Fitting Models to Data

Statistics 427: R Programming

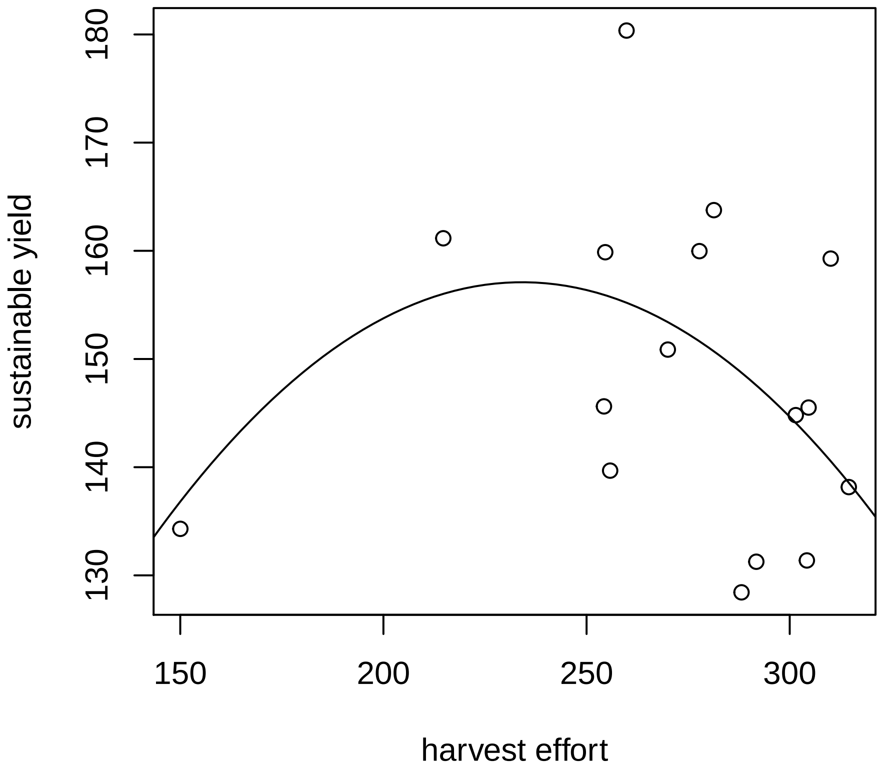
Module 17

2020

Bubba Shrimp (not Dr. Shemp)

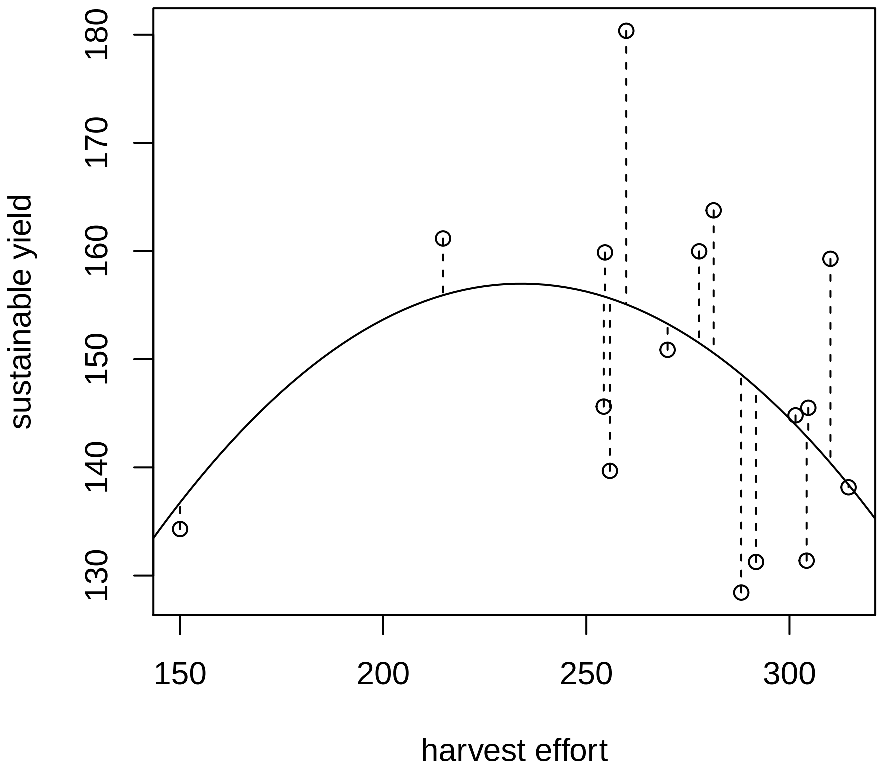
Quadratic model fit with graphs

```
gump=read.csv('https://webpages.uidaho.edu/~renaes/Data/shrimp%20yield.csv',header=T)
with(gump,plot(effort,yield,xlab='harvest effort',ylab='sustainable yield'))

xlo=100             # Low value of effort for calculating quadratic
xhi=350             # High value of effort
x=xlo+(0:100)*(xhi-xlo)/100  # Range of effort values
a=0.002866
b=1.342
y=-a*x^2+b*x   # yield calculated for range of effort values
with(gump,plot(effort,yield,xlab='harvest effort',ylab='sustainable yield'))
points(x,y,type='l')  #  Add the quadratic model to the plot
```



More Bubba: least squares fitted model

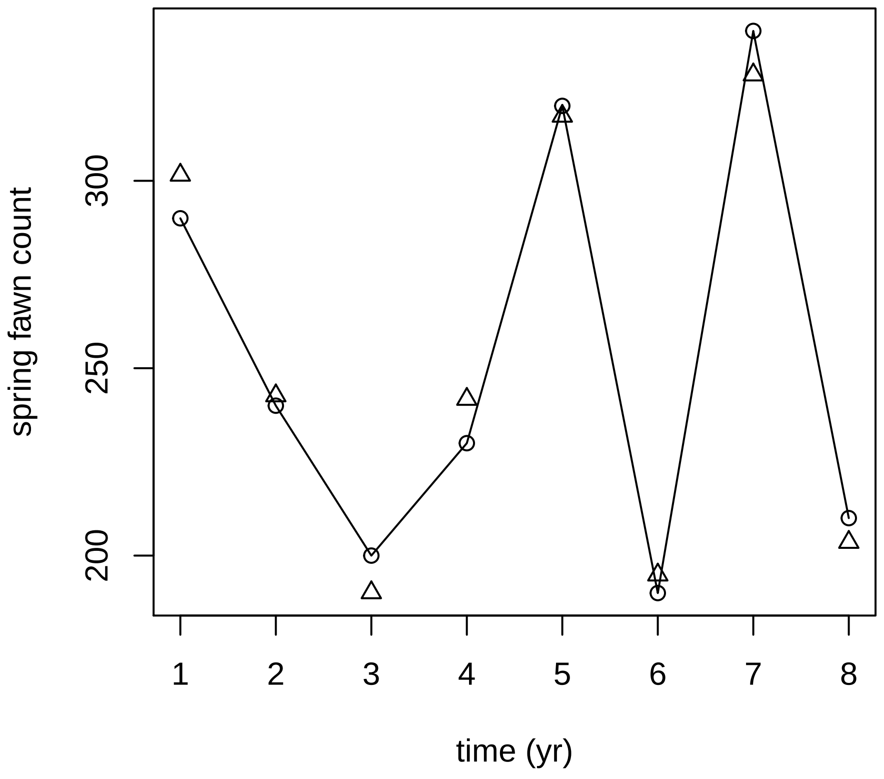
```
# Draw a scatterplot of sustainable yield vs.
# harvesting effort for shrimp data from the Gulf of Mexico,
# with parabolic yield-effort curve fitted and superimposed.
#=============================================================
# Fit parabolic yield-effort curve with least squares.
#-------------------------------------------------------------
attach(gump)
X=cbind(effort,effort^2)  # 1st column of X is effort, 2nd column is effort squared
b=solve(t(X)%*%X,t(X)%*%yield)  # Least squares solution
h=b[1,]           # Coefficient of effort
g=b[2,]           # Coefficient of effort squared
# Overlay fitted quadratic model on scatterplot
elo=100             # Low value of effort for calculating quadratic
ehi=350             # High value of effort
eff=elo+(0:100)*(ehi-elo)/100  # Range of effort values
sy=h*eff+g*eff^2    # Sustainable yield calculated for range of
                    #  effort values
plot(effort,yield,xlab='harvest effort',ylab='sustainable yield')
points(eff,sy,type='l')  #  Add the quadratic model to the plot
x0=effort
y0=yield
x1=effort
y1=h*effort+g*effort^2
segments(x0,y0,x1,y1,lty=2)
```



```
detach(gump)
```

Pronghorn prediction model

```
# Calculate least squares estimates for pronghorn data.
# Variable for prediction is y, spring fawn count.
# Predictor variables are size of adult pronghorn population (u),
# annual inches precipitation (v), winter severity index (w;  scale
# of 1:mild-5:severe. Time plot of the data along with overlayed
# predictions.
#==================================================================
y=c(290,240,200,230,320,190,340,210)
u=c(920,870,720,850,960,680,970,790)
v=c(13.2,11.5,10.8,12.3,12.6,10.6,14.1,11.2)
w=c(2,3,4,2,3,5,1,3)
bambi=data.frame(cbind(y,u,v,w))
attach(bambi)
# Calculate least squares intercept and slope
n=length(y)                # Number of observations is n
X=matrix(1,n,4)            # Form the X matrix: col 1 has 1's,
X[,2]=u                    #  cols 2-4 have predictor variables.
X[,3]=v
X[,4]=w
b=solve(t(X)%*%X,t(X)%*%y)   # LS est b, t() is transpose function
# OR you can use b=solve((t(X)%*%X)%*%t(X)%*%y)
# Draw a time plot of data, with overlayed least squares predictions
plot((1:8),y,type='o',xlab='time (yr)',ylab='spring fawn count')
ypred=X%*%b  # Calculate predicted y values at values of predictors.
points((1:8),ypred,pch=2)  # Plot predicted values w/ points
```

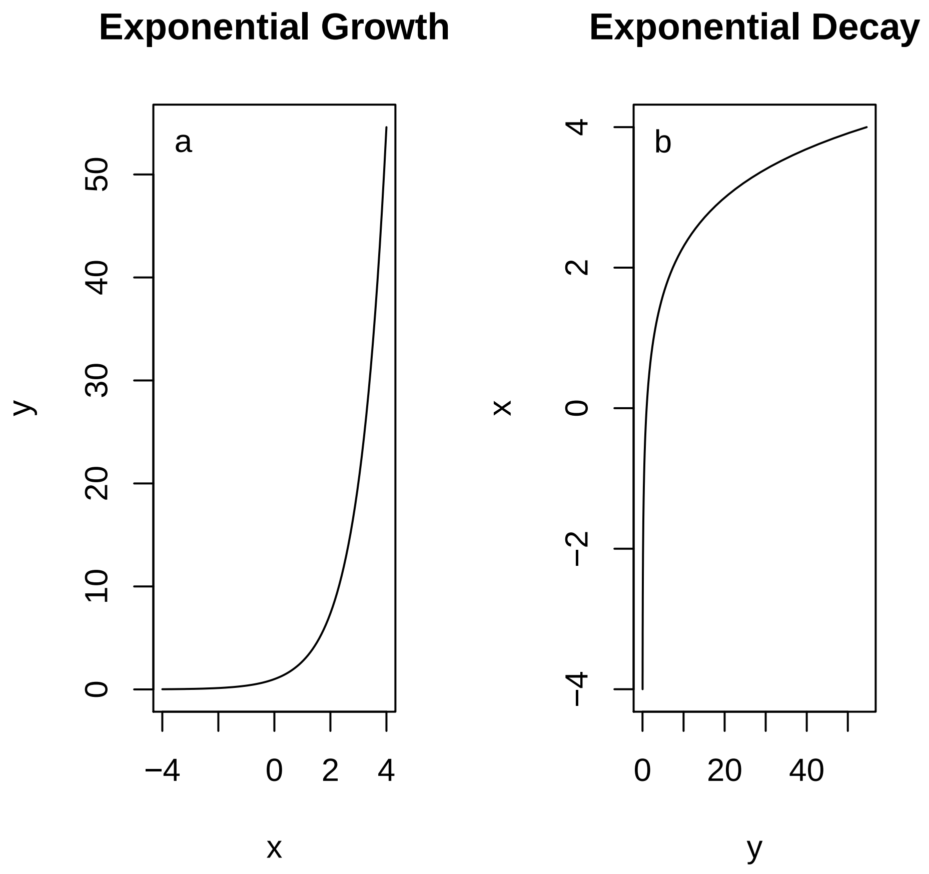


```
# print
cat('least squares intercept and coefficients: ',b)
```

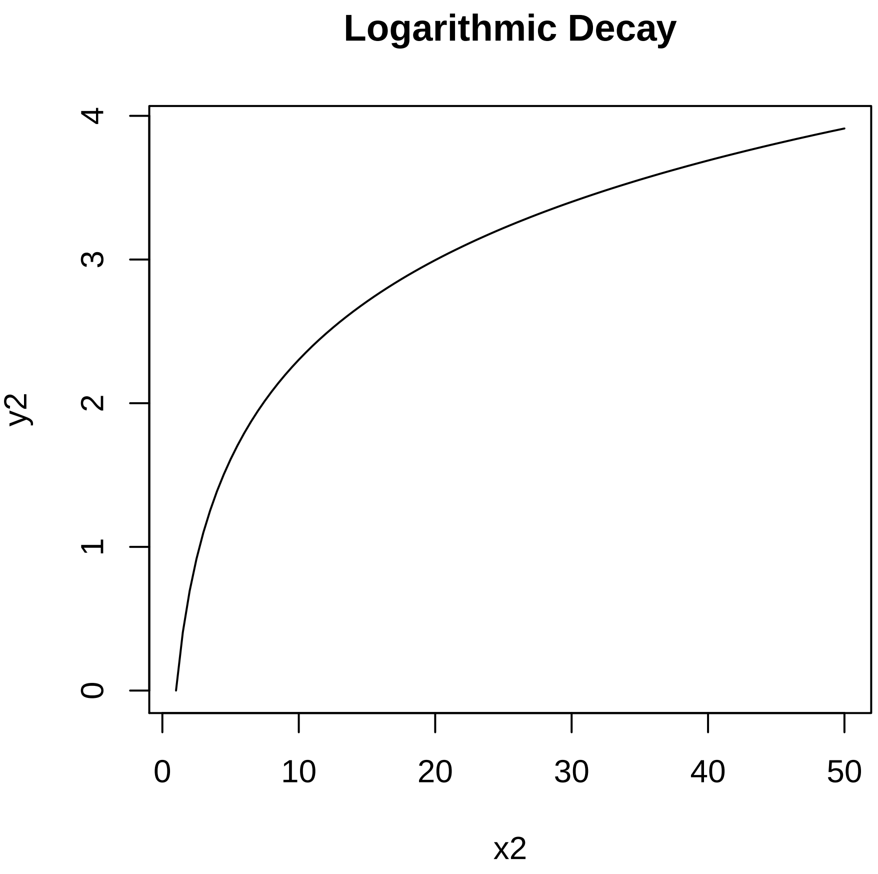
```
least squares intercept and coefficients:  -592.2012 0.3382175 40.15039 26.29461
```

Exponential growth model

```r
x=4*(-50:50)/50
y=exp(x)
par(mfrow=c(1,2))
plot(x,y,type='l',main='Exponential Growth')
text(-3.25,53,'a')
plot(y,x,type='l',main='Exponential Decay')
text(5,3.8,'b')
```

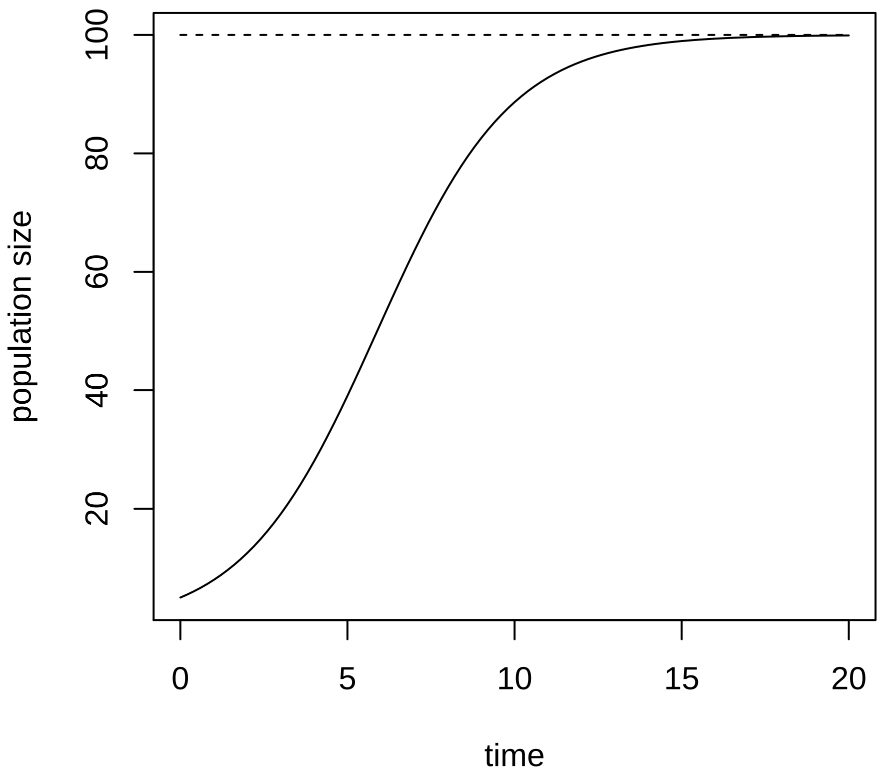


```r
par(mfrow=c(1,1)) # sets graphing window back to default
x2=25*(2:100)/50
y2=log(x2)
plot(x2,y2,type='l',main='Logarithmic Decay')
```



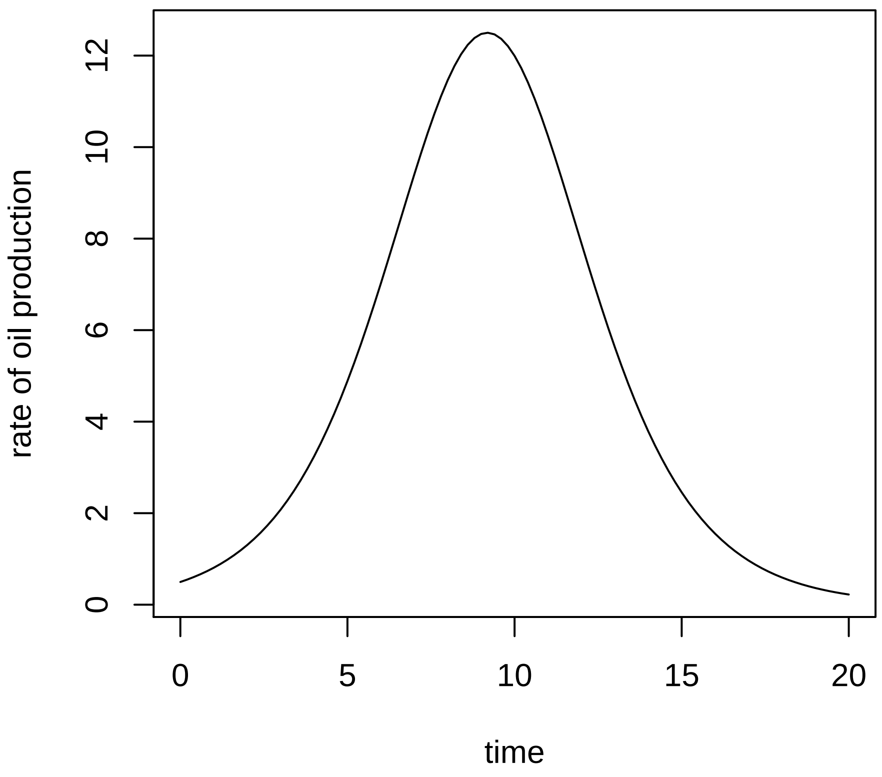
Logistic growth

```
k=100       # Population levels off at k
b=0.5        # Higher value of b will result in quicker approach to k
m=5         # Initial population size m
t=(0:100)*20/100  # Range of values of time t from 0 to 20
n=k/(1+((k-m)/m)*exp(-b*t))  # Logistic population growth function
plot(t,n,type='l',xlab='time',ylab='population size')  # Plot n vs t
k.lvl=numeric(length(t))+k;      # Vector with elements all equal to k
points(t,k.lvl,type='l',lty=2)    # Add dashed line at the level of k
```



Hubbert's model of oil production

```
k=100      # Maximum amount of recoverable resource
b=0.5       # Resource depleted faster if b is larger
m=1        # Initial amount produced is m
t=(0:100)*20/100  # Range of values of time t from 0 to 20
s=0.01      # Small interval of time
change.n=k/(1+((k-m)/m)*exp(-b*(t+s)))-k/(1+((k-m)/m)*exp(-b*t))
           # Amount of oil extracted between time t and time t+s
rate.n=change.n/s  # Rate of oil production between time t and time t+s
plot(t,rate.n,type='l',lty=1,xlab='time',ylab='rate of oil production')
```



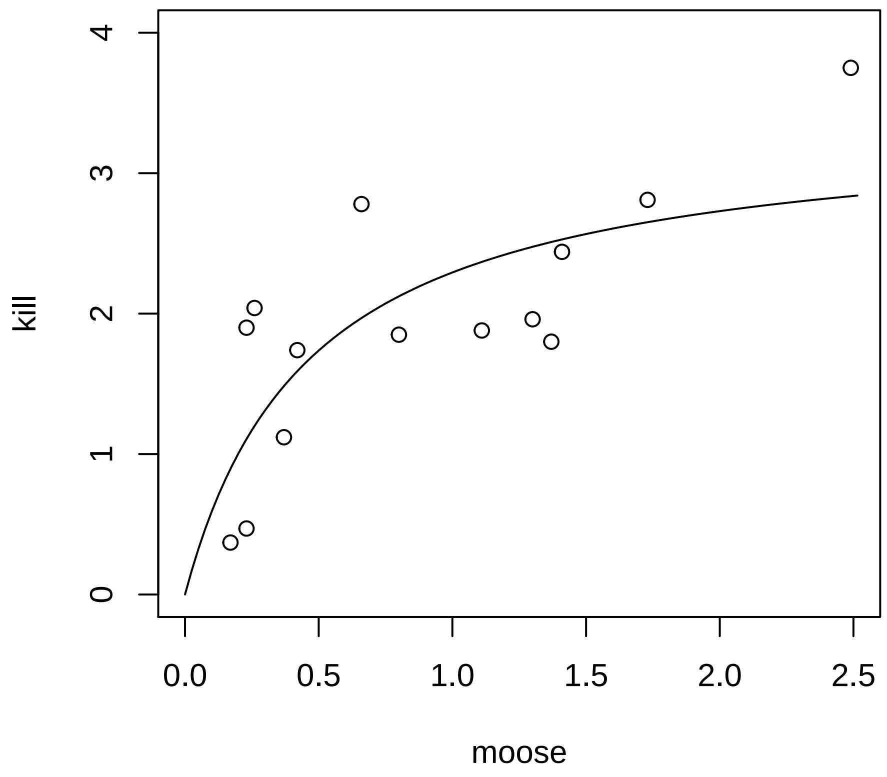
Fitting nonlinear model to wolf

```
# Calculate nonlinear least squares estimates for
# parameters b1 (maximum feeding rate) and b2 (half saturation
# constant) in the rectangular hyperbolic equation for feeding
# rate (Holling type 2 functional response, Monod nutrient uptake
# rate, Michaelis-Menten enzyme equation).  The equation is
# rate = (b1*prey)/(b2 + prey)
#
# Here 0<b1, 0<b2, and "prey" is the density, concentration, or
# abundance of prey, substrate, or nutrient.
#
# Data in example are moose density (# per 1000 km^2) and number
# killed per wolf in 100 d
#==================================================================
# Input the data
moose=c(.17,.23,.23,.26,.37,.42,.66,.80,1.11,1.30,1.37,
1.41,1.73,2.49)
kill=c(.37,.47,1.90,2.04,1.12,1.74,2.78,1.85,1.88,1.96,
1.80,2.44,2.81,3.75)
# Calculate initial values using a linearization transform.  The
#  transform allows initial values to be calculated with a multiple
#  regression without intercept: rate*prey = b1*prey - b2*rate.
yy=kill*moose
xx=cbind(moose,kill)
bb=solve(t(xx)%*%xx,t(xx)%*%t(t(yy)))
b1.0=bb[1]
b2.0=-bb[2]
# Use nls() function to minimize sum of squares with an
```

```r
# iterative numerial method. The object "curve.fit" will be a list
# of results from the calculations
curve.fit=nls(kill~b1*moose/(b2+moose),start=list(b1=b1.0,b2=b2.0))
# Print the results of the calculations. Assign values to b1
# and b2 for plotting the fitted model
summary(curve.fit)
```

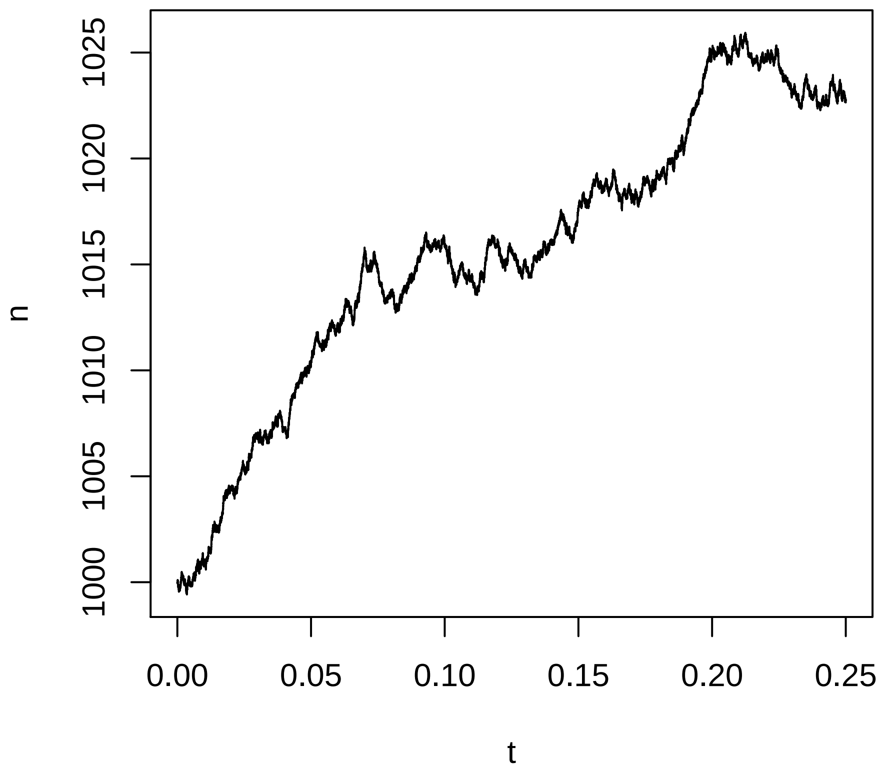
```

Formula: kill ~ b1 * moose/(b2 + moose)

Parameters:
   Estimate Std. Error t value Pr(>|t|)
b1   3.3720     0.6717   5.020 0.000299 ***
b2   0.4705     0.2665   1.766 0.102875
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.6312 on 12 degrees of freedom

Number of iterations to convergence: 7
Achieved convergence tolerance: 3.345e-06
```

```r
b=coef(curve.fit)
b1=b[[1]] # double [[]] eliminates names of vector
b2=b[[2]]
# Calculate fitted rate curve for range of values of prey; store the values in "prey.vals"
# Range is from 0 to slightly beyond max(prey) -> can be changed. Values of fitted rate curve are in "f
prey.vals=(0:100)*1.01*max(moose)/100
fitted.curve=prey.vals*b1/(b2+prey.vals)
# Plot the data in a scatterplot with fitted rate equation
plot(moose,kill,ylim=c(0,4),xlim=c(0,2.5))
points(prey.vals,fitted.curve,type='l')
```



Simulation of investment with randomly fluctuating prices

```
days=60               # Time horizon of simulation (trading days)
r=0.06                 # Annual average return of 6%
dt=1/115200           # 115200 minutes in 240 8-hr trading days/year
sig=.2*r*sqrt(dt)     # Spread constant for noise
mnts=days*8*60        # Number of minutes in 60 trading days
x=numeric(mnts+1)     # Vector to contain the log-prices
t=x                   # Vector to contain the accumulated times
x[1]=log(1000)        # Initial investment amount
t[1]=0                # Initial time

w=rnorm(mnts,0,sig) # Generate vector of normal noises outside loop
                      #  the loop (more efficient)
for (i in 1:mnts) {
  dx=r*dt+w[i]        # Change in log-price during one minute
  x[i+1]=x[i]+dx      # New price after one minute
  t[i+1]=t[i]+dt      # New accumulated time after one minute
}
n=exp(x)              # Change log-prices to prices
plot(t,n,type='l')  # Plot prices vs time
```



n
1000
1005
1010
1015
1020
1025
0.00
0.05
0.10
0.15
0.20
0.25
t