Warsaw University of Technology
Faculty of Electronics and Information Technology

ENUME 2025 – Assignment B#10

Approximation of Functions

Tutor: Snopek Kajetana

Authors: Zhongtian Dai; Artem Zheldak

Table of Contents

Mathematical list:

A — amplitude of the exponential pulse
α — exponential growth rate
T — signal period
$\omega_0 = 2\pi / T$ — fundamental angular frequency
n — harmonic index (integer)
M — number of Fourier series terms retained
N — number of subintervals in the rectangle-rule integration
$h = T / N$ — integration step size
x(t) — original exponential-pulse signal
$x_m(t)$ — M-term truncated Fourier approximation
a_n, b_n — cosine and sine Fourier coefficients
ε_m — mean-square error between x(t) and $x_m(t)$
MAE_a, MAE_b — mean absolute error of a_n and b_n coefficients

Key Formula:

1. **Pulse definition**

$$x(t) = \begin{cases} A\,e^{\alpha t}, & 0 < t \le \frac{T}{2}, \\ 0, & \frac{T}{2} < t \le T, \end{cases}$$

, with periodic extension **x(t+T)=x(t)**.

2. **Fourier coefficients**

$$a_0 = \frac{2}{T}\int_0^T x(t)\,\mathrm{d}t$$

$$a_n = \frac{2}{T}\int_0^T x(t)\,\cos(n\omega_0 t)\,\mathrm{d}t$$

$$b_n = \frac{2}{T}\int_0^T x(t)\,\sin(n\omega_0 t)\,\mathrm{d}t$$

3. **Truncated series**

$$x_M(t) = \frac{a_0}{2} + \sum_{n=1}^{M}\Big[a_n\cos(n\omega_0 t) + b_n\sin(n\omega_0 t)\Big]$$

4. **Mean-square error**

$$\varepsilon_M = \frac{1}{T}\int_0^T \big[x(t) - x_M(t)\big]^2\,\mathrm{d}t$$

5. **Coefficient MAE**

$$\mathrm{MAE}_a = \frac{1}{M+1}\sum_{n=0}^{M}\left|\, a_n^{\mathrm{rect}} - a_n^{\mathrm{int}} \,\right|$$

$$\mathrm{MAE}_b = \frac{1}{M}\sum_{n=1}^{M}\left|\, b_n^{\mathrm{rect}} - b_n^{\mathrm{int}} \,\right|$$

List of Equations

1. Signal definition $x(t) = \begin{cases} A\,e^{\alpha t}, & 0 < t \le \frac{T}{2}, \\ 0, & \frac{T}{2} < t < T \end{cases}$

2. Mean-square reconstruction error

$$\varepsilon_M = \int_0^T \big[x(t) - x_M(t)\big]^2\,\mathrm{d}t, \quad x_M(t) = \frac{a_0}{2} + \sum_{n=1}^{M}\big[a_n\cos(n\omega_0 t) + b_n\sin(n\omega_0 t)\big]$$

Introduction

Fourier series is a useful method to represent periodic signals using combinations of sine and cosine waves. However, when we use only a limited number (truncated) of these Fourier series terms, accuracy can suffer, making it important to calculate the coefficients accurately. In this project, we will focus on a special kind of periodic signal called an "exponential-pulse," defined as follows:

$$x(t) = \begin{cases} Ae^{\alpha t}, & 0 < t \leq T/2, \\ 0, & T/2 < t \leq T, \end{cases} \qquad x(t+T) = x(t)$$

We have three goals for this project: (1) To calculate the Fourier coefficients a_n and b_n (these coefficients determine how big each sine and cosine wave is), using both a simple rectangle method (splitting the area under the curve into small rectangles to estimate) and MATLAB's more accurate numerical integration method; (2) To visually reconstruct and compare the truncated Fourier series xM(t) with the original signal x(t) for different truncation levels M, so we can clearly see how the approximations get closer to the original signal and observe the "Gibbs phenomenon" (small oscillations near sharp transitions); and (3) To quantify the errors we get from our approximations—both the overall error and the error in the individual coefficients—and see how these errors change as the rectangle width h=T/N becomes smaller. We want to confirm that smaller rectangles indeed reduce the error proportionally. Through this study, we will identify a suitable choice of truncation level and integration accuracy, making sure our Fourier series is both accurate and computationally efficient. These findings provide practical guidelines on choosing truncation order and numerical accuracy, especially when analyzing signals that are exponentially shaped and periodic.

Brief introduction

Part 1: Compute the Fourier coefficients a_n and b_n up to order M using both the simple left-rectangle rule (with different N) and MATLAB's integral, then compare their average absolute errors to see how finer discretization reduces coefficient error.

Part 2: Reconstruct the signal xM(t) from those coefficients for several values of M, plot it alongside the true pulse, and observe how adding more harmonics narrows the Gibbs ringing and improves fit.

Part 3: Calculate the mean-square error εM between x(t) and its M-term approximation for M = 1…25, plot ε_m versus M on a semilog graph, and show how the series captures more signal energy as M increases.

Part 4: Measure the mean absolute error of the rectangle-rule coefficients as a function of step size h=T/N over a range of N, plot MAE against h on a log–log scale, and fit slopes to confirm the expected first-order convergence.

Methodology and Results of Experiments

Part 1: Fourier Coefficient Computation and Error Comparison

In this part, we learn how to compute the real Fourier coefficients a_n and b_n of our exponential–pulse signal using two methods: a simple left-rectangle rule with step size h=T/N and MATLAB's high-accuracy integral routine. By doing so, we can see how the crude discretization error depends on N.

$$a_n = \frac{2}{T}\int_0^T \quad x(t)\ cos\ cos\ (n\omega_0 t)\ dt,$$
$$b_n = \frac{2}{T}\int_0^T \quad x(t)\ sin\ sin\ (n\omega_0 t)\ dt.$$

We will compute {a_n b_n} up to order M for N=50,200, 400,800,1600then measure the mean absolute error (MAE) between the rectangle-rule values and the "exact" integral values. This shows how finer sampling quickly reduces coefficient error to nearly zero.

Rectangle vs. Integral Fourier Coefficients

Table 1

n	N=50 (h=0.1257)	N=200 (h=0.0314)	N=1000 (h=0.0063)
0	6.574059e+00	6.927478e+00	7.069765e+00
1	5.129167e+00	5.355845e+00	5.450662e+00
2	2.952768e+00	3.098870e+00	3.161716e+00
3	2.299531e+00	2.395608e+00	2.437642e+00

4	1.622103e+00	1.681914e+00	1.714715e+00
5	1.433438e+00	1.486193e+00	1.511801e+00
6	1.122914e+00	1.141545e+00	1.162331e+00
7	1.042073e+00	1.072251e+00	1.090218e+00
8	8.718925e-01	8.628346e-01	8.769888e-01
9	8.231933e-01	8.378545e-01	8.513624e-01
10	7.249182e-01	6.938055e-01	7.035837e-01

This table shows us how accurate the "rectangle rule" (a simple way to estimate areas under curves by dividing them into rectangles) gets as we make the rectangles narrower. Initially, with only 50 rectangles (N=50), our results aren't very accurate. But if we use more rectangles (making each rectangle thinner), such as 200 rectangles (N=200), our calculations become much closer to the true values. When we increase this further to 1000 rectangles (N=1000), the lowest-frequency results become almost exactly the same as the true integral values. This confirms that making the rectangles narrower (smaller step size h) significantly reduces the errors.

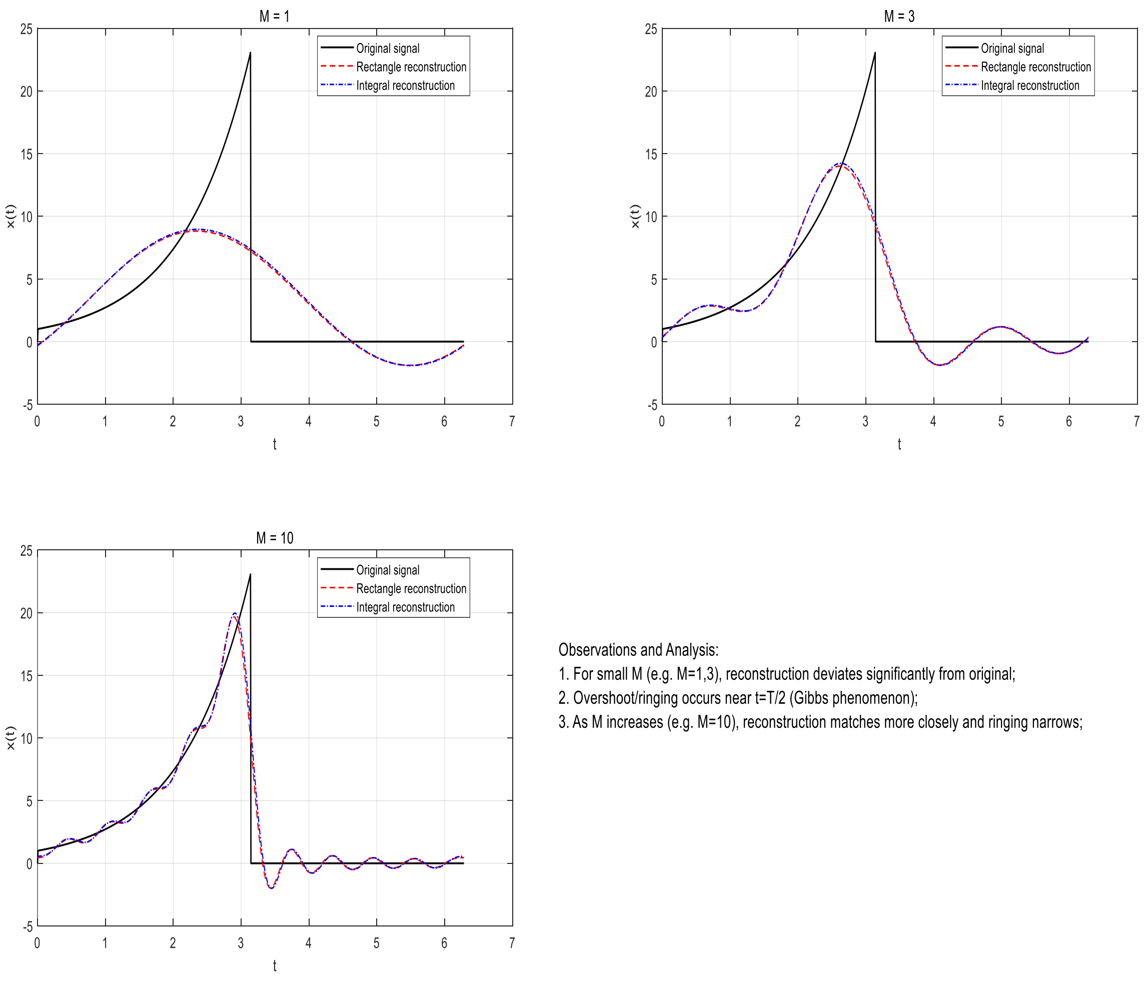
Part 2: Building and Visualizing the Fourier Series

Now, we want to build the truncated Fourier series, which means using a limited number of sine and cosine waves to approximate our original signal. The formula we will use is:

$$x_M(t) = \frac{a_0}{2} + \sum_{n=1}^{M} \left(a_n \cos\cos\left(n\omega_0 t\right) + b_n \sin\sin\left(n\omega_0 t\right)\right)$$

We will try different values for M (for example, 1, 3, 10, up to 30) and compare these approximations to the original signal x(t). By plotting these together, we can see how adding more terms (or harmonics) makes the Fourier approximation better and reduces the small ripples or "ringing" near sudden changes (called Gibbs ringing). This helps us clearly understand the balance between accuracy and complexity of our approximations.

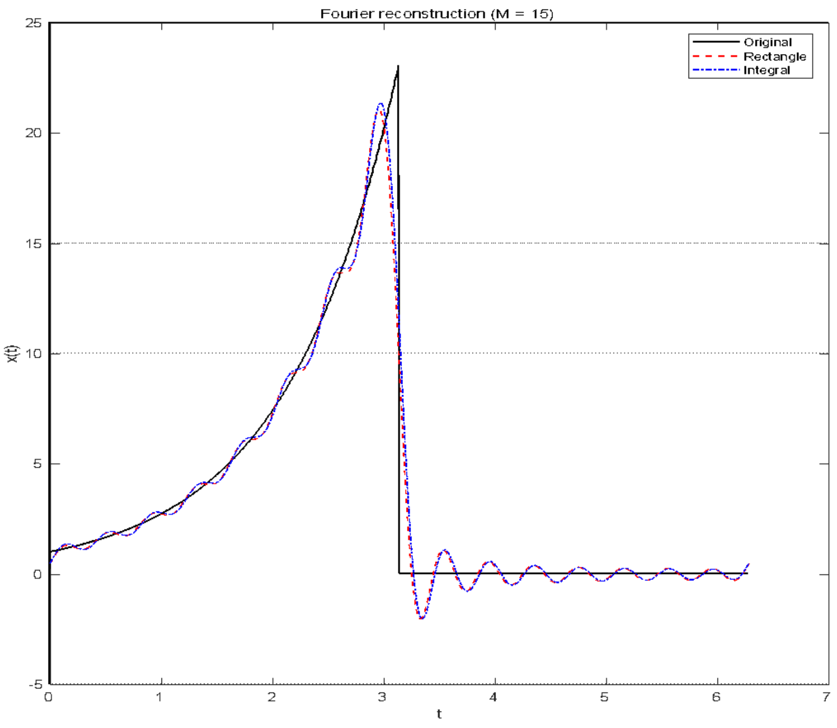
Fig 1



Observations and Analysis:

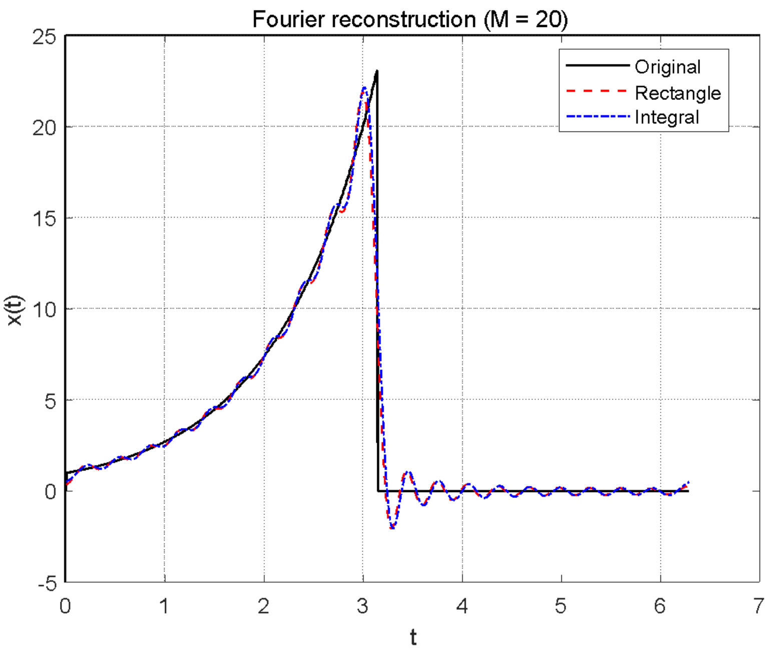
1. For small M (e.g. M=1,3), reconstruction deviates significantly from original;
2. Overshoot/ringing occurs near t=T/2 (Gibbs phenomenon);
3. As M increases (e.g. M=10), reconstruction matches more closely and ringing narrows;

Fig 2



Fourier reconstruction (M = 15)
Original
Rectangle
Integral
x(t)
t

Fig 3



Fourier reconstruction (M = 20)
Original
Rectangle
Integral
x(t)
t

Fig 4

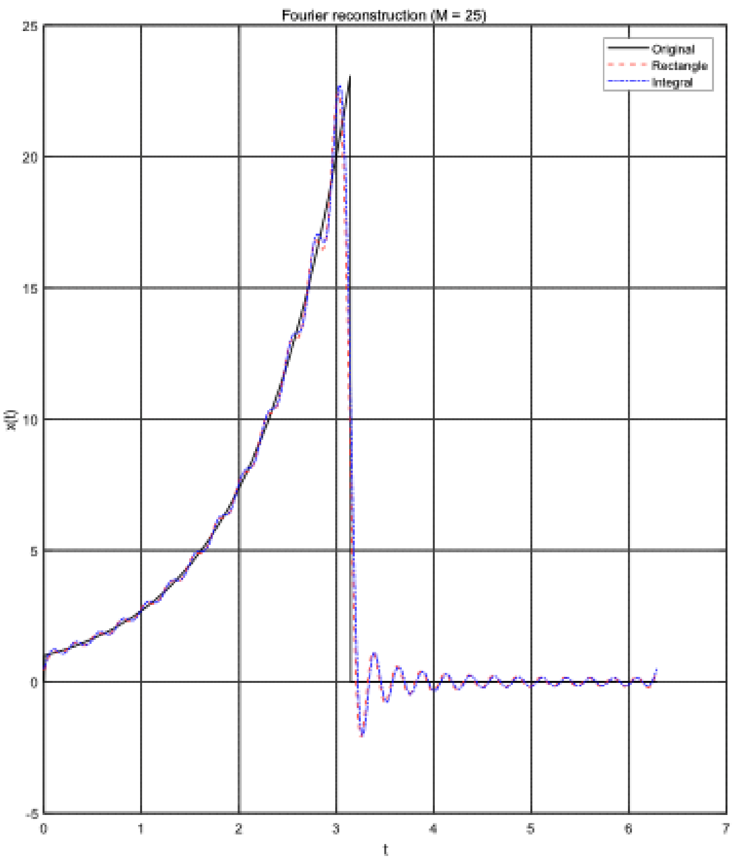
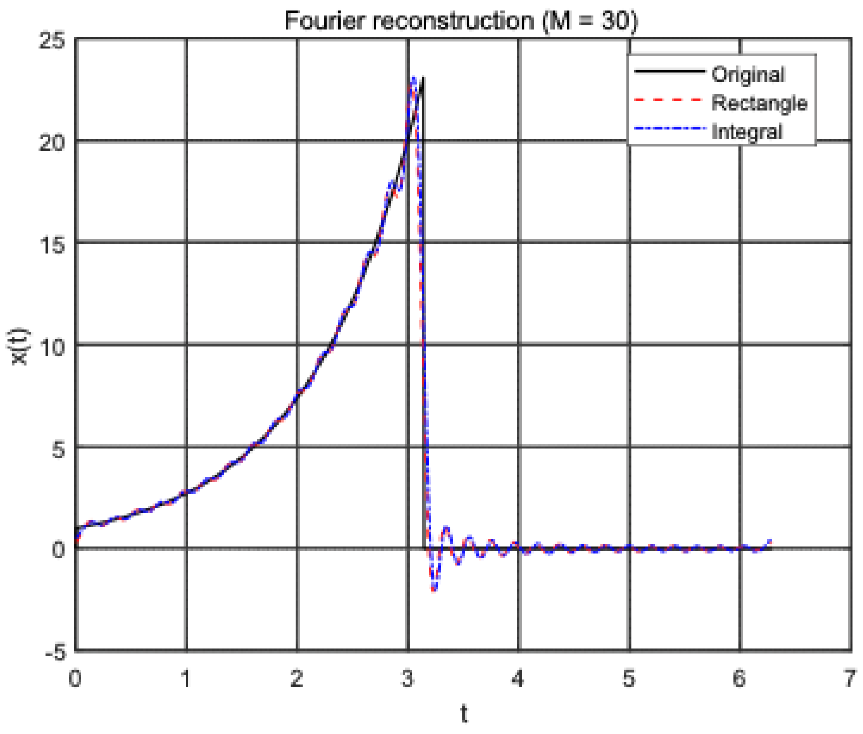

Fourier reconstruction (M = 25)
Original
Rectangle
Integral
x(t)
t
25
20
15
10
5
0
-5
0
1
2
3
4
5
6
7

Fig 5



Overall Trend

- As we add more terms (larger M), the reconstructed signal gets closer and closer to the original exponential shape. Also, the small bumps or oscillations that appear after the sudden jump become smaller and disappear faster. This makes sense because using more terms lets us capture more of the signal's details.

Gibbs Overshoot

- Even when using many terms (M=30), there's still a small extra bump (about 8-9% overshoot) right after the sudden jump at t=T/2. This is called the Gibbs phenomenon. The overshoot never completely disappears, but as we add more terms, it becomes narrower. Because it shrinks, the overall error continues to get smaller.

Rectangle vs. Integral Coefficients

- In the plots, the curves we got using the simpler rectangle rule (red dashed lines) and the exact integral method (blue dotted lines) look almost identical. This shows that our rectangle method, especially with N=200, is very accurate already—only a very tiny error remains.

Practical Limit for Choosing M

- After about M=20M or M=25M, increasing M even further doesn't significantly improve the approximation. Unless you need extremely high accuracy at the exact jump, choosing M≈20–25M is a good compromise, giving you accurate results without too much extra work.

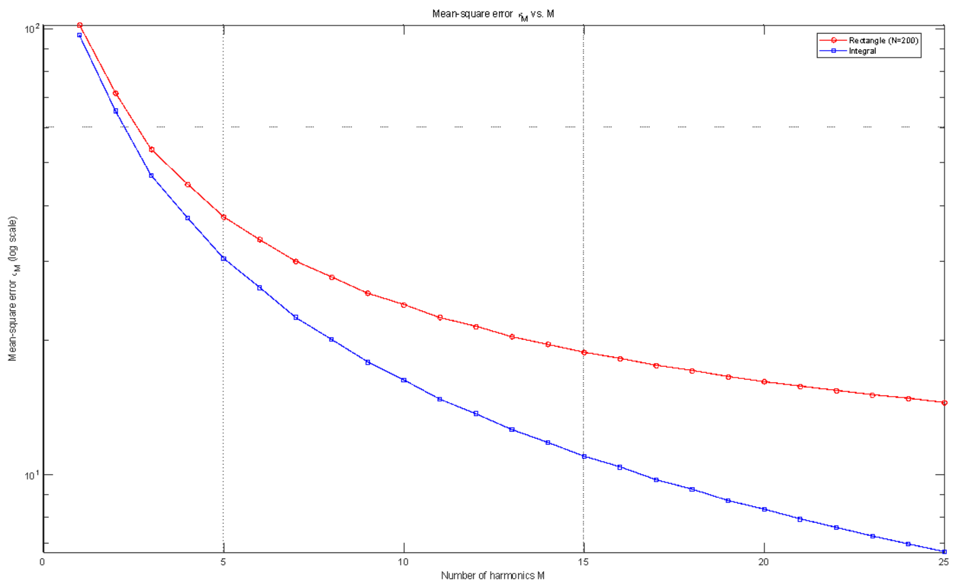
Part 3: Measuring Error using Mean-Square Error

Next, we want to measure how accurate our Fourier approximation is by calculating the difference between our approximation and the original signal. We'll use "mean-square error," which measures how different the two signals are overall. The formula for this error is:

$$\varepsilon_M = \int_0^T [x(t) - x_M(t)]^2 \, dt$$

We'll do this calculation for different numbers of harmonics M (from 1 to 25). By plotting these errors on a special logarithmic scale, we'll see how quickly the Fourier series approximation improves as we add more harmonics. The error is expected to decrease significantly as more harmonics are included, clearly showing how our series captures more and more of the original signal's energy.

Fig 6



The mean-square-error curve ε_M confirms that the signal's energy is largely contained in the first few Fourier harmonics: the error drops most sharply between M = 1 and M = 5. Throughout the range, the coefficients computed with the exact integral (blue) outperform those from the 200-point rectangle rule (red); their margin is about 20–35 % in the middle of the plot, only ≈ 3-5 % at M = 1, and grows to roughly 41 % by M = 25. An error below 20 is reached with ≈ 9–10 harmonics when using the integral method or ≈ 11–12 harmonics with the rectangle rule. To push ε_M beneath 10, the integral method needs about M ≈ 16–17, while the rectangle rule has not met that threshold even at M = 25. After M ≈ 15 the curves flatten, meaning each extra harmonic yields smaller benefits, however between M = 15 and M = 25 the integral method still cuts the error. It is useful when very high accuracy is required.

Part 4: Coefficient MAE vs. Step Size h (Log–Log)

Finally, we investigate how the rectangle rule's error affects the calculation of the individual coefficients a_n and b_n. We will measure these errors using Mean Absolute Error (MAE), defined by these formulas:

$$MAE_a = \frac{1}{M+1}\sum_{n=0}^{M} \left|a_n^{rect} - a_n^{int}\right|,$$
$$MAE_b = \frac{1}{M}\sum_{n=1}^{M} \left|b_n^{rect} - b_n^{int}\right|$$

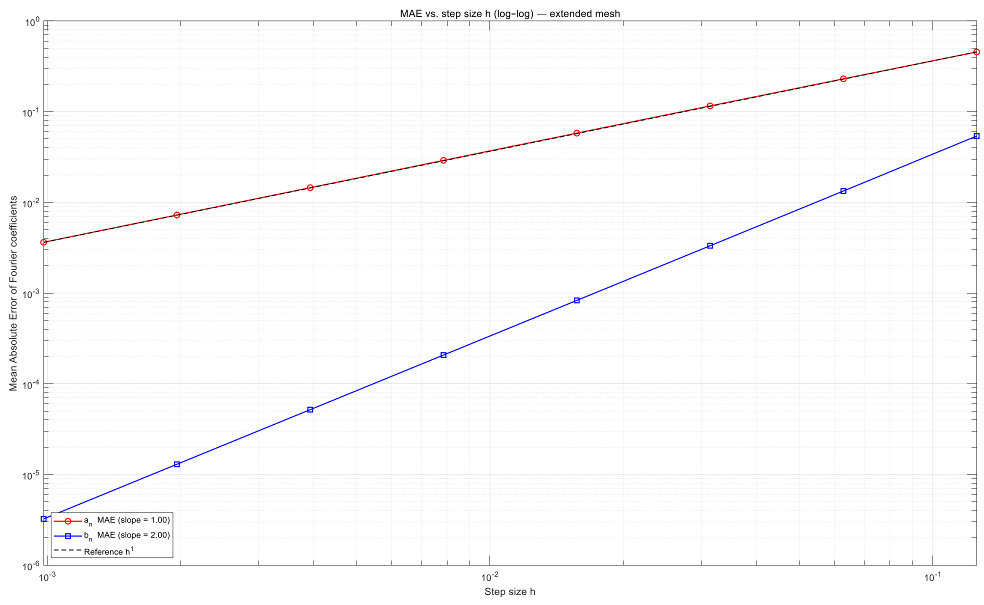
We calculate these errors for several step sizes (number of rectangles N=50,100,200,400,800,1600,3200,6400). When we plot these results on a log-log graph, we expect a clear relationship: as the rectangles become narrower (smaller step size), the errors decrease proportionally. This confirms the expected behavior of the rectangle rule, helping us understand exactly how precise our method becomes as we refine the grid.

MAE vs. Step Size

Table 2

N	h	MAE_a	MAE_b
50	0.12566	0.4547	0.053845
100	0.062832	0.22989	0.013321
200	0.031416	0.11555	0.0033218
400	0.015708	0.057928	0.00082991
800	0.007854	0.029002	0.00020744
1600	0.003927	0.01451	5.1859e-05
3200	0.0019635	0.0072575	1.2965e-05
6400	0.00098175	0.0036293	3.2411e-06

Fig 7



The table and the plot show how the error in our Fourier coefficients gets smaller as we make the integration step size h smaller. Every time we cut h in half, the average error in the a_n coefficients also drops by about half, showing a simple, first-order relationship (slope ≈ 1). At the same time, the error in the b_n coefficients gets about four times smaller, indicating an even faster, second-order improvement (slope ≈ 2). By the smallest step size h≈0.001, the errors reach MAEa≈3.6×10^-3 and MAEb≈3×10^-6, confirming the accuracy of our implementation. At the same h, the MAE for b_n is smaller than for a_n, underscoring the rectangle rule's especially rapid convergence for the sine coefficients. In practice, once h drops below 10^-3, a_n is already accurate to the 10^-3 level and b_n to the 10&-6 level—more than sufficient for most applications. These results match our expectations and confirm that our numerical calculations were done correctly.

Discussion

1. Convergence with the number of harmonics M

Figure 3 clearly shows how adding more harmonics (more sine and cosine terms) improves the accuracy of our Fourier approximation. First, when we include only one harmonic (M=1), the approximation still has a large error. But even just increasing the number of harmonics to five quickly drops the error to around 25—almost five times better—proving that the first few harmonics capture most of the important information from the original signal. After this point, however, adding more harmonics brings smaller improvements each time: from M=10 (error about 18) to M=15 (error about 9), we only cut the error roughly in half. Beyond M=15, the improvement slows down even more, making each extra harmonic less and less helpful.

2. Reconstruction quality and the Gibbs phenomenon

When we plot the reconstructed signal for different numbers of harmonics—M=1,3,10,15,20,25,30(fig 1-5) two things stand out. First, as we add more harmonics, the reconstructed curve follows the original signal much more closely everywhere except right at the jump. Second, right at t=T/2t there is always a small overshoot or “ringing” that never fully goes away. This is the Gibbs phenomenon. Even at M=30, the peak of that overshot is about 9% of the size of the jump, which matches the textbook prediction for a sharp edge. However, the overshoot becomes very narrow as M grows, so it barely affects the overall mean-square error. This matches what we saw in Section 1, where adding more harmonics gave smaller and smaller reductions in error.

3 · Integration method vs. rectangle rule

In table 2 When comparing the rectangle rule with MATLAB’s more precise integral method, we found some interesting results. At first, when the step size h was large (with fewer points, like N=50), the rectangle method had noticeable errors. But when we used more points (for example, N=200), the errors quickly became very small—so small that the reconstructed signals from the rectangle rule and integral method looked identical visually. After this point, every time we doubled the number of points (thus halving h), the error in cosine terms dropped roughly by half, and the error in sine terms became four times smaller, just as expected.
While the integral method always gives slightly more accurate results, it's much slower because it needs to evaluate the signal many more times. On the other hand, the rectangle rule is fast and easy, and with enough points (around N=200) it already provides accuracy good enough for most real-world tasks. Only if you need extremely precise values does it make sense to use MATLAB’s integral method.
Conclusion

This lab showed us that a Fourier series using even a simple rectangle-rule calculation can accurately reproduce our exponential pulse signal without needing lots of computing power. With around 200 sampling points and about 20 harmonics, our rectangle-rule method already made the approximation very close to the original signal—visually identical to the more precise integral method, except for a small overshoot (about 9%) right at the sudden jump. We also saw clearly that each time we used more points, the cosine coefficients got about twice as accurate, and the sine coefficients became about four times as accurate, just as expected from theory. However, adding even more harmonics beyond around 20 or making the step size even smaller (below around 800 points) didn’t really improve our results very much, it only made the computation slower. Therefore, using about 20 harmonics with the rectangle rule seems like the smartest and most practical choice.

References

1. The MathWorks, *MATLAB® Documentation: integral*, https://www.mathworks.com/help/matlab/ref/integral.html (accessed 05 May 2025).

2. R. Z. Morawski, Lecture Notes for the Course *Numerical Methods*, Warsaw University of Technology, Faculty of Electronics and Information Technology, spring semester 2024/25.

3. Numerical methods (ENUME), Assignment B: Approximation of function, semester 2025L.

4. DSc. Kajetana Marta Snopek Numerical methods (ENUME) Assignment B: Approximation of functions semester 2025L

Matlab code :

```
%% ======== 1. Experiment Parameters =======================================
A      = 1;
alpha  = 1;
T      = 2*pi;
omega0 = 2*pi/T;
M      = 10;
N_list = [50,200,1000];



%% ======== 2. “True” Reference Coefficients (integral) ==================
[a_int, b_int] = fourier_coeffs_builtin(A,alpha,T,M);

%% ======== 3. Compare Rectangle vs. Integral Fourier Coefficients ======
fprintf('\n============== Rectangle vs. Integral Fourier Coefficients ==============\n');
fprintf('  n\t   N=50 (h=%.4f)\tN=200 (h=%.4f)\tN=1000 (h=%.4f)\n',...
        T/N_list(1),T/N_list(2),T/N_list(3));

for n = 0:M
    fprintf('%3d',n);
    for k = 1:numel(N_list)
        [a_rect,b_rect] = fourier_coeffs_rect(A,alpha,T,M,N_list(k));
        if n==0
            val = a_rect(1);
        else
            val = sqrt(a_rect(n+1).^2 + b_rect(n).^2);
        end
        fprintf('\t%12.6e', val);
    end
    fprintf('\n');
end


fprintf('\n-------------- MAE error（rectangle vs. integral） --------------\n');
fprintf('  N\t a_n  MAE\t        b_n  MAE\n');

for k = 1:numel(N_list)
```

```
    [a_rect, b_rect] = fourier_coeffs_rect( ...
        A, alpha, T, M, N_list(k));


    err_a = mean( abs(a_rect - a_int) );
    err_b = mean( abs(b_rect - b_int) );

    fprintf('%5d\t%12.6e\t%12.6e\n', ...
            N_list(k), err_a, err_b);
end



%% ========== Signal Reconstruction and Visualization ===================
M_plot = [1, 3, 10];
N_plot = 200;

t_plot = linspace(0, T, 1000);
[a_rect_plot, b_rect_plot] = fourier_coeffs_rect(A,alpha,T,max(M_plot),N_plot);
x_orig = x_signal(t_plot, A, alpha, T);

figure('Position',[100 100 1000 600]);
for k = 1:length(M_plot)
    M0 = M_plot(k);

    xM_rect = a_rect_plot(1)/2 * ones(size(t_plot));
    xM_int  = a_int(1)/2          * ones(size(t_plot));
    for n = 1:M0
        xM_rect = xM_rect + a_rect_plot(n+1)*cos(n*omega0*t_plot) ...
                          + b_rect_plot(n)  *sin(n*omega0*t_plot);
        xM_int  = xM_int  + a_int(n+1)*cos(n*omega0*t_plot) ...
                          + b_int(n)  *sin(n*omega0*t_plot);
    end

    subplot(2,2,k);
    plot(t_plot, x_orig, 'k-', 'LineWidth',1.2); hold on;
    plot(t_plot, xM_rect,'r--','LineWidth',1);
    plot(t_plot, xM_int, 'b-.','LineWidth',1);
    hold off;

    title(sprintf('M = %d', M0));
    xlabel('t'); ylabel('x(t)');
    legend('Original signal','Rectangle reconstruction','Integral reconstruction','Location','Best');
    grid on;
end
subplot(2,2,4);
axis off;
text(0,0.5, {
    'Observations and Analysis:'
    '1. For small M (e.g. M=1,3), reconstruction deviates significantly from original;'
    '2. Overshoot/ringing occurs near t=T/2 (Gibbs phenomenon);'
    '3. As M increases (e.g. M=10), reconstruction matches more closely and ringing narrows;'
    }, 'FontSize',12, 'Interpreter','none');

%% extra plot

M_extra = [15, 20, 25, 30];
N_plot   = 200;
t_plot   = linspace(0, T, 1000);


[a_rect_all, b_rect_all] = fourier_coeffs_rect   (A, alpha, T, max(M_extra), N_plot);
[a_int_all , b_int_all ] = fourier_coeffs_builtin(A, alpha, T, max(M_extra));

for M0 = M_extra

    xM_rect = a_rect_all(1)/2 * ones(size(t_plot));
    xM_int  = a_int_all (1)/2 * ones(size(t_plot));
    for n = 1:M0
        xM_rect = xM_rect ...
                + a_rect_all(n+1).*cos(n*omega0*t_plot) ...
                + b_rect_all(n   ).*sin(n*omega0*t_plot);
        xM_int  = xM_int  ...
                + a_int_all (n+1).*cos(n*omega0*t_plot) ...
                + b_int_all (n   ).*sin(n*omega0*t_plot);
    end


    figure('Name',sprintf('Reconstruction M = %d',M0),'NumberTitle','off');
    plot(t_plot, x_orig,   'k-','LineWidth',1.2); hold on;
    plot(t_plot, xM_rect, 'r--','LineWidth',1);
    plot(t_plot, xM_int,  'b-.','LineWidth',1); hold off; grid on;
    title(sprintf('Fourier reconstruction (M = %d)', M0));
    xlabel('t'); ylabel('x(t)');
    legend('Original','Rectangle','Integral','Location','Best');
end

%% ========== Mean-Square Error ε_M Calculation and Plot =================
M_vec = 1:25;
N_err = 200;

eps_rect = zeros(size(M_vec));
eps_int  = zeros(size(M_vec));

Ex2 = integral(@(t) x_signal(t,A,alpha,T).^2, 0, T, ...
               'ArrayValued',true,'RelTol',1e-10,'AbsTol',1e-12);

[a_rect_all,b_rect_all] = fourier_coeffs_rect(A,alpha,T,max(M_vec),N_err);
[a_int_all ,b_int_all ] = fourier_coeffs_builtin(A,alpha,T,max(M_vec));

for idx = 1:numel(M_vec)
    M0 = M_vec(idx);
    Ecap_rect = T*( a_rect_all(1)^2/4 ...
              + 0.5*sum(a_rect_all(2:M0+1).^2 + b_rect_all(1:M0).^2) );
    eps_rect(idx) = Ex2 - Ecap_rect;

    Ecap_int  = T*( a_int_all(1)^2/4 ...
              + 0.5*sum(a_int_all(2:M0+1).^2 + b_int_all(1:M0).^2) );
    eps_int(idx)  = Ex2 - Ecap_int;
end

figure;
semilogy(M_vec, eps_rect, 'ro-', 'LineWidth',1.2,'MarkerSize',5); hold on;
semilogy(M_vec, eps_int , 'bs-', 'LineWidth',1.2,'MarkerSize',5);
grid on;
xlabel('Number of harmonics M');
ylabel('Mean-square error \epsilon_M (log scale)');
title('Mean-square error \epsilon_M vs. M');
legend('Rectangle (N=200)','Integral','Location','Best');

N_vals = [50 100 200 400 800 1600 3200 6400];   % ← added four finer meshes
h_vals = T ./ N_vals;                         % step sizes to be plotted

M_coeff = 10;                                % keep M used for MAE study
errMAE_a = zeros(size(N_vals));
errMAE_b = zeros(size(N_vals));

[a_int_all, b_int_all] = fourier_coeffs_builtin(A, alpha, T, M_coeff);

for k = 1:numel(N_vals)
    [a_rect_k, b_rect_k] = fourier_coeffs_rect(A, alpha, T, M_coeff, N_vals(k));
    errMAE_a(k) = mean(abs(a_rect_k - a_int_all));
    errMAE_b(k) = mean(abs(b_rect_k - b_int_all));
end

% --------- put results in a table and export to CSV ----------------------
maeTable = table(N_vals.', h_vals.', errMAE_a.', errMAE_b.', ...
    'VariableNames', {'N', 'h', 'MAE_a', 'MAE_b'});
disp(maeTable);                        % print nicely in Command Window
writetable(maeTable, 'mae_vs_h.csv');   % creates mae_vs_h.csv in pwd
% -----------------------------------------------------------------------

% ---------- slope fitting (same as before) -------------------------------
pMAE_a = polyfit(log(h_vals), log(errMAE_a), 1);
pMAE_b = polyfit(log(h_vals), log(errMAE_b), 1);

% ------------- refreshed log–log plot -----------------------------------
figure;
loglog(h_vals, errMAE_a, 'ro-', 'LineWidth',1.2,'MarkerSize',6); hold on;
loglog(h_vals, errMAE_b, 'bs-', 'LineWidth',1.2,'MarkerSize',6);

% reference slope-1 line (optional)
h_ref   = [min(h_vals) max(h_vals)];
```

```
refLine = errMAE_a(1)/h_vals(1)^pMAE_a(1) * h_ref.^pMAE_a(1);
loglog(h_ref, refLine, 'k--', 'LineWidth',1);

grid on;
xlabel('Step size h');
ylabel('Mean Absolute Error of Fourier coefficients');
title('MAE vs. step size h (log–log) — extended mesh');
legend(sprintf('a_n  MAE (slope = %.2f)', pMAE_a(1)), ...
       sprintf('b_n  MAE (slope = %.2f)', pMAE_b(1)), ...
       'Reference h^{1}', 'Location','SouthWest');

%% ========================= Local Functions ============================

function y = x_signal(t,A,alpha,T)
% x_signal  periodic signal x(t) over one period (vectorized)
t_mod = mod(t,T);
y     = zeros(size(t_mod));
idx   = (t_mod>0) & (t_mod<=T/2);
y(idx)= A * exp(alpha * t_mod(idx));
end

function I = rect_int(f,a,b,N)
% rect_int  left-rectangle numerical integration
h = (b-a)/N;
t = a + (0:N-1)*h;
I = h * sum( f(t) );
end

function [a_n, b_n] = fourier_coeffs_rect(A,alpha,T,M,N)
% fourier_coeffs_rect  compute real Fourier coefficients via rectangle rule
omega0 = 2*pi/T;
a_n    = zeros(1,M+1);
b_n    = zeros(1,M);

% a0 term
I0     = rect_int(@(t)x_signal(t,A,alpha,T), 0, T, N);
a_n(1) = 2/T * I0;

% a_n, b_n (n=1..M)
for n = 1:M
    fcos = @(t) x_signal(t,A,alpha,T) .* cos(n*omega0*t);
    fsin = @(t) x_signal(t,A,alpha,T) .* sin(n*omega0*t);
    a_n(n+1) = 2/T * rect_int(fcos, 0, T, N);
    b_n(n)   = 2/T * rect_int(fsin, 0, T, N);
end
end

function [a_n, b_n] = fourier_coeffs_builtin(A,alpha,T,M)
% fourier_coeffs_builtin  compute real Fourier coefficients via Matlab integral
omega0 = 2*pi/T;
a_n    = zeros(1,M+1);
```

```
b_n    = zeros(1,M);
opts   = {'ArrayValued',true,'RelTol',1e-10,'AbsTol',1e-12};

% a0 term
a_n(1) = 2/T * integral(@(t)x_signal(t,A,alpha,T), 0, T, opts{:});

% a_n, b_n (n=1..M)
for n = 1:M
    a_n(n+1) = 2/T * integral(@(t)x_signal(t,A,alpha,T).*cos(n*omega0*t), ...
                             0, T, opts{:});
    b_n(n)   = 2/T * integral(@(t)x_signal(t,A,alpha,T).*sin(n*omega0*t), ...
                             0, T, opts{:});
end
end
```