BLASFEO reference guide

Gianluca Frison

October 7, 2017

Contents

Chapter 1

Introduction

BLASFEO - BLAS For Embedded Optimization.

BLASFEO is a library providing basic linear algebra routines performance-optimized for rather small matrices (fitting in cache). A detailed introduction to BLASFEO can be found in the ArXiv paper at the URL

`https://arxiv.org/abs/1704.02457`

Chapter 2

BLASFEO implementations

BLASFEO comes in three implementations:

- reference (RF)
- high-performance (HP)
- wrapper to BLAS and LAPACK (WR)

Chapter 3

Matrix and vector data types

The fundamental data types in BLASFEO are the C structures `d_strmat` and `d_strvec`, defining respectively a double-precision matrix and vector (and similarly `s_strmat` and `s_strvec` for single-precision matrix and vector).

Some structure members are common to all BLASFEO implementations, some others are specific to some BLASFEO implementation. Therefore, some structure members should be considered public (typically, the common members), some others should be considered as private (and typically directly used only from advanced users and targeting specific BLASFEO implementations). Below only the public members are documented.

3.1 `d_strmat` structure

The structure `d_strmat` defines the (double-precision) matrix type in BLASFEO. Structures and routines for single-precision are analogue.

3.1.1 `d_strmat` definition

```
struct d_strmat
    {
    int m;
    int n;
    double *pA;
    int memory_size;
    };
```

where the structure members are

m number of rows in the matrix

n number of columns in the matrix

pA pointer to the first element of the matrix

memory_size size (in bytes) of the memory referenced by the structure

3.1.2 `strmat` management

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

Populates the structure defining a $m \times n$ matrix, referenced by `sA` and internally dynamically allocated the memory referenced by the structure. The use of this routine is intended for prototype and off-line use, and it not advised in performance-critical code, where the routine `d_create_strmat` should be employed instead.

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

Frees the memory allocated by the routine `d_allocate_strmat`.

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

Computes the size (in bytes) of the memory referenced by the structure defining a $m \times n$ matrix. The memory has to be externally allocated to use in the `d_create_strmat` routine.

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

Populates the structure defining a $m \times n$ matrix, referenced by `sA`. The memory referenced by the structure should be allocated externally and provided to the routine using the `memory` pointer. It should typically be aligned to 64-byte boundaries (the typical cache line size). The amount of memory referenced by the structure is computed using the `d_size_strmat` routine.

3.1.3 `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);
```

3.1.4 `strmat` print

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

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

3.2 `d_strvec` structure

The structure `d_strvec` defines the (double-precision) vector type in BLASFEO. Structures and routines for single-precision are analogue.

3.2.1 `d_strvec` definition

```
struct d_strmat
    {
    int m;
    double *pa;
    int memory_size;
    };
```

where the structure members are

m number of elements in the vector

pA pointer to the first element of the vector

memory_size size (in bytes) of the memory referenced by the structure

3.2.2 strvec management

```
void d_allocate_strvec(int m, struct d_strvec *sx);
```

Populates the structure defining a $m \times 1$ vector, referenced by `sx` and internally dynamically allocated the memory referenced by the structure. The use of this routine is intended for prototype and off-line use, and it not advised in performance-critical code, where the routine `d_create_strvec` should be employed instead.

```
void d_free_strvec(struct d_strvec *sx);
```

Frees the memory allocated by the routine `d_allocate_strvec`.

```
int d_size_strvec(int m);
```

Computes the size (in bytes) of the memory referenced by the structure defining a $m \times 1$ vector. The memory has to be externally allocated to use in the `d_create_strvec` routine.

```
void d_create_strvec(int m, struct d_strvec *sx, void *memory);
```

Populates the structure defining a $m \times 1$ vector, referenced by `sx`. The memory referenced by the structure should be allocated externally and provided to the routine using the `memory` pointer. There are no alignment requirements for memory to be used by the vector structure. The amount of memory referenced by the structure is computed using the `d_size_strvec` routine.

3.2.3 strvec conversion

```
void d_cvt_vec2strvec(int m, double *x, struct d_strvec *sx, int xi);

void d_cvt_strvec2vec(int m, struct d_strvec *sx, int xi, double *x);
```

3.2.4 strvec print

```
void d_print_strvec(int m, struct d_strvec *sx, int xi);

void d_print_tran_strvec(int m, struct d_strvec *sx, int xi);
```