

# Query languages with structural and analytic properties

Thomas Muñoz, Cristian Riveros, Domagoj Vrgoč

## 1 MATLANG syntax and semantics

We assume that we have a supply of matrix variables. The definition of an instance  $I$  on MATLANG is a function defined on a nonempty set  $var(I) = \{A, B, M, C, \dots\}$ , that assigns a concrete matrix to each element (matrix *name*) of  $var(I)$ .

Every expression  $e$  is a matrix of  $var(I)$  (*base* matrix, if you will) or a result of an operation over matrices.

The syntax of MATLANG expressions is defined by the following grammar. Every sentence is an expression itself.

$$\begin{aligned}
 e &= M && \text{(matrix variable)} \\
 \text{let } M = e_1 \text{ in } e_2 &&& \text{(local binding)} \\
 e^* &&& \text{(conjugate transpose)} \\
 \mathbf{1}(e) &&& \text{(one-vector)} \\
 \text{diag}(e) &&& \text{(diagonalization of a vector)} \\
 e_1 \cdot e_2 &&& \text{(matrix multiplication)} \\
 \text{apply}[f](e_1, \dots, e_n) &&& \text{(pointwise application of } f)
 \end{aligned}$$

The operations used in the semantics of the language are defined over complex numbers.

- **Transpose:** if  $A$  is a matrix then  $A^*$  is its conjugate transpose.
- **One-vector:** if  $A$  is a  $n \times m$  matrix then  $\mathbf{1}(A)$  is the  $m \times 1$  column vector full of ones.
- **Diag:** if  $v$  is a  $m \times 1$  column vector then  $\text{diag}(v)$  is the matrix

$$\begin{bmatrix}
 v_1 & 0 & 0 & \dots & 0 \\
 0 & v_2 & 0 & \dots & 0 \\
 \dots & \dots & \dots & \dots & \dots \\
 0 & 0 & 0 & \dots & v_m
 \end{bmatrix}$$

- **Matrix multiplication:** if  $A$  is a  $n \times m$  matrix and  $B$  is a  $m \times p$  matrix then  $A \cdot B$  is the  $n \times p$  matrix with  $(AB)_{ij} = \sum_{k=1}^n A_{ik} B_{kj}$ .
- **Pointwise application:** if  $A^{(1)}, \dots, A^{(n)}$  are  $m \times p$  matrices, then  $\text{apply}[f](A^{(1)}, \dots, A^{(n)})$  is the  $m \times p$  matrix  $C$  where  $C_{ij} = f(A_{ij}^{(1)}, \dots, A_{ij}^{(n)})$ .

The formal semantics have a set of rules for an expression  $e$  to be valid on an instance  $I$ , this is,  $e$  successfully evaluates to a matrix  $A$  on the instance  $I$ . This success is denoted as  $e(I) = A$ . Here  $I[M := A]$  denotes the instance that is equal to  $I$  except that maps  $M$  to the matrix  $A$ .

Expression	Condition for validity
$(\text{let } M = e_1 \text{ in } e_2)(I) = B$	$e_1(I) = A, e_2(I[M := A]) = B$
$e^*(I) = A^*$	$e(I) = A$
$\mathbf{1}(e)(I) = \mathbf{1}(A)$	$e(I) = A$
$\text{diag}(e)(I) = \text{diag}(A)$	$e(I) = A, A \text{ is a column vector}$
$e_1 \cdot e_2(I) = A \cdot B$	$\# \text{ columns of } A = \# \text{ rows of } B$
$\text{apply}[f](e_1, \dots, e_n)(I) = \text{apply}[f](A_1, \dots, A_n)$	$\forall k, e_k(I) = A \text{ and all } A_k \text{ have the same dimentions}$

## 2 Adding canonical vectors to MATLAB

One thing that we cannot do in MATLAB is to obtain a specific entry of a matrix. This entry is expected to be a  $1 \times 1$  matrix. We can do this by adding the standard unit vectors  $e_j$  where

$$e_i = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \rightarrow i\text{-th position}$$

We know show some examples of what can we express with this new feature. For ilustrative reasons, we asume that all the dimentions are well suited for the corresponding operation.

- Get  $A_{ij}$  with  $e_i^* \cdot A \cdot e_j$ .
- The expression  $e_i \cdot e_j^*$  is the matrix that has a 1 in the position  $i, j$  and zero everywhere else.
- Given a vector  $v$ , the expression  $v \cdot e_i^*$  is the matrix

$$\begin{bmatrix} 0 & \cdots & v_1 & \cdots & 0 \\ 0 & \cdots & v_2 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & v_n & \cdots & 0 \end{bmatrix}$$

- Replace column  $j$  of  $A$  with zeros:  $A(I - e_j \cdot e_j^*)$ .
- Replace column  $j$  of  $A$  with a vector  $v$ :  $A(I - e_j \cdot e_j^*) + v \cdot e_j^*$ .

Note that  $I = \text{diag}(\mathbf{1}(A))$  and the sum of matrices can be implemented as  $\text{apply}[+](A, B)$ .

## 3 Connection with logic

## 4 Expressions as functions

## 5 Core of Matlab and R

### MATLAB

The basic operations of MATLAB over matrices are:

- **mldivide**(A, B): returns  $x$  such that  $Ax = B$ .
- **descomposition**(A): returns a decomposition or factorization  $LU, LDL, QR$ , Cholesky, etc.
- **inv**(A): returns  $A^{-1}$ .
- **multiplication**: compute  $A \cdot B$ .
- **transpose**(A): returns  $A^T$ .
- **conjugate transpose**(A): returns  $A'$ .
- **matrix power**(A, k): returns  $A^k$ .
- **eigen**(A): returns the eigenvectors and the eigenvalues matrices of  $A$ .
- **funm**(A, f): returns matrix  $B$  with elements  $b_{ij} = f(a_{ij})$ .
- **crossprod**(a, b): vectorial product, returns  $c$  such that  $c \perp a, b$ .
- **dotprod**(a,b): returns  $a \cdot b$ .
- **diag**(v):  $v$  vector. Returns the following matrix:

$$\begin{bmatrix} v_1 & 0 & 0 & \dots & 0 \\ 0 & v_2 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & v_n \end{bmatrix}$$

- **diag**(A): given matrix  $A$ , it returns

$$\begin{bmatrix} a_{11} \\ a_{22} \\ \vdots \\ a_{nn} \end{bmatrix}$$

- **det**(A): returns the determinant of  $A$ .
- **zeros**(n, m): returns a  $n \times m$  matrix full of zeros.
- **ones**(n, m): returns a  $n \times m$  matrix full of ones.
- **A[i, j]**: you can get  $A_{ij}$ .

## R

The basic operations of the language R over matrices are:

- **A%%B**: matrix multiplication.
- **A\*B**: pointwise multiplication.
- **t(A)**: transpose.

- **diag(v)**: returns the matrix

$$\begin{bmatrix} v_1 & 0 & 0 & \dots & 0 \\ 0 & v_2 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & v_n \end{bmatrix}$$

- **diag(A)**: Returns the vector

$$\begin{bmatrix} a_{11} \\ a_{22} \\ \vdots \\ a_{nn} \end{bmatrix}$$

- **diag(k)**:  $k$  scalar. It creates the  $k \times k$  identity matrix.
- **matrix(k, n, m)**: returns the  $n \times m$  matrix, where every entry is equal to  $k$ .
- **solve(A, b)**: returns  $x$  such that  $Ax = b$ .
- **solve(A)**: returns  $A^{-1}$ .
- **det(A)**: determinant of  $A$ .
- **y<-eigen(A)**: stores the eigenvalues of  $A$  in `y$val` and the eigenvectors in `y$vec`.
- **y<-svd(A)**: it computes and stores the following:
  - `y$d`: vector of the singular values of  $A$ .
  - `y$u`: matrix of the left singular vectors of  $A$ .
  - `y$v`: matrix of the right singular vectors of  $A$ .
- **R<-chol(A)**: Cholesky factorization,  $R'R = A$ .
- **y<-qr(A)**:  $QR$  decomposition, stored in `y$qr`.
- **cbind(A,B, v, ...)**: joins matrices and vector horizontally, returns a matrix.
- **rbind(A,B, v, ...)**: joins matrices and vector vertically, returns a matrix.
- **rowMeans(A)**: returns the vector of the averages over the rows of  $A$ .
- **colMeans(A)**: returns the vector of the averages over the columns of  $A$ .
- **rowSums(A)**: returns the vector of the sums over the rows of  $A$ .
- **colSums(A)**: returns the vector of the sums over the columns of  $A$ .
- **outer(A, B, f)**: applies  $f(\cdot, \cdot)$ . Returns matrix  $C$  of entries  $c_{ij} = f(a_{ij}, b_{ij})$ .
- **A[i, j]**: you can get  $A_{ij}$ .