SLAP
Simple Linear Algebra Package

Andrea Marchi

5 luglio 2023

Indice

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:

```
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:

```
typedef struct _matd{
        unsigned int n_rows;
        unsigned int n_cols;
        double *data; // row-major matrix data array
} 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):

```
matd* new_matd(unsigned int num_rows, unsigned int num_cols)
{
        int i;
        // create a new double matrix
        if(num_rows == 0) { /*SLAP_ERROR(INVALID_ROWS);*/ return 0; } // dovrebbe ritornare
            NULL
        if(num_cols == 0) { /*SLAP_ERROR(INVALID_COLS);*/ return 0; } // dovrebbe ritornare
            NULL

        matd *m = calloc(1, sizeof(*m)); // allocate space for the struct
        // CONTROLLARE LA MATRICE CREATA ( SLAP_CHECK(m) )
        m->n_rows = num_rows;
        m->n_cols = num_cols;
        m->data = calloc(m->n_rows*m->n_cols, sizeof(*m->data));
        // CONTROLLARE I DATI CREATI ( SLAP_CHECK(m->data) )
        for(i=0; i<num_rows*num_cols; i++) m->data[i] = 0; // set to zero

        return m;
}

void free_mat(matd *matrix)
{
        free(matrix->data); // delete the data
        free(matrix); // delete the data structure
}
```

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

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

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

2 Basic operations

Mettere un pannello (anche una tabella) che riassume le operazioni, il nome delle funzioni e come si usano.

2.1 Equality

2.2 Addition and sottration

2.3 Multiplication

2.3.1 Scalar-matrix multiplication

2.3.2 Vector-matrix multiplication

2.3.3 Matrix-matrix multiplication

[1] Usare le macro mi permetterebbe di risparmiare tempo nella allocazione dei parametri delle funzioni. Negli algoritmi potrei usare l'accesso diretto all'array `data`.

A Random numbers

Algoritmi di generazione dei numeri casuali

A.1 Sampling from Gaussian distribution

metodi di sampling descritti sul mio quaderno dei ponti integrali, con qualche indicazione in piu'