HPIPM guide

F.M. Smaldone

May 8, 2020

Contents

1 Dense QP

This is the structure that fits the best with gait generation MPC.
The dense QP is a QP in the form

$$
\begin{aligned}
\min_{v,s} \quad & \frac{1}{2}\begin{bmatrix} v \\ 1 \end{bmatrix}^T \begin{bmatrix} H & g \\ g^T & 0 \end{bmatrix} \begin{bmatrix} v \\ 1 \end{bmatrix} + \frac{1}{2}\begin{bmatrix} s^l \\ s^u \\ 1 \end{bmatrix}^T \begin{bmatrix} Z^l & 0 & z^l \\ 0 & Z^u & z^u \\ (z^l)^T & (z^u)^T & 0 \end{bmatrix} \begin{bmatrix} s^l \\ s^u \\ 1 \end{bmatrix} \\
\text{s.t.} \quad & Av = b, \\
& \begin{bmatrix} \underline{v} \\ \underline{d} \end{bmatrix} \leq \begin{bmatrix} J_{b,v} \\ C \end{bmatrix} v + \begin{bmatrix} J_{s,v} \\ J_{s,g} \end{bmatrix} s^l, \\
& \begin{bmatrix} J_{b,v} \\ C \end{bmatrix} v - \begin{bmatrix} J_{s,v} \\ J_{s,g} \end{bmatrix} s^u \leq \begin{bmatrix} \overline{v} \\ \overline{d} \end{bmatrix}, \\
& s^l \geq \underline{s}^l, \\
& s^u \geq \underline{s}^u,
\end{aligned}
$$

where v are the primal variables, s^l (s^u) are the slack variables of the soft lower (upper) constraints. The matrices $J_{\dots}$ are made of rows from identity matrices. Furthermore, note that the constraint matrix with respect to v is the same for the upper and the lower constraints.

2 Makefile

The Makefile for make compilation must start with these lines

```
include ../../Makefile.rule
include ../../Makefile.external_blas

LIBS = $(TOP)/lib/libhpipm.a $(BLASFEO_PATH)/lib/libblasfeo.a $(LIBS_EXTERNAL_BLAS)
```

Then if you need a datafile

```
OCP_QP_DATA = data/getting_started_data.o
```

```
OBJS_EXMP = $(OCP_QP_DATA) example_d_dense_qp.o
```

otherwise it is sufficient to write

```
OBJS_EXMP = example_d_dense_qp.o
```

Note that the file extension is '.o', while the actual script is in '.c' extension. The makefile will in fact generate an object code for each correspondent '.c' file.
Type then

```
obj: $(OBJS_EXMP)
$(CC) -o  example.out $(OBJS_EXMP) $(LIBS)
```

and eventually clean

```
clean:
rm -f *.o
rm -f data/*.o
rm -f *.out
rm -f libhpipm.a
```

3 Code guide for dense QP

3.1 Include

Include the following headers

```
#include <stdlib.h>
#include <stdio.h>
#include <sys/time.h>
#include <blasfeo_d_aux_ext_dep.h>
#include <hpipm_d_dense_qp_ipm.h>
#include <hpipm_d_dense_qp_dim.h>
#include <hpipm_d_dense_qp.h>
#include <hpipm_d_dense_qp_sol.h>
#include <hpipm_timing.h>
```

It is also useful to initialize

```
int ii, jj;
int hpipm_status;
int rep, nrep=10;
struct timeval tv0, tv1;
```

3.2 Create structure

Start creating the structure as follows

```
int dim_size = d_dense_qp_dim_memsize(int N);
void *dim_mem = malloc(dim_size);
struct d_dense_qp_dim dim;
d_dense_qp_dim_create(&dim, dim_mem);
```

3.3 Populate structure

Once created, a dense QP structure can be populated using the global conversion routine

```
int qp_size = d_dense_qp_memsize(&dim);
void *qp_mem = malloc(qp_size);
struct d_dense_qp qp;
d_dense_qp_create(&dim, &qp, qp_mem);
d_dense_qp_set_all(double *H, double *g, double *A, double *b,
 int *idxb, double *d_lb, double *d_ub, double *C, double *d_lg, double *d_ug, double *Zl,
  double *Zu, double *zl, double *zu, int *idxs, double *d_ls,
   double *d_us, struct d_dense_qp *qp);
```

which is useful when all the structure fields have to be populated at once. Use the NULL pointer if a field must be left empty.
Alternatively individual structure fields can be filled with the setter routine

```
void d_ocp_qp_set(char *field, int *stage, void *value, struct d_ocp_qp *qp);
```

where the fields are the ones like H, g, A, b, idxb, ... etc.

3.4 Create solution structure

It is necessary to create a solution structure

```
int qp_sol_size = d_dense_qp_sol_memsize(&dim);
void *qp_sol_mem = malloc(qp_sol_size);

struct d_dense_qp_sol qp_sol;
d_dense_qp_sol_create(&dim, &qp_sol, qp_sol_mem);
```

3.5 IPM solver

Create the structure for the IPM (interior point method) solver

```
int ipm_arg_size = d_dense_qp_ipm_arg_memsize(&dim);
void *ipm_arg_mem = malloc(ipm_arg_size);

struct d_dense_qp_ipm_arg arg;
d_dense_qp_ipm_arg_create(&dim, &arg, ipm_arg_mem);
d_dense_qp_ipm_arg_set_default(mode, &arg);
```

Moreover, specific settings can be modified as

```
d_dense_qp_ipm_arg_set_mu0(&mu0, &arg);
d_dense_qp_ipm_arg_set_iter_max(&iter_max, &arg);
d_dense_qp_ipm_arg_set_alpha_min(&alpha_min, &arg);
d_dense_qp_ipm_arg_set_mu0(&mu0, &arg);
d_dense_qp_ipm_arg_set_tol_stat(&tol_stat, &arg);
d_dense_qp_ipm_arg_set_tol_eq(&tol_eq, &arg);
d_dense_qp_ipm_arg_set_tol_ineq(&tol_ineq, &arg);
d_dense_qp_ipm_arg_set_tol_comp(&tol_comp, &arg);
d_dense_qp_ipm_arg_set_reg_prim(&reg_prim, &arg);
d_dense_qp_ipm_arg_set_warm_start(&warm_start, &arg);
d_dense_qp_ipm_arg_set_pred_corr(&pred_corr, &arg);
d_dense_qp_ipm_arg_set_ric_alg(&ric_alg, &arg);
```

The IPM workspace struct must also be initialized

```
int ipm_size = d_dense_qp_ipm_ws_memsize(&dim, &arg);
void *ipm_mem = malloc(ipm_size);

struct d_dense_qp_ipm_ws workspace;
d_dense_qp_ipm_ws_create(&dim, &arg, &workspace, ipm_mem);
```

3.6 Solve QP with IPM routine

Solve in this way

```
hpipm_timer timer;
hpipm_tic(&timer);

for(rep=0; rep<nrep; rep++)
{
 &qp_sol);
 &qp_sol);

// call solver
d_dense_qp_ipm_solve(&qp, &qp_sol, &arg, &workspace);
d_dense_qp_ipm_get_status(&workspace, &hpipm_status);
}

double time_ipm = hpipm_toc(&timer) / nrep;
```

Solution informations are printed as follows

```
    printf("\nHPIPM returned with flag %i.\n", hpipm_status);
if(hpipm_status == 0)
{
printf("\n -> QP solved!\n");
}
else if(hpipm_status==1)
{
printf("\n -> Solver failed! Maximum number of iterations reached\n");
}
else if(hpipm_status==2)
{
printf("\n -> Solver failed! Minimum step lenght reached\n");
}
else if(hpipm_status==2)
{
printf("\n -> Solver failed! NaN in computations\n");
}
else
{
printf("\n -> Solver failed! Unknown return flag\n");
}
```

3.7 Get solution

The QP solution is retrieved as

```
int nu_max = ? (maybe N)
double *u = malloc(nu_max*sizeof(double));

printf("\nu = \n");
```

```
for(ii=0; ii<=N; ii++)
{
d_ocp_qp_sol_get_u(ii, &qp_sol, u);
}
```

3.8 Free memory

```
free(dim_mem);
free(qp_mem);
free(qp_sol_mem);
free(ipm_arg_mem);
free(ipm_mem);

free(u);
```