STAT 380: Computational Statistics
2016 FINAL EXAMINATION

Student Solution Document

Time allowed: 3 Hours

This examination paper comprises 16 pages.
Answer ALL questions.
The marks total 70.

Name: Ash Midgley

1. Two people decide to play a dice rolling game. Each player has a die that they roll n times. The winner is the player with the highest mean value after n rolls. The two players decide to make it a race, checking the mean value after each roll to determine who is in the lead.

   (a) [2 marks] Write a function that simulates $n = 30$ die rolls for each player. The only input should be the value n. The output should be a vector of n rolls for each player. Test your function for $n = 30$.

```
rolls = function(n) {
    p1 = rep(NA, n)
    p2 = rep(NA, n)
    for (i in 1:n) {
        r1 = sample(1:6, 1)
        p1[i] = r1
        r2 = sample(1:6, 1)
        p2[i] = r2
    }
    return(list(p1, p2))
}
rolls(30)
## [[1]]
##  [1] 3 3 3 2 4 3 6 4 4 3 5 6 3 4 4 6 2 6 1 3 3 2 2 1 4 6 6 4 5 6
##
## [[2]]
##  [1] 1 1 3 1 3 2 3 2 2 2 3 1 4 1 3 2 5 6 6 2 5 4 2 2 5 2 1 4 6 1
```

   (b) [2 marks] Write a function to find the mean values. The input is a single vector that contains the realization of a sequence of n die rolls. The output is a vector of length n that gives the mean value after each roll (i.e. the ith element of the vector is the mean value of all rolls up to an including the ith roll). Test your function using the input vector x below.

```
x = c(1, 3, 5, 5, 2, 1, 1, 3, 4, 1, 6, 6, 4, 6, 5,
    2, 5, 3, 3, 4, 6, 3, 1, 1, 1, 4, 6, 5, 2, 4)
```

```
findMeans = function(vec) {
    n = length(vec)
    meanvec = rep(NA, n)
    for (i in 1:n) {
        meanvec[i] = mean(vec[1:i])
    }
    return(meanvec)
}
findMeans(x)
##  [1] 1.000000 2.000000 3.000000 3.500000 3.200000 2.833333 2.571429 2.
##  [9] 2.777778 2.600000 2.909091 3.166667 3.230769 3.428571 3.533333 3.
## [17] 3.529412 3.500000 3.473684 3.500000 3.619048 3.590909 3.478261 3.
## [25] 3.280000 3.307692 3.407407 3.464286 3.413793 3.433333
```

   (c) [2 marks] Write a function that finds who is leading after each roll. The inputs are a vector for each player giving the mean value after every roll. The output is a vector of length n that gives the leader after each roll (you should use the value 1 if player

1 is leading, 2 if player 2 is leading and 3 if it is a tie). Test your function with the input vectors $y1$ and $y2$ below:

```
y1 = c(5, 3.5, 3.67, 3.5, 3.2, 3.33, 3.43, 3.25, 3.56,
    3.5, 3.55, 3.5, 3.46, 3.43, 3.53, 3.69, 3.65, 3.72,
    3.74, 3.85, 3.76, 3.77, 3.7, 3.67, 3.64, 3.69,
    3.63, 3.71, 3.69, 3.73)
y2 = c(6, 3.5, 4.33, 3.5, 4, 3.67, 4, 3.62, 3.33, 3.5,
    3.36, 3.25, 3.46, 3.36, 3.33, 3.25, 3.12, 3, 3.16,
    3.25, 3.29, 3.32, 3.43, 3.33, 3.24, 3.23, 3.26,
    3.29, 3.21, 3.23)
```

```
leader = function(p1, p2) {
    n = length(p1)
    leadervec = rep(NA, n)
    for (i in 1:n) {
        p1M = p1[i]
        p2M = p2[i]
        if (p1M > p2M) {
            leadervec[i] = 1
        } else {
            if (p1M < p2M) {
                leadervec[i] = 2
            } else {
                leadervec[i] = 3
            }
        }
    }
    return(leadervec)
}
leader(y1, y2)
##  [1] 2 3 2 3 2 2 2 2 1 3 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

(d) [2 marks] Write a function that finds the number of lead changes. The function takes as input a vector giving the leader after each roll. The output is a single value that records the number of lead changes throughout the game (this includes changes into and out of a tie). Test your function using the input vector z below:

```
z = c(1, 3, 2, 2, 2, 2, 2, 2, 2, 3, 3, 1, 1, 1, 1,
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2)
```

```
leadChanges = function(lvec) {
    n = length(lvec)
    leadC = 0
    prevL = lvec[1]
    for (i in 2:n) {
        currL = lvec[i]
        if (currL != prevL) {
            leadC = leadC + 1
        }
        prevL = currL
    }
```

```
    return(leadC)
}
leadChanges(z)
## [1] 5
```

(e) [3 marks] Write a function that simulates the entire game. The input is the number of rolls n, defaulted to 30. The output should consist of

- A vector for each player that gives the mean value after each roll;
- A vector giving the leader after each roll;
- The number of lead changes in the game.

Use your function to simulate a game when $n = 100$.

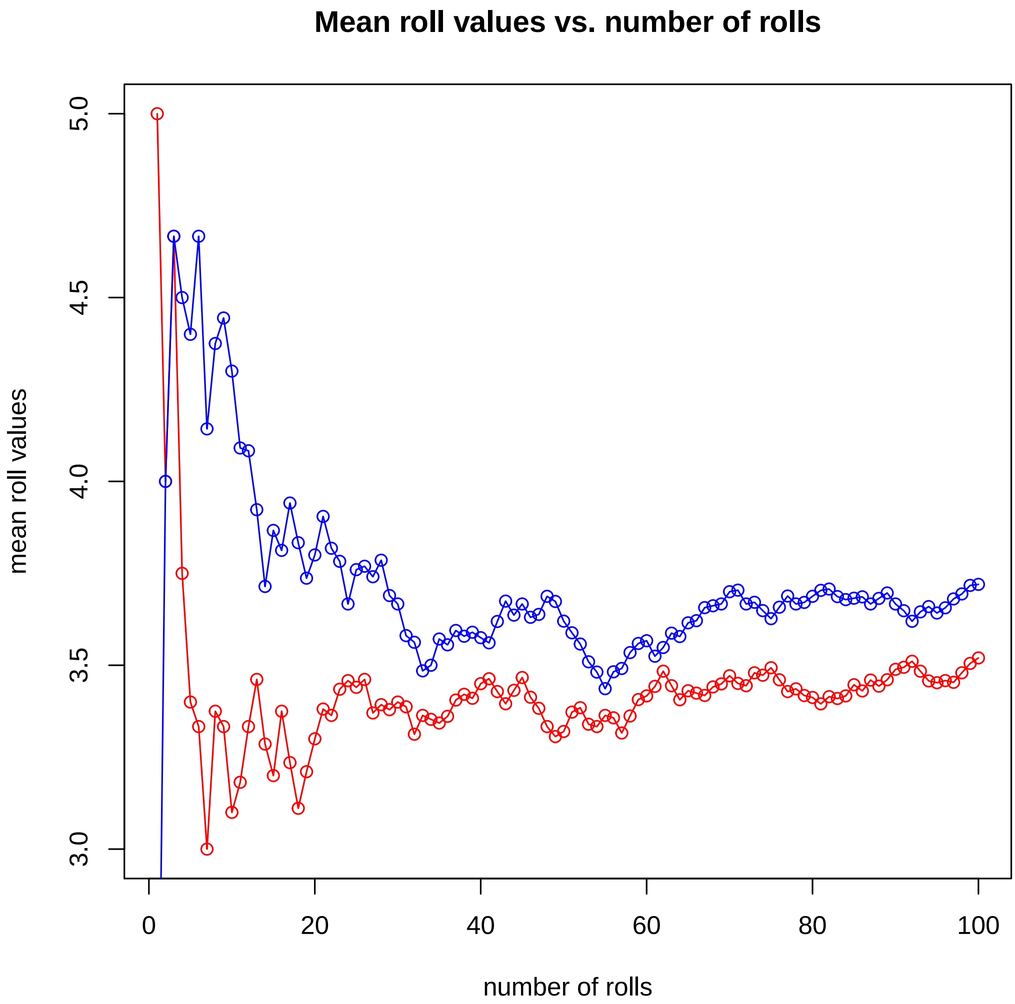
```
simGame = function(n = 30) {
    rollvals = rolls(n)
    p1rolls = rollvals[[1]]
    p2rolls = rollvals[[2]]
    p1M = findMeans(p1rolls)
    p2M = findMeans(p2rolls)
    leaderVec = leader(p1M, p2M)
    leadChanges = leadChanges(leaderVec)
    return(list(p1M, p2M, leaderVec, leadChanges))
}
out = simGame(100)
out
## [[1]]
##   [1] 5.000000 4.000000 4.666667 3.750000 3.400000 3.333333 3.000000 3
##   [9] 3.333333 3.100000 3.181818 3.333333 3.461538 3.285714 3.200000 3
##  [17] 3.235294 3.111111 3.210526 3.300000 3.380952 3.363636 3.434783 3
##  [25] 3.440000 3.461538 3.370370 3.392857 3.379310 3.400000 3.387097 3
##  [33] 3.363636 3.352941 3.342857 3.361111 3.405405 3.421053 3.410256 3
##  [41] 3.463415 3.428571 3.395349 3.431818 3.466667 3.413043 3.382979 3
##  [49] 3.306122 3.320000 3.372549 3.384615 3.339623 3.333333 3.363636 3
##  [57] 3.315789 3.362069 3.406780 3.416667 3.442623 3.483871 3.444444 3
##  [65] 3.430769 3.424242 3.417910 3.441176 3.449275 3.471429 3.450704 3
##  [73] 3.479452 3.472973 3.493333 3.460526 3.428571 3.435897 3.417722 3
##  [81] 3.395062 3.414634 3.409639 3.416667 3.447059 3.430233 3.459770 3
##  [89] 3.460674 3.488889 3.494505 3.510870 3.483871 3.457447 3.452632 3
##  [97] 3.453608 3.479592 3.505051 3.520000
##
## [[2]]
##   [1] 2.000000 4.000000 4.666667 4.500000 4.400000 4.666667 4.142857 4
##   [9] 4.444444 4.300000 4.090909 4.083333 3.923077 3.714286 3.866667 3
##  [17] 3.941176 3.833333 3.736842 3.800000 3.904762 3.818182 3.782609 3
##  [25] 3.760000 3.769231 3.740741 3.785714 3.689655 3.666667 3.580645 3
##  [33] 3.484848 3.500000 3.571429 3.555556 3.594595 3.578947 3.589744 3
##  [41] 3.560976 3.619048 3.674419 3.636364 3.666667 3.630435 3.638298 3
##  [49] 3.673469 3.620000 3.588235 3.557692 3.509434 3.481481 3.436364 3
##  [57] 3.491228 3.534483 3.559322 3.566667 3.524590 3.548387 3.587302 3
##  [65] 3.615385 3.621212 3.656716 3.661765 3.666667 3.700000 3.704225 3
##  [73] 3.671233 3.648649 3.626667 3.657895 3.688312 3.666667 3.670886 3
```

```
##  [81] 3.703704 3.707317 3.686747 3.678571 3.682353 3.686047 3.666667 3
##  [89] 3.696629 3.666667 3.648352 3.619565 3.645161 3.659574 3.642105 3
##  [97] 3.680412 3.693878 3.717172 3.720000
##
## [[3]]
##   [1] 1 3 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [38] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [75] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##
## [[4]]
## [1] 2
```

(f) [4 marks] Plot a line graph of mean value (y-axis) against the number of rolls (x-axis) using the output from the simulation you conducted in part (e). The data for the two players should be plotted in different colours. Ensure your plot is appropriately labeled.

Note: if you were unable to successfully implement the function in part (e) you should use the mean vectors $y1$ and $y2$ given in part (c).

```
plot(1:100, out[[1]], type = "o", col = "red", main = "Mean roll values v
    ylab = "mean roll values", xlab = "number of rolls")
lines(out[[2]], type = "o", col = "blue")
```



(g) [2 marks] Give a step-by-step description of how you could use the functions specified above to estimate the average number of lead changes in a $n = 100$ game using simulation.
- I would first define the m value and set the corresponding probability vector to size m
- I would then loop from 1 to m
- For each iteration I would simulate the rolls with n=100, then use this to find the mean roll vectors for each player, then find the current leader vector and then use this to find the number of lead changes
- Once I have found the lead changes for that iteration, I would input it into the probability vector at the current index
- Once the m iterations are complete and we have filled the probability vector, I would find the mean value of the prob vector which would give us the average number of lead changes in a n=100 game

2. The dataset **state** contains information on 50 US states and is made up of built-in datasets *state.abb*, *state.division* and *state.x77*.

```r
state = data.frame(state = state.abb, region = state.division,
    state.x77)
```

(a) [2 marks] We wish to recode the factor variable *region*. Replace both the *Pacific* and *Mountain* regions by a new region: *West*.

```r
library(dplyr)
##
## Attaching package:  'dplyr'
## The following objects are masked from 'package:stats':
##
##     filter, lag
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
levels(state$region) = c("New England", "Middle Atlantic",
    "South Atlantic", "East South Central", "West South Central",
    "East North Central", "West North Central", "West",
    "West")
```

(b) [1 mark] Add a variable to **state** called *Density*. It is given as

$$Density = \frac{1000 \times Population}{Area}.$$

```r
state = mutate(state, Density = ((1000 * Population)/Area))
```

(c) [1 mark] Find the median and interquartile range of the murder rate across all 50 states.

```r
median(state$Murder)
## [1] 6.85
quantile(state$Murder)
##      0%     25%     50%     75%    100%
##  1.400   4.350   6.850  10.675  15.100
```

(d) [2 marks] Find the total population of each region.

```r
summarise(group_by(state, region), sum(state$Population))
## Source: local data frame [8 x 2]
##
##                 region sum(state$Population)
##                 (fctr)                 (dbl)
## 1          New England                212321
## 2      Middle Atlantic                212321
## 3       South Atlantic                212321
## 4   East South Central                212321
## 5   West South Central                212321
## 6   East North Central                212321
## 7   West North Central                212321
## 8                 West                212321
```

(e) [2 marks] Obtain a new data frame that only features states with *Population* more than 750 and *Frost* less than 170.

```
d2 = filter(state, Population > 750 | Frost < 150)
d2

##    state              region Population Income Illiteracy Life.Exp Murd
## 1     AL East South Central       3615   3624        2.1    69.05   15
## 2     AZ                West       2212   4530        1.8    70.55    7
## 3     AR West South Central       2110   3378        1.9    70.66   10
## 4     CA                West      21198   5114        1.1    71.71   10
## 5     CO                West       2541   4884        0.7    72.06    6
## 6     CT         New England       3100   5348        1.1    72.48    3
## 7     DE      South Atlantic        579   4809        0.9    70.06    6
## 8     FL      South Atlantic       8277   4815        1.3    70.66   10
## 9     GA      South Atlantic       4931   4091        2.0    68.54   13
## 10    HI                West        868   4963        1.9    73.60    6
## 11    ID                West        813   4119        0.6    71.87    5
## 12    IL East North Central      11197   5107        0.9    70.14   10
## 13    IN East North Central       5313   4458        0.7    70.88    7
## 14    IA West North Central       2861   4628        0.5    72.56    2
## 15    KS West North Central       2280   4669        0.6    72.58    4
## 16    KY East South Central       3387   3712        1.6    70.10   10
## 17    LA West South Central       3806   3545        2.8    68.76   13
## 18    ME         New England       1058   3694        0.7    70.39    2
## 19    MD      South Atlantic       4122   5299        0.9    70.22    8
## 20    MA         New England       5814   4755        1.1    71.83    3
## 21    MI East North Central       9111   4751        0.9    70.63   11
## 22    MN West North Central       3921   4675        0.6    72.96    2
## 23    MS East South Central       2341   3098        2.4    68.09   12
## 24    MO West North Central       4767   4254        0.8    70.69    9
## 25    NE West North Central       1544   4508        0.6    72.60    2
## 26    NH         New England        812   4281        0.7    71.23    3
## 27    NJ     Middle Atlantic       7333   5237        1.1    70.93    5
## 28    NM                West       1144   3601        2.2    70.32    9
## 29    NY     Middle Atlantic      18076   4903        1.4    70.55   10
## 30    NC      South Atlantic       5441   3875        1.8    69.21   11
## 31    OH East North Central      10735   4561        0.8    70.82    7
## 32    OK West South Central       2715   3983        1.1    71.42    6
## 33    OR                West       2284   4660        0.6    72.13    4
## 34    PA     Middle Atlantic      11860   4449        1.0    70.43    6
## 35    RI         New England        931   4558        1.3    71.90    2
## 36    SC      South Atlantic       2816   3635        2.3    67.96   11
## 37    TN East South Central       4173   3821        1.7    70.11   11
## 38    TX West South Central      12237   4188        2.2    70.90   12
## 39    UT                West       1203   4022        0.6    72.90    4
## 40    VA      South Atlantic       4981   4701        1.4    70.08    9
## 41    WA                West       3559   4864        0.6    71.72    4
## 42    WV      South Atlantic       1799   3617        1.4    69.48    6
## 43    WI East North Central       4589   4468        0.7    72.48    3
##    HS.Grad Frost   Area   Density
## 1     41.3    20  50708 71.290526
## 2     58.1    15 113417 19.503249
```

```
## 3      39.9    65  51945  40.619886
## 4      62.6    20 156361 135.570890
## 5      63.9   166 103766  24.487790
## 6      56.0   139   4862 637.597696
## 7      54.6   103   1982 292.129162
## 8      52.6    11  54090 153.022740
## 9      40.6    60  58073  84.910371
## 10     61.9     0   6425 135.097276
## 11     59.5   126  82677   9.833448
## 12     52.6   127  55748 200.850255
## 13     52.9   122  36097 147.186747
## 14     59.0   140  55941  51.143169
## 15     59.9   114  81787  27.877291
## 16     38.5    95  39650  85.422446
## 17     42.2    12  44930  84.709548
## 18     54.7   161  30920  34.217335
## 19     52.3   101   9891 416.742493
## 20     58.5   103   7826 742.908255
## 21     52.8   125  56817 160.356935
## 22     57.6   160  79289  49.452005
## 23     41.0    50  47296  49.496786
## 24     48.8   108  68995  69.091963
## 25     59.3   139  76483  20.187493
## 26     57.6   174   9027  89.952365
## 27     52.5   115   7521 975.003324
## 28     55.2   120 121412   9.422462
## 29     52.7    82  47831 377.913905
## 30     38.5    80  48798 111.500471
## 31     53.2   124  40975 261.989018
## 32     51.6    82  68782  39.472536
## 33     60.0    44  96184  23.746153
## 34     50.2   126  44966 263.754837
## 35     46.4   127   1049 887.511916
## 36     37.8    65  30225  93.167907
## 37     41.8    70  41328 100.972706
## 38     47.4    35 262134  46.682231
## 39     67.3   137  82096  14.653576
## 40     47.8    85  39780 125.213675
## 41     63.5    32  66570  53.462521
## 42     41.6   100  24070  74.740341
## 43     54.5   149  54464  84.257491
```

(f) [3 marks] Regress *Income* on *HS.Grad*. Find a 95% confidence interval for the effect of graduation rate on income. Interpret the confidence interval.

```
m1 = lm(state$Income ~ state$HS.Grad)
confint(m1)
##                      2.5 %     97.5 %
## (Intercept)    1000.70545 2861.50393
## state$HS.Grad    29.83854   64.48606
```

The confidence interval is between 29.83854 and 64.48606

3. (a) [3 marks] State the difference between a list and a vector. Give an example where a list is preferred over a vector.
   - Lists allow storage of multiple individual objects under one object name
   - Vectors are only allowed to made up of a single class, whereas lists allow objects of different classes
   - Each element in a list can be a vector, a matrix or another list etc
   - Example below shows how vectors dont store vectors properly but lists can
   EXAMPLE:

```
workingList = list(c(1, 2, 3), c("a", "b"))
errorVec = c(c(1, 2, 3), c("a", "b"))
workingList
## [[1]]
## [1] 1 2 3
##
## [[2]]
## [1] "a" "b"
errorVec
## [1] "1" "2" "3" "a" "b"
```

   (b) [4 marks] We have measured the height of 30 kahikatea trees around Otago. We have estimated the mean tree height and used a large-sample confidence interval based on the central limit theorem. Give step-by-step directions for how we could use simulation to estimate the coverage of the large-sample confidence interval.

   - Set number of simulations (m)
   - Find true mean based on theta
   - Make a vector with sample sizes across the range
   - store the length of this vec in variable n
   - set a coverage vec to this size
   - Iterate through from 1 to n
   - For each iteration, find the confidence interval
   - Check whether the true mean is within this confidence interval
   - If so add 1 to index of coverage vec, otherwise 0
   - After we have finished iterating, we can view the coverage in a date frame

4. Researchers are interested in the number of faults in rolls of fabric as a function of the length of the roll. The first 5 rows of the data are in the table below.

   (a) [6 marks] Use Poisson regression via the `glm()` function to fit a model of the form:

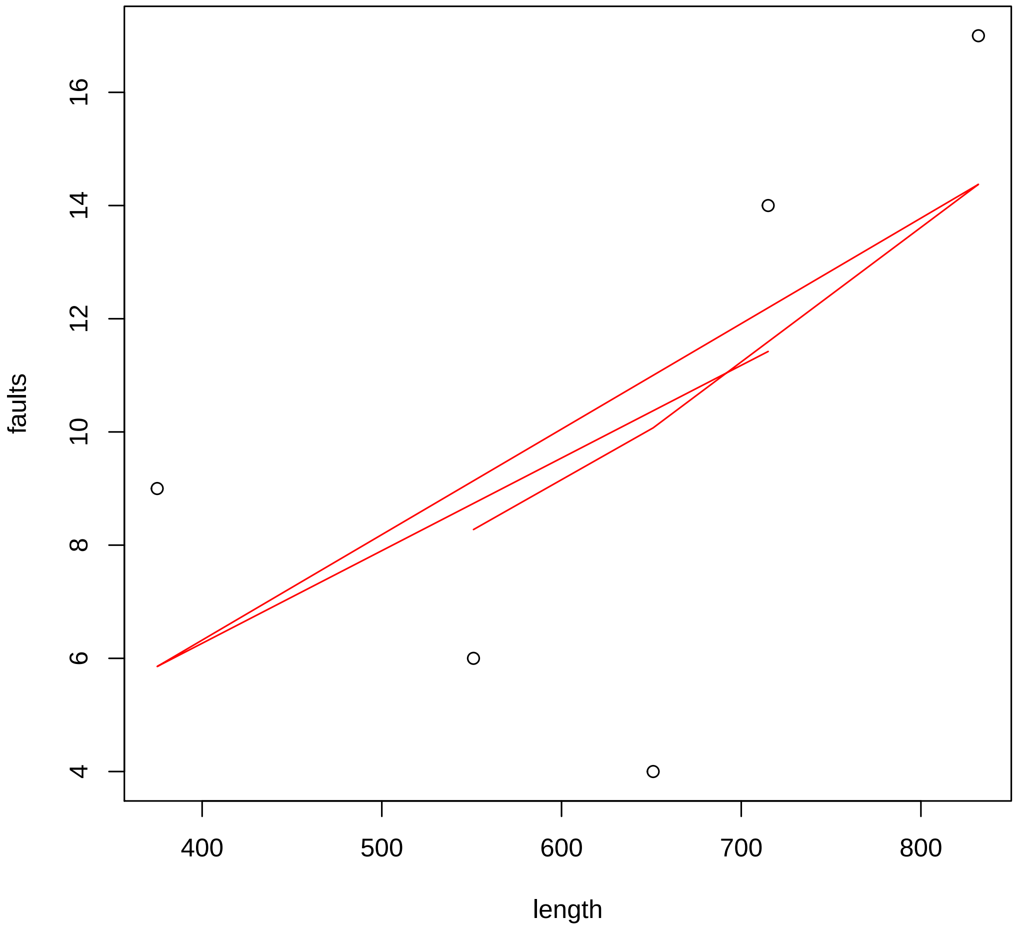
$$\log(\lambda_i) = \beta_0 + \beta_1 x_i$$

   where λ_i is expected number of faults in a roll of length x_i. Provide a brief summary of your model fit, including a plot of the expected number of faults vs roll length. This plot should include a line showing the fitted values under the model.

Faults	Length
6	551
4	651
17	832
9	375
14	715

Table 1: The number of faults in rolls of fabric as a function of the length of the roll

```
faults = c(6, 4, 17, 9, 14)
length = c(551, 651, 832, 375, 715)
fit = glm(faults ~ length, family = poisson)
fit
##
## Call:  glm(formula = faults ~ length, family = poisson)
##
## Coefficients:
## (Intercept)       length
##    1.030585     0.001965
##
## Degrees of Freedom: 4 Total (i.e. Null);  3 Residual
## Null Deviance:     12.11
## Residual Deviance: 7.892  AIC: 32.04
plot(length, faults)
lines(length, fit$fitted.values, col = "red")
```



(b) [2 marks] Use your parameter estimates from part (a) to estimate θ, the value of x_i when the expected number of faults is exactly 5.

```
logval = log(5)
theta = (logval - beta[1])/beta[2]
## Error in beta[1]:  object of type 'closure' is not subsettable
theta
## Error in eval(expr, envir, enclos):  object 'theta' not
found
```

(c) [6 marks] Use the parametric bootstrap to find a 95% confidence interval for θ.

```
Q <- 999
bhatboot = matrix(NA, Q, 2)
thetaboot = rep(NA, Q)

for (i in 1:Q) {
    r = rbinom(length(faults), faults, prob = fit$fit)
    fitboot <- glm(r ~ length, family = poisson)
    bhatboot[i, ] = fitboot$coef
    thetaboot[i] = (logval - bhatboot[i, 1])/bhatboot[i,
```

```
        2]
}
## Warning in rbinom(length(faults), faults, prob = fit$fit):
NAs produced
## Warning in glm.fit(x = structure(numeric(0), .Dim = c(0L,
2L), .Dimnames = list(:  no observations informative at iteration
1
## Warning:  glm.fit:  algorithm did not converge
## Error in fit$qr[1L:nr, 1L:nvars]:  subscript out of bounds
ci = 2 * mean(thetaboot) - quantile(thetaboot, c(0.975,
    0.025))
## Error in quantile.default(thetaboot, c(0.975, 0.025)):
missing values and NaN's not allowed if 'na.rm' is FALSE
ci
## Error in eval(expr, envir, enclos):  object 'ci' not found
```

(d) [2 marks] Briefly explain the difference between parametric and non-parametric bootstrapping.
Parametric:
- Assume some parametric model for the underlying population Ftheta
- Estimate the parameters of thsi model
- Resample from this model using the estimated parameters to estimate uncertainty

Non-parametric:
- Make few assumptions about the underlying distribution
- Use Fhat as an estimator for F
- Fhat is non-parametric estimator for F

(e) [5 marks] Use JAGS implemented through R using R2jags to redo the analysis described above. Use normal $\mathcal{N}(0, 10000)$ (mean and variance) prior distributions for the parameters β_0 and β_1. As output, report traceplots for the parameters of the model and report statistics that provide posterior summaries.

```
library("R2jags")
## Loading required package:  rjags
## Loading required package:  coda
## Linked to JAGS 4.2.0
## Loaded modules:  basemod,bugs
##
## Attaching package:  'R2jags'
## The following object is masked from 'package:coda':
##
##    traceplot
model = function() {
    for (i in 1:n) {
        logit(p[i]) <- b[1] + b[2] * length[i]
    }
    for (i in 1:2) {
```

```
        b[i] ~ dnorm(0, 1e-04)
    }
    theta <- (logit(5) - b[i])/b[2]
}
data = c("Faults", "Length", "n")

inits = function() {
    list(b = rnorm(2, mean = beta, sd = 0.1))
}

params = c("b", "theta")
jagsfit = jags(data = data, inits, parameters.to.save = params,
    n.iter = 1e+05, model.file = model)
## module glm loaded
## Error in rnorm(2, mean = beta, sd = 0.1):  invalid arguments
traceplot(as.mcmc(jagsfit), ask = F)
## Error in traceplot(as.mcmc(jagsfit), ask = F): error in
evaluating the argument 'x' in selecting a method for function
'traceplot':  Error in as.mcmc(jagsfit) :  object 'jagsfit'
not found
jfsum = jagsfit$BUGS$sum
## Error in eval(expr, envir, enclos):  object 'jagsfit'
not found
```

(f) [2 marks] From your analysis in part (e), what is a 95% credible interval for θ?

```
theta95 = jfsum[4, c(3, 7)]
## Error in eval(expr, envir, enclos):  object 'jfsum' not
found
```

5. The pdf for a random variable with support on $(0, 1)$ is given by

$$f_X(x) = \begin{cases} 12(x^2 - x^3) & 0 \leq x \leq 1 \\ \\ 0 & \text{otherwise} \end{cases}$$

and with cumulative density function (cdf) given by

$$F_X(x) = \begin{cases} 0 & x < 0 \\ 4x^3 - 3x^4 & 0 \leq x \leq 1 \\ 1 & x > 1 \end{cases}$$

This distribution has a mode at 2/3.

(a) [4 marks] Write an R function called `dfx` that returns the pdf of the above distribution for any x.

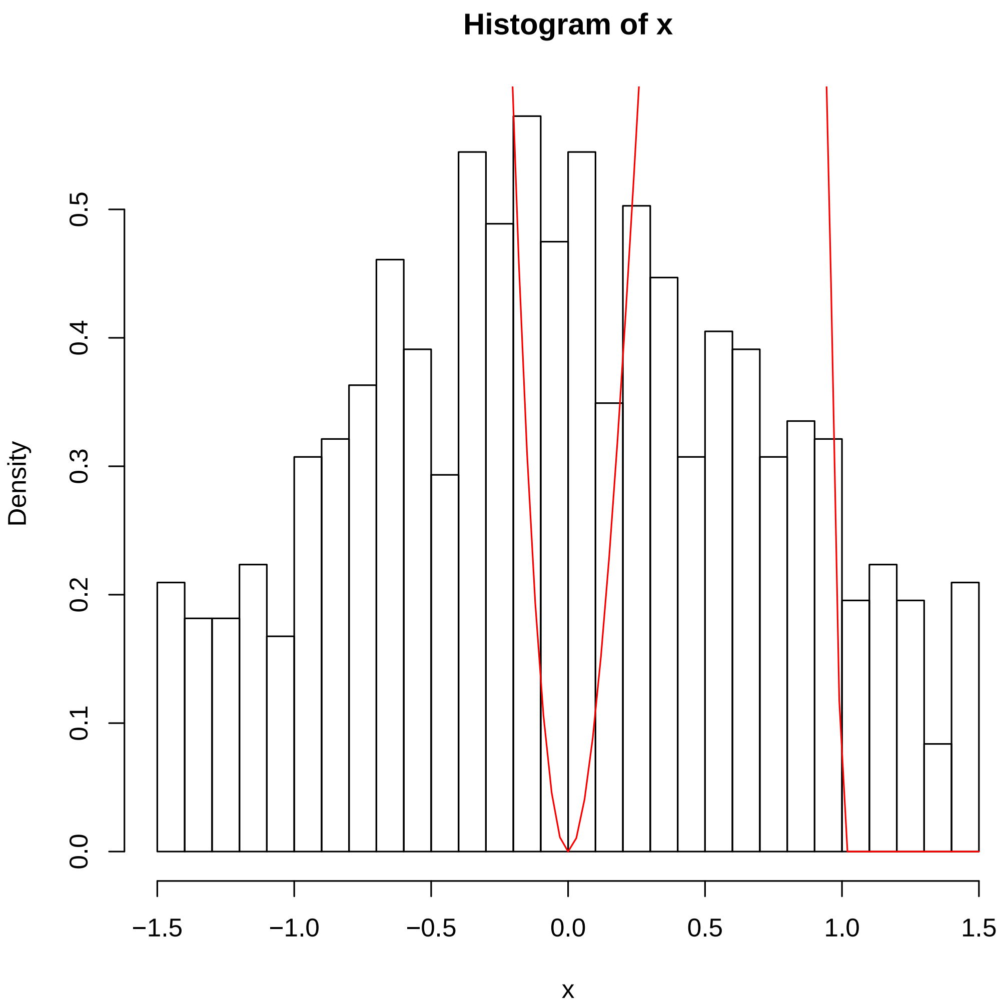
```
dfx = function(x) {
    brk = x <= 1
    result = brk * (12 * (x^2 - x^3))
    return(result)
}
```

(b) [4 marks] Write an R function called `rfx` that uses rejection sampling to generate exactly n samples from the above distribution with the number of samples defaulted to 1.

```
rfx = function(n = 1) {
    M = 1/sqrt(2 * pi)
    u = runif(n)
    x = runif(n, -1.5, 1.5)
    keep = which(u < dnorm(x)/M)
    y = x[keep]
    return(y)
}
```

(c) [3 marks] Use your functions to generate 10000 samples from the distribution with pdf $f_X(x)$. Display them using an appropriately labelled histogram with 25 breaks. You should overlay in red the corresponding probability density function.

```
x = rfx(1000)
hist(x, breaks = 25, prob = TRUE)
curve(dfx(x), add = T, col = "red")
```



(d) [1 mark] Based on your simulations what is the expected value of our random variable $\mathbb{E}[X]$?

```
0
## [1] 0
```