Concrete Compressive Strength

CTC-34\_Lab2: Evolução Gramátical

Matheus Vidal & Pedro Alves

17/11/2019

## Definição da Gramática

# Limpa Área de trabalho  
rm(list = ls())  
  
#set seed  
set.seed(165465)  
  
# Não Terminais  
V <- c("<exp>","<op>","<var>")  
  
# Terminais  
op <- c("+","-","/","\*")  
  
# Geração de constantes  
max\_const <- 3.15  
const\_quant <- 10000  
amostra <- 10000  
const <- runif(const\_quant,min = 0,max=max\_const)  
  
max\_const <- 10  
const\_quant <- 1000  
amostra <- 1000  
const <- c(const, runif(const\_quant,min = max\_const/2,max=max\_const\*1.5))  
  
max\_const <- 100  
const\_quant <- 1000  
amostra <- 1000  
const <- c(const, runif(const\_quant,min = max\_const/2,max=max\_const\*1.5))  
  
  
max\_const <- 1000  
const\_quant <- 1000  
amostra <- 1000  
const <- c(const, runif(const\_quant,min = max\_const/2,max=max\_const\*1.5))  
  
max\_const <- 10000  
const\_quant <- 1000  
amostra <- 1000  
const <- c(const, runif(const\_quant,min = max\_const/2,max=max\_const\*1.5))  
  
max\_const <- 100000  
const\_quant <- 1000  
amostra <- 1000  
const <- c(const, runif(const\_quant,min = max\_const/2,max=max\_const\*1.5))  
  
max\_const <- 1000000  
const\_quant <- 1000  
amostra <- 1000  
const <- c(const, runif(const\_quant,min = max\_const/2,max=max\_const\*1.5))  
  
max\_const <- 10000000  
const\_quant <- 1000  
amostra <- 1000  
const <- c(const, runif(const\_quant,min = max\_const/2,max=max\_const\*1.5))  
  
max\_const <- 100000000  
const\_quant <- 1000  
amostra <- 1000  
const <- c(const, runif(const\_quant,min = max\_const/2,max=max\_const\*1.5))  
  
var <- rep(c("x","y","z","u","v","w","p","q"),25000)  
var <- c(var,const)  
sigma <- c(op,var)  
  
  
# Símbolo Inicial  
S <- "<exp>"  
  
# Produções  
basicas <- c(rep(c("<var>"),1000),rep(c("<var><op><exp>"),5000),rep(c("<var><op><exp>"),0))  
seno <- c(rep(c("sin(<var>)"),35),rep(c("sin(<exp>)"),15),rep(c("sin(<var>)<op><exp>"),20),rep(c("sin(<exp>)<op><exp>"),10))  
aseno <- c(rep(c("asin(<var>)"),35),rep(c("asin(<exp>)"),15),rep(c("asin(<var>)<op><exp>"),20),rep(c("asin(<exp>)<op><exp>"),10))  
senoh <- c(rep(c("sinh(<var>)"),35),rep(c("sinh(<exp>)"),15),rep(c("sinh(<var>)<op><exp>"),20),rep(c("sinh(<exp>)<op><exp>"),10))  
asenoh <- c(rep(c("asinh(<var>)"),35),rep(c("asinh(<exp>)"),15),rep(c("asinh(<var>)<op><exp>"),20),rep(c("asinh(<exp>)<op><exp>"),10))  
cosseno <- c(rep(c("cos(<var>)"),35),rep(c("cos(<exp>)"),15),rep(c("cos(<var>)<op><exp>"),20),rep(c("cos(<exp>)<op><exp>"),10))  
acosseno <- c(rep(c("acos(<var>)"),35),rep(c("acos(<exp>)"),15),rep(c("acos(<var>)<op><exp>"),20),rep(c("acos(<exp>)<op><exp>"),10))  
cossenoh <- c(rep(c("cosh(<var>)"),35),rep(c("cosh(<exp>)"),15),rep(c("cosh(<var>)<op><exp>"),20),rep(c("cosh(<exp>)<op><exp>"),10))  
acossenoh <- c(rep(c("acosh(<var>)"),35),rep(c("acosh(<exp>)"),15),rep(c("acosh(<var>)<op><exp>"),20),rep(c("acosh(<exp>)<op><exp>"),10))  
tangente <- c(rep(c("tan(<var>)"),35),rep(c("tan(<exp>)"),15),rep(c("tan(<var>)<op><exp>"),20),rep(c("tan(<exp>)<op><exp>"),10))  
atangente <- c(rep(c("atan(<var>)"),35),rep(c("atan(<exp>)"),15),rep(c("atan(<var>)<op><exp>"),20),rep(c("atan(<exp>)<op><exp>"),10))  
tangenteh <- c(rep(c("tanh(<var>)"),35),rep(c("tanh(<exp>)"),15),rep(c("tanh(<var>)<op><exp>"),20),rep(c("tanh(<exp>)<op><exp>"),10))  
atangenteh <- c(rep(c("atanh(<var>)"),35),rep(c("atanh(<exp>)"),15),rep(c("atanh(<var>)<op><exp>"),20),rep(c("atanh(<exp>)<op><exp>"),10))  
logaritmo <- rep(c(rep(c("log(<var>)"),35),rep(c("log(<exp>)"),15),rep(c("log(<var>)<op><exp>"),20),rep(c("log(<exp>)<op><exp>"),10)),5)  
exp\_natural <- rep(c(rep(c("exp(<var>)"),35),rep(c("exp(<exp>)"),15),rep(c("exp(<var>)<op><exp>"),20),rep(c("exp(<exp>)<op><exp>"),10)),5)  
exponencial <- rep(c(rep(c("<exp>^(<var>)"),35),rep(c("<exp>^(<exp>)"),15),rep(c("<exp>^(<var>)<op><exp>"),20),rep(c("<exp>^(<exp>)<op><exp>"),10)),2)  
modulo <- rep(c(rep(c("abs(<var>)"),35),rep(c("abs(<exp>)"),15),rep(c("abs(<var>)<op><exp>"),20),rep(c("abs(<exp>)<op><exp>"),10)),2)  
quadrada <- rep(c(rep(c("sqrt(<var>)"),35),rep(c("sqrt(<exp>)"),15),rep(c("sqrt(<var>)<op><exp>"),20),rep(c("sqrt(<exp>)<op><exp>"),10)),5)  
  
  
P <- list(c(basicas,quadrada,seno,aseno,asenoh,cosseno,cossenoh,acosseno,senoh,cossenoh,logaritmo,tangente,atangenteh,atangente,tangenteh,  
 exp\_natural,exponencial,modulo),op,var)

## Definição dos Cromossomos

Cada cromossomo é gerado aleatóriamente.

cromo\_size <- 200  
cromo\_quant <- 100  
max\_gene <- 10000000  
p.mutacao <- c(0,0,0.01,0.03,0.05,0.1,0.2,0.35,0.55,0.80,1,1,1)  
p.cruzamento <- 0.4  
if(max\_gene < cromo\_size\*cromo\_quant){  
 max\_gene <- cromo\_size\*cromo\_quant  
}  
first\_sorteados\_fraction <- 10  
order\_size\_fraction <- 3  
prop\_filhos <- 2  
C <- matrix(sample(max\_gene,max\_gene),cromo\_quant,cromo\_size) # Ci = C[i,]

## Carregamento dos Dados de Treinamento

trainingData <- read.csv(file="training.csv", header=TRUE, sep=",")

## Gerando Derivação

Gera-se aqui as expressões matemáticas a partir de cada cromossomo.

S <- "<exp>"  
derivacao <- rep(S,cromo\_quant) # símbolo inicial  
endCromo <- array(0,cromo\_quant)  
for (i in 1:cromo\_quant) {  
 S <- "<exp>"  
 j <- 0  
 count <- 0  
 while (S!="<NA>"){  
 count <- count + 1  
 # evitar loop infinito, para cromossomos mal escolhidos:  
 if (count > cromo\_size){  
 tmp <- stringr::str\_match(derivacao[i], ".\*?<")  
 tmp <- substr(tmp,1,nchar(tmp)-1)  
 if (is.na(tmp)){  
 last.char <- substr(tmp,nchar(derivacao[i]),nchar(derivacao[i]))  
 if (!last.char%in%var){  
 tmp <- substr(derivacao[i],1,nchar(derivacao[i])-1)   
 }  
 } else {  
 last.char <- substr(tmp,nchar(tmp),nchar(tmp))  
 if (!last.char%in%var){   
 # se terminar em operador  
 if (last.char%in%op){  
 tmp <- substr(tmp,1,nchar(tmp)-1)   
 } else if (substr(tmp,nchar(tmp)-1,nchar(tmp))=="^("){  
 tmp <- substr(tmp,1,nchar(tmp)-2)  
 } else if (substr(tmp,nchar(tmp)-5,nchar(tmp))=="asinh("){  
 tmp <- substr(tmp,1,nchar(tmp)-6)  
 } else if (substr(tmp,nchar(tmp)-5,nchar(tmp))=="acosh("){  
 tmp <- substr(tmp,1,nchar(tmp)-6)  
 } else if (substr(tmp,nchar(tmp)-5,nchar(tmp))=="atanh("){  
 tmp <- substr(tmp,1,nchar(tmp)-6)  
 } else if (substr(tmp,nchar(tmp)-1,nchar(tmp))=="h("){  
 tmp <- substr(tmp,1,nchar(tmp)-5)  
 } else if (substr(tmp,nchar(tmp)-4,nchar(tmp))=="sqrt("){  
 tmp <- substr(tmp,1,nchar(tmp)-5)  
 } else if (substr(tmp,nchar(tmp)-4,nchar(tmp))=="asin("){  
 tmp <- substr(tmp,1,nchar(tmp)-5)  
 } else if (substr(tmp,nchar(tmp)-4,nchar(tmp))=="acos("){  
 tmp <- substr(tmp,1,nchar(tmp)-5)  
 } else if (substr(tmp,nchar(tmp)-4,nchar(tmp))=="atan("){  
 tmp <- substr(tmp,1,nchar(tmp)-5)  
 } else if (last.char=="("){  
 tmp <- substr(tmp,1,nchar(tmp)-4)   
 }  
 }  
 last.char <- substr(tmp,nchar(tmp),nchar(tmp))  
 # se terminar em operador  
 if (last.char%in%op){  
 tmp <- substr(tmp,1,nchar(tmp)-1)   
 }  
 endCromo[i] <- -1  
 }  
 # colocar parênteses restantes  
 abrindo <- stringr::str\_count(tmp,"\\(")  
 fechando <- stringr::str\_count(tmp,"\\)")  
 add <- strrep(")",abrindo-fechando)  
 tmp <- paste(tmp,add,sep = "")  
   
 derivacao[i] <- tmp  
 S <- "<NA>"  
 }  
 j <- j%%cromo\_size + 1  
   
 # Faz substituições  
 if (S=="<exp>"){  
 rule <- P[[1]][1+C[i,j]%%length(P[[1]])] # determina a regra a ser aplicada  
 derivacao[i] <- sub(S, rule, derivacao[i]) # atualiza derivacao por substituição  
 S <- stringr::str\_match(derivacao[i], "<(.\*?)>") # acha o primeiro <..>  
 S <- paste("<",S[,2],">",sep = "") # novo simbolo  
 } else if (S=="<op>"){  
 rule <- P[[2]][1+C[i,j]%%length(P[[2]])] # determina a regra a ser aplicada  
 derivacao[i] <- sub(S, rule, derivacao[i]) # atualiza derivacao por substituição  
 S <- stringr::str\_match(derivacao[i], "<(.\*?)>") # acha o primeiro <..>  
 S <- paste("<",S[,2],">",sep = "") # novo simbolo  
 } else if (S=="<var>"){  
 rule <- P[[3]][1+C[i,j]%%length(P[[3]])] # determina a regra a ser aplicada  
 derivacao[i] <- sub(S, rule, derivacao[i]) # atualiza derivacao por substituição  
 S <- stringr::str\_match(derivacao[i], "<(.\*?)>") # acha o primeiro <..>  
 S <- paste("<",S[,2],">",sep = "") # novo simbolo  
 }  
 }  
 endCromo[i] <- endCromo[i] + count  
}  
derivacao

## [1] "v/tan(y)"   
## [2] "sqrt(exp(v+p+cosh(p))-p-v/x-w+cosh(v))"   
## [3] "u+2.23955080801388/asinh(x)"   
## [4] "x"   
## [5] "y-v"   
## [6] "p\*p+sqrt(z)"   
## [7] "sqrt(w)+w"   
## [8] "u/sqrt(asinh(z))-u-98.3543772716075\*exp(y)^(v)-atanh(q)/sqrt(q)-y-q-q-exp(abs(q))"   
## [9] "y^(x-y/2.16979381801793-sinh(v)+x/w/y/u\*p)/exp(y)"   
## [10] "y/q-u"   
## [11] "u\*exp(z)-z+exp(y)"   
## [12] "u+p"   
## [13] "z+z-z\*w\*exp(2.41571084397146)"   
## [14] "x/u-exp(w)"   
## [15] "z+u\*x/asin(x)"   
## [16] "u-log(p)"   
## [17] "v"   
## [18] "z-x+1.9828120994498+u\*q/u"   
## [19] "cos(x\*sqrt(q))+exp(z)"   
## [20] "log(q\*z/v+1294.53106690198-p/u^(v)^(v))"   
## [21] "z\*w\*sqrt(p)+v-q\*x/exp(p)\*w"   
## [22] "v"   
## [23] "p\*z-sqrt(exp(y\*v+sqrt(sqrt(y\*abs(z)))\*y+w+cosh(1.41732286289334-p+u-p)+p\*y))^(v/tanh(y)\*sqrt(u)-v-z-w+x)"   
## [24] "1.86170721313683+u\*v-p\*y-y"   
## [25] "v\*y+tan(q)"   
## [26] "sin(y)"   
## [27] "z"   
## [28] "log(y)^(0.0349425449967384)\*cosh(w)"   
## [29] "w+z"   
## [30] "sqrt(x+cosh(y)+q+p\*14926824.8514272/exp(q-sqrt(q)-v/v+log(1.50212439384777))/exp(z))"   
## [31] "q/w\*u+u-w+q-sqrt(y)\*acos(q/sinh(u))"   
## [32] "cos(x\*w\*u+652308.284537867)^(p)"   
## [33] "w-p+sqrt(abs(v))\*x-y-exp(v)/p"   
## [34] "q-sqrt(q)"   
## [35] "v-z\*cosh(p)-acos(x)"   
## [36] "sinh(y)+v"   
## [37] "sqrt(q)"   
## [38] "x+log(y)"   
## [39] "u"   
## [40] "u/u-104.482182813808/abs(x-w+v-sqrt(q))"   
## [41] "y-exp(u\*q+q-p-v-asin(q))"   
## [42] "w-z/x\*y+atan(sqrt(y))/log(w)^(w)/q+q"   
## [43] "sqrt(cosh(u-x/y-exp(z)))"   
## [44] "p"   
## [45] "x-y-sqrt(z)-atan(log(w)+w/8349412.43985668)"   
## [46] "u\*exp(u-z-v\*abs(p))"   
## [47] "v-p+1.93361547149252"   
## [48] "p+q"   
## [49] "y/v+log(u)"   
## [50] "x+8.57818119460717-sqrt(z)"   
## [51] "x\*x-cos(x)"   
## [52] "z+atan(u)"   
## [53] "v/v+q\*z+w+y/p+1.46019304501824-11.6554226796143\*cosh(sqrt(p/x-p+cosh(w)))"   
## [54] "sqrt(z)"   
## [55] "z+w+w-w"   
## [56] "u+p+y/x"   
## [57] "z+cos(p+77878.7357732654\*sinh(z)-sinh(v))+atanh(u+v-exp(12594.0873823129))"   
## [58] "sqrt(q)"   
## [59] "acos(z)"   
## [60] "w"   
## [61] "p\*log(z)"   
## [62] "p\*v+z\*q/v/tanh(z+y+u-p/y)"   
## [63] "q/cos(q)+v-sin(x-atan(x))+cos(cosh(exp(y)))"   
## [64] "y-u+u\*x\*log(p+u\*tan(sinh(p)/z\*p)-w)"   
## [65] "p+exp(w)"   
## [66] "q-acos(v)"   
## [67] "q"   
## [68] "w+sqrt(x)"   
## [69] "v+w"   
## [70] "w\*tan(q)"   
## [71] "x-cosh(z+tan(w))/sqrt(z)"   
## [72] "v-q/sinh(u)\*w-sqrt(x)^(v)"   
## [73] "1.3008682862157/v/exp(u-p/x-tan(y))"   
## [74] "w/w+sqrt(2.25925432539079\*x-p\*log(x))-sqrt(q)-x-tanh(q)^(p-2.3127512157429/y-atan(u))"   
## [75] "q"   
## [76] "sqrt(p+w-999213.432194665\*w+z/x-y/1.32852541686734/y\*q-u-exp(3.13486336466158))"   
## [77] "exp(sqrt(p))/x+y+exp(1.98409532607766/sinh(z)\*log(z))-q\*acos(y)"   
## [78] "0.628235456894617/z-q\*asinh(u)-cosh(u)/z/y-log(log(u-w\*p\*u\*x\*z-abs(q)))"   
## [79] "y\*y-x"   
## [80] "w"   
## [81] "x+v"   
## [82] "x/y+1.63233152932953\*tanh(u)"   
## [83] "q/z\*p+v"   
## [84] "log(x+q)"   
## [85] "x"   
## [86] "u+acos(z)^(z)/q\*x/w/13804364.2393313\*z\*v/tanh(cosh(q\*q-q\*x)+q\*q+cos(u))\*sqrt(q-log(z))-w+log(v)-q/tan(w)+z\*y\*y\*14183.8945331983/x/exp(x)"  
## [87] "x"   
## [88] "exp(11154805.1750287)\*x\*q-sinh(y)/z-u-w/log(y)"   
## [89] "w+exp(y)\*cos(y)"   
## [90] "u-x/v/q\*w\*log(x)"   
## [91] "1.03488615900278"   
## [92] "sqrt(z)+y-p+z\*x-v/sqrt(y)^(z)"   
## [93] "p^(y)-x-q-w"   
## [94] "q\*y+cosh(w)+x"   
## [95] "q+v-y"   
## [96] "x/x"   
## [97] "q+v/exp(z)"   
## [98] "x/2.87060482936213-x+v\*cosh(x)"   
## [99] "x\*atan(q+sqrt(z))/p"   
## [100] "v\*y\*w\*abs(w)^(0.0971202179091051-atanh(q))\*p/5465.85498843342-tanh(q)\*u+u\*x-q-137169725.052081"

## Avaliação dos Cromossomos

Avalia quão bem estão cada cromossomo pelo seus respectivos valores de erro quadrático médio.

options(warn = -1)  
training.model <- trainingData[,c("strength")]  
erroQuadraticoMedio <- rep(Inf,cromo\_quant)  
for (k in 1:cromo\_quant){  
 training.data <- c()  
 for (i in 1:length(training.model)){  
 x <- trainingData[i,2]  
 y <- trainingData[i,3]  
 z <- trainingData[i,4]  
 u <- trainingData[i,5]  
 v <- trainingData[i,6]  
 w <- trainingData[i,7]  
 p <- trainingData[i,8]  
 q <- trainingData[i,9]  
 valor <- suppressWarnings(eval(parse(text=derivacao[k])))   
 training.data <- c(training.data,valor)  
 }  
 erroQuadraticoMedio[k] <- mean((training.data - (training.model))^2)   
}  
options(warn = 0)  
erroQuadraticoMedio

## [1] NaN NaN Inf 0.05154777 0.36266019  
## [6] 0.29813175 0.83324259 Inf NaN 623.93697007  
## [11] 1.69902933 0.36500144 5.65134265 Inf NaN  
## [16] Inf 0.10277478 NaN 3.97007001 NaN  
## [21] 0.11765367 0.10277478 NaN 1.60354680 0.13635021  
## [26] 0.14383323 0.18190592 NaN 0.37455799 NaN  
## [31] NaN 0.26940546 Inf 0.43500996 3.01277710  
## [36] 0.09182322 0.06490272 Inf 0.10605711 NaN  
## [41] 0.77641610 NaN Inf 0.10827206 0.37231403  
## [46] 0.40556086 1.55139642 0.12650189 NaN 67.75983461  
## [51] 1.29956003 0.22137168 NaN 0.21289813 0.37455799  
## [56] Inf NaN 0.06490272 0.87816969 0.11911191  
## [61] NaN NaN 0.40938331 NaN 3.29452480  
## [66] 2.83271027 0.14935169 0.56393354 0.17395544 0.18240757  
## [71] Inf Inf Inf NaN 0.14935169  
## [76] NaN Inf NaN 0.74977703 0.11911191  
## [81] 0.11138038 Inf NaN 1.75402404 0.05154777  
## [86] NaN 0.05154777 NaN 1.71985668 NaN  
## [91] 0.40833916 NaN 0.73216680 1.41108091 0.22183285  
## [96] NaN 0.06416851 0.30952445 Inf NaN

## Cromossomo Vencedor da Primeira Geração

# Cromossomo vencedor:  
vencedor <- which.min(erroQuadraticoMedio)  
vencedor

## [1] 4

raizErroQuadraticoMedioMin <- sqrt(erroQuadraticoMedio[vencedor])  
derivacao\_vencedora <-derivacao[vencedor]  
derivacao\_vencedora

## [1] "x"

## Torneio de Cromossomos

Do algoritmo genético, é feito o torneio de cromossomos, de modo a escolher o de melhor erro quadrático médio a cada disputa entre dois deles.

sorteados <- c()  
invalidosNanInf <- which(erroQuadraticoMedio %in% c(NaN,Inf))  
invalidosVazio <- which(derivacao %in% c(""))  
invalidos <- c(invalidosNanInf,invalidosVazio,which(duplicated(derivacao)))  
validos <-c(1:length(derivacao))[which(!c(1:length(derivacao)) %in% invalidos)]  
  
first\_sorteados <- length(validos)%/%first\_sorteados\_fraction  
order\_size <- (order\_size\_fraction\*length(validos))%/%first\_sorteados\_fraction  
ordenados <- validos[order(erroQuadraticoMedio[validos])][1:order\_size]  
sorteados <- validos[order(erroQuadraticoMedio[validos])][1:first\_sorteados]  
  
for (i in (first\_sorteados+1):(order\_size\*prop\_filhos+first\_sorteados+1)){  
 sorteio <- rep(NA,cromo\_quant)  
 pos <- sample(ordenados,2) # sorteio dois crommossomos  
 sorteio[pos] <- pos  
 sorteados <- c(sorteados,which.min(erroQuadraticoMedio[sorteio]))  
}  
  
for (i in (order\_size\*prop\_filhos+first\_sorteados+2):cromo\_quant){  
 sorteio <- rep(NA,cromo\_quant)  
 pos <- sample(validos,2) # sorteio dois crommossomos  
 sorteio[pos] <- pos  
 sorteados <- c(sorteados,which.min(erroQuadraticoMedio[sorteio]))  
}  
  
C <- C[sorteados,]  
endCromo <- endCromo[sorteados]  
  
sorteados

## [1] 4 97 37 36 17 36 97 25 37 4 36 67 36 4 97 37 44 44 4 44 17 26 97 60 17  
## [26] 37 25 44 21 97 37 97 60 37 26 81 67 63 59 81 98 27 60 25 44 36 65 70 44 32  
## [51] 81 21 54 69 7 26 24 27 6 79 36 79 50 69 46 79 60 54 81 81 12 27 41 5 51  
## [76] 44 68 25 12 37 21 59 68 50 32 93 91 60 60 36 95 55 91 27 67 27 81 32 39 60

## Cruzamento de Cromossomos

COnforme o algoritmo genético, os cromossomos que vencedores de cada disputa são cruzados em pares.

i <- (first\_sorteados+1)  
while (i < length(sorteados)){  
 # avalia a possibilidade de cruzamento de cada par  
 vai.cruzar <- runif(1)  
 if (vai.cruzar<=p.cruzamento){ # realiza cruzamento  
 pto.cruza <- endCromo[i]  
 if(pto.cruza > endCromo[i+1])  
 {  
 pto.cruza <- endCromo[i+1]  
 }  
 pto.cruza <- sample(pto.cruza-1,1)  
 index\_pai <- i  
 index\_mae <- i+1  
 tmp <- C[index\_pai,(pto.cruza+1):cromo\_size]  
 C[index\_pai,(pto.cruza+1):cromo\_size] <- C[index\_mae,(pto.cruza+1):cromo\_size]   
 C[index\_mae,(pto.cruza+1):cromo\_size] <- tmp  
 }  
 i <- i+2 # pula o proximo cromossomo (estamos olhando os pares)  
}

## Mutação

Aqui, verifica-se a possibilidade de mutação de qualquer gene de qualquer cromossomo.

vai.mutar <- matrix(runif(cromo\_quant\*cromo\_size),cromo\_quant,cromo\_size)  
divisa\_prob <- cromo\_quant%/%first\_sorteados\_fraction  
for (i in (first\_sorteados+1):cromo\_quant){  
 linhaMutar <- vai.mutar[i,]  
 prob<-p.mutacao[i%/%divisa\_prob+1]  
 if(prob\*endCromo[i]<0.3)  
 {  
 prob <- 0.3/endCromo[i]  
 }  
 id <- which(linhaMutar < prob)  
 for (j in 1:length(id)){  
 coluna <- id[j]  
 C[i,coluna] <- sample(max\_gene,1)  
 }  
}

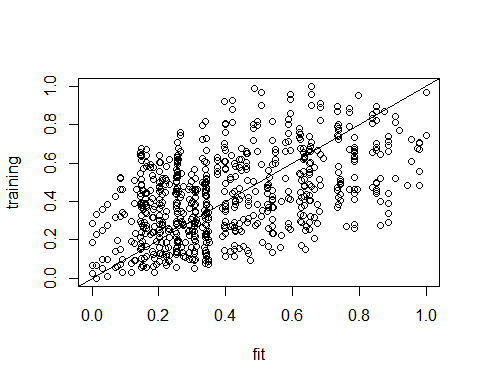
## Comparando com o Próprio training.csv (fit)

Aqui vamos comparar o quão bem nosso cromossomo vencedor se adequa ao dataset de treinamento.

fit.data <- c()  
for (i in 1:dim(trainingData)[1]){  
 x <- trainingData[i,2]  
 y <- trainingData[i,3]  
 z <- trainingData[i,4]  
 u <- trainingData[i,5]  
 v <- trainingData[i,6]  
 w <- trainingData[i,7]  
 p <- trainingData[i,8]  
 q <- trainingData[i,9]  
 fit.data <- c(fit.data,eval(parse(text=derivacao\_vencedora)))  
}

Graficamente, os dados calculados pelo cromossomo vencedor devem estar próximos da reta, como visto a seguir:

plot(fit.data,training.model,  
 xlab="fit",ylab="training")  
 abline(a=0,b=1)



## Após N Gerações

Vamos repetir o processo acima para mais N gerações.

ngeracoes <- 0 # Contador de Gerações do 0  
N <- 5 # número de gerações  
ngeracoes <- ngeracoes + N  
for (g in 1:N){  
 ## Gerando Derivação  
 if(g%%5 == 0){  
 cat("Geração: ", g,"\n")  
 }  
 S <- "<exp>"  
 derivacao <- rep(S,cromo\_quant) # símbolo inicial  
 endCromo <- array(0,cromo\_quant)  
 for (i in 1:cromo\_quant) {  
 S <- "<exp>"  
 j <- 0  
 count <- 0  
 while (S!="<NA>"){  
 count <- count + 1  
 # evitar loop infinito, para cromossomos mal escolhidos:  
 if (count > cromo\_size){  
 tmp <- stringr::str\_match(derivacao[i], ".\*?<")  
 tmp <- substr(tmp,1,nchar(tmp)-1)  
 if (is.na(tmp)){  
 last.char <- substr(tmp,nchar(derivacao[i]),nchar(derivacao[i]))  
 if (!last.char%in%var){  
 tmp <- substr(derivacao[i],1,nchar(derivacao[i])-1)   
 }  
 } else {  
 last.char <- substr(tmp,nchar(tmp),nchar(tmp))  
 if (!last.char%in%var){   
 # se terminar em operador  
 if (last.char%in%op){  
 tmp <- substr(tmp,1,nchar(tmp)-1)   
 } else if (substr(tmp,nchar(tmp)-1,nchar(tmp))=="^("){  
 tmp <- substr(tmp,1,nchar(tmp)-2)  
 } else if (substr(tmp,nchar(tmp)-5,nchar(tmp))=="asinh("){  
 tmp <- substr(tmp,1,nchar(tmp)-6)  
 } else if (substr(tmp,nchar(tmp)-5,nchar(tmp))=="acosh("){  
 tmp <- substr(tmp,1,nchar(tmp)-6)  
 } else if (substr(tmp,nchar(tmp)-5,nchar(tmp))=="atanh("){  
 tmp <- substr(tmp,1,nchar(tmp)-6)  
 } else if (substr(tmp,nchar(tmp)-1,nchar(tmp))=="h("){  
 tmp <- substr(tmp,1,nchar(tmp)-5)  
 } else if (substr(tmp,nchar(tmp)-4,nchar(tmp))=="sqrt("){  
 tmp <- substr(tmp,1,nchar(tmp)-5)  
 } else if (substr(tmp,nchar(tmp)-4,nchar(tmp))=="asin("){  
 tmp <- substr(tmp,1,nchar(tmp)-5)  
 } else if (substr(tmp,nchar(tmp)-4,nchar(tmp))=="acos("){  
 tmp <- substr(tmp,1,nchar(tmp)-5)  
 } else if (substr(tmp,nchar(tmp)-4,nchar(tmp))=="atan("){  
 tmp <- substr(tmp,1,nchar(tmp)-5)  
 } else if (last.char=="("){  
 tmp <- substr(tmp,1,nchar(tmp)-4)   
 }  
 }  
 last.char <- substr(tmp,nchar(tmp),nchar(tmp))  
 # se terminar em operador  
 if (last.char%in%op){  
 tmp <- substr(tmp,1,nchar(tmp)-1)   
 }  
 endCromo[i] <- -1  
 }  
 # colocar parênteses restantes  
 abrindo <- stringr::str\_count(tmp,"\\(")  
 fechando <- stringr::str\_count(tmp,"\\)")  
 add <- strrep(")",abrindo-fechando)  
 tmp <- paste(tmp,add,sep = "")  
   
 derivacao[i] <- tmp  
 S <- "<NA>"  
 }  
 j <- j%%cromo\_size + 1  
 if (S=="<exp>"){  
 rule <- P[[1]][1+C[i,j]%%length(P[[1]])] # determina a regra a ser aplicada  
 derivacao[i] <- sub(S, rule, derivacao[i]) # atualiza derivacao por substituição  
 S <- stringr::str\_match(derivacao[i], "<(.\*?)>") # acha o primeiro <..>  
 S <- paste("<",S[,2],">",sep = "") # novo símbolo  
 } else if (S=="<op>"){  
 rule <- P[[2]][1+C[i,j]%%length(P[[2]])] # determina a regra a ser aplicada  
 derivacao[i] <- sub(S, rule, derivacao[i]) # atualiza derivacao por substituição  
 S <- stringr::str\_match(derivacao[i], "<(.\*?)>") # acha o primeiro <..>  
 S <- paste("<",S[,2],">",sep = "") # novo símbolo  
 } else if (S=="<var>"){  
 rule <- P[[3]][1+C[i,j]%%length(P[[3]])] # determina a regra a ser aplicada  
 derivacao[i] <- sub(S, rule, derivacao[i]) # atualiza derivacao por substituição  
 S <- stringr::str\_match(derivacao[i], "<(.\*?)>") # acha o primeiro <..>  
 S <- paste("<",S[,2],">",sep = "") # novo símbolo  
 }  
 }  
 endCromo[i] <- endCromo[i] + count  
 }  
   
   
 ## Avaliação dos Cromossomos  
   
 training.model <- trainingData[,c("strength")]  
 erroQuadraticoMedio <- rep(0,cromo\_quant)  
 for (k in 1:cromo\_quant){  
 training.data <- c()  
 for (i in 1:length(training.model)){  
 x <- trainingData[i,2]  
 y <- trainingData[i,3]  
 z <- trainingData[i,4]  
 u <- trainingData[i,5]  
 v <- trainingData[i,6]  
 w <- trainingData[i,7]  
 p <- trainingData[i,8]  
 q <- trainingData[i,9]  
 valor <- suppressWarnings(eval(parse(text=derivacao[k])))   
 training.data <- c(training.data,valor)  
 }  
 erroQuadraticoMedio[k] <- mean((training.data - (training.model))^2)   
 }  
  
 ## Torneio de Cromossomos  
   
 sorteados <- c()  
 invalidosNanInf <- which(erroQuadraticoMedio %in% c(NaN,Inf))  
 invalidosVazio <- which(derivacao %in% c(""))  
 invalidos <- c(invalidosNanInf,invalidosVazio,which(duplicated(derivacao)))  
 validos <-c(1:length(derivacao))[which(!c(1:length(derivacao)) %in% invalidos)]  
   
 first\_sorteados <- length(validos)%/%first\_sorteados\_fraction  
 order\_size <- (order\_size\_fraction\*length(validos))%/%first\_sorteados\_fraction  
 ordenados <- validos[order(erroQuadraticoMedio[validos])][1:order\_size]  
 sorteados <- validos[order(erroQuadraticoMedio[validos])][1:first\_sorteados]  
   
 if(length(sorteados)>0)  
 {  
 if(sqrt(erroQuadraticoMedio[sorteados[1]])<raizErroQuadraticoMedioMin)  
 {  
 raizErroQuadraticoMedioMin<-sqrt(erroQuadraticoMedio[sorteados[1]])  
 cromossomo\_vencedor <- C[sorteados[1],]  
 derivacao\_vencedora <- derivacao[sorteados[1]]  
 cat("Achou pto de mínimo: ",raizErroQuadraticoMedioMin," com derivação: ",derivacao\_vencedora,"\n")  
 }  
 }  
   
   
 for (i in (first\_sorteados+1):(order\_size\*prop\_filhos+first\_sorteados+1)){  
 sorteio <- rep(NA,cromo\_quant)  
 pos <- sample(ordenados,2) # sorteio dois crommossomos  
 sorteio[pos] <- pos  
 sorteados <- c(sorteados,which.min(erroQuadraticoMedio[sorteio]))  
 }  
   
 for (i in (order\_size\*prop\_filhos+first\_sorteados+2):cromo\_quant){  
 sorteio <- rep(NA,cromo\_quant)  
 pos <- sample(validos,2) # sorteio dois crommossomos  
 sorteio[pos] <- pos  
 sorteados <- c(sorteados,which.min(erroQuadraticoMedio[sorteio]))  
 }  
 C <- C[sorteados,]  
   
 ## Cruzamento de Cromossomos  
   
 i <- (first\_sorteados+1)  
 while (i < length(sorteados)){  
 # avalia a possibilidade de cruzamento de cada par  
 vai.cruzar <- runif(1)  
 if (vai.cruzar<=p.cruzamento){ # realiza cruzamento  
 pto.cruza <- endCromo[i]  
 if(pto.cruza > endCromo[i+1])  
 {  
 pto.cruza <- endCromo[i+1]  
 }  
 pto.cruza <- sample(pto.cruza-1,1)  
 index\_pai <- i  
 index\_mae <- i+1  
 tmp <- C[index\_pai,(pto.cruza+1):cromo\_size]  
 C[index\_pai,(pto.cruza+1):cromo\_size] <- C[index\_mae,(pto.cruza+1):cromo\_size]   
 C[index\_mae,(pto.cruza+1):cromo\_size] <- tmp  
 }  
 i <- i+2 # pula o próximo cromossomo (estamos olhando os pares)  
 }  
   
 ## Mutação  
   
 vai.mutar <- matrix(runif(cromo\_quant\*cromo\_size),cromo\_quant,cromo\_size)  
 divisa\_prob <- cromo\_quant%/%first\_sorteados\_fraction  
 for (i in (first\_sorteados+1):cromo\_quant){  
 linhaMutar <- vai.mutar[i,]  
 prob<-p.mutacao[i%/%divisa\_prob+1]  
 if(prob\*endCromo[i]<0.3)  
 {  
 prob <- 0.3/endCromo[i]  
 }  
 id <- which(linhaMutar < prob)  
 for (j in 1:length(id)){  
 coluna <- id[j]  
 C[i,coluna] <- sample(max\_gene,1)  
 }  
 }  
  
}

## Achou pto de mínimo: 0.2207719 com derivação: q+v\*exp(u)   
## Geração: 5   
## Achou pto de mínimo: 0.2159772 com derivação: sin(x)

## Encontra as Últimas Derivações  
  
## Gerando Derivação  
S <- "<exp>"  
derivacao <- rep(S,cromo\_quant) # símbolo inicial  
endCromo <- array(0,cromo\_quant)  
for (i in 1:cromo\_quant) {  
 S <- "<exp>"  
 j <- 0  
 count <- 0  
 while (S!="<NA>"){  
 count <- count + 1  
 # evitar loop infinito, para cromossomos mal escolhidos:  
 if (count > cromo\_size){  
 tmp <- stringr::str\_match(derivacao[i], ".\*?<")  
 tmp <- substr(tmp,1,nchar(tmp)-1)  
 if (is.na(tmp)){  
 last.char <- substr(tmp,nchar(derivacao[i]),nchar(derivacao[i]))  
 if (!last.char%in%var){  
 tmp <- substr(derivacao[i],1,nchar(derivacao[i])-1)   
 }  
 } else {  
 last.char <- substr(tmp,nchar(tmp),nchar(tmp))  
 if (!last.char%in%var){   
 # se terminar em operador  
 if (last.char%in%op){  
 tmp <- substr(tmp,1,nchar(tmp)-1)   
 } else if (substr(tmp,nchar(tmp)-1,nchar(tmp))=="^("){  
 tmp <- substr(tmp,1,nchar(tmp)-2)  
 } else if (substr(tmp,nchar(tmp)-5,nchar(tmp))=="asinh("){  
 tmp <- substr(tmp,1,nchar(tmp)-6)  
 } else if (substr(tmp,nchar(tmp)-5,nchar(tmp))=="acosh("){  
 tmp <- substr(tmp,1,nchar(tmp)-6)  
 } else if (substr(tmp,nchar(tmp)-5,nchar(tmp))=="atanh("){  
 tmp <- substr(tmp,1,nchar(tmp)-6)  
 } else if (substr(tmp,nchar(tmp)-1,nchar(tmp))=="h("){  
 tmp <- substr(tmp,1,nchar(tmp)-5)  
 } else if (substr(tmp,nchar(tmp)-4,nchar(tmp))=="sqrt("){  
 tmp <- substr(tmp,1,nchar(tmp)-5)  
 } else if (substr(tmp,nchar(tmp)-4,nchar(tmp))=="asin("){  
 tmp <- substr(tmp,1,nchar(tmp)-5)  
 } else if (substr(tmp,nchar(tmp)-4,nchar(tmp))=="acos("){  
 tmp <- substr(tmp,1,nchar(tmp)-5)  
 } else if (substr(tmp,nchar(tmp)-4,nchar(tmp))=="atan("){  
 tmp <- substr(tmp,1,nchar(tmp)-5)  
 } else if (last.char=="("){  
 tmp <- substr(tmp,1,nchar(tmp)-4)   
 }  
 }  
 last.char <- substr(tmp,nchar(tmp),nchar(tmp))  
 # se terminar em operador  
 if (last.char%in%op){  
 tmp <- substr(tmp,1,nchar(tmp)-1)   
 }  
 endCromo[i] <- -1  
 }  
 # colocar parênteses restantes  
 abrindo <- stringr::str\_count(tmp,"\\(")  
 fechando <- stringr::str\_count(tmp,"\\)")  
 add <- strrep(")",abrindo-fechando)  
 tmp <- paste(tmp,add,sep = "")  
   
 derivacao[i] <- tmp  
 S <- "<NA>"  
 }  
 j <- j%%cromo\_size + 1  
 if (S=="<exp>"){  
 rule <- P[[1]][1+C[i,j]%%length(P[[1]])] # determina a regra a ser aplicada  
 derivacao[i] <- sub(S, rule, derivacao[i]) # atualiza derivacao por substituição  
 S <- stringr::str\_match(derivacao[i], "<(.\*?)>") # acha o primeiro <..>  
 S <- paste("<",S[,2],">",sep = "") # novo símbolo  
 } else if (S=="<op>"){  
 rule <- P[[2]][1+C[i,j]%%length(P[[2]])] # determina a regra a ser aplicada  
 derivacao[i] <- sub(S, rule, derivacao[i]) # atualiza derivacao por substituição  
 S <- stringr::str\_match(derivacao[i], "<(.\*?)>") # acha o primeiro <..>  
 S <- paste("<",S[,2],">",sep = "") # novo símbolo  
 } else if (S=="<var>"){  
 rule <- P[[3]][1+C[i,j]%%length(P[[3]])] # determina a regra a ser aplicada  
 derivacao[i] <- sub(S, rule, derivacao[i]) # atualiza derivacao por substituição  
 S <- stringr::str\_match(derivacao[i], "<(.\*?)>") # acha o primeiro <..>  
 S <- paste("<",S[,2],">",sep = "") # novo símbolo  
 }  
 }  
 endCromo[i] <- endCromo[i] + count  
}

## Escolha do Cromossomo Vencedor

Escolhemos o melhor cromossomo, i.e., o de menor erro quadrático médio.

training.model <- trainingData[,c("strength")]  
erroQuadraticoMedio <- rep(0,cromo\_quant)  
for (k in 1:cromo\_quant){  
 training.data <- c()  
 for (i in 1:length(training.model)){  
 x <- trainingData[i,2]  
 y <- trainingData[i,3]  
 z <- trainingData[i,4]  
 u <- trainingData[i,5]  
 v <- trainingData[i,6]  
 w <- trainingData[i,7]  
 p <- trainingData[i,8]  
 q <- trainingData[i,9]  
 valor <- suppressWarnings(eval(parse(text=derivacao[k])))   
 training.data <- c(training.data,valor)  
 }  
 erroQuadraticoMedio[k] <- mean((training.data - (training.model))^2)   
}  
  
# Cromossomo vencedor:  
vencedor <- which.min(erroQuadraticoMedio)  
erroQuadraticoMedio

## [1] 4.664615e-02 4.874021e-02 5.154777e-02 5.184144e-02 5.211251e-02  
## [6] 5.184144e-02 2.631337e-01 6.490272e-02 5.184144e-02 4.664615e-02  
## [11] 6.851778e+01 2.352580e+00 1.154915e-01 1.464036e-01 1.371803e+00  
## [16] 7.520026e-02 5.184144e-02 1.032677e-01 7.520026e-02 5.154777e-02  
## [21] 4.664615e-02 NaN 1.493517e-01 6.914655e-02 7.974003e-02  
## [26] 5.184144e-02 5.184144e-02 6.490272e-02 5.184144e-02 1.504166e-01  
## [31] 7.520026e-02 5.154777e-02 4.874021e-02 4.664615e-02 6.490272e-02  
## [36] 4.874021e-02 1.880879e-01 5.211251e-02 Inf 1.116635e-01  
## [41] 3.647248e-01 5.211251e-02 6.490272e-02 4.664615e-02 NaN  
## [46] 7.520026e-02 1.870698e+00 5.211251e-02 3.012557e-01 4.747224e-02  
## [51] 7.520026e-02 9.954620e+15 1.780074e-01 NaN 1.274813e+01  
## [56] 4.575196e+00 7.974003e-02 1.341232e-01 1.027748e-01 1.455511e-01  
## [61] 3.647248e-01 1.026730e-01 1.343123e-01 1.974899e+00 NaN  
## [66] 5.001060e-01 1.493517e-01 5.154777e-02 NaN NaN  
## [71] NaN 5.154777e-02 5.154777e-02 7.520026e-02 1.680780e-01  
## [76] 2.467269e-01 1.470602e-01 1.265019e-01 Inf 1.250311e-01  
## [81] Inf 3.020551e-01 1.493517e-01 9.881691e-01 1.739554e-01  
## [86] 9.355575e-02 Inf NaN NaN NaN  
## [91] NaN Inf 1.060571e-01 4.315055e+01 5.154777e-02  
## [96] NaN 2.106236e-01 1.536217e-01 1.546272e-01 1.624715e-01

vencedor

## [1] 1

derivacao\_vencedora <- derivacao[vencedor]  
sqrt(erroQuadraticoMedio[vencedor])

## [1] 0.2159772

derivacao\_vencedora

## [1] "sin(x)"

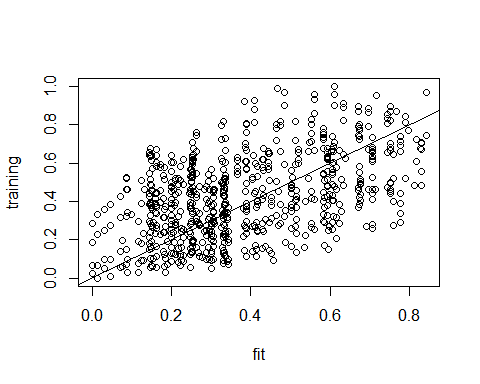
## Comparando com o Próprio training.csv (fit)

Novamente, aqui vamos comparar o quão bem nosso cromossomo vencedor se adequa ao dataset de treinamento.

fit.data <- c()  
for (i in 1:dim(trainingData)[1]){  
 x <- trainingData[i,2]  
 y <- trainingData[i,3]  
 z <- trainingData[i,4]  
 u <- trainingData[i,5]  
 v <- trainingData[i,6]  
 w <- trainingData[i,7]  
 p <- trainingData[i,8]  
 q <- trainingData[i,9]  
 fit.data <- c(fit.data,eval(parse(text=derivacao\_vencedora)))  
}

Graficamente, os dados calculados pelo cromossomo vencedor devem estar próximos da reta, como visto a seguir:

plot(fit.data,training.model,  
 xlab="fit",ylab="training")  
 abline(a=0,b=1)



## Carregamento dos Dados de Teste

testingData <- read.csv(file="testing.csv", header=TRUE, sep=",")

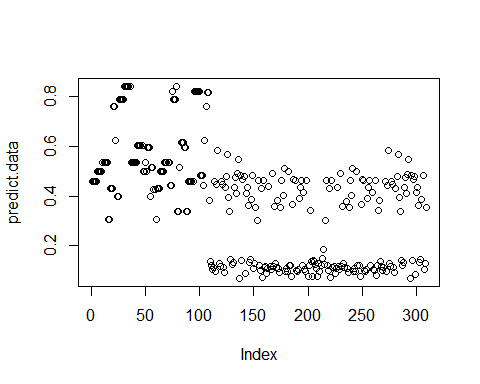
## Predição de Dados

Vamos aplicar o cromossomo vencedor para predizer o dado de saída a partir de entradas advindas de testing.csv.

predict.data <- c()  
for (i in 1:dim(testingData)[1]){  
 x <- testingData[i,2]  
 y <- testingData[i,3]  
 z <- testingData[i,4]  
 u <- testingData[i,5]  
 v <- testingData[i,6]  
 w <- testingData[i,7]  
 p <- testingData[i,8]  
 q <- testingData[i,9]  
 predict.data <- c(predict.data,eval(parse(text=derivacao\_vencedora)))  
}

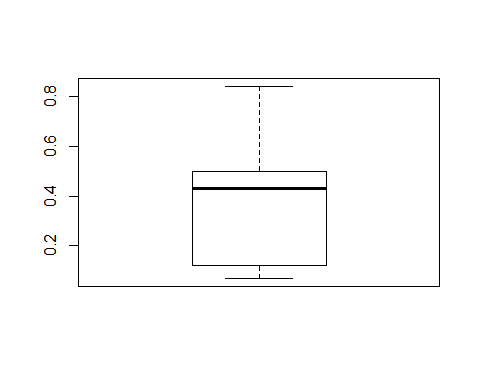
Graficamente, os dados preditos são observados a seguir:

#plot  
plot(predict.data)



Análise boxplot, para visualizar se há outliers.

# boxplot  
boxplot(predict.data)



## Construindo predicted.csv

Construção do arquivo csv a ser submetido para averiguação na plataforma Kaggle.

ID <- testingData[c("ID")]  
result <- data.frame(ID,predict.data)  
colnames(result) <- c('ID', 'strength')  
write.table(result,file="predicted.csv",col.names = c('ID','strength'), row.names = FALSE,sep = ",", quote = FALSE)  
result.data <- read.csv(file="predicted.csv", header=TRUE, sep=",")