Blatt 6

Burr & Lübeck

7 6 2020

Aufgabe 1

$ f(x,y) = x y $ s.t: $ y = x - 2 $

Elimination von y:

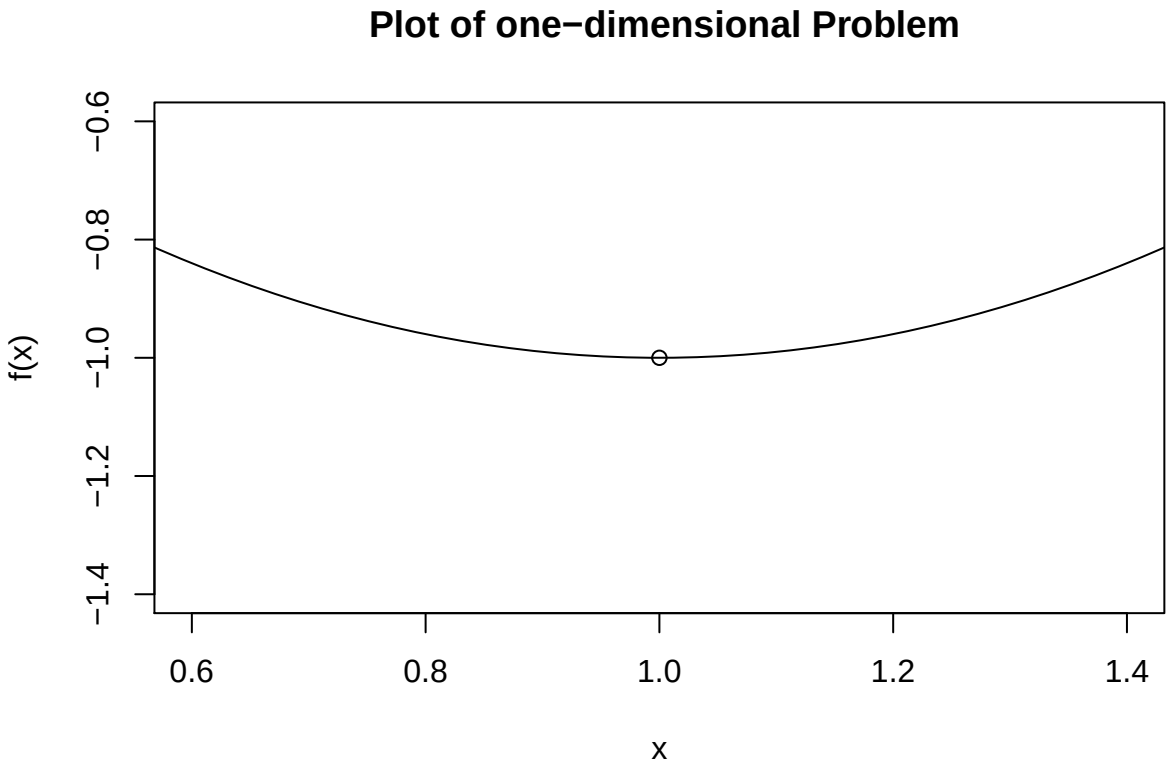
$ f(x) = x^2 - 2x $

Diese Funktion soll opitmiert werden. Ich gehe davon aus, dass damit Minimierung gemeint ist.

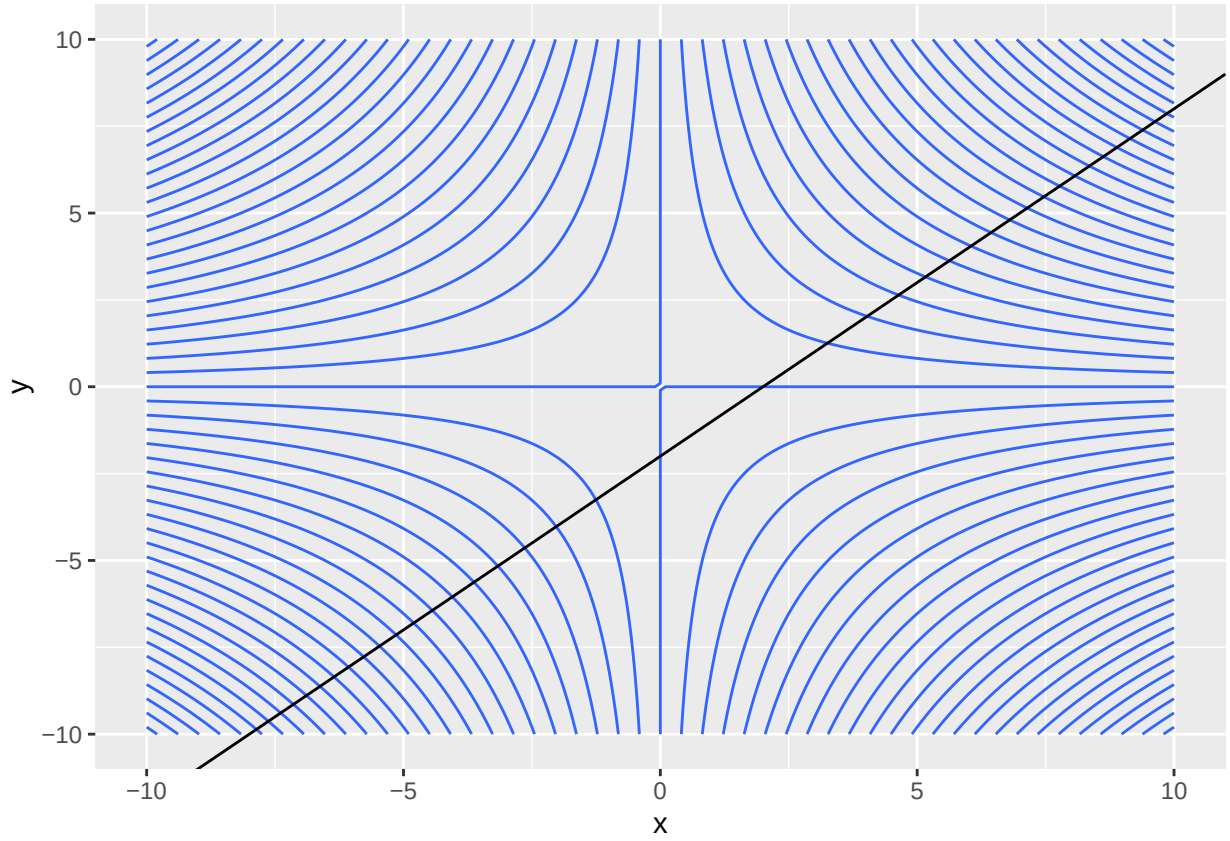
```
library(ggplot2)

fx = function(x) x**2 - 2*x

fxy = function(x) x[1] *x[2]


optim(method="BFGS", fn=fx, par=0)
```

```
## $par
## [1] 1
##
## $value
## [1] -1
##
## $counts
## function gradient
##        6        3
##
## $convergence
## [1] 0
##
## $message
## NULL
```

Optim konvergiert zu x=1 mit f(x)=-1. Durch Wiedereinsetzen in die Nebenbedingung ergibt sich y=-1.



Durch die Elimination wird das Problem im Eindimensionalen durch eine einfache, konvexe Parabel repräsentiert.



Die Isogewinnlinien zeigen in Richtung des Ursprungs bzw der Achsen. Allerdings sind wegen der Nebenbedinung nur die Punkte zugelassen, die auf der schwarzen Linie liegen. Daher ist die beste Lösung bei (1,-1).

Aufgabe 2

```
one_lambda_ea = function(f,lambda=2,parent,evals,lower,upper,
                         repair="border",
                         method="plus", sigma0=0.0625,
                         tau=0.25){

  #initialize
  n = length(parent)
  sigma = sigma0
  fitness_p = f(parent)
  evals_left = evals -1

  while(evals_left > 0){

    #schwefel
    sigma=sigma * exp(rnorm(n=1,mean=0,sd=tau))
    lambda= min(lambda, evals_left)
    children = parent + sigma * matrix(rnorm(lambda * n), nrow=n)

    # repair unvalid children
```

```
    # repair = border
    if(repair == "border"){
      children[children < lower] = lower
      children[children > upper] = upper
    }
    # repair = modulo
    if(repair == "modulo"){
      children[children < lower] = (upper - (lower- children[children < lower])) %% (upper - lower)
      children[children > upper] = (lower + (children[children < lower] - upper)) %% (upper - lower)
    }

    # repair = mirror
    if(repair=="mirror"){
      children[children < lower] = lower + (lower - children[children < lower])
      children[children >upper] = upper - (children[children < lower] - upper)
    }


    # evaluate children
    fitness_c = apply(children,2,f)
    evals_left = evals_left - lambda

    # selectionn
    if(method == "plus") {
      min_i = which.min(c(fitness_p, fitness_c))
      parent = rbind(parent, t(children))[min_i, ]
      fitness_p =c(fitness_p, fitness_c)[min_i]
    }
    else {
      min_ind = which.min(fitness_c)
      parent = children[, min_ind]
      fitness_p = fitness_c[min_ind]
    }

    }
    return(list(x=parent, fx=fitness_p))
  }
```

```
f = function(x) x[1]^3 * sin(x[1]+1) - x[2]^3 * cos(x[2])
```

Border Method

```
set.seed(08062020)
#imitialize randoms sample
reps=1000
pp = matrix(runif(reps, min=-10,max=10), ncol=2)


df_b = data.frame(matrix(ncol=3, nrow=0))
colnames(df_b) = c("X1", "X2", "Fx")

# iterate over sample and save results in data frame
for (i in 1:((reps/2)-1)){
```

```
 df_b[i,1] = one_lambda_ea(f,parent=pp[i,], evals=100,
                           lower=-10,upper=10, method="border")$x[1]
 df_b[i,2] = one_lambda_ea(f,parent=pp[i,], evals=100,
                           lower=-10,upper=10, method="border")$x[2]
 df_b[i,3] = one_lambda_ea(f,parent=pp[i,], evals=100,
                           lower=-10,upper=10, method="border")$fx

}
```

Mirror Method

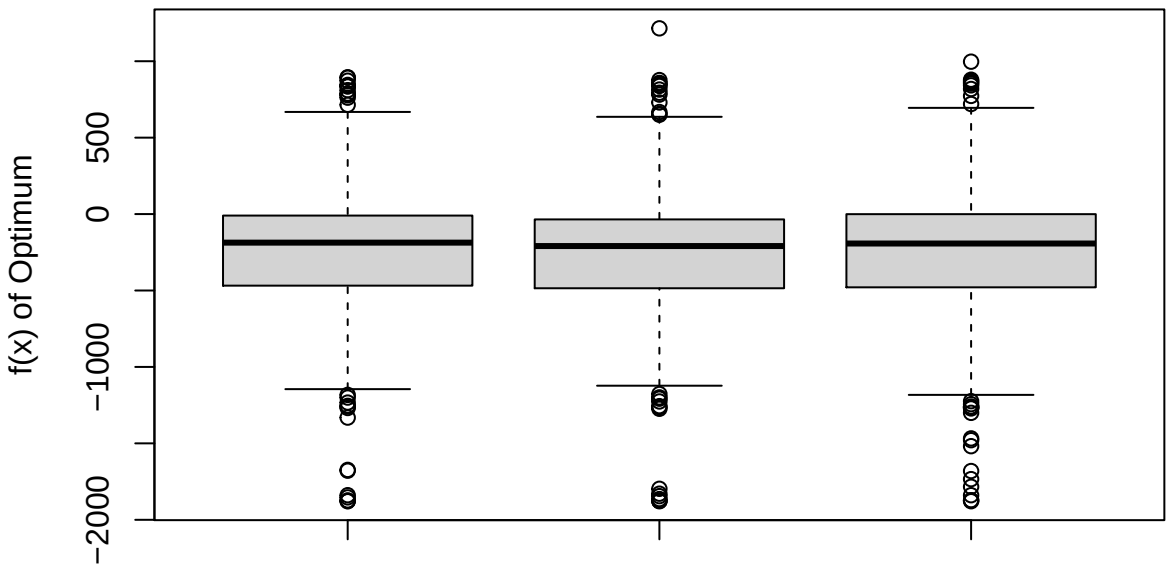
```
df_mi = data.frame(matrix(ncol=3, nrow=0))
colnames(df_b) = c("X1", "X2", "Fx")
for (i in 1:((reps/2)-1)){

 df_mi[i,1] = one_lambda_ea(f,parent=pp[i,], evals=100,
                           lower=-10,upper=10, method="mirror")$x[1]
 df_mi[i,2] = one_lambda_ea(f,parent=pp[i,], evals=100,
                           lower=-10,upper=10, method="mirror")$x[2]
 df_mi[i,3] = one_lambda_ea(f,parent=pp[i,], evals=100,
                           lower=-10,upper=10, method="mirror")$fx

}
```

Modulo Method

```
df_mo = data.frame(matrix(ncol=3, nrow=0))
colnames(df_b) = c("X1", "X2", "Fx")
for (i in 1:((reps/2)-1)){

 df_mo[i,1] = one_lambda_ea(f,parent=pp[i,], evals=100,
                           lower=-10,upper=10, method="modulo")$x[1]
 df_mo[i,2] = one_lambda_ea(f,parent=pp[i,], evals=100,
                           lower=-10,upper=10, method="modulo")$x[2]
 df_mo[i,3] = one_lambda_ea(f,parent=pp[i,], evals=100,
                           lower=-10,upper=10, method="modulo")$fx

}
```

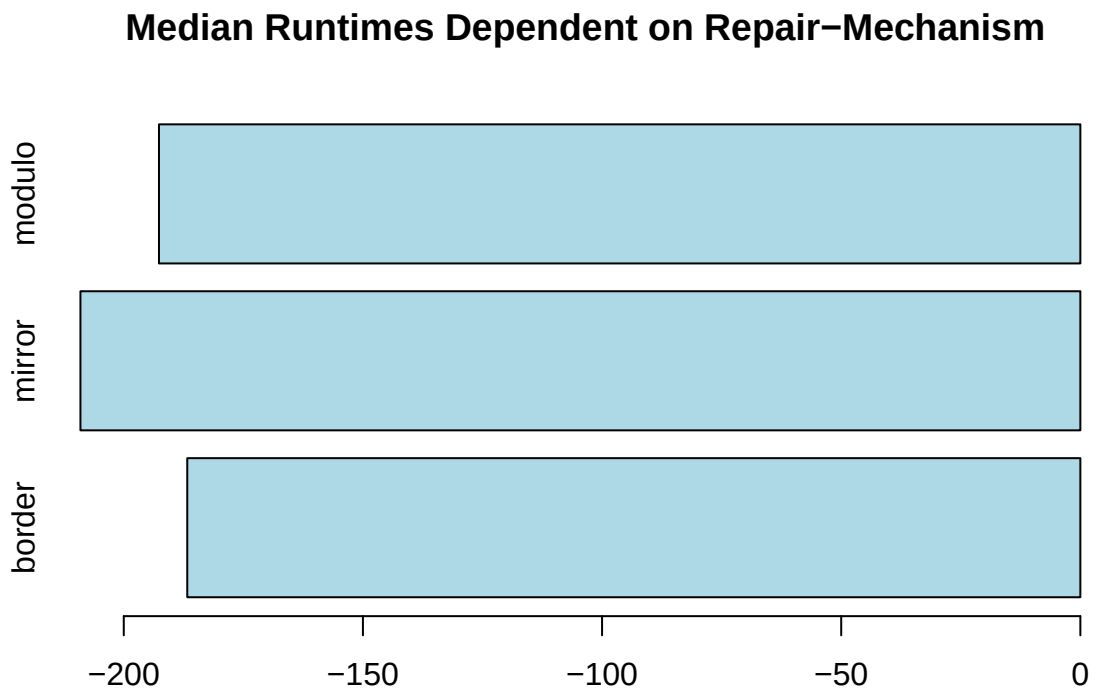
```
opt_b = df_b[,3]

opt_mi = df_mi[,3]

opt_mo = df_mo[,3]
```

```
boxplot(opt_b,opt_mi,opt_mo,
        ylab="f(x) of Optimum")
```



In den Boxplots ähneln sich die Verteilungen der gefundenen Optima der drei Einstellungen sehr stark. Da es einige Ausreiser gibt, wählen wir als Maß der zentralen Tendenz den Median.

```
med_b = median(opt_b)

med_mi = median(opt_mi)

med_mo = median(opt_mo)

barplot(height=c(med_b,med_mi,med_mo), horiz=TRUE, names.arg = c("border", "mirror", "modulo"), col="li
```



Plottet man nur die Mediane der 3 Varianten, lassen sich gewisse Unterschiede erkennen. Hier findet die Parametereinstellung "Modulo" im Schnitt die kleinsten, also besten Optima. Das ist - für mich - auf den ersten Blick überraschend, weil es die Variante ist, die die geringste Info beibehält. Vielleicht ist das aber auch gerade die Stärke, dass durch den Einsatz des Modulos zusätzliches "Chaos" erzeugt wird, was ja die Stärke von Evolutionären Algorithmen ist. Gerade hier, wo lokale Optima vor allem an den Rändern liegen, kann das helfen.

```
wilcox.test(opt_b,opt_mo)
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  opt_b and opt_mo
## W = 122692, p-value = 0.6912
## alternative hypothesis: true location shift is not equal to 0
```

Wenn man allerdings die Unterschiede inferenzstatistisch absichern möchte, findet man keinen signifikanten Unterschied, was bei einer doch großen Stichprobe von n=500 durchaus gegen einen starken Effekt spricht. Die deskriptiven Unterschiede sollten also mit Vorsicht interpretiert werden.

Aufgabe 3

```
strafopt = function(f,g,h,
                    xnull=0,ceta=1,gamma=2,
                    evals=100){
  ##########################################
  #Iterative approach to loss function optimazitation
```

```
  #f: target function
  #g & h: inequality constraints with zero
  #(which is no restriction of generality)



  #intit
  k = 1
  c = ceta
  x_k = xnull
  evals_left = evals

  while (evals_left>0){  #give a maximal budget

    res = optim(fn=(f+c*g+c*h),par=x_k, method="BFGS")
    x_k = res$x

    if (g(x_k)<=0 & h(x_k)<=0){
      break  ## valid result is found: break loop

    }
    else{
      k = k+1
      c = gamma * c
      next #continue with next iteration

    }

  }
  return(res)


}
```