Special Topics in Applied Mathematics I
Solution 1

XIA JIAHAN

February 3, 2026

Question 1

Introduction to Iterative Eigenvalue Methods

Direct methods for computing eigenvalues (based on characteristic polynomials) differ from iterative methods which are necessary for large, sparse matrices where $O(n^3)$ operations are prohibitive. Iterative algorithms typically rely on repeated matrix-vector multiplications to approximate eigenpairs. 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 μ. 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 μ. 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 μ 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 $k = 0, 1, 2, \ldots$, we compute:

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

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

Mathematical Analysis of Convergence. Assume A is diagonalizable with eigenvalues ordered $|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n|$ and corresponding linearly independent eigenvectors $\{v_1, \ldots, v_n\}$. 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 $k \to \infty$, 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 $b_k \to \frac{c_1 v_1}{\|c_1 v_1\|}$ (up to a global phase scaling). The convergence rate is linear and governed by the spectral gap ratio $|\lambda_2/\lambda_1|$.

Estimating the eigenvalue: Rayleigh quotient

Once we have an approximate eigenvector x, a natural estimate of its eigenvalue is the *Rayleigh quotient*

$$R(x) = \frac{x^\top A x}{x^\top x}.$$

If x is exactly an eigenvector v, then $R(v) = \lambda$ because $v^\top A v = v^\top (\lambda v) = \lambda\, v^\top v$. When x is close to an eigenvector, $R(x)$ typically gives a good scalar approximation of the associated eigenvalue, so one can monitor the sequence $R(b_k)$ during power iteration.

Example: Consider a diagonal matrix with an irrational eigenvalue:

$$M = \begin{pmatrix} \pi & 0 \\ 0 & 1 \end{pmatrix}, \quad \lambda_1 = \pi \approx 3.14159.$$

Let's estimate λ_1 using power iteration starting with $x_0 = (1, 1)^\top$. The Rayleigh quotient $R(x_k)$ provides the estimate at each step.

- **Step 0:** $x_0 = (1, 1)^\top$.

$$R(x_0) = \frac{1 \cdot \pi + 1 \cdot 1}{1 + 1} = \frac{\pi + 1}{2} \approx 2.0708.$$

- **Step 1:** $x_1 = Mx_0 = (\pi, 1)^\top$.

$$R(x_1) = \frac{\pi^3 + 1}{\pi^2 + 1} \approx \frac{31.006 + 1}{9.870 + 1} \approx 2.9446.$$

- **Step 2:** $x_2 = Mx_1 = (\pi^2, 1)^\top$.

$$R(x_2) = \frac{\pi^5 + 1}{\pi^4 + 1} \approx \frac{306.02 + 1}{97.41 + 1} \approx 3.1198.$$

- **Step 3:** $x_3 = Mx_2 = (\pi^3, 1)^\top$.

$$R(x_3) = \frac{\pi^7 + 1}{\pi^6 + 1} \approx \frac{3020.29 + 1}{961.39 + 1} \approx 3.1394.$$

After three steps, the estimate 3.1394 is very close to $\pi \approx 3.1416$.

Question 2

Mathematical Foundation of Monte Carlo Integration

Monte Carlo (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 $f : \Omega \to \mathbb{R}$ 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 Ω, 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 $N \to \infty$, 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 σ 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.

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

```python
import numpy as np

def f(x):
    return np.exp(x)

def g(x):
    return 1 + x

def main():
    np.random.seed(42)
    N = 10000
    U = np.random.rand(N)
    Y = f(U)
    C = g(U)
    mu_c = 1.5

    # Standard Monte Carlo
    mc_estimate = np.mean(Y)
    mc_variance = np.var(Y, ddof=1) / N

    # Control Variates
    cov_matrix = np.cov(Y, C)
    c_star = cov_matrix[0, 1] / cov_matrix[1, 1]
    Y_cv = Y - c_star * (C - mu_c)
    cv_estimate = np.mean(Y_cv)
    cv_variance = np.var(Y_cv, ddof=1) / N

    print(f"Standard Monte Carlo Estimate: {mc_estimate:.6f}")
    print(f"Standard MC Variance: {mc_variance:.10f}")
    print("-" * 30)
    print(f"Control Variates Estimate: {cv_estimate:.6f}")
    print(f"Control Variates Variance: {cv_variance:.10f}")
    print(f"Variance Reduction Factor: {mc_variance / cv_variance:.2f}x
        ")

if __name__ == "__main__":
    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.]
```