Day 14: Computing Eigenvalues in Practice

From Theory to Implementation: Algorithms Powering Modern AI

1. The Computational Challenge

While eigenvalues provide deep insight into matrix behavior, computing them for large matrices presents significant challenges:

- Solving $\det(A - \lambda I) = 0$ is computationally expensive ($O(n!)$ for an $n \times n$ matrix).
- For large matrices (common in AI), we need efficient, approximate methods.
- Often, we only need a subset of eigenvalues (largest/smallest few) rather than the full spectrum.

This has led to the development of sophisticated numerical algorithms that leverage matrix structure and iterative approaches.

2. Power Iteration: Finding the Dominant Eigenvector

The Power Iteration method is one of the simplest algorithms for finding the eigenvector corresponding to the largest eigenvalue (in magnitude).

Power Iteration Algorithm

1. Choose a random initial vector $\mathbf{b}_0$ with $\|\mathbf{b}_0\| = 1$
2. For $k = 1, 2, \ldots$ until convergence:

$$\mathbf{v}_k = A\mathbf{b}_{k-1}$$
$$\mathbf{b}_k = \frac{\mathbf{v}_k}{\|\mathbf{v}_k\|}$$
$$\lambda_k = \mathbf{b}_k^T A \mathbf{b}_k$$

3. Return $(\lambda_k, \mathbf{b}_k)$ as approximate dominant eigenpair

Convergence: The error decreases geometrically as $O(|\lambda_2/\lambda_1|^k)$, where λ_1 is the dominant eigenvalue and λ_2 is the second largest. This can be slow if $|\lambda_2/\lambda_1| \approx 1$.

3. Extensions to Power Iteration

- **Inverse Iteration:** For finding eigenvalues closest to a specific value μ, use $(A - \mu I)^{-1}$ instead of A.

- **Rayleigh Quotient Iteration:** An accelerated version that updates μ at each step with the current eigenvalue estimate.
- **Orthogonal Iteration:** Extends power iteration to find multiple eigenvectors simultaneously.

4. The QR Algorithm: The Workhorse for Dense Problems

For dense matrices, the QR algorithm is the standard method for computing all eigenvalues. It's based on repeatedly applying QR decomposition:

QR Algorithm (Basic Version)

1. Set $A_0 = A$
2. For $k = 1, 2, \ldots$ until convergence:

$$\begin{aligned} Q_k R_k &= A_{k-1} \quad \text{(QR decomposition)} \\ A_k &= R_k Q_k \end{aligned}$$

3. The matrices A_k converge to Schur form, with eigenvalues on the diagonal

Enhancements: Practical implementations use several optimizations:

- **Hessenberg reduction:** First reduce A to upper Hessenberg form to reduce computational cost.
- **Shifts:** Use shift strategies (Wilkinson shift) to accelerate convergence.
- **Deflation:** Once an eigenvalue converges, continue with a smaller submatrix.

The overall complexity is $O(n^3)$, making it suitable for medium-sized matrices but prohibitive for very large ones.

5. Krylov Subspace Methods: For Large Sparse Matrices

For the massive sparse matrices common in AI (graph Laplacians, Hessians), we use Krylov subspace methods:

Lanczos Algorithm (Symmetric Matrices)

- Builds an orthonormal basis for the Krylov subspace $\mathcal{K}_m(A, v) = \text{span}\{v, Av, A^2v, \ldots, A^{m-1}v\}$
- Projects A onto this subspace to form a tridiagonal matrix T_m
- The eigenvalues of T_m approximate those of A
- Particularly efficient for finding extreme (largest/smallest) eigenvalues

For non-symmetric matrices, the Arnoldi iteration generalizes this approach, producing an upper Hessenberg matrix instead of a tridiagonal one.

6. Modern Applications in AI

Where These Algorithms Power AI

- **Principal Component Analysis (PCA):** Truncated SVD (which computes singular values/eigenvalues) uses Lanczos methods for large datasets. Libraries like Facebook's PCA implementation and sklearn.decomposition.TruncatedSVD use these techniques.
- **PageRank:** The original Google algorithm is essentially power iteration on the web graph matrix.
- **Recommender Systems:** Eigenvalue methods help in dimensionality reduction for collaborative filtering.
- **Graph Neural Networks:** Spectral graph convolutions rely on efficient computation of graph Laplacian eigenvectors.
- **Deep Learning Optimization:** Second-order optimization methods (like K-FAC) use Lanczos to approximate the inverse Hessian.

7. Practical Implementation and Libraries

In practice, AI practitioners rarely implement these algorithms from scratch. Key libraries include:

- **LAPACK:** Implements the QR algorithm for dense matrices (used by NumPy, SciPy).
- **ARPACK:** Implements the Arnoldi and Lanczos methods for sparse matrices.

- **TRLAN:** Thick-restart Lanczos method for large-scale eigenvalue problems.
- **CuSOLVER:** NVIDIA's GPU-accelerated eigensolver library.

8. Example: Power Iteration Step-by-Step

Let's apply power iteration to $A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$ with initial vector $\mathbf{b}_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$:

$$\text{Step 1:} \quad \mathbf{v}_1 = A\mathbf{b}_0 = \begin{bmatrix} 2 \\ 1 \end{bmatrix}, \quad \mathbf{b}_1 = \frac{\mathbf{v}_1}{\|\mathbf{v}_1\|} \approx \begin{bmatrix} 0.894 \\ 0.447 \end{bmatrix}, \quad \lambda_1 \approx \begin{bmatrix} 0.894 & 0.447 \end{bmatrix} \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} 0.894 \\ 0.447 \end{bmatrix} \approx 2.8$$

$$\text{Step 2:} \quad \mathbf{v}_2 = A\mathbf{b}_1 \approx \begin{bmatrix} 2.236 \\ 1.788 \end{bmatrix}, \quad \mathbf{b}_2 \approx \begin{bmatrix} 0.781 \\ 0.625 \end{bmatrix}, \quad \lambda_2 \approx 2.95$$

$$\text{Step 5:} \quad \mathbf{b}_5 \approx \begin{bmatrix} 0.709 \\ 0.709 \end{bmatrix}, \quad \lambda_5 \approx 2.999$$

$$\text{Converged:} \quad \mathbf{b} \approx \begin{bmatrix} 0.707 \\ 0.707 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad \lambda = 3$$

Key Takeaway

Practical eigenvalue computation requires sophisticated algorithms tailored to matrix properties (size, sparsity, symmetry). While power iteration provides a simple introduction, industrial-strength applications rely on the QR algorithm for dense matrices and Krylov subspace methods (Lanczos, Arnoldi) for large sparse problems. These algorithms form the computational backbone of many AI techniques, from PCA to PageRank to graph neural networks, enabling us to extract deep insights from high-dimensional data.