Solutions: Assignment #3 STA410H1F/2102H1F

1. (a) Since $E(\boldsymbol{V}^T A \boldsymbol{V}) = \mathrm{tr}(A)$, we have

$$\mathrm{Var}(\widehat{\mathrm{tr}}(A)) = \frac{1}{m}\mathrm{Var}(\boldsymbol{V}^T A \boldsymbol{V}) = \frac{1}{m}\left\{E\left[(\boldsymbol{V}^T A \boldsymbol{V})^2\right] - \mathrm{tr}(A)^2\right\}$$

and so it suffices to minimize

$$E\left[(\boldsymbol{V}^T A \boldsymbol{V})^2\right] = \sum_{i=1}^n \sum_{j=1}^n \sum_{k=1}^n \sum_{\ell=1}^n a_{ij} a_{k\ell} E(V_i V_j V_k V_\ell)$$

Since we are assuming that $V_1, \cdots, V_n$ are independent with mean 0 and variance 1, it follows that $E(V_i V_j V_k V_\ell) = 0$ or 1 unless $i = j = k = \ell$. Thus

$$E[(\boldsymbol{V}^T A \boldsymbol{V})^2] = \sum_{i=1}^n a_{ii}^2 E(V_i^4) + \text{constant}$$

Thus it suffices to find a distribution with mean 0 and variance 1 with minimal fourth moment. Note that

$$E(V_i^4) = \mathrm{Var}(V_i^2) + 1$$

and $\mathrm{Var}(V_i^2)$ is minimized if V_i^2 is constant (with probability 1). Since $E(V_i)$ and $E(V_i^2)$ must equal 0 and 1, respectively, it follows that $\mathrm{Var}(\widehat{\mathrm{tr}(A)}$ is minimized by taking the elements of $\boldsymbol{V}_i$ to be ± 1 each with probability 1/2.
Alternatively, we can use Jensen's inequality: $E(X^2) \geq [E(X)]^2$ with equality iff X is constant. Setting $X = V_i^2$, it follows that $E(V_i^4) \geq 1$ (since $E(V_i^2) = 1$) with equality if $V_i^2 = 1$ or $V_i = \pm 1$; the condition $E(V_i) = 0$ implies that $V_i = \pm 1$ each with probability 1/2.

(b) Defining

$$H = X(X^T X)^{-1} X^T = \begin{pmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{pmatrix},$$

we have

$$\begin{aligned} H\begin{pmatrix} \boldsymbol{V} \\ \boldsymbol{0} \end{pmatrix} &= \begin{pmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{pmatrix}\begin{pmatrix} \boldsymbol{V} \\ \boldsymbol{0} \end{pmatrix} \\ &= \begin{pmatrix} H_{11}\boldsymbol{V} \\ H_{21}\boldsymbol{V} \end{pmatrix} \end{aligned}$$

and so $H_{11}\boldsymbol{V}$ is the first ℓ components of the right hand side above. Likewise,

$$\begin{aligned} H\begin{pmatrix} H_{11}^{k-1}\boldsymbol{V} \\ \boldsymbol{0} \end{pmatrix} &= \begin{pmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{pmatrix}\begin{pmatrix} H_{11}^{k-1}\boldsymbol{V} \\ \boldsymbol{0} \end{pmatrix} \\ &= \begin{pmatrix} H_{11}^k\boldsymbol{V} \\ H_{21}H_{11}^{k-1}\boldsymbol{V} \end{pmatrix} \end{aligned}$$

with $H_{11}^k \boldsymbol{V}$ the first ℓ components of the right hand side above.
(c) There are a number of different ways to proceed here. One approach is to use the R function `set.seed` so that the function `leverage` uses the same stream of random numbers for both designs. Another approach (shown below) is to modify `leverage` to take two designs as inputs.
Here is a modification of leverage:

```
leverage <- function(x1,x2,w,r=10,m=100) {
               qrx1 <- qr(x1)
               qrx2 <- qr(x2)
               n <- nrow(x1)
               lev1 <- NULL
               lev2 <- NULL
               for (i in 1:m) {
                   v <- ifelse(runif(n)>0.5,1,-1)
                   v[-w] <- 0
                   v01 <- qr.fitted(qrx1,v)
                   v02 <- qr.fitted(qrx2,v)
                   f1 <- v01
                   f2 <- v02
                   for (j in 2:r) {
                      v01[-w] <- 0
                      v02[-w] <- 0
                      v01 <- qr.fitted(qrx1,v01)
                      v02 <- qr.fitted(qrx2,v02)
                      f1 <- f1 + v01/j
                      f2 <- f2 + v02/j
                      }
                   lev1 <- c(lev1,sum(v*f1))
                   lev2 <- c(lev2,sum(v*f2))
                   }
                se1 <- exp(-mean(lev1))*sd(lev1)/sqrt(m)
                se2 <- exp(-mean(lev2))*sd(lev2)/sqrt(m)
                se.diff <- 0.5*(exp(-mean(lev1))+exp(-mean(lev2)))*
                                  sd(lev1-lev2)/sqrt(m)
                lev1 <- 1 - exp(-mean(lev1))
                lev2 <- 1 - exp(-mean(lev2))
                r <- list(lev=c(lev1,lev2),std.err=c(se1,se2,se.diff))
                r
                }
```

We define the two design matrices as follows:

```
> x <- c(1:1000)/1000
> X1 <- 1
> for (k in 1:5) X1 <- cbind(X1,cos(2*k*pi*x),sin(2*k*pi*x))
> library(splines) # loads the library of functions to compute B-splines
> X2 <- cbind(1,bs(x,df=10))
```

and use the function above to compute the leverages. (The third element of `$std.err` is an estimate of the standard error of the difference.) The following R code is used to compute the leverages for the two designs for the 20 subsets of 50 observations; we set $r = 15$ and $m = 1000$ (although m in particular could be taken to be much smaller).

```
> levs <- NULL
> for (k in 1:20) {
+     w <- c(((k-1)*50+1):(50*k))
+     r <- leverage(X1,X2,w,m=1000,r=15)
+     levs <- rbind(levs,r$lev)
+     }
```

Figure 1 shows the leverages for the two designs; note that the leverages for the sinusoidal design are nearly constant while for the B-spline design, the leverages of the subsets of observations near 0 and 1 are much greater than those closer to 0.5. (Due to the symmetry of both designs, the leverage of points 1–50, 51–100 etc should be the same as the leverage of points 951-1000, 901-950 etc, respectively; this is reflected in Figure 2.) The standard errors for the estimates of the leverages are less than 0.016 and the standard errors of the differences leverages between the two designs are less than 0.01 with the exception of the two extreme subsets where the standard error of the difference is about 0.03.
What are the implications of this? Both the B-spline and sinusoidal design are useful for estimating a smooth function $g(x)$ for $0 \le x \le 1$ in the model $y_i = g(x_i) + \varepsilon_i$ $(i = 1, \cdots, n)$ where we approximate g by either B-splines or sinusoids. The results suggest that the B-spline design makes "better" use of the observations where x_i is close to either 0 or 1; in other words, these observations have higher influence on the B-spline estimates. (This higher influence is desirable if the model is a good approximation but is potentially dangerous otherwise.) To illustrate this, we generate $\{y_i : i = 1, \cdots, 1000\}$ using the following R code:

```
> x <- c(1:1000)/1000
> y <- exp(-x)*cos(6*pi*x) + rnorm(1000,0,0.1)
> library(splines)
> X1 <- NULL
> for (k in 1:5) X1 <- cbind(X1,cos(2*k*pi*x),sin(2*k*pi*x))
> X2 <- bs(x,df=10)
> r1 <- lm(y~X1)
```

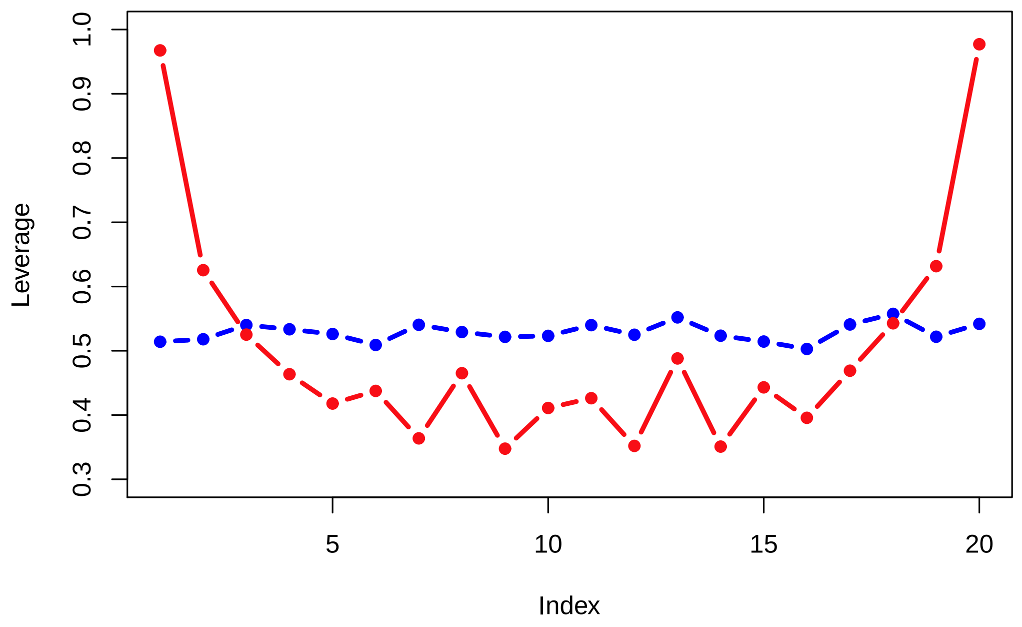

Figure 1: Estimated leverages for sinusoidal (blue) and B-spline (red) designs

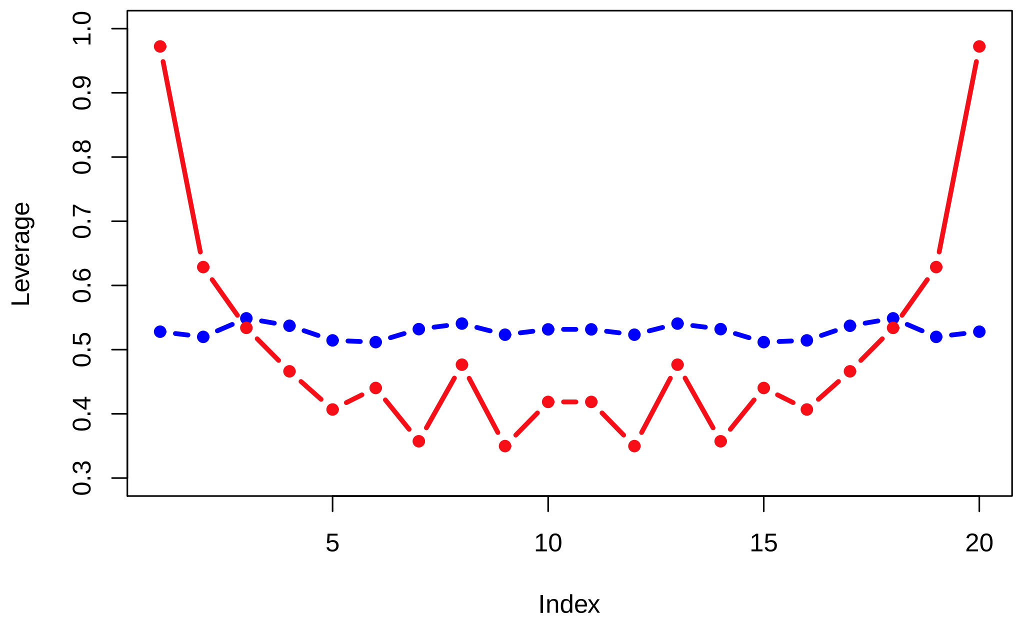

Figure 2: Estimated leverages for sinusoidal (blue) and B-spline (red) designs assuming symmetry

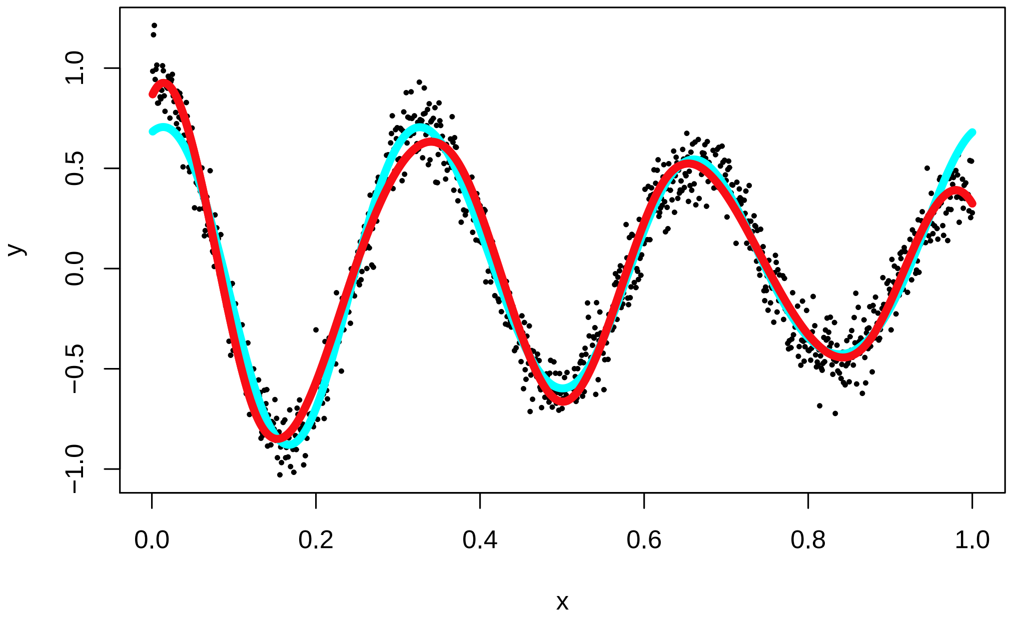

Figure 3: Simulated data with B-spline (red) and sinusoid (cyan) estimates of $g(x) = \exp(-x)\cos(6\pi x)$.

```
> r2 <- lm(y~X2)
> plot(x,y,pch=20)
> lines(x,r1$fitted.values,col="cyan",lwd=5)
> lines(x,r2$fitted.values,col="red",lwd=5)
```

In Figure 3, the B-spline estimate of g is shown in red and the sinusoid estimate is shown in cyan. Note that the B-spline estimate seems to fit the data much better than the sinusoid estimate for values of x close to 0 and 1; the two estimates are very similar for values of x away from 0 and 1.

2. (a) If X has a Gamma distribution then $E(X) = \alpha/\lambda$ and $\mathrm{Var}(X) = \alpha/\lambda^2$ or equivalently, $\alpha = E(X)^2/\mathrm{Var}(X)$ and $\lambda = E(X)/\mathrm{Var}(X)$. Therefore, if $\bar{x}$ and s^2 are the sample mean and variance of $x_1, \cdots, x_n$, method of moments estimators are $\widehat{\alpha} = \bar{x}^2/s^2$ and $widehat\lambda = \bar{x}/s^2$.
(b) Define $\psi(\alpha)$ and $\psi'(\alpha)$ to be the first two derivatives of $\ln\Gamma(\alpha)$, i.e. ψ and ψ' are the digamma and trigamma functions. For fixed α and λ, the score vector is

$$\begin{pmatrix} \displaystyle\sum_{i=1}^{n} \ln(x_i) + n\{\ln(\lambda) - \psi(\alpha)\} \\ \displaystyle n\alpha/\lambda - \sum_{i=1}^{n} x_i \end{pmatrix}$$

and the Fisher information matrix is

$$\begin{pmatrix} n\psi'(\alpha) & -n/\lambda \\ -n/\lambda & n\alpha/\lambda^2 \end{pmatrix}.$$

The following function implements the Newton-Raphson algorithm to compute the MLE; it also retains the initial (method of moments) estimate as well as the one-step estimate.

```
gammamle <- function(x,eps=1.e-6) {
              n <- length(x)
              alpha <- mean(x)^2/var(x)
              alpha1 <- alpha
              lambda <- mean(x)/var(x)
              theta <- c(alpha,lambda)
              theta0 <- theta
              score1 <- sum(log(x)) + n*(log(lambda) - digamma(alpha))
              score2 <- n*alpha/lambda-sum(x)
              score <- c(score1,score2)
              iter <- 0
              partialsteps <- 0
              while (max(abs(score))>eps) {
                 info11 <- n*trigamma(alpha)
                 info12 <- -n/lambda
                 info22 <- n*alpha/lambda^2
                 info <- matrix(c(info11,info12,info12,info22),ncol=2)
                 mult <- 1
                 repeat {
                     theta1 <- theta + mult*solve(info,score)
                     if (min(theta1)<=0) {
                          mult <- 0.9*mult
                          partialsteps <- partialsteps + 1
                          }
                       else break
                     }
                 theta <- theta1
                 alpha <- theta[1]
                 lambda <- theta[2]
                 score1 <- sum(log(x)) + n*(log(lambda) - digamma(alpha))
                 score2 <- n*alpha/lambda-sum(x)
                 score <- c(score1,score2)
                 iter <- iter + 1
                 if (iter==1) theta2 <- theta
```

```
                  }
                info11 <- n*trigamma(alpha)
                info12 <- -n/lambda
                info22 <- n*alpha/lambda^2
                info <- matrix(c(info11,info12,info12,info22),ncol=2)
                cat("number of iterations = ",iter, "partial steps = ",
                    partialsteps, "\n")
                r <- list(mle=theta,init=theta0,onestep=theta2,
                      varcovar=solve(info))
                r
                }
```

The following illustrates its use for $n = 100$ with $\alpha = 0.5, 10$, and 0.01 and $\lambda = 1$.

```
> x <- rgamma(100,shape=0.5)
> r <- gammamle(x)
number of iterations =  4 partial steps =  0
> r
$mle
[1] 0.4648531 1.0717564
$init
[1] 0.4973883 1.1467688
$onestep
[1] 0.462355 1.065997
$varcovar
            [,1]        [,2]
[1,] 0.002905840 0.006699649
[2,] 0.006699649 0.040156790
> x <- rgamma(100,shape=10)
> r <- gammamle(x)
number of iterations =  3 partial steps =  0
> r
$mle
[1] 12.145770  1.160724
$init
[1] 11.596944  1.108275
$onestep
[1] 12.120641  1.158323
$varcovar
          [,1]       [,2]
[1,] 2.8716893 0.27443626
```

```
[2,] 0.2744363 0.02733607
> x <- rgamma(100,0.01)
> r <- gammamle(x)
number of iterations =  5 partial steps =  0
> r
$mle
[1] 0.01099171 0.93276209
$init
[1] 0.01632313 1.38518841
$onestep
[1] 0.008382625 0.711353622
$varcovar
             [,1]         [,2]
[1,] 1.221363e-06 0.0001036455
[2,] 1.036455e-04 0.8003418746
```

Supplemental problems:

3. (a) This is straightforward — if $\widehat{\boldsymbol{\beta}}_\lambda$ minimizes the objective function g, it also minimizes $a \times g(\boldsymbol{\beta}) + b$ for any $a > 0$ and b. In this case $a = 1/\lambda > 0$ and $b = -\tau/\lambda$.

(b) For $\boldsymbol{\beta} \in \mathcal{C}$,

$$\frac{1}{\lambda}\left\{\sum_{i=1}^{n}(y_i - \bar{y} - \boldsymbol{x}_i^T\boldsymbol{\beta})^2 - \tau\right\} = 0$$

for all $\lambda > 0$ and so

$$\lim_{\lambda\downarrow 0}\frac{1}{\lambda}\left\{\sum_{i=1}^{n}(y_i - \bar{y} - \boldsymbol{x}_i^T\boldsymbol{\beta})^2 - \tau\right\} = 0.$$

On the other hand, for $\boldsymbol{\beta} \notin \mathcal{C}$,

$$\left\{\sum_{i=1}^{n}(y_i - \bar{y} - \boldsymbol{x}_i^T\boldsymbol{\beta})^2 - \tau\right\} > 0$$

and so

$$\lim_{\lambda\downarrow 0}\frac{1}{\lambda}\left\{\sum_{i=1}^{n}(y_i - \bar{y} - \boldsymbol{x}_i^T\boldsymbol{\beta})^2 - \tau\right\} \to +\infty.$$

Thus for λ close to 0, $\widehat{\boldsymbol{\beta}}_\lambda$ minimizes

$$\sum_{j=1}^{p}|\beta_j| + h_\lambda(\boldsymbol{\beta})$$

where $h_\lambda(\boldsymbol{\beta}) = 0$ for $\boldsymbol{\beta} \in \mathcal{C}$ and $h_\lambda(\boldsymbol{\beta})$ is arbitrarily large for $\boldsymbol{\beta} \notin \mathcal{C}$. From this, we deduce that $\widehat{\boldsymbol{\beta}}_\lambda \to \widehat{\boldsymbol{\beta}}_0$ where $\widehat{\boldsymbol{\beta}}_0$ minimizes $\sum_{j=1}^{p}|\beta_j|$ on the set $\mathcal{C}$.

(c) Any $\boldsymbol{\beta}$ in $\mathcal{C}$ must satisfy the (first order) condition

$$X^T X\boldsymbol{\beta} = X^T \boldsymbol{y}$$

where X is the usual design matrix and $\boldsymbol{y}$ is the vector of responses. If we write each β_j as $\beta_j^+ - \beta_j^-$ (where $\beta_j^+ \geq 0$ and $\beta_j^- \geq 0$) so that $|\beta_j| = \beta_j^+ + \beta_j^-$, it follows that $\widehat{\boldsymbol{\beta}}_0 = \widehat{\boldsymbol{\beta}}^+ - \widehat{\boldsymbol{\beta}}^-$ minimizes

$$\sum_{j=1}^{p}(\beta_j^+ + \beta_j^-)$$

subject to

$$X^T X(\boldsymbol{\beta}^+ - \boldsymbol{\beta}^-) = X^T\boldsymbol{y} \quad \text{and} \quad \boldsymbol{\beta}^+ \geq \mathbf{0},\ \boldsymbol{\beta}^- \geq \mathbf{0}$$

which is a linear programming problem.

4. (a) For λ less than the threshold, there are a number of fixed point algorithms that can be used to solve the equation $g'(x^*) = 0$. The most obvious is the Newton-Raphson algorithm

$$x_{k+1} = x_k - \frac{g'(x_k)}{g''(x_k)}$$

where

$$\begin{aligned} g'(x) &= 2x - 2\alpha + \lambda\gamma\frac{|x|^\gamma}{x} \\ g''(x) &= 2 - \lambda\gamma(1-\gamma)|x|^{\gamma-2} \end{aligned}$$

Another possibility is the simple fixed point iteration

$$x_{k+1} = \alpha - \lambda\gamma\frac{|x_k|^\gamma}{2x_k}.$$

A "good" starting value for both algorithms is $x_0 = \alpha$, which minimizes g when $\lambda = 0$.

```
NR <- function(alpha,gamma,lambda,start,maxiter=50,eps=1.e-8) {
          g2 <- 2-gamma
          test <- 2*((2-2*gamma)/g2)^(1-gamma)*abs(alpha)^g2/g2
          if (lambda>=test) r <- 0
            else {
              if(missing(start)) start <- alpha
              x <- start
              gp <- 2*x - 2*alpha + lambda*gamma*abs(x)^gamma/x
              iter <- 0
              while (abs(gp)>eps && iter<maxiter) {
                 gpp <- 2 - lambda*gamma*(1-gamma)*abs(x)^(-g2)
                 x <- x - gp/gpp
                 gp <- 2*x - 2*alpha + lambda*gamma*abs(x)^gamma/x
                 iter <- iter + 1
                 }
              if (iter > maxiter) print("no convergence")
                else {
                   r <- x
                   print(paste("convergence in",iter,"iterations"))
                   }
              }
           r
           }

FP <- function(alpha,gamma,lambda,start,maxiter=50,eps=1.e-8) {
          g2 <- 2-gamma
          test <- 2*((2-2*gamma)/g2)^(1-gamma)*abs(alpha)^g2/g2
          if (lambda>=test) r <- 0
            else {
              if(missing(start)) start <- alpha
              x <- start
              gp <- 2*x - 2*alpha + lambda*gamma*abs(x)^gamma/x
              iter <- 0
              while (abs(gp)>eps && iter<maxiter) {
                 x <- alpha - lambda*gamma*abs(x)^gamma/(2*x)
                 gp <- 2*x - 2*alpha + lambda*gamma*abs(x)^gamma/x
                 iter <- iter + 1
                 }
              if (iter > maxiter) print("no convergence")
                else {
                   r <- x
                   print(paste("convergence in",iter,"iterations"))
                   }
              }
           r
           }
```

The functions can be used as follows:

```
> r <- NR(alpha=3,gamma=1/2,lambda=0.3)
[1] "convergence in 2 iterations"
> r
[1] 2.956380
> r <- FP(3,1/2,0.3)
```

```
[1] "convergence in 4 iterations"
> r
[1] 2.956380
```

Notice that the Newton-Raphson algorithm converges somewhat more quickly; this is true for other starting values as well although the difference seems to be less pronounced as the starting value moves away from α.

(b) The objective function in (2) is a function of the parameters $\beta_0, \beta_1, \cdots, \beta_p$. If we fix the values of all but one of the parameters (say β_j) then for the fixed values of β_k (for $k \neq j$), the objective function (2) becomes a function of β_j, and has the form (up to a constant that depends on β_k for $k \neq j$) of g. Thus for fixed β_k ($k \neq j$), we can minimize (2) with respect to β_j. If we cycle through the parameters $\beta_0, \beta_1, \cdots, \beta_p$ (using a coordinate descent algorithm), we will at each step decrease the value of the objective function (2) — the sequence of objective function values will converge to a limit although this limit is not necessarily equal to the minimum of (2) (due to the lack of convexity).

(c) This is quite difficult! Define

$$h(t) = g(\alpha t) = \alpha^2 \left(t^2 - 2t + \lambda |t|^\gamma |\alpha|^{\gamma - 2} \right).$$

For $t < 0$, $h'(t) < 0$ and thus $h(t)$ is strictly decreasing on the interval $(-\infty, 0]$. For $t > 1$, $h'(t) > 0$ and so $h(t)$ is strictly increasing on the interval $(1, \infty)$. Thus g is minimized at $x = t\alpha$ for some $t \in [0, 1]$. If 0 is a minimizer of g then we must have

$$|\alpha|^{2-\gamma} \left(t^2 - 2t \right) + \lambda t^\gamma \geq 0$$

for all $0 \leq t \leq 1$. In other words,

$$\lambda \geq |\alpha|^{2-\gamma} \max_{0 \leq t \leq 1} t^{1-\gamma}(2 - t)$$

Using calculus, it is easy to verify that the right hand side above is maximized for $t = 2(1-\gamma)/(2-\gamma)$ and so 0 minimizes g if, and only if,

$$\lambda \geq |\alpha|^{2-\gamma} \left(\frac{2}{2-\gamma} \right) \left(\frac{2(1-\gamma)}{2-\gamma} \right)^{1-\gamma}$$

Moreover, if $\gamma < 1$ and $\alpha \neq 0$ then strict inequality implies that 0 is the unique minimizer of g. If equality holds then g is minimized at $x \in \{0, 2\alpha(1-\gamma)/(2-\gamma)\}$; note that this set contains a single point when $\gamma = 1$.

5. We can write $\boldsymbol{x}_0 = \sum_{i=1}^{n} a_i \boldsymbol{v}_i$ where $a_i = \boldsymbol{x}_0^T \boldsymbol{v}_i$. Note that

$$\boldsymbol{x}_k = \frac{A^k \boldsymbol{x}_0}{\|A^k \boldsymbol{x}_0\|_2} = \frac{1}{\|A^k \boldsymbol{x}_0\|_2} \sum_{i=1}^{n} a_i \lambda_i^k \boldsymbol{v}_i = \frac{\lambda_1^k}{\|A^k \boldsymbol{x}_0\|_2} \sum_{i=1}^{n} a_i (\lambda_i/\lambda_1)^k \boldsymbol{v}_i$$

and

$$\begin{aligned}\mu_k &= \frac{\boldsymbol{x}_k^T A \boldsymbol{x}_k}{\boldsymbol{x}_k^T \boldsymbol{x}_k} \\ &= \frac{\lambda^{2k+1}\sum_{i=1}^n a_i^2(\lambda_i/\lambda_1)^{2k+1}}{\lambda^{2k}\sum_{i=1}^n a_i^2(\lambda_i/\lambda_1)^{2k}} \\ &= \lambda_1 \frac{\sum_{i=1}^n a_i^2(\lambda_i/\lambda_1)^{2k+1}}{\sum_{i=1}^n a_i^2(\lambda_i/\lambda_1)^{2k}}\end{aligned}$$

Note that as $k \to \infty$,

$$\sum_{i=1}^n a_i^2(\lambda_i/\lambda_1)^{2k} \to a_1^2$$

since $\lambda_i/\lambda_1 < 1$ for $i = 2, \cdots, n$ and hence $(\lambda_i/\lambda_1)^{2k} \to 0$ as $k \to \infty$ for $i = 2, \cdots, n$. Thus $\mu_k \to \lambda_1$ as $k \to \infty$.

(b) If the maximum eigenvalue is not unique (that is, $\lambda_1 = \lambda_2 = \cdots = \lambda_\ell$) then from the proof in part (a), we have

$$\sum_{i=1}^n a_i^2(\lambda_i/\lambda_1)^{2k} \to a_1^2 + \cdots + a_\ell^2$$

as $k \to \infty$ and so μ_k still converges to the maximum eigenvalue.

6. (a) Since B has its eigenvalues in the interval $(-1, 1)$, we can write

$$A^{-1} = (I - B)^{-1} = \sum_{k=0}^\infty B^k$$

and so

$$\text{tr}(A^{-1}) = \sum_{k=0}^\infty \text{tr}(B^k).$$

(b) We want to write $A = \alpha(I - B)$ so that $B = I - \alpha^{-1}A$. If $\lambda_1, \cdots, \lambda_n$ are the eigenvalues of A then $1 - \lambda_1/\alpha, \cdots, 1 - \lambda_n/\alpha$ are the eigenvalues of B. To apply the formula for $\text{tr}(A^{-1})$, we require the eigenvalues of B to lie in the interval $(-1, 1)$:

$$-1 < 1 - \lambda_i/\alpha < 1 \;\text{ or }\; 0 < \lambda_i/\alpha < 2 \;\text{ for all } i.$$

If μ is an upper bound for $\lambda_1, \cdots, \lambda_n$, it follows that we can take $\alpha > \mu/2$.

(c) We know that $0 < \lambda_1, \cdots, \lambda_n \le \|A\|_\infty$. By definition, $\|A\|_\infty = \sup \|A\boldsymbol{v}\|_\infty$ where the supremum is taken over vectors $\boldsymbol{v}$ with $\|\boldsymbol{v}\|_\infty = 1$. Let $\boldsymbol{a}_i$ be the i-th row of A and note that for $\|\boldsymbol{v}\|_\infty = 1$,

$$|\boldsymbol{a}_i \boldsymbol{v}| \le \sum_{j=1}^n |a_{ij}|$$

with equality if $v_j = \text{sign}(a_{ij})$. From this, it follows that

$$\|A\|_\infty = \max_{1 \le i \le n} \sum_{j=1}^n |a_{ij}|.$$