Report Front Page

Team Details

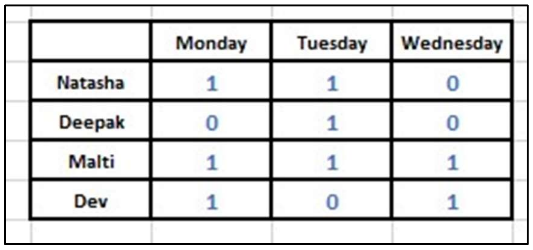
Manipulation of Matrices Using Racket

CONTENTS

S.NO	TOPIC
1	INTRODUCTION
2	MATRIX OPERATIONS
3	DESIGN ABSTRACTIONS
4	IMPLEMENTATION
5	CHEAT SHEET

INTRODUCTION

In mathematics, a matrix is a rectangular array or table of numbers, symbols or expressions, arranged in rows and columns. For example, a student attendance record can be considered as a matrix as shown below.



	Monday	Tuesday	Wednesday
Natasha	1	1	0
Deepak	0	1	0
Malti	1	1	1
Dev	1	0	1

Matrices are used for various applications in our day-to-day life. Based on the application and requirement the following operations can be performed on matrices:

1.Create a matrix of given order (m x n)

2.Add, subtract and multiply two given matrices

3.Transpose of a given matrix

4.Determinant of a given matrix

5.Diagonal elements of a given matrix

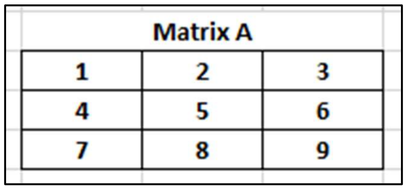
6.Lower and upper triangular matrix

7.Rank of a given matrix

8.Inverse of a given matrix.

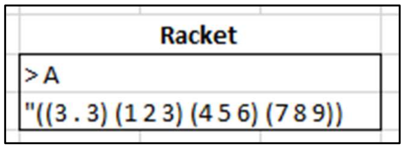
This report explains our attempt to implement those operations to make handling of matrices in racket easier for the programming community.

We have used list of lists to represent a matrix. For example:



Matrix A

1	2	3
4	5	6
7	8	9



Racket

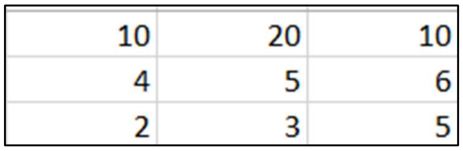
```
>A
"((3 . 3) (1 2 3) (4 5 6) (7 8 9))
```

MATRIX OPERATIONS

Addition of 2 Matrices

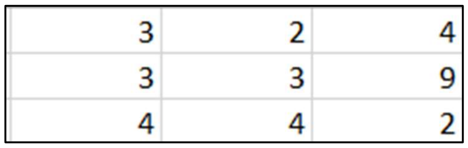
- Two matrices may be added only if they have the same dimension; that is, they must have the same number of rows and columns.
- Addition is accomplished by adding corresponding elements of the matrices.

Matrix A



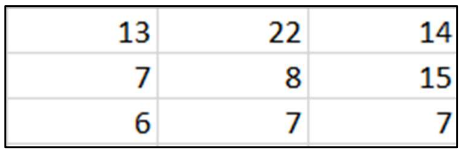
10	20	10
4	5	6
2	3	5

Matrix B



3	2	4
3	3	9
4	4	2

Sum of matrices A and B

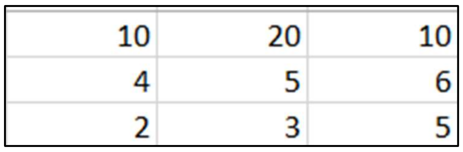


13	22	14
7	8	15
6	7	7

Difference of 2 Matrices

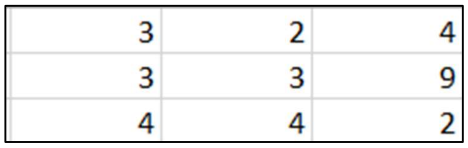
- Two matrices may be subtracted only if they have the same dimension; that is, they must have the same number of rows and columns.
- Subtracted is accomplished by subtracting corresponding elements of the matrices.

Matrix A



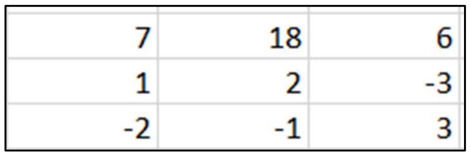
10	20	10
4	5	6
2	3	5

Matrix B



3	2	4
3	3	9
4	4	2

Difference of 2 matrices A and B

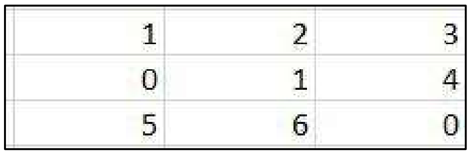


7	18	6
1	2	-3
-2	-1	3

Transpose of Matrix

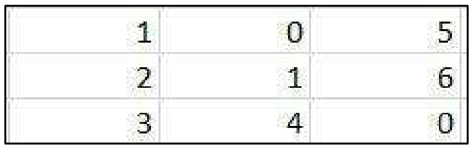
- The transpose A^T of a matrix A can be obtained by reflecting the elements along its main diagonal.
- Repeating the process on the transposed matrix returns the elements to their original position.

Matrix A



1	2	3
0	1	4
5	6	0

Transpose of matrix

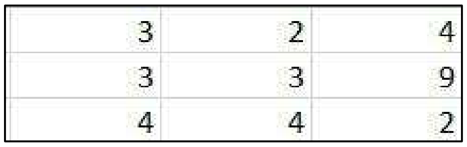


1	0	5
2	1	6
3	4	0

Multiplication of 2 Matrices

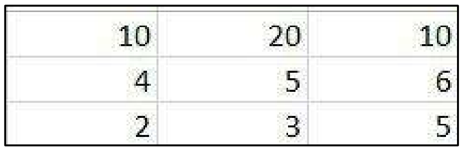
- The number of columns of the 1st matrix must equal the number of rows of the 2nd matrix.
- And the result will have the same number of rows as the 1st matrix, and the same number of columns as the 2nd matrix.

Matrix A



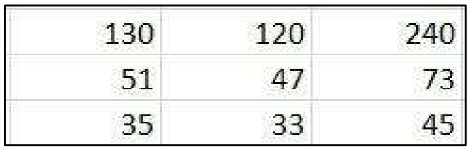
3	2	4
3	3	9
4	4	2

Matrix B



10	20	10
4	5	6
2	3	5

Multiplication of 2 matrices A and B

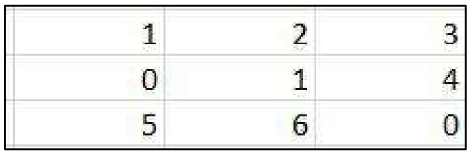


130	120	240
51	47	73
35	33	45

Inverse of Matrix

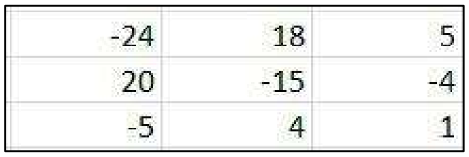
- To find the inverse of a 2x2 matrix: swap the positions of a and d, put negatives in front of b and c, and divide everything by the determinant (ad-bc).

Matrix A



1	2	3
0	1	4
5	6	0

Inverse of matrix



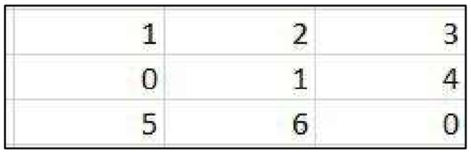
-24	18	5
20	-15	-4
-5	4	1

Determinant of Matrix

- To work out the determinant of a 3×3 matrix: Multiply a by the determinant of the 2×2 matrix that is not in a's row or column. Likewise, for b, and for c. Sum them up, but remember the minus in front of the b.

Matrix A

Determinant = 1

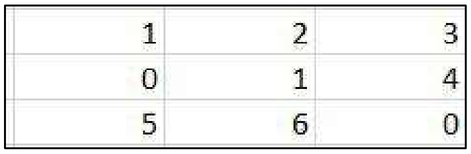


1	2	3
0	1	4
5	6	0

Upper Triangular Matrix

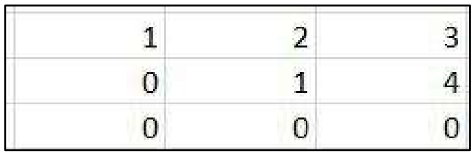
- A square matrix is called upper triangular if all the entries below the main diagonal are zero.

Matrix A



1	2	3
0	1	4
5	6	0

Upper Triangular Matrix

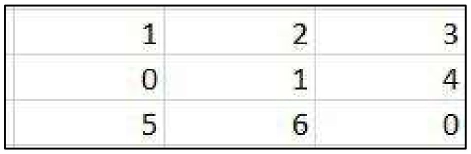


1	2	3
0	1	4
0	0	0

Lower Triangular Matrix

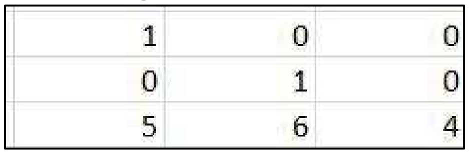
- A square matrix is called lower triangular if all the entries above the main diagonal are zero.

Matrix A



1	2	3
0	1	4
5	6	0

Lower Triangular Matrix

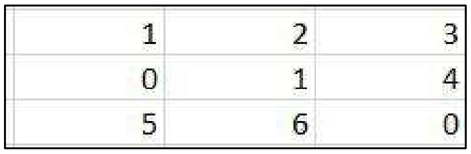


1	0	0
0	1	0
5	6	4

Diagonal Matrix

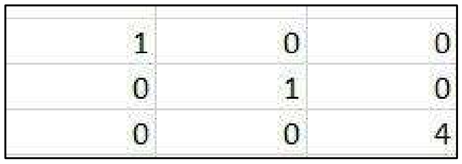
- An identity matrix of any size, or any multiple of it (a scalar matrix), is a diagonal matrix.
- A diagonal matrix is sometimes called a scaling matrix, since matrix multiplication with it results in changing scale (size). Its determinant is the product of its diagonal values.

Matrix A



1	2	3
0	1	4
5	6	0

Diagonal Matrix



1	0	0
0	1	0
0	0	4

DESIGN ABSTRACTIONS

make-shape

It defines the shape of the matrix

It is similar to shape in python

```
(make-shape  Z1 Z2) -> pair?

Z1: number? (rows)
Z2: number? (cols)
```

Examples

```
>(make-shape 5 3)
'(5 . 3)
> (make-shape 2 2)
'(2 . 2)
```

get-rows

It extracts the ‘rows’ from a ‘shape’ object

```
(get-rows S) -> number?

S: pair? (shape)
```

Examples

```
> (get-rows (make-shape 5 3))
5
> (get-rows '(2 . 2))
2
```

get-cols

It extracts the ‘cols’ from a ‘shape’ object

```
(get-cols S) -> number?

S: pair? (shape)
```

Examples

```
> (get-cols (make-shape 5 3))
3
> (get-cols '(2 . 2))
2
```

make-matrix

It defines a matrix from given shape and list of lists

```
(make-matrix S lst) -> list?

S: pair? (shape)
lst: list? (elements)
```

Examples

```
> (make-matrix '(2 . 2) '((1 2) (3 4)))
'((2 . 2) (1 2) (3 4))
> (make-matrix (make-shape 3 2) '((1 2) (3 4) (5 6)))
'((3 . 2) (1 2) (3 4) (5 6))
```

get-ele

It extracts the elements of a matrix

```
(get-ele matrix) -> list?

matrix: list? (matrix)
```

Examples

```
> (get-ele '((3 . 2) (1 2) (3 4) (5 6)))
'((1 2) (3 4) (5 6))
```

get-shape

It extracts the shape of a matrix

```
(get-shape matrix) -> pair?

matrix: list? (matrix)
```

Examples

```
> (get-shape '((2 . 2) (1 2) (3 4)))
'(2 . 2)
```

get

It returns the ith row jth column element of a matrix.

```
(get matrix i j) -> number?

matrix: list? (matrix)
i: number? (row-index)
j: number? (column-index)
```

Examples

```
> (define a '((3 . 3) (1 2 3) (4 5 6) (7 8 9)))
> (get a 2 1)
8
```

add

It returns the addition of two matrices

```
(add matrix) -> list?

matrix: list? (matrix)
```

Examples

```
> (define m1 '((2 . 2) (1 2) (3 4)))
> (define m2 '((2 . 2) (2 4) (8 9)))
> (add m1 m2)
'((2 . 2) (3 6) (11 13))
> (define m3 '((1 . 2) ((2 4))))
> (add m1 m3)
. . Shape Error
```

sub

It returns the difference of two matrices

```
(add matrix) -> list?

matrix: list? (matrix)
```

Examples

```
> (define m1 '((2 . 2) (1 2) (3 4)))
> (define m2 '((2 . 2) (2 4) (8 9)))
> (sub m1 m2)
'((2 . 2) (-1 -2) (-5 -5))
> (define m3 '((1 . 2) ((2 4))))
> (sub m1 m3)
. . Shape Error
```

diagonal

It returns the diagonal of a matrix

```
(diagonal matrix) -> list?

matrix: list? (matrix)
```

Examples

```
> (define shape (make-shape 3 3))
> (define example (make-matrix shape '((1 2 3) (4 5 6) (7 8 9))))
> (diagonal example)
'(9 5 1)
> (diagonal '((2 . 3) (1 2 3) (4 5 6)))
. . Not a Square Matrix
```

transpose

It returns the transpose of a matrix

```
(transpose matrix) -> list?

matrix: list? (matrix)
```

Examples

```
> (define shape (make-shape 3 3))
> (define example (make-matrix shape '((1 2 3) (4 5 6) (7 8 9))))
> (transpose example)
'((3 . 3) (1 4 7) (2 5 8) (3 6 9))
```

s-multi

It multiplies a scalar with the given matrix

```
(s-multi matrix val) -> list?

matrix: list?
val: number? (scalar)
```

Examples

```
> (define c '((2 . 3) (1 2 3) (4 5 6)))
> (s-multi c 10)
'((2 . 3) (10 20 30) (40 50 60))
```

multiply

Multiplication of two matrices

```
(multiply matrix1 matrix2) -> list?

matrix1: list?
matrix2: list?
```

Examples

```
> (multiply '((2 . 3) (1 2 3) (4 5 6)) '((3 . 2) (7 8) (9 10) (11
12)))
'((2 . 2) (58 64) (139 154))
> (multiply '((2 . 2) (1 2) (3 4)) '((1 . 2) (1 2)))
. . Multiplication Shape Error
```

lower-tri

It extracts the lower triangle of the matrix

```
(lower-tri matrix) -> list?

matrix: list?
```

Examples

```
> (define shapeB (make-shape 4 4))
> (define B (make-matrix shapeB '((1 2 3 4) (1 2 3 4) (5 6 7 8) (5 6
7 9))))
> (lower-tri B)
'((4 . 4) (1 0 0 0) (1 2 0 0) (5 6 7 0) (5 6 7 9))
> (lower-tri '((2 . 3) (1 2 3) (4 5 6)))
. . Square Matrix Expected
```

upper-tri

It extracts the lower triangle of the matrix

```
(upper-tri matrix) -> list?

matrix: list?
```

Examples

```
> (define shapeB (make-shape 4 4))
> (define B (make-matrix shapeB '((1 2 3 4) (1 2 3 4) (5 6 7 8) (5 6
7 9))))
> (upper-tri B)
'((4 . 4) (1 2 3 4) (0 2 3 4) (0 0 7 8) (0 0 0 9))
```

det

It finds the determinant of a matrix

```
(det matrix) -> list?

matrix: list?
```

rank

Find the rank of a matrix

```
(reduced-echelon matrix) -> list?

matrix: list?
```

IMPLEMENTATION

make-shape

```
(define (make-shape rows cols)
  (cons rows cols))
```

get-rows

```
(define (get-rows shape)
  (car shape))
```

get-cols

```
(define (get-cols shape)
  (cdr shape))
```

make-matrix

```
(define (make-matrix shape lst)
      (append (cons shape '())
              (append lst '())))
```

get-ele

```
(define (get-ele matrix)
  (cdr matrix))
```

get-shape

```
(define (get-shape matrix)
  (car matrix))
```

get

```
(define (get matrix i j)
  (list-ref (list-ref (get-ele matrix) i) j))
```

transpose

```
(define trans
  (lambda (xss)
    (cond
      [(empty? xss)          empty]
      [(empty? (first xss)) empty]
      [else                  (define first-column    (map first xss))
                             (define other-columns   (map rest  xss))
                             (cons first-column
                                   (trans other-columns))])))
(define (transpose matrix)
  (append (list (get-shape matrix)) (trans (get-ele matrix))))
```

diagonal

```
(define (d_helper m size)
  (cond
    [(empty? m) (list)]
    [(= size -1) (list)]
    [else (append (list (list-ref (list-ref m (+ size 1)) size))
(d_helper m (- size 1)))]))
(define (diagonal matrix)
  (cond
    [(= (get-rows (get-shape matrix)) (get-cols (get-shape matrix)))
(d_helper matrix (- (get-cols (get-shape matrix)) 1))]
    [else (error "Not a Square Matrix")]))
```

Helper-function: op-lists (Helps Add, Subtract)

```
(define (op-lists op a b)
  (map op a b))
```

Helper-function: helper (Iterates two matrices row by row to perform op)

```
(define (helper op A B size)
  (cond
    [(= size 0) '()]
    [else (append (list (op-lists op (car A) (car B))) (helper op
(cdr A) (cdr B) (- size 1)))]))
```

Helper-function: operation (checks for shape error w.r.t. Add and subtract)

```
(define (operation op x y)
  (cond
    [(and (= (get-rows (get-shape x)) (get-rows (get-shape y))) (=
(get-cols (get-shape x)) (get-cols (get-shape y))))
```

```
     (append (list (get-shape x)) (helper op (get-ele x) (get-ele y)
(get-rows (get-shape x))))]
    [else (error "Shape Error")]))
```

add

```
(define (add x y)
  (operation + x y))
```

sub

```
(define (sub x y)
  (operation - x y))
```

Helper-function: multiplier (Scales a list with specified value)

```
(define (multiplier lst val)
  (cond
    [(empty? lst) (list)]
    [else (append (list (* (car lst) val)) (multiplier (cdr lst)
val))]))
```

s-multi

```
(define (help A val)
  (cond
    [(empty? A) (list)]
    [else (append (list (multiplier (car A) val)) (help (cdr A)
val))]))

(define (s-multi matrix val)
  (append (list (get-shape matrix)) (help (get-ele matrix) val)))
```

multiply

```
(define mult_mat
  (λ (A B)
    (Trans_Mat (map (λ (x) (mul_Mat_vec A x))
                    (Trans_Mat B)))))
(define Trans_Mat
  (λ (A)
    (apply map (cons list A))))

(define mul_Mat_vec
  (λ (A v)
    (map (λ (x) (apply + (map * x v)))
         A)))
(define (multiply matrix1 matrix2)
  (cond
    [(= (get-cols (get-shape matrix1)) (get-rows (get-shape
matrix2))) (append (list (make-shape (get-rows (get-shape matrix1))
(get-cols (get-shape matrix2)))) (mult_mat (get-ele matrix1) (get-
ele matrix2)))]
    [else (error "Multiplication Shape Error")]))
```

lower-tri

```
(define (ele-j A i j n)
  (cond
    [(= j n) (list)]
    [(> j i) (append (list 0) (ele-j A i (+ j 1) n))]
    [else (append (list (list-ref (list-ref A i) j)) (ele-j A i (+ j
1) n))]))

(define (ele-i A i n)
  (cond
    [(= i n) (list)]
    [else (append (list (ele-j A i 0 n)) (ele-i A (+ i 1) n))]))

(define (lower-tri matrix)
  (cond
    [(= (get-rows (get-shape matrix)) (get-cols (get-shape matrix)))
(append (list (get-shape matrix)) (ele-i (get-ele matrix) 0 (get-
rows (get-shape matrix))))]
    [else (error "Square Matrix Expected")]))
```

upper-tri

```
(define (ele-jj A i j n)
  (cond
    [(= j n) (list)]
    [(< j i) (append (list 0) (ele-jj A i (+ j 1) n))]
    [else (append (list (list-ref (list-ref A i) j)) (ele-jj A i (+
j 1) n))]))

(define (ele-ii A i n)
  (cond
    [(= i n) (list)]
    [else (append (list (ele-jj A i 0 n)) (ele-ii A (+ i 1) n))]))

(define (upper-tri matrix)
  (cond
    [(= (get-rows (get-shape matrix)) (get-cols (get-shape matrix)))
(append (list (get-shape matrix)) (ele-ii (get-ele matrix) 0 (get-
rows (get-shape matrix))))]
    [else (error "Square Matrix Expected")]))
```

det

```
(define (find-index pred l)
    (let loop ((index 0) (l l))
      (cond ((null? l) #f)
            ((pred (car l)) index)
            (else (loop (add1 index) (cdr l))))))
(define (swap l i1 i2)
    (let loop ((l l) (i1 (min i1 i2)) (i2 (max i1 i2)))
      (if (= 0 i1)
        (cons (list-ref l i2)
              (replace-index (cdr l) (sub1 i2) (car l)))
        (cons (car l)
              (loop (cdr l) (sub1 i1) (sub1 i2))))))
(define (replace-index l i v)
    (if (= i 0)
      (cons v (cdr l))
      (cons (car l) (replace-index (cdr l) (add1 i) v))))
(define (det m)
    (let ((index (find-index (lambda (r) (not (= 0 (car r)))) m)))
      (if (not index)
        0
        (let ((swap-multiplier (if (= index 0) 1 -1))
              (m-swapped (if (= index 0) m (swap m 0 index))))
          (let ((submatrix (gaussian-submatrix m-swapped)))
            (if (null? submatrix)
              (caar m-swapped)
              (* swap-multiplier
                 (caar m-swapped)
                 (det submatrix))))))))
(define (gaussian-submatrix m)
    (let ((first-row (car m)))
      (map (lambda (row)
             (let ((mult (/ (car row)
                            (car first-row))))
               (map (lambda (above current)
                      (- current (* mult above)))
                    (cdr first-row)
                    (cdr row))))
           (cdr m))))
```

rank

```
(define (reduced-echelon M)
  (matrix-row-echelon M #t #t))
```

CHEAT SHEET

1. make-shape: Create a shape object of the form '(rows . cols).
2. get-rows: Extract the rows from a shape object.
3. get-cols: Extract the cols from a shape object.
4. make-matrix: create a matrix (list of lists) from given shape and elements.
5. get-ele: Extract the elements from the matrix.
6. get-shape: Extract the shape of the matrix.
7. get: It returns the i^{th} row j^{th} column element of a matrix.
8. transpose: It returns the transpose of the matrix (helper: trans).
9. diagonal: It extracts the diagonal elements of the matrix (helper: d_helper).
10. add: It returns the sum of two matrices (helper: operation, helper, op-lists).
11. sub: It returns the difference of two matrices (helper: operation, helper, op-lists).
12. s-multi: Scalar Multiplication of a matrix (helper: multiplier, help)
13. multiply: Multiplication of two matrices (helper: mult_mat, Trans_Mat, mul_Mat_vec)
14. lower-tri: Lower Triangle of a Matrix (helper: ele-i, ele-j)
15. upper-tri: Upper Triangle of a Matrix (helper: ele-ii, ele-jj)
16. det: Determinant of a matrix (helper: find-index, swap, replace-index, gaussian-submatrix)
17. rank: Rank of a matrix