`acados` reference

September 17, 2016

Chapter 1

Interfaces

1.1 OCP QP interface

This interface describes a Quadratic Programming (QP) problem with Optimal Control Problem (OCP) structure, as follows:

$$\begin{aligned}
\underset{\substack{x_0,\ldots,x_k \\ u_0,\ldots,u_{N-1}}}{\text{minimize}} \quad & \sum_{k=0}^{N-1} \frac{1}{2} \begin{bmatrix} x_k \\ u_k \end{bmatrix}^\top \begin{bmatrix} Q_k & S_k \\ S_k^\top & R_k \end{bmatrix} \begin{bmatrix} x_k \\ u_k \end{bmatrix} + \begin{bmatrix} q_k \\ r_k \end{bmatrix}^\top \begin{bmatrix} x_k \\ u_k \end{bmatrix} && \text{(1.1a)} \\
& \quad + \frac{1}{2} x_N^\top Q_N x_N + q_N^\top x_N \\
\text{subject to} \quad & x_{k+1} = A_k x_k + B_k u_k + b_k, && k = 0, \ldots, N-1, && \text{(1.1b)} \\
& \underline{u}_k \leq u_k \leq \overline{u}_k, && k = 0, \ldots, N-1, && \text{(1.1c)} \\
& \underline{x}_k \leq x_k \leq \overline{x}_k, && k = 0, \ldots, N, && \text{(1.1d)} \\
& \underline{c}_k \leq C_k^x x_k + C_k^u u_k \leq \overline{c}_k, && k = 0, \ldots, N-1, && \text{(1.1e)} \\
& \underline{c}_N \leq C_N^x x_k \leq \overline{c}_N. && && \text{(1.1f)}
\end{aligned}$$

The `C` interface looks like

```
int ocp_qp_SOLVERNAME(
    int N, int *nx, int *nu, int *nb, int *nc,
    double **A, double **B, double **b,
    double **Q, double **S, double **R, double **q, double **r,
    int **idxb, double **lb, double **ub,
    double **Cx, double **Cu, double **lc, double **uc,
    double **x, double **u, double **pi, double **lam,
    struct ocp_qp_SOLVERNAME_args *args, void *work);
```

where `SOLVERNAME` is the name of the specific solver.

`N` [input] is the horizon length.

`nx` [input] is the vector of the state sizes n_u at the different stages, such that `nx[k]` is the state size at stage `k`.

`nu` [input] is the vector of the input sizes n_x at the different stages, such that `nu[k]` is the input size at stage `k`.

`nb` [input] is the vector of the bound sizes n_b at the different stages, such that `nb[k]` is the bound size at stage `k`. The value of `nb[k]` is smaller or equal to `nx[k]+nu[k]`.

`nc` [input] is the vector of the general polytopic constraint sizes n_c at the different stages, such that `nc[k]` is the general polytopic constraint size at stage `k`.

A [input] is the vector of size N of the pointers to the first element of the matrices A_k, such that `A[k]` is the pointer to the first element of the matrix A_k, and `A[k][0]` is the first element of the matrix A_k. The matrix referenced by the pointer `A[k]` is stored in column-major (or Fortran-like) order, in a vector of `nx[k+1]`$\times$`nx[k]` double-precision floating-point numbers.

B [input] is the vector of size N of the pointers to the first element of the matrices B_k, such that `B[k]` is the pointer to the first element of the matrix B_k, and `B[k][0]` is the first element of the matrix B_k. The matrix referenced by the pointer `B[k]` is stored in column-major (or Fortran-like) order, in a vector of `nx[k+1]`$\times$`nu[k]` double-precision floating-point numbers.

b [input] is the vector of size N of the pointers to the first element of the vectors b_k, such that `b[k]` is the pointer to the first element of the vector b_k, and `b[k][0]` is the first element of the vector b_k. The vector referenced by the pointer `b[k]` is stored in a vector of `nx[k+1]`$\times 1$ double-precision floating-point numbers.

Q [input] is the vector of size $N+1$ of the pointers to the first element of the matrices Q_k, such that `Q[k]` is the pointer to the first element of the matrix Q_k, and `Q[k][0]` is the first element of the matrix Q_k. The matrix referenced by the pointer `Q[k]` is stored in column-major (or Fortran-like) order, in a vector of `nx[k]`$\times$`nx[k]` double-precision floating-point numbers.

S [input] is the vector of size N of the pointers to the first element of the matrices S_k, such that `S[k]` is the pointer to the first element of the matrix S_k, and `S[k][0]` is the first element of the matrix S_k. The matrix referenced by the pointer `S[k]` is stored in column-major (or Fortran-like) order, in a vector of `nu[k]`$\times$`nx[k]` double-precision floating-point numbers.

R [input] is the vector of size N of the pointers to the first element of the matrices R_k, such that `R[k]` is the pointer to the first element of the matrix R_k, and `R[k][0]` is the first element of the matrix R_k. The matrix

referenced by the pointer `R[k]` is stored in column-major (or Fortran-like) order, in a vector of `nu[k]`×`nu[k]` double-precision floating-point numbers.

q [input] is the vector of size $N+1$ of the pointers to the first element of the vectors q_k, such that `q[k]` is the pointer to the first element of the vector q_k, and `q[k][0]` is the first element of the vector q_k. The vector referenced by the pointer `q[k]` is stored in a vector of `nx[k]`×`1` double-precision floating-point numbers.

r [input] is the vector of size N of the pointers to the first element of the vectors r_k, such that `r[k]` is the pointer to the first element of the vector r_k, and `r[k][0]` is the first element of the vector r_k. The vector referenced by the pointer `r[k]` is stored in a vector of `nu[k]`×`1` double-precision floating-point numbers.

idxb [input] is the vector of size $N+1$ of the pointers to the first element of the integer vectors idxb_k describing the indexes of the corresponding upper and lower bounds in `lb` and `ub`, such that `idxb[k]` is the pointer to the index of the first bound at stage k, and `idxb[k][0]` is index of the first bound at stage k. The indexes in `idxb[k]` correspond to the position of the constrained components in the variables vector $\begin{bmatrix} x_k \\ u_k \end{bmatrix}$: therefore a bound on the first state component has index `0`, a bound on the last state component has index `nx[k]-1`, a bound on the first input component has index `nx[k]` and a bound on the last input component has index `nx[k]+nu[k]-1`. The vector referenced by the pointer `idxb[k]` is stored in a vector of `nb[k]`×`1` integer numbers.

lb [input] is the vector of size $N+1$ of the pointers to the first element of the vectors $\begin{bmatrix} \underline{x}_k \\ \underline{u}_k \end{bmatrix}$, such that `lb[k]` is the pointer to the first element of the vector $\begin{bmatrix} \underline{x}_k \\ \underline{u}_k \end{bmatrix}$, and `lb[k][0]` is the first element of the vector $\begin{bmatrix} \underline{x}_k \\ \underline{u}_k \end{bmatrix}$. The vector referenced by the pointer `lb[k]` is stored in a vector of `nb[k]`×`1` double-precision floating-point numbers.

ub [input] is the vector of size $N+1$ of the pointers to the first element of the vectors $\begin{bmatrix} \overline{x}_k \\ \overline{u}_k \end{bmatrix}$, such that `ub[k]` is the pointer to the first element of the vector $\begin{bmatrix} \overline{x}_k \\ \overline{u}_k \end{bmatrix}$, and `ub[k][0]` is the first element of the vector $\begin{bmatrix} \overline{x}_k \\ \overline{u}_k \end{bmatrix}$. The vector referenced by the pointer `ub[k]` is stored in a vector of `nb[k]`×`1` double-precision floating-point numbers.

Cx [input] is the vector of size $N+1$ of the pointers to the first element of the matrices C_k^x, such that `Cx[k]` is the pointer to the first element of the matrix C_k^x, and `Cx[k][0]` is the first element of the matrix C_k^x. The

matrix referenced by the pointer `Dx[k]` is stored in column-major (or Fortran-like) order, in a vector of `nc[k]`×`nx[k]` double-precision floating-point numbers.

Cu [input] is the vector of size N of the pointers to the first element of the matrices C_k^u, such that `Cu[k]` is the pointer to the first element of the matrix C_k^u, and `Cu[k][0]` is the first element of the matrix C_k^u. The matrix referenced by the pointer `Cu[k]` is stored in column-major (or Fortran-like) order, in a vector of `nc[k]`×`nu[k]` double-precision floating-point numbers.

lc [input] is the vector of size $N+1$ of the pointers to the first element of the vectors $\underline{c}_k$, such that `lc[k]` is the pointer to the first element of the vector $\underline{c}_k$, and `lc[k][0]` is the first element of the vector $\underline{c}_k$. The vector referenced by the pointer `ld[k]` is stored in a vector of `nc[k]`×`1` double-precision floating-point numbers.

uc [input] is the vector of size $N+1$ of the pointers to the first element of the vectors $\overline{c}_k$, such that `uc[k]` is the pointer to the first element of the vector $\overline{c}_k$, and `uc[k][0]` is the first element of the vector $\overline{c}_k$. The vector referenced by the pointer `uc[k]` is stored in a vector of `nc[k]`×`1` double-precision floating-point numbers.

x [output] is the vector of size $N+1$ of the pointers to the first element of the vectors x_k (states), such that `x[k]` is the pointer to the first element of the vector x_k, and `x[k][0]` is the first element of the vector x_k. The vector referenced by the pointer `x[k]` is stored in a vector of `nx[k]`×`1` double-precision floating-point numbers.

u [output] is the vector of size $N+1$ of the pointers to the first element of the vectors u_k (controls or inputs), such that `u[k]` is the pointer to the first element of the vector u_k, and `u[k][0]` is the first element of the vector u_k. The vector referenced by the pointer `u[k]` is stored in a vector of `nu[k]`×`1` double-precision floating-point numbers.

pi [output] is the vector of size N of the pointers to the first element of the vectors π_k (Lagrange multipliers of the (state space system) equality constraints), such that `pi[k]` is the pointer to the first element of the vector π_k, and `pi[k][0]` is the first element of the vector π_k. The vector referenced by the pointer `pi[k]` is stored in a vector of `nx[k+1]`×`1` double-precision floating-point numbers.

lam [output] is the vector of size $N+1$ of the pointers to the first element of the vectors λ_k (Lagrange multipliers of the inequality constraints), such that `lam[k]` is the pointer to the first element of the vector λ_k, and `lam[k][0]` is the first element of the vector λ_k. The vector referenced by the pointer `lam[k]` is stored in a vector of `(2*nb[k]+2*ng[k])`×`1` double-precision floating-point numbers, corresponding to (in the same order) `lb`, `ub`, `lc`, `uc`..

args [input] is the pointer to a structure of type `ocp_qp_SOLVERNAME_args` that defines the arguments (as e.g. maximum number of iterations, minimum step size, ...) passed to the specific solver.

work [workspace] is the pointer to the working space used by the specific solver. The working space size (in bytes) is returned by a call to the function `ocp_qp_SOLVERNAME_workspace_size_bytes(int N, int *nx, int *nu, int *nb, int *nc, int **idxb, struct ocp_qp_solver_args *args)`.

Furthermore, the function returns an int, that is defined in the following enum (TODO change the names to something better!!!):

ACADOS_SUCCESS Solution successfully found.

ACADOS_MAXITER Maximum number of iterations reached.

ACADOS_MINSTEP Minumum step size reached (in IPs, probably unfeasible problem).

1.1.1 Examples

MPC problem

In the MPC problem, the initial state is fixed. This can be modeled by choosing `nx[0]=0`, i.e. not considering the initial state as an optimization variable. As a consequence, e.g. the matrix `A[0]` has size `nx[1]`$\times$`0`, the matrix `Q[0]` has size `0`$\times$`0`, and the vector `q[0]` has size `0`$\times$`1`. The information about the known value of x_0 and the matrix A_0 are used to compute the value of the vector `b[0]`, that is initialized to $b_0 + A_0 \cdot x_0$.

1.2 Condensing Interface

This is the interface to the condensing routine, first discovered in [**?**]. The notation used here is partly based on [**?**].

```
int condensing_NAME(condensing_in *input, condensing_out *output,
        condensing_memory *mem, condensing_workspace *work);
```

The condensing function relies on different structures, explained next.

input The input struct has a number of fields

1.3 Integrator Interface with Sensitivity Propagation

The C code interface looks like

```
int ocp_qp_NAME(integrator_in *input,
      struct integrator_NAME_opts *opts, double *work);
```

where the input struct is defined as:

```
typedef struct integrator_in_{
    int nSteps, int nSystems, int nX, int nXA, int nP, int nOD,
    double *steps, int *nx, int *nxa, double **S0, double *mu,
    double **x, double **xa, double *p, double *od, double **xOut,
    double **sensOut, double *muOut, double *hessOut, int nOutputs,
    int *nGridOutputs, double **gridOutputs, double **muOutputs,
    double **valOutputs, double **sensOutputs, function_call *sys,
    function_call *out, double *variables} integrator_in;
```

where `NAME` is the name of the specific integration routine. Additionally, a function struct has been defined as:

```
typedef struct function_call_{
    int dimIn, int dimOut, bool *sparsity, bool linear,
    void (*fun)(double*,double*), void (*jac)(double*,double*),
    bool forward, bool backward, bool hessian,
    void (*vde_forw)(double*,double*), void (*vde_adj)(double*,double*),
    void (*vde_hess)(double*,double*)} function_call;
```

If we like the latter embedding of structs, we can avoid some of the fields in the *integrator_in* struct by moving them somewhere else!

For fixed step integrators:

`nSteps` [input] is the number of integration steps (in case of no step size control).

`steps` [input] is the vector of size *nSteps*, containing the step size for each integration step.

Dimensions of the dynamic system(s):

`nSystems` [input] is the number of dynamic subsystems in order of dependency (states of subsystem k only depend on subsystems $1, 2, \ldots, k-1$ etc).

`flag_linear` [input] is the vector of size *nSystems*, containing the flag whether the subsystem is linear or not (0 or 1, should be part of options OR see the function_call struct).

`nx` [input] is the vector of size *nSystems*, containing the number of differential states for this integrator.

`nxa` [input] is the vector of size *nSystems*, containing the number of algebraic states for this integrator.

`nX` [input] is the **total** number of differential states for this integrator.

`nXA` [input] is the **total** number of algebraic states for this integrator.

`nP` [input] is the number of free parameters for this integrator (sensitivities with respect to these parameters are computed).

`nOD` [input] is the number of online data, i.e. fixed parameters for this integrator (no sensitivities are computed).

Sensitivity analysis for the integrator:

`S0` [input] is the vector of size $nSystems$ of the pointers to the matrices containing the forward seeds (the dimensions of these matrices are $nx[0] \times (nx[0] + np)$, $nx[1] \times (nx[0] + nx[1] + np)$, ... etc).

`mu` [input] is the vector of dimension nX, containing the backward seed.

`flag_forward` [input] is the flag for first order forward sensitivity propagation (0 or 1, should be part of options OR see the function_call struct).

`flag_backward` [input] is the flag for first order backward sensitivity propagation (0 or 1, should be part of options OR see the function_call struct).

`flag_hessian` [input] is the flag for second order Hessian propagation (0 or 1, should be part of options OR see the function_call struct).

Input and output data to the integrator:

`x` [input] is the vector of size $nSystems$ of the pointers to the initial values of the differential states for this integrator.

`xOut` [output] is the vector of size $nSystems$ of the pointers to the end values of the differential states as a result of calling this integrator.

`sensOut` [output] is the vector of size $nSystems$ of the pointers to the end values of the forward sensitivities as a result of calling this integrator.

`muOut` [output] is the vector of size $nX+nP$ containing the backward derivatives as a result of calling this integrator.

`hessOut` [output] is the matrix of size $(nX + nP) \times (nX + nP)p$ containing the second order derivatives as a result of calling this integrator.

`xa` [input,output] is the vector of size $nSystems$ of the pointers to the initial **guess** of the algebraic states for this integrator (the guess can be updated by the integrator).

`p` [input] is the vector of size nP containing the free parameters for this integrator (sensitivities can be computed).

`od` [input] is the vector of size nOD containing the online data, i.e. fixed parameters for this integrator (no sensitivities are computed).

Extra outputs to be evaluated:

`nOutputs` [input] is the number of extra output functions (with each their own dimension).

`dimOutputs` [input] is the vector of size *nOutputs*, containing the dimension for each of the output functions (moved to function_call struct!).

`sparsityOutputs` [input] is the matrix of dimension $nOutputs \times (nX + nXA + nX)$, defining the sparsity pattern for each output function with respect to the differential states, the algebraic variables and the state derivatives (moved to function_call struct!).

`nGridOutputs` [input] is the vector of size *nOutputs*, containing the number of grid points on which each output function should be evaluated.

`gridOutputs` [input] is the vector of size *nOutputs* of the pointers to the grid points on which each output function should be evaluated.

`muOutputs` [input] is the vector of size *nOutputs* of the pointers to the backward seeds for each of the grid points defined by *gridOutputs*.

`valOutputs` [output] is the vector of size *nOutputs* of the pointers to the outputs evaluated on the grid points defined by *gridOutputs*.

`sensOutputs` [output] is the vector of size *nOutputs* of the pointers to the forward sensitivities of the outputs evaluated on the grid points defined by *gridOutputs*.

The functions (including AD) to be evaluated:

(This changed using the function_call struct!!)

`rhs` [input] the function pointer to evaluate the right-hand side.

`jac` [input] the function pointer to evaluate the full Jacobian.

`vde_forw` [input] the function pointer to evaluate the forward variational differential equations.

`vde_adj` [input] the function pointer to evaluate the adjoint equations.

`vde_hess` [input] the function pointer to evaluate the second order sensitivity equations.

`out_vde_forw` [input] is the vector of size *nOutputs*, containing the function pointers to evaluate the forward sensitivity equations for the outputs.

`out_vde_adj` [input] is the vector of size *nOutputs*, containing the function pointers to evaluate the adjoint equations for the output functions.

`out_vde_hess` [input] is the vector of size *nOutputs*, containing the function pointers to evaluate the second order sensitivity equations for the outputs.

Other memory:

variables [input,output] is the pointer to the warm variables space used by the specific solver from one call to the other. The variables space size (in doubles) is returned by a call to the function `integrator_NAME_variables_double(integrator_in *input, struct integrator_solver_opts *opts)`.

opts [input] is the pointer to a structure of type `integrator_NAME_opts` that defines the arguments (as e.g. maximum number of steps, minimum step size, desired tolerance, ...) passed to the specific solver.

work [workspace] is the pointer to the working space used by the specific solver. The working space size (in doubles) is returned by a call to the function `integrator_NAME_workspace_double(integrator_in *input, struct integrator_solver_opts *opts)`.

Furthermore, the function returns an int, that is defined in the following enum (TODO change the names to something better!!!):

ACADOS_SUCCESS Solution successfully found.

ACADOS_MAXITER Maximum number of integration steps reached.

ACADOS_MINSTEP Minumum step size reached.