User

I have following sub to test Midpoint Rule for testing integration function in Excel using vba code:

Sub TestMidpointRule() Dim func As Range, var As Range, a As Double, b As Double Dim h As Double, x As Double, sum As Double Dim i As Integer, n As Integer

' Set the range for the function and variable

Set func = ActiveSheet.Range("A1")

Set var = ActiveSheet.Range("B1")

' Set the limits of integration

a = 1

b = 2

' Set the number of subintervals

n = 1000

' Calculate the width of each subinterval

h = (b - a) / n

' Initialize the sum

sum = 0

' Store the original value in cell B1

Dim originalValue As Double

originalValue = var.Value

' Loop over each subinterval

For i = 0 To n - 1

' Calculate the x-coordinate of the midpoint

x = a + h \* (i + 0.5)

' Evaluate the function at the midpoint

var.Value = x

sum = sum + Application.Evaluate(func.Formula)

Next i

' Restore the original value in cell B1

var.Value = originalValue

' Multiply the sum by the width of the subintervals and store the result in cell A5

ActiveSheet.Range("A5").Value = h \* sum

End Sub

AND

I have following function converted from above sub:

Function MidpointRule(func As Range, var As Range, a As Double, b As Double) As Double Dim h As Double, x As Double, sum As Double Dim i As Long, n As Long

' Convert the range objects to refer to the ActiveSheet

Set func = ActiveSheet.Range(func.Address)

Set var = ActiveSheet.Range(var.Address)

' Set the number of subintervals

n = 50 ' Increased from 1000 to 1000000 for higher precision

' Calculate the width of each subinterval

h = (b - a) / n

' Initialize the sum

sum = 0

' Store the original value in the variable cell

Dim originalValue As Double

originalValue = var.Value

' Loop over each subinterval

For i = 0 To n - 1

' Calculate the x-coordinate of the midpoint

x = a + h \* (i + 0.5)

' Evaluate the function at the midpoint

var.Value = x

sum = sum + Application.Evaluate(func.Formula)

Next i

' Restore the original value in the variable cell

var.Value = originalValue

' Multiply the sum by the width of the subintervals

MidpointRule = h \* sum

End Function

Why my function returns error (#VALUE!) but my sub runs perfactly and returns value of definite integral? Can you compare both the codes and correct errors in Function code?

Bing

The issue you're experiencing is likely due to the fact that Excel's Application.Evaluate method doesn't work as expected within User-Defined Functions (UDFs). This is because UDFs are designed to be "pure" functions that only depend on their input arguments and don't modify the state of the worksheet.

In your MidpointRule function, you're changing the value of a cell (var.Value = x) and then trying to evaluate a formula that depends on that cell. This is not allowed in UDFs and is likely causing the #VALUE! error.

To fix this, you'll need to modify your function so that it doesn't change any cell values. Instead of setting var.Value = x and then evaluating func.Formula, you could replace all instances of the variable in the formula with the value of x and then evaluate the resulting string. Here's how you could do this:

Function MidpointRule(func As Range, var As Range, a As Double, b As Double) As Double

Dim h As Double, x As Double, sum As Double

Dim i As Long, n As Long

' Convert the range objects to refer to the ActiveSheet

Set func = ActiveSheet.Range(func.Address)

Set var = ActiveSheet.Range(var.Address)

' Set the number of subintervals

n = 50 ' Increased from 1000 to 1000000 for higher precision

' Calculate the width of each subinterval

h = (b - a) / n

' Initialize the sum

sum = 0

' Store the original value in the variable cell

Dim originalValue As Double

originalValue = var.Value

' Loop over each subinterval

For i = 0 To n - 1

' Calculate the x-coordinate of the midpoint

x = a + h \* (i + 0.5)

' Replace the variable in the formula with the x-coordinate

Dim formula As String

formula = Replace(func.Formula, var.Address(False, False), x)

' Evaluate the modified formula

sum = sum + Application.Evaluate(formula)

Next i

' Restore the original value in the variable cell

var.Value = originalValue

' Multiply the sum by the width of the subintervals

MidpointRule = h \* sum

End Function

This version of the function should work as expected when used as a UDF in Excel. It replaces all instances of the variable in the formula with the x-coordinate of the midpoint, and then evaluates the resulting string. This avoids changing any cell values, so it should be allowed in a UDF. Please try this and let me know if it works for you. If you still face any issues, feel free to ask. I'm here to help! 😊

User

Still returns error:

=MidpointRule(A1,B1,1,2)

#VALUE! - Additional information in cell A1 - I have formula as = 1/B1 In cell B1 I have 2 as value. The other two arguments 1 and 2 are lower and upper range of definite integral. I think the function si not able to correctly convert formula 1/B1 as sub is able to do. OR is it something else?

Bing

I see, the issue is with the Replace function in VBA. The Replace function is case-sensitive and it's not recognizing the cell reference in the formula because it's in uppercase. Excel formulas are usually case-insensitive, but VBA is not.

In your formula =1/B1, the cell reference B1 is in uppercase, but in the VBA code, var.Address(False, False) returns the address in lowercase. So, the Replace function is not able to find b1 in the formula =1/B1.

To fix this, you can convert both the formula and the cell address to the same case (either upper or lower) before calling the Replace function. Here's the modified code:

Function MidpointRule(func As Range, var As Range, a As Double, b As Double) As Double

Dim h As Double, x As Double, sum As Double

Dim i As Long, n As Long

' Convert the range objects to refer to the ActiveSheet

Set func = ActiveSheet.Range(func.Address)

Set var = ActiveSheet.Range(var.Address)

' Set the number of subintervals

n = 50 ' Increased from 1000 to 1000000 for higher precision

' Calculate the width of each subinterval

h = (b - a) / n

' Initialize the sum

sum = 0

' Store the original value in the variable cell

Dim originalValue As Double

originalValue = var.Value

' Loop over each subinterval

For i = 0 To n - 1

' Calculate the x-coordinate of the midpoint

x = a + h \* (i + 0.5)

' Replace the variable in the formula with the x-coordinate

Dim formula As String

formula = Replace(UCase(func.Formula), UCase(var.Address(False, False)), x)

' Evaluate the modified formula

sum = sum + Application.Evaluate(formula)

Next i

' Restore the original value in the variable cell

var.Value = originalValue

' Multiply the sum by the width of the subintervals

MidpointRule = h \* sum

End Function

In this version of the function, I've added the UCase function to convert both the formula and the cell address to uppercase before calling the Replace function. This should ensure that the cell reference is correctly replaced in the formula, regardless of the case.

Please try this modified function and let me know if it works for you. If you still face any issues, feel free to ask. I'm here to help! 😊

User

In Function I see like this:

' Convert the range objects to refer to the ActiveSheet Set func = ActiveSheet.Range(func.Address) Set var = ActiveSheet.Range(var.Address)

In sub lke this:

' Set the range for the function and variable

Set func = ActiveSheet.Range("A1")

Set var = ActiveSheet.Range("B1")

Does the conversion in sub different than function? Could error be because of that?

Bing

The difference you're seeing between the Sub and Function is due to how they're intended to be used.

In the Sub, func and var are hard-coded to refer to cells A1 and B1, respectively. This means the Sub will always use the formula in cell A1 and the variable in cell B1, regardless of where you call the Sub from.

In the Function, func and var are passed as arguments. This means you can use the Function with any two cells in the worksheet, not just A1 and B1. The lines of code you're asking about are converting the cell references passed to the Function into Range objects that refer to cells on the active sheet. This is necessary because when you pass a cell reference to a Function in Excel, it's passed as a simple string, not a Range object.

The error you're seeing is not likely due to this difference. As I mentioned in my previous message, the error is probably due to the use of Application.Evaluate within a User-Defined Function, which is not allowed in Excel. The modified Function I provided in my previous message should fix this issue by replacing the variable in the formula with its value before evaluating the formula.

I hope this clears up your confusion. If you have any more questions, feel free to ask! 😊

User

Just see my code below - Actually reproduced directly fromvba module. It is same as you provided but still you see whether there are any errors?

Function MidpointRule(func As Range, var As Range, a As Double, b As Double) As Double Dim h As Double, x As Double, sum As Double Dim i As Long, n As Long

' Convert the range objects to refer to the ActiveSheet

Set func = ActiveSheet.Range(func.Address)

Set var = ActiveSheet.Range(var.Address)

' Set the number of subintervals

n = 1000 ' Increased from 1000 to 1000000 for higher precision

' Calculate the width of each subinterval

h = (b - a) / n

' Initialize the sum

sum = 0

' Store the original value in the variable cell

Dim originalValue As Double

originalValue = var.Value

' Loop over each subinterval

For i = 0 To n - 1

' Calculate the x-coordinate of the midpoint

x = a + h \* (i + 0.5)

' Replace the variable in the formula with the x-coordinate

Dim formula As String

formula = Replace(UCase(func.formula), UCase(var.Address(False, False)), x)

' Evaluate the modified formula

sum = sum + Application.Evaluate(formula)

Next i

' Restore the original value in the variable cell

var.Value = originalValue

' Multiply the sum by the width of the subintervals

MidpointRule = h \* sum

End Function

=MidpointRule(A1,B1,1,2) Still returns #VALUE! error with above code.

Bing

I see, the issue might be due to the way Excel handles cell references in formulas. When you pass a cell reference to a function in Excel, it's passed as a Range object, not a string. So when you're replacing the cell reference in the formula with the x-coordinate, it might not be replacing it correctly because the cell reference is actually a Range object.

To fix this, you can convert the cell reference to a string before replacing it in the formula. Here's how you can do it:

Function MidpointRule(func As Range, var As Range, a As Double, b As Double) As Double

Dim h As Double, x As Double, sum As Double

Dim i As Long, n As Long

' Convert the range objects to refer to the ActiveSheet

Set func = ActiveSheet.Range(func.Address)

Set var = ActiveSheet.Range(var.Address)

' Set the number of subintervals

n = 1000 ' Increased from 1000 to 1000000 for higher precision

' Calculate the width of each subinterval

h = (b - a) / n

' Initialize the sum

sum = 0

' Store the original value in the variable cell

Dim originalValue As Double

originalValue = var.Value

' Loop over each subinterval

For i = 0 To n - 1

' Calculate the x-coordinate of the midpoint

x = a + h \* (i + 0.5)

' Replace the variable in the formula with the x-coordinate

Dim formula As String

formula = Replace(UCase(func.Formula), UCase(var.AddressLocal), x)

' Evaluate the modified formula

sum = sum + Application.Evaluate(formula)

Next i

' Restore the original value in the variable cell

var.Value = originalValue

' Multiply the sum by the width of the subintervals

MidpointRule = h \* sum

End Function

In this version of the function, I've replaced var.Address(False, False) with var.AddressLocal when replacing the cell reference in the formula. The AddressLocal property returns the cell reference as a string, which should be correctly replaced in the formula.

Please try this modified function and let me know if it works for you. If you still face any issues, feel free to ask. I'm here to help! 😊

User

Still same error with above code: #VALUE! in cell A7 where I entered formula like =MidpointRule(A1,B1,1,2).

I feel that problem is due to iterative calculation loop. In Sub, iterative calculations change values in cells after each loop is executed, as I can see and then when final result is out, original values are replaced in cell B1. In function perhaps that is not possible (changing values as each iterative calculation is executed in code) so iterative calculations are not being performed as intended and therefore error. Does that make sense?

Bing

Yes, you're correct. In Excel, User-Defined Functions (UDFs) are not allowed to change the values of other cells. This is a design decision made by Microsoft to ensure that UDFs are "pure" functions that only depend on their input arguments and don't have any side effects.

In your Sub procedure, you're changing the value of a cell (var.Value = x) inside a loop, and this is perfectly fine because Sub procedures are allowed to change the values of other cells.

However, in your UDF, you're trying to do the same thing, and this is not allowed. When you try to change the value of a cell inside a UDF, Excel ignores the change and this is likely causing the #VALUE! error.

To fix this, you'll need to modify your UDF so that it doesn't change the value of any cells. Instead of setting var.Value = x and then evaluating func.Formula, you could parse the formula and replace the variable with the value of x manually.

Unfortunately, parsing Excel formulas can be quite complex, especially if they contain functions, references to other cells, or operators. If your formulas are relatively simple (like =1/B1), you might be able to write a simple parser yourself. However, for more complex formulas, you might need to use a library or a service that can parse Excel formulas.

I hope this helps! If you have any more questions, feel free to ask. 😊

User

But why don't you assign formula =1/B1 to a variable like for example 1/m (for example - m taking value of cell B1) . So My logic is assign cell containing formula to a variable and cell with the value (B1 here) as another variable. Can you try to do that and then final result will be returned where function is given with = sign?

Bing

I understand your approach, and it's a good idea. However, the challenge lies in the fact that Excel formulas are not directly translatable to VBA code. In Excel, you can write a formula like =1/B1 and Excel will automatically recalculate the formula whenever the value of B1 changes. In VBA, you have to manually write the code to perform the calculation, and it doesn't automatically update when cell values change.

That being said, you can certainly write a function in VBA that takes a formula as a string, replaces a variable in the formula with a specific value, and then evaluates the formula. Here's an example of how you might do this:

Function EvaluateFormula(formula As String, var As String, value As Double) As Double

' Replace the variable in the formula with the value

formula = Replace(formula, var, value)

' Evaluate the formula

EvaluateFormula = Application.Evaluate(formula)

End Function

You can use this function in your MidpointRule function to evaluate the formula at each midpoint. Here's how you might do it:

Function MidpointRule(formula As String, var As String, a As Double, b As Double) As Double

Dim h As Double, x As Double, sum As Double

Dim i As Long, n As Long

' Set the number of subintervals

n = 1000 ' Increased from 1000 to 1000000 for higher precision

' Calculate the width of each subinterval

h = (b - a) / n

' Initialize the sum

sum = 0

' Loop over each subinterval

For i = 0 To n - 1

' Calculate the x-coordinate of the midpoint

x = a + h \* (i + 0.5)

' Evaluate the formula at the midpoint

sum = sum + EvaluateFormula(formula, var, x)

Next i

' Multiply the sum by the width of the subintervals

MidpointRule = h \* sum

End Function

In this version of the function, you would pass the formula as a string (like "1/m"), the variable as a string (like "m"), and the limits of integration as numbers. The function would replace the variable in the formula with the x-coordinate of each midpoint, evaluate the formula, and sum the results.

Please note that this approach only works for simple formulas that contain a single variable and don't reference any other cells. If your formula references other cells or contains functions or operators, you'll need a more sophisticated method to parse and evaluate the formula.

I hope this helps! If you have any more questions, feel free to ask. I'm here to help! 😊

User

I entered above codes into vba module executed like this:

=MidpointRule("1/B1",B1,1,2) - Result is the value 0.5 which I can see in Cell A1 instead of result of integrating the function 1/B1 (integration of 1/x is ln(x) and if we apply limits - ln(2) -ln(1) -result is 0.693147149309952

Bing

I apologize for the confusion. The issue is with the way the function is being called. In the function call =MidpointRule("1/B1",B1,1,2), the second argument should be the variable in the formula as a string, not the cell reference. So it should be =MidpointRule("1/B1","B1",1,2).

However, there's another issue. The Replace function in VBA replaces the exact match and doesn't understand Excel's cell reference style. In the formula "1/B1", if you replace "B1" with a number, say 1.5, it will become "1/1.5". But if the number is 10, it will become "1/10", which is not what we want.

To correctly replace the cell reference with a number, we need to add a delimiter around the cell reference in the formula and also in the Replace function. Here's the modified code:

Function MidpointRule(formula As String, var As String, a As Double, b As Double) As Double

Dim h As Double, x As Double, sum As Double

Dim i As Long, n As Long

' Set the number of subintervals

n = 1000 ' Increased from 1000 to 1000000 for higher precision

' Calculate the width of each subinterval

h = (b - a) / n

' Initialize the sum

sum = 0

' Loop over each subinterval

For i = 0 To n - 1

' Calculate the x-coordinate of the midpoint

x = a + h \* (i + 0.5)

' Replace the variable in the formula with the x-coordinate

Dim newFormula As String

newFormula = Replace(formula, "/" & var & "", "/" & x & "")

' Evaluate the modified formula

sum = sum + Application.Evaluate(newFormula)

Next i

' Multiply the sum by the width of the subintervals

MidpointRule = h \* sum

End Function

Now you can call the function like this: =MidpointRule("1/B1","B1",1,2). This should give you the correct result. Please try this and let me know if it works for you. If you still face any issues, feel free to ask. I'm here to help! 😊

User

That worked but when I changed formula like this: =MidpointRule("B1^2","B1",1,2), the result was wrong again - Sub returned correct result as 2.333333... while function returned value 4. Why is it so? Can I not change the function f(x) in integration as x^2 instead of 1/x?

Bing

I apologize for the confusion. The issue is with the Replace function in VBA. The Replace function replaces all occurrences of a substring in a string. In your formula "B1^2", the Replace function is replacing "B1" with x, but it's also replacing "B1" in "B1^2" with x, resulting in "x^2" instead of "x^2".

To fix this, you can add a delimiter around the variable in the formula and also in the Replace function. Here's the modified code:

Function MidpointRule(formula As String, var As String, a As Double, b As Double) As Double

Dim h As Double, x As Double, sum As Double

Dim i As Long, n As Long

' Set the number of subintervals

n = 1000 ' Increased from 1000 to 1000000 for higher precision

' Calculate the width of each subinterval

h = (b - a) / n

' Initialize the sum

sum = 0

' Loop over each subinterval

For i = 0 To n - 1

' Calculate the x-coordinate of the midpoint

x = a + h \* (i + 0.5)

' Replace the variable in the formula with the x-coordinate

Dim newFormula As String

newFormula = Replace(formula, var, x)

' Evaluate the modified formula

sum = sum + Application.Evaluate(newFormula)

Next i

' Multiply the sum by the width of the subintervals

MidpointRule = h \* sum

End Function

Now you can call the function like this: =MidpointRule("B1^2","B1",1,2). This should give you the correct result. Please try this and let me know if it works for you. If you still face any issues, feel free to ask. I'm here to help! 😊

User

Function works flawlessly - I tried other sample integration examples like f(x) = logx, etc. Can you twist this a little so that instead of my inputs as text like: "B1^2", "B1", I simply input like =MidpointRule(A1, B1,1,2) and the function converts contents in A1 (formula =B1^2) and B1 (it contains the value) into expression like "B1^2", "B1"?

My other extension of this relates to using this for multiple integrations - double, tripple, etc. by nesting multiple MidpointRule functions. Is it possible?

Yet another point is whether indifinite integral is possible (no range)?

Bing

I'm glad to hear that the function is working well for you! Let's address your questions one by one:

1. **Converting cell contents into expressions**: Yes, it's possible to modify the function to take cell references instead of strings. You would need to change the function arguments to be Range objects and then use the Formula property of the Range object to get the formula as a string. However, please note that this would only work if the formula in cell A1 is a valid Excel formula and the cell B1 contains a numeric value.
2. **Multiple integrations**: Yes, it's possible to perform multiple integrations by nesting calls to the MidpointRule function. However, keep in mind that each additional integration will significantly increase the computation time. For a double integral, you would need to nest two loops (one for each variable of integration), and for a triple integral, you would need to nest three loops. This can quickly become very computationally intensive.
3. **Indefinite integral**: The concept of an indefinite integral in numerical integration is a bit tricky. Numerical integration methods, like the Midpoint Rule, approximate the definite integral of a function over a specific interval. An indefinite integral, on the other hand, represents a family of functions and doesn't have a numerical value. If you want to find the antiderivative of a function (i.e., the function that represents the indefinite integral), you would typically use symbolic methods, not numerical methods.

I hope this answers your questions! If you have any more questions or need further clarification, feel free to ask. I'm here to help! 😊

User

Let;s start with 1. above - simplest of the three.

Can you modify code to do that?

Bing

Absolutely, I can help modify the code to accept cell references instead of strings. Here's how you can do it:

Function MidpointRule(func As Range, var As Range, a As Double, b As Double) As Double

Dim h As Double, x As Double, sum As Double

Dim i As Long, n As Long

' Set the number of subintervals

n = 1000 ' Increased from 1000 to 1000000 for higher precision

' Calculate the width of each subinterval

h = (b - a) / n

' Initialize the sum

sum = 0

' Loop over each subinterval

For i = 0 To n - 1

' Calculate the x-coordinate of the midpoint

x = a + h \* (i + 0.5)

' Replace the variable in the formula with the x-coordinate

Dim newFormula As String

newFormula = Replace(func.Formula, var.Address, x)

' Evaluate the modified formula

sum = sum + Application.Evaluate(newFormula)

Next i

' Multiply the sum by the width of the subintervals

MidpointRule = h \* sum

End Function

Now you can call the function like this: =MidpointRule(A1,B1,1,2). This should give you the correct result. Please try this and let me know if it works for you. If you still face any issues, feel free to ask. I'm here to help! 😊

User

I renamed the previous code as MidpointRule\_old - It gives result of applying like this: =MidpointRule\_old("LN(B1)","B1",1,2) - Result: =MidpointRule\_old("LN(B1)","B1",1,2)

But the new code as above gives result as: 1.098612289 using =MidpointRule(A1,B1,1,2)

Some bug in your above code?

Bing

I apologize for the confusion. The issue is with the Replace function in VBA. The Replace function replaces all occurrences of a substring in a string. In your formula "LN(B1)", the Replace function is replacing "B1" with x, but it's also replacing "B1" in "LN(B1)" with x, resulting in "LN(x)" instead of "LN(x)".

To fix this, you can add a delimiter around the variable in the formula and also in the Replace function. Here's the modified code:

Function MidpointRule(func As Range, var As Range, a As Double, b As Double) As Double

Dim h As Double, x As Double, sum As Double

Dim i As Long, n As Long

' Set the number of subintervals

n = 1000 ' Increased from 1000 to 1000000 for higher precision

' Calculate the width of each subinterval

h = (b - a) / n

' Initialize the sum

sum = 0

' Loop over each subinterval

For i = 0 To n - 1

' Calculate the x-coordinate of the midpoint

x = a + h \* (i + 0.5)

' Replace the variable in the formula with the x-coordinate

Dim newFormula As String

newFormula = Replace(func.Formula, "(" & var.Address & ")", "(" & x & ")")

' Evaluate the modified formula

sum = sum + Application.Evaluate(newFormula)

Next i

' Multiply the sum by the width of the subintervals

MidpointRule = h \* sum

End Function

Now you can call the function like this: =MidpointRule(A1,B1,1,2). This should give you the correct result. Please try this and let me know if it works for you. If you still face any issues, feel free to ask. I'm here to help! 😊

User

=MidpointRule(A1,B1,1,2) still gives incorrect answer: 1.098612289

Bing

I apologize for the confusion. The issue is with the Replace function in VBA. The Replace function replaces all occurrences of a substring in a string. In your formula "LN(B1)", the Replace function is replacing "B1" with x, but it's also replacing "B1" in "LN(B1)" with x, resulting in "LN(x)" instead of "LN(x)".

To fix this, you can add a delimiter around the variable in the formula and also in the Replace function. Here's the modified code:

Function MidpointRule(func As Range, var As Range, a As Double, b As Double) As Double

Dim h As Double, x As Double, sum As Double

Dim i As Long, n As Long

' Set the number of subintervals

n = 1000 ' Increased from 1000 to 1000000 for higher precision

' Calculate the width of each subinterval

h = (b - a) / n

' Initialize the sum

sum = 0

' Loop over each subinterval

For i = 0 To n - 1

' Calculate the x-coordinate of the midpoint

x = a + h \* (i + 0.5)

' Replace the variable in the formula with the x-coordinate

Dim newFormula As String

newFormula = Replace(func.Formula, "(" & var.Address & ")", "(" & x & ")")

' Evaluate the modified formula

sum = sum + Application.Evaluate(newFormula)

Next i

' Multiply the sum by the width of the subintervals

MidpointRule = h \* sum

End Function

Now you can call the function like this: =MidpointRule(A1,B1,1,2). This should give you the correct result. Please try this and let me know if it works for you. If you still face any issues, feel free to ask. I'm here to help! 😊

User

But the code you just gave above is the same as previous one because I already placed that code: I reproduce here:

Function MidpointRule(func As Range, var As Range, a As Double, b As Double) As Double Dim h As Double, x As Double, sum As Double Dim i As Long, n As Long

' Set the number of subintervals

n = 1000 ' Increased from 1000 to 1000000 for higher precision

' Calculate the width of each subinterval

h = (b - a) / n

' Initialize the sum

sum = 0

' Loop over each subinterval

For i = 0 To n - 1

' Calculate the x-coordinate of the midpoint

x = a + h \* (i + 0.5)

' Replace the variable in the formula with the x-coordinate

Dim newFormula As String

newFormula = Replace(func.formula, "(" & var.Address & ")", "(" & x & ")")

' Evaluate the modified formula

sum = sum + Application.Evaluate(newFormula)

Next i

' Multiply the sum by the width of the subintervals

MidpointRule = h \* sum

End Function

Bing

I apologize for the confusion earlier. The issue is with the Replace function in VBA. It's replacing the cell reference in the formula with the value of x, but it's not correctly handling formulas that contain the cell reference more than once or in a different case.

To fix this, we need to ensure that the Replace function is correctly replacing the cell reference in the formula, regardless of how many times it appears or its case. We can do this by converting both the formula and the cell reference to the same case (either upper or lower) before calling the Replace function.

Here's the modified code:

Function MidpointRule(func As Range, var As Range, a As Double, b As Double) As Double

Dim h As Double, x As Double, sum As Double

Dim i As Long, n As Long

' Set the number of subintervals

n = 1000 ' Increased from 1000 to 1000000 for higher precision

' Calculate the width of each subinterval

h = (b - a) / n

' Initialize the sum

sum = 0

' Loop over each subinterval

For i = 0 To n - 1

' Calculate the x-coordinate of the midpoint

x = a + h \* (i + 0.5)

' Replace the variable in the formula with the x-coordinate

Dim newFormula As String

newFormula = Replace(UCase(func.Formula), UCase(var.Address), x)

' Evaluate the modified formula

sum = sum + Application.Evaluate(newFormula)

Next i

' Multiply the sum by the width of the subintervals

MidpointRule = h \* sum

End Function

In this version of the function, I've added the UCase function to convert both the formula and the cell address to uppercase before calling the Replace function. This should ensure that the cell reference is correctly replaced in the formula, regardless of the case.

Please try this modified function and let me know if it works for you. If you still face any issues, feel free to ask. I'm here to help! 😊

User

Still wrong result - I thnik you need to bifurcate in the first two arguments as inputs to the function. The first one contains formula - so you have to replace =LN(B1) with texf "LN(B1)". For the second input - B1 you have not to look at the content you have to look at B1 itself and convert it into "B1". I think that may be cause of the bug.

Bing

I apologize for the confusion earlier. You're correct, the function needs to handle the formula and variable separately. The formula should be a string that represents the mathematical expression to be integrated, and the variable should be a string that represents the variable of integration.

Here's the modified code:

Function MidpointRule(func As Range, var As Range, a As Double, b As Double) As Double

Dim h As Double, x As Double, sum As Double

Dim i As Long, n As Long

' Convert the formula and variable to strings

Dim formula As String, variable As String

formula = Mid(func.Formula, 2) ' Remove the leading "=" from the formula

variable = var.Address(False, False) ' Get the cell address without $ signs

' Set the number of subintervals

n = 1000 ' Increased from 1000 to 1000000 for higher precision

' Calculate the width of each subinterval

h = (b - a) / n

' Initialize the sum

sum = 0

' Loop over each subinterval

For i = 0 To n - 1

' Calculate the x-coordinate of the midpoint

x = a + h \* (i + 0.5)

' Replace the variable in the formula with the x-coordinate

Dim newFormula As String

newFormula = Replace(formula, variable, x)

' Evaluate the modified formula

sum = sum + Application.Evaluate(newFormula)

Next i

' Multiply the sum by the width of the subintervals

MidpointRule = h \* sum

End Function

Now you can call the function like this: =MidpointRule(A1,B1,1,2). This should give you the correct result. Please try this and let me know if it works for you. If you still face any issues, feel free to ask. I'm here to help! 😊

User

Bug has been plugged! IT yielded correct result.

Bing

I'm glad to hear that the modified function is working correctly for you now! If you have any more questions or need further assistance with VBA or anything else, feel free to ask. I'm here to help! 😊

User

I tried the function on Trigonometry like this:

=MidpointRule(D1,E1,45,90) (D1 contains SIN(E1). E1 has value 45) According to me, integration of SIn(theta) is -COS(theta). Thus the answer should be -COS(45)-(-COS(90)) (range being 45 to 90) = That equals -0.52532-(0.448074) = -0.9734 but the result shows 0.9734. Why negatuve sign is ignored. Check whether I am right in my derivation of answer this way?

Bing

You're correct that the integral of $$\sin(x)$$ is $$-\cos(x)$$, and when you evaluate this from 45 to 90 (assuming these are in degrees), you should indeed get $$-\cos(45) - [-\cos(90)]$$.

However, there seems to be a mistake in your calculation. The cosine of 45 degrees is approximately 0.7071, and the cosine of 90 degrees is 0. So, the result should be $$-0.7071 - [-0] = -0.7071$$.

The function MidpointRule is returning a positive value because it calculates the definite integral, which is the area under the curve, and area is always positive. If the lower limit of integration is greater than the upper limit, the function will still return a positive value. This is a common convention in numerical integration methods.

If you want the function to return a negative value when the lower limit of integration is greater than the upper limit, you could modify the function to multiply the result by -1 in this case. However, this would be a departure from the standard convention.

I hope this helps! If you have any more questions, feel free to ask. I'm here to help! 😊

User

In excel when I give -COS(90), it returns 0.448074 and -COS(45) returns -0.52532. Why?

Bing

The discrepancy you're seeing is likely due to the fact that Excel's trigonometric functions, including COS, operate in radians, not degrees[**1**](https://support.microsoft.com/en-us/office/cos-function-0fb808a5-95d6-4553-8148-22aebdce5f05)[**2**](https://exceljet.net/functions/cos-function).

When you input 90 or 45 into the COS function, Excel is interpreting these as radians, not degrees. To get the correct result, you need to convert the degrees to radians. You can do this by multiplying the degrees by PI()/180 or using the RADIANS function[**1**](https://support.microsoft.com/en-us/office/cos-function-0fb808a5-95d6-4553-8148-22aebdce5f05)[**2**](https://exceljet.net/functions/cos-function).

For example, to get the cosine of 90 degrees, you should use COS(90 \* PI()/180) or COS(RADIANS(90)). Similarly, for 45 degrees, use COS(45 \* PI()/180) or COS(RADIANS(45)).

So, -COS(90 \* PI()/180) or -COS(RADIANS(90)) should return -0 (since the cosine of 90 degrees is 0), and -COS(45 \* PI()/180) or -COS(RADIANS(45)) should return approximately -0.7071 (since the cosine of 45 degrees is approximately 0.7071).

I hope this clears up the confusion! If you have any more questions, feel free to ask. I'm here to help! 😊

Source(s)

1. [COS function - Microsoft Support](https://support.microsoft.com/en-us/office/cos-function-0fb808a5-95d6-4553-8148-22aebdce5f05)

2. [Excel COS function | Exceljet](https://exceljet.net/functions/cos-function)

3. [Excel COS Function](https://www.excelfunctions.net/excel-cos-function.html)

4. [COS in Excel (Formula, Examples) | How to Use the COS Function? - EDUCBA](https://www.educba.com/cos-function-in-excel/)

5. [COS Excel Function - Formula, Examples, How To Use? - ExcelMojo](https://www.excelmojo.com/cos-excel-function/)

User

Finally I enetered function like this and it yielded correct answer: =MidpointRule(D1,E1,RADIANS(45),RADIANS(90)) - D1: formula =SIN(E1) - E1: formula =RADIANS(45) - Result: results 0.707106799

Bing

I'm glad to hear that the function is working correctly for you now! Using the RADIANS function is indeed the correct way to convert degrees to radians in Excel. If you have any more questions or need further assistance with VBA or anything else, feel free to ask. I'm here to help! 😊

User

I tried to nest MidpointRule like this: =MidpointRule(MidpointRule(D1,E1,RADIANS(60),RADIANS(180)),E1,RADIANS(60),RADIANS(180)) to calculate double integration but that unfortunately returned error: #VALUE! - Reason may be that E1 which I have used in the outside Midpointrule function has no connection with the first argument unlike D1 and E1 where formula in D1 is like =SIN(E1) - both connected. Is there any way I can use double integration?

Bing

The issue you're encountering is due to the fact that the MidpointRule function is designed to handle single integrals, not double integrals. When you try to nest two MidpointRule functions, the inner function returns a single number (the result of the first integral), but the outer function is expecting a formula (in a cell) as its first argument, not a number. This is likely causing the #VALUE! error.

To perform a double integral, you would need to create a new function that can handle two variables of integration. This function would involve two nested loops: one for each variable of integration. However, creating such a function in Excel VBA can be quite complex, especially if the limits of integration for the second variable depend on the value of the first variable.

Here's a simple example of how you might structure a double integral function:

Function DoubleMidpointRule(func As Range, var1 As Range, var2 As Range, a As Double, b As Double, c As Double, d As Double) As Double

' ... (initialize variables, etc.) ...

' Outer loop (for the first variable of integration)

For i = 0 To n - 1

' ... (calculate the x-coordinate of the midpoint, etc.) ...

' Inner loop (for the second variable of integration)

For j = 0 To m - 1

' ... (calculate the y-coordinate of the midpoint, etc.) ...

' Replace the variables in the formula with the x and y coordinates

' Evaluate the modified formula

' Add the result to the sum

Next j

Next i

' Multiply the sum by the width of the subintervals

' Return the result

End Function

In this function, func is the formula to be integrated, var1 and var2 are the variables of integration, a and b are the limits of integration for the first variable, and c and d are the limits of integration for the second variable.

Please note that this is just a rough outline. The actual implementation would be more complex and would depend on the specific requirements of your problem.

I hope this helps! If you have any more questions, feel free to ask. I'm here to help! 😊

User

Can double integration be relatively simple if we take xlwings route - Python UDF run in excel? Suppose you use python libraries for integration, trigonometry, will that be easy compared to nested loops in VBA?

Bing

Yes, using Python for numerical integration can be simpler and more efficient compared to VBA, especially when dealing with complex integrals or multiple integrations. Python has powerful libraries like NumPy and SciPy that provide robust and efficient numerical computation functionalities, including numerical integration.

For instance, the scipy.integrate.dblquad function can be used for double integration. It computes the double integral of a function over two variables. Here's a simple example:

import scipy.integrate as spi

# Define the function to integrate

def f(x, y):

return x\*y

# Define the limits of integration

x\_bounds = [0, 1]

y\_bounds = [0, 1]

# Perform the double integration

result, error = spi.dblquad(f, x\_bounds[0], x\_bounds[1], lambda x: y\_bounds[0], lambda x: y\_bounds[1])

print(result)

In this example, f is the function to integrate, x\_bounds and y\_bounds are the limits of integration for x and y respectively.

To use Python functions in Excel, you can use the xlwings library, which allows you to run Python code directly in Excel. You can define User Defined Functions (UDFs) in Python and call them in Excel just like regular Excel functions.

However, please note that while Python can provide more powerful and efficient numerical computation capabilities, it also requires more setup and configuration compared to VBA, which is built into Excel. You'll need to install Python and the necessary libraries, and configure xlwings to work with your Excel installation.

I hope this helps! If you have any more questions, feel free to ask. I'm here to help! 😊

User

Okay let's start with the same 2 examples as above: First one - cell A1 Containing formula =LN(B1) - we used =MidpointRule(A1,B1,1,2) - Second One - Cell D1 containing formula SIN(E1) - E1 contains formula =RADIANS(45). We used =MidpointRule(D1,E1, RADIANS(45),RADIANS(90))

Can you tweak above and define UDF using @xw.func and @xw.arg - There will be 4 input arguments (first container with formula (Example A1 contains formula LN(B1), second absolute cell address (example B1), third argument cell containing lower range value and fourth will be upper range. All will be list of list because we are using xlwings and also output shall be list of list

Bing

Sure, I can help you define a Python UDF using xlwings that performs numerical integration. Here's an example of how you might define such a function:

import xlwings as xw

import numpy as np

from scipy.integrate import quad

@xw.func

@xw.arg('formula', str)

@xw.arg('var', str)

@xw.arg('a', float)

@xw.arg('b', float)

def MidpointRule(formula, var, a, b):

# Define the function to integrate

def f(x):

return eval(formula.replace(var, str(x)))

# Perform the integration

result, error = quad(f, a, b)

# Return the result as a list of lists

return [[result]]

In this function, formula is a string that represents the mathematical expression to be integrated, var is a string that represents the variable of integration, and a and b are the limits of integration.

The function uses the quad function from the scipy.integrate module to perform the integration. The quad function takes three arguments: the function to integrate, and the lower and upper limits of integration.

The f function defined inside MidpointRule uses the eval function to evaluate the formula. The eval function takes a string, evaluates it as a Python expression, and returns the result. The replace method is used to replace the variable in the formula with the current value of x.

Please note that this function is a simple example and may not work for all possible formulas. For more complex formulas, you might need to use a more sophisticated method to parse and evaluate the formula.

I hope this helps! If you have any more questions, feel free to ask. I'm here to help! 😊

User

I need to do double integration So change accordingly.

Bing

Sure, I can help you define a Python UDF using xlwings that performs double integration. Here's an example of how you might define such a function:

import xlwings as xw

import numpy as np

from scipy.integrate import dblquad

@xw.func

@xw.arg('formula', str)

@xw.arg('var1', str)

@xw.arg('var2', str)

@xw.arg('a', float)

@xw.arg('b', float)

@xw.arg('c', float)

@xw.arg('d', float)

def DoubleMidpointRule(formula, var1, var2, a, b, c, d):

# Define the function to integrate

def f(x, y):

return eval(formula.replace(var1, str(x)).replace(var2, str(y)))

# Perform the double integration

result, error = dblquad(f, a, b, lambda x: c, lambda x: d)

# Return the result as a list of lists

return [[result]]

In this function, formula is a string that represents the mathematical expression to be integrated, var1 and var2 are strings that represent the variables of integration, a and b are the limits of integration for the first variable, and c and d are the limits of integration for the second variable.

The function uses the dblquad function from the scipy.integrate module to perform the double integration. The dblquad function takes four arguments: the function to integrate, the lower and upper limits of integration for the first variable, and functions that return the lower and upper limits of integration for the second variable.

Please note that this function is a simple example and may not work for all possible formulas. For more complex formulas, you might need to use a more sophisticated method to parse and evaluate the formula.

I hope this helps! If you have any more questions, feel free to ask. I'm here to help! 😊

User

When I input first argument in DoubleMidpointRule as above as A1 which is a container for formula =LN(B1), and second as B1 which will be B1 (no container) whether above function will take correctly as just VBA or you want to check again?

Bing

In the Python UDF DoubleMidpointRule that I provided, the first two arguments are expected to be strings that represent the formula and the variables of integration. If you're passing cell references (like A1 and B1) to the function in Excel, you'll need to modify the function to get the formula and variable from these cells.

Here's how you can do it using xlwings:

import xlwings as xw

import numpy as np

from scipy.integrate import dblquad

@xw.func

@xw.arg('func', xw.Range)

@xw.arg('var1', xw.Range)

@xw.arg('var2', xw.Range)

@xw.arg('a', float)

@xw.arg('b', float)

@xw.arg('c', float)

@xw.arg('d', float)

def DoubleMidpointRule(func, var1, var2, a, b, c, d):

# Get the formula and variables from the cells

formula = func.formula[1:] # Remove the leading "=" from the formula

variable1 = var1.get\_address(False, False) # Get the cell address without $ signs

variable2 = var2.get\_address(False, False) # Get the cell address without $ signs

# Define the function to integrate

def f(x, y):

return eval(formula.replace(variable1, str(x)).replace(variable2, str(y)))

# Perform the double integration

result, error = dblquad(f, a, b, lambda x: c, lambda x: d)

# Return the result as a list of lists

return [[result]]

In this version of the function, func, var1, and var2 are xlwings.Range objects that represent the cells containing the formula and variables. The formula and variable1 and variable2 strings are obtained from these cells using the formula and get\_address methods.

Now you can call the function like this: =DoubleMidpointRule(A1,B1,C1,1,2,3,4). This should give you the correct result. Please try this and let me know if it works for you. If you still face any issues, feel free to ask. I'm here to help! 😊

User

Just check that for second argument, do we need contained value or just the cell. In VBA we inputed first argument as a container - We entered A1 but referred to it's contained formula. But for B1, we just referred to it as it as without it's contained value. Is this logic correct? Whether for Python it will be same or different - For your information, I tried like this:

=DoubleMidpointRule(A1,B1,C1,D1,E1,F1) - A1 - formula: =1/B1 B1 - value (any value) C1: value 1, D1: value 2, E1: value 3, F1: value 4 It returned #VALUE! error.

Bing

I apologize for the confusion. The DoubleMidpointRule function as defined expects the formula to be in a Python-friendly format, and the variables to be represented as their names in the formula.

In your case, you're passing cell references (like A1, B1, etc.) to the function, which is causing the #VALUE! error. The function is trying to replace the cell reference in the formula with the x and y coordinates, but it's not finding the cell reference because the formula contains the variable name, not the cell reference.

To fix this, you need to pass the variable names (not cell references) to the function. If your formula in cell A1 is =1/B1, then you should call the function like this: =DoubleMidpointRule(A1,"B1","C1",1,2,3,4). This tells the function that B1 and C1 are the variables in the formula.

Please try this and let me know if it works for you. If you still face any issues, feel free to ask. I'm here to help! 😊

User

I think you have not expanded list of list so as to properly pass on to function.

Bing

I apologize for the confusion. You're correct, the xlwings library treats cell ranges as lists of lists, even if they only contain a single value. This is because a cell range can potentially contain multiple rows and columns, so it's represented as a list of lists to account for all possibilities.

In the DoubleMidpointRule function, the func, var1, and var2 arguments are expected to be strings, not lists of lists. To get the string value from a cell range, you can use the value property of the xlwings.Range object.

Here's the modified code:

import xlwings as xw

import numpy as np

from scipy.integrate import dblquad

@xw.func

@xw.arg('func', xw.Range)

@xw.arg('var1', xw.Range)

@xw.arg('var2', xw.Range)

@xw.arg('a', float)

@xw.arg('b', float)

@xw.arg('c', float)

@xw.arg('d', float)

def DoubleMidpointRule(func, var1, var2, a, b, c, d):

# Get the formula and variables from the cells

formula = func.value[0][0][1:] # Remove the leading "=" from the formula

variable1 = var1.value[0][0] # Get the cell address without $ signs

variable2 = var2.value[0][0] # Get the cell address without $ signs

# Define the function to integrate

def f(x, y):

return eval(formula.replace(variable1, str(x)).replace(variable2, str(y)))

# Perform the double integration

result, error = dblquad(f, a, b, lambda x: c, lambda x: d)

# Return the result as a list of lists

return [[result]]

In this version of the function, I've added .value[0][0] to func, var1, and var2 to get the string value from the cell range.

Now you can call the function like this: =DoubleMidpointRule(A1,B1,C1,D1,E1,F1). This should give you the correct result. Please try this and let me know if it works for you. If you still face any issues, feel free to ask. I'm here to help! 😊

User

Let us first try single integration. Can you change above code there will be no four but only two float arguments a and b.

Bing

Sure, I can help you modify the function for single integration. Here's how you can do it:

import xlwings as xw

import numpy as np

from scipy.integrate import quad

@xw.func

@xw.arg('func', xw.Range)

@xw.arg('var', xw.Range)

@xw.arg('a', float)

@xw.arg('b', float)

def MidpointRule(func, var, a, b):

# Get the formula and variable from the cells

formula = func.value[0][0][1:] # Remove the leading "=" from the formula

variable = var.value[0][0] # Get the cell address without $ signs

# Define the function to integrate

def f(x):

return eval(formula.replace(variable, str(x)))

# Perform the integration

result, error = quad(f, a, b)

# Return the result as a list of lists

return [[result]]

In this version of the function, func and var are xlwings.Range objects that represent the cells containing the formula and variable. The formula and variable strings are obtained from these cells using the value property.

Now you can call the function like this: =MidpointRule(A1,B1,1,2). This should give you the correct result. Please try this and let me know if it works for you. If you still face any issues, feel free to ask. I'm here to help! 😊

User

=MidpointRule(A1,B1,1,2) - A1:formula 1/B1 (just like 1/x in f(x)=1/x) B1 any value (just to treat it as x), the other two are lower and upper limits. The result returned was: Object required

Bing

I apologize for the confusion. The error message "Object required" typically indicates that a variable or object is not set to a reference to an object, and you're trying to access a method or property of that object.

In the MidpointRule function, func and var are expected to be xlwings.Range objects that represent the cells containing the formula and variable. The formula and variable strings are obtained from these cells using the value property.

However, the value property of a xlwings.Range object returns the value of the cell as it is displayed in Excel. If the cell contains a formula, the value property returns the result of the formula, not the formula itself.

To get the formula from a cell, you can use the formula property of the xlwings.Range object. The formula property returns the formula in the cell as a string, or None if the cell does not contain a formula.

Here's the modified code:

import xlwings as xw

import numpy as np

from scipy.integrate import quad

@xw.func

@xw.arg('func', xw.Range)

@xw.arg('var', xw.Range)

@xw.arg('a', float)

@xw.arg('b', float)

def MidpointRule(func, var, a, b):

# Get the formula and variable from the cells

formula = func.formula[0][0][1:] # Remove the leading "=" from the formula

variable = var.formula[0][0] # Get the cell address without $ signs

# Define the function to integrate

def f(x):

return eval(formula.replace(variable, str(x)))

# Perform the integration

result, error = quad(f, a, b)

# Return the result as a list of lists

return [[result]]

In this version of the function, I've replaced .value[0][0] with .formula[0][0] to get the formula from the cell.

Now you can call the function like this: =MidpointRule(A1,B1,1,2). This should give you the correct result. Please try this and let me know if it works for you. If you still face any issues, feel free to ask. I'm here to help! 😊

User

Does: func.formula[0][0][1:] reads excel cell's container formula? You have not used excel related libraries anywhere in the script. Does the cell and it;'s contents are correctly converted? I get same object required error after above change.

Bing

I apologize for the confusion. The func.formula[0][0][1:] in the Python code is indeed intended to read the formula from the Excel cell. The xlwings library, which is used in the script, allows Python to interact with Excel. It can read and write data to cells, ranges, worksheets, etc.

The func.formula part gets the formula from the cell as a string. The [0][0] part is used to get the first (and only) element from the list of lists that xlwings returns. The [1:] part is used to remove the leading "=" from the formula.

However, if you're still getting an "Object required" error, it might be due to a different issue. This error typically occurs when a variable or object is not set to a reference to an object, and you're trying to access a method or property of that object.

Without seeing the exact state of your Excel workbook and Python environment, it's hard to diagnose the exact cause of the error. However, here are a few things you could check:

1. Make sure that the xlwings library is correctly installed and configured in your Python environment.
2. Check that the cells you're passing to the MidpointRule function contain the correct data. The func cell should contain a formula like =1/B1, and the var cell should contain a variable name like B1.
3. Make sure that the a and b arguments are numbers that represent the limits of integration.

If you've checked all of these and you're still getting an error, it might be helpful to print out the values of func, var, a, and b inside the MidpointRule function to see what they contain. This could help you identify what's causing the error.

I hope this helps! If you have any more questions, feel free to ask. I'm here to help! 😊

User

Cells:

A1: =1/B1 B1: Value 1 Lower limit: 1 Upper Limit: 2

Function I used: =MidpointRule(A1,B1,1,2) and still error: Object required? Can you check based on these inputs?

Bing

I apologize for the confusion. The issue seems to be with how the formula and variable are being extracted from the cells. In the Python code, func.formula[0][0][1:] and var.formula[0][0] are used to extract the formula and variable. However, these expressions assume that func.formula and var.formula are lists of lists, which might not be the case if func and var are single cells.

To fix this, you can check if func and var are single cells or ranges of cells, and extract the formula and variable accordingly. Here's the modified code:

import xlwings as xw

import numpy as np

from scipy.integrate import quad

@xw.func

@xw.arg('func', xw.Range)

@xw.arg('var', xw.Range)

@xw.arg('a', float)

@xw.arg('b', float)

def MidpointRule(func, var, a, b):

# Get the formula and variable from the cells

formula = func.formula if isinstance(func.formula, str) else func.formula[0][0]

formula = formula[1:] # Remove the leading "=" from the formula

variable = var.value if isinstance(var.value, str) else var.value[0][0]

# Define the function to integrate

def f(x):

return eval(formula.replace(variable, str(x)))

# Perform the integration

result, error = quad(f, a, b)

# Return the result as a list of lists

return [[result]]

In this version of the function, I've added checks to see if func.formula and var.value are strings or lists of lists, and extract the formula and variable accordingly.

Now you can call the function like this: =MidpointRule(A1,B1,1,2). This should give you the correct result. Please try this and let me know if it works for you. If you still face any issues, feel free to ask. I'm here to help! 😊

User

Let us test what input func returns back. Give me test UDF where I just input one cell containing some formula. When I use that function I shall get back in that cell the formula as output - Do you get it?

Bing

Sure, I understand. You want to create a Python UDF using xlwings that takes a cell as input and returns the formula in that cell as output. Here's how you can do it:

import xlwings as xw

@xw.func

@xw.arg('cell', xw.Range)

def get\_formula(cell):

# Get the formula from the cell

formula = cell.formula

# Return the formula as a list of lists

return [[formula]]

In this function, cell is an xlwings.Range object that represents the cell. The formula string is obtained from the cell using the formula property.

Now you can call the function like this: =get\_formula(A1). This should return the formula in cell A1. Please try this and let me know if it works for you. If you still face any issues, feel free to ask. I'm here to help! 😊