Supplementary Information 2
Seeing the Forest for the trees: Assessing genetic offset predictions with Gradient Forest.

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```
library(gradientForest)
```

```
## Loading required package: extendedForest
```

```
## extendedForest 1.6.1
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
cols <- viridisLite::turbo(10)
pchs <- c(0, 1, 2, 15, 23)
options(warn = -1)
```

Run GF model Function

```
runMod <- function(envPop, alFreq, ntree=500, corr.threshold=0.5){
  # envPop is a dataframe of environments
  # alfreq is the pattern of allele frequency across that environment
  alFreq1 <- data.frame(alFreq)
  #names(alFreq1) <- "alFreq1"
  gfMod <- gradientForest(data=data.frame(envPop, alFreq1),
                          predictor.vars=names(envPop),
                          response.vars=names(alFreq1),
                          corr.threshold=corr.threshold,
                          ntree=ntree,
                          trace=F)

  CI <- cumimp(gfMod, "envPop")
  return(list(CI=CI, IsZero=sum(CI$y)==0, R2=gfMod$result))
}

runMod_multi <- function(envPop, alFreq, ntree=500, corr.threshold=0.5){
  # envPop is a dataframe of environments
  # alfreq is the pattern of allele frequency across that environment
  alFreq1 <- data.frame(alFreq)
  #save the different environment names
  pred.vars<-names(envPop)
```

```
  resp.vars <- names(alFreq1)
  gfMod <- gradientForest(data=data.frame(envPop, alFreq1),
                          predictor.vars=pred.vars,
                          response.vars=resp.vars,
                          corr.threshold=corr.threshold,
                          ntree=ntree,
                          trace=F)
  #create a list of CI for each environment and an IsZero entry for each one as well
  CI_env <- list()
  CI_locus_env <- list()
  IsZero <- list()
  for(i in 1:length(pred.vars)){
    CI_env[[pred.vars[i]]]<-cumimp(gfMod, pred.vars[i], type="Overall")
    IsZero[[pred.vars[i]]]<-sum(CI_env[[pred.vars[i]]]$y)==0
    CI_locus_env[[pred.vars[i]]]<-cumimp(gfMod, pred.vars[i], type="Species")
  }
  #save as a list of lists, including the overall CI (gfMod$Y)
  return(list(CI_env=CI_env, CI=gfMod$Y, IsZero=IsZero, R2=gfMod$result, CI_locus_env))
}
```

Output 1: *Cumulative importance prediction as a function of the environment.* We found that in sparse sampling (see below), this sometimes output all 0's

Output 2: *R2* This is the proportion of variance in the allele frequency that is explained by the model.

First comparison: few vs. many populations sampled along a cline

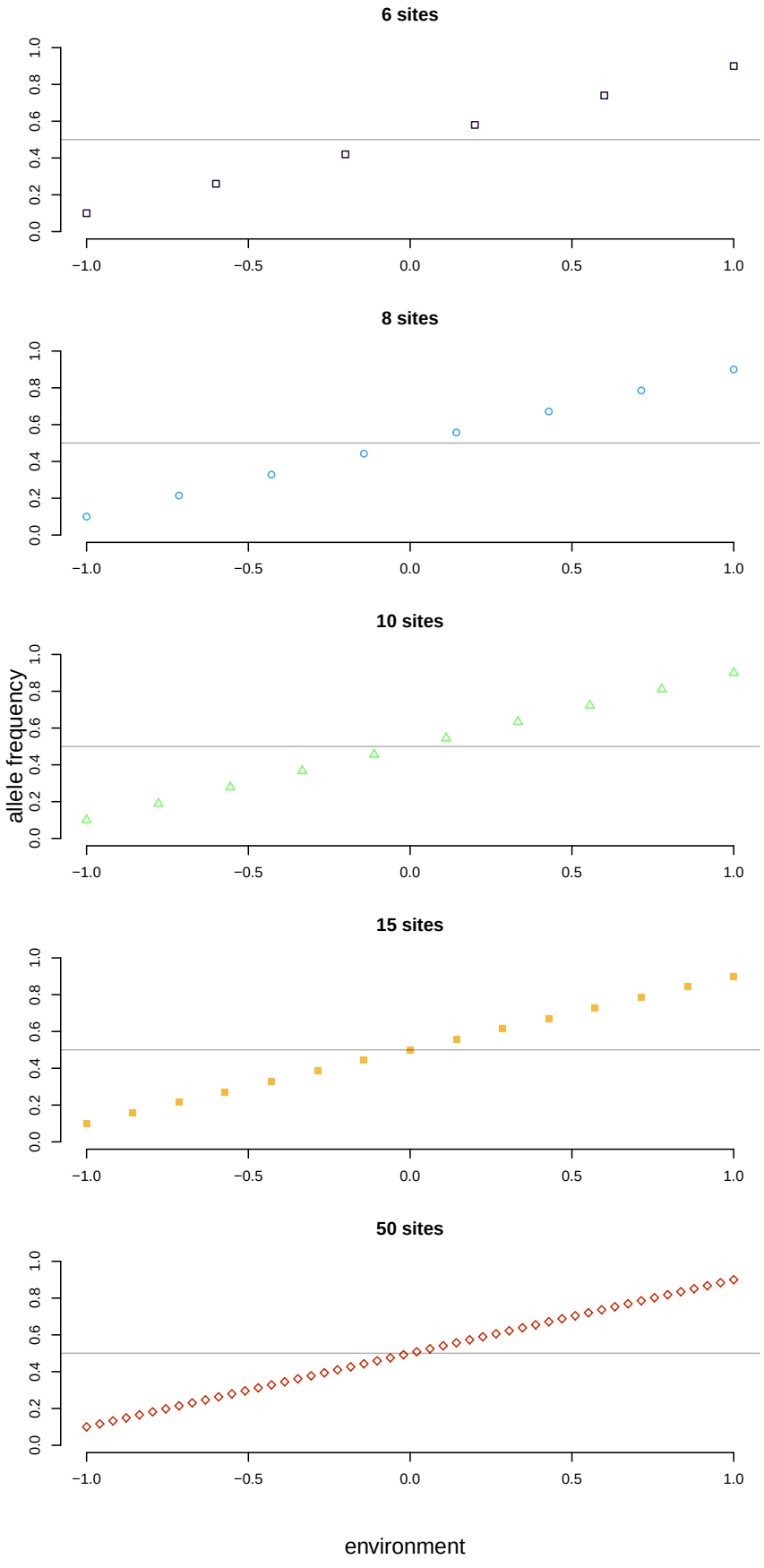
First take home message: GF outputs depends on sampling scheme

Sometimes in pop gen we only have a few populations sampled. Here we explore the consequences of sampling design on GF outputs. All sampling schemes were based on the same relationship betweeen allele frequency and the environment. We sampled 6, 8, 10, 15, or 50 populations across an environmental gradient.

```
## 'data.frame':    6 obs. of  1 variable:
##  $ envPop: num  -1 -0.6 -0.2 0.2 0.6 1
```

The following plot shows the relationship between allele frequency and the environment is the same for the different sampling schemes:

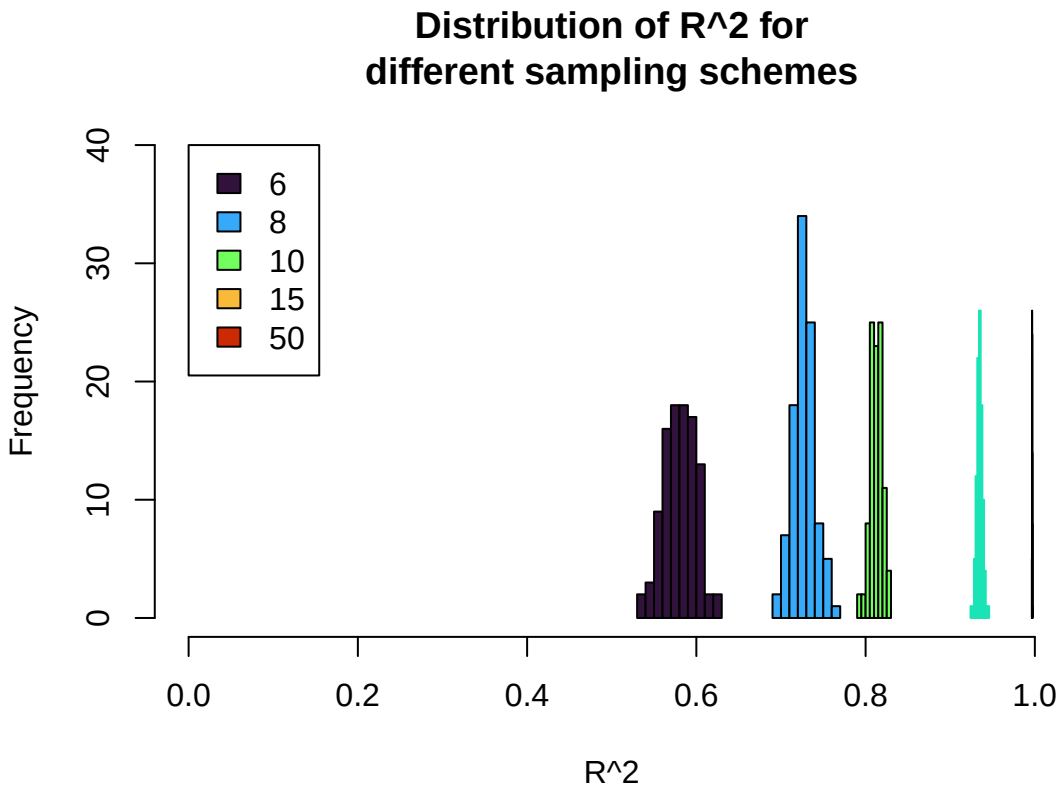
```
par(mar=c(3,1,3,0.4), mfrow=c(5,1), oma=c(4,4,0,0))
plot(envPop_short$envPop, alFreq_L1_short, xlab="environment", ylab="allele freq.", col=cols[1], pch=pch
abline(h=0.5, col=rgb(0,0,0,0.3))
plot(envPop_med$envPop, alFreq_L1_med, xlab="environment", ylab="allele freq.", col=cols[3], pch=pchs[2]
abline(h=0.5, col=rgb(0,0,0,0.3))
plot(envPop_med2$envPop, alFreq_L1_med2, xlab="environment", ylab="allele freq.", col=cols[5], pch=pchs
abline(h=0.5, col=rgb(0,0,0,0.3))
plot(envPop_long$envPop, alFreq_L1_long, xlab="environment", ylab="allele freq.", col=cols[7], pch=pchs
abline(h=0.5, col=rgb(0,0,0,0.3))
plot(envPop_long2$envPop, alFreq_L1_long2, xlab="environment", ylab="allele freq.", col=cols[9], pch=pch
abline(h=0.5, col=rgb(0,0,0,0.3))
mtext("environment", side=1, outer=TRUE, line=2)
mtext("allele frequency", side=2, outer=TRUE, line=1)
```


6 sites
8 sites
10 sites
15 sites
50 sites
allele frequency
environment
0.0
0.2
0.4
0.6
0.8
1.0
−1.0
−0.5
0.0
0.5
1.0

For each sampling scheme, we conducted 100 replicate simulations of GF with 500 trees.

The R^2 value from GF is typically interpreted to be similar to a regression. However, it increases with the number of sites sampled along the gradient, as shown in the next graph:

```
# Histogram of R2 values
hist(CI_short_R2, xlim=c(0,1), ylim=c(0,40), col=cols[1], xlab="R^2", main="Distribution of R^2 for\ndi
hist(CI_med_R2, xlim=c(0,1), col=cols[3], add=TRUE )
hist(CI_med2_R2, xlim=c(0,1), col=cols[5], add=TRUE)
hist(CI_long_R2, xlim=c(0,1), col=cols[7], add=TRUE , border = cols[4])
hist(CI_long2_R2, xlim=c(0,1), col=cols[9], add=TRUE )

legend(0, 40, fill=cols[seq(1,9, by=2)], legend = samp_sizes)
```

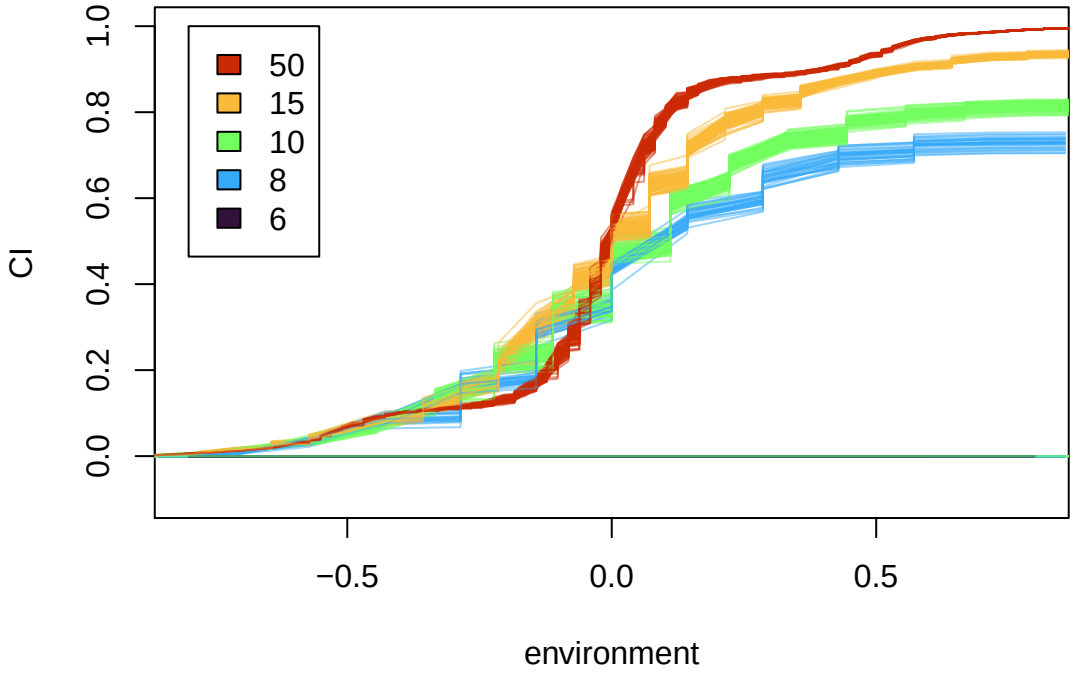


The above two graphs illustrate that the R^2 value from a gradient forests output is not similar to a R^2 value from a regression (which should be the same in all 5 of these cases).

The next figure illustrates that the shape and height of the cumulative importance curve is also sensitive to the sampling design:

```
# CI vs. env
sequ <- seq(1, length(CI_short), by=3)
plot(CI_short[[1]]$x, CI_short[[1]]$y, type="l", col=cols[1], ylim=c(-0.1,1), ylab="CI", xlab="environm
for (i in sequ){
  points(CI_short[[i]]$x, CI_short[[i]]$y, type="l", col=adjustcolor(cols[1], 0.5))
  points(CI_med[[i]]$x, CI_med[[i]]$y, type="l", col=adjustcolor(cols[3], 0.5))
  points(CI_med2[[i]]$x, CI_med2[[i]]$y, type="l", col=adjustcolor(cols[5], 0.5))
  points(CI_long[[i]]$x, CI_long[[i]]$y, type="l", col=adjustcolor(cols[7], 0.5))
  points(CI_long2[[i]]$x, CI_long2[[i]]$y, type="l", col=adjustcolor(cols[9], 0.5))
}
```

```
legend(-0.8, 1, fill=rev(cols[seq(1,9, by=2)]), legend = rev(samp_sizes))
```



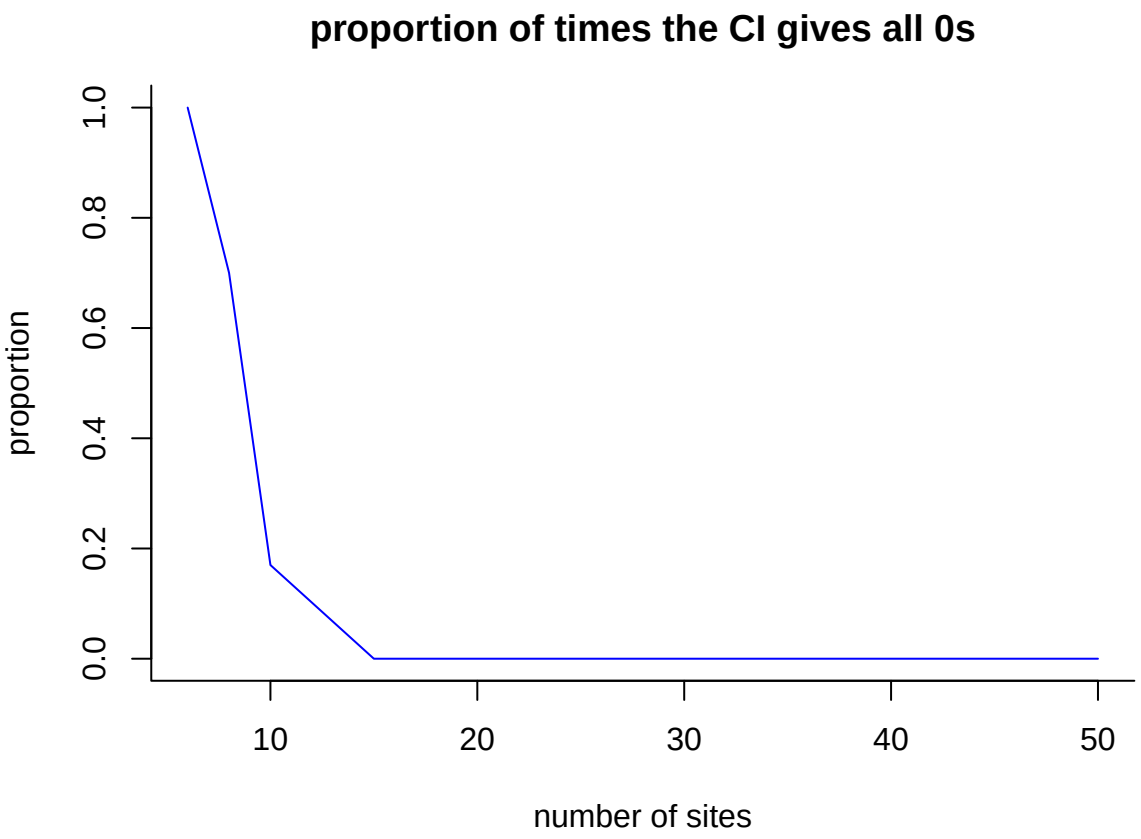
The above plot highlights some interesting behavior.

First, the height of the CI curve depends on the sampling scheme. Sampling more sites along the gradient results in a higher overall CI.

Second, the shape of the CI curve depends on the sampling scheme. Sampling more sites along the gradient results in more sigmodal shaped CI function near the intermediate environment. For example, the actual allele frequency turnover is the same across all 5 sampling schemes. But the CI function makes it look like the sampling scheme with 50 samples is very steep near `environment=0` compared to the other schemes. This highlights issues with interpreting GF outputs as allele frequency change along a gradient and calculating GF offsets from that.

Third, there is an behavior with the distribution of CI values, but it's hard to see. With the sparsest sampling scheme (6 sites), the CI is always 0. With less sparse sampling schemes (e.g., 8 or 10 sites), the CI is *sometimes* 0, but it's hard to see because all the lines overlap at the horizontal line at y=0. We will visualized that in the next plot:

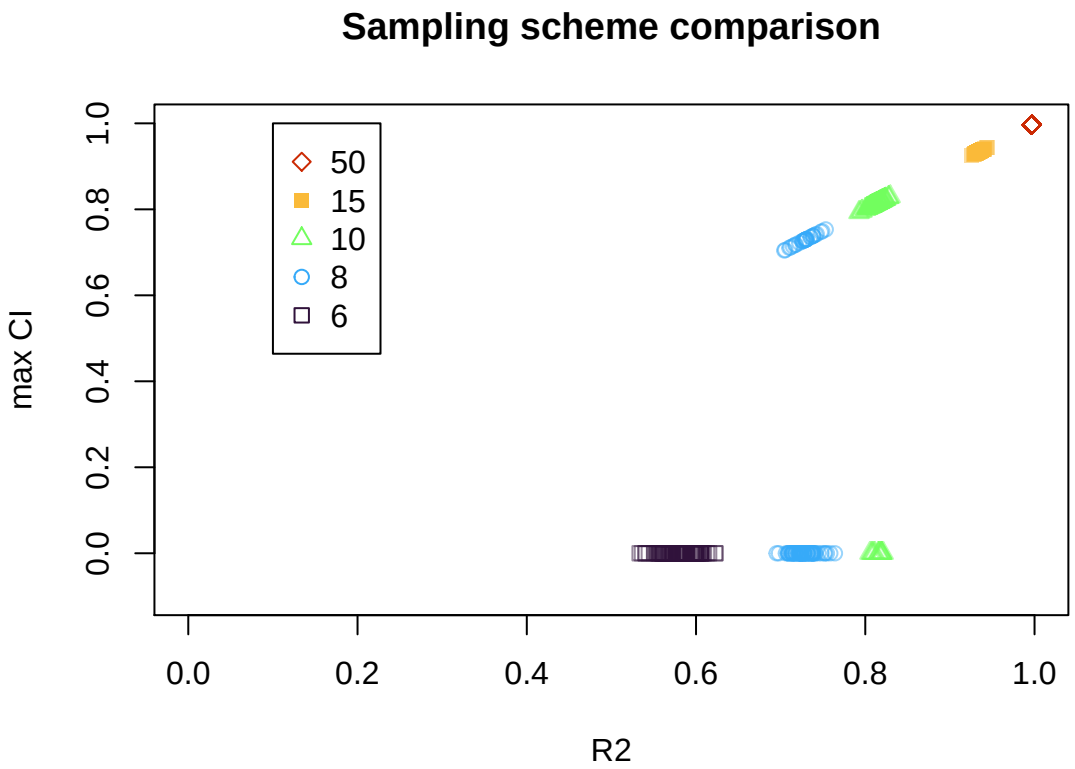
```
# Proportion of 0s
par(mar=c(4,4,3,0.4), mfrow=c(1,1), oma=c(0,0,0,0), bty="l")
plot(samp_sizes, c(CI_short_prop0, CI_med_prop0, CI_med2_prop0, CI_long_prop0, CI_long2_prop0), xlab="nu
     type="l", col="blue", ylab= "proportion",
     main="proportion of times the CI gives all 0s", ylim=c(0,1))
```



The above plot shows that sometimes GF output all 0s for the cumulative importance function, and that the proportion of times this happened was higher for sparser sampling schemes. We did not find increasing the number of trees changed these proportions.

Another way to visualize it as the distribution of R^2 vs. maximum CI for each replicate simulation:

```
sequ <- seq(1, length(CI_short), by=3)
plot(CI_short[[3]], max(CI_short[[1]]$y), col=adjustcolor(cols[1], 0.5), ylim=c(-0.1,1), xlab="R2", ylab
for (i in sequ){
  points(CI_short[[i+2]], max(CI_short[[i]]$y),  col=adjustcolor(cols[1], 0.5), pch=pchs[1])
  points(CI_med[[i+2]], max(CI_med[[i]]$y),  col=adjustcolor(cols[3], 0.5), pch=pchs[2])
  points(CI_med2[[i+2]], max(CI_med2[[i]]$y),  col=adjustcolor(cols[5], 0.5), pch=pchs[3])
  points(CI_long[[i+2]], max(CI_long[[i]]$y), col=adjustcolor(cols[7], 0.5), pch=pchs[4])
  points(CI_long2[[i+2]],max(CI_long2[[i]]$y),  col=adjustcolor(cols[9], 0.5), pch=pchs[5])
}

legend(0.1, 1, col =rev(cols[seq(1,9, by=2)]), legend = rev(samp_sizes), pch=rev(pchs))
```



Maybe this weird behavior is GF's way of weighting individual loci that it has more confidence in (e.g, denser sampling). However, it does raise issues for the way that populations are sampled unequally along gradients in nature. In our simulations we had equal number of sites sampled along each environmental gradient (e.g., a 10 x 10 matrix with latitudinal gradient and a longitudinal gradient), so the effect of unequal sampling along different gradients warrants further investigation.

Second comparison: A steep cline vs. less steep cline vs. a reverse cline

Second take home message: GF outputs are the same regardless of slope of cline

For this comparison, we assumed equal sampling (15 sites along the cline) and manipulated the pattern of allele frequency along the cline.

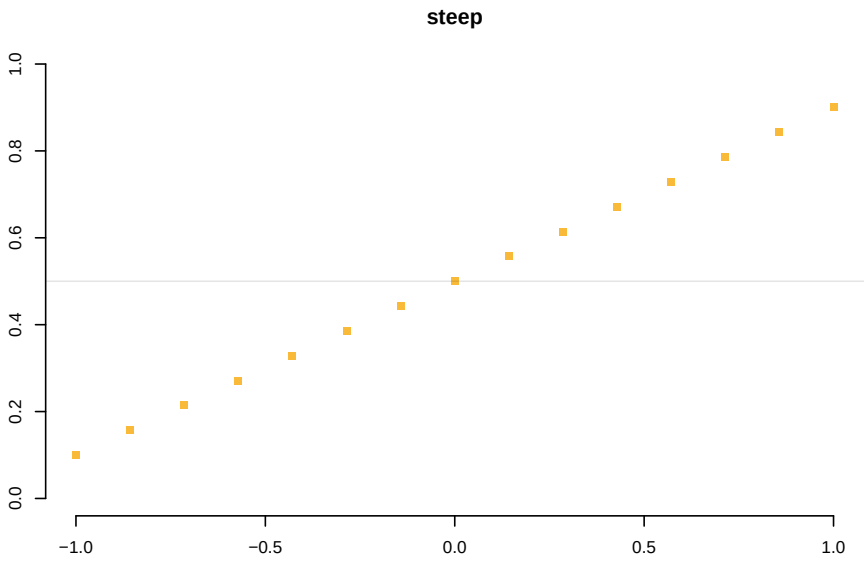
Code used from first section:

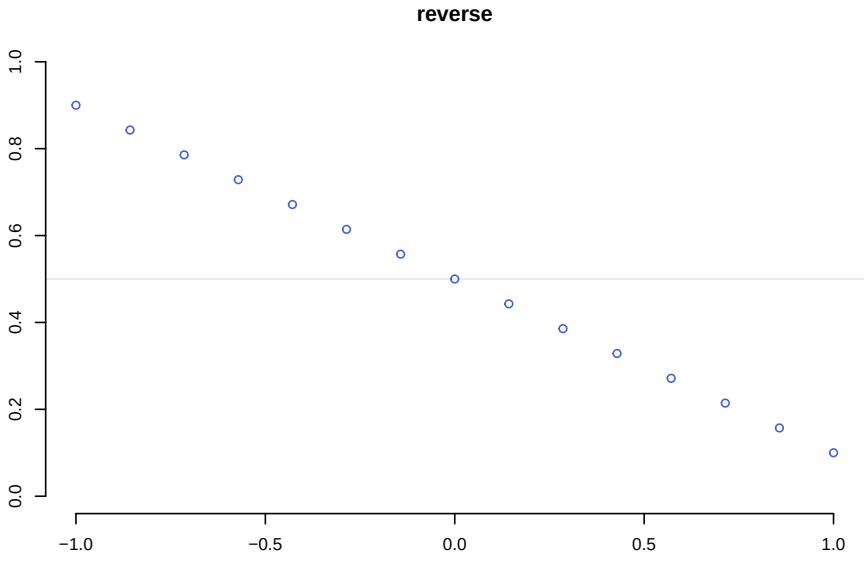
```
envPop_long <- data.frame(envPop=seq(-1,1,length.out=15)) #create env
#create four allele frequency gradients of varying steepness
alFreq_L1_long<-seq(0.1,0.9,length.out=15)
CI_long <- replicate(100,runMod(envPop_long, alFreq_L1_long))
(CI_long_R2 <- (unlist(CI_long[seq(3,length(CI_long), by=3)])))
```

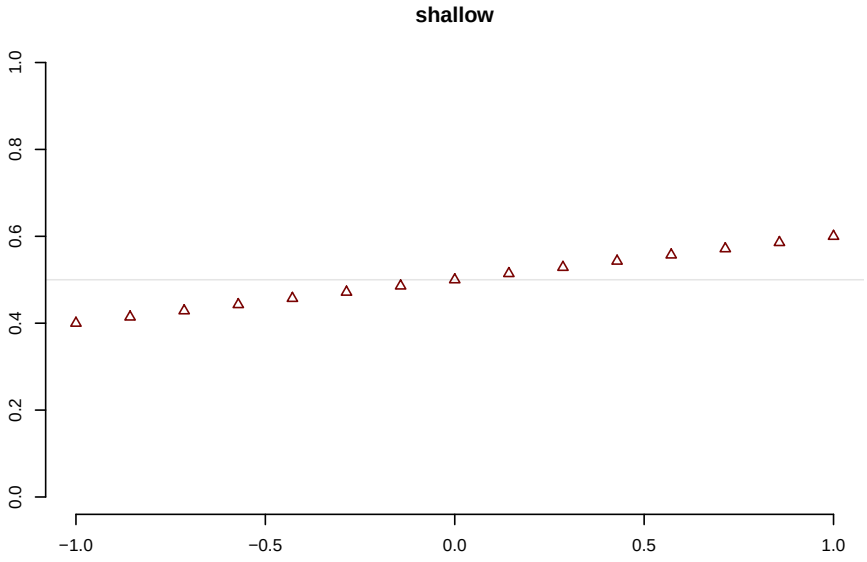
```
alFreq_L1_rev<- rev(alFreq_L1_long)
CI_rev <- replicate(100,runMod(envPop_long, alFreq_L1_rev))
CI_rev_R2 <- (unlist(CI_rev[seq(3,length(CI_rev), by=3)]))

alFreq_L1_shallow<-seq(0.4,0.6,length.out=15)
CI_shallow <- replicate(100,runMod(envPop_long, alFreq_L1_shallow))
CI_shallow_R2 <- (unlist(CI_shallow[seq(3,length(CI_shallow), by=3)]))
```

As you will see below, all of these clines have the same R^2 value and same max CI (e.g., the same proportion of variance explained by the model). The results highlight, however, why the GF output cannot be called a "vulnerability": the output is agnostic to the direction of allele frequency change across the gradient.

```
par(mar=c(3,1,3,0.4), mfrow=c(3,1), oma=c(4,4,0,0))
plot(envPop_long$envPop, alFreq_L1_long, xlab="environment", ylab="allele freq.", col=cols[7], pch=pchs
abline(h=0.5, col=rgb(0,0,0,0.1))
plot(envPop_long$envPop, alFreq_L1_rev, xlab="environment", ylab="allele freq.", col=cols[2], pch=pchs[1
abline(h=0.5, col=rgb(0,0,0,0.1))
plot(envPop_long$envPop, alFreq_L1_shallow, xlab="environment", ylab="allele freq.", col=cols[10], pch=p
abline(h=0.5, col=rgb(0,0,0,0.1))
```

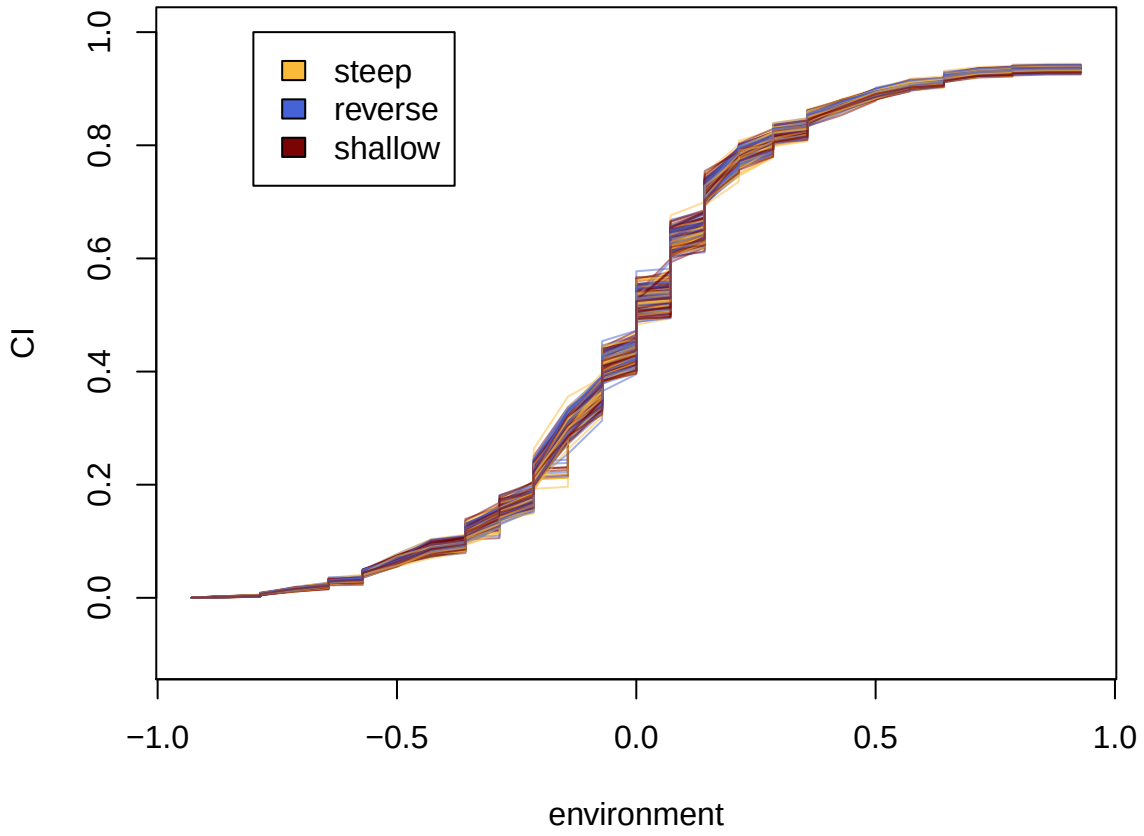

steep
1.0
0.8
0.6
0.4
0.2
0.0
−1.0
−0.5
0.0
0.5
1.0


reverse
1.0
0.8
0.6
0.4
0.2
0.0
−1.0
−0.5
0.0
0.5
1.0


shallow
1.0
0.8
0.6
0.4
0.2
0.0
−1.0
−0.5
0.0
0.5
1.0

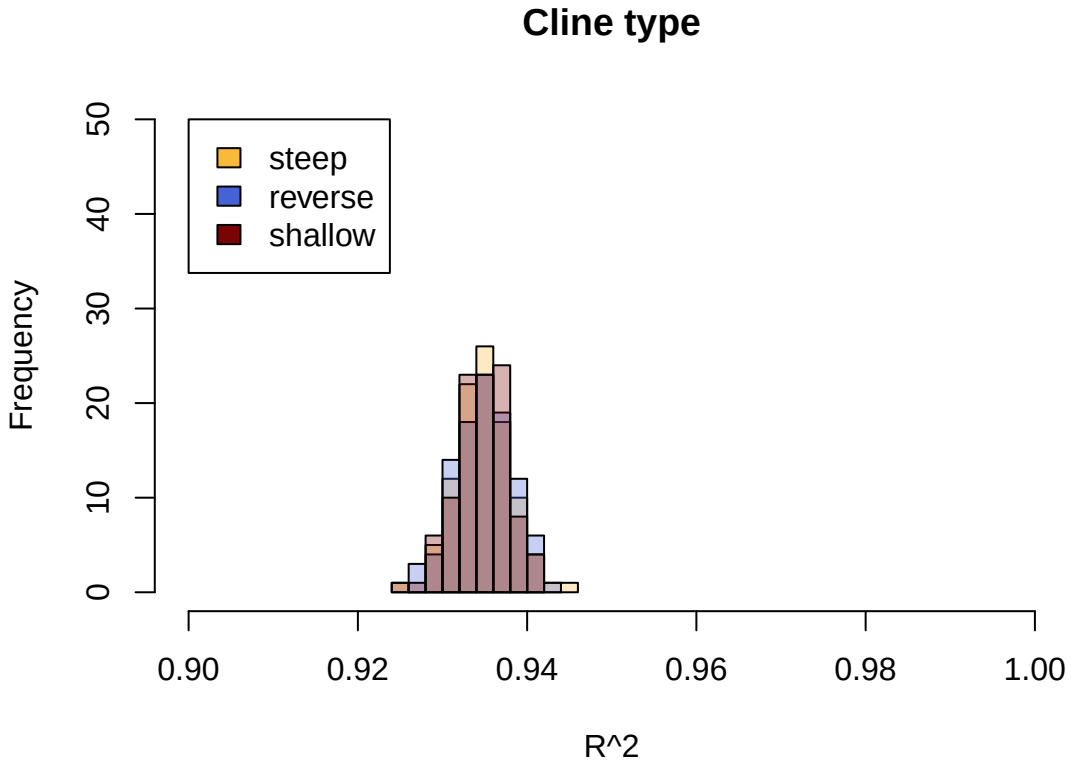
All these relationships give essentially the same CI curve:

```
par(mfrow=c(1,1), mar=c(4,4,1,1))
# CI vs. env
sequ <- seq(1, length(CI_short), by=3)
plot(CI_long[[1]]$x, CI_long[[1]]$y, type="l", col=cols[7], ylim=c(-0.1,1), ylab="CI", xlab="environmen
for (i in sequ){
  points(CI_long[[i]]$x, CI_long[[i]]$y, type="l", col=adjustcolor(cols[7], 0.5))
  points(CI_rev[[i]]$x, CI_rev[[i]]$y, type="l", col=adjustcolor(cols[2], 0.5))
  points(CI_shallow[[i]]$x, CI_shallow[[i]]$y, type="l", col=adjustcolor(cols[10], 0.5))
}

legend(-0.8, 1, fill=cols[c(7,2,10)], legend = c("steep", "reverse", "shallow"))
```



And same R^2 function:

```
hist(CI_long_R2, xlim=c(0.9,1), col=adjustcolor(cols[7], 0.3), ylim=c(0,50), main="Cline type", xlab="R
hist(CI_rev_R2, xlim=c(0,1), col=adjustcolor(cols[2], 0.3), add=TRUE )
hist(CI_shallow_R2, xlim=c(0,1), col=adjustcolor(cols[10], 0.3), add=TRUE )
legend(0.9, 50, fill=cols[c(7,2,10)], legend = c("steep", "reverse", "shallow"))
```



This section highlighted out the GF output is driven by the proportion of variance the model can explain in the data, which is the same for all shapes of allele frequency curves. To further illustrate this, we will consider in the next section changing the shape of the relationship between a.f. and environment.

Third comparison: Non-monotonic relationships with allele frequency

Third take home message: GF outputs are the same regardless of the magnitude of the non-monotonic relationship with allele frequency, when they have the same shape

For this comparison, we assumed equal sampling (15 sites along the cline) and manipulated the pattern of allele frequency along the cline.

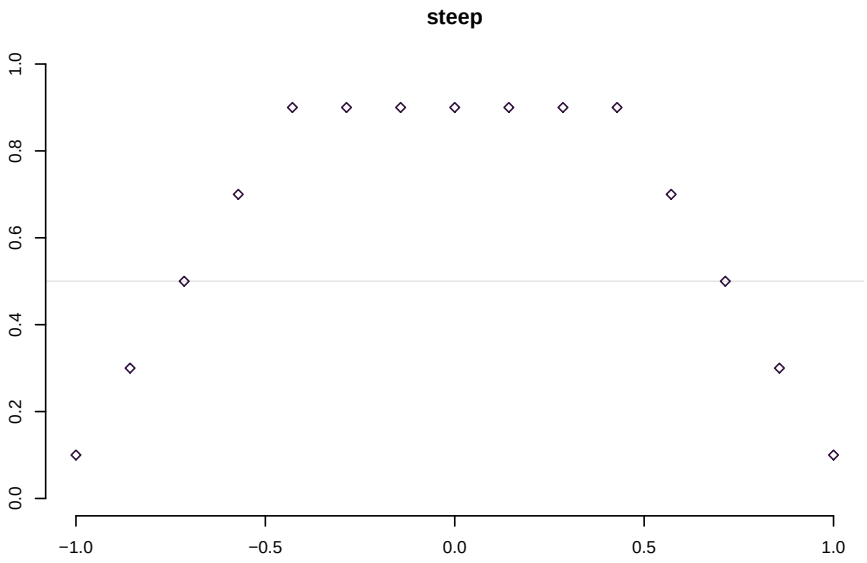
```
envPop_long <- data.frame(envPop=seq(-1,1,length.out=15)) #create env
alFreq_nonMon<- c(seq(0.1,0.9, length.out=5), rep(0.9,5), seq(0.9,0.1, length.out=5))
CI_nonMon <- replicate(100,runMod(envPop_long, alFreq_nonMon))
CI_nonMon_R2 <- (unlist(CI_nonMon[seq(3,length(CI_nonMon), by=3)]))

alFreq_nonMon_rev<- 1-alFreq_nonMon
CI_nonMon_rev <- replicate(100,runMod(envPop_long, alFreq_nonMon_rev))
CI_nonMon_rev_R2 <- (unlist(CI_nonMon_rev[seq(3,length(CI_nonMon_rev), by=3)]))

alFreq_nonMon_shallow<- c(seq(0.4,0.6, length.out=5), rep(0.6,5), seq(0.6,0.4, length.out=5))
CI_nonMon_shallow <- replicate(100,runMod(envPop_long, alFreq_nonMon_shallow))
CI_nonMon_shallow_R2 <- (unlist(CI_nonMon_shallow[seq(3,length(CI_nonMon_shallow), by=3)]))
```

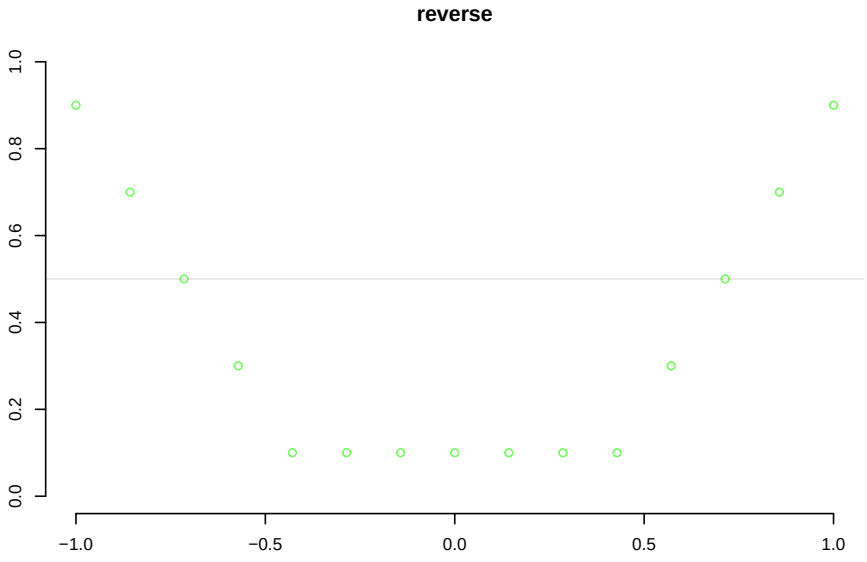
The shallow non-monotonic case gives the following error in GF, but still runs and produces outputs:

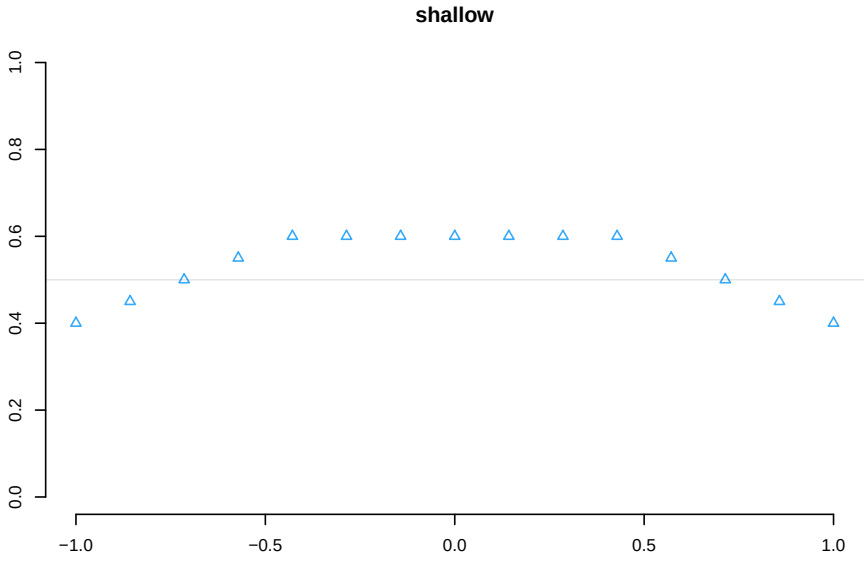
```
Warning in randomForest.default(m, y, ...): The response has five or fewer
unique values. Are you sure you want to do regression?
```

Plot the patterns of allele frequency as a function of the environment:

```
par(mar=c(3,1,3,0.4), mfrow=c(3,1), oma=c(4,4,0,0))
plot(envPop_long$envPop, alFreq_nonMon, xlab="environment", ylab="allele freq.", col=cols[1], pch=pchs[5
abline(h=0.5, col=rgb(0,0,0,0.1))
plot(envPop_long$envPop, alFreq_nonMon_rev, xlab="environment", ylab="allele freq.", col=cols[5], pch=pc
abline(h=0.5, col=rgb(0,0,0,0.1))
plot(envPop_long$envPop, alFreq_nonMon_shallow, xlab="environment", ylab="allele freq.", col=cols[3], pc
abline(h=0.5, col=rgb(0,0,0,0.1))
```

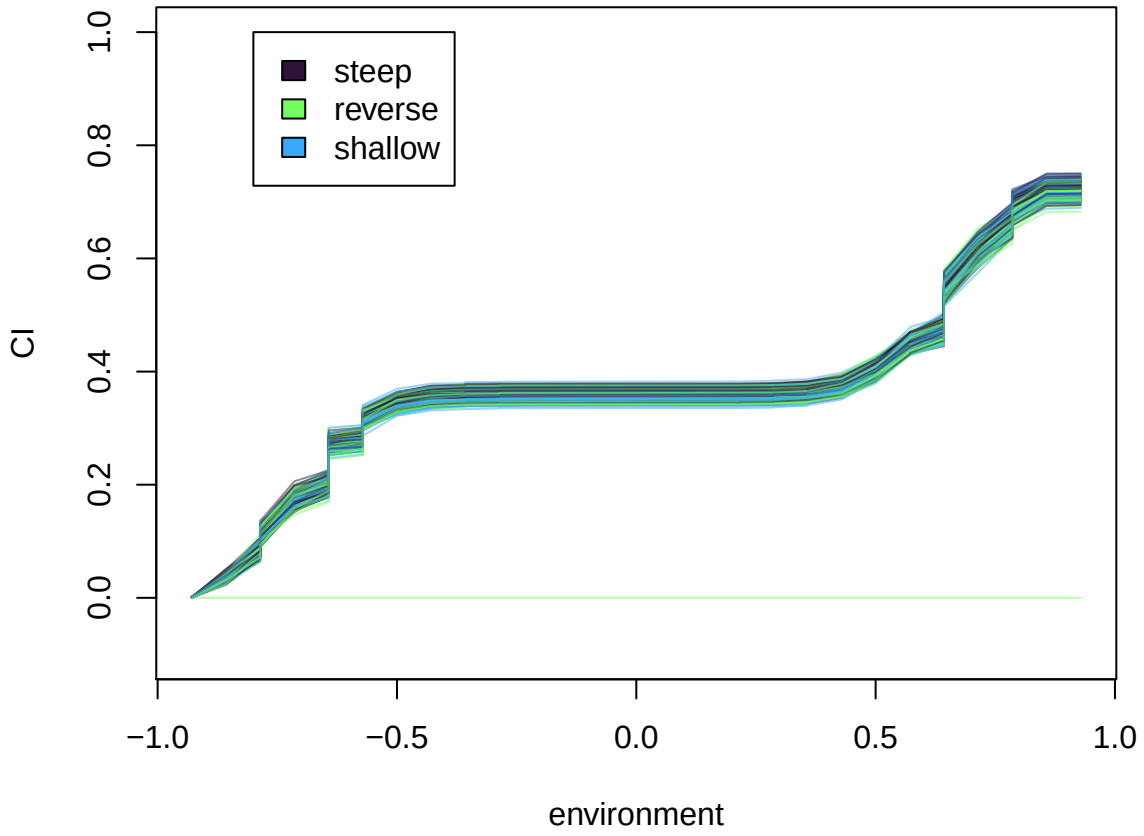

steep
1.0
0.8
0.6
0.4
0.2
0.0
−1.0
−0.5
0.0
0.5
1.0


reverse
1.0
0.8
0.6
0.4
0.2
0.0
−1.0
−0.5
0.0
0.5
1.0


shallow
1.0
0.8
0.6
0.4
0.2
0.0
−1.0
−0.5
0.0
0.5
1.0

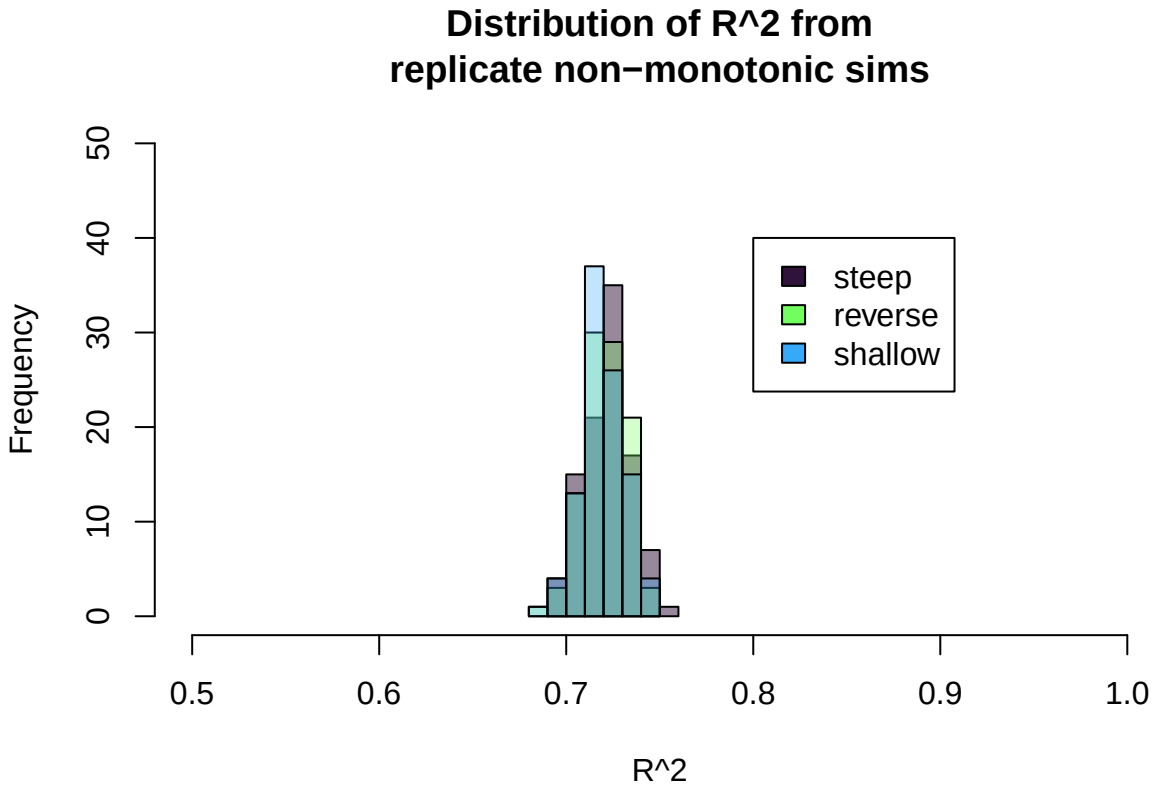
All these relationships give essentially the same CI curve:

```
par(mfrow=c(1,1), mar=c(4,4,1,1))
# CI vs. env
sequ <- seq(1, length(CI_nonMon), by=3)
plot(CI_nonMon[[1]]$x, CI_nonMon[[1]]$y, type="l", col=cols[7], ylim=c(-0.1,1), ylab="CI", xlab="enviro
for (i in sequ){
  points(CI_nonMon[[i]]$x, CI_nonMon[[i]]$y, type="l", col=adjustcolor(cols[1], 0.5))
  points(CI_nonMon_rev[[i]]$x, CI_nonMon_rev[[i]]$y, type="l", col=adjustcolor(cols[5], 0.5))
  points(CI_nonMon_shallow[[i]]$x, CI_nonMon_shallow[[i]]$y, type="l", col=adjustcolor(cols[3], 0.5))
}

legend(-0.8, 1, fill=cols[c(1,5,3)], legend = c("steep", "reverse", "shallow"))
```



And same R^2 function:

```
hist(CI_nonMon_R2, xlim=c(0.5,1), col=adjustcolor(cols[1], 0.5), ylim=c(0,50), xlab="R^2", main=
      "Distribution of R^2 from\nreplicate non-monotonic sims")
hist(CI_nonMon_rev_R2, xlim=c(0,1), col=adjustcolor(cols[5], 0.3), add=TRUE )
hist(CI_nonMon_shallow_R2, xlim=c(0,1), col=adjustcolor(cols[3], 0.3), add=TRUE )
legend(0.8, 40, fill=cols[c(1,5,3)], legend = c("steep", "reverse", "shallow"))
```



The above two sections illustrate that the GF CI curves do not correspond the amount of turnover in allele frequency. If we consider turnover in allele frequency in the mathematical sense, it is the amount of change in allele frequency per unit change in the environment. In the examples in the above two sections, steeper clines have greater amount of change in allele frequency per unit change in the environment than shallower clines. Hence, if the maximum CI of an allele reflected the turnover, alleles with steeper clines should have greater maximum CI. However, the above two sections show this is not the case when comparing alleles with similar patterns but different amounts of frequency change across the gradient. The maximum CI is, instead, driven by how much variation in allele frequency the gradient-forests model explains, which is the same for steep, shallow, and reverse clines of the same general pattern across the environmental gradient.

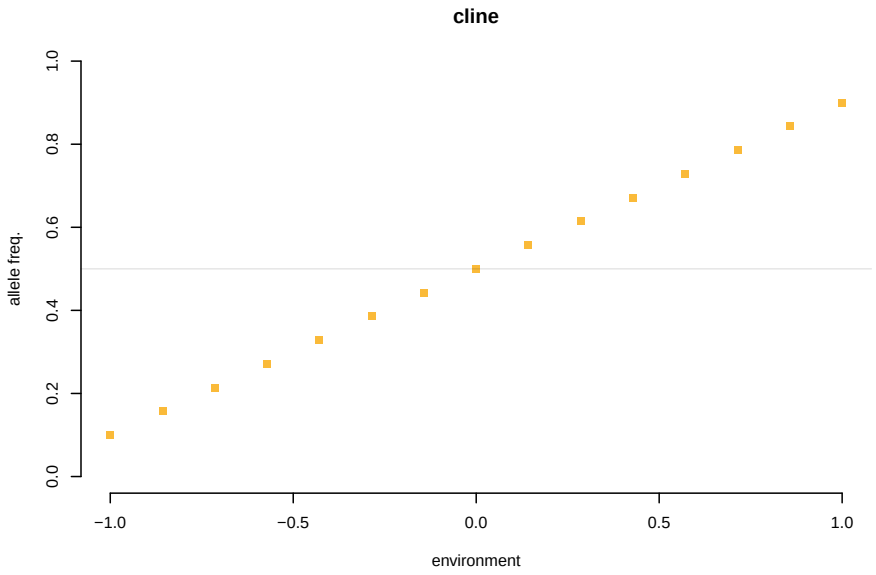
The non-monotonic clines in the above example all have the same allele frequency at the extremes of the environmental variable. If the genetic basis of adaptation was monogenic (e.g. caused by a single allele), and fitness followed the same non-monotonic pattern (e.g., individuals with the focal allele had highest fitness in the intermediate values of the environment, and individuals with the alternate allele had highest fitness in the extreme values of the environment), then this conceptual example illustrates issues with calling the GF-offset a genetic offset or genomic vulnerability. In this hypothetical case, individuals of the same genotype would have the same fitness at extreme values of the environment, and hence the genetic offset should be 0. But GF would predict a maximum offset (e.g. the vertical distance in the CI evaluated an environment of -1 and the CI evaluated an environment of +1 in the previous plot).

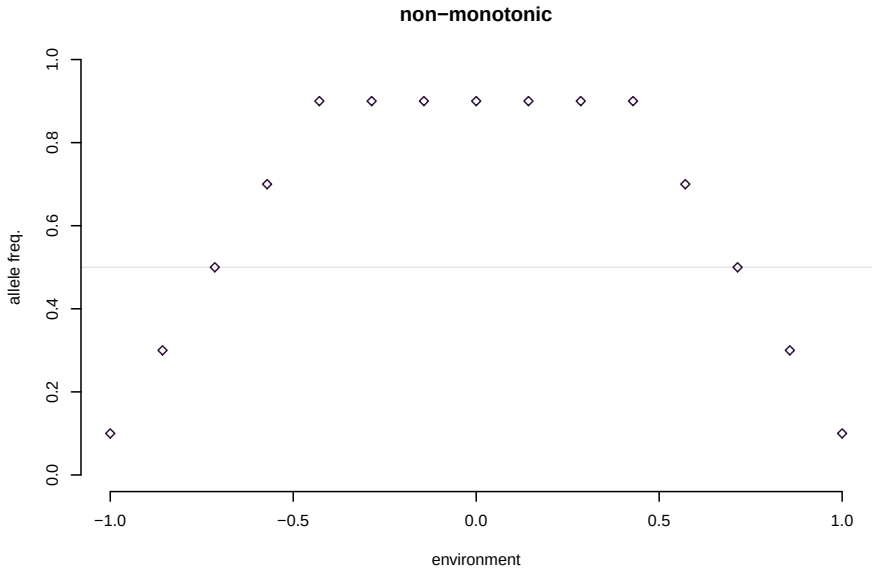
Fourth comparison: A steep cline vs. a non-monotonic cline

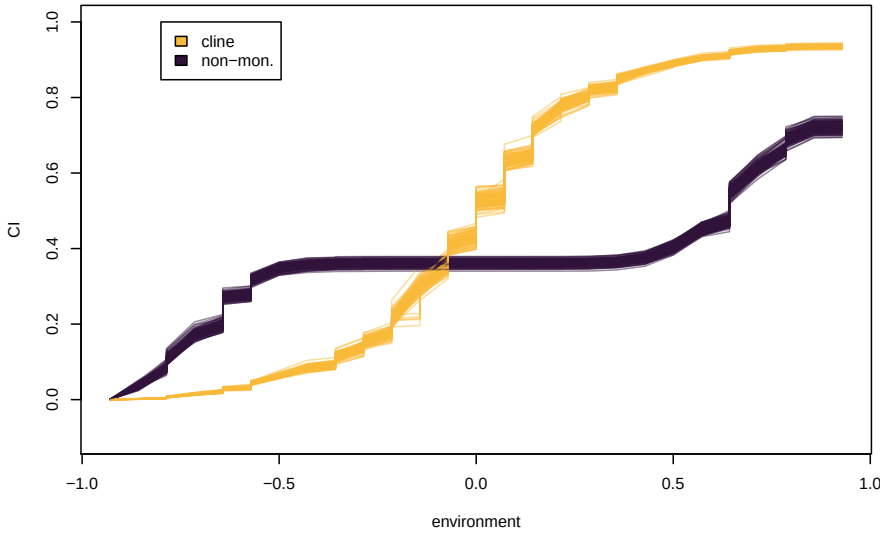
Fourth take home message: The shape of the CI function reflects the pattern of turnover in allele frequency

For this comparison, we assumed equal sampling (15 sites along the cline). We compared the linear cline from the second comparison with a non-monotonic clines from the third comparison.

```r
par(mar=c(4,4,3,0.4), mfrow=c(3,1))
plot(envPop_long$envPop, alFreq_L1_long, xlab="environment", ylab="allele freq.", col=cols[7], pch=pchs
abline(h=0.5, col=rgb(0,0,0,0.1))

plot(envPop_long$envPop, alFreq_nonMon, xlab="environment", ylab="allele freq.", col=cols[1], pch=pchs[5
abline(h=0.5, col=rgb(0,0,0,0.1))

# CI vs. env
sequ <- seq(1, length(CI_nonMon), by=3)
plot(CI_nonMon[[1]]$x, CI_nonMon[[1]]$y, type="l", col=cols[1], ylim=c(-0.1,1), ylab="CI", xlab="enviro
for (i in sequ){
  points(CI_nonMon[[i]]$x, CI_nonMon[[i]]$y, type="l", col=adjustcolor(cols[1], 0.5))
  points(CI_long[[i]]$x, CI_long[[i]]$y, type="l", col=adjustcolor(cols[7], 0.5))
}

legend(-0.8, 1, fill=cols[c(7,1)], legend = c("cline", "non-mon."))
```


cline
allele freq.
0.0
0.2
0.4
0.6
0.8
1.0
−1.0
−0.5
0.0
0.5
1.0
environment


non−monotonic
allele freq.
0.0
0.2
0.4
0.6
0.8
1.0
−1.0
−0.5
0.0
0.5
1.0
environment


cline
non−mon.
CI
0.0
0.2
0.4
0.6
0.8
1.0
−1.0
−0.5
0.0
0.5
1.0
environment

The non-monotonic pattern has a flat CI in the intermediate values of the environment that have very little turnover. However, the clinal pattern, which is linear, results in a non-linear CI function. This illustrates a potential problem with interpreting the GF-offset as a genetic offset. For the cline at extreme values of the environment, a very small GF-offset would be calculated, and at intermediate values of the environment, a very large GF-offset would be calculated. However, the actual "genetic offset" (in terms of allele frequency change per unit environment change) is the same at all values of the environment.

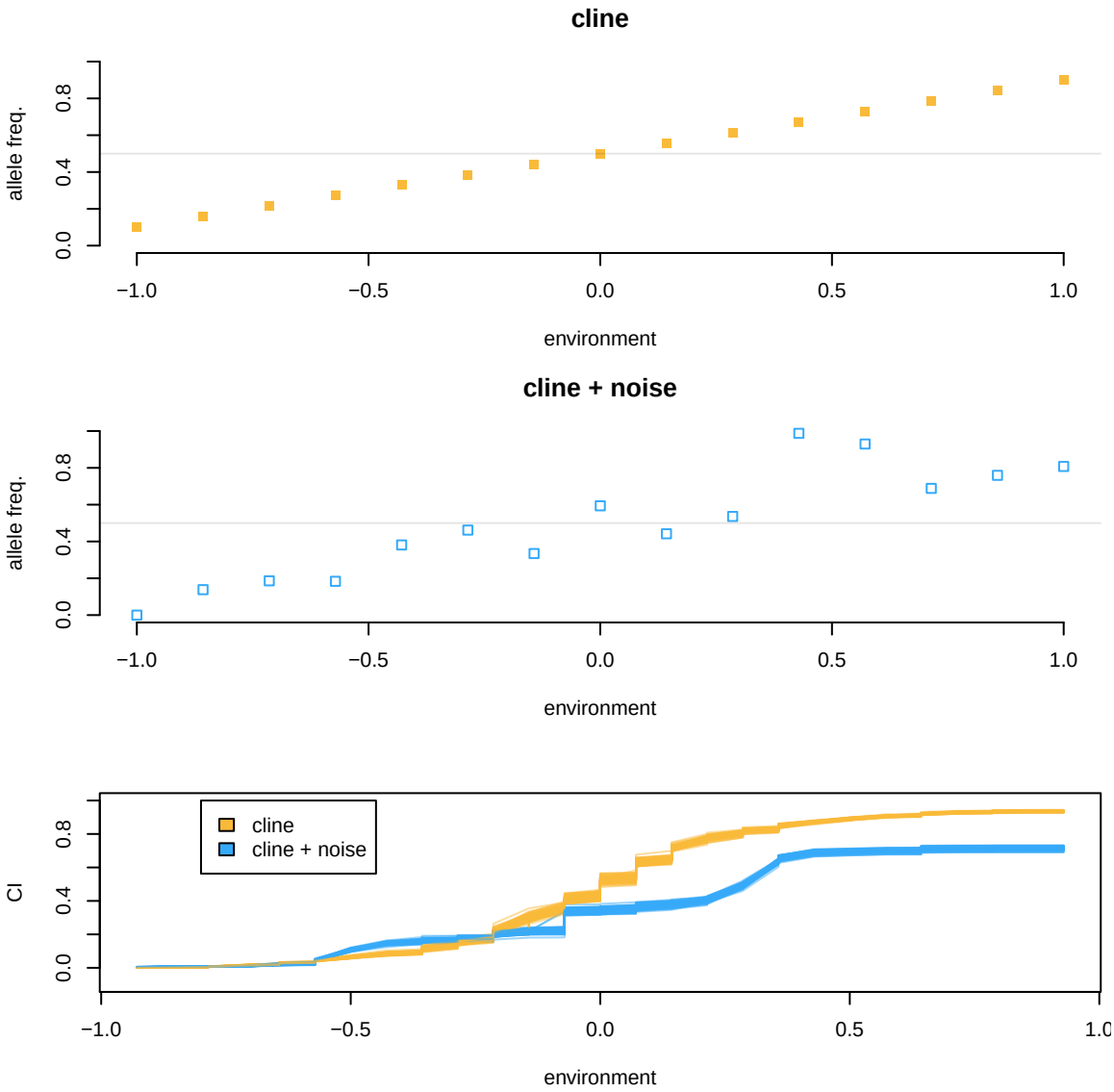
A steep cline vs. more randomness in the cline

Fifth take home message: The maximum value of the CI function reflects the amount of variation in the data that the model explains.

For this comparison, we assumed equal sampling (15 sites along the cline). We compared the linear cline from the second comparison with the same cline, but random noise added.

```
alFreq_L1_long_error <- alFreq_L1_long + rnorm(length(alFreq_L1_long), 0, 0.1)
alFreq_L1_long_error[alFreq_L1_long_error>1] <- 1
alFreq_L1_long_error[alFreq_L1_long_error<0] <- 0
CI_long_error <- replicate(100,runMod(envPop_long, alFreq_L1_long_error))
CI_long_error_R2 <- (unlist(CI_long_error[seq(3,length(CI_long_error), by=3)]))
```

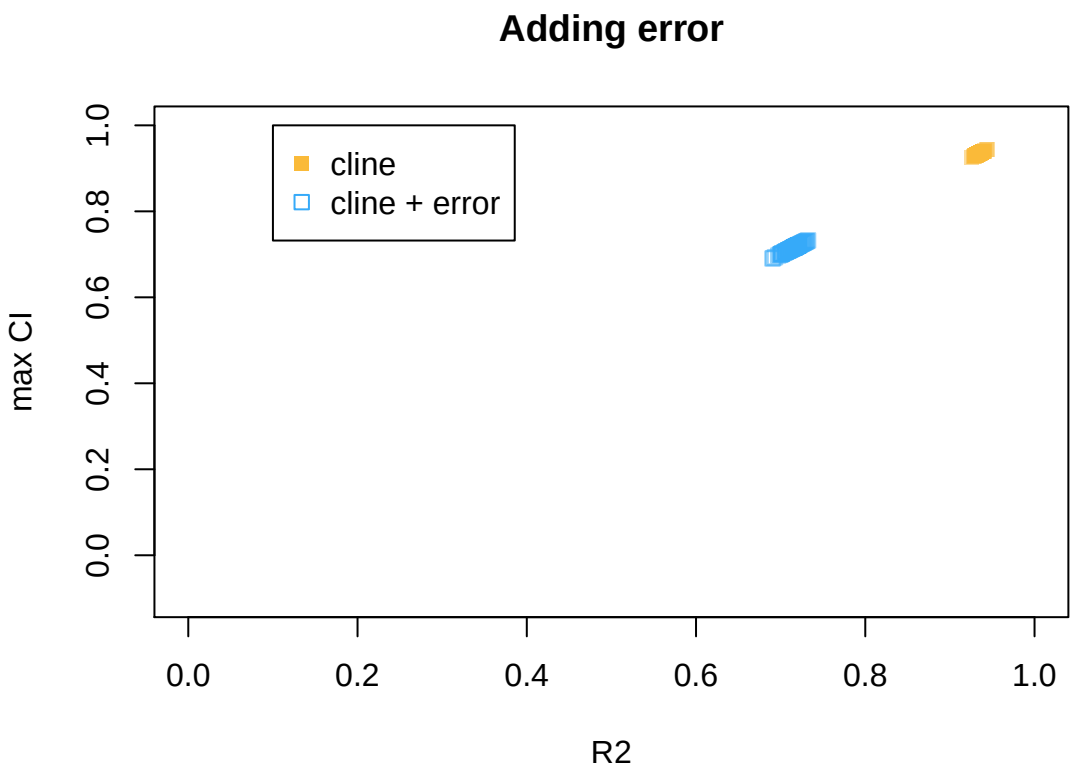
```
par(mar=c(4,4,3,0.4), mfrow=c(3,1))
plot(envPop_long$envPop, alFreq_L1_long, xlab="environment", ylab="allele freq.", col=cols[7], pch=pchs
abline(h=0.5, col=rgb(0,0,0,0.1))

plot(envPop_long$envPop, alFreq_L1_long_error, xlab="environment", ylab="allele freq.", col=cols[3], pc
abline(h=0.5, col=rgb(0,0,0,0.1))

# CI vs. env
sequ <- seq(1, length(CI_long), by=3)
plot(CI_long[[1]]$x, CI_long[[1]]$y, type="l", col=cols[7], ylim=c(-0.1,1), ylab="CI", xlab="environmen
for (i in sequ){
  points(CI_long_error[[i]]$x, CI_long_error[[i]]$y, type="l", col=adjustcolor(cols[3], 0.5))
  points(CI_long[[i]]$x, CI_long[[i]]$y, type="l", col=adjustcolor(cols[7], 0.5))
}

legend(-0.8, 1, fill=cols[c(7,3)], legend = c("cline", "cline + noise"))
```



The above example shows that adding noise (as might occur due to genetic drift) changes the shape of the CI curve, and also lowers the maximum value that can be obtained. That is because the maximum value of the CI for an allele depends on how much variance in the allele frequency as a function of the environment the model can explain. When there is more noise in the data, the model explains less of the variance in allele frequency.

In addtion, we see the "all zeros in the CI despite high R^2" problem again with the noisy data, but it appeared to only happen once (and may not happen when the markdown renders).

```
sequ <- seq(1, length(CI_short), by=3)
plot(CI_long[[3]], max(CI_long[[1]]$y), col=adjustcolor(cols[7], 0.5), ylim=c(-0.1,1), xlab="R2", ylab=
for (i in sequ){
  points(CI_long[[i+2]], max(CI_long[[i]]$y), col=adjustcolor(cols[7], 0.5), pch=pchs[4])
  points(CI_long_error[[i+2]],max(CI_long_error[[i]]$y),  col=adjustcolor(cols[3], 0.5), pch=pchs[1])
}

legend(0.1, 1, col =cols[c(7,3)], legend = c("cline", "cline + error"), pch=pchs[c(4,1)])
```



Appendix: Varying the parameters

Here we examine: (1) varying number of trees and other parameters (2) varying number of loci and environments

testing smaller number of trees

The following chunk tests what happens when 50, instead of 500 trees are used in the gradient forest.

```
### 50 trees ####
envPop_short <- data.frame(envPop=seq(-1,1,length.out=6)) #create env
alFreq_L1_short<-seq(0.1,0.9,length.out=6)
CI_short <- replicate(100,runMod(envPop_short, alFreq_L1_short, 50))
(CI_short_prop0 <- sum(unlist(CI_short[seq(2,length(CI_short), by=3)]))/100)
```

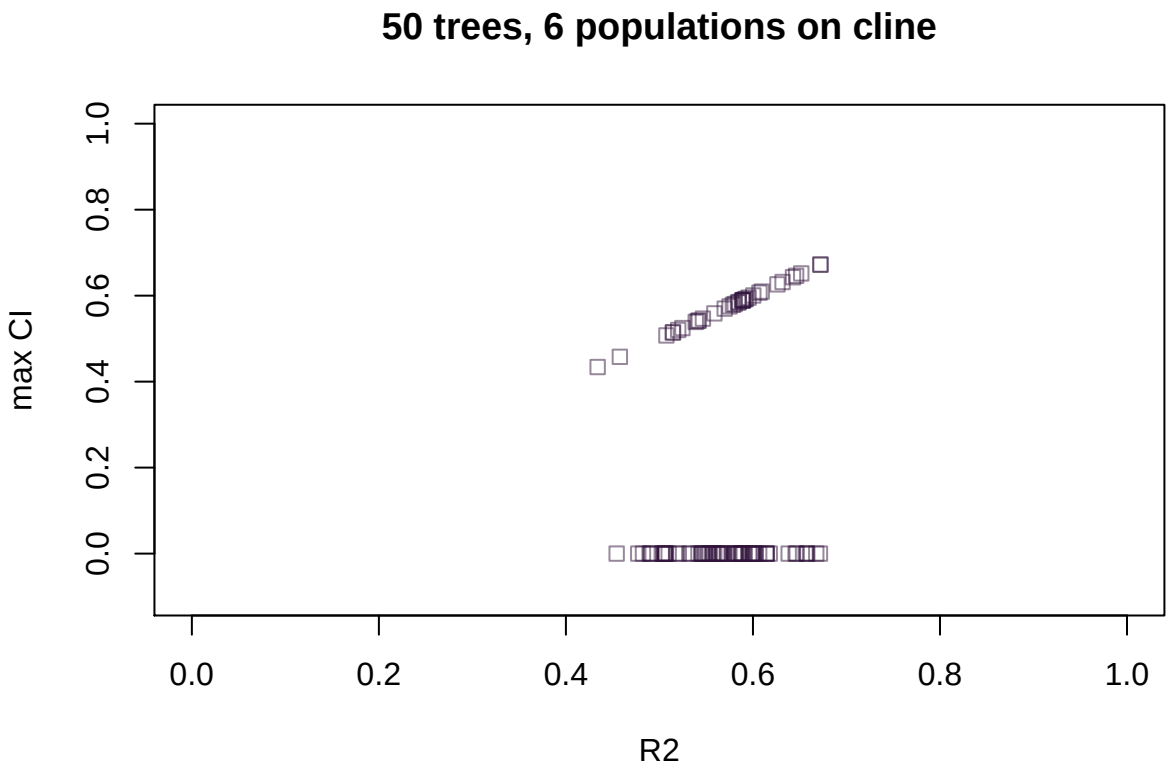
```
## [1] 0.65
```

```
(CI_short_R2 <- (unlist(CI_short[seq(3,length(CI_short), by=3)])))
```

```
##    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq
## 0.5640594 0.5146315 0.5388191 0.5060984 0.5888075 0.6262222 0.6028817 0.6716384
##    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq
## 0.6428197 0.5142203 0.5074620 0.5891499 0.5200636 0.5797859 0.5058276 0.5840260
##    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq
## 0.5464937 0.6516609 0.6006369 0.5098625 0.5448255 0.6382043 0.4906079 0.5513094
##    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq
## 0.6316946 0.5959444 0.6461828 0.6576784 0.5925074 0.5408996 0.5341026 0.4339150
##    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq
```

```
## 0.5848557 0.6069635 0.5785703 0.6093869 0.5988525 0.5869832 0.4909673 0.5453111
##    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq
## 0.4894957 0.6580032 0.6006830 0.5697750 0.5658817 0.5484657 0.5062534 0.5207086
##    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq
## 0.5681984 0.4542661 0.6003900 0.6179509 0.4773472 0.5843916 0.5040819 0.4824090
##    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq
## 0.6676968 0.5319057 0.5069736 0.5421320 0.5969174 0.5571251 0.5876186 0.5820417
##    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq
## 0.5177913 0.6145400 0.5882389 0.4576746 0.5587660 0.5892426 0.5417823 0.6013371
##    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq
## 0.5455950 0.6456101 0.5893254 0.6722149 0.5902102 0.5538899 0.6563736 0.5672238
##    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq
## 0.5025077 0.5565976 0.5621897 0.6142942 0.6468090 0.5800299 0.6147809 0.5619541
##    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq
## 0.5887611 0.5805008 0.6066636 0.5749816 0.5953550 0.5245828 0.5848650 0.5809833
##    alFreq    alFreq    alFreq    alFreq
## 0.5909567 0.5525944 0.5586372 0.6721210
```

```
par(mfrow=c(1,1))
plot(CI_short[[3]], max(CI_short[[1]]$y), col=adjustcolor(cols[1], 0.5), ylim=c(-0.1,1), xlab="R2", ylab
for (i in sequ){
  points(CI_short[[i+2]], max(CI_short[[i]]$y),  col=adjustcolor(cols[1], 0.5), pch=pchs[1])
}
```



With 50 trees, there is higher variance in the R^2, but fewer replicates randomly have a max CI=0.

testing larger number of trees

The following chunk tests what happens when 3000, instead of 500 trees are used in the gradient forest.

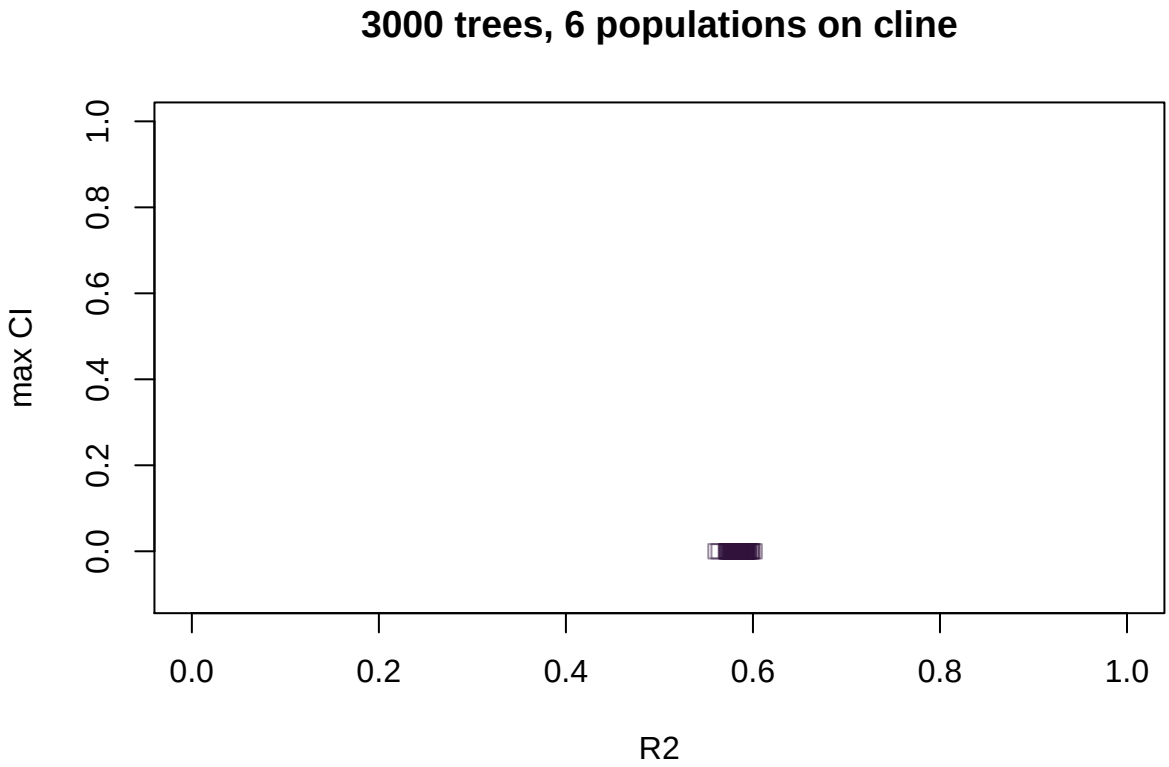
```
### 3000 Trees
envPop_short <- data.frame(envPop=seq(-1,1,length.out=6)) #create env
alFreq_L1_short<-seq(0.1,0.9,length.out=6)
CI_short <- replicate(100,runMod(envPop_short, alFreq_L1_short, 3000))
(CI_short_prop0 <- sum(unlist(CI_short[seq(2,length(CI_short), by=3)]))/100)
```

```
## [1] 1
```

```
(CI_short_R2 <- (unlist(CI_short[seq(3,length(CI_short), by=3)])))
```

```
##    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq
## 0.5822242 0.5876016 0.5888111 0.5761212 0.5887489 0.5822095 0.5933892 0.5863021
##    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq
## 0.5852664 0.5897681 0.5864104 0.5818532 0.5791071 0.5959563 0.5847642 0.5910439
##    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq
## 0.5950287 0.5826453 0.5949270 0.5785810 0.5930226 0.5599576 0.5795856 0.5832636
##    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq
## 0.5762809 0.5771733 0.5980986 0.5985381 0.5938065 0.5916142 0.5819014 0.5862796
##    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq
## 0.5919149 0.5884928 0.5768099 0.5770434 0.5827178 0.5710193 0.5717712 0.5797755
##    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq
## 0.5867210 0.5841803 0.5835819 0.5752422 0.5860649 0.5890658 0.5768574 0.5994981
##    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq
## 0.5825852 0.5797343 0.5792954 0.5732948 0.6017210 0.5891882 0.5783587 0.5804352
##    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq
## 0.5788849 0.5837896 0.5818375 0.5839265 0.5928426 0.5909676 0.5853117 0.5843738
##    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq
## 0.5792581 0.5823283 0.5836936 0.5925291 0.5719179 0.5881464 0.5912848 0.5896243
##    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq
## 0.5884925 0.5859716 0.5871760 0.5726268 0.5876209 0.5796322 0.5633249 0.5826731
##    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq
## 0.5888460 0.5794107 0.5893843 0.5809165 0.5740343 0.5851063 0.5852536 0.5744621
##    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq
## 0.5800276 0.5958652 0.5931320 0.5843662 0.5919345 0.5818019 0.5813584 0.5784764
##    alFreq    alFreq    alFreq    alFreq
## 0.5815395 0.5894890 0.5792373 0.5819016
```

```
par(mfrow=c(1,1))
plot(CI_short[[3]], max(CI_short[[1]]$y), col=adjustcolor(cols[1], 0.5), ylim=c(-0.1,1), xlab="R2", ylab
for (i in sequ){
  points(CI_short[[i+2]], max(CI_short[[i]]$y),  col=adjustcolor(cols[1], 0.5), pch=pchs[1])
}
```



With 3000 trees, there is lower variance in the R^2, but all replicates have a max CI=0. That was weird. Moving on.

testing higher corr threshold

The following chunk tests if different correlation thresholds in the gradient forest affects the output.

```
### Compare the same cline for 10 vs. 50 environment sites ####
envPop_short <- data.frame(envPop=seq(-1,1,length.out=8)) #create env
alFreq_L1_short<-seq(0.1,0.9,length.out=8)
CI_short <- replicate(100,runMod(envPop_short, alFreq_L1_short, 500, 0.9))
print("Proportion of times we get all 0's in the CI with 8 sites, 500 trees, cor threshold 0.9:")
```

```
## [1] "Proportion of times we get all 0's in the CI with 8 sites, 500 trees, cor threshold 0.9:"
```

```
(CI_short_prop0 <- sum(unlist(CI_short[seq(2,length(CI_short), by=3)]))/100)
```

```
## [1] 0.69
```

```
CI_short <- replicate(100,runMod(envPop_short, alFreq_L1_short, 500, 0.5))
print("Proportion of times we get all 0's in the CI with 8 sites, 500 trees, cor threshold 0.5:")
```

```
## [1] "Proportion of times we get all 0's in the CI with 8 sites, 500 trees, cor threshold 0.5:"
```

```
(CI_short_prop0 <- sum(unlist(CI_short[seq(2,length(CI_short), by=3)]))/100)
```

```
## [1] 0.65
```

```
CI_short <- replicate(100,runMod(envPop_short, alFreq_L1_short, 500, 0.1))
print("Proportion of times we get all 0's in the CI with 8 sites, 500 trees, cor threshold 0.1:")
```

```
## [1] "Proportion of times we get all 0's in the CI with 8 sites, 500 trees, cor threshold 0.1:"
```

```
(CI_short_prop0 <- sum(unlist(CI_short[seq(2,length(CI_short), by=3)]))/100)
```

```
## [1] 0.65
```

```
CI_short <- replicate(100,runMod(envPop_short, alFreq_L1_short, 50, 0.1))
print("Proportion of times we get all 0's in the CI with 8 sites, 50 trees, cor threshold 0.1:")
```

```
## [1] "Proportion of times we get all 0's in the CI with 8 sites, 50 trees, cor threshold 0.1:"
```

```
(CI_short_prop0 <- sum(unlist(CI_short[seq(2,length(CI_short), by=3)]))/100)
```

```
## [1] 0.14
```

```
CI_short <- replicate(100,runMod(envPop_short, alFreq_L1_short, 1000, 0.1))
print("Proportion of times we get all 0's in the CI with 8 sites, 1000 trees, cor threshold 0.1:")
```

```
## [1] "Proportion of times we get all 0's in the CI with 8 sites, 1000 trees, cor threshold 0.1:"
```

```
(CI_short_prop0 <- sum(unlist(CI_short[seq(2,length(CI_short), by=3)]))/100)
```

```
## [1] 0.89
```

```
CI_short <- replicate(100,runMod(envPop_short, alFreq_L1_short, 10, 0.1))
print("Proportion of times we get all 0's in the CI with 8 sites, 10 trees, cor threshold 0.1:")
```

```
## [1] "Proportion of times we get all 0's in the CI with 8 sites, 10 trees, cor threshold 0.1:"
```

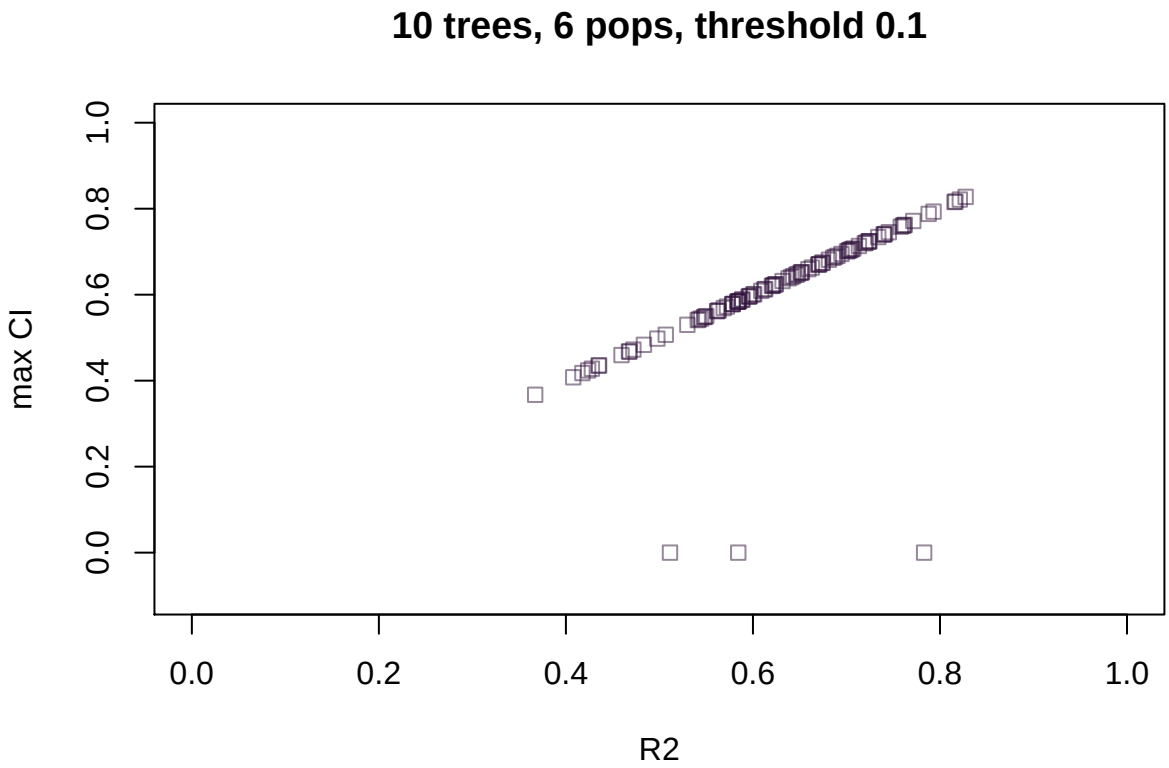
```
(CI_short_prop0 <- sum(unlist(CI_short[seq(2,length(CI_short), by=3)]))/100)
```

```
## [1] 0.03
```

```
(CI_short_R2 <- (unlist(CI_short[seq(3,length(CI_short), by=3)])))
```

```
##    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq
## 0.5835723 0.6122003 0.6946944 0.6630016 0.6130569 0.7209615 0.5781991 0.4977504
##    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq
## 0.6738302 0.7931634 0.4176383 0.7239505 0.5501379 0.6241533 0.5954438 0.5837610
##    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq
## 0.4680550 0.6202902 0.5783525 0.6002781 0.5424306 0.6857510 0.6013343 0.4355068
##    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq
## 0.4674215 0.7133445 0.6812561 0.5780690 0.7245465 0.6316897 0.6212312 0.4277747
##    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq
## 0.7611870 0.8212090 0.6707627 0.6462856 0.5884784 0.7603513 0.6593050 0.5067709
##    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq
## 0.4349268 0.8274235 0.7029493 0.5854262 0.5298670 0.5886232 0.7830584 0.7623808
##    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq
## 0.6240996 0.6746856 0.6409556 0.5722041 0.6696402 0.7714659 0.7878698 0.5634724
##    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq
## 0.6510078 0.5485780 0.4078315 0.5687487 0.7058317 0.5969481 0.5847519 0.6479356
##    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq
## 0.5843026 0.5409360 0.4721812 0.6707543 0.5446764 0.6517732 0.7402107 0.6088672
##    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq
## 0.7043676 0.6903815 0.6877923 0.5958771 0.5112976 0.7397239 0.7074111 0.7453011
##    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq
## 0.7340539 0.3671461 0.6219419 0.5618588 0.7412025 0.6523862 0.4594859 0.4833906
##    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq    alFreq
## 0.6381154 0.7005917 0.7580838 0.7021783 0.6430845 0.5619208 0.7236326 0.8165032
##    alFreq    alFreq    alFreq    alFreq
## 0.7195849 0.8155736 0.4237842 0.5477945
```

```r
par(mfrow=c(1,1))
plot(CI_short[[3]], max(CI_short[[1]]$y), col=adjustcolor(cols[1], 0.5), ylim=c(-0.1,1), xlab="R2", ylab
for (i in sequ){
  points(CI_short[[i+2]], max(CI_short[[i]]$y),  col=adjustcolor(cols[1], 0.5), pch=pchs[1])
}
```



We wondered if this may relate to the number of trees far exceeding the number of possible splits given only a few samples along a gradient. When the number of requested permutations exceeds the number possible in the data, the result is a large number of identical trees.

Errors arose when the number of tress was *less* then the number of samples in the data.

testing multiple environmental predictors

Here, we compare two models. The first "univariate" model only includes the true environmental predictor. The second "multivariate model" includes the true predictor and two random environmental variables.

Each has 15 sites sampled along the true environmental gradient.

```r
envPop_short <- data.frame(envPop=seq(-1,1,length.out=15),
                           env2 = rnorm(15),
                           env3 = rnorm(15)) #create env
alFreq_L1_short<-seq(0.1,0.9,length.out=15)

uni <- runMod(data.frame(envPop=envPop_short[,1]), alFreq_L1_short)
multi <- runMod_multi(envPop_short, alFreq_L1_short)

print("The univariate model, which only includes the true predictor, has an R^2 of:")
print(uni$R2)
```

```
print("The multivariate model, which includes the true predictor and two random predictors, has an R^2 o
print(multi$R2)

# How are the CI functions affected by including multiple predictors?
plot(uni$CI$x, uni$CI$y, main="compare univariate (black circ.)\nto multivariate (blue tri.)")
points(multi$CI_env$envPop$x, multi$CI_env$envPop$y, col="blue", pch=2)
```

The above R^2 results make sense. When more random predictors are included, the random forest would choose these sometimes to build the model, and this would lower the overal predictive power of the model compared to a model that only includes the true predictor.

The plot shows that when adding additional random environmental predictors, it affects the total CI of the model for the true environmental predictor. In addition this would affect genetic offset calculations near the environmental extremes.

Testing multiple loci and one environment

Here we ask, does adding more loci to the model affect the results for one locus? Based on first princples, it should not, because each locus is an independent model. we created a second allele with frequencies sampled from a beta distribution, and a third allele with frequencies sampled randomly from the first locus. The second allele does not have a positive R^2 in a univariate model, but the third allele does. Here, we test if the statistics output from the 1-locus-1-environment model are the same as the multi-locus-multi-environment model.

```
L1 = seq(0.1,0.9,length.out=15)
L2 = rbeta(15, 1, 1)
set.seed(9126)
L3 = sample(L1)
envPop_short <- data.frame(envPop=seq(-1,1,length.out=15)) #create env
alFreq_L1_short<-data.frame(L1,L2, L3)
# use "uni" model above
uni
# uni_L2 <- runMod(envPop_short, L2) this returns NULL
uni_L3 <- runMod(envPop_short, L3)
uni_L3
uni_L3_2 <- runMod(envPop_short, L3)
uni_L3_2

multi_loc <- runMod_multi(envPop_short, alFreq_L1_short)
multi_loc

plot(uni$CI$y, multi_loc[[5]]$envPop$L1$y, xlab="CI L1 one locus", ylab="CI L1 three locus")
abline(0,1)

plot(uni_L3$CI$y, uni_L3_2$CI$y, xlab="CI L3 one locus", ylab="CI L3 one locus again")
abline(0,1)

plot(uni_L3$CI$y, multi_loc[[5]]$envPop$L3$y[2:43], xlab="CI L3 one locus", ylab="CI L3 three locus")
abline(0,1)

print("The R^2 for L1 from the one-locus model:")
print(uni$R2)
print("The R^2 for L1 from the multi-locus model:")
print(multi_loc$R2[1])
```

```r
print("The R^2 for L3 from the one-locus model:")
print(uni_L3$R2)
print("The R^2 for L3 from the multi-locus model:")
print(multi_loc$R2[2])
```

The last analysis shows that adding more loci does not affect the CI curves for a single locus, above and beyond what happens from run to run given the randomness in the forest.