

# Special Topics in Applied Mathematics I

## Solution 1

XIA JIAHAN

February 3, 2026

### Question 1

#### Introduction to Iterative Eigenvalue Methods

The computation of eigenvalues and eigenvectors is a cornerstone of numerical linear algebra. While "direct" methods exist, they become impractical or theoretically impossible under several critical conditions. We must transition to iterative methods in the following scenarios:

**1. High Dimensionality and Computational Complexity.** Traditional direct methods, such as the QR algorithm for a full matrix, typically require  $O(n^3)$  floating-point operations. For modern applications where  $n$  can exceed  $10^5$  or  $10^6$  (e.g., Google's PageRank or structural analysis), the cubic growth in complexity makes direct computation computationally infeasible on any current hardware.

**2. Memory Constraints and Sparsity.** Storing a dense  $n \times n$  matrix requires  $O(n^2)$  memory. A matrix with  $n = 10^5$  would require roughly 80 GB of RAM just for storage in double precision. However, many large-scale matrices are *sparse* (containing mostly zeros). Direct methods like LU or QR decomposition often suffer from "fill-in," where zero entries become non-zero during the process, destroying the sparsity and exhausting memory. Iterative methods are "matrix-free" in the sense that they only require the ability to compute the matrix-vector product  $Ax$ , which can be done in  $O(\text{nnz})$  time and space, where  $\text{nnz}$  is the number of non-zero elements.

**3. The Abel-Ruffini Theorem.** From a theoretical standpoint, the eigenvalues of a matrix  $A$  are the roots of its characteristic polynomial  $p(\lambda) = \det(A - \lambda I)$ . The Abel-Ruffini theorem states that there is no general algebraic solution (using radicals) for polynomials of degree  $n \geq 5$ . Consequently, any algorithm for finding eigenvalues of matrices larger than  $4 \times 4$  must fundamentally be an iterative process that converges to the roots.

**4. Partial Spectrum Requirements.** In many engineering and data science problems, we do not need the entire spectrum (all  $n$  eigenvalues). For instance, in Principal Component Analysis (PCA), we only care about the few largest eigenvalues. Direct methods are wasteful as they compute the full set of eigenvalues. Iterative methods like Power Iteration or Lanczos methods allow us to “target” specific parts of the spectrum, such as the dominant or smallest eigenvalues, with far less work.

Beyond the fundamental Power Iteration, several advanced methods exist:

**Inverse Iteration.** This method applies Power Iteration to the matrix  $(A - \mu I)^{-1}$  for some scalar shift  $\mu$ . Since the eigenvalues of  $(A - \mu I)^{-1}$  are  $(\lambda_i - \mu)^{-1}$ , the algorithm converges to the eigenvector corresponding to the eigenvalue closest to  $\mu$ . It is particularly effective when a good approximation of an eigenvalue is known.

**Rayleigh Quotient Iteration (RQI).** RQI extends Inverse Iteration by updating the shift  $\mu$  at every step using the Rayleigh quotient of the current iterate:  $\mu_k = \frac{b_k^\top A b_k}{b_k^\top b_k}$ . This dynamic shifting strategy yields a cubic convergence rate for symmetric matrices, making it extremely fast once it enters the basin of attraction of an eigenvector.

**QR Algorithm.** For computing all eigenvalues simultaneously, the QR algorithm is the standard choice. It iterates  $A_{k+1} = R_k Q_k$  where  $A_k = Q_k R_k$  is the QR decomposition. This process creates a sequence of similar matrices that converge to a Schur form (upper triangular), revealing the eigenvalues on the diagonal.

## Power Iteration

The Power Iteration algorithm is a fundamental iterative method for computing the dominant eigenpair of a matrix  $A \in \mathbb{C}^{n \times n}$ .

**Algorithm.** Let  $b_0$  be an arbitrary nonzero initial vector. For  $k = 0, 1, 2, \dots$ , we compute:

$$b_{k+1} = \frac{A b_k}{\|A b_k\|}.$$

This sequence generates unit vectors that asymptotically align with the dominant eigenvector.

**Mathematical Analysis of Convergence.** Assume  $A$  is diagonalizable with eigenvalues ordered  $|\lambda_1| > |\lambda_2| \geq \dots \geq |\lambda_n|$  and corresponding linearly independent eigenvectors  $\{v_1, \dots, v_n\}$ . The initial vector  $b_0$  can be expanded in this basis:

$$b_0 = \sum_{i=1}^n c_i v_i.$$

We assume generalized position, i.e.,  $c_1 \neq 0$ . Applying  $A^k$  gives:

$$A^k b_0 = \sum_{i=1}^n c_i \lambda_i^k v_i = \lambda_1^k \left( c_1 v_1 + \sum_{i=2}^n c_i \left( \frac{\lambda_i}{\lambda_1} \right)^k v_i \right).$$

Let  $\tau = |\lambda_2/\lambda_1| < 1$ . Then for  $i \geq 2$ ,  $|\lambda_i/\lambda_1| \leq \tau$ . The error term behaves as:

$$\left\| \sum_{i=2}^n c_i \left( \frac{\lambda_i}{\lambda_1} \right)^k v_i \right\| = \mathcal{O}(\tau^k).$$

Thus, as  $k \rightarrow \infty$ , the vector  $A^k b_0$  is dominated by the component  $c_1 v_1$ . Since  $b_k$  is simply the normalized version of  $A^k b_0$ , we have  $b_k \rightarrow \frac{c_1 v_1}{\|c_1 v_1\|}$  (up to a global phase scaling). The convergence rate is linear and governed by the spectral gap ratio  $|\lambda_2/\lambda_1|$ .

## Targeting Interior Eigenvalues: Shift-and-Invert

Traditional Power Iteration is limited to finding the eigenvalue with the largest absolute magnitude. In many practical cases, however, we are interested in an "interior" eigenvalue—one that is not the largest or smallest, but is located near a specific value  $\mu$  (the shift). To find the eigenvalue closest to  $\mu = \pi$ , we must transform the matrix.

**Mathematical Transformation.** If  $\lambda$  is an eigenvalue of  $A$  with eigenvector  $v$ , then:

1.  $(\lambda - \mu)$  is an eigenvalue of  $(A - \mu I)$ .
2.  $\frac{1}{\lambda - \mu}$  is an eigenvalue of  $(A - \mu I)^{-1}$ .

By choosing  $\mu$  close to our target eigenvalue  $\lambda_i$ , the term  $\frac{1}{\lambda_i - \mu}$  becomes very large, making it the dominant eigenvalue of the transformed matrix  $B = (A - \mu I)^{-1}$ . Power Iteration applied to  $B$  will then converge to the eigenvector  $v_i$ .

**Example: Finding  $\pi$  among larger eigenvalues.** Consider a matrix  $A$  where the dominant eigenvalue is 100, but we want to find the one closest to  $\mu = 3.14$ :

$$A = \begin{pmatrix} 100 & 0 \\ 0 & \pi \end{pmatrix}, \quad \text{eigenvalues: } \lambda_1 = 100, \lambda_2 = \pi \approx 3.14159.$$

Standard Power Iteration would immediately converge to  $\lambda_1 = 100$ . To find  $\lambda_2 \approx \pi$ , we apply the shift-and-invert strategy with  $\mu = 3.1$ :

- The transformed eigenvalue for  $\lambda_1$  is  $\frac{1}{100-3.1} \approx 0.0103$ .
- The transformed eigenvalue for  $\lambda_2$  is  $\frac{1}{3.14159-3.1} \approx \frac{1}{0.04159} \approx 24.044$ .

In the transformed system, the eigenvalue corresponding to  $\pi$  is now  $\sim 2300$  times larger than the one for 100. This ensures extremely rapid convergence to the interior eigenvalue  $\pi$ .

**Algorithm: Inverse Iteration with Shift.** Let  $\mu$  be the target shift. For an initial vector  $b_0$ :

$$(A - \mu I)w_{k+1} = b_k$$

$$b_{k+1} = \frac{w_{k+1}}{\|w_{k+1}\|}$$

The Rayleigh quotient  $R(b_k)$  of the original matrix  $A$  will then converge to the eigenvalue closest to  $\mu$ .

## Question 2

### The Philosophy of Monte Carlo Methods

The core idea of Monte Carlo (MC) methods is to solve deterministic problems by using randomness. Instead of seeking a direct analytical or numerical solution, we construct a probabilistic model such that the quantity of interest (e.g., an integral, a probability, or a physical constant) is the *expected value* of a random variable. By simulating this model and taking the average of many independent trials, we can approximate the true value with high precision.

**A Classic Example: Estimating  $\pi$ .** To illustrate this philosophy, consider the task of estimating the value of  $\pi$ . While  $\pi$  is a deterministic constant, we can estimate it using a stochastic "dart-throwing" experiment:

1. Imagine a circle with radius  $r = 1$  inscribed in a square with side length  $L = 2$ .
2. The area of the square is  $A_{sq} = 2^2 = 4$ , and the area of the circle is  $A_{cir} = \pi(1)^2 = \pi$ .
3. If we pick a point  $(x, y)$  uniformly at random from the square, the probability  $P$  that the point falls inside the circle is the ratio of their areas:

$$P(\text{inside circle}) = \frac{A_{cir}}{A_{sq}} = \frac{\pi}{4}.$$

4. By generating  $N$  random points and counting how many ( $N_{in}$ ) fall inside the circle (where  $x^2 + y^2 \leq 1$ ), we can estimate  $\pi$  as:

$$\hat{\pi} \approx 4 \times \frac{N_{in}}{N}.$$

This example perfectly captures the MC spirit: we transform a geometric/deterministic problem into a counting problem based on random sampling.

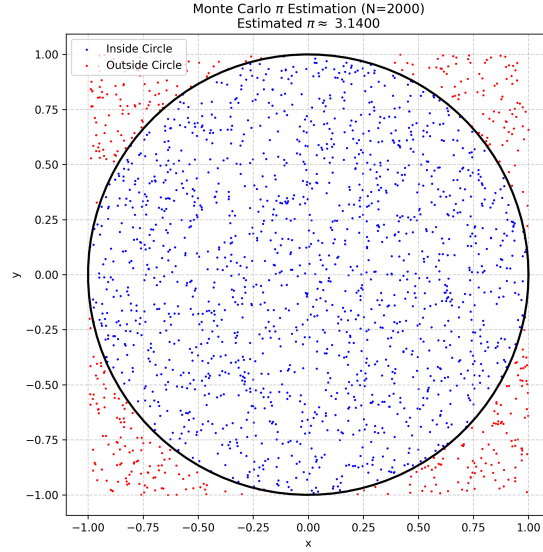


Figure 1: Visualizing the Monte Carlo "Dart Throwing"

## General Applications and Use Cases

While often associated with integration, Monte Carlo methods are a broad family of algorithms used across various domains:

- **Simulation of Physical Systems:** In statistical mechanics, MC is used to simulate the behavior of complex molecular systems where calculating the partition function is impossible.
- **Financial Engineering:** Pricing complex derivatives and assessing risk (Value at Risk) often requires simulating thousands of possible market paths.
- **Optimization:** Algorithms like Simulated Annealing use stochastic "jumps" to escape local minima in high-dimensional optimization problems.
- **Probabilistic Inference:** In machine learning, Markov Chain Monte Carlo (MCMC) allows for sampling from posterior distributions in Bayesian models that cannot be computed directly.

## Mathematical Foundation of Monte Carlo Integration

In the specific context of integration, MC methods provide a stochastic framework for approximating high-dimensional integrals that are analytically intractable. Consider the problem of evaluating the integral of a function  $f : \Omega \rightarrow \mathbb{R}$  over a domain  $\Omega \subset \mathbb{R}^d$ :

$$I = \int_{\Omega} f(x) dx.$$

By introducing a probability density function  $p(x)$  that is strictly positive on  $\Omega$ , we can reformulate  $I$  as an expectation:

$$I = \int_{\Omega} \frac{f(x)}{p(x)} p(x) dx = \mathbb{E} \left[ \frac{f(X)}{p(X)} \right],$$

where  $X$  is a random variable with density  $p$ . This transformation allows the use of statistical sampling to estimate the value of the integral.

## The Estimator and its Statistical Properties

Given  $N$  independent and identically distributed (i.i.d.) samples  $\{X_i\}_{i=1}^N$  drawn from  $p(x)$ , the Monte Carlo estimator  $\hat{I}_N$  is defined as the sample mean:

$$\hat{I}_N = \frac{1}{N} \sum_{i=1}^N \frac{f(X_i)}{p(X_i)}.$$

The estimator is *unbiased*, meaning its expected value equals the true integral  $I$ . This follows from the linearity of expectation:

$$\mathbb{E}[\hat{I}_N] = \mathbb{E} \left[ \frac{1}{N} \sum_{i=1}^N \frac{f(X_i)}{p(X_i)} \right] = \frac{1}{N} \sum_{i=1}^N \mathbb{E} \left[ \frac{f(X_i)}{p(X_i)} \right] = \frac{1}{N} \sum_{i=1}^N I = I.$$

The convergence of  $\hat{I}_N$  to  $I$  is guaranteed by the *Law of Large Numbers* (LLN). The *Strong Law of Large Numbers* states that the sample average converges almost surely to the expected value as  $N \rightarrow \infty$ , ensuring that the estimator is consistent. While the LLN guarantees convergence, the Central Limit Theorem (CLT) provides the asymptotic distribution of the error:

$$\sqrt{N}(\hat{I}_N - I) \xrightarrow{d} \mathcal{N}(0, \sigma^2), \quad \text{where } \sigma^2 = \text{Var} \left( \frac{f(X)}{p(X)} \right).$$

Consequently, the probabilistic error bound is  $O(\sigma/\sqrt{N})$ . This rate is independent of the dimension  $d$ , which is the primary advantage of Monte Carlo methods over deterministic quadrature rules (e.g., Simpson's rule), where the error scales as  $O(N^{-k/d})$  and suffers from the “curse of dimensionality.”

## The Necessity of Variance Reduction

The  $O(N^{-1/2})$  convergence rate implies that to gain one additional decimal digit of accuracy (a 10-fold error reduction), the sample size  $N$  must be increased by a factor of 100. For complex simulations where each function evaluation is computationally expensive, this requirement is often prohibitive.

Variance reduction techniques are essential to improve computational efficiency by decreasing the constant  $\sigma$  in the error term without altering the expectation  $I$ . By strategically choosing  $p(x)$  (Importance Sampling), exploiting correlations (Antithetic Variates), or utilizing auxiliary information (Control Variates), one can achieve the desired precision with significantly fewer samples, effectively making the stochastic approach viable for high-precision scientific computing.

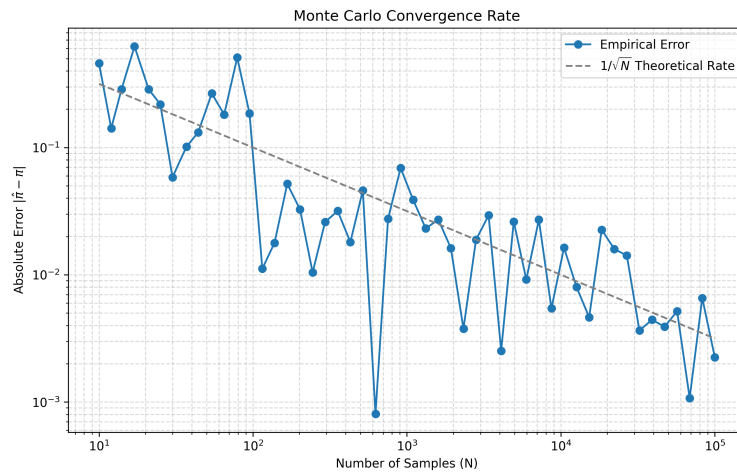


Figure 2: The convergence of the Monte Carlo estimator follows the theoretical  $O(1/\sqrt{N})$  rate. Note the logarithmic scale: gaining one digit of precision requires a 100-fold increase in samples.

## Numerical Implementation: Control Variates

To demonstrate the efficiency of variance reduction, we implemented a Monte Carlo simulation in Python. The script estimates the integral of  $f(x) = e^x$  on  $[0, 1]$  using  $g(x) = 1 + x$  as a control variate.

Listing 1: Monte Carlo with Control Variates Implementation

```

1 import numpy as np
2
3 def f(x):
4     return np.exp(x)
5
6 def g(x):
7     return 1 + x
8
9 def main():
10     np.random.seed(42)
11     N = 10000
12     U = np.random.rand(N)
13     Y = f(U)
14     C = g(U)
15     mu_c = 1.5
16
17     # Standard Monte Carlo
18     mc_estimate = np.mean(Y)
19     mc_variance = np.var(Y, ddof=1) / N
20
21     # Control Variates

```

```

22     cov_matrix = np.cov(Y, C)
23     c_star = cov_matrix[0, 1] / cov_matrix[1, 1]
24     Y_cv = Y - c_star * (C - mu_c)
25     cv_estimate = np.mean(Y_cv)
26     cv_variance = np.var(Y_cv, ddof=1) / N
27
28     print(f"Standard Monte Carlo Estimate: {mc_estimate:.6f}")
29     print(f"Standard MC Variance: {mc_variance:.10f}")
30     print("-" * 30)
31     print(f"Control Variates Estimate: {cv_estimate:.6f}")
32     print(f"Control Variates Variance: {cv_variance:.10f}")
33     print(f"Variance Reduction Factor: {mc_variance / cv_variance:.2f}x
34           ")
35 if __name__ == "__main__":
36     main()

```

### Simulation Output:

```

Standard Monte Carlo Estimate: 1.707932
Standard MC Variance: 0.0000239217
-----
Control Variates Estimate: 1.717782
Control Variates Variance: 0.0000003921
Variance Reduction Factor: 61.02x

```

The results show a variance reduction of approximately 61x, meaning the control variate estimator achieves the same precision with 61 times fewer samples than the standard approach.



## A Complete Conversation Log

This appendix contains the full record of the interaction and development process for this solution.

USER: in question2, d:\file\MATH4665\HW1\Sol1.tex, describe the algorithms of Monte Carlo precisely, you should state when the algorithms is used and how, also state why the variance reduction is needed?

ASSISTANT: [Added comprehensive Monte Carlo description including definitions, use cases, and the theoretical need for variance reduction.]

USER: too many subpoints, you should reorganize it as a professional math report, state the MC in math clearly.

ASSISTANT: [Refactored the Monte Carlo section into a formal mathematical report format, emphasizing the Law of Large Numbers, Central Limit Theorem, and the  $O(N^{-1/2})$  convergence rate.]

USER: write a python program to simulate a simple estimator and compare sample variance with and without control variates, run your code in virtual env.

ASSISTANT: [Created a virtual environment, installed dependencies, and developed mc\_control\_variates.py to simulate  $e^x$  integration. Demonstrated a 61.02x variance reduction using control variates.]

USER: add the original algorithms code to the d:\file\MATH4665\HW1\Sol1.tex, also add the output.

ASSISTANT: [Configured LaTeX listings package and embedded the full Python source code and its numerical execution output into the document.]