Final Presentation

Memory Efficient Signal Reconstruction from Phaseless Coefficients of a Linear Mapping

Naveed Haghani
nhaghan1@math.umd.edu

Project Advisor:
Dr. Radu Balan
rvbalan@cscamm.umd.edu
Professor of Applied Mathematics, University of Maryland
Department of Mathematics
Center for Scientific Computation and Mathematical Modeling
Norbert Wiener Center

Problem Overview

Original Signal

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_n \end{bmatrix} \epsilon\ \mathbb{C}^n$$

Transformation

Transformation

$$c = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ \vdots \\ c_m \end{bmatrix} \epsilon\ \mathbb{C}^m$$

Transformation Magnitudes

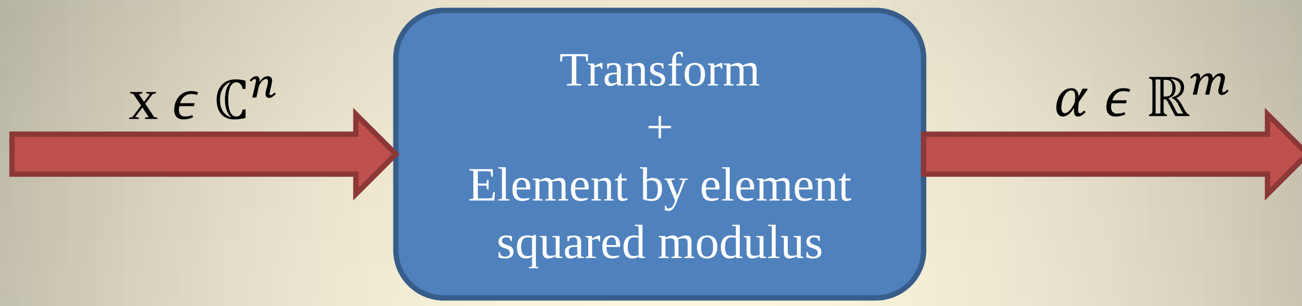
$$\alpha = \begin{bmatrix} |c_1|^2 \\ |c_2|^2 \\ \vdots \\ \vdots \\ |c_m|^2 \end{bmatrix} \epsilon\ \mathbb{R}^m$$

Reconstruct Original Signal

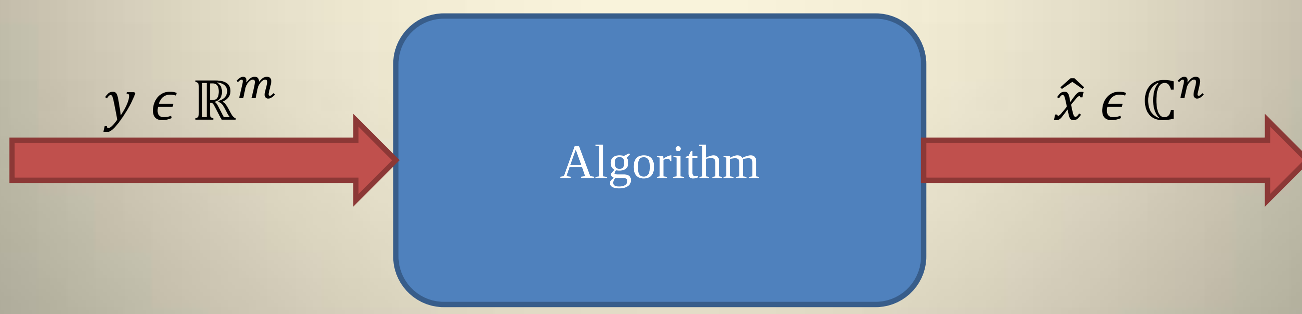
Original Signal Approximation

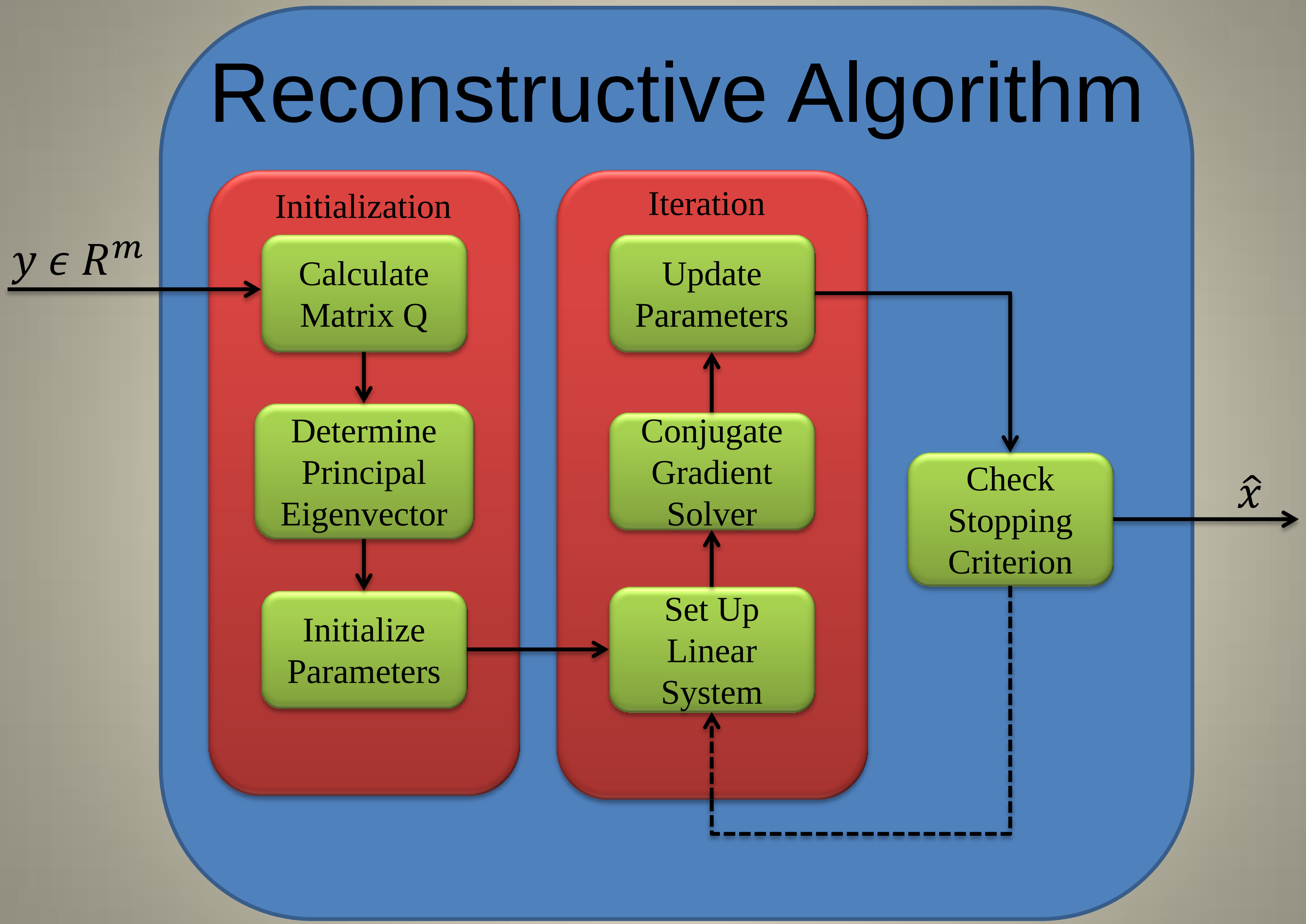
$$\hat{x} = \begin{bmatrix} \hat{x}_1 \\ \hat{x}_2 \\ \vdots \\ \vdots \\ \hat{x}_n \end{bmatrix} \epsilon\ \mathbb{C}^n$$

Approach



$$y = \alpha + \sigma \cdot \nu, \qquad \sigma \cdot \nu: \textit{Gaussian noise}$$




Reconstructive Algorithm
$y \in R^m$
Initialization
Calculate Matrix Q
Determine Principal Eigenvector
Initialize Parameters
Iteration
Update Parameters
Conjugate Gradient Solver
Set Up Linear System
Check Stopping Criterion
$\hat{x}$

Transformation $T(x) = c$

- Redundant linear transformation
- Maps vector in $\mathbb{C}^n$ to $\mathbb{C}^m$
  - $m = R \cdot n$
  - R is redundancy of the Transformation $T(x)$
- Defined by m vectors in $\mathbb{C}^n$ labeled $f_{1:m}$ such that:

$$T(x) = c = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ \vdots \\ c_m \end{bmatrix} \text{ and } c_i = \langle x, f_i \rangle$$

Transformation $T(x) = c$

- Weighted Discrete Fourier Transform

$$B_j = Discrete\ Fourier\ Transform \left\{ \begin{bmatrix} w_1^{(j)} & & 0 \\ & \ddots & \\ 0 & & w_n^{(j)} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_n \end{bmatrix} \right\}$$

$$for\ 1 \le j \le R\ randomly\ generated\ arrays\ of\ complex\ weights \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ \vdots \\ w_n \end{bmatrix}$$

$$T(x) = c = \frac{1}{\sqrt{R \cdot n}} \cdot \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ \vdots \\ B_R \end{bmatrix}$$

Algorithm Initialization

- Solve regularized least squares:

$$\min_u \|y - \alpha(u)\| + 2\lambda\|u\|^2$$ [4]

$$\approx \min_u \|y\|^2 + 2\langle(\lambda I - Q)u, u\rangle$$ [4]

$$Q = \sum_{k=1}^{m} y_k f_k f_k^*$$ [4]

Algorithm Initialization

$$Q = \sum_{k=1}^{m} y_k f_k f_k^*$$ [4]

e: *principal eigenvector of* Q^+
a: *associated eigenvalue of* Q
ρ: *constant between* $(0, 1)$

$$\hat{x}^{(0)} = e\sqrt{\frac{(1-\rho)\cdot a}{\sum_{k=1}^{m}|\langle e, f_k\rangle|^4}}$$ [4]

Algorithm Initialization (cont.)

- Regularization variables:
  - μ
    - Controls step size between successive approximations
    - Larger μ → smaller jumps
  - λ
    - Penalizes approximations large in magnitude
- Initialization
  - $\mu_0 = \lambda_0 = \rho \cdot a$ [4]

Algorithm Iteration

- Work in Real space
  - $\xi = \begin{bmatrix} real(\hat{x}) \\ imag(\hat{x}) \end{bmatrix}$
- Solve linear system $A\xi^{(t+1)} = b$, where

$$A = \sum_{k=1}^{m} (\Phi_k \xi^{(t)}) \cdot (\Phi_k \xi^{(t)})^* + (\lambda_t + \mu_t) \cdot I$$ [4]

$$b = \left(\sum_{k=1}^{m} y_k \Phi_k + \mu_t \cdot I \right) \cdot \xi^t$$ [4]

$\Phi_k = \phi_k \phi_k^T + J \phi_k \phi_k^T \mathrm{J}^\mathrm{T}$, where $\phi_k = \begin{bmatrix} real(f_k) \\ imag(f_k) \end{bmatrix}$ and $\mathrm{J} = \begin{bmatrix} 0 & -I \\ I & 0 \end{bmatrix}$

$\xi^{(t+1)}$ = next approximation

Algorithm Iteration (cont.)

- Update λ, μ

$$\lambda_{t+1} = \gamma\lambda_{t+1}, \qquad \mu_{t+1} = \max\left(\gamma\mu_t, \mu^{min}\right), \qquad \text{where } 0 < \gamma < 1$$ [4]

- Stopping criterion

$$\sum_{k=1}^{m} \left| y_k - \left|\left\langle x^{(t)}, f_k \right\rangle\right|^2 \right|^2 > \sum_{k=1}^{m} \left| y_k - \left|\left\langle x^{(t-1)}, f_k \right\rangle\right|^2 \right|^2$$

Big Picture

Pre-processing	Reconstructive Algorithm	Post-processing
▪ Retrieve input data ▪ Transform signal ▪ Generate noise vector ▪ Add noise ▪ Call reconstructive algorithm, $LS\_Algorithm()$	▪ Retrieve principal eigenvector ▪ Initialize approximation ▪ Iterate recursive algorithm until error is minimized	▪ Load output data ▪ Calculate Bias/Variance/MSE ▪ Calculate CRLB ▪ Plot results

Testing

- Test over SNR levels
  - [-30, -20, -10, 0, 10, 20, 30] dB
- Test over different realizations of random noise.
  - $n = 1{,}000$
  - $1{,}000$ noise realizations
- Find the bias of the mean, variance, the mean squared error, and CRLB

Parallelization

Loop over all inputs:

parfor — Loop over all SNR levels:

parfor — Loop over all noise realizations:

$LS\_Algorithm()$

end

end

end

Parallelization (cont.)

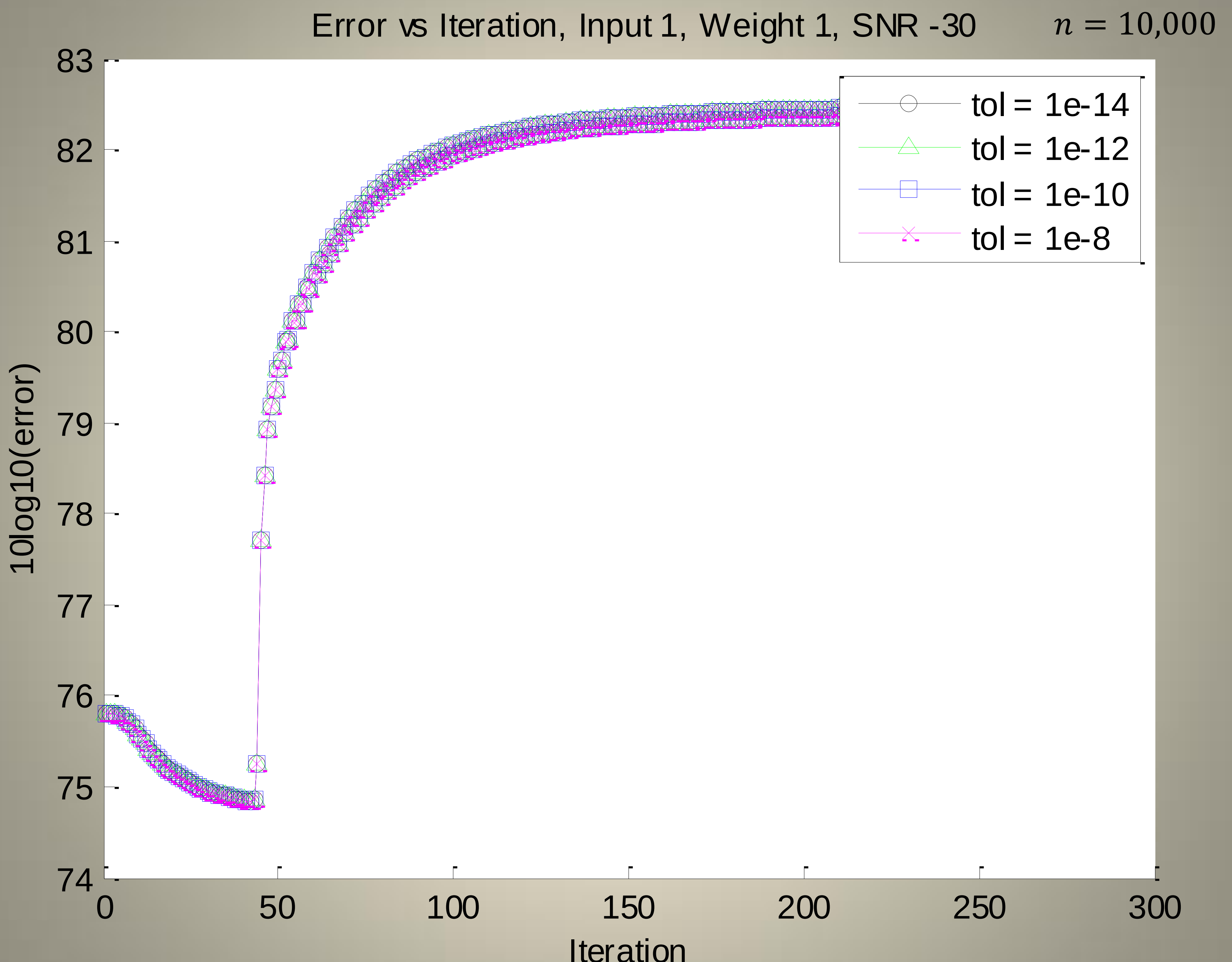
- Noise Realization loop iterations > SNR level loop iterations
- Parallelizing here increases efficiency
  - Reduces thread downtime
  - Increases maximum number of potential threads
- Problem: no mutual exclusion
  - Needed for random number generation
  - Solution: Independent random number streams

Power Method tol

- Power Method takes up $\approx 50\%$ of program time
- Reducing the Power Method tolerance will increase program performance speed
- Studied the cost-benefit relation of several tolerance values.
  - $n = 10{,}000$

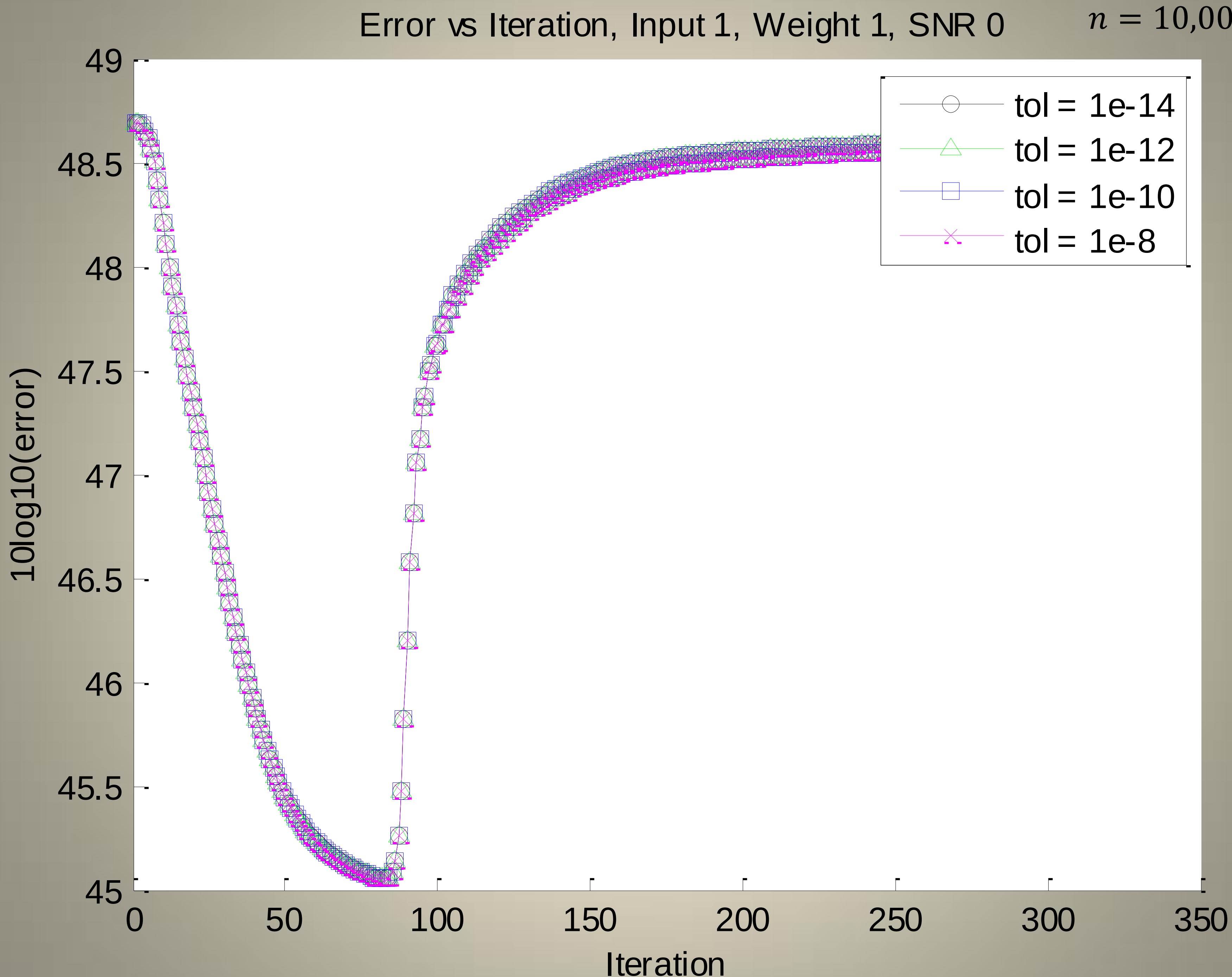
Time Consumption by Function

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
LS_Algorithm	1	181.935 s	0.377 s	
Power_Method	1	94.735 s	5.717 s	
Q_u_compute	12717	89.018 s	4.378 s	
Conjugate_Gradient	219	86.267 s	2.215 s	
A_u_compute	7957	82.393 s	7.321 s	
adjTransformation	20893	82.302 s	82.302 s	
Transformation	29071	79.532 s	79.532 s	
RHS_compute	219	1.659 s	0.093 s	

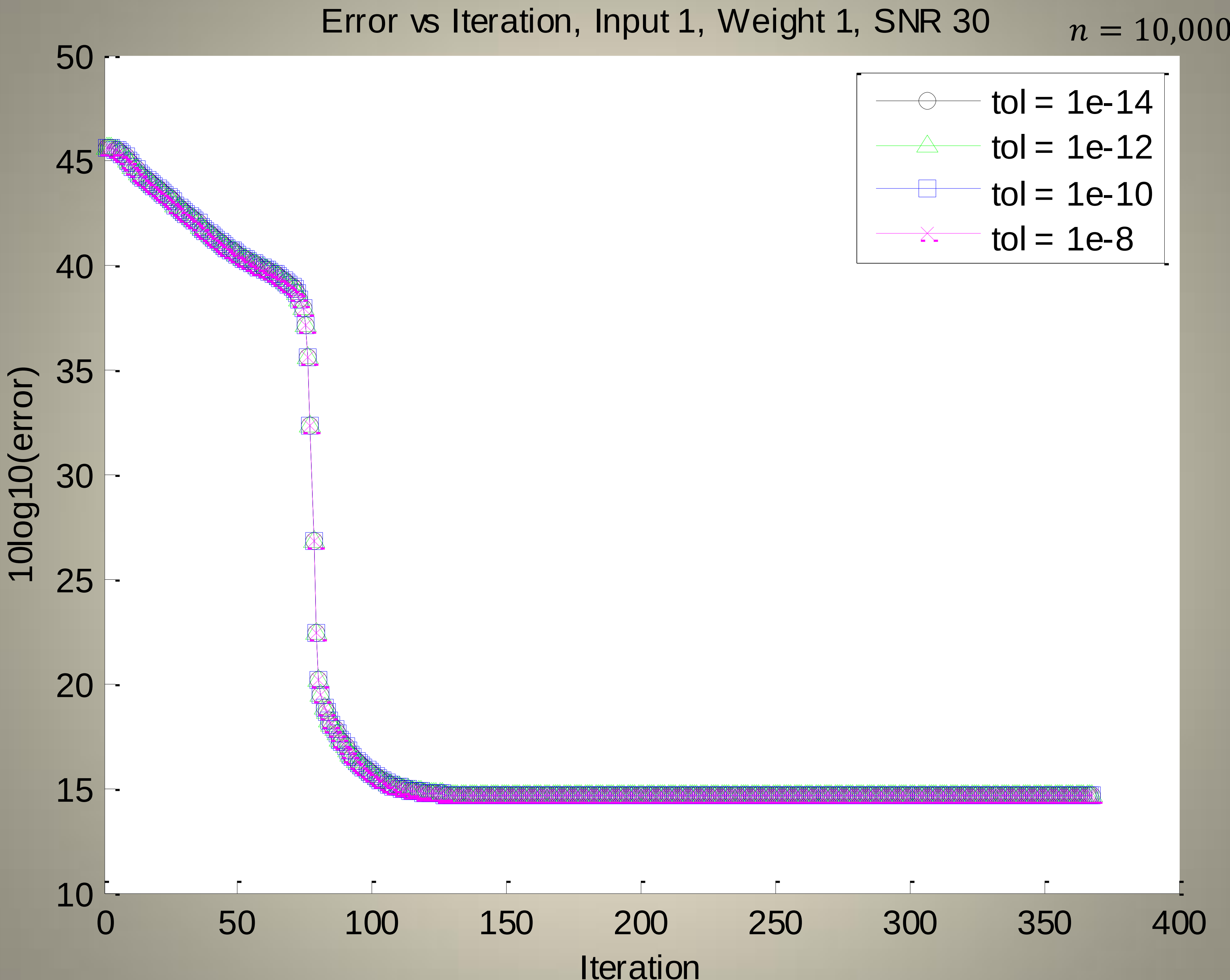


Error vs Iteration, Input 1, Weight 1, SNR -30
$n = 10{,}000$
10log10(error)
Iteration
83
82
81
80
79
78
77
76
75
74
0
50
100
150
200
250
300
tol = 1e-14
tol = 1e-12
tol = 1e-10
tol = 1e-8

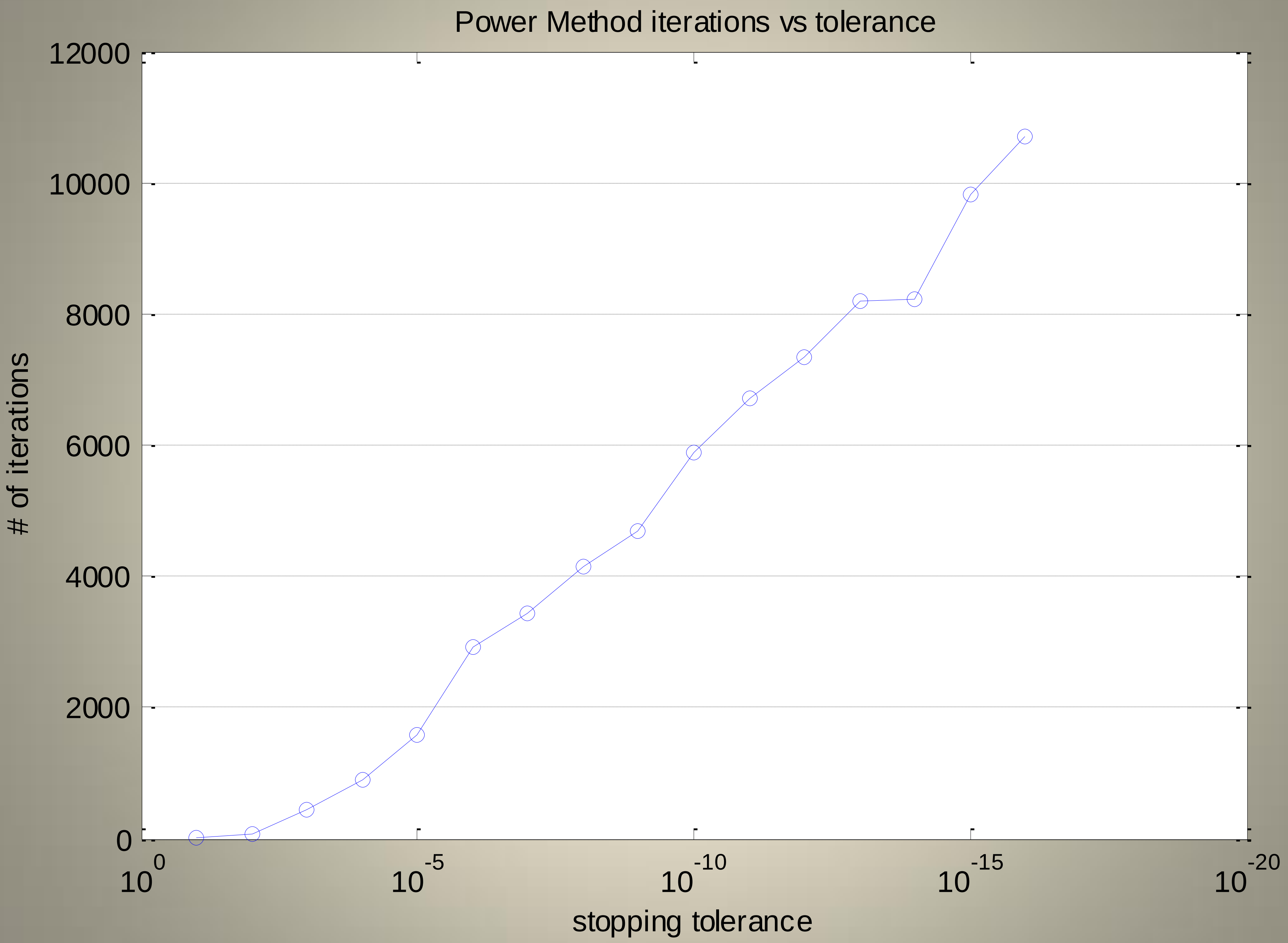
$n = 10{,}000$



Error vs Iteration, Input 1, Weight 1, SNR 0
10log10(error)
Iteration
tol = 1e-14
tol = 1e-12
tol = 1e-10
tol = 1e-8
49
48.5
48
47.5
47
46.5
46
45.5
45
0
50
100
150
200
250
300
350



Error vs Iteration, Input 1, Weight 1, SNR 30
$n = 10{,}000$
tol = 1e-14
tol = 1e-12
tol = 1e-10
tol = 1e-8
10log10(error)
50
45
40
35
30
25
20
15
10
0
50
100
150
200
250
300
350
400
Iteration



Power Method iterations vs tolerance
12000
10000
8000
6000
4000
2000
0
# of iterations
10^{0}
10^{-5}
10^{-10}
10^{-15}
10^{-20}
stopping tolerance

Power Method tol (cont.)

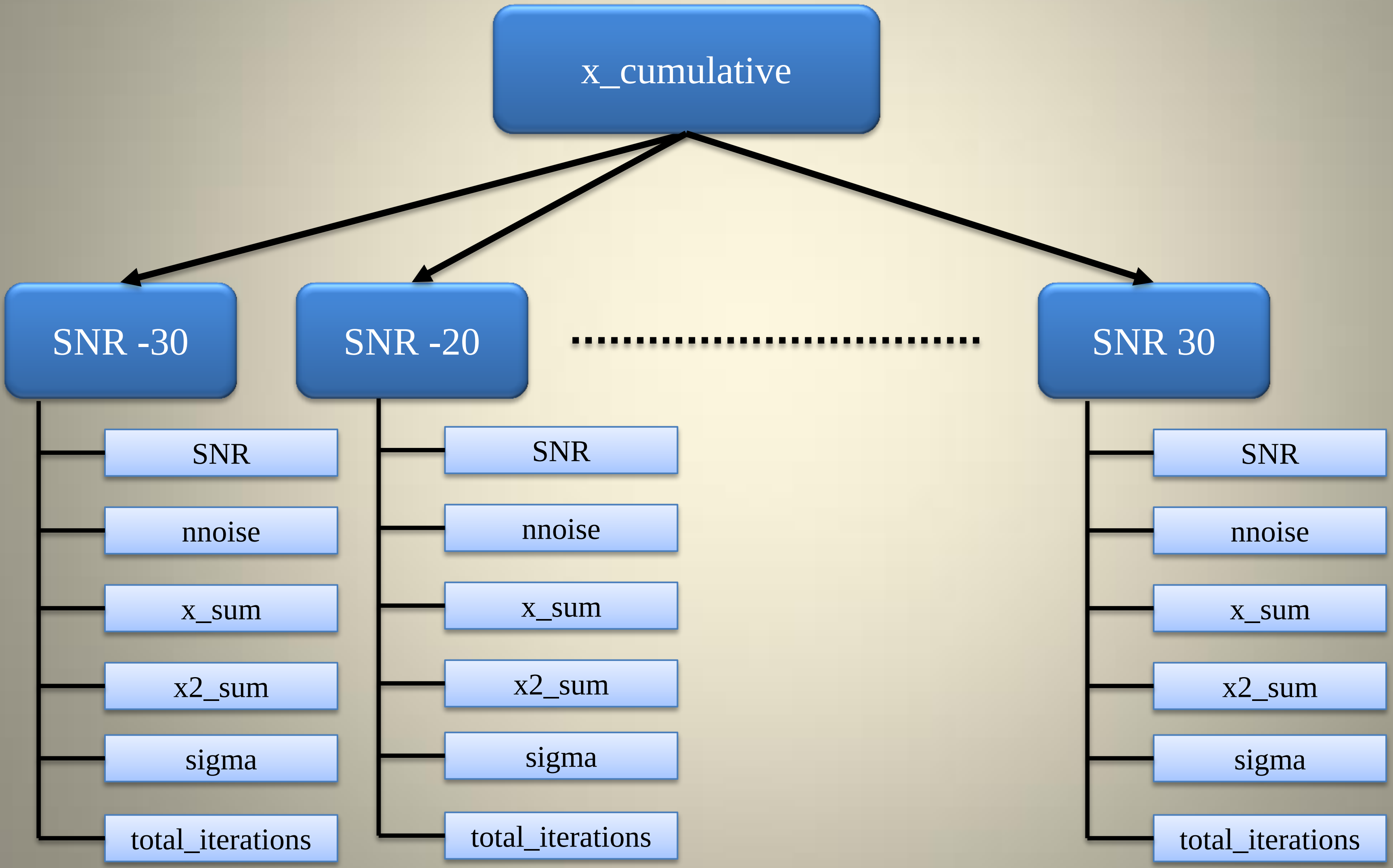
- Reducing the Power Method tolerance from 10^{-14} to as low as 10^{-8} does not affect program accuracy
- Reducing Power Method tolerance from 10^{-14} to 10^{-8} cuts approximately 50% of Power Method runtime
  – Translates to approximately 25% of total program runtime

Stored Output ($output.mat$)

For a given input-weight combination, variables are stored in an output $.mat$ file

SNR_level	Array of SNR values
input_num	Input file number
weight_num	Weight file number
x_solution	Original signal
x_cumulative	Array of structure results
param	Structure of all parameters

Stored Output (cont.)



Bias of the mean

$$x\_sum_i = x\_sum_i + \hat{x}_i \ \ \forall i$$

$$bias\_vec = Sample\ Mean\ - Original\ Signal(x)$$

$$bias\_vec = \frac{1}{nnoise} x\_sum\ - x$$

$$Bias = \sum_{i=1}^{n} |bias\_vec_i|^2$$

Variance and MSE

- Each noise realization:

$$x2\_sum_i = x2\_sum_i + |\hat{x}_i|^2 \ \ \forall i$$

$$Var = \frac{\sum_{i=1}^{n}\left(x2_{sum_i} - \frac{\left|x_{sum_i}\right|^2}{nnoise}\right)}{nnoise - 1}$$

$$MSE = \left(1 - \frac{1}{nnoise}\right) \cdot Var + Bias$$

Cramer Rao Lower Bound

- Gives lower bound for the variance of the unbiased estimator

$$CRLB = 10 \cdot \log_{10}\left(\frac{\check{\sigma}^2}{4} \cdot (trc - 1)\right)$$

$$trc = trace\left(\tilde{A}^{-1}\right)$$

$$\check{\sigma} = noise\ std.\ dev.$$

$$\tilde{A} = modified\ Fisher\ information\ matrix$$

$$\tilde{A} = \sum_{k=1}^{m} \Phi_k \xi \xi^T \Phi_k^T + \frac{1}{\|x\|^2} \cdot J \xi \xi^T J^T$$

[4]

Cramer Rao Lower Bound (cont.)

$$trace(\tilde{A}^{-1}) = \sum_{k=1}^{2n} \langle \tilde{A}^{-1} e_k, e_k \rangle$$

$$trc = 0$$

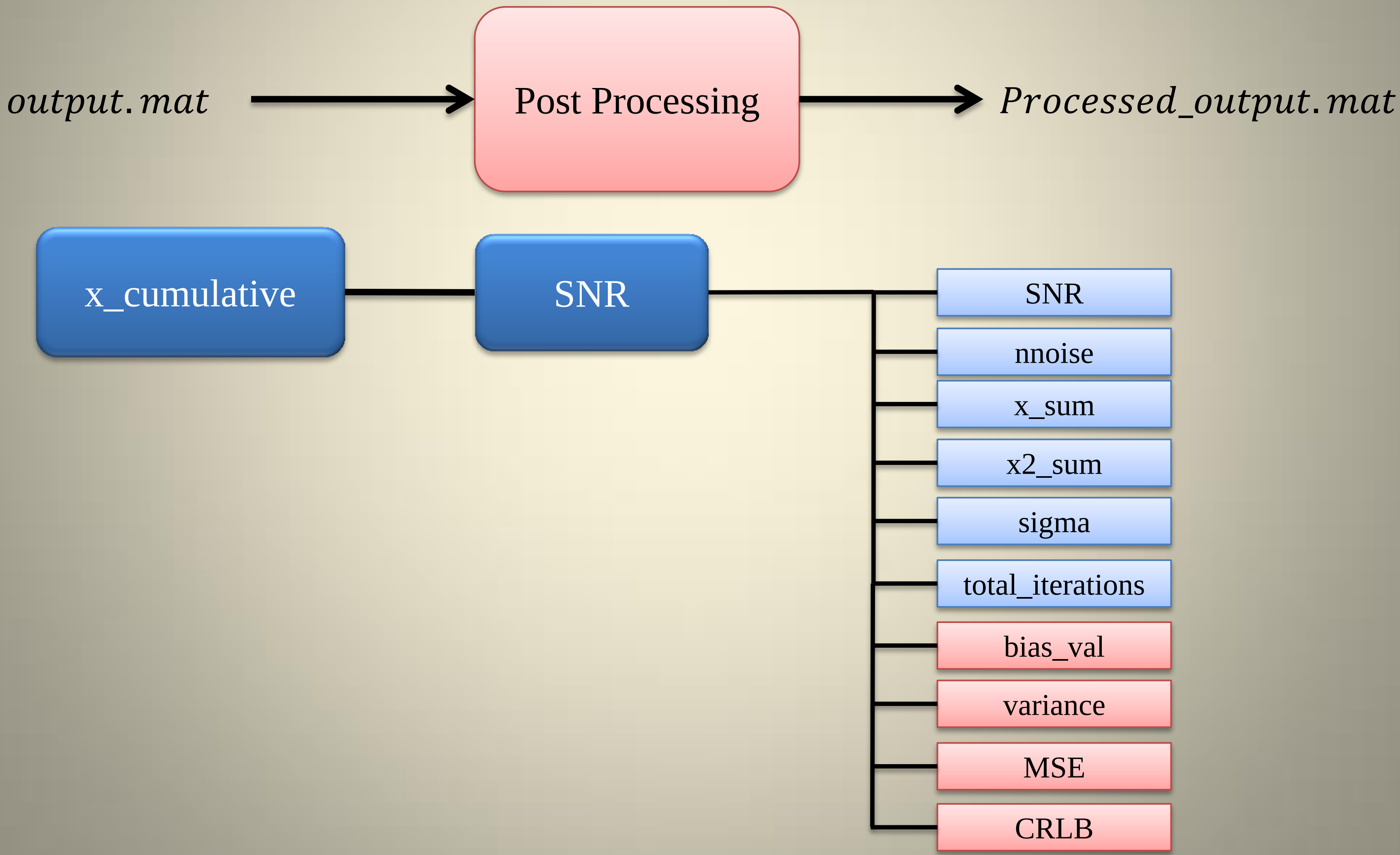
$$Loop\ k = 1:2n$$

$$solve\ for\ z:\ \ \tilde{A}z = e_k$$

$$trc = trc + \langle z, e_k \rangle$$

$$end$$

Post Processing output



Parallelization (PostProcessing)

Loop over all output files:

    Calculate trc for CRLB

    Loop over all SNR levels:

        Calculate Bias

        Calculate Variance

        Calculate MSE

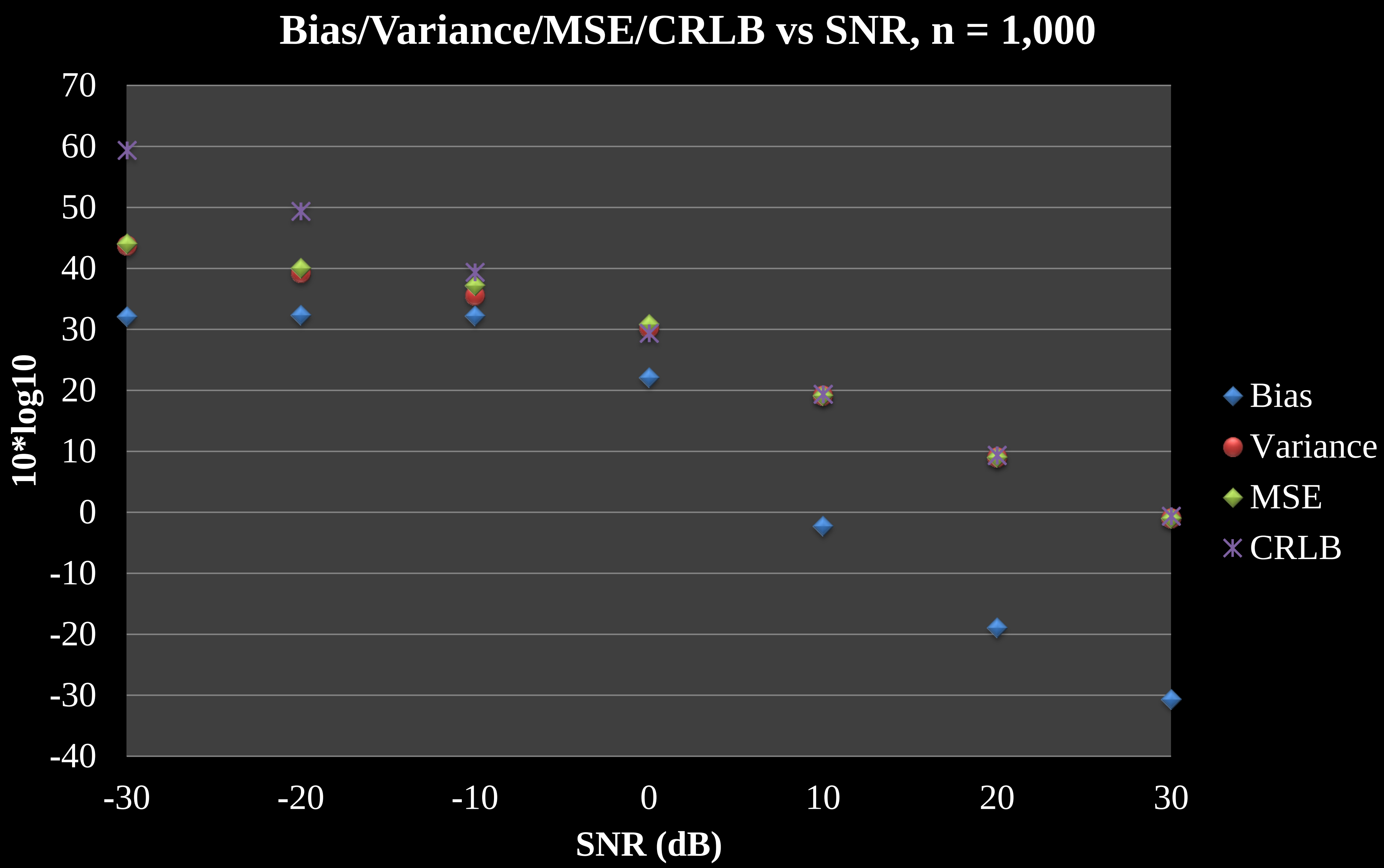
        Calculate Cramer Rao Lower Bound

    end

    Plot and save results

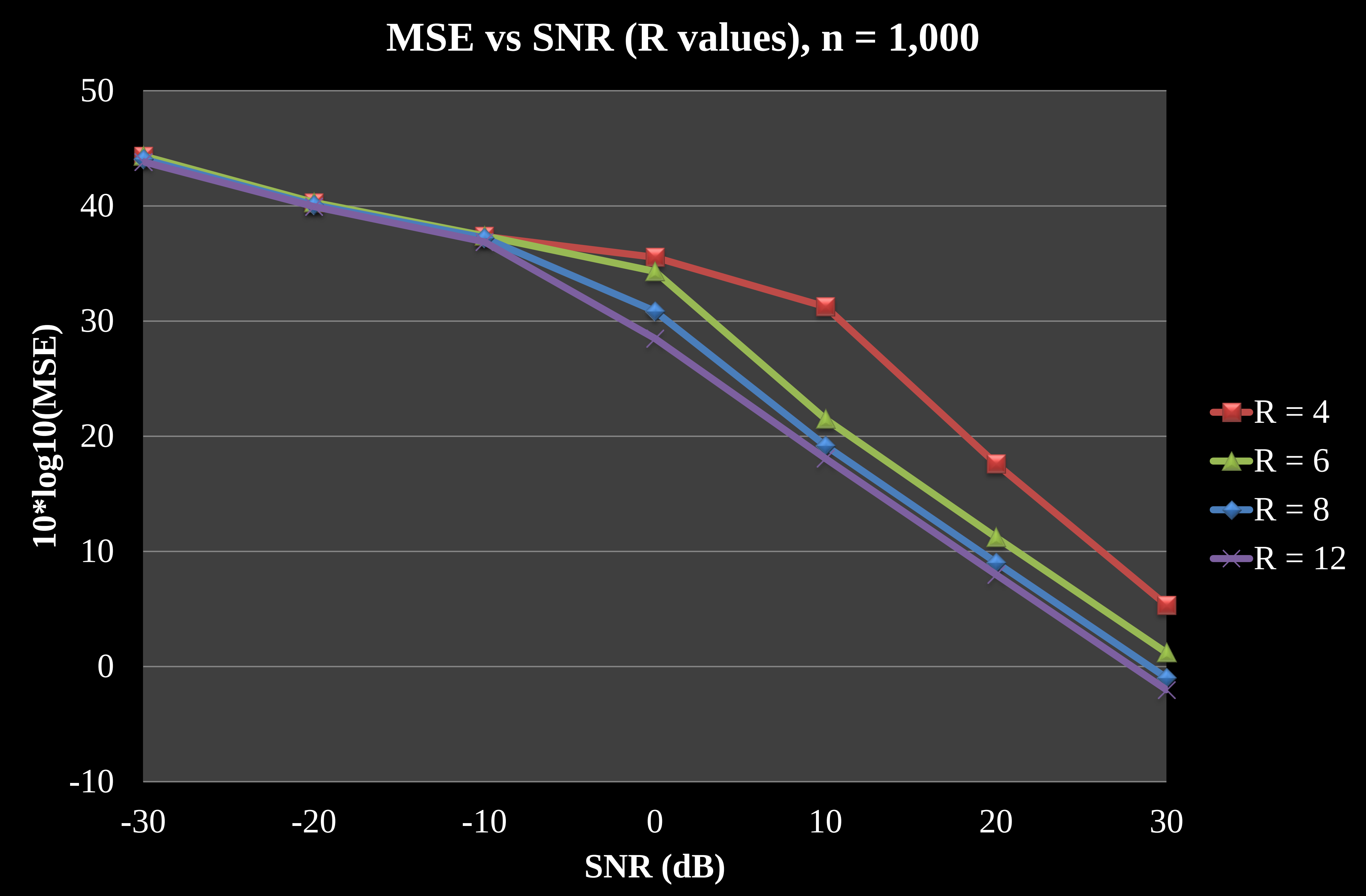
end

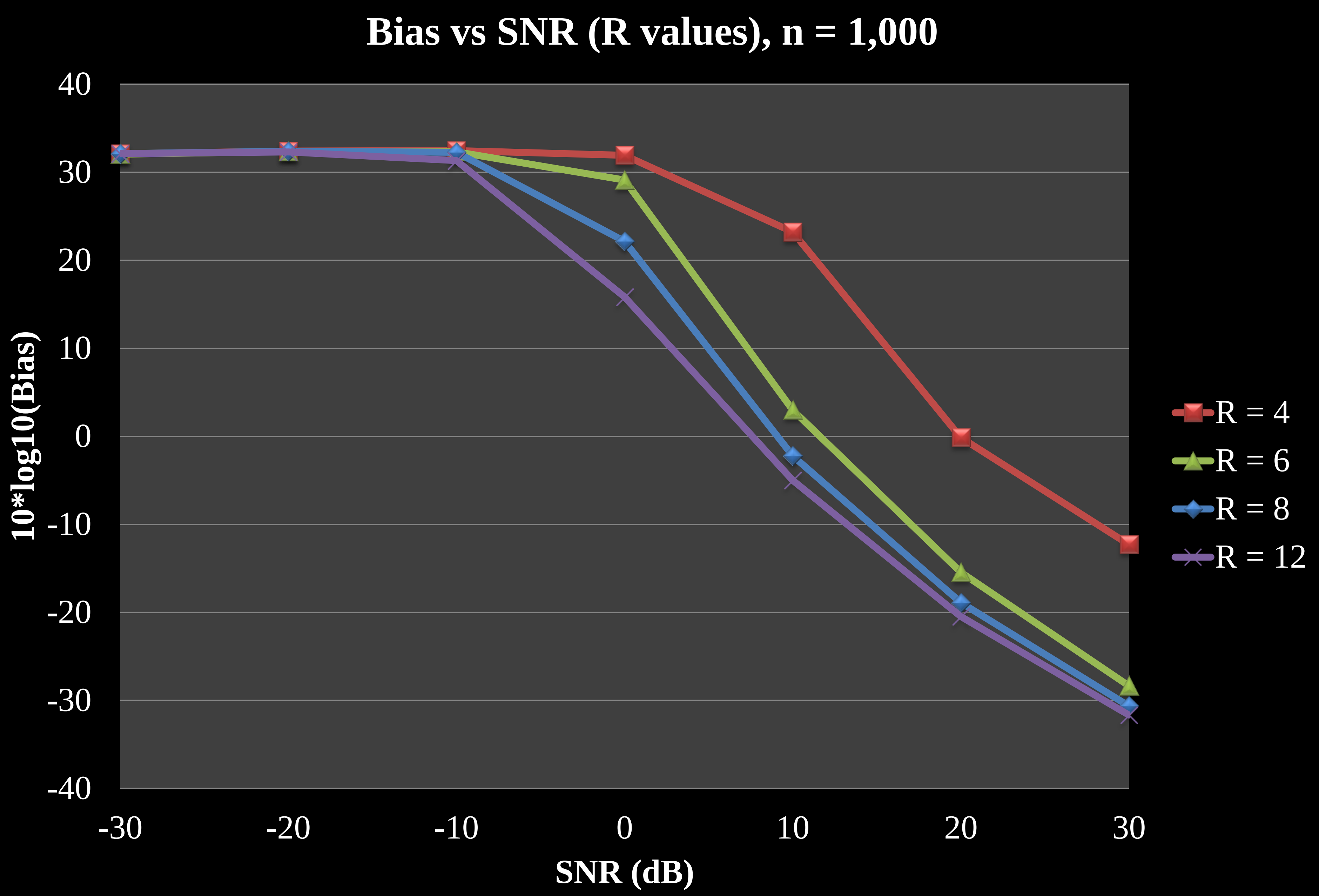
parfor


Bias/Variance/MSE/CRLB vs SNR, n = 1,000
10*log10
70
60
50
40
30
20
10
0
-10
-20
-30
-40
-30
-20
-10
0
10
20
30
SNR (dB)
Bias
Variance
MSE
CRLB

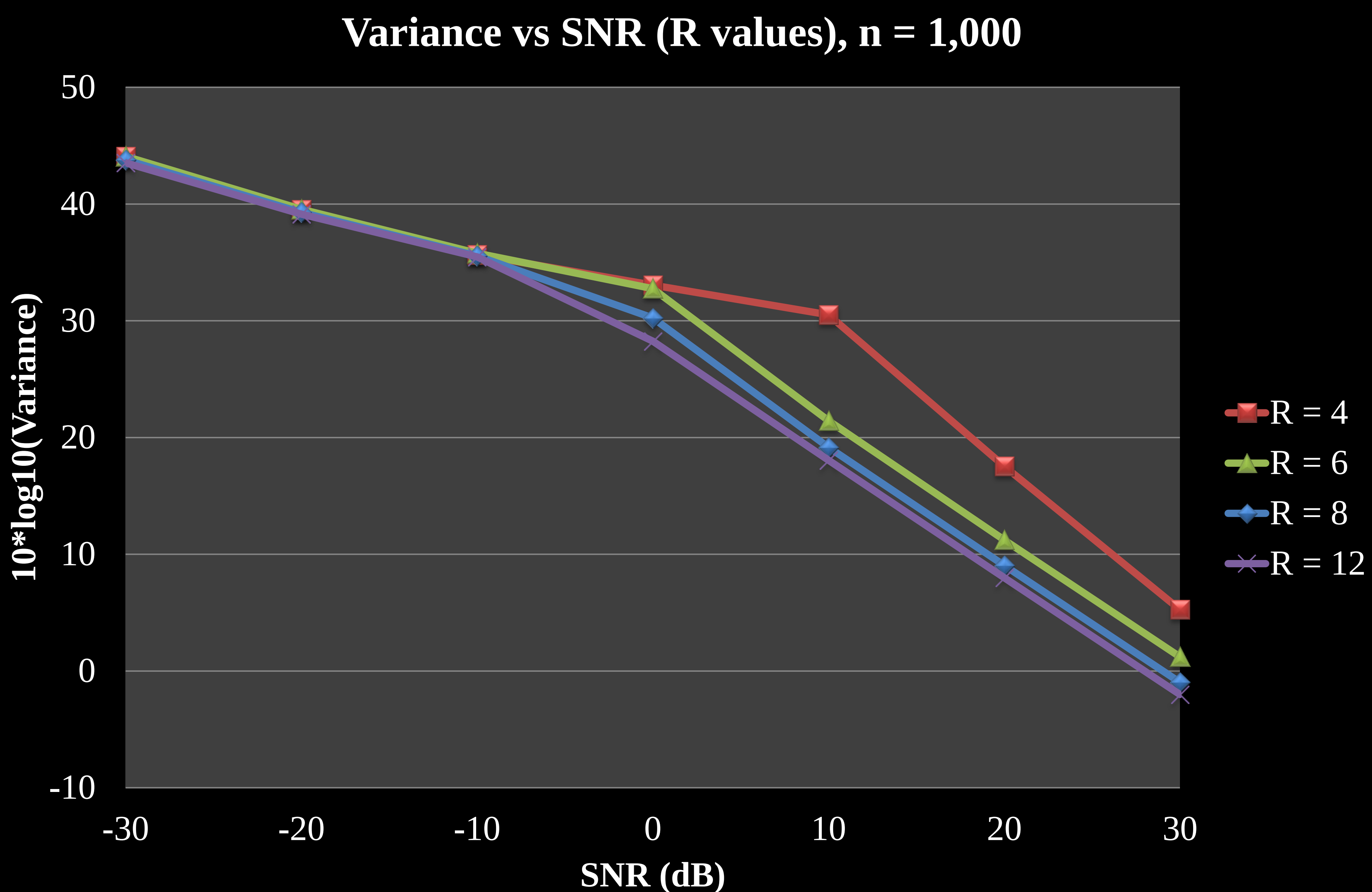
Parameter Study

- Vary R over values:
  - [4, 6, 8, 12]
- Vary Gamma over values:
  - [.5 .90 .95 .99]
- Study the affects on program runtime, number of $LS\_Algorithm()$ iterations, and solution accuracy

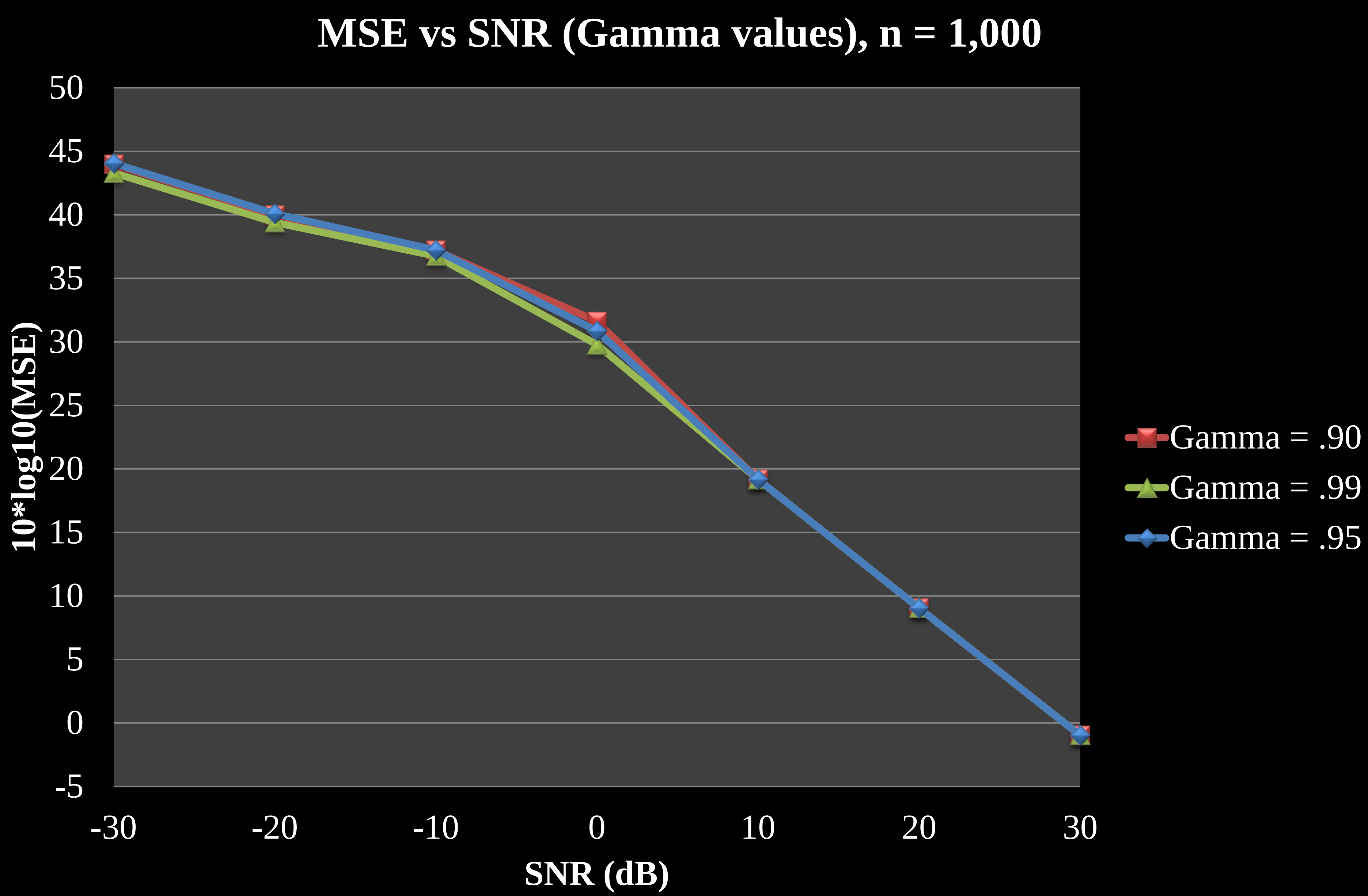

MSE vs SNR (R values), n = 1,000
10*log10(MSE)
50
40
30
20
10
0
-10
-30
-20
-10
0
10
20
30
SNR (dB)
R = 4
R = 6
R = 8
R = 12

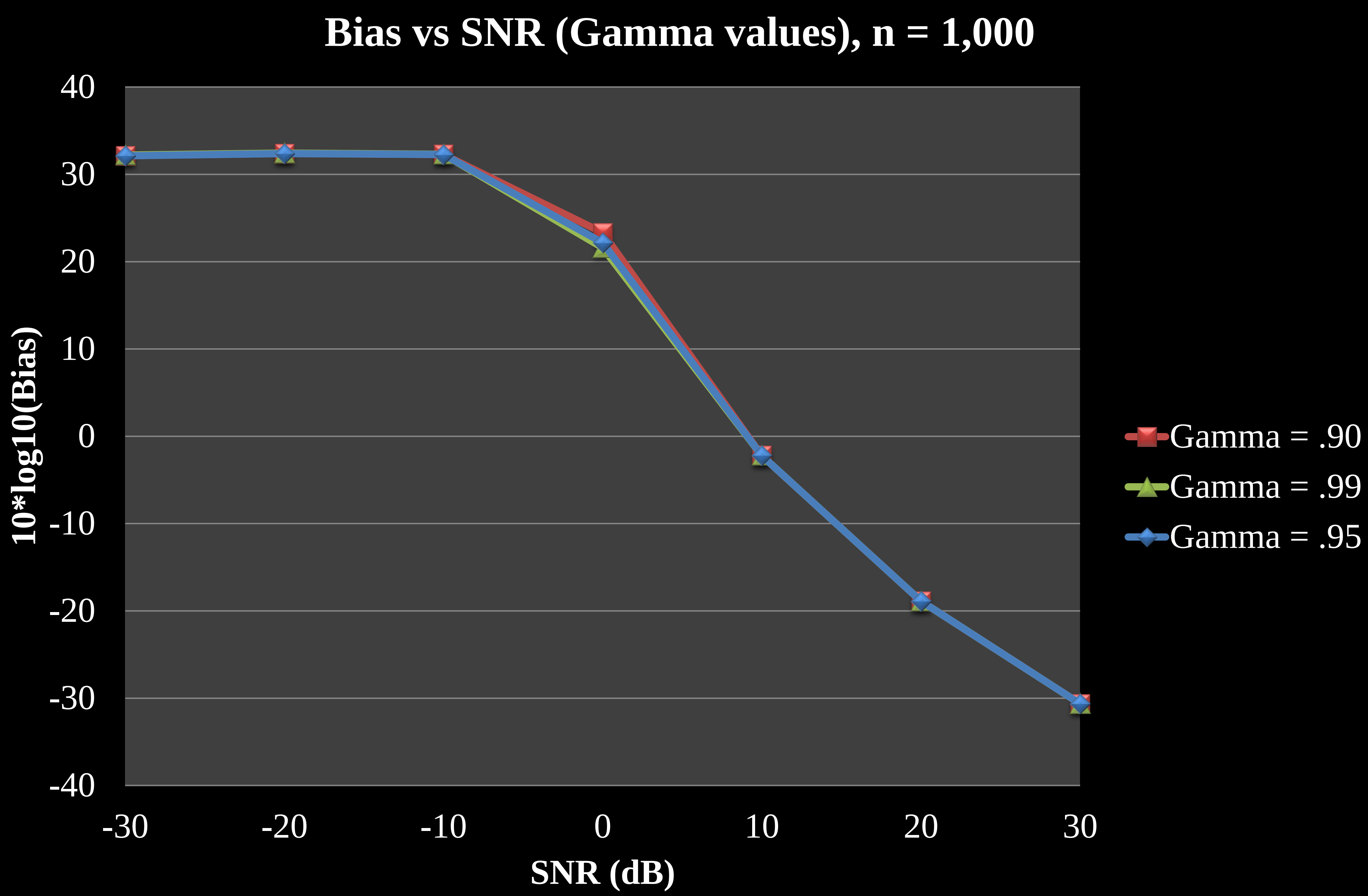


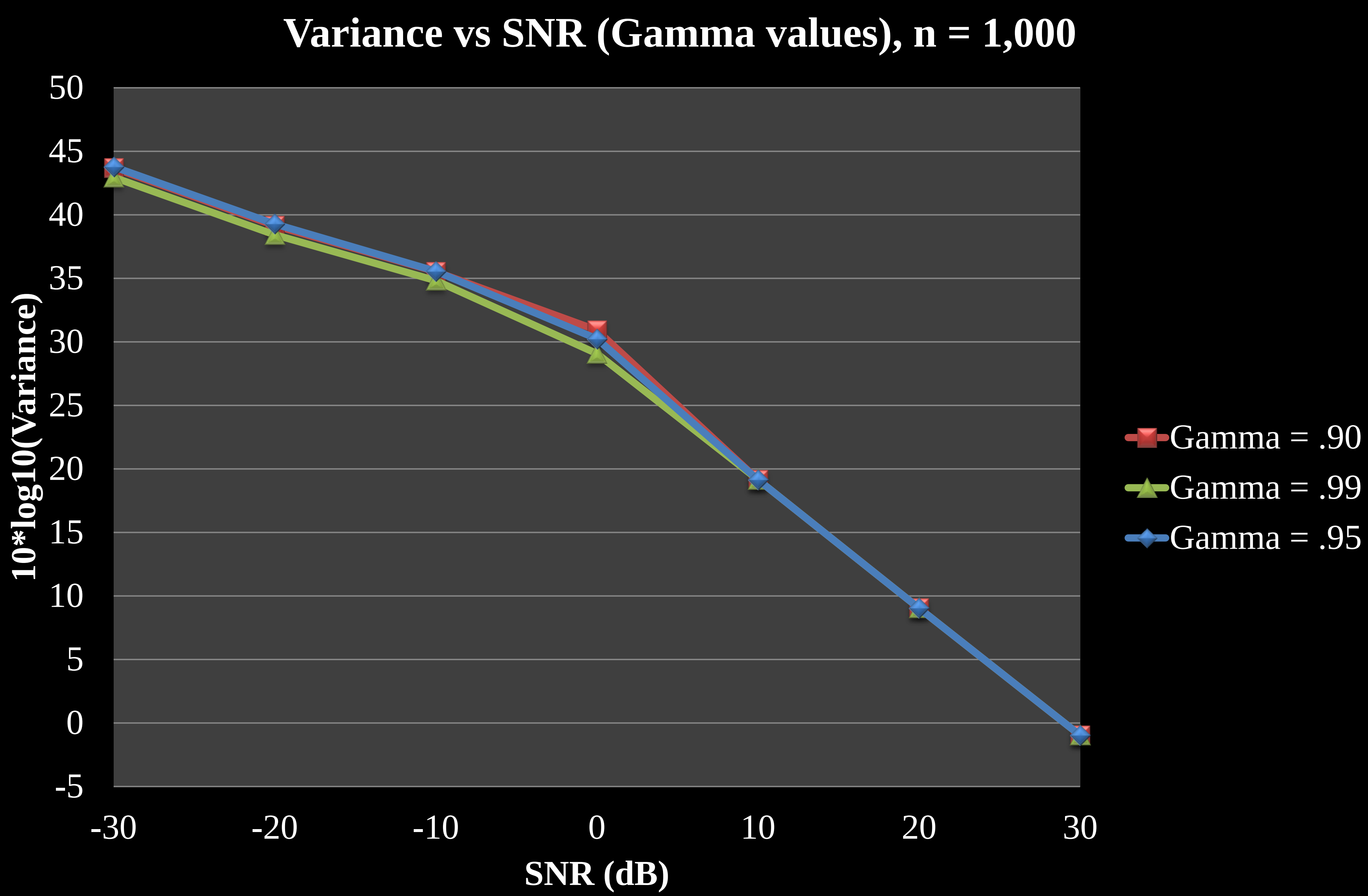
Bias vs SNR (R values), n = 1,000
40
30
20
10
0
-10
-20
-30
-40
10*log10(Bias)
-30
-20
-10
0
10
20
30
SNR (dB)
R = 4
R = 6
R = 8
R = 12



Variance vs SNR (R values), n = 1,000
10*log10(Variance)
50
40
30
20
10
0
-10
-30
-20
-10
0
10
20
30
SNR (dB)
R = 4
R = 6
R = 8
R = 12

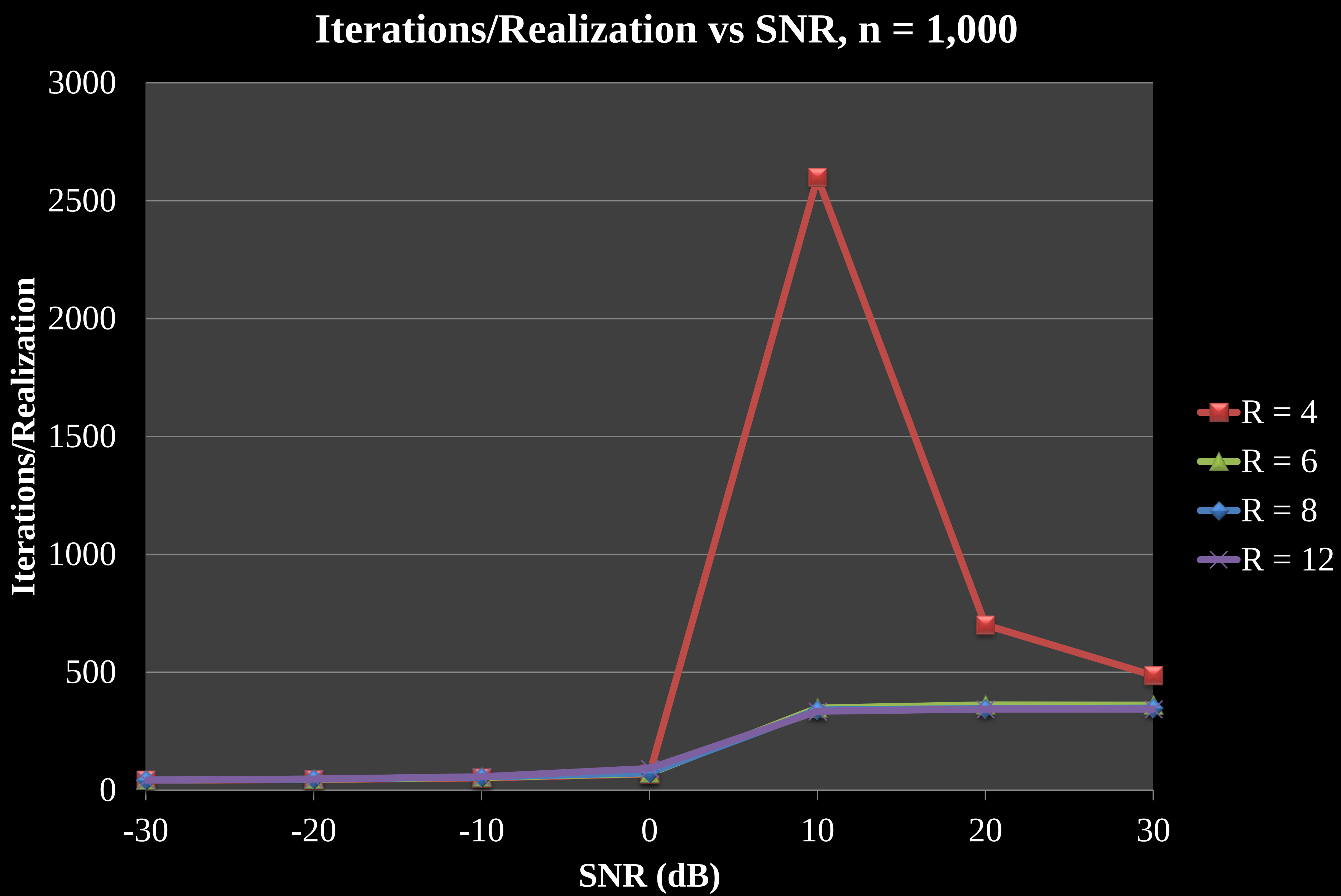

MSE vs SNR (Gamma values), n = 1,000
50
45
40
35
30
25
20
15
10
5
0
-5
10*log10(MSE)
-30
-20
-10
0
10
20
30
SNR (dB)
Gamma = .90
Gamma = .99
Gamma = .95


Bias vs SNR (Gamma values), n = 1,000
10*log10(Bias)
40
30
20
10
0
-10
-20
-30
-40
-30
-20
-10
0
10
20
30
SNR (dB)
Gamma = .90
Gamma = .99
Gamma = .95

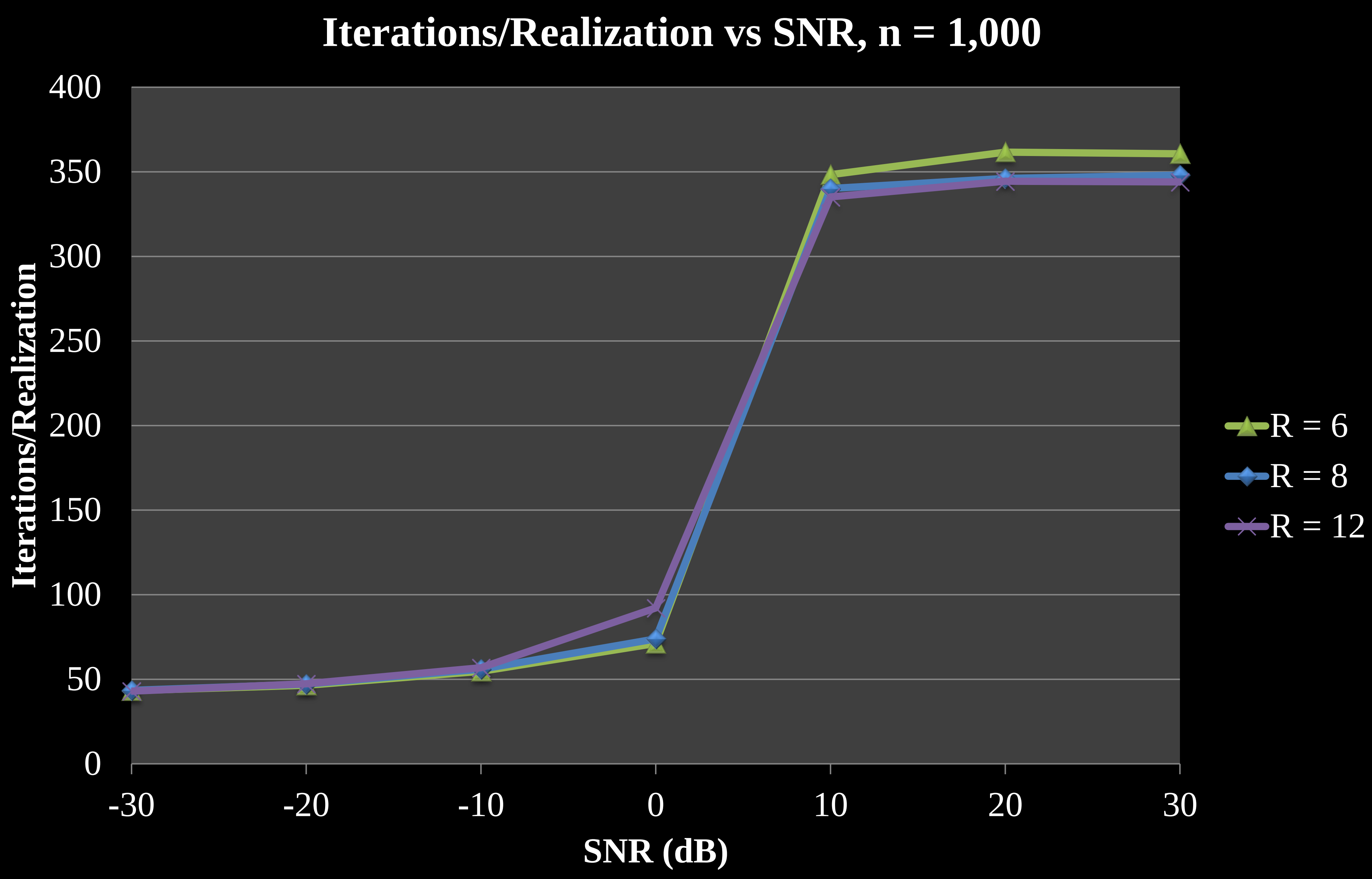

Variance vs SNR (Gamma values), n = 1,000
10*log10(Variance)
50
45
40
35
30
25
20
15
10
5
0
-5
-30
-20
-10
0
10
20
30
SNR (dB)
Gamma = .90
Gamma = .99
Gamma = .95

R vs $LS\_Algorithm$ Iterations

	Iterations/Realization (n = 10,000)			
SNR	**R = 4**	**R = 6**	**R = 8**	**R = 12**
-30	42.6	43.2	43.5	43.0
-20	45.3	46.5	47.1	47.3
-10	52.2	54.7	55.9	56.9
0	69.1	71.2	73.9	92.2
10	2599.0	348.3	340.1	335.2
20	700.9	361.7	346.1	344.4
30	486.6	360.6	348.0	344.0



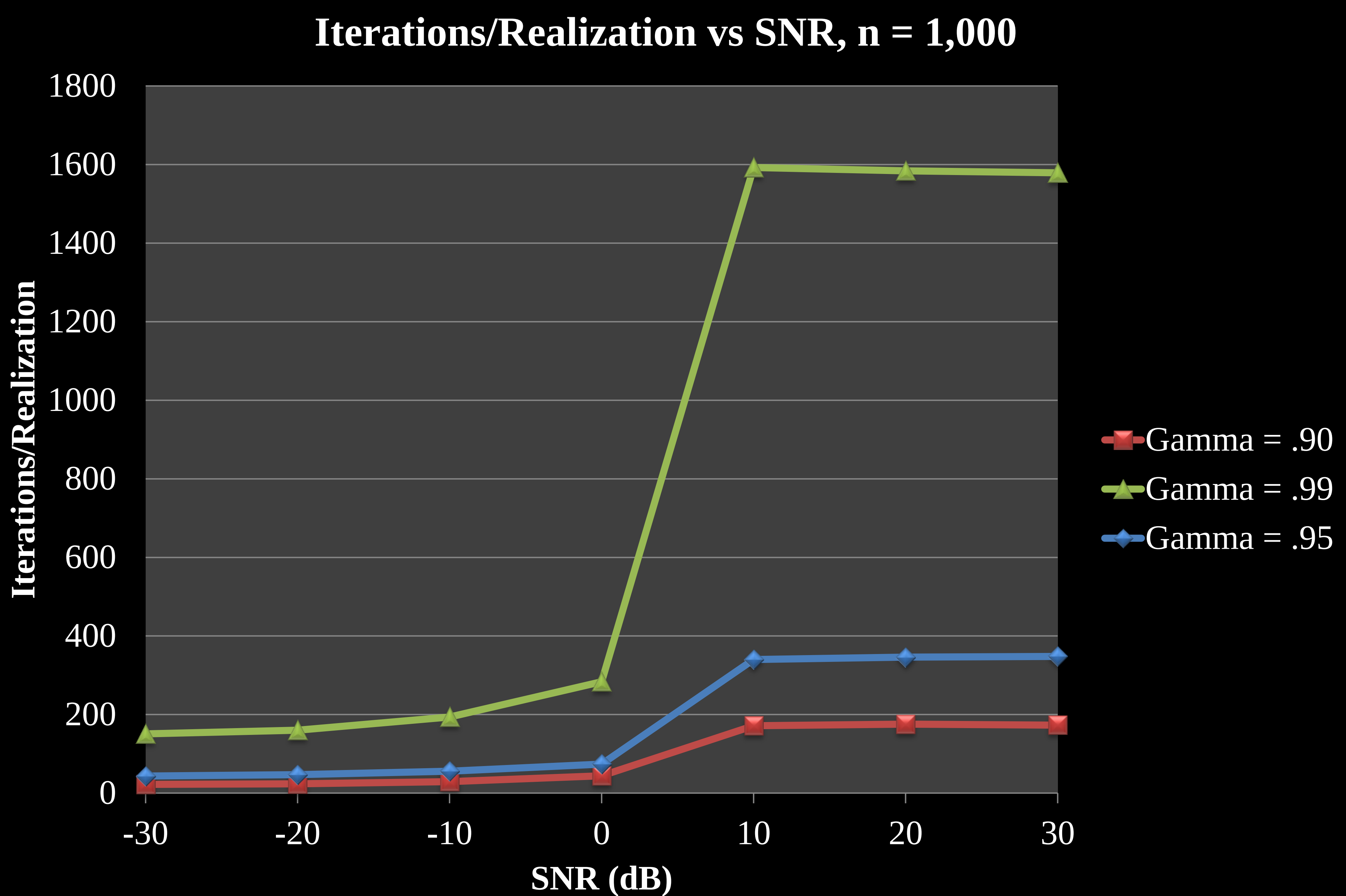
Iterations/Realization vs SNR, n = 1,000
3000
2500
2000
1500
1000
500
0
Iterations/Realization
-30
-20
-10
0
10
20
30
SNR (dB)
R = 4
R = 6
R = 8
R = 12



Iterations/Realization vs SNR, n = 1,000
Iterations/Realization
SNR (dB)
400
350
300
250
200
150
100
50
0
-30
-20
-10
0
10
20
30
R = 6
R = 8
R = 12

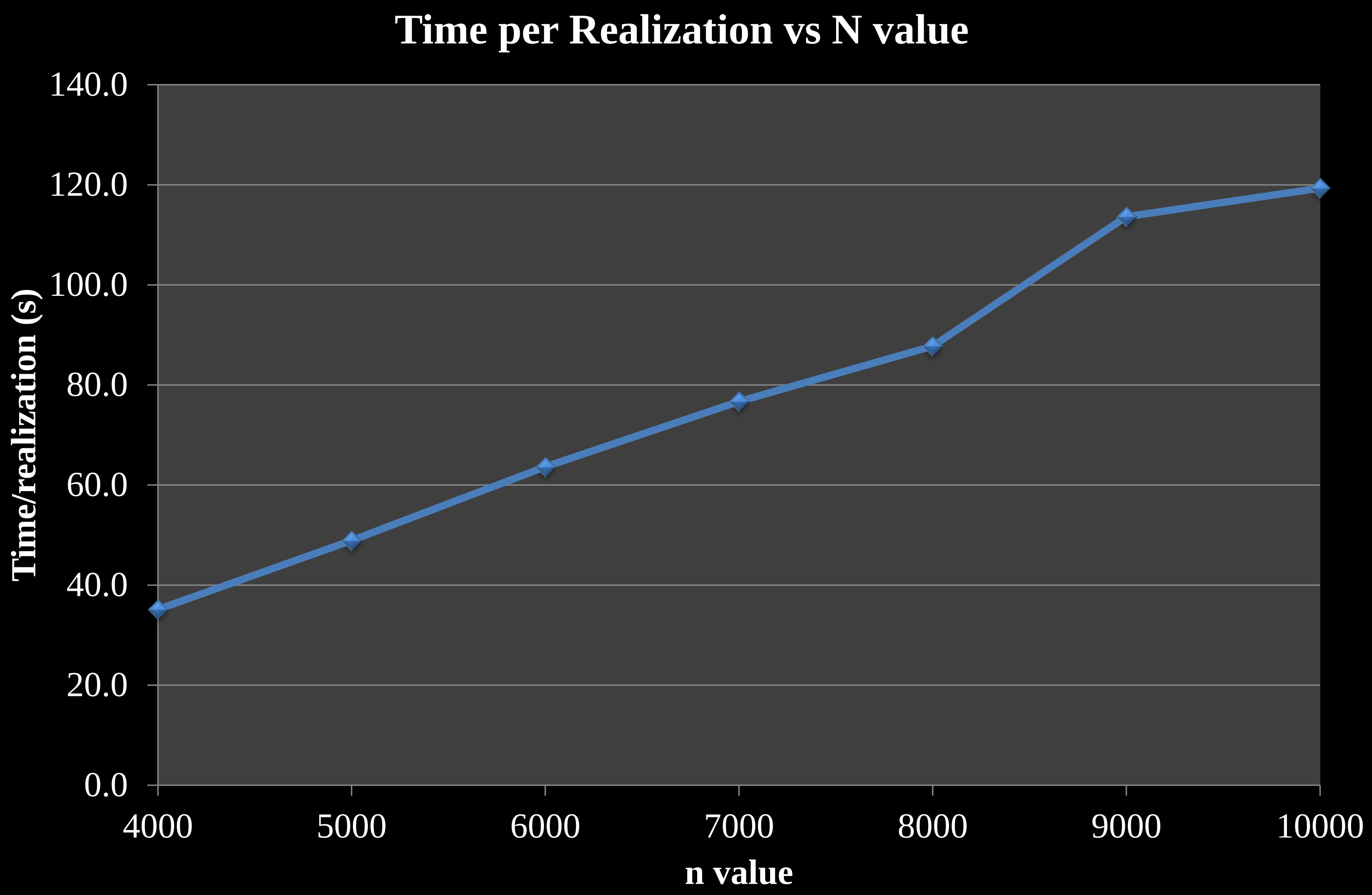
Gamma vs LS_Algorithm Iterations

	Iterations/Realization (n = 10,000)		
SNR	**Gamma = .99**	**Gamma = .95**	**Gamma = .90**
-30	150.9	43.5	22.4
-20	160.3	47.1	23.6
-10	193.7	55.9	29.3
0	283.4	73.9	44.0
10	1592.5	340.1	171.5
20	1584.2	346.1	175.7
30	1579.1	348.0	173.0


Iterations/Realization vs SNR, n = 1,000
Iterations/Realization
1800
1600
1400
1200
1000
800
600
400
200
0
-30
-20
-10
0
10
20
30
SNR (dB)
Gamma = .90
Gamma = .99
Gamma = .95

Computational Complexity

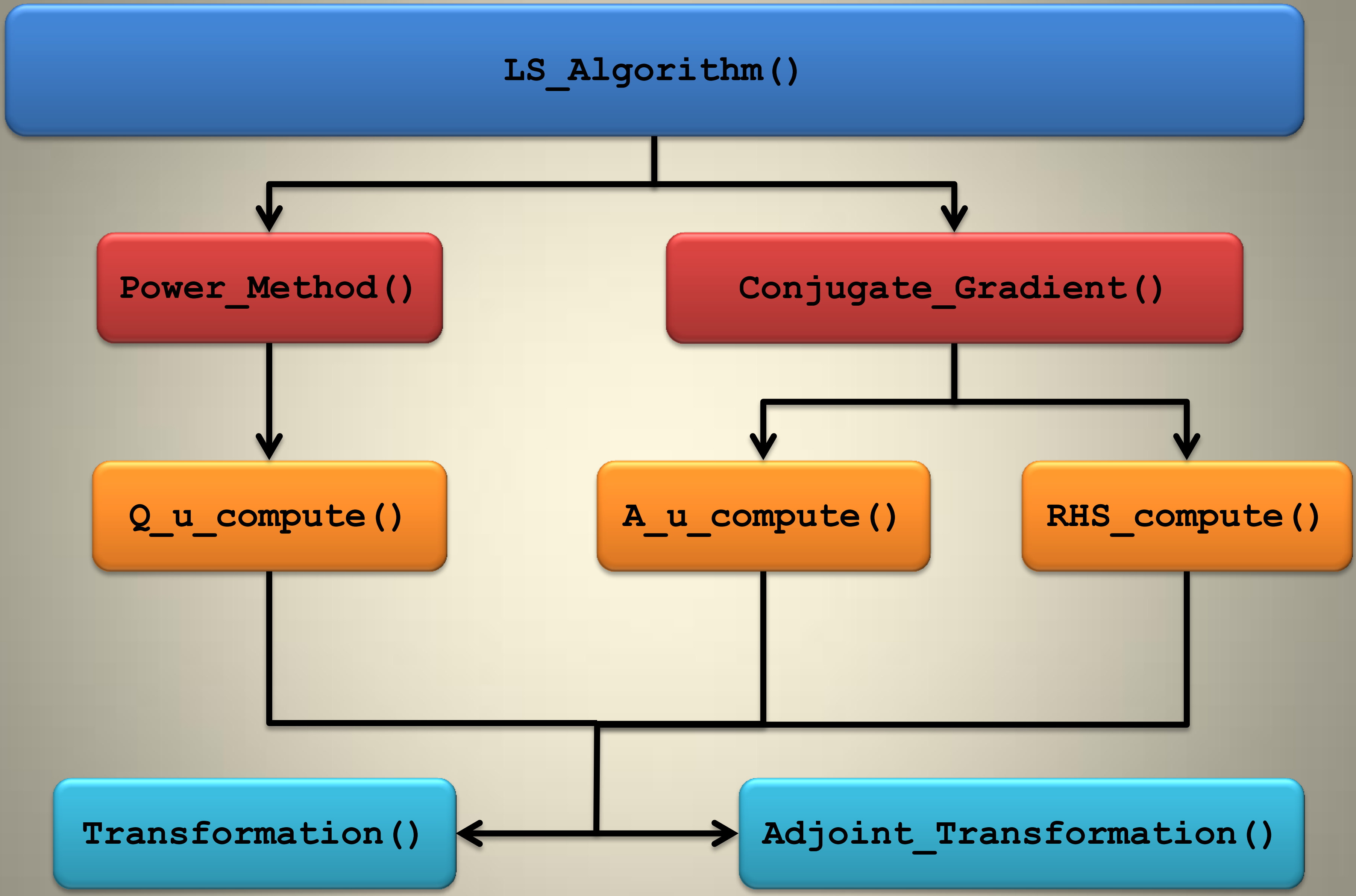
n value	Total Time (s)	# of trials	Time/trial (s)
4000	3514.3	100	35.143
5000	4892.2	100	48.922
6000	6364.3	100	63.643
7000	7669.4	100	76.694
8000	8775.6	100	87.756
9000	11364.8	100	113.648
10000	11934.7	100	119.347



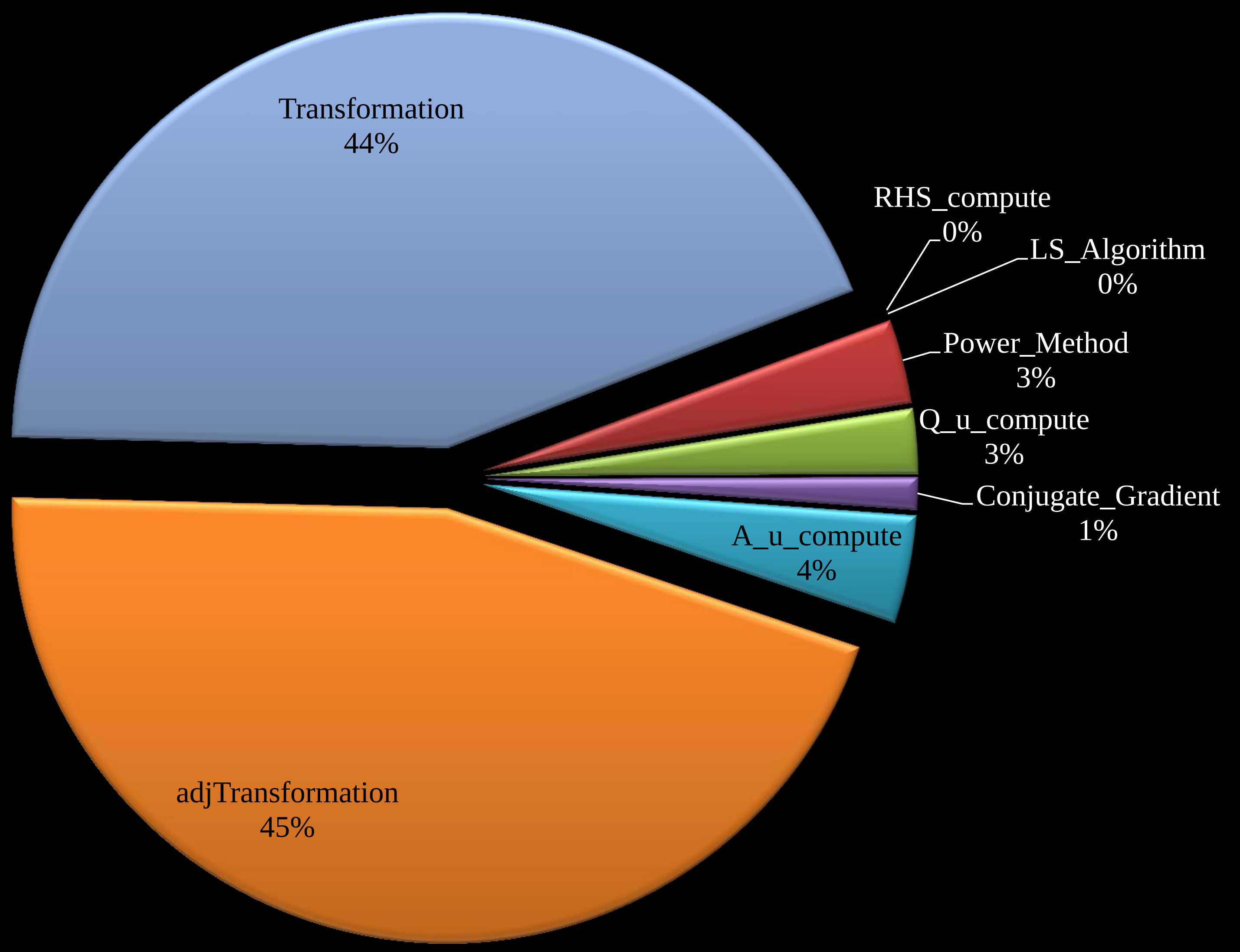
Time per Realization vs N value
Time/realization (s)
140.0
120.0
100.0
80.0
60.0
40.0
20.0
0.0
4000
5000
6000
7000
8000
9000
10000
n value

Computational Complexity (cont.)

- Majority of time is spent in $Transformation()$ and Adjoint_$Transformation()$
- Most time spent executing Fourier and Inverse Fourier transforms.
- Fourier Transform: $O(n \cdot \log(n))$

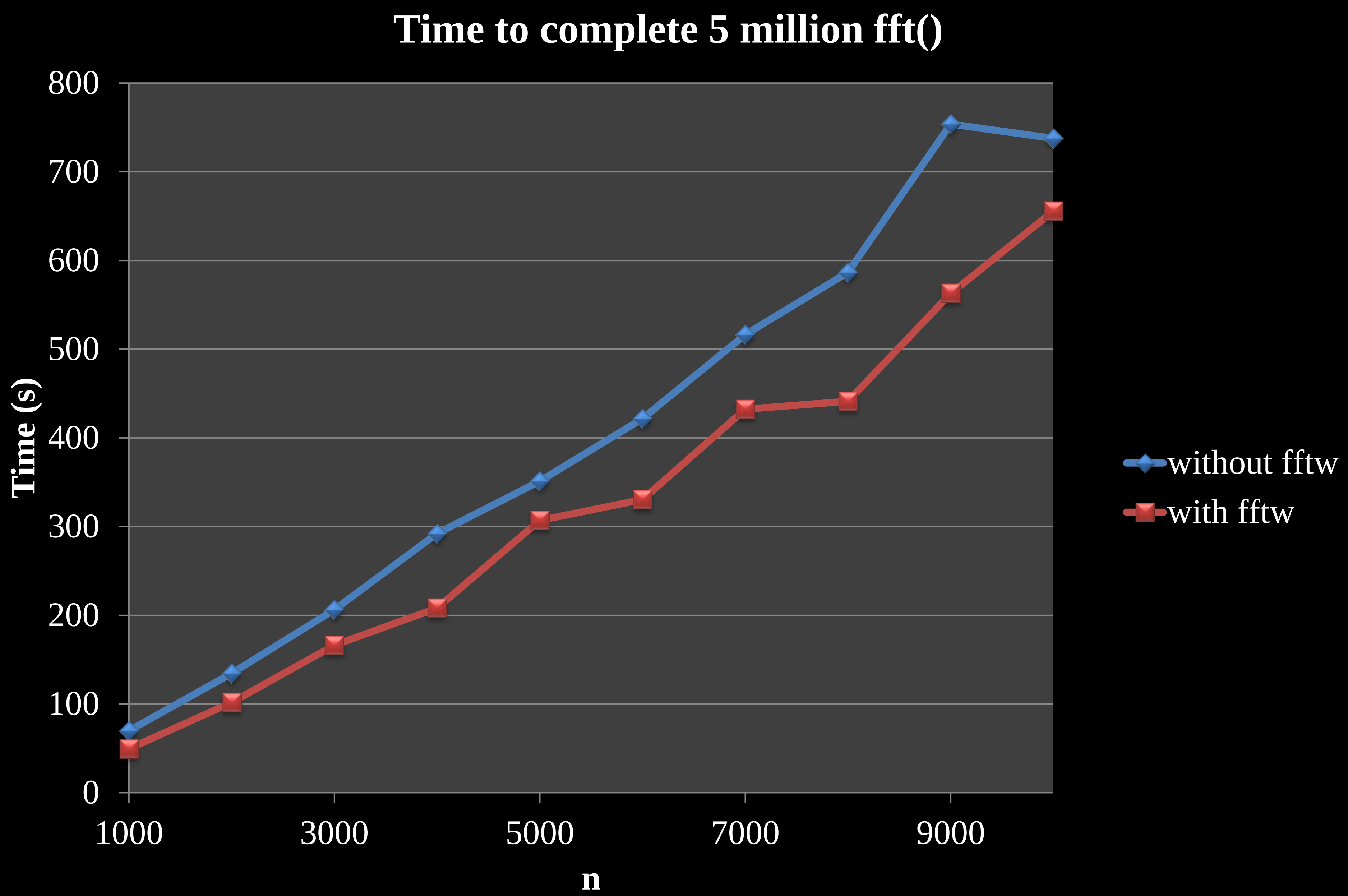

LS_Algorithm()
Power_Method()
Conjugate_Gradient()
Q_u_compute()
A_u_compute()
RHS_compute()
Transformation()
Adjoint_Transformation()

Time Consumption by Function



$fft()$ Speedup

- Use Fastest Fourier Transform in the West
- $fftw('planner','exhaustive')$
- Searches for optimal Fast Fourier Transform algorithm
- Utilizes most efficient implementation for all $fft()$ and $ifft()$ calculations.


Time to complete 5 million fft()
800
700
600
500
400
300
200
100
0
Time (s)
1000
3000
5000
7000
9000
n
without fftw
with fftw

$fft()$ speedup

n	Speedup
1000	1.41
2000	1.32
3000	1.24
4000	1.40
5000	1.14
6000	1.28
7000	1.19
8000	1.33
9000	1.34
10000	1.13

FFTW

- Program results:
  - No speedup after initial implementation
  - $fftw()$ does not get applied to parallel threads
- Solution: call $fftw()$ from each individual thread
  - Single trial results ($n = 1{,}000$, $nnoise = 2{,}000$):
    - Without $fftw()$: 14,072 seconds
    - With $fftw()$: 13,726 seconds
    - Speedup: 1.025

Schedule

October	✓ Post processing framework ✓ Database generation
November	✓ MATLAB implementation of iterative recursive least squares algorithm
December	✓ Validate modules written so far
February	✓ Implement power iteration method ✓ Implement conjugate gradient
By March 15	✓ Validate power iteration and conjugate gradient
March 15 – April 15	▪ Test on synthetic databases ▪ Extract metrics
April 15 – end of semester	▪ Write final report

Deliverables

- Final Report
- Program Code
- Input data sets (input files and weight files)
- Output data
- Output charts and graphs

References

[1] R. Balan, On Signal Reconstruction from Its Spectrogram, Proceedings of the CISS Conference, Princeton, NJ, May 2010.

[2] R. Balan, P. Casazza, D. Edidin, On signal reconstruction without phase, Appl.Comput.Harmon.Anal. 20 (2006), 345-356.

[3] R. Balan, Reconstruction of signals from magnitudes of redundant representations. 2012.

[4] R. Balan, Reconstruction of signals from magnitudes of redundant representations: the complex case. 2013.

[5] Christensen, Ole. "Frames in Finite-dimensional Inner Product Spaces."*Frames and Bases*. Birkhäuser Boston, 2008. 1-32.

[6] Allaire, Grâegoire, and Sidi Mahmoud Kaber. *Numerical linear algebra*. Springer, 2008.

[7] Shewchuk, Jonathan Richard. "An introduction to the conjugate gradient method without the agonizing pain." (1994).