刘程华 2018011687 计91

Prob1. For $f(x)=\left(10x_1^2+x_2^2\right)/2$, try three step size rules on $\mathrm{p}.71$ respectively. Present figures like those on p. 74 and $\mathrm{p}.75$

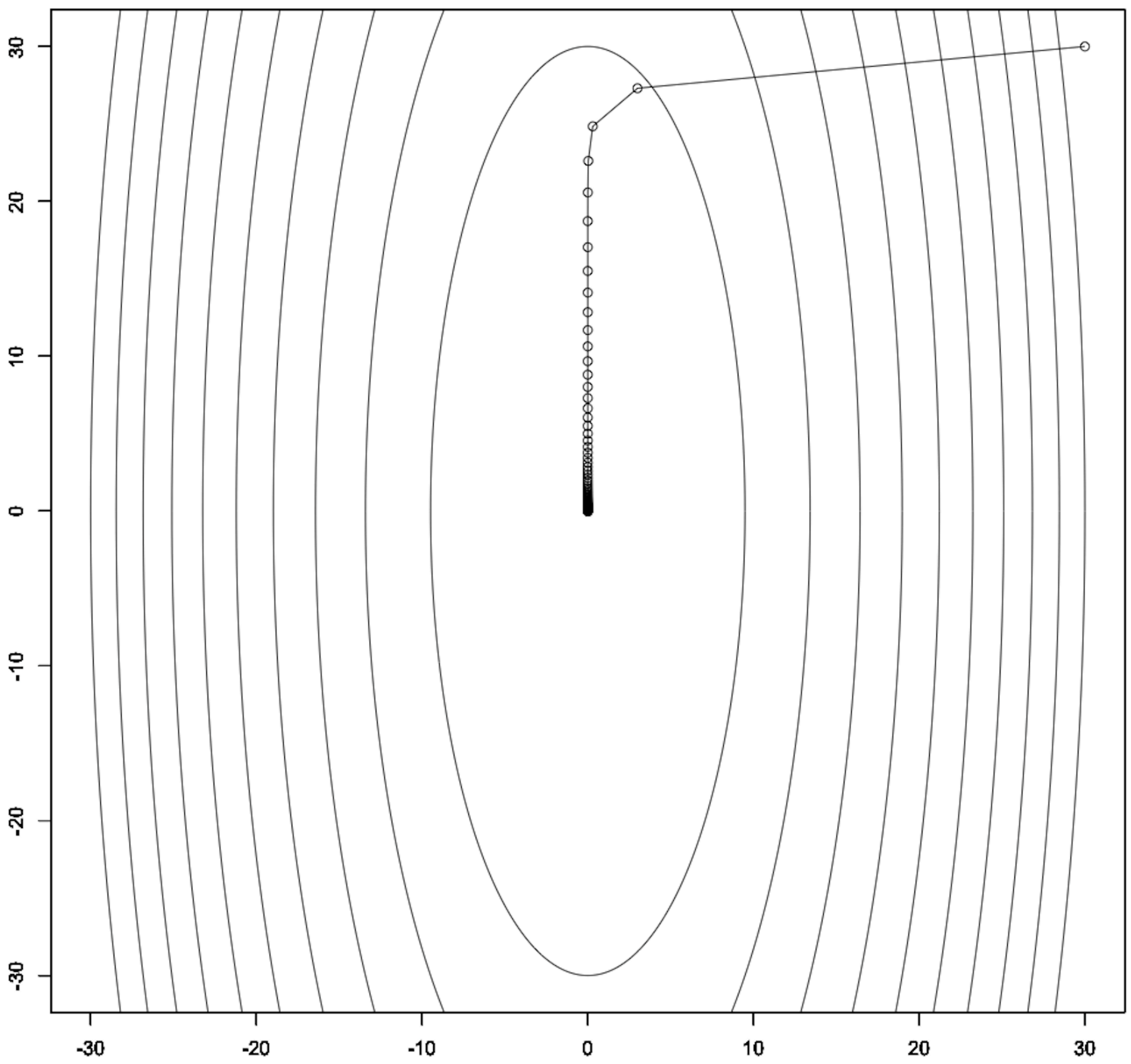
set up function and x_0

```
x0 = c(30,30)
f = function(x){
  return(sum((c(10,1)*x)^2)/2)
}
df = function(x){
  return(c(10,1)*2*x)
}

```

Fixed: $\alpha^{(t)}$ constant

```
step = 9e-2
epsilon = 1e-4
ans = c()
x = x0
while(1){
  ans = rbind(ans,x)
  if(sum(df(x)^2)<epsilon^2)
    break
  x = x-step*df(x)
}
for(r in (1:10)*900){
  theta = (0:360)*pi/180
  x = cos(theta)*sqrt(r)/sqrt(10)
  y = sin(theta)*sqrt(r)
  plot(x,y,xlim=c(-30,30),ylim=c(-30,30),type='l')
  par(new=TRUE)
}
plot(ans[,1],ans[,2],type='o',xlim=c(-30,30),ylim=c(-30,30))
```



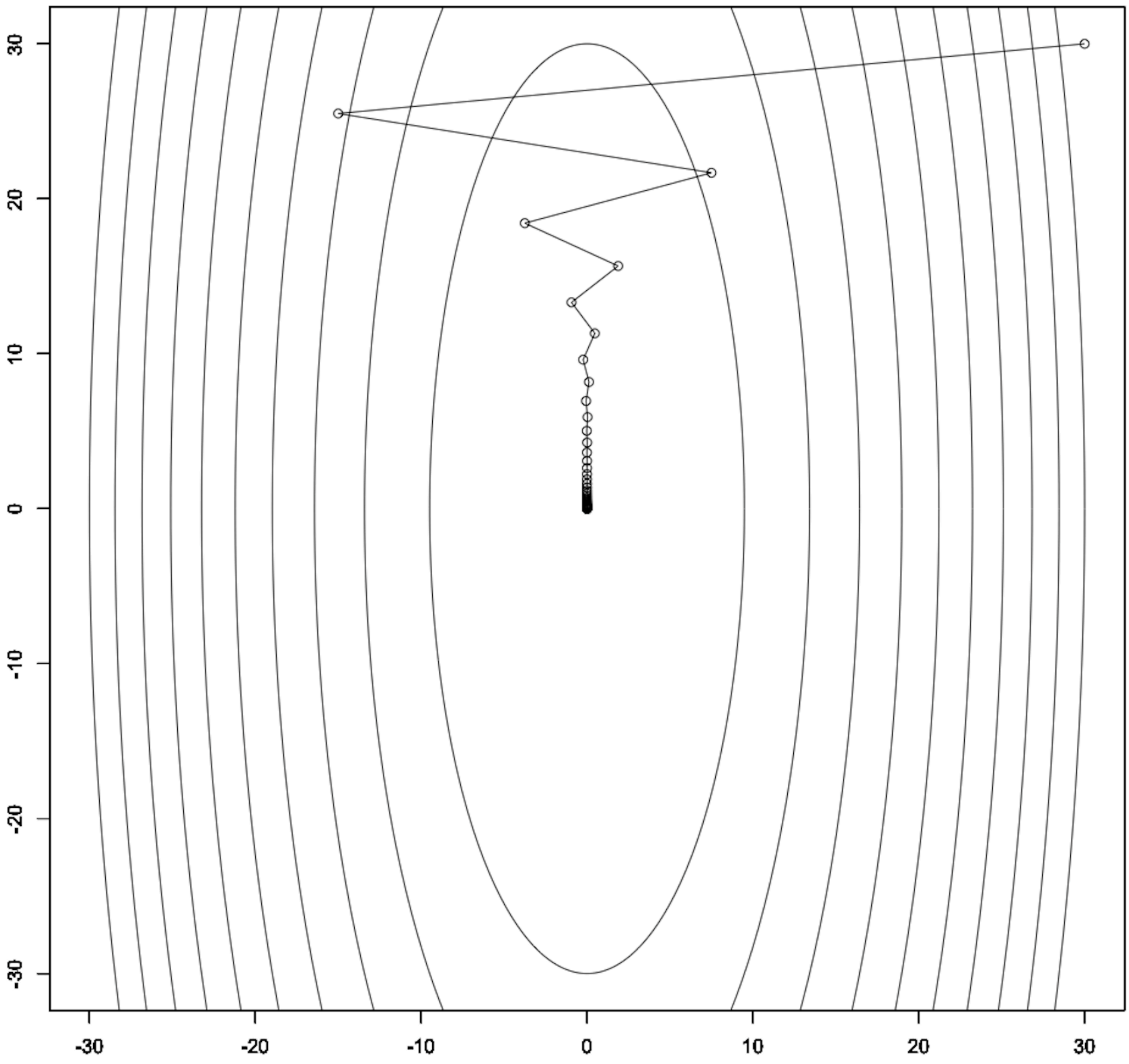
Algorithm stops iteration after 135 times.

Backtracking line search

```
step = 1.5e-1
epsilon = 1e-4
beta = 0.9
t=1/2
x = x0
ans = c()
while(1){
  ans = rbind(ans,x)
  if(sum(df(x)^2)<epsilon^2)
    break
  if(f(x-step*df(x))>f(x)-step*t*sum(df(x)^2))
    step = step*beta
  x = x-step*df(x)
}
for(r in (1:10)*900){
```

```
  theta = (0:360)*pi/180
  x = cos(theta)*sqrt(r)/sqrt(10)
  y = sin(theta)*sqrt(r)
  plot(x,y,xlim=c(-30,30),ylim=c(-30,30),type='l')
  par(new=TRUE)
}
plot(ans[,1],ans[,2],type='o',xlim=c(-30,30),ylim=c(-30,30))

```



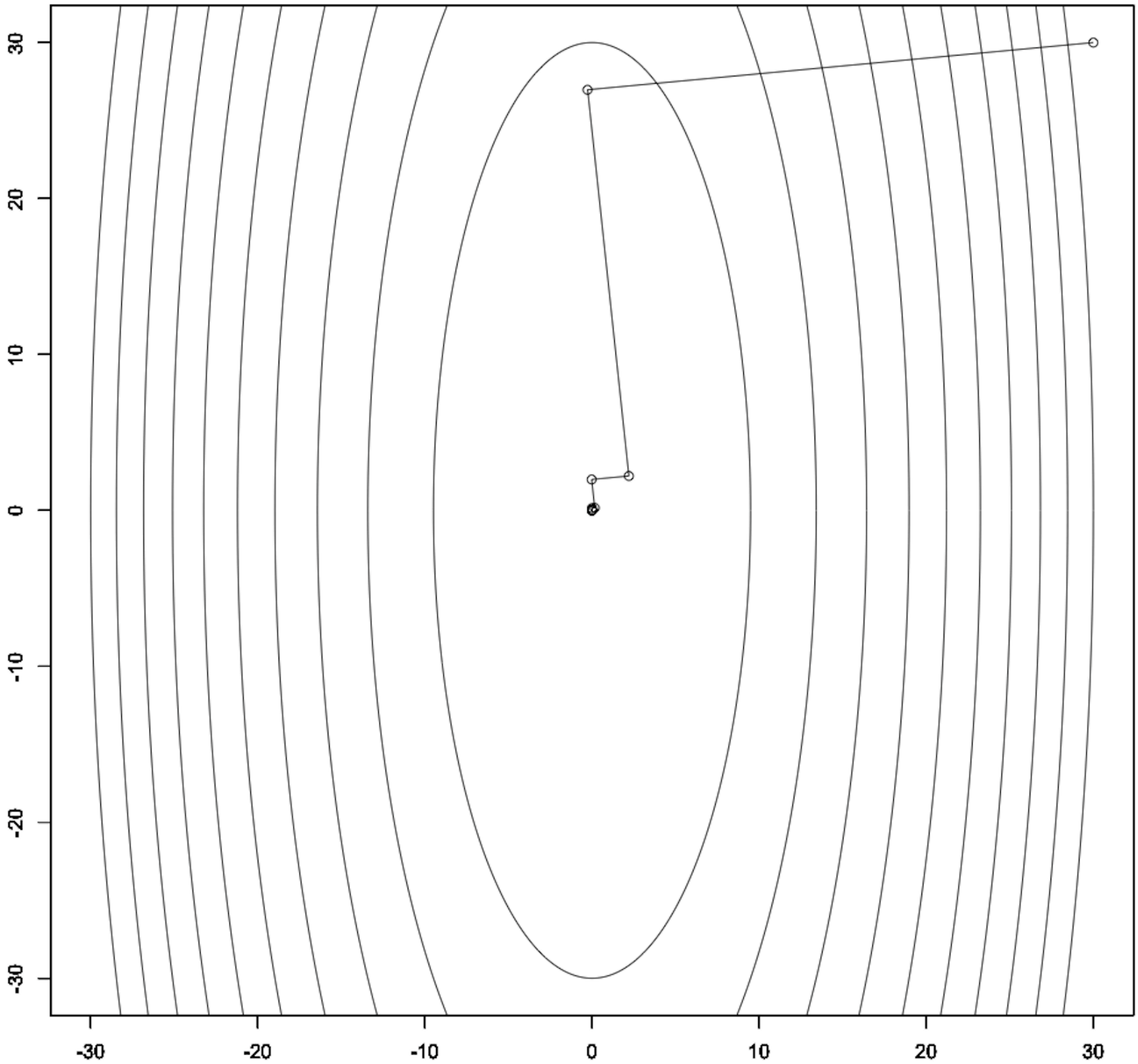
Algorithm stops iteration after 79 times.

Exact line search: minimize

```
epsilon = 1e-4
A = matrix(c(10,0,0,1),2,2)
x = x0
ans = c()
while(1){
```

```
  ans = rbind(ans,x)
  if(sum(df(x)^2)<epsilon^2)
    break
  step = (t(x)%*%A%*%df(x))/(t(df(x)%*%A%*%df(x)))
  step = as.numeric(step)
  x = x-step*df(x)
}
for(r in (1:10)*900){
  theta = (0:360)*pi/180
  x = cos(theta)*sqrt(r)/sqrt(10)
  y = sin(theta)*sqrt(r)
  plot(x,y,xlim=c(-30,30),ylim=c(-30,30),type='l')
  par(new=TRUE)
}
plot(ans[,1],ans[,2],type='o',xlim=c(-30,30),ylim=c(-30,30))

```



Algorithm stops iteration after 12 times.

Prob2. Recall the least-squares error function for linear regression:

$$\operatorname{Error}(\boldsymbol{\beta})=\frac{1}{2}\left(y-\boldsymbol{x}^{\prime} \boldsymbol{\beta}\right)^2$$

This objective function encodes a belief that bigger errors are much worse than smaller errors. One problem with using a squared error function is that outliers can have a big impact on the result. An alternative that is more robust to outliers is the absolute error (or L_1 error):

$$\text{Error }(\boldsymbol{\beta})=\left|y-\boldsymbol{x}^{\prime} \boldsymbol{\beta}\right|=\left|y-\left(\beta_0+\beta_1 x_1+\cdots+\beta_{p-1} x_{p-1}\right)\right|$$

Your goal now is to develop a gradient-descent learning rule $\left(x^{(t+1)}=x^{(t)}-\alpha^{(t)} f^{\prime}\left(x^{(t)}\right)\right.$) for this specific objective function.

Consider the following one-dimensional smooth function:

$$l_\delta(x)=\begin{cases}\frac{1}{2 \delta} x^2, & |x|<\delta \\ |x|-\frac{\delta}{2}, & \text{otherwise.}\end{cases}$$

When $\delta \rightarrow 0$, the smooth function $l_\delta(x)$ and the absolute value function $|x|$ will get closer.

We have

$$\left(\nabla L_\delta(x)\right)_i=\begin{cases}\operatorname{sign}\left(x_i\right), & \left|x_i\right|>\delta \\ \frac{x_i}{\delta}, & \left|x_i\right| \leqslant \delta\end{cases}$$

Thus

$$\frac{d \text{ Error }(\boldsymbol{\beta})}{d \beta_i}=\operatorname{sgn}\left(y-x^{\prime} \beta\right) x_i$$

Which could be substituted into the algorithm($\beta^{(t+1)}=\beta^{(t)}-\alpha^{(t)} \nabla f\left(\beta^{(t)}\right)$).