Homework 4 Gaussian Procceses

BenLarson

April 28, 2016

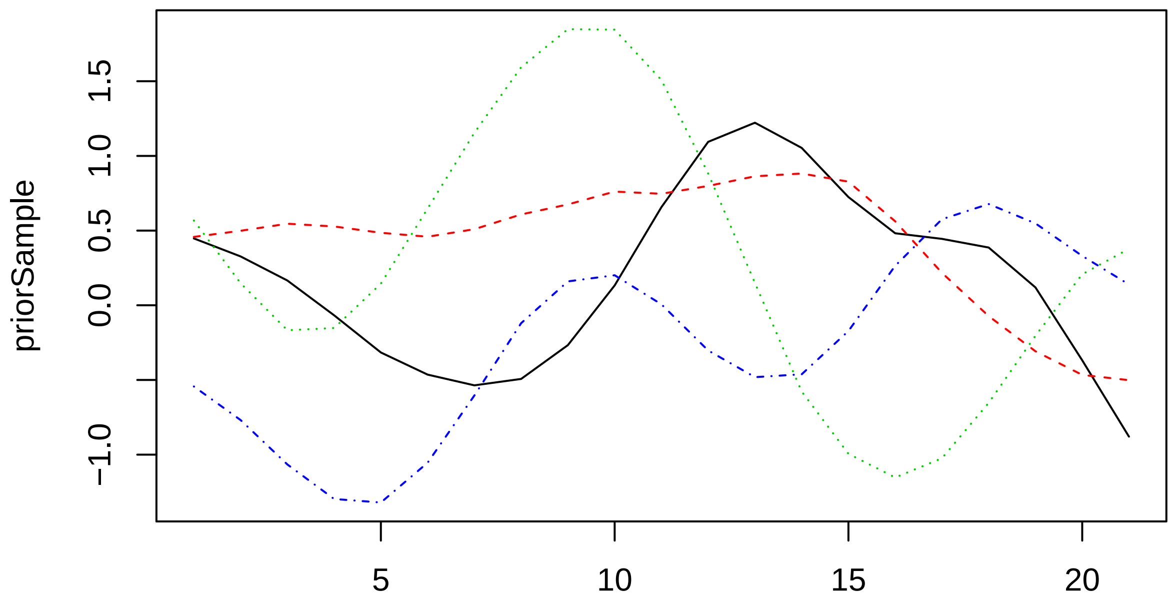
```
require(knitr)
```

```
## Loading required package: knitr
```

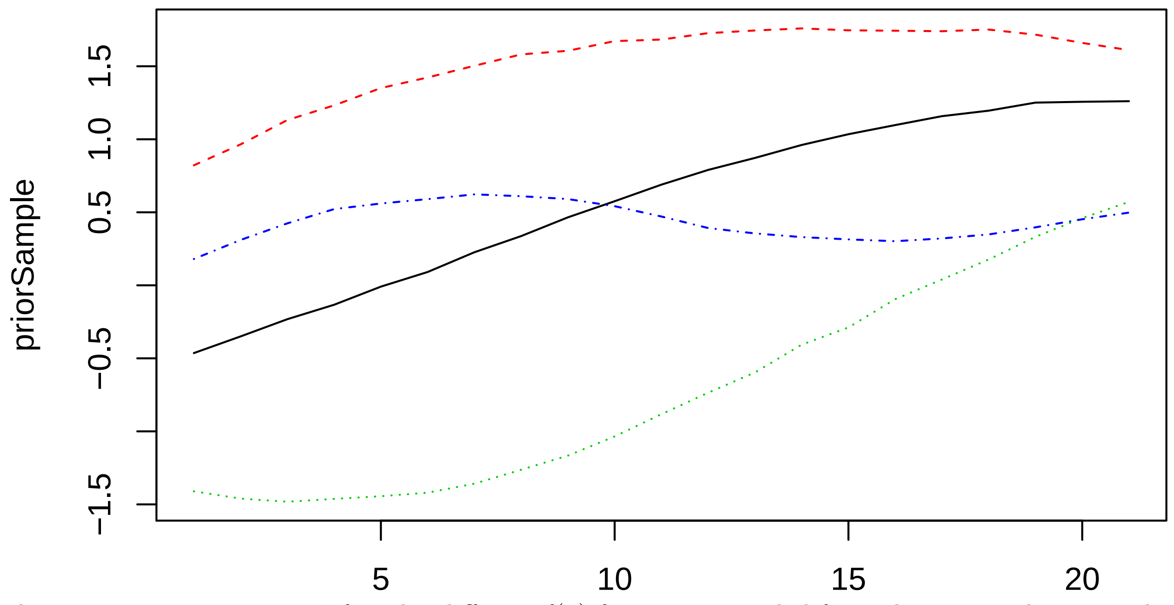
```
# opts_chunk$set(tidy.opts=list(width.cutoff=60))

opts_knit$set(root.dir= '.')
k_Lcov = function(x,lam)
{
  cc = matrix(1,nrow=length(x[,1]),ncol=length(x[,1]) )
  x1 = cc%*%x
  x1 = x1/length(x[,1])
  a = x - x1
  a = t(a)%*%a
  a = a/(length(x[,1])-1)
  # cc = exp(-1/(2*lam)*a)
return(a)
}

k_cov = function(x,lam)
{
  # exp(-1/(2*lam)* (x - mean(x) )^2)
  xbar = colMeans(x)
  # xbar = diag(xbar)
  # print(xbar)
  c= matrix(0,nrow=length(x[1,]),ncol=length(x[1,]) )
  for (i in 1:length(x[,1]))
  {
    temp = x[i,]-xbar
    c = c + outer(temp,temp)
  }
  c = c/(length(x[,1])-1)
  return(c)
}
k = function(x,x_,lam)
{
  #t(x) repiclate then x replicate then subtract then square
  n = length(x)
  n_ = length(x_)
  x = replicate(n_,x) #rep not dot product
  x_ = t(replicate(n,x_))
  r=exp(-1/(2*lam)*( x- x_ )^2)
  return(r)
}

dra = function(n, p , nr )
```

```
{
  x = matrix(0,n,p)
  for (i in 1:nr) {
    u = replicate(p, runif(n)) #make x a matrix
    s = cov(u)
    L = chol(s)
    x[i] = norm(L%*%t(L) - s)
    # x[i] = u%*%L
  }
  x
}
dra_prior = function(x, x_,point, lam)
{
  n = point
  p = matrix(0,nrow=point,ncol=n)
  ss = k(x,x_,lam)#this is LAMBDA Maybe change back to k_cov
  diag(ss) = diag(ss) +0.0001
  L = t(chol(ss)) #chol2inv
  d = length(L[1,])
  #n sample size
  #d dimension
  Z = matrix(rnorm(n*d),nrow=d,ncol=n)
  mu = 0# 1:d
  r = mu + L%*%Z
}
dra_samp = function(ss, point)
{
  n = point
  p = matrix(0,nrow=point,ncol=n)
  L = t(chol(ss))
  d = length(L[1,])
  #n sample size
  #d dimension
  Z = matrix(rnorm(n*d),nrow=d,ncol=n)
  mu = 0# 1:d
  r = mu + L%*%Z
}
```

```
n = 5 #row
p = 4 #col
x = seq(-1,1, by= 0.1)
x_=seq(-1,1, by =0.1)
priorSample = dra_prior(x,x_,p, 0.1) #row,col,nr, lam
matplot(priorSample, type='l')
```



```
n = 5 #row
p = 4 #col
x = seq(-1,1, by= 0.1)
x_=seq(-1,1, by =0.1)
priorSample = dra_prior(x,x_,p, 1.0) #row,col,nr, lam
matplot(priorSample, type='l')
```

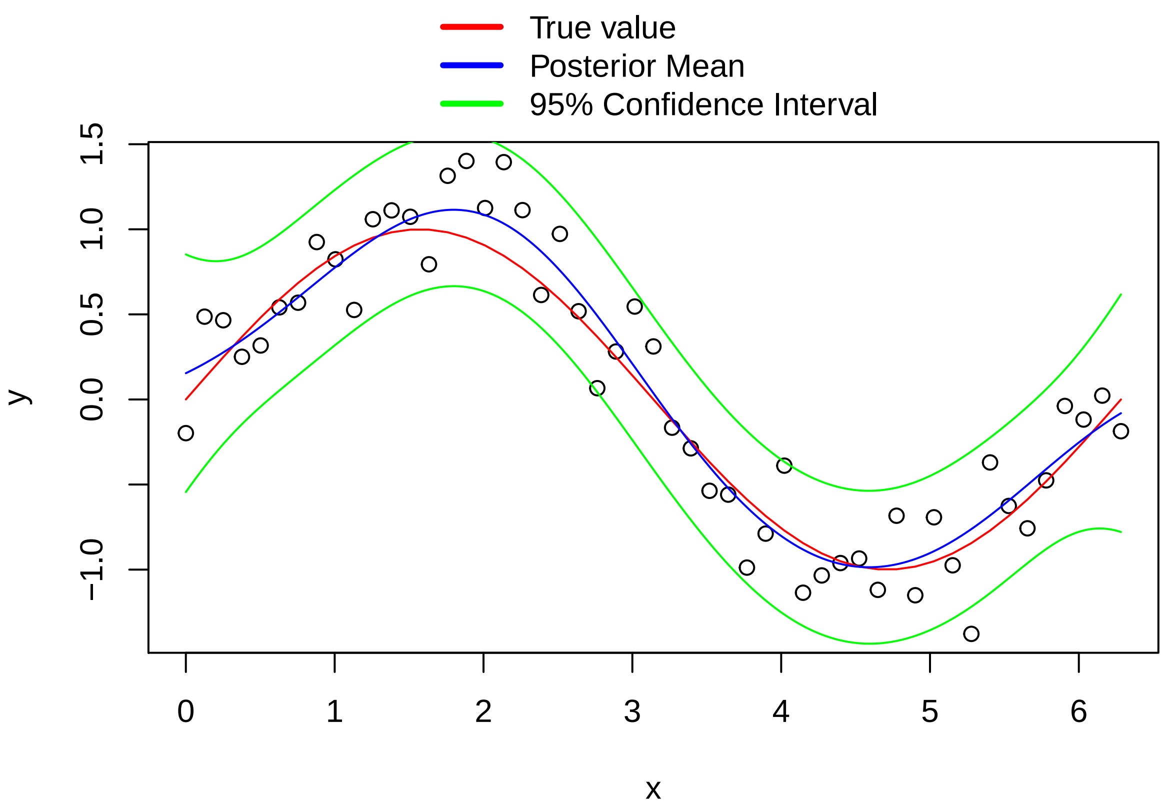


For the gaussain process prior I found 4 different f(x) functions sampled from the prior. These are plotted for lambda 0.1 and 1.0. I found that the smaller lambda the higher smoothness in the function and with smaller lambda values the plot became jagged. The method for sampling from this multivariate gaussian followed the method found in: https://journal.r-project.org/archive/2013-2/hofert.pdf
To sample you use the forumula:

$$X = \mu + AZ$$

Where X is the array of samples, μ is the mean, A is the decomposition of the matrix $\Sigma = AA^T$, and Z is a random vector. I found this using the cholsky decomposition.

```r
sigma_noise = 0.5
x = seq(0,2*pi,2*pi/50)
e = rnorm(x,0,sigma_noise^2)
y = sin(x)+e
plot(x,y)
lines(x,sin(x),col='red')
xx = seq(0,2*pi,2*pi/1000)
lambda=1.0
ker = k(x,x,lambda)
diag(ker) = diag(ker)+sigma_noise  #   sigma_noise
k_1 = chol2inv(chol(ker))
kxx_x = k(xx,x,lambda)
## lambda big... singular problem want it to not be all 1s, all 0s. L controls this.

ystar = kxx_x%*%(k_1%*%y)
kxx_xx = k(xx,xx,lambda)
kx_xx = k(x,xx,lambda)
vstar = kxx_xx - kxx_x %*% (k_1 %*% kx_xx)
lines(xx,ystar,col='blue')
lines(xx,ystar+1.96*sqrt(diag(vstar)),col='green')
lines(xx,ystar-1.96*sqrt(diag(vstar)),col='green')
legend("bottom", inset=c(0.0,1.0),xpd=TRUE,bty='n', c("True value","Posterior Mean", "95% Confidence In
       col = c("red", "blue","green"), lwd=3)
```



For this problem we plotted sin(x) for 1 cycle. Then we added gaussian noise for some parameter. Then a gaussian process was used. The function we are comes from the gaussian process from above:

$$y = f(x) + \epsilon$$

$$f \sim N(0, k(x, x'))$$

with kernel matrix:

$$k(x,x') = exp(\frac{1}{\lambda}(x-x')^2)$$

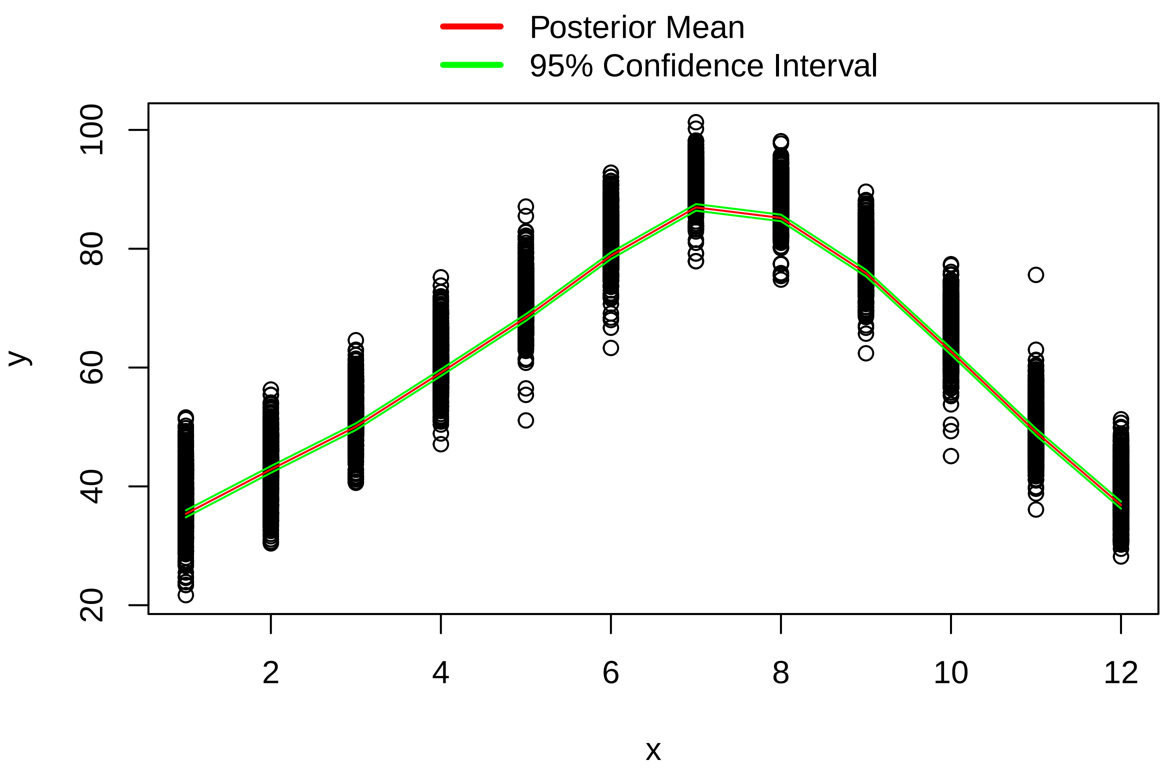
We find the kernel between the vectors x the true data and x' the samples we are finding. We can then use the equations to plot the posterior predictive distribution:

$$\mu = k(x',x)k(x,x')^{-1}y$$

$$\Sigma = k(x',x') - k(x',x)k(x,x)^{-1}k(x,x')$$

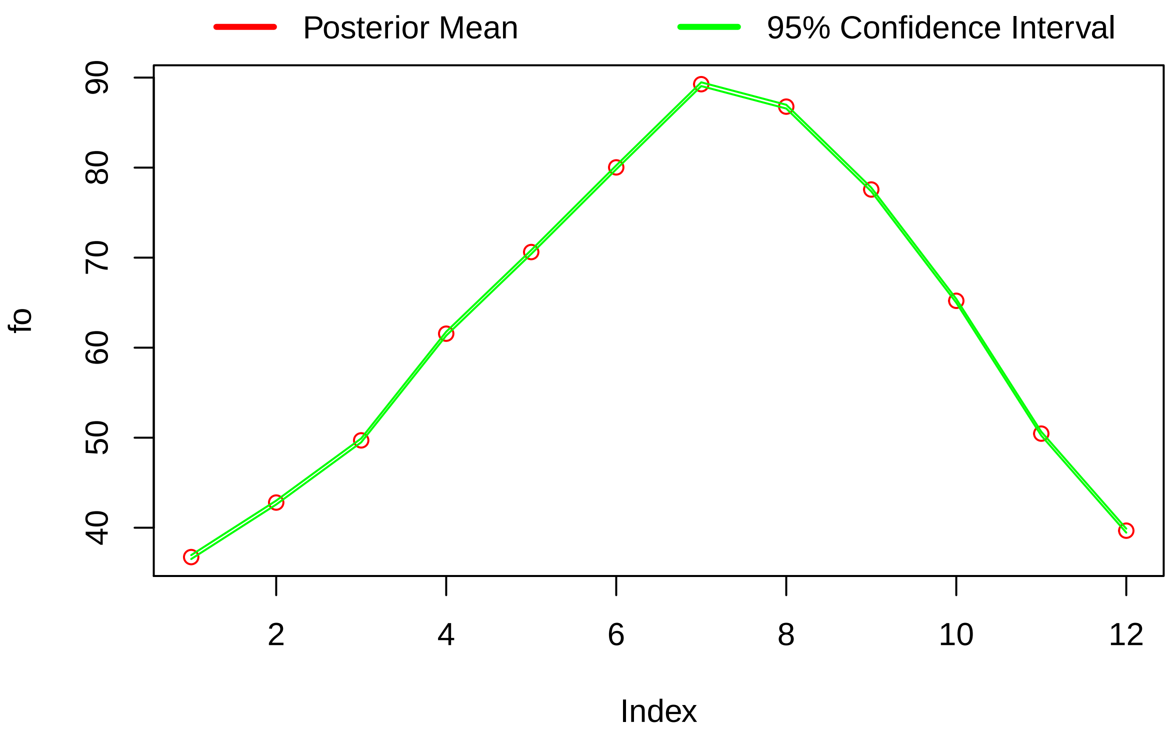
To plot this predictive posterior distribution we plot the array of μ and add/subtract the sqrt of the diagnol of the Σ matrix as that is the variance of our distribution at those points.
From experimentation we can see that the higher λ values will give smoother results that can almost exactly approximate the real data. This is because the smoothness value is increased. If we decrease the values of λ we get less smoothness as this increases the distance between x and x' in our kernel function. This is related to the corrleation of x and x'. I found that any $1.5 \geq \lambda 1 \geq 0.5$ came very close to the exact approximation of the true function. $\lambda \geq 1.5$ and the results would become very smooth possibly underfitting the data and $\lambda \leq 0.5$ the results would be very jagged trying to fit each of the noise points and possible overfitting the true function.

```r
# slc = read.csv('/Users/Benjamin/Documents/PROBMOD/Hmk5/SaltLakeTemperatures.csv')
slc = read.csv('SaltLakeTemperatures.csv')
slc <- slc[ slc$AVEMAX != -9999 , ]
slc$Year <- sapply(slc$DATE, function(x) substring(x, first=1,last=4))
slc$Month <-sapply(slc$DATE, function(x) substring(x, first=5,last=6 ))

x = as.numeric(slc$Month)
xx = seq(1,12,1)
y = as.numeric(slc$AVEMAX)
sigma_noise = 50
lambda=1.0
ker = k(x,x,lambda)
diag(ker) = diag(ker)+sigma_noise
k_1 = chol2inv(chol(ker))

kxx_x = k(xx,x,lambda)
# lambda big... singular problem want it to not be all 1s, all 0s. L controls this.
ystar = kxx_x%*%(k_1%*%y)
kxx_xx = k(xx,xx,lambda)
kx_xx = k(x,xx,lambda)

vstar = kxx_xx - kxx_x %*% (k_1 %*% kx_xx)
d = data.frame(xx,ystar,diag(vstar))

plot(x,y)
lines(xx,ystar,col='red')
lines(xx,ystar+1.96*sqrt(diag(vstar)),col='green')
lines(xx,ystar-1.96*sqrt(diag(vstar)),col='green')
legend("bottom", inset=c(0.0,1.0),xpd=TRUE,bty='n', c("Posterior Mean", "95% Confidence Interval"),
       col = c("red", "green"), lwd=3)
```

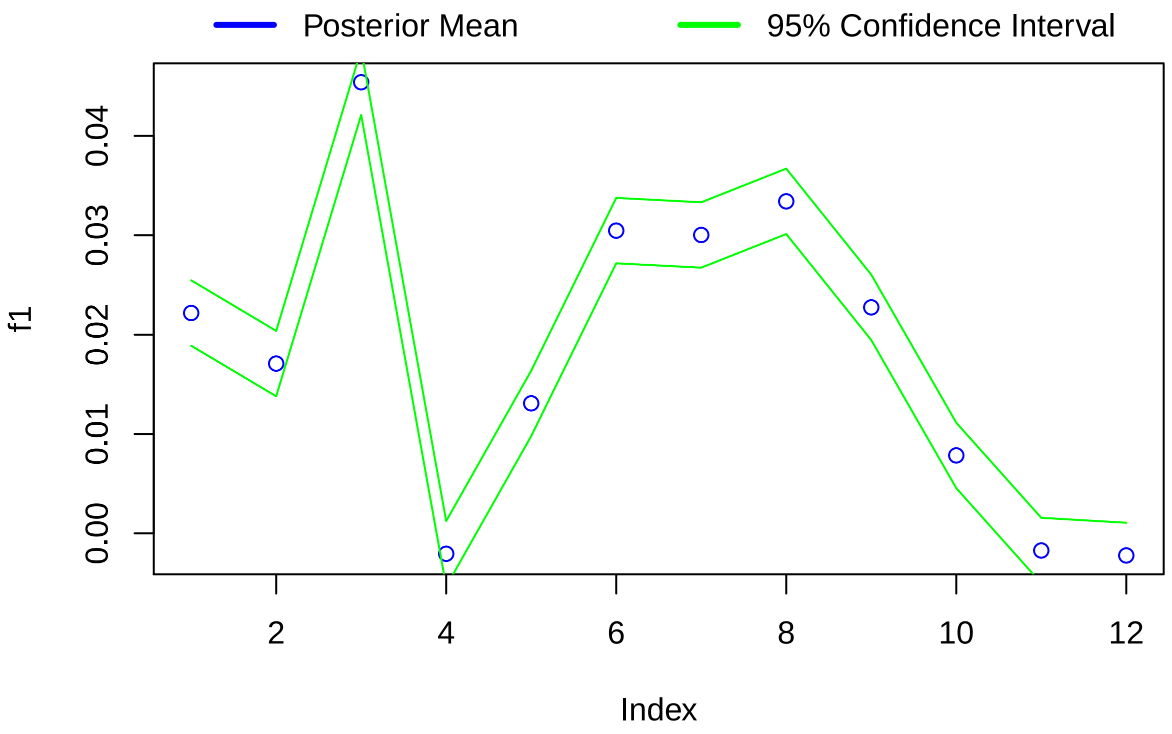


```
y1 = slc$AVEMAX
y1[is.na(y1)]<-0
xo = slc[slc[,"Year"]==1906,"Month"]
yo = slc[slc[,"Year"]==1906,"AVEMAX"]
x = as.numeric(slc$Month)
z =as.numeric(slc$Year)-1906

f1s = seq(1,12,1)
fos = seq(1,12,1)

sigma_noise = 1
lambda=1.0

k2 = k(f1s,x,lambda)

z0 = matrix(z, nrow=length(f1s), ncol=length(z), byrow = TRUE)
k1 = z0*k2

ker = k(x,x,lambda)
zout = outer(z,z)
ker1 = ker+ zout*ker

diag(ker1) = diag(ker1)+sigma_noise
k_1 = chol2inv(chol(ker1))

k2k1= rbind(k2,k1)
pmc = k2k1%*%k_1%*%y1
fo = k2%*%k_1%*%y1
f1 = k1%*%k_1%*%y1
```

```
ks_s = k(f1s,f1s,lambda)
m0 = matrix(0,nrow=length(f1s),ncol=length(f1s))
mc = cbind(ks_s,m0)
mr = cbind(m0,ks_s)
post_cov = rbind(mc,mr) - k2k1%*%k_1%*%t(k2k1)

plot(fo,col='red')
lines(fo+1.96*sqrt(diag(post_cov))[0:length(fo)],col='green')
lines(fo-1.96*sqrt(diag(post_cov))[0:length(fo)],col='green')
legend("bottom", inset=c(0.0,1.0),xpd=TRUE,bty='n',horiz=TRUE, c("Posterior Mean", "95% Confidence Inte
       col = c("red", "green"), lwd=3)
```



```
plot(f1,col='blue')
lines(f1+1.96*sqrt(diag(post_cov))[length(fo):length(fo)*2],col='green')
lines(f1-1.96*sqrt(diag(post_cov))[length(fo):length(fo)*2],col='green')
legend("bottom", inset=c(0.0,1.0),xpd=TRUE,bty='n',horiz=TRUE, c("Posterior Mean", "95% Confidence Inte
       col = c("blue", "green"), lwd=3)
```



From the last plot we are looking at the slope of the equation:

$$temperature = f_1(x) + f_0(x)z + \epsilon$$

Where x is the month, z is the year, and ϵ is some added noise to the model. The slope is indicative of the amount of rising temperature over the span of 100 years as given in the data. We are sampling from the equation:

$$\begin{bmatrix} y \\ f_1^* \\ f_0^* \end{bmatrix} = N\left[0, \begin{bmatrix} k(y,y) & k(y,f_0^*) & k(y,f_1^*) \\ k(f_0^*,y) & k(f_0^*,f_0^*) & k(f_0^*,f_1^*) \\ k(f_1^*,y) & k(f_1^*,f_0^*) & K(f_1^*,f_1^*) \end{bmatrix}\right]$$

Using a similar method for finding the posterior mean and covariance matrix that we have used before we get the mean posterior:

$$\mu = \begin{bmatrix} k(f_0^*,y) \\ k(f_1^*,y) \end{bmatrix} k(y,y)^{-1} y$$

And posterior covariance:

$$\Sigma = \begin{bmatrix} k(f_0^*,f_0^*) & k(f_0^*,f_1^*) \\ k(f_1^*,f_0^*) & k(f_1^*,f_1^*) \end{bmatrix} - \begin{bmatrix} k(f_0^*,y) \\ k(f_1^*,y) \end{bmatrix} k(y,y)^{-1} k(y,f_0^*) k(y,f_1^*)$$

These values are found using f* ranges of 1:12 for each month. The interesting part is including Z in the equations. To find k(y,y) we need to include the z term for f_0^* and f_1^* using the equations:

$$f_1^* = k(f_1^*, x)$$

$$k(y,y) = k(x,x) + k(x,x)z^T z + \epsilon$$

The transpose is just so that the k(x,x) matrix is multiplied by Z as the variable of interest. This will give us the slope of our model in the form of f_1^*. This is the last plot. Looking at this posterior mean for the slope (change over years) we see that the overall values of the temperature are increasing. Even if by a small margin. Modifying the parameters did change the values, however the results weren't changed drastically. Larger lambda values, $\lambda \geq 10$ would allow the november slope to be negative and showed less variance between all the data points while $\lambda \leq 1$ was strictly positive and had sharp changes in the spring, while maintaining smooth variance during summer and fall.