User

I have following xlwings python script:

import xlwings as xw
import numpy as np
from scipy.integrate import quad

@xw.func
@xw.arg('func', xw.Range)
@xw.arg('var')
@xw.arg('a')
@xw.arg('b')
def MidpointRule(func, var, a, b):
# Convert a and b to float
a = float(a)
b = float(b)

# Get the formula from the cell

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

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

# Get the variable from the cell

variable = str(var.value)

# Define the function to integrate

def f(x):

B1 = x # Define 'B1' before calling eval

return eval(formula.replace(variable, str(B1)))

# Perform the integration

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

# Return the result as a list of lists

return [[result]]

When I am using the function as: =MidpointRule(A1,B1,1,2) [cell A1 formula =1/B1, cell B1 some value (it is to be used as variable actually like x - I am evaluating integration of 1/x so B1 is infact x. The last two are lower and upper limits of integral. I get error:

"AttributeError: 'float' object has no attribute 'value'
^^^^^^^^^
variable = str(var.value)
File ""D:\OneDrive - 0yt2k\xlwings\_double\_integration\_example.py"", line 20, in MidpointRule
^^^^^^^^^^^
ret = func(\*args)
File ""D:\Programs\Python\Lib\site-packages\xlwings\udfs.py"", line 502, in call\_udf
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
res = call\_udf(script, fname, args, this\_workbook, FromVariant(caller))
File ""D:\Programs\Python\Lib\site-packages\xlwings\com\_server.py"", line 235, in CallUDF
^^^^^^^^^^^
return func(\*args)
File ""D:\Programs\Python\Lib\site-packages\win32com\server\policy.py"", line 639, in *invokeex*
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
return S\_OK, -1, self.*invokeex*(dispid, lcid, wFlags, args, None, None)
File ""D:\Programs\Python\Lib\site-packages\win32com\server\policy.py"", line 310, in *invoke*
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
return self.*invoke*(dispid, lcid, wFlags, args)
File ""D:\Programs\Python\Lib\site-packages\win32com\server\policy.py"", line 305, in *Invoke*"

Bing

The error message you're seeing is because you're trying to access the value attribute of a float object, which doesn't exist. In your function MidpointRule, you're expecting var to be an xlwings.Range object, but it seems like you're passing a float instead when you call the function with =MidpointRule(A1,B1,1,2).

In this case, B1 is a float, not an xlwings.Range object. Therefore, when you try to access var.value, it throws an AttributeError because float objects do not have a value attribute.

To fix this issue, you need to ensure that var is an xlwings.Range object. You can do this by changing the @xw.arg('var') decorator to @xw.arg('var', xw.Range). This will ensure that var is treated as an xlwings.Range object, and you can safely access the value attribute.

Here's the corrected 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) # Ensure 'var' is an xlwings.Range object

@xw.arg('a')

@xw.arg('b')

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

# Convert a and b to float

a = float(a)

b = float(b)

# Get the formula from the cell

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

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

# Get the variable from the cell

variable = str(var.value)

# Define the function to integrate

def f(x):

B1 = x # Define 'B1' before calling eval

return eval(formula.replace(variable, str(B1)))

# Perform the integration

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

# Return the result as a list of lists

return [[result]]

Now, when you call the function with =MidpointRule(A1,B1,1,2), B1 should be treated as an xlwings.Range object, and you should be able to access var.value without any issues. 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!

Bing

I'm glad to hear that it worked! If you have any more questions or need further help with Python, xlwings, or anything else, feel free to ask. I'm here to help! 😊

User

Does this function also allows integration of trigonometric functions like SIN, COS, or some additional libraries will need to be imported for that?

Bing

Yes, the function you've written can handle the integration of trigonometric functions like sin and cos. Python's eval function, which you're using to evaluate the formula, has access to the Python's built-in functions, including sin and cos from the math module.

However, you need to make sure that you're using the correct syntax in your Excel formula. The sin and cos functions should be written in lowercase, and the argument should be in radians. For example, if you want to integrate the function sin(x) from 0 to π, you would write the formula in Excel as =SIN(B1), and call your function as =MidpointRule(A1,B1,0,3.14159).

Please note that Python's trigonometric functions expect the argument to be in radians, not degrees. If you want to use degrees, you'll need to convert them to radians first. You can do this in Excel using the RADIANS function, or in Python using the radians function from the math module.

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

User

Error is:

"NameError: name 'SIN' is not defined
File ""<string>"", line 1, in <module>
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
return eval(formula.replace(variable, str(B1)))
File ""D:\OneDrive - 0yt2k\xlwings\_double\_integration\_example.py"", line 25, in f
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
return \_quadpack.\_qagse(func,a,b,args,full\_output,epsabs,epsrel,limit)
File ""D:\Programs\Python\Lib\site-packages\scipy\integrate\_quadpack\_py.py"", line 577, in \_quad
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
retval = \_quad(func, a, b, args, full\_output, epsabs, epsrel, limit,
File ""D:\Programs\Python\Lib\site-packages\scipy\integrate\_quadpack\_py.py"", line 465, in quad
^^^^^^^^^^^^^
result, error = quad(f, a, b)
File ""D:\OneDrive - 0yt2k\xlwings\_double\_integration\_example.py"", line 28, in MidpointRule
^^^^^^^^^^^
ret = func(\*args)
File ""D:\Programs\Python\Lib\site-packages\xlwings\udfs.py"", line 502, in call\_udf
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
res = call\_udf(script, fname, args, this\_workbook, FromVariant(caller))
File ""D:\Programs\Python\Lib\site-packages\xlwings\com\_server.py"", line 235, in CallUDF
^^^^^^^^^^^
return func(\*args)
File ""D:\Programs\Python\Lib\site-packages\win32com\server\policy.py"", line 639, in *invokeex*
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
return S\_OK, -1, self.*invokeex*(dispid, lcid, wFlags, args, None, None)
File ""D:\Programs\Python\Lib\site-packages\win32com\server\policy.py"", line 310, in *invoke*
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
return self.*invoke*(dispid, lcid, wFlags, args)
File ""D:\Programs\Python\Lib\site-packages\win32com\server\policy.py"", line 305, in *Invoke*"

I think you need to include some additional commands to check is it is trigonometric functions then convert CAPS into small.

Bing

You're correct. The error is because the Python eval function is case-sensitive and it doesn't recognize 'SIN' as it does 'sin'. Excel, on the other hand, is not case-sensitive when it comes to function names.

To handle this, you can modify your Python code to convert the formula to lowercase before evaluating it. Here's how you can do it:

import xlwings as xw

import numpy as np

from scipy.integrate import quad

import math

@xw.func

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

@xw.arg('var', xw.Range) # Ensure 'var' is an xlwings.Range object

@xw.arg('a')

@xw.arg('b')

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

# Convert a and b to float

a = float(a)

b = float(b)

# Get the formula from the cell

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

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

# Convert the formula to lowercase

formula = formula.lower()

# Get the variable from the cell

variable = str(var.value)

# Define the function to integrate

def f(x):

B1 = x # Define 'B1' before calling eval

return eval(formula.replace(variable, str(B1)))

# Perform the integration

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

# Return the result as a list of lists

return [[result]]

This should solve the issue. Now, when you write =SIN(B1) in Excel, it will be converted to =sin(b1) in Python, and eval will be able to recognize and evaluate it correctly.

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

If I do this then my earlier function in A1 =1/B1 will not work because lowercase range objects are not allowed. Do it in some other way that both trigo and non-trigo are supported.

Bing

You're correct, my apologies for the oversight. Lowercasing the entire formula would indeed cause issues with range references. Instead, we can create a dictionary that maps the uppercase function names to their lowercase equivalents, and then replace only the function names in the formula. Here's how you can do it:

import xlwings as xw

import numpy as np

from scipy.integrate import quad

import math

@xw.func

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

@xw.arg('var', xw.Range) # Ensure 'var' is an xlwings.Range object

@xw.arg('a')

@xw.arg('b')

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

# Convert a and b to float

a = float(a)

b = float(b)

# Get the formula from the cell

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

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

# Map of Excel function names to Python function names

func\_map = {

'SIN': 'sin',

'COS': 'cos',

'TAN': 'tan',

# Add more if needed

}

# Replace function names in the formula

for excel\_func, python\_func in func\_map.items():

formula = formula.replace(excel\_func, python\_func)

# Get the variable from the cell

variable = str(var.value)

# Define the function to integrate

def f(x):

B1 = x # Define 'B1' before calling eval

return eval(formula.replace(variable, str(B1)))

# Perform the integration

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

# Return the result as a list of lists

return [[result]]

This code will replace the Excel function names (like 'SIN') with their Python equivalents (like 'sin'), while leaving the rest of the formula (including range references like 'B1') unchanged. This should allow both your =1/B1 and =SIN(B1) formulas to work correctly.

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

Why you have gard coded:
def f(x):
B1 = x # Define 'B1' before calling eval
return eval(formula.replace(variable, str(B1)))

It can be C1, D2 or any cell?

Bing

You're correct, the cell reference could be any cell, not just B1. The line B1 = x is used to define a variable B1 that can be used in the formula you're evaluating. If your formula uses a different cell reference, you would need to adjust this line accordingly.

However, to make the function more flexible and work with any cell reference, you can modify the code to use the variable directly in the eval function. Here's how you can do it:

import xlwings as xw

import numpy as np

from scipy.integrate import quad

import math

@xw.func

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

@xw.arg('var', xw.Range) # Ensure 'var' is an xlwings.Range object

@xw.arg('a')

@xw.arg('b')

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

# Convert a and b to float

a = float(a)

b = float(b)

# Get the formula from the cell

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

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

# Map of Excel function names to Python function names

func\_map = {

'SIN': 'sin',

'COS': 'cos',

'TAN': 'tan',

# Add more if needed

}

# Replace function names in the formula

for excel\_func, python\_func in func\_map.items():

formula = formula.replace(excel\_func, python\_func)

# Get the variable from the cell

variable = str(var.value)

# 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, the eval function directly replaces the variable with the value of x, so it should work with any cell reference. 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 caused error:

"NameError: name 'B1' is not defined
File ""<string>"", line 1, in <module>
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
return eval(formula.replace(variable, str(x)))
File ""D:\OneDrive - 0yt2k\xlwings\_double\_integration\_example.py"", line 37, in f
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
return \_quadpack.\_qagse(func,a,b,args,full\_output,epsabs,epsrel,limit)
File ""D:\Programs\Python\Lib\site-packages\scipy\integrate\_quadpack\_py.py"", line 577, in \_quad
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
retval = \_quad(func, a, b, args, full\_output, epsabs, epsrel, limit,
File ""D:\Programs\Python\Lib\site-packages\scipy\integrate\_quadpack\_py.py"", line 465, in quad
^^^^^^^^^^^^^
result, error = quad(f, a, b)
File ""D:\OneDrive - 0yt2k\xlwings\_double\_integration\_example.py"", line 40, in MidpointRule
^^^^^^^^^^^
ret = func(\*args)
File ""D:\Programs\Python\Lib\site-packages\xlwings\udfs.py"", line 502, in call\_udf
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
res = call\_udf(script, fname, args, this\_workbook, FromVariant(caller))
File ""D:\Programs\Python\Lib\site-packages\xlwings\com\_server.py"", line 235, in CallUDF
^^^^^^^^^^^
return func(\*args)
File ""D:\Programs\Python\Lib\site-packages\win32com\server\policy.py"", line 639, in *invokeex*
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
return S\_OK, -1, self.*invokeex*(dispid, lcid, wFlags, args, None, None)
File ""D:\Programs\Python\Lib\site-packages\win32com\server\policy.py"", line 310, in *invoke*
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
return self.*invoke*(dispid, lcid, wFlags, args)
File ""D:\Programs\Python\Lib\site-packages\win32com\server\policy.py"", line 305, in *Invoke*"

On using Trigonometry it caused error:

"NameError: name 'sin' is not defined. Did you mean: 'bin'?
File ""<string>"", line 1, in <module>
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
return eval(formula.replace(variable, str(x)))
File ""D:\OneDrive - 0yt2k\xlwings\_double\_integration\_example.py"", line 37, in f
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
return \_quadpack.\_qagse(func,a,b,args,full\_output,epsabs,epsrel,limit)
File ""D:\Programs\Python\Lib\site-packages\scipy\integrate\_quadpack\_py.py"", line 577, in \_quad
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
retval = \_quad(func, a, b, args, full\_output, epsabs, epsrel, limit,
File ""D:\Programs\Python\Lib\site-packages\scipy\integrate\_quadpack\_py.py"", line 465, in quad
^^^^^^^^^^^^^
result, error = quad(f, a, b)
File ""D:\OneDrive - 0yt2k\xlwings\_double\_integration\_example.py"", line 40, in MidpointRule
^^^^^^^^^^^
ret = func(\*args)
File ""D:\Programs\Python\Lib\site-packages\xlwings\udfs.py"", line 502, in call\_udf
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
res = call\_udf(script, fname, args, this\_workbook, FromVariant(caller))
File ""D:\Programs\Python\Lib\site-packages\xlwings\com\_server.py"", line 235, in CallUDF
^^^^^^^^^^^
return func(\*args)
File ""D:\Programs\Python\Lib\site-packages\win32com\server\policy.py"", line 639, in *invokeex*
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
return S\_OK, -1, self.*invokeex*(dispid, lcid, wFlags, args, None, None)
File ""D:\Programs\Python\Lib\site-packages\win32com\server\policy.py"", line 310, in *invoke*
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
return self.*invoke*(dispid, lcid, wFlags, args)
File ""D:\Programs\Python\Lib\site-packages\win32com\server\policy.py"", line 305, in *Invoke*"

Bing

I apologize for the confusion. The eval function in Python evaluates a string as a Python expression within the current namespace. If the string contains a function name like sin, eval will look for a definition of sin in the current namespace. If it doesn't find one, it will raise a NameError.

To fix this, you can import the math module and add its functions to the namespace in which eval is called. You can do this by creating a dictionary that maps function names to their corresponding functions in the math module, and passing this dictionary as the second argument to eval.

Here's how you can modify your code:

import xlwings as xw

import numpy as np

from scipy.integrate import quad

import math

@xw.func

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

@xw.arg('var', xw.Range) # Ensure 'var' is an xlwings.Range object

@xw.arg('a')

@xw.arg('b')

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

# Convert a and b to float

a = float(a)

b = float(b)

# Get the formula from the cell

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

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

# Map of Excel function names to Python function names

func\_map = {

'SIN': 'sin',

'COS': 'cos',

'TAN': 'tan',

# Add more if needed

}

# Replace function names in the formula

for excel\_func, python\_func in func\_map.items():

formula = formula.replace(excel\_func, python\_func)

# Get the variable from the cell

variable = str(var.value)

# Define the function to integrate

def f(x):

# Create a dictionary with the math module's functions and the variable

namespace = {variable: x, \*\*math.\_\_dict\_\_}

return eval(formula, namespace)

# 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, the eval function has access to all functions in the math module, so it can evaluate formulas containing sin, cos, and other mathematical functions. It should also correctly handle any variable name.

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's go back to the code which worked flawlessly. See below:

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) # Ensure 'var' is an xlwings.Range object
@xw.arg('a')
@xw.arg('b')
def MidpointRule(func, var, a, b):
# Convert a and b to float
a = float(a)
b = float(b)

# Get the formula from the cell

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

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

# Get the variable from the cell

variable = str(var.value)

# Define the function to integrate

def f(x):

B1 = x # Define 'B1' before calling eval

return eval(formula.replace(variable, str(B1)))

# Perform the integration

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

# Return the result as a list of lists

return [[result]]

I want you to focus on hard coded B1 - this part of the code:

def f(x):
B1 = x # Define 'B1' before calling eval
return eval(formula.replace(variable, str(B1)))

Before above part, you have defined: variable = str(var.value)

Instead of B1=x, can't you use variable and replace it with x (in place of B1=x) so that, in place of B1 suppose the cell is say C1 or D1 or anything, it can run without errors because that would have been passed on to variable instead of B1? Do you get what I mean?

Bing

Yes, I understand what you mean. You want to replace the hardcoded B1 with the variable cell reference that you're passing to the function. This can be achieved using Python's exec function to dynamically create a variable with the name stored in variable. Here's how you can modify your 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) # Ensure 'var' is an xlwings.Range object

@xw.arg('a')

@xw.arg('b')

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

# Convert a and b to float

a = float(a)

b = float(b)

# Get the formula from the cell

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

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

# Get the variable from the cell

variable = str(var.value)

# Define the function to integrate

def f(x):

exec(f"{variable} = x") # Create a variable with the name stored in 'variable'

return eval(formula)

# 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, the exec function creates a variable with the name stored in variable and assigns it the value of x. This should allow your function to work with any cell reference.

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

Error has now changed to:

"SyntaxError: cannot assign to literal here. Maybe you meant '==' instead of '='?
^^^
1.0 = x
File ""<string>"", line 1
^^^^^^^^^^^^^^^^^^^^^^^
exec(f""{variable} = x"") # Create a variable with the name stored in 'variable'
File ""D:\OneDrive - 0yt2k\xlwings\_double\_integration\_example.py"", line 24, in f
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
return \_quadpack.\_qagse(func,a,b,args,full\_output,epsabs,epsrel,limit)
File ""D:\Programs\Python\Lib\site-packages\scipy\integrate\_quadpack\_py.py"", line 577, in \_quad
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
retval = \_quad(func, a, b, args, full\_output, epsabs, epsrel, limit,
File ""D:\Programs\Python\Lib\site-packages\scipy\integrate\_quadpack\_py.py"", line 465, in quad
^^^^^^^^^^^^^
result, error = quad(f, a, b)
File ""D:\OneDrive - 0yt2k\xlwings\_double\_integration\_example.py"", line 28, in MidpointRule
^^^^^^^^^^^
ret = func(\*args)
File ""D:\Programs\Python\Lib\site-packages\xlwings\udfs.py"", line 502, in call\_udf
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
res = call\_udf(script, fname, args, this\_workbook, FromVariant(caller))
File ""D:\Programs\Python\Lib\site-packages\xlwings\com\_server.py"", line 235, in CallUDF
^^^^^^^^^^^
return func(\*args)
File ""D:\Programs\Python\Lib\site-packages\win32com\server\policy.py"", line 639, in *invokeex*
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
return S\_OK, -1, self.*invokeex*(dispid, lcid, wFlags, args, None, None)
File ""D:\Programs\Python\Lib\site-packages\win32com\server\policy.py"", line 310, in *invoke*
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
return self.*invoke*(dispid, lcid, wFlags, args)
File ""D:\Programs\Python\Lib\site-packages\win32com\server\policy.py"", line 305, in *Invoke*"

Bing

I apologize for the confusion. The error is due to the fact that exec is trying to assign a value to a literal (1.0 in this case), which is not allowed in Python. This happens because variable is holding the value of the cell, not the cell reference itself.

To fix this, you need to pass the cell reference (like 'B1', 'C1', etc.) to the function instead of the cell value. You can do this by changing the @xw.arg('var', xw.Range) decorator to @xw.arg('var'). This will ensure that var is treated as a string, and you can safely use it as a variable name.

Here's the corrected code:

import xlwings as xw

import numpy as np

from scipy.integrate import quad

import math

@xw.func

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

@xw.arg('var') # Treat 'var' as a string

@xw.arg('a')

@xw.arg('b')

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

# Convert a and b to float

a = float(a)

b = float(b)

# Get the formula from the cell

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

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

# Get the variable from the cell

variable = str(var)

# Define the function to integrate

def f(x):

exec(f"{variable} = x") # Create a variable with the name stored in 'variable'

return eval(formula)

# Perform the integration

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

# Return the result as a list of lists

return [[result]]

Now, when you call the function with =MidpointRule(A1,"B1",1,2), B1 should be treated as a string, and exec should be able to create a variable with the name B1 without any issues. 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](#message|)
That worked! Thanks.
[assistant]=(#message|)
You're welcome! I'm glad to hear that it worked! If you have any more questions or need further help with Python, xlwings, or anything else, feel free to ask. I'm here to help! 😊

User

That one worked. But now let us try to fix this final bug so that when I give input in the function - say B1, it reads as string instead of my manually inputting "B1".

Bing

I understand your requirement. You want to pass the cell reference (like 'B1') directly to the function, without having to enclose it in quotes. This can be achieved by modifying the @xw.arg('var') decorator to @xw.arg('var', str). This will ensure that var is treated as a string, and you can safely use it as a variable name.

Here's the corrected code:

import xlwings as xw

import numpy as np

from scipy.integrate import quad

import math

@xw.func

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

@xw.arg('var', str) # Treat 'var' as a string

@xw.arg('a')

@xw.arg('b')

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

# Convert a and b to float

a = float(a)

b = float(b)

# Get the formula from the cell

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

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

# Get the variable from the cell

variable = var

# Define the function to integrate

def f(x):

exec(f"{variable} = x") # Create a variable with the name stored in 'variable'

return eval(formula)

# Perform the integration

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

# Return the result as a list of lists

return [[result]]

Now, when you call the function with =MidpointRule(A1,B1,1,2), B1 should be treated as a string, and exec should be able to create a variable with the name B1 without any issues. 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

Why this error even though I changed: @xw.arg('var', str)?

"AttributeError: type object 'str' has no attribute 'reader'
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
pipeline = accessors.get(convert, convert).reader(options)
File ""D:\Programs\Python\Lib\site-packages\xlwings\conversion\_*init*\_.py"", line 75, in read
^^^^^^^^^^^^^^^^
args[i] = conversion.read(
File ""D:\Programs\Python\Lib\site-packages\xlwings\udfs.py"", line 440, in call\_udf
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
res = call\_udf(script, fname, args, this\_workbook, FromVariant(caller))
File ""D:\Programs\Python\Lib\site-packages\xlwings\com\_server.py"", line 235, in CallUDF
^^^^^^^^^^^
return func(\*args)
File ""D:\Programs\Python\Lib\site-packages\win32com\server\policy.py"", line 639, in *invokeex*
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
return S\_OK, -1, self.*invokeex*(dispid, lcid, wFlags, args, None, None)
File ""D:\Programs\Python\Lib\site-packages\win32com\server\policy.py"", line 310, in *invoke*
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
return self.*invoke*(dispid, lcid, wFlags, args)
File ""D:\Programs\Python\Lib\site-packages\win32com\server\policy.py"", line 305, in *Invoke*"

Is there any other change required in the code?

Bing

I apologize for the confusion. The @xw.arg('var', str) decorator is not working as expected because xlwings does not support string conversion for function arguments out of the box.

However, you can achieve your goal by a workaround. You can pass the cell reference as a string by appending a single quote before it. Excel treats anything following a single quote as a string. So, you can call your function like this: =MidpointRule(A1, 'B1', 1, 2). This way, 'B1' will be treated as a string, and you can use it as a variable name in your Python code.

Here's your code which should work with this approach:

import xlwings as xw

import numpy as np

from scipy.integrate import quad

@xw.func

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

@xw.arg('var') # Treat 'var' as a string

@xw.arg('a')

@xw.arg('b')

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

# Convert a and b to float

a = float(a)

b = float(b)

# Get the formula from the cell

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

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

# Get the variable from the cell

variable = var

# Define the function to integrate

def f(x):

exec(f"{variable} = x") # Create a variable with the name stored in 'variable'

return eval(formula)

# Perform the integration

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

# Return the result as a list of lists

return [[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

Can you use some other additional library, pass on B1 to that which then converts into string and then use above code out of the box?

Bing

I understand your requirement, but unfortunately, the way Excel and xlwings work doesn't allow us to pass a cell reference like 'B1' directly as a string without enclosing it in quotes. This is because Excel treats anything not enclosed in quotes as a cell reference or a formula, not a string.

However, there's a workaround that might help. You can create a helper function in Excel that takes a cell reference and returns it as a string. Here's how you can do it:

1. Press Alt + F11 to open the VBA editor.
2. Click Insert -> Module to create a new module.
3. In the module, paste the following VBA code:

Function CellToString(ref As Range) As String

CellToString = ref.Address(False, False)

End Function

1. Press Ctrl + S to save, and Alt + Q to close the VBA editor.

Now, you have a custom Excel function CellToString that takes a cell reference and returns it as a string. You can use this function to pass a cell reference to your Python function as a string. For example, you can call your function like this: =MidpointRule(A1, CellToString(B1), 1, 2).

Please note that this workaround requires enabling macros in your Excel workbook, which might have security implications. Always be careful when running macros from sources you don't trust.

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

User

You said:

I apologize for the confusion. The @xw.arg('var', str) decorator is not working as expected because xlwings does not support string conversion for function arguments out of the box.

How about @xw.arg('var') and then converting it to string? Is that posssible using some other excel related library in python?

Bing

The @xw.arg('var') decorator without a type specification treats the argument as an xlwings.Range object by default. This is why you can access properties like var.value in your function. If you try to convert it to a string, you'll get a string representation of the xlwings.Range object, not the cell reference.

Unfortunately, xlwings does not provide a built-in way to pass cell references as strings directly. The library is designed to work with cell values and ranges, not cell references. This is why you need to use a workaround like the one I suggested earlier (passing the cell reference as a string by appending a single quote before it).

There are other libraries for working with Excel files in Python, like openpyxl or pandas, but they also don't support passing cell references as strings directly from Excel. They are more focused on manipulating Excel data within Python, not on interacting with Excel in real time like xlwings does.

I understand that this might not be the ideal solution for your use case, but it's a limitation of the way Excel and xlwings interact. I hope this clarifies the issue. If you have any more questions, feel free to ask. I'm here to help! 😊

User

Okay, let us change the dimension completely. Let's have only 3 srguments - A1 and two others a & b in case of your code. Then in this part:

def f(x):
exec(f"{variable} = x") # Create a variable with the name stored in 'variable'
return eval(formula)

change or create x out of the box. Do you get it?

Bing

Yes, I understand your requirement. You want to modify the function to take only three arguments: a formula in cell A1, and two numbers for the limits of integration. The variable in the formula will be 'x', which will be defined within the function itself.

Here's how you can modify your code:

import xlwings as xw

import numpy as np

from scipy.integrate import quad

@xw.func

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

@xw.arg('a')

@xw.arg('b')

def MidpointRule(func, a, b):

# Convert a and b to float

a = float(a)

b = float(b)

# Get the formula from the cell

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

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

# Define the function to integrate

def f(x):

return eval(formula)

# 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, 'x' is the variable of integration, and it's defined by the quad function when it calls f(x). The formula in cell A1 should be written in terms of 'x'. For example, if you want to integrate the function sin(x) from 0 to π, you would write the formula in Excel as =SIN(x), and call your function as =MidpointRule(A1,0,3.14159).

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

We are very near to the simple solution withn 3 arguments. In Cell A1 when I input 1/x, it is treated as string and function results in error, but when I enter =1/x, although cell A1 show's excel's internal error #NAME!, the function works. So just change your code so that it takes A1 input as string and not formula like =1/x.

Bing

I understand your requirement. You want to input the formula as a string in cell A1, without the leading '=' sign. This can be achieved by modifying the way the formula is extracted from the cell. Here's how you can modify your code:

import xlwings as xw

import numpy as np

from scipy.integrate import quad

@xw.func

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

@xw.arg('a')

@xw.arg('b')

def MidpointRule(func, a, b):

# Convert a and b to float

a = float(a)

b = float(b)

# Get the formula from the cell

formula = func.value # Get the cell value directly as a string

# Define the function to integrate

def f(x):

return eval(formula.replace('x', str(x))) # Replace 'x' with the value of 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.value is used to get the cell value directly as a string. The replace function is used to replace 'x' with the value of x in the formula. Now, you can input the formula as a string in cell A1, like '1/x', and call your function as =MidpointRule(A1,1,2).

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

So that worked finally - Now let's go to next stage. Trignometry compatibility of this function. When I input sin(x) in cell A1, it results in error:

Bing

The error is likely because the Python eval function does not recognize sin as it's not defined in the current namespace. To fix this, you can import the math module and add its functions to the namespace in which eval is called. Here's how you can modify your code:

import xlwings as xw

import numpy as np

from scipy.integrate import quad

import math

@xw.func

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

@xw.arg('a')

@xw.arg('b')

def MidpointRule(func, a, b):

# Convert a and b to float

a = float(a)

b = float(b)

# Get the formula from the cell

formula = func.value # Get the cell value directly as a string

# Define the function to integrate

def f(x):

# Create a dictionary with the math module's functions

namespace = math.\_\_dict\_\_

# Add 'x' to the namespace

namespace['x'] = x

return eval(formula, namespace)

# 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, the eval function has access to all functions in the math module, so it can evaluate formulas containing sin, cos, and other mathematical functions. It should also correctly handle 'x' as the variable of integration.

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 same error as before import math:

"NameError: name 'sin' is not defined. Did you mean: 'bin'?
File ""<string>"", line 1, in <module>
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
return eval(formula.replace('x', str(x))) # Replace 'x' with the value of x
File ""D:\OneDrive - 0yt2k\xlwings\_double\_integration\_example.py"", line 20, in f
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
return \_quadpack.\_qagse(func,a,b,args,full\_output,epsabs,epsrel,limit)
File ""D:\Programs\Python\Lib\site-packages\scipy\integrate\_quadpack\_py.py"", line 577, in \_quad
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
retval = \_quad(func, a, b, args, full\_output, epsabs, epsrel, limit,
File ""D:\Programs\Python\Lib\site-packages\scipy\integrate\_quadpack\_py.py"", line 465, in quad
^^^^^^^^^^^^^
result, error = quad(f, a, b)
File ""D:\OneDrive - 0yt2k\xlwings\_double\_integration\_example.py"", line 23, in MidpointRule
^^^^^^^^^^^
ret = func(\*args)
File ""D:\Programs\Python\Lib\site-packages\xlwings\udfs.py"", line 502, in call\_udf
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
res = call\_udf(script, fname, args, this\_workbook, FromVariant(caller))
File ""D:\Programs\Python\Lib\site-packages\xlwings\com\_server.py"", line 235, in CallUDF
^^^^^^^^^^^
return func(\*args)
File ""D:\Programs\Python\Lib\site-packages\win32com\server\policy.py"", line 639, in *invokeex*
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
return S\_OK, -1, self.*invokeex*(dispid, lcid, wFlags, args, None, None)
File ""D:\Programs\Python\Lib\site-packages\win32com\server\policy.py"", line 310, in *invoke*
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
return self.*invoke*(dispid, lcid, wFlags, args)
File ""D:\Programs\Python\Lib\site-packages\win32com\server\policy.py"", line 305, in *Invoke*"

Bing

I apologize for the confusion. The issue is that the eval function does not have access to the sin function from the math module in its current namespace. To fix this, you can use the locals() function to get the current local symbol table, which includes the math module since it was imported. Here's how you can modify your code:

import xlwings as xw

import numpy as np

from scipy.integrate import quad

import math

@xw.func

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

@xw.arg('a')

@xw.arg('b')

def MidpointRule(func, a, b):

# Convert a and b to float

a = float(a)

b = float(b)

# Get the formula from the cell

formula = func.value # Get the cell value directly as a string

# Define the function to integrate

def f(x):

# Create a dictionary with the current local symbol table

namespace = locals()

return eval(formula, namespace)

# 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, the eval function has access to all local variables, including the math module, so it can evaluate formulas containing sin, cos, and other mathematical functions. It should also correctly handle 'x' as the variable of integration.

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

My code is like this:
import xlwings as xw
import numpy as np
from scipy.integrate import quad
import math

@xw.func
@xw.arg('func', xw.Range)
@xw.arg('a')
@xw.arg('b')
def MidpointRule(func, a, b):
# Convert a and b to float
a = float(a)
b = float(b)

# Get the formula from the cell

formula = func.value # Get the cell value directly as a string

# Define the function to integrate

def f(x):

# Create a dictionary with the current local symbol table

namespace = locals()

return eval(formula, namespace)

# Perform the integration

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

# Return the result as a list of lists

return [[result]]

I use funcion like this: =MidpointRule(A1,1,2) Cell A1 contains string sin(x) (sin in lowercase)

Error:

"NameError: name 'sin' is not defined. Did you mean: 'bin'?
File ""<string>"", line 1, in <module>
^^^^^^^^^^^^^^^^^^^^^^^^
return eval(formula, namespace)
File ""D:\OneDrive - 0yt2k\xlwings\_double\_integration\_example.py"", line 22, in f
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
return \_quadpack.\_qagse(func,a,b,args,full\_output,epsabs,epsrel,limit)
File ""D:\Programs\Python\Lib\site-packages\scipy\integrate\_quadpack\_py.py"", line 577, in \_quad
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
retval = \_quad(func, a, b, args, full\_output, epsabs, epsrel, limit,
File ""D:\Programs\Python\Lib\site-packages\scipy\integrate\_quadpack\_py.py"", line 465, in quad
^^^^^^^^^^^^^
result, error = quad(f, a, b)
File ""D:\OneDrive - 0yt2k\xlwings\_double\_integration\_example.py"", line 25, in MidpointRule
^^^^^^^^^^^
ret = func(\*args)
File ""D:\Programs\Python\Lib\site-packages\xlwings\udfs.py"", line 502, in call\_udf
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
res = call\_udf(script, fname, args, this\_workbook, FromVariant(caller))
File ""D:\Programs\Python\Lib\site-packages\xlwings\com\_server.py"", line 235, in CallUDF
^^^^^^^^^^^
return func(\*args)
File ""D:\Programs\Python\Lib\site-packages\win32com\server\policy.py"", line 639, in *invokeex*
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
return S\_OK, -1, self.*invokeex*(dispid, lcid, wFlags, args, None, None)
File ""D:\Programs\Python\Lib\site-packages\win32com\server\policy.py"", line 310, in *invoke*
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
return self.*invoke*(dispid, lcid, wFlags, args)
File ""D:\Programs\Python\Lib\site-packages\win32com\server\policy.py"", line 305, in *Invoke*"

Bing

I apologize for the confusion. The issue is that the eval function does not have access to the sin function from the math module in its current namespace. To fix this, you can use the locals() function to get the current local symbol table, which includes the math module since it was imported. Here's how you can modify your code:

import xlwings as xw

import numpy as np

from scipy.integrate import quad

import math

@xw.func

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

@xw.arg('a')

@xw.arg('b')

def MidpointRule(func, a, b):

# Convert a and b to float

a = float(a)

b = float(b)

# Get the formula from the cell

formula = func.value # Get the cell value directly as a string

# Define the function to integrate

def f(x):

# Create a dictionary with the current local symbol table

namespace = {\*\*locals(), \*\*math.\_\_dict\_\_}

return eval(formula, namespace)

# 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, the eval function has access to all local variables, including the math module, so it can evaluate formulas containing sin, cos, and other mathematical functions. It should also correctly handle 'x' as the variable of integration.

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 applied the function like this:

=MidpointRule(A1,60,180)

in Cell A1 - I have sin(x) The answer returned is -0.353952911

But When I use Midpointrule defined via vba code and applied like =MidpointRule(D1,E1,RADIANS(60),RADIANS(180)) (here E1 is just like x)

answer returned is 1.50. vba code is:

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

Why Python returns wrong value?

Bing

The discrepancy between the results of the Python and VBA functions is due to the difference in the way the trigonometric function sin(x) is interpreted in the two languages.

In Python, the math.sin function expects the argument to be in radians, while in Excel, the SIN function expects the argument to be in degrees.

In your Python function, you're passing the limits of integration as 60 and 180, which are being interpreted as radians. This is equivalent to approximately 3438 and 10313 degrees, respectively, which is why you're getting an unexpected result.

To get the correct result, you need to convert the limits of integration from degrees to radians before passing them to the Python function. You can do this in Excel using the RADIANS function. So instead of =MidpointRule(A1,60,180), you should use =MidpointRule(A1,RADIANS(60),RADIANS(180)).

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

User

Changed accordingly and worked. Now same code with changes for double integration.

Bing

I'm glad to hear that the single integration function worked for you! Now, let's extend this to a double integration function.

In Python, you can perform double integration using the dblquad function from the scipy.integrate module. The dblquad function takes two functions as arguments that define the lower and upper limits of the second integral.

Here's how you can modify your code to perform double integration:

import xlwings as xw

import numpy as np

from scipy.integrate import dblquad

import math

@xw.func

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

@xw.arg('a')

@xw.arg('b')

@xw.arg('c')

@xw.arg('d')

def DoubleIntegral(func, a, b, c, d):

# Convert the limits to float

a = float(a)

b = float(b)

c = float(c)

d = float(d)

# Get the formula from the cell

formula = func.value # Get the cell value directly as a string

# Define the function to integrate

def f(y, x): # Note the order of the arguments

# Create a dictionary with the current local symbol table

namespace = {\*\*locals(), \*\*math.\_\_dict\_\_}

return eval(formula, namespace)

# 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, the dblquad function is used to perform the double integration. The formula in cell A1 should be written in terms of 'x' and 'y'. For example, if you want to integrate the function sin(x\*y) over the rectangle [1,2]x[3,4], you would write the formula in Excel as =SIN(x\*y), and call your function as =DoubleIntegral(A1,1,2,3,4).

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

Double integral means for example integration of integration of 1/x. So integration of 1/x is logx and integration of logx is xlogx -x. So When I apply function like say: =DoubleIntegral(C1,1,2,1,2) the answer is 0.693147181 But even in single integral like this: =MidpointRule(C1,1,2) (as you can see I have taken same cell and same parameters) the answer is the same: 0.693147181 Is there something wrong in my understanding about double integral?

Bing

I see where the confusion is coming from. The concept of a double integral is not simply integrating the result of a single integral again. Rather, it's used to integrate a function of two variables over a region in a plane.

In your example, you're performing a single integral of the function $$\frac{1}{x}$$ over the interval [1,2], which results in $$\ln(2) \approx 0.693147181$$.

When you're trying to perform a double integral of the same function over the rectangle [1,2]x[1,2], it's not equivalent to integrating $$\ln(x)$$ (the result of the single integral) again. Instead, you're integrating the function $$\frac{1}{x}$$ over a two-dimensional region. In this case, the function $$\frac{1}{x}$$ is not a suitable integrand for a double integral, because it's not a function of two variables.

If you want to perform a double integral, your integrand should be a function of two variables. For example, you could use the function $$f(x,y) = x^2 + y^2$$, and integrate it over the rectangle [1,2]x[1,2]. This would give you the volume under the surface $$z = x^2 + y^2$$ above the rectangle [1,2]x[1,2].

I hope this clarifies the concept of double integrals. If you have any more questions, feel free to ask. I'm here to help! 😊

User

So if my function is not compatible with double integration, why did it not returned error when I applied python xw function as above and instead it gave same answer as single integration?

Bing

The double integral of a function over a region is computed by integrating the function with respect to one variable, and then integrating the result with respect to the other variable. When you performed the double integral of the function $$\frac{1}{x}$$ over the rectangle [1,2]x[1,2], the function was integrated with respect to 'y' first, but since the function does not depend on 'y', this first integral just gave you the function back, multiplied by the width of the interval for 'y', which is 2-1=1. Then, the result was integrated with respect to 'x' over the interval [1,2], which is why you got the same result as the single integral.

In other words, the double integral of the function $$\frac{1}{x}$$ over the rectangle [1,2]x[1,2] is equivalent to the single integral of the function $$\frac{1}{x}$$ over the interval [1,2], because the function does not depend on 'y'.

This is not an error, but rather a consequence of the definition of the double integral and the specific function and region you chose. If you choose a function that depends on both 'x' and 'y', you will see a different result for the double integral compared to the single integral. I hope this clarifies your question! 😊

User

Oh! that means my inderstanding about double integration is wrong that it is nesting of single integrations multiple times! Okay - thanks for clarifying.

But then I tried like this:

=DoubleIntegral(D1,1,2,3,4) - D1 has formula in string format as 1/xy

Why that has resulted into error:

"NameError: name 'xy' is not defined
File ""<string>"", line 1, in <module>
^^^^^^^^^^^^^^^^^^^^^^^^
return eval(formula, namespace)
File ""D:\OneDrive - 0yt2k\xlwings\_double\_integration\_example.py"", line 64, in f
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
return \_quadpack.\_qagse(func,a,b,args,full\_output,epsabs,epsrel,limit)
File ""D:\Programs\Python\Lib\site-packages\scipy\integrate\_quadpack\_py.py"", line 577, in \_quad
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
retval = \_quad(func, a, b, args, full\_output, epsabs, epsrel, limit,
File ""D:\Programs\Python\Lib\site-packages\scipy\integrate\_quadpack\_py.py"", line 465, in quad
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
quad\_r = quad(f, low, high, args=args, full\_output=self.full\_output,
File ""D:\Programs\Python\Lib\site-packages\scipy\integrate\_quadpack\_py.py"", line 1233, in integrate
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
return \_quadpack.\_qagse(func,a,b,args,full\_output,epsabs,epsrel,limit)
File ""D:\Programs\Python\Lib\site-packages\scipy\integrate\_quadpack\_py.py"", line 577, in \_quad
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
retval = \_quad(func, a, b, args, full\_output, epsabs, epsrel, limit,
File ""D:\Programs\Python\Lib\site-packages\scipy\integrate\_quadpack\_py.py"", line 465, in quad
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
quad\_r = quad(f, low, high, args=args, full\_output=self.full\_output,
File ""D:\Programs\Python\Lib\site-packages\scipy\integrate\_quadpack\_py.py"", line 1233, in integrate
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
return \_NQuad(func, ranges, opts, full\_output).integrate(\*args)
File ""D:\Programs\Python\Lib\site-packages\scipy\integrate\_quadpack\_py.py"", line 1179, in nquad
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
return nquad(func, [temp\_ranges, [a, b]], args=args,
File ""D:\Programs\Python\Lib\site-packages\scipy\integrate\_quadpack\_py.py"", line 775, in dblquad
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
result, error = dblquad(f, a, b, lambda x: c, lambda x: d)
File ""D:\OneDrive - 0yt2k\xlwings\_double\_integration\_example.py"", line 67, in DoubleIntegral
^^^^^^^^^^^
ret = func(\*args)
File ""D:\Programs\Python\Lib\site-packages\xlwings\udfs.py"", line 502, in call\_udf
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
res = call\_udf(script, fname, args, this\_workbook, FromVariant(caller))
File ""D:\Programs\Python\Lib\site-packages\xlwings\com\_server.py"", line 235, in CallUDF
^^^^^^^^^^^
return func(\*args)
File ""D:\Programs\Python\Lib\site-packages\win32com\server\policy.py"", line 639, in *invokeex*
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
return S\_OK, -1, self.*invokeex*(dispid, lcid, wFlags, args, None, None)
File ""D:\Programs\Python\Lib\site-packages\win32com\server\policy.py"", line 310, in *invoke*
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
return self.*invoke*(dispid, lcid, wFlags, args)
File ""D:\Programs\Python\Lib\site-packages\win32com\server\policy.py"", line 305, in *Invoke*"

I think you need to modify code or should I change string as 1/x\*y or something compatible with your code in cell D1?

Bing

The error you're seeing is because the string '1/xy' is being interpreted as a single variable named 'xy', not as 'x' times 'y'. To fix this, you should change the string in cell D1 to '1/(x\*y)'. This will ensure that 'x' and 'y' are recognized as separate variables.

Here's how you can modify your code:

import xlwings as xw

import numpy as np

from scipy.integrate import dblquad

import math

@xw.func

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

@xw.arg('a')

@xw.arg('b')

@xw.arg('c')

@xw.arg('d')

def DoubleIntegral(func, a, b, c, d):

# Convert the limits to float

a = float(a)

b = float(b)

c = float(c)

d = float(d)

# Get the formula from the cell

formula = func.value # Get the cell value directly as a string

# Define the function to integrate

def f(x, y): # Note the order of the arguments

# Create a dictionary with the current local symbol table

namespace = {\*\*locals(), \*\*math.\_\_dict\_\_}

return eval(formula, namespace)

# 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]]

Now, when you call the function with =DoubleIntegral(D1,1,2,3,4), where D1 contains the string '1/(x\*y)', it should work correctly.

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

After I inserted in D1 as 1/(x\*y), the answer of double integral with above code provided by you is: 0.199406017417594. Formula used by me: =DoubleIntegral(D1,1,2,3,4)

Can you explain me step by step how this answer is arrived at to check whether it is really correct or not?

Bing

Sure, I'd be happy to explain how the double integral is computed.

The double integral of a function over a region in the plane is a way of adding up the values of the function at all points in the region. In your case, the function is $$f(x, y) = \frac{1}{x \cdot y}$$ and the region is the rectangle with corners at (1, 3), (1, 4), (2, 3), and (2, 4).

The dblquad function from the scipy.integrate module computes the double integral by dividing the region into small rectangles, approximating the value of the function at the center of each rectangle, multiplying by the area of the rectangle, and adding up all these values. The more rectangles, the better the approximation.

Here's a step-by-step breakdown of the computation:

1. The region of integration is defined by the limits of 'x' from 1 to 2 and 'y' from 3 to 4.
2. The function $$f(x, y) = \frac{1}{x \cdot y}$$ is evaluated at a grid of points within this region.
3. The value of the function at each point is multiplied by the area of a small rectangle around that point. The size of the rectangle is determined by the precision of the dblquad function.
4. All these values are added up to give the approximate value of the double integral.
5. The dblquad function repeats this process with smaller and smaller rectangles until the change in the result is below a certain threshold, at which point it returns the result.

The result you got, 0.199406017417594, is the value of the double integral of $$\frac{1}{x \cdot y}$$ over the rectangle [1,2]x[3,4] computed by this method.

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

User

I cheked answers as per your code with online integration calculator sites like wolfarm and they match!

Bing

That's great to hear! I'm glad the results from the Python code matched with the online calculators. It's always a good idea to cross-verify results, especially when dealing with complex calculations like double integrals. If you have any more questions or need further help with Python, math, or anything else, feel free to ask. I'm here to help! 😊

User

You have used from scipy.integrate import dblquad to find double integration. I believe that Python itself has background coding in C++. Whether there is any library in C++ to access these functions directly instead of having python interpreter in between? I think in that case using cffi library as bridge, we can call function directly from dll compiled from C++ code created proprietary way. That wpuld be mush faster than using scipy.integrate?

Bing

Yes, you're correct that many scientific computing libraries in Python, including SciPy, leverage C++ (or sometimes Fortran or C) for performance-critical sections of code. This allows them to provide the speed of compiled languages with the ease of use of Python.

If you want to call these functions directly from C++, there are a few options:

1. **Use the original C++ libraries**: Many functions in SciPy are based on well-established C++ libraries. For example, the dblquad function in SciPy is a wrapper around the QUADPACK library, which is written in Fortran. You could use these libraries directly from your C++ code.
2. **Use a C++ numerical library**: There are several numerical libraries available for C++ that provide similar functionality to SciPy, such as Boost.Numeric, Eigen, or Armadillo. These libraries are designed to be used directly from C++ and can provide excellent performance.
3. **Use Python's C API or a tool like SWIG or pybind11**: Python provides a C API that allows you to call Python code from C or C++, and tools like SWIG (Simplified Wrapper and Interface Generator) or pybind11 can make this process easier. However, this would still involve the Python interpreter, so it may not provide the performance benefits you're looking for.
4. **Use the CFFI library**: As you mentioned, the CFFI (C Foreign Function Interface) library allows you to call C functions from Python. If you have a DLL compiled from C++ code, you could use CFFI to call functions from that DLL. However, creating a C-compatible DLL from C++ code can be complex, depending on the features of C++ you're using.

Please note that while C++ can provide performance benefits, it also comes with increased complexity and reduced ease of use compared to Python. The best choice depends on your specific needs and constraints.

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