Problem Set 4 SOLUTIONS

CSCI E-5a: Programming in R

Let's clear the environment:

```
rm( list = ls() )
```

Now let's load in the R objects for this problem set:

```
load( "Problem Set 4 R Objects.Rdata" )

ls()
```

```
## [1] "problem.2.data"
## [2] "problem.3.number.of.items.sold.vector"
## [3] "problem.3.transaction.id.vector"
## [4] "problem.3.widget.model.name.vector"
## [5] "problem.5.stock.price.raw.data.vector"
## [6] "problem.6.number.of.items.sold.vector"
## [7] "problem.6.widget.model.names.vector"
```

Problem 1: Newton's Method

In Unit 2, we reviewed the theory of quadratic polynomials, which are polynomials of this form:

$$f(x) \;=\; ax^2 + bx + c$$

Now we're going to see another way to solve this equation, known as *Newton's method*.

The basic idea of Newton's method is that we start with an initial guess for the root of the equation with the understanding that this is only an approximation to the correct answer.

Then we take this initial guess, denoted x_1, and transform it into a new value, denoted x_2, by using a special formula.

What makes the formula special is that the new value is a better approximation to the root than the first guess.

That's a pretty amazing formula!

For a quadratic polynomial, the special formula is:

$$x_2 \;=\; \frac{a \cdot x_1^2 - c}{2a \cdot x_1 + b}$$

What's even more amazing is that we can then use the special formula once again with our second, improved approximation, and that will give us an even better approximation to the root:

$$x_3 \;=\; \frac{a \cdot x_2^2 - c}{2a \cdot x_2 + b}$$

We can then repeat that process once again to obtain an even better approximation.

In general, we have:

$$x_{i+1} \;=\; \frac{a \cdot x_i^2 - c}{2a \cdot x_i + b}$$

So we're repetitively using the output of one computation as the input to the next computation.

Each time we use the formula, we get a better approximation to the root.

The derivation of the special formula involves some advanced mathematics that is not really the concern of our course, so we're not going to worry about how to derive it.

To be honest, Newton's method doesn't *always* work, and it depends a lot on how close your initial guess is to the real root.

What's important to understand is how the method works: we're always feeding the result of one computation back into the formula to get a better estimate.

If we run this process for a number of times, we'll end up with a very good numerical approximation to the root (maybe).

Those formulas might look a little complicated, and it looks like this method is going to require a lot of computation.

On the other hand, the process is highly repetitive, so we can automate it with a `for()` loop.

In this problem, we're going to study the polynomial:

$$f(x) \;=\; -2x^2 + 28x - 90$$

Part (a): Create variables

Create variables to represent the coefficients of this quadratic polynomial.

There's nothing to report here, but write your code clearly so the TAs can understand what you're doing.

Solution

First, let's define some variables:

```
a <- -2

b <- 28

c <- -90
```

Part (b): Roots

Calculate all the roots of this quadratic polynomial. Report each root using a separate `cat()` statement, displaying the value with 5 decimal places.

Solution

The discriminant of the quadratic polynomial is:

```
discriminant <-
  b^2 - 4 * a * c

cat(
  "Discriminant:",
  formatC(
    discriminant,
    format = "f",
    digits = 2
  )
)
```

```
## Discriminant: 64.00
```

Since the discriminant is positive, there are two distinct roots.

The first root is:

```
first.root <-
  (-b - sqrt( b^2 - 4 * a * c ) ) / (2 * a)

cat(
  "First root:",
  formatC(
    first.root,
    format = "f",
    digits = 2
  )
)
```

```
## First root: 9.00
```

The first root is:

```
second.root <-
  (-b + sqrt( b^2 - 4 * a * c ) ) / (2 * a)

cat(
  "Second root:",
  formatC(
    second.root,
    format = "f",
    digits = 2
  )
)
```

```
## Second root: 5.00
```

So the two roots are 5 and 9, which you can easily check.

Part (c): Second approximation

Now we're going to use Newton's method to approximate the first root of the polynomial.

We'll start with an initial estimate of 4 for this root.

Now use the initial estimate as the input for the special formula to obtain a better estimate of the first root, and store this back into the current estimate variable. Report the value of this second approximation using a `cat()` statement, displaying the value with 5 decimal places.

Make sure you use 5 decimal places, because we're trying to study this numerical proces in detail.

Solution

```
current.estimate <- 4

current.estimate <-
  (a * current.estimate^2 - c) /
  (2 * a * current.estimate + b)

cat(
  "Second approximation:",
  formatC(
    current.estimate,
    format = "f",
    digits = 5
  )
)
```

```
## Second approximation: 4.83333
```

Part (d): Third approximation

Now use the result of part (b) as the input for the special formula to obtain a better estimate of the root, and store this back into the current estimate variable. Report the value of this third approximation using a `cat()` statement, displaying the value with 5 decimal places.

Make sure you use 5 decimal places, because we're trying to study this numerical proces in detail.

Solution

```
current.estimate <-
  (a * current.estimate^2 - c) /
  (2 * a * current.estimate + b)

cat(
  "Third approximation:",
  formatC(
    current.estimate,
    format = "f",
    digits = 5
  )
)
```

```
## Third approximation: 4.99359
```

Part (e): Fourth approximation

Now use the result of part (b) as the input for the special formula to obtain a better estimate of the root, and store this back into the current estimate variable. Report the value of this third approximation using a `cat()` statement, displaying the value with 5 decimal places.

Make sure you use 5 decimal places, because we're trying to study this numerical proces in detail.

Solution

```
current.estimate <-
  (a * current.estimate^2 - c) /
  (2 * a * current.estimate + b)

cat(
  "Fourth approximation:",
  formatC(
    current.estimate,
    format = "f",
    digits = 5
  )
)
```

```
## Fourth approximation: 4.99999
```

Part (f): Second root

Now we're going to calculate the second root, but this time we'll automate the process.

We'll start with an initial estimate of 12, and we'll use 5 repetitions of the special formula.

Construct a `for()` loop that repeatedly applies the special formula to the current estimate to produce a better estimate, feeding this value back into formula in the next iteration.

Create a numbered list which prints out a short `cat()` statement displaying the value of the current estimate at the beginning of the loop, before applying the special formula. Use 5 decimal places for the numerical value.

When you're all done, print out the final value of the current estimate using a `cat()` statement displaying the value with 5 decimal places.

Solution

```
current.estimate <- 12

number.of.iterations <- 5

for( index in 1:number.of.iterations ) {

  cat(
    index,
    ". ",
    "Current estimate: ",
    formatC(
      current.estimate,
      format = "f",
```

```
      digits = 5
    ),
    "\n",
    sep = ""
  )

  current.estimate <-
    (a * current.estimate^2 - c) /
    (2 * a * current.estimate + b)
}
```

```
## 1. Current estimate: 12.00000
## 2. Current estimate: 9.90000
## 3. Current estimate: 9.13966
## 4. Current estimate: 9.00456
## 5. Current estimate: 9.00001
```

```
cat(
  "\n\tFinal estimate:",
  formatC(
    current.estimate,
    format = "f",
    digits = 5
  )
)
```

```
##
##  Final estimate: 9.00000
```

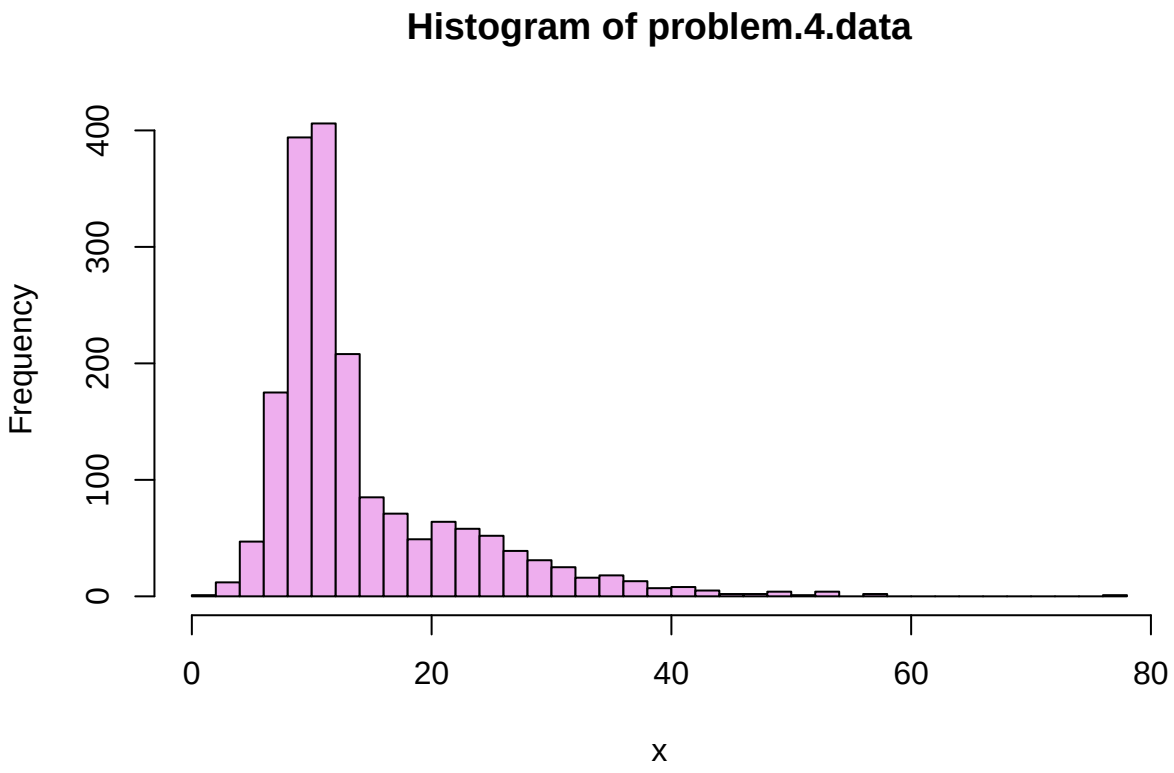
End of problem 1

Problem 2: Histogram

Construct a histogram for the values in the vector `problem.2.data`. Be sure to include a main title and titles for the x-axis and y-axis. Specify the color of the bars, and explicitly indicate that the number of breaks should be 50.

Solution

```
hist(
    problem.2.data,
    main = "Histogram of problem.4.data",
    xlab = "x",
    ylab = "Frequency",
    col = "plum2",
    breaks = 50
)
```



End of problem 2

Problem 3: Sales Report

WiDgT offers five different models:

Model	Price
Classic WiDgT	4.99
WiDgT 2.0	5.99
WiDgT 3k	8.99
Quadcore WiDgT	10.99
WiDgT Mach 5	12.99

Here is data for five sales, listing the widget model and the number of widgets sold.

Transaction	Model	Number of Items
4171	WiDgT 2.0	50
1683	WiDgT Mach 5	65
7719	WiDgT 3k	10
5684	Classic WiDgT	25
9125	WiDgT 3k	40

When you loaded the R objects at the beginning of this problem set, they included three vectors for this problem:

- The vector `problem.3.transaction.id.vector` contains the transaction ids for each of the five transactions.
- The vector `problem.3.widget.model.name.vector` contains the widget model name for each of the five transactions.
- The `problem.3.number.of.items.sold.vector` contains the number of items sold in the transaction.

Your job is to create a sales report for these five sales. Your report should include:

- The transaction ID.
- The widget model name.
- The price per item.
- The number of items sold.
- The total sales amount of the transaction.

Finally, report the total amount of all the sales.

For this problem, you should use the vectors that were loaded in. You'll have to determine how to convert the widget model name to a price, and how to calculate the total sales amount of each transaction. You should use a `for` loop to automate the overall process. Report all results using `cat()` statements, displaying each value with 2 decimal places.

You're on your own for this one – you'll have to figure out how to design and implement this. You might find it useful to use some vectorized operations, but you must use a `for` loop to automate the report

generation. Get creative with simple applications of newline characters and tab characters to supply some basic formatting for your report.

Other than that, we just want to see a nice final report.

Solution

```
price.lookup.vector <-
    c(
        "Classic" = 4.99,
        "Widget 2.0" = 5.99,
        "Widget 3k" = 8.99,
        "Quadcore Widget" = 10.99,
        "Widget Mach 5" = 12.99
    )
```

```
total.overall.sales <- 0

{
for( i in 1:length( problem.3.number.of.items.sold.vector ) ) {

    current.price.per.item <-
        price.lookup.vector[
             problem.3.widget.model.name.vector[ i ]
         ]

    number.of.items.sold <-
        problem.3.number.of.items.sold.vector[ i ]

    transaction.sales.amount <-
        current.price.per.item * number.of.items.sold

    total.overall.sales <-
        total.overall.sales +
        transaction.sales.amount

    cat(
        "Transaction ID:",
        problem.3.transaction.id.vector[ i ],
        "\n"
    )

    cat( "\t\tModel name:",
         problem.3.widget.model.name.vector[ i ],
        "\n")

    cat( "\t\tPrice per item:",
         current.price.per.item,
         "\n"
    )

    cat(
        "\t\tNumber of items:",
        problem.3.number.of.items.sold.vector[ i ],
        "\n"
```

```
    )

    cat(
        "\t\tTransaction total sales amount:",
        formatC(
          x = transaction.sales.amount,
          format = "f",
          digits = 2
        ),
        "\n"
    )

    cat( "\n" )
}

cat(
    "\nTotal Overall Sales:",
    total.overall.sales
)}
```

```
## Transaction ID: 4171
##      Model name: Widget 2.0
##      Price per item: 5.99
##      Number of items: 50
##      Transaction total sales amount: 299.50
##
## Transaction ID: 1683
##      Model name: Widget Mach 5
##      Price per item: 12.99
##      Number of items: 65
##      Transaction total sales amount: 844.35
##
## Transaction ID: 7719
##      Model name: Widget 3k
##      Price per item: 8.99
##      Number of items: 10
##      Transaction total sales amount: 89.90
##
## Transaction ID: 5684
##      Model name: Classic
##      Price per item: 4.99
##      Number of items: 25
##      Transaction total sales amount: 124.75
##
## Transaction ID: 9125
##      Model name: Widget 3k
##      Price per item: 8.99
##      Number of items: 40
##      Transaction total sales amount: 359.60
##
##
## Total Overall Sales: 1718.1
```

End of problem 3

Problem 4: Loan Amortization

In this problem, we'll use iteration to calculate two loan amortization problems.

First, we want you to show us that you can construct an amortization schedule by hand, and that's parts (a) through (d).

Then we'll look at another problem which would require a lot of work to calculate by hand, and we'll automate this computation by using iteration.

Part (a): Payment Amount

Marie takes out a loan for $2,000, and pays it back in three annual installments. The interest rate for the loan is 3.5%.

Calculate the payment amount for Marie's annual payments so that she pays the loan off completely at the end of 3 years. Save this value in a variable, and report your result using a `cat()` statement, displaying this value with 2 decimal places.

Solution

Let's start by defining some variables:

```
loan.amount <- 2000

number.of.payments <- 3

interest.rate <- 0.035
```

First, we calculate the discount factor:

```
discount.factor <-
    1 / (1 + interest.rate)
```

Now we can calculate the annual payment amount:

```
annual.payment.amount <-
    (loan.amount * interest.rate) /
    (1 - discount.factor^number.of.payments)

cat(
    "Annual payment amount:",
    formatC(
        annual.payment.amount,
        format = "f",
        digits = 2
    )
)
```

```
## Annual payment amount: 713.87
```

Part (b): First year

For year 1, determine:

- The loan balance at the beginning of the year
- The loan balance at the end of the year, before Marie makes a payment.
- The loan balance at the end of the year, after Marie makes a payment.

Report each value using a separate `cat()` statement, displaying the value using 2 decimal places.

Solution

```
loan.balance.at.beginning.of.year <-
    loan.amount

cat(
    "Loan balance at beginning of year 1:",
    formatC(
        loan.balance.at.beginning.of.year,
        format = "f",
        digits = 2
    )
)
```

```
## Loan balance at beginning of year 1: 2000.00
```

```
loan.balance.at.end.of.year.before.payment <-
    loan.balance.at.beginning.of.year *
    (1 + interest.rate)

cat(
    "Loan balance at end of year 1, before payment:",
    formatC(
        loan.balance.at.end.of.year.before.payment,
        format = "f",
        digits = 2
    )
)
```

```
## Loan balance at end of year 1, before payment: 2070.00
```

```
loan.balance.at.end.of.year.after.payment <-
    loan.balance.at.end.of.year.before.payment -
    annual.payment.amount

cat(
    "Loan balance at end of year 1, after payment:",
    formatC(
        loan.balance.at.end.of.year.after.payment,
        format = "f",
        digits = 2
    )
)
```

```
## Loan balance at end of year 1, after payment: 1356.13
```

Part (c): Second year

For year 2, determine:

- The loan balance at the beginning of the year
- The loan balance at the end of the year, before Marie makes a payment.
- The loan balance at the end of the year, after Marie makes a payment.

Report each value using a separate `cat()` statement, displaying the value using 2 decimal places.

Solution

```
loan.balance.at.beginning.of.year <-
    loan.balance.at.end.of.year.after.payment

cat(
    "Loan balance at beginning of year 2:",
    formatC(
        loan.balance.at.beginning.of.year,
        format = "f",
        digits = 2
    )
)
```

```
## Loan balance at beginning of year 2: 1356.13
```

```
loan.balance.at.end.of.year.before.payment <-
    loan.balance.at.beginning.of.year *
    (1 + interest.rate)

cat(
    "Loan balance at end of year 2, before payment:",
    formatC(
        loan.balance.at.end.of.year.before.payment,
        format = "f",
        digits = 2
    )
)
```

```
## Loan balance at end of year 2, before payment: 1403.60
```

```
loan.balance.at.end.of.year.after.payment <-
    loan.balance.at.end.of.year.before.payment -
    annual.payment.amount

cat(
    "Loan balance at end of year 2, after payment:",
    formatC(
```

```
        loan.balance.at.end.of.year.after.payment,
        format = "f",
        digits = 2
    )
)
```

```
## Loan balance at end of year 2, after payment: 689.73
```

Part (d): Third year

For year 3, determine:

- The loan balance at the beginning of the year
- The loan balance at the end of the year, before Marie makes a payment.
- The loan balance at the end of the year, after Marie makes a payment.

Report each value using a separate `cat()` statement, displaying the value using 2 decimal places.

Solution

```
loan.balance.at.beginning.of.year <-
    loan.balance.at.end.of.year.after.payment

cat(
    "Loan balance at beginning of year 3:",
    formatC(
        loan.balance.at.beginning.of.year,
        format = "f",
        digits = 2
    )
)
```

```
## Loan balance at beginning of year 3: 689.73
```

```
loan.balance.at.end.of.year.before.payment <-
    loan.balance.at.beginning.of.year *
    (1 + interest.rate)

cat(
    "Loan balance at end of year 3, before payment:",
    formatC(
        loan.balance.at.end.of.year.before.payment,
        format = "f",
        digits = 2
    )
)
```

```
## Loan balance at end of year 3, before payment: 713.87
```

```r
loan.balance.at.end.of.year.after.payment <-
    loan.balance.at.end.of.year.before.payment -
    annual.payment.amount

cat(
    "Loan balance at end of year 3, after payment:",
    formatC(
        loan.balance.at.end.of.year.after.payment,
        format = "f",
        digits = 2
    )
)
```

```
## Loan balance at end of year 3, after payment: -0.00
```

Part (e): Automating the calculation

Now we'll take on a more challenging problem, and solve by automating the computations using iteration.

Taylor takes out a loan for $200,000, to be paid back in 10 annual installments. The interest rate for the loan is 2.8%.

Construct a report that displays the loan amortization schedule for all 10 years. For each year, your report should include:

- The loan balance at the beginning of the year.
- The loan balance at the end of the year, before Taylor makes a payment.
- The loan balance at the end of the year, after Taylor makes a payment.

Do *not* calculate all of these values by hand – you won't receive credit if you do. Instead, set up a `for` loop that will automate this proces.

Try to use some simple applications of newline characters and tab characters to supply some basic formatting for your report.

Since the loan is for 10 years, the `for` loop will have 10 iterations, but that's all that I'll say. Otherwise, you're on your own for this one.

Here's a suggestion: in Module 4, we worked on an example in lecture, and another example in the exercise. If you write your code properly, you should be able to change the values of a few variables and everything will work properly. So write your code, and try it out on the worked examples. If you can get it to work with them, then you should have little difficulty with this problem.

Solution

Let's start by defining some variables:

```r
loan.amount <- 200000

number.of.payments <- 10

interest.rate <- 0.028
```

First, we calculate the discount factor:

```
discount.factor <-
    1 / (1 + interest.rate)
```

Now we can calculate the annual payment amount:

```
annual.payment.amount <-
    (loan.amount * interest.rate) /
    (1 - discount.factor^number.of.payments)

cat(
    "Annual payment amount:",
    formatC(
        annual.payment.amount,
        format = "f",
        digits = 2
    )
)
```

```
## Annual payment amount: 23207.42
```

Now we'll construct a `for` loop to automate these calculations:

```
loan.balance.at.beginning.of.year <-
    loan.amount

for( current.year in 1:number.of.payments ) {

    cat( "Year:", current.year, "\n" )

    cat(
        "Loan balance at beginning of year :",
        i,
        ": ",
        formatC(
            loan.balance.at.beginning.of.year,
            format = "f",
            digits = 2
        ),
        "\n",
        sep = ""
    )

    loan.balance.at.end.of.year.before.payment <-
        loan.balance.at.beginning.of.year *
            (1 + interest.rate)

    loan.balance.at.end.of.year.after.payment <-
        loan.balance.at.end.of.year.before.payment -
            annual.payment.amount

    cat(
        "Loan balance at end of year ",
        i,
```

```
        ", before payment: ",
        formatC(
            loan.balance.at.end.of.year.before.payment,
            format = "f",
            digits = 2
        ),
        "\n",
        sep = ""
    )

    cat(
        "Loan balance at end of year ",
        i,
        ", after payment: ",
        formatC(
            loan.balance.at.end.of.year.after.payment,
            format = "f",
            digits = 2
        ),
        "\n",
        sep = ""
    )

    cat( "\n" )


    loan.balance.at.beginning.of.year <-
        loan.balance.at.end.of.year.after.payment
}
```

```
## Year: 1
## Loan balance at beginning of year :5: 200000.00
## Loan balance at end of year 5, before payment: 205600.00
## Loan balance at end of year 5, after payment: 182392.58
##
## Year: 2
## Loan balance at beginning of year :5: 182392.58
## Loan balance at end of year 5, before payment: 187499.57
## Loan balance at end of year 5, after payment: 164292.16
##
## Year: 3
## Loan balance at beginning of year :5: 164292.16
## Loan balance at end of year 5, before payment: 168892.34
## Loan balance at end of year 5, after payment: 145684.92
##
## Year: 4
## Loan balance at beginning of year :5: 145684.92
## Loan balance at end of year 5, before payment: 149764.09
## Loan balance at end of year 5, after payment: 126556.68
##
## Year: 5
## Loan balance at beginning of year :5: 126556.68
## Loan balance at end of year 5, before payment: 130100.26
```

```
## Loan balance at end of year 5, after payment: 106892.84
##
## Year: 6
## Loan balance at beginning of year :5: 106892.84
## Loan balance at end of year 5, before payment: 109885.84
## Loan balance at end of year 5, after payment: 86678.43
##
## Year: 7
## Loan balance at beginning of year :5: 86678.43
## Loan balance at end of year 5, before payment: 89105.42
## Loan balance at end of year 5, after payment: 65898.00
##
## Year: 8
## Loan balance at beginning of year :5: 65898.00
## Loan balance at end of year 5, before payment: 67743.15
## Loan balance at end of year 5, after payment: 44535.73
##
## Year: 9
## Loan balance at beginning of year :5: 44535.73
## Loan balance at end of year 5, before payment: 45782.73
## Loan balance at end of year 5, after payment: 22575.31
##
## Year: 10
## Loan balance at beginning of year :5: 22575.31
## Loan balance at end of year 5, before payment: 23207.42
## Loan balance at end of year 5, after payment: 0.00
```

End of problem 4

Problem 5: Moving Average

The vector `problem.5.stock.price.raw.data.vector` contains closing stock prices for Krispee Yummm!! stock.

Part (a): Determining the number of trading days

Determine how many trading days are represented by this data. Report your result using a `cat()` statement, displaying the value with 0 decimal places.

Solution

```
number.of.trading.days <-
  length( problem.5.stock.price.raw.data.vector )

cat(
  "Number of trading days:",
  formatC(
    x = number.of.trading.days,
    format = "f",
    digits = 0
  )
)
```

```
## Number of trading days: 50
```

Part (b): Creating the trading days index vector

Using your result in part (a), construct the vector of trading day indices.

Report the first 5 elements of this vector using a `cat()` statement, displaying each value with 0 decimal places.

Solution

```
trading.day.index.vector <- 1:number.of.trading.days

cat(
  "Trading day index vector:",
  formatC(
    x = head( x = trading.day.index.vector, n = 5 ),
    format = "f",
    digits = 0
  )
)
```

```
## Trading day index vector: 1 2 3 4 5
```

Part (c): Constructing the vector of smoothed values

Now we'll construct a vector consisting of the smoothed values for these stock prices.

- First, allocate storage for the smoothed values vector using the value you calculated in part (a).
- Next, populate the first element of the smoothed values vector with the raw stock price value for the first day.
- Then construct a `for` loop that will automate the process of calculating moving averages for the stock price.
- Finally, populate the last element of the smoothed values vector with the raw stock price value for the last day.

When you're all done, report the first 5 elements of the smoothed values vector using a `cat()` statement, displaying each value with 2 decimal places.

Solution

First, let's pre-allocate the storage for the smoothed values vector

```
smoothed.values.vector <-
  numeric( number.of.trading.days )
```

Next, we'll populate the first element of the smoothed values vector with the raw stock price for day 1:

```
smoothed.values.vector[ 1 ] <-
  problem.5.stock.price.raw.data.vector[ 1 ]
```

Now we'll construct the `for` loop:

```
for( trading.day.index in 2:(number.of.trading.days - 1) ) {
  smoothed.values.vector[ trading.day.index ] <-
    (problem.5.stock.price.raw.data.vector[ trading.day.index - 1 ] +
       problem.5.stock.price.raw.data.vector[ trading.day.index ] +
       problem.5.stock.price.raw.data.vector[ trading.day.index + 1 ] ) /
    3
}
```

Next, we'll populate the last element of the smoothed values vector with the raw stock price for the last day:

```
smoothed.values.vector[ number.of.trading.days ] <-
  problem.5.stock.price.raw.data.vector[ number.of.trading.days ]
```

Finally we can display the first 5 elements of the vector of smoothed values:

```
cat(
  "Smoothed values vector:",
  formatC(
    x = head( x = smoothed.values.vector, n = 5),
    format = "f",
    digits = 2
  )
)
```

```
## Smoothed values vector: 33.55 58.79 58.69 58.35 50.06
```

Part (d): Constructing the line graph

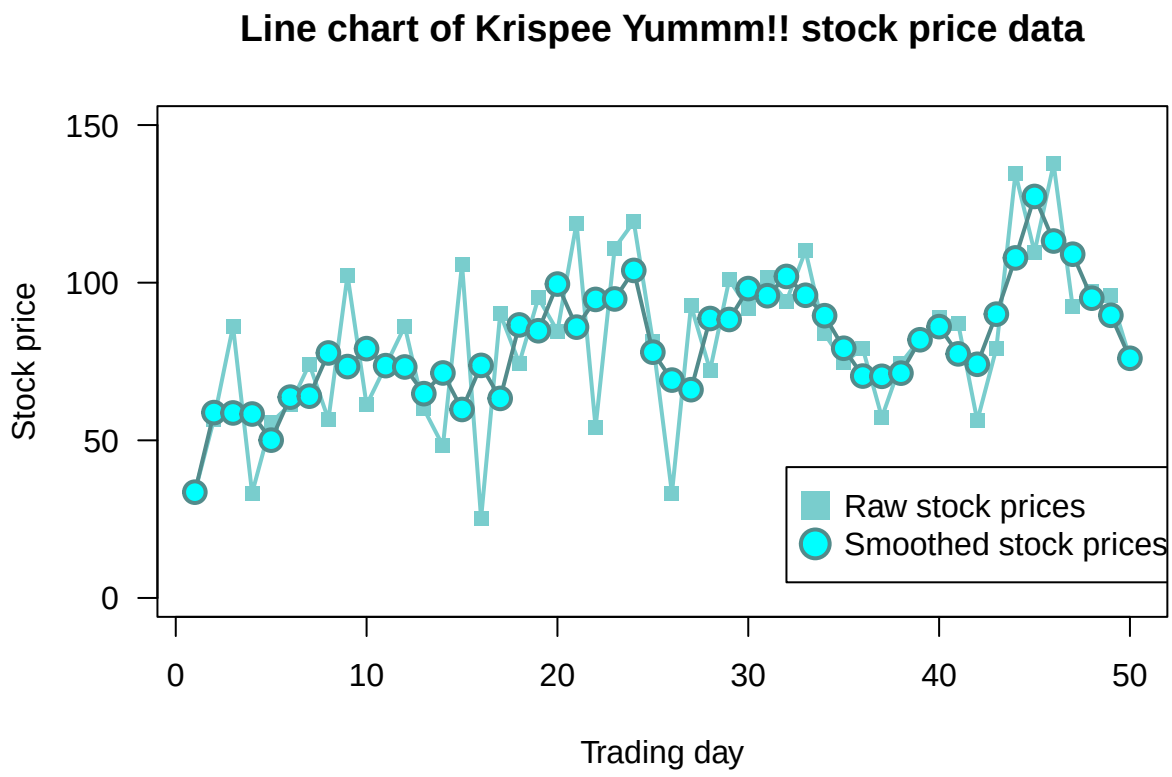
Construct a line graph with the raw stock data and the smoothed moving average values. Be sure to specify the range of the x- and y-axes, and include a main title, titles for the x and y-axes, and a legend.

Hint: we strongly recommend that the values on the y axis range from 0 to 150.

Solution

```
plot(
  x = NULL,
  xlim = c(1, number.of.trading.days),
  ylim = c(0, 150),
  main = "Line chart of Krispee Yummm!! stock price data",
  xlab = "Trading day",
  ylab = "Stock price",
  las = 1
)

lines(
  x = trading.day.index.vector,
  y = problem.5.stock.price.raw.data.vector,
  lty = "solid",
  lwd = 2,
  col = "darkslategray3"
)

points(
  x = trading.day.index.vector,
  y = problem.5.stock.price.raw.data.vector,
  pch = 15,
  lwd = 2,
  col = "darkslategray3"
)

lines(
  x = trading.day.index.vector,
  y = smoothed.values.vector,
  lty = "solid",
  lwd = 2,
  col = "darkslategray4"
)

points(
  x = trading.day.index.vector,
  y = smoothed.values.vector,
  pch = 21,
  cex = 1.5,
  lwd = 2,
  bg = "cyan",
  col = "darkslategray4"
)

legend(
  x = 32,
```

```
  y = 41.5,
  legend = c( "Raw stock prices", "Smoothed stock prices" ),
  pch = c( 15, 21),
  col = c( "darkslategray3", "darkslategray4"),
  pt.bg = "cyan",
  pt.lwd = 2,
  pt.cex = 2
)
```



End of problem 5

Problem 6: Invalid Values Reporter

In the last lecture, we saw how to use lookup vectors to repair invalid values in data.

This is an important practical skill, but it would also be nice to have a report of all the invalid values that are observed in our data so that they can be investigated.

Prelude: The `setdiff()` function

For this problem, we're going to learn a new function called `setdiff()`.

In this part, I'll give you all the background that you need for the problem.

The `set.diff()` function takes two input arguments that are vectors, denoted A and B, and returns a vector consisting of all the elements of A that are **not** elements of B.

You might need a moment to unpack that definition.

Just remember: if a value is contained in A and it's also contained in B, then `setdiff()` will **not** return that value.

The vectors don't have to be coordinated or have any special relationship; `setdiff()` treats the input arguments as just sets or collections of objects without any order.

There's nothing to report for this section

Let's see an example of this:

Let A be a vector consisting of the values from 1 to 5 in increments of 1:

```
A <- 1:5

A
```

```
## [1] 1 2 3 4 5
```

Let B be a vector consisting of the values from 1 to 3 in increments of 1.

```
B <- 1:3

B
```

```
## [1] 1 2 3
```

What is `setdiff(A, B)`?

Remember that A consists of the values 1 to 5.

The numbers 1, 2, and 3 are all contained in B, so these values are **not** in `setdiff(A, B)`.

The numbers 4 and 5 are *not* contained in B, so these values **are** in `setdiff(A, B)`.

Let's try this out:

```
setdiff(A, B)
```

```
## [1] 4 5
```

OK, let's get started.

Part (a): Obtaining the unique values.

Recall the WiDgT product line:

Model	Price
Classic WiDgT	4.99
WiDgT 2.0	5.99
WiDgT 3k	8.99
Quadcore WiDgT	10.99
WiDgT Mach 5	12.99

In this problem, data on widget sales is recorded in two vectors:

- The vector `problem.6.widget.model.names.vector` contains the name of the widget model in the transaction.
- The vector `problem.6.number.of.items.sold.vector` contains the number of items sold in the transaction.

Construct a vector consisting of a set of unique representatives for the values in `problem.6.widget.model.vector`. (The preceding sentence contains a subtle hint for the name of the function to use.) Save this vector of unique representatives in a variable, and report it using a `cat()` statement, displaying each value with 2 decimal places.

There are a lot of elements of this vector, and you'll need to be sure that this `cat()` statement doesn't run off the page, so you'll need to use the `fill = TRUE` option. Review Section 5: Reporting Vectors in Lecture 3 Module 1: Vector Basics for information about this option.

Solution

```
model.names.unique.representatives.vector <-
  unique( problem.6.widget.model.names.vector )

cat( "Model names unique representatives:",
     model.names.unique.representatives.vector,
     fill = TRUE)
```

```
## Model names unique representatives: WiDgT Mach 5 Classic WiDgT WiDgT 3k
## Classsic WiDgT 2.0 Quadcore WiDgT Widget 5 Classic Widget Widget Mach5
## Widget Mch 5 Widget 2 Widget 3000
```

Part (b): Valid model names vector

Construct a vector consisting of the valid names for the widget models. That is, there are 5 different models of widget, so you should construct a vector with 5 elements, each of which is the correct version of one of the models. Report your result with a `cat()` statement.

Solution

```
valid.model.names.vector <-
  c( "Classic WiDgT", "WiDgT 2.0", "WiDgT 3k",
     "Quadcore WiDgT", "WiDgT Mach 5" )

cat( "Valid model names vector:", valid.model.names.vector )
```

```
## Valid model names vector: Classic WiDgT WiDgT 2.0 WiDgT 3k Quadcore WiDgT WiDgT Mach 5
```

Part (c): Selecting the invalid model names

In this part, we want to construct a vector consisting of the *invalid* model names.

That is, we want to construct a vector consisting of the model names that occur in our data, but are not valid model names.

In other words, we want to construct a vector consisting of the values that occur in the vector of unique representatives that do **not** occur in the value of valid widget model names.

Use the `setdiff()` function along with the vectors you created in parts (a) and (b) to construct a vector consisting of unique representatives of the invalid model names that occur in `problem.6.model.names.vector`. Save this vector in a variable, and report the first 5 values using a `cat()` statement.

Solution

```
invalid.model.names.vector <-
  setdiff(
    model.names.unique.representatives.vector,
    valid.model.names.vector
  )

cat(
    "Invalid model names:",
    head( x = invalid.model.names.vector, n = 12),
    fill = TRUE
)
```

```
## Invalid model names: Classsic Widget 5 Classic Widget Widget Mach5 Widget Mch 5
## Widget 2 Widget 3000
```

Part (d): Invalid name reporter

Use a `for` loop to write a short report enumerating each of the invalid values in `problem.6.model.names.vector`. Remember, to "enumerate" the invalid values you should use a separate `cat()` statement for each invalid value, and the statements should be numbered as "1.", "2.", and so on.

Solution

```
for( i in 1:length( invalid.model.names.vector ) ) {
  cat( i,
       ". Invalid model name: ",
       invalid.model.names.vector[ i ],
       "\n",
       sep = "")
}
```

```
## 1. Invalid model name: Classsic
## 2. Invalid model name: Widget 5
## 3. Invalid model name: Classic Widget
## 4. Invalid model name: Widget Mach5
## 5. Invalid model name: Widget Mch 5
## 6. Invalid model name: Widget 2
## 7. Invalid model name: Widget 3000
```

Here's another approach using a counter variable:

```
counter <- 1

for( invalid.model.name in invalid.model.names.vector ) {
  cat(
    counter,
    ". Invalid model name: ",
    invalid.model.name,
    "\n",
    sep = ""
  )

  counter <- counter + 1
}
```

```
## 1. Invalid model name: Classsic
## 2. Invalid model name: Widget 5
## 3. Invalid model name: Classic Widget
## 4. Invalid model name: Widget Mach5
## 5. Invalid model name: Widget Mch 5
## 6. Invalid model name: Widget 2
## 7. Invalid model name: Widget 3000
```

Part (e): Total sales amount

What is the total sales amount for this set of transactions? Repair the invalid model names and then calculate the total sales amount. You can are encouraged to use the price lookup vector that you constructed in Problem 3 to convert valid model names to their price. Report the total sales amount with a `cat()` statement, displaying each value with 2 decimal places.

Solution

```
repair.model.names.lookup.vector <-
  c(
    "Classic WiDgT" = "Classic WiDgT",
    "WiDgT 2.0" = "WiDgT 2.0",
    "WiDgT 3k" = "WiDgT 3k",
    "Quadcore WiDgT" = "Quadcore WiDgT",
    "WiDgT Mach 5" = "WiDgT Mach 5",

    "Classsic" = "Classic WiDgT",
    "Widget 5" = "WiDgT Mach 5",
    "Classic Widget" = "Classic WiDgT",
    "Widget Mach5" = "WiDgT Mach 5",
    "Widget Mch 5" = "WiDgT Mach 5",
    "Widget 2" = "WiDgT 2.0",
    "Widget 3000" = "WiDgT 3k"
  )
```

Now we can repair the invalid values:

```
repaired.model.names.vector <-
  repair.model.names.lookup.vector[
    problem.6.widget.model.names.vector
  ]
```

Let's check this by using the `unique()` function once again:

```
unique( repaired.model.names.vector )
```

```
## [1] "WiDgT Mach 5"   "Classic WiDgT"  "WiDgT 3k"       "WiDgT 2.0"
## [5] "Quadcore WiDgT"
```

```
widget.model.price.lookup.vector <-
    c(
        "Classic WiDgT" = 4.99,
        "WiDgT 2.0" = 5.99,
        "WiDgT 3k" = 8.99,
        "Quadcore WiDgT" = 10.99,
        "WiDgT Mach 5" = 12.99
    )

widget.model.price.lookup.vector
```

```
##  Classic WiDgT      WiDgT 2.0       WiDgT 3k Quadcore WiDgT   WiDgT Mach 5
##           4.99           5.99           8.99          10.99          12.99
```

Now we can convert these valid model names to prices:

```
model.price.vector <-
  widget.model.price.lookup.vector[
    repaired.model.names.vector
  ]
```

Let's check the first 8 values of this vector, just to make sure that our process is working properly:

```
head( x = model.price.vector, n = 8 )
```

```
##  WiDgT Mach 5 Classic WiDgT      WiDgT 3k Classic WiDgT  WiDgT Mach 5
##         12.99          4.99          8.99          4.99         12.99
## Classic WiDgT     WiDgT 2.0 Classic WiDgT
##          4.99          5.99          4.99
```

Now we can calculate the total sales amount using a dot product:

```
total.sales.amount <-
  sum( model.price.vector * problem.6.number.of.items.sold.vector )
```

Finally, we can print out the final answer:

```
cat( "Total sales amount:",
     formatC(
       x = total.sales.amount,
       format = "f",
       big.mark = ",",
       digits = 2
     )
)
```

```
## Total sales amount: 7,469,693.00
```