BLASFEO reference guide

Gianluca Frison

January 6, 2018

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 `blasfeo_dmat` and `blasfeo_dvec`, defining respectively a double-precision matrix and vector (and similarly `blasfeo_smat` and `blasfeo_sec` 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 `blasfeo_dmat` structure

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

3.1.1 `blasfeo_dmat` definition

```
struct blasfeo_dmat
    {
    int m;
    int n;
    double *pA;
    int memsize;
    };
```

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

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

3.1.2 `blasfeo_dmat` management

```
void blasfeo_allocate_dmat(int m, int n, struct blasfeo_dmat *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 `blasfeo_create_dmat` should be employed instead.

```
void blasfeo_free_dmat(struct blasfeo_dmat *sA);
```

Frees the memory allocated by the routine `blasfeo_allocate_dmat`.

```
int blasfeo_memsize_dmat(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 `blasfeo_create_dmat` routine.

```
void blasfeo_create_dmat(int m, int n, struct blasfeo_dmat *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 `blasfeo_memsize_dmat` routine.

3.1.3 `blasfeo_dmat` packing

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

void blasfeo_pack_tran_dmat(int m, int n, double *A, int lda, struct blasfeo_dmat *sA,
     int ai, int aj);

void blasfeo_unpack_dmat(int m, int n, struct blasfeo_dmat *sA, int ai, int aj,
     double *A, int lda);

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

3.1.4 `blasfeo_dmat` printing

```
void blasfeo_print_dmat(int m, int n, struct blasfeo_dmat *sA, int ai, int aj);

void blasfeo_print_tran_dmat(int m, int n, struct blasfeo_dmat *sA, int ai, int aj);
```

3.2 `blasfeo_dvec` structure

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

3.2.1 `blasfeo_dvec` definition

```
struct blasfeo_dvec
    {
    int m;
    double *pa;
    int memsize;
    };
```

where the structure members are

m number of elements in the vector

pA pointer to the first element of the vector

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

3.2.2 `blasfeo_dvec` management

```
void blasfeo_allocate_dvec(int m, struct blasfeo_dvec *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 `blasfeo_create_dvec` should be employed instead.

```
void blasfeo_free_dvec(struct blasfeo_dvec *sx);
```

Frees the memory allocated by the routine `blasfeo_allocate_dvec`.

```
int blasfeo_memsize_dvec(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 `blasfeo_create_dvec` routine.

```
void blasfeo_create_dvec(int m, struct blasfeo_dvec *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 `blasfeo_memsize_dvec` routine.

3.2.3 `blasfeo_dvec` packing

```
void blasfeo_pack_dvec(int m, double *x, struct blasfeo_dvec *sx, int xi);
```

```
void blasfeo_unpack_dvec(int m, struct blasfeo_dvec *sx, int xi, double *x);
```

3.2.4 `blasfeo_dvec` printing

```
void blasfeo_print_dvec(int m, struct blasfeo_dvec *sx, int xi);
```

```
void blasfeo_print_tran_dvec(int m, struct blasfeo_dvec *sx, int xi);
```