Midterm Exam

36-350, Statistical Computing, Fall 2012

12 October 2012

INSTRUCTIONS: Provide all answers in your bluebook. Only work in the bluebook will be graded. Clearly indicate which problem each answer goes with.

There are 100 points in all. There are twelve questions; you have gotten a randomly-chosen set of six. Questions have been matched across tests to make sure everyone has an equally hard exam.

The exam will be curved.

Explain your reasoning when possible, even for the multiple choice questions; this will help us to give partial credit.

1. (50) Suppose that we have a function `gamma.mle`, which takes in a data vector and returns the maximum likelihood estimate of a gamma distribution fitted to that data, including both the parameters, and the value of the log-likelihood at the maximum.

```
gamma_ml_dist <- function(x,B=100) {                                  # 1
  mle.fit <- gamma.mle(x)                                             # 2
  mle.ll <- mle.fit$loglik                                            # 3
  params <- mle.fit$params                                            # 4
  boot.logliks <- vector(length = B) {                                # 5
  for (b in 1:B) {                                                    # 6
    random.draws <- vector(length=length(x)) {                        # 7
    for (i in seq(along=x)) {                                         # 8
      random.draws[i] <- rgamma(1,shape=params[1],scale=params[2])    # 9
    }                                                                 # 10
    boot.logliks[b] <- gamma.mle(random.draws)$params                 # 11
  }                                                                   # 12
  p <- mean(boot.logliks <= mle.ll)                                   # 13
  return(p)                                                           # 14
}                                                                     # 15
```

   (a) (15) Explain, in words, what this is supposed to do.

   (b) (10) There is a bug. Find it, explain why it's wrong, and explain how to fix it. (You do not need to say exactly what the buggy output will be.)

   (c) (10) Replace the inner `for` loop with a single, non-iterative line. (You will get more partial credit the closer you can get to one line.)

   (d) (15) Replace both the inner and the outer `for` loop with at most two lines, with no iteration.

2. (50) You will recall from 36-202 that **logistic** regression is technique which works rather like linear regression, but is used when the dependent variable is a binary category, rather than a continuous real number. It is implemented in R through the function

```
glm(formula, family=binomial(link="logit"), data)
```

Like `lm`, this returns a complicated object containing estimated coefficients, standard errors, residuals, etc.

Now consider the following code.

```
income.coefs <- vector(length=nlevels(voter_survey$state))                # 1
for (state in 1:nlevels(voter_survey$state)) {                            # 2
  this_state <- NULL                                                      # 3
  for (voter in 1:nrow(voter_survey)) {                                   # 4
    if (voter_survey$state[voter] == levels(voter_survey$state)[state]) { # 5
      if (is.null(this_state)) {                                          # 6
        this_state <- as.data.frame(voter_survey[voter,])                 # 7
      } else {                                                            # 8
        this_state <- cbind(this_state, voter_survey[voter,])             # 9
      }                                                                   # 10
    }                                                                     # 11
  }                                                                       # 12
  republican_vs_income <- glm(voted_republican ~ income + race,           # 13
    data=this_state, family=binomial(link="logit"))                       # 14
  income.coefs[state] <- coefficients(republican_vs_income)["income"]     # 15
}                                                                         # 16
plot(states$mean_income, income.coefs, xlab="State's per-capita income",  # 17
  ylab="Coefficient for voting Republican as income inceases")            # 18
```

(a) (15) Explain, in words, what this is supposed to do.

(b) (10) The code, as written, contains a bug. Find it, explain why it's wrong, and explain how to fix it. (You do not need to say exactly what the buggy output will be.)

(c) (15) How would you replace the inner, but not the outer, `for` loop with at most two lines of code, containing no iteration?

(d) (10) How would you replace both `for` loops with at most six lines of code, containing no iteration?

3. (10) Consider the following code:

```
sortifnot <- function(x) {
  sorted <- sort(x)
  if ((x = sorted)) {
    return(TRUE)
  } else {
    return(sorted)
  }
}
```

What is the desired output for the following?

```
sortifnot(c(3,1,2,Inf))
```

(a) `TRUE`

(b) `FALSE`

(c) The vector `1 2 3 Inf`

(d) The vector `Inf 3 2 1`

(e) The vector `3 1 2 Inf`

4. (10) Refer to the code in Question 3. What is the *actual* output for the following?

```
sortifnot(c(3,1,2,Inf))
```

   (a) `TRUE`

   (b) `FALSE`

   (c) The vector `1 2 3 Inf`

   (d) `TRUE` with a warning message about "the condition has length > 1"

   (e) Trying to define the function will produce an error

5. (10) Refer to the code in Question 3. Which of the following will fix the function?

   (a) The function works as is.

   (b) Replace `if((x = sorted))` with `if(x <- sorted)`

   (c) Replace `if((x = sorted))` with `if(identical(x,sorted))`

   (d) Replace `if((x = sorted))` with `if(x == sorted)`

   (e) Exchange `return(TRUE)` and `return(sorted)`

6. (10) Consider the following code:

```
permute_test <- function(x,y,assoc=cor,B=100) {                    # 1
  stopifnot(length(x)==length(y))                                  # 2
  observed.association <- assoc(x,y)                               # 3
  permuted.associations <- replicate(B, assoc(sample(x),y))        # 4
  pv <- mean((observed.association = permuted.associations))       # 5
  return(pv)                                                       # 6
}                                                                  # 7
```

What is this supposed to do?

(a) Test whether the association between vectors `x` and `y` is significant, by randomly permuting `x`, re-calculating the association for the shuffled data, and returning `TRUE` or `FALSE` depending on whether the actual association is stronger than `B`% of the shuffled values.

(b) Find the sampling distribution for the association between `x` and `y` under the null hypothesis of no association, by randomly permuting `x` `B` times, re-calculating the association each time, then returning at what quantile of this distribution the observed association lies.

(c) Finding the bias of `assoc` as an estimator of the association between `x` and `y`, by making `B` draws from the sampling distribution of `x`, calculating the association for each draw, and then returning the average difference between the observed association and each draw.

(d) Finding a predictive variance for the association between `x` and `y`, by randomly perturbing `x` `B` times, re-calculating the association, and taking the mean dispersion of these associations around the observed association; this gives a margin of error for how much the sample association should change on new data.

(e) The code is meaningless pseudo-statistical rubbish, designed to look impressive and fool the easily intimidated.

7. (10) Consider the code in Problem 6. What will happen when I try to run the following?

```
z <- rnorm(100,mean=12,sd=10)
permute_test(z,z,B=1e4)
```

   (a) It depends on what `assoc` has been defined to be in the global environment (if anything).

   (b) The number 1.0 exactly.

   (c) A random, but strictly positive, fraction close to 0.

   (d) A random number of mean 0.

   (e) A random number, approximately 6/5.

8. (10) Consider the code in Problem 6. What will fix the code?

   (a) It does not need to be fixed.

   (b) It cannot be fixed because it makes no sense.

   (c) Define an `assoc` function before running this.

   (d) `replicate` is being used inappropriately, and must be replaced with a `for` loop or `apply`.

   (e) Change `=` to `>=` in line 5.

9. (10) Consider the following code, which tries to determine whether the majority of the volume of a sphere is located within a thin shell at the surface.

```
surface.dominated <- function(r,dim,shell=0.01) {                # 1
  stopifnot(r>0, r-shell>0,dim>0,dim==round(dim))                 # 2
  total.volume <- d.sphere.volume(r,dim)                         # 3
  excluded.volume <- d.sphere.volume(r-shell,dim)                # 4
  return(excluded.volume/total.volume < 0.5)                     # 5
}                                                                # 6
# Calculate the volume of a sphere in d dimensions               # 7
d.sphere.volume <- function(r,dim) {                             # 8
  return(pi^(dim/2) * r^dim / gamma(1+dim/2))                    # 9
}                                                                # 10
```

What has to be done to be able to evaluate whether 300-dimensional sphere of radius 10 has most of its volume in a shell of thickness 0.005?

(a) Modify the definition of `surface.dominated` to bring the function `d.sphere.volume` inside it; otherwise, it cannot find the function.

(b) Re-define `surface.dominated` with `shell=0.05`; otherwise, it will use a shell thickness of 0.01.

(c) Run `surface.dominated(10,300,0.005)`

(d) Both (a) and (b) are necessary.

(e) Something else.

10. (10) Consider the following code, which tries to be a (very simple) version of *stochastic gradient descent*. This works like gradient descent, but at each stage it only evaluates the model on a single randomly-chosen data point, because working with the full data would take too long.

```
stoch.grad.descent <- function(f,x,df,max.iter=1e6,rate=1e-6) {      # 1
  stopifnot(require(numDeriv))                                       # 2
  for (t in 1:max.iter) {                                            # 3
    g <- stoch.grad(f,x,df)                                          # 4
    x <- x - (rate/t)*g                                              # 5
  }                                                                  # 6
  return(x)                                                          # 7
}                                                                    # 8
stoch.grad <- function(f,x,df) {                                     # 9
  i <- sample(1:nrow(df),size=1)                                     # 10
  noisy.f <- function(x) { return(f(x, data=df[i,])) }               # 11
  stoch.grad <- grad(noisy.f,x)                                      # 12
  return(stoch.grad)                                                 # 13
}                                                                    # 14
```

Suppose that the `mse` function has been set up to take a parameter vector called `x` and a data frame called `data`, and return a single real number. Suppose also that `old.est` is an old estimate of the model's parameters, and `sales2012` is a data frame of this year's data. What must be done to run stochastic gradient descent, starting from the old estimate, for ten million steps?

(a) Run `stoch.grad.descent(mse,old.est,sales2012,1e7)`.

(b) The definition of `stoch.grad` must be moved inside `stoch.grad.descent`, otherwise the latter won't be able to find the function.

(c) Re-define `stoch.grad.descent` with `max.iter=1e7`, otherwise it will do only `1e6` = one million steps.

(d) Both (b) and (c).

(e) Both (b), and moving the `require(numDeriv)` command inside the definition of `stoch.grad`.

11. (10)

```
lower <- 1
upper <- 2
bracketing.doublings <- function(x) {
 while (upper < x) {
   lower <- upper
   upper <- 2*upper
 }
 return(c(lower,upper))
}
sapply(c(37,193,45),bracketing.doublings)
```

What comes next?

(a)
```
     [,1] [,2] [,3]
[1,]   32  128   32
[2,]   64  256   64
```

(b)
```
Error:  object 'upper' not found
```

(c)
```
Error:  object 'lower' not found
```

(d)
```
     [,1] [,2] [,3]
[1,]   32  128  128
[2,]   64  256  256
```

(e)
```
     [,1] [,2] [,3]
[1,]   32  128  256
[2,]   64  256  512
```

12. (10)

```
x <- c(-1,1)
f <- function(x) {
  y <- mean(x)
  sd <- function(z) { return((z-y)^2) }
  return(sd)
}
f(x)
sd(x)
```

What comes next?

(a) `1 1`, because that is $(-1-0)^2$ and $(1-0)^2$.

(b) `1.414214`, because that is $\sqrt{(-1-0)^2+(1-0)^2}$.

(c) `1`, because that is $\sqrt{\frac{(-1-0)^2+(1-0)^2}{2}}$.

(d) `2`, because that is $(-1-0)^2+(1-0)^2$.

(e) Something else.