EX.5.2.4.a, Sauer3
Apply the composite Simpson's Rule with $m = 1$, 2 and 4 panels to the following integral. Compute the error by comparing with the exact value from calculus.

$$\text{(a)} \qquad \int_0^1 xe^x \, dx$$

EX.5.2.4.a, Sauer3, solution, Langou

Colab: `https://colab.research.google.com/drive/190libB0ObBkh-uVgzJS-fZAHpD_-EFs0`

Compute the value of the integral using calculus.

Integration by parts gives:

$$\int_0^1 xe^x \, dx = [xe^x]_0^1 - \int_0^1 e^x \, dx = [xe^x - e^x]_0^1 = 1.$$

We get

$$\int_0^1 xe^x \, dx = 1.$$

Apply the composite Simpson's Rule with $m = 1$, 2 and 4 panels to the following integral.

- **one panel,** $m = 1$:
  $h = \frac{1}{2}$: $x_0 = 0$, $x_1 = \frac{1}{2}$, and $x_2 = 1$,

  $$\frac{h}{3}(y_0 + 4y_1 + y_2) = \frac{h}{3}(f(x_0) + 4f(x_1) + f(x_2)).$$

  In Python,

```
integral1 = (1/2)/3*( f(0) + 4*f(1/2) + f(1) )
```

```
1.0026207283
```

- **two panels,** $m = 2$:
  $h = \frac{1}{4}$: $x_0 = 0$, $x_1 = \frac{1}{4}$, $x_2 = \frac{1}{2}$, $x_3 = \frac{3}{4}$, and $x_4 = 1$,

  $$\frac{h}{3}(y_0 + 4y_1 + 2y_2 + 4y_3 + y_4) = \frac{h}{3}(f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + f(x_4)).$$

  In Python,

```
integral2 = (1/4)/3*( f(0) + 4*f(1/4) + 2*f(1/2) + 4*f(3/4) + f(1) )
```

```
1.0001690471
```

- **four panels,** $m = 4$:
  $h = \frac{1}{8}$: $x_0 = 0$, $x_1 = \frac{1}{8}$, $x_2 = \frac{1}{4}$, $x_3 = \frac{3}{8}$, $x_4 = \frac{1}{2}$, $x_5 = \frac{5}{8}$, $x_6 = \frac{3}{4}$, $x_7 = \frac{7}{8}$, and $x_8 = 1$,

  $$\frac{h}{3}(y_0 + 4y_1 + 2y_2 + 4y_3 + 2y_4 + 4y_5 + 2y_6 + 4y_7 + y_8)$$

  $$\frac{h}{3}(f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) + 4f(x_5) + 2f(x_6) + 4f(x_7) + f(x_8))$$

  In Python,

```
integral3 = (1/8)/3*( f(0) + 4*f(1/8) + 2*f(1/4) + 4*f(3/8)
    + 2*f(1/2) + 4*f(5/8) + 2*f(3/4) + 4*f(7/8) + f(1) )
```

```
1.0000106501
```

- We see that, as the number of panels, m, increases, the approximations composite Simpson's Rule approximate the integral better and better. (We get closer to 1, the true value.)

Compute the error by comparing with the exact value from calculus.

Since we know the exact value of our integral (it is 1), we can compute the "true" errors associated with the three different h. (Note: In general, we will not know the value of the integral we are trying to approximate – since we are trying to approximate it – so, in general we cannot exactly compute the errors of the approximations.)
We get:
$h = \frac{1}{2}$ true error$= 2.6 \cdot 10^{-3}$
$h = \frac{1}{4}$ true error$= 1.7 \cdot 10^{-4}$
$h = \frac{1}{8}$ true error$= 1.1 \cdot 10^{-5}$
We see that, as the number of panels, m, increases, the "true" error of the approximation composite Simpson's Rule goes to zero.

(optional) derive an error bound and compare the "true error" with your "error bound"

We can also compare with the error term given in Equation (5.26):

$$\frac{(b-a)h^4}{180} f^{(iv)}(c).$$

Here $a = 0$, $b = 1$, h varies, it is $\frac{1}{2}$, $\frac{1}{4}$, or $\frac{1}{8}$, and c is a point in (a, b). We have

$$f^{(iv)}(x) = 4e^x + xe^x,$$

we note that $f^{(iv)}$ is a continuous positive increasing function, and so

$$\sup_{x \in (0,1)} \{|f^{(iv)}(x)|\} = f^{(iv)}(1) = 5e.$$

(The supremum is attained on the right side.) A formula to bound the error in term of h is therefore

$$\frac{5e}{180} h^4.$$

Let us check all this:
$h = \frac{1}{2}$ true error$= 2.6 \cdot 10^{-3}$ bound on the error $= 4.7 \cdot 10^{-3}$
$h = \frac{1}{4}$ true error$= 1.7 \cdot 10^{-4}$ bound on the error $= 2.9 \cdot 10^{-4}$
$h = \frac{1}{8}$ true error$= 1.1 \cdot 10^{-5}$ bound on the error $= 1.8 \cdot 10^{-5}$
(1) We see that the true errors are always less than their associated error bounds. (They better be!)
(2) We see that the error bounds are *descriptive* (or we can also say *sharp*), i.e. they are "pretty close" from the errors they bound.

python helper code

The complete python code to compute the quantities is:

```
import numpy as np
import scipy.integrate

f = lambda x : x * np.exp( x )
a = 0.
b = 1.
```

```
# method 0: calculus
# computing the exact value by hand, an antiderivative of x * exp(x) is
# x * exp(x) - exp(x) so we get

integral0 = 1.

print("integral0 =", f"{integral0:20.16f}")

# method 1:
# using scipy.integrate.quadrature

integral1 = scipy.integrate.quadrature( f, a, b)[0]

print("integral1 =", f"{integral1:20.16f}", "   error =", f"{abs(integral1-integ

# method 2:
# composite Simpson's Rule with m = 1, 2 and 4 panels

integral2 = (1/2)/3*( f(0) + 4*f(1/2) + f(1) )
integral3 = (1/4)/3*( f(0) + 4*f(1/4) + 2*f(1/2) + 4*f(3/4) + f(1) )
integral4 = (1/8)/3*( f(0) + 4*f(1/8) + 2*f(1/4) + 4*f(3/8)
    + 2*f(1/2) + 4*f(5/8) + 2*f(3/4) + 4*f(7/8) + f(1) )

h=1./2.; print("integral2 =", f"{integral2:20.16f}",
      "   error =", f"{abs(integral2-integral1):6.1e}",
      "   error bound=", f"{5.*exp(1.)/180.*h**4:6.1e}" )
h=1./4.; print("integral3 =", f"{integral3:20.16f}",
      "   error =", f"{abs(integral3-integral1):6.1e}",
      "   error bound=", f"{5.*exp(1.)/180.*h**4:6.1e}" )
h=1./8.; print("integral4 =", f"{integral4:20.16f}",
      "   error =", f"{abs(integral4-integral1):6.1e}",
      "   error bound=", f"{5.*exp(1.)/180.*h**4:6.1e}" )
```

The line `f = lambda x : x*exp(x)` creates the function `f`. The line `integral0= 1.` declares and initializes the variable `integral0` to `1.`. This is the "true" value of our integral that we try to approximate. Then `integral1` is computed using `scipy.integrate.quadrature`. Then `integral2`, `integral3`, and `integral4` represent the composite Simpson's Rule with $m = 1$, 2 and 4, respectively.