NUMERICAL LINEAR ALGEBRA EXAM CHEATSHEET
6th DECEMBER 2023

1. Basics and LU Factorisation

The p-norm of a vector is defined as:

$$||\mathbf{x}||_p = \left(\sum_i^n |x_i|^p \right)^{\frac{1}{p}}$$

The induced p-norm of a matric is defined as:

$$||\mathbf{A}||_p = \max_{x \neq 0} \frac{||\mathbf{Ax}||_p}{||\mathbf{x}||_p}$$

A few important examples here are:

(a) The matrix norm induced by the ∞-norm is the maximum row sum.
(b) The matrix norm induced by the 1-norm is the maximum column sum.
(c) The matrix norm induced by the 2-norm is the square root of the spectral radius of $\mathbf{A}^T\mathbf{A}$:

$$||\mathbf{A}||_2 = \sqrt{\rho(\mathbf{A}^T\mathbf{A})}, \text{ where } \rho(\mathbf{A}^T\mathbf{A}) = \max\{|\lambda|, \ \lambda \text{ is an eigenvalue of } \mathbf{A}^T\mathbf{A}\}$$

The condition number of a matrix in the p-norm is,

$$\kappa_p(\mathbf{A}) = ||\mathbf{A}||_p ||\mathbf{A}^{-1}||_p,$$

for invertible $\mathbf{A}$ and ∞ otherwise. A matrix for which κ_p is large (which is an ill-defined term, unfortunately) is said to be ill-conditioned. Typically, solving systems involving ill-conditioned matrices causes large round off errors. This is encapsulated in the following theorem.

Theorem 1. Let $\mathbf{Ax} = \mathbf{b}$ and $(\mathbf{A} + \Delta\mathbf{A})\hat{\mathbf{x}} = \mathbf{b}$. If $\kappa_p(\mathbf{A})||\Delta\mathbf{A}||_p < ||\mathbf{A}||_p$, then

$$\frac{||\mathbf{x} - \hat{\mathbf{x}}||_p}{||\mathbf{x}||_p} \leq \frac{\kappa_p(\mathbf{A})}{1 - \kappa_p(\mathbf{A})\frac{||\Delta\mathbf{A}||_p}{||\mathbf{A}||_p}} \frac{||\Delta\mathbf{A}||_p}{||\mathbf{A}||_p}.$$

Another important property is that for $\mathbf{A}$ such that it's eigenvalues are real, denoting $|\lambda_1| \geq |\lambda_2| \geq ... \geq |\lambda_n|$, one has:

$$\kappa_2(\mathbf{A}) = \frac{|\lambda_1|}{|\lambda_n|} \geq 1$$

Algorithm 1 Algorithm LU. LU factorisation

Input: $\mathbf{A} \in \mathbb{R}^{n \times n}$
Output: $\mathbf{L}, \mathbf{U} \in \mathbb{R}^{n \times x}$, the LU factorisation of $\mathbf{A}$

```
L = I, U = A
for k=1,2...n-1 do
    for j = k+1...n do
        l_jk = u_jk / u_kk
        (u_jk, ..., u_jn) = (u_jk, ..., u_jn) − l_jk(u_kk, ..., u_kn)
    end for
end for
```

Algorithm LU has cost $C(n) = \frac{2}{3}n^3 + \frac{1}{2}n^2 - \frac{7}{6}n$. In combination with Forward and Backward substitution (FS,BS, both with $C(n) = n^2$), Algorithm LU can be used as a form of Gaussian elimination,

Tristan Jacquel, December 2023.

to solve linear systems $\mathbf{Ax} = \mathbf{b}$, by factorising $\mathbf{LUx} = \mathbf{b}$, then using FS to solve $\mathbf{Ly} = \mathbf{b}$ and finally using BS to solve $\mathbf{Ux} = \mathbf{y}$.

In this form, Algorithm GE will have cost $C(n) = \frac{2}{3}n^3 + \frac{5}{2}n^2 - \frac{7}{6}n$. This version of Gaussian elimination is highly unstable. We can achieve better stability by introducing partial pivoting, permuting the rows of $\mathbf{A}$ and hence of $\mathbf{U}, \mathbf{L}$ using a permutation matrix $\mathbf{P}$, so as to minimize $|u_{ik}|$ at each step. The cost of GEPP is $C(n) = \frac{2}{3}n^3 + \frac{5}{2}n^2 - \frac{7}{6}n$.

Algorithm GEPP is backwards stable with the error bound

$$||\Delta\mathbf{A}||_\infty \leq p(n)2^{n-1}\varepsilon_m||\mathbf{A}||_\infty,$$

with $p(n), \varepsilon_m$ being some 3rd degree polynomial in n and the machine epsilon respectively.

2. QR Factorisation

Where the LU factorisation turned a matrix into an upper and lower triangular matrix, QR factorisation turns a matrix into an orthogonal matrix and $\mathbf{Q}$ (recall orthogonal means $\mathbf{QQ}^T = \mathbf{I}$) and a non-singular upper triangular matrix $\mathbf{R}$. One possible implementation of QR decomposition is the Gram Schmidt method:

Algorithm 2 Algorithm GS. Gram Schmidt orthonormalisation

Input: $\mathbf{A} \in \mathbb{R}^{n\times n}$
Output: $\mathbf{Q}, \mathbf{R} \in \mathbb{R}^{n\times x}$, the QR decomposition of $\mathbf{A}$

1: $\mathbf{R} = \mathbf{0}$
2: **for** j=1,2...n-1 **do**
3: $\quad \mathbf{q}_j = \mathbf{a}_j$
4: $\quad$ **for** k = 1,2,...,j-1 **do**
5: $\quad\quad r_{kj} = \mathbf{q}_k^T\mathbf{a}_j$
6: $\quad\quad \mathbf{q}_j = \mathbf{q}_j - r_{kj}\mathbf{q}_k$
7: $\quad$ **end for**
8: $\quad r_{jj} = ||\mathbf{q}_j||_2 = \sqrt{\mathbf{q}_j^T\mathbf{q}_j}$
9: $\quad \mathbf{q}_j = \mathbf{q}_j/r_{jj}$
10: **end for**

Algorithm GS has computational cost $C(n) = 2n^3 + n^2 + n$, giving the overall SQR (solve using QR) algorithm a cost $C(n) = 2n^3 + 4n^2 + n$. This version of GS is known to be unstable, so we can modify line 5 to $r_{kj} = \mathbf{q}_k^T\mathbf{q}_j$. The new algorithm gives the same answer in exact arithmetic, but is much more stable using FLOPs.

Theorem 2. The computed matrices $\hat{\mathbf{Q}}, \hat{\mathbf{R}}$ found using MGS satisfy:

$$\hat{\mathbf{Q}}^T\hat{\mathbf{Q}} = \mathbf{I} + \Delta\mathbf{I}, \quad \hat{\mathbf{Q}}\hat{\mathbf{R}} = \mathbf{A} + \Delta\mathbf{A},$$

where $||\Delta\mathbf{I}||_2$ and $||\Delta\mathbf{A}||_2$ are bounded above as:

$$||\Delta\mathbf{I}||_2 \leq \frac{\alpha_1(n)\varepsilon_m\kappa_2(\mathbf{A})}{1 - \alpha_1(n)\varepsilon_m\kappa_2(\mathbf{A})},$$

$$||\Delta\mathbf{A}||_2 \leq \alpha_2(n)\varepsilon_m||\mathbf{A}||_2.$$

3. The Householder Algorithm

Based on the Householder reflection $\mathbf{H}_k\mathbf{x} = \mathbf{x} - 2\mathbf{w}\mathbf{w}^T\mathbf{x}$, we can find an alternative method of QR decomposition.

As above, QR has cost $C(n) = \frac{8}{3}n^3 + O(n^2)$. However, since for SQR, solving the system doesn't require us to explicitly compute Q, we can get around this by multiplying as we go with $\mathbf{Q}$. (i.e. computing $\mathbf{y} = \mathbf{Q}_{n-1}...\mathbf{Q}_1\mathbf{b}$ from right to left using MV) This saves on cost leaving us with cost $C(n) = \frac{4}{3}n^3 + O(n^2)$.

Lemma 3. The matrix $\mathbf{R}_k = \mathbf{Q}_k...\mathbf{Q}_1\mathbf{A}$ has zeros below the diagonal in columns 1 to k, for $1 \leq k \leq n-1$.

Algorithm 3 Algorithm QR. Householder QR factorisation.

Input: $\mathbf{A} \in \mathbb{R}^{n\times n}$
Output: $\mathbf{Q}, \mathbf{R} \in \mathbb{R}^{n\times x}$, the QR decomposition of $\mathbf{A}$
1: $\mathbf{Q} = \mathbf{I}, \mathbf{R} = \mathbf{A}$
2: **for** k=1,2...n-1 **do**
3: $\mathbf{u} = (r_{kk}, ..., r_{nk}) \in \mathbb{R}^{n-k+1}$
4: $\mathbf{v} = \mathbf{u} + \text{sign}(u_1)||\mathbf{u}||_2\mathbf{e}_1$
5: $\mathbf{w} = \mathbf{v}/||\mathbf{v}||_2$
6: $\mathbf{H}_k = \mathbf{I} - 2\mathbf{w}\mathbf{w}^T \in \mathbb{R}^{(n-k+1)\times(n-k+1)}$
7: $\mathbf{Q}_k = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{H}_k \end{bmatrix}$
8: $\mathbf{R} = \mathbf{Q}_k\mathbf{R}$
9: $\mathbf{Q} = \mathbf{Q}\mathbf{Q}k$
10: **end for**

Theorem 4. Let $\mathbf{Ax} = \mathbf{b}$ and let $\hat{\mathbf{x}}$ be the computed solution using SQR with the Householder decomposition. Then,

$$(\mathbf{A} + \Delta\mathbf{A})\hat{\mathbf{x}} = \mathbf{b},$$

with the error/perturbation $\Delta\mathbf{A}$ being bounded above as

$$||\Delta\mathbf{A}||_2 \leq \alpha n^{5/2}\varepsilon_m||\mathbf{A}||_2.$$

4. Iterative Methods

We consider general iterative methods of the following form:

Algorithm 4 Algorithm GI. General Iterative Method.

Input: $\mathbf{A} = \mathbf{M} + \mathbf{N} \in \mathbb{R}^{n\times n},\ \mathbf{b} \in \mathbb{R}^n,\ \mathbf{x}_0 \in \mathbf{R}^n, \varepsilon_r > 0$
Output: $\mathbf{x}_k \in \mathbb{R}^n$, such that $\mathbf{A}\mathbf{x}_k \approx \mathbf{b}$
1: $k = 0$
2: **while** $||\mathbf{A}\mathbf{x}_k - \mathbf{b}||_p > \varepsilon_r$ **do**
3: $k = k + 1$
4: Compute $\mathbf{y}_k = \mathbf{b} - \mathbf{N}\mathbf{x}_{k-1}$
5: Solve $\mathbf{M}\mathbf{x}_k = \mathbf{y}_k$
6: **end while**

The key is that we split the matrix $\mathbf{A}$ such that $\mathbf{Mx} = \mathbf{b}$ is easy to solve (e.g. $\mathbf{M}$ is diagonal.)

Theorem 5. Let $\mathbf{e}_k = \mathbf{x} - \mathbf{x}_k = \mathbf{R}^k\mathbf{e}_0$, with $\mathbf{R} = -\mathbf{M}^{-1}\mathbf{N}$. Then,

$$||\mathbf{e}_k||_p \leq ||\mathbf{R}||_p^k||\mathbf{e}_0||_p.$$

This guarantees convergence provided $||\mathbf{R}||_p < 1$.

The Jacobi method sets $\mathbf{M} = \mathbf{D},\ \mathbf{N} = \mathbf{L} + \mathbf{U}$, where $\mathbf{D}$ is the strictly diagonal part of $\mathbf{A}$, so we can solve $\mathbf{Mx} = \mathbf{y}$ by simple division. Applying the above theorem gives that a sufficient, but not necessary condition for convergence is that $\mathbf{A}$ is strictly diagonally dominant, which means

$$|a_ii| > \sum_{j\neq i} |a_{ij}|, \quad \forall i = 1, 2, ..., n.$$

The Jacobi method has cost $C(n) = 2n^2 + 2n$ *per iteration.*

The Gauss-Seidel method splits the matrix such that $\mathbf{M} = \mathbf{L} + \mathbf{D},\ \mathbf{N} = \mathbf{U}$, so that we can solve $\mathbf{Mx} = y$ using FS. Just like the Jacobi method, a sufficient but not necessary condition for convergence is $\mathbf{A}$ being strictly diagonally dominant. The Gauss-Seidel method has cost $C(n) = 2n^2$, so to leading order, the Jacobi and Gauss-Seidel methods are equally costly. (note that one can adjust the Jacobi method to have equal cost to the Gauss-Seidel method by noting $\mathbf{L} + \mathbf{U}$ has zeros on the diagonal and modifying MV accordingly, which saves on $2n$ operations)

5. Polynomial and Least Square Fitting

Suppose we are given a set of points $\{a_i, b_i\}_{i=1}^m$, and we want to find the degree $m-1$ polynomial that goes through these points. This corresponds to solving the linear system $\mathbf{Ax} = \mathbf{b}$, with the entries of the *Vandermonde matrix* $\mathbf{A}$ given by

$$a_{ij} = a_i^{j-1}.$$

Vandermonde matrices are very ill-conditioned, especially when the number of points m is large. There is a risk of overfitting when solving for a degree $m-1$ polynomial, so often we only fit for a degree n polynomial, thus using $\mathbf{A} \in \mathbb{R}^n$. In these cases, it is often impossible to exactly solve the linear system, hence we only seek to minimize

$$||\mathbf{Ax} - \mathbf{b}||_2.$$

Theorem 6. Let $\mathbf{A} \in \mathbb{R}^{m\times n}, \mathbf{b} \in \mathbb{R}^m$. A vector $\mathbf{x} \in \mathbb{R}^n$ minimises $||\mathbf{Ax} - \mathbf{b}||_2$ if and only if it solves the normal equations

$$\mathbf{A}^T\mathbf{Ax} = \mathbf{A}^T\mathbf{b}.$$

Least square fitting, as this is called, is almost never done by solving the normal equations because by applying Theorem 1, we see that our error is only bounded by $\kappa_p(\mathbf{A}^T\mathbf{A}) = \kappa_p(\mathbf{A})^2$,[1] which is highly undesirable as it amplifies error for an ill-conditioned matrix (which the Vandermonde matrix is!)

In the general case, $\mathbf{A}^T\mathbf{A}$ doesn't need to be invertible and so the minimizer is not unique. If and only if $m \geq n$ and $\text{rank}(\mathbf{A}) = n$, then $\mathbf{A}^T\mathbf{A}$ is invertible and the normal equations have the unique solution

$$\mathbf{x} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b}.$$

The matrix $\mathbf{A}^\dagger = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T$ is the *Moore-Penrose pseudo-inverse*. If $\mathbf{A}$ is invertible, then $\mathbf{A}^\dagger = \mathbf{A}^{-1}$ and hence $\kappa_p(\mathbf{A}) = ||\mathbf{A}||_p||\mathbf{A}^\dagger||_p$ in this case.

The better alternative to the normal equations is using QR. However, since $\mathbf{R} \in \mathbb{R}^{m\times n}$, $m \geq n$ is upper triangular, it is sufficient to only use the reduced QR factorisation such that $\mathbf{R}_1 \in \mathbb{R}^{n\times n}$ and $\mathbf{Q}_1 \in \mathbb{R}^{m\times n}$.

Algorithm 5 Algorithm LSQ-QR. Least Squares using QR factorisation.

Input: $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{b} \in \mathbb{R}^m$
Output: $\mathbf{x} \in \mathbb{R}^n$ that minimizes $||\mathbf{Ax} - \mathbf{b}||_2$
1: Compute the reduced QR factorisation $\mathbf{A} = \mathbf{Q}_1\mathbf{R}_1$.
2: Compute $\mathbf{y} = \mathbf{Q}_1^T\mathbf{b} \in \mathbb{R}^n$.
3: Solve $\mathbf{R}_1\mathbf{x} = \mathbf{y}$ using BS.

We can use algorithm QR almost as written for this purpose, but now the loop over k now runs all the way to n and we need to zero out the terms below the diagonal in each column. The computational cost of using QR in this way is $C(n, m) = 2n^2m - \frac{2}{3}n^3 + O(m^2 + n^2 + mn)$.

Theorem 7. Let $\mathbf{x}$ be the minimiser of $||\mathbf{Ax} - \mathbf{b}||_2$ and let $\hat{\mathbf{x}}$ be the computes minimiser using QR. Then, $\hat{\mathbf{x}}$ is the minimiser of $||(\mathbf{A} + \Delta\mathbf{A})\mathbf{x} - \mathbf{b}||_2$, where with $\alpha > 0$ is a small integer constant, $||\Delta\mathbf{A}||_2$ is bounded above by

$$||\Delta\mathbf{A}||_2 \leq \alpha n^{3/2} m\varepsilon_m||\mathbf{A}||_2.$$

If $\text{rank}(\mathbf{A}) = \text{rank}(\mathbf{A} + \Delta\mathbf{A}) = n$, and $\kappa_2(\mathbf{A})||\Delta\mathbf{A}||_2 < ||\mathbf{A}||_2$, then

$$\frac{||\mathbf{x} - \hat{\mathbf{x}}||_2}{||\mathbf{x}||_2} \leq \frac{\kappa_2(\mathbf{A})}{1 - \kappa_2(\mathbf{A})\frac{||\Delta\mathbf{A}||_2}{||\mathbf{A}||_2}} \frac{||\Delta\mathbf{A}||_2}{||\mathbf{A}||_2} \left(2 + (\kappa_2(\mathbf{A}) + 1)\frac{||\mathbf{r}||_2}{||\mathbf{A}||_2||\mathbf{x}||_\mathbf{2}}\right),$$

where we defined $\mathbf{r} = \mathbf{b} - \mathbf{Ax}$.

[1] This can be proved trivially using the singular value decomposition.

6. Singular Value Decomposition

A factorisation of $\mathbf{A}$ of the form $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ is called a singular value decomposition if $\mathbf{U} \in \mathbb{R}^{m\times m}$, $\mathbf{V} \in \mathbb{R}^{n\times n}$ and $\mathbf{\Sigma}$ is diagonal with positive entries $\sigma_1, \sigma_2, ..., \sigma_p$ with $p = \min(m, n)$. The columns of $\mathbf{U}, \mathbf{V}$ are called the left and right singular vectors of $\mathbf{A}$.

The SVD is very useful for looking at if $\mathbf{A}$ is of full rank. That is, $\text{rank}(\mathbf{A}) = \text{rank}(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T) = \text{rank}(\Sigma)$. So, the rank of $\mathbf{A}$ is the number of non-zero singular values.

Theorem 8. Let $\mathbf{A} \in \mathbb{R}^{m\times n}$ with $m \geq n$. the condition number of $\mathbf{A}$ is given by

$$\kappa_2(\mathbf{A}) = \frac{\sigma_{\max}}{\sigma_{\min}} \geq 1$$

The SVD can also be used in Least Square problems and is the most stable (but also the most costly) method for such purposes. Just as with QR factorisation, we work with the reduced SVD, since the last $m - n$ rows of $\mathbf{\Sigma}$ are zero.

Algorithm 6 Algorithm LSQ-SVD. Least Squares using SVD.

Input: $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{b} \in \mathbb{R}^m$
Output: $\mathbf{x} \in \mathbb{R}^n$ that minimizes $||\mathbf{A}\mathbf{x} - \mathbf{b}||_2$
1: Compute the reduced SVD $\mathbf{A} = \mathbf{U}_1\mathbf{\Sigma}_1\mathbf{V}_1^T$.
2: Compute $\mathbf{z} = \mathbf{U}_1^T\mathbf{b}$.
3: Solve $\mathbf{\Sigma}_1\mathbf{y} = \mathbf{z}$.
4: Compute $\mathbf{x} = \mathbf{V}\mathbf{y}$

7. Eigenvalue Problems

Famously, the eigenvalue problem, which is equivalent to the problem of solving for the roots of polynomials, is not generally possible in finitely many operations.

Before we delve into some approximations, a useful quantity to define is the Rayleigh coefficient

$$r_{\mathbf{A}}(\mathbf{x}) = \frac{\mathbf{x}^T\mathbf{A}\mathbf{x}}{\mathbf{x}^T\mathbf{x}}.$$

The first method is simple. Power iteration relies on the observation that given a general vector $\mathbf{x}$, repeated multiplication by $\mathbf{A}$ causes the component in the direction of the dominant eigenvalue to grow the fastest.

Algorithm 7 Algorithm PI. Power Iteration

Input: $\mathbf{A} \in \mathbb{R}^{n\times n}$, (we work with symmetric matrices) $\mathbf{z}^{(0)} \in \mathbb{R}^n\backslash\{\mathbf{0}\}, \varepsilon > 0$
Output: $\mathbf{z}^{(k)} \in \mathbb{R}^n, \lambda^{(k)} \in \mathbb{R}$ with $\mathbf{A}\mathbf{z}^{(k)} \approx \lambda^{(k)}\mathbf{z}^{(k)}$
1: $k = 0$
2: $\lambda^{(0)} = r_{\mathbf{A}}(\mathbf{z}^{(0)})$
3: **while** $||\mathbf{A}\mathbf{z}^{(k)} - \lambda^{(k)}\mathbf{z}^{(k)}||_2 > \varepsilon$ **do**
4: $\quad k = k + 1$
5: $\quad \mathbf{z}^{(k)} = \mathbf{A}\mathbf{z}^{(k-1)}$
6: $\quad \mathbf{z}^{(k)} = \mathbf{z}^{(k)}/||\mathbf{z}^{(k)}||_2$
7: $\quad \lambda^{(k)} = r_{\mathbf{A}}(\mathbf{z}^{(k)})$
8: **end while**

Power Iteration has cost $C(n) = 4n^2 + 5n + 1$ *per iteration*. On the condition that our guess $\mathbf{z}^{(0)}$ has a component in the $\mathbf{x}_1$ direction and $|\lambda_1| > |\lambda_2| \geq ... \geq |\lambda_n|$,[2] we have that speed of convergence of the eigenvalues and vectors are:

$$||\sigma_k\mathbf{z}^{(k)} - \mathbf{x}_1||_2 \leq \alpha_1 \left(\frac{|\lambda_2|}{|\lambda_1|}\right)^k$$

[2]Note the strict inequality because otherwise we'd get a linear combination of $\mathbf{x}_1, \mathbf{x}_2$ etc. if there were repeated eigenvalues.

$$|\lambda^{(k)} - \lambda_1| \leq \alpha_2 \left(\frac{|\lambda_2|}{|\lambda_1|}\right)^{2k}$$

Where $\sigma_k = \frac{|c_1\lambda_1^k|}{c_1\lambda_1^k}$.

Power Iteration has the problem that it can only easily be used to find λ_1 and λ_n by applying it to $\mathbf{A}^{-1}$. A similar algorithm which can be used is Shifted Inverse Iteration, SII. SII is obtained simply by introducing a shifting parameter s and modifying line 5 of PI to $\mathbf{z}^{(k)} = (\mathbf{A} - s\mathbf{I})^{-1}\mathbf{z}^{(k-1)}$. Essentially, we are applying PI to $(\mathbf{A} - s\mathbf{I})^{-1}$. If we are smart about this, we solve for $(\mathbf{A} - s\mathbf{I})^{-1}$ *once only*, thus incurring a one time cost of order $O(n^3)$ and then retaining the cost per iteration of order $O(n^2)$.

SII converges to whichever $\lambda_a - s$ is closest to 0, unless there is one or more such that $\lambda_i = s$, then it converges to the next closest. Since SII is effectively the same as PI, it converges in the same way, but with $\lambda_2 \to \lambda_a - s$ and $\lambda_1 \to \lambda_b - s$, where λ_b is the next closest eigenvalue to s.

Recall the caveat with PI that it needs the component in the $\mathbf{x}_1$ direction to be non-zero to converge to λ_1. Orthogonal Iteration, OI, applies PI to an orthogonal matrix, so that we get estimates for every eigenvector/eigenvalue *assuming all eigenvalues are unique*. It will converge as fast as each power iteration would. In practice OI is quite unstable, so it is better to use QR factorisation directly.

Algorithm 8 Algorithm QR-Eig. QR algorithm for eigenvalues.

Input: $\mathbf{A} \in \mathbb{R}^{n\times n}$, (we work with symmetric matrices) $\varepsilon > 0$
Output: $\mathbf{\Lambda}^{(k)} \in \mathbb{R}^{n\times n}$, a diagonal matrix whose entries approximate the eigenvalues of $\mathbf{A}$.
1: $k = 0$
2: $\Lambda^{(0)} = \mathbf{A}$
3: **while** $||\mathbf{\Lambda}^{(k)} - \mathbf{\Lambda}^{(k)}||_1 > \varepsilon$ or $k < 1$ **do**
4: $\quad k = k + 1$
5: $\quad$ Compute the QR decomposition $\mathbf{\Lambda}^{(k-1)} = \mathbf{Q}^{(k)}\mathbf{R}^{(k)}$
6: $\quad \mathbf{\Lambda}^{(k)} = \mathbf{R}^{(k)}\mathbf{Q}^{(k)}$
7: **end while**

QR-Eig has cost *per iteration* of order $O(n^3)$ since we have to use MM and QR, which each incur cost of order $O(n^3)$.