DynamicNLPModels

David Cole, Sungho Shin, Francois Pacaud

August 26, 2022

Contents

Part I

Introduction

Chapter 1

Introduction

DynamicNLPModels.jl is a package for Julia designed for representing linear model predictive control (MPC) problems. It includes an API for building a model from user defined data and querying solutions.

> **Note**
>
> This documentation is also available in PDF format.

1.1 Installation

To install this package, please use

```
using Pkg
Pkg.add(url="https://github.com/MadNLP/DynamicNLPModels.jl.git")
```

or

```
pkg> add https://github.com/MadNLP/DynamicNLPModels.jl.git
```

1.2 Overview

DynamicNLPModels.jl can construct both sparse and condensed formulations for MPC problems based on user defined data. We use the methods discussed by Jerez et al. to eliminate the states and condense the problem. DynamicNLPModels.jl constructs models that are subtypes of `AbstractNLPModel` from NLPModels.jl enabling both the sparse and condensed models to be solved with a variety of different solver packages in Julia. DynamicNLPModels was designed in part with the goal of solving linear MPC problems on the GPU. This can be done within MadNLP.jl using MadNLPGPU.jl.

The general sparse formulation used within DynamicNLPModels.jl is

$$
\begin{aligned}
\min_{s,u,v}\ & s_N^\top Q_f s_N + \frac{1}{2}\sum_{i=0}^{N-1} \begin{bmatrix} s_i \\ u_i \end{bmatrix}^\top \begin{bmatrix} Q & S \\ S^\top & R \end{bmatrix} \begin{bmatrix} s_i \\ u_i \end{bmatrix} \\
\text{s.t. } & s_{i+1} = As_i + Bu_i + w_i \quad \forall i = 0, 1, \cdots, N-1 \\
& u_i = Ks_i + v_i \quad \forall i = 0, 1, \cdots, N-1 \\
& g^l \leq Es_i + Fu_i \leq g^u \quad \forall i = 0, 1, \cdots, N-1 \\
& s^l \leq s_i \leq s^u \quad \forall i = 0, 1, \cdots, N \\
& u^l \leq u_t \leq u^u \quad \forall i = 0, 1, \cdots, N-1 \\
& s_0 = \bar{s}
\end{aligned}
$$

where s_i are the states, u_i are the inputs, N is the time horizon, $\bar{s}$ are the initial states, and Q, R, A, and B are user defined data. The matrices Q_f, S, K, E, and F and the vectors w, g^l, g^u, s^l, s^u, u^l, and u^u are optional data. v_t is only needed in the condensed formulation, and it arises when K is defined by the user to ensure numerical stability of the condensed problem.

The condensed formulation used within DynamicNLPModels.jl is

$$
\begin{aligned}
\min_{\boldsymbol{v}}\ & \frac{1}{2}\boldsymbol{v}^\top \boldsymbol{H}\boldsymbol{v} + \boldsymbol{h}^\top \boldsymbol{v} + \boldsymbol{h}_0 \\
\text{s.t. } & d^l \leq \boldsymbol{J}\boldsymbol{v} \leq d^u.
\end{aligned}
$$

Chapter 2

Bug reports and support

This package is new and still undergoing some development. If you encounter a bug, please report it through Github's issue tracker.

Part II

Quick Start

Chapter 3

Getting Started

DynamicNLPModels.jl takes user defined data to construct a linear MPC problem of the form

$$\begin{aligned}
\min_{s,u,v}\ & s_N^\top Q_f s_N + \frac{1}{2}\sum_{i=0}^{N-1} \begin{bmatrix} s_i \\ u_i \end{bmatrix}^\top \begin{bmatrix} Q & S \\ S^\top & R \end{bmatrix} \begin{bmatrix} s_i \\ u_i \end{bmatrix} \\
\text{s.t. } & s_{i+1} = As_i + Bu_i + w_i \quad \forall i = 0, 1, \cdots, N-1 \\
& u_i = Ks_i + v_i \quad \forall i = 0, 1, \cdots, N-1 \\
& g^l \leq Es_i + Fu_i \leq g^u \quad \forall i = 0, 1, \cdots, N-1 \\
& s^l \leq s_i \leq s^u \quad \forall i = 0, 1, \cdots, N \\
& u^l \leq u_i \leq u^u \quad \forall i = 0, 1, \cdots, N-1 \\
& s_0 = \bar{s}.
\end{aligned}$$

This data is stored within the struct `LQDynamicData`, which can be created by passing the data `s0`, `A`, `B`, `Q`, `R` and `N` to the constructor as in the example below.

```
using DynamicNLPModels, Random, LinearAlgebra

Q  = 1.5 * Matrix(I, (3, 3))
R  = 2.0 * Matrix(I, (2, 2))
A  = rand(3, 3)
B  = rand(3, 2)
N  = 5
s0 = [1.0, 2.0, 3.0]

lqdd = LQDynamicData(s0, A, B, Q, R, N; **kwargs)
```

`LQDynamicData` contains the following fields. All fields after `R` are keyword arguments:

- `ns`: number of states (determined from size of `Q`)
- `nu`: number of inputs (determined from size of `R`)
- `N` : number of time steps
- `s0`: a vector of initial states

- `A` : matrix that is multiplied by the states that corresponds to the dynamics of the problem. Number of columns is equal to `ns`
- `B` : matrix that is multiplied by the inputs that coresonds to the dynamics of the problem. Number of columns is equal to `nu`
- `Q` : objective function matrix for system states from $0, 1, \cdots, (N-1)$
- `R` : objective function matrix for system inputs from $0, 1, \cdots, (N-1)$
- `Qf`: objective function matrix for system states at time N
- `S` : objective function matrix for system states and inputs
- `E` : constraint matrix multiplied by system states. Number of columns is equal to `ns`
- `F` : constraint matrix multiplied by system inputs. Number of columns is equal to `nu`
- `K` : feedback gain matrix. Used to ensure numerical stability of the condensed problem. Not necessary within the sparse problem
- `w` : constant term within dynamic constraints. At this time, this is the only data that is time varying. This vector must be length `ns * N`, where each set of `ns` entries corresponds to that time (i.e., entries `1:ns` correspond to time 0, entries `(ns + 1):(2 * ns)` corresond to time 1, etc.)
- `sl` : lower bounds on state variables
- `su` : upper bounds on state variables
- `ul` : lower bounds on ipnut variables
- `uu` : upper bounds on input variables
- `gl` : lower bounds on the constraints $Es_i + Fu_i$
- `gu` : upper bounds on the constraints $Es_i + Fu_i$

3.1 SparseLQDynamicModel

A `SparseLQDynamicModel` can be created by either passing `LQDynamicData` to the constructor or passing the data itself, where the same keyword options exist which can be used for `LQDynamicData`.

```
sparse_lqdm = SparseLQDynamicModel(lqdd)

# or

sparse_lqdm = SparseLQDynamicModel(s0, A, B, Q, R, N; **kwargs)
```

The `SparseLQDynamicModel` contains four fields:

- `dynamic_data` which contains the `LQDynamicData`
- `data` which is the `QPData` from QuadraticModels.jl. This object also contains the following data:
  - `H` which is the Hessian of the linear MPC problem
  - `A` which is the Jacobian of the linear MPC problem such that $\text{lcon} \leq Az \leq \text{ucon}$

  - `c` which is the linear term of a quadratic objective function
  - `c0` which is the constant term of a quadratic objective function

- `meta` which contains the `NLPModelMeta` for the problem from NLPModels.jl
- `counters` which is the `Counters` object from NLPModels.jl

!!! Note The `SparseLQDynamicModel` requires that all matrices in the `LQDynamicData` be the same type. It is recommended that the user be aware of how to most efficiently store their data in the `Q`, `R`, `A`, and `B` matrices as this impacts how efficiently the `SparseLQDynamicModel` is constructed. When `Q`, `R`, `A`, and `B` are sparse, building the `SparseLQDynamicModel` is much faster when these are passed as sparse rather than dense matrices.

3.2 DenseLQDynamicModel

The `DenseLQDynamicModel` eliminates the states within the linear MPC problem to build an equivalent optimization problem that is only a function of the inputs. This can be particularly useful when the number of states is large compared to the number of inputs.

A `DenseLQDynamicModel` can be created by either passing `LQDynamicData` to the constructor or passing the data itself, where the same keyword options exist which can be used for `LQDynamicData`.

```
dense_lqdm = DenseLQDynamicModel(lqdd)

# or

dense_lqdm = DenseLQDynamicModel(s0, A, B, Q, R, N; **kwargs)
```

The `DenseLQDynamicModel` contains five fields:

- `dynamic_data` which contains the `LQDynamicData`
- `data` which is the `QPData` from QuadraticModels.jl. This object also contains the following data:
  - `H` which is the Hessian of the condensed linear MPC problem
  - `A` which is the Jacobian of the condensed linear MPC problem such that $\text{lcon} \leq Az \leq \text{ucon}$
  - `c` which is the linear term of the condensed linear MPC problem
  - `c0` which is the constant term of the condensed linear MPC problem
- `meta` which contains the `NLPModelMeta` for the problem from NLPModels.jl
- `counters` which is the `Counters` object from NLPModels.jl
- `blocks` which contains the data needed to condense the model and then to update the condensed model when `s0` is reset.

The `DenseLQDynamicModel` is formed from dense matrices, and this dense system can be solved on a GPU using MadNLP.jl and MadNLPGPU.jl For an example script for performing this, please see the the examples directory of the main repository.

3.3 API functions

An API has been created for working with `LQDynamicData` and the sparse and dense models. All functions can be seen in the API Manual section. However, we give a short overview of these functions here.

- `reset_s0!(LQDynamicModel, new_s0)`: resets the model in place with a new `s0` value. This could be called after each sampling period in MPC to reset the model with a new measured value
- `get_s(solver_ref, LQDynamicModel)`: returns the optimal solution for the states from a given solver reference
- `get_u(solver_ref, LQDynamicModel)`: returns the optimal solution for the inputs from a given solver reference; when K is defined, the solver reference contains the optimal v values rather than optimal u values, adn this function converts v to u and returns the u values
- `get_*`: returns the data of * where * is an object within `LQDynamicData`
- `set_*!`: sets the value within the data of * for a given entry to a user defined value

Part III

API Manual

Chapter 4

API Manual

`DynamicNLPModels.DenseLQDynamicBlocks` – Type.

Struct containing block matrices used for creating and resetting the `DenseLQDynamicModel`. A and B matrices are given in part by Jerez, Kerrigan, and Constantinides in section 4 of "A sparse and condensed QP formulation for predictive control of LTI systems" (doi:10.1016/j.automatica.2012.03.010). States are eliminated by the equation $x = Ax_0 + Bu + \hat{A}w$ where $x = [x_0^T, x_1^T, ..., x_N^T]$ and $u = [u_0^T, u_1^T, ..., u_{N-1}^T]$

- `A` : block A matrix given by Jerez et al. with $n_s(N+1)$ rows and ns columns
- `B` : block B matrix given by Jerez et al. with $n_s(N)$ rows and nu columns
- `Aw` : length $n_s(N+1)$ vector corresponding to the linear term of the dynamic constraints
- `h` : $n_u(N) \times n_s$ matrix for building the linear term of the objective function. Just needs to be

multiplied by `s0`.

- `h01`: ns x ns matrix for building the constant term fo the objective function. This can be found by

taking s_0^T `h01` s_0

- `h02`: similar to `h01`, but one side is multiplied by `Aw` rather than by `As0`. This will just

be multiplied by `s0` once

- `h_constant` : linear term in the objective function that arises from `Aw`. Not a function of `s0`
- `h0_constant`: constant term in the objective function that arises from `Aw`. Not a function of `s0`
- `d` : length $n_c(N)$ term for the constraint bounds corresponding to E and F. Must be multiplied by `s0` and

subtracted from `gl` and `gu`. Equal to the blocks (E + FK) A (see Jerez et al.)

- `dw` : length $n_c(N)$ term for the constraint bounds that arises from `w`. Equal to the blocks (E + FK) Aw
- `KA` : size $n_u(N)$ x ns matrix. Needs to be multiplied by `s0` and subtracted from `ul` and uu to update

the algebraic constraints corresponding to the input bounds

- `KAw`: similar to KA, but it is multiplied by Aw rather than A

See also `reset_s0!`

source

`DynamicNLPModels.DenseLQDynamicModel` – Method.

```
DenseLQDynamicModel(dnlp::LQDynamicData; implicit = false)     -> DenseLQDynamicModel
DenseLQDynamicModel(s0, A, B, Q, R, N; implicit = false ...) -> DenseLQDynamicModel
```

A constructor for building a `DenseLQDynamicModel <: QuadraticModels.AbstractQuadraticModel`

Input data is for the problem of the form

$$\begin{aligned}
\min \frac{1}{2} \sum_{i=0}^{N-1} (s_i^T Q s_i + 2u_i^T S^T x_i + u_i^T R u_i) + \frac{1}{2} s_N^T Q_f s_N \\
\text{s.t. } s_{i+1} = As_i + Bu_i + w_i \quad \forall i = 0, 1, ..., N-1 \\
u_i = Kx_i + v_i \quad \forall i = 0, 1, ..., N-1 \\
gl \leq Es_i + Fu_i \leq gu \quad \forall i = 0, 1, ..., N-1 \\
sl \leq s \leq su \\
ul \leq u \leq uu \\
s_0 = s0
\end{aligned}$$

Data is converted to the form

$$\begin{aligned}
\min \frac{1}{2} z^T H z \\
\text{s.t. } \text{lcon} \leq Jz \leq \text{ucon} \\
\text{lvar} \leq z \leq \text{uvar}
\end{aligned}$$

Resulting `H`, `J`, `h`, and `h0` matrices are stored within `QuadraticModels.QPData` as `H`, `A`, `c`, and `c0` attributes respectively

If K is defined, then u variables are replaced by v variables. The bounds on u are transformed into algebraic constraints, and u can be queried by `get_u` and `get_s` within `DynamicNLPModels.jl`

Keyword argument `implicit = false` determines how the Jacobian is stored within the `QPData`. If `implicit = false`, the full, dense Jacobian matrix is stored. If `implicit = true`, only the first `nu` columns of the Jacobian are stored with the Linear Operator `LQJacobianOperator`.

source

`DynamicNLPModels.LQDynamicData` – Type.

```
LQDynamicData{T,V,M,MK} <: AbstractLQDynData{T,V}
```

A struct to represent the features of the optimization problem

$$
\begin{aligned}
\min \frac{1}{2} \sum_{i=0}^{N-1} (s_i^T Q s_i + 2u_i^T S^T x_i + u_i^T R u_i) + \frac{1}{2} s_N^T Q_f s_N \\
\text{s.t. } s_{i+1} = As_i + Bu_i + w_i \quad \forall i = 0, 1, ..., N-1 \\
u_i = Kx_i + v_i \quad \forall i = 0, 1, ..., N-1 \\
g^l \leq Es_i + Fu_i \leq g^u \quad \forall i = 0, 1, ..., N-1 \\
s^l \leq s \leq s^u \\
u^l \leq u \leq u^u \\
s_0 = s0
\end{aligned}
$$

Attributes include:

- `s0`: initial state of system
- `A` : constraint matrix for system states
- `B` : constraint matrix for system inputs
- `Q` : objective function matrix for system states from 0:(N-1)
- `R` : objective function matrix for system inputs from 0:(N-1)
- `N` : number of time steps
- `Qf`: objective function matrix for system state at time N
- `S` : objective function matrix for system states and inputs
- `ns`: number of state variables
- `nu`: number of input varaibles
- `E` : constraint matrix for state variables
- `F` : constraint matrix for input variables
- `K` : feedback gain matrix
- 'w' : constant term for dynamic constraints
- `sl`: vector of lower bounds on state variables
- `su`: vector of upper bounds on state variables
- `ul`: vector of lower bounds on input variables
- `uu`: vector of upper bounds on input variables
- `gl`: vector of lower bounds on constraints
- `gu`: vector of upper bounds on constraints

see also `LQDynamicData(s0, A, B, Q, R, N; ...)`

source

`DynamicNLPModels.LQDynamicData` – Method.

```
LQDynamicData(s0, A, B, Q, R, N; ...) -> LQDynamicData{T, V, M, MK}
```

A constructor for building an object of type `LQDynamicData` for the optimization problem

$$\begin{aligned}
\min \frac{1}{2} & \sum_{i=0}^{N-1} (s_i^T Q s_i + 2u_i^T S^T x_i + u_i^T R u_i) + \frac{1}{2} s_N^T Q_f s_N \\
\text{s.t. } & s_{i+1} = As_i + Bu_i + w_i \quad \forall i = 0, 1, ..., N-1 \\
& u_i = Kx_i + v_i \quad \forall i = 0, 1, ..., N-1 \\
& gl \leq Es_i + Fu_i \leq gu \quad \forall i = 0, 1, ..., N-1 \\
& sl \leq s \leq su \\
& ul \leq u \leq uu \\
& s_0 = s0
\end{aligned}$$

- `s0`: initial state of system
- `A` : constraint matrix for system states
- `B` : constraint matrix for system inputs
- `Q` : objective function matrix for system states from 0:(N-1)
- `R` : objective function matrix for system inputs from 0:(N-1)
- `N` : number of time steps

The following attributes of the `LQDynamicData` type are detected automatically from the length of s0 and size of R

- `ns`: number of state variables
- `nu`: number of input varaibles

The following keyward arguments are also accepted

- `Qf = Q`: objective function matrix for system state at time N; dimensions must be ns x ns
- `S = nothing`: objective function matrix for system state and inputs
- `E = zeros(eltype(Q), 0, ns)` : constraint matrix for state variables
- `F = zeros(eltype(Q), 0, nu)` : constraint matrix for input variables
- `K = nothing` : feedback gain matrix
- `w = zeros(eltype(Q), ns * N)` : constant term for dynamic constraints
- `sl = fill(-Inf, ns)`: vector of lower bounds on state variables
- `su = fill(Inf, ns)` : vector of upper bounds on state variables
- `ul = fill(-Inf, nu)`: vector of lower bounds on input variables
- `uu = fill(Inf, nu)` : vector of upper bounds on input variables
- `gl = fill(-Inf, size(E, 1))` : vector of lower bounds on constraints
- `gu = fill(Inf, size(E, 1))` : vector of upper bounds on constraints

source

`DynamicNLPModels.LQJacobianOperator` – Type.

`LQJacobianOperator{T, V, M}`

Struct for storing the implicit Jacobian matrix. All data for the Jacobian can be stored in the first nu columns of J. This struct contains the needed data and storage arrays for calculating Jx, J^Tx, and $J^T\Sigma J$. Jx and J^Tx are performed through extensions to `LinearAlgebra.mul!()`.

Attributes

- `truncated_jac1`: Matrix of first nu columns of the Jacobian corresponding to Ax + Bu constraints
- `truncated_jac2`: Matrix of first nu columns of the Jacobian corresponding to state variable bounds
- `truncated_jac3`: Matrix of first nu columns of the Jacobian corresponding to input variable bounds
- `N` : number of time steps
- `nu` : number of inputs
- `nc` : number of algebraic constraints of the form gl <= Es + Fu <= gu
- `nsc`: number of bounded state variables
- `nuc`: number of bounded input variables (if K is defined)
- `SJ1`: placeholder for storing data when calculating `ΣJ`
- `SJ2`: placeholder for storing data when calculating `ΣJ`
- `SJ3`: placeholder for storing data when calculating `ΣJ`
- `H_sub_block`: placeholder for storing data when adding `J^T ΣJ` to the Hessian

source

`DynamicNLPModels.SparseLQDynamicModel` – Method.

SparseLQDynamicModel(dnlp::LQDynamicData) -> SparseLQDynamicModel SparseLQDynamicModel(s0, A, B, Q, R, N; ...) -> SparseLQDynamicModel A constructor for building a `SparseLQDynamicModel <: QuadraticModels.AbstractQuadraticModel` Input data is for the problem of the form

$$\begin{aligned}
\min\ & \frac{1}{2}\sum_{i=0}^{N-1}(s_i^TQs_i + 2u_i^TS^Tx_i + u_i^TRu_i) + \frac{1}{2}s_N^TQ_fs_N \\
\text{s.t. } & s_{i+1} = As_i + Bu_i + w_i \quad \forall i = 0, 1, ..., N-1 \\
& u_i = Kx_i + v_i \quad \forall i = 0, 1, ..., N-1 \\
& gl \le Es_i + Fu_i \le gu \quad \forall i = 0, 1, ..., N-1 \\
& sl \le s \le su \\
& ul \le u \le uu \\
& s_0 = s0
\end{aligned}$$

Data is converted to the form

$$\begin{aligned}
\min\ & \frac{1}{2}z^THz \\
\text{s.t. } & \text{lcon} \le Jz \le \text{ucon} \\
& \text{lvar} \le z \le \text{uvar}
\end{aligned}$$

Resulting H and J matrices are stored as QuadraticModels.QPData within the SparseLQDynamicModel struct and variable and constraint limits are stored within NLPModels.NLPModelMeta

If K is defined, then u variables are replaced by v variables, and u can be queried by get_u and get_s within DynamicNLPModels.jl

source

DynamicNLPModels.add_jtsj! – Method.

```
add_jtsj!(H::M, Jac::LQJacobianOperator{T, V, M}, Σ::V, alpha::Number = 1, beta::Number = 1)
```

Generates Jac' Σ Jac and adds it to the matrix H.

alpha and beta are scalar multipliers such beta H + alpha Jac' Σ Jac is stored in H, overwriting the existing value of H

source

DynamicNLPModels.get_A – Method.

```
get_A(LQDynamicData)
get_A(SparseLQDynamicModel)
get_A(DenseLQDynamicModel)
```

Return the value of A from LQDynamicData or SparseLQDynamicModel.dynamic_data or DenseLQDynamicModel.dynamic_dat

source

DynamicNLPModels.get_B – Method.

```
get_B(LQDynamicData)
get_B(SparseLQDynamicModel)
get_B(DenseLQDynamicModel)
```

Return the value of B from LQDynamicData or SparseLQDynamicModel.dynamic_data or DenseLQDynamicModel.dynamic_dat

source

DynamicNLPModels.get_E – Method.

```
get_E(LQDynamicData)
get_E(SparseLQDynamicModel)
get_E(DenseLQDynamicModel)
```

Return the value of E from LQDynamicData or SparseLQDynamicModel.dynamic_data or DenseLQDynamicModel.dynamic_dat

source

DynamicNLPModels.get_F – Method.

```
get_F(LQDynamicData)
get_F(SparseLQDynamicModel)
get_F(DenseLQDynamicModel)
```

Return the value of F from LQDynamicData or SparseLQDynamicModel.dynamic_data or DenseLQDynamicModel.dynamic_dat

source

DynamicNLPModels.get_K – Method.

```
get_K(LQDynamicData)
get_K(SparseLQDynamicModel)
get_K(DenseLQDynamicModel)
```

Return the value of K from LQDynamicData or SparseLQDynamicModel.dynamic_data or DenseLQDynamicModel.dynamic_dat

source

DynamicNLPModels.get_N – Method.

```
get_N(LQDynamicData)
get_N(SparseLQDynamicModel)
get_N(DenseLQDynamicModel)
```

Return the value of N from LQDynamicData or SparseLQDynamicModel.dynamic_data or DenseLQDynamicModel.dynamic_dat

source

DynamicNLPModels.get_Q – Method.

```
get_Q(LQDynamicData)
get_Q(SparseLQDynamicModel)
get_Q(DenseLQDynamicModel)
```

Return the value of Q from LQDynamicData or SparseLQDynamicModel.dynamic_data or DenseLQDynamicModel.dynamic_dat

source

DynamicNLPModels.get_Qf – Method.

```
get_Qf(LQDynamicData)
get_Qf(SparseLQDynamicModel)
get_Qf(DenseLQDynamicModel)
```

Return the value of Qf from LQDynamicData or SparseLQDynamicModel.dynamic_data or DenseLQDynamicModel.dynamic_da

source

DynamicNLPModels.get_R – Method.

```
get_R(LQDynamicData)
get_R(SparseLQDynamicModel)
get_R(DenseLQDynamicModel)
```

Return the value of R from LQDynamicData or SparseLQDynamicModel.dynamic_data or DenseLQDynamicModel.dynamic_dat

source

DynamicNLPModels.get_S – Method.

```
get_S(LQDynamicData)
get_S(SparseLQDynamicModel)
get_S(DenseLQDynamicModel)
```

Return the value of S from LQDynamicData or SparseLQDynamicModel.dynamic_data or DenseLQDynamicModel.dynamic_dat

source

DynamicNLPModels.get_gl – Method.

```
get_gl(LQDynamicData)
get_gl(SparseLQDynamicModel)
get_gl(DenseLQDynamicModel)
```

Return the value of gl from LQDynamicData or SparseLQDynamicModel.dynamic_data or DenseLQDynamicModel.dynamic_dat

source

DynamicNLPModels.get_gu – Method.

```
get_gu(LQDynamicData)
get_gu(SparseLQDynamicModel)
get_gu(DenseLQDynamicModel)
```

Return the value of gu from LQDynamicData or SparseLQDynamicModel.dynamic_data or DenseLQDynamicModel.dynamic_da

source

DynamicNLPModels.get_jacobian – Method.

```
get_jacobian(lqdm::DenseLQDynamicModel) -> LQJacobianOperator
get_jacobian(Jac::AdjointLinearOpeartor{T, LQJacobianOperator}) -> LQJacobianOperator
```

Gets the LQJacobianOperator from DenseLQDynamicModel (if the QPdata contains a LQJacobian Operator) or returns the LQJacobian Operator from the adjoint of the LQJacobianOperator

source

DynamicNLPModels.get_ns – Method.

```
get_ns(LQDynamicData)
get_ns(SparseLQDynamicModel)
get_ns(DenseLQDynamicModel)
```

Return the value of ns from LQDynamicData or SparseLQDynamicModel.dynamic_data or DenseLQDynamicModel.dynamic_da

source

DynamicNLPModels.get_nu – Method.

```
get_nu(LQDynamicData)
get_nu(SparseLQDynamicModel)
get_nu(DenseLQDynamicModel)
```

Return the value of nu from LQDynamicData or SparseLQDynamicModel.dynamic_data or DenseLQDynamicModel.dynamic_da

source

DynamicNLPModels.get_s – Method.

gets(solutionref, lqdm::SparseLQDynamicModel) -> s <: vector gets(solutionref, lqdm::DenseLQDynamic-Model) -> s <: vector

Query the solution s from the solver. If lqdm <: SparseLQDynamicModel, the solution is queried directly from solution_ref.solution If lqdm <: DenseLQDynamicModel, then solution_ref.solution returns u (if K = nothing) or v (if K <: AbstactMatrix), and s is found form transforming u or v into s using A, B, and K matrices.

source

DynamicNLPModels.get_s0 – Method.

```
get_s0(LQDynamicData)
get_s0(SparseLQDynamicModel)
get_s0(DenseLQDynamicModel)
```

Return the value of s0 from LQDynamicData or SparseLQDynamicModel.dynamic_data or DenseLQDynamicModel.dynamic_da

source

DynamicNLPModels.get_sl – Method.

```
get_sl(LQDynamicData)
get_sl(SparseLQDynamicModel)
get_sl(DenseLQDynamicModel)
```

Return the value of sl from LQDynamicData or SparseLQDynamicModel.dynamic_data or DenseLQDynamicModel.dynamic_dat

source

DynamicNLPModels.get_su – Method.

```
get_su(LQDynamicData)
get_su(SparseLQDynamicModel)
get_su(DenseLQDynamicModel)
```

Return the value of su from LQDynamicData or SparseLQDynamicModel.dynamic_data or DenseLQDynamicModel.dynamic_da

source

DynamicNLPModels.get_u – Method.

getu(solutionref, lqdm::SparseLQDynamicModel) -> u <: vector getu(solutionref, lqdm::DenseLQDynamicModel) -> u <: vector

Query the solution u from the solver. If K = nothing, the solution for u is queried from solution_ref.solution

If K <: AbstractMatrix, solution_ref.solution returns v, and get_u solves for u using the K matrix (and the A and B matrices if lqdm <: DenseLQDynamicModel)

source

DynamicNLPModels.get_ul – Method.

```
get_ul(LQDynamicData)
get_ul(SparseLQDynamicModel)
get_ul(DenseLQDynamicModel)
```

Return the value of ul from LQDynamicData or SparseLQDynamicModel.dynamic_data or DenseLQDynamicModel.dynamic_dat

source

DynamicNLPModels.get_uu – Method.

```
get_uu(LQDynamicData)
get_uu(SparseLQDynamicModel)
get_uu(DenseLQDynamicModel)
```

Return the value of uu from LQDynamicData or SparseLQDynamicModel.dynamic_data or DenseLQDynamicModel.dynamic_da

source

DynamicNLPModels.get_w – Method.

```
get_w(LQDynamicData)
get_w(SparseLQDynamicModel)
get_w(DenseLQDynamicModel)
```

Return the value of w from LQDynamicData or SparseLQDynamicModel.dynamic_data or DenseLQDynamicModel.dynamic_dat

source

DynamicNLPModels.reset_s0! – Method.

```
reset_s0!(lqdm::SparseLQDynamicModel, s0)
reset_s0!(lqdm::DenseLQDynamicModel, s0)
```

Resets s0 within lqdm.dynamic_data. For a SparseLQDynamicModel, this updates the variable bounds which fix the value of s0. For a DenseLQDynamicModel, also resets the constraint bounds on the Jacobian and resets the linear and constant terms within the objective function (i.e., lqdm.data.c and lqdm.data.c0). This provides a way to update the model after each sample period.

source

DynamicNLPModels.set_A! – Method.

```
set_A!(LQDynamicData, row, col, val)
set_A!(SparseLQDynamicModel, row, col, val)
set_A!(DenseLQDynamicModel, row, col, val)
```

Set the value of entry A[row, col] to val for LQDynamicData, SparseLQDynamicModel.dynamic_data, or DenseLQDynamicModel.dynamic_data

source

DynamicNLPModels.set_B! – Method.

```
set_B!(LQDynamicData, row, col, val)
set_B!(SparseLQDynamicModel, row, col, val)
set_B!(DenseLQDynamicModel, row, col, val)
```

Set the value of entry B[row, col] to val for LQDynamicData, SparseLQDynamicModel.dynamic_data, or DenseLQDynamicModel.dynamic_data

source

DynamicNLPModels.set_E! – Method.

```
set_E!(LQDynamicData, row, col, val)
set_E!(SparseLQDynamicModel, row, col, val)
set_E!(DenseLQDynamicModel, row, col, val)
```

Set the value of entry E[row, col] to val for LQDynamicData, SparseLQDynamicModel.dynamic_data, or DenseLQDynamicModel.dynamic_data

source

DynamicNLPModels.set_F! – Method.

```
set_F!(LQDynamicData, row, col, val)
set_F!(SparseLQDynamicModel, row, col, val)
set_F!(DenseLQDynamicModel, row, col, val)
```

Set the value of entry F[row, col] to val for LQDynamicData, SparseLQDynamicModel.dynamic_data, or DenseLQDynamicModel.dynamic_data

source

DynamicNLPModels.set_K! – Method.

```
set_K!(LQDynamicData, row, col, val)
set_K!(SparseLQDynamicModel, row, col, val)
set_K!(DenseLQDynamicModel, row, col, val)
```

Set the value of entry K[row, col] to val for LQDynamicData, SparseLQDynamicModel.dynamic_data, or DenseLQDynamicModel.dynamic_data

source

DynamicNLPModels.set_Q! – Method.

```
set_Q!(LQDynamicData, row, col, val)
set_Q!(SparseLQDynamicModel, row, col, val)
set_Q!(DenseLQDynamicModel, row, col, val)
```

Set the value of entry Q[row, col] to val for LQDynamicData, SparseLQDynamicModel.dynamic_data, or DenseLQDynamicModel.dynamic_data

source

DynamicNLPModels.set_Qf! – Method.

```
set_Qf!(LQDynamicData, row, col, val)
set_Qf!(SparseLQDynamicModel, row, col, val)
set_Qf!(DenseLQDynamicModel, row, col, val)
```

Set the value of entry Qf[row, col] to val for LQDynamicData, SparseLQDynamicModel.dynamic_data, or DenseLQDynamicModel.dynamic_data

source

DynamicNLPModels.set_R! – Method.

```
set_R!(LQDynamicData, row, col, val)
set_R!(SparseLQDynamicModel, row, col, val)
set_R!(DenseLQDynamicModel, row, col, val)
```

Set the value of entry R[row, col] to val for LQDynamicData, SparseLQDynamicModel.dynamic_data, or DenseLQDynamicModel.dynamic_data

source

DynamicNLPModels.set_S! – Method.

```
set_S!(LQDynamicData, row, col, val)
set_S!(SparseLQDynamicModel, row, col, val)
set_S!(DenseLQDynamicModel, row, col, val)
```

Set the value of entry S[row, col] to val for LQDynamicData, SparseLQDynamicModel.dynamic_data, or DenseLQDynamicModel.dynamic_data

source

DynamicNLPModels.set_gl! – Method.

```
set_gl!(LQDynamicData, index, val)
set_gl!(SparseLQDynamicModel, index, val)
set_gl!(DenseLQDynamicModel, index, val)
```

Set the value of entry gl[index] to val for LQDynamicData, SparseLQDynamicModel.dynamic_data, or DenseLQDynamicModel.dynamic_data

source

DynamicNLPModels.set_gu! – Method.

```
set_gu!(LQDynamicData, index, val)
set_gu!(SparseLQDynamicModel, index, val)
set_gu!(DenseLQDynamicModel, index, val)
```

Set the value of entry gu[index] to val for LQDynamicData, SparseLQDynamicModel.dynamic_data, or DenseLQDynamicModel.dynamic_data

source

DynamicNLPModels.set_s0! – Method.

```
set_s0!(LQDynamicData, index, val)
set_s0!(SparseLQDynamicModel, index, val)
set_s0!(DenseLQDynamicModel, index, val)
```

Set the value of entry sO[index] to val for LQDynamicData, SparseLQDynamicModel.dynamic_data, or DenseLQDynamicModel.dynamic_data

source

DynamicNLPModels.set_sl! – Method.

```
set_sl!(LQDynamicData, index, val)
set_sl!(SparseLQDynamicModel, index, val)
set_sl!(DenseLQDynamicModel, index, val)
```

Set the value of entry sl[index] to val for LQDynamicData, SparseLQDynamicModel.dynamic_data, or DenseLQDynamicModel.dynamic_data

source

DynamicNLPModels.set_su! – Method.

```
set_su!(LQDynamicData, index, val)
set_su!(SparseLQDynamicModel, index, val)
set_su!(DenseLQDynamicModel, index, val)
```

Set the value of entry su[index] to val for LQDynamicData, SparseLQDynamicModel.dynamic_data, or DenseLQDynamicModel.dynamic_data

source

DynamicNLPModels.set_ul! – Method.

```
set_ul!(LQDynamicData, index, val)
set_ul!(SparseLQDynamicModel, index, val)
set_ul!(DenseLQDynamicModel, index, val)
```

Set the value of entry ul[index] to val for LQDynamicData, SparseLQDynamicModel.dynamic_data, or DenseLQDynamicModel.dynamic_data

source

DynamicNLPModels.set_uu! – Method.

```
set_uu!(LQDynamicData, index, val)
set_uu!(SparseLQDynamicModel, index, val)
set_uu!(DenseLQDynamicModel, index, val)
```

Set the value of entry uu[index] to val for LQDynamicData, SparseLQDynamicModel.dynamic_data, or DenseLQDynamicModel.dynamic_data

source

LinearOperators.reset! – Method.

```
LinearOperators.reset!(Jac::LQJacobianOperator{T, V, M})
```

Resets the values of attributes SJ1, SJ2, and SJ3 to zero

source