**EX.5.2.6.a, Sauer3**

Apply the composite Midpoint Rule with $m = 1$, 2 and 4 panels to approximate the following integral.

$$\text{(a)} \qquad \int_0^{\frac{\pi}{2}} \frac{1-\cos x}{x^2}\, dx$$

**Some comments:**

- We note that in zero the function
$$f(x) = \frac{1-\cos x}{x^2}$$
is not defined. This is why we need an open rule (called in the book: "open Newton-Cotes method" or "midpoint rule"). A closed rule would use $f(0)$ and that would be a problem.

- To repeat, for this integral, Simpson's rule would utterly fail becasue $f(0)$ is not defined, so we need to use a midpoint rule. (To avoid 0.)

- We also note that this is not that bad since
$$\lim_{x\to 0} f(x) = \lim_{x\to 0} \frac{1-\cos x}{x^2} = \frac{1}{2}.$$
So we could very easily (and legitimally) extend $f$ by continuity at 0, by defining $f(0) = 1/2$, and then we could use a closed rule. (Be careful when $x$ gets close to 0 though. But this issue will be for "open" and "closed" rules.)

- We note that there is no closed-form formula for this integral and so the only way to obtain a value for this integral is by numerical integration.



**EX.5.2.6.a, Sauer3, solution, Langou**

Colab: `https://colab.research.google.com/drive/1fTO6pJti-7Ie_69CSiZX1ajpu2q64PWX`

- **declare and initialize the function** `f`:

```
import numpy as np
f = lambda x : ( 1 - np.cos(x) ) / ( x ** 2 )
```

- **getting a reference "trusted" value using** `scipy`:

```
import scipy.integrate
integral1 = scipy.integrate.quadrature( f, 0., 1.)[0]
```

```
integral1 =   0.4863853762296211
```

- **one panel,** $m = 1$:
$h = 1$: $w = \frac{1}{2}$,
$$hf(w)$$
In Python,

```
integral2 = 1*f(1/2)
```

```
integral2 =   0.4896697524385090    error = 3.28e-03
```

- **two panels,** $m = 2$:
  $h = \frac{1}{2}$: $w_1 = \frac{1}{4}$, $w_2 = \frac{3}{4}$,

  $$hf(w_1) + hf(w_2)$$

  In Python,

```
integral3 = 1/2*( f(1/4) + f(3/4) )
```

```
integral3 =   0.4871994095381125    error = 8.14e-04
```

- **four panels,** $m = 4$:
  $h = \frac{1}{4}$: $w_1 = \frac{1}{8}$, $w_2 = \frac{3}{8}$, $w_3 = \frac{5}{8}$, $w_4 = \frac{7}{8}$,

  $$h(f(w_1) + f(w_2) + f(w_3) + f(w_4))$$

  In Python,

```
integral4 = 1/4*( f(1/8) + f(3/8)+ f(5/8) + f(7/8) )
```

```
integral4 =   0.4865884490383869    error = 2.03e-04
```

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

(optional) derive an error bound and compare the "true error" with your "error bound". The "true error" is computed using the trusted value given by `scipy`.

We can also compare with the error term given in Equation (5.27):

$$\frac{(b-a)h^2}{24} f^{('')}(c).$$

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

$$f"(x) = \frac{x^2 \cos(x) - 4x \sin(x) - 6 \cos(x) + 6}{x^4}$$

This function, $f"(x)$, is complicated to analyze. First we extend the function, $f"(x)$, at 0 by continuity. And then, we can show that this (extended) function is bounded, for example, as follows:

$$\text{for all } x, \quad -0.084 \leq f"(x) \leq 0.051.$$

So that

$$\text{for all } x, \quad |f"(x)| \leq 0.084.$$

Using this bound, a formula to bound the error in term of $h$ is therefore

$$0.0036h^2.$$

Let us check all this:

```
integral1 =   0.4863853762296211
integral2 =   0.4896697524385090    error = 3.3e-03    error bound= 3.6e-03
integral3 =   0.4871994095381125    error = 8.1e-04    error bound= 9.0e-04
integral4 =   0.4865884490383869    error = 2.0e-04    error bound= 2.2e-04
```

(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:

```python
import numpy as np
import scipy.integrate

f = lambda x : ( 1 - np.cos(x) ) / ( x ** 2 )
a = 0.
b = 1.

# method 0: calculus
# Well, there is no elementary functions to integrate this integral,
# so calculus is not helpful here and so we need to numerical methods
# to get an approximation. Go Numerical Analysis!

# method 1:
# using scipy.integrate.quadrature

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

print("integral1 =", f"{integral1:20.16f}")

# method 2:
# composite Midpoint Rule with m= 1, 2 and 4 panels

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

h=1.; print("integral2 =", f"{integral2:20.16f}",
      "   error =", f"{abs(integral2-integral1):6.1e}",
      "   error bound=", f"{0.0036 * h**2:6.1e}" )
h=1./2.; print("integral3 =", f"{integral3:20.16f}",
      "   error =", f"{abs(integral3-integral1):6.1e}",
      "   error bound=", f"{0.0036 * h**2:6.1e}" )
h=1./4.; print("integral4 =", f"{integral4:20.16f}",
      "   error =", f"{abs(integral4-integral1):6.1e}",
      "   error bound=", f"{0.0036 * h**2:6.1e}" )
```

(optional) For this integral, we can also use Simpson's Rule with m = 1, 2 and 4 panels, by extending by $f$ by continuity at 0.

```python
# method 3:
# we use composite Simpson's Rule with m = 1, 2 and 4 panels
# but we extend f by continuity with f(0) = 1/2.
```

```python
# So instead of calling f(0), we use f0 which is a variable that is 1/2.

f0 = 0.5
integral2 = (1/2)/3*( f0 + 4*f(1/2) + f(1) )
integral3 = (1/4)/3*( f0 + 4*f(1/4) + 2*f(1/2) + 4*f(3/4) + f(1) )
integral4 = (1/8)/3*( f0 + 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) )

print("integral2 =", f"{integral2:20.16f}", "   error =", f"{abs(integral2-integr
print("integral3 =", f"{integral3:20.16f}", "   error =", f"{abs(integral3-integr
print("integral4 =", f"{integral4:20.16f}", "   error =", f"{abs(integral4-integr
```

```
integral2 =   0.4863961173143160    error = 1.07e-05
integral3 =   0.4863860396094815    error = 6.63e-07
integral4 =   0.4863854175739799    error = 4.13e-08
```