

# BLASFEO reference guide

Gianluca Frison

October 7, 2016

# Contents

|          |                             |          |
|----------|-----------------------------|----------|
| <b>1</b> | <b>Introduction</b>         | <b>2</b> |
| <b>2</b> | <b>Matrix data type</b>     | <b>3</b> |
| 2.1      | strmat definition . . . . . | 3        |
| 2.1.1    | BLASFEO . . . . .           | 3        |
| 2.1.2    | BLAS . . . . .              | 4        |
| 2.2      | strmat management . . . . . | 4        |
| 2.3      | strmat conversion . . . . . | 4        |
| 2.4      | strmat print . . . . .      | 4        |

# Chapter 1

## Introduction

BLASFEO - BLAS For Embedded Optimization.

## Chapter 2

# Matrix data type

The fundamental data type in BLASFEO is a C struct defining a matrix, called **strmat**. Depending on the chosen linear algebra library, the struct is defined differently.

### 2.1 strmat definition

#### 2.1.1 BLASFEO

```
struct d_strmat
{
  int bs;
  int m;
  int n;
  int pm;
  int cn;
  double *pA;
  double *dA;
  int use_dA;
  int memory_size;
};
```

where the struct members are

**bs** height of the panel

**m** number of rows

**n** number of columns

**pm** number of rows of the matrix as allocated in memory, used for memory alignment

**cn** number of rows of the matrix as allocated in memory, used for memory alignment

**pA** pointer to a  $pm \times pn$  array of doubles, the first element is aligned to cache line size

**dA** pointer to a  $\min(m,n)$  array of doubles, used e.g. to store the inverse of the diagonal of the matrix

**use\_dA** flag to tell if dA contains useful information

**memory\_size** size of the memory (in bytes) needed for pA and pD

### 2.1.2 BLAS

```
struct d_strmat
{
int m; // rows
int n; // cols
double *pA; // pointer to a m*n array of doubles
int memory_size; // size of needed memory
};
```

**m** number of rows

**n** number of columns

**pA** pointer to a  $m \times n$  array of doubles

**memory\_size** size of the memory (in bytes) needed for pA

## 2.2 strmat management

```
void d_allocate_strmat(int m, int n, struct d_strmat *sA);
```

```
void d_free_strmat(struct d_strmat *sA);
```

```
int d_size_strmat(int m, int n);
```

```
void d_create_strmat(int m, int n, struct d_strmat *sA, void *memory);
```

## 2.3 strmat conversion

```
void d_cvt_mat2strmat(int m, int n, double *A, int lda, struct d_strmat *sA,
int ai, int aj);
```

```
void d_cvt_tran_mat2strmat(int m, int n, double *A, int lda, struct d_strmat *sA,
int ai, int aj);
```

```
void d_cvt_strmat2mat(int m, int n, struct d_strmat *sA, int ai, int aj,
double *A, int lda);
```

```
void d_cvt_tran_strmat2mat(int m, int n, struct d_strmat *sA, int ai, int aj,
double *A, int lda);
```

## 2.4 strmat print

```
void d_print_strmat(int m, int n, struct d_strmat *sA, int ai, int aj);
```