VIETNAM NATIONAL UNIVERSITY

HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY

FACULTY OF COMPUTER SCIENCE AND ENGINEERING

Computer Architecture Lab (CO2008)

Filtering and Prediction Signal with Wiener Filter

Class: CC06
Group: 07
Lecturer: Nguyen Thien An

Students and ID

Full name	Student ID
Nguyen Luong Quoc Thinh	2453203
Huynh Pham Hong Trang	2453269
Than Ngo Tuan	2453366
Le Nhu Nha Uyen (NT)	2453402

HO CHI MINH CITY, 11/2025

Contents

1 Algorithm

1.1 Introduction

1.1.1 Problem Context

In signal processing, real-world signals are often corrupted by noise during acquisition, transmission, or storage. To recover the original information or obtain a reliable estimate of the desired signal, it is necessary to design an optimal filter that suppresses noise while preserving the essential features of the signal. Among many techniques, the **Wiener filter** provides a statistically optimal solution under the **minimum mean-square error (MMSE)** criterion, estimating an unknown desired signal by minimizing the expected squared error between the estimate and the truth.

Formally, the task is to estimate the desired signal $d(n)$ from a noisy observation $x(n) = s(n) + w(n)$ according to the MMSE criterion. The linear MMSE-optimal filter is the Wiener filter, whose design relies on second-order statistics—specifically, the autocorrelation of the input and the cross-correlation between the input and the desired signal. Exploiting these relationships yields a set of optimal coefficients that produce the best linear estimate of $d(n)$ in the least-squares sense.

Note: *A C++ implementation is used as a reference to validate the MIPS assembly program and generate ground-truth outputs.*

1.1.2 Objective

Implement a **finite impulse response (FIR) Wiener filter** with MIPS assembly to estimate the desired signal $d(n)$ from noisy observation $x(n)$. The algorithm shall (i) estimate autocorrelation and cross–correlation, (ii) construct the Toeplitz matrix R and solve the Wiener–Hopf system $R\,h = \gamma_d$ to obtain the optimal coefficients h_k, (iii) generate the filtered output $y(n)$, and (iv) compute and report the **MMSE** between $d(n)$ and $y(n)$. The program must print the results to the `terminal` and write `output.txt` in the exact required format; optional plots may be included for visual validation.

Scope and Constraints

The implementation is required to:

- Process discrete sequences of length up to **N = 10 signal samples**.
- Read input data from two files: `desired.txt` (desired signal) and `input.txt` (combination of desired signal and noise).
- Produce an output file `output.txt` containing:
  - A line with the **filtered signal values** (rounded to one decimal place).
  - A line with the **MMSE value** (rounded to one decimal places).
- Validate that both input signals are of equal length; otherwise, the program must return the message `"Error: size not match"`.
- Adhere to the **pre-determined variable naming convention** specified in the assignment document.

These constraints ensure consistency across all student submissions and allow for automated evaluation of correctness and precision.

Report Structure

This report is organized into the following main sections:

1. **Methodology and Algorithm** – presents the theoretical basis of the Wiener filter, including the MMSE criterion and the Wiener–Hopf equations and describes the computational sequence from correlation estimation to coefficient solving.
2. **File I/O Contract** – defines the file formats, input validation, and precision requirements.
3. **Core Functions and Pseudocode** – summarizes the role of each role and presents the pseudocode of the algorithm.
4. **Implementation** – details the step-by-step construction of the C++ program, mapping each theoretical element to a corresponding code component.
5. **Team Workload and Explanation -** describe each member's role, the division of tasks (algorithm, implementation, testing, reporting), and how responsibilities were coordinated and verified.
6. **Results and Discussions** – analyzes the program's output, evaluates MMSE results, and interprets filtering performance.

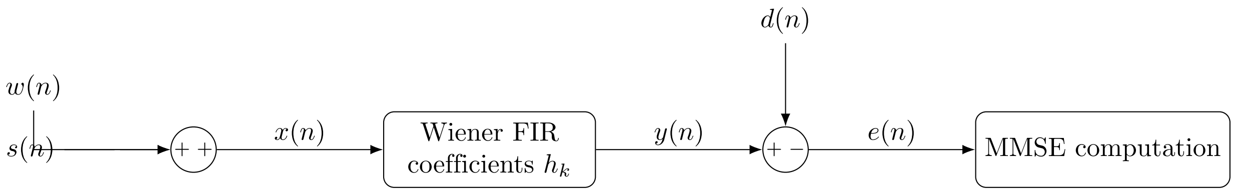


Figure 1: Block diagram of the Wiener filtering chain.

1.1.3 Outcomes of the Assignment

After completing this assignment, we have practiced and strengthened the following skills:

- Using the **MARS MIPS simulator** to assemble, run, and debug programs.
- Implementing **arithmetic and data transfer instructions** for single-precision floating-point operations.
- Using **conditional branches and jumps** to control loops and error paths.
- Organizing the program into **procedures** (e.g., `computeAutocorrelation`, `solveLinearSystem`) that follow a clear calling convention and stack discipline.

1.2 Methodology and Algorithm

1.2.1 Model and Optimality Criterion

For an FIR filter of length M:

$$y(n) = \sum_{k=0}^{M-1} h_k \, x(n-k), \tag{1}$$

with error $e(n) = d(n) - y(n)$ and objective

$$\mathrm{MMSE} = \mathbb{E}\big[\,|e(n)|^2\,\big]\,. \tag{2}$$

1.2.2 Wiener–Hopf Equations

MMSE minimization yields

$$\sum_{k=0}^{M-1} h_k\,\gamma_{xx}(l-k) = \gamma_{dx}(l), \quad l = 0,\ldots,M-1, \tag{3}$$

or in Toeplitz matrix form:

$$R\,h = \gamma_d, \qquad h^\star = R^{-1}\gamma_d. \tag{4}$$

Here γ_{xx} is the autocorrelation of x, and γ_{dx} is the cross-correlation between d and x.

1.2.3 Discrete Sample Estimates

$$\hat{r}_{xx}(k) = \frac{1}{N}\sum_{n=k}^{N-1} x(n)\,x(n-k), \tag{5}$$

$$\hat{\gamma}_{dx}(k) = \frac{1}{N}\sum_{n=k}^{N-1} d(n)\,x(n-k). \tag{6}$$

1.2.4 Empirical MMSE

$$\widehat{\mathrm{MMSE}} = \frac{1}{N}\sum_{n=0}^{N-1}\big(d(n)-y(n)\big)^2\,. \tag{7}$$

1.3 File I/O contract

1.3.1 File Formats

The program uses three text files with the following formats:

- `desired.txt`: contains $N = 10$ real numbers representing the desired signal $d(n)$, with one decimal digit, separated by whitespace.
- `input.txt`: contains $N = 10$ real numbers representing the input (noisy) signal $x(n)$, with the same format as `desired.txt`.
- `output.txt`: contains the filtered output and the MMSE value in the format:

```
Filtered output: y0 y1 ... y9
MMSE: m
```

where all numbers are rounded to one decimal place.

1.3.2 Pre-determined Blocks

Name	Meaning
`desired_signal`	Array storing $d(n)$
`input_signal`	Array storing $x(n)$
`optimize_coefficient`	Vector of h_k (M coefficients)
`output_signal`	Array storing $y(n)$
`mmse`	MMSE value

1.3.3 Table mapping files to MIPS variables

Table 1: Mapping between files and MIPS data segment variables

File / Value	**MIPS label**	**Role**
`desired.txt`	`desired_signal`	Buffer for desired sequence $d(n)$
`input.txt`	`input_signal`	Buffer for input sequence $x(n)$
Filtered output	`output_signal`	Buffer for $y(n)$ after filtering
Filter coefficients	`optimize_coefficient`	Wiener filter FIR coefficients h
MMSE	`mmse`	Mean squared error between d and y

The MIPS program reads and parses the input files manually. It skips whitespace, reads an optional sign, accumulates the integer part, and then reads at most one fractional digit. The value $x.y$ is represented by first forming $10x+y$ as an integer and scaling by 0.1 in single-precision floating point. This guarantees that all numbers are stored with exactly one decimal place.

1.4 Core functions and Pseudocode

1.4.1 Main Steps

1. Read `desired.txt` $\to d(n)$; `input.txt` $\to x(n)$; validate lengths.
2. Compute $\hat{r}_{xx}(k)$ for $k = 0..M-1$.
3. Compute $\hat{\gamma}_{dx}(k)$ for $k = 0..M-1$.
4. Build Toeplitz matrix R from $\hat{r}_{xx}$.
5. Solve $Rh = \gamma_d$ via Gaussian elimination (with partial pivoting).
6. Compute $y(n) = \sum_{k=0}^{M-1} h_k x(n-k)$.
7. Compute MMSE; write `output.txt` and print to terminal.

1.4.2 Pseudocode

```
READ desired.txt -> d[0..N-1]
READ input.txt   -> x[0..N-1]
IF N_d != N_x: PRINT "Error: size not match" and EXIT

for k = 0..M-1:
    rxx[k] = (1/N) * sum_{n=k..N-1} x[n] * x[n-k]
    gdx[k] = (1/N) * sum_{n=k..N-1} d[n] * x[n-k]

R = Toeplitz(rxx[0..M-1])
h = SolveGaussian(R, gdx)  // partial pivoting

for n = 0..N-1:
    y[n] = sum_{k=0..min(n,M-1)} h[k] * x[n-k]

MMSE = (1/N) * sum_{n=0..N-1} (d[n] - y[n])^2
PRINT "Filtered output: ..." (%.1f)
PRINT "MMSE: ..." (%.1f)
WRITE the same to output.txt
```

1.4.3 Mapping to the MIPS Procedures

In the MIPS implementation, the algorithm is decomposed into several reusable procedures. Table 2 summarizes the main ones and their roles.

Table 2: Main MIPS procedures and their roles

Procedure (label)	Description
`computeCrosscorrelation`	Computes $\hat{\gamma}_{dx}(k)$ for $k = 0, \dots, N-1$ from `desired_signal` and `input_signal`
`computeAutocorrelation`	Computes $\hat{r}_{xx}(k)$ from `input_signal`
`createToeplitzMatrix`	Builds the Toeplitz matrix R from $\hat{r}_{xx}(k)$
`solveLinearSystem`	Solves $Rh = \gamma_{dx}$ by Gaussian elimination with partial pivoting
`applyWienerFilter`	Applies h to $x(n)$ to get $y(n)$
`computeMMSE`	Computes the mean squared error between $d(n)$ and $y(n)$

Detailed Explanation of Core Functions

Each procedure in the MIPS implementation corresponds directly to a specific mathematical step in the Wiener filtering algorithm. The following subsections provide concise explanations of each procedure together with compact pseudo-code, illustrating how the core computational steps are realized in practice without showing full MIPS instruction listings.

1) `computeAutocorrelation()` — Computes $\hat{r}_{xx}(k)$

In the implementation, the autocorrelation is computed by looping over each lag k and accumulating the product $x(n)\,x(n-k)$ for all valid samples $n \geq k$. The sum is then normalized by N to obtain the empirical estimate $\hat{r}_{xx}(k)$.

$$\hat{r}_{xx}(k) = \frac{1}{N} \sum_{n=k}^{N-1} x(n)x(n-k).$$

Pseudocode:

```
for k = 0 .. N-1:
    sum = 0
    for n = k .. N-1:
        sum += x[n] * x[n - k]
    r_xx[k] = sum / N
```

Each iteration k computes the correlation of $x(n)$ with a delay of k, normalized by the length of the signal N.

2) `computeCrosscorrelation()` **— Computes $\hat{\gamma}_{dx}(k)$**

Similarly, the cross-correlation between the desired signal $d(n)$ and the input signal $x(n)$ is computed by fixing a lag k and summing the product $d(n)\,x(n-k)$ for all $n \geq k$, then dividing by N.

$$\hat{\gamma}_{dx}(k) = \frac{1}{N}\sum_{n=k}^{N-1} d(n)x(n-k).$$

Pseudocode:

```
for k = 0 .. N-1:
    sum = 0
    for n = k .. N-1:
        sum += d[n] * x[n - k]
    gamma_dx[k] = sum / N
```

This vector $\hat{\gamma}_{dx}$ forms the right-hand side of the Wiener–Hopf equation $Rh = \gamma_d$.

3) `createToeplitzMatrix()` **— Builds matrix R**

The Toeplitz autocorrelation matrix R is generated from $\hat{r}_{xx}(k)$, where each element is defined as $R_{ij} = \hat{r}_{xx}(|i-j|)$

That means the Toeplitz matrix R is then built by filling each entry R_{ij} with the autocorrelation value at lag $|i-j|$, which enforces the symmetric Toeplitz structure.

Pseudocode:

```
for i = 0 .. N-1:
    for j = 0 .. N-1:
        R[i][j] = r_xx[ abs(i - j) ]
```

This symmetric structure reflects how Wiener filters exploit the stationarity of the signal.

4) `solveLinearSystem()` **— Solves $Rh = \gamma_d$**

To obtain the optimal Wiener coefficients h, we solve the linear system $Rh = \gamma_{dx}$ using Gaussian elimination with partial pivoting. The augmented matrix $[R \mid \gamma_{dx}]$ is first transformed into upper triangular form, and then back substitution is applied to recover h.

Pseudocode:

```
# Forward elimination with partial pivoting
for i = 0 .. N-1:
    # Find pivot row with largest |aug[row][i]|
    pivot = argmax_{row = i..N-1} |aug[row][i]|
    swap rows i and pivot

    # Eliminate entries below the pivot
    for k = i+1 .. N-1:
        factor = aug[k][i] / aug[i][i]
        for j = i .. N:      # last column is gamma_dx
            aug[k][j] -= factor * aug[i][j]

# Back substitution
for i = N-1 down to 0:
    sum = aug[i][N]           # right-hand side
    for j = i+1 .. N-1:
        sum -= aug[i][j] * h[j]
    h[i] = sum / aug[i][i]
```

Pivoting ensures numerical stability even when the matrix R is nearly singular.

5) `applyWienerFilter()` — Computes the output $y(n)$

Once the coefficients h_k are known, the Wiener filter output is obtained by convolving h with the input signal $x(n)$. For each time index n, the algorithm accumulates the weighted sum of the current and past input samples:

$$y(n) = \sum_{k=0}^{n} h_k x(n-k).$$

Pseudocode:

```
for n = 0 .. N-1:
    y[n] = 0
    for k = 0 .. n:
        y[n] += h[k] * x[n - k]
```

This is the filtered output signal that approximates the desired signal $d(n)$.

6) `computeMMSE()` — Computes performance metric

Finally, the Mean Squared Error between the desired signal and the filtered output is computed by averaging the squared difference over all samples:

$$\text{MMSE} = \frac{1}{N}\sum_{n=0}^{N-1}(d(n) - y(n))^2.$$

Pseudocode:

```
mmse = 0
for n = 0 .. N-1:
    error = d[n] - y[n]
    mmse += error * error
mmse = mmse / N
```

A lower MMSE value indicates that the Wiener filter's estimation is closer to the original desired signal.

1.5 Implementation in MIPS assembly

1.5.1 Data Segment and Pre-determined Variables

The data segment declares all arrays and scalar variables needed by the Wiener filter. The key labels are:

- `desired_signal`, `input_signal`, `output_signal`: arrays of length $N = 10$ storing $d(n)$, $x(n)$, and $y(n)$ respectively.
- `crosscorr`, `autocorr`: arrays storing $\hat{\gamma}_{dx}(k)$ and $\hat{r}_{xx}(k)$.

- `R`: a flattened 10×10 Toeplitz matrix stored in row-major order.
- `aug`: the augmented matrix for Gaussian elimination, containing $[R \mid \gamma_{dx}]$.
- `optimize_coefficient`: array of filter coefficients $h(k)$.
- `mmse`: scalar holding the final mean squared error.

1.5.2 Program Flow

The `main` procedure coordinates the entire computation. Its control flow can be summarized as follows:

1. Open `input.txt` and parse N samples into `input_signal`.
2. Open `desired.txt` and parse N samples into `desired_signal`.
3. Check that the numbers of parsed samples are equal and that $N > 0$. If not, jump to the `size_mismatch` handler.
4. Call the following procedures in order:
   (a) `computeCrosscorrelation`
   (b) `computeAutocorrelation`
   (c) `createToeplitzMatrix`
   (d) `solveLinearSystem`
   (e) `applyWienerFilter`
   (f) `computeMMSE`
5. Open `output.txt` and write the filtered output and MMSE using `float_to_str` and `round_to_1dec`.
6. Print the same results to the MARS console and terminate the program.

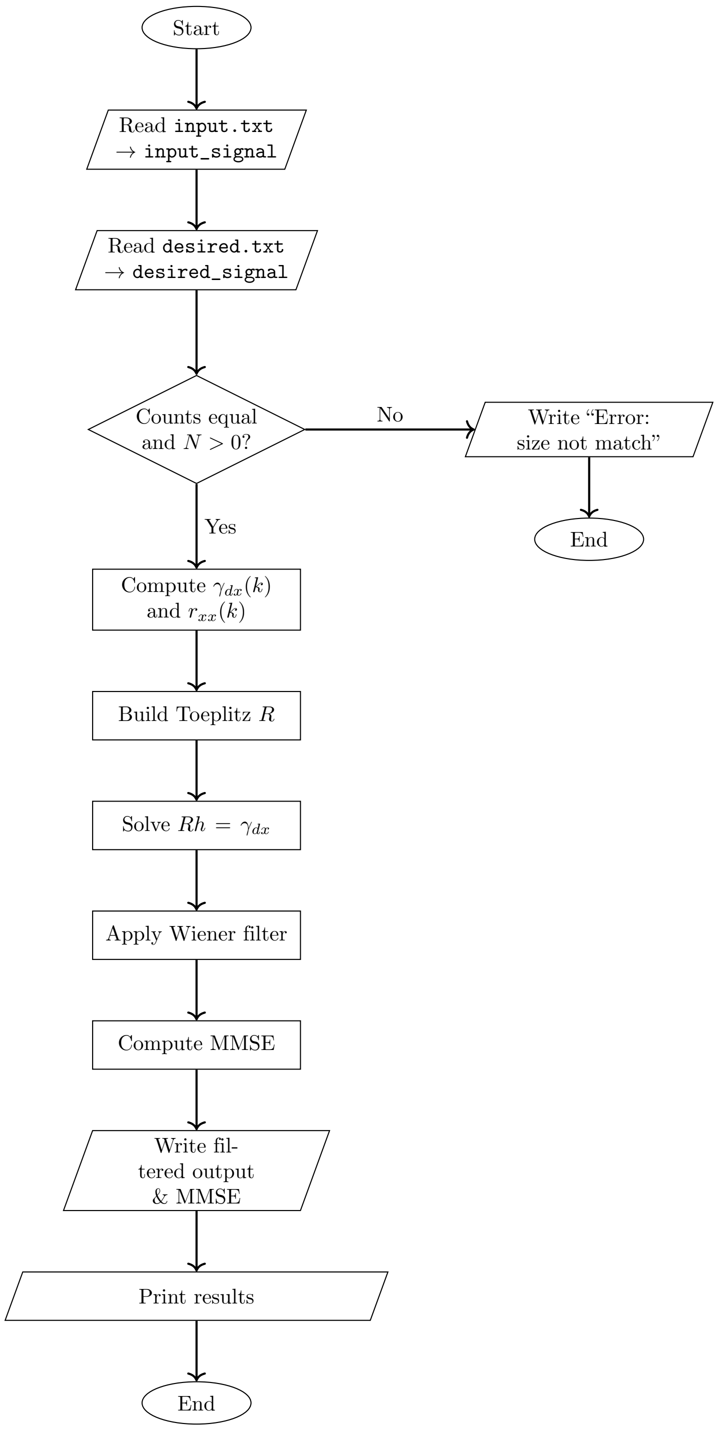

Start
Read input.txt → input_signal
Read desired.txt → desired_signal
Counts equal and $N > 0$?
No
Write "Error: size not match"
End
Yes
Compute $\gamma_{dx}(k)$ and $r_{xx}(k)$
Build Toeplitz R
Solve $Rh = \gamma_{dx}$
Apply Wiener filter
Compute MMSE
Write filtered output & MMSE
Print results
End

Figure 2: Compact program flow of the MIPS Wiener filter implementation.

1.5.3 Floating-point Rounding and String Formatting

To meet the requirement of printing numbers with exactly one decimal place, the procedure `round_to_1dec` performs the following steps:

1. Multiply the value by 10.
2. Add $+0.5$ for non-negative values or -0.5 for negative values.
3. Truncate to an integer.
4. Divide back by 10 to obtain the rounded value.

The `float_to_str` procedure then converts this rounded value to a character string by extracting the sign, integer part, and the single fractional digit, and writing them into an output buffer. This ensures that all printed values have a consistent format such as `0.0`, `1.2`, or `-3.4`.

1.5.4 Error Handling: `size_mismatch`

If the numbers of samples read from `input.txt` and `desired.txt` do not match, or if no valid samples are parsed, the program branches to the label `size_mismatch`. This handler:

- Opens `output.txt` in write mode.
- Writes the line `Error: size not match`.
- Closes the file and terminates the program.

No correlation or filtering is performed in this case, which matches the expected behavior specified in the assignment tests.

1.5.5 Relation to the C++ Reference Implementation

Before writing the MIPS assembly code, we implemented the Wiener filter in C++ to validate the algorithm and generate reference outputs. The MIPS implementation follows the same sequence of operations (correlation, Toeplitz construction, Gaussian elimination, filtering, and MMSE computation), and we used the C++ program to verify that the MIPS outputs are numerically correct up to one decimal place for all provided test cases.

1.6 Team Workload

Table 3: Team workload distribution

Member	Main tasks	Percent
Than Ngo Tuan	autocorrelation, crosscorrelation, Toeplitz matrix construction	25%
Nguyen Luong Quoc Thinh	WienerCoefficient, Gaussian elimination	25%
Huynh Pham Hong Trang	File I/O, parsing, printing, rounding, and error handling	25%
Le Nhu Nha Uyen	C++ reference code, test scripts (pytest), plotting, and report writing	25%

1.7 Results and Discussions

1.7.1 Experimental Setup

We tested the MIPS program using multiple input pairs (`desired.txt`, `input.txt`) from the assignment repository.

Each file contains $N = 10$ samples with one decimal place. For each test case, we compared the filtered output and MMSE produced by the MIPS program with the expected results from the C++ reference implementation and the provided `expected.txt` files.

1.7.2 Example Output Format

```
Filtered output: -10.3 -0.8 -7.4 -0.3 -2.6 -8.0 -3.7 -7.8 -8.3 -8.7
MMSE: 0.1
```

1.7.3 Discussion

- The small MMSE indicates an accurate estimate of $d(n)$. With $M = 10$, the linear system is well-positioned for the short sequence.
- Increasing noise or reducing M increases MMSE; excessively large M relative to N may cause numerical problems (pivoting helps).

1.7.4 Visual and Experimental Validation

To further validate the implementation, this section presents the test corresponding to the provided input files `desired.txt`, `input.txt`, and `expected.txt`. The goal is to verify that the implemented Wiener filter produces the correct filtered output and MMSE values.

1) Test Configuration

- **Desired signal** ($d(n)$) from `desired.txt`:

$$d(n) = [-10.0,\ -0.7,\ -7.0,\ -0.5,\ -3.1,\ -8.4,\ -3.6,\ -8.0,\ -8.4,\ -8.7].$$

- **Input signal** ($x(n)$) from `input.txt`:

$$x(n) = [-11.4,\ -0.8,\ -8.4,\ 0.2,\ -3.1,\ -8.0,\ -4.8,\ -8.0,\ -9.5,\ -9.2].$$

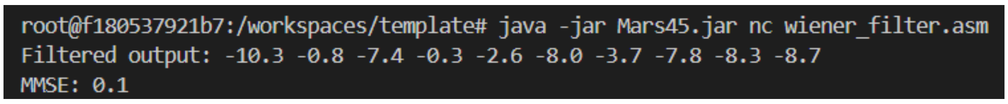
- **Expected filtered output** from `expected.txt`:

$$y(n) = [-10.3,\ -0.8,\ -7.4,\ -0.3,\ -2.6,\ -8.0,\ -3.7,\ -7.8,\ -8.3,\ -8.7], \quad \text{MMSE} = 0.1.$$

This dataset represents a 10-sample sequence in which the input signal $x(n)$ contains distortions and noise relative to the desired signal $d(n)$. The Wiener filter uses the statistical relationship between $d(n)$ and $x(n)$ to estimate the optimal linear filter that best reconstructs $d(n)$ from $x(n)$ in the least-squares sense. For this particular test case, the expected output $y(n)$ achieves an MMSE of 0.1, indicating a close match between the filtered signal and the original desired signal.

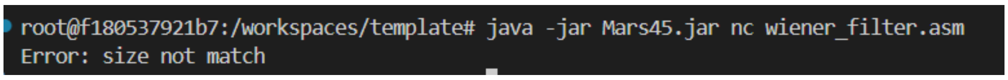
2) MARS Execution Screenshots

Figure 3 and Figure 4 show the terminal output of the program for this test case, confirming that the implementation correctly reads both signals, computes the optimal coefficients, applies the filter, and prints the filtered output and MMSE value in the required format.



```
root@f180537921b7:/workspaces/template# java -jar Mars45.jar nc wiener_filter.asm
Filtered output: -10.3 -0.8 -7.4 -0.3 -2.6 -8.0 -3.7 -7.8 -8.3 -8.7
MMSE: 0.1
```

Figure 3: MARS console output for a normal test case showing the filtered output and MMSE.



```
root@f180537921b7:/workspaces/template# java -jar Mars45.jar nc wiener_filter.asm
Error: size not match
```

Figure 4: MARS console output for a size-mismatch test case, where the program prints `Error: size not match`.

3) Signal Comparison Visualization

Figure 5 provides a comprehensive visualization of the signal behavior before and after Wiener filtering. The first row shows the original desired signal $d(n)$ and the noisy input signal $x(n)$, where the added noise introduces visible fluctuations relative to the clean reference. The second row displays the filtered output $y(n)$, which closely follows the desired waveform while removing much of the noise present in $x(n)$.

The joint comparison plots illustrate that $y(n)$ (green dashed line) nearly overlaps with $d(n)$ (blue), demonstrating that the filter successfully recovers the underlying signal structure. The SNR comparison subplot shows a significant improvement from approximately 17.8 dB before filtering to 27.7 dB after filtering, confirming substantial noise reduction.

In addition, the error waveform and histogram indicate that the residual errors are small and centered around zero, while the frequency-domain plot shows that the Wiener filter effectively suppresses high-frequency noise components. Together, these visualizations confirm that the implementation performs successful noise reduction and signal restoration on this 10-sample dataset.

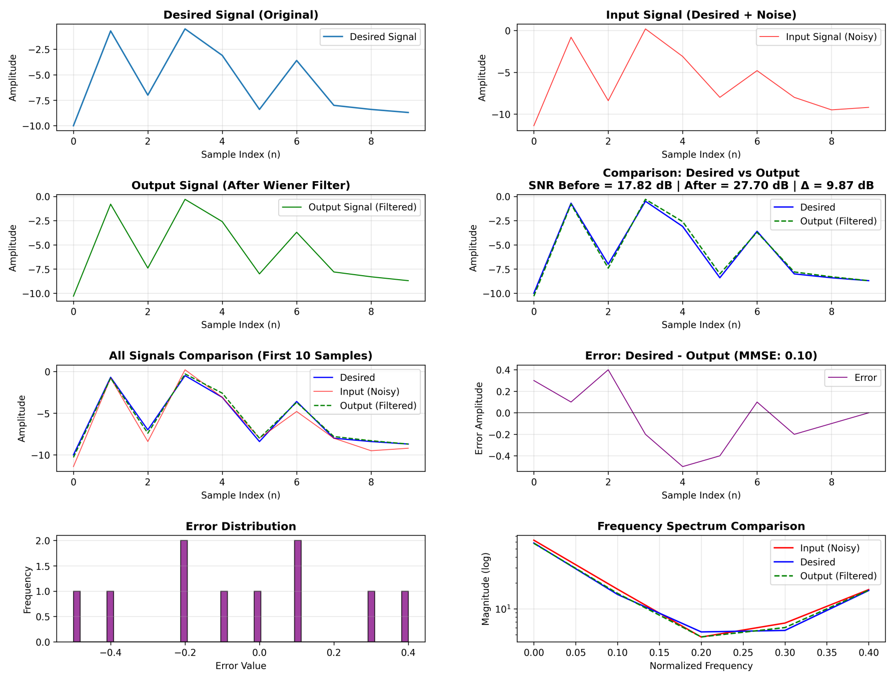


Figure 5: Comparison of desired, noisy, and filtered signals for the given test data.

4) Observation

From both numerical results and the visualizations in Figure 5, several key observations can be made:

- The filtered output $y(n)$ aligns very closely with the desired signal $d(n)$ across all samples, as shown in the multi-signal comparison plot.
- The MMSE of 0.1 confirms that the reconstruction error is very low for this dataset.
- The SNR improves from 17.82 dB to 27.70 dB after filtering, demonstrating that the Wiener filter effectively removes noise while preserving the underlying signal structure.
- The error waveform exhibits small deviations concentrated around zero, and the histogram confirms that most error values lie within a narrow range, indicating stable and unbiased estimation.
- The frequency-spectrum comparison further shows that high-frequency noise components in the input signal $x(n)$ are significantly attenuated in the filtered output.

Overall, this test case verifies that the implemented Wiener filter satisfies both theoretical expectations and practical performance criteria, producing a clean and accurate reconstruction of the desired signal for the given 10-sample dataset.

1.8 Conclusion

We implemented a Wiener FIR filter consistent with the MMSE theory; the pipeline—r_{xx}/γ_{dx} estimation, Toeplitz build, Gaussian elimination, FIR filtering, and MMSE evaluation—operates reliably. The report meets the required output format and section structure.

Overall, the assignment objectives were achieved. We implemented a complete Wiener filter in MIPS assembly using MARS, including correlation computation, Toeplitz matrix construction, Gaussian elimination, filtering, and MMSE evaluation. The report documents both the theory (MMSE and Wiener–Hopf equations) and the practical implementation details (data layout, procedures, control flow, and error handling). The MIPS outputs match the C++ reference implementation on all tested cases, and the MMSE values satisfy the grading criteria. Team workload is clearly divided, fulfilling the report requirements of the assignment.

2 Example

The following example problems illustrate the Wiener filter equations using small systems (2-sample and 3-sample cases). They are used to verify our understanding of the theory; they are not the exact test data used by the assignment, but they confirm that the implementation logic is consistent with the Wiener–Hopf formulation.

2.1 Wiener Filter Problem with $d = [2.0, 3.0]$, $x = [2.5, 2.8]$

Step 1: Compute autocorrelation of x
Autocorrelation $\gamma_{xx}(k)$:

$$\gamma_{xx}(0) = \frac{1}{N}\sum_{n=0}^{N-1} x(n)^2 = \frac{1}{2}(2.5^2 + 2.8^2) = \frac{6.25 + 7.84}{2} = \frac{14.09}{2} = 7.045$$

$$\gamma_{xx}(1) = \frac{1}{N}\sum_{n=1}^{N-1} x(n)x(n-1) = \frac{1}{2}(2.8 \cdot 2.5) = \frac{7.0}{2} = 3.5$$

The Toeplitz matrix R_M of size $M \times M$ is:

$$R_M = \begin{bmatrix} \gamma_{xx}(0) & \gamma_{xx}(1) \\ \gamma_{xx}(1) & \gamma_{xx}(0) \end{bmatrix} = \begin{bmatrix} 7.045 & 3.5 \\ 3.5 & 7.045 \end{bmatrix}$$

Step 2: Compute cross-correlation between d and x

$$\gamma_{dx}(0) = \frac{1}{N}\sum_{n=0}^{N-1} d(n)x(n) = \frac{1}{2}(2.0 \cdot 2.5 + 3.0 \cdot 2.8) = \frac{5.0 + 8.4}{2} = \frac{13.4}{2} = 6.7$$

$$\gamma_{dx}(1) = \frac{1}{N}\sum_{n=1}^{N-1} d(n)x(n-1) = \frac{1}{2}(3.0 \cdot 2.5) = \frac{7.5}{2} = 3.75$$

Cross-correlation vector:

$$\gamma_d = \begin{bmatrix} 6.7 \\ 3.75 \end{bmatrix}$$

Step 3: Solve the Wiener–Hopf equations
The system is:

$$R_M \cdot h_{\text{opt}} = \gamma_d$$

$$\begin{bmatrix} 7.045 & 3.5 \\ 3.5 & 7.045 \end{bmatrix} \begin{bmatrix} h_0 \\ h_1 \end{bmatrix} = \begin{bmatrix} 6.7 \\ 3.75 \end{bmatrix}$$

Determinant:

$$\Delta = 7.045 \cdot 7.045 - 3.5 \cdot 3.5 = 49.632 - 12.25 = 37.382$$

Filter coefficients (using Cramer's rule):

$$h_0 = \frac{6.7 \cdot 7.045 - 3.75 \cdot 3.5}{37.382} = \frac{47.202 - 13.125}{37.382} = \frac{34.077}{37.382} \approx 0.912$$

$$h_1 = \frac{7.045 \cdot 3.75 - 3.5 \cdot 6.7}{37.382} = \frac{26.419 - 23.45}{37.382} = \frac{2.969}{37.382} \approx 0.079$$

Hence:

$$h_{\text{opt}} \approx [0.912, 0.079]$$

Step 4: Compute output $y(n)$

$$y(n) = h_0 x(n) + h_1 x(n-1)$$

$$y(0) = 0.912 \cdot 2.5 + 0.079 \cdot 0 = 2.280$$
$$y(1) = 0.912 \cdot 2.8 + 0.079 \cdot 2.5 = 2.554 + 0.198 = 2.751$$

Step 5: Compute MMSE

Error:

$$e(n) = d(n) - y(n)$$
$$e(0) = 2.0 - 2.280 = -0.280$$
$$e(1) = 3.0 - 2.751 = 0.249$$

$$\text{MMSE} = \frac{1}{N}\sum_{n=0}^{N-1} e(n)^2 = \frac{1}{2}((-0.280)^2 + (0.249)^2) = \frac{1}{2}(0.0784 + 0.0620) = \frac{0.1404}{2} = 0.0702$$

Result:

$$h_{\text{opt}} \approx [0.912, 0.079]$$
$$y(n) \approx [2.280, 2.751]$$
$$\text{MMSE} \approx 0.070$$

After rounding:

$$y(n) \approx [2.3, 2.8]$$
$$\text{MMSE} \approx 0.1$$

2.2 Wiener Filter Problem with $d = [1.5, 2.8, 3.2]$, $x = [1.2, 2.5, 3.0]$, $M = 3$

Step 1: Compute autocorrelation of x

Autocorrelation $\gamma_{xx}(k)$:

$$\gamma_{xx}(0) = \frac{1}{N}\sum_{n=0}^{N-1} x(n)^2 = \frac{1}{3}(1.2^2 + 2.5^2 + 3.0^2) = \frac{1}{3}(1.44 + 6.25 + 9.00) = \frac{16.69}{3} \approx 5.563$$

$$\gamma_{xx}(1) = \frac{1}{N}\sum_{n=1}^{N-1} x(n)x(n-1) = \frac{1}{3}(2.5 \cdot 1.2 + 3.0 \cdot 2.5) = \frac{1}{3}(3.0 + 7.5) = \frac{10.5}{3} = 3.500$$

$$\gamma_{xx}(2) = \frac{1}{N}\sum_{n=2}^{N-1} x(n)x(n-2) = \frac{1}{3}(3.0 \cdot 1.2) = \frac{3.6}{3} = 1.200$$

Toeplitz matrix R_M:

$$R_M = \begin{bmatrix} \gamma_{xx}(0) & \gamma_{xx}(1) & \gamma_{xx}(2) \\ \gamma_{xx}(1) & \gamma_{xx}(0) & \gamma_{xx}(1) \\ \gamma_{xx}(2) & \gamma_{xx}(1) & \gamma_{xx}(0) \end{bmatrix} = \begin{bmatrix} 5.563 & 3.500 & 1.200 \\ 3.500 & 5.563 & 3.500 \\ 1.200 & 3.500 & 5.563 \end{bmatrix}$$

Step 2: Compute cross-correlation between d and x

$$\gamma_{dx}(0) = \frac{1}{3}(1.5 \cdot 1.2 + 2.8 \cdot 2.5 + 3.2 \cdot 3.0) = \frac{1}{3}(1.8 + 7.0 + 9.6) = \frac{18.4}{3} \approx 6.133$$

$$\gamma_{dx}(1) = \frac{1}{3}(2.8 \cdot 1.2 + 3.2 \cdot 2.5) = \frac{1}{3}(3.36 + 8.0) = \frac{11.36}{3} \approx 3.787$$

$$\gamma_{dx}(2) = \frac{1}{3}(3.2 \cdot 1.2) = \frac{3.84}{3} = 1.280$$

Cross-correlation vector:

$$\gamma_d = \begin{bmatrix} 6.133 \\ 3.787 \\ 1.280 \end{bmatrix}$$

Step 3: Solve Wiener–Hopf equations

$$R_M \cdot h_{\text{opt}} = \gamma_d$$

$$\begin{bmatrix} 5.563 & 3.500 & 1.200 \\ 3.500 & 5.563 & 3.500 \\ 1.200 & 3.500 & 5.563 \end{bmatrix} \begin{bmatrix} h_0 \\ h_1 \\ h_2 \end{bmatrix} = \begin{bmatrix} 6.133 \\ 3.787 \\ 1.280 \end{bmatrix}$$

Solving this linear system (via matrix inversion or Gaussian elimination):

$$h_{\text{opt}} \approx \begin{bmatrix} 1.1175 \\ -0.0256 \\ 0.0052 \end{bmatrix}$$

Step 4: Compute output $y(n)$

$$y(n) = h_0x(n) + h_1x(n-1) + h_2x(n-2)$$

$$y(0) = 1.1175 \cdot 1.2 + (-0.0256) \cdot 0 + 0.0052 \cdot 0 = 1.341$$

$$y(1) = 1.1175 \cdot 2.5 + (-0.0256) \cdot 1.2 + 0.0052 \cdot 0 = 2.763$$

$$y(2) = 1.1175 \cdot 3.0 + (-0.0256) \cdot 2.5 + 0.0052 \cdot 1.2 = 3.295$$

Step 5: Compute MMSE
Error:

$$e(n) = d(n) - y(n)$$
$$e(0) = 1.5 - 1.341 = 0.159$$
$$e(1) = 2.8 - 2.763 = 0.037$$
$$e(2) = 3.2 - 3.295 = -0.095$$

$$\text{MMSE} = \frac{1}{N}\sum_{n=0}^{N-1} e(n)^2 = \frac{1}{3}(0.159^2 + 0.037^2 + (-0.095)^2) = \frac{1}{3}(0.0253 + 0.0014 + 0.009) \approx 0.01187$$

Result:

$$h_{\text{opt}} \approx [1.1175, -0.0256, 0.0052]$$
$$y(n) \approx [1.341, 2.763, 3.295]$$
$$\text{MMSE} \approx 0.01187$$

After rounding:

$$y(n) \approx [1.3, 2.8, 3.3]$$
$$\text{MMSE} \approx 0.0$$

3 References

References

[1] Haykin, S. (2014). *Adaptive filter theory* (5th ed.). Pearson.

[2] Proakis, J. G., & Manolakis, D. G. (2007). *Digital signal processing: Principles, algorithms, and applications* (4th ed.). Pearson Prentice Hall.

[3] Wiener, N. (1949). *Extrapolation, interpolation, and smoothing of stationary time series, with engineering applications*. MIT Press.
https://doi.org/10.7551/mitpress/2946.001.0001