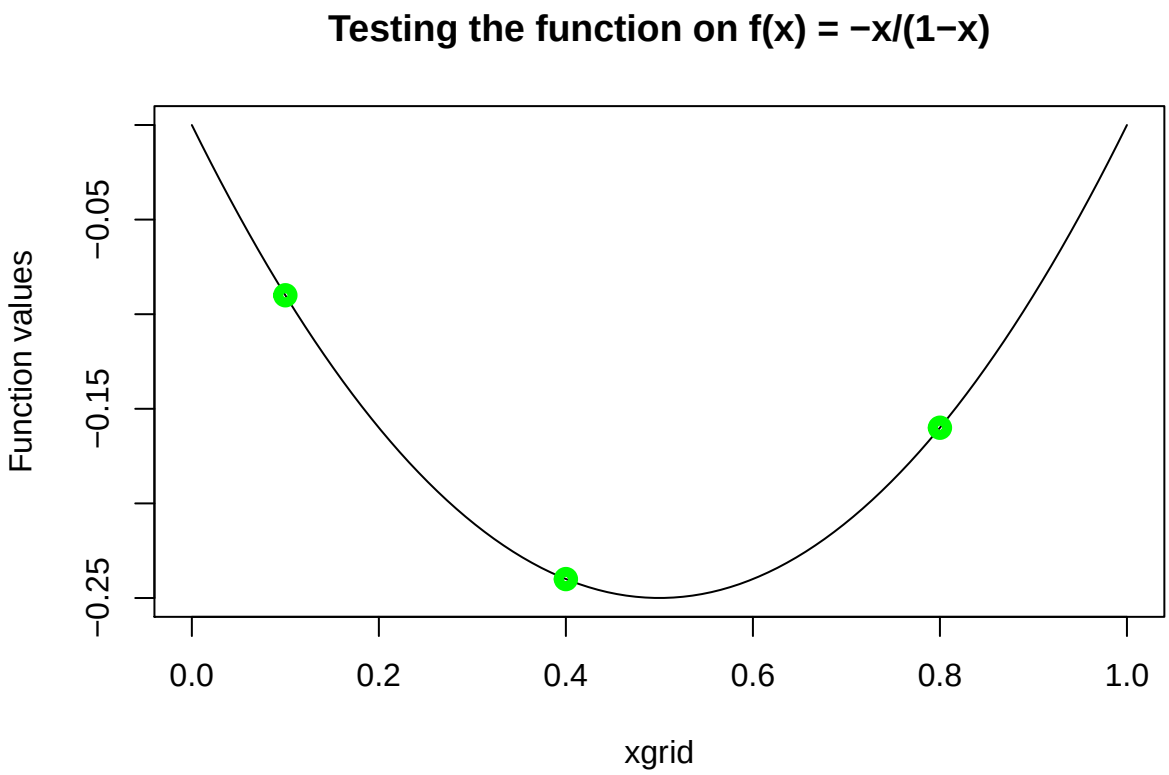
732A90: Lab 2

David Björelind, davbj395

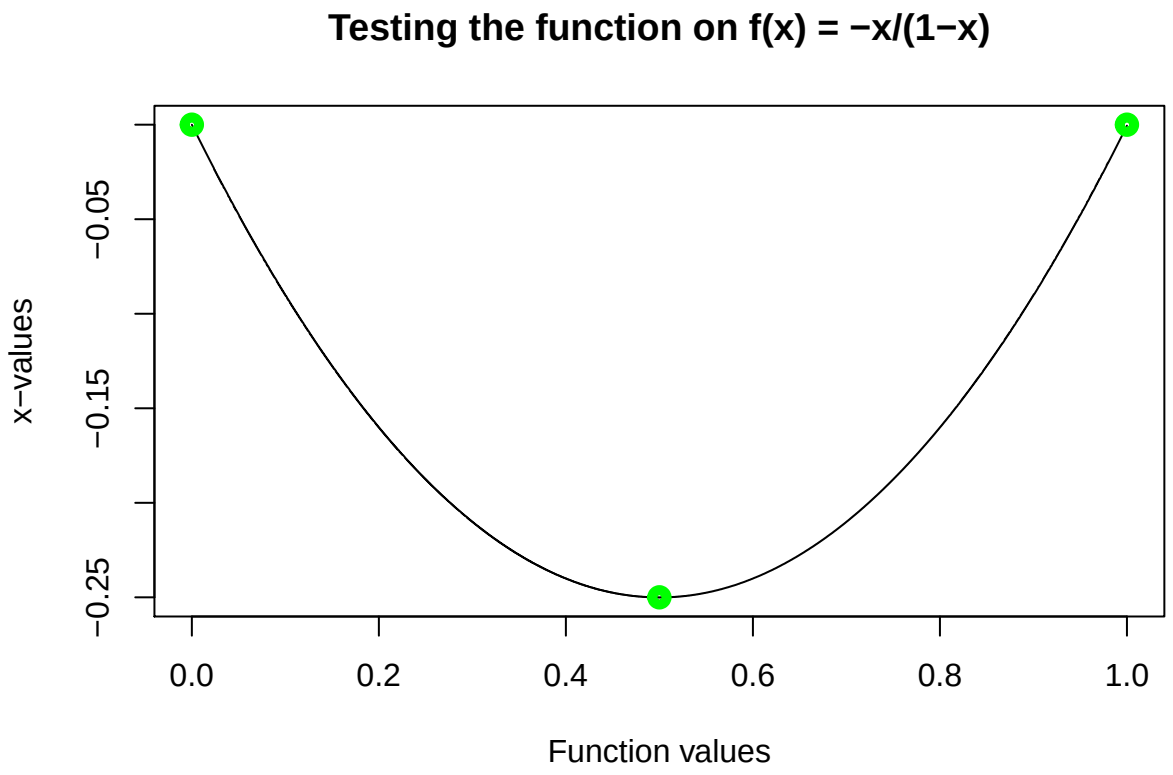
24/11/2020

Question 1: Optimizing parameters

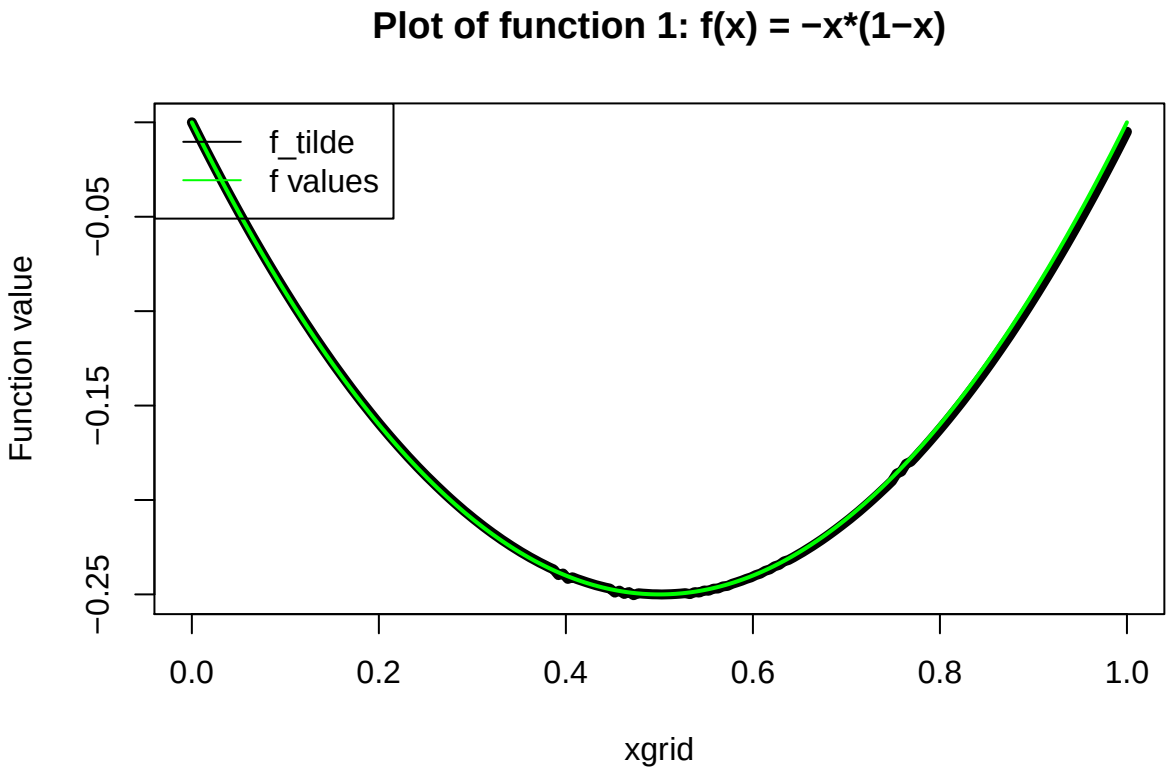
1:

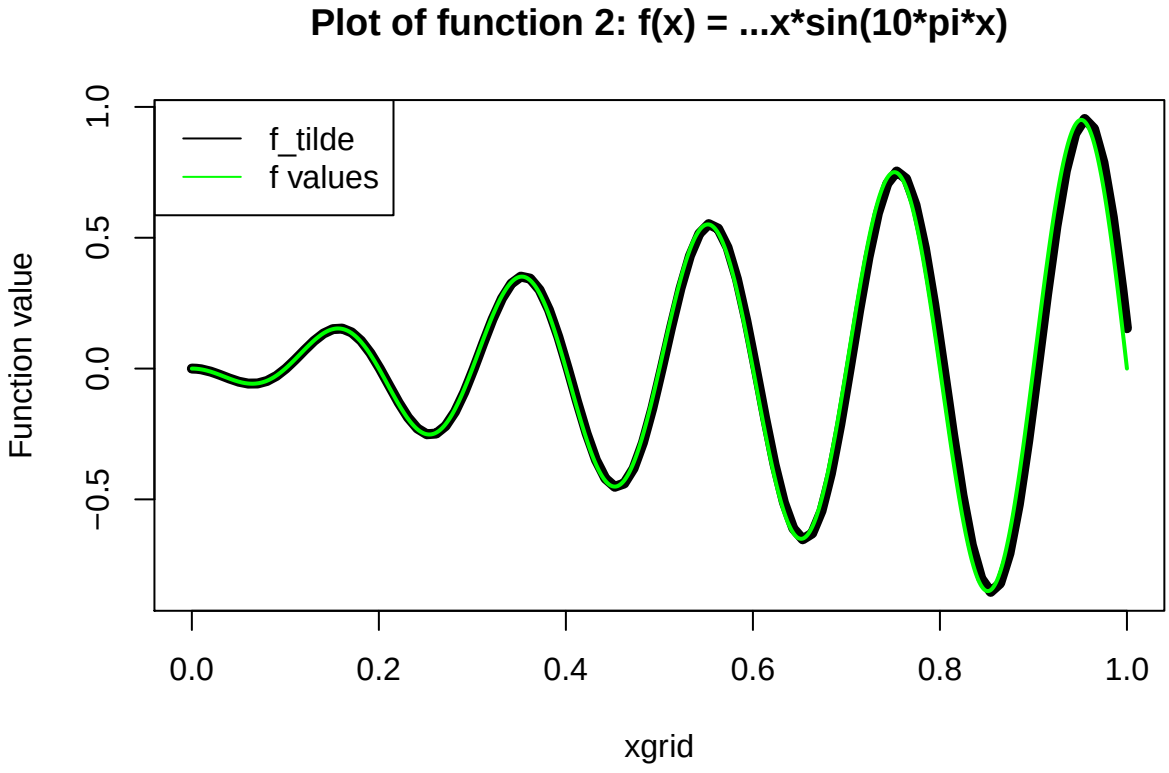


2:


Testing the function on f(x) = −x/(1−x)
x−values
−0.05
−0.15
−0.25
0.0
0.2
0.4
0.6
0.8
1.0
Function values

3:


Plot of function 1: f(x) = −x*(1−x)
f_tilde
f values
Function value
−0.25
−0.15
−0.05
0.0
0.2
0.4
0.6
0.8
1.0
xgrid


Plot of function 2: f(x) = ...x*sin(10*pi*x)
f_tilde
f values
Function value
−0.5
0.0
0.5
1.0
0.0
0.2
0.4
0.6
0.8
1.0
xgrid

Both functions get a very good fit, the approximated functions basically fits on top of the 'real' functions. This is probably due to the large amount of intervals used. At the extreme turns of the second function, the interpolated function is a little bit less smooth. When using a smaller amount of intervals, the functions are not fitted so well. Especially the second one. This is due to it being less smooth with more variations for the function values.

Question 2: Maximizing likelihood

The PDF of a normal distribution is $\frac{1}{\sigma\sqrt{2\pi}} * e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$. The Likelihood then becomes

$L = (\frac{1}{2\pi\sigma^2})^{n/2} * exp(\frac{-\sum_{i=1}^{n}(x_i-\mu)^2}{2\sigma^2})$ after multiplying the PDF over i=1 to n.

The log-likehood is then $-\frac{n}{2}log(\frac{1}{2\pi\sigma^2}) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\mu)^2$ after taking the log of the likelihood. for this expression, we want to find how to pick μ and σ so that we maximize it. The derivative of the log-likelihood is calculated with regards to μ and σ and set $= 0$ to find the optimal values.

$\frac{dL}{d\mu} = \frac{2n(\bar{x}-\mu)}{2\sigma^2} = 0$

$\frac{dL}{d\sigma} = -\frac{n}{\sigma} + \frac{1}{\sigma^3}\sum_{i=1}^{n}(\bar{x}-\mu)^2 = 0$

This produces an MLE estimator for μ which is $\hat{\mu} = \bar{x}$, the sample mean. The MLE estimator for σ is $\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i-\bar{x})^2$.

2:

```
## MLE mu:  1.275528
```

```
## MLE sigma:  2.005976
```

```
## log-likelihood:  -211.5069
```

3:

Bad idea to try to maximize likelihood! The log function is monotonically increasing, which makes the maximum values remain at the same x-values is in the original function. This is essential when maximizing. We apply it to the likelihood function because it makes it way easier to calculate the MLE estimators for the parameters. The log-likehood function will also become more smooth, which is helpful when trying to find optimal values using a solver.

4:

Here follows a table with optimal parameters, best log-likelihood value, and iteration counts for all the cases:

```
##                          mu    sigma log-likelihood count: fn count: gr conv
## CG                 1.275528 2.005977      -211.5069       274        43    0
## CG with gradients  1.275528 2.005976      -211.5069        53        17    0
## BFGS               1.275528 2.005977      -211.5069        37        15    0
## BFGS with gradient 1.275528 2.005977      -211.5069        38        15    0
```

Here follows a table with gradients for all the cases:

```
##                       mu gradient sigma gradient
## CG                  -2.907985e-06  -1.966048e-06
## CG with gradients    2.463027e-07  -1.927738e-08
## BFGS                 1.902664e-06  -2.438016e-05
## BFGS with gradient   1.366082e-06  -3.826563e-06
```

From the results we can that in all of the cases, the algorithm converged (given **0** as convergence value). This is further confirmed by seeing that the gradients of optimal parameters are (close to) zero. When using CG we get a great advantage from using gradients, whereas when using BFGS there is no advantage in using gradients. This means that the "finite-difference approximation" does the job as well as the gradient does at finding new points.

The optimal parameters were also found by the maximum likelihood estimators using their analytical expressions!

The recommended setting would be **BFGS** without gradient. It gets the same number of iterations as with gradient, making it simple to use.

Include all code for this report

```
knitr::opts_chunk$set(echo = TRUE, warning=FALSE, message=FALSE)
# Include packages here
xgrid = seq(0,1, length.out = 100)

f_tilde = function(a, x){
  res = a[1] + a[2]*x + a[3]*x^2
  return(res)
}

squared_error = function(a, param){
  f_vals = f_tilde(a, param[1:3])
  error = (param[4]-f_vals[1])^2 + (param[5]-f_vals[2])^2 + (param[6]-f_vals[3])^2
  return(error)
}

interpolate = function(x_start, func){
  a = c(0,0,0)
  param = c(x_start, func(x_start)) # contains x_vals and y_vals
  opti = optim(par = a, fn = squared_error, param = param)
  return(opti)
}

# Testing the created functions
x_start = c(0.1, 0.4, 0.8)
test_func = function(x){
  return(-x*(1-x))
}
opti = interpolate(x_start, test_func)
plot(x = xgrid, y = f_tilde(opti$par, xgrid), main = "Testing the function on f(x) = -x/(1-x)", ylab ="
lines(x = x_start, y = test_func(x_start), col = 'green', type='p', lwd = '5')
approx_func = function(nint, func){
  xgrid = seq(0,1, length.out = nint)
```

```
## SECOND APPROACH ##
  res = c()
  for (i in 1:nint){
    start = 0+1/nint*(i-1)
    end = start+1/nint
    #cat("start: ", start, " end: ", end, "\n")
    opti = interpolate(c(start, (start+end)/2, end), func)
    val = f_tilde(opti$par, c(start, (start+end)/2))
    res = c(res, val)
  }
  return(res)

## FIRST APPROACH ##
#  opti = interpolate(c(0, 0.5, 1), func)
#  res = f_tilde(opti$par, xgrid)
#  return(res)
}

test = approx_func(1000, test_func)
plot(x = seq(0,1,length.out = length(test)), y = test, type = 'l', xlab = "Function values", ylab ="x-va
lines(x = c(0, 0.5, 1), y = test_func(c(0, 0.5, 1)), col = 'green', type='p', lwd = '5')
f_1 = function(x){
  return(-x*(1-x))
}
f_2 = function(x){
  return(-x*sin(10*pi*x))
}
n_intervals = 100

large_grid = seq(0,1,length.out = 1000)
f_tilde_1 = approx_func(n_intervals, f_1)
f_tilde_2 = approx_func(n_intervals, f_2)
xgrid = seq(0,1,length.out = length(f_tilde_1))
# Plot of f1
plot(x = xgrid, y = f_tilde_1, type = 'l', lwd=5, main="Plot of function 1: f(x) = -x*(1-x)", ylab = "Fu
lines(x = large_grid, y = f_1(large_grid), col='green', lwd=2)
legend("topleft", c("f_tilde","f values"),
       col=c("black", "green"), lty=1, cex=1)

# Plot of f2
plot(x = xgrid, y = f_tilde_2, type = 'l', lwd=5, main="Plot of function 2: f(x) = -x*sin(10*pi*x)", yla
lines(x = large_grid, y = f_2(large_grid), col='green', lwd=2)
legend("topleft", c("f_tilde","f values"),
       col=c("black", "green"), lty=1, cex=1)

load(file = 'data.rdata')

loglike = function(x, mu, sigma){
  log = sum(dnorm(x, mean = mu, sd = sigma, log = TRUE))
  return(log)
}

mle_mu = function(x){
```

```
  return(mean(x))
}
mle_sigma = function(x){
  mean = mean(x)
  a = (x-mean)^2
  res = sum(a)/length(x)
  return(sqrt(res))
}

cat("MLE mu: ", mle_mu(data), "\n")
cat("MLE sigma: ", mle_sigma(data), "\n")
cat("log-likelihood: ", loglike(data, mle_mu(data), mle_sigma(data)), "\n")
grads = function(par, x){
  n = length(x)
  mean = mean(x)
  grad_mu = n*(mean-par[1])/par[2]^2
  a = (x-par[1])^2
  grad_sigma = sum(a)/par[2]^3 - n/par[2]
  return(c(grad_mu, grad_sigma))
}

loglike = function(par, x){
  log = sum(dnorm(x, mean = par[1], sd = par[2], log = TRUE))
  return(log)
}

# Matrix to store the answers
results = matrix(nrow = 4, ncol = 6)
rownames(results) = c("CG", "CG with gradients", "BFGS", "BFGS with gradient")
colnames(results) = c("mu", "sigma", "log-likelihood", "count: fn", "count: gr", "conv")

# Using Conjugate Gradient Method
init = c(0,1)
opti_CG = optim(par = init, fn = loglike, x=data, method=c("CG"), control=list(fnscale=-1))
results[1,] = c(opti_CG$par, opti_CG$value, opti_CG$counts, opti_CG$convergence)

# Using CG with gradients
opti_CG_g = optim(par = init, fn = loglike, gr = grads , x=data, method=c("CG"), control=list(fnscale=-
results[2,] = c(opti_CG_g$par, opti_CG_g$value, opti_CG_g$counts, opti_CG_g$convergence)

# Using BFGS
opti_BFGS = optim(par = init, fn = loglike, x=data, method=c("BFGS"), control=list(fnscale=-1))
results[3,] = c(opti_BFGS$par, opti_BFGS$value, opti_BFGS$counts, opti_BFGS$convergence)

# Using BFGS with gradients
opti_BFGS_g = optim(par = init, fn = loglike, gr = grads, x=data, method=c("BFGS"), control=list(fnscal
results[4,] = c(opti_BFGS_g$par, opti_BFGS_g$value, opti_BFGS_g$counts, opti_BFGS_g$convergence)

print(results)
cat("\n")
gradients = matrix(nrow = 4, ncol = 2)
rownames(gradients) = c("CG", "CG with gradients", "BFGS", "BFGS with gradient")
colnames(gradients) = c("mu gradient", "sigma gradient")
```

```
for (i in 1:4){
  gradients[i,] = grads(results[i,1:2], data)
}
print(gradients)
```