Smooth Factorizations in Dynamical Systems

Andrew Binder

July 29, 2009
The standard eigenvalue problem is of the form

\[ Ax = \lambda x \]

where \( A \) is a matrix, \( \lambda \) is an eigenvalue, and \( x \) is the corresponding eigenvector. The eigenvalues must satisfy the characteristic equation

\[ \det(A - \lambda I) = 0. \]
The **nonlinear eigenproblem** is a generalization of the standard eigenvalue problem. The nonlinear problem is of the form

$$ A(\lambda)x = 0 \quad \text{or} \quad y^*A(\lambda) = 0 $$

where $A(\lambda)$ is a matrix whose entries are functions dependent on the value $\lambda$, $\lambda$ is the nonlinear eigenvalue, and $x$ and $y^*$ are the right and left nonlinear eigenvectors respectively. If $A(\lambda) = B - \lambda I$, the problem reduces to the standard eigenvalue problem. The nonlinear eigenvalues must be the solutions of the characteristic equation

$$ \det A(\lambda) = 0. $$
Quadratic Eigenproblem

Example

Quadratic Eigenproblem:

\[ A_2 \lambda^2 + A_1 \lambda + A_0 = 0 \]
Example

Quadratic Eigenproblem:

$$A_2 \lambda^2 + A_1 \lambda + A_0 = 0$$

Applications:

- Structural Dynamics
Quadratic Eigenproblem:

\[ A_2 \lambda^2 + A_1 \lambda + A_0 = 0 \]

Applications:
- Structural Dynamics
- Vibrational Problems
Quadratic Eigenproblem

Example

Quadratic Eigenproblem:

\[ A_2 \lambda^2 + A_1 \lambda + A_0 = 0 \]

Applications:

- Structural Dynamics
- Vibrational Problems
- Fluid Dynamics
Definition

Matrix decomposition is the factorization of a matrix into the product of new matrices.
Matrix Decomposition

Definition

Matrix decomposition is the factorization of a matrix into the product of new matrices.

- QR Decomposition.
- LU Decomposition.
Rank Revealing LU Matrix

Rank Deficient $A$

$P_1 A P_2 = LU$

$= \begin{bmatrix} L_{11} & 0 \\ L_{21} & I \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ 0 & 0 \end{bmatrix}$
Rank Revealing LU Matrix

**Rank Deficient A**

\[ P_1 A P_2 = LU \]
\[ = \begin{bmatrix} L_{11} & 0 \\ L_{21} & I \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ 0 & 0 \end{bmatrix} \]

**Example**

\[ A = \begin{bmatrix} 2 & 4 & 5 \\ 2 & 4 & 3 \\ 3 & 6 & 1 \end{bmatrix} \]
Rank Revealing LU Matrix

**Rank Deficient A**

\[ P_1 A P_2 = LU \]

\[ = \begin{bmatrix} L_{11} & 0 \\ L_{21} & I \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ 0 & 0 \end{bmatrix} \]

**Example**

\[ P_1 A P_2 = \begin{bmatrix} 6 & 1 & 3 \\ 4 & 5 & 2 \\ 4 & 3 & 2 \end{bmatrix} \]
Rank Revealing LU Matrix

### Rank Deficient A

$$P_1AP_2 = LU$$

$$= \begin{bmatrix} L_{11} & 0 \\ L_{21} & I \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ 0 & 0 \end{bmatrix}$$

### Example

$$LU = \begin{bmatrix} 1 & 0 & 0 \\ \frac{2}{3} & 1 & 0 \\ \frac{2}{3} & \frac{7}{13} & 1 \end{bmatrix} \begin{bmatrix} 6 & 1 & 3 \\ 0 & 4\frac{1}{3} & 0 \\ 0 & 0 & 0 \end{bmatrix}$$
Rank Revealing QR Matrix

**Rank Deficient A**

\[ AP = QR \]
\[ = Q \begin{bmatrix} R_{11} & R_{12} \\ 0 & 0 \end{bmatrix} \]
Determining Nonlinear Eigenvalues Through Minimization

Goal

Find a $\lambda$ so that $\det A(\lambda) = 0$
Determining Nonlinear Eigenvalues Through Minimization

Goal
- Find a $\lambda$ so that $\det A(\lambda) = 0$

Plan
- Guess the nonlinear eigenvalue
- Perform rank revealing decomposition
- Minimize lower right block
- Repeat steps using new guess until eigenvalue is found
Newton’s Minimization Technique

**Problem**

- \( f(x) = 0 \)

- \( f(\lambda) = \|U_{22}(\lambda)\|_F^2 \approx \|U_{22}(\lambda_0) + U'_{22}(\lambda_0)(\lambda - \lambda_0)\|_F^2 = 0 \)

**Iterative Method**

- \( x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \)
Newton’s Minimization Technique

Problem

- \( f'(x) = 0 \)

- \( f'(\lambda) = \frac{d}{d\lambda} \| U_{22}(\lambda) \|_F^2 \approx \frac{d}{d\lambda} \| U_{22}(\lambda_0) + U'_{22}(\lambda_0)(\lambda - \lambda_0) \|_F^2 = 0 \)

Iterative Method

- \( x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)} \)
Smooth Decomposition of a Nonsingular Matrix

Lemma

All full column rank matrices $A(\lambda) \in C^k$ with nonsingular leading principle submatrices have a unique $L(\lambda)U(\lambda) \in C^k$ decomposition.

Proof.

- Assume $A(\lambda) = L(\lambda)U(\lambda)$
- Determine entries of $L(\lambda)$ and $U(\lambda)$
Smooth Decomposition of a Matrix Nonsingular at a Point

**Theorem**

Let \( A(\lambda) \in C^k \) such that \( A(\lambda_0) \) is nonsingular. Assume there’s a permutation matrix \( P \) such that \( PA(\lambda_0) = L_0 U_0 \), where \( L_0 \) is unit lower triangular and \( U_0 \) is upper triangular. Then, there is a neighborhood \( N(\lambda_0) \) such that

\[
PA(\lambda) = L(\lambda) U(\lambda) \quad \forall \lambda \in N(\lambda_0),
\]

with \( L(\lambda_0) = L_0, \ U(\lambda_0) = U_0; \ L(\lambda), U(\lambda) \in C^k, \ L(\lambda) \) unit lower triangular matrix, and \( U(\lambda) \) upper triangular.

**Proof.**

- Locally perturb \( A(\lambda_0) \) using Taylor’s Theorem
- Create lower triangular matrices so that the perturbation becomes upper triangular.
Smooth Decomposition of a General Matrix

**Theorem**

Let $A(\lambda) \in C^k$ be a $n \times n$ matrix such that $A(\lambda_0)$ has a column rank of $n - m$, $m \leq n - 1$. Assume there are permutation matrices $P_1$, $P_2$ such that $P_1A(\lambda_0)P_2 = L_0U_0$, where $L_0$ is a block unit lower triangular matrix and $U_0$ is a block upper triangular matrix. Then, there is a neighborhood $N(\lambda_0)$ such that

$$P_1A(\lambda)P_2 = L(\lambda)U(\lambda) \quad \forall \lambda \in N(\lambda_0),$$

with $L(\lambda_0) = L_0$, $U(\lambda_0) = U_0$; $L(\lambda)$, $U(\lambda) \in C^k$, $L(\lambda)$ a block unit lower triangular matrix, $U(\lambda)$ a block upper triangular matrix.
Step 1: Given an initial approximation $\lambda_0$ to $\lambda^*_\star$

Step 2: Compute

$$A(\lambda_i) \text{ and } A'(\lambda_i), \ i = 0, 1, \ldots$$

Step 3: Compute the $LU$ decomposition with complete column pivoting of $A(\lambda_i)$:

$$P_1A(\lambda_i)P_2 = L(\lambda_i)U(\lambda_i)$$
LU Algorithm for Computation of Nonlinear Eigenvalues

Step 4: Compute

\[ U'_{2,2}(\lambda_i) = (L^{-1}_i P_1 A'(\lambda_i) P_2)_{2,2} - (L^{-1}_i P_1 A'(\lambda_i) P_2)_{2,1} (U^{(i)-1}_{1,1} U^{(i)}_{1,2}) \]

Step 5: Compute

\[ \lambda_{i+1} = \lambda_i - \frac{(\text{col } U'_{2,2}(\lambda_i))^H \cdot \text{col } U_{2,2}(\lambda_i)}{||U'_{2,2}(\lambda_i)||^2_F}. \]

Step 6: If the desired accuracy is attained, stop the iteration. Otherwise, repeat steps 2-6.
Step 3: Compute the $LU$ decomposition with complete column pivoting of $A(\lambda_i)$:

$$A(\lambda_i)P = Q(\lambda_i)R(\lambda_i)$$

Step 5: Compute

$$\lambda_{i+1} = \lambda_i - \frac{(\text{col } R'_{2,2}(\lambda_i))^H \cdot \text{col } R_{2,2}(\lambda_i)}{\|R'_{2,2}(\lambda_i)\|_F^2}.$$
Theorem of Numerical Rank Determination

Property

Let $AP = QR$ be a rank revealing decomposition. Then, the diagonals of $R$ have the property that

$$|r_{1,1}| \geq \cdots \geq |r_{t,t}| \gg |r_{t+1,t+1}| \geq \cdots \geq |r_{n,n}|.$$
Theory of Numerical Rank Determination

Property

Let \( AP = QR \) be a rank revealing decomposition. Then, the diagonals of \( R \) have the property that

\[
|r_{1,1}| \geq \cdots \geq |r_{t,t}| \gg |r_{t+1,t+1}| \geq \cdots \geq |r_{n,n}|.
\]

\[
|r_{t+1,t+1}| \leq \epsilon |r_{1,1}| \leq |r_{t,t}|
\]
Theory of Numerical Rank Determination

Property

Let $AP = QR$ be a rank revealing decomposition. Then, the diagonals of $R$ have the property that

$$|r_{1,1}| \geq \cdots \geq |r_{t,t}| \gg |r_{t+1,t+1}| \geq \cdots \geq |r_{n,n}|.$$

$$\frac{|r_{t+1,t+1}|}{|r_{1,1}|} \leq \epsilon \leq \frac{|r_{t,t}|}{|r_{1,1}|}.$$
# 4 × 4 Time Comparison

**Table:** Time [ms] Comparison of 4 × 4 Algorithm Performance

<table>
<thead>
<tr>
<th>Nonlinear Matrix</th>
<th>LU Average</th>
<th>QR Average</th>
<th>Ratio of Averages (QR / LU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>4.369</td>
<td>16.141</td>
<td>3.695</td>
</tr>
<tr>
<td>Q, E</td>
<td>4.445</td>
<td>15.943</td>
<td>3.587</td>
</tr>
<tr>
<td>Q, S</td>
<td>4.472</td>
<td>16.088</td>
<td>3.597</td>
</tr>
<tr>
<td>Q, E, S</td>
<td>4.568</td>
<td>16.332</td>
<td>3.575</td>
</tr>
</tbody>
</table>
10 x 10 Time Comparison

Table: Time [ms] Comparison of 10 x 10 Algorithm Performance

<table>
<thead>
<tr>
<th>Nonlinear Matrix</th>
<th>LU Average</th>
<th>QR Average</th>
<th>Ratio of Averages (QR / LU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>9.632</td>
<td>44.052</td>
<td>4.574</td>
</tr>
<tr>
<td>Q, E</td>
<td>9.829</td>
<td>44.035</td>
<td>4.480</td>
</tr>
<tr>
<td>Q, S</td>
<td>10.088</td>
<td>44.643</td>
<td>4.425</td>
</tr>
<tr>
<td>Q, E, S</td>
<td>10.166</td>
<td>46.169</td>
<td>4.541</td>
</tr>
</tbody>
</table>
### 100 x 100 Time Comparison

**Table:** Time [ms] Comparison of 100 x 100 Algorithm Performance

<table>
<thead>
<tr>
<th>Nonlinear Matrix</th>
<th>LU Average</th>
<th>QR Average</th>
<th>Ratio of Averages (QR / LU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>391.154</td>
<td>1696.066</td>
<td>4.336</td>
</tr>
<tr>
<td>Q, E</td>
<td>362.494</td>
<td>1630.445</td>
<td>4.498</td>
</tr>
<tr>
<td>Q, S</td>
<td>393.234</td>
<td>1634.839</td>
<td>4.157</td>
</tr>
<tr>
<td>Q, E, S</td>
<td>389.039</td>
<td>1650.813</td>
<td>4.243</td>
</tr>
</tbody>
</table>
### Table: Average Iteration Comparison of 10 x 10 Algorithm Performance

<table>
<thead>
<tr>
<th>Nonlinear Matrix</th>
<th>LU</th>
<th>QR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Number</td>
<td>Time/Iter [ms]</td>
</tr>
<tr>
<td>Q</td>
<td>4.40</td>
<td>2.19</td>
</tr>
<tr>
<td>Q, E</td>
<td>4.44</td>
<td>2.21</td>
</tr>
<tr>
<td>Q, S</td>
<td>4.51</td>
<td>2.24</td>
</tr>
<tr>
<td>Q, E, S</td>
<td>4.38</td>
<td>2.32</td>
</tr>
</tbody>
</table>
Newton Steffensen Method

Cubic Convergence Iterative Formula

Applying Steffensen’s acceleration method to Newton’s root finding method generates an iterative formula with cubic convergence. Let \( f(x_*) = 0 \) and let \( x_0 \) be sufficiently close to \( x_* \), then the successive iterative approximations are determined by

\[
x_{n+1} = x_n - \frac{f^2(x_n)}{f'(x_n)(f(x_n) - f(x_n^*))}
\]

where

\[
x_n^* = x_n - \frac{f(x_n)}{f'(x_n)}.
\]
Newton Steffensen Method

\[ x_{n+1} = x_n - \frac{f'(x_n)^2}{f''(x_n)(f'(x_n) - f'(x_n^*))} \]

where

\[ x_n^* = x_n - \frac{f'(x_n)}{f''(x_n)} \cdot \frac{f'(x_n)}{f''(x_n)} \cdot \frac{f'(x_n)}{f''(x_n)} \cdot \frac{f'(x_n)}{f''(x_n)} \]

- \( f'(\lambda) = (\text{col } U'_{2,2}(\lambda_i))^H \cdot \text{col } U_{2,2}(\lambda_i) \)
- \( f''(\lambda) = \| U'_{2,2}(\lambda_i) \|_F^2 \)
Cubic Time Comparison

Table: Time [ms] Comparison of 10 x 10 Cubic Convergence Algorithm Performance

<table>
<thead>
<tr>
<th>Nonlinear Matrix</th>
<th>LU Average</th>
<th>QR Average</th>
<th>Ratio of Averages (QR / LU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>12.323</td>
<td>57.906</td>
<td>4.699</td>
</tr>
<tr>
<td>Q, E</td>
<td>12.124</td>
<td>57.504</td>
<td>4.743</td>
</tr>
<tr>
<td>Q, S</td>
<td>13.311</td>
<td>62.117</td>
<td>4.667</td>
</tr>
<tr>
<td>Q, E, S</td>
<td>12.422</td>
<td>59.012</td>
<td>4.751</td>
</tr>
</tbody>
</table>
Cubic Iteration Comparison

**Table:** Average Iteration Comparison of $10 \times 10$ Cubic Convergence Algorithm Performance

<table>
<thead>
<tr>
<th>Nonlinear Matrix</th>
<th>LU</th>
<th>QR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Number</td>
<td>Time/Iter [ms]</td>
</tr>
<tr>
<td>Q</td>
<td>3.26</td>
<td>3.775</td>
</tr>
<tr>
<td>Q, E</td>
<td>3.19</td>
<td>3.799</td>
</tr>
<tr>
<td>Q, S</td>
<td>3.48</td>
<td>3.823</td>
</tr>
<tr>
<td>Q, E, S</td>
<td>3.18</td>
<td>3.884</td>
</tr>
</tbody>
</table>
### Lines where the most time was spent

<table>
<thead>
<tr>
<th>Line Number</th>
<th>Code</th>
<th>Calls</th>
<th>Total Time</th>
<th>% Time</th>
<th>Time Plot</th>
</tr>
</thead>
<tbody>
<tr>
<td>59</td>
<td>([Q, R, P] = \text{apqr}(Ak));</td>
<td>5033</td>
<td>43.345 s</td>
<td>95.4%</td>
<td></td>
</tr>
<tr>
<td>82</td>
<td>(dR22 = \text{Knt}^{'<em>}\text{QAP}^{}\text{Knt} - \text{Knt}^{'</em>}\text{QAP...})</td>
<td>5033</td>
<td>0.859 s</td>
<td>1.9%</td>
<td></td>
</tr>
<tr>
<td>76</td>
<td>(\text{QAP} = \text{Q}^{'*}\text{dAk}^{}\text{P};)</td>
<td>5033</td>
<td>0.223 s</td>
<td>0.5%</td>
<td></td>
</tr>
<tr>
<td>92</td>
<td>(\text{guess} = \text{guess} - \text{coldR22'}^{}\text{colR2...})</td>
<td>5033</td>
<td>0.158 s</td>
<td>0.3%</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>(\text{Ak} = \text{Mnot} + \text{Mone}^{}\text{guess} + \text{Mtwo}^{'*}\text{...})</td>
<td>5033</td>
<td>0.118 s</td>
<td>0.3%</td>
<td></td>
</tr>
<tr>
<td>All other lines</td>
<td></td>
<td></td>
<td>0.724 s</td>
<td>1.6%</td>
<td></td>
</tr>
<tr>
<td>Totals</td>
<td></td>
<td></td>
<td>45.426 s</td>
<td>100%</td>
<td></td>
</tr>
</tbody>
</table>
Super Quadratic Convergence

Goal

- Approximate $f'(x_n^*)$ using previously calculated values.
Super Quadratic Convergence

Goal

- Approximate $f'(x_n^*)$ using previously calculated values.
- Add another term in the Taylor’s Series expansion approximation.
Super Quadratic Convergence

Goal

- Approximate $f'(x_n^*)$ using previously calculated values.
- Add another term in the Taylor’s Series expansion approximation.
- Solve for $f'(x_n^*)$. 
Vibrating rail track resting on sleepers (lateral supports)
Vibrating Train Tracks

- Vibrating rail track resting on sleepers (lateral supports)
- Initially modeled as a partial differential equation
Vibrating Train Tracks

- Vibrating rail track resting on sleepers (lateral supports)
- Initially modeled as a partial differential equation
- Discretized and turned into a quadratic eigenvalue problem with $10 \times 10$ matrices
Vibrating Train Tracks

- Vibrating rail track resting on sleepers (lateral supports)
- Initially modeled as a partial differential equation
- Discretized and turned into a quadratic eigenvalue problem with $10 \times 10$ matrices
- Eigenvalues are explicitly known. There exist multiple eigenvalues.
Vibrating rail track resting on sleepers (lateral supports)
Initially modeled as a partial differential equation
Discretized and turned into a quadratic eigenvalue problem with 10 × 10 matrices
Eigenvalues are explicitly known. There exist multiple eigenvalues.
Algorithm was successful within an error tolerance of $10^{-15}$
Vibrating Train Tracks

\[ U = \begin{bmatrix}
0.66 & 0 & 0 & 0 & -0.25 & 0.31 & 0.31 & -0.25 & 0 & 0 \\
0 & 0.66 & 0 & 0 & 0.31 & -0.25 & 0 & 0 & 0.31 & -0.25 \\
0 & 0 & 0.66 & -0.25 & 0 & 0 & 0.31 & 0 & -0.25 & 0.31 \\
0 & 0 & 0 & 0.57 & 0 & 0 & -0.13 & 0.31 & 0.22 & 0.12 \\
0 & 0 & 0 & 0 & 0.42 & -0.01 & 0.12 & 0.22 & -0.15 & 0.12 \\
0 & 0 & 0 & 0 & 0 & 0.42 & -0.14 & 0.13 & 0.11 & 0.22 \\
0 & 0 & 0 & 0 & 0 & 0 & 0.25 & -0.08 & 0.25 & -0.08 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.22 & 0 & 0.22 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix} \]
Next Step

- Analyze the super quadratic algorithm
- Take advantage of matrix structure such as symmetry
- Determine all eigenvalues in a region
- MATLAB polynomial nonlinear eigenvalue solver