https://doi.org/10.26219/heal.aueb.5793

SCHOOL OF INFORMATION SCIENCES & TECHNOLOGY

DEPARTMENT OF STATISTICS

POSTGRADUATE PROGRAM

A Reinforcement Learning approach on matrix triangularization using rotations

by

Alexandros Lolos

A THESIS

Submitted to the Department of Statistics
of the Athens University of Economics and Business
in partial fulfillment of the requirements for
the degree of Master of Science in Statistics

Athens, Greece
November 2023

https://doi.org/10.26219/heal.aueb.5793

https://doi.org/10.26219/heal.aueb.5793

ΣΧΟΛΗ ΕΠΙΣΤΗΜΩΝ & ΤΕΧΝΟΛΟΓΙΑΣ ΤΗΣ ΠΛΗΡΟΦΟΡΙΑΣ
ΤΜΗΜΑ ΣΤΑΤΙΣΤΙΚΗΣ
ΜΕΤΑΠΤΥΧΙΑΚΟ ΠΡΟΓΡΑΜΜΑ

Προσέγγιση της τριγωνοποίησης πινάκων μέσω περιστροφών με την χρήση ενισχυτικής μάθησης

Αλέξανδρος Λώλος

ΔΙΑΤΡΙΒΗ
Που υποβλήθηκε στο Τμήμα Στατιστικής
του Οικονομικού Πανεπιστημίου Αθηνών
ως μέρος των απαιτήσεων για την απόκτηση
Διπλώματος Μεταπτυχιακών Σπουδών στη Στατιστική

Αθήνα
Νοέμβριος 2023

https://doi.org/10.26219/heal.aueb.5793

https://doi.org/10.26219/heal.aueb.5793

ACKNOWLEDGEMENTS

First of all, I would like to thank my advisor, Prof. P. Dellaportas, for giving me the opportunity to work on this thesis. Without his contribution and support this project would have not been possible.

Finally, as always I would like to thank my parents and my brothers for their unwavering support and love throughout my life.

https://doi.org/10.26219/heal.aueb.5793

https://doi.org/10.26219/heal.aueb.5793

ABSTRACT

Alexandros Lolos

A Reinforcement Learning approach on matrix triangularization using rotations

November 2023

In this thesis, we will discuss in detail about Householder transformation, Givens rotations, Cordic Givens rotations and Gram-Schmidt, along with its modified variant. All of the mentioned methods are QR decomposition methods with differences in complexity and error. Furthermore, we will discuss about the basic theory of Reinforcement Learning and explore the differences between algorithms such as SARSA and Q-Learning.

In the experimentation section we will design the QR decomposition as a board game. With the use of Reinforcement Learning and specifically the Q-Learning algorithm we will guide the Givens rotation into alternative paths for our algorithms, that in specific matrix input cases can reach the QR decomposition state in less iterations.

https://doi.org/10.26219/heal.aueb.5793

https://doi.org/10.26219/heal.aueb.5793

ΠΕΡΙΛΗΨΗ

Αλέξανδρος Λώλος

Προσέγγιση της τριγωνοποίησης πινάκων μέσω περιστροφών με την χρήση ενισχυτικής μάθησης

Νοέμβριος 2023

Σε αυτήν τη διπλωματική εργασία θα μελετήσουμε λεπτομερώς τον μετασχηματισμό Householder, τις περιστροφές Givens, τις περιστροφές Cordic Givens, τον αλγόριθμο Gram-Schmidt και την τροποποιημένη έκδοσή του. Όλες αυτές οι μέθοδοι στις οποίες αναφερθήκαμε μέχρι στιγμής είναι μέθοδοι για την παραγοντοποίηση QR με διαφορές σε πολυπλοκότητα και σφάλμα. Επίσης θα συζητήσουμε για την βασική θεωρία της ενισχυτικής μάθησης και θα εξερευνήσουμε τις διαφορές μεταξύ αλγορίθμων όπως ο SARSA και ο Q-Learning.

Στο κεφάλαιο των πειραματισμών θα σχεδιάσουμε την παραγοντοποίηση QR σαν ένα επιτραπέζιο. Με τη χρήση ενισχυτικής μάθησης και συγκεκριμένα του Q-Learning αλγορίθμου θα οδηγήσουμε τις περιστροφές Givens σε εναλλακτικά μονοπάτια για τον αλγόριθμό μας, όπου για συγκεκριμένες εισόδους πινάκων θα οδηγηθούν στη παραγοντοποίηση QR σε λιγότερες επαναλήψεις.

https://doi.org/10.26219/heal.aueb.5793

https://doi.org/10.26219/heal.aueb.5793

Contents

https://doi.org/10.26219/heal.aueb.5793

https://doi.org/10.26219/heal.aueb.5793

List of Figures

https://doi.org/10.26219/heal.aueb.5793

List of Algorithms

https://doi.org/10.26219/heal.aueb.5793

Chapter 1

Introduction

1.1 Motivation

The QR decomposition holds a significant place in the history of the solutions of systems of linear equations which often manifests as a consequence of the minimization of the least squares and the eigenvalue and eigenvector problem. While QR decomposition dates back to 1961 [3] as per Gilbert Strang [1] it "is still considered one of the most important methods developed so far" and it is widely used in many library packages such as LAPACK.

As with all tools, similarly with algorithms we want to refine them to perform fast, accurately and be precise no matter the instance under which they operate. The term 'fast' is often conceived as a time polynomial in relation to the iterations our algorithm has to perform.

In our case, QR decomposition is indeed in a cubic relationship to the input of the data assuming a square matrix of size n. While this boundary may seem trivial in comparison to a lot of non-polynomial algorithms, when we need to use the same algorithm a countless amount of times then every bit of reduction in complexity, would have a significant impact.

While the long arc of the development of numerical algorithms by traditional techniques of human ingenuity has led to very significant achievements over the past decades, there are cases where the problem becomes so complex that we are in need of modern approaches.

In recent years, the incorporation of predictions in algorithm design has gained a significant popularity [8], however in order to predict a proper pathing for our algorithm we need to accumulate data in a reasonable manner.

By using Reinforcement Learning we can explore alternative paths for the completion of the QR decomposition in instances where we can perform multiple steps at the cost of a single step. These paths would be undetectable under sequential order QR factorization

https://doi.org/10.26219/heal.aueb.5793

methods such as Givens rotations or Householder transformation. Under this premise we can gather data that not only executes the algorithm, but actually performs the algorithm faster than its sequential counterparts in cases where we do not have multiplicative machines to perform operations.

A newfound perspective can be gained in incorporating board game solutions into numerical algorithm problems [11], where they utilize the AlphaZero framework [13] developed by DeepMind for board games in order to achieve faster convergence of matrix diagonalization in Hamiltonian matrices. By using Reinforcement learning, they managed to find paths that, in most cases, required less steps to achieve sufficient convergence of the matrix diagonalization (eigenvalues) compared to algorithms used in practice.

The thought process used in this publication consists of designing the rules required to perform the Givens rotations (Jacobi iteration method) as a board game. The purpose is to zero out elements simultaneously in the upper and lower diagonal parts of the matrix by applying the rotation on both sides of the matrix $\mathbf{G}^\intercal\mathbf{A}\mathbf{G}$, while choosing the path that achieves the most reduction of the non diagonal elements to zero, while satisfying enough strengthening of the main diagonal. This approach incorporates the framework mentioned earlier.

Furthermore, publications by DeepMind [2] [7], have shed light on the use of Reinforcement Learning in the discovery of novel techniques to achieve better performances on matrix multiplications and sorting.

1.2 Contribution

We set out to determine if a tool such as Reinforcement Learning can be used in this specific endeavour to reduce the complexity of performing the QR decomposition on matrices with specific properties. Given the premises under which this research was conducted, the answer would be negative in cases where the matrices do not possess a specific underlying pattern. Meanwhile, in cases where the matrices contain routes that lead to a possible path of less iterative steps to accomplish a matrix QR decomposition, the results seem promising. However, we have to account for the broad ecosystem that is developed around such an established method as QR decomposition in order for our results to qualify as positive.

A small gain in constant complexity of the non-parallel Givens rotation can be achieved by following a different path to that of the sequential Givens rotation, in cases where the matrix input is discrete with relatively small range to that of the size of the input.

Furthermore, we provide a Givens rotations modification to fit in the necessities of

https://doi.org/10.26219/heal.aueb.5793

specifying QR decomposition as a board game. Additionally, the Cordic algorithm has been adjusted to fit the purposes of the board game for future development.

1.3 Thesis Outline

The chapters contained in this thesis have the following outline:

Chapter 2: We present the theory and algorithms of the most used QR decomposition methods, accompanied with modified versions of those algorithms in order to fit the board game needs.

Chapter 3: The theory and definitions of Reinforcement Learning that were deemed necessary for the development of the experiment are presented here.

Chapter 4: QR decomposition is designed as a board game; the rules are being fleshed out, and there is a discussion on the difficulties that emerged in the process. Furthermore we present the results of the experiments.

Chapter 5: We discuss conclusions and future work.

https://doi.org/10.26219/heal.aueb.5793

Chapter 2

QR Decomposition

The story of QR decomposition dates back to many decades and intersects with many important problems throughout the history of numerical computations. Two of those important problems are the minimization of least squares which coincides with the solution for large systems of linear equations, and the eigenvalue-eigenvector problem [1].

In this chapter we will discuss about algorithms such as the Givens rotation and the Cordic givens rotation and modify them according to the needs of the board game we will create in Chapter 4. Furthermore, we will discuss algorithms such as the Householder transformation and the classical and modified Gram Schmidt algorithm.

Moreover, we will dedicate a short segment about the differences in complexity between most of the algorithms mentioned so far, at the end of this Chapter.

2.1 Givens Rotations

2.1.1 Brief Introduction

As stated in the explanations for Givens rotations given [4] and [1], one such rotation will take a 2-dimensional vector $\begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix}$ and rotate it, on the plane it resides, by a degree of ϕ. By performing this task, we now have a tool at our disposal in order to rotate the vector in an angle such that it will reside on one of the axes. By doing so, the vector will now have coordinates $\begin{bmatrix} 0 \\ \mathbf{y}' \end{bmatrix}$ or $\begin{bmatrix} \mathbf{x}' \\ 0 \end{bmatrix}$. This allows us to perform operations on our matrix in order to drive it into creating an upper triangular form where all the cells below the main diagonal are set to zero.

While it is as simple as that, let's consider all the specifics that support the whole mechanism.

https://doi.org/10.26219/heal.aueb.5793

2.1.2 Definition and basic properties

According to [4], given a vector $\mathbf{a} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} \in \mathbb{R}^2$, there exists an orthogonal matrix $\mathbf{G}$ such that $\mathbf{G}^\intercal \mathbf{a} = \begin{bmatrix} \pm\|\mathbf{a}\|_2 \\ 0 \end{bmatrix}$.

$\mathbf{G} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}$ where $c^2 + s^2 = 1$. Then

$$\mathbf{G}^\intercal \mathbf{G} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} c & -s \\ s & c \end{bmatrix} = \begin{bmatrix} c^2 + s^2 & 0 \\ 0 & c^2 + s^2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

which means that a Givens rotation is an orthogonal matrix. By rotating in each way we can "create" zeros in any of our vector coordinates. As stated by [1, equation 3.29]

$$\begin{bmatrix} \pm\|\mathbf{a}\|_2 \\ 0 \end{bmatrix} = \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix}$$

in order to now identify the appropriate angle [1] we have

$$\cos\phi = \frac{a_1}{r}, \qquad \sin\phi = \frac{a_2}{r}, \qquad \text{where } r = \sqrt{{a_1}^2 + {a_2}^2}.$$

In the usual case, the Givens rotation algorithm will start its journey from the $a_{2,1}$ cell, pair it with the $a_{1,1}$ cell into a 2-d vector $\mathbf{a}$ and eliminate the $a_{2,1}$ entry as explained above and rotate all the 2-d vectors that belong in the same rows by a degree ϕ.

https://doi.org/10.26219/heal.aueb.5793

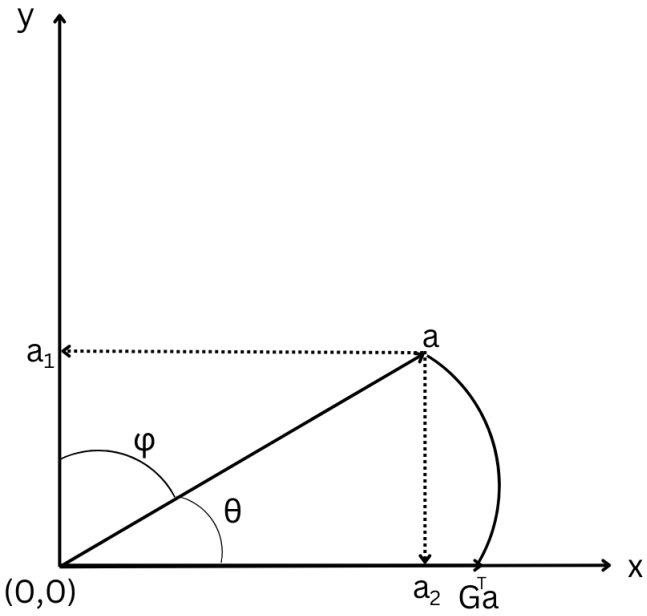


Figure 2.1: Vector Givens rotation. Adapted from [1, figure (1.3), p. 11]

Let's consider the notation $a_{1,m}^{(\phi)}$, an element that its 2-d vector was rotated by ϕ degrees.

$$\begin{bmatrix} \cos\phi & -\sin\phi & \dots & 0 \\ \sin\phi & \cos\phi & & \vdots \\ \vdots & & 1 & \\ 0 & \dots & & 1 \end{bmatrix} \begin{bmatrix} a_{1,1} & \dots & & a_{1,m} \\ a_{2,1} & & & \\ \vdots & & & \vdots \\ a_{n,1} & a_{n,2} & \dots & a_{n,m} \end{bmatrix} = \begin{bmatrix} \pm\|\mathbf{a}\|_2 & \dots & & a_{1,m}^{(\phi)} \\ 0 & & & \\ \vdots & & & \vdots \\ a_{n,1} & a_{n,2} & \dots & a_{n,m} \end{bmatrix}$$

The next step of the algorithm would be to perform the same task again, pairing the cell $a_{3,1}$ with the cell $a_{1,1}$ which has now become the norm of the previous 2-d vector. That way it will continue the entire process until it has zeroed out all the elements below $a_{1,1}$.

This process is strengthening the cell at $i = 1, j = 1$, until the iterative process leads $a_{1,1}$ into becoming the norm of the n dimensional vector $\mathbf{a}_n = (a_{1,1}, a_{2,1}, ...a_{n,1})$ while performing ϕ degree rotation in each two row pair 2-d vector next to the 2-d vector we work with.

After the process of zeroing out one column the final matrix will look like this:

$$\begin{bmatrix} \|\mathbf{a}\|_n & \dots & a_{1,m}^{(\phi_1)} \\ 0 & & a_{1,m}^{(\phi_1 \& \phi_2)} \\ \vdots & & \vdots \\ 0 & a_{n,2}^{(\phi_{n-1})} & \dots & a_{n,m}^{(\phi_{n-1})} \end{bmatrix}$$

https://doi.org/10.26219/heal.aueb.5793

2.1.3 The Algorithm

For our case of the problem, we want an algorithm that will have the relaxation of choosing the 2-d vector among all the below diagonal entries without explicitly pairing the cell with the $a_{1,j}$ component. In that case, we will have the notation of k (shift parameter) that exhibits the distance in terms of rows between the chosen element to zero out and the element that will now possess its magnitude.

It is important to also consider fixing the numerical instability of the original Givens rotation algorithm. In cases where the denominator is smaller, this seems to render poor results according to Gene H. Golub in the book Matrix computations [5, section 5.1.11].

While this is not the original algorithm, as it would not traverse the matrix and zero out each lower diagonal element sequentially, one could trivially implement it to do so.

The modified version of the algorithm is the one that will be used in the experiment and it was considered redundant to have both in this thesis.

Algorithm 1: Givens Rotations QR Decomposition

Input : A - input matrix, i - row index, j - column index, k - shift parameter
Result: Transformed matrix A^*, Rotation matrix G

$x = A_{i-k,j}$
$y = A_{i,j}$
if $|x| > |y|$ **then**
  $t = \frac{y}{x}$
  $c = \frac{1}{\sqrt{1+t^2}}$
  $s = c \cdot t$
else
  $t = \frac{x}{y}$
  $s = \frac{1}{\sqrt{1+t^2}}$
  $c = s \cdot t$

Unstable Version
$r = \sqrt{y^2 + x^2}$
$c = \frac{x}{r}$
$s = \frac{-y}{r}$

$G =$ Identity matrix of size $n \times n$
$G_{i,i} = c$, $G_{i-k,i-k} = c$,
$G_{i-k,i} = s$, $G_{i,i-k} = -s$
$A^* = G \cdot A$
return G, A^*

https://doi.org/10.26219/heal.aueb.5793

2.2 Cordic Givens Rotation

Again this method provides us with the computation of the QR decomposition. While a lot of the methods discussed here require the computation of operations such as roots, which are quite inefficient, an advantage that Cordic Givens rotations provides is that it requires only shifts and additions in order to rotate vectors [17] [10].

Instead of using roots as in the usual Givens rotation algorithm, we can approximate those values by iteratively converging towards the optimal degree of rotation. These angle adjustments, through hardware friendly operations, make this method suitable for implementation in systems with limited computational capacity.

As with the Given rotation algorithm, Cordic Givens rotation is also modified to fit the purposes of the board game discussed in Chapter 4, where the Reinforcement learning agent needs the relaxation to choose the vector pair in any square of the matrix. As such, this algorithm is implemented to require as input the (i, j, k) coordinates which signify the specific lower triangular element and also the distance k where we pair the 2-d vector in hopes to tackle many collinear vectors in one iteration.

Algorithm 2: Cordic Givens Rotations QR Decomposition

Input : A - input matrix, i - row index, j - column index, k - shift parameter, l - number of iterations

Result: Transformed matrix A^*, Rotation matrix G

$\mathrm{Nc} = \prod_{i=0}^{l-1} \frac{1}{\sqrt{1+2^{-2i}}}$

m = # of cols of A

n = # of rows of A

G = Identity matrix of size $n \times n$

$x, y, u, v =$ **Rotation Mode**$(A_{i-k,j:\mathsf{m}}, A_{i,j:\mathsf{m}}, G_{,i-k}, G_{,i}, l, \mathrm{Nc})$

$A_{i-k,j:\mathsf{m}} = x$

$A_{i,j:\mathsf{m}} = y$

$G_{,i-k} = u$

$G_{,i} = v$

return G, A^*

https://doi.org/10.26219/heal.aueb.5793

Algorithm 3: Cordic Givens Rotation Mode

Input: x - input matrix, y - row index, u - column index, v - shift parameter, l - number of iterations

if $x_1 < 0$ **then**
  $x = -x$, $u = -u$, $y = -y$, $v = -v$

for i *in* 0 *to* l **do**
  $x_o = x$, $u_o = u$
  if $y_1 < 0$ **then**
    $x = x - y\ 2^{-i}$, $u = u - v\ 2^{-i}$, $y = y + x0\ 2^{-i}$, $v = v + u0\ 2^{-i}$
  else
    $x = x + y\ 2^{-i}$, $u = u + v\ 2^{-i}$, $y = y - x0\ 2^{-i}$, $v = v - u0\ 2^{-i}$

$y_1 = 0$

$x = \mathrm{Nc}\ x$, $u = \mathrm{Nc}\ u$, $y = \mathrm{Nc}\ y$, $v = \mathrm{Nc}\ v$

return x, y, u, v

https://doi.org/10.26219/heal.aueb.5793

2.3 Householder Reflections

2.3.1 Brief Introduction

While Givens rotation is one way to perform a QR decomposition of a matrix by iteratively zeroing out elements by rotating them in a certain angle while working with 2-d vectors, the Householder transformation, also known as reflection, can give us several advantages when we need to zero out the whole column of a matrix, thus reducing the amount of iterations of matrix multiplications we need to perform in comparison to the Givens rotation.

The idea is very similar, we can find a Householder transformation such that the vector of interest will now reside on one of the axes of the n dimensional coordinate system.

2.3.2 Definition and basic properties

As stated by [4], with a slight manipulation of the definition to fit the purposes of the real case, we have the following Definition (3.3.2.2): Let $u \in \mathbb{R}^n$ be a vector of unit length ($\|u\|_2 = 1$). Then $H = I - 2uu^\intercal$ is said to be a Householder transformation or (Householder) reflector. The exact same can be said for the case where $u \in \mathbb{C}^n$ by using the Hermitian transpose instead, but the complex domain is out of the scope of our interest.

- Any vector x that is perpendicular to u is left unchanged:

$$(I - 2uu^\intercal)x = x - 2u(u^\intercal x) = x.$$

- Any vector x can be written as $x = x + u^\intercal xu$ where x is perpendicular to u, and $u^H xu$ is the component of x in the direction of u. Then

$$(I - 2uu^\intercal)x = (I - 2uu^\intercal)(x + u^\intercal xu) = x + u^\intercal xu - 2uu^\intercal xu$$

$$= x + u^\intercal xu - 2u^\intercal xu^\intercal u = x - u^\intercal xu.$$

The Householder transformation used for the purposes of the QR decomposition is a special case, where we can find a hyperplane s.t it mirrors our original vector $\mathbf{x}$ onto one of the axis coordinates. The now reflected vector $\mathbf{x}'$ maintains its original magnitude but becomes linearly dependent on the first standard basis vector. u is perpendicular to the

https://doi.org/10.26219/heal.aueb.5793

space with respect to which we will reflect.

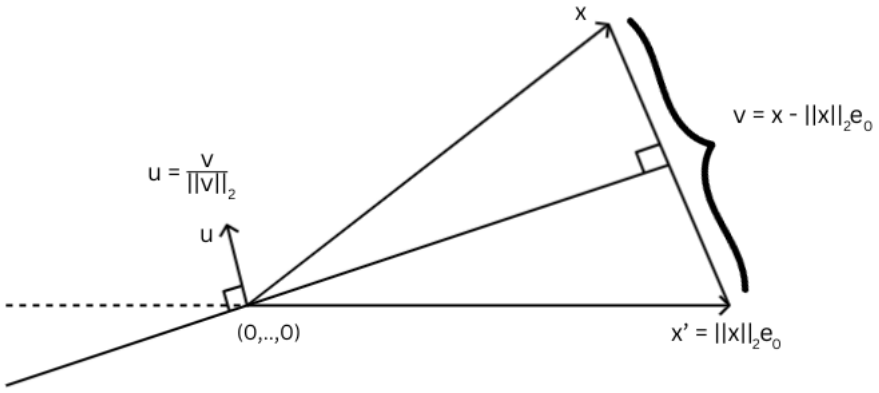


Figure 2.2: Vector hyperplane reflection. Adapted from [4, figure (3.3.2.1), p. 120]

$$\mathbf{v} = \mathbf{x} - \|\mathbf{x}\|_2 e_0 \text{ or } \mathbf{v} = \mathbf{x} - u^\intercal \mathbf{x} u, u = \mathbf{v}/\|\mathbf{v}\|_2$$

Now that we have all the information we need, by simple vector operations we get that $\mathbf{x} + \mathbf{x} - 2u^\intercal \mathbf{x} u = (I - 2uu^\intercal)\mathbf{x}$, where the orthogonal matrix we will use to reflect our vector is $H = (I - 2uu^\intercal)$ and the orthogonal matrix Q which will be the matrix-matrix product of all those H reflections will be $H_n \dots H_2 H_1$.

Algorithm 4: Householder QR Decomposition

Data: A_{nxm} matrix
Result: Q, R
for k *in* $1 : m$ **do**
  $x = A_{k:n,k}$
  $a = \|\mathbf{x}\|_2$
  $e_k =$ standard basis vector k
  $u = x - a \cdot e_k$
  $v = u/\|u\|_2$
  $H = I - 2uu^\intercal$
  $R = H \cdot A$
  $Q = H \cdot Q$
end

https://doi.org/10.26219/heal.aueb.5793

2.4 Modified Gram Schmidt

2.4.1 Brief Introduction

The modified Gram Schmidt (MGS) method differs almost nowhere in comparison to the classical Gram-Schmidt (GS) method, expect for an error correction step which proves to have significant value when it comes to the accumulated error of the floating-point arithmetic that adds up in the GS method.

From a mathematical perspective the properties of the GS method hold true. If someone could perform the computation while having infinite precision, the resulting Q matrix would be orthogonal and would be the same in both methods. Due to the use of modern computers, the method is impacted by numerical instability and often loss of orthogonality. This leads to the requirement of extra steps in order to produce meaningful results [[1], p.9].

While this modified version provides a significant reduction in numerical instability, it is still considered inferior to other methods mentioned such as Givens rotation and Householder reflections both in numerical stability as well as in practical applications.

While the differences between each QR method's floating-point arithmetic error are out of the scope of this thesis, one could refer to [12] for a formal comparison.

2.4.2 Definition and basic properties

As stated by [1], the classical Gram-Schmidt orthogonalization method is a way to obtain orthonormal vectors $Q_k = \{q_1, q_2, ..., q_k\}$, $1 \leq k \leq N$, where as N we denote the number of columns of our matrix.

In order to produce those orthonormal vectors, we can sequentially subtract from our columns $A_k = \{a_1, a_2, ..., a_k\}$, their projection onto the space Q_{k-1}: $a_k = a_k - \sum_{j=1}^{k-1} \text{proj}_{a_j}(a_k)$.

In this way, our matrix A will end up being the Q orthogonal matrix. However, due to the accumulation of rounding errors, this matrix will not be quite orthogonal and is often in need of re-orthogonalization.

In modified Gram-Schmidt (MGS), for each vector we modify all the subsequent vectors to be orthogonal to our current vector. So when we adjust a vector with MGS we use this vector to adjust all the subsequent vectors too.

This change in the order of computations will introduce smaller errors in comparison to orthogonalizing our vector to all previous orthogonal vectors all at once.

This is a modified version of the MGS algorithm that overwrites matrix A (a) with

https://doi.org/10.26219/heal.aueb.5793

the orthogonal matrix Q (a) [[4], p.112]. It does not change anything in terms of complexity beyond the fact that it lowers the memory consumption of having to manage two large matrices instead of one.

Algorithm 5: Modified Gram-Schmidt QR Decomposition

Data: Input matrix A (a) with n rows and m columns
Result: Orthogonal matrix Q (a) and upper triangular matrix R (ρ)
for $j = 1$ *to* m **do**
  for $k = 1$ *to* $j - 1$ **do**
    $\rho_{k,j} = \langle a_{k,}, a_{,j} \rangle$
    $a_{,j} = a_{,j} - \rho_{k,j} a_{k,}$
  end
  $\rho_{j,j} = \|a_{,j}\|_2$
  $a_{,j} = \dfrac{a_{,j}}{\rho_{j,j}}$
end

2.5 Complexity comparison

Here, we will compare the algorithms according to floating-point operations, or flops, that the algorithm requires. All basic operations: addition, subtraction, multiplication, division and the square root, count as one flop [15].

For a general $n \times m$ matrix where $m \leq n$,

Algorithm	$n \times m$	$n \times n$
Householder	$2nm^2 - \frac{2}{3}m^3$	$\frac{4}{3}n^3$
Givens	$3nm^2 - m^3$	$2n^3$
MGS	$2nm^2$	$2n^3$
GS	$2nm^2$	$2n^3$

Table 2.1: Comparison of the flop count between different QR decomposition algorithms

Householder [16]

The dominant terms for each iteration are engulfed in the matrix-matrix product of H and A in order to produce $R = H \cdot A$, which is the upper triangular matrix.

Considering $H = I - 2uu^\intercal$, at iteration k we get:

- $(2(n - k + 1) - 1)(m - k + 1)$ flops for the product of $u^\intercal$ and A

https://doi.org/10.26219/heal.aueb.5793

- $(n-k+1)(m-k+1)$ flops for the outer product of u and $u^\intercal \cdot A$
- $(n-k+1)(m-k+1)$ flops for the subtraction

which we can sum to roughly $4(n-k+1)(m-k+1)$ flops for iteration k.

Total flops: $\sum_{k=1}^{m} 4(n-k+1)(m-k+1) \approx 2nm^2 - \frac{2}{3}m^3$
For the special case where $n = m$ (square matrix) we get $\frac{4}{3}n^3$ flops.

Givens [6]

In the case of Givens rotations we need to count the flops of the operations of the product of a 2×2 matrix with a 2×1 vector, which results in four multiplications and two additions. At iteration k we get $6(n-k+1)(m-k+1)$ flops.

Total flops: $\sum_{k=1}^{m} 6(n-k+1)(m-k+1) \approx 3nm^2 - m^3$

Modified Gram-Schmidt [15]

- $\rho_{k,j} = \langle a_{k,}, a_{,j} \rangle$: n multiplications, n-1 subtractions.
- $a_{,j} = a_{,j} - \rho_{k,j} a_{k,}$: n multiplications, n subtractions.

A single column vector element requires 4n flops, then the total flops are $\sum_{j=1}^{m} \sum_{k=1}^{j-1} 4n = 2nm^2$. Additionally, for the special case where $n = m$ (square matrix) we get $2n^3$ flops. The unmodified version of the algorithm has the same exactly flop count since there is almost no difference besides the error correction step.

https://doi.org/10.26219/heal.aueb.5793

Chapter 3

Reinforcement Learning

Reinforcement Learning (RL) is one of the areas of Machine Learning. In RL an agent can interact with his environment, take actions and learn by the experience he accumulates throughout this process. By arriving in many different states, he can learn throughout the feedback process of rewards. The rewards will signify if the agent is doing well in his journey or not. Through trial and error the agent manages to often find optimal paths in order to arrive at his end goal. This Chapter is a summary of information from [14].

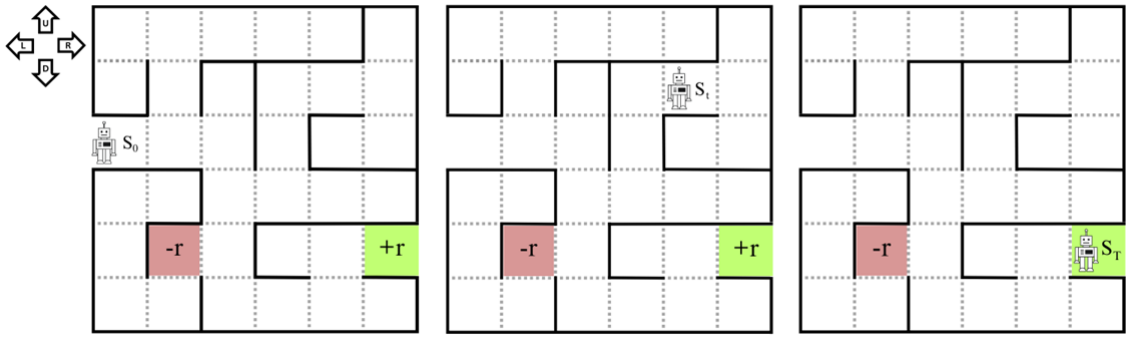


Figure 3.1: Different states through a maze.

3.1 Markov Decision Process

The definition of the Markov Decision Process for Reinforcement Learning is a summary of [14].

- Set of states $s \in S$
- Set of actions $a \in A$

https://doi.org/10.26219/heal.aueb.5793

- Transition Probability: The probability of moving from state s to state s' given that we chose action a at state s.

$$p(s', r|s, a) = Pr(S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a),$$
$$\sum_{s \in S} \sum_{r \in R} p(s', r|s, a) = 1, \quad \forall s \in S, a \in A(s)$$

- Reward Function: Defines the reward r the agent receives upon arrival at the state s' having taken action a at state s.

$$r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}, r(s, a, s') = \mathbb{E}[R_t|S_t = s, A_t = a, S_{t+1} = s']$$

- Discount Factor γ

3.1.1 Discounted Rewards

The goal of our agent is to maximize the cumulative reward in the long run by prioritizing future rewards over short term rewards. It is important to formalize a definition of the sum of the rewards we can get throughout our environment's possible paths:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T \tag{3.1}$$

We can break the reward into chunks of rewards for each time step that self define the notion of the episode. The first reward, either negative or positive, starts at time step $t + 1$ and ends at time step T.

Now, each of our episodes will have a step horizon (finite or infinite) and we get the formal definition of the sum of rewards, which we want to maximize throughout the series of episodes, in which our agent trains.

The episode types can be classified as episodic tasks and continuing tasks, where the first has a clear depiction of the end state of all episodes, while the latter does not break down clearly into separate episodes and can continue indefinitely.

By using the discount factor γ, we can fine tune our agent into being shortsighted or farsighted. An agent with a $\gamma = 1$ will consider strongly the future rewards, in contrast to an agent with $\gamma = 0$ that explicitly considers momentary gains.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-2} R_T = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{3.2}$$

If the reward is constant even if the time horizon of possible steps is infinite, by using

https://doi.org/10.26219/heal.aueb.5793

a discount factor $\gamma < 1$, we can bound the cumulative reward to be finite.

$$G_t = \sum_{k=0}^{\infty} \gamma^k = \frac{1}{1-\gamma} \tag{3.3}$$

3.1.2 Policies and Value functions

A value function in the case of Reinforcement Learning can capture the quality of an agent's position in any given state.

$$v_\pi(s) = E_\pi[G_t|S_t = s] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s] \quad \forall s \in S \tag{3.4}$$

As defined in (3.4), the value function under a policy π given a state s is the expected reward we will collect by following that specific policy starting from state s, while the notion of taking an action given a specific state under a specific policy will be defined as:

$$q_\pi(s,a) = E_\pi[G_t|S_t = s, A_t = a] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a] \quad \forall s \in S \tag{3.5}$$

A fundamental property of Reinforcement Learning is the need for satisfaction of a recursive relationship between s and its successor states, known as the **Bellman equation** for u_π:

$$\begin{aligned} u_\pi(s) &= E_\pi[G_t|S_t = s] \\ &= E_\pi[\sum_{k=0}^{\infty} R_{t+1} + \gamma G_{t+1}|S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma E_\pi[G_{t+1}|S_{t+1} = s']] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma u_\pi(s')] \quad \forall s \in S \end{aligned} \tag{3.6}$$

This equation averages over all possibilities with weight for each possibility the probability of occurring.

Policies can be deterministic or stochastic. In deterministic policies the same action will be taken when we arrive in a specific state, while in stochastic policies we will choose the action derived from a probability distribution in any given state.

https://doi.org/10.26219/heal.aueb.5793

3.2 Temporal Difference

Temporal Difference methods (TD) do not require a model of the environment in order to work. As such, there is no need for the transition probabilities and the rewards that would come after them.

TD methods work in an online fully incremental fashion. While with other methods such as Monte Carlo, we must wait until the end of the episode in order to get the return, working with the TD methods we only must wait one time step. Learning from each step of the episode can be very beneficial in cases where the episodes are very long or do not end at all.

By denoting the parameter λ, in TD(λ) we get an online update method that updates the estimates in each step with TD(0), or every λ steps for TD(λ) and for $\lambda = \mathrm{T}$ we get the Monte Carlo method, where at step T we conclude the end of the episode. This way we can see that the Monte Carlo method is a special case of the more general TD method group.

While the TD methods offer far greater generalization, they introduce greater variance in comparison to the Monte Carlo method.

3.2.1 Sarsa and Q-Learning

Here we will discuss about Sarsa and Q-Learning, two well known reinforcement learning algorithms. Both methods are Temporal Difference methods that update their learning in each step of the episode, but present key differences between them.

The key differences between the Sarsa and the Q-learning algorithm are:

- Sarsa is on-policy while Q-learning is off-policy:

  This means that for Sarsa the policy that is trying to learn and the one that chooses the next actions are the same, while for Q-Learning the policy that is trying to learn can be different from the one the makes the decisions. Q-Learning basically learns the optimal policy by using the maximum reward from the next state under any action regardless of the action the agent will take.

- $\gamma Q(S', A')$ and $\max_a Q(S', a)$

  Sarsa updates the value function's state-action pair using a discounted next state-action pair, while Q-learning updates the current state-action pair using the max value of the next state-action pair.

https://doi.org/10.26219/heal.aueb.5793

- Risk differences

  As discussed in the Cliff Walking example in [14], Sarsa is a more conservative method, because Sarsa learns by taking "safe" paths. As such, in environments with highly negative reward areas, it seems to outperform Q-Learning, while Q-Learning, on the other hand, takes risks more often trying to maximize the reward in the process and ends up overestimating the values of certain situations. In general settings Q-Learning seems to perform the best amongst those two methods. The absence in risk in our problem formulation in Chapter 4 is the reason we will use Q-Learning as our method of choice.

Algorithm 6: Sarsa (On-policy TD control) for estimating q^*

Data: Algorithm parameters: step size $\alpha \in (0, 1]$, small $\epsilon > 0$

Result: Learned Q-values $Q(s, a)$

Initialize $Q(s, a)$ for all $s \in S^+$, $a \in A(s)$ arbitrarily except that $Q(\text{terminal}, \cdot) = 0$

while *not converged* **do**

  Initialize state S

  Choose action A from state S using a policy derived from Q (e.g., ϵ-greedy)

  while S *is not terminal* **do**

    Take action A, observe reward R, and new state S';

    Choose action A' from state S' using a policy derived from Q (e.g., ϵ-greedy)

    $Q(S, A) \leftarrow Q(S, A) + \alpha \left[R + \gamma Q(S', A') - Q(S, A)\right]$

    $S \leftarrow S'$

    $A \leftarrow A'$

[14]

https://doi.org/10.26219/heal.aueb.5793

Algorithm 7: Q-learning (Off-policy TD control) for estimating π^*

Data: Algorithm parameters: step size $\alpha \in (0, 1]$, small $\epsilon > 0$

Result: Learned Q-values $Q(s, a)$

Initialize $Q(s, a)$ for all $s \in S^+$, $a \in A(s)$ arbitrarily except that $Q(\text{terminal}, \cdot) = 0$

while *not converged* **do**

  Initialize state S

  while S *is not terminal* **do**

    Choose action A from state S using a policy derived from Q (e.g., ϵ-greedy)

    Take action A, observe reward R, and new state S'

    $Q(S, A) \leftarrow Q(S, A) + \alpha \left[R + \gamma \max_a Q(S', a) - Q(S, A)\right]$

    $S \leftarrow S'$

[14]

As we will see in our next Chapter, in environments with rich state-action spaces, it is often the case to substitute the Q-table, where we save the Q-values for every state-action pair, with approximators like neural networks. This way we can estimate the Q-values for each state-action pair without the need of exponentially larger memory in cases where the environment gets bigger according to input. While for our case the problem did not dictate further development with reinforcement learning, in the case where our findings were fruitful, in matrices larger than 7×7 or more we would need to rely on estimators for our Q-tables because the memory's complexity is roughly $\mathcal{O}(2^n)$ for a matrix with n rows and columns.

https://doi.org/10.26219/heal.aueb.5793

Chapter 4

QR Decomposition as a board game

In this experiment, we will consider combining Reinforcement Learning with the QR decomposition in order to find alternative paths that may lead to reduction of the steps needed by our algorithm.

In the usual case of Givens Rotation, the algorithm would zero out elements sequentially, starting from the top left corner and ending at the bottom right, where we finally get the upper triangular matrix.

$$\begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{bmatrix} \xrightarrow{\textbf{Step 1}} \begin{bmatrix} x & x & x & x & x \\ 0 & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{bmatrix} \xrightarrow{\textbf{Step 2}} \begin{bmatrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{bmatrix} \xrightarrow{\textbf{Step 3}} \dots \xrightarrow{\textbf{Step 4}} \begin{bmatrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \end{bmatrix}$$

$$\xrightarrow{\textbf{Step 5}} \begin{bmatrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & 0 & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \end{bmatrix} \xrightarrow{\textbf{Step 6}} \dots \xrightarrow{\textbf{Step 10}} \begin{bmatrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & 0 & x & x & x \\ 0 & 0 & 0 & x & x \\ 0 & 0 & 0 & 0 & x \end{bmatrix}$$

Or, with a slight modification of the loop statements could start from the bottom left and end again at the bottom right. This difference in the path of the algorithm will not affect the solution of the upper triangular matrix, since the solution is unique in cases where no two columns are linearly dependent.

$$\begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{bmatrix} \xrightarrow{\textbf{Step 1}} \begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ 0 & x & x & x & x \end{bmatrix} \xrightarrow{\textbf{Step 2}} \begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \end{bmatrix} \xrightarrow{\textbf{Step 3}} \dots \xrightarrow{\textbf{Step 4}} \begin{bmatrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \end{bmatrix}$$

$$\xrightarrow{\textbf{Step 5}} \begin{bmatrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & 0 & x & x & x \end{bmatrix} \xrightarrow{\textbf{Step 6}} \dots \xrightarrow{\textbf{Step 10}} \begin{bmatrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & 0 & x & x & x \\ 0 & 0 & 0 & x & x \\ 0 & 0 & 0 & 0 & x \end{bmatrix}$$

Now, let's consider how our algorithm can zero out elements. As explained in the theory for Givens Rotation in Chapter 3, we can rotate a 2-d vector in such a way that it would now reside on one of the axes without loss of information, because the magnitude would be preserved. Let's consider, for simplicity, a dense (without zeros) 5×5 matrix with discrete entries.

https://doi.org/10.26219/heal.aueb.5793

$$A = \begin{bmatrix} 1 & 2 & 1 & 4 & 7 \\ 3 & 4 & 5 & 1 & 1 \\ 6 & 11 & 2 & 7 & 1 \\ 2 & -3 & 1 & 7 & -1 \\ 8 & -9 & -3 & 1 & -3 \end{bmatrix}$$

Now, in order to get the first step for our matrix, we will calculate and then apply a Givens Rotation that follows the theory outlined in Chapter 3.

$$G = \begin{bmatrix} c & s & 0 & 0 & 0 \\ -s & c & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

c = 0.3162278, s = 0.9486833

Considering the matrix multiplication rules, this rotation will not only affect the 2-d vector that we want to change from $\begin{bmatrix} \mathbf{x_1} \\ \mathbf{x_2} \end{bmatrix}$ to $\begin{bmatrix} \|\mathbf{x}\|_2 \\ \mathbf{0} \end{bmatrix}$, but it will also affect all the 2-d vectors from the first two rows of our matrix, by rotating them by an angle of $\phi = 71.56505°$.

As we can see in Figure 4.1 the vector with coordinates (1,3) needs to be rotated by ϕ. Following on the matrix transformation performed by Givens Rotation, we can see how the coordinates of the values will change, considering the last two lines being 2-d vectors.

$$\begin{pmatrix} 6 & 11 & 2 & 7 & 1 \\ 2 & -3 & 1 & 7 & -1 \\ 8 & -9 & -3 & 1 & -3 \\ 1 & 2 & 1 & 4 & 7 \\ 3 & 4 & 5 & 1 & 1 \end{pmatrix} \xrightarrow{G \cdot A} \begin{pmatrix} 6 & 11 & 2 & 7 & 1 \\ 2 & -3 & 1 & 7 & -1 \\ 8 & -9 & -3 & 1 & -3 \\ 3.16 & 4.43 & 5.06 & 2.21 & 3.16 \\ 0 & -0.63 & 0.63 & -3.48 & -6.32 \end{pmatrix}$$

Before G rotation — After G rotation

https://doi.org/10.26219/heal.aueb.5793

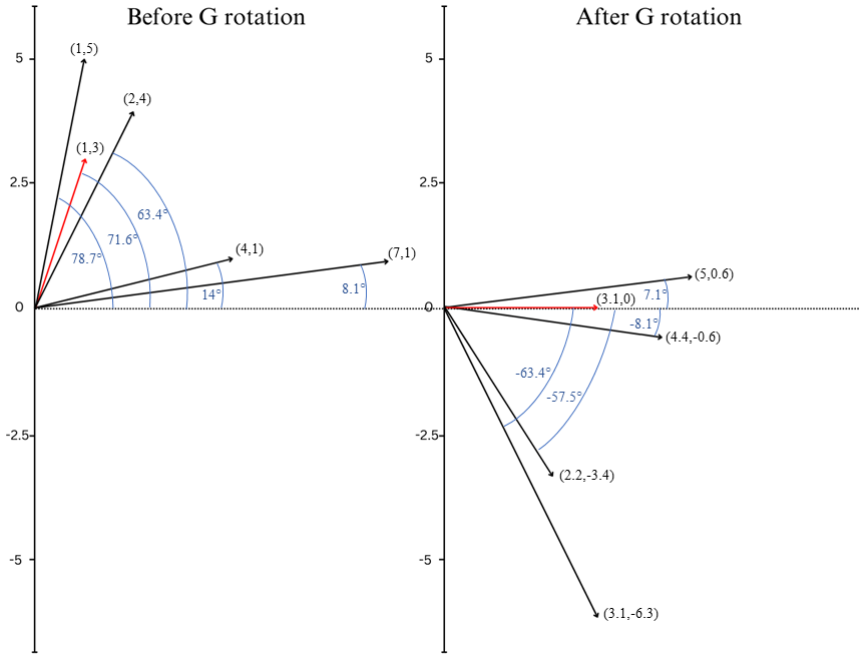


Figure 4.1: 2-d vectors before and after a Givens rotation.

The probability of two vectors being linearly dependent should approach zero as long as the matrix values are random and have many decimals. However, in the case where the matrix values are integers, there should be combinations of vectors that can be nullified together in cases where they are linearly dependent. As such, we can see in the following graph that both entries at $a_{5,1}$ and $a_{5,2}$ were eliminated in one step due to linear dependence.

$$\begin{pmatrix} 6 & 11 & 2 & 7 & 1 \\ 2 & -3 & 1 & 7 & -1 \\ 8 & -9 & -3 & 1 & -3 \\ 1 & 2 & 1 & 4 & 7 \\ 3 & 6 & 5 & 1 & 1 \end{pmatrix} \xrightarrow{G \cdot A} \begin{pmatrix} 6 & 11 & 2 & 7 & 1 \\ 2 & -3 & 1 & 7 & -1 \\ 8 & -9 & -3 & 1 & -3 \\ 3.16 & 6.32 & 5.06 & 2.21 & 3.16 \\ 0 & 0 & 0.63 & -3.48 & -6.32 \end{pmatrix}$$

Before G rotation — After G rotation

https://doi.org/10.26219/heal.aueb.5793

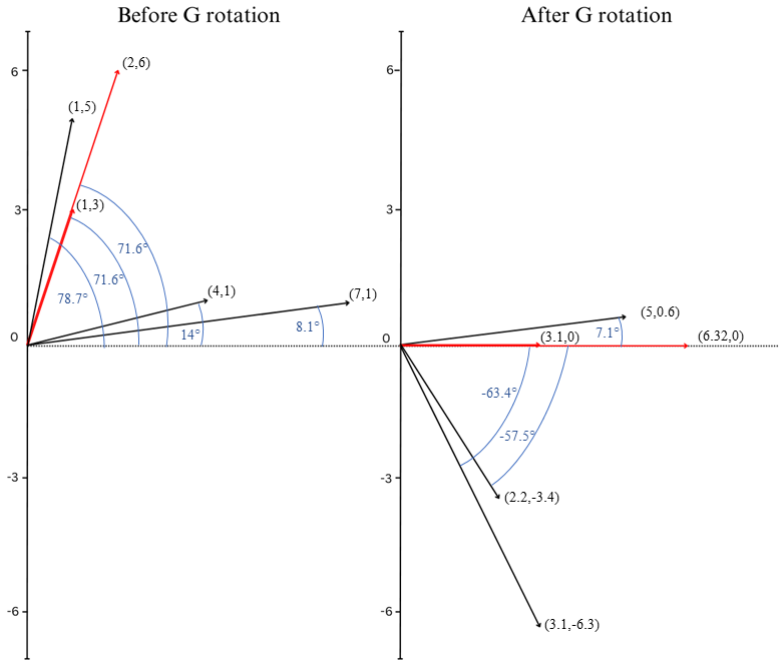


Figure 4.2: 2-d vectors before and after a Givens rotation (Linear dependent case)

4.1 Creating the environment

In order to create the environment for our RL method to work sufficiently, we need to present the rules in an appropriate manner. First of all, we need to realize the 5×5 matrix we are working with, as a board game. In this board game, the agent (the computer) has to zero out all the elements of the lower diagonal of a 5×5 matrix. In order to do so, we need to specify how he perceives his environment and which actions he can perform in order to "play" the game and reach his goal. Furthermore, in order for our agent to learn throughout the series of episodes he will play the game, we will give back to him some reward for zeroing out elements of the lower diagonal matrix or penalize him for not progressing towards his goal.

It is essential to identify which are the **states**, the **actions** and how he takes decisions and receives **reward** as the game unfolds. For the rest of this section, we will formally specify the set of states the agent can arrive at, the set of actions he can perform in each state, and the reward or penalty he will receive by participating in this scheme using the Q-learning algorithm.

https://doi.org/10.26219/heal.aueb.5793

States Space

The complexity of the State space: $\sum_{k=0}^{t} \binom{t}{k} = \mathcal{O}(2^t) = 1024, t = \frac{n(n-1)}{2} = 10,$ n = 5 (rows) for a 5×5 matrix.

Due to it being the sum of the binomial coefficients, we get a state space with complexity $\mathcal{O}(2^t)$, which gets quite computationally expensive, fairly fast, once we get above a 5×5 matrix. For this reason, we chose a small matrix to perform this experiment.

Example:

$$\begin{bmatrix} x & x & x & x & x \\ 0 & x & x & x & x \\ a_{31} & 0 & x & x & x \\ a_{41} & 0 & a_{43} & x & x \\ a_{51} & a_{52} & 0 & a_{54} & x \end{bmatrix}$$

The state space is represented by a binary vector containing all the cells that belong below the diagonal. The number of cells below the diagonal is $n \times (n-1)/2$ or $nm - n \times (n+1)/2$ if the matrix happens to be rectangular with n rows and m columns, where $n > m$. For a 5×5 matrix, there are 1,024 unique states, where these unique states are all the possible combinations of ones and zeros in the lower diagonal elements.

$$\mathrm{B} = \begin{cases} 0 & , \text{if } a_{i,j} = 0 \\ 1 & , \text{else} \end{cases}$$

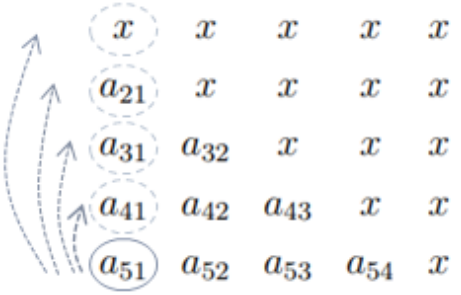
Using the above indicator function, we will produce a binary vector of: $\mathrm{B} = \Big[0, 1, 0, 1, 0, 1, 1, 1, 0, 1\Big]$. If an element of the lower diagonal is zero, we will get 0 for the element in the binary vector and 1 if the element below the diagonal takes different values. We can map onto the discrete 1,024 states by transforming this binary vector: $0 \times 2^9 + ... + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$, to decimal, where in this particular example, we get back the state 366, which is useful in traversing the Q-table where we preserve all the state-action pair values.

https://doi.org/10.26219/heal.aueb.5793

Action Space

The complexity of the Action space: $\sum_{i=1}^{n-1} i^2 = \frac{n(n-1)(2n-1)}{6} = \mathcal{O}(n^3) = 30,$ n = 5 (rows) for a 5×5 matrix.

For each element of the lower diagonal we want to zero out, we can choose an element on the same column to pair it in order to create the 2-d vector we want to rotate. Meanwhile, for the traditional Givens rotation, most of the algorithms pair the elements with the first element of the same column.



All possible pairs for a 2-d vector creation with $a_{5,1}$

With this rule set for the actions: the element a_{51} can be paired with any of the elements $\{a_{41}, a_{31}, a_{21}, a_{11}\}$ to form a 2-d vector, while the element a_{41} can only be paired with $\{a_{31}, a_{21}, a_{11}\}$ etc. Furthermore, we cannot eliminate values residing on the main diagonal and above it, because this would lead to catastrophic failure in the creation of the upper triangular matrix.

More formally, the action pairs consist of all the possible forward row exchange combinations of the Givens rotations. Again, for example, the element a_{51} will be paired with any of the elements above it, and the Givens rotation will affect all the 2-d vectors residing at the same rows.

$A_{5\times5}$:

$$\begin{bmatrix} x & x & x & x & x \\ a_{21} & x & x & x & x \\ a_{31} & a_{32} & x & x & x \\ a_{41} & a_{42} & a_{43} & x & x \\ a_{51} & a_{52} & a_{53} & a_{54} & x \end{bmatrix} \xrightarrow[G_{5,1,1}]{\begin{pmatrix} 1&0&0&0&0 \\ 0&1&0&0&0 \\ 0&0&1&0&0 \\ 0&0&0&c&s \\ 0&0&0&-s&c \end{pmatrix}} \begin{bmatrix} x & x & x & x & x \\ a_{21} & x & x & x & x \\ a_{31} & a_{32} & x & x & x \\ a^*_{41} & a^*_{42} & a^*_{43} & x^* & x^* \\ 0 & a^*_{52} & a^*_{53} & a^*_{54} & x^* \end{bmatrix}$$

https://doi.org/10.26219/heal.aueb.5793

$$\begin{bmatrix} x & x & x & x & x \\ a_{21} & x & x & x & x \\ a_{31} & a_{32} & x & x & x \\ a_{41} & a_{42} & a_{43} & x & x \\ a_{51} & a_{52} & a_{53} & a_{54} & x \end{bmatrix} \xrightarrow[G_{5,1,2}]{\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & c & 0 & s \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & -s & 0 & c \end{pmatrix}} \begin{bmatrix} x & x & x & x & x \\ a_{21} & x & x & x & x \\ a_{31}^* & a_{32}^* & x^* & x^* & x^* \\ a_{41} & a_{42} & a_{43} & x & x \\ 0 & a_{52}^* & a_{53}^* & a_{54}^* & x^* \end{bmatrix}$$

$$\begin{bmatrix} x & x & x & x & x \\ a_{21} & x & x & x & x \\ a_{31} & a_{32} & x & x & x \\ a_{41} & a_{42} & a_{43} & x & x \\ a_{51} & a_{52} & a_{53} & a_{54} & x \end{bmatrix} \xrightarrow[G_{5,1,3}]{\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & c & 0 & 0 & s \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & -s & 0 & 0 & c \end{pmatrix}} \begin{bmatrix} x & x & x & x & x \\ a_{21}^* & x^* & x^* & x^* & x^* \\ a_{31} & a_{32} & x & x & x \\ a_{41} & a_{42} & a_{43} & x & x \\ 0 & a_{52}^* & a_{53}^* & a_{54}^* & x^* \end{bmatrix}$$

$$\begin{bmatrix} x & x & x & x & x \\ a_{21} & x & x & x & x \\ a_{31} & a_{32} & x & x & x \\ a_{41} & a_{42} & a_{43} & x & x \\ a_{51} & a_{52} & a_{53} & a_{54} & x \end{bmatrix} \xrightarrow[G_{5,1,4}]{\begin{pmatrix} c & 0 & 0 & 0 & s \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ -s & 0 & 0 & 0 & c \end{pmatrix}} \begin{bmatrix} x^* & x^* & x^* & x^* & x^* \\ a_{21} & x & x & x & x \\ a_{31} & a_{32} & x & x & x \\ a_{41} & a_{42} & a_{43} & x & x \\ 0 & a_{52}^* & a_{53}^* & a_{54}^* & x^* \end{bmatrix}$$

By considering only the forward exchanges, where we zero an element on the ith row exchanging with the jth row $\forall i > j$, we end up with a total of $\sum_{n=1}^{n-1} n^p$ possible exchanges. As stated before, this geometric series forms a polynomial of degree $p+1$, for $p=2$ we get a polynomial of 3rd degree, resulting in $\mathcal{O}(n^3)$ complexity for the action space.

$$\begin{bmatrix} x & x & x & x & x \\ a_{21}^{(1)} & x & x & x & x \\ a_{31}^{(2)} & a_{32}^{(2)} & x & x & x \\ a_{41}^{(3)} & a_{42}^{(3)} & a_{43}^{(3)} & x & x \\ a_{51}^{(4)} & a_{52}^{(4)} & a_{53}^{(4)} & a_{54}^{(4)} & x \end{bmatrix}$$

We could similarly consider pairing elements with elements below them during the zeroing process, but the action space would become larger without any obvious benefit.

Because we will use the ϵ-greedy Q-learning algorithm, with a probability that will decay throughout the course of the episodes, we will either take a random action, or choose the action that uses the max value of the next state-action pair, with corresponding

https://doi.org/10.26219/heal.aueb.5793

probabilities as shown below.

$$A(s) = \begin{cases} \text{non greedy} & , prob : \frac{\epsilon}{|A(s)|} \\ \max_a Q(S', a) & , prob : 1 - \epsilon + \frac{\epsilon}{|A(s)|} \end{cases}$$

Reward Function

In order for our agent to "learn" throughout the series of episodes in which he explores the board game, we can reward him "positively" for achieving progress, and penalize him for not doing so.

Every time the agent takes an action (Givens Rotation), he is rewarded by default with a -1. If the agent succeeds in zeroing out an element of the lower diagonal in the process then he is rewarded with a -1 for the action cost and a +1 for each zeroed out element.

By using the indicator function that we defined in the State Space part of this Chapter, we will denote as $\sum_{i=0}^{k} B_{i,t}$ the sum of ones in the binary vector which corresponds to the state space we are at time step t.

Reward $= \sum_{i=0}^{k} B_{i,t-1} - \sum_{i=0}^{k} B_{i,t} - 1$, for our case of 5×5 matrix k=10.

Example:

$$\begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{bmatrix} \rightarrow \begin{bmatrix} x & x & x & x & x \\ 0 & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{bmatrix}$$

$$B_{i,t-1} = \Big[1,1,1,1,1,1,1,1,1,1,1\Big] \rightarrow B_{i,t} = \Big[0,1,1,1,1,1,1,1,1,1,1\Big]$$

Reward $= \sum_{i=0}^{k} B_{i,t-1} - \sum_{i=0}^{k} B_{i,t} - 1 = 0$

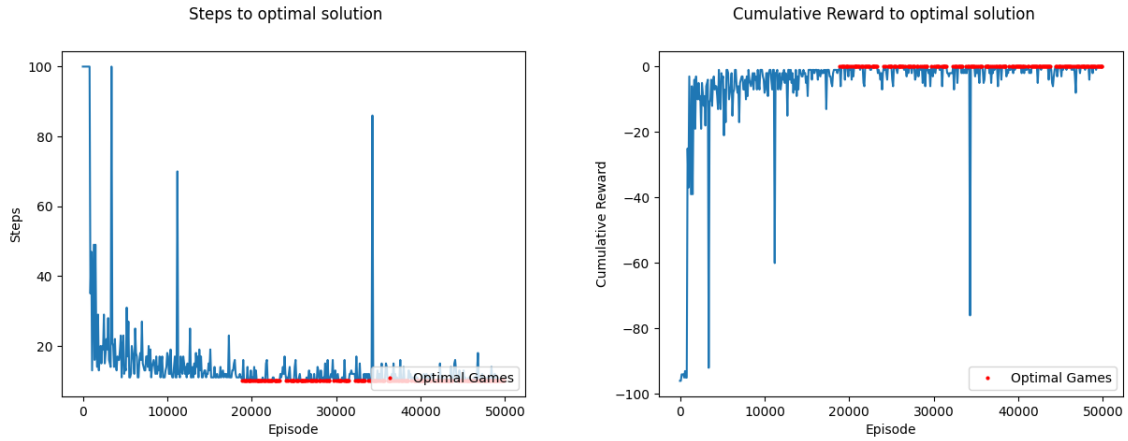
4.1.1 Dense Matrix Case: Results

For a real matrix input, the Q-learning algorithm explores the environment by taking different actions each time it observes new states. It learns through the feedback mechanism we call reward and finally converges in finding a path with as few steps as possible in order to create the upper triangular matrix.

https://doi.org/10.26219/heal.aueb.5793

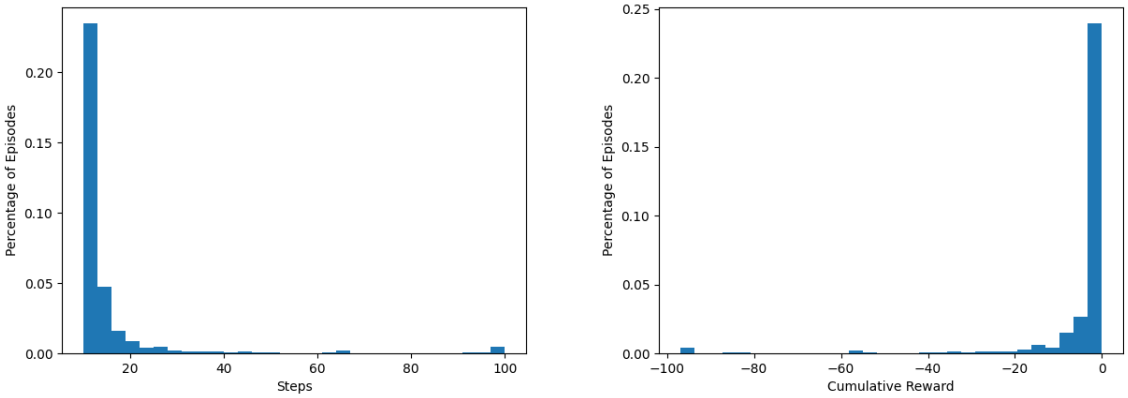
As expected during the experiment analysis, the algorithm paths that will be discovered, will not offer any gain in the number of steps in cases where the matrix inputs are real numbers with many decimals.

Throughout various times that the search was performed, the number of steps did not fall below the expected 10 steps for a 5×5 matrix. We can see that, for a specific matrix input with real numbers, the algorithm converges to the optimal solution the traditional algorithm would offer. For this specific series of episodes, the algorithm played the game for 20,000 episodes of maximum 100 actions until it found one of the possible "perfect" solutions for the Givens rotation path. During the experimentation, there were cases where the algorithm needed more than 50,000 episodes to find a path with 10 actions.



(a) Dense Matrix Steps (b) Dense Matrix Rewards

Figure 4.4: Steps and rewards for each episode.



(a) Steps histogram (b) Reward histogram

Figure 4.5: Histograms of steps and cumulative rewards for 50000 episodes.

Below we will see some of the most common observed paths that were explored by the algorithm.

https://doi.org/10.26219/heal.aueb.5793

Path 1

$$\begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{bmatrix} \xrightarrow{\textbf{Step 1}} \begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ 1 & x & x & x & x \end{bmatrix} \xrightarrow{\textbf{Step 2}} \begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ 2 & x & x & x & x \\ 1 & x & x & x & x \end{bmatrix} \xrightarrow{\textbf{Step 3}} \begin{bmatrix} x & x & x & x & x \\ 3 & x & x & x & x \\ x & x & x & x & x \\ 2 & x & x & x & x \\ 1 & x & x & x & x \end{bmatrix}$$

$$\xrightarrow{\textbf{Step 4}} \dots \xrightarrow{\textbf{Step 10}} \begin{bmatrix} x & x & x & x & x \\ 3 & x & x & x & x \\ 4 & 6 & x & x & x \\ 2 & 5 & 9 & x & x \\ 1 & 7 & 8 & 10 & x \end{bmatrix}$$

Path 2

$$\begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{bmatrix} \xrightarrow{\textbf{Step 1}} \begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ 1 & x & x & x & x \\ x & x & x & x & x \end{bmatrix} \xrightarrow{\textbf{Step 2}} \begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ 1 & x & x & x & x \\ 2 & x & x & x & x \end{bmatrix} \xrightarrow{\textbf{Step 3}} \begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ 1 & x & x & x & x \\ 2 & 3 & x & x & x \end{bmatrix}$$

$$\xrightarrow{\textbf{Step 4}} \dots \xrightarrow{\textbf{Step 10}} \begin{bmatrix} x & x & x & x & x \\ 7 & x & x & x & x \\ 4 & 8 & x & x & x \\ 1 & 5 & 9 & x & x \\ 2 & 3 & 6 & 10 & x \end{bmatrix}$$

Path 3

$$\begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{bmatrix} \xrightarrow{\textbf{Step 1}} \begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ 1 & x & x & x & x \end{bmatrix} \xrightarrow{\textbf{Step 2}} \begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ 2 & x & x & x & x \\ 1 & x & x & x & x \end{bmatrix} \xrightarrow{\textbf{Step 3}} \begin{bmatrix} x & x & x & x & x \\ 3 & x & x & x & x \\ x & x & x & x & x \\ 2 & x & x & x & x \\ 1 & x & x & x & x \end{bmatrix}$$

$$\xrightarrow{\textbf{Step 4}} \dots \xrightarrow{\textbf{Step 10}} \begin{bmatrix} x & x & x & x & x \\ 3 & x & x & x & x \\ 5 & 6 & x & x & x \\ 2 & 8 & 9 & x & x \\ 1 & 4 & 7 & 10 & x \end{bmatrix}$$

We will define the paths as the combinations of row, column, and k-shift values that the modified version of the Givens rotation algorithm would need as input in order to operate outside of RL. The values of i, j range from 1 to n and the values of k would range from 1 to $n-1$. For example $i = 2, j = 1, k = 1$ would be the creation of the 2-d vector $\begin{bmatrix} \alpha_{1,1} \\ \alpha_{2,1} \end{bmatrix}$ where we want to zero the $\alpha_{2,1}$ element. If our matrix happened to be rectangular then the ranges of the values would be different, where i would range from 1 to n (rows), while j would range from 1 to m (columns) and k still would range from 1 to $n-1$.

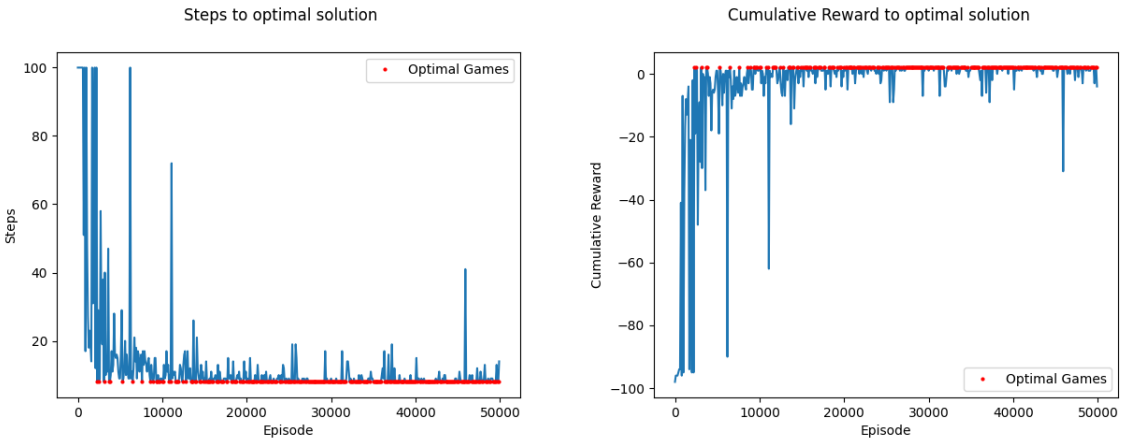
https://doi.org/10.26219/heal.aueb.5793

Path 1$_{i,j,k}$	**Path 2**$_{i,j,k}$	**Path 3**$_{i,j,k}$
5,1,2	4,1,3	5,1,4
4,1,1	5,1,4	4,1,3
2,1,1	5,2,4	2,1,1
3,1,2	3,1,1	5,2,3
4,2,2	4,2,1	3,1,2
3,2,1	5,3,1	3,2,1
5,2,3	2,1,1	5,3,2
5,3,2	3,2,1	4,2,2
4,3,1	4,3,1	4,3,1
5,4,1	5,4,1	5,4,1

4.1.2 Dense Matrix (collinear) Case: Results

For this case, our matrix has values which allow us to combine vectors that can be rotated together in one iteration, similarly to the example we had in the beginning of this Chapter. We can observe that the path provided by this RL setup, can perform the Givens rotation in 8 steps instead of the 10 steps the traditional Givens rotation path would need for a 5×5 matrix.

While in this case, the Q-learning algorithm converged into finding the optimal game of 8 steps to create the upper triangular matrix within only 5,000 episodes, in general there were cases where it needed more than 50,000 episodes to find the optimal solution.

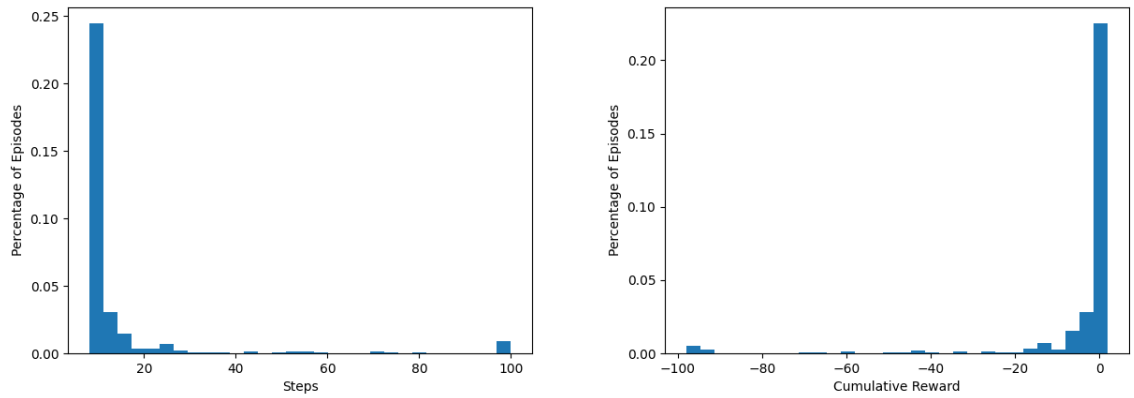


(a) Discrete Matrix Steps

(b) Discrete Matrix Rewards

Figure 4.6: Steps and rewards for each episode.

https://doi.org/10.26219/heal.aueb.5793



(a) Steps histogram (b) Reward histogram

Figure 4.7: Histograms of steps and cumulative rewards for 50000 episodes.

Below we will see some of the most common observed paths that were explored by the algorithm.

Path 1

$$\begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{bmatrix} \xrightarrow{\textbf{Step 1}} \begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ 1 & 1 & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{bmatrix} \xrightarrow{\textbf{Step 2}} \begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ 1 & 1 & x & x & x \\ x & x & x & x & x \\ 2 & 2 & x & x & x \end{bmatrix} \xrightarrow{\textbf{Step 3}} \begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ 1 & 1 & x & x & x \\ 3 & x & x & x & x \\ 2 & 2 & x & x & x \end{bmatrix}$$

$$\xrightarrow{\textbf{Step 4}} \dots \xrightarrow{\textbf{Step 8}} \begin{bmatrix} x & x & x & x & x \\ 4 & x & x & x & x \\ 1 & 1 & x & x & x \\ 3 & 6 & 7 & x & x \\ 2 & 2 & 5 & 8 & x \end{bmatrix}$$

Path 2

$$\begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{bmatrix} \xrightarrow{\textbf{Step 1}} \begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ 1 & 1 & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{bmatrix} \xrightarrow{\textbf{Step 2}} \begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ 1 & 1 & x & x & x \\ x & x & x & x & x \\ 2 & 2 & x & x & x \end{bmatrix} \xrightarrow{\textbf{Step 3}} \begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ 1 & 1 & x & x & x \\ 3 & x & x & x & x \\ 2 & 2 & x & x & x \end{bmatrix}$$

$$\xrightarrow{\textbf{Step 4}} \dots \xrightarrow{\textbf{Step 8}} \begin{bmatrix} x & x & x & x & x \\ 4 & x & x & x & x \\ 1 & 1 & x & x & x \\ 3 & 5 & 7 & x & x \\ 2 & 2 & 6 & 8 & x \end{bmatrix}$$

Path 3

$$\begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{bmatrix} \xrightarrow{\textbf{Step 1}} \begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ 1 & 1 & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{bmatrix} \xrightarrow{\textbf{Step 2}} \begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ 1 & 1 & x & x & x \\ x & x & x & x & x \\ 2 & 2 & x & x & x \end{bmatrix} \xrightarrow{\textbf{Step 3}} \begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ 1 & 1 & x & x & x \\ 3 & x & x & x & x \\ 2 & 2 & x & x & x \end{bmatrix}$$

$$\xrightarrow{\textbf{Step 4}} \dots \xrightarrow{\textbf{Step 8}} \begin{bmatrix} x & x & x & x & x \\ 4 & x & x & x & x \\ 1 & 1 & x & x & x \\ 3 & 6 & 7 & x & x \\ 2 & 2 & 5 & 8 & x \end{bmatrix}$$

Once again we will observe the paths as the combinations of row, column, and k-shift values that the modified version of the Givens rotation algorithm would need as input in order to operate outside of RL. Here are the paths for the matrix case that includes

https://doi.org/10.26219/heal.aueb.5793

collinear 2-d vectors.

Path 1$_{i,j,k}$	**Path 2**$_{i,j,k}$	**Path 3**$_{i,j,k}$
3,1,1	3,1,1	3,1,1
5,1,1	5,1,1	5,1,1
4,1,2	4,1,2	4,1,3
2,1,1	2,1,1	2,1,1
5,3,2	4,2,2	5,3,2
4,2,2	5,3,2	5,2,2
4,3,1	4,3,1	4,3,1
5,4,1	5,4,1	5,4,2

https://doi.org/10.26219/heal.aueb.5793

Chapter 5

Conclusions

In this thesis we evaluate the capabilities of Reinforcement Learning when applied to numerical algorithms specifically for the case of QR decomposition. By designing the QR decomposition as a board game we managed to let Reinforcement Learning scavenge for a solution on its own by using as a stepping stone the Givens rotation method. Furthermore, modifications of Cordic Givens rotation and Givens rotation are provided, with the modified algorithm we provided for Givens rotation being the one used in the board game in Chapter 4.

The experimentation part evaluated cases where the matrix contains real or discrete numbers. Using the Givens rotation method, only in the discrete case, we can find alternative paths to that of the sequential Givens rotation that can execute the algorithm in less steps. However, in order to identify a path with less steps, we need to "learn" on our matrix with a tool (Reinforcement Learning) that has an immensely larger computational complexity. By providing paths that contain fewer steps we would be able to benefit in an online setting, but it is seems that there is nothing to be gained in comparison to using very well established methods, such as adaptive filtering. Furthermore, the non-sequential paths we provide, in cases where the input allows us to reduce the steps of Givens rotation, would result in conflict with parallel methods that have been developed throughout the years, as such our method offering again no gain in cases where we possess multiple machines for our computations.

Throughout the time this thesis was developed, a publication with a similar mindset emerged [11], that applied the Reinforcement Learning method in a more proper and structured way, in a more appropriate setting such as the Eigenvalues. The area is considerably unexplored and there is a lot to be learned by using Reinforcement Learning and models to predict the outcome of numerical algorithms in order to bypass the computational costs of the traditional algorithms.

https://doi.org/10.26219/heal.aueb.5793

Bibliography

[1] Apolinário, Jr, J. [2009], *QRD-RLS Adaptive Filtering*, 1st edn, Springer.

[2] Fawzi, A. et al. [2022], 'Discovering faster matrix multiplication algorithms with reinforcement learning', *Nature* .

[3] Francis, J. [1961], 'The QR transformation a unitary analogue to the LR transformation', *The Computer Journal* .

[4] Geijn, R. v. d. and Myers, M. E. [2023], *Advanced Linear Algebra: Foundations to Frontiers*, University of Texas.

[5] Golub., G. H. and Loan, C. F. V. [1996], *Matrix computations*, 3rd edn, University of Texas.

[6] Higham, N. J. [2002], *Accuracy and Stability of Numerical Algorithms*, 2nd edn, SIAM.

[7] Mankowitz, D. J. et al. [2023], 'Faster sorting algorithms discovered using deep reinforcement learning', *Nature* .

[8] Mitzenmacher, M. and Vassilvitskii, S. [1961], 'Algorithms with predictions', *The Computer Journal* .

[9] Pereira, W. S., Lotfi, A. and Langou, J. [2022], 'Numerical analysis of givens rotation', *https://arxiv.org/abs/2211.04010* .

[10] Ren, M. [2013], *MSc Thesis: Cordic-Based Givens QR Decomposition For Mimo Detectors*, Georgia Institute of Technology.

[11] Romero, P. et al. [2023], 'Matrix diagonalization as a board game: Teaching an eigensolver the fastest path to solution', *https://arxiv.org/abs/2306.10075* .

[12] Roncero, D. F. [2014], *MSc Thesis: A study of QR decomposition and Kalman filter implementations*, KTH.

[13] Silver, D. et al. [2017], 'Mastering the game of go without human knowledge', *Nature* .

https://doi.org/10.26219/heal.aueb.5793

[14] Sutton, R. S. and Barto, A. G. [2018], *Reinforcement Learning: An Introduction*, 2nd edn, MIT Press.

[15] Trefethen, L. N. [1997], *Numerical Linear Algebra*, 1st edn, SIAM.

[16] Vandenberghe, L. [2023], *ECE133A Applied Numerical Computing Additional Lecture Notes*, UCLA.

[17] Volder, J. E. [1959], 'The cordic trigonometric computing technique', *IRE Transactions on Electronic Computers* .