sympy, scipy, and integration

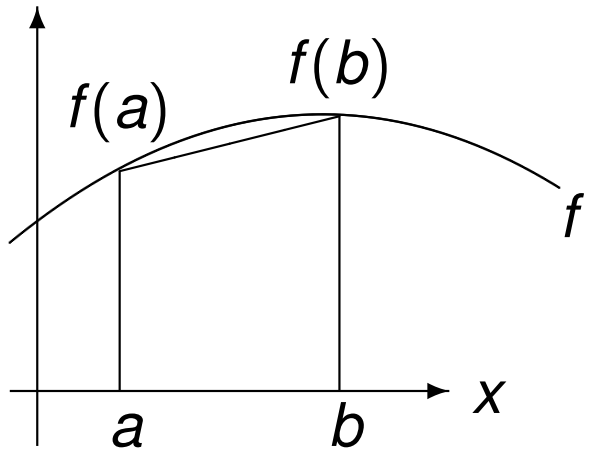
1. Functions using Functions
   - functions as arguments of other functions
   - the one-line if-else statement
   - functions returning multiple values
   - the composite trapezium rule
2. Constructing Integration Rules with `sympy`
   - about sympy
   - algebraic degree of accuracy
3. Optional and Keyword Arguments
   - default values
   - numerical integration in scipy
   - optional and keyword arguments

MCS 507 Lecture 5
Mathematical, Statistical and Scientific Software
Jan Verschelde, 30 August 2023

sympy, scipy, and integration

1. Functions using Functions
   - functions as arguments of other functions
   - the one-line if-else statement
   - functions returning multiple values
   - the composite trapezium rule
2. Constructing Integration Rules with `sympy`
   - about sympy
   - algebraic degree of accuracy
3. Optional and Keyword Arguments
   - default values
   - numerical integration in scipy
   - optional and keyword arguments

the trapezoidal rule



$$\int_a^b f(x)dx \approx \left(\frac{b-a}{2}\right)(f(a)+f(b))$$

functions as arguments

Interactively, in a `lambda` statement:

```
>>> t = lambda f,a,b: (b-a)*(f(a)+f(b))/2
>>> from math import exp
>>> t(exp,0.5,0.7)
0.3662473978170604
```

Using functions in a script:

```
def traprule(fun, left, right):
    """
    trapezoidal rule for fun(x) over [left, right]
    """
    return (right-left)*(fun(left) + fun(right))/2
```

the whole script

```
def traprule(fun, left, right):
    """
    trapezoidal rule for fun(x) over [left, right]
    """
    return (right-left)*(fun(left) + fun(right))/2

import math
S = 'integrating exp() over '
print(S + '[a,b]')
A = float(input('give a : '))
B = float(input('give b : '))
Y = traprule(math.exp, A, B)
print(S + '[%.1E,%.1E] : ' % (A, B))
print('the approximation : %.15E' % Y)
E = math.exp(B) - math.exp(A)
print('  the exact value : %.15E' % E)
```

running `traprule.py`

At the command prompt `$` we type:

```
$ python traprule.py
integrating exp() over [a,b]
give a : 0.5
give b : 0.7
integrating exp() over [5.0E-01,7.0E-01] :
the approximation : 3.662473978170604E-01
  the exact value : 3.650314367703484E-01
```

sympy, scipy, and integration

1. Functions using Functions
   - functions as arguments of other functions
   - the one-line if-else statement
   - functions returning multiple values
   - the composite trapezium rule
2. Constructing Integration Rules with `sympy`
   - about sympy
   - algebraic degree of accuracy
3. Optional and Keyword Arguments
   - default values
   - numerical integration in scipy
   - optional and keyword arguments

if-else on one line

In case the statements in if-else tests are short,
there is the one-line syntax for the if-else statement.

The illustration below avoids the exception
`"ValueError: math domain error"` which is triggered
when `asin` gets a value outside $[-1, +1]$.

```
from math import asin, acos

X = float(input("give a number : "))
SAFE_ASIN = (asin(X) if -1 <= X <= +1 \
    else "undefined")
print(('asin(%f) is' % X), SAFE_ASIN)
```

if-else in lambda

The one-line if-else statement is very useful
in the lambda statement to define functions:

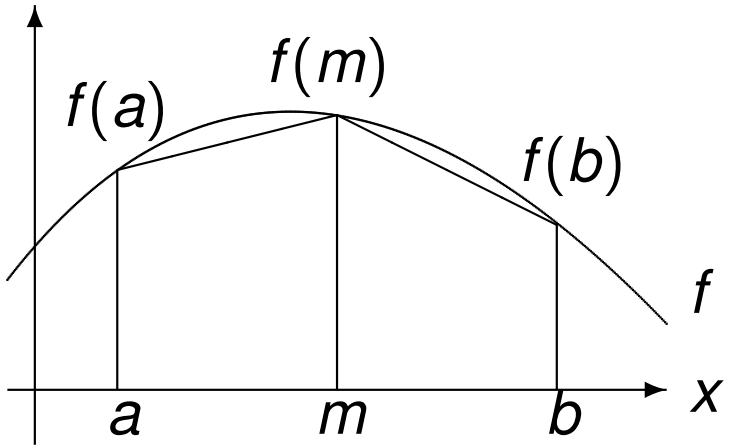
```
from numpy import NaN

ACOS_FUN = \
lambda x: (acos(x) if -1 <= x <= +1 else NaN)
Y = float(input("give a number : "))
print(('acos(%f) is' % Y), ACOS_FUN(Y))
```

From `numpy` we import `NaN`,
`NaN` is the "Not a Number" float.

sympy, scipy, and integration

1. Functions using Functions
   - functions as arguments of other functions
   - the one-line if-else statement
   - functions returning multiple values
   - the composite trapezium rule
2. Constructing Integration Rules with `sympy`
   - about sympy
   - algebraic degree of accuracy
3. Optional and Keyword Arguments
   - default values
   - numerical integration in scipy
   - optional and keyword arguments

an error estimate



$$\left(\frac{b-a}{4}\right)\left[f(a)+f\left(\frac{a+b}{2}\right)\right]+\left(\frac{b-a}{4}\right)\left[f\left(\frac{a+b}{2}\right)+f(b)\right]$$

$$=(b-a)(f(a)+f(b))/4+(b-a)f(m)/2, \quad m=(a+b)/2$$

returning multiple values

A new `traprule` returns the approximation
and an error estimate:

```
def traprule(fun, left, right):
    """
    This trapezoidal rule for fun(x) on [left,right]
    returns an approximation and an error estimate.
    """
    approx = (right-left)*(fun(left) + fun(right))/2
    middle = (left+right)/2
    result = approx/2 + (right-left)*fun(middle)/2
    return result, abs(approx-result)
```

Note that we return the more accurate approximation
as the approximation for the integral.

the main script

```
import math
S = 'integrating exp() over '
print(S + '[a,b]')
A = float(input('give a : '))
B = float(input('give b : '))
Y, E = traprule(math.exp, A, B)
print(S + '[%.1E,%.1E] : ' % (A, B))
print('the approximation : %.15E' % Y)
print('an error estimate : %.4E' % E)
E2 = math.exp(B) - math.exp(A)
print('  the exact value : %.15E' % E2)
```

running `traprule2.py`

At the command prompt `$` we type:

```
$ python traprule2.py
integrating exp() over [a,b]
give a : 0.5
give b : 0.7
integrating exp() over [5.0E-01,7.0E-01] :
the approximation : 3.653355789475811E-01
an error estimate : 9.1182E-04
  the exact value : 3.650314367703484E-01
```

sympy, scipy, and integration

1. Functions using Functions
   - functions as arguments of other functions
   - the one-line if-else statement
   - functions returning multiple values
   - the composite trapezium rule
2. Constructing Integration Rules with `sympy`
   - about sympy
   - algebraic degree of accuracy
3. Optional and Keyword Arguments
   - default values
   - numerical integration in scipy
   - optional and keyword arguments

composite quadrature rules

A composite quadrature rule over $[a, b]$ consists in

1. dividing $[a, b]$ in n subintervals, $h = (b - a)/n$;
2. applying a quadrature rule to each subinterval $[a_i, b_i]$, with $a_i = a + (i - 1)h$, and $b_i = a + ih$, for $i = 1, 2, \ldots, n$;
3. summing up the approximations on each subinterval.

Example: $[a, b] = [0, 1]$, $n = 4$: $h = 1/4$, and
the subintervals are $[0, 1/4]$, $[1/4, 1/2]$, $[1/2, 3/4]$, $[3/4, 1]$.

the composite trapezium rule

For $n = 1$:

$$\int_a^b f(x)dx \approx \frac{b-a}{2}\left(f(a)+f(b)\right).$$

For $n > 1$, $h = (b-a)/n$, $a_i = a + (i-1)h$, $b_i = a + ih$:

$$\begin{aligned}
\int_a^b f(x)dx &= \sum_{i=1}^{n} \int_{a_i}^{b_i} f(x)dx \\
&= \sum_{i=1}^{n} \frac{h}{2}\left(f(a_i)+f(b_i)\right) \\
&= \frac{h}{2}\left(f(a)+f(b)\right) + h\sum_{i=1}^{n-1} f(a+ih)
\end{aligned}$$

Observe that $a_i = b_{i-1}$ and the value $f(b_{i-1})$ at the right of the $(i-1)$-th interval equals the value $f(a_i)$ at the left of the i-th interval.

an application of vectorization

Exercise 1:

1. Write a Python script with a function to evaluate the composite trapezium rule on any given function.

   Test the rule on $\int_0^1 e^{-x^2} \sin(x) dx$,

   for an increasing number of function evaluations.

Exercise 2:

1. Apply vectorization to evaluate the function on a vector for the evaluate of the composite trapezoidal rule.

   Test the vectorized version also on $\int_0^1 e^{-x^2} \sin(x) dx$.

   For sufficiently large number of function evaluations, compare the execution times for the version of exercise 1 with the vectorized version.

sympy, scipy, and integration

1. Functions using Functions
   - functions as arguments of other functions
   - the one-line if-else statement
   - functions returning multiple values
   - the composite trapezium rule

2. Constructing Integration Rules with `sympy`
   - about sympy
   - algebraic degree of accuracy

3. Optional and Keyword Arguments
   - default values
   - numerical integration in scipy
   - optional and keyword arguments

about sympy

- SymPy is pure Python package for symbolic computation, integrated in SageMath and part of the SciPy stack.
- The *pure* means that, in contrast to SageMath, no modifications to the Python scripting language have been made.
- SageMath is licensed under the GNU GPL, whereas the license of SymPy is BSD.
- A comparison of SymPy versus SageMath is given at `https://github.com/sympy/sympy/wiki/SymPy-vs.-Sage`.
- To try online point your browser to `http://live.sympy.org`.
- A short introduction in book format (57 pages) is *Instant SymPy Starter* by Ronan Lamy, Packt Publishing, 2013.

sympy, scipy, and integration

1. Functions using Functions
   - functions as arguments of other functions
   - the one-line if-else statement
   - functions returning multiple values
   - the composite trapezium rule
2. Constructing Integration Rules with `sympy`
   - about sympy
   - algebraic degree of accuracy
3. Optional and Keyword Arguments
   - default values
   - numerical integration in scipy
   - optional and keyword arguments

making integration rules

An integration rule with 3 function evaluations:

$$\int_a^b f(x)dx \approx w_a f(a) + w_m f\left(\frac{a+b}{2}\right) + w_b f(b)$$

We evaluate f at a, $(a+b)/2$, and b.

We look for 3 weights: w_a, w_m, and w_b.

With 3 unknown weights we can require that all quadrics are integrated correctly because:

1. the integration operator is a linear operator and
2. it suffices to require that the 3 basic functions 1, x, and x^2 are integrated correctly.

solving a linear system

$$
\begin{aligned}
\int_a^b 1\,dx = b - a &= w_0 + w_1 + w_2 \\
\int_a^b x\,dx = \frac{b^2}{2} - \frac{a^2}{2} &= w_0 a + w_1 \left(\frac{a+b}{2} \right) + w_2 b \\
\int_a^b x^2\,dx = \frac{b^3}{3} - \frac{a^3}{3} &= w_0 a^2 + w_1 \left(\frac{a+b}{2} \right)^2 + w_2 b^2
\end{aligned}
$$

In matrix format:

$$
\begin{bmatrix} 1 & a & a^2 \\ 1 & (a+b)/2 & ((a+b)/2)^2 \\ 1 & b & b^2 \end{bmatrix}
\begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix}
=
\begin{bmatrix} b-a \\ b^2/2 - a^2/2 \\ b^3/3 - a^3/3 \end{bmatrix}
$$

using sympy

```
from sympy import var, Function
A, B, WA, WM, WB = var('A, B, WA, WM, WB')
RULE = lambda f: WA*f(A) + WM*f((A+B)/2) + WB*f(B)
F = Function('F')
print('making the rule', RULE(F))
# the basic functions are 1, x, and x**2
B0 = lambda x: 1
B1 = lambda x: x
B2 = lambda x: x**2
# require that B0, B1, B2 are integrated exactly
from sympy import integrate
X = var('X')
E0 = RULE(B0) - integrate(B0(X), (X, A, B))
E1 = RULE(B1) - integrate(B1(X), (X, A, B))
E2 = RULE(B2) - integrate(B2(X), (X, A, B))
```

script `makerule.py` continued

```
print('solving 3 equations :')
print(E0, '== 0')
print(E1, '== 0')
print(E2, '== 0')
# the equations are easy to solve:
from sympy import solve, Subs, factor
R = solve((E0, E1, E2), (WA, WM, WB))
print(R)
```

the output of the script:

```
solving 3 equations :
A - B + WA + WB + WM == 0
A**2/2 + A*WA - B**2/2 + B*WB + WM*(A/2 + B/2) == 0
A**3/3 + A**2*WA - B**3/3 + B**2*WB \
    + WM*(A/2 + B/2)**2 == 0
{WA: -A/6 + B/6, WM: -2*A/3 + 2*B/3, WB: -A/6 + B/6}
```

testing the rule

```
V = RULE(F)
S = Subs(V, (WA, WM, WB), (R[WA], R[WM], R[WB])).doit()
FORMULA = factor(S)
print 'Simpson formula :', FORMULA

from sympy import lambdify
SIMPSON = lambdify((F, A, B), FORMULA)

# verifying if every quadric is integrated exactly
C0, C1, C2 = var('C0, C1, C2')
QUADRIC = lambda x: C0 + C1*x + C2*x**2
from sympy import simplify
APPROX = simplify(SIMPSON(QUADRIC, A, B))
print('approximation :', APPROX)
EXACT = integrate(QUADRIC(X), (X, A, B))
print('  exact value :', EXACT)
print('    the error :', APPROX - EXACT)
```

the Simpson rule

end of output of the script `makerule.py`:

```
Simpson formula : -(A - B)*(F(A) + F(B) \
    + 4*F(A/2 + B/2))/6
approximation : -0.333333333333333*A**3*C2 - 0.5*A**2*C1 \
    - 1.0*A*C0 + 0.333333333333333*B**3*C2 + 0.5*B**2*C1 \
    + 1.0*B*C0
  exact value : -A**3*C2/3 - A**2*C1/2 - A*C0 \
    + B**3*C2/3 + B**2*C1/2 + B*C0
    the error : 0
```

We derived the Simpson rule:

$$\int_a^b f(x)dx \approx \frac{b-a}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right].$$

sympy, scipy, and integration

1. Functions using Functions
   - functions as arguments of other functions
   - the one-line if-else statement
   - functions returning multiple values
   - the composite trapezium rule
2. Constructing Integration Rules with `sympy`
   - about sympy
   - algebraic degree of accuracy
3. Optional and Keyword Arguments
   - default values
   - numerical integration in scipy
   - optional and keyword arguments

default values

Examples of default values are

- output & tests for developing & debugging
- numerical tolerances and parameters

```
>>> def diff(f,x,h=1.0e-6):
...     return (f(x+h)-f(x))/h
...
>>> from math import sin
>>> diff(sin,0)
0.9999999999998334
>>> diff(sin,0,1.0e-4)
0.9999999983333334
>>> diff(sin,0,1.0e-10)
1.0
```

sympy, scipy, and integration

1. Functions using Functions
   - functions as arguments of other functions
   - the one-line if-else statement
   - functions returning multiple values
   - the composite trapezium rule
2. Constructing Integration Rules with `sympy`
   - about sympy
   - algebraic degree of accuracy
3. Optional and Keyword Arguments
   - default values
   - numerical integration in scipy
   - optional and keyword arguments

numerical integration in scipy

The module `scipy.integrate.quadpack`
exports a general purpose technique from the library QUADPACK.

R. Piessens, E. de Doncker-Kapenga, C. W. Überhuber,
and D. Kahaner:
QUADPACK: A subroutine package for automatic integration.
Springer-Verlag, 1983.

QUADPACK is a Fortran library.

License: public domain.

scipy.integrate.romberg

```
>>> from scipy.integrate import romberg
>>> from scipy import exp, sin
>>> f = lambda x: exp(-x**2)*sin(x)
>>> r = romberg(f,0,1,show=True)
Romberg integration of <function vfunc at 0x1016f7cf8> from [0, 1]

 Steps  StepSize   Results
     1  1.000000  0.154780
     2  0.500000  0.264078  0.300511
     4  0.250000  0.287239  0.294960  0.294590
     8  0.125000  0.292845  0.294714  0.294697  0.294699
    16  0.062500  0.294236  0.294699  0.294698  0.294698  0.294698
    32  0.031250  0.294583  0.294698  0.294698  0.294698  0.294698  0.294698

The final result is 0.29469818225 after 33 function evaluations.
>>> r
0.2946981822496237
```

scipy.integrate.simps

```
>>> from scipy.integrate import simps
>>> from scipy import exp, sin
>>> f = lambda x: exp(-x**2)*sin(x)
>>> from scipy import linspace
>>> x = linspace(0,1,30); y = f(x)
>>> simps(y,x)
0.29469711257835451
>>> x = linspace(0,1,60); y = f(x)
>>> simps(y,x)
0.29469806974810181
>>> x = linspace(0,1,120); y = f(x)
>>> simps(y,x)
0.2946981693991728 7
>>> x = linspace(0,1,240); y = f(x)
>>> simps(y,x)
0.29469818071537968
```

sympy, scipy, and integration

1. Functions using Functions
   - functions as arguments of other functions
   - the one-line if-else statement
   - functions returning multiple values
   - the composite trapezium rule
2. Constructing Integration Rules with `sympy`
   - about sympy
   - algebraic degree of accuracy
3. Optional and Keyword Arguments
   - default values
   - numerical integration in scipy
   - optional and keyword arguments

arguments of variable length

Consider the area computation of a square or rectangle.

The dimensions of a rectangle are length and width,
but for a square we only need the length.

$\rightarrow$ functions whose number of arguments is variable.

The arguments which may or may not appear
when the function is called are collected in a tuple.

Python syntax:

```
def < name > ( < args > , * < tuple > ) :
```

The name of the `tuple` must

- appear after all other arguments `args`,
- and be preceded by `*`.

area of square or rectangle

```
def area ( length , *width ):
    """
    returns area of rectangle
    """
    if len(width) == 0:              # square
        return length**2
    else:                          # rectangle
        return length*width[0]
```

Observe the different meanings of $*$!

```
print('area of square or rectangle')
L = int(input('give length : '))
W = int(input('give width : '))
if W == 0:
    A = area(L)
else:
    A = area(L, W)
print('the area is', A)
```

using keywords

If arguments are optional, then we may identify the extra arguments of a function with keywords.

Instead of `A = area(L, W)`
we require `A = area(L, width=W)`.

Python syntax:

```
def < f > ( < a > , * < t > , ** < dict > ) :
```

The name of the dictionary `dict` must

- appear at the very end of the arguments,
- and be preceded by `**`.

optional arguments

```
def area ( length , **width ):
    """
    returns area of rectangle
    """
    if len(width) == 0:              # square
        return length**2
    else:                          # rectangle
        result = length
        for each in width:
            result *= width[each]
        return result
```

observe the access to the dictionary ...

```
# input of L and W omitted
if W == 0:
    A = area(L)
else:
    A = area(L, width=W)
print('the area is', A)
```

Calling `area(L, W)` no longer possible.

checking the formal parameter name

```
def area ( length , **width ):
    """
    returns area of rectangle
    """
    if len(width) == 0:              # square
        return length**2
    else:                          # rectangle
        result = length
        if 'width' in width:
            result *= width['width']
            return result
        else:
            print('Please provide width.')
            return -1
```

the main program

```
print('area of square or rectangle')
L = int(input('give length : '))
W = int(input('give width (0 if square) : '))
if W == 0:
    A = area(L)
else:
    A = area(L, width=W)
print('the area is', A)
if W != 0:
    print('the next call fails...')
    A = area(L, h=W)
    print('the returned value is', A)
```

Summary + Additional Exercises

`SciPy` provides a Python interface to `QUADPACK`, public domain software for numerical quadrature.

Additional Exercises:

3. Let $H(x) = 0$ for $x < 0$ and $H(x) = 1$ for $x \geq 0$.
   Use `lambda` to define `H`.
4. Write your own Python code for the composite Simpson rule.
   The input parameters are `f`, `a`, `b`, and `n`, the number of function evaluations.
   Compare with `scipy.integrate.simps`.
5. To integrate $f(x) = e^{-x^2} \sin(x)$ over $[0, 1]$, `romberg` is clearly superior over `simps`.
   Do `help(quad)` after `from scipy.integrate import quad` and compare the performance of this general-purpose function to `romberg` and `simps`.

more exercises

6. Extend your Python function for the composite Simpson rule with an estimate for the error.
7. Adjust the `sympy` script to compute an integration rule that integrates all cubics exactly.
8. Develop a function to compute the volume of a cube, or general parallelepiped. For a cube, only one parameter will be given, otherwise, the user must specify length, width, and height of the parallelepiped.