TMA373 Assignment 2

Edoardo Mangia
edoardom@student.chalmers.se

Bibiana Farinha
bibiana@student.chalmers.se

February 2025

Contents

Task 1: Composite Trapezoidal Rule

In this task, we will write a MATLAB function to calculate the composite trapezoidal rule for a given function f over the interval $[a, b]$, where $a < b$, and with N discretization points. Note that because the grid is uniform, the step size is always the same, $h = h_k$ for $k = 1, 2, \ldots, N$, and it is computed by the difference between b and a divided by N.

In the implementation, we defined the step size h, and computed the trapezoidal rule from the formula that we studied during the classes:

$$\begin{aligned} T(h) &= h\left(\frac{1}{2}f(x_0) + f(x_1) + \cdots + f(x_{N-1}) + \frac{1}{2}f(x_N)\right) \\ &= \frac{h}{2}f(x_0) + h\sum_{i=1}^{N-1} f(x_0 + ih) + \frac{h}{2}f(x_N) \approx \int_a^b f(x)\,dx \end{aligned}$$

Implementation

```
function t = mytrapezoidalrule(f, a, b, N)
    % step size
    h = (b-a)/N;

    % computing trapezoidal rule
    t = h*f(a)/2 + h*f(b)/2;
    t = t + h*sum(f((a + h):h:(b - h)));
end
```

The second part of the task is to test the function *mytrapezoidalrule* and compare it to the known true value of the integral.

For that purpose we have defined the following variables, which will be the arguments to the function we want to test:

$$f(x) = \frac{xe^x}{(x+1)^2}, \quad a = 0, \quad b = 1$$

After computing the trapezoidal rules t_10 and t_100 (with $N = 10$ and $N = 100$ respectively), we created the variable *true_value* and calculated the absolute difference between *true_value* and t_10 and t_100, the errors.

Test

```
% defining parameters
f = @(x) (x .* exp(x)) ./ (x + 1).^2;
a = 0;
b = 1;

% computing trapezoidal rule for N=10 and N=100
t_10 = mytrapezoidalrule(f, a, b, 10)
t_100 = mytrapezoidalrule(f, a, b, 100)

% computing error between true value and t_10 and t_100
true_value = (exp(1) - 2) / 2
error_10 = abs(t_10 - true_value);
error_100 = abs(t_100 - true_value);
fprintf('error 10N = %.8f\n', error_10)
fprintf('error 100N = %.8f\n', error_100)
```

Output

```
t_10 =
   0.358875036660782
t_100 =
   0.359138244098885
true_value =
   0.359140914229523

error 10N = 0.00026588
error 100N = 0.00000267
```

The test output gave very small errors, which indicates that the implementation is accurate and the numerical approximation is close to the true integral value.

As expected, the error when $N = 10$ is significantly larger than when $N = 100$. More discretization points translate into a more accurate approximation: when the step size decreases, the linear segments better conform to the function's shape, leading to a more precise approximation.

Task 2: Composite Midpoint Rule

The second task involves repeating the same process as task 1, but this time for the composite midpoint rule.

Implementation

```
function m = mymidpointrule(f, a, b, N)
    % step size
    h = (b-a)/N;

    % computing midpoint rule
    m = h*sum(f(a + h/2:h:b - h/2));
end
```

In the implementation, we defined once again the same step size h, and computed the midpoint rule from the formula that we studied during the classes:

$$\begin{aligned} M(h) &= h\left(f\left(\frac{x_0+x_1}{2}\right)+f\left(\frac{x_1+x_2}{2}\right)+\cdots+f\left(\frac{x_{N-1}+x_N}{2}\right)\right) \\ &= h\sum_{i=0}^{N-1} f\left(x_i+\frac{h}{2}\right) \approx \int_a^b f(x)\,dx. \end{aligned}$$

Test

```
% defining parameters
f = @(x) (x .* exp(x)) ./ (x + 1).^2;
a = 0;
b = 1;

% computing midpoint rule for N=10 and N=100
m_10 = mymidpointrule(f, a, b, 10)
m_100 = mymidpointrule(f, a, b, 100)

% computing error between true value and t_10 and t_100
true_value = (exp(1) - 2) / 2
error_10 = abs(m_10 - true_value);
error_100 = abs(m_100 - true_value);
fprintf('error 10N = %.8f\n', error_10)
fprintf('error 100N = %.8f\n', error_100)
```

Output

```
m_10 =
   0.359273423670188
t_100 =
   0.359142249251510
true_value =
   0.359140914229523

error 10N = 0.00013251
error 100N = 0.00000134
```

Once again, the test output gave a very small error, which indicates that the implementation is accurate and the numerical approximation is close to the true integral value.

The same observations from task one can be made regarding the influence that N has on the error values.

Task 3: Error of the Composite Trapezoidal Rule

In this task, we want to analyse the error of the composite trapezoidal rule as the number of discretization points N varies. In theory, if N is increased by a factor 2, then the error will converge to $1/2^2$.

In the implementation, we first defined some useful variables, such as the function f, and the interval bounds. Note that f is the following simple function:

$$f(x) = \frac{(2-x)^2}{\exp(x)}$$

Implementation

```
% definiting function and parameters
f = @(x) ((2 - x).^2) ./ exp(x);
a = 0;
b = 1;
true_value = integral(f, a, b)

for n =0:7
    N =2^ n ;
    T = mytrapezoidalrule(f, a, b, N);
    err = abs (T - true_value) ;
    if n ~= 0
        X =[ "N = ", num2str(N),
             "err = " , num2str(err),
             "reduction factor = ", num2str( err / err_prev )];
        disp ( X ) ;
    end
    err_prev = err ; % previous error
end
```

Testing the implementation, we get the results in table 1. As expected, the error reduction factor converges to $\frac{1}{2^2}$ as N increases by the factor of 2, thus verifying the convergence property:

$$\left| T(h) - \int_a^b f(x)\,dx \right| \leq C\frac{1}{N^2}$$

Output

N	**Error**	**Reduction Factor**
2	0.1422	0.25769
4	0.035825	0.25194
8	0.0089738	0.25049
16	0.0022445	0.25012
32	0.0005612	0.25003
64	0.00014031	0.25001
128	3.5077e-05	0.25

Table 1