STAT:7400 Computer Intensive Statistics
Homework Assignments

Luke Tierney

Spring 2017

Assignment 1

Due on Monday, January 23, 2017.

This assignment will give you an opportunity to familiarize yourself with the computing environment you will be using this semester and to practice writing up computing assignments.

Before starting this assignment you should read the coding standards document[1] and the brief introduction[2] to the Git version management system.

You should track your work using a git repository (you can use the same repository for both problems). Your git repository should have at least 2 commits. When you are finished with the homework, run the command

```
git log -p > hw1-gitlog.txt
```

inside of your Git repository. This command writes the Git log to the text file `hw1-gitlog.txt`. Include this file in your submission.

1. The Pareto distribution has density

$$f(x|\alpha,\beta) = \frac{\beta\alpha^\beta}{x^{\beta+1}}$$

for $0 < \alpha < x$ and $\beta > 0$. Write a C function to evaluate the Pareto density. Also write a main program that reads values for the arguments x, α, and β, in that order, from standard input and prints the value of $f(x|\alpha,\beta)$ to standard output. I will test your program using commands of the form

```
echo 4.0 2.0 3.0 | ./paretodens
```

In your written report include the code in an appendix, explain how to compile and run the program, and show a few (two or three) examples of its use. Include the program files as text files with `.c` extensions in your submission as well.

2. Write an R function `dpareto` that computes the density of a Pareto distribution. Your function should behave like other R density functions, such as `dgamma` or `dbeta`.

Your submission should include a file `dpareto.R` suitable for reading into R with the `source` function. I will test your code using the expressions

[1] http://www.stat.uiowa.edu/ luke/classes/STAT7400/coding.html
[2] http://www.stat.uiowa.edu/ luke/classes/STAT7400/git.html

```
source("dpareto.R")
dpareto(x, a, b)
```

with various definitions of `x`, `a`, and `b`.

In your written report include the code for the function in line, show an example of its use, and show plots of the density for two different sets of parameters. Include the file `dpareto.R` in your submission as well.

Some points to keep in mind:

- Negative values of the parameters are not meaningful; how is this handled for other distributions?
- Densities are defined on the entire real line; they are zero outside of the support of the distribution.
- Make sure that your code is consistent with the coding standards.
- It is a good idea to compile your C code with all useful warnings enabled and to eliminate any warnings you find. For the `gcc` compiler using the options `-Wall -pedantic` enables the most useful warnings.
- Make sure the text and code of your writeup are easy to read. Use appropriate fonts and margins.
- R functions are generally vectorized, i.e. return vectors of results when the arguments are vectors. Your R function should do this as well.

You should submit your assignment electronically using Icon. Your submission should include

- your writeup as a PDF file
- one or more source code files containing your programs for Problem 1 and Problem 2.
- your Git log file.

Submit your work as a single compressed tar file. If your work is in a directory `mywork` then you can create a compressed tar file with the command

```
tar czf mywork.tar.gz mywork
```

Solutions and Comments

General Comments

- Make sure your name is on your writeup!
- Don't include things not asked for (editor temporary files like `foo ~`, `.git` subdirectories, executables, etc).
- Large margins waste a lot of space. Make sure to set margins appropriately.
- Make sure your code is readable in your writeup. Blank lines between every line of code make the code hard to read.
- Use appropriate fonts for text and code.
- Avoid using file names that contain spaces. Manu sools don't work well with such file names.
- If you are asked to submit your work in an archive you should include *everything* in the archive. Don't submit your writeup separately. Instructions may vary in later submissions, so be sure to read and follow instructions.
- Do not include screen shots unless you are showing a graphical user interface. For command line interfaces you should only include the text of the interaction.
- If you are asked to put code in a appendix, do so.
- if you are asked to include code in line, do.
- If you are asked to use a particular name for a file or a function you *must* must match that name *exactly*, *including capitalization*. Otherwise code that uses your file of function, or automated test run on them, will fail.
- R code files should usually only include code, not tests or examples (unless they are commented out in a useful way).

Problems

1.
   - Make sure your program compiles on the Linux systems at least, and check it again after you make changes, even small ones.
   - Densities are usually defined for all real arguments. If a distribution has limited support then the density is zero outside the support.

- It is a good idea to compile your C programs with as many warnings enabled as possible. With the `gcc` compiler on the Linux systems, using the flags `-Wall -pedantic` does this. If you want to use C99 features also add `-std=c99`. You should try to eliminate the warnings from your code. This will help finding errors and make it more likely that your code will work on other systems.
- Programs in any language other than machine language are intended to be read by humans. You should try to make your programs as readable as possible.
  - Proper indentation of looping and conditional structures makes programs more readable. There are even some languages, such as Python, where indentation is part of the language syntax! For C, the `emacs` editor can help you indent your code–hitting the TAB key will make emacs indent to the place it thinks appropriate. If the code is not being placed where you think it should go then you probably have a syntax error, such as a missing or extra parenthesis or a missing semicolon. You can also use the program `indent` to indent your C source code according to a reasonable style.
  - I have a strong prference for indenting by 4 spaces at each level.
  - Spaces around operators and after commas also help make your code more readable.
  - Avoid having tab characters in your code — your editor should be able to replace those with spaces.
  - Review the coding standards document[3] and try to follow them.
  - The `indent` program can help; a reasonable result is obtained with

    ```
    indent -kr -nce -nut foo.c -o foo-indent.c
    ```
- You should use variables of type `double` (double precision) for floating point calculations, not the single precision type `float`.
- If there are range restrictions on your parameters then you should check for those and do something to signal violations in some way. Do not just return whatever value the formula happens to compute — it will be invalid.
- The problem asked you to write a separate function to compute the Pareto density, not compute it in the main program. It is important to follow instructions like this if your code is to work properly with other code.
- When asked to show examples you should show the terminal interaction in a displayed paragraph in an appropriate font; do not describe what happens in words and do not show screen shots.
- Be sure to put your code in an appendix as requested.
- I used to following code to test your programs:

[3] http://www.stat.uiowa.edu/ luke/classes/STAT7400/coding/coding.html

```
echo 3 2 1  | ./paretodens   # 0.2222
echo 1 2 3  | ./paretodens   # 0.0
echo 3 -2 1 | ./paretodens   # error
echo 3 2 -1 | ./paretodens   # error
```

2.
- Densities are usually defined for all real arguments. If a distribution has limited support then the density is zero outside the support.
- Programs in any language other than machine language are intended to be read by humans. You should try to make your programs as readable as possible.
  - Proper indentation of looping and conditional structures makes programs more readable. There are even some languages, such as Python, where indentation is part of the language syntax! For R, the `emacs` editor with the ESS package can help you indent your code–hitting the TAB key will make emacs indent to the place it thinks appropriate. If the code is not being placed where you think it should go then you probably have a syntax error, such as a missing or extra parenthesis or a missing semicolon.
  - Spaces around operators and after commas also help make your code more readable.
  - Review the coding standards document[4] and try to follow them.
  - The `formatR` package can be useful, for example:

    ```
    Rscript -e 'formatR::tidy_source("foo.R")'
    ```
- If there are range restrictions on your parameters then you should check for those and do something to signal violations in some way. You can either return `NA`, and perhaps signal a warning, or signal an error by calling `stop`. Do not just return whatever value the formula happens to compute — it will be invalid. Also do not return an error message as a text string as this makes life much harder for code that uses your function.
- Functions like `dgamma` return `NaN` and signal a warning for invalid parameters. It is best to follow convention unless there is a compelling reason not to.
- You were explicitly asked to name your file `dpareto.R`. Doing this is important if other code, such as my test code, is going to look for a file with that name.
- Your file should only contain code defining the function, not test code. Test code can be placed in a different file.
- You should take advantage of vectorized arithmetic whenever possible. Loops and looping functions like `lapply` and friends will be much less efficient.

[4] http://www.stat.uiowa.edu/ luke/classes/STAT7400/coding/coding.html

- To match the behavior of other density functions your function should be vectorized. Your handling of bad parameter values and restrictions on the support of the distribution should also take vector input into account. The `ifelse` function can be useful for this.
- The standard density functions all support a `log` argument. Yours should do this as well.
- If you support the `log` argument, it is almost always better to compute on the log scale and return `exp` of the result than the other way around.
- When plotting a function, line plots for the continuous parts make more sense than point plots. Some thought is needed on how to handle discontinuities.
- When plotting two related densities it is a good idea to help the reader compare the results either by plotting two densities on the same plot or by placing two plots with identical axes next to each other.
- Be consistent in using `=` or `<-` for assignment; `<-` is strongly preferred.
- Make sure your result is returned visibly.
- Make sure to distinguish between the logical operators `&&` and `||` and the vectorized functions `&` and `|`.
- Calls to `return()` are only needed for early exit.
- Be sure to include the function code in line in your writeup.
- I used the following code to test your functions:

```
dpareto(3, 2, 1)     # 0.2222222
dpareto(1, 2, 3)     # 0.0
dpareto(3, -2, 1)    # error
dpareto(3, 2, -1)    # error
dpareto(3 : 5, 2, 1) # 0.2222222 0.1250000 0.0800000
dpareto(1 : 5, 2, 1) # 0.0 0.0 0.2222222 0.1250000 0.0800000
dpareto(6, 2 : 4, 1) # 0.05555556 0.08333333 0.11111111
dpareto(3, 2, 1, log = TRUE) # -1.504077
```

Assignment 2

Due on Monday, January 30, 2017.

1. Redo Problem 2 from Assignment 1. Improve your definition of the `dpareto` function for computing the Pareto density; in particular make sure that the function is properly vectorized, handles the range restriction and invalid parameter values properly, and that the optional `log` argument is supported. Also make sure that your graphs properly show the discontinuity of the densities.

2. Several mechanisms are available for calling C code from R. The simplest is the `.C` interface. Modify your C function for computing the Pareto density so it can be used with the `.C` interface, and include some examples of its use in your writeup.

   The *Writing R Extensions* manual provides documentation for the `.C` interface as well as other interfaces. A simple example is provided on the class web site that you may find helpful. The files in this example show how to use the `.C` interface as well as the richer `.Call` interface.

   Your submission should include the new C source file and a source file with the R code for calling your C function. Track your work using Git and include your Git log in your submission.

Be sure to follow the coding standards. Tools are available to help:

- For C code, the `indent` program on Linux/Mac systems; these have different control options on Linux and Mac.
- For R code, the `formatR` package, in particular the `tidy_source` function in that package.

You should submit your assignment electronically using Icon. Your submission should include

- your writeup as a PDF file
- files with your R code for Problem 1 and Problem 2
- a file with your C code for Problem 2
- the Git log for your work

Submit your work as a single compressed tar file. If your work is in a directory `mywork` then you can create a compressed tar file with the command

```
tar czf mywork.tar.gz mywork
```

Solutions and Comments

General comments:

- Don't include things not asked for (editor temporary files like `foo ~`, `.git` subdirectories, executables, shared libraries, etc).
- Make sure the file you upload is a `gzip`-compressed `tar` file.
- Avoid excessively large margins.

1.
   - Please follow the coding standards on use of spaces and indentation and avoiding long lines.
   - Calls to `return()` are only needed for early exit.
   - Make sure your `log = TRUE` code does works correctly.
   - Do not write error or warning messages with `cat` or return strings — use `stop` to signal errors and `warning` for warnings.
   - Use brief but informative error messages.
   - Make sure your code is not too complex and/or inefficient.
   - You should not use loops if vectorized operations can be used.
   - It is usually numerically better to compute the log density and exponentiate than the other way around.
   - One possible definition:

```
dpareto <- function(x, a, b, log = FALSE) {
   nx <- length(x)
   na <- length(a)
   nb <- length(b)
   n <- max(nx, na, nb)
   if (nx < n) x <- rep(x, length.out = n)
   if (na < n) a <- rep(a, length.out = n)
   if (nb < n) b <- rep(b, length.out = n)
   ld <- ifelse(a > 0 & b > 0,
                ifelse(x > a,
                       log(b) + b * log(a) - (b + 1) * log(x),
                       log(0)),
                NaN)
   if (log) ld
   else exp(ld)
}
```

   This could be simpler if the `ifelse` function did not base its result size entirely on the first argument.

- These are the tests I used:

```
dpareto(3,-2, 1)  # bad parameter
dpareto(3,2, -1)  # bar parameter
stopifnot(all.equal(dpareto(3,2,1), 0.2222222222))
stopifnot(all.equal(dpareto(1,2,3), 0.0))
stopifnot(all.equal(dpareto(3:5,2, 1), c(0.2222222222, 0.1250000, 0.0800000)))
stopifnot(all.equal(dpareto(1:5,2, 1), c(0.0, 0.0, 0.2222222222, 0.1250000, 0.0800000)))
stopifnot(all.equal(dpareto(6,2:4, 1), c(0.05555555556, 0.08333333333, 0.11111111111)))
stopifnot(all.equal(log(dpareto(1:5,2, 1)), dpareto(1:5,2, 1, log = TRUE)))
stopifnot(all.equal(dpareto(1:6,1:2, 1),
                    c(0.0, 0.0, 0.11111111111, 0.125, 0.04, 0.05555555556)))
stopifnot(all.equal(dpareto(1, 2, 1:2), c(0, 0)))
```

2. - Your C code should compile without errors or warnings. You can use the `PKG_CFLAGS` environment variable to set C compiler flags to enable all warnings; e.g. with

     ```
     env PKG_CFLAGS="-Wall -pedantic" R CMD SHLIB ...
     ```

   - It is not a good idea to allocate vectors that might be large as local variables. These will be allocated on the process stack and the space available for the stack is not very large.
   - Follow the coding guide on use of spaces, indentation, and long lines.
   - Standard convention in R density functions is to return `NaN` for bad parameter values.
   - Do not use `printf` for messages; use the C level warning/error calls.
   - It is usually numerically better to compute the log density and exponentiate than the other way around.
   - In C code you should only warn about bad parameter values once, not on every loop iteration.
   - You should not use loops in R if vectorized operations can be used.
   - I used the same test code as in the first problem.

Assignment 3

Due on Monday, February 6, 2017.

1. Two numerical approximations to the derivative of a function f at a point x are the forward difference quotient

$$\delta_F(f, x, h) = \frac{f(x+h) - f(x)}{h}$$

and the central, or symmetric, difference quotient

$$\delta_S(f, x, h) = \frac{f(x+h) - f(x-h)}{2h}$$

for a step size h. A third option that is available when the function f is analytic near x is

$$\delta_C(f, x, h) = \frac{\Im(f(x+hi))}{h}$$

where $i = \sqrt{-1}$ is the imaginary unit and $\Im(z)$ is the imaginary part of the complex number z.

Chose a few functions and argument values and examine these approximations graphically by plotting the approximations against $-\log_2 h$ for h values in the range $2^{-1}, \ldots, 2^{-64}$. Some functions and argument values you might consider:

$$\begin{aligned}
f_1(x) &= \sin(x) && \text{at } x = 1 \\
f_2(x) &= 10000 \sin(x) && \text{at } x = 1 \\
f_3(x) &= \tan(x) && \text{at } x = 1.59 \\
f_4(x) &= \phi(x) && \text{at } x = 0.5
\end{aligned}$$

where ϕ is the standard normal density. Comment on the behavior you see. Can you suggest a guideline for choosing the step size h?

2. Create an R package `pareto` that contains a function `dpareto` to compute the density of the Pareto distribution. Include an example in the help page and some test code in a `tests` directory. Your package should pass `R CMD check` without errors or warnings. Your writeup should contain a simple example of using your package, and you should include your package as a source package file created by `R CMD build` in your submission archive file.

   The *Writing R Extensions* manual provides documentation on creating R packages. The function `package.skeleton` may help you get started. There is also a small sample package available called `AddOne` that you can start with. You can unpack the package sources with the command

```
tar zxf AddOne_1.0-1.tar.gz
```

You can find further documentation, tutorials, and tools by searching the web, e.g. for "create R package."

Track your work on your package using Git and include your Git log in your submission.

You should submit your assignment electronically using Icon. Your submission should include

- your writeup as a PDF file
- a source code package as created by `R CMD build`.

Submit your work as a single compressed tar file. If your work is in a directory `mywork` then you can create a compressed tar file with the command

```
tar czf mywork.tar.gz mywork
```

In addition, you should commit your source code to your UI GitHub repository in a directory named `pareto`. After your commit your repository should look like

```
<your repo>/
    README.md
    pareto/
        DESCRIPTION
        NAMESPACE
        README
        man/
            ...
        R/
            ...
        tests/
            ...
```

Solutions and Comments

General comments:

- Don't include things not asked for (editor temporary files like `foo ~`, `.git` subdirectories, executables, shared libraries, etc).
- Use appropriate fonts for text and code in your writeup.
- Do not leave excess spaces between code lines.
- Don't use margins that are too large.
- Please use an 11 or 12 point font.

1.
   - If you use separate plots for symmetric and forward differences then you should use common axes.
   - Numerical derivatives are often used in optimization.
   - It is important to choose a step size that is not too large or too small. This balances the *truncation error* (h too large) against the *round-off error* (h too small).
   - Central differences can use larger step sizes but require more function evaluations.
   - Simple calculus can help understand why central differences will be more accurate than forward differences for a given small h value.
   - Complex differences avoid the round-off error, but require the algorithm computing the function to be able to handle complex arguments.
   - Dennis and Schnabel (1983) recommend for forward difference quotients

     $$h = \sqrt{\eta}\max\{x, t_x\}$$

     where η is the relative error in computing $f(x)$ and t_x is the typical size of x.
   - Using this rule, if $\eta = 10^D$ with D the number of accurate base 10 digits in $f(x)$ then the number of accurate digits in $\delta_F(f, x, h)$ is about $D/2$.
   - Their recommendation for central difference quotients is

     $$h = \sqrt[3]{\eta}\max\{x, t_x\}$$

     For $\eta = 10^D$ the number of accurate digits in the approximate derivative should be about $2D/3$.
   - Extrapolation methods can be useful.

2.
- Be sure to check your code into GitHub.
- It is usually best to only place source files under version control, not package tar balls or test results.
- You were asked to name your package `pareto`, not `Pareto`, or `dpareto`, or something else.
- Your package should pass `R CMD check` without errors, warnings or notes.
- It is usually better to explicitly export the public functions in your `NAMESPACE` file.
- Package tests:
  - Your should include test code in a `tests` directory.
  - Your tests should try to test all important cases.
  - Be careful about floating point equality tests.
- If you include a `README` file then its contents should be appropriate for a user of your package.
- Make sure your help file includes useful information.
- Please follow the coding standards on use of spaces, avoiding long lines, and proper indentation.
- Vectorization should work for `x`, `a`, and `b`.
- The tests I used:

```
stopifnot(is.na(dpareto(3,-2, 1)))
stopifnot(is.na(dpareto(3,2, -1)))
stopifnot(all.equal(dpareto(3,2,1), 0.2222222222))
stopifnot(all.equal(dpareto(1,2,3), 0.0))
stopifnot(all.equal(dpareto(3:5,2, 1),
                    c(0.2222222222, 0.1250000, 0.0800000)))
stopifnot(all.equal(dpareto(1:5,2, 1),
                    c(0.0, 0.0, 0.2222222222, 0.1250000, 0.0800000)))
stopifnot(all.equal(dpareto(6,2:4, 1),
                    c(0.05555555556, 0.08333333333, 0.11111111111)))
stopifnot(all.equal(log(dpareto(1:5,2, 1)),
                    dpareto(1:5,2, 1, log = TRUE)))
stopifnot(all.equal(dpareto(6,1,2:4),
                    c(0.0092592593, 0.0023148148, 0.0005144033)))
stopifnot(all.equal(dpareto(1:6,1:2, 1),
                    c(0.0, 0.0, 0.11111111111, 0.125, 0.04, 0.05555555556)))
stopifnot(all.equal(dpareto(1, 2, 1:2), c(0, 0)))
```

Assignment 4

Due on Monday, February 13, 2017.

1. The Longley data, available as the variable `longley` in the package `datasets`, provides a well-known example for a highly collinear regression, in particular in the regression of the `Employed` variable on the other six variables plus an intercept. Using this data set as an illustration, this problem explores the accuracy of the QR and Cholesky factorization approaches for fitting a regression.

   (a) Write an R function that uses the package `gmp` to compute the exact coefficients, rounded to the nearest double precision numbers, of a least squares fit of a vector y to the columns of a matrix X. You may find the functions `as.bigq`, `solve`, and `as.double` useful. Your function should allow the arguments to be either floating point or arbitrary precision rational numbers.

   (b) Write an R function that uses the Cholesky factorization of the cross product matrix to compute the coefficients of a least squares fit of a vector y to the columns of a matrix X. Your function should take an optional argument `center` with default `FALSE`; if `center` is `TRUE` then X should contain only non-constant columns and you should mean center the columns before forming the Cholesky factorization (the functions `sweep` and `apply` or `colMeans` may be useful). When centering is used the model includes an intercept, which should be estimated and included in the result.

   (c) Use these functions and the function `lm.fit`, which uses QR factorization, to compare the accuracy of coefficient estimates obtained by the QR and Cholesky approaches for the Longley data. Does it help to apply the Cholesky factorization to mean-centered data? For obtaining the exact coefficients, keep in mind that the `longley` data set is a floating point approximation to the actual decimal data. Looking at the printed representation should show you how to convert this approximation to the exact values as rationals.

2. A random vector Y has a multivariate normal distribution with mean zero and

covariance matrix

$$C = \begin{bmatrix} 1 & a & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ a & 1 & a & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & a & 1 & a & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & a & 1 & a & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \dots & a & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \dots & 1 & a & 0 \\ 0 & 0 & 0 & 0 & 0 & \dots & a & 1 & a \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & a & 1 \end{bmatrix}$$

The log-likelihood for an observed vector y, dropping additive constants, is

$$-\frac{1}{2}\log\det C - \frac{1}{2}y^T C^{-1} y$$

(a) Write an R function to compute the log likelihood for a vector `y` and a scalar `a` using dense matrix methods. The functions `chol` and `backsolve` may be useful. You may also find it useful to use *matrix indexing*; for example for a square matrix `M` the expression

```
M[cbind(2 : nrow(M), 1 : (nrow(M) - 1))]
```

extracts the sub-diagonal of `M`.

(b) Rewrite your function to use sparse matrix methods provided by the `Matrix` package to take advantage of the sparseness of the covariance matrix. The functions `bandSparse`, `solve`, and `chol` may be useful.

(c) Compare the performance of your two functions on data vectors of different lengths. The `system.time` function may be useful. You should see significantly better performance of the sparse matrix approach or a large matrix.

Your submission should include a pdf file with your writeup and text files with `.R` extensions with your code for each problem.

You should submit your assignment electronically using Icon. Submit your work as a single compressed tar file. If your work is in a directory `mywork` then you can create a compressed tar file with the command

```
tar czf mywork.tar.gz mywork
```

Solutions and Comments

1. (a) Here is a simple function for computing the exact regression coefficients that assumes `X` is a matrix and `y` a vector:

```
exact.fit <- function(X, y) {
    bY <- as.bigq(y)
    bX <- as.bigq(X)
    bXt <- t(bX)
    as.double(solve(bXt %*% bX, bXt %*% bY))
}
```

   (b) A function to compute the regression coefficients by Cholesky factorization, with optional mean centering, is

```
chol.fit <- function(X, y, center = FALSE) {
    if (center) {
        cm <- colMeans(X)
        Xm <- sweep(X, 2, cm)
        beta <- chol.fit(Xm, y - mean(y))
        c(mean(y) - crossprod(cm, beta), beta)
    }
    else {
        R <- chol(t(X) %*% X)
        z <- t(X) %*% as.vector(y)
        as.vector(backsolve(R, forwardsolve(t(R), z)))
    }
}
```

   - This uses `forwardsolve` and `backsolve` to solve the equations. Using `solve` is less efficient and less accurate.

   (c)
   - Looking at the printed version of the `longley` data frame suggests that the original data were specified in decimal form with at most three digits after the decimal point.
   - Many decimal fractions are not exactly representable as binary fractions, so the floating point values in the data frame `longley` are *not* exactly equal to the values from the original paper.
   - The expressions

```
> X <- cbind(1, as.matrix(longley[, -7]))
> y <- longley[,7]
> XE <- as.bigq(round(1000 * X)) / as.bigq(1000)
> yE <- as.bigq(round(1000 * y)) / as.bigq(1000)
```

     produce in the variables `XE` and `yE` exact rational representations of the exact decimal data.

- The relative differences between the coefficients for the exact fit to the binary values in the `longley` frame and the fit to these data reflect the effect of decimal to binary rounding in the binary data:

```
> (exact.fit(XE, yE) - exact.fit(X, y)) / exact.fit(XE, yE)
[1] -7.835386e-16 -2.833260e-14 -8.911123e-15 -8.586763e-16  4.029450e-15
[6]  6.354484e-14 -6.069607e-16
```

- *Printed* versions of the coefficients, using the default settings of the number of digits to print, are almost the same:

```
> exact.fit(XE,yE)
[1] -3.482259e+03  1.506187e-02 -3.581918e-02 -2.020230e-02 -1.033227e-02
[6] -5.110411e-02  1.829151e+00
> chol.fit(X,y)
[1] -3.482259e+03  1.506187e-02 -3.581918e-02 -2.020230e-02 -1.033227e-02
[6] -5.110411e-02  1.829151e+00
```

  This is not surprising since computations are done in double precision. To see the differences you need to compute errors or relative errors:

```
> exact <- exact.fit(XE, yE)
> (chol.fit(X,y) - exact) / exact
[1] -1.022182e-08 -2.124171e-08 -2.732136e-08 -7.397719e-09 -4.537949e-09
[6]  4.874051e-08 -9.988415e-09
> (as.vector(lm.fit(X,y)$coef) - exact) / exact
[1] -1.697667e-15 -3.984991e-14 -2.712081e-15 -8.586763e-16 -3.357875e-15
[6] -2.783481e-14 -1.699490e-15
```

  The errors for `lm.fit` are very close in magnitude to the machine unit. The errors for the Cholesky approach are substantially larger. Mean centering improves the Cholesky result but they remain worse than the results obtained using the QR approach.

```
> (chol.fit(X[, -1], y, TRUE) - exact) / exact
[1] -1.120460e-13 -9.147053e-13 -3.843406e-13 -1.009803e-13 -5.490126e-14
[6]  1.166482e-12 -1.086460e-13
```

- The approaches using Cholesky factorization are less accurate because of the loss in accuracy in forming the cross product matrix. Mean centering improves this, but some accuracy is still lost.

2. A function to create the dense covariance matrix is

```
mkC0 <- function(n, a) {
  m <- diag(rep(1, n))
  m[cbind(2 : n, 1 : (n - 1))] <- a
  m[cbind(1 : (n - 1), 2 : n)] <- a
  m
}
```

A simple implementation of the log likelihood function that closely follows the mathematical definition is

```
llik0 <- function(y, a) {
  C <- mkC0(length(y), a)
  -0.5 * log(det(C)) - 0.5 * t(y) %*% solve(C) %*% y
}
```

This has several issues:

- It is almost always a bad idea to compute an inverse; it is more work than needed if you only wnat to solve one system of equations, and it will decrease numerical accuracy. It is usually better to compute and use an appropriate decomposition.
- It is almost always a bad idea to compute a determinant and then take its logarithm.
- If a matrix decomposition is already available then this can be used to compute the log determinant efficiently and accurately.
- This function returns a 1×1 matrix rather than a simple vector of length 1.

The most natural decomposition to use for covariance matrices is the Cholesky decomposition. R functions for computing the Cholesky decomposition of a matrix C produce an upper triangular matrix R such that $C = R^T R$. In terms of this R the quadratic form in the log likelihood is

$$y^T C^{-1} y = y^T (R^T R)^{-1} y = y^T R^{-1} R^{-T} y = z^T z$$

where $z = R^{-T} y$, or z solves $R^T z = y$. Furthermore, $\frac{1}{2} \log(\det(C))$ is equal to the sum of the logarithms of the diagonal elements of R. This leads to the definition

```
llikD <- function(y, a) {
  C <- mkC0(length(y), a)
  R <- chol(C)
  z <- forwardsolve(t(R), y)
  - sum(log(diag(R))) -0.5 * sum(z ^ 2)
}
```

A small improvement is to use the optional arguments to `forwardsolve` to avoid the transpose:

```
llikD <- function(y, a) {
  C <- mkC0(length(y), a)
  R <- chol(C)
  z <- forwardsolve(R, y, upper.tri = TRUE, transpose = TRUE)
  - sum(log(diag(R))) -0.5 * sum(z ^ 2)
}
```

Using the Matrix package a sparse representation of the covariance matrix can created by

```
library(Matrix)

mkC <- function(n, a) {
   d0 <- rep(1, n)
   d1 <- rep(a, n - 1)
   bandSparse(n, k = 0 : 1, diag = list(d0, d1), symm = TRUE)
}
```

The Cholesky factorization of the sparse covariance matrix is also sparse, and the sparse matrix method for the `solve` generic function will take advantage of this and compute the solution z to $R^T z = y$ efficiently. This leads to the definition

```
llikS <- function(y, a) {
  C <- mkC(length(y), a)
  R <- chol(C)
  z <- solve(t(R), y)
  - sum(log(diag(R))) -0.5 * sum(z ^ 2)
}
```

I do not see an obvious way to avoid the transpose.

The overhead associated with the sparse matrix support in the Matrix package is significant; as a result the dense matrix approach is faster for data vectors with less than about 150 observations. But the computational complexity of the dense matrix approach is $O(n^3)$ whereas the sparse matrix approach is $O(n)$; so the sparse matrix approach is much faster for larger vectors, and also uses much less memory.

Some notes:

- `sum(x ^ 2)` is simpler than `t(x) %*% x` and produces a scalar rather than a 1×1 matrix.
- It is best to separate installing packages from scripts that might be run many times.

- You should properly present the timing results with supporting tables and graphs and explanatory text.
- `system.time` is not very accurate for computations taking less than a second.
  - You can replicate the computations in a loop to get more accurate results.
  - The `microbenchmark` package provides a more sophisticated structure for timing experiments.
- Make sure your functions take the arguments requested in the problem.
- Make sure you provide your code in a text file with a `.R` extension.
- Don't create a dense matrix and them convert to a sparse one — that takes $O(n^2)$ time and space. Use `bandSparse` to create the sparse matrix directly, which is $O(n)$ is time and space.
- Once you have the Cholesky factorization you do not need the inverse.
- Once you have the Cholesky factorization you can compute the log determinant you need as the sum of the logarithms of the diagonals. Do *not* compute the product of the diagonals and then the logarithm!
- You do *not* want to compute the inverse of the sparse matrix or its Cholesky factor: they are not sparse!
- Using `forwardsolve` with a sparse matrix silently converts the sparse matrix to a dense matrix.
- Speed of likelihood calculations is important since they are needed many times in iterative computations of maximum likelihood estimates.
- Space efficiency is also important if you want to be able to handle large data sets.
- No matter how fast or space-efficient your function is, it is of no use if it gets the wrong answer!

Assignment 5

Due on Monday, February 20, 2017.

1. Revise your R package `pareto` to include functions `ppareto` to compute the CDF and `qpareto` to compute the quantile function, or inverse CDF, for the Pareto distribution. Use the corresponding functions for the Gamma distribution as a guide. Be sure to document your functions and to include test code.

2. Revise your R package `pareto` to include a short *vignette* describing the package. Section 1.4 of the Writing R Extensions manual describes vignettes briefly. You can read more about these ideas in two short articles in the R News newsletter about the underlying Sweave technology. There is also a manual for Sweave available:

   - Friedrich Leisch, Sweave, Part I: Mixing R and LaTeX, *R News* 2 (3), December, 2002.
   - Friedrich Leisch, Sweave, Part II: Package Vignettes, *R News* 3 (2), October, 2003.
   - Friedrich Leisch, Sweave User Manual, 2003.

   The `AddOne` package has been modified to include a short vignette.

   Writing a vignette involves using LaTeX, but the amount of LaTeX needed for a simple vignette is minimal—you should be able to figure out what is needed from these references and the example in the `AddOne` package.

   Your vignette should include both a graph and a table.

   If you prefer, you can use `knitr` to create your vignette. If you do use `knitr`, then in your `DESCRIPTION` file add `knitr` to the `Suggests:` entry, and add an entry entry

   ```
   VignetteBuilder: knitr
   ```

Your package should pass `R CMD check` without errors, warnings, or notes.

Your submission should consist of a source archive for your package created by `R CMD build`; a separate writeup is not needed.

Please submit your package to Icon in a compressed `tar` archive created with

```
tar czf mywork.tar.gz pareto_xyz.tar.gz
```

In addition, you should commit your revised package source code to your UI GitHub repository in the directory `pareto`.

Solutions and Comments

1.
   - Please follow the coding standards on use of spaces, indentation, and long lines.
   - Please indent by 4 spaces for each level.
   - Your package should pass `R CMD check` without errors, warnings or notes.
   - Your package should contain test code.
   - Make sure your test code is useful.
   - You should try to be as numerically accurate as possible in the tails.
   - The tests I ran are available in

     `http://www.stat.uiowa.edu/~luke/classes/STAT7400/HW5tests.R`

2.
   - Be sure you know how to include graphs and tables in Sweave or knitr documents.
   - Try to avoid excessive margins in LaTeXdocuments.
   - Make sure you write in reasonable sentences and paragraphs.
   - Make sure your plots have reasonable captions and /or legends.

Some notes:

- Sweave is a tool for *literate data analysis*.
- Literate data analysis is motivated by the notion of *literate programming* started by Donald Knuth with his book on the implementation of TeX.
- Literate data analysis is one way to support *reproducible research*.
- Other tools are available for other data analysis frameworks and other output formats.
  - An alternative for R is the `knitr` package.
  - `knitr` supports `markdown` as well as LaTeX.
- R now supports alternatives to Sweave to be used for generating vignettes; the `knitr` package is sometimes used.

Assignment 6

Due on Monday, February 27 2017.

1. One of the characteristics of leukemia is an excess of white blood cells. The white blood cell count at diagnosis can be used to aid in predicting a patient's survival time after diagnosis, with high white blood cell counts indicating a low expected survival time. Feigl and Zelen (Biometrics, 1965) show survival times in weeks and white blood cell counts (WBC) at diagnosis for 33 patients who died of acute leukemia. The patients were classified as AG positive or AG negative depending on the presence or absence of certain characteristics in the white blood cells. Table 1 shows data for AG positive and AG negative patients. The data set is also available online or as `leuk` in the `MASS` package.

WBC	Survival Time	AG	WBC	Survival Time	AG
2300	65	1	4400	56	0
750	156	1	3000	65	0
4300	100	1	4000	17	0
2600	134	1	1500	7	0
6000	16	1	9000	16	0
10500	108	1	5300	22	0
10000	121	1	10000	3	0
17000	4	1	19000	4	0
5400	39	1	27000	2	0
7000	143	1	28000	3	0
9400	56	1	31000	8	0
32000	26	1	26000	4	0
35000	22	1	21000	3	0
100000	1	1	79000	30	0
100000	1	1	100000	4	0
52000	5	1	100000	43	0
100000	65	1			

Table 1: Feigl and Zelen Leukemia Data

One possible model for the life times is a Weibull distribution with density

$$f(t_i|\alpha_i, \gamma) = \frac{\gamma}{\alpha_i}\left(\frac{t_i}{\alpha_i}\right)^{\gamma-1} \exp\{-(t_i/\alpha_i)^\gamma\}$$

with $\log \alpha_i = \beta_0 + \beta_1 x_i + \beta_2 u_i$, where

$$x_i = \log(\text{WBC}_i/10000)$$

and $u_i = 1$ if the patient is AG positive and $u_i = 0$ if the patient is AG negative.

(a) Show that the log likelihood for this model, after dropping additive terms constant in the parameters, can be written as

$$\ell(\beta_0, \beta_1, \beta_2, \delta) = -n \log \delta + \sum (z_i - e^{z_i})$$

with $\delta = 1/\gamma$ and $z_i = (\log t_i - \beta_0 - \beta_1 x_i - \beta_2 u_i)/\delta$.

(b) Find the maximum likelihood estimates of the β_i and δ, along with approximate standard errors. You may find the R functions `nlm`, `optim`, and `nlminb` useful. The `hessian` argument to these functions may be useful for computing standard errors.

2. This problem examines an approach to improving numerical approximations known as *Richardson extrapolation* and applies it to numerical differentiation. Suppose $g(h)$ is an approximation to a quantity g such that

$$g(h) = g + h^n g_n + O(h^{n+m})$$

as $h \to 0$ for some values of $n > 0$ and $m > 0$. Computing the approximation again with h replaced by h/t (common choices are $t = 2$ or $t = 10$) produces

$$g(h/t) = g + (h/t)^n g_n + O((h/t)^{n+m}).$$

The difference is

$$g(h/t) - g(h) = (1 - t^n)(h/t)^n g_n + O(h^{n+m})$$

as $h \to 0$. This can be used to approximate the leading error term in $g(h/t)$ and to produce the *extrapolated* approximation

$$r(h) = g(h/t) + \frac{g(h/t) - g(h)}{t^n - 1}.$$

This approximation satisfies $r(h) = g + O(h^{n+m})$ as $h \to 0$ and thus has a smaller order error term.

Use this approach to extrapolate the forward and central difference quotient approximations to the derivative and assess the results graphically for a range of h values and several t values using the functions you considered in a previous assignment.

Your submission should include a writeup as a PDF file.

Please submit your writeup to Icon in a compressed `tar` archive created with

```
tar czf mywork.tar.gz writeup.pdf
```

[Your can replace `writeup` and `mywork` with whatever names you like.]

Solutions and Comments

1. Some notes:

   - The log likelihood given in part (a) of the problem drops additive terms constant in the parameters.
   - Most optimization methods need initial estimates. These can often be obtained from plots of the data, such as Figure 1 or by fitting a simpler model.

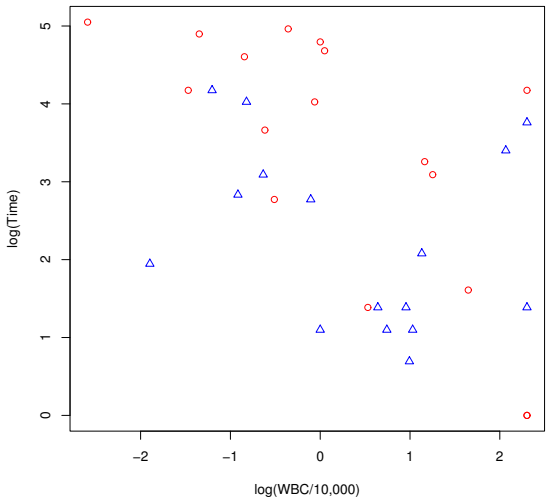


Figure 1: Feigl and Zelen leukemia survival data. Red points are AG positive, blue points are AG negative.

   - On the log time scale the Weibull model is linear with constant variance errors, so a simple least squares fit can be used for initial values. The errors do not have mean zero, so the constant term would need adjusting. An initial value for the scale parameter can be obtained from the OLS estimated standard deviation and the standard deviation of the extreme value distribution. The least squares summary is

```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   2.4995     0.3195   7.824  9.9e-09
x            -0.5709     0.1647  -3.467  0.00161
u             0.9883     0.4361   2.266  0.03081

Residual standard error: 1.249 on 30 degrees of freedom
```

The results obtained by `optim` are

	β_0	β_1	β_2	δ
Estimate	2.9945	-0.31024	1.020125	1.04066
Stand. Err	0.2839	0.13135	0.378119	0.14486

Make sure you report SE's, not variances.

- The log-likelihood is concave in β for a given δ and unimodal in δ for a given β, so it is fairly well behaved and very good initial values are not needed. If very poor initial values are used then the problem may be poorly scaled and default convergence criteria may need some help, for example by specifying scaling information.
- In general, if an interior optimum is expected it is a good idea to transform parameters in optimization problems so they are not constrained. In this case this suggests using $\log \delta$ in the optimization; the estimates and standard errors can then be adjusted to the original scale via invariance and the delta method, respectively. In this case the problem is sufficiently well behaved that this is not needed except for very poor starting values.
- It is a good idea to compare more complex models to simpler ones, to try different starting values, to experiment with different convergence criteria, and to make sure the results make sense.
- With `optim` you either need to minimize the negative log likelihood or use the `control` argument to get `optim` to maximize the function.
- Other methods of estimating standard errors are available, such as bootstrapping, but different bootstrapping methods may be estimating different things.

2.
- To use Richardson extrapolation you need to determine that the approximation has the required behavior and the appropriate value of n, which is $n = 1$ for forward and $n = 2$ for symmetric difference quotients. This is part of the problem and should be discussed in the writeup.
- Comparing the extrapolated and unextrapolated approximations in a single graph is helpful; separate graphs for assessing the truncation error and round-off error may help.
- The accuracy of the extrapolated forward difference approach is comparable to the accuracy of symmetric differences.

Assignment 7

Due on Monday, March 6, 2017.

1. Revise your R package `pareto` to compute the pareto density, CDF, and quantile function using C code. A version of the `AddOne` package that uses C code for its computations can serve as an example. An updated version makes more sophisticated use of C routine registration that may become mandatory for CRAN submissions soon.

   Your package should pass `R CMD check` without errors or warnings. You should submit your package as a source package file created by `R CMD build`.

   Also commit and push your revised package code to your class GitHub repository.

2. This problem is based on the BrainWeb artificial brain imaging data base, which is used for assessing the performance of brain image processing methods. The data are available on the workstations in

   `/group/statsoft/data/Brainweb/images`

   The motivating problem is classifying individual volume elements (or *voxels*) in an image into the three major brain tissue types (gray matter, white matter, and cerebro-spinal fluid) based on MR images. Several image types are available; this problem will look at a T1 image, the type most commonly used for tissue classification. The Wikibedia page

   `http://en.wikipedia.org/wiki/Magnetic_resonance_imaging`

   provides some useful background.

   Brain images are large 3D arrays and some care is used in storing them to reduce size and allow for efficient access to full images or slices of images. A number of structured file formats are in use, including Minc, Analyze, and Niftii. The data we will use is in a simple binary form that can easily be read using `readBin`. A simple function to do this is

```
readVol <- function(fname, dim = c(181, 217, 181)) {
    f<-gzfile(fname, open="rb")
    on.exit(close(f))
    b<-readBin(f,"integer",prod(dim),size = 1, signed = FALSE)
    array(b, dim)
}
```

The data have been scaled and rounded to an integer value between 0 and 255 and encoded as an unsigned byte.

The particular image we will use is stored in the file

```
/group/statsoft/data/Brainweb/images/T1/t1_icbm_normal_1mm_pn3_rf20.rawb.gz
```

In this problem we will ignore the spatial structure of the images and fit a normal mixture model to the image intensities. This model can then be used to classify the voxels into the three tissue types.

(a) Read the data and use the `image` function to plot a middle slice along each axis of the three-dimensional array.

(b) A first step in brain tissue classification is to identify the part of the image corresponding to the brain. This is often done by specialized pre-processing methods; a mask image, in the same format as the T1 image, with volume elements in the brain coded as one and elements outside the brain as zero is available in

```
/group/statsoft/data/Brainweb/images/mask.rawb.gz
```

Read in the mask and plot an estimate of the density of the image intensities for voxels in the brain. You should see three peaks; the lower one corresponds to cerebro-spinal fluid (CSF), the middle one to gray matter, and the higher one to white matter.

(c) Write a function to fit a normal mixture model by the EM algorithm and use the function to fit a model to the intensities within the brain. You can use the single step `EMmix1` functon from the class notes.

(d) Write a function that takes the intensities and the normal moxture parameters and returns for each voxel the index of the most likely tissue class for that voxel. The E step computations and the function `max.col` may be useful. Show the result using the `image` function for the same slices used in the first part of this problem.

(e) The EM calculation may be rather slow. Use profiling tools `Rprof` and `summaryRprof` to see where your code is spending most of its time.

(f) Can you improve peformance by taking advantage of the fact that the intensities can take on only a relatively small number of distinct values?

Your submission should include your source package archive for Problem 1 and a pdf file with your writeup, text files with a `.R` extension with your code for Problem 2.

You should submit your assignment electronically using Icon. Submit your work as a single compressed tar file. If your work is in a directory `mywork` then you can create a compressed tar file with the command

```
tar czf mywork.tar.gz mywork
```

Solutions and Comments

1.
   - It is a good idea to use the registration mechanism.
     - When using the registration mechanism then it is a good idea to make as many definitions static as possible; using a single file for your C code helps.
     - If you want to use multiple files use the `attribute_hidden` prefix.
     - It is also a good idea to use optional arguments to `useDynLib` to define variables for your entry points which You can use instead of using strings in your `.C` calls — this can improve performance.
   - Your package should pass `R CMD check` without errors, warnings, or notes.
   - Make sure to include a reasonable set of tests in your code.
   - When using `.C` you should check that scalar arguments like `log.p` have at least one element.
   - You should use the constants `NA_REAL`, `R_NaN`, `R_PosInf` or `R_NegInf` in C code when you need these values.
   - If you issue warnings make sure to do so only once per call — do not call `warning` or `error` inside your C loop.
   - Be sure to follow the coding standards in you R and your C code.
   - Make sure you push your changes to your class GitHub repository.
   - You should only export those variable that you want to be public; don't just export all variables.

2. Some notes:

   - It helps to use a good qualitative color scheme for showing the classification results in part (d).
   - The axes and labels provided by default in image plots are not useful in this case.
   - It is a good idea to use an appropriate aspect ratio for these images.
   - It is a good idea to describe the convergence criterion used for the EM algorithm. A relative error on the order of 10^{-8} might be reasonable. Using too large a tolerance can cause problems.
   - The classification can be computed with a vectorized expression, for example as

```
tclass <- function(x, theta) {
    mu <- theta$mu
    sigma <- theta$sigma
    p <- theta$p
```

```
    M <- length(mu)

    Ez <- outer(x, 1:M, function(x, i) p[i] * dnorm(x, mu[i], sigma[i]))
    max.col(Ez, ties.method = "first")
}
```

The image can then be filled in with

```
cl <- array(NA, dim(T1))
cl[mask] <- tclass(v, res$pars)
```

- For part (f), there are only a few hundred distinct intensity values, and the function `tabulate` can be used to determine how often each occurs. The EM computations can be rewritten to carry out each distinct calculation once and multiply by the appropriate count. This is easiest to do by modifying the EM step to take a weights argument:

```
EMmix1W <- function(x, w, theta) {
    mu <- theta$mu
    sigma <- theta$sigma
    p <- theta$p
    M <- length(mu)

    ## E step
    Ez <- outer(x, 1:M, function(x, i) p[i] * dnorm(x, mu[i], sigma[i]))
    Ez <- sweep(Ez, 1, rowSums(Ez), "/")
    EzW <- sweep(Ez, 1, w, `*`)
    colSums.EzW <- colSums(EzW)

    ## M step
    xpW <- sweep(EzW, 1, x, "*")
    mu.new <- colSums(xpW) / colSums.EzW

    sqRes <- outer(x, mu.new, function(x, m) (x - m)^2)
    sigma.new <- sqrt(colSums(EzW * sqRes) / colSums.EzW)

    p.new <- colSums.EzW / sum(colSums.EzW)

    ## pack up result
    list(mu = mu.new, sigma = sigma.new, p = p.new)
}
```

This can be tested and compared to the original EM step with

```
counts <- table(as.vector(v))
vv <- sort(unique(as.vector(v)))
```

```
EMmix1W(vv, counts, theta)
```

- Binning or discretization can be useful for some large data set computations; for example

  `http://vita.had.co.nz/papers/bigvis.html`
  `https://github.com/hadley/bigvis`

Assignment 8

Due on Monday, March 27, 2017.

1. Suppose $Y_i = m(x_i) + \varepsilon_i$ with $x_i = \frac{i}{n+1}$,

$$m(x) = 1 - \sin(5x)^2 e^{-4x}$$

and the ε_i $i.i.d$ $\mathrm{N}(0, \sigma^2)$ random variables. Consider the following estimators of m avaiable in R:

`smooth.spline`

`lowess`

`supsmu`

`loess`

The fit produced by `gam` in packages `mgcv` for the model `y ~ s(x)`

all using default smoothing parameter settings.

(a) For $n = 50$ and $\sigma = 0.1$ use simulation to estimate the bias and standard error of the estimates at each x_i and show the results graphically.

(b) The average mean square error

$$aMSE(\widehat{m}) = \frac{1}{n}\sum_{i=1}^{n} E[(\widehat{m}(x_i) - m(x_i))^2]$$

is a measure of the overall quality of the estimator that approximates the integrated mean square error. Use simulation to estimate the average mean square error of the estimators for $n = 50$ and a range of σ values and summarize the results in tables and/or graphs. Comment on any performance differences you see in this example.

Make sure your simulation sample sizes are sufficient to support any conclusions.

Include your code in an appendix to your writeup and as a separate text file. Your code should make it easy to change the parameters of the simulation.

2. Modify your `pareto` package to include functions `p.dpareto`, `p.ppareto`, and `p.qpareto` that are implemented using C code that uses Open MP to allow the loop over elements of the argument vectors to be run in parallel. Your new functions should take one additional argument, `P`, that specifies the number of computational threads to use for the loop. Try your function on vectors of various lengths using one and two threads and try to determine how large the vector has to be for the parallel approach to improve performance. Look at the

elapsed time values provided by `system.time`; because of the simplicity of the density you may need a fairly long vector to see the effect of parallelization.

Some notes on Open MP are available. The notes about the use within R are out of date but the overview of Open MP is still relevant. Use of Open MP is also discussed briefly in the *Writing R Extensions* manual. A modified version of the `AddOne` package is available that uses Open MP is available as well. This package contains a function `p.AddOne` similar to the one you are asked to write.

Please use the Linux systems for this as performance of Open MP on Windows and Mac OS X is not satisfactory for this problem.

Your package should pass `R CMD check` without errors or warnings. You should submit your package as a source package file created by `R CMD build`.

Also commit and push your revised package code to your class GitHub repository.

Your submission should include your revized `pareto` package, a pdf file with your writeup and text files with a `.R` extension with your code for each of the problems.

You should submit your assignment electronically using Icon. Submit your work as a single compressed tar file. If your work is in a directory `mywork` then you can create a compressed tar file with the command

```
tar czf mywork.tar.gz mywork
```

Solutions and Comments

1.
   - Please follow the coding standards on use of spaces, indentation and long lines.
   - Descriptions of simulation experiments should always include information about the simulation design, in particular the number of replicates used.
   - Simulation results should include simulation standard errors.
   - You need to think about the simulation standard errors to determine an appropriate simulation sample size. You may need a preliminary experiment to do this.
     - A simulation sample size of 100 is much too small.
     - Even 10,000 leaves substantial uncertainty about absolute magnitudes.
     - Some simulation variance reduction techniques can make comparisons much more accurate.
   - Smoothness of the mean function and signal to noise ratio will affect the performance of smoothing methods.

2. OpenMP is a very useful framework for taking advantage of multicore computiers. It is fairly easy to use for simple parallelizations, though some care is needed to avoid pitfalls.

   Some notes:

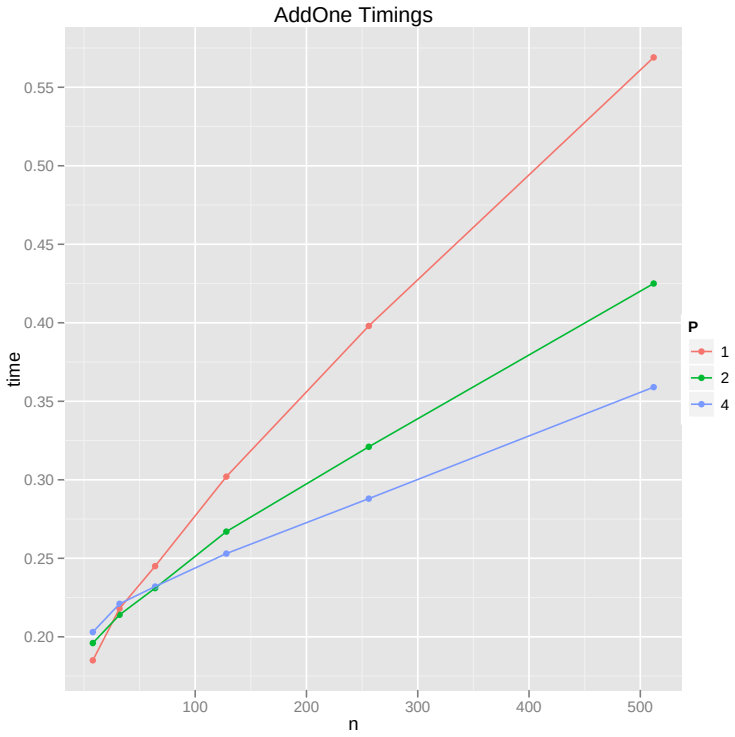
   - You need to make sure OpenMP is enabled; this is done by compiler and linker flags set in `Makevars`.
   - To check that your code is using multiple threads you can
     - use `htop`; for example, watch `htop` while running
       ```
       x <-as.double(1 : 64)
       system.time(for (i in 1: 100000) p.AddOne(x, P = 4))
       ```
     - check that for large enough n your `user` time exceeds your `elapsed` time (the `user` time is cumulative over all threads in the process).
   - You need to be careful only to call functions that are *thread-safe* in the body of a parallel loop.
     - Basic arithmetic functions are safe.
     - For anything else assume it is not safe unless you have checked and made sure that it is.
     - In particular, *do not* call any functions that could allocate memory in R or signal errors or warnings.
   - Ahmdahls Law: if the proportion f of a computation is parallelizable, then using P processors will reduce computation time by a factor of

     $$1 - f + \frac{f}{P}$$

If half your computation is in the sequential portion then you can expect at most a factor of 2 speed-up no matter how many threads you use.

  - This is an approximation.
  - The parallel and sequential components will vary approximately linarly with problem size n.
  - The sequential poriton includes both setup and memory traffic.
  - There is a cost of synchronization associated with coordinating multiple threads.
  - To make f as large as possible you should try to put all $O(n)$ operations into the parallel `for` loop.

- You can use *profiling* (see `?Rprof`) to see how much time your sequential code spends in the `.C` call.
- Some timing results for the `AddOne` example:



- As always, make sure to follow the coding guidelines.

Some test code is available in

`http://www.stat.uiowa.edu/~luke/classes/STAT7400/paretoOMPtests.R`

Assignment 9

Due on Monday, April 10, 2017.

1. Returning again to the setting or Problem 1 from Assignment 8, repeat the problem using the estimates produced by

   `gam`

   `rpart`

   `gbm`

   `randomForest`

   `nnet`.

   Be sure to use appropriate simulation sample sizes and to include simulation standard errors in your report.

You should submit your assignment electronically using Icon. Submit your work as a single compressed tar file. If your work is in a directory `mywork` then you can create a compressed tar file with the command

```
tar czf mywork.tar.gz mywork
```

Solutions and Comments

1. (a) Please follow the coding standards.

   (b) If there are parameters you need to specify for an algorithm you should state what you used, and why, in your report.

   (c) For simulation studies you should include some indication of accuracy in plots and tables.

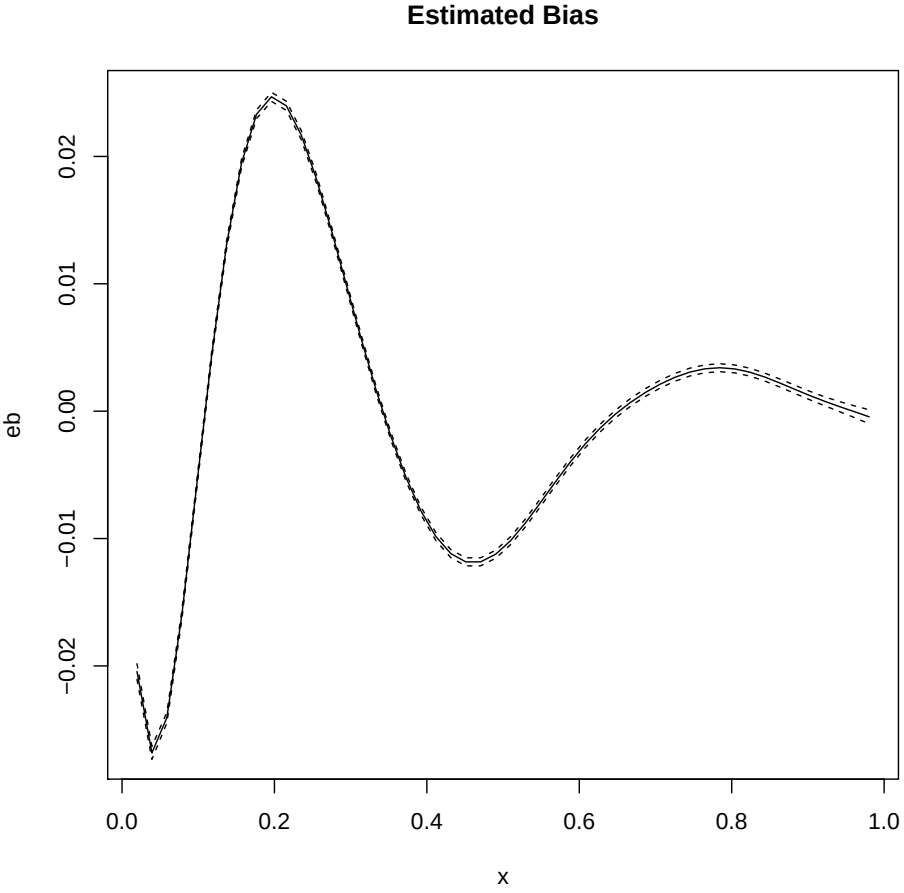
   (d) A simulation size of 1,000 or 2,000 for a simple problem such as this is too small. You should use at least 10,000 independent replicates unless there are compelling reasons not to.

   One way to run the simulation for `gam` (this takes about 90 seconds on my laptop):

```
library(mgcv)

m <- function(x) 1 - sin(5 * x)^2 * exp(-5 * x)
n <- 50
x <- (1 : n) / (n + 1)

R <- 10000

sim <- function(fitfun, sigma = 0.1) {
    sapply(seq_len(R), function(i) {
        y <- rnorm(n, m(x), sigma)
        predict(fitfun(x, y))
    })
}

yhat <- sim(function(x, y) gam(y ~ s(x)))
```

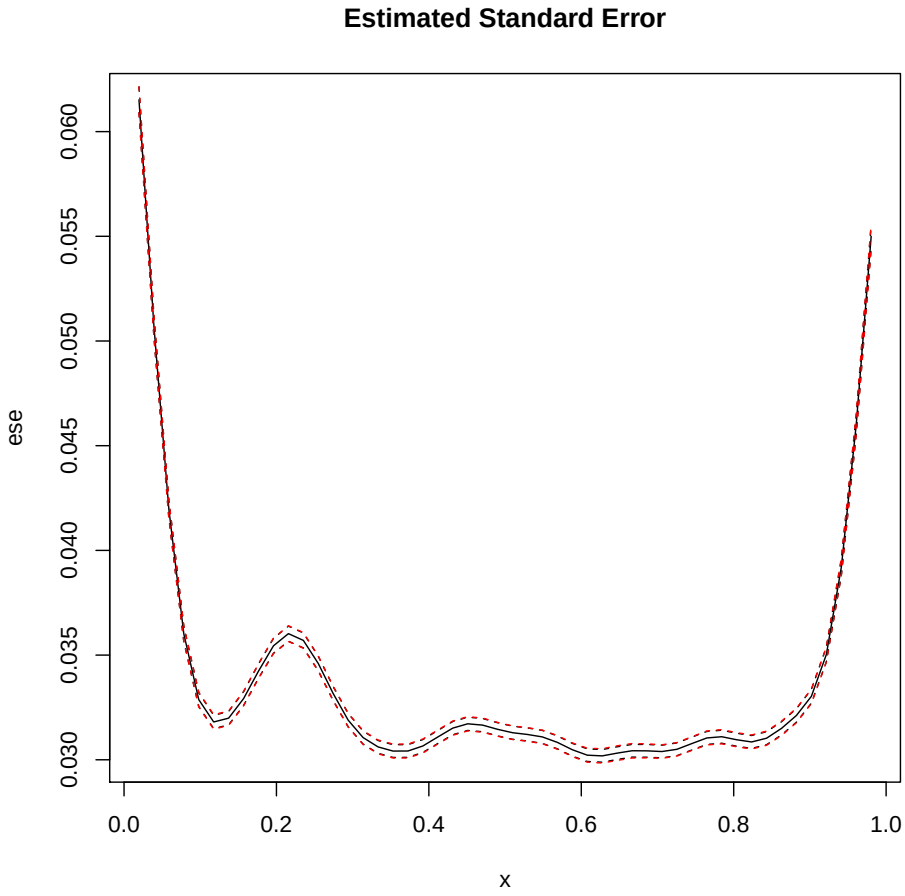
   A plot of the estimated bias, with dashed lines showing $\pm$ one simulation standard error:

```
eb <- rowMeans(yhat) - m(x)
ese <- apply(yhat, 1, sd)
plot(x, eb, type = "l", main = "Estimated Bias")
lines(x, eb +  ese / sqrt(R), lty = 2)
lines(x, eb -  ese / sqrt(R), lty = 2)
```



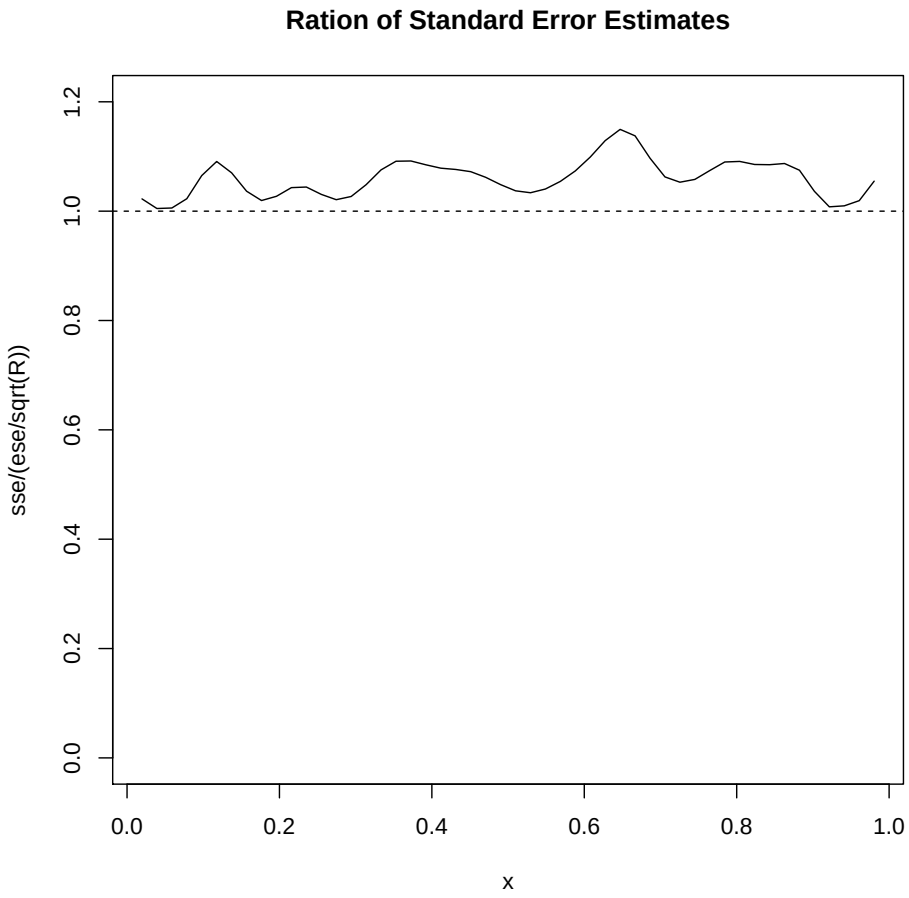
Assuming the estimates are approximately normal we can use the gamma (scaled χ^2) distribution of the sample variance and the delta method to obtain approximate simulation standard errors on the standard errors of the function estimates. If we do not use normality then we can estimate the variance of the sample variance using the variance of the squared deviations from the sample mean.

```
plot(x, ese, type = "l", main = "Estimated Standard Error")

## Chi-square approximation with delta method
lines(x, ese * (1 + sqrt(1 / R)), lty = 2)
lines(x, ese * (1 - sqrt(1 / R)), lty = 2)

## SE of squared deviations with delta method
sse <- apply(sweep(yhat, 1, rowMeans(yhat)) ^ 2, 1, sd) / ese / sqrt(2 * R)
lines(x, ese + sse, lty = 2, col = "red")
lines(x, ese - sse, lty = 2, col = "red")
```



The estimates assuming normality are slightly smaller:

```
plot(x, sse / (ese / sqrt(R)), type = "l", ylim = c(0, 1.2),
     main = "Ration of Standard Error Estimates")
abline(h = 1, lty = 2)
```



The estimates average mean square error and the simulation standard error of the estimate:

```
> ase <- colMeans(sweep(yhat, 1, m(x)) ^ 2)
> amse <- mean(ase)
> sea <- sd(ase) / sqrt(R)
> sea / amse
[1] 0.005583625
```

Assignment 10

Due on Monday, April 17, 2017.

1. (a) Modify your `pareto` package to include a function `rpareto`, written in R only, for generating random variables from a Pareto distribution. Your function should follow the standard usage of random number generator functions in R. Your package should pass `R CMD check` without errors, warnings, or notes. In your writeup explain what method you used to implement your generator and show some usage examples.

   (b) Add a function `rcpareto` to your package that takes the same arguments as `rpareto` and generates its random numbers in C code. The section of the R extension manual on *Random number generation* provides the information you need for this. Again explain your method and show some examples of use in your writeup.

You should submit your assignment electronically using Icon. Submit your work as a single compressed tar file. If your work is in a directory `mywork` then you can create a compressed tar file with the command

```
tar czf mywork.tar.gz mywork
```

Solutions and Comments

1. Some notes:

   - Your package should pass `R CMD check` without errors, warnings, or notes.
   - Follow the coding standards on avoiding long lines, proper indentation, and use of spaces.
   - Make sure you are generating from the correct distribution. A simple check:

     ```
     ppareto(rpareto(n, a, b), a, b)
     ```

     should look like a sample from a uniform distribution on $[0, 1]$.
   - Given that you have a working `qpareto` it makes sense to start with that for your inversion-based generator.
   - Checking generator output can be tricky:
     - Lots of statistical tests are available, but they do fail some of the time even when things are working properly.
     - Using a small α and fixing the generator and seed can help.
   - Your code should check for bad parameter values.
   - You should follow the convention that the length of `n` is used as the number of variables to generate if it is greater than 1.
   - Some of you are still using argument recycling code that is too inefficient — it uses vastly too much memory. You have something like

     ```
     rep(a, n)[1:n]
     ```

     This has several problems:
     - If `a` has length 1 then this is a little inefficient — you do not need the `[1:n]`.
     - If `a` has length n, then you are creating a vector of length n^2 and selecting the first n elements.

     What you want is

     ```
     rep(a, length.out = n)
     ```

     or

     ```
     rep_len(a, n)
     ```

   - Allocating large arrays on the C stack is a bad idea — allocate temporaries on the heap in R or using the C level allocation functions.

- You should place your `GetRNGstate/PutRNGstate` calls *outside* the loop.
- You should use R's random number generator functions in your C code.
  - This allows the user to change the underlying generator by standard mechanisms.
  - This also allows the user to reproduce identical sequences of random numbers using the `set.seed` function.
- You should *not* set the seed in your code.

Some test code is available in

`http://www.stat.uiowa.edu/~luke/classes/STAT7400/paretoRNGtests.R`

Assignment 11

Due Monday, April 24, 2017

1. The objective of this problem is to give you some experience running simple parallel computations in R. You will use the `parallel` package for this. The manual for the `package` is available at

   `http://stat.ethz.ch/R-manual/R-devel/library/parallel/html/00Index.html`.

   The package vignette may also be useful:

   `http://stat.ethz.ch/R-manual/R-devel/library/parallel/doc/parallel.pdf`.

   A simple example along the lines of this problem is given in

   `http://homepage.stat.uiowa.edu/~luke/classes/STAT7400/examples/parallel/gamma.R`.

   Some examples illustrating how to pass function and variable definitions from the master to the workers and loading packages on the workers are given in

   `http://homepage.stat.uiowa.edu/~luke/classes/STAT7400/examples/parallel/export.R`.

   (a) Returning to the setting or Problem 1 from Assignment 8, write a function that takes the number of simulation replications `R`, the sample size `n` for the data sets ($n = 50$ is what you used previously), and the error standard deviation `s` as arguments and returns a matrix with one row for each x_i and one column for the estimated bias and estimates standard error at each x_i for the fit produced by the `gam` estimator in the `mgcv` package.

   (b) Write a function that takes a computational cluster as its first argument, followed by the arguments of the previous function, splits the simulation across the cluster, and returns a result in the same form as the previous function.

   (c) Use your functions for $n = 50$, $\sigma = 0.2$, and $R = 10,000$ on a cluster of at least 2 workers and confirm that you are getting a a significant reduction in elapsed time using the parallel approach. Also run your simulation twice with the same seed and confirm that you are getting identical results.

You should submit your assignment electronically using Icon. Submit your work as a single compressed tar file. If your work is in a directory `mywork` then you can create a compressed tar file with the command

```
tar czf mywork.tar.gz mywork
```

Solutions and Comments

1. Some notes:

   - You should shut down your cluster using `stopCluster`.
   - Make sure your function signature matches the specification.
   - For any parallel simulation it is a good idea to use a parallel RNG.
   - When parallelising code using a distributed memory approach:
     - It is a goo idea to put as much into your parallel function as possible.
     - This allows you to debug and test as much of the code a possible in a sequential setting.
     - It is also important to limit the amount of data that has to be transferred. So perform data reductions on the worker as much as possible.
   - The objects returned by `makeCluster` represent a set of separate R processes.
   - By default, these processes communicate with the master over *sockets*.
   - Other communication mechanisms are possible (e.g. MPI).
   - This framework supports parallel computing using one or more scatter-compute-gather operations.
   - Much more sophisticated parallel algorithms are possible but are much harder to learn to program. The `Rmpi` package might be useful for implementing some of these.
   - Nevertheless, many problems can be usefully parallelized with just scatter-compute-gather steps.
   - If you need to use a sequence of such steps you may need to minimize the amount of data communicated between workers and master between the steps for the parallelization to be effective.
   - The `snow` package provides some timing and visualization tools for understanding how much time is spent in communication.
   - A very major problem in general parallel computing is the possibility of deadlock. This will not happen in this simple framework.
   - It is possible to start worker jobs running that either do not terminate or would take too long to wait for. At the moment these have to be terminated by external means.