Exam 2018-08-27

Maximilian Pfundstein

2019-03-12

Contents

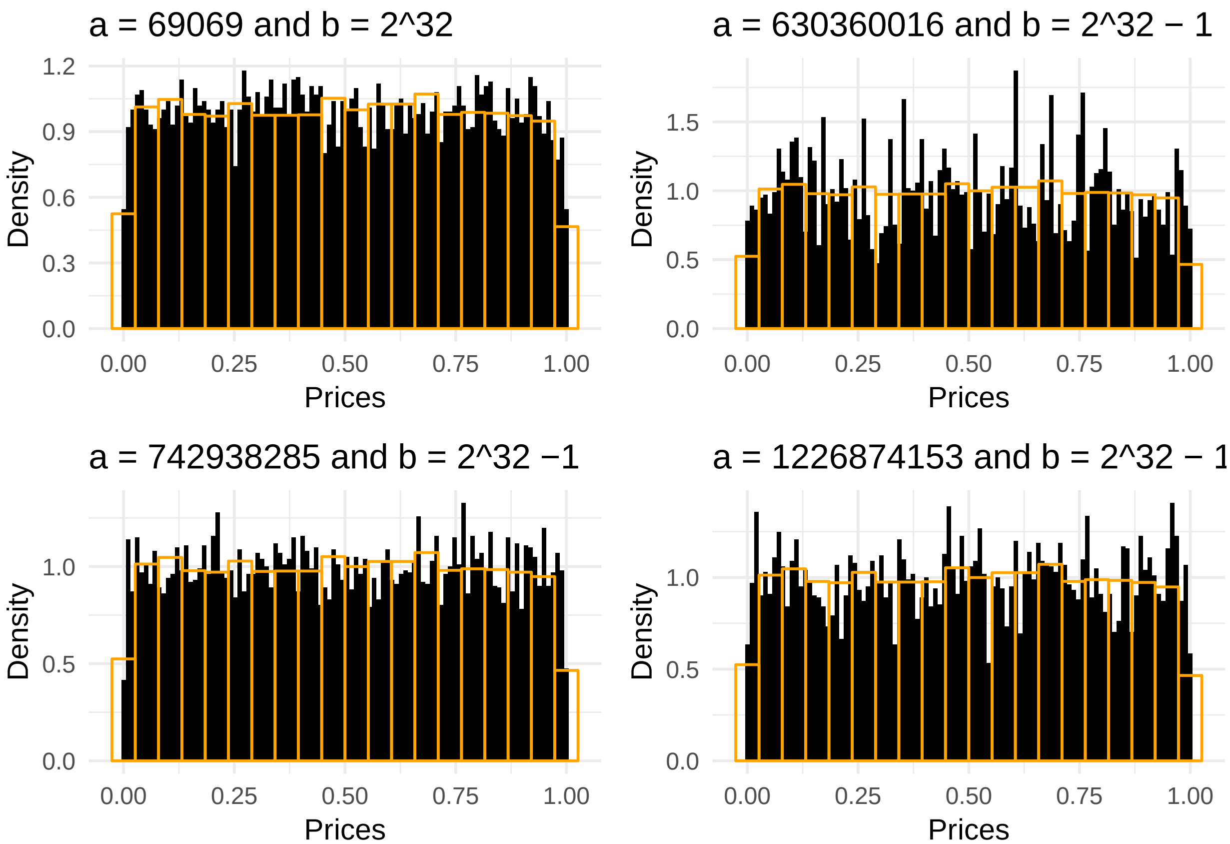
1 Assignment 1 (3 points)

1.1 Question 1.1

```
rng_runif = function(a, m, x_zero, nmax) {

  if (!(a >= 0 && a < m)) stop("a in [0, m) is required")
  if (nmax%%1 != 0) stop("nmax has to be an integer")

  storage = vector(mode = "numeric", length = nmax+1)
  storage[1] = x_zero

  for (i in 1:(nmax-1)) {
    storage[i+1] = ((a * storage[i]) %% m)
  }

  return(storage[2:length(storage)]/m)
}
```

1.2 Question 1.2

```
prs_1 = rng_runif(69069, 2^32, 9999, 10000)
prs_2 = rng_runif(630360016, (2^32-1), 690690, 10000)
prs_3 = rng_runif(742938285, (2^32-1), 690690, 10000)
prs_4 = rng_runif(1226874153, (2^32-1), 690690, 10000)

unif_samples = runif(10000)
```




```r
print(ks.test(x = prs_1, y = "punif"))
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  prs_1
## D = 0.010645, p-value = 0.2072
## alternative hypothesis: two-sided
```

```r
print(ks.test(x = prs_2, y = "punif"))
```

```
## Warning in ks.test(x = prs_2, y = "punif"): ties should not be present for
## the Kolmogorov-Smirnov test

##
##  One-sample Kolmogorov-Smirnov test
##
## data:  prs_2
## D = 0.018419, p-value = 0.002262
## alternative hypothesis: two-sided
```

```r
print(ks.test(x = prs_3, y = "punif"))
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  prs_3
## D = 0.0063916, p-value = 0.8086
## alternative hypothesis: two-sided
```

```r
print(ks.test(x = prs_4, y = "punif"))
```

```
## Warning in ks.test(x = prs_4, y = "punif"): ties should not be present for
## the Kolmogorov-Smirnov test

##
##  One-sample Kolmogorov-Smirnov test
##
## data:  prs_4
## D = 0.013235, p-value = 0.06018
## alternative hypothesis: two-sided
```

```r
print(ks.test(x = unif_samples, y = "punif"))
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  unif_samples
## D = 0.0093707, p-value = 0.3436
## alternative hypothesis: two-sided
```

Answer: We see that the second and the last settings have a low-p value and have to be rejected at a significance level of $\alpha = 10\%$

- `a = 1` mod `p` for every prime divisor `p` of `m`
- `a = 1` mod `4` if `4` divides `m`
- `c` and m have to be relatively prime (no common divisirs bar 2)

In our case `c = 1`.

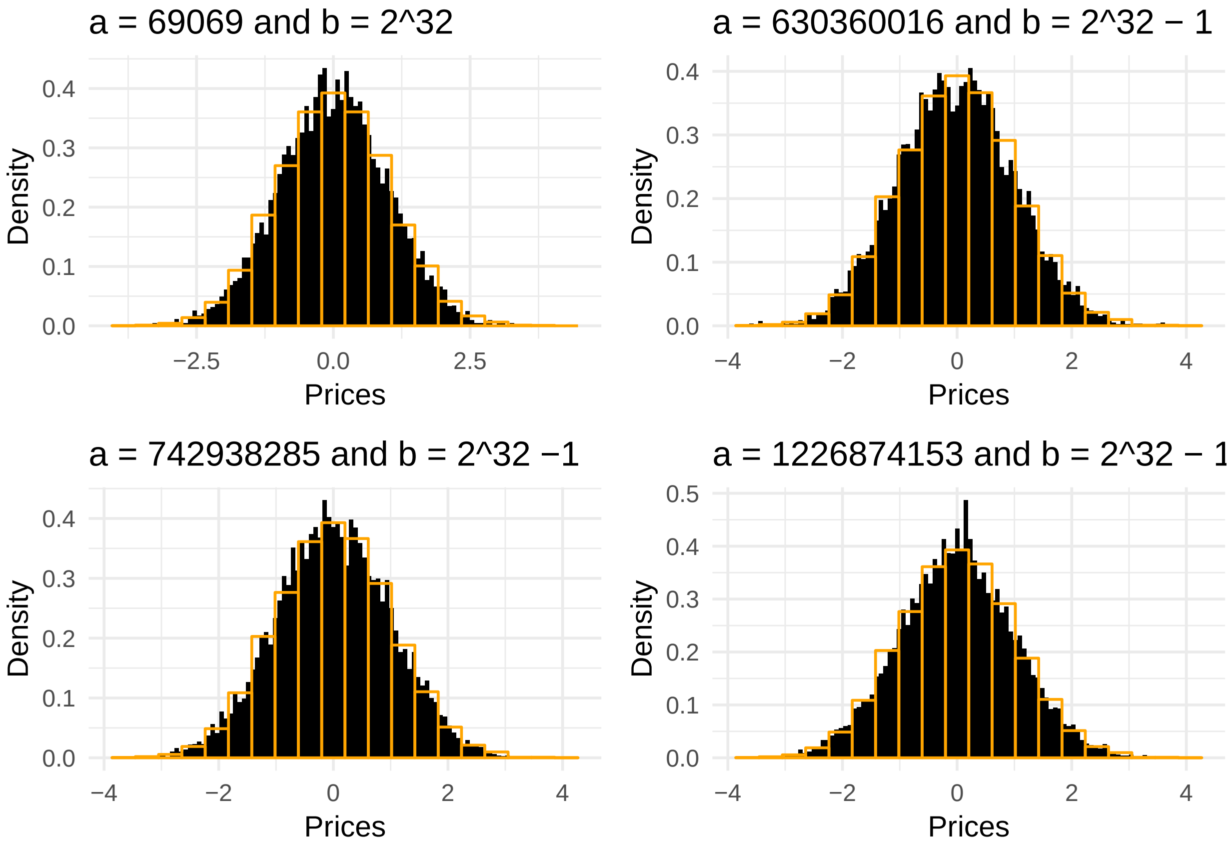
1.3 Question 1.3

```r
rng_rnorm = function(n, a, m) {
  if (!(a >= 0 && a < m)) stop("a in [0, m) is required")

  n_half = ceiling(n/2)

  storage = vector(mode = "numeric", length = n)

  storage_theta = rng_runif(a = a, m = m, x_zero = 123456789, nmax = n_half) * 2 * pi
  storage_d = rng_runif(a = a, m = m, x_zero = 12345, nmax = n_half)

  for (i in 1:n_half) {
    storage[2*i] = sqrt(-2 * log(storage_d[i])) * cos(storage_theta[i])
    storage[2*i+1] = sqrt(-2 * log(storage_d[i])) * sin(storage_theta[i])
  }

  return(storage)
}
```

```r
rng_normal_1 = rng_rnorm(10000, a = 69069, m =2^32)
rng_normal_2 = rng_rnorm(10000, a = 630360016, m = 2^32-1)
rng_normal_3 = rng_rnorm(10000, a = 742938285, m = 2^32-1)
rng_normal_4 = rng_rnorm(10000, a = 1226874153, m = 2^32-1)
```

```r
norm_samples = rnorm(10000)
```

```
## Warning: Removed 2 rows containing non-finite values (stat_bin).

## Warning: Removed 2 rows containing non-finite values (stat_bin).

## Warning: Removed 2 rows containing non-finite values (stat_bin).

## Warning: Removed 2 rows containing non-finite values (stat_bin).
```




```r
print(ks.test(x = rng_normal_1, y = "pnorm"))
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  rng_normal_1
## D = 0.0047014, p-value = 0.9799
## alternative hypothesis: two-sided
```

```r
print(ks.test(x = rng_normal_2, y = "pnorm"))
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  rng_normal_2
## D = 0.015241, p-value = 0.01921
## alternative hypothesis: two-sided
```

```r
print(ks.test(x = rng_normal_3, y = "pnorm"))
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  rng_normal_3
## D = 0.006856, p-value = 0.7351
## alternative hypothesis: two-sided
```

```r
print(ks.test(x = rng_normal_4, y = "pnorm"))
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  rng_normal_4
## D = 0.0094938, p-value = 0.3283
## alternative hypothesis: two-sided
```

```r
print(ks.test(x = norm_samples, y = "pnorm"))
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  norm_samples
## D = 0.0059442, p-value = 0.8716
## alternative hypothesis: two-sided
```

2 Assignment 2

2.1 Question 2.1

```r
interpolate = function(A, X, func, gradient = NULL) {

  # Interpolation function which is so be used
  f_tilde = function(x, a0, a1, a2) a0 + a1 * x + a2 * x^2

  # Error function, here MSE
  f_error = function(A., X. = X, func. = func) {
    return(sum((func.(X.) - f_tilde(X., A.[1], A.[2], A.[3]))^2))
  }

  # Optimize f_error for parameters in A
  res = optim(A, f_error, gradient, method = "CG")

  # Now define with optimized parameters
  f_tilde = function(x) res$par[1] + res$par[2] * x + res$par[3] * x^2

  # Return parameters and function
  return(list(A = res$par, f_tilde = f_tilde))
}
```

2.2 Question 2.2

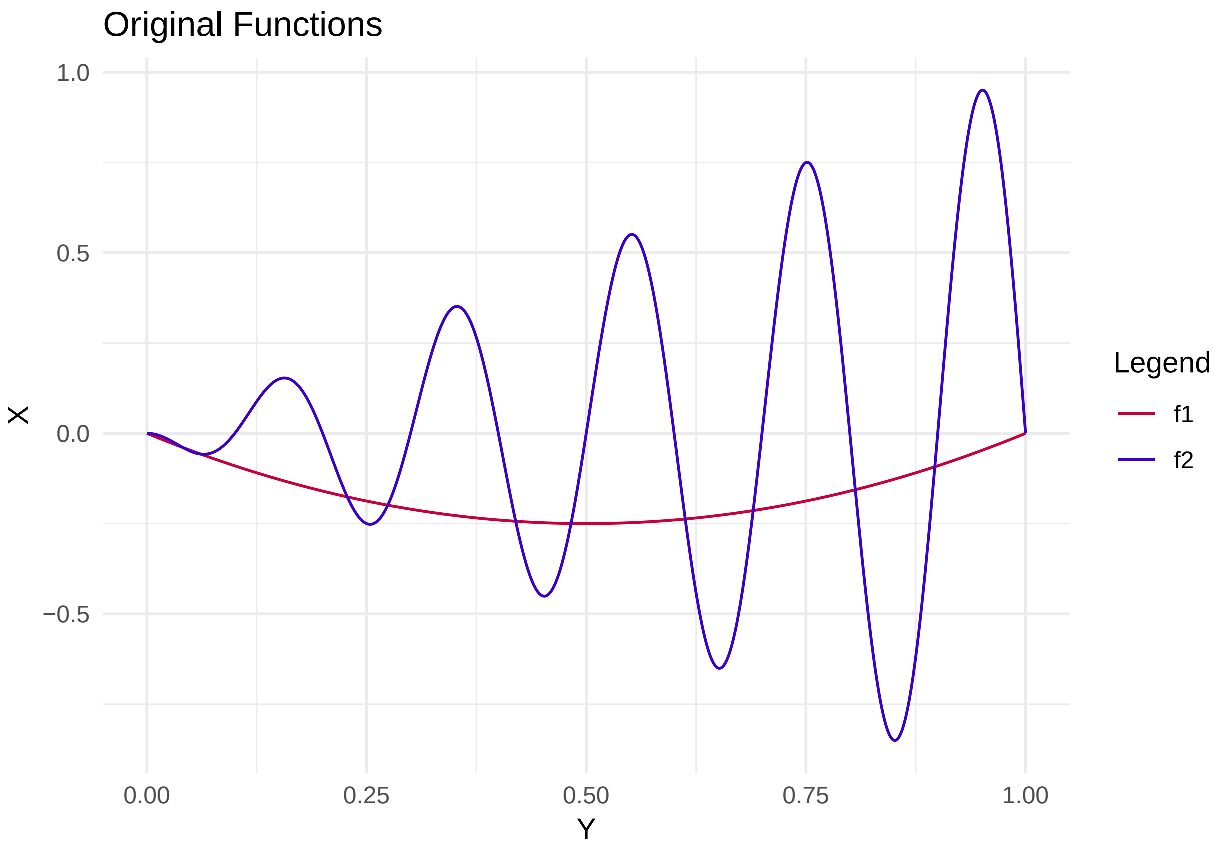
```
f_approximate = function(func_target, bins, A_init, func_target_gradient = NULL) {

  # Initialize interval length and return matrix
  upper_boundary = 0
  interval_length = 1/bins
  res = matrix(0, nrow = bins, ncol = 5)
  colnames(res) = c("lower_boundary", "upper_boundary", "a0", "a1", "a2")

  # Approximate for each bin
  for (i in 1:bins) {
    lower_boundary = upper_boundary
    upper_boundary = lower_boundary + interval_length

    # We rely on three known points
    known_values = c(func_target(lower_boundary),
                     func_target((lower_boundary + upper_boundary)/2),
                     func_target(upper_boundary))

    # Now we optimize for this interval using the previous function
    interpolated = interpolate(func = func_target, X = known_values, A = A_init,
                               gradient = func_target_gradient)$A

    # And fill in the matrix with all necesarry values
    res[i, 1] = lower_boundary
    res[i, 2] = upper_boundary
    res[i, 3:5] = interpolated
  }

  return(res)
}
```

2.3 Question 2.3

Definition of the original functions.

```
f1 = function(x) -x * (1 - x)
f2 = function(x) -x * sin(10 * pi * x)
```

The look like the following.



Now we perform the interpolation.

```
f1_res = f_approximate(f1, bins = 1000, rep(0.001, 3))
head(f1_res)
```

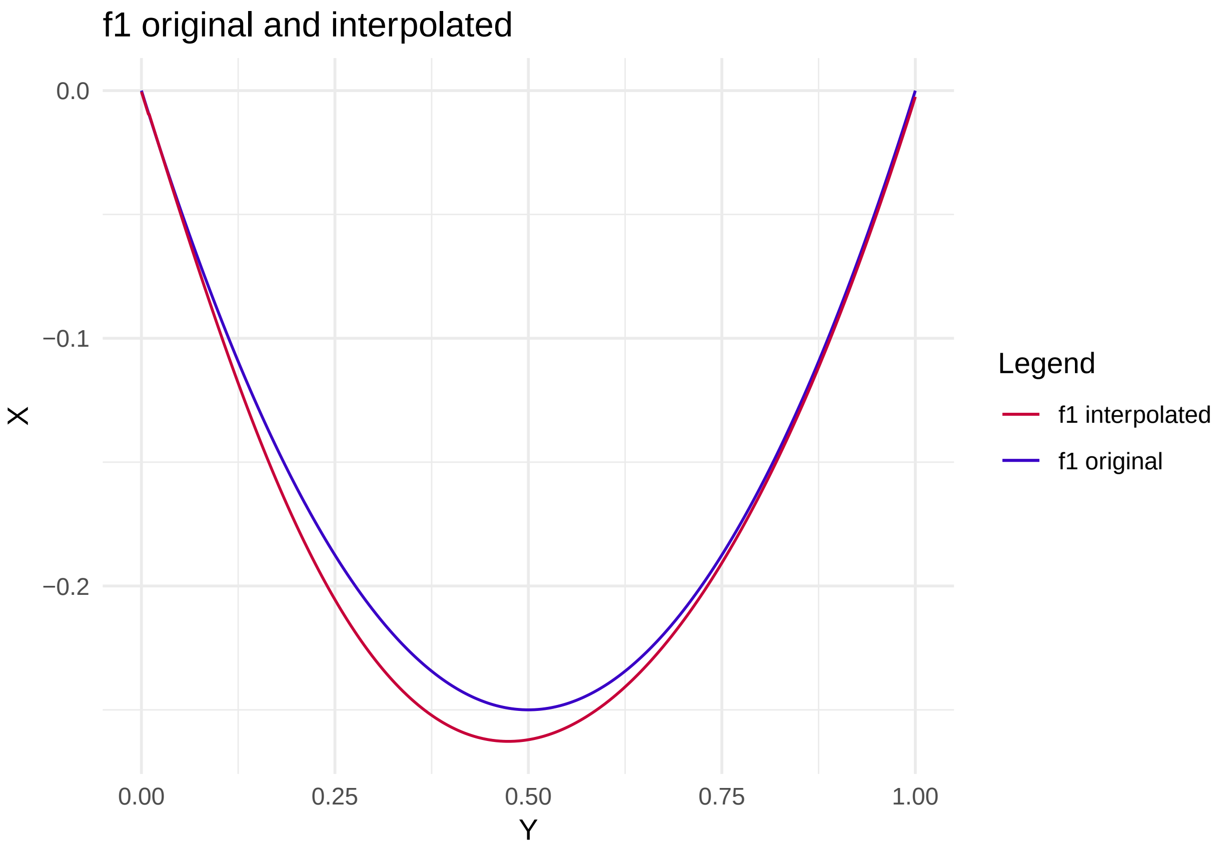
```
##      lower_boundary upper_boundary           a0           a1          a2
## [1,]          0.000          0.001 0.0005006436 0.0009993759 0.001000001
## [2,]          0.001          0.002 0.0015013388 0.0009983774 0.001000004
## [3,]          0.002          0.003 0.0025023634 0.0009952641 0.001000015
## [4,]          0.003          0.004 0.0035032151 0.0009902827 0.001000038
## [5,]          0.004          0.005 0.0045040036 0.0009833186 0.001000080
## [6,]          0.005          0.006 0.0055047103 0.0009743782 0.001000146
```

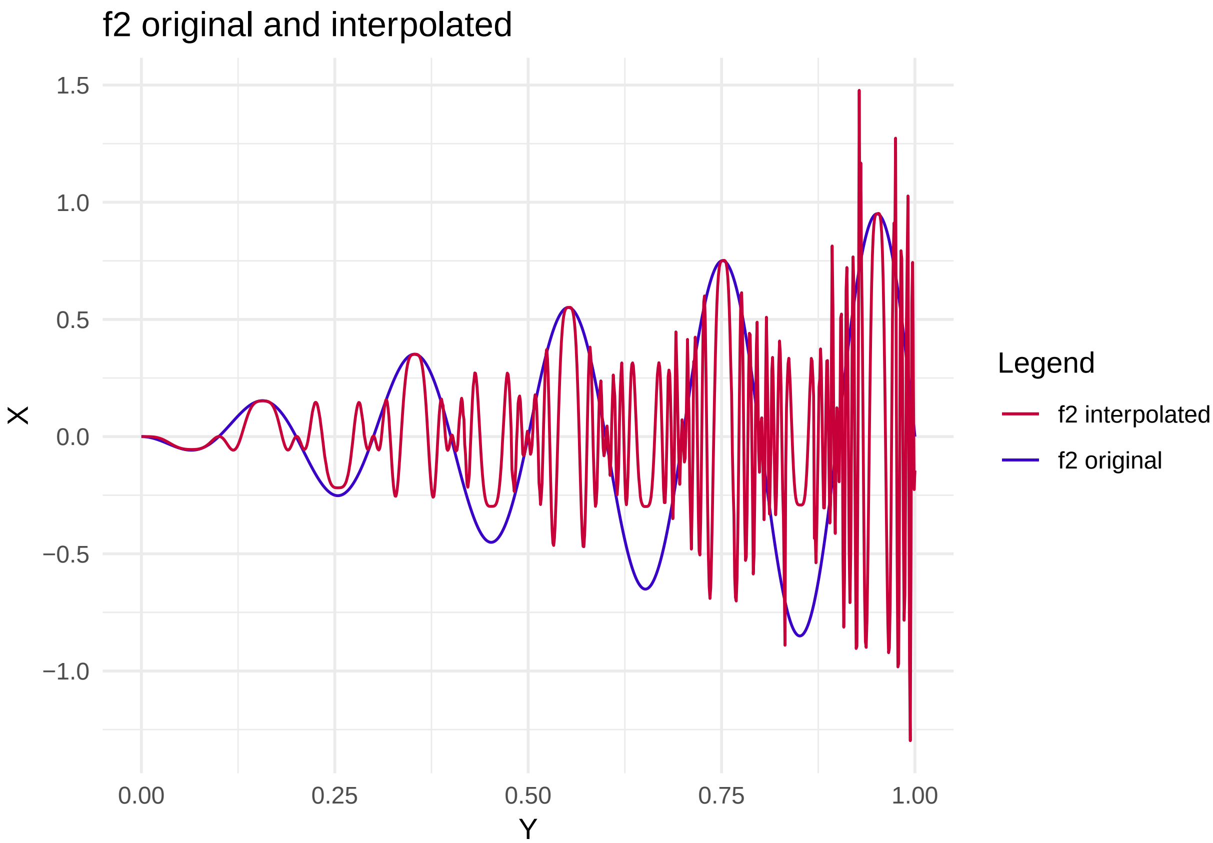
```
f2_res = f_approximate(f2, bins = 1000, rep(0.005, 3))
head(f2_res)
```

```
##      lower_boundary upper_boundary            a0          a1          a2
## [1,]          0.000          0.001  3.370842e-07 0.005000065 0.005000000
## [2,]          0.001          0.002  4.342633e-07 0.005000379 0.005000000
## [3,]          0.002          0.003 -1.133435e-07 0.005001007 0.005000000
## [4,]          0.003          0.004 -2.782254e-06 0.005001948 0.004999999
## [5,]          0.004          0.005 -9.768568e-06 0.005003205 0.004999998
## [6,]          0.005          0.006 -2.396517e-05 0.005004783 0.004999995
```

```
f_tilde = function(x, a0, a1, a2) a0 + a1 * x + a2 * x^2

df$f1_y_interpolated = sapply(sequence, FUN = function(x) {
  target_row = f1_res[x >= f1_res[,1] & x < f1_res[,2]]
  return(-f_tilde(x, target_row[3], target_row[4], target_row[5]))
```

```
})

df$f2_y_interpolated = sapply(sequence, FUN = function(x) {
  target_row = f2_res[x >= f2_res[,1] & x < f2_res[,2]]
  return(f_tilde(x, target_row[3], target_row[4], target_row[5]))
})
```




f2 original and interpolated
1.5
1.0
0.5
0.0
−0.5
−1.0
X
0.00
0.25
0.50
0.75
1.00
Y
Legend
f2 interpolated
f2 original