The general survival function of a Weibull regression model can be specified as

$$ S(t) = \exp(\lambda t ^ \gamma). $$

By introducing the exponent γ in the term below, we allow the hazard to change over time. Hence, we do not need to assume a constant hazard function across time of follow up. Just as a reminder in the Possion regression model our hazard function was just equal to λ. In case of a Weibull regression model our hazard function is

$$ h(t) = \gamma \lambda t ^ {\gamma -1} $$

where

$$ \lambda = \exp(\alpha + \beta_1 x_{female} + \beta_2 x_{age}). $$

Using this more complex hazard function we can fit changes in the hazard across time of follow up.

So now let's get started with loading the data set and setting up the variables.

```
# 1. Prefix ------------------------------
-------------------------------------
# Remove all files from ls
rm(list = ls())
# Loading packages
require(survival)
require(flexsurv)
require(optimx)
require(numDeriv)
require(dplyr)
require(tibble)
require(car)
# 2. Loading dataset ---------------------------------
---------------------------
#Reading the example data set lung from the survival package
lung <- as.data.frame(survival::lung)
#Recode dichotomous vairables
lung$female <- ifelse(lung$sex == 2, 1, 0)
lung$status_n <- ifelse(lung$status == 2, 1, 0)
```

If we now want to use the likelihood function to fit our Weibull regression model we first need to specify our likelihood function. The general likelihood function for survival model can be written as

$$ \ln L_i = d_i \ln h(t_i) + \ln S(t_i). $$

By substituting our previous defined hazard and survival function we get

$$ \ln L = d \ln(\gamma \lambda t ^ {\gamma - 1}) + \exp(\lambda) t ^ \gamma $$

for the log likelihood function of our Weibull model. To find the estimates of our Weibull model that best fit our data, we need to find the maximum of this function. Hence, the next step is to implement this function in `R` so that we can use it for our `optimx()` call.

```
# 3. Define log-likelihood function for Weibull regression model
--------------
negll <- function(par){
#Extract guesses for alpha, gamma, beta1 and beta2
gamma <- par[1]
alpha <- par[2]
beta1 <- par[3]
beta2 <- par[4]
#Define dependent and independent variables
t <- lung$time
d <- lung$status_n
x1 <- lung$female
x2 <- lung$age
#Calculate lambda and gamma
lambda <- (alpha + beta1 * x1 + beta2 * x2)
egamma <- exp(gamma)
#Estimate negetive log likelihood value
val <- -sum(d * (log(egamma * t ^ (egamma - 1)) + lambda) -
exp(lambda) * t ^ egamma)
return(val)
}
```

Additionally, we can pass the analytical gradient function of our likelihood function to our `optimx()` call to improve our estimates. After partially deriving our log likelihood function $\ln L_i$ for α, γ and β_i, we yield the following equations for the gradient of $\ln L_i$.

$$
\sum d_i - \exp(\lambda_i) t_i ^ {\exp(\gamma_i)} = 0
$$

$$
\sum (d_i \ln(t_i) - t_i \exp(\gamma_i) \ln(t_i) \exp(\lambda_i)) \exp(\gamma_i) + d_i = 0
$$

$$
\sum d_i * x_{ij} - \exp(\lambda_i) x_{ij} t_i ^ {\exp(\gamma_i)} = 0
$$

Using these equations we get the following function for our gradient in `R`.

```
# 4. Define gradient function for Weibull regression model
--------------------
negll.grad <- function(par){
#Extract guesses for alpha, gamma, beta1 and beta2
gamma <- par[1]
alpha <- par[2]
beta1 <- par[3]
beta2 <- par[4]
#Define dependent and independent variables
t <- lung$time
d <- lung$status_n
x1 <- lung$female
x2 <- lung$age
```

```
#Create output vector
n <- length(par[1])
gg <- as.vector(rep(0, n))
#Calculate lambda
lambda <- (alpha + beta1 * x1 + beta2 * x2)
#Calculate partial gradient functions
gg[1] <- -sum((d * log(t) -
t ^ exp(gamma) * log(t) * exp(lambda)) * exp(gamma) + d)
gg[2] <- -sum(d - exp(lambda) * t ^ exp(gamma))
gg[3] <- -sum(d * x1 - exp(lambda) * x1 * t ^ exp(gamma))
gg[4] <- -sum(d * x2 - exp(lambda) * x2 * t ^ exp(gamma))
return(gg)
}
```

Let's do some quality check on our gradient functions. For this we compare the estimates of our gradient functions with the approximation from the `numDeriv::numgrad()` function.

```
# 4.1 Compare gradient functiona with numeric approximation of gradient
=======
# compare gradient at 1, 0, 0, 0
mygrad <- negll.grad(c(1, 0, 0, 0))
numgrad <- grad(x = c(1, 0, 0, 0), func = negll)
all.equal(mygrad, numgrad)
## [1] TRUE
```

All good, we get the same results. Now is the time to get all functions and data together and pass them to our `optimx()` call to get the maximum likelihood estimates for our Weibull model.

```
# 5. Find minimum of log-likelihood function
----------------------------------
# Passing names to the values in the par vector improves readability of
results
opt <- optimx(par = c(gamma = 1, alpha = 0, beta_female = 0, beta_age =
0),
fn = negll,
gr = negll.grad,
hessian = TRUE,
control = list(trace = 0, all.methods = TRUE))
# Show results for optimisation alogrithms, that convergered (convcode
== 0)
summary(opt, order = "value") %>%
rownames_to_column("algorithm") %>%
filter(convcode == 0) %>%
select(algorithm, gamma, alpha, beta_female, beta_age, value)
## algorithm gamma alpha beta_female beta_age value
## 1 newuoa 0.282294079 -8.828209 -0.5067097 0.01625468 1147.054
## 2 nlminb 0.282296689 -8.828269 -0.5067118 0.01625527 1147.054
## 3 BFGS 0.282291577 -8.828211 -0.5067102 0.01625510 1147.054
## 4 bobyqa 0.282291455 -8.828164 -0.5067134 0.01625432 1147.054
## 5 L-BFGS-B 0.282257770 -8.828116 -0.5065362 0.01625512 1147.054
## 6 Nelder-Mead 0.004418373 0.702934 -0.4134191 -0.10870858 1271.990
## 7 nlm -23.931210572 -1.433331 -0.6337426 -88.76207627 931946.566
## 8 Rcgmin 1.000000000 0.000000 0.0000000 0.00000000 6629012.776
```

According to our table the `newuoa` algorithm from the `{minqa}` package yielded the best estimates. The `newuoa` algorithm was developed to find the minimum of a function without information about the analytical gradient function. Instead the algorithm uses a quadratic approximation of the gradient function to minimise the function of interest. Interestingly, the `newuoa` algorithm yielded a higher likelihood than the `nlminb` algorithm that uses the analytical gradient function.

Let's now compare our results with the results from the `flexsurvreg()` function from the `{flexsurv}` package.

```
# 6. Estimate regression coeficents using flexsurvreg
-------------------------
weibull_model <- flexsurvreg(Surv(time, status_n == 1) ~ female + age,
data = lung,
dist = "weibullph")
# 7. Comparing results from optimx and flexsurvreg
----------------------------
weibull_results <- unname(coef(weibull_model))
coef_opt <- coef(opt)
lapply(1:nrow(coef_opt), function(i){
opt_name <- attributes(coef_opt)$dimnames[[1]][i]
mle_weibl1 <- (coef_opt[i, 1] - weibull_results[1])
mle_weibl2 <- (coef_opt[i, 2] - weibull_results[2])
mle_weibl3 <- (coef_opt[i, 3] - weibull_results[3])
mle_weibl4 <- (coef_opt[i, 4] - weibull_results[4])
mean_dif <- mean(mle_weibl1, mle_weibl2, mle_weibl3, mle_weibl4,
na.rm = TRUE)
data.frame(opt_name, mean_dif)
}) %>%
bind_rows() %>%
filter(!is.na(mean_dif)) %>%
mutate(mean_dif = abs(mean_dif)) %>%
arrange(mean_dif)
## opt_name mean_dif
## 1 newuoa 1.264708e-06
## 2 nlminb 1.345317e-06
## 3 BFGS 3.766647e-06
## 4 bobyqa 3.888642e-06
## 5 L-BFGS-B 3.757310e-05
## 6 Nelder-Mead 2.778770e-01
## 7 Rcgmin 7.177047e-01
## 8 Rvmmin 7.177047e-01
## 9 CG 2.864370e+00
## 10 nlm 2.421351e+01
```

We can see that the differences between our estimates and the estimates we would have gotten if we used `flexsurvreg()` to fit our model, are close to null. At least for the estimates yielded by the `newuoa` algorithm.

Since we found the point estimates for our Weibull regression, we can now take the next step and calculate confidence intervals (CIs) for our estimates. For this we will use the Hessian matrix of our model. If you cannot follow the code below, please take a look at my previouse

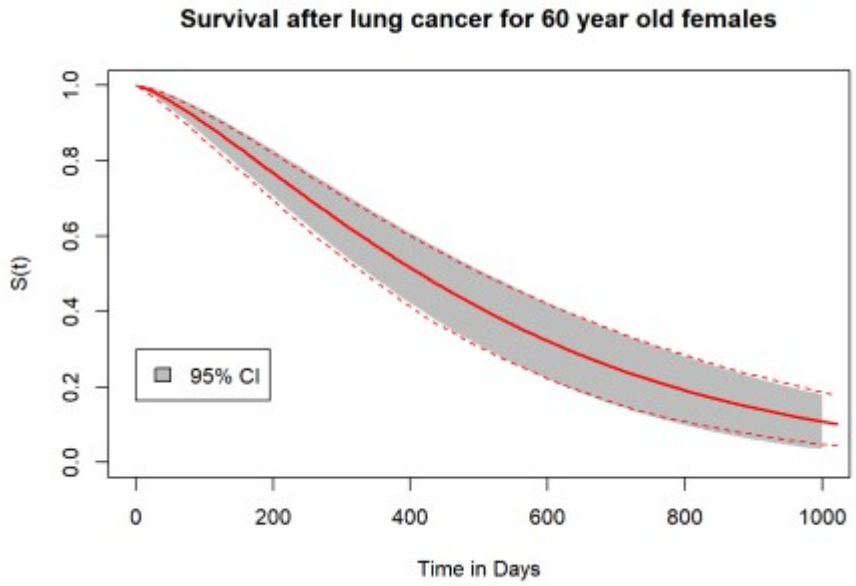
post where I explained how to compute CIs for estimates of a logistic regression model using the same approach.

```
# 8. Estimate the standard error ------------------------------
---------------
#Extract hessian matrix for Newuoa optimisation
hessian_m <- attributes(opt)$details["newuoa", "nhatend"][[1]]
# Estimate se based on hession matrix
fisher_info <- solve(hessian_m)
prop_se <- sqrt(diag(fisher_info))
# Compare the estimated se from our model with the one from the
flexsurv model
# Note use res.t to get the estimates on the reale scale without
transformaiton
ses <- data.frame(se_newuoa = prop_se,
se_felxsurvreg = weibull_model$res.t[, "se"]) %>%
print()
## se_newuoa se_felxsurvreg
## shape 0.061883266 0.061869493
## scale 0.777802040 0.777206348
## female 0.167066129 0.167066105
## age 0.009188029 0.009170499
all.equal(ses[,"se_newuoa"], ses[, "se_felxsurvreg"])
## [1] "Mean relative difference: 0.0006171813"
# 9. Estimate 95%CIs using estimation of SE
-----------------------------------
# Extracting estimates from Newuoa optimisaiton
coef_test <- coef(opt)["newuoa",]
# Compute 95%CIs
upper <- coef_test + 1.96 * prop_se
lower <- coef_test - 1.96 * prop_se
# Print 95%CIs
data.frame(Estimate = coef_test,
CI_lower = lower,
CI_upper = upper,
se = prop_se)
## Estimate CI_lower CI_upper se
## gamma 0.28229408 0.161002878 0.40358528 0.061883266
## alpha -8.82820947 -10.352701468 -7.30371747 0.777802040
## beta_female -0.50670974 -0.834159350 -0.17926012 0.167066129
## beta_age 0.01625468 -0.001753855 0.03426322 0.009188029
```

The best way to understand your survival model is plotting its basic functions. So let's take a look at the survival function (\(S(t)\)) of our model.

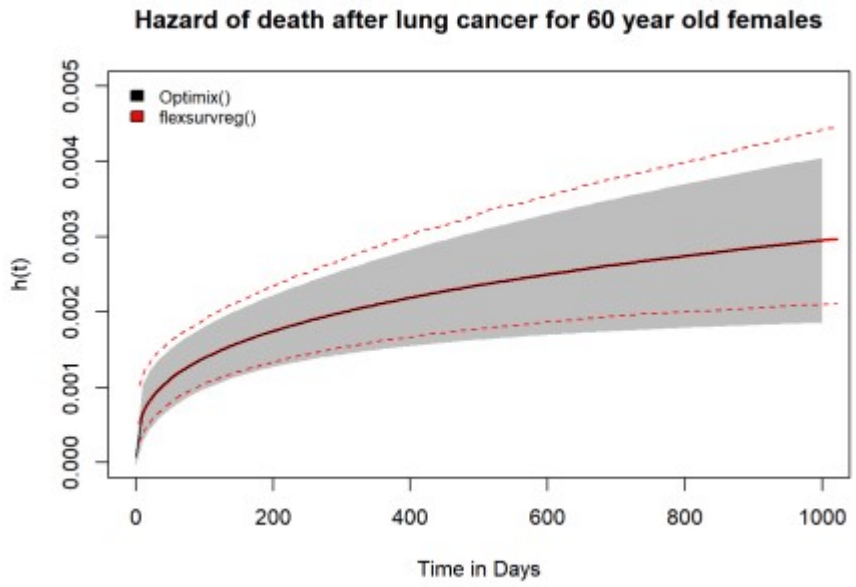
```
# 10. Plot survival curve with 95% CI ------------------------------
-----------
# 10.1 Use Delta Method to compute CIs across time of follow-up
===============
# Get coefficents for Newuoa optimisation
newuoa_coef <- coef(opt)["newuoa", ]
# Compute CIs for a 60 year of female across time
surv_optim_female <- lapply(as.list(seq(0.01, 1000.01, 10)),
```

```
function(t){
g <- paste("exp(-exp(alpha + beta_female + 60 * beta_age) *", t,
"^ exp(gamma))")
fit <- deltaMethod(newuoa_coef, g, solve(hessian_m))
data.frame(time = t,
estimate = fit[, "Estimate"],
ci_low = fit[, "2.5 %"],
ci_up = fit[, "97.5 %"])
}) %>%
bind_rows()
# 10.2 Plot survival curve with CIs ==============================
=============
plot(surv_optim_female$time,
surv_optim_female$estimate,
ylim = c(0, 1),
type = "n",
xlab = "Time in Days",
ylab = "S(t)",
main = "Survival after lung cancer for 60 year old females")
polygon(c(surv_optim_female$time, rev(surv_optim_female$time)),
c(surv_optim_female$ci_low, rev(surv_optim_female$ci_up)),
border = NA,
col = "grey")
lines(surv_optim_female$time,
surv_optim_female$estimate)
plot(weibull_model, type = "survival",
newdata = data.frame(age = 60,
female = 1),
add = TRUE)
legend(0, 0.3,
fill = "grey",
"95% CI")
```



Additionally, we can also plot our hazard function (\(h(t)\)).

```
# 11. Plot hazard curve with 95% CI ------------------------------
-------------
# 10.1 Use Delta Method to compute CIs across time of follow-up
===============
# Get coefficents for Newuoa optimisation
newuoa_coef <- coef(opt)["newuoa", ]
# Compute CIs for a 60 year of female across time
haz_optim_female <- lapply(as.list(seq(0.01, 1000.01, 10)),
function(t){
g <- paste("exp(gamma) * exp(alpha + beta_female + 60 * beta_age) *",
t,
"^ (exp(gamma) - 1)")
fit <- deltaMethod(newuoa_coef, g, solve(hessian_m))
data.frame(time = t,
estimate = fit[, "Estimate"],
ci_low = fit[, "2.5 %"],
ci_up = fit[, "97.5 %"])
}) %>%
bind_rows()
# 10.2 Plot hazard curve with CIs ==============================
===============
plot(haz_optim_female$time,
haz_optim_female$estimate,
ylim = c(0, 0.005),
type = "n",
xlab = "Time in Days",
ylab = "h(t)",
main = "Hazard of death after lung cancer for 60 year old females")
polygon(c(haz_optim_female$time, rev(haz_optim_female$time)),
c(haz_optim_female$ci_low, rev(haz_optim_female$ci_up)),
border = NA,
col = "grey")
plot(weibull_model, type = "hazard",
newdata = data.frame(age = 60,
female = 1),
add = TRUE)
lines(haz_optim_female$time,
haz_optim_female$estimate)
legend("topleft",
inset = 0.01,
cex = 0.8,
fill = c("black", "red"),
legend = c("Optimix()", "Flexregsurv()"),
box.lty = 0)
```



Interestingly, we see quite some differences between our estimates for the CI and `flexregsurv()`'s estimates. Unfortunately, I didn't find a reason for this difference yet. So if you have a guess, please let me know.