Note: Code is provided in both Matlab and R. Or rather, Matlab code has been ported to R. The Matlab code is heavily commented, but the R code has relatively few comments.

General comments:

⋆ Please save your code in one file

⋆ and indent your code as well.

⋆ Also try to give a brief (?) justification for questions, instead of a yes/no answer unless the justification is trivial can be given in one sentence\(^1\).

1. Monahan, 2.9

(a) Yes, the computed average can be smaller. \([1]\]

(b) Some examples:

i. A numerical example (at least for R):

\[
\begin{align*}
x &= 2^{53} + 1 \\
y &= 2^{53} + 2
\end{align*}
\]

\[
(x+y)/2 == 2^{53}
\]

1TRUE

ii. A not so numerical example. Let \(\epsilon\) be the machine epsilon of R / Matlab / etc, and let \(x, y, z\) be floating point numbers such that \(x = y = z + \frac{\epsilon}{2}\). Then:

\[
\frac{x + y}{2} = \frac{2z}{2} = z < z + \frac{\epsilon}{2}
\]

iii. The standard example. With two significant digits, consider \(\frac{51 + 51}{2} = 50\).

Any example which:

i. works \([4]\]

ii. and where the significant digits are stated if it’s not 2 (see comments) \([2]\)

will give you full credit. \([4+2 = 6]\]

(c) It is not possible in base 2. Here is a sketch proof (using 2 significant digits). Suppose we keep the same exponentiation. Looking at the combination of pairs (00, 01, 10, 11, 100, 110), we note that any \(\frac{x+y}{2}\) is never less than any \(x\) or \(y\). Now, if we have different exponentiation of \(x\) than \(y\), eg \(x \geq y\), we must have \(\frac{x+y}{2} \geq y\).

\(^1\)but then you might as well write the sentence.
The breakdown is as follows:

i. Impossible in base 2
   and either:
   A. explain why we can narrow it down to a few possibilities
      and enumerate these possibilities
   B. Or give a convincing proof

\[ 1+1+1 = 3 \quad \text{or} \quad 1+2 = 3 \]

Grand Total: \[ 1 + 6 + 3 = 10 \]

Comments:

- Try not to miss out each part of the question (especially part a).
- Should prove part c) instead of giving a yes / no.
- Some of you gave examples without stating what significant digits you are using. This is relatively (somewhat) okay since the question says you can use two digits, but becomes rather egregious when your example isn’t using two digits of significance. So I chopped off two digits from your score\(^2\).

\(^2\)Aren’t you happy the question didn’t ask for 6 significant digits?
2. Monahan, 2.12

*After Sept 13 corrections.* We want to compute $H_n$ until the value no longer changes with $n$. As there is no function in Matlab like `signif`, code will only be given in R here.  

R code:

```r
NewHS <- function()
{
  # Monahan 2.12
  
  # Note that R isn’t like Matlab where we enter
  # $H_n[1] = 1$ ; we will get an error message since we
  # didn’t define $H_n$.
  
  # We could however do $H_n = c(1,1.5)$, and then $H_n[3] = 0.3$
  # etc wouldn’t be a problem.
  
  # If we wanted $H_n[1] = 1$, we’d have to define $H_n$, so we can
  # have
  
  Hn = vector("numeric")
  Hn[1] = 1 # no error here!
  Hn[2] = 1.5

  n = 2

  while (Hn[n] != Hn[n-1])
  {
    n = n+1
    Hn[n] = signif(Hn[n-1] + signif(1/n,4),4)
  }
  return(list(Hn = Hn[n], n = n))
}
```

with output

```r
$Hn
 [1] 8.446

$n
 [1] 2000
```
Now, we want to compare this (these) value(s) to the stopping point with what can be guessed analytically, by equating $\epsilon_m = (n + 1)^{-1}/H_n$.

Thus, we can write:

$$\epsilon_m = \frac{1}{n+1} H_n = \frac{H_{n+1} - H_n}{H_n}$$

and thus:

```r
NewHSAn<-function(epsilon)
{
    # Monahan 2.12
    # Finding H_n analytically
    Hn = vector("numeric")
    Hn[1] = 0.5772
    Hn[2] = signif(0.5772 + signif(log(2),4) ,4)
    n = 2
    while (((1/(n+1))/ Hn) > epsilon)
    {
        n = n+1
        Hn[n] = signif(0.5772 + signif(log(n),4),4)
    }
    return(list(Hn = Hn[n], n = n))
}
```

with output:

```r
> NewHSAn(0.0005)
$Hn
 [1] 6.33

$n
 [1] 315
```
Breakdown:

(a) Calculating $H_n$ until function no longer changes with $n$
   i. Either the value of $n$ or $H_n$. [1]
   ii. Code to find $H_n$. [8]
   iii. Not using `signif` always (see comments), which distorts answers. [-1]
   iv. Code is excessively inefficient. [-1]
   v. Any other errors.
   * Total: [1+8 = 9]

(b) Finding $n$ (or $H_n$) analytically
   i. Either the value of $n$ or $H_n$. [1]
   iii. “Simplified” too early (see comments), or did not always use `signif` and wasn’t penalized in previous part. [-1]
   iv. Code is excessively inefficient. [-1]
   v. Any other errors.
   * Total: [1+4 = 5]
   * Grand Total: [9 + 5 = 14]

Comments:

- There are several interpretations of this question, but alas, iii) is the correct one.
  i. Without placing any restrictions on $n$, eg, so $\frac{1}{3} = 0.\overline{3}$ instead of 0.3333 (4sf), find $n$ (and $H_n$) such that $H_{n+1} - H_n < 0.0001$.
  ii. Placing restrictions on $n$ to 4sf, eg, $\frac{1}{3} = 0.3333$, find $n$ (and $H_n$) such that $H_{n+1} = H_n$.
  iii. Placing restrictions on $n$ and $H_n$ to be 4sf, find $n$ (and $H_n$) such that $H_{n+1} = H_n$.

However, for those who have done i) or ii), here are outputs for them.

For i), we have $H_n = 9.7876$ and $n = 10000$.
For ii), we have $H_n = 9.3668$ and $n = 10000$.

- Note that you get a different answer if you solve:

$$\epsilon_m = \frac{H_{n+1}}{H_n} - 1$$

and that gives $n = 262$.

\[\text{perhaps a more fitting word might be mistake(?)\}}\]
2. Monahan, 2.12

*Before Sept 13 corrections.* While the question has been modified, the ‘old’ solution will be kept here, but take a look at the second function\(^5\)

Calculating \(H_n\) until the value no longer changes with \(n\) will take a terribly long time (possibly a few weeks) to do. However, here is (naive) code in Matlab and R to find the value of \(H_n\).

(a) ...Matlab...

```matlab
function [HS,i] = HS()

    \% Monahan 2.12
    \% To compute H_n until the value no longer changes with n
    tic
    HS = 1;
    check = -1;
    i = 2;

    while HS > 0;
        HS = HS + 1/i;
        if check == HS;
            disp(['We have convergence at H_n = ', num2str(HS), ' with i = ', num2str(i)]);
            toc
            return;
        end
        check = HS;
    end
    i = i +1;
    \% Sometimes, instead of printing out every iteration so you know
    \% the function isn’t in an infinite loop, you might want to print
    \% out every k^th iteration, particularly since printing every
    \% iteration leads to (greater) computational time.
    if mod(i,1000000) == 0
        i
    HS
    end
    check = HS;
end
end
```

(b) ...R...

---

\(^5\) There is a Google interview question which goes like this: You are at an (infinite) parking lot over the integers, and you start at \{0\}. Your car is parked somewhere along the integers, and you have forgotten where your car is. You can move +1 or −1 position at a time. Give an algorithm to find your car in the least amount of time. Finding your car means stepping into the ‘integer lot’ that has your car.
HS <- function()
{
  # Monahan 2.12
  # To compute $H_n$ until the value no longer changes with n
  tic = proc.time()[3]
  HS = 1
  check = -1
  i = 2
  while (HS > 0)
  {
    HS = HS + 1/i
    if (check == HS)
    {
      toc = proc.time()[3] - tic
      return(list(HS = HS, i = i, timetaken = toc))
    }
    i = i + 1
    if (i %% 1000000 == 0)
    {
      print(i)
      print(HS)
    }
  }
  check = HS
}

Thankfully, here is a (more) computationally efficient way of finding the value of $H_n$ where it converges, given by [Knu97], page 160. The below text is taken almost verbatim from the book.

We define $r_k(x)$ to be the number $x$ rounded to $k$ decimal places. So for example, we have for $k = 1$:

$$
\text{"H}_{n(1)} = \sum_{i=1}^{\infty} r_1(1/i) = 1 + 0.5 + 0.3 + 0.3 + 0.2 + 0.2 + 0.1 + \ldots + 0.1 = 3.9
$$

So we can fix $k$ to be the number of decimal places (depending on machine epsilon), and follow the algorithm:

(a) Set $x_h = 1$, $S = 1$.

(b) Set $x_e = x_h + 1$, and find $r_k(\frac{1}{x_e}) = r$.

(c) Find $x_h$, the largest $x$ for which $r_k(\frac{1}{x}) = r$. 
(d) Set $S = S + (x_h - x_e + 1)$, and repeat ii until convergence.

Thus, here are functions ...

(a) ... in Matlab

We first create the following function to do step iii in the above algorithm.

```matlab
function [startval, endval, searchpara, done] = FindRange(startval, k, searchpara)
    % Monahan 2.12
    % This function takes in an input, startval and k and finds the value
    % $r_k(1/startval)$. It then gives an output endval such that where
    % $x \in [startval, endval]$, we have $r_k(1/x)$ to be the same value.
    % The input searchpara will define the region where we search.
    valuetomatch = floor(10^k*(1/startval) + 0.5)/10^k;
    stop = 0;
    % The below snippet will do the following:
    % Check if $r_k(1/(startval + searchpara))$ has the same value as our
    % value to match. If yes, we need to widen our search parameter, and
    % we will double it. If it is lesser than our value to match, then
    % the range of values we need to check ranges from
    % $[startval, startval + searchpara]$
    yes = 0;
    while stop ~= 1;
        checkval = floor(10^k*(1/(startval+searchpara)) + 0.5)/10^k;
        if checkval < valuetomatch
            stop = 1;
        elseif checkval == valuetomatch
            yes = 1;
            searchpara = searchpara * 2;
        elseif checkval > valuetomatch
            % This should never happen, but just to prevent infinite loops
            error('Error!')
            return;
        end
    end
    % Keep the smaller one for use next time we run this function, if we
    % multiplied search para by 2
    if yes == 1
        searchpara = searchpara/2;
    end
```
% Now, we need to find the range of values, and we will need to
% search through LB+1, LB+2, ... UB-1, UB
LB = searchpara;
UB = searchpara*2;

done = 0;
stop = 0;
while stop ~=1;
    % Check cases
    checkvalUB1 = floor(10^k*(1/(startval+UB)) + 0.5)/10^k;
    checkvalUB2 = floor(10^k*(1/(startval+UB-1)) + 0.5)/10^k;
    if checkvalUB2 == valuetomatch && checkvalUB1 < valuetomatch
        stop = 1;
        endval = startval+UB-1;
        if checkvalUB1 == 0;
            done = 1;
        end
    end
    checkvalLB1 = floor(10^k*(1/(startval+LB)) + 0.5)/10^k;
    checkvalLB2 = floor(10^k*(1/(startval+LB-1)) + 0.5)/10^k;
    if checkvalLB2 == valuetomatch && checkvalLB1 < valuetomatch
        stop = 1;
        endval = startval+LB-1;
        if checkvalLB1 == 0;
            done = 1;
        end
    end
    % else
    tmp = floor((LB+UB)/2);
    checkval = floor(10^k*(1/(startval+tmp)) + 0.5)/10^k;
    if checkval == valuetomatch
        LB = tmp;
    elseif checkval < valuetomatch
        UB = tmp;
    end
end
end

This isn’t the most efficient function to find the range of values, but it already provides a massive speed-up than in the first (naive) function. We then can use the following function:

function [HS,startval] = FindHS()

% Monahan 2.12
tic
startval = 2;
searchpara = 1;
HS = 1;
k = 17;

% Note: k = 17 is what we will use. To check we can easily do
% floor(10^(17)*1/3 + 0.5)/10^(17) == floor(10^(18)*1/3 + 0.5)/10^(18)

stop = 0;
i = 0;

while stop ~=1
    i = i + 1;
    [startval, endval, searchpara, done] = FindRange(startval,k,searchpara);
    HS = HS + (endval - startval + 1)* floor(10^k*(1/(startval))
        + 0.5)/10^k;
    startval = endval + 1;
    if mod(i,1000000) == 0
        HS
        startval
        searchpara
        % save the values, just in case
        % One good reason for saving is that when startval and
        % searchpara are relatively large, it takes longer for each
        % 1000000th iteration. So if you are easily flustered and
        % unsure whether Matlab (or R) is hanging, you can just
        % stop the function, change the parameters, and load in
        % the current HS, endval, and searchpara.
        fname = 'harmonic';
        feval('save',fname,'HS','endval','searchpara');
    end
    if done == 1
        stop = 1;
        toc
    end
end
end

This should take about two hours (roughly) to run, and converges at \( \approx 37.3705 \). Interestingly, for extremely high values of \( k \), the first function will go into an infinite loop unless some modification is done.
(b) ...in R

The same two functions are included here.

```r
findrange<-function(startval,k,searchpara)
{
  valuetomatch = floor(10^-k*(1/startval) + 0.5)/10^-k
  stop = 0
  yes = 0

  while (stop != 1)
  {
    checkval = floor(10^-k*(1/(startval+searchpara)) + 0.5)/10^-k
    if (checkval < valuetomatch)
    {
      stop = 1
    }
    else if (checkval == valuetomatch)
    {
      yes = 1
      searchpara = searchpara * 2
    }
    else if (checkval > valuetomatch)
    {
      stop('Error!')
    }
  }

  if (yes == 1)
  {
    searchpara = searchpara/2
  }

  LB = searchpara
  UB = searchpara*2

  done = 0;
  stop = 0;
  while (stop !=1)
  {
    checkvalUB1 = floor(10^-k*(1/(startval+UB)) + 0.5)/10^-k
    checkvalUB2 = floor(10^-k*(1/(startval+UB-1)) + 0.5)/10^-k
    if (checkvalUB2 == valuetomatch && checkvalUB1 < valuetomatch)
    {
      stop = 1
      endval = startval+UB-1
    }
  }
```
if (checkvalUB1 == 0)
{
    done = 1
}

checkvalLB1 = floor(10^k*(1/(startval+LB)) + 0.5)/10^k
checkvalLB2 = floor(10^k*(1/(startval+LB-1)) + 0.5)/10^k
if (checkvalLB2 == valuetomatch && checkvalLB1 < valuetomatch)
{
    stop = 1
    endval = startval+LB-1
    if (checkvalLB1 == 0)
    {
        done = 1
    }
}
tmp = floor((LB+UB)/2)
checkval = floor(10^k*(1/(startval+tmp)) + 0.5)/10^k
if (checkval == valuetomatch)
{
    LB = tmp
}
else if (checkval < valuetomatch)
{
    UB = tmp
}
return(list(startval = startval, endval = endval, searchpara = searchpara, done = done))

and:

findHS<-function()
{
    tic = proc.time()[3]
    startval = 2
    searchpara = 1
    HS = 1
    k = 17
    stop = 0
    i = 0

    while (stop !=1)
    {

\begin{verbatim}

i = i + 1
results <- findrange(startval,k,searchpara)
startval = results$startval
endval = results$endval
searchpara = results$searchpara
done = results$done
HS = HS + (endval - startval + 1)* floor(10^k*(1/(startval))
     + 0.5)/10^-k
startval = endval + 1
if (i %% 1000000 == 0)
{
    print(HS)
    print(startval)
    print(searchpara)
    save(HS,startval,searchpara, file = "results.rData")
}
if (done == 1)
{
    stop = 1;
    toc = proc.time()[3]
}
}
return(list(HS = HS, startval = startval, totaltime=toc))


\end{verbatim}

Note that this ported version of the R code runs much slower than its Matlab equivalent, and isn’t optimized.
Comparing this to the stopping point analytically is the same as before (just note the change in sf), and thus is omitted.
3. Monahan, 2.14

(a) When \( t \to \infty \), cancellation will be serious, since \( \frac{1}{t} \approx \frac{1}{\sqrt{t^2 + a}} \) 

We can write:

\[
\frac{1}{t^2} - \frac{1}{\sqrt{t^2 + a}} = \frac{\sqrt{t^2 + a} - t}{t\sqrt{t^2 + a}} = \frac{\sqrt{t^2 + a} - t}{t\sqrt{t^2 + a}} \cdot \frac{\sqrt{t^2 + a} + t}{\sqrt{t^2 + a} + t} = \frac{a}{t(t^2 + a) + t^2\sqrt{t^2 + a}}
\]

(b) We have two cases:

i. \( t \to 0 \), since both terms will be \( \approx 1 \).

ii. \( t \to \pm \infty \) since both terms \( \approx 0 \)

For both, we can write:

\[
e^{-2t^2} - e^{-8t^2} = e^{-2t^2}(1 - e^{-t^2})(1 - e^{-t^2})(1 + e^{-2t^2} + e^{-4t^2})
\]

(c) When \( t \to \infty \), then \( e^{t+s} \approx e^t \)

We can write:

\[
\log(e^{t+s} - e^t) = \log(e^{t}(e^s - 1)) = \log e^t + \log(e^s - 1) = t + \log(e^s - 1)
\]

(d) When \( t \to \infty \), we have \( (1 + e^{-t})^2 \approx 1 \).

We can write:

\[
[1 + e^{-t}]^2 e^{-t} = [1 + 2e^{-t} + e^{-2t} - 1] t^2 e^{-t} = (2e^{-t} + e^{-2t})(t^2 e^{-t}) = 2t^2 e^{-2t} + t^2 e^{-3t}
\]
Comments:

• What was required for this question was a where does $t$ tend to for there to be catastrophic cancellation, so plots etc weren’t needed. Those that went the extra mile had some extra credit (+4)
4. Monahan, 2.19

We are given Samuelson’s inequality:

$$|x_i - \bar{x}| \leq s\sqrt{n-1}$$  \hspace{1cm} (†)

and we want to bound the cancellation:

$$\sum_{i=1}^{n} (x_i - \bar{x})^2 = \sum_{i=2}^{n} (x_i - x_1)^2 - n(x_1 - \bar{x})^2$$

by giving a bound on the ratio:

$$n(x_1 - \bar{x})^2 / \sum (x_i - x_1)^2$$

Note that we want something better than a simple upper bound, since we have:

$$\frac{n(x_1 - \bar{x})^2}{\sum (x_i - x_1)^2} = \frac{n(x_1 - \bar{x})^2}{\sum_{i=1}^{n} (x_i - \bar{x})^2 + n(x_1 - \bar{x})^2} \leq \frac{n(x_1 - \bar{x})^2}{n(x_1 - \bar{x})^2} = 1$$

which this doesn’t require (†) at all. But we can write:

$$\frac{n(x_1 - \bar{x})^2}{\sum (x_i - x_1)^2} = \frac{n(x_1 - \bar{x})^2}{\sum_{i=1}^{n} (x_i - \bar{x})^2 + n(x_1 - \bar{x})^2}$$

$$= \frac{1}{\frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n(x_1 - \bar{x})^2} + 1}$$

$$= \frac{1}{\frac{ns^2}{n(x_1 - \bar{x})^2} + 1}$$

$$\leq \frac{1}{\frac{ns^2}{ns^2(n-1)} + 1}$$

$$= \frac{1}{\frac{n-1}{n} + 1}$$

$$= \frac{n-1}{n}$$

Thus we have:

$$n(x_1 - \bar{x})^2 / \sum (x_i - x_1)^2 \leq \frac{n-1}{n}$$
Breakdown:

- Something worse than an upper bound of 1 \[ \leq 2 \]
- (Only) an upper bound of 1 (depending on presentation) \[ \leq 5 \]
- An attempt at a proof + running simulations to guess an upper bound \[ \leq 7 \]
- A proof showing an upper bound of the form \( \frac{n-1}{n} \) or \( \frac{n}{n+1} \) \[ 10 \]
- Minor errors in proof which shows the correct UB \[-1 \]
- * Extra Credit: State the lower bound \[ 1 \]

Comment: It’s a bit ambiguous which (sample?) variance we’re looking at, since we can have \( s^2 = \frac{1}{n} \sum (x_i - \bar{x})^2 \) or \( s^2 = \frac{1}{n-1} \sum (x_i - \bar{x})^2 \). You’d get the same bound anyway.
5. Monahan, 3.2

One good way of writing an algorithm is to try out a (small) test case. So for example, if we had \( L_{5 \times 5} = (a_{ij}) \) and \( k = 3 \), we would have to solve:

\[
L x = \begin{pmatrix}
  a_{11} & 0 & 0 & 0 & 0 \\
  a_{21} & a_{22} & 0 & 0 & 0 \\
  a_{31} & a_{32} & a_{33} & 0 & 0 \\
  a_{41} & a_{42} & a_{43} & a_{44} & 0 \\
  a_{51} & a_{52} & a_{53} & a_{54} & a_{55}
\end{pmatrix} \begin{pmatrix}
  x_1 \\
  x_2 \\
  x_3 \\
  x_4 \\
  x_5
\end{pmatrix} = \begin{pmatrix}
  a_{11}x_1 \\
  a_{21}x_1 + a_{22}x_2 \\
  a_{31}x_1 + a_{32}x_2 + a_{33}x_3 \\
  a_{41}x_1 + a_{42}x_2 + a_{43}x_3 + a_{44}x_4 \\
  a_{51}x_1 + a_{52}x_2 + a_{53}x_3 + a_{54}x_4 + a_{55}x_5
\end{pmatrix} = \begin{pmatrix}
  0 \\
  0 \\
  1 \\
  0 \\
  0
\end{pmatrix} = e_3
\]

This naturally leads to the following reasoning.

(b) The solution vector \( x = (x_1, \ldots, x_n)^T \) will have \( x_1 = x_2 = \ldots = x_{k-1} = 0 \). [4]
(c) But if these values of the solution vector are 0, then we necessarily do not need the first \( k - 1 \) columns of \( L \), since \( \sum_{i=1}^{k-1} a_{ni}x_i = 0 \). [4]

(a) We can now try to come up with an algorithm...

i. ... in Matlab

    function [xvec] = mon32(L,k)

    % Monahan 3.2
    % An algorithm which solves for x in Lx = e_k

    % We first check for square matrix
    if size(L,1) ~= size(L,2);
        error('L is not a lower triangular matrix!');
    end

    % We check that L is lower triangular
    % Could use istril(L), but doesn't work for all Matlab versions
    if all(all(tril(L) == L)) ~= 1
        error('L is not a lower triangular matrix!');
    end

    % Can't be solved if L(k,k) = 0
    if L(k,k) == 0
        error('Unable to solve for x with this L!');
    end

    % Good to go now

    dim = size(L,2);
    xvec = zeros(dim,1);
    xvec(k) = 1/L(k,k);
if k < dim;
    for i = (k+1):dim
        if L(i,i) == 0
            % Still possible to solve even if some off diagonals are zero
            if L(i,k:(i-1)) * xvec(k:(i-1)) ~= 0
                error('Unable to solve for x with this L!');
                return;
            else
                xvec(i) = xvec(i-1);
            end
        else
            vecsum = -dot(L(i,k:(i-1)),xvec(k:i-1));
            xvec(i) = vecsum/L(i,i);
        end
    end
end

ii. ... in R

mon3.2<-function(L,k)
{
    # Monahan 3.2
    # An algorithm which solves for x in Lx = e_k

    if( nrow(L) != ncol(L))
    {
        stop('L is not a lower triangular matrix!')
    }

    Lcheck = L
    upper.tri(L,diag = FALSE)
    Lcheck[upper.tri(L,diag = FALSE)] = 0
    if( identical(L,Lcheck) != 1)
    {
        stop('L is not a lower triangular matrix!')
    }

    if( L[k,k] == 0)
    {
        stop('Unable to solve for x with this L!')
    }

    size = nrow(L)
    xvec = mat.or.vec(size,1)
    xvec = as.matrix(xvec)
xvec[k] = 1/L[k,k]

if (k == size)
{
    return(xvec)
}

for (i in (k+1):size)
{
    if ((L[i,i] == 0))
    {
        if (L[i,k:(i-1)] %*% xvec[k:(i-1)] != 0)
        {
            stop('Unable to solve for x with this L!'
        }
        else
        {
            xvec[i] = xvec[i-1]
        }
    }
    else
    {
        sum = -crossprod(L[i,k:(i-1)], xvec[k:(i-1)]);
        xvec[i] = sum/L[i,i]
    }
    return(xvec)
}

Let’s test the functions out:

i. in Matlab
L = tril(normrnd(0,1,20,20))
evec = zeros(20,1)
evec(10) = 1
xvec = mon32(L,10)

L * xvec
all(L * xvec == evec)

>1  %(?)
>0  %(?)

Running the above snippet repeatedly, we do get 1 some of the time (i.e. they are identical). But for big matrices we probably won’t. Still, it’s a nice way to test your code with small matrices. To modify it for large matrices, you
could check:

\[
\text{norm}(L \ast xvec - evec, 2)
\]
and see how close \(\|Lx - e_k\|_2\) is to 0.

ii. in R

\[
L = \text{mat.or.vec}(20,20)
\]

\[
L[\text{lower.tri}(L, \text{diag} = \text{TRUE})] = \text{rnorm}(210, \text{mean}=0, \text{sd}=1)
\]

\[
k = 10
\]

\[
evec = \text{mat.or.vec}(20,1)
\]

\[
evec = \text{as.matrix}(evec)
\]

\[
evec[k] = 1
\]

\[
xvec = \text{mon3.2}(L,10)
\]

\[
\text{identical}(L \ast\!\!\ast xvec, evec)
\]

\[
> \[1\] \text{TRUE} #(?)
\]

\[
> \[1\] \text{FALSE} #(?)
\]

Same with above - we do get TRUE some of the time as well. Correspondingly, we can also use:

\[
\text{norm}(L \ast\!\!\ast xvec - evec, \text{type} = ~"2")
\]

(d) We want an algorithm to find \(L^{-1}\) from \(L\), and we hope we can use the previous parts to help us do this. Again, consider a small test case, say with \(L = 3\). With an abuse of notation, we have:

\[
LL^{-1} = \begin{pmatrix}
a_{11} & 0 & 0 \\
a_{21} & a_{22} & 0 \\
a_{31} & a_{32} & a_{33}
\end{pmatrix}
\begin{pmatrix}
a_{11}^{-1} & 0 & 0 \\
a_{21}^{-1} & a_{22}^{-1} & 0 \\
a_{31}^{-1} & a_{32}^{-1} & a_{33}^{-1}
\end{pmatrix}
= \begin{pmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{pmatrix}
= I
\]

and realize that \(LL^{-1}_i = e_i\), where \(L^{-1}_i\) denotes the \(i^{\text{th}}\) column of \(L^{-1}\), and \(e_i\) the \(i^{\text{th}}\) basis vector. So we can reuse our algorithm in part a) and overwrite \(L\) with \(L^{-1}\) because of part c).

Thus, here is code:

i. in Matlab

\[
\text{function [L] = ConvertLtoInverse(L)}
\]

\[
\% Monahan 3.2
\]

\[
\% An algorithm which converts a lower triangular matrix L to its inverse
\]

\[
\% Can't be solved if any diag value = 0
\]

\[
\text{if sum(diag(L) == 0) \text{~} = 0}
\]
error('Unable to solve for x with this L!');
return;
end

for i = 1:length(L);
    L(:,i) = mon32(L,i);
end

end

ii. in R
L2inverse<-function(L)
{
    # Monahan 3.2
    # An algorithm which converts a lower triangular matrix L
    # to its inverse

    if (sum(as.integer(diag(L) == 0)) != 0)
    {
        stop('Unable to solve for x with this L!')
    }

    for (i in 1:nrow(L))
    {
        L[,i] = mon3.2(L,i)
    }
    return(L)
}

Let’s test the functions out:
i. in Matlab
L = tril(normrnd(0,1,20,20))
Linv = ConvertLtoInverse(L)
all(all(L*Linv == eye(20))) % sadly, you won’t get 1
norm(L * Linv - eye(20), 2)

We can use the same idea as above (check norms), but I’m okay with someone
printing out I given by $LL^{-1}$ or $L^{-1}L$, or printing out 0 in $L^{-1} - L^{-1} = 0$
for $L^{-1}$ given by another algorithm.

ii. in R
L = mat.or.vec(20,20)
L[lower.tri(L,diag = TRUE)] = rnorm(210,mean=0,sd=1)
Linv = L2inverse(L)
identical(L%*%Linv, diag(x=20)) *Won’t get TRUE
norm(L %*% Linv - diag(20), type = "2")
Output should ideally give **TRUE**, but doesn’t happen. But could check matrix norms.

**Breakdown:**

(a) Part a)
   i. Presence of code which computes $x$ for fixed $L, k$. $\leq 2$
   ii. Presence of code which computes most $x$ for any $L, k$. $[5]$
   iii. Failure to account for some $L$ (see comments). $[-1]$
   iv. Other minor mistakes in code. $[-1]$

(b) Part b) $[4]$

(c) Part c) $[4]$

(d) Part d)
   i. Presence of code which computes $L^{-1}$ for fixed rows and columns. $[\leq 2]$
   ii. Presence of code which computes $L^{-1}$. $[5]$
   iii. Not overwriting $L$ with $L^{-1}$. $[-1]$
   iv. Other minor mistakes in code. $[-1]$

(e) Testing of code on generated matrix in question.
   i. Testing for part a) with $L_{20 \times 20}$ and $k = 10$ by showing that the $x$ you get satisfies $Lx = e_k$ (or similar). $[1]$
   ii. Testing for part d) with $L_{20 \times 20}$ and showing that $LL^{-1} = I_{20}$ (or similar). $[1]$
   iii. Either:
      A. Not showing output (you can show output by printing - though not recommended in future) and giving a one liner like “I tested my code and it works, honest” for each question. $[-1]$
      B. Asking the grader to compare matrices. (This is subjective so see comments below) $[-1]$
      C. Code error in generating lower triangular matrix. $[-1]$

* Extra credit: Not printing output but instead showing that your functions work - eg by considering matrix norm. $[2]$
* Extra credit: (many) checks (and stops) to test for errors. $[2]$
Comments:

- A gentle entreaty: Please indent your code. Reading unindented code is like reading this entire document in this font.

- Try not to use loops in Matlab or R - this really slows down computational time. If you have two vectors, say:

\[ x = (x_1, x_2, \ldots, x_n) \quad a = (a_1, \ldots, a_n) \]

and you want to find \( \sum a_i x_i \), note this is the same as \( x \cdot a = x^T a \).

- On that note, instead of using \( t(x) \%*\% a \) in R, it is actually much faster to use \( \text{crossprod}(x \%*\% a) \).

- We can’t throw out (all) lower triangular matrices with some 0s on the diagonal, because some of them are still solvable. To test if your code in part a) fulfills requirements, I tested it first on an arbitrary 3 by 3 lower triangular matrix, and then with:

\[
L = \begin{pmatrix}
1 & 0 & 0 \\
2 & 3 & 0 \\
4 & 5 & 6 \\
7 & 5 & 6 \\
\end{pmatrix}
\quad \text{and} \quad k = 1, 2, 3, 4
\]

to check for errors. Remember - just because your code works with the “standard” case doesn’t mean it’ll work for all cases.

You should have this habit of testing your code on small cases.

- Usually, to check if a coded algorithm works, it’s nice to try to break it. Two ways of breaking an algorithm (at least for this question). If you have a lot of if and elseif / elif / else if statements, are all these conditions accounted for? Second way - if you are dividing by anything, will there ever be a case where you divide by zero?

- Regarding output - I’m a okay for output to be printed out for this assignment. Most of the time though, you should write Matlab or R code to actually check if two matrices (or vectors) are similar instead. I have deducted marks if I see something like: “This is \( L^{-1} \) generated by another algorithm, this is \( L^{-1} \) from my algorithm, they are equal.” I doubt you checked all entries of \( L^{-1} \). But if you printed out a zero matrix or the identity matrix, or something “easy” to check, you get the marks for this.

---

6 with tiny font size.

7 Your function should be able to at least give an error (or rather, stop) for \( k = 1, 3, 4 \) (eg, say it is unsolvable), but solve for \( xvec \) when \( k = 2 \), giving \( x \approx (0, 0.3333, -0.2778, -0.2778)^T \). No marks were deducted for input \( L \) being non-triangular (nice to have in future to debug code), but marks were deducted if input was lower triangular, but you didn’t manage to calculate \( x \).

8 See the 2 4 6 experiment [Wat06].

9 Eg, is \( Lx \) similar to \( e_k \) and is \( LL^{-1} \) similar to \( I \)? Also we can check the norm of their differences instead.
6. Monahan, 3.10

Proof. \((-\Rightarrow)\) Suppose \(A\) is positive definite.

We factor \(A\) as \(PDPT\) with \(PT = P^{-1}\). Now let \(\lambda_1, \ldots, \lambda_n\) be the (positive) eigenvalues of \(A\). Let \(C\) be the matrix with \(\sqrt{\lambda_1}, \ldots, \sqrt{\lambda_n}\) on its diagonal.

Then we have \(D = C^2 = CT C\). Writing \(B = PCPT\), we have \(B\) to be positive definite as well since \(\sqrt{\lambda_i}\) are all positive.

Now, write:

\[ A = PDPT = PCCTCP = (PCPT)^T PCPT = B^T B. \]

Now, the eigenvalues of \(B\) are strictly positive implies that it is invertible, which implies that its columns are linearly independent.

Since \(B\) has linearly independent columns, we can factor \(B = QR\). In particular, we have \(Q\) to be an orthonormal square matrix and \(R\) to be a upper triangular square matrix.

Therefore, we have:

\[ A = B^T B = (QR)^T QR = R^T Q^T QR = R^T R \]

i.e. which implies the Cholesky decomposition will work.

\((\Rightarrow)\) Suppose the Cholesky decomposition works, i.e. we factor \(A = LL^T\), where \(L\) is lower triangular. We use the hint and realize that:

\[
\det \begin{pmatrix} A_{n\times n} & B_{n\times m} \\ \hline \hline \hline 0 & D_{m\times m} \end{pmatrix} = \det \begin{pmatrix} A_{n\times n} & 0 \\ \hline \hline \hline C_{m\times n} & D_{m\times m} \end{pmatrix} = \det(A) \det(D) \quad (\dagger)
\]

We have:

\[
A^{[k]} = \begin{pmatrix} A^{[k-1]} & a^{[k]} \\ a^{[k]T} & A_{kk} \end{pmatrix} = \begin{pmatrix} L^{[k-1]} & 0 \\ \hline \hline \hline l^{[k]}T & L_{kk} \end{pmatrix} \begin{pmatrix} L^{[k-1]} & l^{[k]} \quad l^{[k]}T \\ \hline \hline \hline 0 & L_{kk} \end{pmatrix} = L^{[k]} L^{[k]T}
\]

and we can write out:

\[
\det(A^{[k]}) = \det(L^{[k]} L^{[k]T})
\]

\[
= \det(L^{[k]}) \det(L^{[k]T})
\]

\[
= \det(L^{[k-1]}) \det(L_{kk}) \det(L^{[k-1]T}) \det(L_{kk})
\]

\[
= \det(L^{[k-1]} L^{[k-1]T}) L_{kk}^2
\]

\[
= \det(A^{[k-1]}) L_{kk}^2
\]

Now, we assume that \(\det(A^{[k]}) \neq 0\). If it was 0, then either \(\det(A^{[k-1]}) = 0\) or \(L_{kk}^2 = 0\).

But since we constructed \(L\) from \(A_{11}\), this can only happen when \(A_{11} = 0\) which implies that \(L^2 = 0\) which implies \(A = 0_{n\times n}\). Now, since \(\det(A^{[1]}) = L^{[1]} L^{[1]T} = L_{11}^2 > 0\), we must have all leading principal minors of \(A\) to be > 0 (by our equation), and thus \(A\) is positive definite.
Breakdown:

(a) Showing $\det(A^{[k]}) = L_{kk}^2 \det(A^{[k-1]})$.
   In particular:
   i. Quoting (from the) textbook or notes that $L_{kk}^2 = A_{kk} - l^{[k]^T} l^{[k]}$ [2]
   ii. Actually showing why they are equal. [2]

   \[2+2 = 4\]

(b) Showing the Cholesky decomposition works if the matrix is positive definite. [3]

(c) Showing a matrix is positive definite if the Cholesky decomposition works.
   i. “We have shown in class” [2]
   ii. A workable proof [3]

   \[ \min\{3, \text{score}\} \]

* Total: \[4 + 3 + 3 = 10\]

**Comment:** Note that a question which says “if and only if” means you have to show both sides.
7. **Monahan, 3.28**

*This question basically looks for an example, and since most people have an example (and this is already 28 pages long), code and such will be omitted.*

Breakdown:

(a) Construct a Vandermonde matrix [3]
(b) Find condition number [1]
(c) Rescale it [3]
(d) Find new condition number [1]
(e) Conclude [2]

* Total: 

[3 + 1 + 3 + 1 + 2 = 10]

** Extra Credit: Running simulations for a wide range of matrices and verifying. [3]

** Extra Credit: Instead of one (or two) scalings, investigating to find the ‘best’ (or at least a better) scaling and condition number. [3]

or

(a) Reading and paraphrasing and understanding (?) certain papers [7]
(b) and citing them. [3]
References
