Report #2

Brett Bono

November 2024

1 Least Squares:

1.1 History:

In the 15th to 17th century, astronomers and geodesist(those in the field of geodesy), sought to provide solutions to the challenges of navigating the Earth's oceans during the golden age of exploration. During these times many scientific observations of celestial bodies and those used for navigation, were taken and recorded. Scientist would try to record the same results under the same conditions but would have different results. In the 18th century, the first clear and concise exposition of the method of least squares was published by Adrien Marie Legendre, a French Mathematician. From Legendre the rest of the idea of least squares was built.

1.2 Over Determined Systems:

1. An **Overdetermined system** is:

   A system of equations where there are more equations than unknowns

   Over Determined systems are almost always inconsistent when constructed with random coefficients. In matrix form $Ax = b$, an example of an over determined system is:

$$\begin{bmatrix} 2 & 1 \\ -3 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} -1 \\ -2 \\ 1 \end{bmatrix}$$

   Again, almost always, overdetermined systems are inconsistent. However, with these systems we can approximate a solution. This approximation is called a **Least Squares Approximation**.

   Using the Overdetermined system, we can obtain a residual, R. Where

$$R = b - Ax$$

R cannot be 0 if the system is overdetermined. However, with the right pick in our vector x, R can be made very small.

We can measure the smallness of R with the 2-Norm:

$$||Ax - b||_2 \text{ ; such that } x \in \mathbb{R}^n$$

1.3 Polynomial Interpolation:

We will review polynomial interpolation in this section and then compare it to a polynomial least squares fitting In the next section.

We can interpolate data by creating a polynomial.

1. Given m distinct inputs, $x_1, x_2, ..., x_m$ with corresponding outputs, $y_1, y_2, ...y_m$, respectively. There exists a unique polynomial that goes through these points.

   The polynomial is at most degree $m - 1$ and has the form:

   $$p(x) = c_0 + c_1 x + c_2 x^2 + ...c_{m-1} x^{m-1}$$

   below, $p(x)$ can also be written discretely by the form $Ax = b$ by using a Vandermonde matrix.

$$\begin{bmatrix} 1 & x_1 & x_1^2 & ... & x_1^{m-1} \\ 1 & x_2 & x_2^2 & ... & x_2^{m-1} \\ 1 & x_3 & x_3^3 & ... & x_3^{m-1} \\ ... & ... & ... & ... & x_4^{m-1} \\ 1 & x_m & x_m^2 & ... & x_m^{m-1} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ ... \\ c_{m-1} \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ ... \\ y_m \end{bmatrix}$$

   This gives us a polynomial interpolate that goes through each point. This interpolate is not perfect. This polynomial exhibits large oscillations near the endpoints, as seen in **Figure 1.1** and **Figure 1.2**

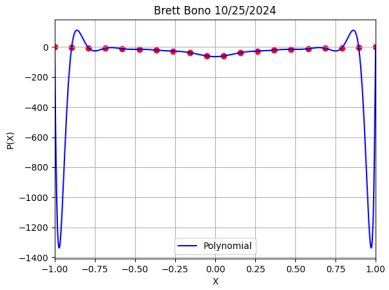

Figure 1.1: Polynomial interpolate with n = 20, degree 19

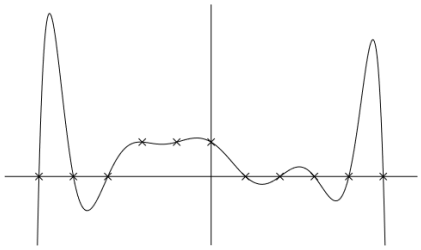

Figure 1.2: Polynomial interpolate with n = 11, degree 10

These oscillations can be reduced by using **Chebyshev points** in the interval [0,1]. Such that we utilize a nonuniform set of interpolation points.

The Chebyshev points follow:

$$x_k = \frac{b-a}{2} \times \cos\left(\frac{\pi(2k+1)}{2(n)}\right) + \frac{b+a}{2}$$

Such that $[a, b] \in [-1, 1]$ and n = number of points

By using Chebyshev points we can reduce the large oscillations in **Figure 1.1** as seen in **Figure 1.3** (not to scale)

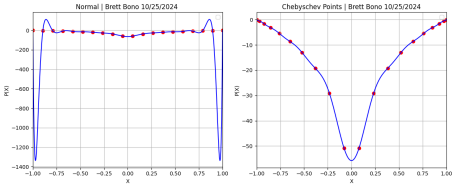

Figure 1.3: Figure 1.1 (Left) $\implies$ Chebyshev (Right)

1.4 Polynomial Least Squares Fitting:

As seen from **section 1.3**, we can apply polynomial interpolation to given data points. However, by applying the least squares approximations seen in **Section 1.2** we can often obtain better results.

By creating a polynomial using the same methods as **Section 1.3** but reducing the degree such that we obtain a overdetermined system, we can reduce the volatility of our interpolation function.

Again, given m distinct inputs, $x_1, x_2, ..., x_m$ with corresponding outputs, $y_1, y_2, ...y_m$

lets now use a $n-1$ degree polynomial such that $n < m$.

This polynomial is a least squares fit if it minimizes the sum of the squares of the deviation from the data. As shown below, using mathematical notation.

$$\Sigma_{i=1}^{m} |p(x_i) - y_i|^2$$

So by least squares we have:

$$\begin{bmatrix} 1 & x_1 & ... & x_1^{n-1} \\ 1 & x_2 & ... & x_2^{n-1} \\ 1 & x_3 & ... & x_3^{n-1} \\ ... & ... & ... & x_4^{n-1} \\ 1 & x_m & ... & x_m^{n-1} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ ... \\ c_{n-1} \end{bmatrix} \approx \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ ... \\ y_m \end{bmatrix}$$

As seen in Figure 1.4, the least squares polynomial minimizes the sum of the squares of the deviation from the data.

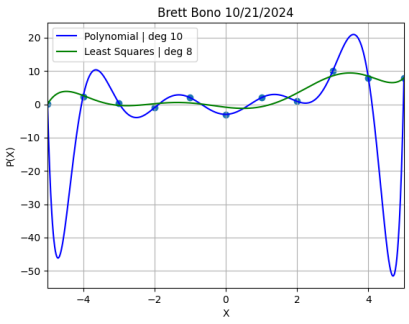

Figure 1.4: Polynomial Interpolation vs Least Squares Fitting

1.5 Ways to Solve Least Squares:

Theorem: *Let $A \in R^{m \times n}$ $(m \geq n)$ and $b \in R^m$ be given. A vector $x \in R^n$ minimizes the residual norm $||r||_2 = ||b - Ax||_2$, thereby solving the least squares problem, $\iff$ $r \perp Range(A)$, that is $A^T r = 0$*

Equivalently, $A^TAx = A^Tb \iff Pb = Ax$, where $P \in R^{m\times m}$ is the orthonormal projection onto $range(A)$

The *pseudo inverse* is the matrix $(A^TA)^{-1}A^T$. The pseudo inverse maps vectors $b \in R^m$ to $x \in R^n$

This information gives us the tools to solve least squares in the following ways.

1.5.1 Normal Equations:

The classical way to solve least squares problems is by solving normal equations. We can do this by setting up a *cholesky facorization* which constructs a factorization $A^TA = R^TR$, such that R is upper triangular.

This factorization reduces $A^TAx = A^Tb$ to $R^TRx = A^Tb$. With this we can solve the system through the following algorithm.

Algorithm 1 Least Squares via Normal Equations

1.) Form the matrix A^TA and the vector A^Tb
2.) Compute Cholesky Factorization $A^TA = R^TR$
3.) Solve the lower triangular system $R^Tw = A^Tb$ for w
4.) Solve the upper triangular System $Rx = w$ for x

1.5.2 QR Factorization:

A more modern approach to solving least squares problems is through a QR factorization. In the QR factorization, $A = QR$ and our orthogonal projector $P = QQ^T$ such that $y = Pb = QQ^Tb$. $y \in Range(A)$ so $QRx = QQ^Tb$ and by left multiplication of Q^T results in $Rx = Q^Tb$.

The algorithm is as follows:

Algorithm 2 Least Squares via QR

1.) Solve the reduced QR where $A = QR$.
2.) Compute vector Q^Tb.
3.) Solve the upper triangular system $Rx = Q^Tb$ for x.

1.5.3 Singular Value Decomposition:

Least squares can also be computed using singular value decomposition. Where $A = U\Sigma V^T$.

The orthogonal projector P is $P = UU^T$. Therefore $y = Pb = UU^Tb$, so $U\Sigma V^Tx = UU^Tb$ and $\Sigma V^Tx = U^Tb$. Below is the algorithm.

Algorithm 3 Least Squares via SVD

1.) Compute Reduced SVD, $A = U\Sigma V^T$
2.) Compute vector $U^T b$
3.) Solve the diagonal system $\Sigma w = U^T b$ for w
4.) Set $x = Vw$

1.6 Application:

Least Squares is used in many industries dealing with data, if not all. Below two use cases of least squares will be briefly talked about.

1.6.1 Astronomy:

In **Section 1.1**, it is mentioned that the least squares idea originated from astronomers and geodesist. Least Squares fitting is still used in astronomy to this day. Astrophysicist use different types of least square fitting methods to fit their need. One of these types of least squares fitting methods is called Transit Least Squares (TLS). TLS is used for detecting planetary transits from a time-series photometry.

As seen in **Figure 1.5**, TLS is being compared to a different least square fitting method called Box Least Squares (BLS). TLS is more reliable than BLS in finding any kind of transiting planet but it is particularly suited for the detection of small planets in long time series from particular telescopes.

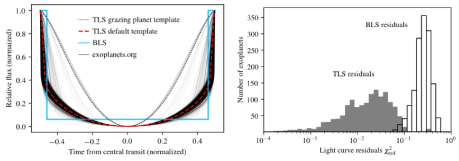

Figure 1.5: Least Squares method comparison BLS vs TLS

TLS Study

1.6.2 Asset Trading Indicator:

Besides astronomy, least squares fitting, is also used a lot in the realm of finance. Polynomial least squares fitting is used for momentum indicators by analyzing trends in asset prices. By fitting a polynomial to historical price data, the method helps determine the best coefficients that represent the price movement over time. This can highlight trends and predict future movements. The polynomial's coefficients provide insight into the rate and direction of change.
Typically used is least squares moving average, which tweaks the polynomial coefficients as new data is being added.

Figure 1.6: LMSA Example: The blue line is the LSMA fitting

LSMA Fitting

2 Google Page Rank:

2.1 History:

In the late 1990's, the usage of the world wide web started gaining momentum. During this time there were many companies that wanted to take advantage of this opportunity by creating search engines that would retrieve web pages from computers globally. One company, Google stood out the most. Google created their search engine strategically. Google had a PageRank algorithm, which quantitatively rated each page on the web based on an "importance" score. The following subsections will discuss this algorithm and what made it unique.

2.2 Importance Ranking:

Google had 3 basic functions for their PageRank algorithm.

1. Crawl the web and locate all web pages with public access
2. Index the data from step 1, so that it can be searched efficiently for relevant keywords or phrases
3. Rate the importance of each web page in the database, so that when user does a search and the subset of web pages in the database with the desired information has been found, the more important pages can be presented first.

Assuming step 1 and 2 are already complete, how can we find the "Importance" of a webpage?

Given each web page has hyperlinks to other webpages, a webpage can be considered important if it has a lot of references to it.

We can label each page x_n such that $n \in \mathbb{N}$ and x is the importance score and n denotes the page. So if $x_1 = 2$, then the first webpage has an importance score of 2.

Google decided to rank x_k by the following equation $x_k = \Sigma_{j \in L_k} \frac{x_j}{n_j}$ where $L_k \subset 1, 2, ..., n$, In other words L_k is the set of page $k's$ backlinks. n_j is the number of outgoing links from page j.

Figure 2.1 shows an example of rankings with 7 web pages and **Table 1** shows the rankings as a system of equations.

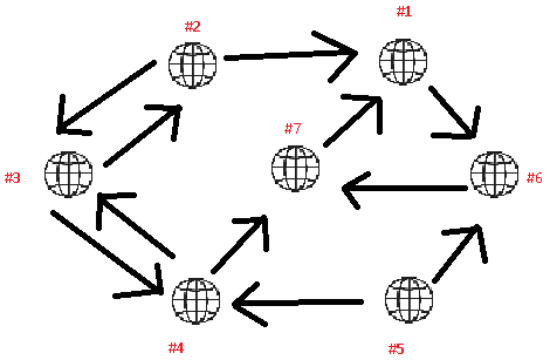

Figure 2.1: Web Page Example

Table 1: System of Equations for Web Pages

Page	**System of Equations**
x_1	$0x_1 + \frac{1}{2}x_2 + 0x_3 + 0x_4 + 0x_5$ $+0x_6 + x_7$
x_2	$0x_1 + 0x_2 + \frac{1}{2}x_3 + 0x_4 + 0x_5$ $+0x_6 + 0x_7$
x_3	$0x_1 + \frac{1}{2}x_2 + 0x_3 + \frac{1}{2}x_4$ $+0x_5 + 0x_6 + 0x_7$
x_4	$0x_1 + 0x_2 + \frac{1}{2}x_3 + 0x_4 + \frac{1}{2}x_5$ $+0x_6 + 0x_7$
x_5	$0x_1 + 0x_2 + 0x_3 + 0x_4 + 0x_5$ $+0x_6 + 0x_7$
x_6	$x_1 + 0x_2 + 0x_3 + 0x_4 + \frac{1}{2}x_5$ $+0x_6 + 0x_7$
x_7	$0x_1 + 0x_2 + 0x_3 + \frac{1}{2}x_4 + 0x_5$ $+x_6 + 0x_7$

We can write the system of equations as a matrix multiplication in the form

$$\begin{bmatrix} 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix}$$

This shows $Ax = x$. A is called a **Column-Stochastic** matrix because in any arbitrary column of A, the entries of that column are non-negative and sum to 1.

Also by definition of **Eigenvector** where, $Ax = \lambda x$ and where λ is the **Eigenvalue**, the value 1 must be the eigenvalue for all of A.

We can call A a "linking matrix". In fact any linking matrix that does not have dangling nodes, a node or pair that is separate from others, is a column-stochastic matrix.

2.3 Modifying Our Linking Matrix A:

For a n page web with no dangling nodes the linking matrix A can be modified to one with unambiguous importance scores by the following equation:

$$\mathbf{M} = (\mathbf{1} - \mathbf{m})\mathbf{A} + \mathbf{mS}$$
$$\text{where } 0 < m < 1,$$
$$n = \text{number of pages},$$
$$S := n \times n \text{ matrix with all entries } \tfrac{1}{n}$$

by creating M, M is a weighted average of A and S and the eigenspace of M is always of dimension 1 if $m \in [0, 1)$.
The proof of why the eigenspace of M is one dimensional is listed directly below.

List of propositions for the proof:

Let $V_1(M)$ be the eigenspace of M for the eigenvalue 1 of a column-stochastic matrix A.

Prop 1: If M is positive and column-stochastic, then any eigenvector in $V_1(M)$ has all positive or all negative components.

Prop 2: let v and w be linearly independent vectors in R^m, $m \geq 2$. Then for some values of s and t that are not both zero, the vector $x = sv + tw$ has both positive and negative components.

Proof. **If M is positive and column-stochastic then $V_1(M)$ is one dimensional.**

Suppose there are two linearly independent eigenvectors v and w in the subspace $V_1(M)$.
For any real numbers s and t that are not both zero, the nonzero vector $x = sv + tw$ must be in $V_1(M)$ and so have components that are all negative or all positive.
But by **Prop 2**, for some choice of s and t the vector x must contain components of mixed sign.
This is a contradiction.
Therefore, $V_1(M)$ cannot contain two linearly independent vectors and so $V_1(M)$ is one dimensional.

□

2.4 Example:

Using **Figure 2.1**, as a web page system, the following python code shows a way to obtain the importance scores for each web page. This code will use the python numpy package to obtain the eigenvector corresponding to the eigenvalue 1.

```
import numpy as np

A = np.matrix([[0, 1/2,  0,   0,  0, 0, 1],
               [0,  0,  1/2,  0,  0, 0, 0],
               [0, 1/2,  0,  1/2, 0, 0, 0],
               [0,  0,  1/2,  0, 1/2,0, 0],
               [0,  0,   0,   0,  0, 0, 0],
               [1,  0,   0,   0, 1/2,0, 0],
               [0,  0,   0,  1/2, 0, 1, 0]])

print("A Matrix: \n", A)
print("\n")

n = 7
m = 0.15

S = np.ones(shape=(7, 7)) * (1 / n)
print("S Matrix: \n", S)
print("\n")

M = (1 - m) * A + m * S
```

```
        print("M Ranking Matrix: \n", M)
        print("\n")

        eigvals, eigvecs = np.linalg.eig(M)

        #Get the eigenvalue corresponding to lamda =
            1
        #This eigenvalue is real
        lamda1 = np.argmax(np.abs(eigvals))

        x = eigvecs[:, lamda1]
        x = x / np.sum(x)
        print(x)
        print("\n")

        print("Mx: \n", M @ x)
```

Listing 1: Python Code

This code outputs the following:

```
A, linking Matrix:
 [[0.  0.5 0.  0.  0.  0.  1. ]
 [0.  0.  0.5 0.  0.  0.  0. ]
 [0.  0.5 0.  0.5 0.  0.  0. ]
 [0.  0.  0.5 0.  0.5 0.  0. ]
 [0.  0.  0.  0.  0.  0.  0. ]
 [1.  0.  0.  0.  0.5 0.  0. ]
 [0.  0.  0.  0.5 0.  1.  0. ]]
```

Figure 2.2: The matrix A

```
S Matrix:
 [[0.14285714 0.14285714 0.14285714 0.14285714 0.14285714 0.14285714
  0.14285714]
 [0.14285714 0.14285714 0.14285714 0.14285714 0.14285714 0.14285714
  0.14285714]
 [0.14285714 0.14285714 0.14285714 0.14285714 0.14285714 0.14285714
  0.14285714]
 [0.14285714 0.14285714 0.14285714 0.14285714 0.14285714 0.14285714
  0.14285714]
 [0.14285714 0.14285714 0.14285714 0.14285714 0.14285714 0.14285714
  0.14285714]
 [0.14285714 0.14285714 0.14285714 0.14285714 0.14285714 0.14285714
  0.14285714]
 [0.14285714 0.14285714 0.14285714 0.14285714 0.14285714 0.14285714
  0.14285714]]
```

Figure 2.3: The matrix S

```
M Ranking Matrix:
 [[0.02142857 0.44642857 0.02142857 0.02142857 0.02142857 0.02142857
  0.87142857]
 [0.02142857 0.02142857 0.44642857 0.02142857 0.02142857 0.02142857
  0.02142857]
 [0.02142857 0.44642857 0.02142857 0.44642857 0.02142857 0.02142857
  0.02142857]
 [0.02142857 0.02142857 0.44642857 0.02142857 0.44642857 0.02142857
  0.02142857]
 [0.02142857 0.02142857 0.02142857 0.02142857 0.02142857 0.02142857
  0.02142857]
 [0.87142857 0.02142857 0.02142857 0.02142857 0.44642857 0.02142857
  0.02142857]
 [0.02142857 0.02142857 0.02142857 0.44642857 0.02142857 0.87142857
  0.02142857]]
```

Figure 2.4: The matrix M

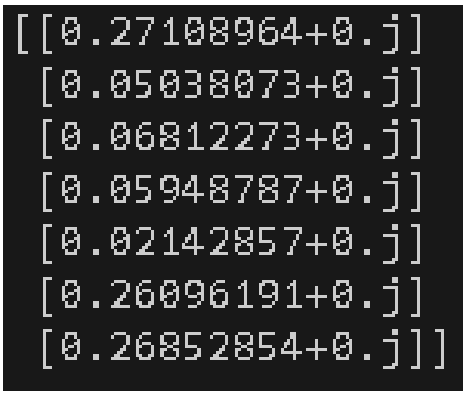

Figure 2.5: The eigenvector x

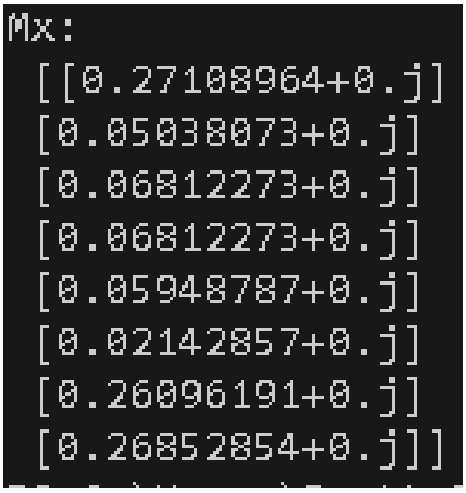

Figure 2.6: The vector Mx which is x(eigenvector corresponding to $\lambda = 1$)

3 Eigenvalues and Eigenvectors:

This section deals obtaining eigenvalues via different algorithms. This section will cover the Rayleigh Quotient, Power Iteration, Rayleigh Quotient Iteration, Inverse Iteration, and lastly, the QR Algorithm. Each algorithm is similar but have different use cases and convergent rates. In these upcoming sections, we will assume that each algorithm deals with matrices that are symmetric.

3.1 Rayleigh Quotient

The **Rayleigh Quotient** of a vector $x \in \mathbb{R}^m$ is the scalar. It is defined as:

$$r(x) := \frac{x^T Ax}{x^T x}$$

If x is an eigenvector, then $r(x) = \lambda$ is the corresponding eigenvalue.
If x is not an eigenvector, then $r(x) = \alpha$, where α is a the least squares solution to $x\alpha \approx Ax$. α is an estimate close to the nearest eigenvector x, but it is not equal to it.

3.2 Power Iteration:

The Power Iteration algorithm uses the Rayleigh Quotient at its core.

The algorithm is as follows:

Algorithm 4 Power Iteration

1.)Let $v^{(0)}$ be an arbitrary vector with $||v^{(0)}|| = 1$
2.)for k = 1,2,...
$w = Av^{(k-1)}$
$v^{(k)} = \frac{w}{||w||}$
$\lambda^{(k)} = (v^{(k)})^T Av^{(k)}$

The power iteration algorithm converges to the largest eigenvalue $\lambda^{(k)}$ and corresponding eigenvector $v^{(k)}$ to a given matrix A.
This algorithm has a convergence rate of $O(|\frac{\lambda_2}{\lambda_1}|^k)$, where $\lambda_1 > \lambda_2 > ... > \lambda_3$

3.3 Rayleigh Quotient Iteration:

Another algorithm similar to power iteration is the Rayleigh Quotient Iteration. The Rayleigh Quotient iteration converges to an eigenvalue/ eigenvector pair for all except a set of measure zero of starting vectors $v^{(0)}$. When it converges, the convergence is cubic. Below is the algorithm:

Algorithm 5 Rayleigh Quotient Iteration

1.)$v^{(0)}$ = arbitrary vector where $||v^{(0)}|| = 1$
2.)$\lambda^{(0)} = (v^{(0)})^T Av^{(0)}$
for $k = 1, 2, ...$
Solve $(A - \lambda^{(k-1)}I)w = v^{(k-1)}$ for w
$v^{(k)} = w/||w||$
$\lambda^{(k)} = (v^{(k)})^T Av^{(k)}$

The Rayleigh Quotient Iteration has a convergence rate of $O(|\lambda^{(k)} - \lambda_J|^3)$, where λ_J is an eigenvalue of the matrix A.

3.4 Inverse Iteration:

Rather than converging to the largest eigenvalue and vector pair, inverse iteration allows for an approximation of an eigenvalue closest to μ if $\mu \in \mathbb{R}$ is not an eigenvalue of the given matrix.

Algorithm 6 Inverse Iteration

1.)Let $v^{(0)}$ be an arbitrary vector with $||v^{(0)}|| = 1$
2.)for k = 1,2,...
solve $((A - \mu I)w = v^{(k-1)}$ for w
$v^{(k)} = \frac{w}{||w||}$
$\lambda^{(k)} = (v^{(k)})^T A v^{(k)}$

3.5 QR Algorithm:

For obtaining all eigenvalues of a given matrix, the QR algorithm can be used. The QR algorithm converges to a matrix with the diagonals being all the eigenvalues of a given matrix A.

Algorithm 7 QR Algorithm

1.) $A^0 = A$
2.)for k = 1,2,...
$Q^{(k)}R^{(k)} = A^{(k-1)}$
$A^{(k)} = R^{(k)}Q^{(k)}$

3.6 Examples

The figures below are examples of the discussed algorithms for the matrix given matrix

$$A = \begin{bmatrix} 5 & 2 & 3 & 4 & 5 \\ 2 & 6 & 7 & 8 & 9 \\ 3 & 7 & 10 & 11 & 12 \\ 4 & 8 & 11 & 15 & 16 \\ 5 & 9 & 12 & 16 & 20 \end{bmatrix}$$

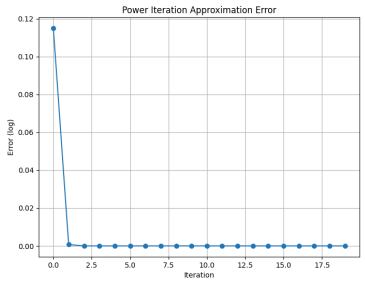

Figure 3.1: Power iteration applied to A — 20 iterations

```
Power Iteration:
Lamda: 47.97428942529308
Vector: [0.16812685 0.31729239 0.42732864 0.54394852 0.62655338]
```

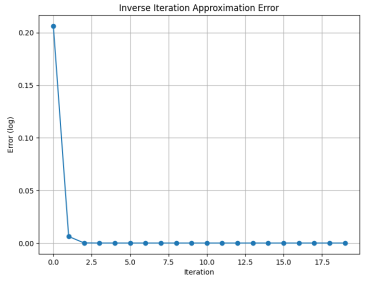

Figure 3.2: Inverse iteration applied to A — 20 iterations

```
Inverse Iteration (mu = 2.5):
Lamda: 2.2883992582374404
Vector: [ 0.31582746  0.41552699  0.5756824  -0.0720044  -0.62529631]
```

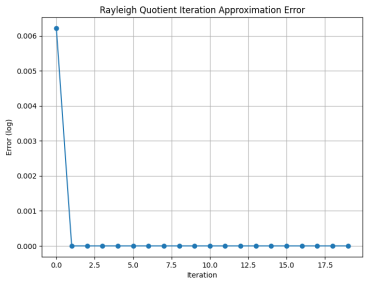

Figure 3.3: Rayleigh Quotient applied to A — 20 iterations

```
Rayleigh Quotient Iteration:
Lamda: 47.97428942529309
Vector: [0.16812685 0.31729239 0.42732864 0.54394852 0.62655338]
```

```
Eigenvalues from QR Algorithm:
[47.97428943  3.93162219  2.28839924  1.17012094  0.6355682 ]

Eigenvectors from QR Algorithm:
[[ 0.16812685 -0.93174462  0.31582596  0.00143142 -0.06194543]
 [ 0.31729239  0.24155496  0.41546903 -0.4756866  -0.66489434]
 [ 0.42732864  0.22844276  0.57568079 -0.01647378  0.65842568]
 [ 0.54394852  0.09516475 -0.07190933  0.77303097 -0.30383524]
 [ 0.62655338 -0.1107282  -0.625348   -0.41937089  0.16804183]]

Eigenvalues from np.linalg.eig:
[47.97428943  3.93162219  2.28839926  0.6355682   1.17012092]

Eigenvectors from np.linalg.eig:
[[ 0.16812685  0.93174405  0.31582746  0.06194541 -0.00147096]
 [ 0.31729239 -0.24155571  0.41552699  0.66490027  0.4756273 ]
 [ 0.42732864 -0.2284438   0.5756824  -0.65842547  0.0164113 ]
 [ 0.54394852 -0.09516463 -0.0720044   0.3038256  -0.77302592]
 [ 0.62655338  0.11072933 -0.62529631 -0.1680366   0.41944976]]
```

Figure 3.4: QR Algorithm applied to A — 20 iterations