Poisson regression, splines, and gradient methods

Prof. Sharpnack

Ebola dataset

```
Indicator, Country, Date, value
"Cumulative number of confirmed probable and suspected Ebola cases",Guinea,2015-03-10,3285.0
"Cumulative number of confirmed Ebola cases",Guinea,2015-03-10,2871.0
"Cumulative number of probable Ebola cases",Guinea,2015-03-10,392.0
"Cumulative number of suspected Ebola cases",Guinea,2015-03-10,22.0
"Cumulative number of confirmed probable and suspected Ebola deaths",Guinea,2015-03-10,2170.0
"Cumulative number of confirmed Ebola deaths",Guinea,2015-03-10,1778.0
"Cumulative number of probable Ebola deaths",Guinea,2015-03-10,392.0
"Cumulative number of confirmed probable and suspected Ebola cases",Liberia,2015-03-10,9343.0
"Cumulative number of confirmed Ebola cases",Liberia,2015-03-10,3150.0
```

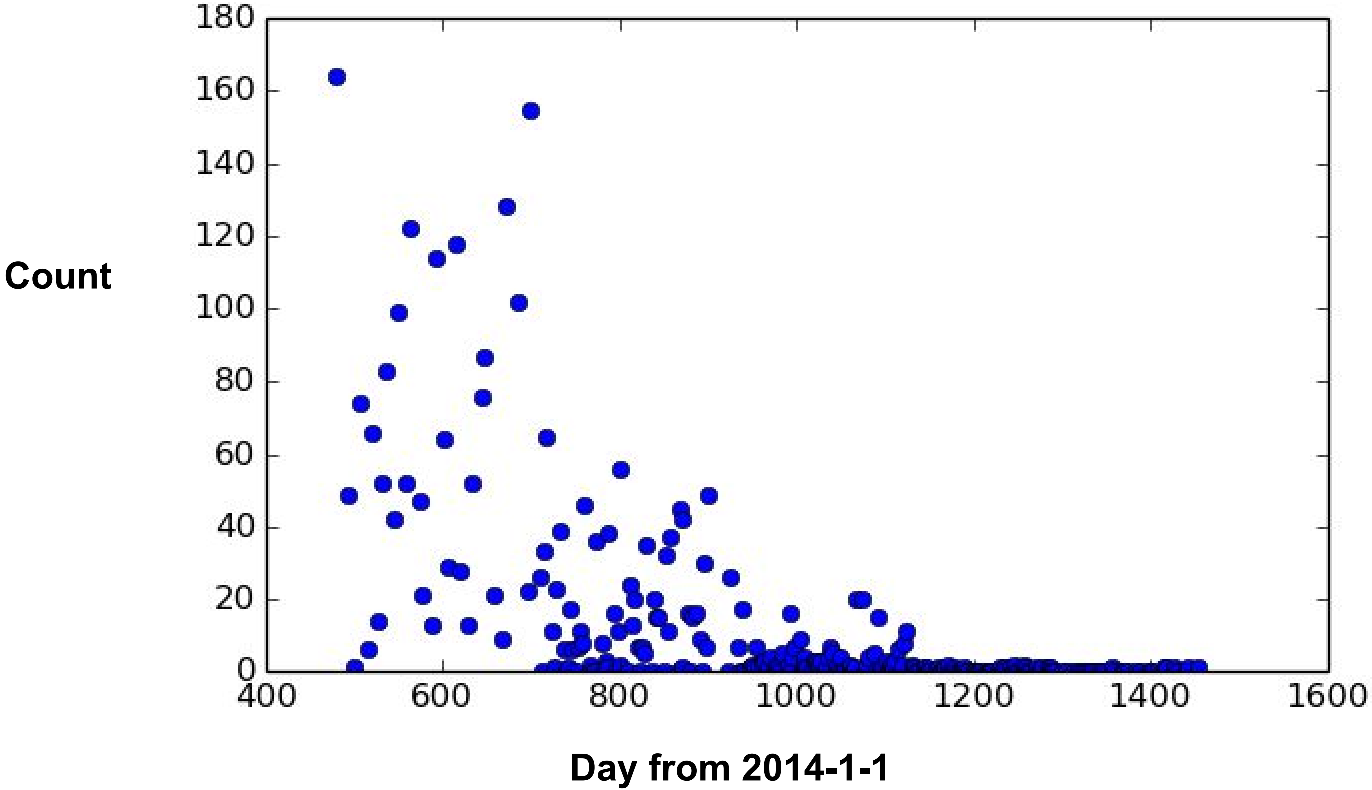
Preprocessing

```
>python
import numpy as np
import datetime as dt

def dateconv(d):
    return dt.datetime.strptime(d,'%Y-%m-%d') - dt.datetime(2014,1,1)

D = np.loadtxt("ebola_data_db_format.csv",dtype=np.str_,delimiter=",")
countries = [s.strip('"') for s in set(D[:,1])]

G = (D[:,1] == 'Guinea')*(D[:,0] == '"Cumulative number of confirmed probable and
suspected Ebola cases"')
```

Preprocessing

```
>python
Y = np.array(D[G,3],dtype='float')
Y = np.array(Y,dtype='int')
X = np.array([dateconv(d).total_seconds()/(12*3600) for d in D[G,2]],dtype='float')
X = np.sort(X)
E = X[1:] - X[:-1]
X = X[1:]
Y = np.sort(Y)
Y = Y[1:] - Y[:-1]
X.shape = (X.shape[0],1)
Y.shape = X.shape
E.shape = X.shape

M = np.hstack((X,E,Y))
np.savetxt("ebola.csv",M)
```

Count data


Count
0
20
40
60
80
100
120
140
160
180
400
600
800
1000
1200
1400
1600
Day from 2014-1-1

Poisson likelihood

$$L(y) = \frac{e^{-\lambda}\lambda^{y}}{y!}$$

$$y_i \sim Pois(\lambda_i)$$
$$\log(\lambda_i / e_i) = z_i^\top \beta$$

exposure

linear in covariates

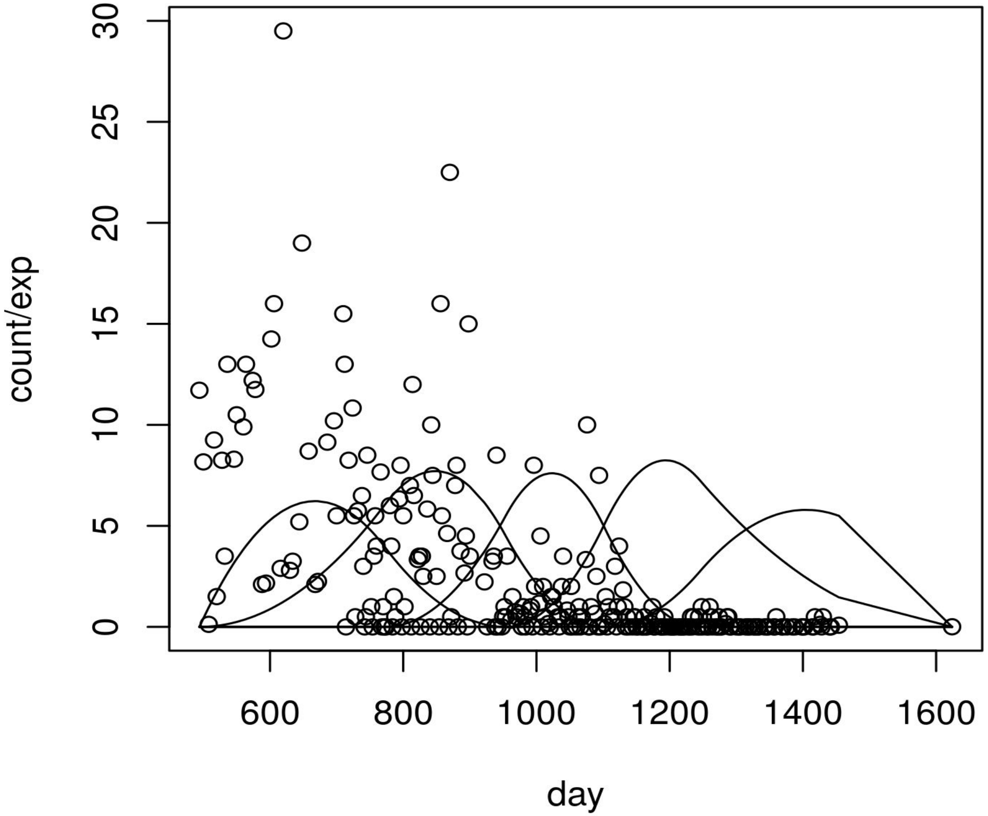
Poisson likelihood

$$-\log\left(\prod_i L(\lambda_i; y_i)\right) = \sum_i -\log L(\lambda_i; y_i)$$

The negative log likelihood is our loss function

$$\propto \sum_i \lambda_i - y_i \log \lambda_i \propto \sum_i e_i e^{z_i^\top \beta} - y_i z_i^\top \beta$$

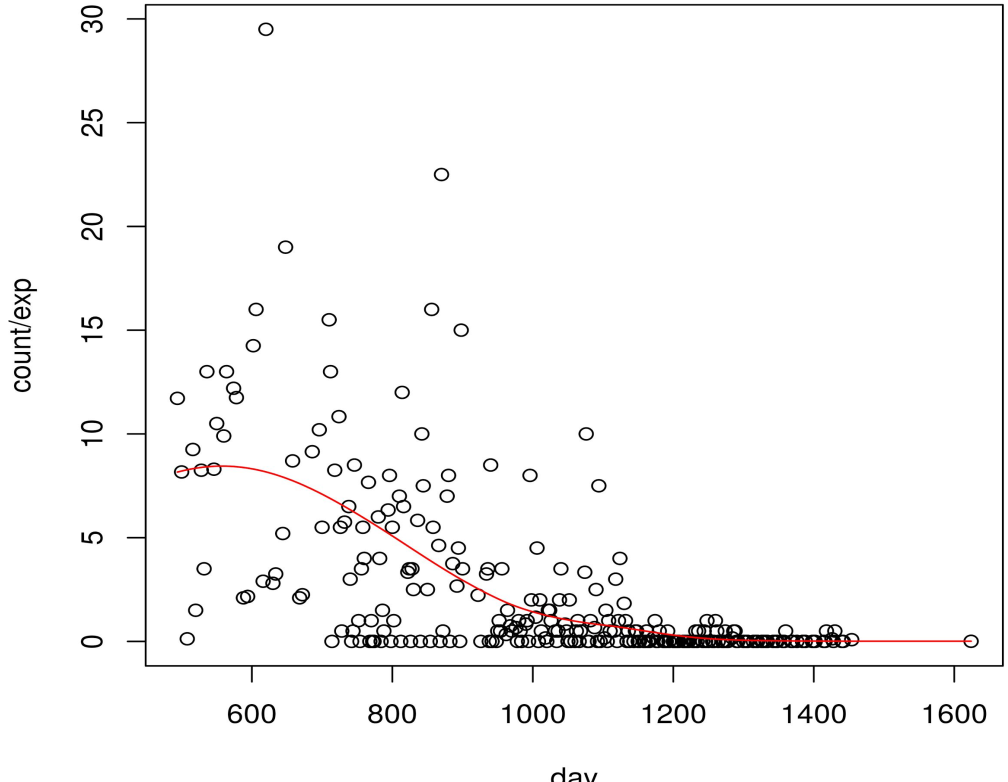
Constructing covariates from B-splines



Poisson regression with B-splines

```
>R

D = read.csv('ebola.csv',header=F,sep=" ")
n = dim(D)[1]
X = D[,1]
Y = D[,3]
E = D[,2]
Bq = bs(X,df=6,degree=2)

pois = glm(Y ~ offset(log(E)) + Bq, family=poisson(link=log))
Yhat = predict(pois,type="response")
```

Poisson regression with B-splines



Poisson likelihood

$$\propto \sum_i \lambda_i - y_i \log \lambda_i \propto \sum_i e_i e^{z_i^\top \beta} - y_i z_i^\top \beta$$

Gradient of - log likelihood / n

$$g_i(\beta) = \left(e_i e^{z_i^\top \beta} - y_i\right) z_i \qquad g(\beta) = \frac{1}{n} \sum_i g_i(\beta)$$

Poisson regression with B-splines

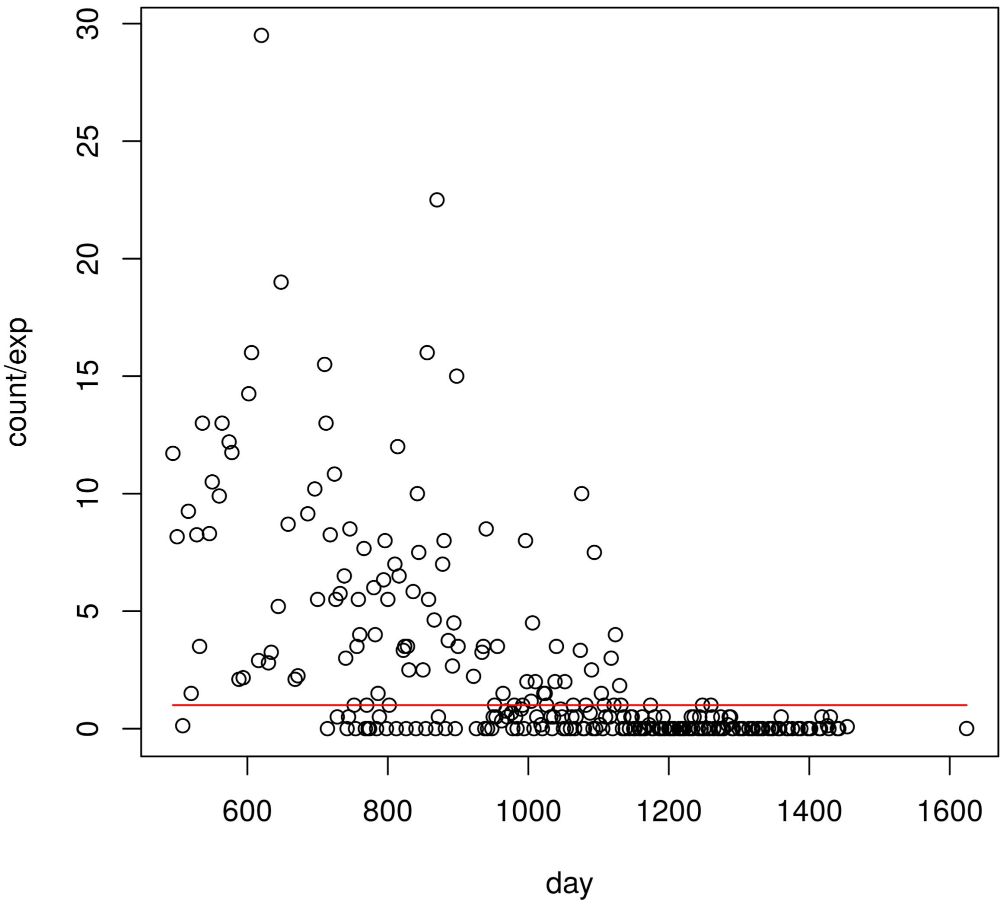
```
>R

pois.grad = function(Y,E,Z,beta){
    weights = (E * exp(Z %*% beta) - Y)
    Gv = weights %*% t(rep(1,p)) * Z
    grad = apply(Gv,2,mean)
    return(grad)
}

t = t+1
m=.1
eta = m/t**0.5
grad = pois.grad(Y,E,Z,beta)
beta = beta - eta * grad
pred = exp(Z%*%beta)
```

Iterative gradient

learning rate schedule:

0.1 / sqrt(t)

t=1

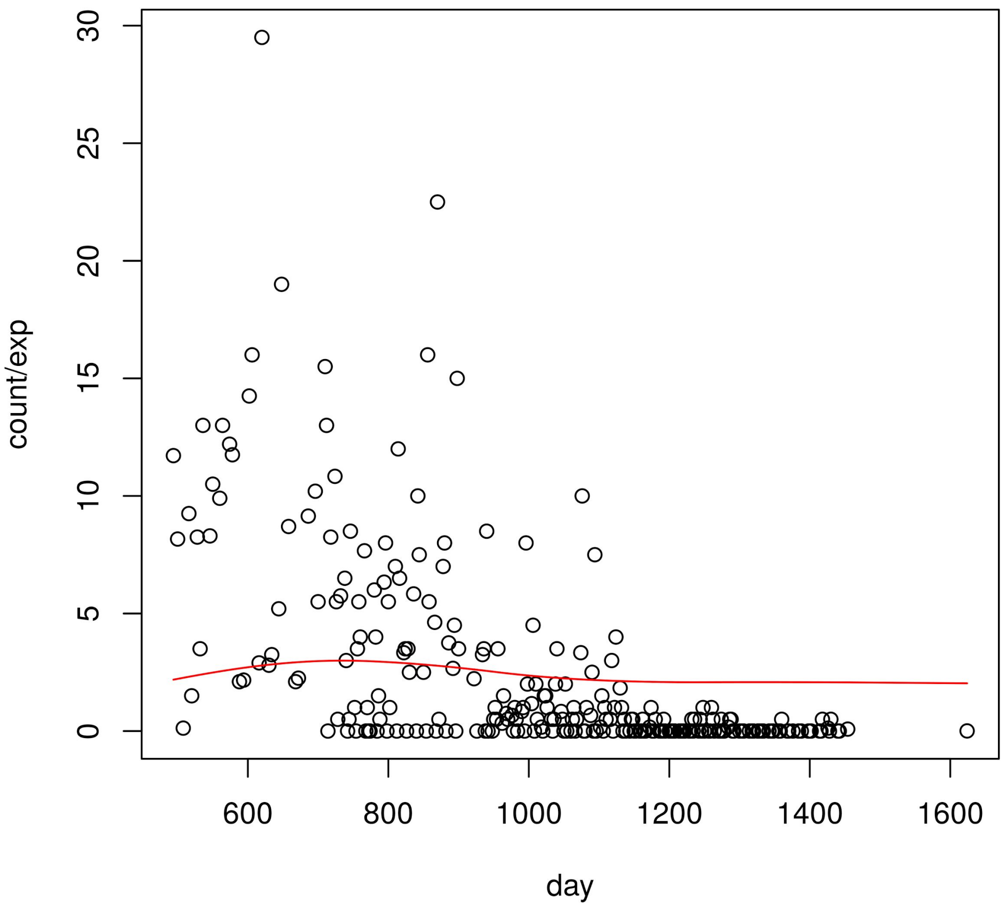


Iterative gradient

learning rate schedule:

0.1 / sqrt(t)

t=2

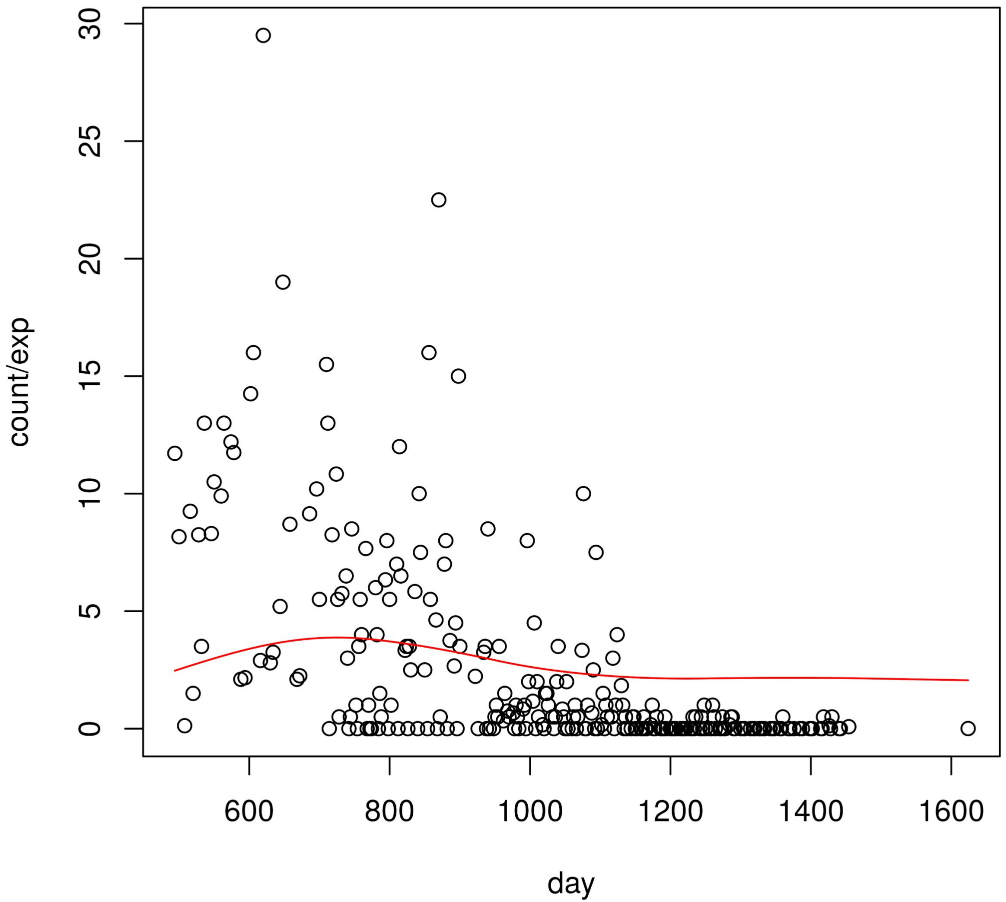


Iterative gradient

learning rate schedule:

0.1 / sqrt(t)

t=3

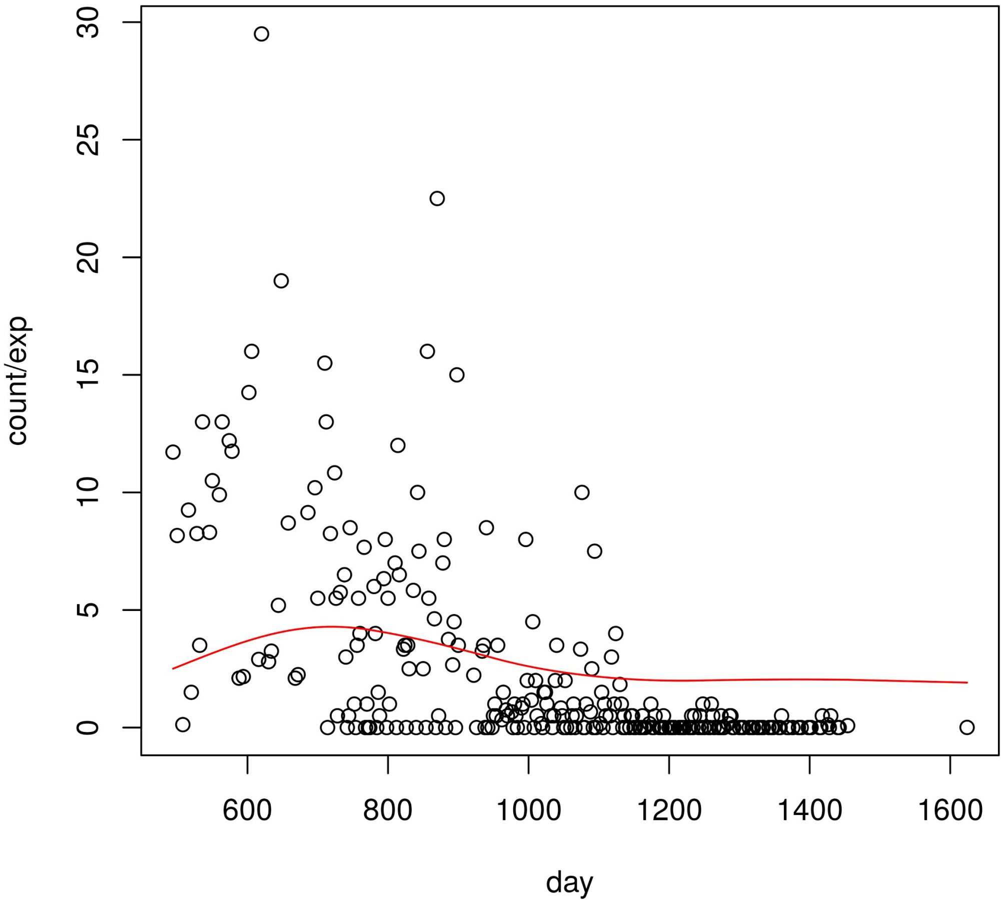


Iterative gradient

learning rate schedule:

0.1 / sqrt(t)

t=4

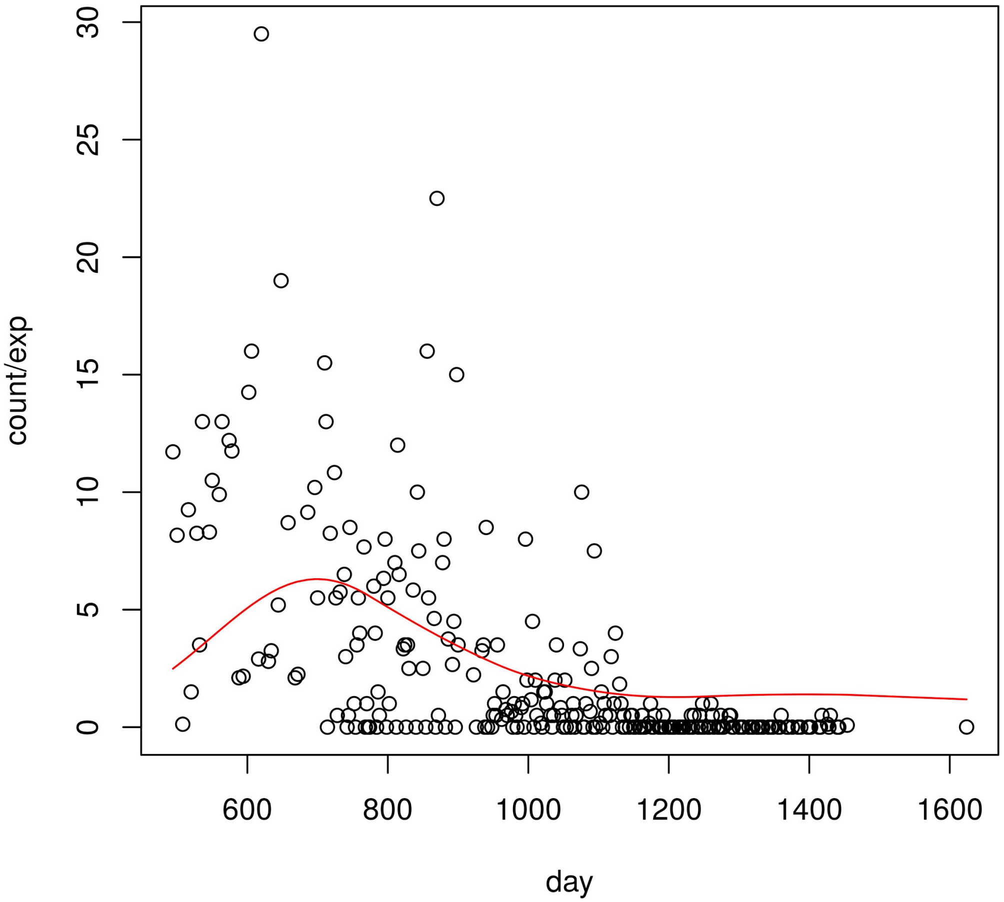


Iterative gradient

learning rate schedule:

0.1 / sqrt(t)

t=15

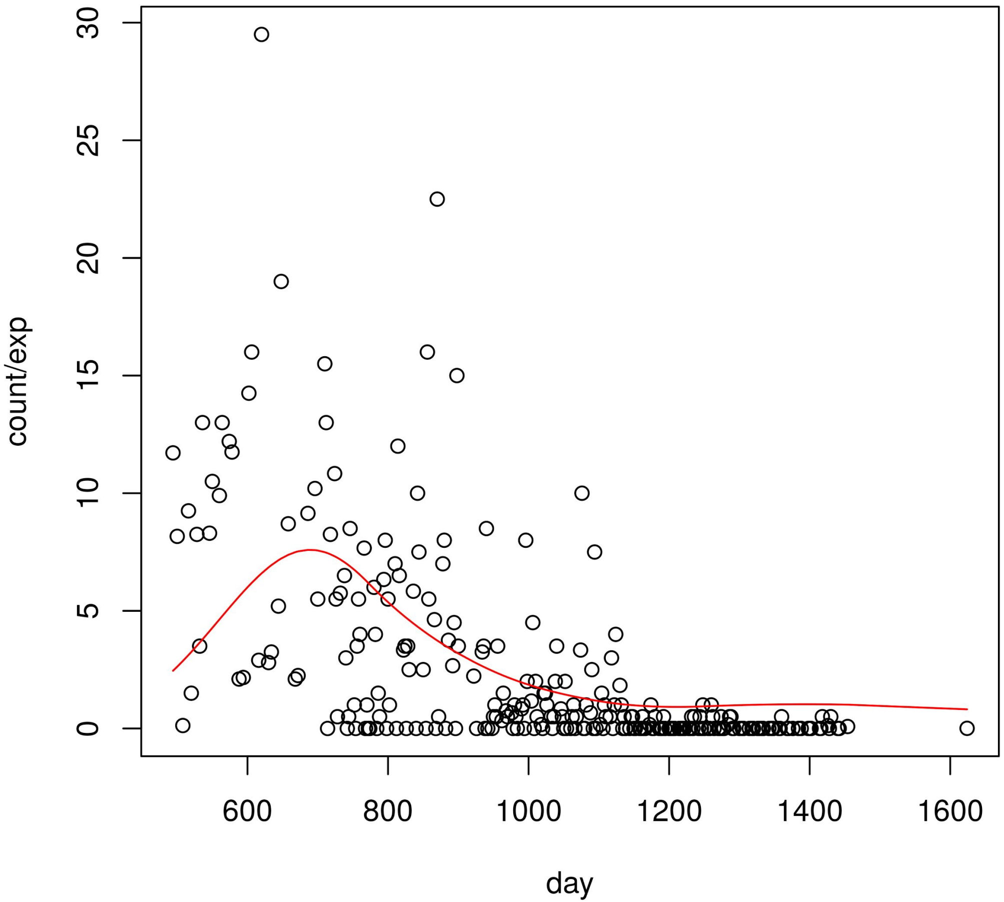


Iterative gradient

learning rate schedule:

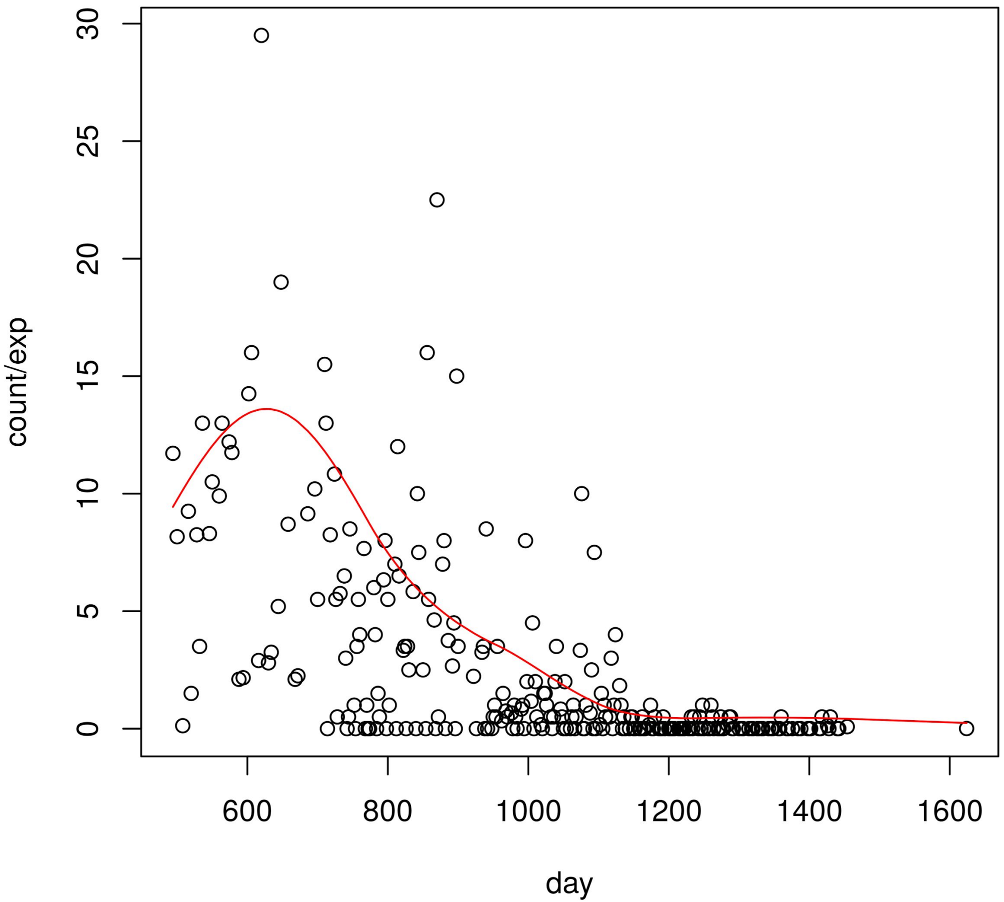
0.1 / sqrt(t)

t=40



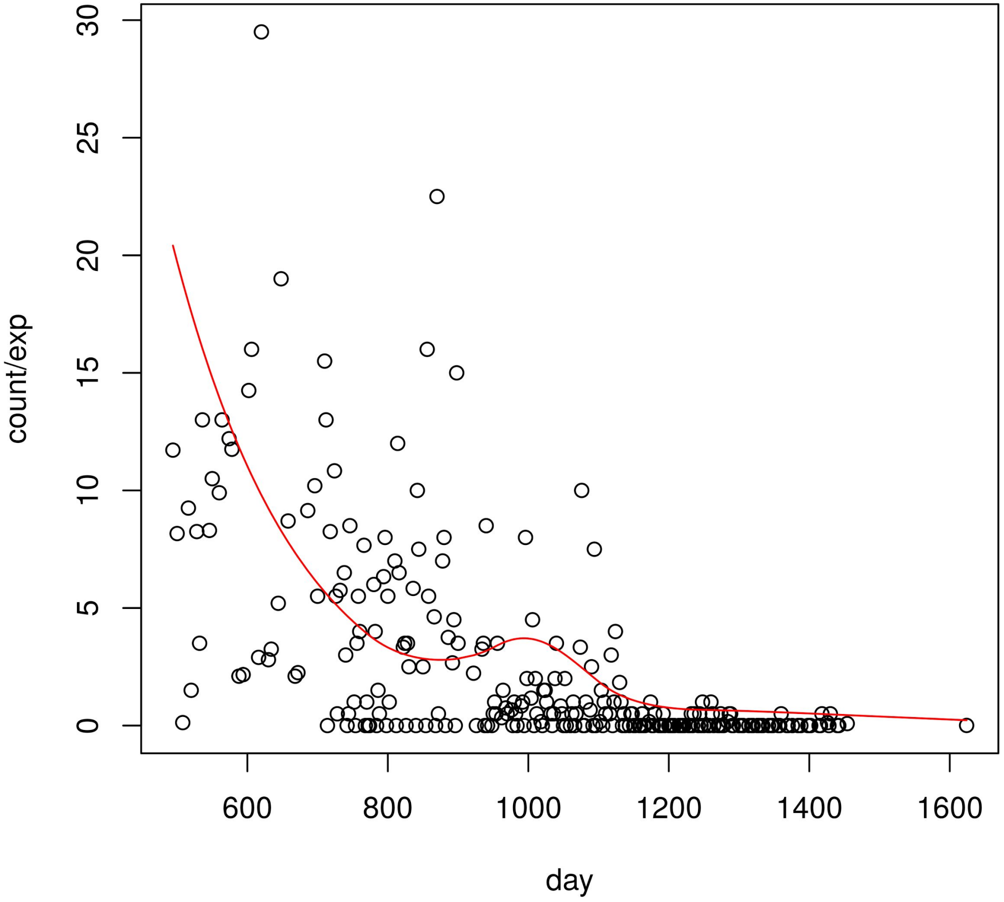
Iterative gradient

Restart



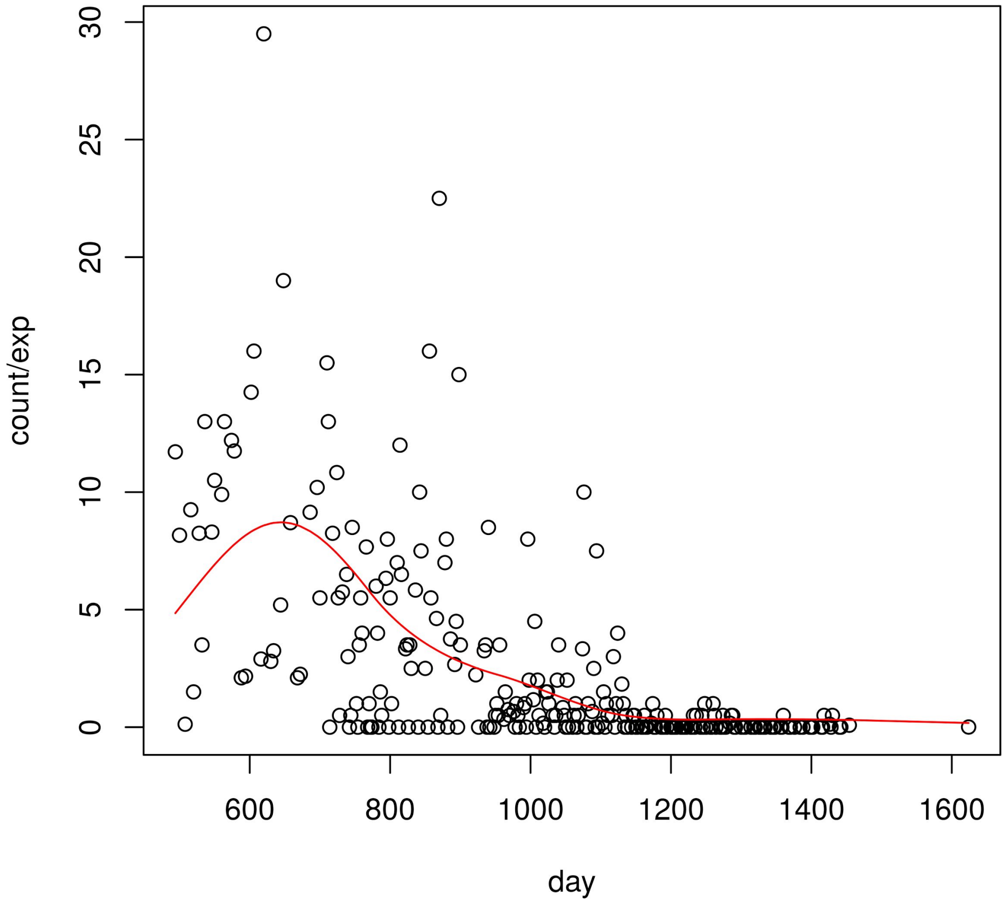
Iterative gradient

Restart



Iterative gradient

Constant learning rate



Iterative gradient

The learning rate plays a huge role in these methods: one reason to use Newton-Raphson

$$g_i(\beta) = \left(e_i e^{z_i^\top \beta} - y_i\right) z_i \qquad g(\beta) = \frac{1}{n} \sum_i g_i(\beta)$$

Stochastic gradient

```
>R
grad = pois.grad(Y,E,Z,beta)
k=40
S = sample(n,k)
gradS = pois.grad(Y[S],E[S],Z[S,],beta)

print(gradS)
-3.6000000 -3.0767723 -1.0682690 -0.6364375  0.3710768  0.6313820  4.3901139

print(grad)
-7.8217054 -3.6451910 -2.6974959 -0.3000890  0.5432673  0.4298444  0.7370615
```

Stochastic gradient

Constant learning rate