

# AwesomePackage

A valid R package for ancestry inference

Jonathon Chow

School of Mathematical Sciences, USTC

2022-10-04



看, 前面漆黑一片, 甚麼也看不到

# Installation

You can install the development version of AwesomePackage from [GitHub](#) with:

```
# install.packages("devtools")  
devtools::install_github("JONATHONCHOW/AwesomePackage")
```

# Hello world

You can use the following code to see if `AwesomePackage` has been successfully installed.

```
library(AwesomePackage)
hello_world()
```

```
## [1] "Data science is fantastic!"
```

You can check out theories and examples at *Articles* in [AwesomePackage](#).

You can refer to *Reference* in [AwesomePackage](#) for the use of functions, and then you can have fun with ancestry inference!

# Data

The package data: `data_TGP`, `data_HGDP`, `map_TGP`, `map_HGDP`.

# Fit PSD model

We consider the sample data from the 1000 Genomes Project (TGP) and use a PSD model to fit these data. We import the pre-trained results directly.

```
load("result/result_fit_psd.RData")
```

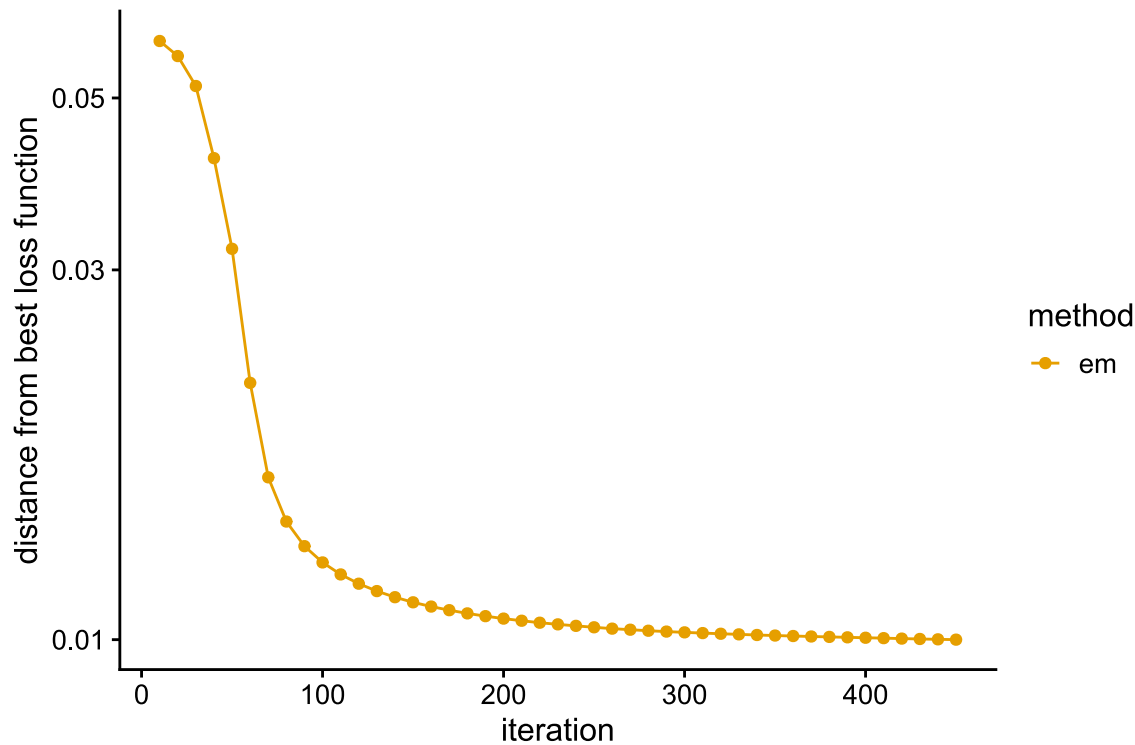
# EM algorithm

We fit the PSD model with EM algorithm, and use the loss function as a stopping criterion. EM algorithm converges slowly. It takes about 450 EM iterations to reach the predetermined accuracy.

```
result_TGP_em <- psd_fit_em(data_TGP, 3, 1e-5, 500)
```

We plot the loss function against the number of iterations using package `ggplot2`, the loss function records once for 10 iterations.

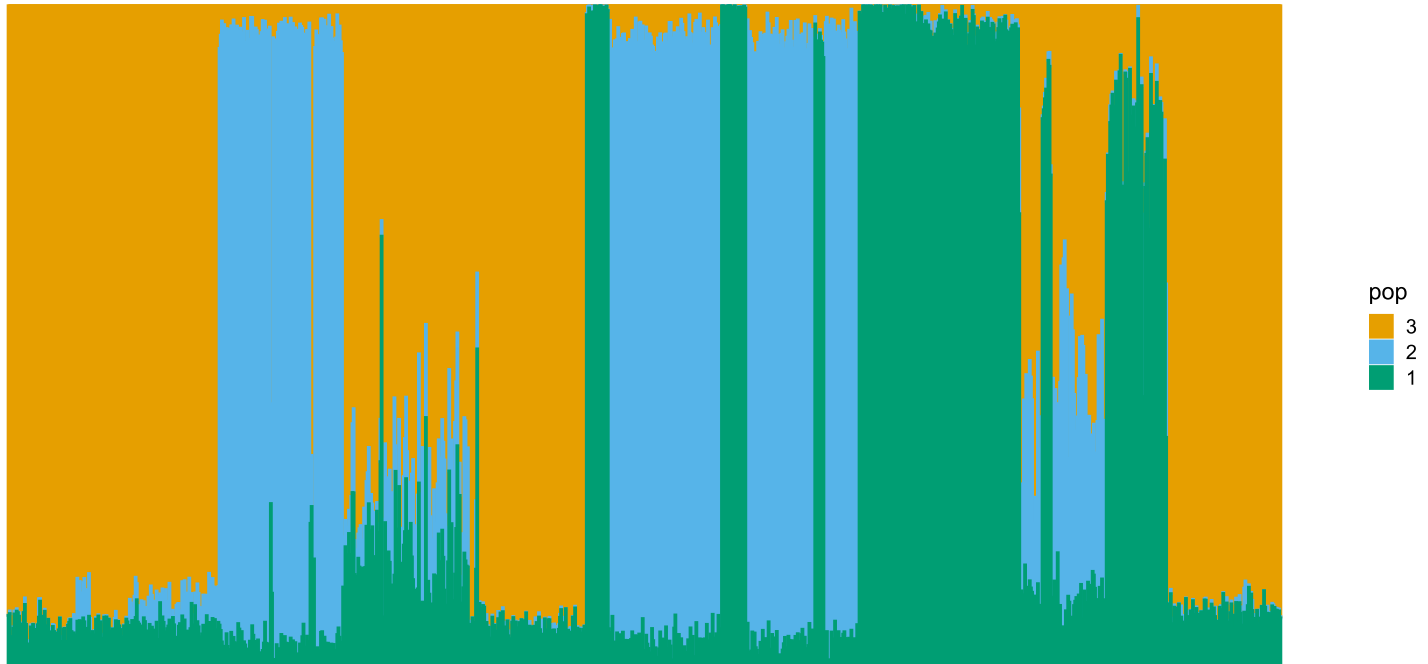
```
L <- result_TGP_em$Loss  
plot_loss(list(L), "em", 10)
```





We plot the ancestral proportions of individuals using package ggplot2. The fitting result of EM algorithm is not very accurate.

```
P <- result_TGP_em$P  
plot_structure(P)
```



We measure the prediction accuracy of the dataset by the maximum likelihood function and the deviance residuals.

```
L[length(L)]
```

```
## [1] -0.5971432
```

```
psd_error(data_TGP, result_TGP_em)
```

```
## [1] 0.3374755
```

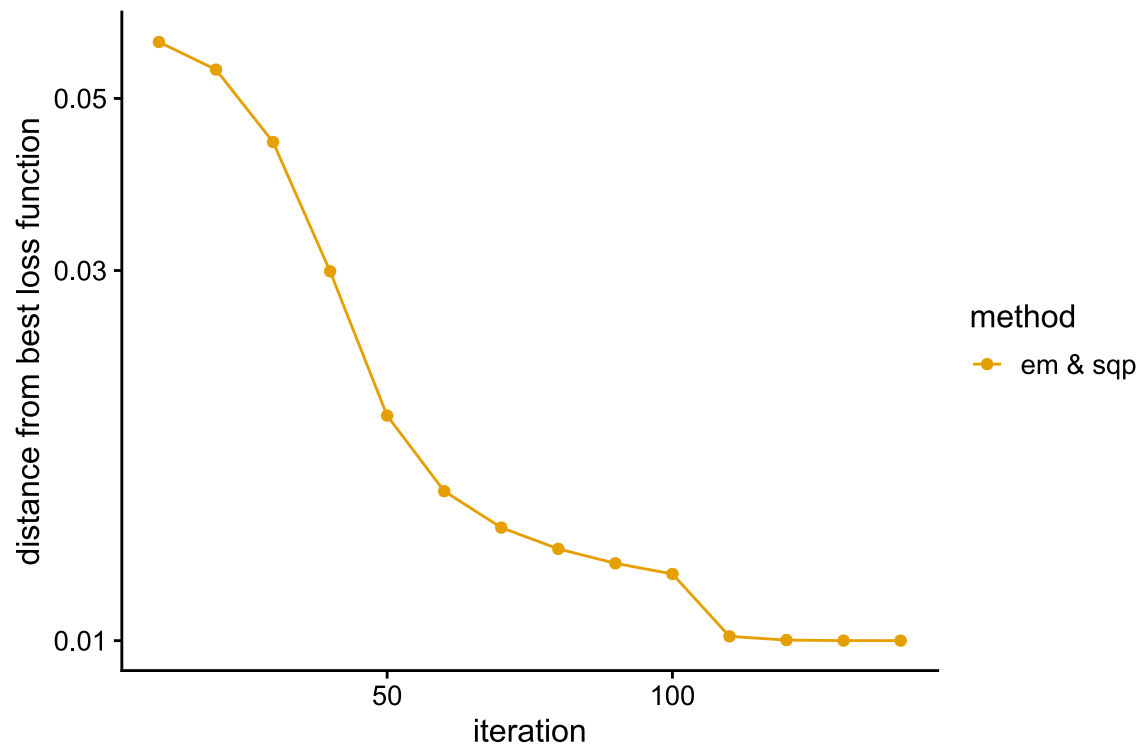
# SQP algorithm

We fit the PSD model with SQP algorithm, and use the loss function as a stopping criterion. Although SQP algorithm converges fast, it is easy to converge to the local optimal value. To prevent this bad scenario, we start with 100 EM iterations, and then only need about 40 SQP iterations to reach the accuracy requirement.

```
result_TGP_sqp <- psd_fit_sqp(data_TGP, 3, 1e-5, 50, 100)
```

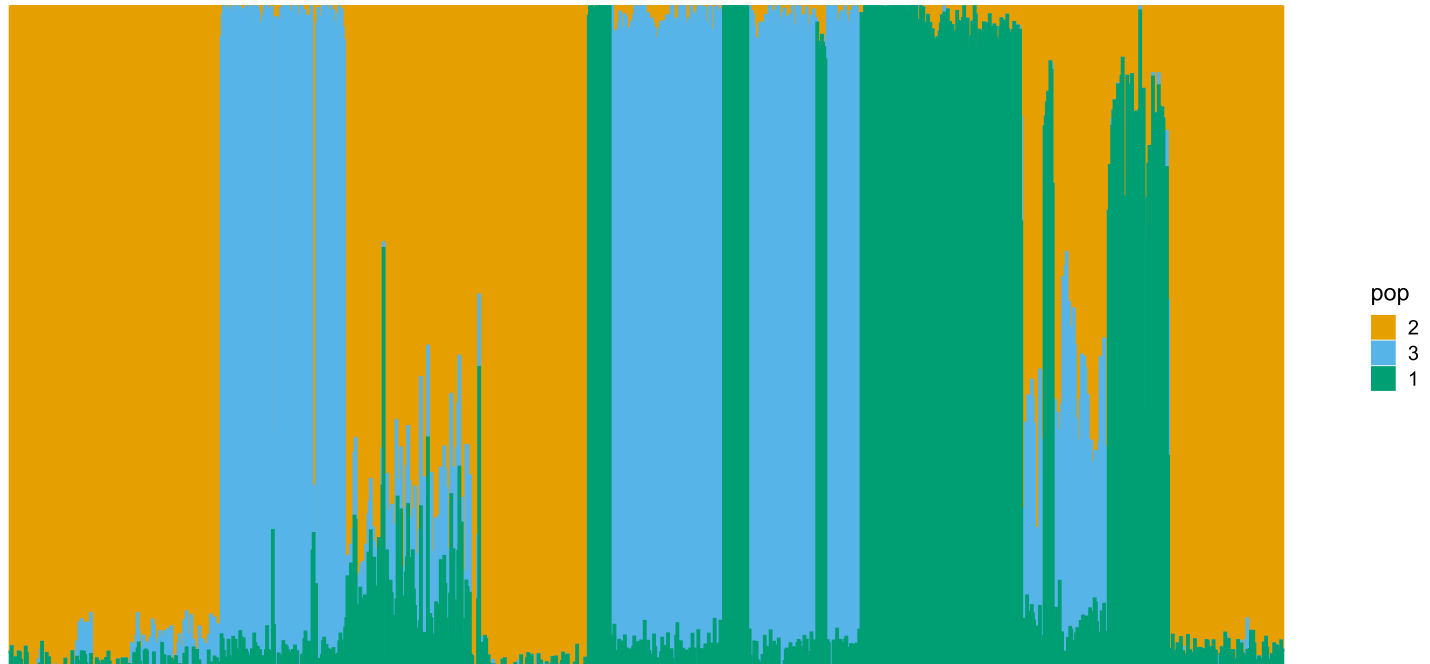
We plot the loss function against the number of iterations using package `ggplot2`, the loss function records once for 10 iterations.

```
L <- result_TGP_sqp$Loss  
plot_loss(list(L), "em & sqp", 10)
```



We plot the ancestral proportions of individuals using package `ggplot2`. Notice that the fitting result of the SQP algorithm is excellent.

```
P <- result_TGP_sqp$P  
plot_structure(P)
```



We measure the prediction accuracy of the dataset by the maximum likelihood function and the deviance residuals.

```
L[length(L)]
```

```
## [1] -0.5969287
```

```
psd_error(data_TGP, result_TGP_sqp)
```

```
## [1] 0.3372611
```

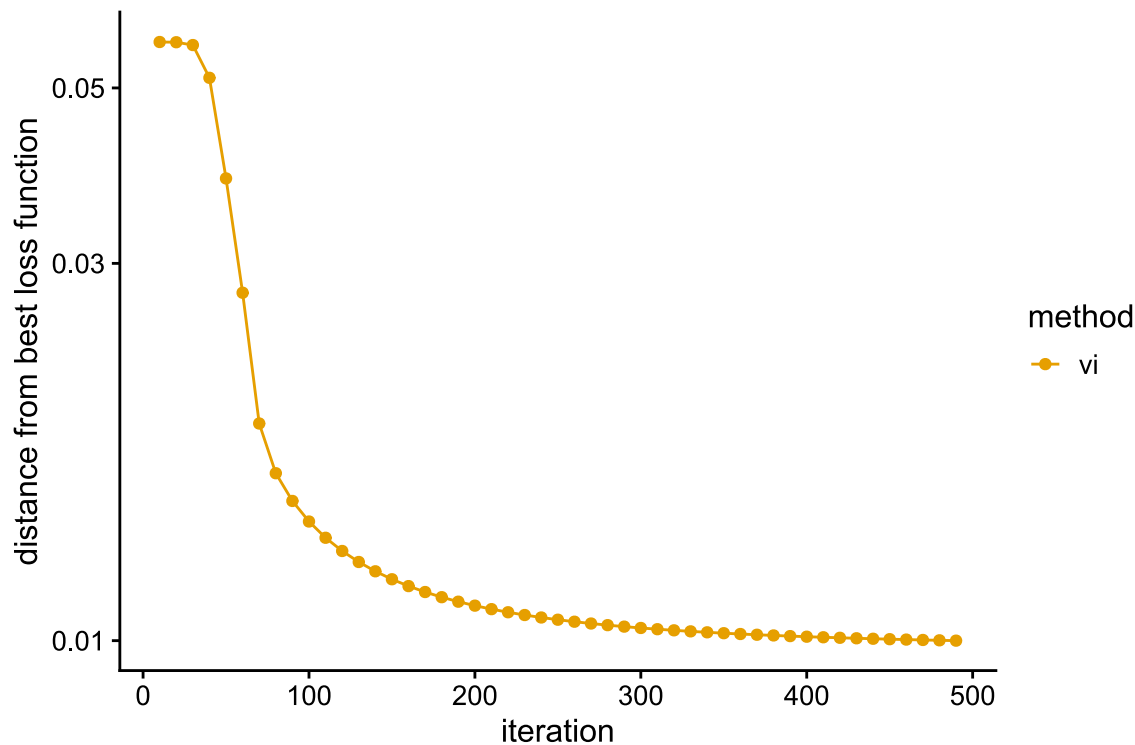
# VI algorithm

We fit the PSD model with VI algorithm, and use the loss function as a stopping criterion. Similar to EM algorithm, the convergence of VI algorithm is also relatively slow, and about 490 iterations are needed to reach the accuracy requirement.

```
result_TGP_vi <- psd_fit_vi(data_TGP, 3, 1e-5, 500)
```

We plot the loss function against the number of iterations using package `ggplot2`, the loss function records once for 10 iterations.

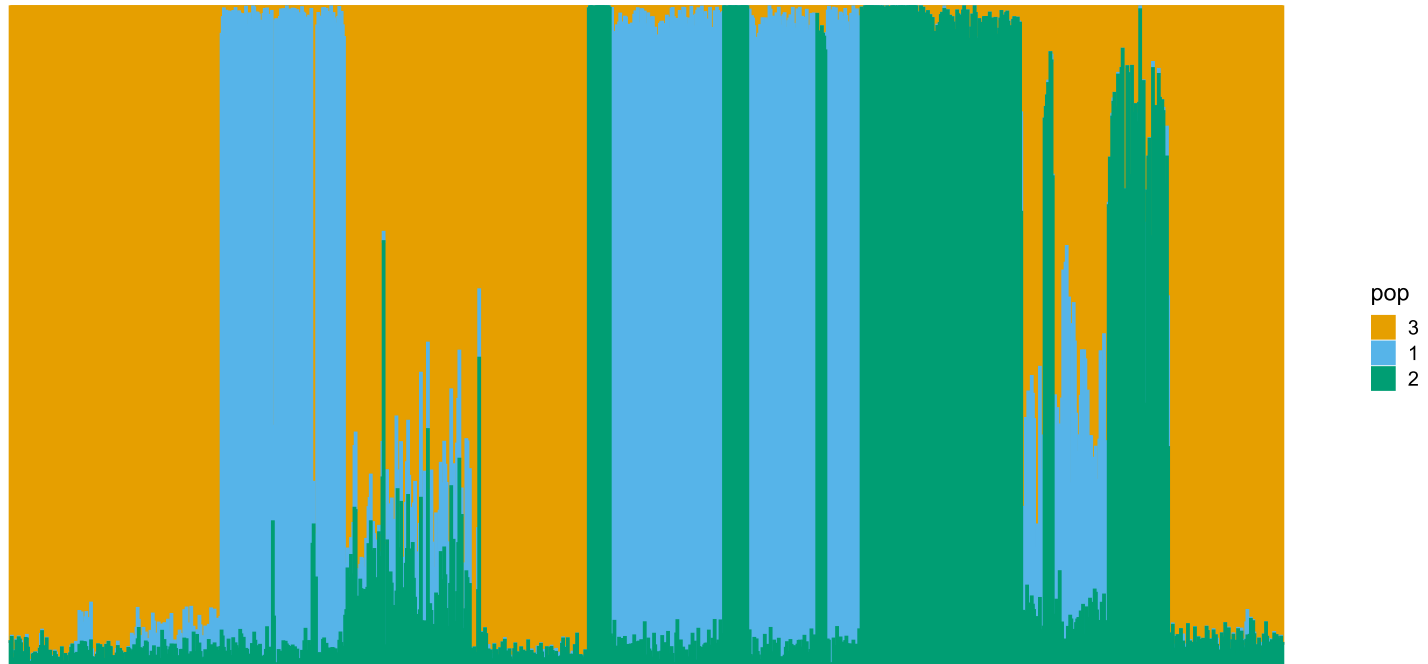
```
L <- result_TGP_vi$Loss  
plot_loss(list(L), "vi", 10)
```





We plot the ancestral proportions of individuals using package `ggplot2`. The fitting result is much better than EM algorithm.

```
P <- result_TGP_vi$P  
plot_structure(P)
```



We measure the prediction accuracy of the dataset by the evidence of lower bound (ELBO), the maximum likelihood function and the deviance residuals.

```
L[length(L)]
```

```
## [1] -0.6103088
```

```
psd_loglikelihood(data_TGP, result_TGP_vi)
```

```
## [1] -0.59751
```

```
psd_error(data_TGP, result_TGP_vi)
```

```
## [1] 0.3378423
```

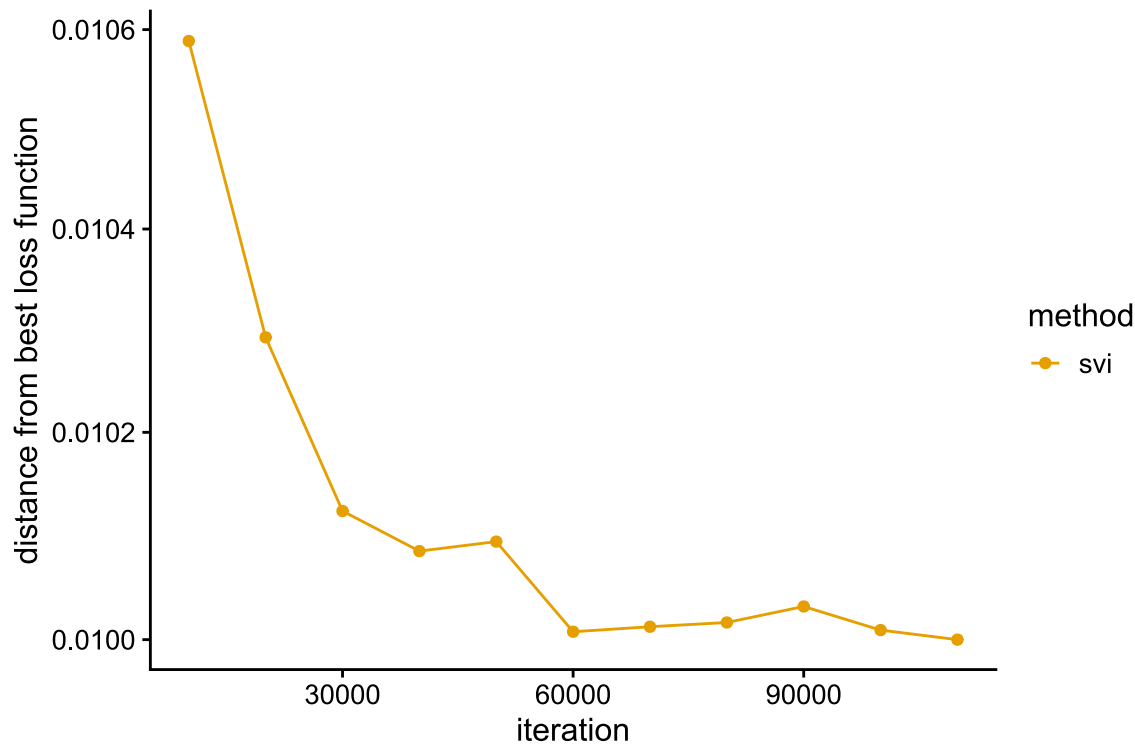
# SVI algorithm

We fit the PSD model with SVI algorithm, and use the loss function as a stopping criterion. We completed the fitting in 13 min, little longer than the previous three algorithms. Although SVI algorithm is known for its fast speed, this is for the dataset with a large number of individuals. It can be predicted that for the complete TGP data, SVI algorithm still needs about the same time, but the time of other algorithms will be greatly increased.

```
result_TGP_svi <- psd_fit_svi(data_TGP, 3,  
                             1e-5, 5e+5, 1e+4, 3,  
                             100, 2000,  
                             5e-2, 1e-1,  
                             1, 0.5)
```

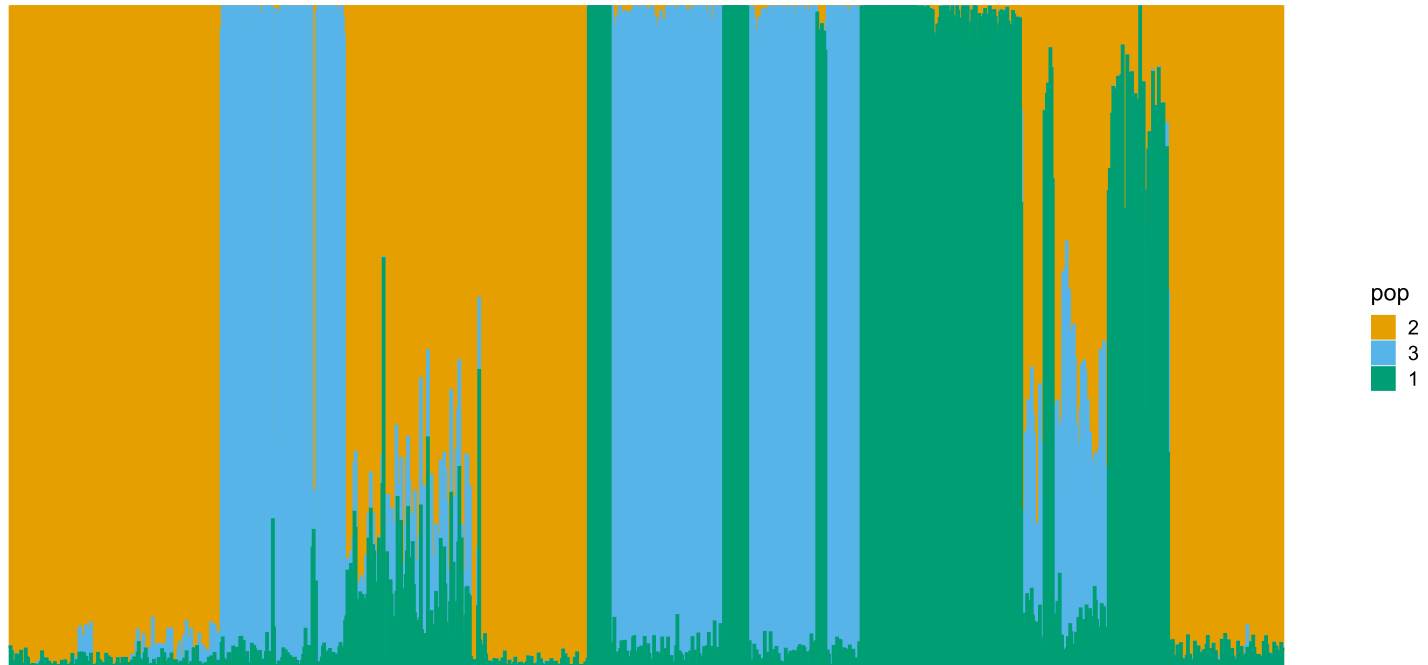
We plot the loss function against the number of iterations using package `ggplot2`, the loss function records once for 10,000 iterations.

```
L <- result_TGP_svi$Loss  
plot_loss(list(L), "svi", 1e+4)
```



We plot the ancestral proportions of individuals using package ggplot2. The convergence accuracy of SVI algorithm is also very good.

```
P <- result_TGP_svi$P  
plot_structure(P)
```



We measure the prediction accuracy of the dataset by the maximum likelihood function of the validation set.

```
result_TGP_svi$MaxLoss
```

```
## [1] -0.614379
```

# Structure plot

We analyze the structure diagrams of the above four algorithms for TGP and HGDP. We import the pre-trained results directly.

```
load("result/result_structure_plot.RData")
```

# TGP

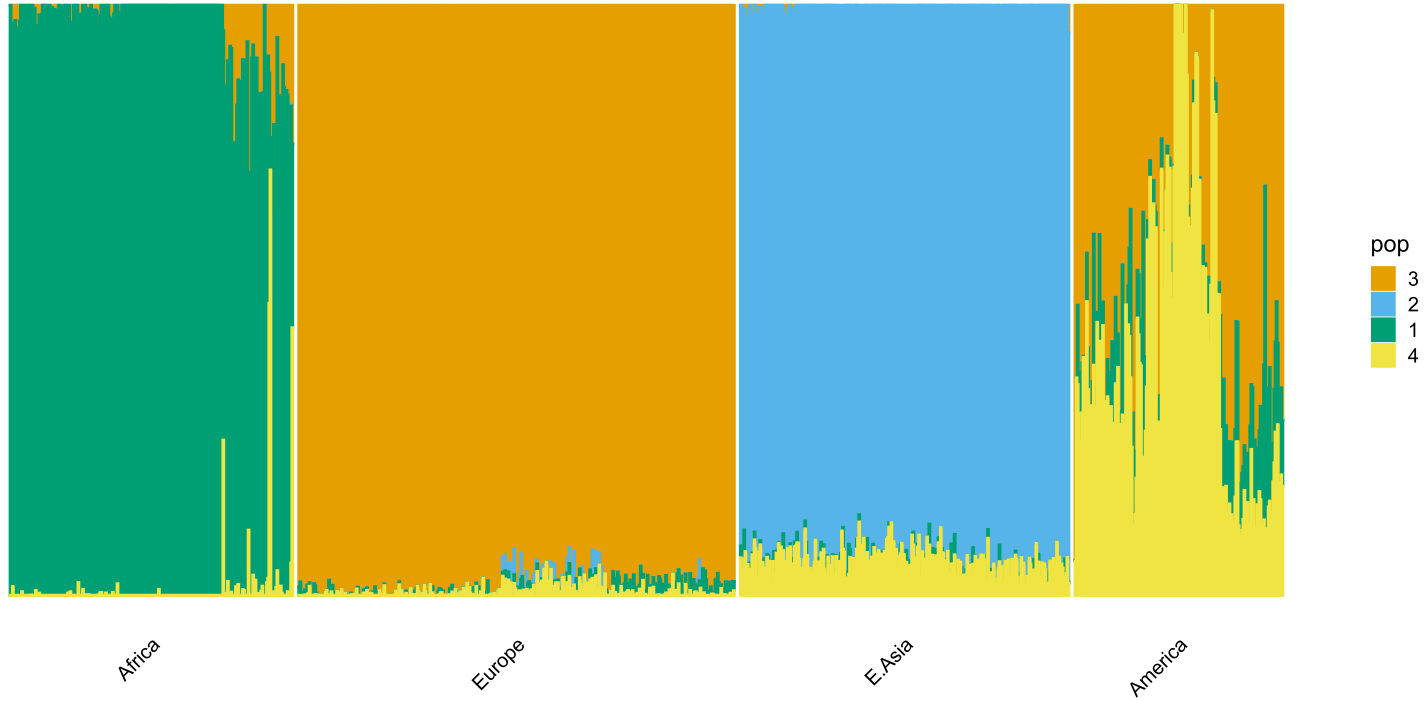
```
result_TGP <- psd_fit_vi(data_TGP, 4, 1e-5, 2000)
```

```
P <- result_TGP$P  
label <- rownames(data_TGP)  
lpop <- unlist(map_TGP[1])  
spop <- unlist(map_TGP[2])  
indiv <- unlist(map_TGP[3])
```



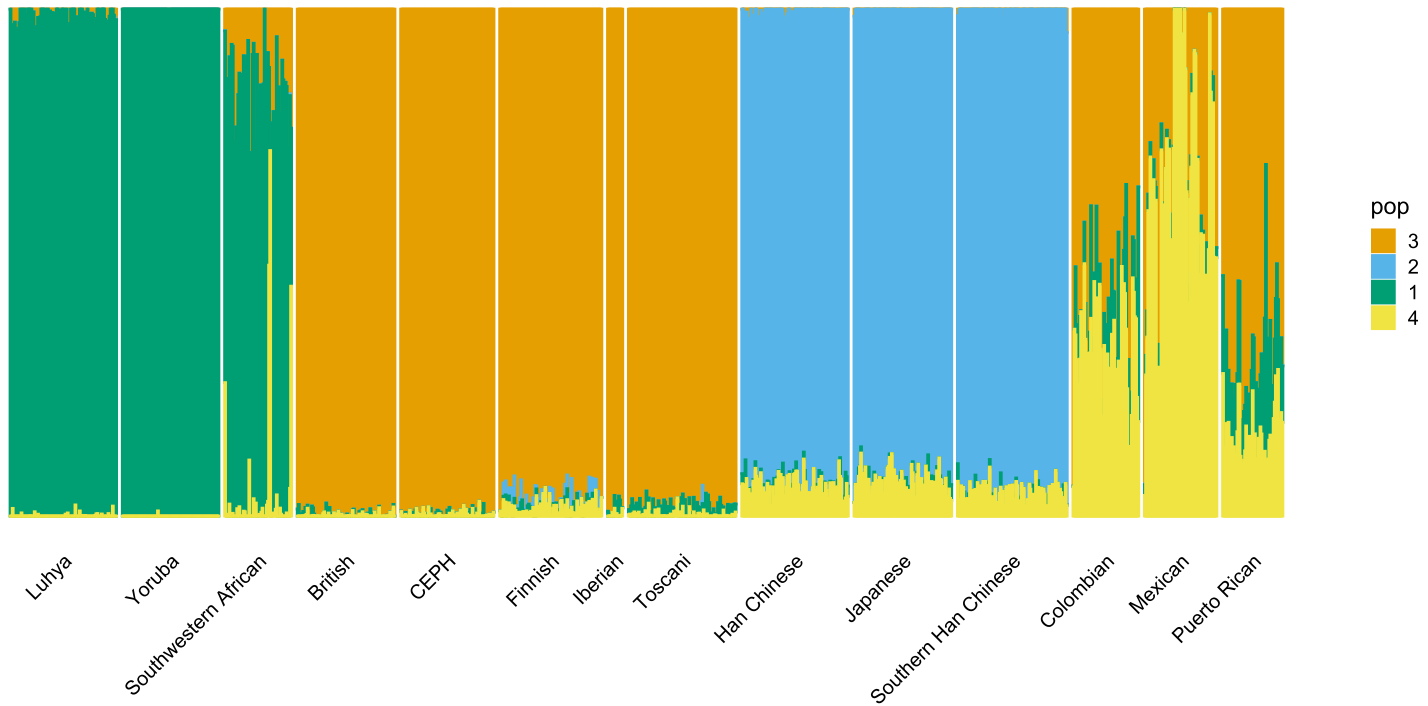
## Large populations.

```
plot_structure(P,  
              label = label,  
              map.indiv = indiv,  
              map.pop = lpop,  
              gap = 5)
```



## Small populations.

```
plot_structure(P,  
              label = label,  
              map.indiv = indiv,  
              map.pop = spop,  
              gap = 5)
```



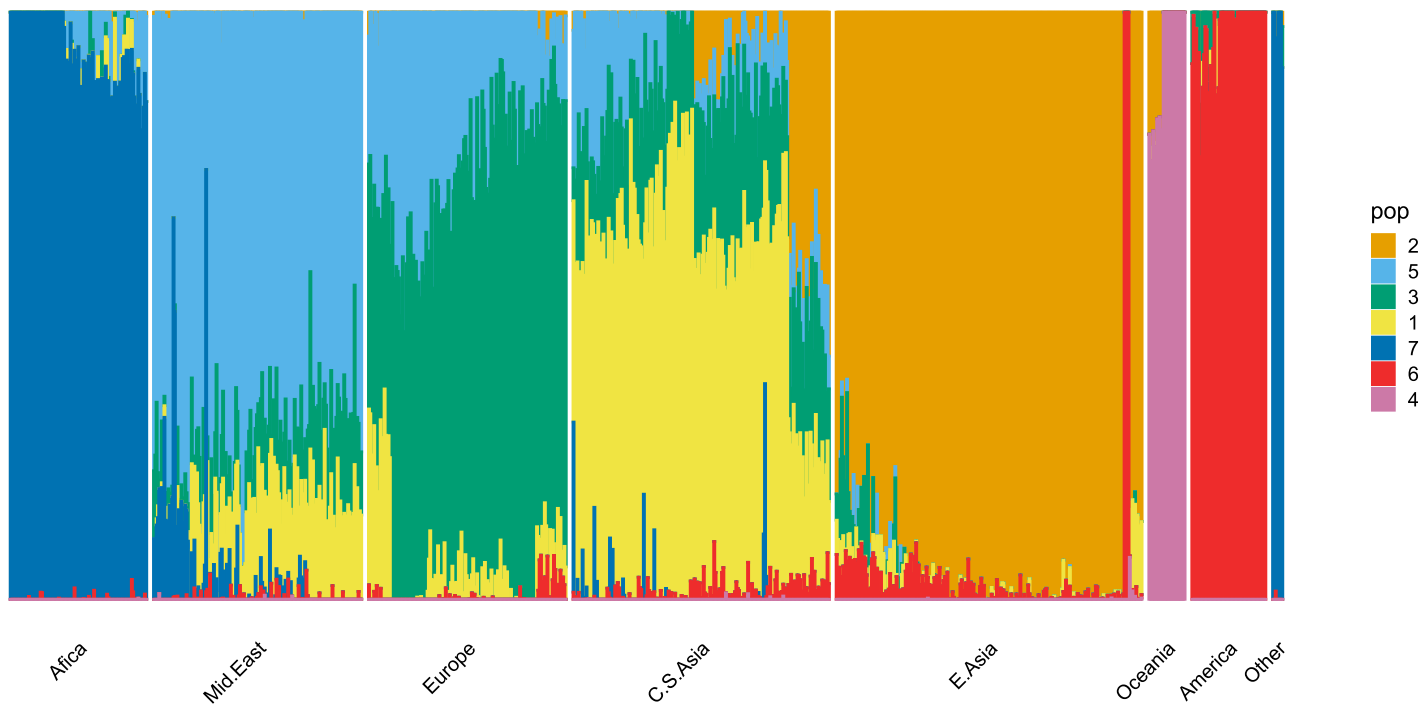
# HGDP

```
result_HGDP <- psd_fit_vi(data_HGDP, 7, 1e-5, 2000)
```

```
P <- result_HGDP$P  
label <- rownames(data_HGDP)  
lpop <- unlist(map_HGDP[1])  
spop <- unlist(map_HGDP[2])  
indiv <- unlist(map_HGDP[3])
```

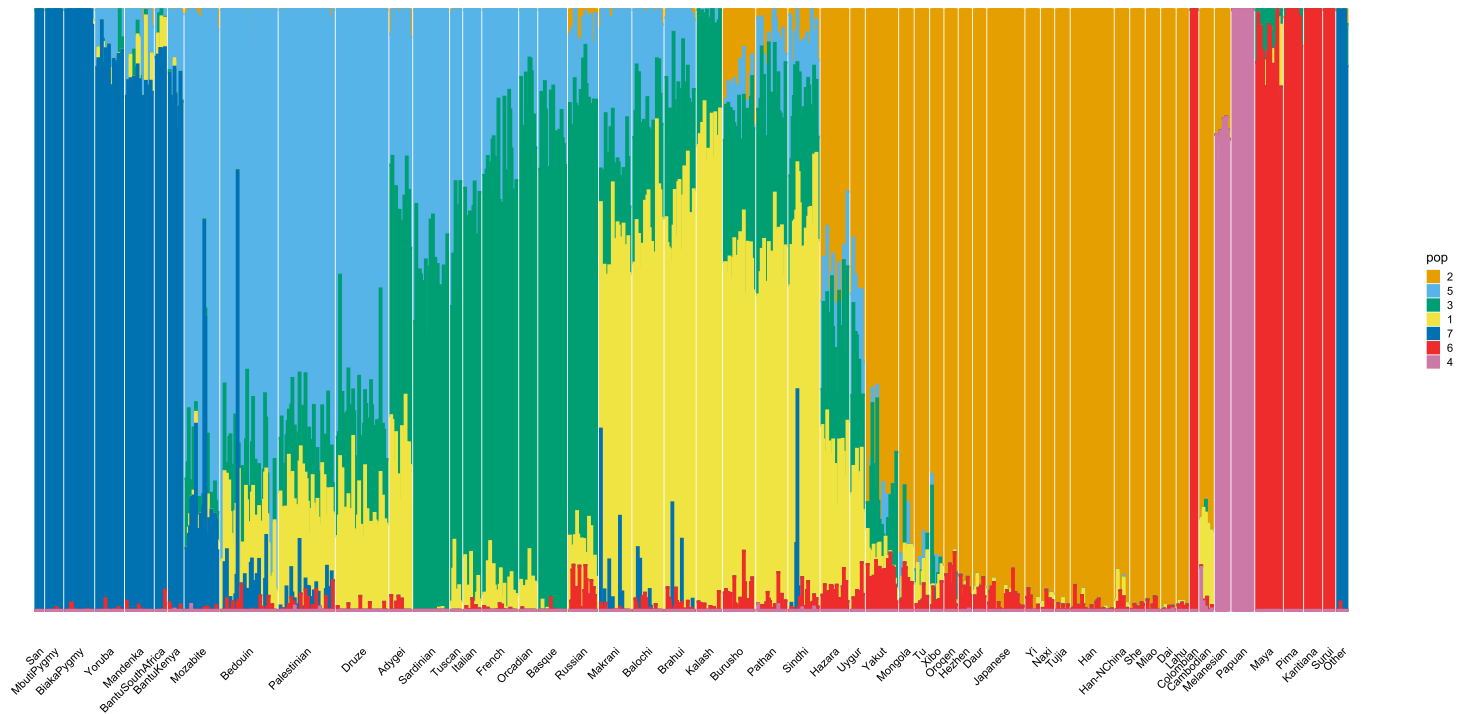
## Large populations.

```
plot_structure(P,  
              label = label,  
              map.indiv = indiv,  
              map.pop = lpop,  
              gap = 5)
```



# Small populations.

```
plot_structure(P,  
              label = label,  
              map.indiv = indiv,  
              map.pop = spop,  
              gap = 3,  
              font.size = 5)
```




# Choose hyper-parameter K

We fit different hyperparameters K respectively, and select the K that makes some indices optimal. We conduct experiments on TPG and HGDP datasets and different algorithms separately. See *Articles* in [AwesomePackage](#) for details.

# Conclusion

"I have to work very hard, very hard, in order to do a little success."

--- Stephen Chow



也不是, 天亮後便會很美的