Engines

Generated by Doxygen 1.8.13

Contents

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

Module Documentation

4.1 Connection_mesh

Functions

- int conn_mesh::init (std::string conn2p_filename)

  init mesh by reading TPFACONNS keyword file
- int conn_mesh::init (std::vector< index_t > &block_m, std::vector< index_t > &block_p, std::vector< value_t > &tran, std::vector< value_t > &tranD)

  init mesh by reading array of left/right neighbours
- int conn_mesh::init_mpfa (std::vector< index_t > &block_m, std::vector< index_t > &block_p, std::vector< index_t > &_fstencil, std::vector< index_t > &_fst_offset, std::vector< value_t > &_ftran, std::vector< value_t > &_rhs, uint8_t _n_dim, index_t _n_matrix, index_t _n_bounds)

  init mesh by reading MPFA connections
- int conn_mesh::init_mpsa (std::vector< index_t > &block_m, std::vector< index_t > &block_p, std::vector< index_t > &_sstencil, std::vector< index_t > &_sst_offset, std::vector< value_t > &_stran, uint8_t _n_dim, index_t _n_matrix, index_t _n_bounds, index_t _n_fracs)

  init mesh by reading MPSA connections
- int conn_mesh::init_pm (std::vector< index_t > &block_m, std::vector< index_t > &block_p, std::vector< index_t > &_stencil, std::vector< index_t > &_st_offset, std::vector< value_t > &_tran, std::vector< value_t > &_rhs, index_t _n_matrix, index_t _n_bounds, index_t _n_fracs)

  init mesh by reading both MPFA and MPSA connections
- int **conn_mesh::init_pm** (std::vector< index_t > &block_m, std::vector< index_t > &block_p, std::vector< index_t > &_stencil, std::vector< index_t > &_st_offset, std::vector< value_t > &_tran, std::vector< value_t > &_rhs, std::vector< value_t > &_tran_biot, std::vector< value_t > &_rhs_biot, index_t _n_matrix, index_t _n_bounds, index_t _n_fracs)
- int conn_mesh::init_const_1d (double trans_const, index_t nb)

  init mesh for 1D reservoir with 'nb' blocks
- int conn_mesh::add_conn (index_t block_m, index_t block_p, value_t trans)

  add a new connection to connection list
- int **conn_mesh::add_conn_mpfa** (index_t block_m, index_t block_p, value_t trans)
- int conn_mesh::reverse_and_sort ()

  reverse connections and sort them by both row and col
- int conn_mesh::reverse_and_sort_dvel ()

  reverse connections and renumerate velocity mappers and sort them by both row and col
- int conn_mesh::reverse_and_sort_mpfa ()

  reverse mpsa connections and sort them by both row and col

- int conn_mesh::reverse_and_sort_mpsa ()
- int conn_mesh::reverse_and_sort_pm ()
- int conn_mesh::add_wells (std::vector< ms_well *> &wells)

  discretize ms wells into reservoir
- int **conn_mesh::add_wells_mpfa** (std::vector< ms_well *> &wells)

Variables

- std::vector< index_t > conn_mesh::block_m

  [n_conns] array of indices of blocks on the minus side of a connection (smaller index)
- std::vector< index_t > conn_mesh::block_p

  [n_conns] array of indices of blocks on the plus side of a connection (bigger index)
- std::vector< value_t > conn_mesh::tran

  [n_conns] array of transissibility values for given connection
- std::vector< value_t > conn_mesh::tranD

  [n_conns] array of diffusion transissibility values for given connection (transmis value)
- std::vector< value_t > conn_mesh::grav_coef

  *[n_conns] array of gravity coefficient for every connection (= (depth[block_m] - depth[block_p]) * g)*
- std::vector< value_t > conn_mesh::velocity

  [n_conns] array of initial velocity values (Decouple - velocity engine)
- std::vector< value_t > conn_mesh::tran_const

  [n_conns] array of temporary const transmissibilities
- uint8_t conn_mesh::n_vars

  Number of unknowns per block.
- std::vector< index_t > conn_mesh::fstencil

  array of indices of blocks are neccessary for each connection
- std::vector< index_t > conn_mesh::fst_offset

  [n_conns + 1] array of offsets of the first block of connection in 'stencil'
- std::vector< index_t > conn_mesh::sstencil

  array of indices of blocks are neccessary for each connection
- std::vector< index_t > conn_mesh::sst_offset

  [n_conns + 1] array of offsets of the first block of connection in 'stencil'
- std::vector< value_t > conn_mesh::stran

  [n_conns] array of transissibility values for given connection
- std::vector< value_t > conn_mesh::stran_biot

  [n_conns] array of transissibility values for biot contribution to the given connection
- std::vector< std::vector< int > > **conn_mesh::stencil**
- std::vector< value_t > **conn_mesh::rhs**
- std::vector< value_t > **conn_mesh::rhs_biot**
- std::vector< value_t > **conn_mesh::f**
- index_t conn_mesh::n_links

  number of non-zero links
- std::vector< value_t > conn_mesh::volume

  [n_blocks] array of volumes of mesh blocks
- std::vector< value_t > conn_mesh::poro

  [n_blocks] array of porosities of mesh blocks
- std::vector< value_t > conn_mesh::depth

  [n_blocks] array of depths
- std::vector< value_t > conn_mesh::heat_capacity

  [n_blocks] array of heat capacity of rock

- std::vector< value_t > conn_mesh::rock_cond
  [n_blocks] array of heat conduction of rock;
- std::vector< value_t > conn_mesh::kin_factor
  [n_blocks] array of kinetic rate constants (dependent on the initial porosity and other factors!);
- std::vector< value_t > conn_mesh::pressure
  [n_blocks] array of initial pressure values
- std::vector< value_t > conn_mesh::composition
  [n_blocks] array of initial composition values
- std::vector< value_t > conn_mesh::temperature
  [n_blocks] array of initial temperature values
- std::vector< value_t > conn_mesh::enthalpy
  [n_blocks] array of initial enthalpy values
- std::vector< value_t > conn_mesh::displacement
  *[n_dim * n_blocks] array of initial displacements values*
- std::vector< value_t > conn_mesh::bc
  *[(1 + n_dim) * n_bounds] array of boundary conditions*
- std::vector< value_t > conn_mesh::bc_prev
  *[(1 + n_dim) * n_bounds] array of boundary conditions*
- std::vector< value_t > conn_mesh::pz_bounds
  *[nc * n_bounds] array of pressures and (inflow) fractions at boundaries*
- std::vector< value_t > conn_mesh::biot
  *[9 * n_blocks] array of biot coefficients of mesh blocks*
- std::vector< value_t > conn_mesh::drained_compressibility
  [n_blocks] array of drained compressibility of mesh blocks
- std::vector< index_t > conn_mesh::op_num
  [n_blocks] array of operator set index for every block

4.1.1 Detailed Description

Parameters and methods in mesh class exposed to Python

4.1.2 Function Documentation

4.1.2.1 reverse_and_sort_mpfa()

```
int conn_mesh::reverse_and_sort_mpfa ( )
```

reverse mpsa connections and sort them by both row and col

[n_blocks] map of connections [block_p, conn_idx] per block

4.1.2.2 reverse_and_sort_mpsa()

```
int conn_mesh::reverse_and_sort_mpsa ( )
```

[n_blocks] map of connections [block_p, conn_idx] per block

4.1.2.3 reverse_and_sort_pm()

```
int conn_mesh::reverse_and_sort_pm ( )
```

[n_blocks] map of connections [block_p, conn_idx] per block

4.2 Engine_methods

Functions

- virtual int engine_base::run (value_t n_days)

  runs simulation for the number of days using internal time management
- virtual int engine_base::run_timestep (value_t deltat, value_t time)

  runs simulation for one timestep starting from particular time
- virtual int engine_base::run_single_newton_iteration (value_t deltat)

  report for one newton iteration
- virtual int **engine_base::solve_linear_equation** ()
- virtual int **engine_base::post_newtonloop** (value_t deltat, value_t time)
- virtual int engine_base::report ()

  reports complete information about well regimes
- virtual int engine_base::print_stat ()

  print statistics for the current run
- virtual int **engine_base::test_assembly** (int n_times, int kernel_number=0, int dump_jacobian=0)
- virtual int **engine_base::test_spmv** (int n_timer, int kernel_number=0, int dump_result=0)
- virtual int engine_base_gpu::run_timestep (value_t deltat, value_t time)

  runs simulation for one timestep starting from particular time
- virtual int **engine_base_gpu::test_assembly** (int n_times, int kernel_number=0, int dump_jacobian_rhs=0)
- virtual int **engine_base_gpu::test_spmv** (int n_times, int kernel_number=0, int dump_result=0)

Variables

- value_t * **engine_base_gpu::X_d**
- value_t * **engine_base_gpu::Xn_d**
- value_t * **engine_base_gpu::dX_d**
- value_t * **engine_base_gpu::RHS_d**
- value_t * **engine_base_gpu::RHS_wells_d**
- std::vector< value_t > **engine_base_gpu::jac_wells**
- value_t * **engine_base_gpu::jac_wells_d**
- std::vector< index_t > **engine_base_gpu::jac_well_head_idxs**
- index_t * **engine_base_gpu::jac_well_head_idxs_d**
- value_t * **engine_base_gpu::op_vals_arr_d**
- value_t * **engine_base_gpu::op_ders_arr_d**
- value_t * **engine_base_gpu::op_vals_arr_n_d**
- std::vector< index_t * > **engine_base_gpu::block_idxs_d**
- value_t * **engine_base_gpu::RV_d**
- value_t * **engine_base_gpu::PV_d**
- value_t * **engine_base_gpu::mesh_tran_d**
- value_t * **engine_base_gpu::mesh_tranD_d**
- value_t * **engine_base_gpu::mesh_hcap_d**
- int **engine_base_gpu::min_grid_size**
- int **engine_base_gpu::grid_size**
- std::vector< int > **engine_base_gpu::assembly_block_sizes**

4.2.1 Detailed Description

Methods of base engine class exposed to Python

4.2.2 Function Documentation

4.2.2.1 print_stat()

```
int engine_base::print_stat ( )  [virtual]
```

print statistics for the current run

This procedure prints all statistics of simultion run

Note

The statistics includes the following items:

- total timesteps n_timesteps_total,
- wasted timesteps n_timesteps_wasted,
- total number of Neton iterations n_newton_total,
- number of wated Newton iterations n_newton_wasted,
- number of linear iterations n_linear_total,
- number of waterd linear iterations n_linear_wasted
- extended OBL statistics

4.2.2.2 report()

```
int engine_base::report ( )  [virtual]
```

reports complete information about well regimes

This method adds averaged and accumulated phase rates over the period since the last report call till current moment (possibly spannin over several timesteps) to time_data_report array.

4.2.2.3 run()

```
int engine_base::run (
            value_t n_days )  [virtual]
```

runs simulation for the number of days using internal time management

This base method runs simulation for particular time period. It use various parameters kept from the previous run in internal structures. The timestep managements is performed based on internal parameters (see 'params' class)

Parameters

n_days	- number of days for simulation

Note

this procedure involve several internal parameters including:

- mesh - mesh information for simulation
- params - all parameters for simulation
- wells - well input used in simulation
- Xn and X - solution for initial and following timesteps

4.2.2.4 run_timestep() [1/2]

```
int engine_base_gpu::run_timestep (
            value_t deltat,
            value_t time )  [virtual]
```

runs simulation for one timestep starting from particular time

This base method runs simulation for a single timetep. For compatibility between different runs, optional value for a current timetep can be provided.

Parameters

n_days	- number of days for simulation
time	- starting time for the timestep

Note

this procedure involve several internal parameters including:

- mesh - mesh information for simulation
- params - all parameters for simulation
- wells - well input used in simulation
- Xn and X - solution for initial and following timesteps

Reimplemented from engine_base.

4.2.2.5 run_timestep() [2/2]

```
int engine_base::run_timestep (
            value_t deltat,
            value_t time )  [virtual]
```

runs simulation for one timestep starting from particular time

This base method runs simulation for a single timetep. For compatibility between different runs, optional value for a current timetep can be provided.

Parameters

n_days	- number of days for simulation
time	- starting time for the timestep

Note

this procedure involve several internal parameters including:

- mesh - mesh information for simulation
- params - all parameters for simulation
- wells - well input used in simulation
- Xn and X - solution for initial and following timesteps

Reimplemented in engine_base_gpu.

4.3 Engine_parameters

Variables

- std::vector< value_t > engine_base::X
  vector of unknowns in the current timestep
- std::vector< value_t > engine_base::Xn
  vector of unknowns in the previous timestep
- value_t engine_base::t
  current timestep
- conn_mesh * engine_base::mesh
  pointer to mesh
- sim_params * engine_base::params
  simulation parameters
- sim_stat engine_base::stat
  simulation statistics
- std::vector< ms_well * > engine_base::wells
  vector of wells
- std::unordered_map< std::string, std::vector< value_t > > engine_base::time_data
  unsorted map containing well information (BHP, rates)

4.3.1 Detailed Description

Parameters in base engine class exposed to Python

4.4 Well_controls

Classes

- class bhp_inj_well_control

  BHP control for injection compositional well.
- class bhp_prod_well_control

  BHP control for production compositional well.
- class rate_inj_well_control

  Volumetric rate control for injection compositional well.
- class rate_prod_well_control

  Volumetric rate control for production compositional well.
- class gt_bhp_temp_inj_well_control

  BHP and temperature control for injection geothermal well.
- class gt_bhp_prod_well_control

  BHP control for production geothermal well.
- class gt_rate_temp_inj_well_control

  Volumetric rate and temperature control for injection geothermal well.
- class gt_rate_prod_well_control

  Volumetric rate control for production geothermal well.
- class gt_mass_rate_enthalpy_inj_well_control

  Mass rate and enthalpy control for injection geothermal well.
- class gt_mass_rate_prod_well_control

  Mass rate control for production geothermal well.

4.4.1 Detailed Description

Methods for well control/constraint exposed to Python

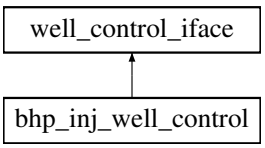
Chapter 5

Class Documentation

5.1 bhp_inj_well_control Class Reference

BHP control for injection compositional well.

Inheritance diagram for bhp_inj_well_control:



Public Member Functions

- **bhp_inj_well_control** (value_t target_pressure_, std::vector< value_t > &injection_stream_)
- virtual int **add_to_jacobian** (value_t dt, index_t well_head_idx, value_t segment_trans, index_t n_block_size, std::vector< value_t > &X, value_t *jacobian_row, std::vector< value_t > &RHS)
- virtual int **add_to_csr_jacobian** (value_t dt, index_t well_head_idx, value_t segment_trans, index_t n_↩ block_size, std::vector< value_t > &X, value_t *jacobian_row, std::vector< value_t > &RHS)
- virtual int **check_constraint_violation** (value_t dt, index_t well_head_idx, value_t segment_trans, index_t n_block_size, std::vector< value_t > &X)
- virtual int **initialize_well_block** (std::vector< value_t > &state_block, const std::vector< value_t > &state↩ _neighbour)

Public Attributes

- value_t **target_pressure**
- std::vector< value_t > **injection_stream**

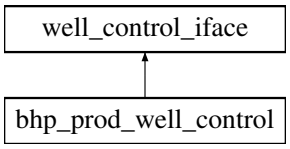
5.1.1 Detailed Description

BHP control for injection compositional well.

5.2 bhp_prod_well_control Class Reference

BHP control for production compositional well.

Inheritance diagram for bhp_prod_well_control:



Public Member Functions

- **bhp_prod_well_control** (value_t target_pressure_)
- virtual int **add_to_jacobian** (value_t dt, index_t well_head_idx, value_t segment_trans, index_t n_block_size, std::vector< value_t > &X, value_t *jacobian_row, std::vector< value_t > &RHS)
- virtual int **add_to_csr_jacobian** (value_t dt, index_t well_head_idx, value_t segment_trans, index_t n_block_size, std::vector< value_t > &X, value_t *jacobian_row, std::vector< value_t > &RHS)
- virtual int **check_constraint_violation** (value_t dt, index_t well_head_idx, value_t segment_trans, index_t n_block_size, std::vector< value_t > &X)
- virtual int **initialize_well_block** (std::vector< value_t > &state_block, const std::vector< value_t > &state_neighbour)

Public Attributes

- value_t **target_pressure**

5.2.1 Detailed Description

BHP control for production compositional well.

5.3 block_perf Struct Reference

Structure for well perforation.

Public Member Functions

- block_perf ()

  transmissibilty between reservoir block and well block

Public Attributes

- index_t **reservoir_block**
- value_t well_index

  the block of reservoir which is perforated

5.3.1 Detailed Description

Structure for well perforation.

5.4 block_well Class Reference

Class for well definition (both segments and controls)

Public Member Functions

- void **set_nc** (int nc_)

Public Attributes

- std::vector< block_perf > **block_perf_list**
- std::vector< std::pair< int, well_control > > **control_list**
- well_control current_control

  a list of well controls and correspondent report timestep
- index_t first_well_block_index

  effective well control to be used in jacobian assembly
- index_t ghost_well_block_index

  index of the first well block, which connects to ghost well block
- index_t NC

  index of the first well block, which connects to ghost well block
- value_t **well_block_volume**
- value_t well_block_trans

  volume of well block
- value_t * current_rates

  transmissibility bewteen well blocks
- value_t current_BHP

  current molar rates taken as influx/outflux of ghost block for production/injection well respectively
- value_t current_temp

  current BHP for well
- well_control_iface * control

  current temperature for well
- well_control_iface * constraint

  set of controls
- std::string name

  set of constraints

5.4.1 Detailed Description

Class for well definition (both segments and controls)

5.5 conn_mesh Class Reference

This class defines mesh and corresponding arrays.

Public Member Functions

- int **init_poro** (std::string poro_filename)
- int **init_grav_coef** (value_t grav_const=9.80665e-5)
- int **save_keyword_compressed** (std::string filename, std::string keyword, value_t *data, index_t length)
- int **get_res_tran** (std::vector< value_t > &res_tran, std::vector< value_t > &res_tranD)
- int **set_res_tran** (std::vector< value_t > &res_tran, std::vector< value_t > &res_tranD)
- int **get_wells_tran** (std::vector< value_t > &wells_tran)
- int **set_wells_tran** (std::vector< value_t > &wells_tran)
- int **save_volume** (std::string filename)
- int **save_poro** (std::string filename)
- int **save_pressure** (std::string filename)
- int **save_zmf** (std::string filename)
- int **save_temperature** (std::string filename)
- int **save_enthalpy** (std::string filename)
- int **save_wells** (std::string filename, std::vector< block_well > &well_list, sim_params &params)
- int init (std::string conn2p_filename)

  init mesh by reading TPFACONNS keyword file
- int init (std::vector< index_t > &block_m, std::vector< index_t > &block_p, std::vector< value_t > &tran, std::vector< value_t > &tranD)

  init mesh by reading array of left/right neighbours
- int init_mpfa (std::vector< index_t > &block_m, std::vector< index_t > &block_p, std::vector< index_t > &↩ _fstencil, std::vector< index_t > &_fst_offset, std::vector< value_t > &_ftran, std::vector< value_t > &_rhs, uint8_t _n_dim, index_t _n_matrix, index_t _n_bounds)

  init mesh by reading MPFA connections
- int init_mpsa (std::vector< index_t > &block_m, std::vector< index_t > &block_p, std::vector< index_t > &_sstencil, std::vector< index_t > &_sst_offset, std::vector< value_t > &_stran, uint8_t _n_dim, index_t _n_matrix, index_t _n_bounds, index_t _n_fracs)

  init mesh by reading MPSA connections
- int init_pm (std::vector< index_t > &block_m, std::vector< index_t > &block_p, std::vector< index_t > &↩ _stencil, std::vector< index_t > &_st_offset, std::vector< value_t > &_tran, std::vector< value_t > &_rhs, index_t _n_matrix, index_t _n_bounds, index_t _n_fracs)

  init mesh by reading both MPFA and MPSA connections
- int **init_pm** (std::vector< index_t > &block_m, std::vector< index_t > &block_p, std::vector< index_t > &↩ _stencil, std::vector< index_t > &_st_offset, std::vector< value_t > &_tran, std::vector< value_t > &_rhs, std::vector< value_t > &_tran_biot, std::vector< value_t > &_rhs_biot, index_t _n_matrix, index_t _n_↩ bounds, index_t _n_fracs)
- int init_const_1d (double trans_const, index_t nb)

  init mesh for 1D reservoir with 'nb' blocks
- int add_conn (index_t block_m, index_t block_p, value_t trans)

  add a new connection to connection list
- int **add_conn_mpfa** (index_t block_m, index_t block_p, value_t trans)
- int reverse_and_sort ()

  reverse connections and sort them by both row and col
- int reverse_and_sort_dvel ()

  reverse connections and renumerate velocity mappers and sort them by both row and col
- int reverse_and_sort_mpfa ()

  reverse mpsa connections and sort them by both row and col

- int reverse_and_sort_mpsa ()
- int reverse_and_sort_pm ()
- int add_wells (std::vector< ms_well *> &wells)

  discretize ms wells into reservoir
- int **add_wells_mpfa** (std::vector< ms_well *> &wells)

Public Attributes

- index_t **n_res_blocks**
- index_t **n_res_well_blocks**
- index_t **n_blocks**
- index_t **n_conns**
- index_t **n_perfs**
- index_t **n_matrix**
- index_t **n_bounds** = 0
- index_t **n_fracs** = 0
- std::vector< index_t > **one_way_to_conn_index_forward**
- std::vector< index_t > **one_way_to_conn_index_reverse**
- std::vector< index_t > **conn_index_to_one_way**
- std::vector< index_t > block_m

  [n_conns] array of indices of blocks on the minus side of a connection (smaller index)
- std::vector< index_t > block_p

  [n_conns] array of indices of blocks on the plus side of a connection (bigger index)
- std::vector< value_t > tran

  [n_conns] array of transissibility values for given connection
- std::vector< value_t > tranD

  [n_conns] array of diffusion transissibility values for given connection (transmis value)
- std::vector< value_t > grav_coef

  *[n_conns] array of gravity coefficient for every connection (= (depth[block_m] - depth[block_p]) * g)*
- std::vector< value_t > velocity

  [n_conns] array of initial velocity values (Decouple - velocity engine)
- std::vector< value_t > tran_const

  [n_conns] array of temporary const transmissibilities
- uint8_t n_vars

  Number of unknowns per block.
- std::vector< index_t > fstencil

  array of indices of blocks are neccessary for each connection
- std::vector< index_t > fst_offset

  [n_conns + 1] array of offsets of the first block of connection in 'stencil'
- std::vector< index_t > sstencil

  array of indices of blocks are neccessary for each connection
- std::vector< index_t > sst_offset

  [n_conns + 1] array of offsets of the first block of connection in 'stencil'
- std::vector< value_t > stran

  [n_conns] array of transissibility values for given connection
- std::vector< value_t > stran_biot

  [n_conns] array of transissibility values for biot contribution to the given connection
- std::vector< std::vector< int > > **stencil**
- std::vector< value_t > **rhs**
- std::vector< value_t > **rhs_biot**
- std::vector< value_t > **f**

- index_t n_links

  number of non-zero links
- std::vector< value_t > volume

  [n_blocks] array of volumes of mesh blocks
- std::vector< value_t > poro

  [n_blocks] array of porosities of mesh blocks
- std::vector< value_t > depth

  [n_blocks] array of depths
- std::vector< value_t > heat_capacity

  [n_blocks] array of heat capacity of rock
- std::vector< value_t > rock_cond

  [n_blocks] array of heat conduction of rock;
- std::vector< value_t > kin_factor

  [n_blocks] array of kinetic rate constants (dependent on the initial porosity and other factors!);
- std::vector< value_t > pressure

  [n_blocks] array of initial pressure values
- std::vector< value_t > composition

  [n_blocks] array of initial composition values
- std::vector< value_t > temperature

  [n_blocks] array of initial temperature values
- std::vector< value_t > enthalpy

  [n_blocks] array of initial enthalpy values
- std::vector< value_t > displacement

  *[n_dim * n_blocks] array of initial displacements values*
- std::vector< value_t > bc

  *[(1 + n_dim) * n_bounds] array of boundary conditions*
- std::vector< value_t > bc_prev

  *[(1 + n_dim) * n_bounds] array of boundary conditions*
- std::vector< value_t > pz_bounds

  *[nc * n_bounds] array of pressures and (inflow) fractions at boundaries*
- std::vector< value_t > biot

  *[9 * n_blocks] array of biot coefficients of mesh blocks*
- std::vector< value_t > drained_compressibility

  [n_blocks] array of drained compressibility of mesh blocks
- std::vector< index_t > op_num

  [n_blocks] array of operator set index for every block

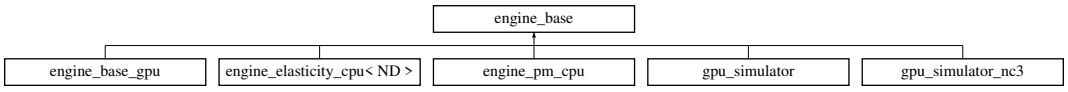
5.5.1 Detailed Description

This class defines mesh and corresponding arrays.

5.6 engine_base Class Reference

This class defines infrastructure for simulation.

Inheritance diagram for engine_base:



Public Member Functions

- virtual const uint8_t **get_n_vars** ()=0
- virtual const uint8_t **get_n_ops** ()=0
- virtual const uint8_t **get_n_comps** ()=0
- virtual const uint8_t **get_n_fl_var** ()
- virtual const uint8_t **get_z_var** ()=0
- virtual int **init** (conn_mesh *mesh_, std::vector< ms_well *> &well_list_, std::vector< operator_set_↩ gradient_evaluator_iface *> &acc_flux_op_set_list_, sim_params *params, timer_node *timer_)=0
- template<uint8_t N_VARS>
  int **init_base** (conn_mesh *mesh_, std::vector< ms_well *> &well_list_, std::vector< operator_set_↩ gradient_evaluator_iface *> &acc_flux_op_set_list_, sim_params *params, timer_node *timer_)
- virtual int **init_jacobian_structure** (csr_matrix_base *jacobian)
- virtual int **assemble_jacobian_array** (value_t dt, std::vector< value_t > &X, csr_matrix_base *jacobian, std::vector< value_t > &RHS, int assemble_kernel=0)=0
- virtual double **calc_newton_residual** ()
- virtual double **calc_newton_residual_L1** ()
- virtual double **calc_newton_residual_L2** ()
- virtual double **calc_newton_residual_Linf** ()
- virtual double **calc_well_residual** ()
- virtual double **calc_well_residual_L1** ()
- virtual double **calc_well_residual_L2** ()
- virtual double **calc_well_residual_Linf** ()
- virtual void **average_operator** (std::vector< value_t > &av_op)
- virtual void **apply_composition_correction** (std::vector< value_t > &X, std::vector< value_t > &dX)
- void **apply_global_chop_correction** (std::vector< value_t > &X, std::vector< value_t > &dX)
- virtual void **apply_local_chop_correction** (std::vector< value_t > &X, std::vector< value_t > &dX)
- void **apply_composition_correction_with_solid** (std::vector< value_t > &X, std::vector< value_t > &dX)
- void **apply_local_chop_correction_with_solid** (std::vector< value_t > &X, std::vector< value_t > &dX)
- void **apply_composition_correction_new** (std::vector< value_t > &X, std::vector< value_t > &dX)
- void **apply_global_chop_correction_new** (std::vector< value_t > &X, std::vector< value_t > &dX)
- void **apply_local_chop_correction_new** (std::vector< value_t > &X, std::vector< value_t > &dX)
- virtual int **apply_newton_update** (value_t dt)
- virtual void **apply_obl_axis_local_correction** (std::vector< value_t > &X, std::vector< value_t > &dX)
- double **evaluate_next_dt** ()
- virtual int **print_timestep** (value_t time, value_t deltat)
- int **print_header** ()
- virtual int run (value_t n_days)

  runs simulation for the number of days using internal time management
- virtual int run_timestep (value_t deltat, value_t time)

  runs simulation for one timestep starting from particular time
- virtual int run_single_newton_iteration (value_t deltat)

  report for one newton iteration
- virtual int **solve_linear_equation** ()
- virtual int **post_newtonloop** (value_t deltat, value_t time)
- virtual int report ()

  reports complete information about well regimes
- virtual int print_stat ()

  print statistics for the current run
- virtual int **test_assembly** (int n_times, int kernel_number=0, int dump_jacobian=0)
- virtual int **test_spmv** (int n_timer, int kernel_number=0, int dump_result=0)

Public Attributes

- std::vector< value_t > X

  vector of unknowns in the current timestep
- std::vector< value_t > Xn

  vector of unknowns in the previous timestep
- value_t t

  current timestep
- conn_mesh * mesh

  pointer to mesh
- sim_params * params

  simulation parameters
- sim_stat stat

  simulation statistics
- std::vector< ms_well * > wells

  vector of wells
- std::unordered_map< std::string, std::vector< value_t > > time_data

  unsorted map containing well information (BHP, rates)
- linsolv_iface * **linear_solver**
- std::vector< operator_set_gradient_evaluator_iface * > **acc_flux_op_set_list**
- uint8_t **n_vars**
- uint8_t **n_ops**
- uint8_t **nc**
- uint8_t **z_var**
- double **min_zc**
- double **max_zc**
- std::vector< value_t > **old_z**
- std::vector< value_t > **new_z**
- uint8_t **nc_fl**
- std::vector< value_t > **old_z_fl**
- std::vector< value_t > **new_z_fl**
- std::vector< value_t > **X_init**
- std::vector< value_t > **PV**
- std::vector< value_t > **RV**
- std::vector< std::vector< index_t > > **block_idxs**
- std::vector< std::vector< value_t > > **op_axis_min**
- std::vector< std::vector< value_t > > **op_axis_max**
- std::vector< value_t > **op_vals_arr**
- std::vector< value_t > **op_ders_arr**
- std::vector< value_t > **op_vals_arr_n**
- std::unordered_map< std::string, std::vector< value_t > > **time_data_report**
- std::vector< value_t > **FIPS**
- csr_matrix_base * **Jacobian**
- std::vector< value_t > **X0**
- std::vector< value_t > **RHS**
- std::vector< value_t > **dX**
- value_t **dt**
- value_t **prev_usual_dt**
- value_t **stop_time**
- value_t **CFL_max**
- index_t **n_newton_last_dt**
- index_t **n_linear_last_dt**
- double **newton_residual_last_dt**

- double **well_residual_last_dt**
- int **linear_solver_error_last_dt**
- timer_node * **timer**
- timer_node **full_step_timer**
- double **full_step_run_timer**
- double **t_full_step**

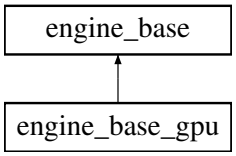
5.6.1 Detailed Description

This class defines infrastructure for simulation.

5.7 engine_base_gpu Class Reference

This class defines infrastructure for simulation.

Inheritance diagram for engine_base_gpu:



Public Member Functions

- virtual uint8_t const **get_n_vars** ()=0
- virtual uint8_t const **get_n_ops** ()=0
- virtual uint8_t const **get_n_comps** ()=0
- virtual uint8_t const **get_z_var** ()=0
- virtual int **init** (conn_mesh *mesh_, std::vector< ms_well *> &well_list_, std::vector< operator_set_↩ gradient_evaluator_iface *> &acc_flux_op_set_list_, sim_params *params, timer_node *timer_)=0
- template<uint8_t N_VARS>
  int **init_base** (conn_mesh *mesh_, std::vector< ms_well *> &well_list_, std::vector< operator_set_↩ gradient_evaluator_iface *> &acc_flux_op_set_list_, sim_params *params, timer_node *timer_)
- virtual int **assemble_jacobian_array** (value_t dt, std::vector< value_t > &X, csr_matrix_base *jacobian, std::vector< value_t > &RHS, int assemble_kernel=0)=0
- void **apply_composition_correction** (std::vector< value_t > &X, std::vector< value_t > &dX)
- void **apply_global_chop_correction** (std::vector< value_t > &X, std::vector< value_t > &dX)
- void **apply_local_chop_correction** (std::vector< value_t > &X, std::vector< value_t > &dX)
- int **apply_newton_update** (value_t dt)
- virtual int run_timestep (value_t deltat, value_t time)

  runs simulation for one timestep starting from particular time
- virtual int **test_assembly** (int n_times, int kernel_number=0, int dump_jacobian_rhs=0)
- virtual int **test_spmv** (int n_times, int kernel_number=0, int dump_result=0)

Public Attributes

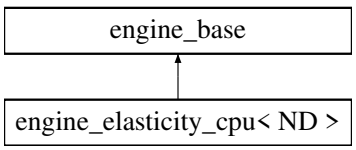
- value_t * **X_d**
- value_t * **Xn_d**
- value_t * **dX_d**
- value_t * **RHS_d**
- value_t * **RHS_wells_d**
- std::vector< value_t > **jac_wells**
- value_t * **jac_wells_d**
- std::vector< index_t > **jac_well_head_idxs**
- index_t * **jac_well_head_idxs_d**
- value_t * **op_vals_arr_d**
- value_t * **op_ders_arr_d**
- value_t * **op_vals_arr_n_d**
- std::vector< index_t * > **block_idxs_d**
- value_t * **RV_d**
- value_t * **PV_d**
- value_t * **mesh_tran_d**
- value_t * **mesh_tranD_d**
- value_t * **mesh_hcap_d**
- int **min_grid_size**
- int **grid_size**
- std::vector< int > **assembly_block_sizes**

5.7.1 Detailed Description

This class defines infrastructure for simulation.

5.8 engine_elasticity_cpu< ND > Class Template Reference

Inheritance diagram for engine_elasticity_cpu< ND >:



Public Member Functions

- const uint8_t **get_n_vars** () override
- const uint8_t **get_n_ops** ()
- const uint8_t **get_n_dim** ()
- const uint8_t **get_n_comps** ()
- const uint8_t **get_z_var** ()
- int **init** (conn_mesh *mesh_, std::vector< ms_well *> &well_list_, std::vector< operator_set_gradient_↩ evaluator_iface *> &acc_flux_op_set_list_, sim_params *params_, timer_node *timer_)
- int **init_base** (conn_mesh *mesh_, std::vector< ms_well *> &well_list_, std::vector< operator_set_↩ gradient_evaluator_iface *> &acc_flux_op_set_list_, sim_params *params_, timer_node *timer_)
- int **init_jacobian_structure_mpsa** (csr_matrix_base *jacobian)
- int **assemble_jacobian_array** (value_t dt, std::vector< value_t > &X, csr_matrix_base *jacobian, std↩ ::vector< value_t > &RHS, int assemble_kernel=0)

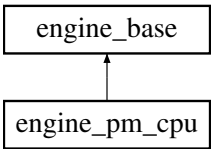
Static Public Attributes

- static const uint8_t **ND_** = ND
- static const uint8_t **N_VARS** = ND
- static const uint8_t **U_VAR** = 0
- static const uint8_t **N_OPS** = ND_
- static const uint8_t **ACC_OP** = 0
- static const uint8_t **FLUX_OP** = 0
- static const uint8_t **N_VARS_SQ** = N_VARS ∗ N_VARS

Additional Inherited Members

5.9 engine_pm_cpu Class Reference

Inheritance diagram for engine_pm_cpu:



Public Member Functions

- const uint8_t **get_n_vars** () override
- const uint8_t **get_n_ops** ()
- const uint8_t **get_n_dim** ()
- const uint8_t **get_n_comps** ()
- const uint8_t **get_z_var** ()
- int **init** (conn_mesh ∗mesh_, std::vector< ms_well ∗> &well_list_, std::vector< operator_set_gradient_↩ evaluator_iface ∗> &acc_flux_op_set_list_, sim_params ∗params_, timer_node ∗timer_)
- int **init_base** (conn_mesh ∗mesh_, std::vector< ms_well ∗> &well_list_, std::vector< operator_set_↩ gradient_evaluator_iface ∗> &acc_flux_op_set_list_, sim_params ∗params_, timer_node ∗timer_)
- int **init_jacobian_structure_pm** (csr_matrix_base ∗jacobian)
- int **assemble_jacobian_array** (value_t dt, std::vector< value_t > &X, csr_matrix_base ∗jacobian, std↩ ::vector< value_t > &RHS, int assemble_kernel=0)
- int **solve_linear_equation** ()
- void **apply_obl_axis_local_correction** (std::vector< value_t > &X, std::vector< value_t > &dX)
- void **extract_Xop** ()
- std::vector< value_t > **calc_newton_dev_L2** ()
- std::vector< value_t > **calc_newton_dev** ()

Public Attributes

- std::vector< value_t > Xop

  vector of unknowns in the current timestep provided for operator evaluation
- value_t **deviation_u**
- value_t **deviation_p**

Static Public Attributes

- static const uint8_t **ND_** = 3
- static const uint8_t **NC_** = 1
- static const uint8_t **NT_** = 4
- static const uint8_t **N_VARS** = NC_ + ND_
- static const uint8_t **U_VAR** = 0
- static const uint8_t **P_VAR** = 3
- static const uint8_t **Z_VAR** = 255
- static const uint8_t **N_OPS** = 2 ∗ NC_
- static const uint8_t **ACC_OP** = 0
- static const uint8_t **FLUX_OP** = NC_
- static const uint8_t **GRAV_OP** = 0
- static const uint8_t **N_VARS_SQ** = N_VARS ∗ N_VARS

5.10 pm::Face Class Reference

Public Member Functions

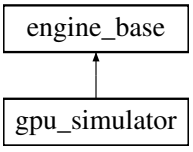
- **Face** (index_t _type, index_t _cell_id1, index_t _cell_id2, index_t _face_id1, index_t _face_id2, value_t _↩ area, std::valarray< value_t > &_n, std::valarray< value_t > &_c)

Public Attributes

- index_t **type**
- index_t **cell_id1**
- index_t **cell_id2**
- index_t **face_id1**
- index_t **face_id2**
- Matrix **n**
- Matrix **c**
- value_t **area**

5.11 gpu_simulator Class Reference

Inheritance diagram for gpu_simulator:



Public Member Functions

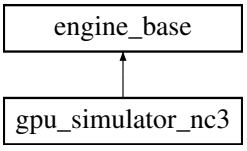
- int **init** (conn_mesh ∗_mesh, std::string table_base_name)
- int **run** (sim_params ∗params)
- int **assemble_jacobian** (value_t dt, int is_first)
- int **gpu_test** (int argc, char ∗∗argv)

Public Attributes

- conn_mesh * **mesh**
- sim_params **params**
- linear_solver_base * **cpu_solver**
- linear_solver_base * **cpu_preconditioner**
- LinearSolver * **gpu_solver**
- Preconditioner * **gpu_preconditioner**
- interp_table * **acc1**
- interp_table * **acc2**
- interp_table * **flu1**
- interp_table * **flu2**
- csr_matrix **Jacobian**
- std::vector< value_t > **RHS**
- std::vector< value_t > **X**
- std::vector< value_t > **Xn**
- std::vector< value_t > **dX**
- float **gpu_assemble_timer**
- value_t * **gpu_acc1_data**
- interp_value_t * **gpu_acc1_res**
- value_t * **gpu_acc2_data**
- interp_value_t * **gpu_acc2_res**
- value_t * **gpu_flu1_data**
- interp_value_t * **gpu_flu1_res**
- value_t * **gpu_flu2_data**
- interp_value_t * **gpu_flu2_res**
- index_t * **gpu_block_m**
- index_t * **gpu_block_p**
- value_t * **gpu_tran**
- value_t * **gpu_PV**
- value_t * **gpu_jac_values**
- index_t * **gpu_jac_rows_ptr**
- index_t * **gpu_jac_cols_ind**
- value_t * **gpu_jac_values_ilu**
- value_t * **gpu_rhs**
- value_t * **gpu_x**
- value_t * **gpu_xn**
- value_t * **gpu_dx**

5.12 gpu_simulator_nc3 Class Reference

Inheritance diagram for gpu_simulator_nc3:



Public Member Functions

- int **init** (conn_mesh *_mesh, std::string table_base_name)
- int **run** (sim_params *params)
- int **assemble_jacobian** (value_t dt, int is_first)
- int **gpu_test** (int argc, char **argv)

Public Attributes

- conn_mesh * **mesh**
- sim_params **params**
- linear_solver_base * **cpu_solver**
- linear_solver_base * **cpu_preconditioner**
- LinearSolver * **gpu_solver**
- Preconditioner * **gpu_preconditioner**
- interp_table_3d * **acc1**
- interp_table_3d * **acc2**
- interp_table_3d * **flu1**
- interp_table_3d * **flu2**
- interp_table_3d * **acc3**
- interp_table_3d * **flu3**
- csr_matrix **Jacobian**
- std::vector< value_t > **RHS**
- std::vector< value_t > **X**
- std::vector< value_t > **Xn**
- std::vector< value_t > **dX**
- double **interpolation_timer**
- value_t * **gpu_acc1_data**
- interp_value_t * **gpu_acc1_res**
- value_t * **gpu_acc2_data**
- interp_value_t * **gpu_acc2_res**
- value_t * **gpu_acc3_data**
- interp_value_t * **gpu_acc3_res**
- interp_value_t * **gpu_acc_n_res**
- value_t * **gpu_flu1_data**
- interp_value_t * **gpu_flu1_res**
- value_t * **gpu_flu2_data**
- interp_value_t * **gpu_flu2_res**
- value_t * **gpu_flu3_data**
- interp_value_t * **gpu_flu3_res**
- index_t * **gpu_block_m**
- index_t * **gpu_block_p**
- value_t * **gpu_tran**
- value_t * **gpu_PV**
- value_t * **gpu_jac_values**
- index_t * **gpu_jac_rows_ptr**
- index_t * **gpu_jac_cols_ind**
- value_t * **gpu_jac_values_ilu**
- value_t * **gpu_rhs**
- value_t * **gpu_x**
- value_t * **gpu_xn**
- value_t * **gpu_dx**
- float * **gpu_update_ratio**

5.13 pm::pm_discretizer::Gradients Struct Reference

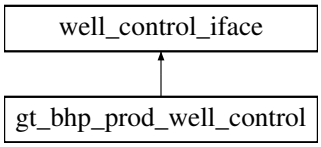
Public Attributes

- std::vector< index_t > **stencil**
- Matrix **mat**
- Matrix **rhs**

5.14 gt_bhp_prod_well_control Class Reference

BHP control for production geothermal well.

Inheritance diagram for gt_bhp_prod_well_control:



Public Member Functions

- **gt_bhp_prod_well_control** (value_t target_pressure_)
- virtual int **add_to_jacobian** (value_t dt, index_t well_head_idx, value_t segment_trans, index_t n_block_size, std::vector< value_t > &X, value_t *jacobian_row, std::vector< value_t > &RHS)
- virtual int **check_constraint_violation** (value_t dt, index_t well_head_idx, value_t segment_trans, index_t n_block_size, std::vector< value_t > &X)
- virtual int **initialize_well_block** (std::vector< value_t > &state_block, const std::vector< value_t > &state↩ _neighbour)

Public Attributes

- value_t **target_pressure**

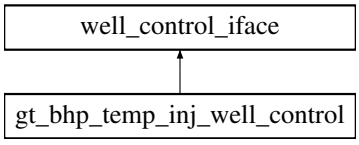
5.14.1 Detailed Description

BHP control for production geothermal well.

5.15 gt_bhp_temp_inj_well_control Class Reference

BHP and temperature control for injection geothermal well.

Inheritance diagram for gt_bhp_temp_inj_well_control:



Public Member Functions

- **gt_bhp_temp_inj_well_control** (std::vector< std::string > phase_names_, index_t n_vars_, value_t target_pressure_, value_t target_temperature_, std::vector< value_t > injection_stream_, operator_set_gradient_evaluator_iface *rate_etor_)
- virtual int **add_to_jacobian** (value_t dt, index_t well_head_idx, value_t segment_trans, index_t n_block_size, std::vector< value_t > &X, value_t *jacobian_row, std::vector< value_t > &RHS)
- virtual int **check_constraint_violation** (value_t dt, index_t well_head_idx, value_t segment_trans, index_t n_block_size, std::vector< value_t > &X)
- virtual int **initialize_well_block** (std::vector< value_t > &state_block, const std::vector< value_t > &state_neighbour)

Public Attributes

- std::vector< std::string > **phase_names**
- index_t **n_vars**
- index_t **n_phases**
- value_t **target_pressure**
- value_t **target_temperature**
- std::vector< value_t > **injection_stream**
- std::vector< value_t > **state**
- std::vector< value_t > **rate_temp_ops**
- std::vector< value_t > **rate_temp_ops_derivs**
- operator_set_gradient_evaluator_iface * **rate_etor**

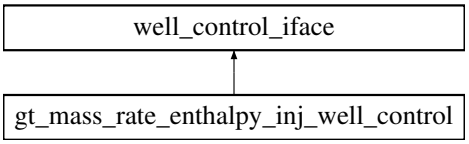
5.15.1 Detailed Description

BHP and temperature control for injection geothermal well.

5.16 gt_mass_rate_enthalpy_inj_well_control Class Reference

Mass rate and enthalpy control for injection geothermal well.

Inheritance diagram for gt_mass_rate_enthalpy_inj_well_control:



Public Member Functions

- **gt_mass_rate_enthalpy_inj_well_control** (std::vector< std::string > phase_names_, index_t target_phase_idx_, index_t n_variables_, std::vector< value_t >injection_stream_, value_t target_rate_, value_t target_enthalpy_, operator_set_gradient_evaluator_iface *rate_etor_)
- virtual int **add_to_jacobian** (value_t dt, index_t well_head_idx, value_t segment_trans, index_t n_block_size, std::vector< value_t > &X, value_t *jacobian_row, std::vector< value_t > &RHS)
- virtual int **check_constraint_violation** (value_t dt, index_t well_head_idx, value_t segment_trans, index_t n_block_size, std::vector< value_t > &X)
- virtual int **initialize_well_block** (std::vector< value_t > &state_block, const std::vector< value_t > &state_neighbour)

Public Attributes

- std::vector< std::string > **phase_names**
- std::vector< value_t > **state**
- std::vector< value_t > **injection_stream**
- std::vector< value_t > **rate_temp_ops**
- std::vector< value_t > **rate_temp_ops_derivs**
- index_t **target_phase_idx**
- index_t **n_variables**
- index_t **n_phases**
- value_t **target_rate**
- value_t **target_enthalpy**
- operator_set_gradient_evaluator_iface * **rate_etor**

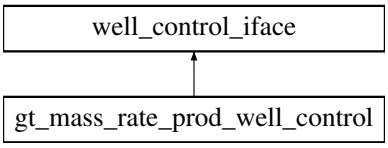
5.16.1 Detailed Description

Mass rate and enthalpy control for injection geothermal well.

5.17 gt_mass_rate_prod_well_control Class Reference

Mass rate control for production geothermal well.

Inheritance diagram for gt_mass_rate_prod_well_control:



Public Member Functions

- **gt_mass_rate_prod_well_control** (std::vector< std::string > phase_names_, index_t target_phase_idx_, index_t n_variables_, value_t target_rate_, operator_set_gradient_evaluator_iface *rate_etor_)
- virtual int **add_to_jacobian** (value_t dt, index_t well_head_idx, value_t segment_trans, index_t n_block_size, std::vector< value_t > &X, value_t *jacobian_row, std::vector< value_t > &RHS)
- virtual int **check_constraint_violation** (value_t dt, index_t well_head_idx, value_t segment_trans, index_t n_block_size, std::vector< value_t > &X)
- virtual int **initialize_well_block** (std::vector< value_t > &state_block, const std::vector< value_t > &state↩ _neighbour)

Public Attributes

- std::vector< std::string > **phase_names**
- std::vector< value_t > **state**
- std::vector< value_t > **rate_temp_ops**
- std::vector< value_t > **rate_temp_ops_derivs**
- index_t **target_phase_idx**
- index_t **n_variables**
- index_t **n_phases**
- value_t **target_rate**
- operator_set_gradient_evaluator_iface * **rate_etor**

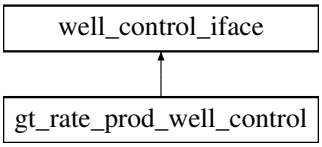
5.17.1 Detailed Description

Mass rate control for production geothermal well.

5.18 gt_rate_prod_well_control Class Reference

Volumetric rate control for production geothermal well.

Inheritance diagram for gt_rate_prod_well_control:



Public Member Functions

- **gt_rate_prod_well_control** (std::vector< std::string > phase_names_, index_t target_phase_idx_, index_t n_vars, value_t target_rate_, operator_set_gradient_evaluator_iface *rate_etor_)
- virtual int **add_to_jacobian** (value_t dt, index_t well_head_idx, value_t segment_trans, index_t n_block_size, std::vector< value_t > &X, value_t *jacobian_row, std::vector< value_t > &RHS)
- virtual int **check_constraint_violation** (value_t dt, index_t well_head_idx, value_t segment_trans, index_t n_block_size, std::vector< value_t > &X)
- virtual int **initialize_well_block** (std::vector< value_t > &state_block, const std::vector< value_t > &state_neighbour)

Public Attributes

- std::vector< std::string > **phase_names**
- std::vector< value_t > **state**
- std::vector< value_t > **rate_temp_ops**
- std::vector< value_t > **rate_temp_ops_derivs**
- index_t **target_phase_idx**
- index_t **n_variables**
- index_t **n_phases**
- value_t **target_rate**
- operator_set_gradient_evaluator_iface * **rate_etor**

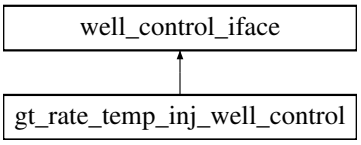
5.18.1 Detailed Description

Volumetric rate control for production geothermal well.

5.19 gt_rate_temp_inj_well_control Class Reference

Volumetric rate and temperature control for injection geothermal well.

Inheritance diagram for gt_rate_temp_inj_well_control:



Public Member Functions

- **gt_rate_temp_inj_well_control** (std::vector< std::string > phase_names_, index_t target_phase_idx_↩ , index_t n_variables_, value_t target_rate_, value_t target_temp_, std::vector< value_t > &injection_↩ stream_, operator_set_gradient_evaluator_iface *rate_etor_)
- virtual int **add_to_jacobian** (value_t dt, index_t well_head_idx, value_t segment_trans, index_t n_block_size, std::vector< value_t > &X, value_t *jacobian_row, std::vector< value_t > &RHS)
- virtual int **check_constraint_violation** (value_t dt, index_t well_head_idx, value_t segment_trans, index_t n_block_size, std::vector< value_t > &X)
- virtual int **initialize_well_block** (std::vector< value_t > &state_block, const std::vector< value_t > &state↩ _neighbour)

Public Attributes

- std::vector< std::string > **phase_names**
- index_t **target_phase_idx**
- index_t **n_equations**
- index_t **n_variables**
- size_t **n_phases**
- value_t **target_rate**
- value_t **target_temperature**
- std::vector< value_t > **injection_stream**
- std::vector< value_t > **state**
- std::vector< value_t > **rate_temp_ops**
- std::vector< value_t > **rate_temp_ops_derivs**
- operator_set_gradient_evaluator_iface * **rate_etor**

5.19.1 Detailed Description

Volumetric rate and temperature control for injection geothermal well.

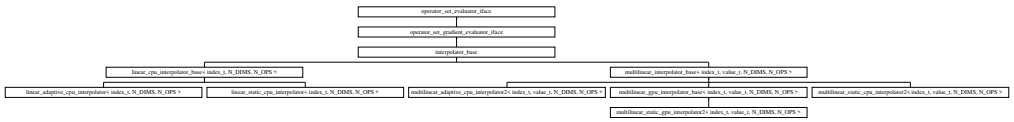
5.20 pm::pm_discretizer::InnerMatrices Struct Reference

Public Attributes

- Matrix **A1**
- Matrix **A2**
- Matrix **Q1**
- Matrix **Q2**
- Matrix **Th1**
- Matrix **Th2**
- Matrix **R1**
- Matrix **R2**
- Matrix **y1**
- Matrix **y2**
- value_t **r1**
- value_t **r2**

5.21 interpolator_base Class Reference

Inheritance diagram for interpolator_base:



Public Member Functions

- interpolator_base (operator_set_evaluator_iface *supporting_point_evaluator, const std::vector< int > &axes_points, const std::vector< double > &axes_min, const std::vector< double > &axes_max)

  Construct an interpolator with predefined parametrization space.
- virtual int init ()

  Initialize interpolator, perform internal sanity checks unavailable at construction time.
- int evaluate (const std::vector< double > &state, std::vector< double > &values)

  Evaluate all operators at the given state (point in parametrization space) Runs timer and calls virtual interpolate routine.
- int evaluate_with_derivatives (const std::vector< double > &states, const std::vector< int > &states_idxs, std::vector< double > &values, std::vector< double > &derivatives)

  Evaluate operators and their gradient for every specified state (point in parametrization space)
- virtual int interpolate (const std::vector< double > &point, std::vector< double > &values)=0

  Compute interpolation for all operators at the given point.
- virtual int interpolate_with_derivatives (const std::vector< double > &points, const std::vector< int > &points_idxs, std::vector< double > &values, std::vector< double > &derivatives)=0

  Compute interpolation and its gradient for all operators at every specified point.
- virtual int get_n_dims ()=0

  Get the number of dimensions in interpolation space Virtual, to be overriden by a child class.
- virtual int get_n_ops ()=0

  Get the number of operators Virtual, to be overriden by a child class.
- int get_axis_n_points (int axis)

Get the number of supporting points for the given axis.

- double get_axis_min (int axis)

  Get the parametrization minimum value for given axis.
- double get_axis_max (int axis)

  Get the parametrization maximum value for given axis.
- uint64_t get_n_interpolations ()

  Get the number of interpolations that took place.
- uint64_t get_n_points_total ()

  Get the total number of supporting points in parameter space.
- uint64_t get_n_points_used ()

  Get the number of supporting points used (evaluated through supporting_point_evaluator) The number is equal to n_points_total for static interpolation methods.

Protected Attributes

- const std::vector< int > axes_points

  number of supporting points along each axis
- const std::vector< double > axes_min

  minimum at each axis
- const std::vector< double > axes_max

  maximum of each axis
- operator_set_evaluator_iface * supporting_point_evaluator

  object which computes operator values for supporting points
- std::vector< double > axes_step

  the distance between neighbor supporting points for each axis
- std::vector< double > axes_step_inv

  inverse of step (to avoid division)
- uint64_t n_interpolations

  Number of interpolations that took place.
- uint64_t n_points_total

  Total number of parametrization points.
- double n_points_total_fp

  Total number of parametrization points in floating point format, to detect index overflow in derived classes.
- uint64_t n_points_used

  Number of parametrization points which were used (equal to n_points_total for static interpolators)
- std::vector< double > new_point_coords

  intermediate storage for supporting point generation
- std::vector< double > new_operator_values

  intermediate storage for supporting point generation

5.21.1 Detailed Description

Interpolator base class

5.21.2 Constructor & Destructor Documentation

5.21.2.1 interpolator_base()

```
interpolator_base::interpolator_base (
            operator_set_evaluator_iface * supporting_point_evaluator,
            const std::vector< int > & axes_points,
            const std::vector< double > & axes_min,
            const std::vector< double > & axes_max )
```

Construct an interpolator with predefined parametrization space.

Parameters

in	*supporting_point_evaluator*	Object used to compute operator values at supporting points
in	*axes_points*	Number of supporting points (minimum 2) along axes
in	*axes_min*	Minimum value for each axis
in	*axes_max*	Maximum for each axis

5.21.3 Member Function Documentation

5.21.3.1 evaluate()

```
int interpolator_base::evaluate (
            const std::vector< double > & state,
            std::vector< double > & values )
```

Evaluate all operators at the given state (point in parametrization space) Runs timer and calls virtual interpolate routine.

Parameters

in	*state*	Coordinates in parametrization space
out	*values*	Interpolated values

Returns

0 if evaluation is successful

5.21.3.2 evaluate_with_derivatives()

```
int interpolator_base::evaluate_with_derivatives (
            const std::vector< double > & states,
            const std::vector< int > & states_idxs,
            std::vector< double > & values,
            std::vector< double > & derivatives )
```

Evaluate operators and their gradient for every specified state (point in parametrization space)

Parameters

in	*states*	Array of coordinates in parametrization space
in	*states_idxs*	Indexes of states in the input array which are marked for evaluation
out	*values*	Interpolated values
out	*derivatives*	Interpolation gradients

Returns

0 if evaluation is successful

5.21.3.3 get_axis_max()

```
value_t interpolator_base::get_axis_max (
            int axis )
```

Get the parametrization maximum value for given axis.

Parameters

axis	index of axis in question

5.21.3.4 get_axis_min()

```
value_t interpolator_base::get_axis_min (
            int axis )
```

Get the parametrization minimum value for given axis.

Parameters

axis	index of axis in question

5.21.3.5 get_axis_n_points()

```
int interpolator_base::get_axis_n_points (
            int axis )
```

Get the number of supporting points for the given axis.

Parameters

axis	index of axis in question

5.21.3.6 get_n_points_total()

```
uint64_t interpolator_base::get_n_points_total ( )
```

Get the total number of supporting points in parameter space.

Returns

the total number of supporting points

5.21.3.7 get_n_points_used()

```
uint64_t interpolator_base::get_n_points_used ( )
```

Get the number of supporting points used (evaluated through supporting_point_evaluator) The number is equal to n_points_total for static interpolation methods.

Returns

the number of supporting points used

5.21.3.8 init()

```
int interpolator_base::init ( )  [virtual]
```

Initialize interpolator, perform internal sanity checks unavailable at construction time.

Returns

int 0 if successful

Reimplemented in multilinear_static_cpu_interpolator2< index_t, value_t, N_DIMS, N_OPS >, multilinear_static_gpu_interpolator2< index_t, value_t, N_DIMS, N_OPS >, and linear_static_cpu_interpolator< index_t, N_DIMS, N_OPS >.

5.21.3.9 interpolate()

```
virtual int interpolator_base::interpolate (
            const std::vector< double > & point,
            std::vector< double > & values )  [pure virtual]
```

Compute interpolation for all operators at the given point.

Parameters

in	*point*	Coordinates in parametrization space
out	*values*	Interpolated values

Returns

0 if interpolation is successful

Implemented in multilinear_interpolator_base< index_t, value_t, N_DIMS, N_OPS >, and linear_cpu_interpolator_base< index_t, N_DIMS, N_OPS >.

5.21.3.10 interpolate_with_derivatives()

```
virtual int interpolator_base::interpolate_with_derivatives (
            const std::vector< double > & points,
            const std::vector< int > & points_idxs,
            std::vector< double > & values,
            std::vector< double > & derivatives )  [pure virtual]
```

Compute interpolation and its gradient for all operators at every specified point.

Parameters

in	*points*	Array of coordinates in parametrization space
in	*points_idxs*	Indexes of points in the points array which are marked for interpolation
out	*values*	Interpolated values
out	*derivatives*	Interpolation gradients

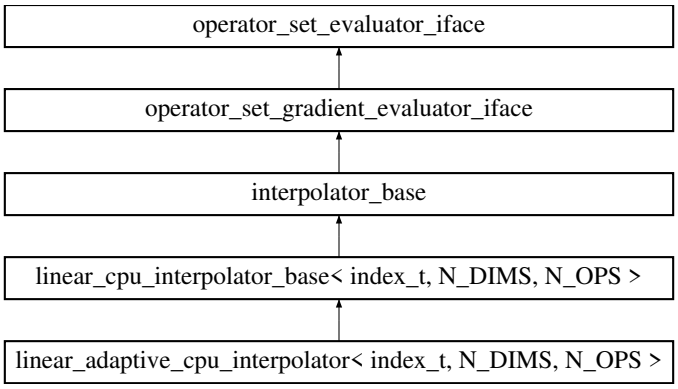
Returns

0 if interpolation is successful

Implemented in multilinear_interpolator_base< index_t, value_t, N_DIMS, N_OPS >, multilinear_adaptive_cpu_interpolator2< index_t, value_t, N_DIMS, N_OPS >, and linear_cpu_interpolator_base< index_t, N_DIMS, N_OPS >.

5.22 linear_adaptive_cpu_interpolator< index_t, N_DIMS, N_OPS > Class Template Reference

Inheritance diagram for linear_adaptive_cpu_interpolator< index_t, N_DIMS, N_OPS >:



Public Member Functions

- **linear_adaptive_cpu_interpolator** (operator_set_evaluator_iface *base_points_generator, const std::vector< int > &axesPoints, const std::vector< double > &axesMin, const std::vector< double > &axesMax)

Public Attributes

- std::unordered_map< index_t, std::array< double, N_OPS > > point_data

  adaptive storage: the values of operators at supporting points actually required

Additional Inherited Members

5.22.1 Detailed Description

template<typename index_t, int N_DIMS, int N_OPS>
class linear_adaptive_cpu_interpolator< index_t, N_DIMS, N_OPS >

Adaptive piecewise linear interpolator

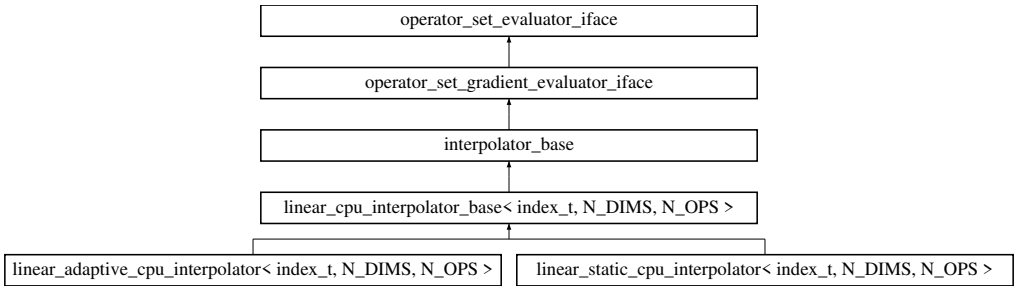
Template Parameters

index_t	index type used for supporting point indexing
N_DIMS	The number of dimensions in paramter space
N_OPS	The number of operators to be interpolated

5.23 linear_cpu_interpolator_base< index_t, N_DIMS, N_OPS > Class Template Reference

Interpolator base for static/adaptive piecewise linear interpolator.

Inheritance diagram for linear_cpu_interpolator_base< index_t, N_DIMS, N_OPS >:



Public Member Functions

- linear_cpu_interpolator_base (operator_set_evaluator_iface *supporting_point_evaluator, const std::vector< int > &axes_points, const std::vector< double > &axes_min, const std::vector< double > &axes_max)

  Construct an interpolator with specified parametrization space.
- int get_n_dims ()

  Get the number of dimensions in interpolation space.
- int get_n_ops ()

  Get the number of operators to be interpolated.
- int interpolate (const std::vector< value_t > &point, std::vector< value_t > &values) override

  Compute interpolation for all operators at the given point.
- int interpolate_with_derivatives (const std::vector< double > &points, const std::vector< int > &points_idxs, std::vector< double > &interp_values, std::vector< double > &derivatives) override

  Compute interpolation and its gradient for all operators at every specified point.

Protected Member Functions

- void find_hypercube (const std::vector< double > &points, std::array< int, N_DIMS > &hypercube, std::array< double, N_DIMS > &scaled_point, const int point_index=0)

  Given the coordinate of a point, the function computes the hypercube where the point is located and its scaled coordinate inside the hypercube.
- void find_simplex (const std::array< int, N_DIMS > &hypercube, const std::array< double, N_DIMS > &scaled_point, std::array< int, N_DIMS > &tri_order, std::array< std::array< int, N_DIMS >, N_DIMS+1 > &simplex)

  Compute which simplex the given point is located in using standard triangulation.
- virtual void get_supporting_point (const std::array< int, N_DIMS > &vertex, std::array< double, N_OPS > &values)=0

  Get values of operators at the given supporting point Implementation depends on underlying storage. If static storage is used, the function simply reads operator values of the given supporting point from the the storage. If adaptive storage is used, the function checks whether the values were computed before, if yes, the value is directly returned; if not, the function computes the values, stores them and then returns.
- index_t get_index_from_vertex (const std::array< int, N_DIMS > &vertex)

  Given a supporting point, compute its index.
- void get_point_from_vertex (const std::array< int, N_DIMS > &vertex, std::vector< double > &point)

  Transfer a vertex to its coordinates.

Protected Attributes

- std::array< std::array< int, N_DIMS >, N_DIMS+1 > standard_simplex

  a standard simplex
- std::array< index_t, N_DIMS > **axes_mult**
- int transform_last_axis

  multiplication factor used for transferring supporting point to point index

5.23.1 Detailed Description

template<typename index_t, int N_DIMS, int N_OPS>
class linear_cpu_interpolator_base< index_t, N_DIMS, N_OPS >

Interpolator base for static/adaptive piecewise linear interpolator.

Template Parameters

index_t	index type used for supporting point indexing
N_DIMS	The number of dimensions in paramter space
N_OPS	The number of operators to be interpolated

5.23.2 Constructor & Destructor Documentation

5.23.2.1 linear_cpu_interpolator_base()

```
template<typename index_t , int N_DIMS, int N_OPS>
linear_cpu_interpolator_base< index_t, N_DIMS, N_OPS >::linear_cpu_interpolator_base (
            operator_set_evaluator_iface * supporting_point_evaluator,
            const std::vector< int > & axes_points,
            const std::vector< double > & axes_min,
            const std::vector< double > & axes_max )
```

Construct an interpolator with specified parametrization space.

Parameters

in	*supporting_point_evaluator*	Object used to compute operators values at supporting points
in	*axes_points*	Number of supporting points (minimum 2) along axes
in	*axes_min*	Minimum value for each axis
in	*axes_max*	Maximum for each axis

5.23.3 Member Function Documentation

5.23.3.1 find_hypercube()

```
template<typename index_t , int N_DIMS, int N_OPS>
void linear_cpu_interpolator_base< index_t, N_DIMS, N_OPS >::find_hypercube (
            const std::vector< double > & points,
```

```
            std::array< int, N_DIMS > & hypercube,
            std::array< double, N_DIMS > & scaled_point,
            const int point_index = 0 )  [protected]
```

Given the coordinate of a point, the function computes the hypercube where the point is located and its scaled coordinate inside the hypercube.

Parameters

in	*points*	The array of coordinates of points
out	*hypercube*	The lower-left vertex of the hypercube
out	*scaled_point*	The scaled coordinate of the given point inside the hypercube
in	*point_index*	Index of the point in the std::vector points The argument point_index is used only when std::vector points consists of multiple points.

5.23.3.2 find_simplex()

```
template<typename index_t , int N_DIMS, int N_OPS>
void linear_cpu_interpolator_base< index_t, N_DIMS, N_OPS >::find_simplex (
            const std::array< int, N_DIMS > & hypercube,
            const std::array< double, N_DIMS > & scaled_point,
            std::array< int, N_DIMS > & tri_order,
            std::array< std::array< int, N_DIMS >, N_DIMS+1 > & simplex )  [protected]
```

Compute which simplex the given point is located in using standard triangulation.

Parameters

in	*hypercube*	The lower-left vertex of the hypercube
in	*scaled_point*	The scaled coordinate of the given point inside the hypercube
out	*tri_order*	The order of the scaled coordinate which is used for 1. finding simplex for standard triangulation 2. computing weights of the barycentric interpolation
out	*simplex*	An array of vertices which forms simplex in N_DIMS-dimensional space

5.23.3.3 get_index_from_vertex()

```
template<typename index_t , int N_DIMS, int N_OPS>
index_t linear_cpu_interpolator_base< index_t, N_DIMS, N_OPS >::get_index_from_vertex (
            const std::array< int, N_DIMS > & vertex )  [protected]
```

Given a supporting point, compute its index.

This function is used as a hash for std::array<int, N_DIMS>.

Parameters

in	*vertex*	The indexes of coordinates the given supporting point along axes

Returns

The index of point among all supporting point

5.23.3.4 get_point_from_vertex()

```
template<typename index_t , int N_DIMS, int N_OPS>
void linear_cpu_interpolator_base< index_t, N_DIMS, N_OPS >::get_point_from_vertex (
            const std::array< int, N_DIMS > & vertex,
            std::vector< double > & point )  [protected]
```

Transfer a vertex to its coordinates.

Parameters

in	*vertex*	The indexes of the given supporting point along axes
out	*point*	The coordinates of the supporting point

5.23.3.5 get_supporting_point()

```
template<typename index_t , int N_DIMS, int N_OPS>
virtual void linear_cpu_interpolator_base< index_t, N_DIMS, N_OPS >::get_supporting_point (
            const std::array< int, N_DIMS > & vertex,
            std::array< double, N_OPS > & values )  [protected], [pure virtual]
```

Get values of operators at the given supporting point Implementation depends on underlying storage. If static storage is used, the function simply reads operator values of the given supporting point from the the storage. If adaptive storage is used, the function checks whether the values were computed before, if yes, the value is directly returned; if not, the function computes the values, stores them and then returns.

Parameters

in	*vertex*	The indexes of coordinates the given supporting point along axes
out	*values*	The operator values at the given point

5.23.3.6 interpolate()

```
template<typename index_t , int N_DIMS, int N_OPS>
int linear_cpu_interpolator_base< index_t, N_DIMS, N_OPS >::interpolate (
```

```
            const std::vector< value_t > & point,
            std::vector< value_t > & values )  [override], [virtual]
```

Compute interpolation for all operators at the given point.

Parameters

in	*point*	Coordinates in parametrization space
out	*values*	Interpolated values

Returns

0 if interpolation is successful

Implements interpolator_base.

5.23.3.7 interpolate_with_derivatives()

```
template<typename index_t , int N_DIMS, int N_OPS>
int linear_cpu_interpolator_base< index_t, N_DIMS, N_OPS >::interpolate_with_derivatives (
            const std::vector< double > & points,
            const std::vector< int > & points_idxs,
            std::vector< double > & interp_values,
            std::vector< double > & derivatives )  [override], [virtual]
```

Compute interpolation and its gradient for all operators at every specified point.

Parameters

in	*points*	Array of coordinates in parametrization space
in	*points_idxs*	Indexes of points in the points array which are marked for interpolation
out	*values*	Interpolated values
out	*derivatives*	Interpolation gradients

Returns

0 if interpolation is successful

Implements interpolator_base.

5.23.4 Member Data Documentation

5.23.4.1 transform_last_axis

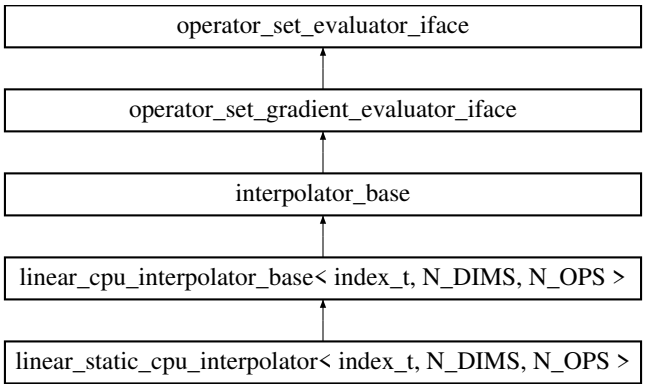
```
template<typename index_t , int N_DIMS, int N_OPS>
int linear_cpu_interpolator_base< index_t, N_DIMS, N_OPS >::transform_last_axis  [protected]
```

multiplication factor used for transferring supporting point to point index

apply transformation z'=1-z for the last axis

5.24 linear_static_cpu_interpolator< index_t, N_DIMS, N_OPS > Class Template Reference

Inheritance diagram for linear_static_cpu_interpolator< index_t, N_DIMS, N_OPS >:



Public Member Functions

- linear_static_cpu_interpolator (operator_set_evaluator_iface *supporting_point_evaluator, const std::vector< int > &axes_points, const std::vector< double > &axes_min, const std::vector< double > &axes_max)

  Construct the interpolator with specified parametrization space.
- int init () override

  Initialize the interpolator by computing all values of supporting points if the storage was not already initialized.

Public Attributes

- std::vector< double > point_data

  static storage: the values of operators at all supporting points

Additional Inherited Members

5.24.1 Detailed Description

template<typename index_t, int N_DIMS, int N_OPS>
class linear_static_cpu_interpolator< index_t, N_DIMS, N_OPS >

Static piecewise linear interpolator

Template Parameters

index_t	index type used for supporting point indexing
N_DIMS	The number of dimensions in paramter space
N_OPS	The number of operators to be interpolated

5.24.2 Constructor & Destructor Documentation

5.24.2.1 linear_static_cpu_interpolator()

```
template<typename index_t , int N_DIMS, int N_OPS>
linear_static_cpu_interpolator< index_t, N_DIMS, N_OPS >::linear_static_cpu_interpolator (
            operator_set_evaluator_iface * supporting_point_evaluator,
            const std::vector< int > & axes_points,
            const std::vector< double > & axes_min,
            const std::vector< double > & axes_max )
```

Construct the interpolator with specified parametrization space.

Parameters

in	*supporting_point_evaluator*	Object used to compute operators values at supporting points
in	*axes_points*	Number of supporting points (minimum 2) along axes
in	*axes_min*	Minimum value for each axis
in	*axes_max*	Maximum for each axis

5.24.3 Member Function Documentation

5.24.3.1 init()

```
template<typename index_t , int N_DIMS, int N_OPS>
int linear_static_cpu_interpolator< index_t, N_DIMS, N_OPS >::init ( )  [override], [virtual]
```

Initialize the interpolator by computing all values of supporting points if the storage was not already initialized.

Returns

int 0 if successful

Reimplemented from interpolator_base.

5.25 linalg::Matrix< T > Class Template Reference

Public Types

- typedef T **Type**

Public Member Functions

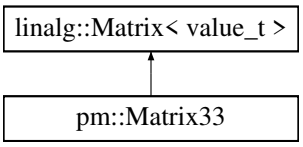
- **Matrix** (index_t _M, index_t _N)
- **Matrix** (const Matrix< T > &m)
- **Matrix** (const std::valarray< T > &v, const index_t _M, const index_t _N)
- index_t **getIndex** (index_t i, index_t j) const
- Matrix< T > **transpose** () const
- void **transposeInplace** ()
- bool **inv** ()
- bool **lu** (std::valarray< size_t > &ri, T *pDet)
- T **det** () const

Public Attributes

- int **M**
- int **N**
- std::valarray< T > **values**
- std::gslice **g**

5.26 pm::Matrix33 Class Reference

Inheritance diagram for pm::Matrix33:



Public Types

- typedef Matrix **Base**

Public Member Functions

- **Matrix33** (value_t kx)
- **Matrix33** (value_t kx, value_t ky, value_t kz)
- **Matrix33** (std::valarray< value_t > _m)

Static Public Attributes

- static const index_t **N** = ND ∗ ND

Additional Inherited Members

5.27 pm::mech_operators Class Reference

Public Member Functions

- ∼**mech_operators** ()
- void **init** (conn_mesh ∗_mesh, pm_discretizer ∗_discr, uint8_t _P_VAR, uint8_t _Z_VAR, uint8_t _U_VAR, uint8_t _N_VARS, uint8_t _N_OPS, uint8_t _NC, uint8_t _ACC_OP, uint8_t _FLUX_OP, uint8_t _GRAV_OP)
- std::vector< value_t > **eval_flux** (int cell_id, int face_id, const std::vector< value_t > &X)
- void **eval_fluxes** (const std::vector< value_t > &X, const std::vector< value_t > &bc_rhs, const std::vector< value_t > &op_vals_arr)
- void **eval_stresses** (const std::vector< value_t > &X, const std::vector< value_t > &bc_rhs, const std↩ ::vector< value_t > &op_vals_arr)
- void **eval_porosities** (const std::vector< value_t > &X, const std::vector< value_t > &bc_rhs)

Public Attributes

- std::vector< std::vector< std::vector< value_t > > > **fluxes**
- std::vector< std::vector< value_t > > **pressures**
- std::vector< std::array< value_t, 6 > > **stresses**
- std::vector< std::array< value_t, 6 > > **total_stresses**
- std::vector< value_t > **eps_vol**
- std::vector< value_t > **porosities**

Protected Attributes

- conn_mesh ∗ **mesh**
- pm_discretizer ∗ **discr**
- uint8_t **P_VAR**
- uint8_t **Z_VAR**
- uint8_t **U_VAR**
- uint8_t **N_VARS**
- uint8_t **N_OPS**
- uint8_t **NC**
- uint8_t **ACC_OP**
- uint8_t **FLUX_OP**
- uint8_t **GRAV_OP**
- std::map< uint8_t, Matrix > **pre_N**
- std::map< uint8_t, Matrix > **pre_R**
- std::map< uint8_t, Matrix > **pre_Ft**
- std::map< uint8_t, Matrix > **pre_F**

Static Protected Attributes

- static const uint8_t **ND** = 3
- static const uint8_t **NT** = 4
- static const uint8_t **N_TRANS_SQ** = NT ∗ NT

5.28 ms_well Class Reference

Base class for multi-segmented well.

Public Member Functions

- void **init_rate_parameters** (int n_vars_, std::vector< std::string > phase_names_, operator_set_gradient↩ _evaluator_iface ∗rate_evaluator_, int thermal_=0)
- int **add_to_jacobian** (double dt, std::vector< value_t > &X, value_t ∗jac_well_head, std::vector< value_t > &RHS)
- int **add_to_csr_jacobian** (double dt, std::vector< value_t > &X, value_t ∗jac_well_head, std::vector< value_t > &RHS)
- int **check_constraints** (double dt, std::vector< value_t > &X)
- int **calc_rates** (std::vector< value_t > &X, std::vector< value_t > &op_vals_arr, std::unordered_map< std↩ ::string, std::vector< value_t >> &time_data)
- int **calc_rates_velocity** (std::vector< value_t > &X, std::vector< value_t > &op_vals_arr, std::unordered↩ _map< std::string, std::vector< value_t >> &time_data, index_t n_blocks)
- int **initialize_control** (std::vector< value_t > &X)

Public Attributes

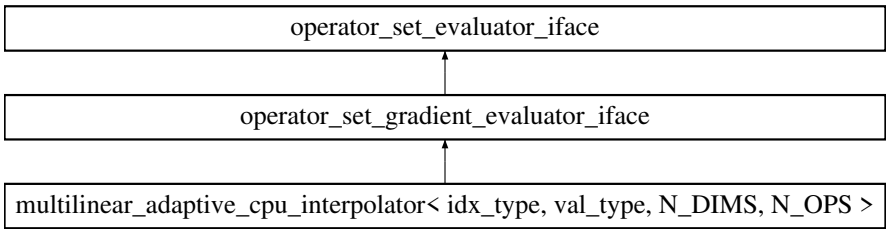
- std::vector< std::tuple< index_t, index_t, value_t > > **perforations**
- value_t **segment_volume**
- value_t **segment_transmissibility**
- value_t **well_head_depth**
- value_t **well_body_depth**
- value_t **segment_depth_increment**
- value_t **segment_diameter**
- value_t **segment_roughness**
- index_t **well_head_idx**
- index_t **well_body_idx**
- index_t **well_head_idx_conn**
- well_control_iface ∗ **control**
- well_control_iface ∗ **constraint**
- operator_set_evaluator_iface ∗ **rate_evaluator**
- std::string **name**
- std::vector< std::string > **phase_names**
- std::vector< value_t > **state**
- std::vector< value_t > **state_neighbour**
- std::vector< value_t > **rates**
- int **n_vars**
- int **n_segments**
- int **n_phases**
- int **thermal**

5.28.1 Detailed Description

Base class for multi-segmented well.

5.29 multilinear_adaptive_cpu_interpolator< idx_type, val_type, N_DIMS, N_OPS > Class Template Reference

Inheritance diagram for multilinear_adaptive_cpu_interpolator< idx_type, val_type, N_DIMS, N_OPS >:



Public Member Functions

- **multilinear_adaptive_cpu_interpolator** (operator_set_evaluator_iface *base_points_generator, std::vector< index_t > &axis_resolution, std::vector< value_t > &axis_min, std::vector< value_t > &axis_max)
- int **benchmark** (index_t n_points, index_t n_blocks, value_t param_space_frac, index_t n_unique_bodies)
- int **evaluate_with_derivatives** (const std::vector< value_t > &state, const std::vector< index_t > &block_idx, std::vector< value_t > &values, std::vector< value_t > &derivatives)
- int **evaluate** (const std::vector< value_t > &state, std::vector< value_t > &values)
- body_data_it_t **add_body_from_points** (const value_t *axis_values, idx_type body_idx)
- int **get_resolution** ()
- value_t **get_axis_min** (int axis)
- value_t **get_axis_max** (int axis)
- uint64_t **get_n_interpolations** ()
- uint64_t **get_n_points_used** ()
- uint64_t **get_n_points_total** ()
- uint64_t **get_n_hypercubes_used** ()
- uint64_t **get_n_hypercubes_total** ()
- double **get_interpolation_timer** ()
- double **get_generation_timer** ()
- double **get_point_generation_timer** ()
- int **clear_body_data** ()
- int **clear** ()

Public Attributes

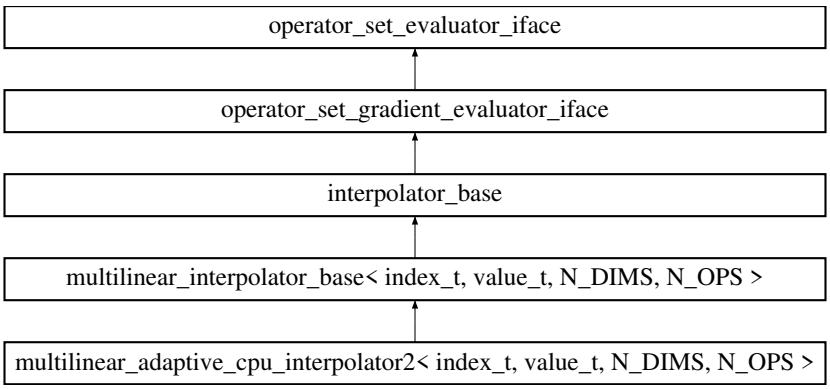
- std::array< unsigned int, N_DIMS > **axis_points**
- std::array< val_type, N_DIMS > **axis_min**
- std::array< val_type, N_DIMS > **axis_max**
- val_type **axis_step** [N_DIMS]
- val_type **axis_step_inv** [N_DIMS]
- idx_type **axis_mult** [N_DIMS]
- idx_type **axis_body_mult** [N_DIMS]

- operator_set_evaluator_iface * **base_points_generator**
- std::unordered_map< idx_type, std::vector< val_type > > **point_data**
- std::unordered_map< idx_type, std::vector< val_type > > **body_data**
- std::vector< std::vector< value_t > > **body_vertex_axes_values**
- std::vector< value_t > **new_ops_values**
- std::vector< val_type > **new_ops_values_interp**
- long long **n_interpolations**
- double **interpolation_timer**
- double **body_generation_timer**
- double **point_generation_timer**

5.30 multilinear_adaptive_cpu_interpolator2< index_t, value_t, N_DIMS, N_OPS > Class Template Reference

Piecewise mulitlinear interpolator with adaptive storage.

Inheritance diagram for multilinear_adaptive_cpu_interpolator2< index_t, value_t, N_DIMS, N_OPS >:



Public Member Functions

- multilinear_adaptive_cpu_interpolator2 (operator_set_evaluator_iface *supporting_point_evaluator, const std::vector< int > &axes_points, const std::vector< double > &axes_min, const std::vector< double > &axes_max)

  Construct the interpolator with specified parametrization space.

Public Attributes

- std::unordered_map< index_t, point_data_t > point_data

  adaptive point storage: the values of operators at requested supporting points Storage is grown dynamically in the process of simulation. Only supporting points that are required for interpolation are computed and added

Protected Member Functions

- const point_data_t & get_point_data (const index_t point_index)

  Get values of operators at a given point Provide a reference to correct location in the adaptive point storage. If the point is not found, compute it first, and then return the reference.
- const hypercube_data_t & get_hypercube_data (const index_t hypercube_index)

  Get values of operators at all vertices of the hypercube. Provide a reference to correct location in the adaptive hypercube storage. If the hypercube is not found, compute it first, and then return the reference.
- int interpolate_with_derivatives (const std::vector< double > &points, const std::vector< int > &points_idxs, std::vector< double > &values, std::vector< double > &derivatives) override

  Compute interpolation and its gradient for all operators at every specified point.

Protected Attributes

- std::unordered_map< index_t, hypercube_data_t > hypercube_data

  adaptive hypercube storage: the values of operators at every vertex of reqested hypercubes Storage is grown dynamically in the process of simulation Only hypercubes that are required for interpolation are computed and added

Additional Inherited Members

5.30.1 Detailed Description

template<typename index_t, typename value_t, uint8_t N_DIMS, uint8_t N_OPS>
class multilinear_adaptive_cpu_interpolator2< index_t, value_t, N_DIMS, N_OPS >

Piecewise mulitlinear interpolator with adaptive storage.

Template Parameters

index_t	type used for indexing of supporting points and hypercubes
value_t	value type used for supporting point storage, hypercube storage and interpolation
N_DIMS	The number of dimensions in paramter space
N_OPS	The number of operators to be interpolated

5.30.2 Constructor & Destructor Documentation

5.30.2.1 multilinear_adaptive_cpu_interpolator2()

```
template<typename index_t , typename value_t , uint8_t N_DIMS, uint8_t N_OPS>
multilinear_adaptive_cpu_interpolator2< index_t, value_t, N_DIMS, N_OPS >::multilinear_↩
adaptive_cpu_interpolator2 (
            operator_set_evaluator_iface * supporting_point_evaluator,
```

```
            const std::vector< int > & axes_points,
            const std::vector< double > & axes_min,
            const std::vector< double > & axes_max )
```

Construct the interpolator with specified parametrization space.

Parameters

in	*supporting_point_evaluator*	Object used to compute operators values at supporting points
in	*axes_points*	Number of supporting points (minimum 2) along axes
in	*axes_min*	Minimum value for each axis
in	*axes_max*	Maximum for each axis

5.30.3 Member Function Documentation

5.30.3.1 get_hypercube_data()

```
template<typename index_t , typename value_t , uint8_t N_DIMS, uint8_t N_OPS>
const hypercube_data_t& multilinear_adaptive_cpu_interpolator2< index_t, value_t, N_DIMS, N_↩
OPS >::get_hypercube_data (
            const index_t hypercube_index )  [protected], [virtual]
```

Get values of operators at all vertices of the hypercube. Provide a reference to correct location in the adaptive hypercube storage. If the hypercube is not found, compute it first, and then return the reference.

Parameters

in	*hypercube_index*	index of hypercube

Returns

operator values at all vertices of the hypercube

Implements multilinear_interpolator_base< index_t, value_t, N_DIMS, N_OPS >.

5.30.3.2 get_point_data()

```
template<typename index_t , typename value_t , uint8_t N_DIMS, uint8_t N_OPS>
const point_data_t& multilinear_adaptive_cpu_interpolator2< index_t, value_t, N_DIMS, N_OPS
>::get_point_data (
            const index_t point_index )  [protected]
```

Get values of operators at a given point Provide a reference to correct location in the adaptive point storage. If the point is not found, compute it first, and then return the reference.

Parameters

in	*point_index*	index of point

Returns

operator values at given point

5.30.3.3 interpolate_with_derivatives()

```
template<typename index_t , typename value_t , uint8_t N_DIMS, uint8_t N_OPS>
int multilinear_adaptive_cpu_interpolator2< index_t, value_t, N_DIMS, N_OPS >::interpolate_↩
with_derivatives (
            const std::vector< double > & points,
            const std::vector< int > & points_idxs,
            std::vector< double > & values,
            std::vector< double > & derivatives )  [override], [protected], [virtual]
```

Compute interpolation and its gradient for all operators at every specified point.

Parameters

in	*points*	Array of coordinates in parametrization space
in	*points_idxs*	Indexes of points in the points array which are marked for interpolation
out	*values*	Interpolated values
out	*derivatives*	Interpolation gradients

Returns

0 if interpolation is successful

Implements interpolator_base.

5.30.4 Member Data Documentation

5.30.4.1 hypercube_data

```
template<typename index_t , typename value_t , uint8_t N_DIMS, uint8_t N_OPS>
std::unordered_map<index_t, hypercube_data_t> multilinear_adaptive_cpu_interpolator2< index↩
_t, value_t, N_DIMS, N_OPS >::hypercube_data  [protected]
```

adaptive hypercube storage: the values of operators at every vertex of reqested hypercubes Storage is grown dynamically in the process of simulation Only hypercubes that are required for interpolation are computed and added

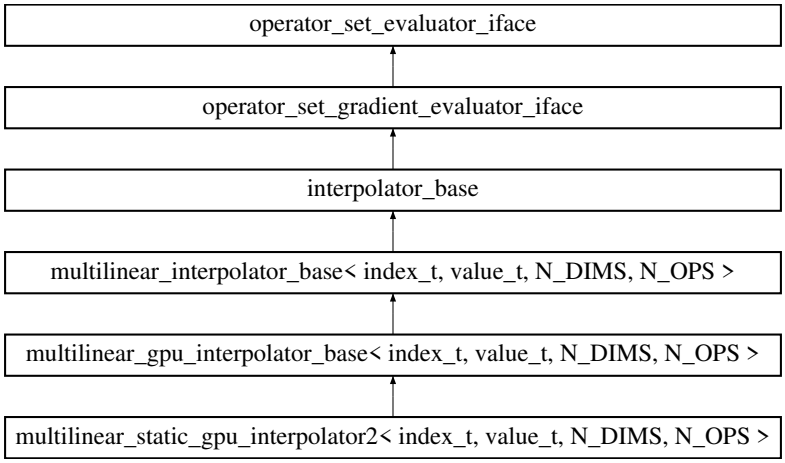
In fact it is an excess storage used to reduce memory accesses during interpolation. Here all values of all vertexes of requested hypercube are stored consecutevely and are accessed via a single index Usage of point_data for interpolation directly would require N_VERTS memory accesses (>1000 accesses for 10-dimensional space)

-

5.31 multilinear_gpu_interpolator_base< index_t, value_t, N_DIMS, N_OPS > Class Template Reference

Piecewise mulitlinear GPU interpolator base class.

Inheritance diagram for multilinear_gpu_interpolator_base< index_t, value_t, N_DIMS, N_OPS >:



Public Member Functions

- multilinear_gpu_interpolator_base (operator_set_evaluator_iface *supporting_point_evaluator, const std::vector< int > &axes_points, const std::vector< double > &axes_min, const std::vector< double > &axes_max)

  Construct the interpolator with specified parametrization space.

Protected Attributes

- const thrust::device_vector< value_t > axes_min_d

  minimum at each axis in value_t type

Additional Inherited Members

5.31.1 Detailed Description

template<typename index_t, typename value_t, uint8_t N_DIMS, uint8_t N_OPS>
class multilinear_gpu_interpolator_base< index_t, value_t, N_DIMS, N_OPS >

Piecewise mulitlinear GPU interpolator base class.

Introduces and initialize basic interpolation data on GPU device

Template Parameters

index_t	type used for indexing of supporting points and hypercubes
value_t	value type used for supporting point storage, hypercube storage and interpolation
N_DIMS	The number of dimensions in paramter space
N_OPS	The number of operators to be interpolated

5.31.2 Constructor & Destructor Documentation

5.31.2.1 multilinear_gpu_interpolator_base()

```
template<typename index_t , typename value_t , uint8_t N_DIMS, uint8_t N_OPS>
multilinear_gpu_interpolator_base< index_t, value_t, N_DIMS, N_OPS >::multilinear_gpu_interpolator_base (
            operator_set_evaluator_iface * supporting_point_evaluator,
            const std::vector< int > & axes_points,
            const std::vector< double > & axes_min,
            const std::vector< double > & axes_max )
```

Construct the interpolator with specified parametrization space.

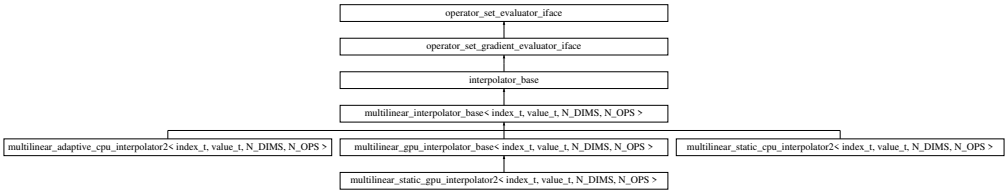
Parameters

in	*supporting_point_evaluator*	Object used to compute operators values at supporting points
in	*axes_points*	Number of supporting points (minimum 2) along axes
in	*axes_min*	Minimum value for each axis
in	*axes_max*	Maximum for each axis

5.32 multilinear_interpolator_base< index_t, value_t, N_DIMS, N_OPS > Class Template Reference

Interpolator base for static/adaptive piecewise mulitlinear interpolator.

Inheritance diagram for multilinear_interpolator_base< index_t, value_t, N_DIMS, N_OPS >:



Public Types

- typedef std::array< value_t, N_OPS > point_data_t
  values of all operators at a given (supporting) point
- typedef std::vector< double > point_coordinates_t
  coordinates of a given point in N_DIMS-dimensional space
- typedef std::array< int, N_DIMS > point_axes_index_t
  indexes of axes of point in parametrized space for each axis
- typedef std::array< value_t, N_VERTS ∗N_OPS > hypercube_data_t
  type for keeping values of all operators at all vertexes of a hypercube
- typedef std::array< index_t, N_VERTS > hypercube_points_index_t
  type for indexing vertexes of a hypercube

Public Member Functions

- multilinear_interpolator_base (operator_set_evaluator_iface *supporting_point_evaluator, const std::vector< int > &axes_points, const std::vector< double > &axes_min, const std::vector< double > &axes_max)

  Construct the interpolator with specified parametrization space.
- int get_n_dims ()

  Get the number of dimensions in interpolation space.
- int get_n_ops ()

  Get the number of operators to be interpolated.
- int interpolate (const std::vector< double > &point, std::vector< double > &values) override

  Compute interpolation for all operators at the given point.
- int interpolate_with_derivatives (const double *point, double *values, double *derivatives)

  Compute interpolation and its gradient for all operators at the given point point.
- int interpolate_with_derivatives (const std::vector< double > &points, const std::vector< int > &points_idxs, std::vector< double > &values, std::vector< double > &derivatives) override

  Compute interpolation and its gradient for all operators at every specified point.

Static Public Attributes

- static const uint16_t N_VERTS = (1 << N_DIMS)

  number of vertexes in interpolation hypercube - N_DIMS-th power of 2

Protected Member Functions

- void get_point_coordinates (index_t point_index, point_coordinates_t &coordinates)

  Get point coordinates in space for given point index.
- void get_hypercube_points (index_t index, hypercube_points_index_t &hypercube_points)

  Get indexes of all vertices for given hypercube.
- virtual const hypercube_data_t & get_hypercube_data (const index_t hypercube_index)=0

  Get values of operators at all vertices of the hypercube Implementation depends on underlying storage.

Protected Attributes

- const std::vector< value_t > axes_min_internal

  minimum at each axis in value_t type
- const std::vector< value_t > axes_max_internal

  maximum of each axis in value_t type
- const std::vector< value_t > axes_step_internal

  the distance between neighbor supporting points for each axis in value_t type
- const std::vector< value_t > axes_step_inv_internal

  inverse of step (to avoid division) in value_t type
- std::array< uint64_t, N_DIMS > axis_point_mult

  mult factor for each axis (for points) to compute global point index
- std::array< uint64_t, N_DIMS > axis_hypercube_mult

  mult factor for each axis (for hypercubes) to compute global hypercubes index

5.32.1 Detailed Description

template<typename index_t, typename value_t, uint8_t N_DIMS, uint8_t N_OPS>
class multilinear_interpolator_base< index_t, value_t, N_DIMS, N_OPS >

Interpolator base for static/adaptive piecewise mulitlinear interpolator.

Interpolation is performed simulataneously for several functions (operators) in multidimensional parameter space In order to do that, the space is uniformly parametrized within range of interest That range along each axis is devided by specific number of equal intervals, forming uniform mesh Each vertex of the mesh represents a supporting point, where operator values are evaluated exactly Using data at supporting points, interpolation is performed

Template Parameters

index_t	type used for indexing of supporting points and hypercubes
value_t	value type used for supporting point storage, hypercube storage and interpolation
N_DIMS	The number of dimensions in paramter space
N_OPS	The number of operators to be interpolated

5.32.2 Constructor & Destructor Documentation

5.32.2.1 multilinear_interpolator_base()

```
template<typename index_t, typename value_t, uint8_t N_DIMS, uint8_t N_OPS>
multilinear_interpolator_base< index_t, value_t, N_DIMS, N_OPS >::multilinear_interpolator_base (
            operator_set_evaluator_iface * supporting_point_evaluator,
            const std::vector< int > & axes_points,
            const std::vector< double > & axes_min,
            const std::vector< double > & axes_max )
```

Construct the interpolator with specified parametrization space.

Parameters

in	*supporting_point_evaluator*	Object used to compute operators values at supporting points
in	*axes_points*	Number of supporting points (minimum 2) along axes
in	*axes_min*	Minimum value for each axis
in	*axes_max*	Maximum for each axis

5.32.3 Member Function Documentation

5.32.3.1 get_hypercube_data()

```
template<typename index_t, typename value_t, uint8_t N_DIMS, uint8_t N_OPS>
virtual const hypercube_data_t& multilinear_interpolator_base< index_t, value_t, N_DIMS, N_OPS
>::get_hypercube_data (
            const index_t hypercube_index )  [protected], [pure virtual]
```

Get values of operators at all vertices of the hypercube Implementation depends on underlying storage.

Parameters

in	*hypercube_index*	index of hypercube

Returns

operator values at all vertices of the hypercube

Implemented in multilinear_static_cpu_interpolator2< index_t, value_t, N_DIMS, N_OPS >, multilinear_static_↩ gpu_interpolator2< index_t, value_t, N_DIMS, N_OPS >, and multilinear_adaptive_cpu_interpolator2< index_t, value_t, N_DIMS, N_OPS >.

5.32.3.2 get_hypercube_points()

```
template<typename index_t, typename value_t, uint8_t N_DIMS, uint8_t N_OPS>
void multilinear_interpolator_base< index_t, value_t, N_DIMS, N_OPS >::get_hypercube_points (
            index_t index,
            hypercube_points_index_t & hypercube_points )  [inline], [protected]
```

Get indexes of all vertices for given hypercube.

Parameters

in	*index*	index of the hyporcube
out	*hypercube_points*	indexes of all vertices of hypercube

5.32.3.3 get_point_coordinates()

```
template<typename index_t, typename value_t, uint8_t N_DIMS, uint8_t N_OPS>
void multilinear_interpolator_base< index_t, value_t, N_DIMS, N_OPS >::get_point_coordinates (
            index_t point_index,
            point_coordinates_t & coordinates )  [inline], [protected]
```

Get point coordinates in space for given point index.

Parameters

in	*point_index*	index of the point
out	*coordinates*	coordinates along all axes

5.32.3.4 interpolate()

```
template<typename index_t, typename value_t, uint8_t N_DIMS, uint8_t N_OPS>
int multilinear_interpolator_base< index_t, value_t, N_DIMS, N_OPS >::interpolate (
            const std::vector< double > & point,
            std::vector< double > & values )  [override], [virtual]
```

Compute interpolation for all operators at the given point.

Parameters

in	*point*	Coordinates in parametrization space
out	*values*	Interpolated values

Returns

0 if interpolation is successful

Implements interpolator_base.

5.32.3.5 interpolate_with_derivatives() [1/2]

```
template<typename index_t, typename value_t, uint8_t N_DIMS, uint8_t N_OPS>
int multilinear_interpolator_base< index_t, value_t, N_DIMS, N_OPS >::interpolate_with_↩
derivatives (
            const double * point,
            double * values,
            double * derivatives )
```

Compute interpolation and its gradient for all operators at the given point point.

Parameters

in	*points*	Coordinates of a point where interpolation is requested
out	*values*	Interpolated values
out	*derivatives*	Interpolation gradients

Returns

0 if interpolation is successful

5.32.3.6 interpolate_with_derivatives() [2/2]

```
template<typename index_t, typename value_t, uint8_t N_DIMS, uint8_t N_OPS>
int multilinear_interpolator_base< index_t, value_t, N_DIMS, N_OPS >::interpolate_with_↩
derivatives (
            const std::vector< double > & points,
            const std::vector< int > & points_idxs,
            std::vector< double > & values,
            std::vector< double > & derivatives )  [override], [virtual]
```

Compute interpolation and its gradient for all operators at every specified point.

Parameters

in	*points*	Array of coordinates in parametrization space
in	*points_idxs*	Indexes of points in the points array which are marked for interpolation
out	*values*	Interpolated values
out	*derivatives*	Interpolation gradients

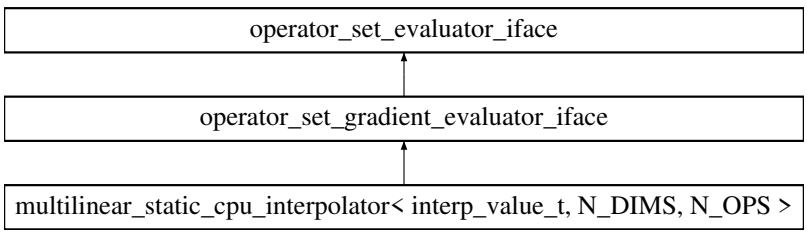
Returns

0 if interpolation is successful

Implements interpolator_base.

5.33 multilinear_static_cpu_interpolator< interp_value_t, N_DIMS, N_OPS > Class Template Reference

Inheritance diagram for multilinear_static_cpu_interpolator< interp_value_t, N_DIMS, N_OPS >:



Public Member Functions

- **multilinear_static_cpu_interpolator** (operator_set_evaluator_iface *base_points_generator, std::vector< index_t > &axis_resolution, std::vector< value_t > &axis_min, std::vector< value_t > &axis_max)

- int **benchmark** (index_t n_points, index_t n_blocks, value_t param_space_frac, index_t n_unique_bodies)
- int **evaluate_with_derivatives** (const std::vector< value_t > &state, const std::vector< index_t > &block_idx, std::vector< value_t > &values, std::vector< value_t > &derivatives)
- int **evaluate_with_derivatives_fake** (const std::vector< value_t > &state, const std::vector< index_t > &block_idx, std::vector< value_t > &values, std::vector< value_t > &derivatives)
- int **evaluate** (const std::vector< value_t > &state, std::vector< value_t > &values)
- void **get_point_state** (interp_index_t point_idx, state_t &state)
- void **get_body_points** (interp_index_t body_idx, body_data_points_t &body_points)
- int **get_resolution** ()
- value_t **get_axis_min** (int axis)
- value_t **get_axis_max** (int axis)
- uint64_t **get_n_points_total** ()
- uint64_t **get_n_hypercubes_used** ()
- uint64_t **get_n_hypercubes_total** ()
- int **clear_body_data** ()
- int **clear** ()

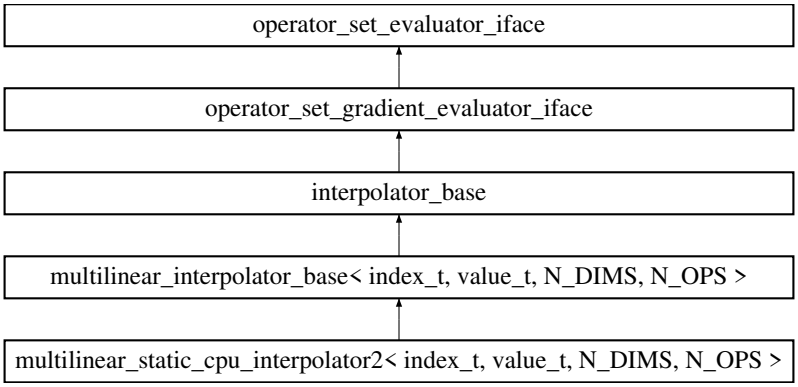
Public Attributes

- std::array< uint32_t, N_DIMS > **axis_points**
- std::array< interp_value_t, N_DIMS > **axis_min**
- std::array< interp_value_t, N_DIMS > **axis_max**
- interp_value_t **axis_step** [N_DIMS]
- interp_value_t **axis_step_inv** [N_DIMS]
- uint64_t **axis_mult** [N_DIMS]
- uint64_t **axis_body_mult** [N_DIMS]
- operator_set_evaluator_iface * **base_points_generator**
- uint64_t **n_interpolations**
- point_data_t **point_data**
- body_data_t **body_data**
- int **fake_mode**

5.34 multilinear_static_cpu_interpolator2< index_t, value_t, N_DIMS, N_OPS > Class Template Reference

Piecewise mulitlinear interpolator with static storage.

Inheritance diagram for multilinear_static_cpu_interpolator2< index_t, value_t, N_DIMS, N_OPS >:



Public Member Functions

- multilinear_static_cpu_interpolator2 (operator_set_evaluator_iface *supporting_point_evaluator, const std::vector< int > &axes_points, const std::vector< double > &axes_min, const std::vector< double > &axes_max)

  Construct the interpolator with specified parametrization space.
- int init () override

  Initialize the interpolator by:

Public Attributes

- std::vector< point_data_t > point_data

  static point storage: the values of operators at all supporting points

Protected Member Functions

- const hypercube_data_t & get_hypercube_data (const index_t hypercube_index)

  Get values of operators at all vertices of the hypercube. Simply provide a reference to correct location in static storage - all values have been already computed.

Protected Attributes

- std::vector< hypercube_data_t > hypercube_data

  static hypercube storage: the values of operators at every vertex of all hypercubes

Additional Inherited Members

5.34.1 Detailed Description

template<typename index_t, typename value_t, uint8_t N_DIMS, uint8_t N_OPS>
class multilinear_static_cpu_interpolator2< index_t, value_t, N_DIMS, N_OPS >

Piecewise mulitlinear interpolator with static storage.

Static storage is initialized in init() method. Two-level storage is used: with operator data at every supporting point and with operator data at all vertices of every hypercube point data may be assigned externally after construction and before init() call to save time hypercube storage then is initialized only and much faster, as does not involve computation of supporting points, only copying

Template Parameters

index_t	type used for indexing of supporting points and hypercubes
value_t	value type used for supporting point storage, hypercube storage and interpolation
N_DIMS	The number of dimensions in paramter space
N_OPS	The number of operators to be interpolated

5.34.2 Constructor & Destructor Documentation

5.34.2.1 multilinear_static_cpu_interpolator2()

```
template<typename index_t , typename value_t , uint8_t N_DIMS, uint8_t N_OPS>
multilinear_static_cpu_interpolator2< index_t, value_t, N_DIMS, N_OPS >::multilinear_static_↩
cpu_interpolator2 (
            operator_set_evaluator_iface * supporting_point_evaluator,
            const std::vector< int > & axes_points,
            const std::vector< double > & axes_min,
            const std::vector< double > & axes_max )
```

Construct the interpolator with specified parametrization space.

Parameters

in	*supporting_point_evaluator*	Object used to compute operators values at supporting points
in	*axes_points*	Number of supporting points (minimum 2) along axes
in	*axes_min*	Minimum value for each axis
in	*axes_max*	Maximum for each axis

5.34.3 Member Function Documentation

5.34.3.1 get_hypercube_data()

```
template<typename index_t , typename value_t , uint8_t N_DIMS, uint8_t N_OPS>
const hypercube_data_t& multilinear_static_cpu_interpolator2< index_t, value_t, N_DIMS, N_OPS
>::get_hypercube_data (
            const index_t hypercube_index )  [protected], [virtual]
```

Get values of operators at all vertices of the hypercube. Simply provide a reference to correct location in static storage - all values have been already computed.

Parameters

in	*hypercube_index*	index of hypercube

Returns

operator values at all vertices of the hypercube

Implements multilinear_interpolator_base< index_t, value_t, N_DIMS, N_OPS >.

5.34.3.2 init()

```
template<typename index_t , typename value_t , uint8_t N_DIMS, uint8_t N_OPS>
int multilinear_static_cpu_interpolator2< index_t, value_t, N_DIMS, N_OPS >::init ( )  [override],
[virtual]
```

Initialize the interpolator by:

1. computing all values of supporting points (if point_data storage was not already initialized)
2. populating hypercube static storage from point storage

Returns

int 0 if successful

Reimplemented from interpolator_base.

5.34.4 Member Data Documentation

5.34.4.1 hypercube_data

```
template<typename index_t , typename value_t , uint8_t N_DIMS, uint8_t N_OPS>
std::vector<hypercube_data_t> multilinear_static_cpu_interpolator2< index_t, value_t, N_DIMS,
N_OPS >::hypercube_data  [protected]
```

static hypercube storage: the values of operators at every vertex of all hypercubes

In fact it is an excess storage used to reduce memory accesses during interpolation. Here all values of all vertexes of every hypercube are stored consecutevely and are accessed via a single index Usage of point_data for interpolation directly would require N_VERTS memory accesses (>1000 accesses for 10-dimensional space)

5.34.4.2 point_data

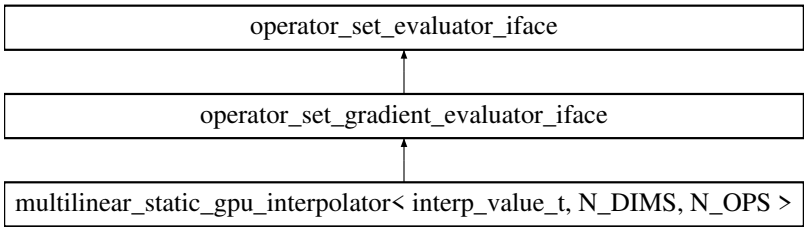
```
template<typename index_t , typename value_t , uint8_t N_DIMS, uint8_t N_OPS>
std::vector<point_data_t> multilinear_static_cpu_interpolator2< index_t, value_t, N_DIMS, N↩
_OPS >::point_data
```

static point storage: the values of operators at all supporting points

Used to store all computed supporting points and to initialize hypercube_data

5.35 multilinear_static_gpu_interpolator< interp_value_t, N_DIMS, N_OPS > Class Template Reference

Inheritance diagram for multilinear_static_gpu_interpolator< interp_value_t, N_DIMS, N_OPS >:



Public Member Functions

- **multilinear_static_gpu_interpolator** (operator_set_evaluator_iface *base_points_generator, std::vector< index_t > &axis_resolution, std::vector< value_t > &axis_min, std::vector< value_t > &axis_max)
- int **benchmark** (index_t n_points, index_t n_blocks, value_t used_paramspace_fraction, index_t n_unique_bodies)
- int **evaluate_with_derivatives** (const std::vector< value_t > &state, const std::vector< index_t > &block_idx, std::vector< value_t > &values, std::vector< value_t > &derivatives)
- virtual int **evaluate_with_derivatives_d** (index_t n_blocks, value_t *state, index_t *block_idx, value_t *values, value_t *derivatives)
- int **evaluate** (const std::vector< value_t > &state, std::vector< value_t > &values)
- void **get_point_state** (interp_index_t point_idx, state_t &state)
- void **get_body_points** (interp_index_t body_idx, body_data_points_t &body_points)
- int **get_resolution** ()
- value_t **get_axis_min** (int axis)
- value_t **get_axis_max** (int axis)
- uint64_t **get_n_interpolations** ()
- uint64_t **get_n_points_used** ()
- uint64_t **get_n_points_total** ()
- uint64_t **get_n_hypercubes_used** ()
- uint64_t **get_n_hypercubes_total** ()
- int **clear_body_data** ()
- int **clear** ()

Public Attributes

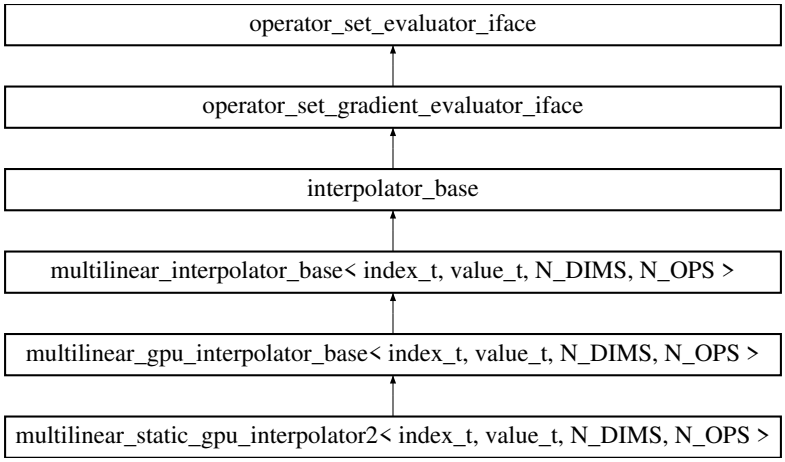
- std::array< uint32_t, N_DIMS > **axis_points**
- std::array< value_t, N_DIMS > **axis_min**
- std::array< value_t, N_DIMS > **axis_max**
- value_t **axis_step** [N_DIMS]
- value_t **axis_step_inv** [N_DIMS]
- uint64_t **axis_mult** [N_DIMS]
- uint32_t **axis_body_mult** [N_DIMS]
- operator_set_evaluator_iface * **base_points_generator**
- uint64_t **n_interpolations**
- point_data_t **point_data**
- body_data_t **body_data**
- interp_value_t * **body_data_gpu**

- value_t * **state_gpu**
- value_t * **values_gpu**
- value_t * **derivatives_gpu**
- index_t * **block_idx_gpu**
- index_t **gpu_data_size**
- int **fake_mode**

5.36 multilinear_static_gpu_interpolator2< index_t, value_t, N_DIMS, N_OPS > Class Template Reference

Piecewise mulitlinear interpolator with static storage.

Inheritance diagram for multilinear_static_gpu_interpolator2< index_t, value_t, N_DIMS, N_OPS >:



Public Member Functions

- multilinear_static_gpu_interpolator2 (operator_set_evaluator_iface *supporting_point_evaluator, const std::vector< int > &axes_points, const std::vector< double > &axes_min, const std::vector< double > &axes_max)

  Construct the interpolator with specified parametrization space.
- int init () override

  Initialize the interpolator by:

Public Attributes

- std::vector< point_data_t > point_data

  static point storage: the values of operators at all supporting points

Protected Member Functions

- const hypercube_data_t & get_hypercube_data (const index_t hypercube_index)

  Get values of operators at all vertices of the hypercube. Simply provide a reference to correct location in static storage - all values have been already computed.

Protected Attributes

- std::vector< hypercube_data_t > hypercube_data

  static hypercube storage: the values of operators at every vertex of all hypercubes

Additional Inherited Members

5.36.1 Detailed Description

template<typename index_t, typename value_t, uint8_t N_DIMS, uint8_t N_OPS>
class multilinear_static_gpu_interpolator2< index_t, value_t, N_DIMS, N_OPS >

Piecewise mulitlinear interpolator with static storage.

Static storage is initialized in init() method. Two-level storage is used: with operator data at every supporting point and with operator data at all vertices of every hypercube point data may be assigned externally after construction and before init() call to save time hypercube storage then is initialized only and much faster, as does not involve computation of supporting points, only copying

Template Parameters

index_t	type used for indexing of supporting points and hypercubes
value_t	value type used for supporting point storage, hypercube storage and interpolation
N_DIMS	The number of dimensions in paramter space
N_OPS	The number of operators to be interpolated

5.36.2 Constructor & Destructor Documentation

5.36.2.1 multilinear_static_gpu_interpolator2()

```
template<typename index_t , typename value_t , uint8_t N_DIMS, uint8_t N_OPS>
multilinear_static_gpu_interpolator2< index_t, value_t, N_DIMS, N_OPS >::multilinear_static_gpu_interpolator2 (
            operator_set_evaluator_iface * supporting_point_evaluator,
            const std::vector< int > & axes_points,
            const std::vector< double > & axes_min,
            const std::vector< double > & axes_max )
```

Construct the interpolator with specified parametrization space.

Parameters

in	*supporting_point_evaluator*	Object used to compute operators values at supporting points
in	*axes_points*	Number of supporting points (minimum 2) along axes
in	*axes_min*	Minimum value for each axis
in	*axes_max*	Maximum for each axis

5.36.3 Member Function Documentation

5.36.3.1 get_hypercube_data()

```
template<typename index_t , typename value_t , uint8_t N_DIMS, uint8_t N_OPS>
const hypercube_data_t& multilinear_static_gpu_interpolator2< index_t, value_t, N_DIMS, N_OPS
>::get_hypercube_data (
            const index_t hypercube_index )  [protected], [virtual]
```

Get values of operators at all vertices of the hypercube. Simply provide a reference to correct location in static storage - all values have been already computed.

Parameters

in	*hypercube_index*	index of hypercube

Returns

operator values at all vertices of the hypercube

Implements multilinear_interpolator_base< index_t, value_t, N_DIMS, N_OPS >.

5.36.3.2 init()

```
template<typename index_t , typename value_t , uint8_t N_DIMS, uint8_t N_OPS>
int multilinear_static_gpu_interpolator2< index_t, value_t, N_DIMS, N_OPS >::init ( )  [override],
[virtual]
```

Initialize the interpolator by:

1. computing all values of supporting points (if point_data storage was not already initialized)
2. populating hypercube static storage from point storage

Returns

int 0 if successful

Reimplemented from interpolator_base.

5.36.4 Member Data Documentation

5.36.4.1 hypercube_data

```
template<typename index_t , typename value_t , uint8_t N_DIMS, uint8_t N_OPS>
std::vector<hypercube_data_t> multilinear_static_gpu_interpolator2< index_t, value_t, N_DIMS,
N_OPS >::hypercube_data  [protected]
```

static hypercube storage: the values of operators at every vertex of all hypercubes

In fact it is an excess storage used to reduce memory accesses during interpolation. Here all values of all vertexes of every hypercube are stored consecutevely and are accessed via a single index Usage of point_data for interpolation directly would require N_VERTS memory accesses (>1000 accesses for 10-dimensional space)

5.36.4.2 point_data

```
template<typename index_t , typename value_t , uint8_t N_DIMS, uint8_t N_OPS>
std::vector<point_data_t> multilinear_static_gpu_interpolator2< index_t, value_t, N_DIMS, N↩
_OPS >::point_data
```

static point storage: the values of operators at all supporting points

Used to store all computed supporting points and to initialize hypercube_data

5.37 pm::pm_discretizer Class Reference

Classes

- struct Gradients
- struct InnerMatrices

Public Member Functions

- void **init** ()
- void **reconstruct_gradients_per_cell** (value_t dt)
- void **calc_all_fluxes** (value_t dt)
- void **calc_all_fluxes_once** (value_t dt)

Public Attributes

- std::vector< std::vector< Face > > **faces**
- std::vector< Matrix33 > **perms**
- std::vector< Matrix33 > **biots**
- std::vector< Stiffness > **stfs**
- std::vector< Matrix > **cell_centers**
- std::vector< Matrix > **u0**
- std::vector< Matrix > **bc**
- std::vector< Matrix > **bc_prev**
- std::vector< value_t > **x_prev**
- value_t **visc**
- value_t **grav**
- value_t **density**

- Matrix **grav_vec**
- std::vector< index_t > **cell_m**
- std::vector< index_t > **cell_p**
- std::vector< index_t > **stencil**
- std::vector< index_t > **offset**
- std::vector< value_t > **tran**
- std::vector< value_t > **rhs**
- std::vector< value_t > **tran_biot**
- std::vector< value_t > **rhs_biot**
- std::vector< Gradients > **grad**
- std::vector< Gradients > **grad_prev**

Static Public Attributes

- static const uint8_t **MIN_FACE_NUM** = 4
- static const uint8_t **MAX_FACE_NUM** = 8
- static const uint8_t **BLOCK_SIZE** = 4
- static const int **MAX_STENCIL** = 15

Protected Member Functions

- Matrix **get_u_face_prev** (const Matrix dr, const index_t cell_id) const
- Matrix **get_ub_prev** (const Face &face) const
- Matrix **calc_grad_prev** (const index_t cell_id) const
- Matrix **calc_grad_cur** (const index_t cell_id) const
- Matrix **calc_vector** (const Matrix &a, const Matrix &rhs, const std::vector< index_t > &stencil) const
- Gradients **merge_stencils** (const std::vector< index_t > &st1, const Matrix &m1, const std::vector< index_t > &st2, const Matrix &m2)
- bool **check_trans_sum** (const std::vector< index_t > &st, const Matrix &a) const
- void **write_trans** (const std::vector< index_t > &st, const Matrix &from)
- void **write_trans_biot** (const std::vector< index_t > &st, const Matrix &from, const Matrix &from_biot)
- std::pair< bool, size_t > **findInVector** (const std::vector< index_t > &vec, const index_t &element)

Protected Attributes

- int **n_cells**
- int **n_faces**
- int **nb_faces**
- std::map< uint8_t, Matrix > **pre_A**
- std::map< uint8_t, Matrix > **pre_u1_mult**
- std::map< uint8_t, Matrix > **pre_u2_mult**
- std::map< uint8_t, Matrix > **pre_rest**
- std::map< uint8_t, Matrix > **pre_rhs_mult**
- std::map< uint8_t, std::map< uint8_t, Matrix > > **pre_cur_rhs**
- Matrix **pre_merged_grad**
- std::vector< index_t > **pre_merged_stencil**
- index_t **st_id**
- Matrix **W**
- std::vector< std::map< uint8_t, InnerMatrices > > **inner**
- uint8_t **counter**
- index_t **id**
- std::vector< index_t >::const_iterator **it_find**
- std::pair< bool, size_t > **res1**
- std::pair< bool, size_t > **res2**

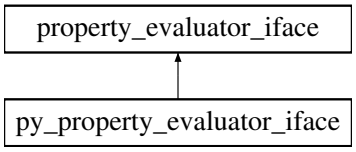
Static Protected Attributes

- static const value_t **darcy_constant** = 0.008526714671916010498687664041994 8
- static const Matrix **I3** = pm::Matrix({ 1,0,0, 0,1,0, 0,0,1 }, ND, ND)
- static const Matrix **I4** = pm::Matrix({ 1,0,0,0, 0,1,0,0, 0,0,1,0, 0,0,0,1 }, ND + 1, ND + 1)

5.38 property_evaluator_iface Class Reference

Virtual interface class for evaluation of physical properties values Implemented mainly by different C++ physical kernels from darts.physics However, pure Python implementation is also possible through inheritance.

Inheritance diagram for property_evaluator_iface:



Public Member Functions

- virtual double evaluate (const std::vector< double > &state)=0

  Compute property values for specified state.
- int evaluate (const std::vector< double > &states, int n_blocks, std::vector< double > &values)

  Compute property values for all specified states A surrogate for vectorized evaluate function.

5.38.1 Detailed Description

Virtual interface class for evaluation of physical properties values Implemented mainly by different C++ physical kernels from darts.physics However, pure Python implementation is also possible through inheritance.

5.38.2 Member Function Documentation

5.38.2.1 evaluate() [1/2]

```
virtual double property_evaluator_iface::evaluate (
            const std::vector< double > & state )  [pure virtual]
```

Compute property values for specified state.

Parameters

state	Coordinates in parameter space, where operators to be evaluated

Returns

double property value

Implemented in py_property_evaluator_iface.

5.38.2.2 evaluate() [2/2]

```
int property_evaluator_iface::evaluate (
            const std::vector< double > & states,
            int n_blocks,
            std::vector< double > & values )  [inline]
```

Compute property values for all specified states A surrogate for vectorized evaluate function.

Parameters

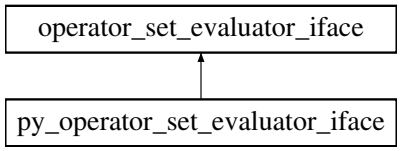
in	*states*	array of states
in	*n_blocks*	the number of states
out	*values*	evaluated property values

Returns

int

5.39 py_operator_set_evaluator_iface Class Reference

Inheritance diagram for py_operator_set_evaluator_iface:

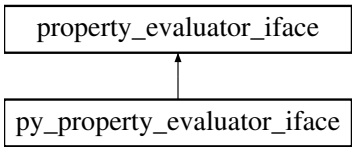


Public Member Functions

- int **evaluate** (const std::vector< value_t > &state, std::vector< value_t > &values)

5.40 py_property_evaluator_iface Class Reference

Inheritance diagram for py_property_evaluator_iface:



Public Member Functions

- value_t evaluate (const std::vector< value_t > &state)

  Compute property values for specified state.
- int **evaluate** (const std::vector< value_t > &states, index_t n_blocks, std::vector< value_t > &values)

5.40.1 Member Function Documentation

5.40.1.1 evaluate()

```
value_t py_property_evaluator_iface::evaluate (
            const std::vector< value_t > & state )  [inline], [virtual]
```

Compute property values for specified state.

Parameters

state	Coordinates in parameter space, where operators to be evaluated

Returns

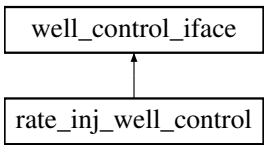
double property value

Implements property_evaluator_iface.

5.41 rate_inj_well_control Class Reference

Volumetric rate control for injection compositional well.

Inheritance diagram for rate_inj_well_control:



Public Member Functions

- **rate_inj_well_control** (std::vector< std::string > phase_names_, index_t target_phase_idx_, index_t n_equations_, index_t n_variables_, value_t target_rate_, std::vector< value_t > &injection_stream_, operator_set_gradient_evaluator_iface *rate_etor_)
- virtual int **add_to_jacobian** (value_t dt, index_t well_head_idx, value_t segment_trans, index_t n_block_size, std::vector< value_t > &X, value_t *jacobian_row, std::vector< value_t > &RHS)
- virtual int **add_to_csr_jacobian** (value_t dt, index_t well_head_idx, value_t segment_trans, index_t n_block_size, std::vector< value_t > &X, value_t *jacobian_row, std::vector< value_t > &RHS)
- virtual int **check_constraint_violation** (value_t dt, index_t well_head_idx, value_t segment_trans, index_t n_block_size, std::vector< value_t > &X)
- virtual int **initialize_well_block** (std::vector< value_t > &state_block, const std::vector< value_t > &state_neighbour)

Public Attributes

- index_t **target_phase_idx**
- index_t **n_equations**
- index_t **n_variables**
- std::vector< std::string > **phase_names**
- value_t **target_rate**
- std::vector< value_t > **injection_stream**
- operator_set_gradient_evaluator_iface * **rate_etor**
- std::vector< value_t > **state**
- std::vector< value_t > **rates**
- std::vector< value_t > **rates_derivs**

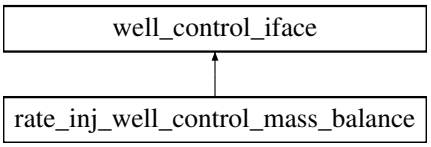
5.41.1 Detailed Description

Volumetric rate control for injection compositional well.

5.42 rate_inj_well_control_mass_balance Class Reference

Rate control based on mass ballance equation for injection compositional well.

Inheritance diagram for rate_inj_well_control_mass_balance:



Public Member Functions

- **rate_inj_well_control_mass_balance** (std::vector< std::string > phase_names_, index_t target_phase_↩ idx_, index_t n_equations_, index_t n_variables_, value_t target_rate_, std::vector< value_t > &injection_↩ stream_, operator_set_evaluator_iface *rate_etor_, operator_set_gradient_evaluator_iface *sources_etor↩ _)
- virtual int **add_to_jacobian** (value_t dt, index_t well_head_idx, value_t segment_trans, index_t n_block_size, std::vector< value_t > &X, value_t *jacobian_row, std::vector< value_t > &RHS)
- virtual int **check_constraint_violation** (value_t dt, index_t well_head_idx, value_t segment_trans, index_t n_block_size, std::vector< value_t > &X)
- virtual int **initialize_well_block** (std::vector< value_t > &state_block, const std::vector< value_t > &state↩ _neighbour)

Public Attributes

- index_t **target_phase_idx**
- index_t **n_equations**
- index_t **n_variables**
- std::vector< std::string > **phase_names**
- value_t **target_rate**
- std::vector< value_t > **injection_stream**
- operator_set_gradient_evaluator_iface * **sources_etor**
- operator_set_evaluator_iface * **rate_etor**
- std::vector< value_t > **state**
- std::vector< value_t > **sources**
- std::vector< value_t > **rates**
- std::vector< value_t > **sources_derivs**

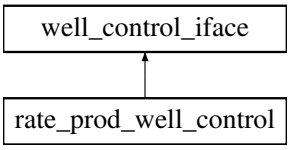
5.42.1 Detailed Description

Rate control based on mass ballance equation for injection compositional well.

5.43 rate_prod_well_control Class Reference

Volumetric rate control for production compositional well.

Inheritance diagram for rate_prod_well_control:



Public Member Functions

- **rate_prod_well_control** (std::vector< std::string > phase_names_, index_t target_phase_idx_, index_t n↩ _equations_, index_t n_variables_, value_t target_rate_, operator_set_gradient_evaluator_iface *rate_etor↩ _)
- virtual int **add_to_jacobian** (value_t dt, index_t well_head_idx, value_t segment_trans, index_t n_block_size, std::vector< value_t > &X, value_t *jacobian_row, std::vector< value_t > &RHS)
- virtual int **check_constraint_violation** (value_t dt, index_t well_head_idx, value_t segment_trans, index_t n_block_size, std::vector< value_t > &X)
- virtual int **initialize_well_block** (std::vector< value_t > &state_block, const std::vector< value_t > &state↩ _neighbour)

Public Attributes

- index_t **target_phase_idx**
- index_t **n_equations**
- index_t **n_variables**
- std::vector< std::string > **phase_names**
- value_t **target_rate**
- operator_set_gradient_evaluator_iface * **rate_etor**
- std::vector< value_t > **state**
- std::vector< value_t > **rates**
- std::vector< value_t > **rates_derivs**

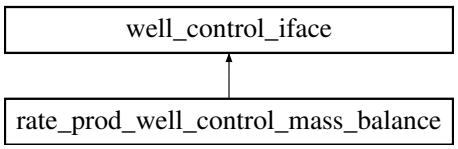
5.43.1 Detailed Description

Volumetric rate control for production compositional well.

5.44 rate_prod_well_control_mass_balance Class Reference

Rate control based on mass ballance equation for production compositional well.

Inheritance diagram for rate_prod_well_control_mass_balance:



Public Member Functions

- **rate_prod_well_control_mass_balance** (std::vector< std::string > phase_names_, index_t target_↩ phase_idx_, index_t n_equations_, index_t n_variables_, value_t target_rate_, operator_set_evaluator_iface ∗rate_etor_, operator_set_gradient_evaluator_iface ∗sources_etor_)
- virtual int **add_to_jacobian** (value_t dt, index_t well_head_idx, value_t segment_trans, index_t n_block_size, std::vector< value_t > &X, value_t ∗jacobian_row, std::vector< value_t > &RHS)
- virtual int **check_constraint_violation** (value_t dt, index_t well_head_idx, value_t segment_trans, index_t n_block_size, std::vector< value_t > &X)
- virtual int **initialize_well_block** (std::vector< value_t > &state_block, const std::vector< value_t > &state↩ _neighbour)

Public Attributes

- index_t **target_phase_idx**
- index_t **n_equations**
- index_t **n_variables**
- std::vector< std::string > **phase_names**
- value_t **target_rate**
- operator_set_gradient_evaluator_iface ∗ **sources_etor**
- operator_set_evaluator_iface ∗ **rate_etor**
- std::vector< value_t > **state**
- std::vector< value_t > **sources**
- std::vector< value_t > **rates**
- std::vector< value_t > **sources_derivs**

5.44.1 Detailed Description

Rate control based on mass ballance equation for production compositional well.

5.45 sim_params Class Reference

Main simulation parameters including tolerances.

Public Types

- enum **newton_solver_t** { **NEWTON_STD** = 0, **NEWTON_GLOBAL_CHOP**, **NEWTON_LOCAL_CHOP**, **NEWTON_INFLECTION_POINT** }
- enum **linear_solver_t** { **CPU_GMRES_CPR_AMG** = 0, **CPU_GMRES_CPR_AMG1R5**, **CPU_GMRES_FS_CPR**, **CPU_GMRES_ILU0**, **CPU_SUPERLU**, **GPU_GMRES_CPR_AMG**, **GPU_GMRES_ILU0**, **GPU_GMRES_CPR_AIPS**, **GPU_GMRES_CPR_AMGX_ILU**, **GPU_GMRES_CPR_AMGX_ILU_SP**, **GPU_GMRES_CPR_AMGX_AMGX**, **GPU_GMRES_AMGX**, **GPU_AMGX**, **GPU_GMRES_CPR_NF**, **GPU_BICGSTAB_CPR_AMGX** }
- enum **nonlinear_norm_t** { **L1** = 0, **L2**, **LINF** }

Public Attributes

- value_t **first_ts**
- value_t **max_ts**
- value_t **mult_ts**
- index_t **max_i_newton**
- index_t **min_i_newton**
- index_t **max_i_linear**
- value_t **tolerance_newton**
- value_t **tolerance_linear**
- index_t **tot_newt_count**
- index_t **log_transform**
- index_t **interface_avg_tmult**
- index_t **trans_mult_exp**
- value_t **obl_min_fac**
- int **assembly_kernel**
- newton_solver_t **newton_type**
- linear_solver_t **linear_type**
- nonlinear_norm_t **nonlinear_norm_type**
- std::vector< value_t > **newton_params**
- std::vector< value_t > **linear_params**
- std::vector< int > **global_actnum**

5.45.1 Detailed Description

Main simulation parameters including tolerances.

5.46 sim_stat Class Reference

Main simulation statistics with active and wasted counts.

Public Attributes

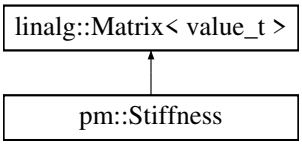
- index_t **n_newton_total**
- index_t **n_linear_total**
- index_t **n_newton_wasted**
- index_t **n_linear_wasted**
- index_t **n_timesteps_total**
- index_t **n_timesteps_wasted**

5.46.1 Detailed Description

Main simulation statistics with active and wasted counts.

5.47 pm::Stiffness Class Reference

Inheritance diagram for pm::Stiffness:



Public Types

- typedef Matrix **Base**

Public Member Functions

- **Stiffness** (value_t la, value_t mu)
- **Stiffness** (std::valarray< value_t > _c)

Static Public Attributes

- static const index_t **N** = SUM_N(ND) * SUM_N(ND)

Additional Inherited Members

5.48 well_control Struct Reference

Structure for well control.

Public Member Functions

- well_control ()

  temperature of injection stream

Public Attributes

- int **control_type**
- value_t control_param

  INJECTOR_BHP or PRODUCER_BHP.
- std::vector< value_t > inj_stream

  BHP (or rate) value.
- value_t inj_temperature

  composition of injection stream

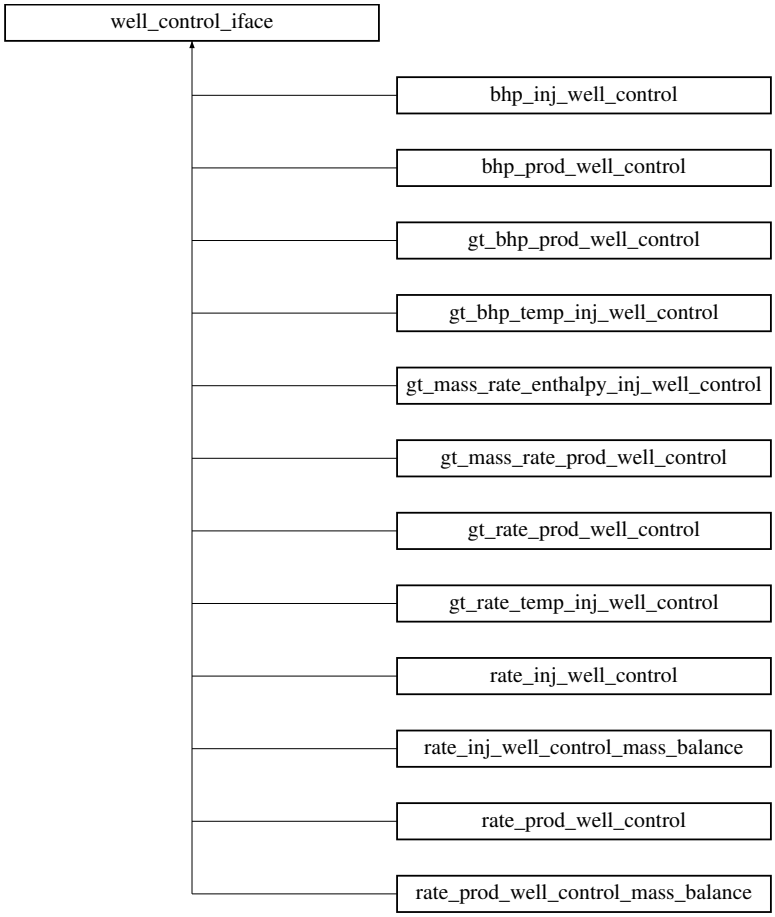
5.48.1 Detailed Description

Structure for well control.

5.49 well_control_iface Class Reference

Base class work well control/constraint.

Inheritance diagram for well_control_iface:



Public Member Functions

- virtual int **add_to_jacobian** (value_t dt, index_t well_head_idx, value_t segment_trans, index_t n_block_size, std::vector< value_t > &X, value_t *jacobian_row, std::vector< value_t > &RHS)=0
- virtual int **check_constraint_violation** (value_t dt, index_t well_head_idx, value_t segment_trans, index_t n_block_size, std::vector< value_t > &X)=0
- virtual int **initialize_well_block** (std::vector< value_t > &state_block, const std::vector< value_t > &state↩_neighbour)=0
- virtual int **add_to_csr_jacobian** (value_t dt, index_t well_head_idx, value_t segment_trans, index_t n_↩block_size, std::vector< value_t > &X, value_t *jacobian_row, std::vector< value_t > &RHS)

Public Attributes

- std::string **name**
- std::vector< index_t > **block_idx**

5.49.1 Detailed Description

Base class work well control/constraint.