

**Deduction for Late Submission of assignment:**

***For Students:***  
Once marked please refer to Moodle  
for your final coursework grade,  
including your Peer Assessment grade.

- **Theory background**

The Wilcoxon signed-rank test is a non-parametric test that is used to compare two related samples, repeated observations or matched samples. Using the Wilcoxon Signed-Rank test, it is judged whether the corresponding data population distribution is the same without assuming that the data is normally distributed. The Wilcoxon signed-rank test is based on three assumptions: 1) Data are paired and come from the same population. 2) Each pair is chosen randomly and independently. 3) The data are measured on at least an interval scale when, as is usual, within-pair differences are calculated to perform the test (though it does suffice that within-pair comparisons are on an ordinal scale). However, the test has a main limitation, that is, when the difference between the groups is zero, the observations are removed. This is of particular concern if the samples are taken from a discrete distribution.

- **Task 1: VBA implementation of the Wilcoxon signed-rank test**



Figure 1.1: Wilcoxon signed-rank test button

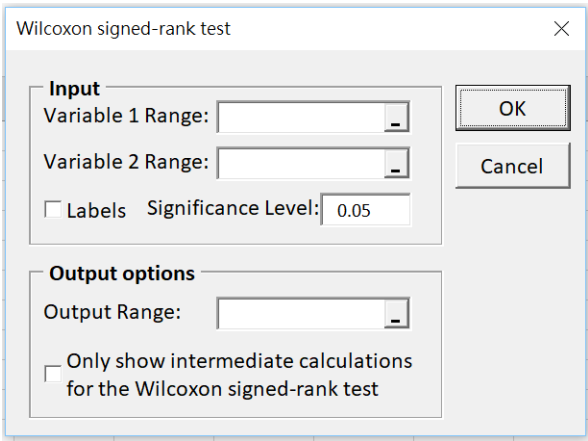


Figure 1.2: User form of the Wilcoxon test

When users are executing the Wilcoxon signed-rank test, click on the button “Wilcoxon signed-rank test” (Figure 1.1) on the excel sheet, and it will show the user form to perform the test. Figure 1.2 is the user form that we have designed for users to input the cell ranges into the dialog and have option to include the labels in the data selected. The default value of the significant values of the test is 5%, but users can input other values which they want to use. Then users can select a specific cell where the outputs will appear.

Calculations for the Wilcoxon signed-rank test									Calculations for the Wilcoxon signed-rank test								
i	A	B	d <sub>i</sub>	d <sub>i</sub> (non-zero)	Abs.d <sub>i</sub>	ranks	signed-ranks		i	Variable 1	Variable 2	d <sub>i</sub>	d <sub>i</sub> (non-zero)	Abs.d <sub>i</sub>	ranks	signed-ranks	
1	135	128	7	7	7	7	7		1	135	128	7	7	7	7	7	
2	110	105	5	5	5	4.5	4.5		2	110	105	5	5	5	4.5	4.5	
3	131	119	12	12	12	9	9		3	131	119	12	12	12	9	9	
4	142	140	2	2	2	1	1		4	142	140	2	2	2	1	1	
5	105	98	7	7	7	7	7		5	105	98	7	7	7	7	7	
6	130	123	7	7	7	7	7		6	130	123	7	7	7	7	7	
7	131	127	4	4	4	2.5	2.5		7	131	127	4	4	4	2.5	2.5	
8	110	115	-5	-5	5	4.5	-4.5		8	110	115	-5	-5	5	4.5	-4.5	
9	125	125	0	exclude	exclude	exclude	exclude		9	125	125	0	exclude	exclude	exclude	exclude	
10	149	145	4	4	4	2.5	2.5		10	149	145	4	4	4	2.5	2.5	

Figure 1.3: The tabular format of output (first-type output)

Wilcoxon signed-rank test	
n	10
nr	9
SR(+)	40.5
SR(-)	4.5
T	4.5
E[T]	22.5
Var(T)	71.25
z	-2.13246
p-value	0.032969
$\alpha$	0.05
result	Reject H0

Figure 1.4: The overall results of output (second-type output)

There are also two type of outputs. The first type is a tabular format with all the intermediate calculations of the test (Figure 1.3), which will only show when the user ticks the option at the bottom of the user form. If the user has the decision to include the labels, the labels of the selected data will be shown on the top of the data column. Otherwise, the labels will be variable 1 and variable 2. The second type of output will only be the basic one with the overall results of the Wilcoxon test (Figure 1.4), including the sums of each positive and negative ranks, the T test statistic, the mean and variance of T, the z-score and the associated p-value.

In addition, we did some testing to predict that interactive warnings could be provided to users in the following possible errors.

**Sufficient data check**

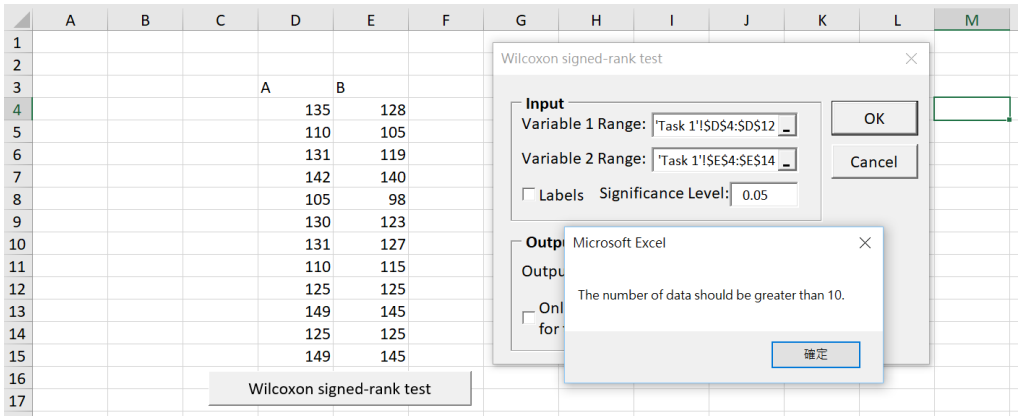


Figure 1.5

Figure 1.5 shows that the range of Variable 1 is D4:D12 which have 9 inputs of data, and range of Variable 2 is E4:E14 indicates that it has 10 inputs. The recommendation of the sample size for a valid test should be at least 10, since normal approximation will fail with fewer observations. Although the input data of variable 2 is sufficient, variable 1 does not have enough data. Thus, there will be a warning showing that the number of data should be greater than 10. Similarly, if the input data of variable 2 are not sufficient but that of variable 1 are sufficient, it will also show this kind of warning.

**Pairwise data check**

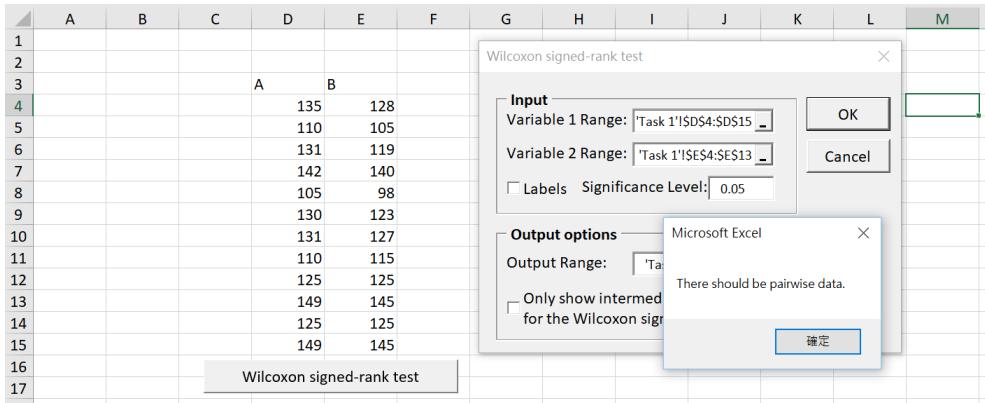


Figure 1.6

Suppose that there are both sufficient data in variable 1 and variable 2. Like figure 1.6, variable 1 range is D4:D15 which have 12 inputs, variable 2 range is E4:E13 which only have 10 input data. In order to calculate the pairwise difference in the output of the test, the number of input data should be the same. Therefore, in this case, the number of input data is different, there will be a warning pop up showing that there should be pairwise data.

**Numeric significance level check**

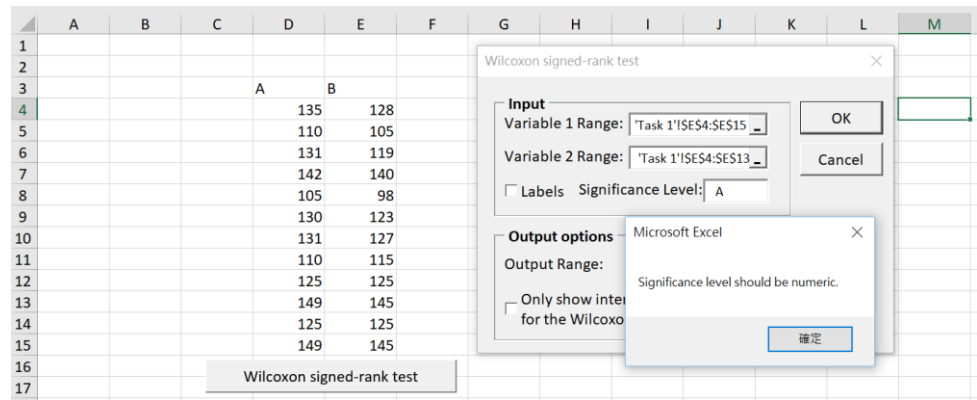


Figure 1.7

In figure 1.7, with the correct input data, the significance level should be a numeric number to create a correct output. In this case, the significance is A which is a non-numeric number. A warning will show that there is an error because significance level should be numeric.

**Incorrect range references check**

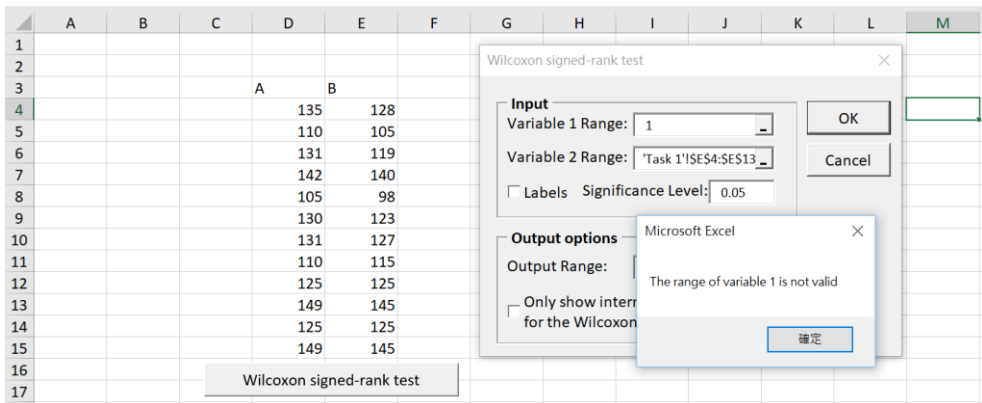


Figure 1.8

It is easy to find that there is an incorrect range reference in figure 1.8. The inputs of variable range cannot be a number or a letter. Therefore, a warning will show that the range of variable 1 is not valid. This sort of error will also be popped up when variable 2 range or output range are wrong range references.

**Existent data check**

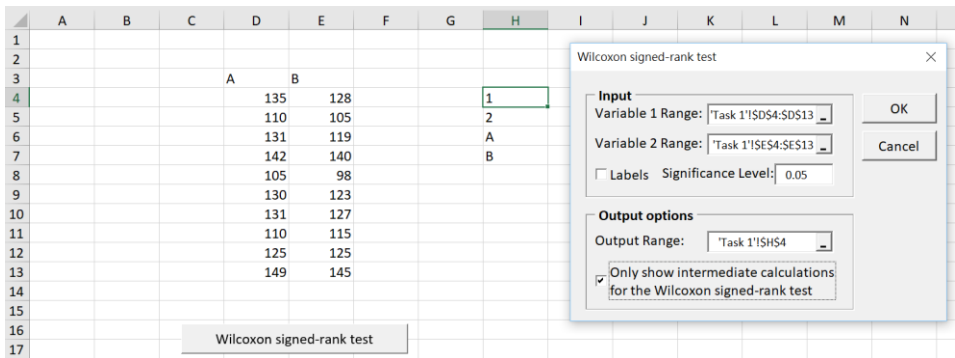


Figure 1.9

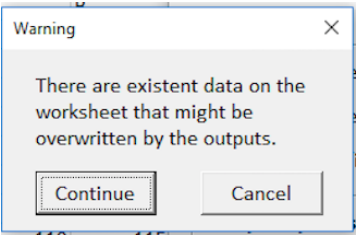


Figure 1.10

If there are existent data on the worksheet where the user would like the output to be shown, there will be a warning appears. It allows the user to make corrections or ignore. Figure 1.9 shows that the content already exists in the output range we need, so in order to ensure that the output will not overwrite the original data, a warning prompt box will pop up (figure 1.10), asking the user to determine again whether to continue the output and reminding the user that this action will overwrite the previous one.

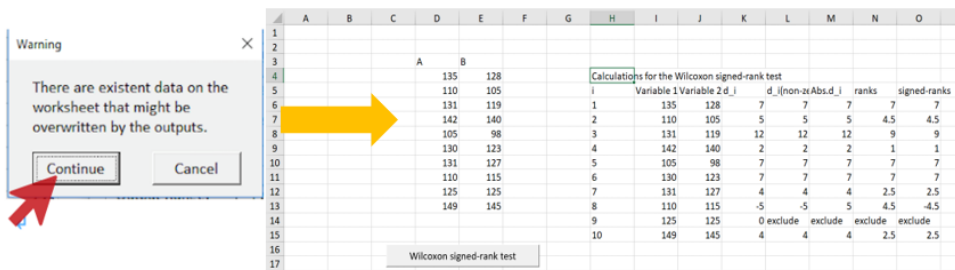


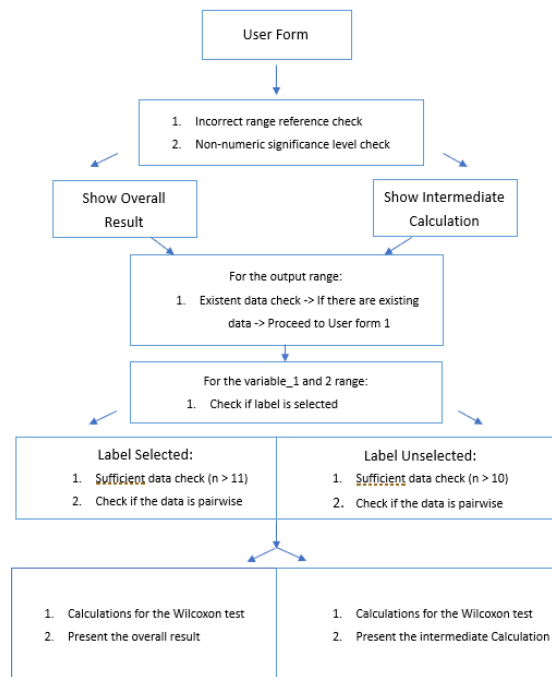
Figure 1.11

If the user chooses “Continue” as figure 1.11 shows, then the new output will overwrite the existent data on the worksheet. On the other hand, if the “Cancel” option is selected, the calculation will stop, and no new output will overwrite the previous one.

**VBA code explanation:**

Our VBA code is stored under Forms: User Form file.

A structure of the code is listed below:



### Step 1: call and initiate user form:

In this step, first we create a module to call the user form. Only when the user clicks the button “Wilcoxon signed-rank test” on the worksheet, the user form will pop up.

```

Sub Wilcoxon_Click()
    UserForm.Show
End Sub
  
```

We initiate the user form and set up the significant level default to 0.05.

```

Private Sub UserForm_Initialize()
    ' Set default alpha as 0.05
    Me.alpha_TB.Text = 0.05
End Sub
  
```

Click to active: The user form will only be active after the “OK” button have been clicked.

## Private Sub OK\_Button\_Click()

Click to stop calculation: If click the “Cancel” button on the user form, it will stop the performance.

```
Private Sub Cancel_Button_Click()  
    Unload Me  
End Sub
```

### Step 2: Checking the requirements before launching calculations

#### 1. Incorrect range reference check:

```
' Incorrect range references check-----  
Dim Var1_blsRange, Var2_blsRange, output_blsRange As Boolean  
On Error Resume Next  
    Var1_blsRange = IsObject(Range(Var1Range_RefEdit.Value))  
    Var2_blsRange = IsObject(Range(Var2Range_RefEdit.Value))  
    output_blsRange = IsObject(Range(Range_RefEdit.Value))  
On Error GoTo 0
```

In the codes above, the variable range\_1, variable range\_2 and output range are been defined as a Boolean variable, meaning it only holds value true or false.

In the following statement, the variable's range are defined as true only if it's from the value returned by the **RefEdit** function (allows user to select a range of cell on a worksheet). It ensures that the range reference should from the range of cell on a worksheet.

This is achieved by using the **IsObject** function which checks if the variables' range satisfied our defined requirement (values from **RefEdit** function) and returns a true or false value.

#### *Error handling:*

The On error resume next line ensures the codes will be executed when run-time error occurs, which ensures the stability of the code. The On Error GoTo 0 ends the error handling procedures.

#### *Warning Message:*

From the above, if the variables and outputs range are false, meaning they are not from the range of a cell on a worksheet, the following code will be executed:

```

If Var1_bIsRange = False Then
    MsgBox "The range of variable 1 is not valid"
    Var1Range_RefEdit.Value = vbNullString
    Var1Range_RefEdit.SetFocus
    Exit Sub
End If

If Var2_bIsRange = False Then
    MsgBox "The range of variable 2 is not valid"
    Var2Range_RefEdit.Value = vbNullString
    Var2Range_RefEdit.SetFocus
    Exit Sub
End If

If output_bIsRange = False Then
    MsgBox "The range of output is not valid"
    Range_RefEdit.Value = vbNullString
    Exit Sub
End If

```

In any of the above scenario, if the range reference is incorrect and error message will appear, and the program will stop executing (Exit Sub), which ensures that the input range reference is correct before it proceeds to calculating the Wilcoxon test.

## 2. Non-numeric significance level check:

```

' Non-numeric significance level check
If Not IsNumeric(Me.alpha_TB.Value) Then
    MsgBox "Significance level should be numeric."
    Me.alpha_TB.Value = 0.05
    Exit Sub
End If

```

*Warning message:*

The above code used If statement to ensure that the input in the significant level cell is numeric. If it is not numeric value, the warning message will appear, and the program will stop executing. In addition, the significant level is set default to 0.05.

The final step before launching the calculation is to select the types of output. There are two types of output: Overall result VS. Intermediate calculations.

### **Step 3: Select two types of output**

#### Output type: Overall result

Set up the check box on the User Form. If it is not ticked, then the following code will be executed and only the basic result of Wilcoxon signed-rank test will show.

```

' Show overall results -----
If Me.OnlyCalculation CheckBox.Value = False Then

```

#### Output type: Intermediated result

Set up the check box on the User Form. If it is ticked, then the following code will be executed and only the intermediate calculation table of the Wilcoxon signed-rank test will show.



```
' Only show the intermediate calculations-----
If Me.OnlyCalculation_CheckBox.Value = True Then
```

#### **Step 4: Checking the output range and label selection check**

Ensuring the output range is the user selected range of cell on the worksheet.

```
' Select the output range
Dim output_rng As Range
Set output_rng = Range(Me.Range_RefEdit.Value)
output_rng.Select
```

Before proceeding to calculation: the following requirements are checked:

##### **1. Existent data check**

```
' Existent data check
Dim results_rng As Range
Set results_rng = Range(output_rng, output_rng.Offset(11, 1))
If WorksheetFunction.CountA(results_rng) <> 0 Then
    UserForm1.Show
    Unload Me
    Exit Sub
End If
```

Checking if there are existing data on the output range cell, if there are existing data on the output range which might be overwritten: **UserForm\_1 will be activated and this user form will be stopped.**

##### **2. Label selection check: If else statement**

- *Without labels:*

```
' Select data without labels
If Me.Label_CB.Value = False Then
    ' Check sufficient data has been selected
    Set rng1 = Range(Var1Range_RefEdit.Value)
    Set rng2 = Range(Var2Range_RefEdit.Value)
    If rng1.Rows.count < 10 Then
        MsgBox "The number of data should be greater than 10."
        Var1Range_RefEdit.Value = vbNullString
        Var1Range_RefEdit.SetFocus
        Exit Sub
    End If
    If rng2.Rows.count < 10 Then
        MsgBox "The number of data should be greater than 10."
        Var2Range_RefEdit.Value = vbNullString
        Var2Range_RefEdit.SetFocus
        Exit Sub
    End If
    ' Check pairwise data has been selected
    If rng1.Rows.count <> rng2.Rows.count Then
        MsgBox "There should be pairwise data."
        Exit Sub
    End If
    ' Paste the 2 variable values
    ActiveCell.Offset(1, 1) = "Variable 1"
    ActiveCell.Offset(1, 2) = "Variable 2"
    icol = Cells(Rows.count, 2).End(xlUp).Offset(0, 0).Column
    ActiveCell(3, icol).Resize(rng1.Rows.count, rng1.Columns.count).Value = rng1.Value
    icol = Cells(Rows.count, 3).End(xlUp).Offset(0, 0).Column
    ActiveCell(3, icol).Resize(rng2.Rows.count, rng2.Columns.count).Value = rng2.Value
```

The above code is activated when the label box is not ticked. Before proceeding further, the code will check if sufficient data have been selected for both range\_1 and range\_2 variables. The minimum amount of data is set to 10, so we will check if the number of data is greater than 10. If the requirement is not satisfied: a warning message will appear, and the program will stop executing. The code will also check if pairwise data has been selected. If the requirement is not satisfied: a warning message will appear, and the program will stop executing. The final step is to paste the selected range\_1 and range\_2 variables for calculation.

- *With labels*

```

Else
' Select data with labels
' Check sufficient data has been selected
Set rng1 = Range(Var1Range_RefEdit.Value)
Set rng2 = Range(Var2Range_RefEdit.Value)
If rng1.Rows.count < 11 Then
MsgBox "The number of data should be greater than 10."
Var1Range_RefEdit.Value = vbNullString
Var1Range_RefEdit.SetFocus
Exit Sub
End If
If rng2.Rows.count < 11 Then
MsgBox "The number of data should be greater than 10."
Var2Range_RefEdit.Value = vbNullString
Var2Range_RefEdit.SetFocus
Exit Sub
End If
' Check pairwise data has been selected
If rng1.Rows.count <> rng2.Rows.count Then
MsgBox "There should be pairwise data."
Exit Sub
End If
' Paste the 2 variable values
icol = Cells(Rows.count, 2).End(xlUp).Offset(0, 0).Column
ActiveCell(2, icol).Resize(rng1.Rows.count, rng1.Columns.count).Value = rng1.Value
icol = Cells(Rows.count, 3).End(xlUp).Offset(0, 0).Column
ActiveCell(2, icol).Resize(rng2.Rows.count, rng2.Columns.count).Value = rng2.Value
End If

```

The only different between with and without labels code is the highlighted part. The minimum amount of data is set to 10, so we will check if the number of data is greater than 11 because we considered the existence of the label.

## Step 5: Calculations

### 1. Wilcoxon Signed-rank test: Intermediate calculation

```

' Add title
ActiveCell = "Calculations for the Wilcoxon signed-rank test"

' Calculations for the Wilcoxon signed-rank test -----
' Create column i
ActiveCell.Offset(1, 0) = "i"
Dim Lst As Long
Lst = Range(ActiveCell.Offset(2, 2), ActiveCell.Offset(2, 2).End(xlDown)).Rows.count
With ActiveCell.Offset(2, 0)
.Value = "1"
.AutoFill Destination:=ActiveCell.Offset(2, 0).Resize(Lst), Type:=xlFillSeries
End With

' Create column d.i
ActiveCell.Offset(1, 3) = "d.i"
ActiveCell.Offset(2, 3).FormulaR1C1 = "=RC[-2]-RC[-1]"
ActiveCell.Offset(2, 3).AutoFill Destination:=ActiveCell.Offset(2, 3).Resize(Lst)

' Create column d.i(non-zero)
ActiveCell.Offset(1, 4) = "d.i(non-zero)"
ActiveCell.Offset(2, 4).FormulaR1C1 = "=IF(OR(RC[-1]>0,RC[-1]<0),RC[-1],""exclude"")"
ActiveCell.Offset(2, 4).AutoFill Destination:=ActiveCell.Offset(2, 4).Resize(Lst)

' Create column Abs.d.i
ActiveCell.Offset(1, 5) = "Abs.d.i"
ActiveCell.Offset(2, 5).FormulaR1C1 = "=IFERROR(ABS(RC[-1]),""exclude"")"
ActiveCell.Offset(2, 5).AutoFill Destination:=ActiveCell.Offset(2, 5).Resize(Lst)

' Create column ranks
ActiveCell.Offset(1, 6) = "ranks"
loop_num = Lst - 1
For i = 0 To loop_num
active_row = 2 + i
start_row = 0 - i
last_row = loop_num - i
ActiveCell.Offset(active_row, 6)
.FormulaR1C1 = "=IFERROR(RANK.AVG(RC[-1],R[" & start_row & "]C[-1]:R[" & last_row & "]C[-1],1),""exclude"")"
Next i

' Create column signed ranks
ActiveCell.Offset(1, 7) = "signed-ranks"
ActiveCell.Offset(2, 7).FormulaR1C1 =
"=IF(RC[-3]>0,RC[-1],IF(RC[-3]<0,-RC[-1],""exclude""))"
ActiveCell.Offset(2, 7).AutoFill Destination:=ActiveCell.Offset(2, 7).Resize(Lst)

```

In step 4, we have pasted the selected range\_1 and range\_2 variables to the relative locations of user-selected output range. So here we first create a column “i” before our variables to specify the number of variables. Then we start to calculate from the pairwise differences to the ranks of the Wilcoxon Signed-rank test. In each column, we assign a title on the top and use functions to calculate the first value. Then use autofill to fill out all the cells in this column to calculate the values with the same functions.

### 2. Wilcoxon Signed-rank test: Overall result

The overall result was calculated based on the previous calculation, and the following code is executed. We assign a title for each result and calculate the values.

```

' Wilcoxon signed-rank test result -----
' Add title
ActiveCell.Offset(0, 9) = "Wilcoxon signed-rank test"
' Calculate n
ActiveCell.Offset(1, 9) = "n"
ActiveCell.Offset(1, 10).FormulaR1C1 = "=COUNT(R[1]C[-10]:R[1] & Lst & "C[-10])"
' Calculate nr
ActiveCell.Offset(2, 9) = "nr"
ActiveCell.Offset(2, 10).FormulaR1C1 = "=R[-1]C-COUNTIF(R[0]C[-7]:R[1] & loop_num & "C[-7], "<0")"
' Calculate SR(+)
ActiveCell.Offset(3, 9) = "SR(+)"
loop_num = loop_num + 1
ActiveCell.Offset(3, 10).FormulaR1C1 = "=SUMIF(R[-1]C[-7]:R[1] & loop_num & "C[-7], ">0", R[-1]C[-4]:R[1] & loop_num & "C[-4])"
' Calculate SR(-)
ActiveCell.Offset(4, 9) = "SR(-)"
loop_num = loop_num + 1
ActiveCell.Offset(4, 10).FormulaR1C1 = "=SUMIF(R[-2]C[-7]:R[1] & loop_num & "C[-7], "<0", R[-2]C[-4]:R[1] & loop_num & "C[-4])"
' Calculate T
ActiveCell.Offset(5, 9).FormulaR1C1 = "T"
ActiveCell.Offset(5, 10).FormulaR1C1 = "=MIN(R[-2]C:R[-1]C)"
' Calculate E[T]
ActiveCell.Offset(6, 9) = "E[T]"
ActiveCell.Offset(6, 10).FormulaR1C1 = "=R[-4]C*(R[-4]C+1)/4"
' Calculate Var(T)
ActiveCell.Offset(7, 9) = "Var(T)"
ActiveCell.Offset(7, 10).FormulaR1C1 = "=R[-5]C*(R[-5]C+1)*(2*R[-5]C+1)/24"
' Calculate z
ActiveCell.Offset(8, 9) = "z"
ActiveCell.Offset(8, 10).FormulaR1C1 = "=R[-3]C-R[-2]C/(R[-1]C^0.5)"
' Calculate p-value
ActiveCell.Offset(9, 9) = "p-value"
ActiveCell.Offset(9, 10).FormulaR1C1 = "=2*(1-NORM.DIST(ABS(R[-1]C),0,1,TRUE))"
' Assign alpha
ActiveCell.Offset(10, 9) = "α"
If Me.alpha_TB.Value < 0.05 Then
    ActiveCell.Offset(10, 10) = Me.alpha_TB.Value
Else
    ActiveCell.Offset(10, 10) = 0.05
End If
' Reject or accept
ActiveCell.Offset(11, 9) = "result"
ActiveCell.Offset(11, 10).FormulaR1C1 = "=IF(R[-1]C>R[-2]C, ""Reject H0"", ""Accept H0"")"

```

Since only the basic result is needed, so we delete the intermediate calculation and only present the result.

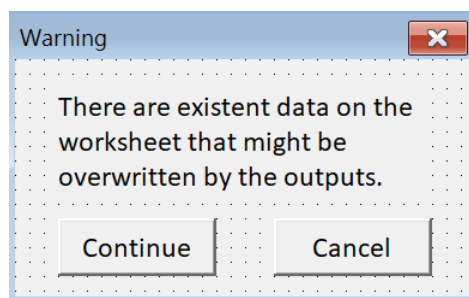
```

' Delete the calculations and keep the results
Range(ActiveCell.Offset(0, 9), ActiveCell.Offset(11, 10)).Select
Selection.Copy
Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone, SkipBlanks _
:=False, Transpose:=False
output_rng.Select
Range(ActiveCell, ActiveCell.Offset(0, 7)).Select
Range(Selection, Selection.End(xlDown)).Select
Selection.ClearContents
Range(ActiveCell.Offset(0, 9), ActiveCell.Offset(11, 10)).Select
Selection.Copy
output_rng.Select
ActiveSheet.Paste
Range(ActiveCell.Offset(0, 9), ActiveCell.Offset(11, 10)).Select
Selection.ClearContents
Range(output_rng, output_rng.Offset(11, 1)).Select
End If

```

### User Form 1:

The VBA user form 1 is activated when there are existent data on the worksheet. Users can decide whether to continue the calculation or stop to avoid overwriting the existent data.



If “Cancel” button has been click, the program will be stopped.

```

Private Sub Cancel_CB_Click()
    Unload Me
End Sub

```

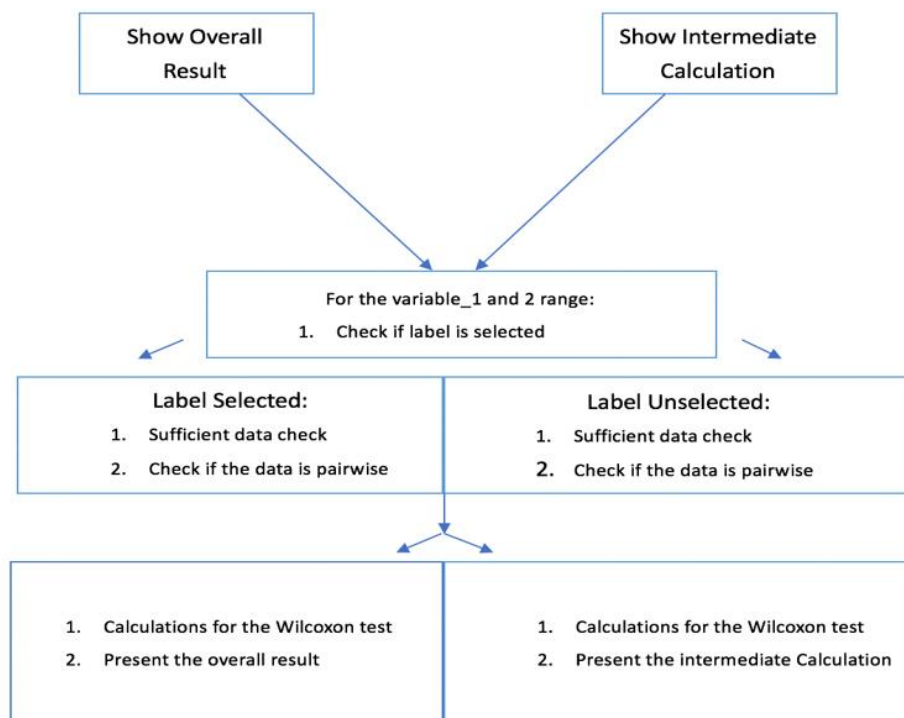
When click “Continue”, the code attached in User Form 1 will be activated.

```

Private Sub Continue_CB_Click()

```

The following process will be activated: the codes for calculation are the same as the code on user form. However, some error checks, such as incorrect range reference check and non-numeric significance level check, have been performed before, so they will not be run again here.



After performing all the steps above, the VBA program successfully replicates the Wilcoxon signed-rank test.

- Task 2: R implementation of the Wilcoxon signed-rank test

```
wilcoxon.test <- function(x, y, significance.level = 0.05,
                          output.type = c("basic", "inter.cal"))
```

Figure 2.1

```
# Set 2 types of output -----
if (missing(output.type)) output.type <- "basic"
if (output.type == "basic") {
  print(wilcoxon.result)
} else if (output.type == "inter.cal") {
  print(wilcoxon.cal)
} else {
  stop("Incorrect type of output: output.type should be 'basic' or 'inter.cal'.")
}
```

Figure 2.2

The function we have created in R is called “wilcoxon.test” (figure 2.1), it has two necessary arguments for the data inputs x and y. There are two optional arguments, one for significance level, and the other for output type. In order to set significance level by default to 5%, “significance.level = 0.05” has been directly specified in the function input. For the other argument output types, there are two options “basic” and “inter.cal” (intermediate calculation), and an if statement was specified within the function (figure 2.2), saying that the output level is “basic” if this argument is missing. However, if the user enters a wrong type of output, not “basic” or “inter.cal”, then the calculation will be stopped immediately.

```
> var1 <- c(135, 110, 131, 142, 105, 130, 131, 110, 125, 149)
> var2 <- c(128, 105, 119, 140, 98, 123, 127, 115, 125, 145)
> wilcoxon.test(x = var1, y = var2, output.type = "basic")
      wilcoxon signed-rank test
n                10
nr                 9
SR.positive       40.5
SR.negative        4.5
wilcoxon.statistic 4.5
mean.T            22.5
var.T             71.25
z                -2.13246
p.value           0.03297
significance.level 0.05
result            reject H0
> wilcoxon.test(x = var1, y = var2, output.type = "inter.cal")
  x_i y_i d_i d_i(non-zero) Abs.d_i ranks signed-ranks
1 135 128 7          7          7 7          7
2 110 105 5          5          5 4.5        4.5
3 131 119 12         12         12 9          9
4 142 140 2          2          2 1          1
5 105 98 7          7          7 7          7
6 130 123 7          7          7 7          7
7 131 127 4          4          4 2.5        2.5
8 110 115 -5         -5          5 4.5       -4.5
9 125 125 0      exclude exclude exclude exclude
10 149 145 4          4          4 2.5        2.5
```

Figure 2.3

In terms of the calculation with the Wilcoxon signed-rank test, we have created two data frames with calculation implementation within the function. The two data frames actually correspond to two different output types as figure 2.3 above shows. In implementation of R codes, we first calculate the values of each column, then use `cbind()` to combine the results into the intermediate calculation data frame. After getting the output of the intermediate calculation, we calculate the values of the basic overall result of the Wilcoxon signed-rank test and use `rbind()` to store all the values in a data frame.

Similar to the VBA version, in R we also created some warning messages for different kinds of cases, and it may have some automatic adjustment for the error where it is applicable.

### Invalid data type check

```
> var2 <- c("128", "105", 119, 140, 98, 123, 127, 115, 125, 145)
> wilcoxon.test(var1,var2)
      wilcoxon signed-rank test
n                10
nr                 9
SR.positive       40.5
SR.negative        4.5
Wilcoxon.statistic 4.5
mean.T            22.5
var.T             71.25
z                -2.13246
p.value           0.03297
significance.level 0.05
result            reject H0
Warning message:
In wilcoxon.test(var1, var2) :
  y is not numeric and is transformed automatically.
```

Figure 2.4

Figure 2.4 is the case that some elements in data input are factors rather than numeric, so the factors can be transformed to numeric automatically if possible.

```
# Automatic adjustments of input -----
# Transform data to numeric
if (any(!is.numeric(x))) warning("x is not numeric and is transformed automatically.")
x <- as.numeric(x)
if (any(!is.numeric(y))) warning("y is not numeric and is transformed automatically.")
y <- as.numeric(y)
```

Figure 2.5

Here we use an if statement with a warning function to check the data type of inputs. If there is any element not numeric number in x or y, it will be automatically transformed into numeric and the calculation of Wilcoxon signed-rank test can be continued. In the meanwhile, it will show a warning message saying that the transforming process has been executed.

### Sufficient data check

```
> var1 <- c(135, 110, 131, 142, 105, 130, 131, 110, 125, 149)
> var2 <- c(128, 105, 119, 140, 98, 123, 127, 115)
> wilcoxon.test(var1,var2)
Error in wilcoxon.test(var1, var2) :
  The number of data in y should be greater than 10.
```

Figure 2.6

Figure 2.6 shows that the data input y only has eight elements, so calculation was stopped and a warning message was also created saying that the number of data in y should be greater than 10.

```
# sufficient data check
if (length(x) < 10 ) stop("The number of data in x should be greater than 10.")
if (length(y) < 10 ) stop("The number of data in y should be greater than 10.")
```

Figure 2.7

A length function is used here to check how many elements in x or y. If the length of x or y is shorter than 10 elements, the calculation will be stopped and give a warning message.

**Pairwise data check**

```
> var1 <- c(135, 110, 131, 142, 105, 130, 131, 110, 125, 149)
> var2 <- c(128, 105, 119, 140, 98, 123, 127, 115, 125, 145, 138, 111)
> wilcoxon.test(var1, var2)
Error in wilcoxon.test(var1, var2) : x and y should be pairwise data.
```

Figure 2.8

If both x and y have more than 10 elements but have different lengths, there will be a problem to perform the Wilcoxon signed-rank test. Figure 2.8 shows that there are 10 elements in “var1” but 12 elements in “var2”, so the calculation was also stopped, and this error reminds that x and y should be pairwise data.

```
# pairwise data check
if (length(x) != length(y)) stop("x and y should be pairwise data.")
```

Figure 2.9

In codes, we use length function to check whether x and y has the same number of elements. If it does not fulfil this condition, then the calculation will be stopped.

**Numeric significance level check**

```
> wilcoxon.test(var1, var2, significance.level = "a" )
Error in wilcoxon.test(var1, var2, significance.level = "a") :
  Significance level should be numeric.
```

Figure 2.10

Significance level should be numeric but in the case of what the figure 2.10 shows above, the significance level is “a”, which is invalid of course, so an error message exists saying that significance level should be numeric.

```
# non-numeric significance level check
if (!is.numeric(significance.level)) stop("Significance level should be numeric.")
```

Figure 2.11

Figure 2.11 shows that an is.numeric function is used to make sure that the input element of the significance level should be numeric or the Wilcoxon function will be stopped.

**Missing values (NAs) manipulation**

```
> var1 <- c(135, 110, 131, 142, 105, 130, NA, 110, 125, 149, 138)
> var2 <- c(128, 105, 119, 140, NA, 123, 127, 115, 125, 145, 182)
> wilcoxon.test(var1, var2)
Error in wilcoxon.test(var1, var2) :
  The number of data after removing NAs should be greater than 10.
In addition: warning message:
In wilcoxon.test(var1, var2) :
  There are NAs in x or y and those pairs are removed automatically.
```

Figure 2.12

The function also carries out automatic adjustments to remove NAs in the data. In the case of above, there is one “NA” in each dataset, so here the NA and its corresponding data in the other vector are dropped to maintain a corresponding relationship between x and y (105 in var1 corresponding to NA in var2 and NA in var1 corresponding to 127 in var2 are removed). After dropping the two pairwise data,

there are only 9 elements in x and y, so the warning message saying that the number of data after removing NAs should be greater than 10.

```
# Remove NAs in inputs
if (any(is.na(wilcoxon.cal))) warning("There are NAs in x or y and those pairs are removed automatically.")
wilcoxon.cal <- wilcoxon.cal[complete.cases(wilcoxon.cal),]
# sufficient data check
if ((dim(wilcoxon.cal["x_i"])[1]) < 10 ) stop("The number of data after removing NAs should be greater than 10.")
```

Figure 2.13

Here we use an `is.na` function to find out any NA value in x or y. Then use the `complete.cases` function to subset all the pairwise data without NAs. After removing NAs, we check that the number of the first dimension of the pairwise data, which equals to the number of data in x, should be greater than 10 as well.