

# SLAP

## Simple Linear Algebra Package

Andrea Marchi

5 luglio 2023

## Indice

### 1 Data type

1

## 1 Data type

Le matrici sono salvate in array monodimensionali *row-major*, ovvero che i dati sulle righe sono sequenziali. Quindi nella indicizzazione degli elementi della matrice con  $n$  righe e  $m$  colonne ( $\mathbb{R}^{n \times m}$ ) si usa la formula:

```
1 matrix[i][j] = array[i*m + j]
```

Le righe vanno da 0 a  $n - 1$ , mentre le colonne vanno da 0 a  $m - 1$ .

Alcune librerie usano la notazione *column-major*, ovvero con indicizzazione `array[j*n+i]`. Avere una funzione che organizza i dati della matrice in colonne o in righe è comodo per quanto riguarda l'ottimizzazione della cache di lettura dei dati sequenziali dalla RAM alla CPU.

Il tipo di dati (essendo in C) è una struttura (struct) e la definizione cambia al variare del tipo di dati base (il C non permette l'uso di template). La struttura base è:

```
1 typedef struct _matd{
2     unsigned int n_rows;
3     unsigned int n_cols;
4     double *data; // row-major matrix data array
5 } matd;
```

dove la lettera finale indica il tipo di variabile usata, in questo caso `double`. I tipi di dati che ha senso utilizzare nella libreria sono:

**d** virgola mobile a doppia precisione (`double`)

**f** virgola mobile (`float`)

**i** intero (`int`)

**b** byte (`unsigned char`)

Per allocare la memoria e liberarla (sempre liberare la memoria dopo averla allocata):

```
1 matd* new_matd(unsigned int num_rows, unsigned int num_cols)
2 {
3     int i;
4     // create a new double matrix
5     if(num_rows == 0) { /*SLAP_ERROR(INVALID_ROWS);*/ return 0; } // dovrebbe ritornare
        NULL
6     if(num_cols == 0) { /*SLAP_ERROR(INVALID_COLS);*/ return 0; } // dovrebbe ritornare
        NULL
7
8     matd *m = calloc(1, sizeof(*m)); // allocate space for the struct
9     // CONTROLLARE LA MATRICE CREATA ( SLAP_CHECK(m) )
10    m->n_rows = num_rows;
```

```

11     m->n_cols = num_cols;
12     m->data = calloc(m->n_rows*m->n_cols, sizeof(*m->data));
13     // CONTROLLARE I DATI CREATI ( SLAP_CHECK(m->data) )
14     for(i=0; i<num_rows*num_cols; i++) m->data[i] = 0; // set to zero
15
16     return m;
17 }
18
19 void free_mat(matd *matrix)
20 {
21     free(matrix->data); // delete the data
22     free(matrix); // delete the data structure
23 }

```

Come setters e getters non potendo usare le operation del C++ e non riuscendo a fare qualcosa di funzionante e decente con le macro<sup>1</sup> uso le funzioni

```

1 double matd_get(matd matrix, unsigned int row, unsigned int col) {
2     return matrix.data[row*matrix.n_cols + col]; // row-major
3 }
4 void matd_set(matd matrix, unsigned int row, unsigned int col, double val) {
5     matrix.data[row*matrix.n_cols + col] = val; // row-major
6 }

```

Dovrei controllare che l'accesso sia corretto (che non cerchi di scrivere/leggere dati non allocati).

---

<sup>1</sup>Usare le macro mi permetterebbe di risparmiare tempo nella allocazione dei parametri delle funzioni. Negli algoritmi potrei usare l'accesso diretto all'array data.