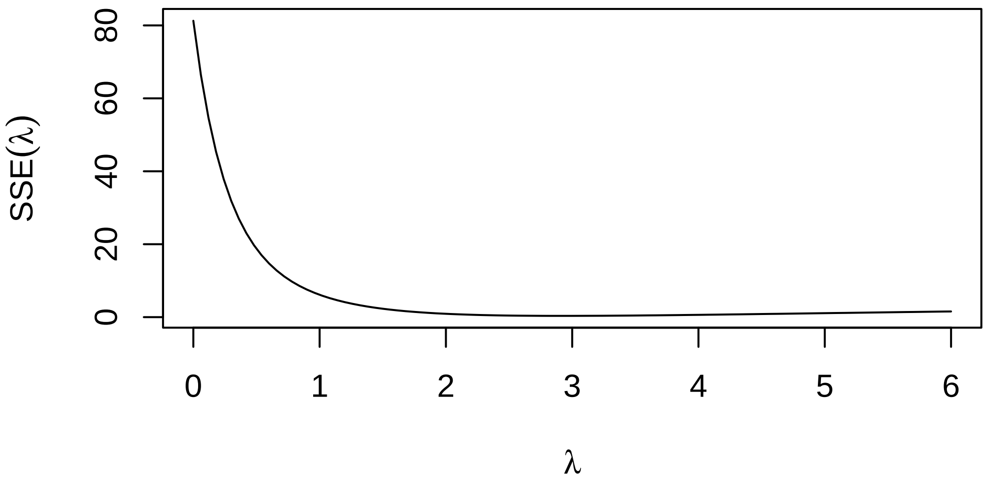


```
# restore option
par(op)
```

As seen above, the choice of h can be arbitrary. In the following attempts, we will use these three values to flip the curve and perform curve fitting as proposed.

```
# define plotting parameters
from_to <- lambda_mean + c(-3, 3)*lambda_sd
```

```r
# plot SSE vs. lambda
curve(SSE(x, df, fmod),
      from = from_to[1], to = from_to[2],
      xlab = bquote(lambda), ylab = bquote(SSE(lambda)))
```



This means we need to know:

1. How to flip the SSE curve is an arbitrary choice, ie. what value of h should be used?
2. What is the range of λ should be included in the fitting?
3. The inverse SSE curve is skewed and is not bell shaped. Is Gaussian curve a good candidate for the fitting?

Curve Flipping and Range Selection

We first address the first two by parameterizing the objective function for $SSE(\lambda)$ minimization as $h - SSE(\lambda)$ so that we can fit a Gaussian curve:

$$\widehat{SSE}(\lambda = x) \simeq h - a \cdot e^{\frac{-1}{2\sigma^2}(x-\mu)^2} \tag{6}$$

where $h > 0$ and $a > 0$.

By doing so, we include two parameters h to control how to flip the curve and a to control the scaling width. However we need to address that the width of a Gaussian curve is then determined by this scaling factor a and the standard deviation σ. We will need to expect how the fitting works and decide our fitting strategy.

In Prof. Hayes' algorithm, if the initial fitting range is not specified, the reflection points calculated by solving $h - SSE(\lambda) = 0$ will be used.

Next, we define a helper function to compute $\widehat{SSE}(\lambda)$ based on a given fitting function `fn()`. In the case of Gaussian fitting, the R base function `dnorm()` is used.

```r
# define a function to compute SSE fitted by the specified curve
SSE_hat <- function(lambdas, h, a, fn, ...){
  sse_fit  <-  abs(h) - abs(a)*fn(lambdas, ...)
  return(sse_fit)
}
```

Finally, we define couple more helper functions to carry on the curve fitting to optimize a specified objective function in order to minimize the residual sum squares:

$$\arg\min_{h,a,\mu,\sigma} \sum_{\lambda_i} \left(\widehat{SSE}(\lambda_i) - SSE(\lambda_i)\right)^2 \tag{7}$$

These helper functions are:

- `fit_optim()`: perform optimization to minimize the objective function using a Gaussian curve.
- `fit_summarize()`: summarize the results.
- `fit_plot()`: plot $\widehat{SSE}(\lambda)$ and $SSE(\lambda)$ for comparison and visually verify the goodness of the fitting (optimization).

```r
# minimize sse(lambda) and plot sse for comparison using Gaussian fit
## pars = {h, a, mean, sd, range_lb, range, n}
fit_optim <- function(pars, pars_control, df, fmod, bestfit){

  update_range <- function(pars){
    # add range_lb and range if either is missing in pars
    # use "reflection points" as endpoints of fitting range
    #print(paste(pars[c("h", "range_lb")]))
    if(is.na(pars$range_lb) | is.na(pars$range)){
      range_by_h <- get_range(pars$h, df, fmod, bestfit)
      if(is.na(pars$range_lb)) pars$range_lb <- range_by_h$lb
      if(is.na(pars$range)) pars$range <- range_by_h$range_std
    }
    #print(paste(pars[c("h", "range_lb")]))
    pars
  }

  # update parameters
  update_pars <- function(b, pars){
    pars_new <- c(b, pars[setdiff(names(pars), names(b))])[names(pars)]
    #print(pars_new[c("h", "range_lb")] %>% paste())
    pars_new <- update_range(pars_new)
    #print(pars_new[c("h", "range_lb")] %>% paste())

    return(pars_new)
  }

  # compute sse_data and sse_fit for residuals
  sse <- function(pars2){
    # determine lower and upper bound of lambda
    lb <- bestfit$lambda - abs(pars2$range_lb)*bestfit$sd
    ub <- lb + abs(pars2$range)*bestfit$sd
    # set up grid for lambda
    lambda <- seq(lb, ub, length.out = pars2$n)
    # compute sse_data and sse_fit
    sse_data <- SSE(lambda, df, fmod)
    sse_fit <- SSE_hat(lambda, pars2$h, pars2$a, dnorm, pars2$mean, pars2$sd)
    # pack into a data frame and return
    return(tibble(lambda = lambda, sse_data = sse_data, sse_fit = sse_fit))
  }

  # objective function: residual sum squares
  fn_obj <- function(b){
    # update parameters
    pars2 <- update_pars(b, pars)

    # compute residuals
    df2 <- sse(pars2)
    residuals <- df2$sse_fit - df2$sse_data
    # return residual sum squares
    return(sum(residuals^2))
  }

  pars <- update_range(pars)

  # optimize with minimization of residual sum squares
  fit <- optim(par = pars[pars_control], fn = fn_obj)

  # retrieve parameters
  pars2 <- update_pars(fit$par, pars) %>% lapply(abs)

  # compute sse_data and sse_fit
  df2 <- sse(pars2)
  rss <- sum((df2$sse_fit - df2$sse_data)^2)
  width <- max(df2$lambda) - min(df2$lambda)
  pars2 <- c(rss = rss, width = width, pars2) %>%
    lapply(signif, digits = 4)
  # alert if minimized values don't match
  if(fit$value != rss){
    print(glue("Not matched! {fit$value} vs. {rss}"))
  }

  return(list(fit = fit, pars = pars2, df_sse = df2))
}

# summarize fit results
fit_summarize <- function(fits, title = "", plot = F, print = F){
  fit <- fits$fit
  pars <- fits$pars
  df_sse <- fits$df_sse

  tbl_par <- as_tibble(pars) %>%
    mutate(title = title) %>%
    relocate(title, rss, width, h, a, mean, sd)
  if(print) print(tbl_par)

  # plot for comparison
  p <- fit_plot(df_sse, pars, title)
  if(plot) print(p)

  # return summary table
  tbl <- tibble(
    method = title,
    lambda = pars$mean,
    std.error =  pars$sd/sqrt(nrow(df)),
    lower_0.95 = lambda_mean + qnorm(0.05/2)*std.error,
    upper_0.95 = lambda_mean + qnorm(1 - 0.05/2)*std.error,
    width = upper_0.95 - lower_0.95)

  return(list(pars = tbl_par, tbl = tbl, p = p))
}

# plot for fit comparison
fit_plot <- function(df_sse, pars, title){
  # fewer digits for easier reading
  pars <- lapply(pars, round, digits = 3)

  # plot comparison;'
  df_sse %>%
    pivot_longer(cols = starts_with("sse"), names_to = "Type") %>%
    ggplot(aes(lambda, value, col = Type)) +
    geom_line() +
    labs(
      x = bquote(lambda),
      y = bquote(SSE(lambda)),
      title = glue(
        "{title}\n",
        "rss = {pars$rss}, width = {pars$width}"),
      subtitle = glue(
        "(mean, sd) = ({pars$mean}, {pars$sd})\n",
        "(h, a) = ({pars$h}, {pars$a})"))
}
```

In `fit_optim()`, we include the following fitting parameters and their initial values in a vector `pars`:

- `h` and `a`: Control how to flip the SSE curve and its scaling width. Both the initial values are set to 1.
- `mean`, `sd`: Control location and shape of the Gaussian curve. The mean and standard deviation

obtained in the best fitting (non-linear regression) are used as initial values.
- `range_lb`: Controls range of λ, lower bound of the fitting range (unit: `sd`). The initial lower bound is set to the 2 standard deviation lower than the initial mean.
- `range`: Controls range of λ, numbers of `sd` to be included in the fitting (unit: `sd`). The initial range to be included is at least 4 standard deviations.
- `n`: Controls how many grid points in the fitting range λ.

If either of `range_lb` and `range` is given, we use the "reflection points" as endpoints of the fitting range. This is defined by solving $h - SSE(\lambda) = 0$.

Also, a parameter control vector `pars_control` is used to specify which parameters are varied to minimize the objective function. For example, `c(1, 2, 3, 4)` means we only vary `h`, `a`, `mean` and `sd` (the 1st, 2nd, 3rd and 4th parameters listed above) to optimize the fitting.

Fitting Strategy I: Fit Without Specifying Initial Fitting Range

In our first attempt, we fit $SSE(\lambda)$ without explicitly specifying initial fitting range. the "reflection points" will be used to bound the fitting range based on the value of h:

- Control (case 1): fit all parameters
- Group 1 (case 2-3): fixed `h` or `a` - how to flip the SSE curve
- Group 2 (case 4-5): fixed `mean` or `sd` - shape of the Gaussian curve
- Group 3 (case 6-8): fixed `range_lb` and/or `range` - fitting range

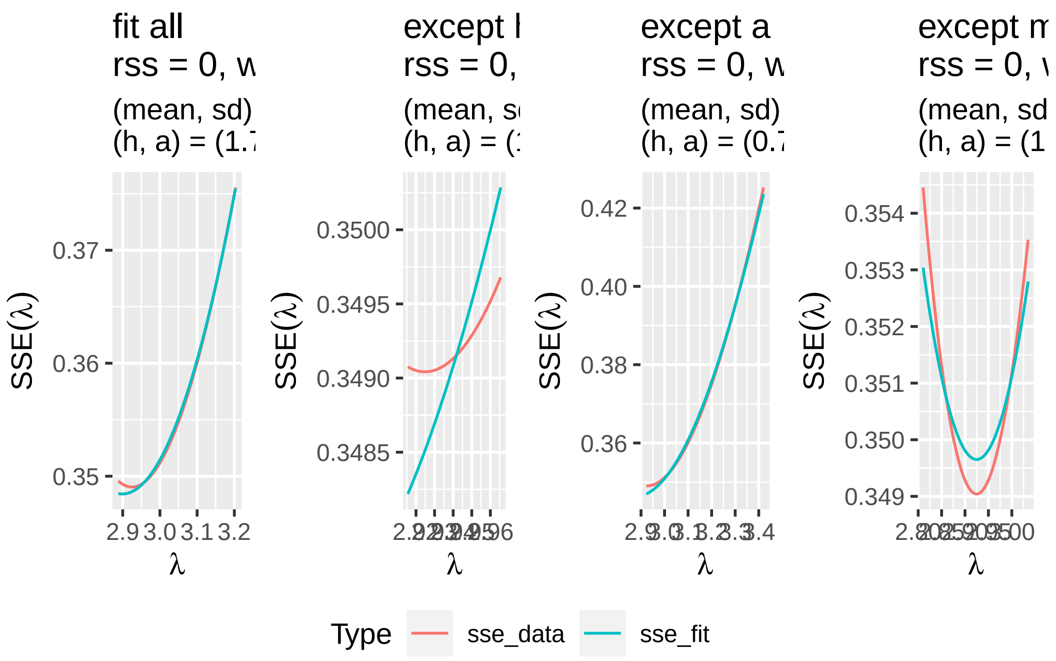
```r
# initial parameters and control which to fit
pars <- list(
  h = 1, a = 1, mean = bestfit$lambda, sd = bestfit$sd,
  range_lb = NA, range = NA, n = 100)

# optimization fit: fit all parameters
opt1 <- c(1, 2, 3, 4, 5, 6) %>%
  fit_optim(pars, pars_control = ., df, fmod, bestfit) %>%
  fit_summarize(title = "fit all")
# optimization fit: fit all parameters except h
opt2 <- c(2, 3, 4, 5, 6) %>%
  fit_optim(pars, pars_control = ., df, fmod, bestfit) %>%
  fit_summarize(title = "except h")
# optimization fit: fit all parameters except a
opt3 <- c(1, 3, 4, 5, 6) %>%
  fit_optim(pars, pars_control = ., df, fmod, bestfit) %>%
  fit_summarize(title = "except a")
# optimization fit: fit all parameters except mean
opt4 <- c(1, 2, 4, 5, 6) %>%
  fit_optim(pars, pars_control = ., df, fmod, bestfit) %>%
  fit_summarize(title = "except mean")
# optimization fit: fit all parameters except sd
opt5 <- c(1, 2, 3, 5, 6) %>%
  fit_optim(pars, pars_control = ., df, fmod, bestfit) %>%
  fit_summarize(title = "except sd")
# optimization fit: fit all parameters except range_lb
```

```r
opt6 <- c(1, 2, 3, 4, 6) %>%
  fit_optim(pars, pars_control = ., df, fmod, bestfit) %>%
  fit_summarize(title = "except range_lb")
```

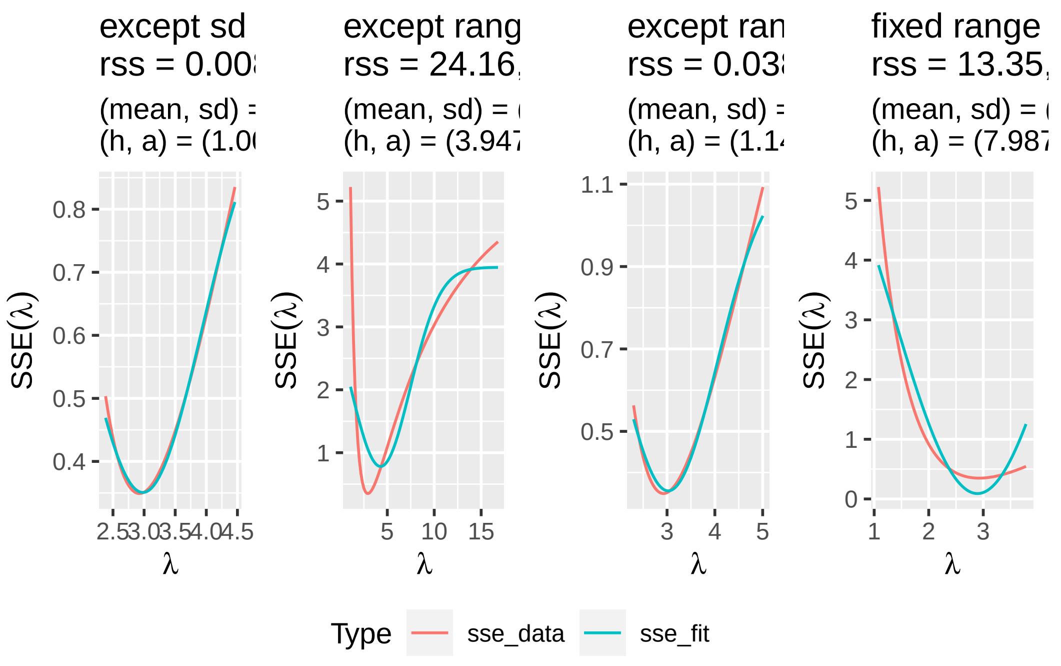
```
Warning in fn(lambdas, ...): NaNs produced
```

```r
# optimization fit: fit all parameters except range
opt7 <- c(1, 2, 3, 4, 5) %>%
  fit_optim(pars, pars_control = ., df, fmod, bestfit) %>%
  fit_summarize(title = "except range")
# optimization fit: fit all parameters with fixed interval
opt8 <- c(1, 2, 3, 4) %>%
  fit_optim(pars, pars_control = ., df, fmod, bestfit) %>%
  fit_summarize(title = "fixed range")

# compare visually
ggarrange(
  opt1$p, opt2$p, opt3$p, opt4$p, opt5$p, opt6$p, opt7$p, opt8$p,
  ncol = 4, common.legend = TRUE, legend = "bottom")
```

```
$`1`
```



```
$`2`
```



```
attr(,"class")
[1] "list"      "ggarrange"
```

```
# summarize in a table
bind_rows(
  opt1$pars, opt2$pars, opt3$pars, opt4$pars,
  opt5$pars, opt6$pars, opt7$pars, opt8$pars) %>% rowid_to_column()
```

rowid	title	rss	width	h	a	mean	sd	range_lb	range	n
1	fit all	1.210e-05	0.31500	1.7400	5.363	2.899	1.5380	0.038640	0.33290	100
2	except h	1.860e-05	0.04998	1.0000	1.916	2.853	1.1710	0.009710	0.05283	100
3	except a	6.410e-05	0.49780	0.7934	1.000	2.871	0.8921	0.002232	0.52620	100
4	except mean	2.840e-05	0.22440	1.0890	2.210	2.925	1.1930	0.120900	0.23720	100
5	except sd	7.528e-03	2.07800	1.0040	1.548	2.980	0.9460	0.574300	2.19600	100
6	except range_lb	2.416e+01	15.71000	3.9470	25.120	4.280	3.1650	1.951000	16.61000	100
7	except range	3.763e-02	2.70500	1.1460	2.032	3.024	1.0260	0.659900	2.85900	100
8	fixed range	1.335e+01	2.70500	7.9870	31.200	2.894	1.5760	1.951000	2.85900	100

```
bind_rows(
  opt1$tbl, opt2$tbl, opt3$tbl, opt4$tbl,
    opt5$tbl, opt6$tbl, opt7$tbl, opt8$tbl) %>% rowid_to_column()
```

rowid	method	lambda	std.error	lower_0.95	upper_0.95	width
1	fit all	2.899	0.15380	2.698558	3.301442	0.6028849
2	except h	2.853	0.11710	2.770488	3.229512	0.4590236
3	except a	2.871	0.08921	2.825152	3.174848	0.3496968
4	except mean	2.925	0.11930	2.766176	3.233824	0.4676474
5	except sd	2.980	0.09460	2.814587	3.185413	0.3708252
6	except range_lb	4.280	0.31650	2.379671	3.620329	1.2406572
7	except range	3.024	0.10260	2.798908	3.201092	0.4021846
8	fixed range	2.894	0.15760	2.691110	3.308890	0.6177806

Control (case 1): fit all parameters

Not surprisingly, the case 1 has the smallest fitting error (`rss`) because of all the parameters are free to change. But this is not a good fitting because if only fit one side of the curve to get the smallest `rss`.

Group 1 (case 2-3): fixed `h` or `a` - how to flip the SSE curve

Similarly to what happened previously, if we fit everything but `h` or `a`, the minimization of `rss` leads to fitting on one side of the curve.

Group 2 (case 4-5): fixed `mean` or `sd` - shape of the Gaussian curve

By fix either `mean` or `sd`, we fix the location and shape of the Gaussian curve. Fixing `mean` leads to a very narrow fitting range (plot 4). Fixing `sd` gives a off-center fitting range (plot5).

Group 3 (case 6-8): fixed `range_lb` and/or `range` - fitting range

The lower bound of the reflection points is very far away from the mean. Therefore, in case 6 and 8, the lower bound of the fitting range (`range_lb`) is fixed and this leads to the fitting curve with a large fitting range and large `rss`. On the other hand if we only fix the width of the fitting raneg (`range`), we get a pretty good fit (plot 7) although it's off-center similarly to case 5.

Based on the results above, in order to get a good fitting, we need to force the fitting range centered and force it to include a wider range. These can be done by fixing `range_lb` and `range` parameters. If we let them be free, the fitting results will be either off-centered or using a very narrow fitting range.

Fitting Strategy II: Fit With Specifying Initial Fitting Range

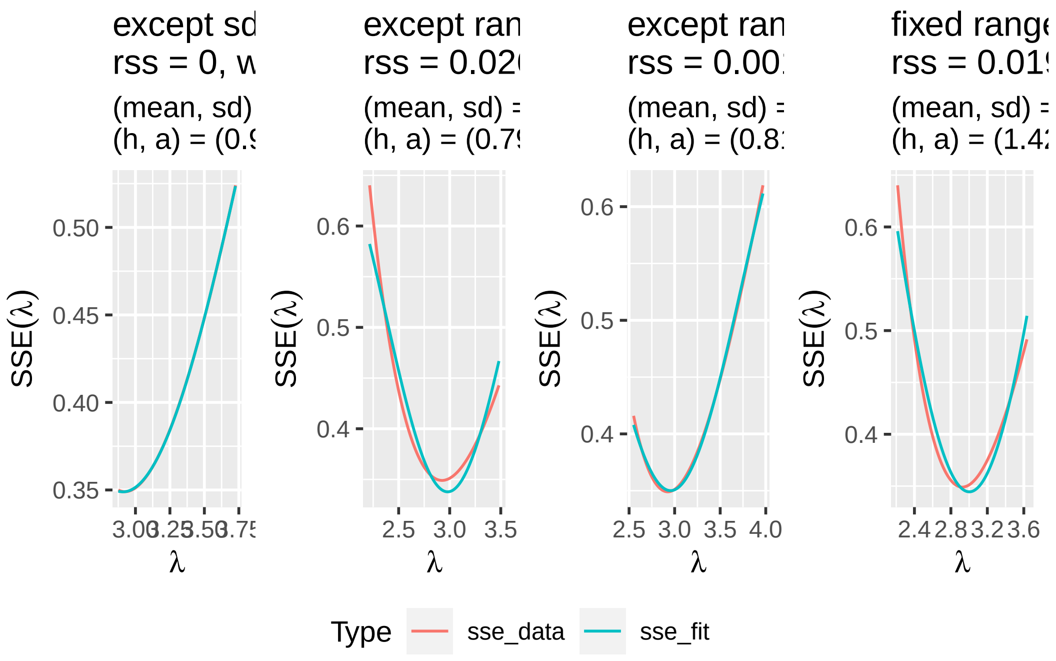
Based on the learning from the previous section, we should control the fitting range. In this study, we will control it with the initial values of the range (the location `range_lb` and the width `range`):

- Group 1 (case 1-4): the initial fitting range is set to be 1.5 standard deviations of the best fitting
- Group 2 (case 5-8): the initial fitting range is set to be 1.5 standard deviations of the best fitting

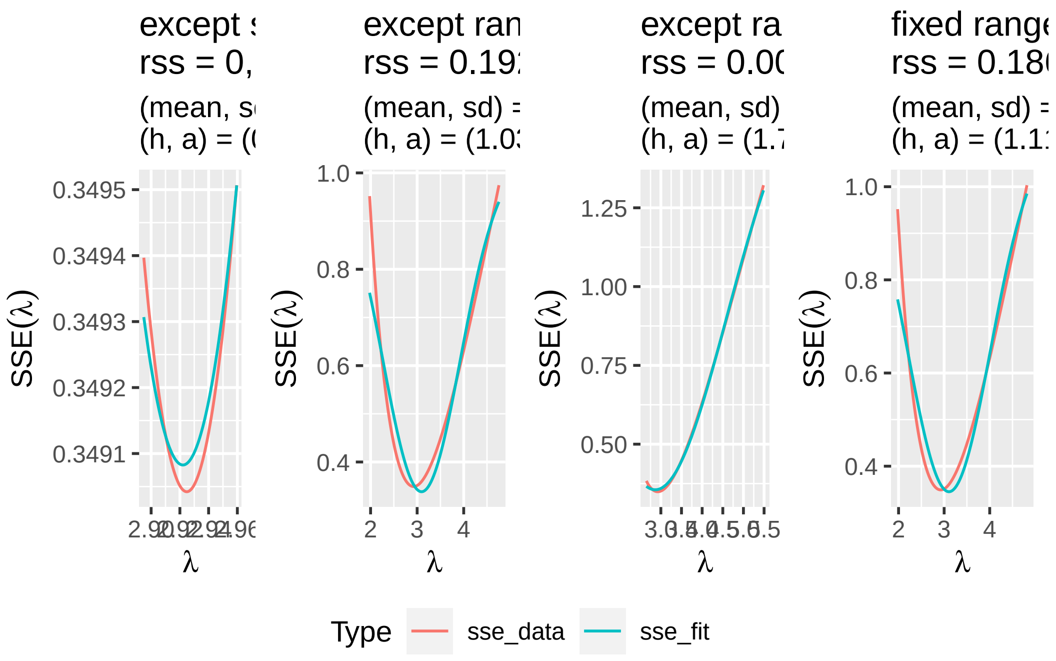
In both groups, the fitting range is centered initially and we try to keep one parameter free at a time in the first three cases (case 1-3 in group 1 and case 5-7 in group 2). In the last set (case 4 and 8) of each group, we will keep both the fitting ranges fixed to the initial values.

```r
# initial parameters and control which to fit
pars <- list(
  h = 1, a = 1, mean = bestfit$lambda, sd = bestfit$sd,
  range_lb = -0.75, range = 1.5, n = 100)
# optimization fit: fit all parameters except sd
opt1 <- c(1, 2, 3, 5, 6) %>%
  fit_optim(pars, pars_control = ., df, fmod, bestfit) %>%
  fit_summarize(title = "except sd")
# optimization fit: fit all parameters except range_lb
opt2 <- c(1, 2, 3, 4, 6) %>%
  fit_optim(pars, pars_control = ., df, fmod, bestfit) %>%
  fit_summarize(title = "except range_lb")
# optimization fit: fit all parameters except range
opt3 <- c(1, 2, 3, 4, 5) %>%
  fit_optim(pars, pars_control = ., df, fmod, bestfit) %>%
  fit_summarize(title = "except range")
# optimization fit: fit all parameters with fixed interval
opt4 <- c(1, 2, 3, 4) %>%
  fit_optim(pars, pars_control = ., df, fmod, bestfit) %>%
  fit_summarize(title = "fixed range")

# initial parameters and control which to fit
pars <- list(
  h = 1, a = 1, mean = bestfit$lambda, sd = bestfit$sd,
  range_lb = -1, range = 3, n = 100)
# optimization fit: fit all parameters except sd
opt5 <- c(1, 2, 3, 5, 6) %>%
  fit_optim(pars, pars_control = ., df, fmod, bestfit) %>%
  fit_summarize(title = "except sd")
# optimization fit: fit all parameters except range_lb
opt6 <- c(1, 2, 3, 4, 6) %>%
  fit_optim(pars, pars_control = ., df, fmod, bestfit) %>%
  fit_summarize(title = "except range_lb")
# optimization fit: fit all parameters except range
opt7 <- c(1, 2, 3, 4, 5) %>%
  fit_optim(pars, pars_control = ., df, fmod, bestfit) %>%
  fit_summarize(title = "except range")
# optimization fit: fit all parameters with fixed interval
opt8 <- c(1, 2, 3, 4) %>%
  fit_optim(pars, pars_control = ., df, fmod, bestfit) %>%
  fit_summarize(title = "fixed range")

# compare visually
ggarrange(
  opt1$p, opt2$p, opt3$p, opt4$p, opt5$p, opt6$p, opt7$p, opt8$p,
    ncol = 4, common.legend = TRUE, legend = "bottom")
```

```
$`1`
```



```
$`2`
```



```
attr(,"class")
[1] "list"      "ggarrange"
```

```r
# summarize in a table
bind_rows(
  opt1$pars, opt2$pars, opt3$pars, opt4$pars,
  opt5$pars, opt6$pars, opt7$pars, opt8$pars) %>% rowid_to_column()
```

rowid	title	rss	width	h	a	mean	sd	range_lb	range	n
1	except sd	0.0000059	0.85030	0.9105	1.3320	2.909	0.9460	0.05191	0.89890	100
2	except range_lb	0.0262100	1.26700	0.7906	0.6964	2.980	0.6134	0.75000	1.33900	100
3	except range	0.0007431	1.41900	0.8101	0.9016	2.955	0.7819	0.39640	1.50000	100
4	fixed range	0.0187200	1.41900	1.4210	2.9110	3.002	1.0790	0.75000	1.50000	100
5	except sd	0.0000002	0.06468	0.8903	1.2830	2.922	0.9460	0.03160	0.06837	100
6	except range_lb	0.1924000	2.77800	1.0350	1.4560	3.095	0.8330	1.00000	2.93600	100
7	except range	0.0039780	2.83800	1.7460	6.0260	2.865	1.7290	0.29220	3.00000	100
8	fixed range	0.1858000	2.83800	1.1180	1.7650	3.106	0.9115	1.00000	3.00000	100

```r
bind_rows(
  opt1$tbl, opt2$tbl, opt3$tbl, opt4$tbl,
  opt5$tbl, opt6$tbl, opt7$tbl, opt8$tbl) %>% rowid_to_column()
```

rowid	method	lambda	std.error	lower_0.95	upper_0.95	width
1	except sd	2.909	0.09460	2.814587	3.185413	0.3708252
2	except range_lb	2.980	0.06134	2.879776	3.120224	0.2404484
3	except range	2.955	0.07819	2.846750	3.153250	0.3064992
4	fixed range	3.002	0.10790	2.788520	3.211480	0.4229602
5	except sd	2.922	0.09460	2.814587	3.185413	0.3708252
6	except range_lb	3.095	0.08330	2.836735	3.163265	0.3265300
7	except range	2.865	0.17290	2.661122	3.338878	0.6777555
8	fixed range	3.106	0.09115	2.821349	3.178651	0.3573014

It is not surprising that the smallest RSS is obtained when we have more fitting parameter free to change. The trade off again (as seen in the previous section), the fitting range is very narrow and might become off centered. The obtained standard deviations (`sd`) have very wide range too. Based on the comparison plots, to ensure we use have a centered reasonable range, we need to keep the two parameters `range_lb` and `range` fixed. This means we cannot let the algorithm to determine the fitting range.

Fitting Strategy III: Grid Search

If we need to specify the fitting range, the remaining question is how the fitted standard deviation changes with the specified fitting range. To understand this, a grid search of `range_lb` and `range` has been performed:

```r
# a helper function to do the fitting used in a grid search setup
search_optim <- function(pars, range_lb.init, range.init){
  # initial parameters and control which to fit
  pars$range_lb = range_lb.init
  pars$range = range.init

  # optimization fit: fit all parameters except sd
  opt <- c(1, 2, 3, 4) %>%
    fit_optim(pars, pars_control = ., df, fmod, bestfit) %>%
    fit_summarize(
      title = glue(
        "init. (h, a, lb, range)= ",
        "({pars$h}, {pars$a}, {pars$range_lb}, {pars$range})"))
}

# initial parameters and control which to fit
pars <- list(
  h = 1, a = 1, mean = bestfit$lambda, sd = bestfit$sd,
  range_lb = NA, range = NA, n = 100)

# grid search
df_pscf <- expand_grid(range_lb.init = seq.int(-20, -5, 1)/10, range.init = seq.int(10, 40, 2)/
  filter(range.init >= 2*abs(range_lb.init)) %>%
  rowwise() %>%
  mutate(opt = list(search_optim(pars, range_lb.init, range.init)))

# summary table
p4_summary <- df_pscf %>%
  hoist(opt, pars = 1, tbl = 2, plot = 3) %>%
  unnest(pars)
p4_summary %>% select(1:2, rss:sd)
```

range_lb.init	range.init	rss	width	h	a	mean	sd
-2.0	4.0	21.470000	3.784	15.0200	110.9000	3.349	2.9470
-1.9	3.8	13.910000	3.595	10.8400	69.7000	3.324	2.5860
-1.9	4.0	14.160000	3.784	14.6900	118.2000	3.372	3.2380
-1.8	3.6	7.944000	3.406	15.2500	121.7000	3.274	3.2150
-1.8	3.8	9.239000	3.595	10.6600	73.2900	3.350	2.7860
-1.8	4.0	9.908000	3.784	11.0100	81.2800	3.415	2.9980
-1.7	3.4	5.115000	3.216	10.1300	67.2400	3.247	2.7020
-1.7	3.6	5.585000	3.406	11.0100	81.7900	3.329	3.0270
-1.7	3.8	6.350000	3.595	9.2090	63.5000	3.387	2.8250
-1.7	4.0	7.051000	3.784	7.7500	50.6100	3.432	2.6950
-1.6	3.2	3.125000	3.027	9.1240	58.7600	3.230	2.6390
-1.6	3.4	3.596000	3.216	7.9640	49.7500	3.288	2.5730
-1.6	3.6	4.148000	3.406	7.3030	44.2500	3.327	2.5000
-1.6	3.8	4.575000	3.595	6.0010	35.1200	3.405	2.4570
-1.6	4.0	4.757000	3.784	8.1600	59.5200	3.433	3.0250
-1.5	3.0	2.015000	2.838	6.0080	30.8000	3.186	2.1380
-1.5	3.2	2.303000	3.027	5.6270	29.4200	3.252	2.1950
-1.5	3.4	2.600000	3.216	5.5270	29.5700	3.301	2.2480

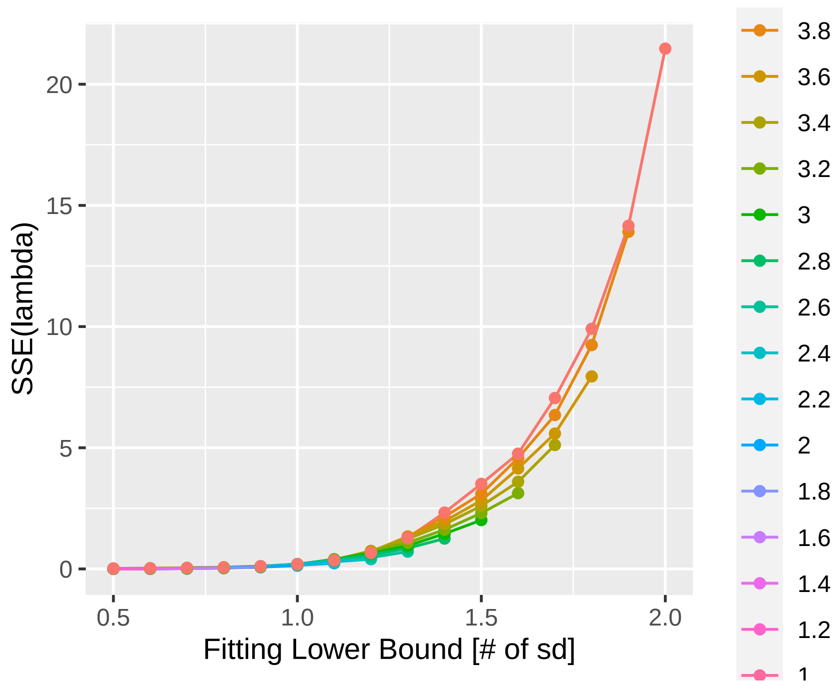
range_lb.init	range.init	rss	width	h	a	mean	sd
-1.5	3.6	2.825000	3.406	5.8960	34.5800	3.348	2.4690
-1.5	3.8	3.099000	3.595	5.6960	34.3100	3.391	2.5510
-1.5	4.0	3.511000	3.784	3.9040	18.6000	3.415	2.0780
-1.4	2.8	1.253000	2.649	4.5290	19.9100	3.153	1.8730
-1.4	3.0	1.452000	2.838	5.2780	26.3700	3.181	2.1050
-1.4	3.2	1.620000	3.027	4.3920	21.1400	3.271	2.0710
-1.4	3.4	1.844000	3.216	3.7200	16.5200	3.296	1.9440
-1.4	3.6	1.973000	3.406	4.3660	22.6700	3.327	2.2520
-1.4	3.8	2.154000	3.595	4.1540	21.5600	3.365	2.2640
-1.4	4.0	2.326000	3.784	2.6840	10.3000	3.369	1.7780
-1.3	2.6	0.714800	2.460	4.2290	18.2300	3.119	1.8510
-1.3	2.8	0.855600	2.649	3.7030	15.4600	3.179	1.8210
-1.3	3.0	0.958200	2.838	4.6700	24.5200	3.209	2.2600
-1.3	3.2	1.083000	3.027	4.3460	21.9500	3.253	2.1810
-1.3	3.4	1.237000	3.216	2.9300	11.6400	3.279	1.7990
-1.3	3.6	1.340000	3.406	2.0770	6.3530	3.281	1.4760
-1.3	3.8	1.333000	3.595	1.4900	3.1710	3.252	1.1040
-1.3	4.0	1.277000	3.784	1.4410	2.9410	3.247	1.0660
-1.2	2.4	0.402600	2.270	3.8240	15.6600	3.102	1.7760
-1.2	2.6	0.468000	2.460	4.1080	19.0400	3.153	2.0090
-1.2	2.8	0.570400	2.649	3.2360	13.2900	3.184	1.8290
-1.2	3.0	0.649400	2.838	3.3130	14.4400	3.207	1.9450
-1.2	3.2	0.732000	3.027	2.5480	9.3530	3.227	1.7040
-1.2	3.4	0.743600	3.216	1.3660	2.6780	3.198	1.0450
-1.2	3.6	0.709400	3.406	1.3150	2.4490	3.193	1.0040
-1.2	3.8	0.679300	3.595	1.3520	2.6160	3.198	1.0350
-1.2	4.0	0.659500	3.784	1.4150	2.9110	3.206	1.0890
-1.1	2.2	0.231700	2.081	2.6060	8.4190	3.081	1.4730
-1.1	2.4	0.294800	2.270	2.2900	7.0290	3.128	1.4340
-1.1	2.6	0.328500	2.460	2.7390	10.0700	3.151	1.6770
-1.1	2.8	0.380600	2.649	2.1460	6.7840	3.164	1.5110
-1.1	3.0	0.400400	2.838	1.2810	2.3710	3.151	1.0110
-1.1	3.2	0.380900	3.027	1.1950	2.0110	3.145	0.9418
-1.1	3.4	0.363000	3.216	1.2350	2.1780	3.149	0.9762
-1.1	3.6	0.352800	3.406	1.3030	2.4810	3.157	1.0360
-1.1	3.8	0.349600	3.595	1.3830	2.8570	3.165	1.1070
-1.1	4.0	0.351900	3.784	1.4680	3.2830	3.173	1.1820
-1.0	2.0	0.131800	1.892	1.8330	4.5180	3.051	1.2010
-1.0	2.2	0.150900	2.081	2.1830	6.6110	3.082	1.4310
-1.0	2.4	0.186600	2.270	1.7080	4.3340	3.110	1.2680
-1.0	2.6	0.204600	2.460	1.3160	2.5680	3.112	1.0590
-1.0	2.8	0.196400	2.649	1.0830	1.6310	3.103	0.8803
-1.0	3.0	0.185800	2.838	1.1180	1.7650	3.106	0.9115
-1.0	3.2	0.180500	3.027	1.1920	2.0680	3.112	0.9782
-1.0	3.4	0.180100	3.216	1.2770	2.4430	3.120	1.0550
-1.0	3.6	0.183500	3.406	1.3670	2.8730	3.127	1.1360
-1.0	3.8	0.189600	3.595	1.4590	3.3450	3.133	1.2180
-1.0	4.0	0.197800	3.784	1.5500	3.8530	3.138	1.3010
-0.9	1.8	0.065590	1.703	1.5790	3.4750	3.038	1.1160

range_lb.init	range.init	rss	width	h	a	mean	sd
-0.9	2.0	0.074390	1.892	2.2780	7.2740	3.059	1.5000
-0.9	2.2	0.094890	2.081	1.3980	2.9300	3.070	1.1130
-0.9	2.4	0.096480	2.270	0.9695	1.2690	3.065	0.8095
-0.9	2.6	0.090150	2.460	1.0020	1.3830	3.067	0.8406
-0.9	2.8	0.087220	2.649	1.0770	1.6630	3.073	0.9103
-0.9	3.0	0.087680	2.838	1.1680	2.0330	3.079	0.9941
-0.9	3.2	0.090600	3.027	1.2640	2.4600	3.085	1.0820
-0.9	3.4	0.095330	3.216	1.3610	2.9330	3.090	1.1700
-0.9	3.6	0.101400	3.406	1.4570	3.4440	3.094	1.2590
-0.9	3.8	0.108700	3.595	1.5510	3.9820	3.096	1.3460
-0.9	4.0	0.116800	3.784	1.6440	4.5520	3.098	1.4320
-0.8	1.6	0.030100	1.514	1.3480	2.5920	3.011	1.0280
-0.8	1.8	0.036700	1.703	1.7240	4.3820	3.032	1.2680
-0.8	2.0	0.043970	1.892	0.9508	1.2390	3.035	0.8183
-0.8	2.2	0.041140	2.081	0.8782	1.0070	3.034	0.7534
-0.8	2.4	0.039220	2.270	0.9596	1.2770	3.038	0.8330
-0.8	2.6	0.039640	2.460	1.0540	1.6280	3.044	0.9235
-0.8	2.8	0.041670	2.649	1.1560	2.0460	3.049	1.0190
-0.8	3.0	0.044860	2.838	1.2580	2.5120	3.053	1.1130
-0.8	3.2	0.048960	3.027	1.3590	3.0170	3.055	1.2070
-0.8	3.4	0.053800	3.216	1.4590	3.5590	3.057	1.3000
-0.8	3.6	0.059300	3.406	1.5560	4.1330	3.056	1.3910
-0.8	3.8	0.065380	3.595	1.6510	4.7360	3.054	1.4800
-0.8	4.0	0.072010	3.784	1.7430	5.3630	3.051	1.5690
-0.7	1.4	0.011620	1.324	1.4140	2.9190	2.990	1.0900
-0.7	1.6	0.016500	1.514	1.1330	1.8990	3.006	0.9654
-0.7	1.8	0.018060	1.703	0.9356	1.2290	3.011	0.8365
-0.7	2.0	0.016120	1.892	0.8404	0.9200	3.009	0.7449
-0.7	2.2	0.016250	2.081	0.9376	1.2410	3.013	0.8423
-0.7	2.4	0.017420	2.270	1.0430	1.6330	3.017	0.9441
-0.7	2.6	0.019320	2.460	1.1500	2.0840	3.021	1.0460
-0.7	2.8	0.021790	2.649	1.2570	2.5820	3.022	1.1470
-0.7	3.0	0.024760	2.838	1.3620	3.1180	3.022	1.2440
-0.7	3.2	0.028180	3.027	1.4640	3.6900	3.021	1.3410
-0.7	3.4	0.032030	3.216	1.5630	4.2920	3.018	1.4340
-0.7	3.6	0.036290	3.406	1.6600	4.9270	3.013	1.5270
-0.7	3.8	0.040950	3.595	1.7540	5.5920	3.006	1.6190
-0.7	4.0	0.045990	3.784	1.8450	6.2820	2.998	1.7100
-0.6	1.2	0.004807	1.135	0.9472	1.2200	2.973	0.8096
-0.6	1.4	0.006323	1.324	0.9799	1.3830	2.986	0.8744
-0.6	1.6	0.007057	1.514	0.9374	1.2730	2.990	0.8659
-0.6	1.8	0.005846	1.703	0.8134	0.8658	2.987	0.7440
-0.6	2.0	0.006392	1.892	0.9295	1.2510	2.991	0.8632
-0.6	2.2	0.007364	2.081	1.0360	1.6550	2.993	0.9670
-0.6	2.4	0.008687	2.270	1.1490	2.1370	2.994	1.0750
-0.6	2.6	0.010330	2.460	1.2590	2.6630	2.993	1.1800
-0.6	2.8	0.012270	2.649	1.3660	3.2270	2.991	1.2810
-0.6	3.0	0.014520	2.838	1.4710	3.8290	2.986	1.3800
-0.6	3.2	0.017060	3.027	1.5710	4.4610	2.980	1.4770

range_lb.init	range.init	rss	width	h	a	mean	sd
-0.6	3.4	0.019910	3.216	1.6700	5.1280	2.972	1.5730
-0.6	3.6	0.023060	3.406	1.7650	5.8220	2.962	1.6670
-0.6	3.8	0.026500	3.595	1.8570	6.5450	2.951	1.7610
-0.6	4.0	0.030240	3.784	1.9460	7.2920	2.937	1.8530
-0.5	1.0	0.001400	0.946	0.9464	1.2450	2.959	0.8292
-0.5	1.2	0.002110	1.135	0.9253	1.2310	2.968	0.8532
-0.5	1.4	0.001797	1.324	0.7031	0.5752	2.969	0.6479
-0.5	1.6	0.001943	1.514	0.7987	0.8474	2.969	0.7534
-0.5	1.8	0.002360	1.703	0.9232	1.2640	2.971	0.8825
-0.5	2.0	0.002964	1.892	1.0350	1.6960	2.972	0.9923
-0.5	2.2	0.003756	2.081	1.1510	2.2000	2.970	1.1040
-0.5	2.4	0.004739	2.270	1.2620	2.7450	2.967	1.2100
-0.5	2.6	0.005922	2.460	1.3720	3.3400	2.961	1.3160
-0.5	2.8	0.007313	2.649	1.4780	3.9670	2.954	1.4170
-0.5	3.0	0.008922	2.838	1.5810	4.6340	2.944	1.5180
-0.5	3.2	0.010750	3.027	1.6800	5.3300	2.933	1.6170
-0.5	3.4	0.012810	3.216	1.7760	6.0510	2.920	1.7130
-0.5	3.6	0.015100	3.406	1.8690	6.8100	2.905	1.8100
-0.5	3.8	0.017620	3.595	1.9590	7.5930	2.888	1.9050
-0.5	4.0	0.020370	3.784	2.0460	8.3980	2.869	1.9980

The summary table is huge and hard to read. It is easier to plot $SSE(\lambda)$ as a function of the fitting range: `range_lb` as for its location in the horizontal axis and `range` as for its width in different color:

```r
# plot the overall comparison
p4_summary %>%
  ggplot(aes(abs(range_lb.init), rss, col = fct_rev(factor(range.init)))) +
  geom_line() +
  geom_point() +
  labs(
    x = "Fitting Lower Bound [# of sd]",
    y = "SSE(lambda)",
    color = "Fitting Range [# of sd]"
  )
```



As seen in the plot, the minimization of $SSE(\lambda)$ will leads the fitting algorithm to use very narrow fitting range. When we force it to use a large fitting range, the $SSE(\lambda)$ becomes bigger. This clearly explains why this algorithm could not give us a stable fitted standard deviation `sd` - there is no global minimum.

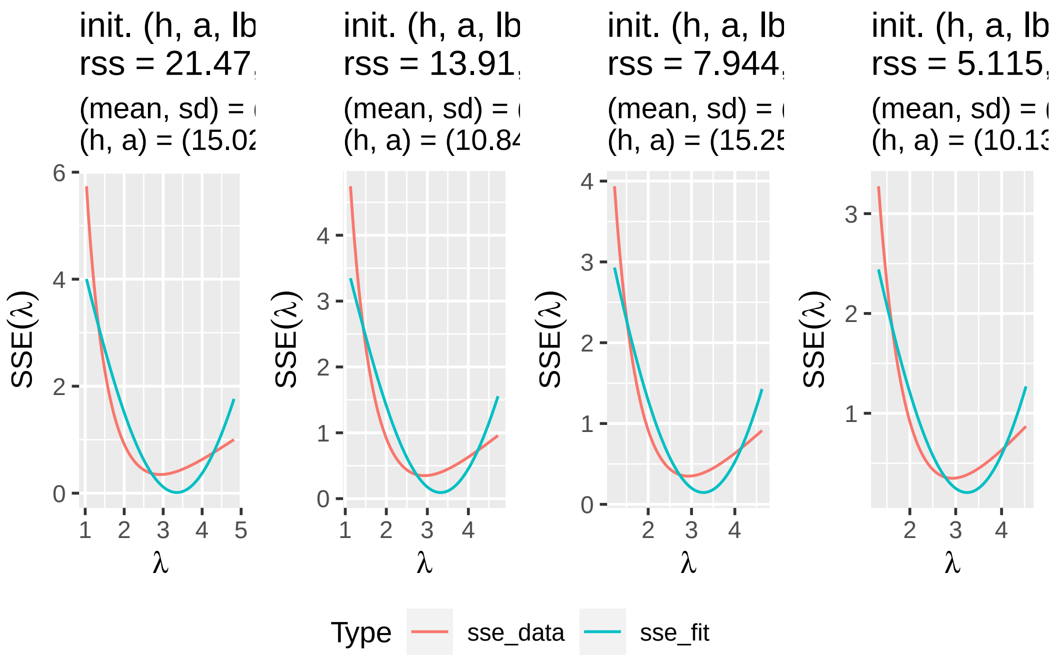
```
# focus on the centered cases
p4_centered <- p4_summary %>% filter(range.init == 2*abs(range_lb.init))
p4_centered %>% select(1:2, title, rss:sd)
```

range_lb.init	range.init	title	rss	width	h	a	mean	sd
-2.0	4.0	init. (h, a, lb, range)= (1, 1, -2, 4)	21.470000	3.784	15.0200	110.900	3.349	2.9470
-1.9	3.8	init. (h, a, lb, range)= (1, 1, -1.9, 3.8)	13.910000	3.595	10.8400	69.700	3.324	2.5860
-1.8	3.6	init. (h, a, lb, range)= (1, 1, -1.8, 3.6)	7.944000	3.406	15.2500	121.700	3.274	3.2150
-1.7	3.4	init. (h, a, lb, range)= (1, 1, -1.7, 3.4)	5.115000	3.216	10.1300	67.240	3.247	2.7020
-1.6	3.2	init. (h, a, lb, range)= (1, 1, -1.6, 3.2)	3.125000	3.027	9.1240	58.760	3.230	2.6390
-1.5	3.0	init. (h, a, lb, range)= (1, 1, -1.5, 3)	2.015000	2.838	6.0080	30.800	3.186	2.1380
-1.4	2.8	init. (h, a, lb, range)= (1, 1, -1.4, 2.8)	1.253000	2.649	4.5290	19.910	3.153	1.8730
-1.3	2.6	init. (h, a, lb, range)= (1, 1, -1.3, 2.6)	0.714800	2.460	4.2290	18.230	3.119	1.8510
-1.2	2.4	init. (h, a, lb, range)= (1, 1, -1.2, 2.4)	0.402600	2.270	3.8240	15.660	3.102	1.7760
-1.1	2.2	init. (h, a, lb, range)= (1, 1, -1.1, 2.2)	0.231700	2.081	2.6060	8.419	3.081	1.4730

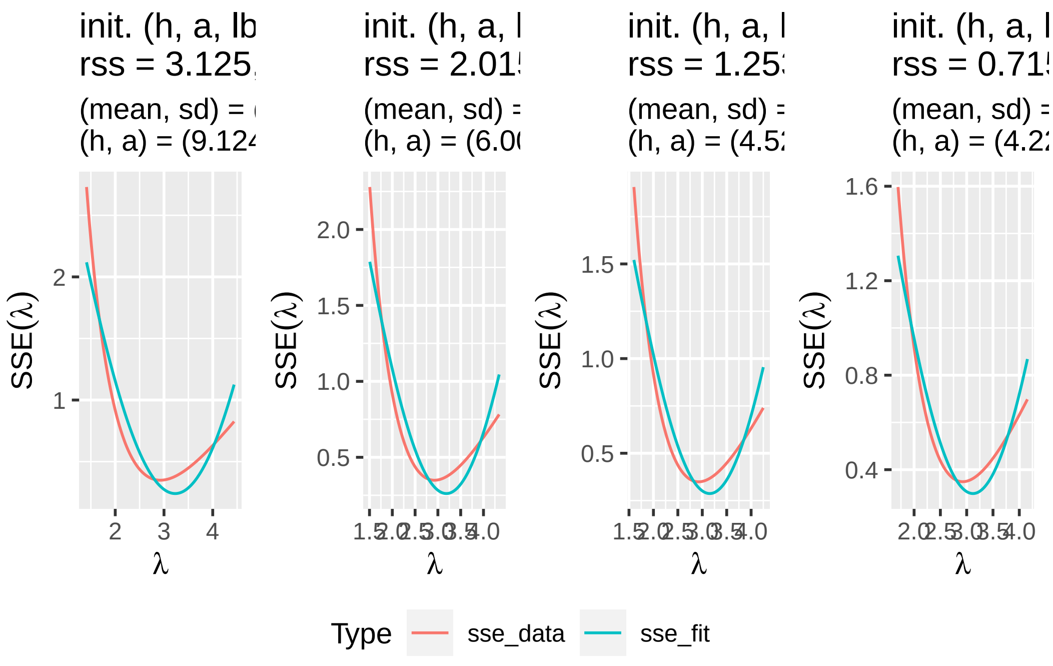
range_lb.init	range.init	title	rss	width	h	a	mean	sd
-1.0	2.0	init. (h, a, lb, range)= (1, 1, -1, 2)	0.131800	1.892	1.8330	4.518	3.051	1.2010
-0.9	1.8	init. (h, a, lb, range)= (1, 1, -0.9, 1.8)	0.065590	1.703	1.5790	3.475	3.038	1.1160
-0.8	1.6	init. (h, a, lb, range)= (1, 1, -0.8, 1.6)	0.030100	1.514	1.3480	2.592	3.011	1.0280
-0.7	1.4	init. (h, a, lb, range)= (1, 1, -0.7, 1.4)	0.011620	1.324	1.4140	2.919	2.990	1.0900
-0.6	1.2	init. (h, a, lb, range)= (1, 1, -0.6, 1.2)	0.004807	1.135	0.9472	1.220	2.973	0.8096
-0.5	1.0	init. (h, a, lb, range)= (1, 1, -0.5, 1)	0.001400	0.946	0.9464	1.245	2.959	0.8292

```r
# plot the centered case
ggarrange(
  plotlist = p4_centered$plot,
  ncol = 4, common.legend = TRUE, legend = "bottom")
```

```
$`1`
```

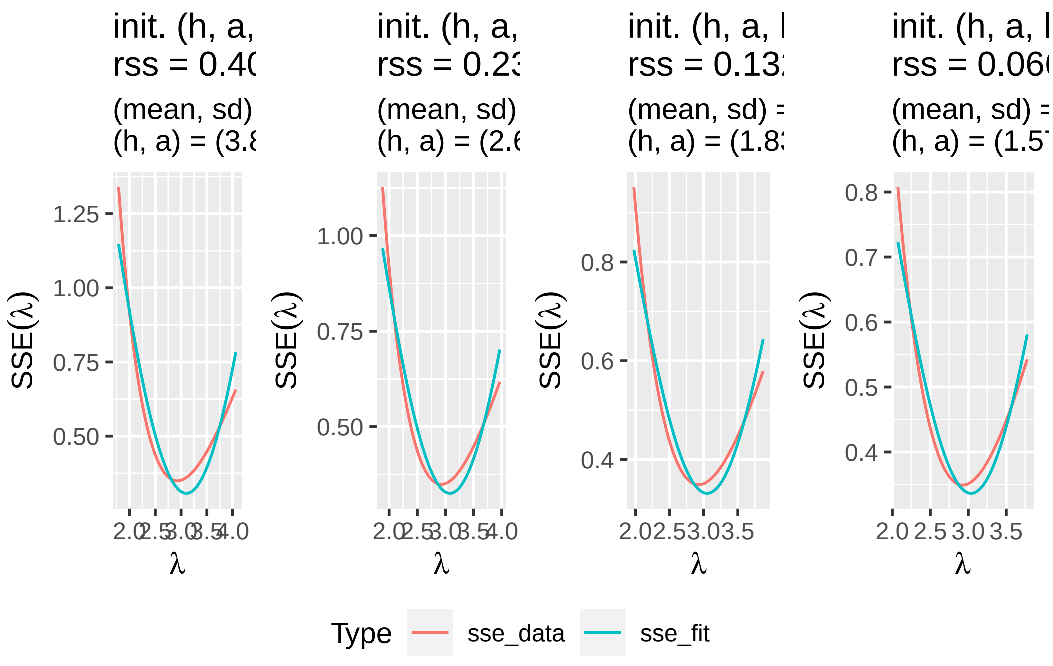


```
$`2`
```



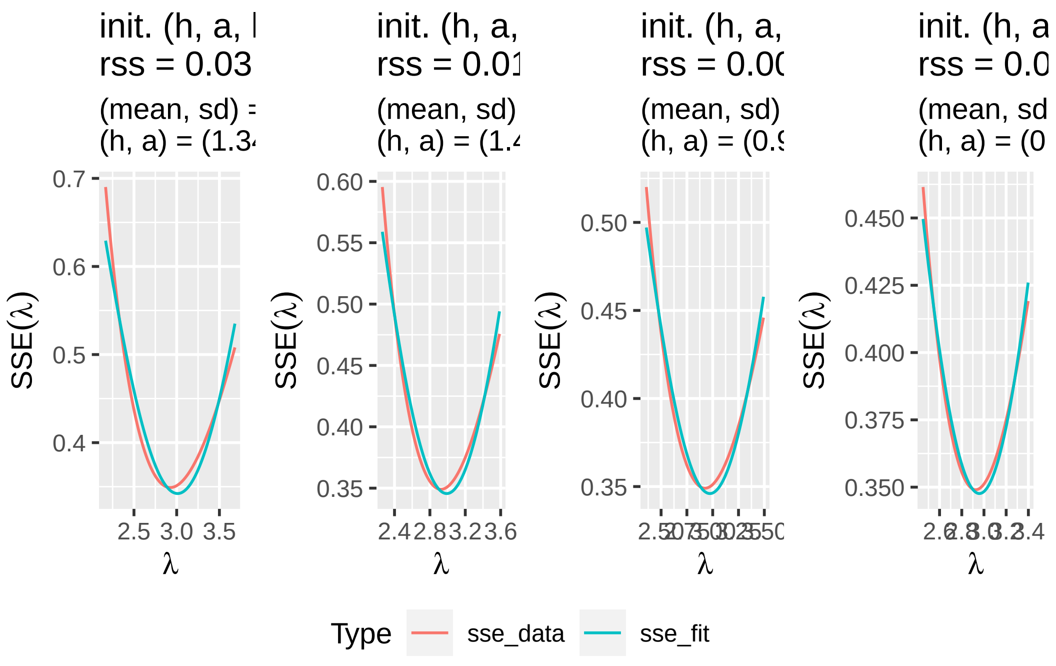
init. (h, a, lb
rss = 3.125
(mean, sd) =
(h, a) = (9.124
init. (h, a, l
rss = 2.015
(mean, sd) =
(h, a) = (6.00
init. (h, a, l
rss = 1.253
(mean, sd) =
(h, a) = (4.52
init. (h, a, l
rss = 0.715
(mean, sd) =
(h, a) = (4.22
SSE(λ)
λ
Type
sse_data
sse_fit

```
$`3`
```



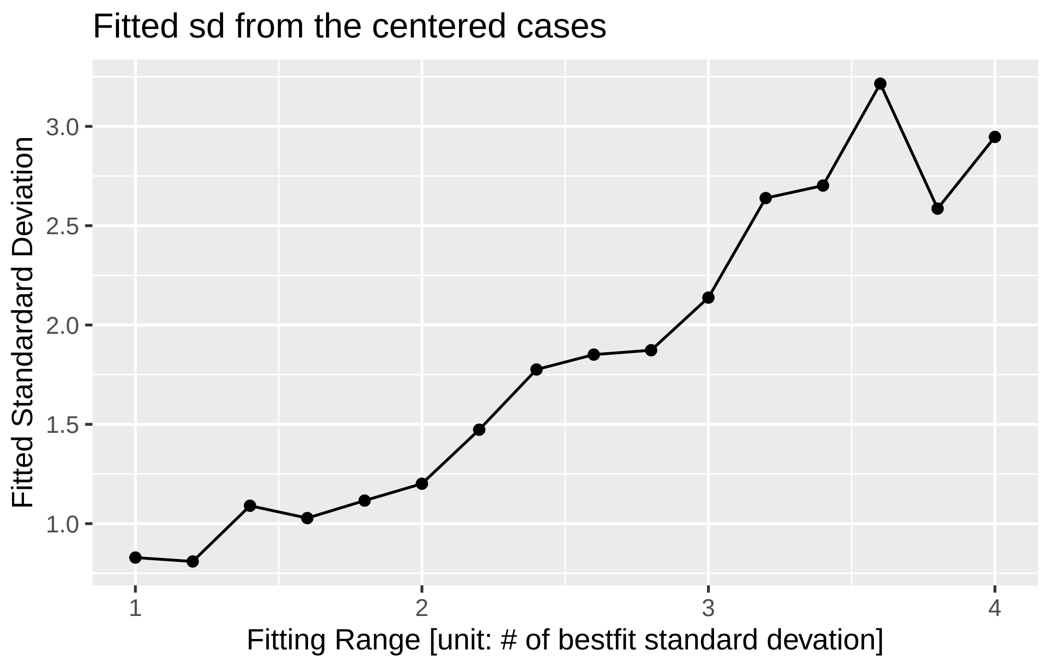
init. (h, a,
rss = 0.40
(mean, sd)
(h, a) = (3.8
init. (h, a,
rss = 0.23
(mean, sd)
(h, a) = (2.6
init. (h, a, l
rss = 0.13
(mean, sd) =
(h, a) = (1.83
init. (h, a, l
rss = 0.06
(mean, sd) =
(h, a) = (1.5
SSE(λ)
λ
Type
sse_data
sse_fit

```
$`4`
```



```
attr(,"class")
[1] "list"      "ggarrange"
```

```
# summary plot for the centered cases
p4_centered %>% ggplot(aes(range.init, sd)) +
  geom_line() +
  geom_point() +
  labs(
    x = "Fitting Range [unit: # of bestfit standard devation]",
    y = "Fitted Standardard Deviation",
    title = "Fitted sd from the centered cases"
  )
```



The algorithm can give a very wide estimated (fitted) standard deviation depending on the initial values of the fitting range (location `range_lb` and width `range`). Even for the cases with forced centered ranges, the obtained fitting results are not stable.

Bootstrapping CI

Bootstrapping is a very powerful method to sample estimates by using random sampling with replacement. The central idea is assuming the bootstrap statistics vary in a similar fashion to the sample statistic of interest. **Resample** of the sample of same sample size with replacement. Bootstrap has two major uses (Wood 2009): 1. Approximating the shape of a sampling distribution (to form a CI) 2. Estimating the spread of an estimator

$$\begin{aligned} \hat{SE}(\underset{\sim}{\hat{\alpha}}) &= \sqrt{\frac{1}{B}\sum_{i=1}^{B}\left(\hat{\alpha}_j^* - \bar{\hat{\alpha}}^*\right)^2} \\ &= \sqrt{\begin{array}{c}\text{Biased sample variance}\\ \text{of bootstrapped estimates}\end{array}} \end{aligned} \tag{8}$$

The assumption of bootstrapping is that variation between each resample (bootstrap sample) gives a good idea of sampling error when sampling a real population. Although the resample is not the real population, it is reasonably similar to sampling from the population under this assumption and thus provides a good estimate or test measure of interest. This is the main reason we choose this method in this study.

In general, the workflow of bootstrap confidence intervals is:

1. resample B stacks of bootstrap samples (same sample size n with replacement)
    - this can be done from observations (non-parametric) or from a known distribution (parametric)
2. calculate B (plug-in) bootstrap estimates of the sample statistic
3. assume these bootstrap statistics vary in a similar fashion to your sample statistic
    - to approximate shape of the distribution CI
    - to obtained spread of the estimator $\hat{\text{SE}}(\underset{\sim}{\hat{\alpha}})$

For each bootstrap sample, we can obtain the parameter estimate using parametric method or non-parametric method.

Parametric Bootstrap CI

Based on the physical model, we can estimate the parameter $\hat{\lambda} = -\frac{1}{x}\ln y$ (Equation 1) using each bootstrap sample and calculate its average (Equation 2):

1. Resample data with the same sample size n
2. For each resample observation, compute the parameter estimate from (x, y)
3. Compute the average estimate
4. Repeat for N times

Non-Parametric Bootstrap CI

For the non-parametric approach, we use the bootstrap sample to refit a nonlinear regression model and obtain the NLR estimate for the parameter.

1. Resample data with the same sample size n
2. For each resample data set, fit a nonlinear regression model

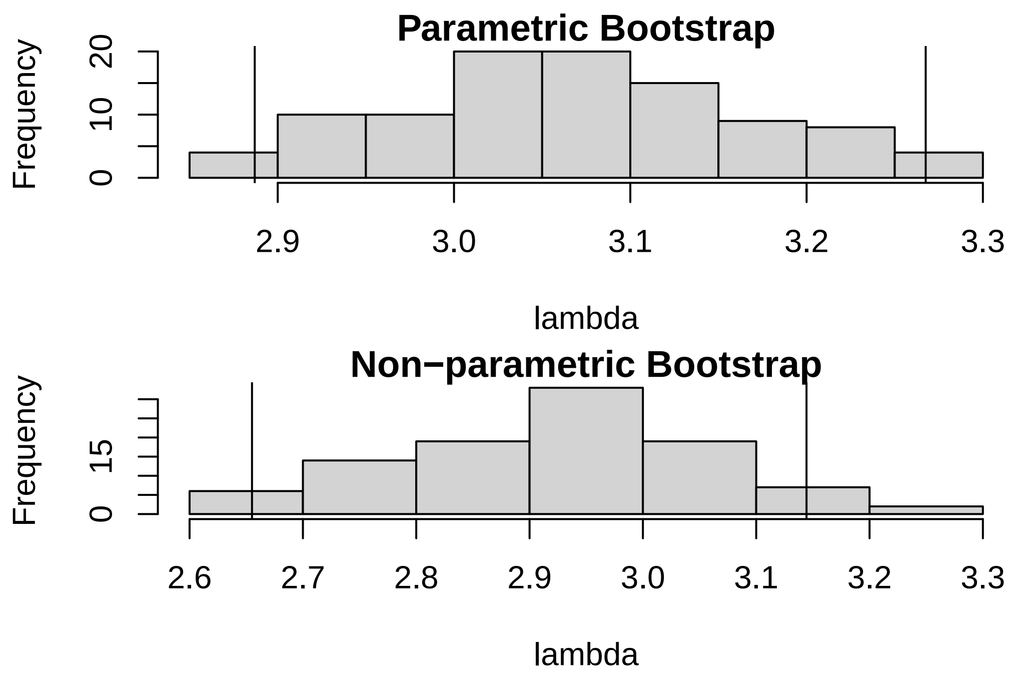
3. Obtain NLR estimate
4. Repeat for N times

Bootstrap CI Results

We then use the distribution of the bootstrap statistics obtained above to approximate the shape of the distribution CI. Furthermore, for computational efficiency, we combine these two into one loop:

```r
# number of bootstrapping samples
N <- 0.1*1000

# bootstrap for N times
boot <- 1:N %>%
  sapply(\(x){
    # resample: index of rows
    idx <- sample(1:n, n, replace = T)
    # resample data
    df_boot <- df[idx, ] %>%
      # bootstrap A: calculate the lambda for each resampled observation
      mutate(estimate = fmod_inv(y, x))

    # bootstrap B: fit nlr model for each resampled data set
    mod_boot <- nls(y ~ fmod(x, lambda), df_boot, start = c(lambda = lambda_mean))

    # return the estimates
    c(A = mean(df_boot$estimate), B = tidy(mod_boot)$estimate)
  }) %>%
  # transpose and convert into a data frame
  t() %>%
  as_tibble()
```

```r
# histogram and CI

# save and restore option
op <- par(pty="m", mfrow=c(2, 1), mar=c(4.2, 4.2, 1, 1))

hist(boot$A, main = "Parametric Bootstrap", xlab = "lambda")
abline(v = quantile(boot$A,  probs = c(0.05/2, 1-0.05/2)))
hist(boot$B, main = "Non-parametric Bootstrap", xlab = "lambda")
abline(v = quantile(boot$B,  probs = c(0.05/2, 1-0.05/2)))
```



```r
# restore option
par(op)
```

```r
# save as a comparison table
tbl_bootA <- tibble(
  method = "Parametric Bootstrap",
  lambda = mean(boot$A),
  std.error = NA,
  lower_0.95 = quantile(boot$A,  probs = 0.05/2),
  upper_0.95 = quantile(boot$A,  probs = 1 - 0.05/2),
  width = upper_0.95 - lower_0.95)

tbl_bootB <- tibble(
  method = "Non-parametric Bootstrap",
  lambda = mean(boot$B),
  std.error = NA,
  lower_0.95 = quantile(boot$B,  probs = 0.05/2),
  upper_0.95 = quantile(boot$B,  probs = 1 - 0.05/2),
  width = upper_0.95 - lower_0.95)
```

Delta Method Normality

Delta method normality is theorem that can be used to derive the distribution of a function of an asymptotically normal variable. Combining the advantages of the two powerful statistic concepts: **large sample normality** and **delta method**, it is very useful to approximate probability distribution for a function of an asymptotically normal statistical estimator from knowledge of the limiting variance of that estimator (Kelley 1928). The main appealing feature of this method is that we don't need know much about the expected value and variance of the function g due to its asymptotic normality.

Univariate Delta Method

If $Y \overset{\bullet}{\sim} N(\mu, \sigma^2)$ and there exist a transformation function g and value μ and $g'(\mu) \neq 0$, then the Taylor series expansion (Taylor approximation) is used to derive variation around a point (Doob 1935):

$$g(Y) \overset{\bullet}{\sim} N\Big(g(\mu),\, [g'(\mu)]^2\, \sigma^2\Big) \tag{9}$$

Now consider i.i.d. sample Y_i for $g(\bar{Y})$ as a sequence of RVs such that $Y_n \overset{\bullet}{\sim} N(\mu, \sigma^2/n)$. For a function g and value θ_0 where $g'(\theta_0)$ exists and is not 0, we approximate the distribution of $g(Y_n)$ using Delta method:

$$g(Y_n) \overset{\bullet}{\sim} N\Big(g(\theta_0),\, [g'(\theta_0)]^2\, \sigma^2/n\Big)$$

Bivariate Delta Method

For a two-variable function $Z = g(X, Y) = -\frac{1}{X} \ln Y$:

$$\begin{aligned}
Z &= g(X, Y) \\
&\approx g(\mu_X, \mu_Y) + (X - \mu_X)\partial_x g(\mu) + (Y - \mu_Y)\partial_y g(\mu) \\
&+ \frac{1}{2}(X - \mu_X)^2 \partial_x^2 g(\mu) + \frac{1}{2}(Y - \mu_Y)^2 \partial_y^2 g(\mu) \\
&+ (X - \mu_X)(Y - \mu_Y)\partial_{xy}^2 g(\mu)
\end{aligned}$$

$$\begin{aligned}
E[Z] &\approx g(\mu_X, \mu_Y) \\
Var(Z) &\approx \big(\partial_x g(\mu_X, \mu_Y)\big)^2 \sigma_X^2 + \big(\partial_y g(\mu_X, \mu_Y)\big)^2 \sigma_Y^2 + 2\sigma_{XY}\partial_x g(\mu_X, \mu_Y)\partial_y g(\mu_X, \mu_Y) \\
\Rightarrow g(X, Y) &\overset{\bullet}{\sim} N\Big(g(\mu_X, \mu_Y),\, [g'(\theta_0)]^2\, \sigma^2/n\Big)
\end{aligned}$$

Delta Method Results

With the estimator of our model parameter $\hat{\lambda}$ (Equation 1), we have $g(Y) = -\frac{1}{x} \ln Y$. Therefore, we can estimate the distribution of the model parameter:

```
# g(Y_n)
gs <- -1/df$x * log(df$y)
```

```r
# CI
tbl_delta1 <- tibble(
  method = "Delta Method Normality",
  lambda = mean(gs),
  std.error =  sd(gs)/sqrt(n),
  lower_0.95 = mean(gs) + qt(0.05/2, n-1) * std.error,
  upper_0.95 = mean(gs) - qt(0.05/2, n-1) * std.error,
  width = upper_0.95 - lower_0.95)
tbl_delta1
```

method	lambda	std.error	lower_0.95	upper_0.95	width
Delta Method Normality	3.067518	0.0993012	2.870483	3.264553	0.3940704

Note that we use t-distribution to compute the $100(1-\alpha)\%$ CI.

Comparison

```r
# comparison
rbind(tbl_nlr, tbl_bootA, tbl_bootB, tbl_delta1, tbl_truth)
```

method	lambda	std.error	lower_0.95	upper_0.95	width
Nonlinear Regression	2.924786	0.0946027	2.812288	3.187712	0.3754246
Parametric Bootstrap	3.066531	NA	2.886955	3.267549	0.3805939
Non-parametric Bootstrap	2.920003	NA	2.655081	3.144379	0.4892980
Delta Method Normality	3.067518	0.0993012	2.870483	3.264553	0.3940704
Truth (All sample sizes = 100)	3.000000	0.1000000	2.801578	3.198422	0.3968434

Conclusion

In this study, we introduce the algorithm developed by Prof. Hayes to estimate the standard deviation of a model parameter by fitting a Gaussian curve. After implementing the algorithm, we perform a extensive study on a simulated data set to test how well the algorithm performs in the estimating the parameter's standard deviation. In this study, we have shown that the obtained fitting results are not stable and greatly depend on the initial fitting values of the parameters. Among them, the parameters associating with the fitting range (the location `range_lb` and the width `range`) have the biggest impacts on the fitting results.

```r
# compare all centered cases
p4_centered %>%
  mutate(method = "PSCF - Centered", std.error = sd/sqrt(n)) %>%
  rename(lambda = mean, fitting_initial_setup = title) %>%
  select(method, fitting_initial_setup, lambda, std.error)
```

method	fitting_initial_setup	lambda	std.error
PSCF - Centered	init. (h, a, lb, range)= (1, 1, -2, 4)	3.349	0.29470
PSCF - Centered	init. (h, a, lb, range)= (1, 1, -1.9, 3.8)	3.324	0.25860
PSCF - Centered	init. (h, a, lb, range)= (1, 1, -1.8, 3.6)	3.274	0.32150
PSCF - Centered	init. (h, a, lb, range)= (1, 1, -1.7, 3.4)	3.247	0.27020
PSCF - Centered	init. (h, a, lb, range)= (1, 1, -1.6, 3.2)	3.230	0.26390
PSCF - Centered	init. (h, a, lb, range)= (1, 1, -1.5, 3)	3.186	0.21380
PSCF - Centered	init. (h, a, lb, range)= (1, 1, -1.4, 2.8)	3.153	0.18730
PSCF - Centered	init. (h, a, lb, range)= (1, 1, -1.3, 2.6)	3.119	0.18510
PSCF - Centered	init. (h, a, lb, range)= (1, 1, -1.2, 2.4)	3.102	0.17760
PSCF - Centered	init. (h, a, lb, range)= (1, 1, -1.1, 2.2)	3.081	0.14730
PSCF - Centered	init. (h, a, lb, range)= (1, 1, -1, 2)	3.051	0.12010
PSCF - Centered	init. (h, a, lb, range)= (1, 1, -0.9, 1.8)	3.038	0.11160
PSCF - Centered	init. (h, a, lb, range)= (1, 1, -0.8, 1.6)	3.011	0.10280
PSCF - Centered	init. (h, a, lb, range)= (1, 1, -0.7, 1.4)	2.990	0.10900
PSCF - Centered	init. (h, a, lb, range)= (1, 1, -0.6, 1.2)	2.973	0.08096
PSCF - Centered	init. (h, a, lb, range)= (1, 1, -0.5, 1)	2.959	0.08292

Even if we force the fitting range to be centered around the best fitted value, the range of the fitted standard deviation `sd` is still very wide. This is because the objective function we use to minimize in **the algorithm doesn't have a global minimum**.

While we cannot get a reliable estimation on the parameter's standard deviation using the proposed algorithm, we can still take advantages of the statistical methods like non-linear regression, bootstrapping CI and Delta method normality. These methods are well developed and have been used in many applications. In this study, we have showed that these methods are capable to obtain pretty good estimation on the model parameter.

```r
# comparison - using statistical methods
rbind(tbl_nlr, tbl_bootA, tbl_bootB, tbl_delta1, tbl_truth)
```

method	lambda	std.error	lower_0.95	upper_0.95	width
Nonlinear Regression	2.924786	0.0946027	2.812288	3.187712	0.3754246
Parametric Bootstrap	3.066531	NA	2.886955	3.267549	0.3805939
Non-parametric Bootstrap	2.920003	NA	2.655081	3.144379	0.4892980
Delta Method Normality	3.067518	0.0993012	2.870483	3.264553	0.3940704
Truth (All sample sizes = 100)	3.000000	0.1000000	2.801578	3.198422	0.3968434

References

Brake, A., and West, R. (2021), "Retrospective dosimetry for nuclear nonproliferation and emergency response," North Carolina State University.

Brown, F. B., Sweezy, J. E., and Hayes, R. B. (2004), "Monte carlo parameter studies and uncertainty analyses with MCNP5."

Doob, J. L. (1935), "The Limiting Distributions of Certain Statistics," *The Annals of Mathematical Statistics*, 6, 160–169. https://doi.org/10.1214/aoms/1177732594.

Hayes, R. B., and O'Mara, R. P. (2019), "Retrospective dosimetry at the natural background level with commercial surface mount resistors," *Radiation Measurements*, 121. https://doi.org/10.1016/j.radmeas.2018.12.007.

Kelley, T. L. (1928), *Crossroads in the mind of man; a study of differentiable mental abilities.*, Stanford Univ. Press. https://doi.org/10.1037/11206-000.

O'Mara, R. P. (2020), *Retrospective dosimetry for nuclear nonproliferation and emergency response*, North Carolina State University.

O'Mara, R., and Hayes, R. (2018), "Dose Deposition Profiles in Untreated Brick Material," *Health Physics*, 114, 414–420. https://doi.org/10.1097/hp.0000000000000843.

Seber, G. A. F., and Wild, C. J. (1989), *Nonlinear regression*, John Wiley & Sons, Inc. https://doi.org/10.1002/0471725315.

Wood, M. (2009), "Bootstrapping confidence levels for hypotheses about quadratic (u-shaped) regression models." https://doi.org/10.48550/ARXIV.0912.3880.