```
library(mvtnorm)

# Grammar functions
# Inputs: tableau, tab, and a vector of constraint weights, w
# return vector of candidate probabilities

# sample tableau
# candidates are rows, constraints are columns
c1 = c(-1,0,0)
c2 = c(0,-2,0)
c3 = c(0,-1,-1)
tab = rbind(c1, c2, c3)
# vector of constraint weights in the same order as columns of tab
w = c(15,8,8)

# Maxent probabilities

# calculate probabilities of candidates given a MaxEnt grammar
ME <-function(tab,w){
        # calculate harmony vector
        h = w %*% t(tab)
        # calculate probabilities of candidates
        p = exp(h)/sum(exp(h))
        return(p)}

# calculate probabilities of candidates given a Normal MaxEnt grammar
NME<-function(tab,w){
        # calculate harmony vector
        h = w %*% t(tab)
        # initialize vector of candidate probabilities
        p = rep(NA,nrow(tab))
        # ncands is the number of candidates
        ncands = nrow(tab)
        # set standard deviation of noise terms
        sd = 1
        # sigma is the covariance matrix of the joint distribution of
noise differences
        # i.e. an (ncands-1)*(ncands-1) matrix with s*sd^2 on the
leading diagonal and sd^2 elsewhere
        sigma = sd^2*(diag(ncands-1)+matrix(1,nrow=ncands-1,
ncol=ncands-1))

        # calculate probability of each candidate
        for (c in 1:ncands){
                # calculate harmony differences between each
candidate and candidate c
                hdiff = h[c]-h[-c]
                # calculate candidate probability of candidate c from
multivariate normal distribution function
                p[c] = pmvnorm(upper=hdiff, sigma=sigma)
        }
        return(p)
}

# NHG probabilities

# calculate the probabilities of candidates given an NHG
NHG<-function(tab,w){
        # calculate harmony vector
        h = w %*% t(tab)
        # initialize vector of candidate probabilities
        p = rep(NA,nrow(tab))
        # set standard deviation of noise terms
        sd = 1

        # calculate probability of each candidate
        for (c in 1:nrow(tab)){
                # calculate vector of harmony differences between
each other candidate and candidate c
                hdiff = h[c]-h[-c]
                # calculate matrix of differences in violations
between candidate c and each other candidate
                difftab = tab[-c,]-
matrix(rep(tab[c,],nrow(tab)-1),nrow=nrow(tab)-1,byrow=TRUE)
                # calculate candidate probability of candidate c from
multivariate normal distribution function
                # use violation difference matrix to calculate
covariance among noise terms, sigma
                p[c] = pmvnorm(upper=hdiff, sigma=(sd^2)*(difftab %*%
t(difftab)))
                }
return(p)
}

# estimate probabilities of candidates given a censored NHG
cNHG <- function(tab,w){
        # initial vector of candidate probabilities
        p_hat = rep(NA,nrow(tab))
        # number of constraints
        nconstraints = ncol(tab)
        # number of samples to use in simulation
        N=10^6
        # generate N weight vectors, each perturbed by censored noise,
combine them into a matrix
        wnoise = matrix(pmax(rnorm(nconstraints*N, mean=w, sd=1),
0),ncol=nconstraints, byrow=TRUE)
        # calculate matrix of harmonies of candidates for each
perturbed weight vector
        hs = wnoise%*%t(tab)
        # calculate observed probabilities of each candidate
        for(c in 1:nrow(tab)){
                # count number of times candidate c wins and divide
by N
                p_hat[c]=sum(apply(hs,1,which.max)==c)/N
                }
        return(p_hat)
}

# apply functions to sample tableau
# MaxEnt
ME(tab,w)
# normal Maxent
NME(tab,w)
# NHG
NHG(tab,w)
# censored NHG
cNHG(tab,w)

########################################
# Fit grammars to Smith & Pater schwa data

# Observed rates of schwa realization in Smith & Pater (2020)
# order of contexts: clitic: CC_s, CC_ss, C_s, C_ss, word: CC_s,
CC_ss, C_s, C_ss

p = c(0.938, 0.914,0.648,0.562,0.833,0.683,0.122,0.09)

# convert probabilities to counts
# 27 subjects * 6 items per context = 162
schwa_count = round(162*p)

# Fit MaxEnt grammars to schwa data

# Constraints - differences in violations in each context, ordered as
above,
# schwa violations - non-schwa violations

# NoSchwa
c1 = rep(-1, 8)
# *CCC
c2 = c(1,1,0,0,1,1,0,0)
# *Clash
c3 = c(1,0,1,0,1,0,1,0)
# Max
c4 = c(rep(1,4),rep(0,4))
# Dep
c5 = c(rep(0,4), rep(-1,4))
# *Cluster
c6 = c(0,0,1,1,0,0,1,1)
# *CCC/iP
```

```
c7 = c(rep(0,4),1,1,0,0)

#######################
# Fit MaxEnt grammar with constraints 1-4 using logistic regression
# (weight of constraint 1 = negation of the intercept)
maxent =glm(cbind(schwa_count, 162-schwa_count)~c2+c3+c4,
family=binomial(link="logit"))
summary(maxent)

# extract fitted schwa probabilities
ppmaxent = fitted(maxent)

# Test for significant differences in weights of constraints between
subsets of contexts
# by adding two-way interactions to the logistic regression MaxEnt
model
twoway =glm(cbind(schwa_count, 162-schwa_count)~(c2+c3+c4)^2,
family=binomial(link="logit"))
summary(twoway)

# fit normal MaxEnt grammar with constraints 1-4 using probit
regression
# in probit regression noise follows a normal distribution with sd =1,
whereas in normal MaxEnt
# sd of noise difference is sqrt(2), if the noise added to candidate
harmonies has sd = 1.
# So to convert coefficients of the probit regression to normal MaxEnt
weights, the must be multiplied by sqrt(2)
# (weight of constraint 1 = -intercept*sqrt(2))
norm_maxent = glm(cbind(schwa_count, 162-schwa_count)~c2+c3+c4,
family=binomial(link="probit"))
summary(norm_maxent)

# convert coefficients to constraint weights
w = coef(norm_maxent)*sqrt(2)
w[1] = -w[1]

# extract fitted schwa probabilities
ppnormmaxent = fitted(norm_maxent)

# Fit models with revised constraint set, adding *CCC/iP and
collapsing Max/Dep into a single constraint
# create Max/Dep constraint
c45 = c4+c5

# Fit MaxEnt grammar with revised constraint set using logistic
regression
# (weight of constraint 1 = negation of the intercept)
maxent2 =glm(cbind(schwa_count, 162-schwa_count)~c2+c3+c45+c7,
family=binomial(link="logit"))
```

```
summary(maxent2)

# extract fitted schwa probabilities
ppmaxent2 = fitted(maxent2)

# Fit normal MaxEnt grammar with revised constraint set using logistic
regression
norm_maxent2 =glm(cbind(schwa_count, 162-schwa_count)~c2+c3+c45+c7,
family=binomial(link="probit"))
summary(norm_maxent2)

# convert coefficients to constraint weights
w = coef(norm_maxent2)*sqrt(2)
w[1] = -w[1]


############################
# Fit NHG grammars to Smith & Pater schwa data

# Tableaux
# violation vectors for schwa and non-schwa candidates in each context
# order of constraints: NoSchwa, *CCC. Clash, Max, Dep, *CC, *CCC/iP
#CC_, _s, /schwa/
c1 = c(0,-1,-1,-1,0,-1,0)
c2 = c(-1,0,0,0,0,-1,0)
#CC_, _ss, /schwa/
c3 = c(0,-1,0,-1,0,-1,0)
c4 = c(-1,0,0,0,0,-1,0)
#C_, _s, /schwa/
c5 = c(0,0,-1,-1,0,-1,0)
c6 = c(-1,0,0,0,0,0,0)
#C_, _ss, /schwa/
c7 = c(0,0,0,-1,0,-1,0)
c8 = c(-1,0,0,0,0,0,0)
#CC_, _s, /0/
c9 = c(0,-1,-1,0,0,-1,-1)
c10 = c(-1,0,0,0,-1,-1,0)
#CC_, _ss, /0/
c11 = c(0,-1,0,0,0,-1,-1)
c12 = c(-1,0,0,0,-1,-1,0)
#C_, _s, /0/
c13 = c(0,0,-1,0,0,-1,0)
c14 = c(-1,0,0,0,-1,0,0)
#C_, _ss, /0/
c15 = c(0,0,0,0,0,-1,0)
c16 = c(-1,0,0,0,-1,0,0)

# combine vectors into matrix of tableaux
tabs = rbind(c1,c2,c3,c4,c5,c6,c7,c8,c9,c10,c11,c12,c13,c14,c15,c16)
```

```
# exclude *CCC/iP
tabs = tabs[,1:6]

# function to calculate residual deviance of a NHG with vector of
constraint weights w,
# given a set of tableaux, tabs, and observed counts of schwa (y1) and
non-schwa (y0) candidates.
devhg<-function(w,tabs,y1,y0){
        p_hat = rep(NA,nrow(tabs)/2)

        for(input in 1:8){
                tab = tabs[c((2*input)-1,2*input),]
                p_hat[input]=NHG(tab,w)[2]
        }
        resdev=2*sum((y1*log((y1/(y1+y0)/p_hat))+y0*log(y0/(y1+y0)/(1-
p_hat))))
        return(resdev)
}

# version of devhg that fixes at 0 the weights of Dep and *CC to
eliminate redundancy
# which is present if all constraint weights are allowed to vary

devhg4<-function(w,tabs,y1,y0){
        p_hat = rep(NA,nrow(tabs)/2)
        for(input in 1:8){
                tab = tabs[c((2*input)-1,2*input),]
                p_hat[input]=NHG(tab,c(w, 0,0))[2]
        }
        resdev=2*sum((y1*log((y1/(y1+y0)/p_hat))+y0*log(y0/(y1+y0)/(1-
p_hat))))
        return(resdev)
}

# Find best fitting constraint weights for NHG

# set initial weights to 1
# 4 constraint weights
initial_w = rep(1,4)
# 6 constraints
initial_w = rep(1,6)
# 7 constraints
initial_w = rep(1,7)

# Use L-BFGS-B algorithm to search for constraint weights that
minimize deviance,
# constraining weights to be non-negative.
# All weights free to vary
soln = optim(initial_w,devhg,tabs=tabs,y1=schwa_count,y0=162-
schwa_count, lower=0, method = "L-BFGS-B")
```

```
# Use Nelder-Mead algorithm to search for constraint weights that
minimize deviance,
# no restrictions on constraint weights
soln = optim(initial_w,devhg,tabs=tabs,y1=schwa_count,y0=162-
schwa_count)

# set w to optimal constraint weights
w = soln$par

# First four constraint weights free to vary, 5 and 6 set to 0
soln = optim(initial_w,devhg4,  tabs=tabs,y1=schwa_count,y0=162-
schwa_count)

# set w to optimal constraint weights
# weights 5 and 6 set to 0
w = c(soln$par, 0, 0)

# calculate fitted probabilities, p_hat, given weight vector, w, and
tableaux, tabs
        p_hat = rep(NA,nrow(tabs)/2)

        for(input in 1:8){
                tab = tabs[c((2*input)-1,2*input),]
                p_hat[input]=NHG(tab,w)[2]
        }

# fit constraint set with CCC/iP, but collapsing Max and Dep into a
single constraint

# collapse Max and Dep into a single constraint
tabs[,4]=tabs[,4]+tabs[,5]
tabs = tabs[,-5]

# version of devhg that fixes weight of *CC at 0, avoiding redundancy
in constraint weights
devhg5<-function(w,tabs,y1,y0){
        p_hat = rep(NA,nrow(tabs)/2)
        for(input in 1:8){
                tab = tabs[c((2*input)-1,2*input),]
                p_hat[input]=NHG(tab,c(w[1:4], 0,w[5]))[2]
        }
        resdev=2*sum((y1*log((y1/(y1+y0)/p_hat))+y0*log(y0/(y1+y0)/(1-
p_hat))))
        return(resdev)
}

# set initial weights to 1
initial_w = rep(1,5)
# Use Nelder-Mead algorithm to search for constraint weights that
```

```
minimize deviance,
# with weight of *CC fixed at 0.
soln = optim(initial_w,devhg5,  tabs=tabs,y1=schwa_count,y0=162-
schwa_count, control=list(maxit=1000))

# set w to optimal constraint weights
w = c(soln$par[1:4], 0, soln$par[5])

# calculate fitted probabilities, p_hat, given weight vector, w, and
tableaux, tabs
        p_hat = rep(NA,nrow(tabs)/2)
        for(input in 1:8){
                tab = tabs[c((2*input)-1,2*input),]
                p_hat[input]=NHG(tab,w)[2]
        }

#########################
# Fit censored NHG

# Function to estimate residual deviance of a censored NHG with vector
of constraint weights, w,
# given a set of tableaux, tabs, and observed counts of schwa (y1) and
non-schwa (y0) candidates.
devcNHG=function(w,tabs,y1,y0){
        p_hat = rep(NA,nrow(tabs)/2)
        for(input in 1:8){
                tab = tabs[c((2*input)-1,2*input),]
                p_hat[input]=cNHG(tab,w)[2]
                }
        resdev=2*sum((y1*log((y1/(y1+y0)/p_hat))+y0*log(y0/(y1+y0)/(1-
p_hat))))
        return(resdev)
}

# Find best-fitting constraint weights for censored NHG
# initial constraint weights for 7 constraints (based on optimum for 6
constraints)
initial_w= c(11, 12, 0, 2, 0, 9, 1)
# initial constraint weights for 6 constraints (based on previous
runs)
initial_w= c(10, 10, 0.3, 1.5, 0.1, 9)

# use Nelder-Mead algorithm to search for constraint weights that
minimize deviance of censored NHG grammar
# Warning: This can take a very long time to run because each
iteration of devcNHG takes a long time to run.
# It may be necessary to increase maxit, or feed interim results back
into the optimization routine to find optimal values.
soln = optim(initial_w,devcNHG,tabs=tabs,y1=schwa_count,y0=162-
schwa_count, control=list(maxit=200))
```

```
# extract optimal weights
w = soln$par

# estimate fitted probabilities
p_hat = rep(NA,nrow(tabs)/2)
for(input in 1:8){
        tab = tabs[c((2*input)-1,2*input),]
        p_hat[input]=cNHG(tab,w)[2]
        }

# estimate residual deviance of model from p_hat
# note this will generally be higher than the deviance reported by
optim() because
# that will typically represent a 'lucky draw' in the simulation
process

y1 = schwa_count
y0 = 162-schwa_count
resdev=2*sum((*log((y1/(y1+y0)/p_hat))+y0*log(y0/(y1+y0)/(1-p_hat))))
```