Mathematical practice final exam 2024

2024-07-08

1.

Solve the following system of equations using the `solve()` function:

$$\begin{pmatrix} 9 & 4 & 12 & 2 \\ 5 & 0 & 7 & 9 \\ 2 & 6 & 8 & 0 \\ 9 & 2 & 9 & 11 \end{pmatrix} \times \begin{pmatrix} x \\ y \\ z \\ t \end{pmatrix} = \begin{pmatrix} 7 \\ 18 \\ 1 \\ 0 \end{pmatrix}$$

```
A <- matrix(c(9, 4, 12, 2,
              5, 0, 7, 9,
              2, 6, 8, 0,
              9, 2, 9, 11),
            nrow = 4, byrow = TRUE)
b <- c(7, 18, 1, 0)
solution <- solve(A, b)
print(solution)
```

```
## [1] -4.2028571 -6.2285714  5.8471429 -0.2128571
```

2.

Execute the following lines which create two vectors of random integers which are chosen with replacement from the integers $0, 1, \ldots, 999$. Both vectors have length 250.

```
xVec <- sample(0:999, 250, replace=T)
yVec <- sample(0:999, 250, replace=T)
```

(a) Create the vector $(y_2 - x_1, \cdots, y_n - x_{n-1})$.

```
vec <- yVec[2:length(yVec)]-xVec[1:(length(xVec))-1]
print(vec)
```

```
##   [1]  320  427  750 -462  184  157  439   83 -206 -448  -22  376 -490  132  611
##  [16] -520 -126   67  488  162  436  -88  353 -625   31  235  521  418  179 -243
##  [31]  -59 -313  364   78  164   41  306 -180  221  370 -232 -190  -56  682 -253
##  [46]  421  231  891 -503   35 -737  698  545  -23  521 -217  816 -260  -96 -357
##  [61]  622  600  548   38 -242 -374  211   20 -313   13   67 -174   20  962 -653
##  [76] -166 -138  457  188 -208  394 -362 -101  506  641   81 -232    7 -112 -796
##  [91]  192  258  171 -103  -14  -39  -19 -203 -407  221  701  -96  -78 -271  424
## [106]  -55 -358  601  316 -374 -572 -549  141  -37 -162 -291   59   39  574  -84
```

```
## [121]  259  601  385 -267 -219  561 -339 -544 -796  276 -270  231  351   34  131
## [136] -545 -540  144   32 -436  302  128 -422  401  357 -514  210   96 -429  -75
## [151]  -10  121  211  437  152  214  485 -454 -277  584  792  318 -840 -107  477
## [166]  738  673 -476  753  814  310  220   84  -15  -17  306  821   68  415 -417
## [181]  -69 -462 -545  452  -27  115 -554  499  552 -214   15  273 -169 -573  466
## [196]  534 -745 -212   -1 -128 -247  767  635  -54 -421  238  -42 -119 -436  446
## [211]  592  359 -481  198 -106  462  103  106 -610  607 -134  204 -107 -383   64
## [226]   -7 -119  743   48  672 -528 -131  379  598  456  401  276 -397 -417  292
## [241]  261 -536  -15  139   91  123  217   70  634
```

(b) Pick out the values in yVec which are > 600.

```
vec_b <- yVec[yVec > 600]
print(vec_b)
```

```
##   [1] 945 636 775 853 695 665 619 811 694 777 917 715 863 771 739 708 607 876
##  [19] 989 814 902 749 999 628 873 666 835 980 835 698 993 673 915 647 765 916
##  [37] 891 998 824 833 873 834 826 677 990 852 672 943 619 902 647 750 892 937
##  [55] 887 868 652 680 920 668 827 727 876 918 914 753 828 964 774 828 671 730
##  [73] 698 993 861 913 722 990 960 910 907 832 666 818 841 940 673 938 834 755
##  [91] 720 646 767 966 946 877 798 646 964 744 604 781 905 975 859 994 615 855
## [109] 742 819 611 749
```

(c) What are the index positions in yVec of the values which are > 600?

```
which(yVec>600)
```

```
##   [1]   2   3   4   7   8  12  13  16  18  19  21  24  27  28  34  35  42  45
##  [19]  47  48  49  53  54  56  58  62  63  64  69  71  75  78  79  82  85  86
##  [37]  87  92  94  95  96  97  98 101 102 106 109 115 118 119 120 121 122 123
##  [55] 124 127 128 133 134 136 140 142 143 145 146 148 149 152 154 155 156 157
##  [73] 158 161 162 163 165 166 167 168 170 171 175 176 178 180 185 189 190 191
##  [91] 193 197 201 203 204 208 211 212 218 221 224 229 230 231 234 235 236 237
## [109] 241 242 244 250
```

(d) Sort the numbers in the vector xVec in the order of increasing values in yVec.

```
vec_d <- xVec[order(yVec)]
print(vec_d)
```

```
##   [1] 362 409 936 479 829 221 154 806 565 122 667 397 918 450 194 352 625 253
##  [19]  25 202 439 859 587  31 994 449 346 599  55 700 425  14 990 820 137 495
##  [37]  53 901 563 647  99 626 974 307 543  51 434 938 107 529 442  71 292 525
##  [55]  16 286 167 537 428 588 264 285 746 517 808 895 308 976 206  20 980 259
##  [73] 480 668 720  50 560 557 736 568 228 193 406 202 311 839 815 653 711 310
##  [91] 680 453 520 811 375 752 642 681 350  20 447 135  57 285  38 114 318 449
## [109] 861 302 199 112 696 946 304 636 755 115 522 576 972 696 687  96 356 200
## [127] 569  22 493 795 919 628 685 891  44 456 838 162 530 641 155 454 545 863
## [145] 673  25 915 159 509 834 747 243 235 835 614 516  34 458 312 289 569 710
## [163] 269 364 911 432 891 297 513 748 213 630 558 259 454 895 633 732 434 275
## [181] 728 125 391 941  21 857  54 710  11  26 751 936 390 614 741 519 189 887
## [199] 845 969 432 884 218 557 256 210 396 595 250 991 701 873 629 723 632 675
## [217] 520 336 790  73 303  18 512 863 748 219 810  88 557 257 502 282 739 376
## [235] 209 625 237  92 197 311 546  11 583 143 222 783  69 159 148 286
```

(e) Pick out the elements in yVec at index positions $1, 4, 7, 10, 13, \cdots$

```
vec_e <- yVec[seq(1, length(yVec), by = 3)]
print(vec_e)
```

```
##  [1]  87 775 853 514 619 811 777 524 266 771 252 739 203 449 451 376 902 239 263
## [20] 873 581 980 378 571 468 130 915 647 765 288  42 824 834 583  47 852 672  16
## [39] 943 619 750 887 868 105 680 668 550 727 918 753 233 774 730   8 913 990  36
## [58] 499 666 841 322  24 407 834 720 482 387 481 571 877 798 215 476 198 406 410
## [77] 781  18 994 486 742 611 173 749
```

3.

For this problem we'll use the (built-in) dataset state.x77.

```
data(state)
state.x77 <- as_tibble(state.x77, rownames  = 'State')
```

a. Select all the states having an income less than 4300, and calculate the average income of these states.

```
a <- state.x77 |> filter(Income < 4300)
mean(pull(a, Income))
```

```
## [1] 3830.6
```

b. Sort the data by income and select the state with the highest income.

```
state.x77 |> arrange(Income) |> tail(1)
```

```
## # A tibble: 1 x 9
##   State  Population Income Illiteracy `Life Exp` Murder `HS Grad` Frost   Area
##   <chr>       <dbl>  <dbl>      <dbl>      <dbl>  <dbl>     <dbl> <dbl>  <dbl>
## 1 Alaska        365   6315        1.5       69.3   11.3      66.7   152 566432
```

```
state.x77 |> summarise(max(Income))
```

```
## # A tibble: 1 x 1
##   `max(Income)`
##           <dbl>
## 1          6315
```

c. Add a variable to the data frame which categorizes the size of population: $<=$ 4500 is `S`, \$ $>$ 4500 \$ is `L`.

d. Find out the average income and illiteracy of the two groups of states, distinguishing by whether the states are small or large.

4.

a. Write a function to simulate `n` observations of (X_1, X_2) which follow the uniform distribution over the square $[0, 1] \times [0, 1]$.

```r
simulate_uniform_points <- function(n) {
  data.frame(
    X1 = runif(n, 0, 1),
    X2 = runif(n, 0, 1)
  )
}
simulate_uniform_points(100)
```

```
##              X1          X2
## 1   0.268085174 0.543229287
## 2   0.717793365 0.220809420
## 3   0.508191325 0.129244578
## 4   0.301141017 0.905568746
## 5   0.258901858 0.361476314
## 6   0.040634166 0.141698580
## 7   0.220868572 0.570115670
## 8   0.100161241 0.154920217
## 9   0.810061165 0.671183354
## 10  0.466554370 0.486041477
## 11  0.575955302 0.133173948
## 12  0.381156693 0.496785611
## 13  0.363996882 0.948546567
## 14  0.624570739 0.947350645
## 15  0.841561614 0.356259312
## 16  0.008412414 0.135911005
## 17  0.375058783 0.001529855
## 18  0.945216957 0.383157329
## 19  0.152458258 0.732789078
## 20  0.389374035 0.631023867
## 21  0.596612588 0.666452381
## 22  0.692512569 0.373256179
## 23  0.801463063 0.575240979
## 24  0.135444667 0.774531035
## 25  0.018632099 0.420657257
## 26  0.928753173 0.385743384
## 27  0.644141050 0.044558597
## 28  0.109655937 0.855839870
## 29  0.847226621 0.678719396
## 30  0.900637557 0.783330139
## 31  0.174322044 0.956216308
## 32  0.599562055 0.144265838
## 33  0.483525358 0.481873573
## 34  0.531860797 0.082286194
## 35  0.621338322 0.283505297
## 36  0.221170856 0.113996632
## 37  0.483836401 0.513389648
## 38  0.931518404 0.633692394
## 39  0.546898287 0.286872324
## 40  0.321169357 0.094804098
## 41  0.569301375 0.378368874
## 42  0.838803876 0.782000584
## 43  0.418920988 0.877371290
## 44  0.690395285 0.999500488
```

```
## 45  0.658945604 0.572461354
## 46  0.204922437 0.147355789
## 47  0.039271365 0.685838931
## 48  0.053084322 0.395108984
## 49  0.453245888 0.449129964
## 50  0.442142399 0.023711957
## 51  0.950831358 0.472277227
## 52  0.707531531 0.781226703
## 53  0.641743653 0.015176494
## 54  0.941452186 0.331391281
## 55  0.714796190 0.402174780
## 56  0.859887239 0.573320421
## 57  0.419478792 0.882892685
## 58  0.506420180 0.273187506
## 59  0.884771709 0.652806625
## 60  0.195964247 0.053565837
## 61  0.761117530 0.885458461
## 62  0.971144022 0.590081746
## 63  0.654653879 0.021490993
## 64  0.423207030 0.149057247
## 65  0.642259988 0.433674170
## 66  0.436404001 0.766440394
## 67  0.866220227 0.527418325
## 68  0.489090039 0.616851244
## 69  0.146335560 0.858308365
## 70  0.800508148 0.016253163
## 71  0.415381782 0.381282128
## 72  0.040873035 0.063229098
## 73  0.893902523 0.321784682
## 74  0.520334365 0.542991204
## 75  0.740471969 0.769495065
## 76  0.827872704 0.560154996
## 77  0.596531007 0.766257207
## 78  0.513014797 0.553913818
## 79  0.614918053 0.094985780
## 80  0.691493512 0.249334797
## 81  0.300231600 0.529035077
## 82  0.658447500 0.721064510
## 83  0.129927119 0.813719159
## 84  0.836862505 0.372095502
## 85  0.765802775 0.922872850
## 86  0.455735386 0.402829402
## 87  0.842363441 0.748055207
## 88  0.259472372 0.798965745
## 89  0.101000473 0.286923036
## 90  0.341237880 0.833872559
## 91  0.743928232 0.288467797
## 92  0.536488675 0.390793624
## 93  0.970062822 0.255896987
## 94  0.549122352 0.845122389
## 95  0.813687239 0.837896186
## 96  0.864921696 0.812178884
## 97  0.492250699 0.206792019
## 98  0.571753340 0.021447888
```

```
## 99  0.641138018 0.312551104
## 100 0.427457929 0.585089791
```

b. Write a function to calculate the proportion of the observations that the distance between (X_1, X_2) and the nearest edge is less than 0.25, and the proportion of them with the distance to the nearest vertex less than 0.25.

```r
calculate_proportions <- function(data) {
  # Distance to nearest edge
  d_edge <- pmin(data$X1, 1 - data$X1, data$X2, 1 - data$X2)
  prop_edge <- mean(d_edge < 0.25)

  # Distance to nearest vertex
  d1 <- sqrt(data$X1^2 + data$X2^2)                    # to (0, 0)
  d2 <- sqrt(data$X1^2 + (1 - data$X2)^2)              # to (0, 1)
  d3 <- sqrt((1 - data$X1)^2 + data$X2^2)              # to (1, 0)
  d4 <- sqrt((1 - data$X1)^2 + (1 - data$X2)^2)        # to (1, 1)
  d_vertex <- pmin(d1, d2, d3, d4)
  prop_vertex <- mean(d_vertex < 0.25)

  # Return result as named vector
  c(proportion_edge = prop_edge,
    proportion_vertex = prop_vertex)
}
calculate_proportions(simulate_uniform_points(100))
```

```
##   proportion_edge proportion_vertex
##              0.78              0.25
```

5.

To estimate π with a Monte Carlo simulation, we draw the unit circle inside the unit square, the ratio of the area of the circle to the area of the square will be $\pi/4$. Then shot K arrows at the square, roughly $K * \pi/4$ should have fallen inside the circle. So if now you shoot N arrows at the square, and M fall inside the circle, you have the following relationship $M = N * \pi/4$. You can thus compute π like so: $\pi = 4 * M/N$. The more arrows N you throw at the square, the better approximation of π you'll have.

```r
n <- 10000

set.seed(1)
points <- tibble("x" = runif(n), "y" = runif(n))
```

Now, to know if a point is inside the unit circle, we need to check whether $x^2 + y^2 < 1$. Let's add a new column to the points tibble, called `inside` equal to `1` if the point is inside the unit circle and `0` if not:

```r
points <- points |>
    mutate(inside = map2_dbl(.x = x, .y = y, ~ifelse(.x**2 + .y**2 < 1, 1, 0))) |>
    rowid_to_column("N")
```

a. Compute the estimation of π at each row, by computing the cumulative sum of the 1's in the `inside` column and dividing that by the current value of `N` column:

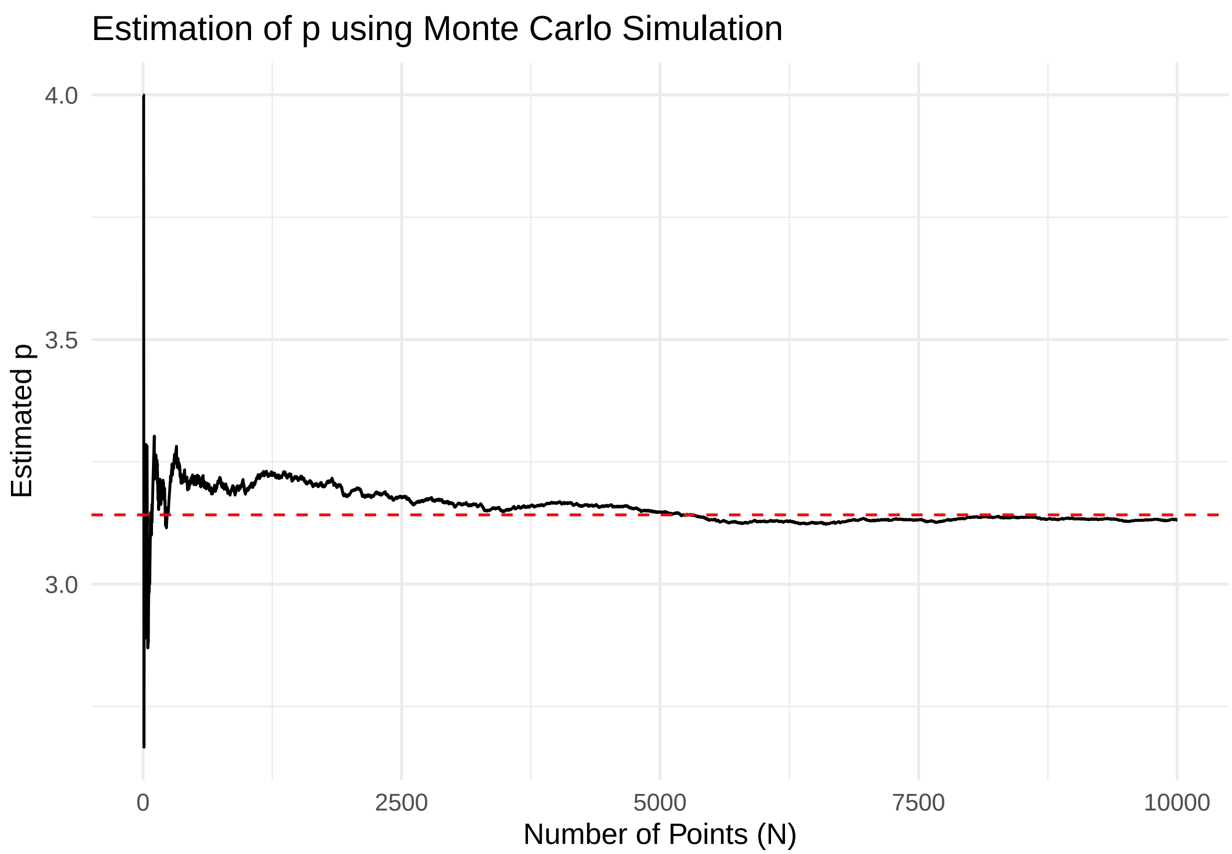
```r
library(tidyverse)


points_with_pi <- points %>%
  mutate(
    cum_inside = cumsum(inside),
    pi_estimate = 4 * cum_inside / N
  )
tail(points_with_pi)
```

```
## # A tibble: 6 x 6
##       N     x     y inside cum_inside pi_estimate
##   <int> <dbl> <dbl>  <dbl>      <dbl>       <dbl>
## 1  9995 0.244 0.691      1       7824        3.13
## 2  9996 0.732 0.788      0       7824        3.13
## 3  9997 0.499 0.814      1       7825        3.13
## 4  9998 0.503 0.478      1       7826        3.13
## 5  9999 0.568 0.602      1       7827        3.13
## 6 10000 0.653 0.111      1       7828        3.13
```

b. Plot the estimates of π against `N`.

```r
ggplot(points_with_pi, aes(x = N, y = pi_estimate)) +
  geom_line() +
  geom_hline(yintercept = pi, color = "red", linetype = "dashed") +
  labs(title = "Estimation of   using Monte Carlo Simulation",
       x = "Number of Points (N)",
       y = "Estimated  ") +
  theme_minimal()
```



6.

Mortality rates per 100,000 from male suicides for a number of age groups and a number of countries are given in the following data frame.

```
suicrates <- tibble(Country = c('Canada', 'Israel', 'Japan', 'Austria', 'France', 'Germany',
'Hungary', 'Italy', 'Netherlands', 'Poland', 'Spain', 'Sweden', 'Switzerland', 'UK', 'USA'),
Age25.34 = c(22,  9, 22, 29, 16, 28, 48,  7,  8, 26,  4, 28, 22, 10, 20),
Age35.44 = c(27, 19, 19, 40, 25, 35, 65,  8, 11, 29,  7, 41, 34, 13, 22),
Age45.54 = c(31, 10, 21, 52, 36, 41, 84, 11, 18, 36, 10, 46, 41, 15, 28),
Age55.64 = c(34, 14, 31, 53, 47, 49, 81, 18, 20, 32, 16, 51, 50, 17, 33),
Age65.74 = c(24, 27, 49, 69, 56, 52, 107, 27, 28, 28, 22, 35, 51, 22, 37))
```

a. Transform `suicrates` into *long* form.

```
library(tidyverse)

#    long
suicrates_long <- suicrates %>%
  pivot_longer(
    cols = starts_with("Age"),
    names_to = "AgeGroup",
    values_to = "Rate"
  )
```

```r
head(suicrates_long)
```

```
## # A tibble: 6 x 3
##   Country AgeGroup  Rate
##   <chr>   <chr>    <dbl>
## 1 Canada  Age25.34    22
## 2 Canada  Age35.44    27
## 3 Canada  Age45.54    31
## 4 Canada  Age55.64    34
## 5 Canada  Age65.74    24
## 6 Israel  Age25.34     9
```

b. Construct side-by-side box plots for the data from different age groups, and comment on what the graphic tells us about the data.

```r
ggplot(suicrates_long, aes(x = AgeGroup, y = Rate)) +
  geom_boxplot() +
  labs(title = "Male Suicide Rates by Age Group",
       x = "Age Group",
       y = "Suicide Rate per 100,000") +
  theme_minimal() +
  coord_flip()
```