# Graphic Models

Yang Le

2019年12月

## 概述

1.DGP
a) Ω：对角线元素ρ，非对角线的元素以0.01的概率等于0.5
b) 问题在于该Ω是否可以用来构造多元正态回归模型（关键在于常数ρ）（hint：条件数需要满足一定性质）。
2.测试Graphical Models中的Glasso和Neighborhood Selection
a) 测试方式：approx参数取T和F（即两种求解方法），测试两个方法选出来的变量是否符合真实的设定。
b) 选用两种指标（Frobenius norm，以及分类评估方法）来判断在模拟得的数据集下哪种方法结果比较好。
c) 对于Neighborhood Selection，只能选出有效集合，不能得到估计值，若用Frobenius norm进行评估，在Neighborhood Selection基础上做一个稀疏的极大似然估计 （glasso 设定rho=0，并指定zero）

## 测试Glasso、Neighborhood Selection

## 自定义函数

### 1) 作图：变量点与边

```
# 测试Glasso、Neighborhood Selection
# 采用Frobenius Norm和MCC评估
########自定义函数##############
# 1) 作图：变量点与边
edges.char<-function(plot.adj){ # 以字符串的形式表示边
  a<-which(plot.adj == T, arr.ind = T)
  a<-rbind(a,cbind(a[,2],a[,1]))
  a<-a[a[,1]>a[,2],]
  a<-a[!duplicated(a),]
  paste(a[,2],a[,1])
}
graphplot<-function(omegahat,omega=NA,main=NA){
  library(igraph)
  true.adj=abs(omegahat)>1e-6 # 如果omega的元素绝对值大于一个很小的数，认为点之间有边
  plot.adj=true.adj
  diag(plot.adj)=0
  temp=graph.adjacency(adjmatrix=plot.adj, mode="undirected")
  temp.degree=apply(plot.adj, 2, sum) # 每个节点边的个数
  V(temp)$color=(temp.degree>2)+3 # 边大于3条的节点颜色不同显示
  if(length(omega)>1){ # if需要与真实的Omega进行对比
    true.adj.omega=abs(omega)>1e-6
    plot.adj.omega=true.adj.omega
    diag(plot.adj.omega)=0
    temp.omega=graph.adjacency(adjmatrix=plot.adj.omega, mode="undirected")
    temp.degree.omega=apply(plot.adj.omega, 2, sum)
    V(temp.omega)$color=(temp.degree.omega>2)+3
    E(temp)$color=ifelse((edges.char(plot.adj) %in% edges.char(plot.adj.omega)),
                         "green","red") # 绿色：正确建立的边；红色：错误建立的边
```

```
    E(temp.omega)$color=ifelse((edges.char(plot.adj.omega) %in% edges.char(plot.adj)),
                               "green","gray") # 灰色：缺少的边
    par(mfrow=c(1,2))
    plot(temp.omega, vertex.size=3, vertex.frame.color="white",
         layout=layout.fruchterman.reingold,vertex.label=NA,
         main="True Omega")
    plot(temp, vertex.size=3, vertex.frame.color="white",
         layout=layout.fruchterman.reingold,vertex.label=NA,
         main="Omega-hat")
  }else{
    par(mfrow=c(1,1))
    plot(temp, vertex.size=3, vertex.frame.color="white",
         layout=layout.fruchterman.reingold,vertex.label=NA,edge.color=grey(0.5),
         main=main)
  }
}
```

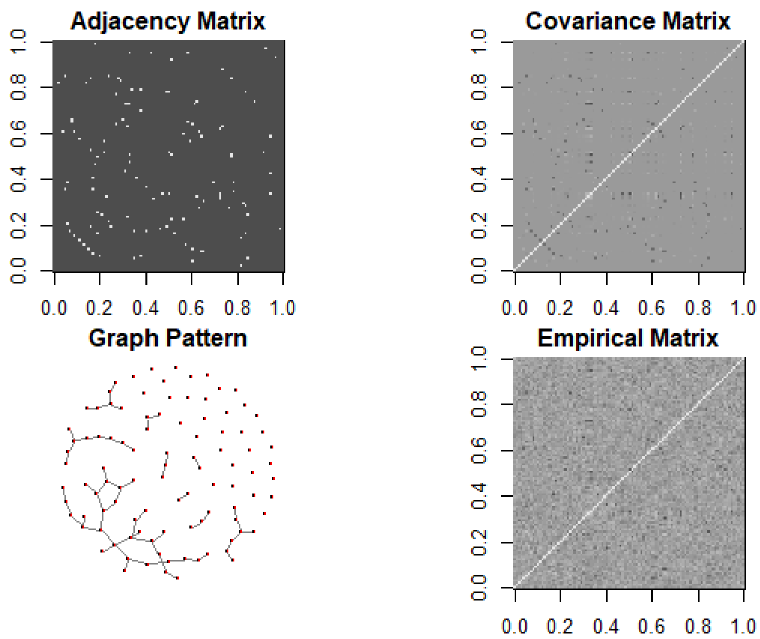
### 2) Frobenius Norm

```
# 如果给定了sigma参变量的值就用omegahat%*%sigma与I计算
FrobeniusNorm<-function(omega,omegahat,sigma=NA){
  if(length(sigma)==1) FN<-sqrt(sum((omega-omegahat)^2))
  else{
    FN<-sqrt(sum((omegahat%*%sigma-diag(1,nrow = nrow(omega)))^2))
  }
  return(FN)
}
```

## 1.构造数据

```
# Ω: 对角线元素p，非对角线的元素以0.01的概率等于0.5
n<-50 # 50条观测
p<-100 # 100个节点
par(mfrow=c(1,2))
library(flare) #Family of Lasso Regression
data<-sugm.generator(n = n, d = p, graph = "random", v = 0.5,
                     prob = 0.01, seed = 0, vis = TRUE)
```

```
## Generating data from the multivariate normal distribution with the random graph structure...
```



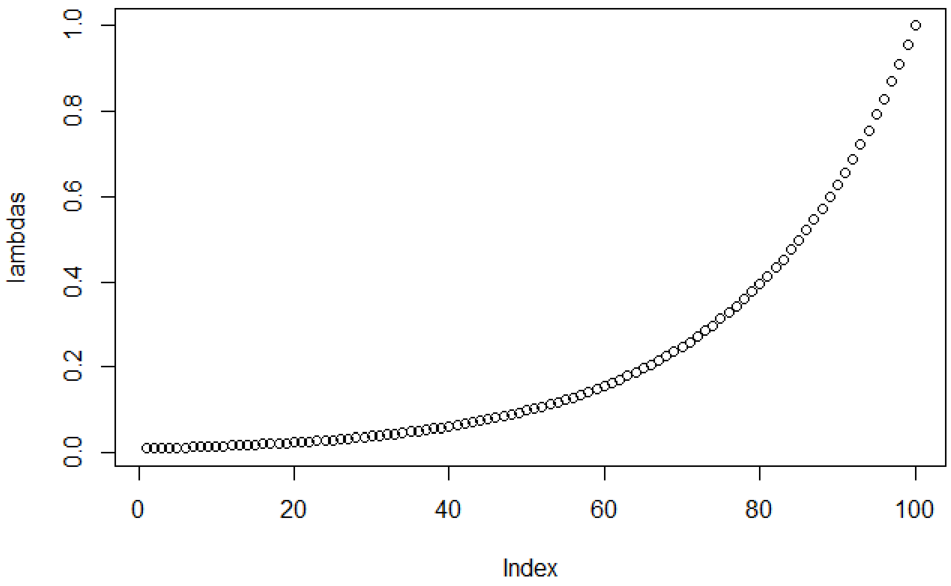

```
## done.
```

```
x<-data$data
omega<-data$omega
sigma<-data$sigma

# graphplot(omega) # 真实的omega确定的节点之间关系
s<-var(x)
```

## 2.测试Graphical Models中的Glasso

```
# lambda的设定
lambda.max<-1
lambda.min<-1e-2
lambdas<-seq(log(lambda.min),log(lambda.max),length.out = 100)
lambdas<-exp(lambdas)
plot(lambdas) # lambda在0.01到1之间取值
```




在不同的lambda取值下测试两种求解方法的结果，testGlasso函数返回给定lambda下glasso结果的frbnsnorm,tpr,fpro,mcc。

```
# 在不同的lambda取值下测试两种求解方法的结果
# testGlasso函数返回给定lambda下glasso结果的frbnsnorm,tpr,fpro,mcc
library(glasso)
library(ModelMetrics)
testGlasso<-function(lambda,s,omega,sigma=NA, approx=FALSE){
  library(glasso)
  library(ModelMetrics)
  weight<-glasso(s,lambda,approx=approx)
  omegahat<-weight$wi
  frbnsnorm<-FrobeniusNorm(omega,omegahat,sigma)
  tpr.glasso<-tpr(abs(omega)>1e-6,abs(omegahat)>1e-6)
  fpr.glasso<-1-tnr(abs(omega)>1e-6,abs(omegahat)>1e-6)
  mcc.glasso<-mcc(abs(omega)>1e-6,abs(omegahat)>1e-6,cutoff=0.5)
  return(c(lambda,frbnsnorm,tpr.glasso,fpr.glasso,mcc.glasso))
}
```

## 串行计算

```
# ## 串行计算
# # approx=FALSE
# system.time(
#   frbnsnorm.exact<-apply(as.array(lambdas), 1, testGlasso, s, omega, sigma, approx=FALSE)
# )
# # approx=TRUE
# system.time(
#   frbnsnorm.aprx<-apply(as.array(lambdas), 1, testGlasso, s, omega, sigma, approx=TRUE)
# )
```

## 并行计算

```
## 并行计算
library(parallel)
# 打开多核
cl <- makeCluster(getOption("cl.cores", 6))
```

```
clusterExport(cl,"FrobeniusNorm")
system.time({
  frbnsnorm.exact<-parApply(cl,as.array(lambdas), 1, testGlasso, s, omega, sigma, approx=FALSE)
})
```

```
##    user  system elapsed
##    0.03    0.00    4.14
```

```
system.time({
  frbnsnorm.aprx<-parApply(cl,as.array(lambdas), 1, testGlasso, s, omega, sigma, approx=TRUE)
})
```

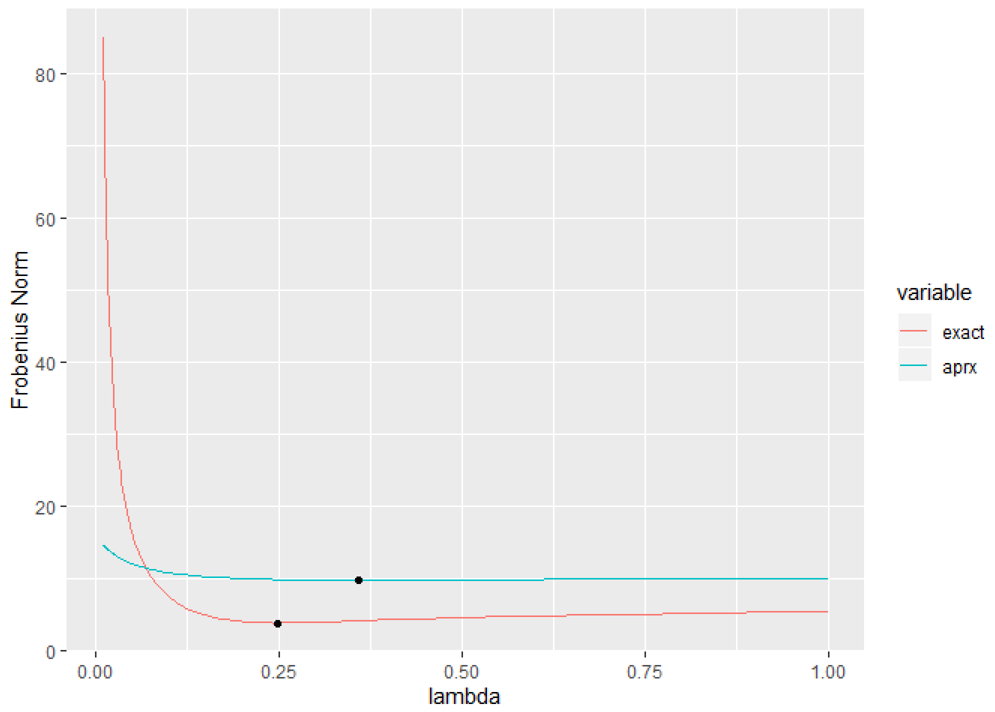
```
##    user  system elapsed
##    0.00    0.00    2.31
```

```
# 关闭并行计算
stopCluster(cl)
# 六核：3s;1.92s approx方法更快

# 结果整理
frbnsnorm<-data.frame(t(frbnsnorm.exact),t(frbnsnorm.aprx)[,2:5])
colnames(frbnsnorm)<-c("lambda","exact","exact.tpr",
                       "exact.fpr","exact.mcc","aprx","aprx.tpr","aprx.fpr","aprx.mcc")
```

## Frobenius Norm评估

```
# Frobenius Norm评估
# 不同的lambda取值下两种求解方法的估计效果
library(ggplot2)
library(reshape2)
library(dplyr)
p.fn.glasso<-frbnsnorm %>%
  select(lambda,exact,aprx) %>%
  melt(id="lambda") %>%
  ggplot(aes(x=lambda))+
  geom_line(aes(y=value,color=variable))+
  ylab("Frobenius Norm")+
  annotate("point",x=frbnsnorm[which.min(frbnsnorm$exact),1],
           y=min(frbnsnorm$exact))+
  annotate("point",x=frbnsnorm[which.min(frbnsnorm$aprx),1],
           y=min(frbnsnorm$aprx))
p.fn.glasso
```



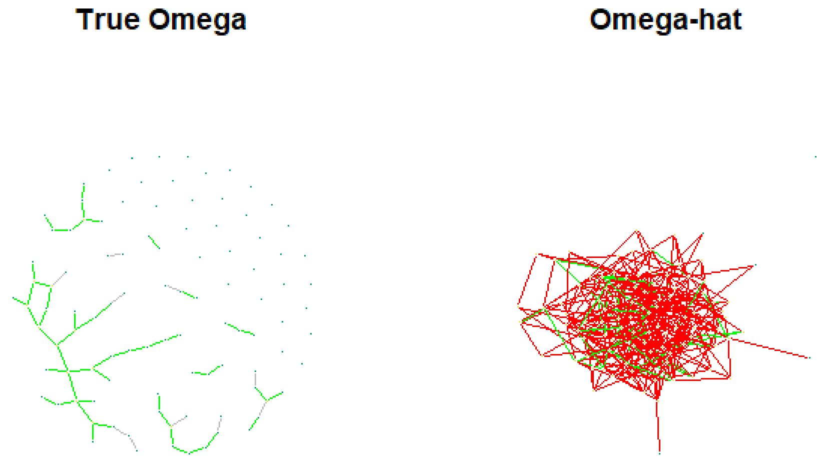

### 最优的lamda

```r
# 两种求解方法各自最优的Lambda
lambda.exact<-frbnsnorm[which.min(frbnsnorm$exact),1]
lambda.aprx<-frbnsnorm[which.min(frbnsnorm$aprx),1]
cat("min(Frobenius Norm)-exact method: best lambda =",
    lambda.exact," Frobenius Norm =",min(frbnsnorm$exact),
    "\nmin(Frobenius Norm)-approx method: best lambda =",
    lambda.aprx," Frobenius Norm =",min(frbnsnorm$aprx))
```

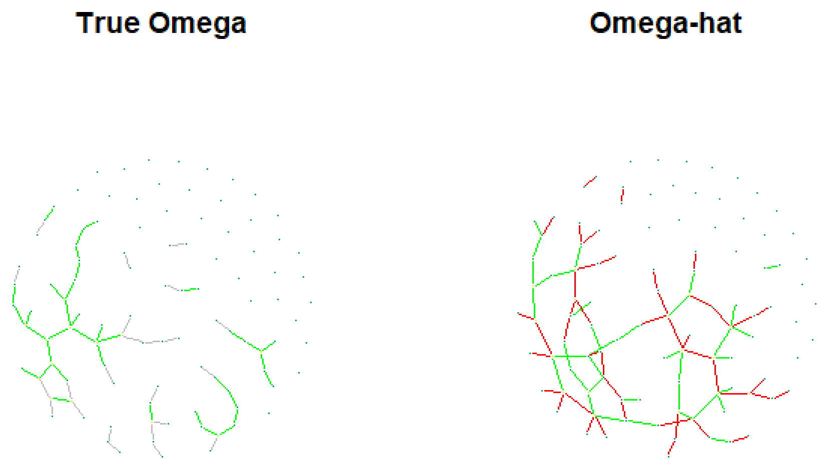
```
## min(Frobenius Norm)-exact method: best lambda = 0.2477076  Frobenius Norm = 3.92209
## min(Frobenius Norm)-approx method: best lambda = 0.3593814  Frobenius Norm = 9.850065
```

### 最优的lambda下的估计效果（exact 和 approx）

```r
# 两种求解方法各自最优的Lambda下的估计效果
weight.exact<-glasso(s,lambda.exact,approx=FALSE)
weight.aprx<-glasso(s,lambda.aprx,approx=TRUE)
graphplot(weight.exact$wi,omega,main="method: exact")
```




```
graphplot(weight.aprx$wi,omega,main="method: approx")# approx=T得到的omegahat更稀疏
```



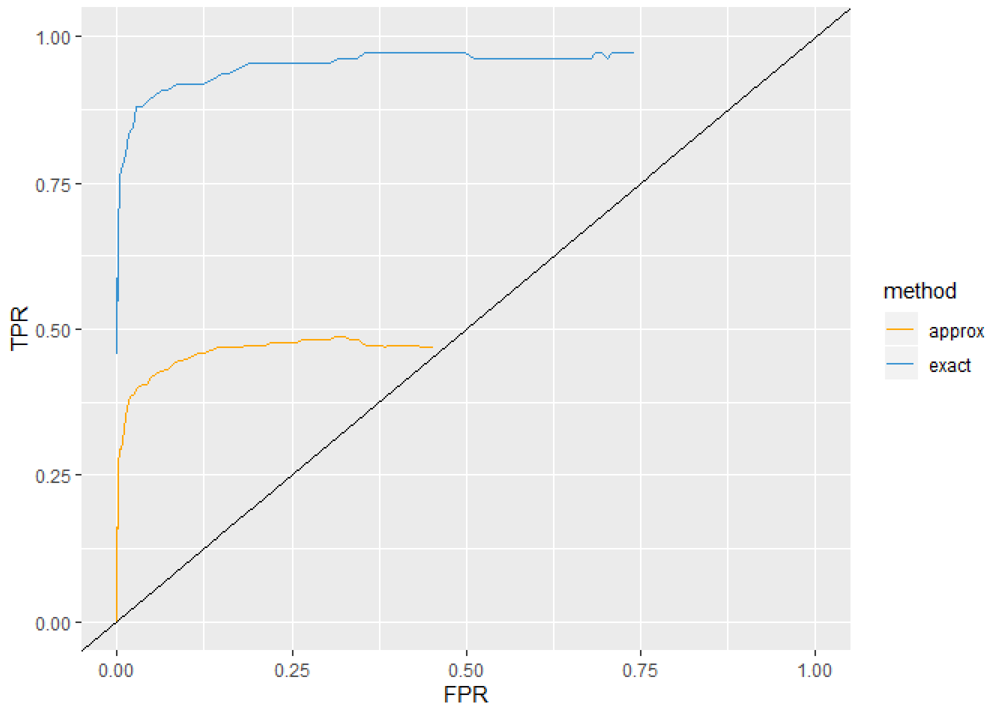

## ROC MCC 评估

（只考虑是否正确选出了相关变量）

### ROC曲线

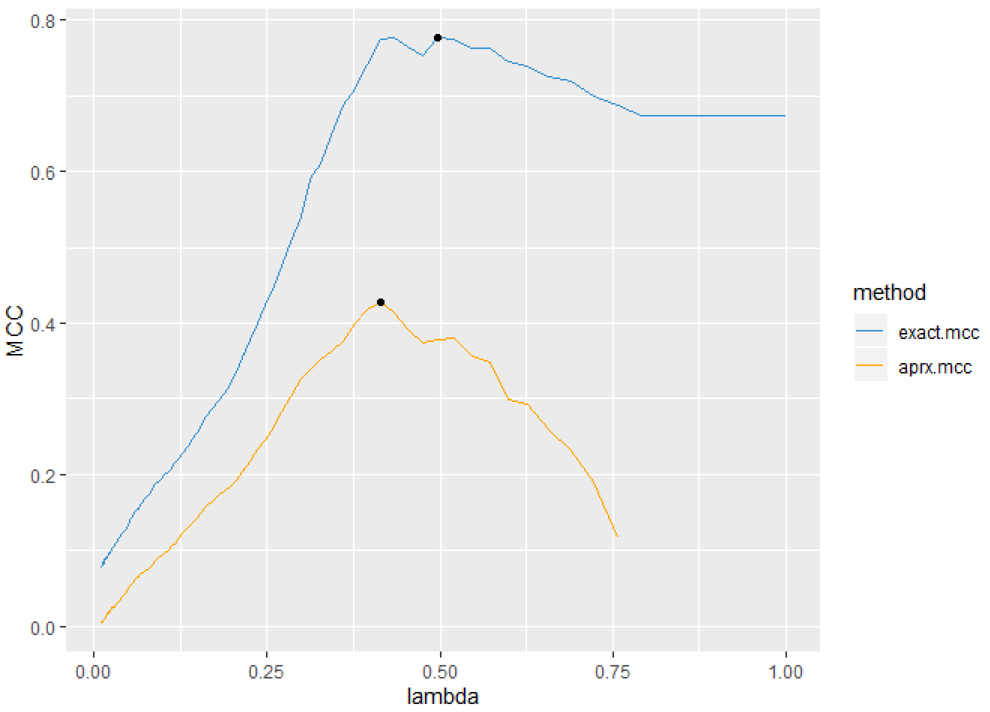
```
# ROC曲线
cols1=c('exact'="#3591d1", 'approx'="orange")
p.roc.glasso<-ggplot(data = frbnsnorm)+
  geom_line(aes(x=exact.fpr,y=exact.tpr,color="exact"))+
  geom_line(aes(x=aprx.fpr,y=aprx.tpr,color="approx"))+
  scale_colour_manual(name="method",values=cols1)+
  ylab("TPR")+xlab("FPR")+xlim(0,1)+ylim(0,1)+
```

```
  geom_abline(intercept=0,slope=1 )
p.roc.glasso
```




## MCC最大时的lambda

```
# MCC最大 最优的lambda
cols2=c('exact.mcc'="#3591d1", 'aprx.mcc'="orange")
p.mcc.glasso<-frbnsnorm %>%
  select(lambda,exact.mcc,aprx.mcc) %>%
  melt(id="lambda") %>%
  ggplot(aes(x=lambda))+
  geom_line(aes(y=value,color=variable))+
  ylab("MCC")+
  annotate("point",x=frbnsnorm[which.max(frbnsnorm$exact.mcc),1],
           y=frbnsnorm[which.max(frbnsnorm$exact.mcc),"exact.mcc"])+
  annotate("point",x=frbnsnorm[which.max(frbnsnorm$aprx.mcc),1],
           y=frbnsnorm[which.max(frbnsnorm$aprx.mcc),"aprx.mcc"])+
  scale_colour_manual(name="method",values=cols2)
p.mcc.glasso
```

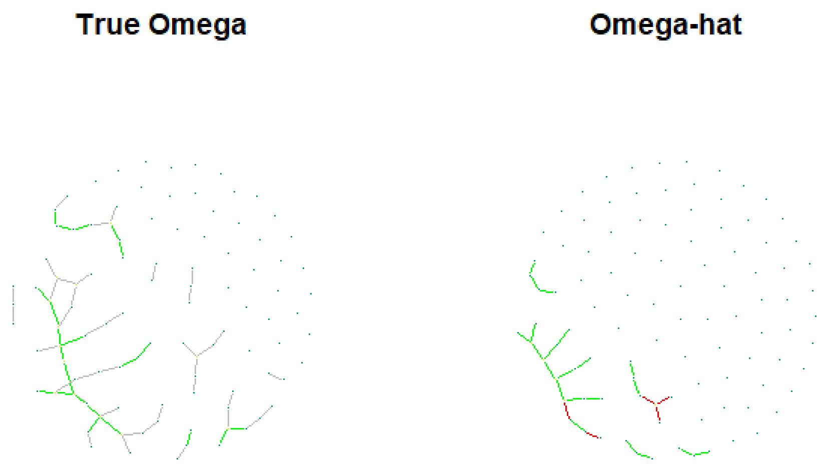



```
# 利用MCC最大得到两种求解方法各自最优的lambda
lambda.exact.mcc<-frbnsnorm[which.max(frbnsnorm$exact.mcc),1]
lambda.aprx.mcc<-frbnsnorm[which.max(frbnsnorm$aprx.mcc),1]
cat("max(MCC)-exact method: best lambda =",
    lambda.exact.mcc," MCC =",max(frbnsnorm$exact.mcc,na.rm = T),
    "\nmax(MCC)-approx method: best lambda =",
    lambda.aprx," MCC =",max(frbnsnorm$aprx.mcc,na.rm = T))
```

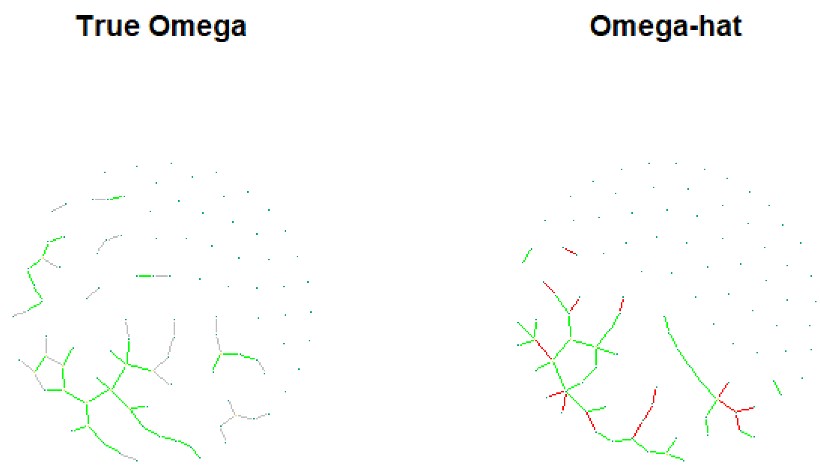
```
## max(MCC)-exact method: best lambda = 0.4977024  MCC = 0.7762399
## max(MCC)-approx method: best lambda = 0.3593814  MCC = 0.4268459
```

### 最优的lambda下的估计效果

```
# 利用MCC最大得到的各自最优的lambda下的估计效果
weight.exact.mcc<-glasso(s,lambda.exact.mcc,approx=FALSE)
weight.aprx.mcc<-glasso(s,lambda.aprx.mcc,approx=TRUE)
graphplot(weight.exact.mcc$wi,omega,main="method: exact") # 绿色: TP 绿色/灰色: TPR
```




```
graphplot(weight.aprx.mcc$wi,omega,main="method: approx") # 红色: FP
```




## 3.测试Space

```
# 不同Lambda下的结果
library(space)
testNS<-function(lambda,x,omega,sigma=NA){
  library(space)
  library(glasso)
  library(ModelMetrics)
  ns<-space.neighbor(x, lambda)
  ParCor<-ns$ParCor
  # TPR FPR MCC
  tpr.ns<-tpr(abs(omega)>1e-6,abs(ParCor)>1e-6)
  fpr.ns<-1-tnr(abs(omega)>1e-6,abs(ParCor)>1e-6)
  mcc.ns<-mcc(abs(omega)>1e-6,abs(ParCor)>1e-6,cutoff=0.5)
```

```r
  # 不存在条件相关的变量对
  iszero=which(abs(ParCor)<1e-6,arr.ind = T)
  # 在结果的基础上glasso, lambda=0
  weight.ns<-glasso(var(x),0,zero = iszero)
  omegahat.ns<-weight.ns$wi
  frbnsnorm.ns<-FrobeniusNorm(omega,omegahat.ns,sigma)
  return(c(lambda,frbnsnorm.ns,nrow(iszero),tpr.ns,fpr.ns,mcc.ns))
}
```

## 串行计算与并行计算

```r
# ## 串行计算
# system.time(
#   frbnsnormNS<-apply(as.array(lambdas), 1, testNS,x,omega,sigma)
# )
## 并行计算
# 打开多核
cl <- makeCluster(getOption("cl.cores", 6))
system.time({
  clusterExport(cl,"FrobeniusNorm")
  frbnsnormNS<-parApply(cl,as.array(lambdas), 1, testNS,x,omega,sigma)
})
```

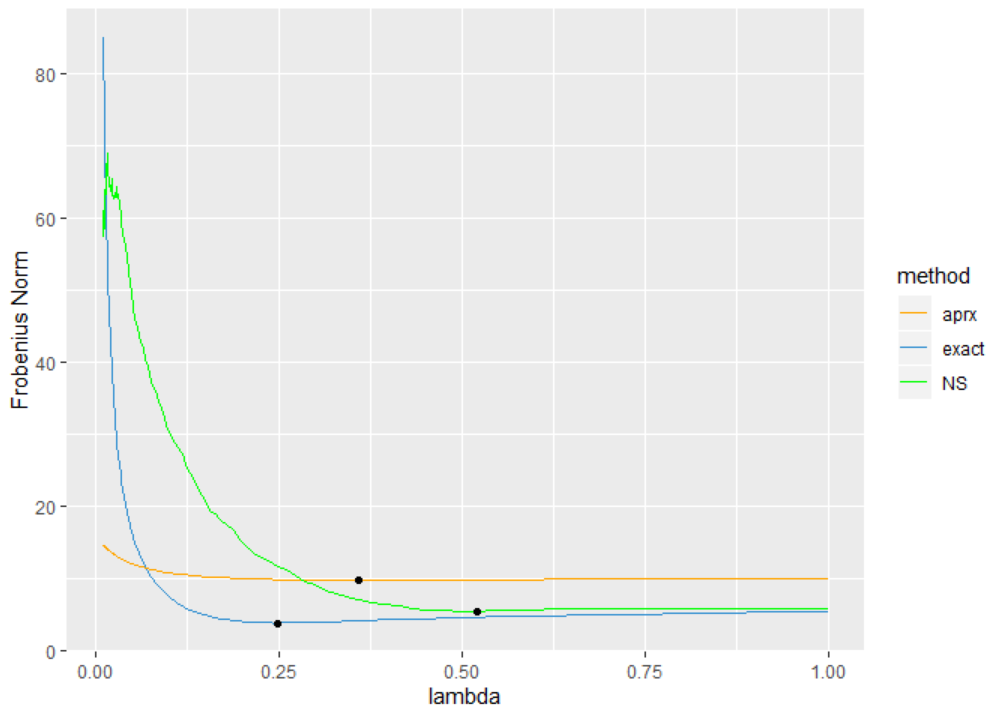
```
##    user  system elapsed
##    0.00    0.02    7.18
```

```r
# 关闭并行计算
stopCluster(cl)
frbnsnormNS<-data.frame(t(frbnsnormNS))
colnames(frbnsnormNS)<-c("lambda","frbnsnorm","n_zero",
                         "tpr.ns","fpr.ns","mcc.ns")
```

## Frobenius Norm评估

### 结果比较

```r
# 与glasso的结果比较
cols1=c('exact'="#3591d1", 'aprx'="orange","NS"="green")
p.fn.all<-p.fn.glasso+
  geom_line(data=frbnsnormNS,aes(x=lambda,y=frbnsnorm,color="NS"))+
  scale_colour_manual(name="method",values=cols1)+
  annotate("point",x=frbnsnormNS[which.min(frbnsnormNS$frbnsnorm),1],
           y=min(frbnsnormNS$frbnsnorm))
p.fn.all
```



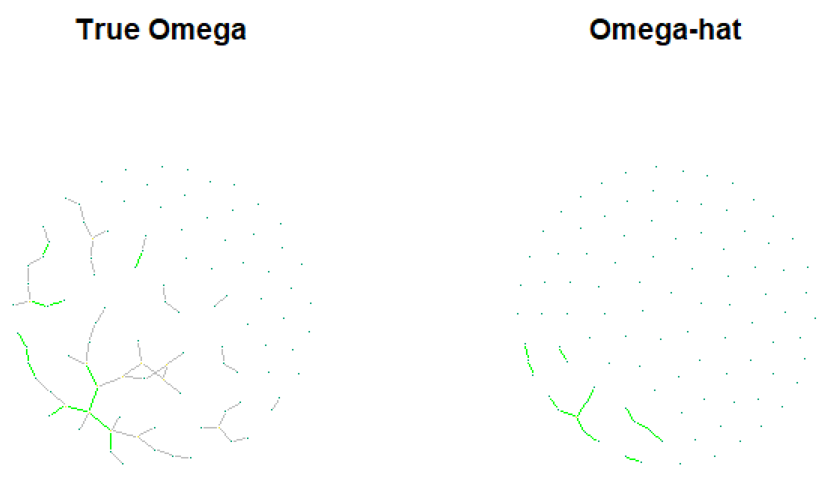

## Frobenius Norm最小的lambda

```r
# Frobenius Norm最小的Lambda
lambda.ns<-frbnsnorm[which.min(frbnsnormNS$frbnsnorm),1]
cat("min(Frobenius Norm)-exact method: best lambda =",
    lambda.exact," Frobenius Norm =",min(frbnsnorm$exact),
    "\nmin(Frobenius Norm)-approx method: best lambda =",
    lambda.aprx," Frobenius Norm =",min(frbnsnorm$aprx),
    "\nmin(Frobenius Norm)-Space: best lambda =",
    lambda.ns," Frobenius Norm =",min(frbnsnormNS$frbnsnorm))
```

```
## min(Frobenius Norm)-exact method: best lambda = 0.2477076  Frobenius Norm = 3.92209
## min(Frobenius Norm)-approx method: best lambda = 0.3593814  Frobenius Norm = 9.850065
## min(Frobenius Norm)-Space: best lambda = 0.5214008  Frobenius Norm = 5.563073
```

## 最优的lambda下的估计效果

```r
ns<-space.neighbor(x, lambda.ns)
iszero=which(abs(ns$ParCor)<1e-6,arr.ind = T)
#weight.ns<-glasso(var(x),0,zero = iszero)
graphplot(ns$ParCor,omega,main="Space")
```

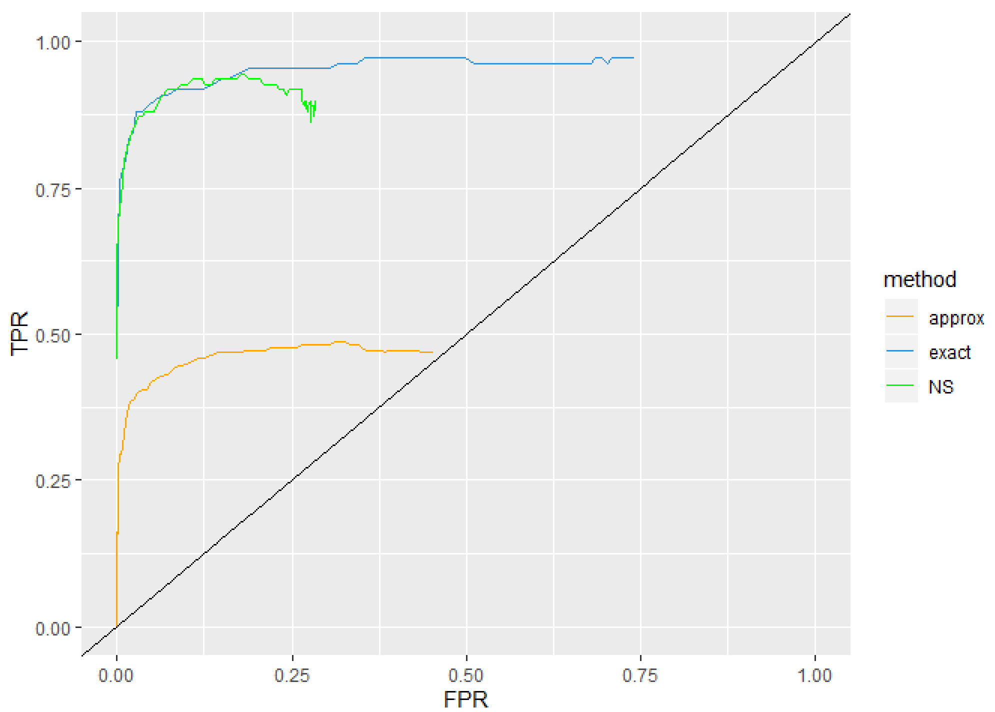



## ROC MCC 评估

（只考虑是否正确选出了相关变量）

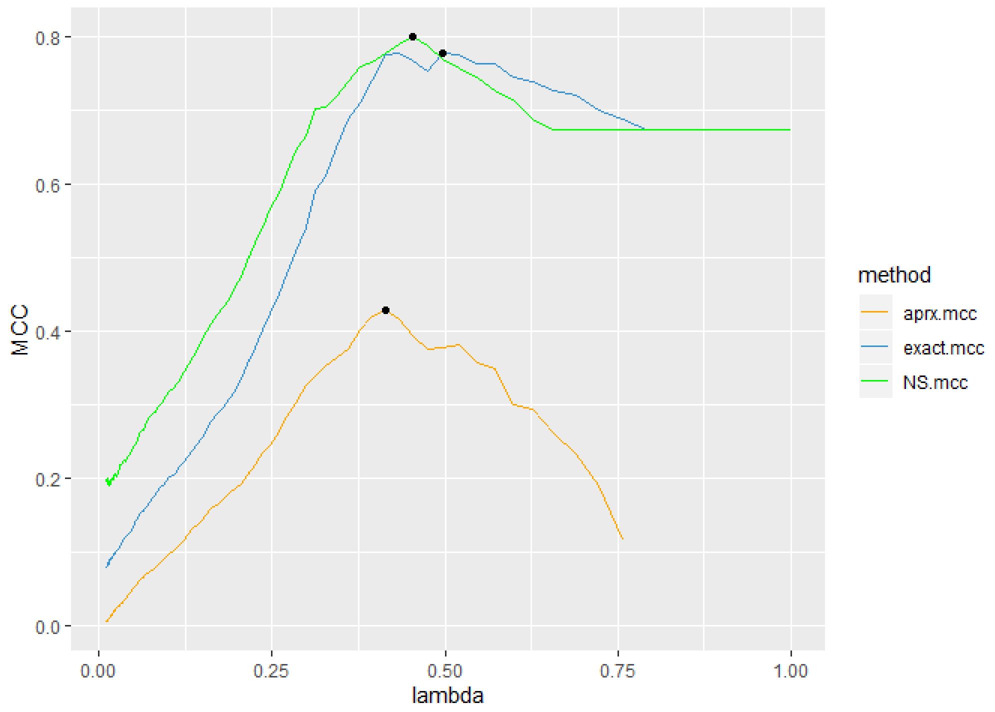
### ROC曲线

```
cols1=c('exact'="#3591d1", 'approx'="orange","NS"="green")
p.roc.all<-p.roc.glasso+
  geom_line(data=frbnsnormNS,aes(x=fpr.ns,y=tpr.ns,color="NS"))+
  scale_colour_manual(name="method",values=cols1)
p.roc.all
```




```
cols2=c('exact.mcc'="#3591d1", 'aprx.mcc'="orange", "NS.mcc"="green")
p.mcc.all<-p.mcc.glasso+
  geom_line(data=frbnsnormNS,aes(y=mcc.ns,color="NS.mcc"))+
  annotate("point",x=frbnsnormNS[which.max(frbnsnormNS$mcc.ns),1],
```

```
           y=max(frbnsnormNS$mcc.ns))+
  scale_colour_manual(name="method",values=cols2)
p.mcc.all
```



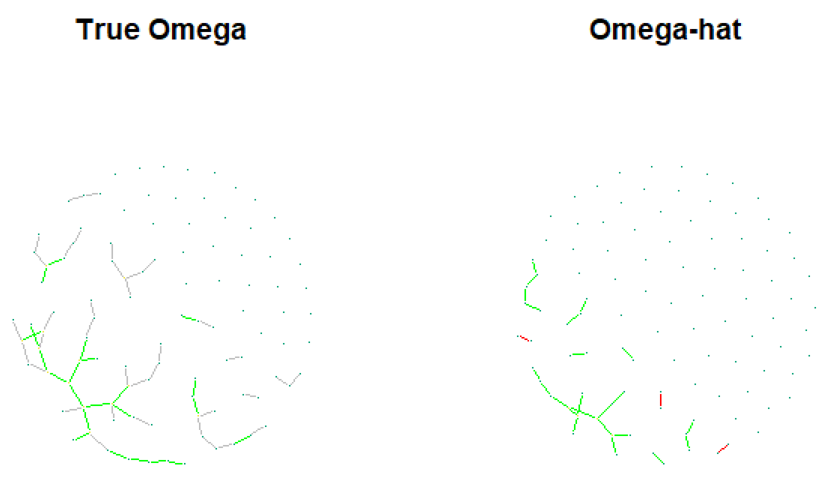

### MCC最大得到的最优的lambda

```
# 利用MCC最大得到的最优的lambda
lambda.nc.mcc<-frbnsnormNS[which.max(frbnsnormNS$mcc.ns),1]
cat("max(MCC)-exact method: best lambda =",
    lambda.exact.mcc," MCC =",max(frbnsnorm$exact.mcc,na.rm = T),
    "\nmax(MCC)-approx method: best lambda =",
    lambda.aprx," MCC =",max(frbnsnorm$aprx.mcc,na.rm = T),
    "\nmax(MCC)-Space: best lambda =",
    lambda.nc.mcc," MCC =",max(frbnsnormNS$mcc.ns,na.rm = T))
```

```
## max(MCC)-exact method: best lambda = 0.4977024  MCC = 0.7762399
## max(MCC)-approx method: best lambda = 0.3593814  MCC = 0.4268459
## max(MCC)-Space: best lambda = 0.4534879  MCC = 0.7986283
```

### 最优的lambda下的估计效果

```
# MCC最大的最优的lambda下的估计效果
ns<-space.neighbor(x, lambda.nc.mcc)
iszero=which(abs(ns$ParCor)<1e-6,arr.ind = T)
#weight.ns<-glasso(var(x),0,zero = iszero)
graphplot(ns$ParCor,omega,main="Space")
```




## 彩蛋：快乐码代码之利用R拼乐高

```
#devtools::install_github("ryantimpe/brickr")
library(beepr)
library(brickr)
library(dplyr)
library(rayshader)

#一层两排
cc = data.frame(
  Level = "A",
  X1 = rep(1,4),
  X2 = rep(1,4)
)
cc %>% bricks_from_table() %>%
  build_bricks()


#加颜色
lego_colors
build_colors()
cc_colors = data.frame(
  .value = 1,
  Color = "Bright blue"
)
cc %>%
   bricks_from_table(cc_colors)%>%
   build_bricks()

#两层两排
cc = data.frame(
  Level = c(rep("A",4),rep("B",4)),
  x1 = rep(1,4),  #A和B对应不同的层
  x2 = rep(2,4) #1和2分别对应不同的颜色
)

cc_colors = data.frame(
  .value = c(1,2),
  Color = c("Bright blue","Bright red")
```

```
)

cc %>%
  bricks_from_table(cc_colors)%>%
  build_bricks()


#多层
1:10 %>%
  purrr::map_df(~dplyr::mutate(cc,Level=LETTERS[.x],x1=.x,x2=.x))%>%
  bricks_from_table()%>%
  build_bricks()


#使用bricks_from_coords()以编程方式构建3D LEGO模型，而不是通过表格手动绘。使用该函数，必须要提供x，y和z的坐
sphere_coords<-expand.grid(
  x=1:10,
  y=1:10,
  z=1:10
)%>%
  mutate(
    dist=sqrt(((x-mean(x))^2+(y-mean(y))^2+(z-mean(z))^2)),
    Color=case_when(
      between(dist,0,5) & (x+y+z)%%6 %in% 0:1~"Bright red",
      between(dist,5,6)~"Bright green"
    ))

sphere_coords%>%
  bricks_from_coords()%>%
  build_bricks()
beep("mario")


#用乐高拼一个马里奥
#mario
mario <- jpeg::readJPEG("D:/download/google/mario.jpg")
plot_map(mario)

#2d马赛克
mario1 = jpeg::readJPEG("D:/download/google/mario.jpg") %>%
  image_to_mosaic(img_size = 50)
mario1 %>% build_mosaic()

#建立乐高马赛克拼图过程图集
mario1%>%build_instructions()

#生成所有必需的插图块的形状和数量。这些插图块按颜色和大小排序,可以在乐高官网的Pick-a-Brick购买
mario1%>%build_pieces()
mario1%>%build_pieces_table()  #形成数据框


#手动拼一个马里奥
x1 = rep(1,25)
x2 = x3 = x4 = x1
x5 = c(rep(2,5),rep(1,15),rep(2,5))
x6 = x7 = x8 = x9 = x10 = x11 = x5
x12 = c(rep(2,4),rep(3,4),rep(1,9),rep(3,4),rep(2,4))
x13 = c(rep(2,4),rep(3,4),rep(2,9),rep(3,4),rep(2,4))
x14 = x13
x15 = c(rep(2,4),rep(1,4),rep(2,9),rep(1,4),rep(2,4))
x16  = x15
```

```r
x17 = c(rep(2,4),rep(1,4),rep(2,2),rep(1,5),rep(2,2),rep(1,4),rep(2,4))
x18 = c(rep(2,4),rep(1,4),rep(4,9),rep(1,4),rep(2,4))
x19 = c(rep(2,4),rep(1,3),rep(4,11),rep(1,3),rep(2,4))
x20 = c(rep(2,4),rep(1,1),rep(4,15),rep(1,1),rep(2,4))
x21 = c(rep(2,4),rep(4,17),rep(2,4))
x22 = c(rep(0,3),rep(4,6),rep(5,7),rep(4,6),rep(0,3))
x23 = c(rep(0,3),rep(4,4),rep(5,11),rep(4,4),rep(0,3))
x24 = c(rep(0,2),rep(4,4),rep(5,5),rep(6,4),rep(5,5),rep(4,3),rep(0,2))
x25 = c(rep(0,2),rep(4,1),rep(5,5),rep(4,1),rep(6,8),rep(4,1),rep(5,4),rep(4,2),rep(0,1))
x26 = c(rep(0,1),rep(4,2),rep(5,3),rep(4,3),rep(6,8),rep(4,3),rep(5,3),rep(4,1),rep(0,1))
x27 = c(rep(4,3),rep(5,1),rep(4,5),rep(6,8),rep(4,5),rep(5,1),rep(4,2))
m1 = data.frame(
  Level = "A",
  x1,x2,x3,x4,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16,x17,x18,x19,x20,x21,x22,x23,x24,x25,x26,x27
)
m_colors = data.frame(
  .value = c(1,2,3,4,5,6,7,8,9,10,11),
  Color = c("Dark azur","Bright red","Bright yellow","Light nougat","Dark brown","Nougat",
            "Brick yellow","White","Black","Dark stone grey","Dark red")
)
# m1 %>% bricks_from_table(m_colors) %>%
#   build_bricks()

a = c("A",rep(0,27))
m2 = rbind(a,a,m1,a,a)
m2[,-1] = apply(m2[,-1],2,as.numeric)
# m2 %>% bricks_from_table(m_colors) %>%
#   build_bricks()

x28 = c(rep(0,1),rep(7,3),rep(4,4),rep(8,2),rep(4,1),rep(6,8),rep(4,1),rep(8,2),rep(4,4),rep(7,2),re
x29 = c(rep(7,3),rep(5,1),rep(4,3),rep(8,3),rep(9,1),rep(6,8),rep(9,1),rep(8,3),rep(4,3),rep(5,1),re
x30 = c(rep(7,3),rep(5,1),rep(4,2),rep(8,3),rep(10,1),rep(9,2),rep(10,1),rep(6,4),rep(10,1),rep(9,2)
x31 = c(rep(7,3),rep(5,2),rep(4,1),rep(8,3),rep(10,1),rep(9,2),rep(10,1),rep(4,4),rep(10,1),rep(9,2)
x32 = c(rep(0,1),rep(7,2),rep(5,2),rep(4,1),rep(8,3),rep(10,2),rep(9,1),rep(10,1),rep(4,4),rep(10,1)
x33 = c(rep(0,1),rep(7,2),rep(5,2),rep(4,2),rep(8,3),rep(10,2),rep(4,6),rep(10,2),rep(8,3),rep(4,2),
x34 = c(rep(0,1),rep(2,2),rep(5,2),rep(4,2),rep(8,5),rep(4,6),rep(8,5),rep(4,2),rep(5,1),rep(2,2),re
x35 = c(rep(0,1),rep(2,2),rep(5,2),rep(4,4),rep(8,2),rep(4,1),rep(7,2),rep(4,2),rep(7,2),rep(4,1),re
x36 = c(rep(0,1),rep(2,2),rep(11,2),rep(6,4),rep(7,2),rep(5,1),rep(7,1),rep(4,4),rep(7,1),rep(5,1),r
x37 = c(rep(0,1),rep(2,2),rep(11,4),rep(6,1),rep(5,3),rep(7,1),rep(4,6),rep(7,1),rep(5,3),rep(6,1),r
x38 = c(rep(0,1),rep(2,3),rep(11,6),rep(6,1),rep(4,8),rep(6,1),rep(11,5),rep(2,3),rep(0,1))
x39 = c(rep(0,2),rep(2,2),rep(11,21),rep(2,2),rep(0,2))
x40 = c(rep(0,2),rep(2,4),rep(11,17),rep(2,4),rep(0,2))
x41 = c(rep(0,2),rep(2,5),rep(11,15),rep(2,5),rep(0,2))
x42 = c(rep(0,3),rep(2,7),rep(11,9),rep(2,7),rep(0,3))
x43 = c(rep(0,3),rep(2,23),rep(0,3))
x44 = c(rep(0,4),rep(2,21),rep(0,4))
x45 = c(rep(0,4),rep(2,8),rep(8,5),rep(2,8),rep(0,4))
x46 = c(rep(0,5),rep(2,6),8,2,8,2,8,2,8,rep(2,6),rep(0,5))
x47 = c(rep(0,6),rep(2,5),8,8,2,8,2,8,8,rep(2,5),rep(0,6))
x48 = c(rep(0,7),rep(2,4),rep(8,7),rep(2,4),rep(0,7))
x49 = c(rep(0,9),rep(2,3),rep(8,5),rep(2,3),rep(0,9))
x50 = c(rep(0,10),rep(2,9),rep(0,10))

m2 %>% cbind(x28,x29,x30,x31,x32,x33,x34,x35,x36,x37,x38,x39,x40,x41,x42,x43,x44,x45,x46,x47,x48,x49,
  bricks_from_table(m_colors) %>%
  build_bricks()
beep("mario")

#3d马赛克
mario1%>%bricks_from_mosaic()%>%build_bricks()
beep("mario")
```