Lecture 3 Exercises

1. For a Bayesian linear regression model with prior on $p(w) = N(w|\mu, D^{-1})$ and q known, show that w_{Bayes} solves $(D + q\phi^T\phi)w_{Bayes} = q\phi^T t + D\mu$

To solve for bayes we take the known posterior probability:

$-2logp(w|t, X) = -2qt^T\phi w + qw^T\phi^T\phi w + (w-\mu)^T D(w-\mu) + const$

To get the best estimate of w, we take the derivative of the posterior probability and set it equal to zero:

$\partial - 2logp(w|t, X)/\partial w = 0$

To take the derivative of the first term:

$\partial - 2qt^T\phi w/\partial w = -2qt^T\phi$

The derivative of the second term uses the property $\partial x^T Ax/\partial x = 2Ax$

$\partial qw^T\phi^T\phi w/\partial w = 2q\phi^T\phi w$

Expanding the third term $(w-\mu)^T D(w-\mu)$ gives:

$= (w^T D - \mu^T D)(w-\mu)$

$= w^T Dw - \mu^T Dw - w^T D\mu + \mu^T D\mu$

$\partial(w^T Dw - \mu^T Dw - w^T D\mu + \mu^T D\mu)/\partial w = 2w^T D - 2\mu^T D$

So we have:

$0 = -2qt^T\phi + 2qw^T\phi^T\phi + 2w^T D - 2\mu^T D$

Move negative terms to the LHS and divide both sides by 2:

$qt^T\phi + \mu^T D = qw^T\phi^T\phi + w^T D$

$qw^T\phi^T\phi + w^T D = qt^T\phi + \mu^T D$

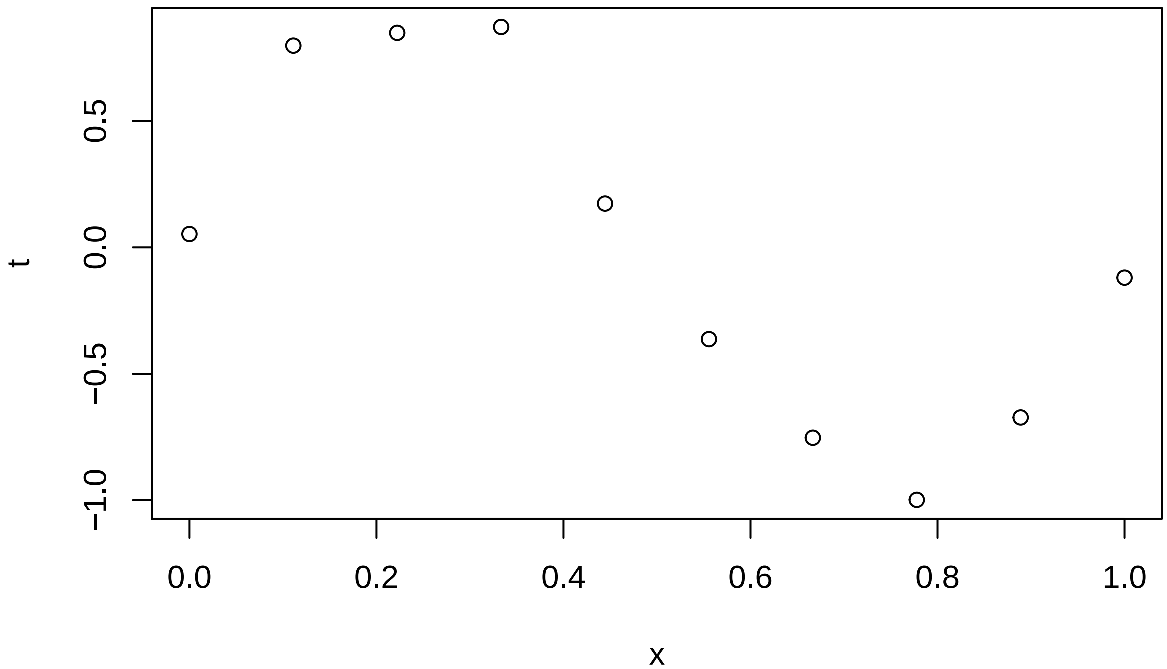
Take the transpose of both sides (note: $D^T = D$:

$q\phi^T\phi w + Dw = q\phi^T t + D\mu$

$(D + q\phi^T\phi)w_{Bayes} = q\phi^T t + D\mu$

2. Curve fitting (pt1) The aim is to learn a smooth function from the cloud of points stored in curve_data.txt using Bayesian linear regression models. In all that follows the prior $p(w) = N(w|0, \delta^{-1}I)$ is used and q, δ are constants specified by the user.

1. Plot the data

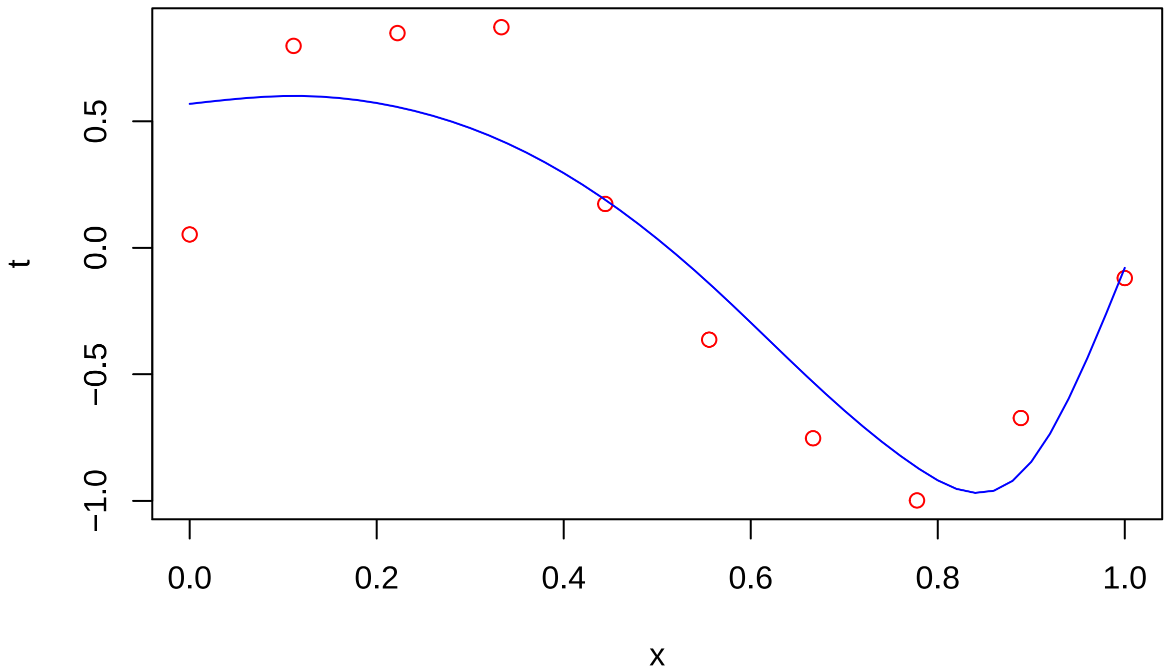


2.2 Write a function in R, called phix that takes as input a scalar x (the input in curve fitting), with values in [0, 1], M the number of bases functions, and a categorical variable that specifies the type of basis used, and returns the vector of basis functions evaluated at x. Hence a call of the function phix(0.3,4,"poly") should return c(1.0000, 0.3000, 0.0900, 0.0270, 0.0081). Code it up so that the option "poly" gives the polynomial bases and "Gauss" the Gaussian kernels with means mui equally spaced in [0, 1], with $\mu = 0$ **and** $mu_M = 1$.

```
phix <- function(x, M, basis) {
  # Initialize vector to return
  u <- rep(0, M+1)
  # \mus for Gauss
  mus <- rep(0, M+1)
  interval <- 1 / M

  if (basis == 'poly') {
    # evaluate x at each value of p, e.g. x^0, x^1, ..., x^M
    for (p in 0:M) u[p+1] <- x^p
  } else if (basis == 'Gauss') {
    # Populate \mus with M intervals in [0,1]
    for (n in 1:M) mus[n+1] = mus[n] + interval
    # Generate Gaussian Kernels
    for (p in 0:M) u[p+1] <- exp((-(x - mus[p+1])^2)/0.1)
  }

  u
}
# Test: round(phix(0.3,4,'poly'), digits = 4) == c(1, 0.3, 0.09, 0.027, 0.0081)
```

2.3. Write a function in R, called post.params, that takes as input the training data, M, the type of basis, the function phix, δ **and q and returns the parameters of the posterior distribution,** w_{Bayes} **and** Q**.**

```r
library(MASS)

constructPhi <- function(features, M, basis_type) {
  phi <- matrix(nrow = length(features), ncol = M + 1)
    # Construct a phi matrix that is phix for each value of phix(x, M, 'Gauss') and append to phi matri
  for (i in 1:(length(features))) {
    phi[i,] = phix(features[i], M, basis_type)
  }
  phi
}

# Part 3: post.params
post.params <- function(training_data, M, basis_type, delta, q) {
  # regularization coefficient
  # training_data = curve_data
  # q = (1/0.1)^2
  # delta = 2.0
  # basis_type = 'Gauss'
  lambda = delta / q

  # Note: this requires the training data to have an x column
  xs = training_data$x
  # M = 9
  phi <- constructPhi(xs, M, basis_type)
  t = training_data$t

  # Slide 10 of Bayessian_regression.pdf
  w_Bayes = ginv(lambda * diag(M+1) + t(phi) %*% phi) %*% t(phi) %*% training_data$t

  D = delta * diag(M + 1)
  Q = D + q * (t(phi) %*% phi)

  # FIXME: RETURN Q?
  list(w_Bayes, Q)
}

M = 9
delta = 2.0
q = (1/0.1)^2

params <- post.params(curve_data, M, 'poly', delta, q)
w_Bayes <- params[[1]]
Q <- params[[2]]
phi <- constructPhi(curve_data$x, M, 'poly')
result <- phi %*% w_Bayes

# Part 4: Plot
plot(curve_data, col = 'red')
# Don't know how to smooth this out :(
library(splines)
lines(predict(splines::interpSpline(curve_data$x, result)), col = 'blue')
```



======= Roger's code:

```
library(ggplot2)
data <- read.table("curve_data.txt")

phix <- function(x, M, option) {
  phi <- rep(0,M)
  if (option == "poly") {
    for (i in 1:(M)) {
      phi[i] <- x**i
    }
  }
  if (option == "Gauss") {
    for (i in 0:(M-1)) {
      phi[i] <- exp(-((x-i/(M))**2)/0.1)
    }

  }
  phi
}
# Test: round(phix(0.3,4,'poly'), digits = 5) == c(0.3000, 0.0900, 0.0270, 0.0081)

post.params <- function(data, M, option, delta, q) {
  # Q = D + qPhiTPhi
  # wbayes = Q-1(qPhiTt + Dmu)

  phi = phix(data$x[1],M, option)
  for (i in 2:length(data$x)) {
    phi_ <- phix(data$x[i], M, option)
    phi = rbind(phi, phi_)
  }
  phi = cbind(rep(1,10),phi)
  phi
```

```r
  Q = diag(delta,M+1) + q*t(phi)%*%phi
  #lambda = delta/q
  w = solve(Q)%*%(q*t(phi)%*%data$t)
  #w = solve(diag((2.0)/((1/0.1)**2),M+1)+t(phi)%*%phi)%*%t(phi)%*%data$t
  t <- phi%*%w
  return(list(Q,w,t))
}

answer <- post.params(data, 9, "poly", 2.0, (1/0.1)**2)

that <- answer[[3]]

p <- ggplot(cbind(data,that), aes(x=data$x, y=data$t)) + geom_point()
p + stat_smooth(method = "loess", formula = that ~ x, size = 1)
```