DynamicNLPModels

David Cole, Sungho Shin, Francois Pacaud

August 12, 2022

Contents

Part I

Introduction

Chapter 1

Introduction

Welcome to the documentation of DynamicNLPModels.jl

Warning

This documentation page is under construction.

Note

This documentation is also available in PDF format.

Chapter 2

What is DynamicNLPModels?

Chapter 3

Bug reports and support

Please report issues and feature requests via the Github issue tracker.

Part II

Quick Start

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 = Ax0 + Bu + \hat{A}w where x = [x0^T, x1^T, ..., xN^T] and u = [u0^T, u1^T, ..., u_{N-1}^T]

- `A` : block A matrix given by Jerez et al. with ns(N + 1) rows and ns columns
- `B` : block B matrix given by Jerez et al. with ns(N) rows and nu columns
- `Aw` : length ns(N + 1) vector corresponding to the linear term of the dynamic constraints
- `h` : nu(N) x ns 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 s0^T h01 s0

- `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 nc(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 nc(N) term for the constraint bounds that arises from `w`. Equal to the blocks (E + FK) Aw
- `KA` : size nu(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

$$minimize \frac{1}{2}\sum_{i=0}^{N-1}(s_i^T Q s_i + 2u_i^T S^T x_i + u_i^T R u_i) + \frac{1}{2}s_N^T Q f s_N subject to s_{i+1} = As_i + Bu_i + w for i = 0, 1, ..., N-1 u_i$$

Data is converted to the form

$$minimize \frac{1}{2}u^T H u + h^T u + h0 subject to Jz \leq gul \leq u \leq uu$$

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

$$minimize \frac{1}{2}\sum_{i=0}^{N-1}(s_i^T Q s_i + 2u_i^T S^T x_i + u_i^T R u_i) + \frac{1}{2}s_N^T Q f s_N subject to s_{i+1} = As_i + Bu_i + w for i = 0, 1, ..., N-1 u_i$$

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 1:(N-1)
- `R` : objective function matrix for system inputs from 1:(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

$$minimize \frac{1}{2}\sum_{i=0}^{N-1}(s_i^T Q s_i + 2u_i^T S^T x_i + u_i^T R u_i) + \frac{1}{2}s_N^T Q f s_N subject to s_{i+1} = As_i + Bu_i + w \forall i = 0, 1, ..., N-1 u_i =$$

- 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 1:(N-1)
- R : objective function matrix for system inputs from 1:(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)` : 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^T x`, and `J^T Sigma J`. `Jx` and `J^T x` 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

$$minimize \frac{1}{2}\sum_{i=0}^{N-1}(s_i^T Q s_i + 2u_i^T S^T x_i + u_i^T R u_i) + \frac{1}{2}s_N^T Q f s_N subject to s_{i+1} = As_i + Bu_i + w for i = 0, 1, ..., N-1 u_i$$

Data is converted to the form

$$minimize \frac{1}{2} z^T H z subject to lcon \leq Jz \leq ucon lvar \leq z \leq uvar$$

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._set_sparse_H! – Method.

```
_set_sparse_H!(H_colptr, H_rowval, H_nzval, Q, R, N; Qf = Q, S = zeros(T, size(Q, 1), size(R,
↪  1))
```

set the data needed to build a SparseArrays.SparseMatrixCSC matrix. Hcolptr, Hrowval, and Hnzval are set so that they can be passed to SparseMatrixCSC() to obtain the H matrix such that z^T H z = sum{i=1}^{N-1} si^T Q s + sum{i=1}^{N-1} u^T R u + sN^T Qf sn .

source

DynamicNLPModels._set_sparse_J! – Method.

```
_set_sparse_J!(J_colptr, J_rowval, J_nzval, A, B, E, F, K, bool_vec, N, nb)
_set_sparse_J!(J_colptr, J_rowval, J_nzval, A, B, E, F, K, N)
```

set the data needed to build a SparseArrays.SparseMatrixCSC matrix. Jcolptr, Jrowval, and J_nzval are set so that they can be passed to SparseMatrixCSC() to obtain the Jacobian, J. The Jacobian contains the data for the following constraints:

Asi + Bui = s{i + 1} gl <= Esi + Fui <= getu

If K is defined, then this matrix also contains the constraints ul <= Kxi + vi <= uu

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 A from LQDynamicData or SparseLQDynamicModel.dynamicdata or DenseLQDynamicModel.dynamicdata

source

DynamicNLPModels.get_B – Method.

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

Return the value B from LQDynamicData or SparseLQDynamicModel.dynamicdata or DenseLQDynamicModel.dynamicdata

source

DynamicNLPModels.get_E – Method.

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

Return the value E from LQDynamicData or SparseLQDynamicModel.dynamicdata or DenseLQDynamicModel.dynamicdata

source

DynamicNLPModels.get_F – Method.

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

Return the value F from LQDynamicData or SparseLQDynamicModel.dynamicdata or DenseLQDynamicModel.dynamicdata

source

DynamicNLPModels.get_K – Method.

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

Return the value K from LQDynamicData or SparseLQDynamicModel.dynamicdata or DenseLQDynamicModel.dynamicdata

source

DynamicNLPModels.get_N – Method.

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

Return the value N from LQDynamicData or SparseLQDynamicModel.dynamicdata or DenseLQDynamicModel.dynamicdata

source

DynamicNLPModels.get_Q – Method.

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

Return the value Q from LQDynamicData or SparseLQDynamicModel.dynamicdata or DenseLQDynamicModel.dynamicdata

source

DynamicNLPModels.get_Qf – Method.

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

Return the value Qf from LQDynamicData or SparseLQDynamicModel.dynamicdata or DenseLQDynamicModel.dynamicdata

source

DynamicNLPModels.get_R – Method.

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

Return the value R from LQDynamicData or SparseLQDynamicModel.dynamicdata or DenseLQDynamicModel.dynamicdata

source

DynamicNLPModels.get_S – Method.

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

Return the value S from LQDynamicData or SparseLQDynamicModel.dynamicdata or DenseLQDynamicModel.dynamicdata

source

DynamicNLPModels.get_gl – Method.

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

Return the value gl from LQDynamicData or SparseLQDynamicModel.dynamicdata or DenseLQDynamicModel.dynamicdata

source

DynamicNLPModels.get_gu – Method.

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

Return the value gu from LQDynamicData or SparseLQDynamicModel.dynamicdata or DenseLQDynamicModel.dynamicdata

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 ns from LQDynamicData or SparseLQDynamicModel.dynamicdata or DenseLQDynamicModel.dynamicdata

source

DynamicNLPModels.get_nu – Method.

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

Return the value nu from LQDynamicData or SparseLQDynamicModel.dynamicdata or DenseLQDynamicModel.dynamicdata

source

DynamicNLPModels.get_s – Method.

gets(solutionref, lqdm::SparseLQDynamicModel) -> s <: vector gets(solutionref, lqdm::DenseLQDynamicModel) -> 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 s0 from LQDynamicData or SparseLQDynamicModel.dynamicdata or DenseLQDynamicModel.dynamicdata

source

DynamicNLPModels.get_sl – Method.

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

Return the value sl from LQDynamicData or SparseLQDynamicModel.dynamicdata or DenseLQDynamicModel.dynamicdata

source

DynamicNLPModels.get_su – Method.

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

Return the value su from LQDynamicData or SparseLQDynamicModel.dynamicdata or DenseLQDynamicModel.dynamicdata

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 ul from LQDynamicData or SparseLQDynamicModel.dynamicdata or DenseLQDynamicModel.dynamicdata

source

DynamicNLPModels.get_uu – Method.

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

Return the value uu from LQDynamicData or SparseLQDynamicModel.dynamicdata or DenseLQDynamicModel.dynamicdata

source

DynamicNLPModels.get_w – Method.

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

Return the value w from LQDynamicData or SparseLQDynamicModel.dynamicdata or DenseLQDynamicModel.dynamicdata

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.dynamicdata, or DenseLQDynamicModel.dynamicdata

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.dynamicdata, or DenseLQDynamicModel.dynamicdata

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.dynamicdata, or DenseLQDynamicModel.dynamicdata

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.dynamicdata, or DenseLQDynamicModel.dynamicdata

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.dynamicdata, or DenseLQDynamicModel.dynamicdata

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.dynamicdata, or DenseLQDynamicModel.dynamicdata

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.dynamicdata, or DenseLQDynamicModel.dynamicdata

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.dynamicdata, or DenseLQDynamicModel.dynamicdata

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.dynamicdata, or DenseLQDynamicModel.dynamicdata

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.dynamicdata, or DenseLQDynamicModel.dynamicdata

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.dynamicdata, or DenseLQDynamicModel.dynamicdata

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 s0[index] to val for LQDynamicData, SparseLQDynamicModel.dynamicdata, or DenseLQDynamicModel.dynamicdata

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.dynamicdata, or DenseLQDynamicModel.dynamicdata

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.dynamicdata, or DenseLQDynamicModel.dynamicdata

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.dynamicdata, or DenseLQDynamicModel.dynamicdata

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.dynamicdata, or DenseLQDynamicModel.dynamicdata

source

`LinearOperators.reset!` – Method.

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

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

source