Understanding Breakthrough Curves

v0.3

Generated by Doxygen 1.9.1

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 isotherms Namespace Reference

defines the Langmuir isotherm class

Data Structures

- class Langmuir

  Class for Langmuir isotherm.

5.1.1 Detailed Description

defines the Langmuir isotherm class

5.2 plotting_util Namespace Reference

Utilities for making plots.

Data Structures

- class Convergence

  Class for performing convergence analysis.
- class Temporal

  class for plotting temporal convergence
- class Spatial

  class for plotting temporal convergence
- class AsymptoticConvergence

  Class for plotting convergence in epsilon (not N or M)
- class TWConvergence

  Class for plotting traveling wave convergence in epsilon (not N or M)

Functions

- def get_u_TW_Langmuir (xi)

  get traveling wave solution
- def calculate_slope_error (x, y, slope, intercept, conf=0.95)

  calculate the error in the slope estimated with linear regression
- def save_figure (fig, name)

  saves figure to out/ directory

Variables

- **ISOTHERM** = Langmuir(1)
- **BASE_DIR** = os.path.abspath(os.path.dirname(__file__))

5.2.1 Detailed Description

Utilities for making plots.

5.2.2 Function Documentation

5.2.2.1 calculate_slope_error()

```
def plotting_util.calculate_slope_error (
              x,
              y,
              slope,
              intercept,
              conf = 0.95 )
```

calculate the error in the slope estimated with linear regression

Parameters

x	np array of x-values of plot
y	numpy array of y-values of plot
slope	float representing slope calculated
intercept	float representing intercept calculated
conf	float in (0, 1) representing confidence fraction. Defaults to 0.95

5.2.2.2 get_u_TW_Langmuir()

```
def plotting_util.get_u_TW_Langmuir (
              xi )
```

get traveling wave solution

Parameters

xi	(np.array) positions to evaluate concentrations in boundary layer

Returns

concentrations inside boundary layer

5.2.2.3 save_figure()

```
def plotting_util.save_figure (
              fig,
              name )
```

saves figure to *out/* directory

Parameters

fig	(matplotlib.pyplot.figure) instance of figure to be saved
name	(str) name of figure to be saved

Chapter 6

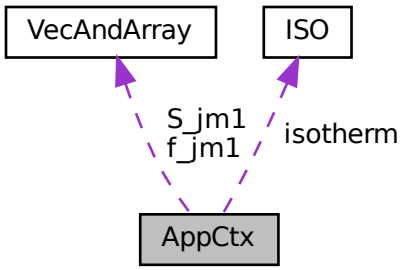
Data Structure Documentation

6.1 AppCtx Struct Reference

Useful parameters for solving, Application context.

`#include <my_petsc.h>`

Collaboration diagram for AppCtx:



Data Fields

- PetscInt j
- PetscInt m
- PetscReal errnorm_all
- PetscReal eps
- ISO isotherm
- PetscReal theta
- PetscReal dx
- PetscReal dtau
- PetscReal * btc
- PetscReal tau_star

- VecAndArray f_jm1
- VecAndArray S_jm1
- bool compute_analytical
- bool make_movie
- bool save_btc
- bool print_header
- PetscViewer viewer
- PetscInt movie_step_interval
- PetscReal omega
- PetscReal b
- PetscReal K
- PetscReal beta
- PetscReal kappa
- PetscReal dtau_est

6.1.1 Detailed Description

Useful parameters for solving, Application context.

6.1.2 Field Documentation

6.1.2.1 b

```
PetscReal AppCtx::b
```

scale factor for beta

6.1.2.2 beta

```
PetscReal AppCtx::beta
```

constant calculated

6.1.2.3 btc

```
PetscReal* AppCtx::btc
```

array for break-through curve

6.1.2.4 compute_analytical

```
bool AppCtx::compute_analytical
```

Whether or not to compute analytical solution

6.1.2.5 dtau

`PetscReal AppCtx::dtau`

time step

6.1.2.6 dtau_est

`PetscReal AppCtx::dtau_est`

estimated dtau for setting it when tau_star unknown

6.1.2.7 dx

`PetscReal AppCtx::dx`

grid spacing

6.1.2.8 eps

`PetscReal AppCtx::eps`

value of ε

6.1.2.9 errnorm_all

`PetscReal AppCtx::errnorm_all`

infinity norm

6.1.2.10 f_jm1

`VecAndArray AppCtx::f_jm1`

Isotherm function at previous time step

6.1.2.11 isotherm

`ISO AppCtx::isotherm`

structure for adsorption isotherm

6.1.2.12 j

`PetscInt AppCtx::j`

time step

6.1.2.13 K

`PetscReal AppCtx::K`

scale factor for kappa

6.1.2.14 kappa

`PetscReal AppCtx::kappa`

constant calculated

6.1.2.15 m

`PetscInt AppCtx::m`

total number of steps

6.1.2.16 make_movie

`bool AppCtx::make_movie`

whether or not to make movie

6.1.2.17 movie_step_interval

`PetscInt AppCtx::movie_step_interval`

step interval for saving movie frames

6.1.2.18 omega

`PetscReal AppCtx::omega`

closeness to steady state at end of simulation

6.1.2.19 print_header

`bool AppCtx::print_header`

whether to print header when displaying results of simulation

6.1.2.20 S_jm1

`VecAndArray AppCtx::S_jm1`

Solid concentration at previous time step

6.1.2.21 save_btc

```
bool AppCtx::save_btc
```

flag to save btc

6.1.2.22 tau_star

```
PetscReal AppCtx::tau_star
```

total simulation time $\tau_\star$

6.1.2.23 theta

```
PetscReal AppCtx::theta
```

fraction of BFD/FFD

6.1.2.24 viewer

```
PetscViewer AppCtx::viewer
```

petsc viewer

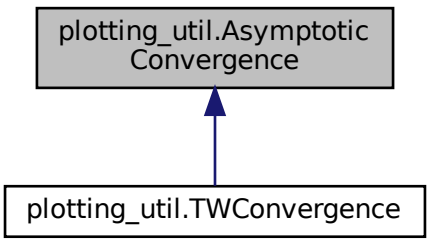
The documentation for this struct was generated from the following file:

- src/my_petsc.h

6.2 plotting_util.AsymptoticConvergence Class Reference

Class for plotting convergence in epsilon (not N or M)

Inheritance diagram for plotting_util.AsymptoticConvergence:



Public Member Functions

- def __init__ (self, Langmuir isotherm, file_prefix, tau_star)

  initialize the class
- def get_other_theory (self, ts)

  get btc associated with other theory
- def plot_btcs (self, ax)

  plot breakthrough curves of numerical simulation and regular perturbation theory
- def plot_error (self, ax, plot_kwargs, line_kwargs=None)

  plot errors as a function of ε

Data Fields

- **isotherm**
- **klasses**
- **es**
- **colors**
- **file_prefix**
- **tau_star**

6.2.1 Detailed Description

Class for plotting convergence in epsilon (not N or M)

Parameters

isotherm	(Langmuir) type of isotherm
klasses	array of solutions to store
es	array of epsilons, hard-coded to 0.02, 0.04, 0.08, 0.16, 0.32, 0.64
colors,list	of colors to plot, hard-coded to plt.cm.viridis_r
tau_star,total	dimensionless time to simulate

6.2.2 Constructor & Destructor Documentation

6.2.2.1 __init__()

```
def plotting_util.AsymptoticConvergence.__init__ (
              self,
             Langmuir isotherm,
              file_prefix,
              tau_star )
```

initialize the class

```
   @param isotherm isotherm to use
   @param file_prefix (str) prefix for looking at files
   @param tau_star (float) total dimensionless time to simulate

   @returns instance of class
```

Reimplemented in plotting_util.TWConvergence.

6.2.3 Member Function Documentation

6.2.3.1 get_other_theory()

```
def plotting_util.AsymptoticConvergence.get_other_theory (
              self,
              ts )
```

get btc associated with other theory

```
   @param ts (iterable) list of physical times to get btc for

   @returns array of concentrations (btc) according to list of times
```

6.2.3.2 plot_btcs()

```
def plotting_util.AsymptoticConvergence.plot_btcs (
              self,
              ax )
```

plot breakthrough curves of numerical simulation and regular perturbation theory

```
   @param ax matplotlib axis for plotting
```

Reimplemented in plotting_util.TWConvergence.

6.2.3.3 plot_error()

```
def plotting_util.AsymptoticConvergence.plot_error (
              self,
              ax,
              plot_kwargs,
              line_kwargs = None )
```

plot errors as a function of ε

```
   @param ax matplotlib axis
   @param plot_kwargs kwargs for plotting
   @param line_kwargs kwargs for lines
```

Reimplemented in plotting_util.TWConvergence.

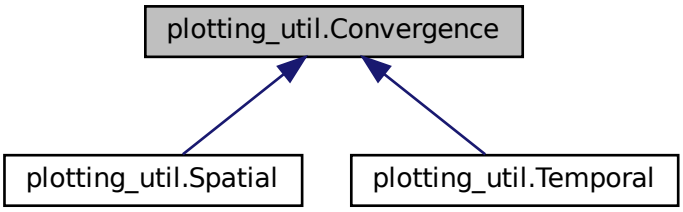
The documentation for this class was generated from the following file:

- src/plotting_util.py

6.3 plotting_util.Convergence Class Reference

Class for performing convergence analysis.

Inheritance diagram for plotting_util.Convergence:



Public Member Functions

- def __init__ (self, str file)

  Initialize the class.
- def plot (self, ax, dict symbol_kwargs, dict line_kwargs, label_line=False)

  plot the figure

Data Fields

- **kappa**
- **e**
- **ns**
- **ms**
- **tau_star**
- **du_inf**
- **ds**

6.3.1 Detailed Description

Class for performing convergence analysis.

Parameters

kappa	(float) value of Langmuir isotherm parameter κ
e	(float) value of ε
ns	(np.array) value of N for each calculation
ms	(np.array) value of M for each calculation
tau_star	(float) value of $\tau_\star$
du_inf	(np.array) infinity norm of error for each calculation

6.3.2 Constructor & Destructor Documentation

6.3.2.1 __init__()

```
def plotting_util.Convergence.__init__ (
              self,
             str file )
```

Initialize the class.

```
    @param file name of data file to analyze

    @return instance of the class
```

Reimplemented in plotting_util.Spatial, and plotting_util.Temporal.

6.3.3 Member Function Documentation

6.3.3.1 plot()

```
def plotting_util.Convergence.plot (
              self,
              ax,
             dict symbol_kwargs,
             dict line_kwargs,
              label_line = False )
```

plot the figure

```
    @param ax matplotlib axis
    @param symbol_kwargs key-word arguments for symbols
    @param line_kwargs line keyword arguments
    @param label_line whether or not to label line

    @return None
```

The documentation for this class was generated from the following file:

- src/plotting_util.py

6.4 ISO Struct Reference

Data Fields

- Fun F
- dFun dF
- int num_params
- double * params
- double * data

6.4.1 Field Documentation

6.4.1.1 data

```
double* ISO::data
```

data used for splines

6.4.1.2 dF

```
dFun ISO::dF
```

derivative of adsorption isotherm

6.4.1.3 F

```
Fun ISO::F
```

adsorption isotherm

6.4.1.4 num_params

```
int ISO::num_params
```

number of parameters used

6.4.1.5 params

```
double* ISO::params
```

parameters fit

The documentation for this struct was generated from the following file:

- src/isotherm.h

6.5 Isotherm Struct Reference

`#include <isotherm.h>`

6.5.1 Detailed Description

Used for adsorption isotherm

The documentation for this struct was generated from the following file:

- src/isotherm.h

6.6 isotherms.Langmuir Class Reference

Class for Langmuir isotherm.

Public Member Functions

- def __init__ (self, k)

  Initialize the class.
- def F (self, c)
- def F_prime (self, c)
- def s (self, c)
- def get_c_rarefaction (self, x, t)
- def shock_lambda (self)

  speed of shock wave
- def get_c_shock (self, x, t)
- def is_rarefaction_wave (self)
- def is_shock_wave (self)
- def get_c_local_equilibrium (self, x, t)

Data Fields

- **k**
- **name**

6.6.1 Detailed Description

Class for Langmuir isotherm.

Parameters

k	The value of κ chosen
name	The name of the isotherm

6.6.2 Constructor & Destructor Documentation

6.6.2.1 __init__()

```
def isotherms.Langmuir.__init__ (
              self,
              k )
```

Initialize the class.

```
   @param k The value of \f$\kappa\f$ chosen
   @param name The name of the isotherm

   @return An instance of the class with specified k
```

6.6.3 Member Function Documentation

6.6.3.1 F()

```
def isotherms.Langmuir.F (
              self,
              c )
```

Returns

$$L(c;k) = (1+\kappa)c/(1+\kappa c)$$

6.6.3.2 F_prime()

```
def isotherms.Langmuir.F_prime (
              self,
              c )
```

Returns

$$L'(c;k) = (1+\kappa)/(1+\kappa c)^2$$

6.6.3.3 get_c_local_equilibrium()

```
def isotherms.Langmuir.get_c_local_equilibrium (
              self,
              x,
              t )
```

Returns

$c(x,t)$ associated with local equilibrium

6.6.3.4 get_c_rarefaction()

```
def isotherms.Langmuir.get_c_rarefaction (
              self,
              x,
              t )
```

Returns

concentration $c(x,t)$ associated with rarefaction wave

6.6.3.5 get_c_shock()

```
def isotherms.Langmuir.get_c_shock (
              self,
              x,
              t )
```

Returns

concentration $c(x,t)$ associated with rarefaction wave

6.6.3.6 is_rarefaction_wave()

```
def isotherms.Langmuir.is_rarefaction_wave (
              self )
```

Returns

whether rarefaction wave or not

6.6.3.7 is_shock_wave()

```
def isotherms.Langmuir.is_shock_wave (
              self )
```

Returns

whether shock wave or not

6.6.3.8 s()

```
def isotherms.Langmuir.s (
              self,
              c )
```

Returns

speed at given concentration in x,t coordinate system

6.6.3.9 shock_lambda()

```
def isotherms.Langmuir.shock_lambda (
              self )
```

speed of shock wave

Returns

float representing speed of shock wave

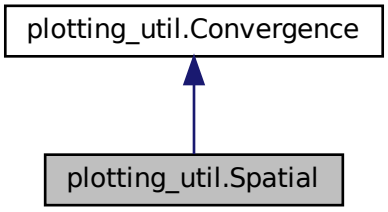
The documentation for this class was generated from the following file:

- src/isotherms.py

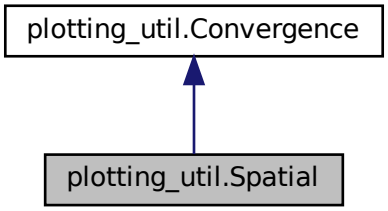
6.7 plotting_util.Spatial Class Reference

class for plotting temporal convergence

Inheritance diagram for plotting_util.Spatial:



Collaboration diagram for plotting_util.Spatial:



Public Member Functions

- def __init__ (self, str file)
  initialize the class

Data Fields

- **ds**
- **m**

6.7.1 Detailed Description

class for plotting temporal convergence

Parameters

m	(float or int?) number of M (time-step intervals), the same for each simulation

See also

Convergence

6.7.2 Constructor & Destructor Documentation

6.7.2.1 __init__()

```
def plotting_util.Spatial.__init__ (
              self,
             str file )
```

initialize the class

```
   @return instancs of class
```

Reimplemented from plotting_util.Convergence.

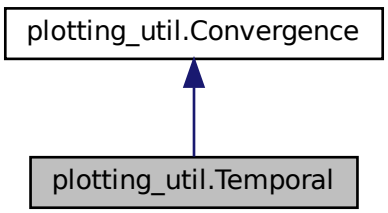
The documentation for this class was generated from the following file:

- src/plotting_util.py

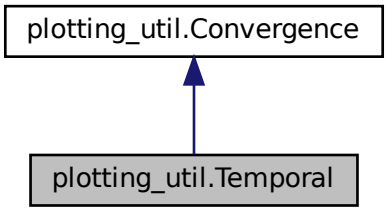
6.8 plotting_util.Temporal Class Reference

class for plotting temporal convergence

Inheritance diagram for plotting_util.Temporal:



Collaboration diagram for plotting_util.Temporal:



Public Member Functions

- def __init__ (self, str file)
  initialize the class

Data Fields

- **ds**
- **n**

6.8.1 Detailed Description

class for plotting temporal convergence

Parameters

n	(float or int?) number of N (spatial intervals), the same for each simulation

See also

Convergence

6.8.2 Constructor & Destructor Documentation

6.8.2.1 __init__()

```
def plotting_util.Temporal.__init__ (
              self,
             str file )
```

initialize the class

```
@return instancs of class
```

Reimplemented from plotting_util.Convergence.

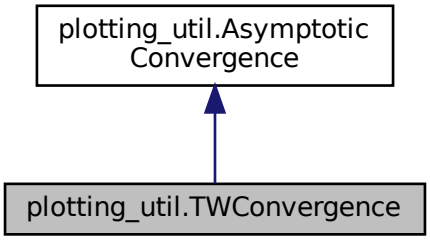
The documentation for this class was generated from the following file:

- src/plotting_util.py

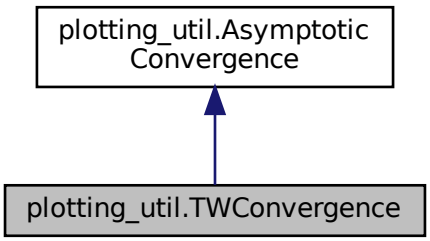
6.9 plotting_util.TWConvergence Class Reference

Class for plotting traveling wave convergence in epsilon (not N or M)

Inheritance diagram for plotting_util.TWConvergence:



Collaboration diagram for plotting_util.TWConvergence:



Public Member Functions

- def __init__ (self, Langmuir isotherm, file_prefix, tau_star)

  initialize the class
- def get_other_theory (self, t, eps)

  get btc associated with other theory
- def plot_btcs (self, ax)

  plot breakthrough curves
- def plot_error (self, ax, plot_kwargs, line_kwargs=None)

  plot error

Additional Inherited Members

6.9.1 Detailed Description

Class for plotting traveling wave convergence in epsilon (not N or M)

Parameters

isotherm	(Langmuir) type of isotherm
klasses	array of solutions to store
es	array of epsilons, hard-coded to 0.02, 0.04, 0.08, 0.16, 0.32, 0.64
colors,list	of colors to plot, hard-coded to plt.cm.viridis_r
tau_star,total	dimensionless time to simulate

6.9.2 Constructor & Destructor Documentation

6.9.2.1 __init__()

```
def plotting_util.TWConvergence.__init__ (
              self,
             Langmuir isotherm,
              file_prefix,
              tau_star )
```

initialize the class

```
   @param isotherm isotherm to use
   @param file_prefix (str) prefix for looking at files
   @param tau_star (float) total dimensionless time to simulate

   @returns instance of class
```

Reimplemented from plotting_util.AsymptoticConvergence.

6.9.3 Member Function Documentation

6.9.3.1 get_other_theory()

```
def plotting_util.TWConvergence.get_other_theory (
              self,
              t,
              eps )
```

get btc associated with other theory

```
   @param t (np.array) list of physical times to get btc for
   @param eps (float) value of $\f\varepsilon\f$ for calculation

   @returns array of concentrations (btc) according to list of times
```

6.9.3.2 plot_btcs()

```
def plotting_util.TWConvergence.plot_btcs (
              self,
              ax )
```

plot breakthrough curves

```
   @param ax matplotlib axis
   @return none
```

Reimplemented from plotting_util.AsymptoticConvergence.

6.9.3.3 plot_error()

```
def plotting_util.TWConvergence.plot_error (
              self,
              ax,
              plot_kwargs,
              line_kwargs = None )
```

plot error

Parameters

ax	matplotlib axis
plot_kwargs	kws for plotting
line_kwargs	kws for line, defaults to None

Returns

none

Reimplemented from plotting_util.AsymptoticConvergence.

The documentation for this class was generated from the following file:

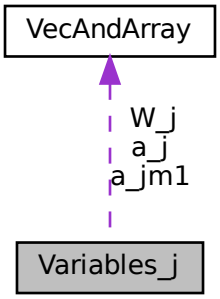
- src/plotting_util.py

6.10 Variables_j Struct Reference

Analytical and numerical variables at time j (and jm1 if analytical)

`#include <my_petsc.h>`

Collaboration diagram for Variables_j:



Data Fields

- VecAndArray W_j
- VecAndArray a_j
- VecAndArray a_jm1
- Vec error

6.10.1 Detailed Description

Analytical and numerical variables at time j (and jm1 if analytical)

6.10.2 Field Documentation

6.10.2.1 a_j

`VecAndArray Variables_j::a_j`

Analytical values of fluid concentration at time j

6.10.2.2 a_jm1

`VecAndArray Variables_j::a_jm1`

Analytical values of fluid concentration at time $j-1$

6.10.2.3 error

`Vec Variables_j::error`

vector comprising errors (presumably b/t analytical and numerical)

6.10.2.4 W_j

`VecAndArray Variables_j::W_j`

Numerical values of fluid concentration at time j

The documentation for this struct was generated from the following file:

- src/my_petsc.h

6.11 VecAndArray Struct Reference

`#include <my_petsc.h>`

Data Fields

- Vec vec
- PetscReal * arr

6.11.1 Detailed Description

Used for converting between vector and array types

6.11.2 Field Documentation

6.11.2.1 arr

```
PetscReal* VecAndArray::arr
```

array part

6.11.2.2 vec

```
Vec VecAndArray::vec
```

vector part

The documentation for this struct was generated from the following file:

- src/my_petsc.h

Chapter 7

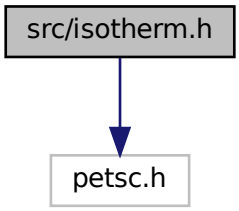
File Documentation

7.1 src/isotherm.h File Reference

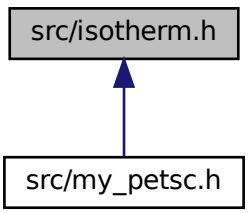
Functions and structs for adsorption isotherm.

```
#include <petsc.h>
```

Include dependency graph for isotherm.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct ISO

Typedefs

- typedef double(∗ Fun) (double c, int num_params, double ∗params, double ∗data)
  F.
- typedef double(∗ dFun) (double c, int num_params, double ∗params, double ∗data)
  dF

Functions

- void initISOParams (ISO ∗isotherm)
  allocate memory for isotherm parameters array
- void freeISOParams (ISO ∗isotherm)
  free isotherm parameters array
- double F_langmuir_dimensionless (double c, int num_params, double ∗params, double ∗data)
  evaluate adsorption isotherm
- double dF_langmuir_dimensionless (double c, int num_params, double ∗params, double ∗data)
  evaluate derivative adsorption isotherm
- void initialize_langmuir_dimensionless (ISO ∗langmuir, double kappa)
  initialize langmuir struct

7.1.1 Detailed Description

Functions and structs for adsorption isotherm.

Author

Robert F. DeJaco

7.1.2 Typedef Documentation

7.1.2.1 dFun

```
typedef double(* dFun) (double c, int num_params, double *params, double *data)
```

dF

Function pointer used for isotherm derivative

7.1.2.2 Fun

```
typedef double(* Fun) (double c, int num_params, double *params, double *data)
```

F.

Function pointer used for isotherm function

7.1.3 Function Documentation

7.1.3.1 dF_langmuir_dimensionless()

```
double dF_langmuir_dimensionless (
            double c,
            int num_params,
            double * params,
            double * data )
```

evaluate derivative adsorption isotherm

Evaluates

$$\frac{1+\kappa}{(1+\kappa c)^2}$$

where $\kappa = p[0]$ and p is the parameters array.

Returns

value of adsorption isotherm

7.1.3.2 F_langmuir_dimensionless()

```
double F_langmuir_dimensionless (
            double c,
            int num_params,
            double * params,
            double * data )
```

evaluate adsorption isotherm

Evaluates

$$\frac{(1+\kappa)\,c}{1+\kappa c}$$

where $\kappa = p[0]$ and p is the parameters array.

Returns

value of adsorption isotherm

7.1.3.3 initialize_langmuir_dimensionless()

```
void initialize_langmuir_dimensionless (
            ISO * langmuir,
            double kappa )
```

initialize langmuir struct

Initializes langmuir struct. Sets number of params to be 1, initializes parameters array, sets F and dF to point to F_langmuir and dF_Langmuir.

7.2 src/my_funcs.hpp File Reference

For calculating analytical solutions.

Functions

- double exp_I0 (double T, double z)

  Calculate exponential term involving bessel function.
- double integrate (double T_jm1, double T_j, double z)

  Integrate expI0 from T_jm1 to T_j.
- double update_solution (double ∗c_j, double ∗a_jm1, double T_jm1, double T_j, double ∗X, int n)

  Updates solution from previous time step.
- void analytical_btc (double ∗, double ∗, double X, int)

  Provides analytical break-through curve.

7.2.1 Detailed Description

For calculating analytical solutions.

Used to calculate analytical solutions or break-through curves

Author

Robert F. DeJaco

7.2.2 Function Documentation

7.2.2.1 analytical_btc()

```
void analytical_btc (
            double * c,
            double * T,
            double X,
            int m )
```

Provides analytical break-through curve.

Parameters

c	fluid concentration
T	times
X	value for distance (1/epsilon)
m	number of times including 0

Returns

void

7.2.2.2 exp_I0()

```
double exp_I0 (
            double T,
            double z )
```

Calculate exponential term involving bessel function.

Evaluates $e^{-z-T} I_0(2\sqrt{zT})$, where I_0 is the modified Bessel function of zeroth order.

Parameters

T	Scaled value for time
z	Scaled value for distance

Returns

value of expression

7.2.2.3 integrate()

```
double integrate (
            double T_jm1,
            double T_j,
            double z )
```

Integrate expI0 from T_jm1 to T_j.

Evaluates

$$\int_{T_{j-1}}^{T_j} e^{-z-t} I_0\left(2\sqrt{zt}\right)\, dt$$

See also

exp_I0

Parameters

T_jm1	scaled time at previous index j-1
T_j	scaled time at current/next index j
z	scaled distance

Returns

value of expression

7.2.2.4 update_solution()

```
double update_solution (
            double * c_j,
            double * a_jm1,
            double T_jm1,
            double T_j,
            double * X,
            int n )
```

Updates solution from previous time step.

Parameters

c_j	new solution at time j
a_jm1	old solution at time j-1
T_j	current/new value of time
T_jm1	previous value of time
X	distance along column
n	last index of distance array

Returns

void

Warning

{not tested, probably uses wrong value of n}

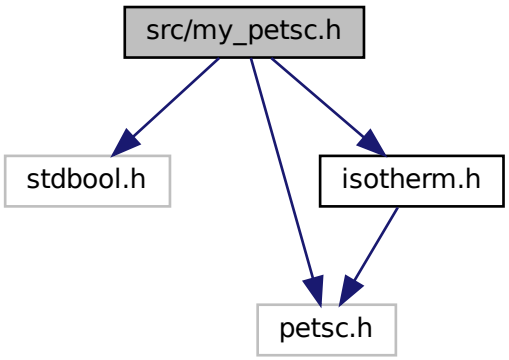
7.3 src/my_petsc.h File Reference

Functions and structs for solving PDE.

```
#include <stdbool.h>
#include <petsc.h>
```

```
#include "isotherm.h"
```

Include dependency graph for my_petsc.h:



Data Structures

- struct VecAndArray
- struct AppCtx

  Useful parameters for solving, Application context.
- struct Variables_j

  Analytical and numerical variables at time j (and jm1 if analytical)

Functions

- void print_user_params (AppCtx u)

  print user params associated with application
- PetscErrorCode InitializeVecAndArray (DM da, VecAndArray *x)

  Initialize Struct containing both vector and array.
- PetscErrorCode DestroyVecAndArray (DM da, VecAndArray *x)

  Destroy Struct containing both vector and array.
- PetscErrorCode one_time_step (SNES snes, DMDALocalInfo *p_info, AppCtx *p_user, Variables_j *p_vars)

  Perform one time step.
- PetscErrorCode simulate (DM da, SNES snes, DMDALocalInfo *p_info, AppCtx *p_user)

  Perform calculations at all time steps.
- PetscErrorCode run (int argc, char **args, int m, double eps, double tau_star, double theta, ISO *isotherm, bool compute_analytical, double *btc, bool make_movie, bool print_header)

  Main function called.
- PetscErrorCode UpdateAnalytical (DMDALocalInfo *info, PetscReal *a_j, PetscReal *a_jm1, AppCtx *user)

  Updates analytical solution.
- PetscErrorCode UpdateSolid (DMDALocalInfo *info, PetscReal *W_j, AppCtx *user)

  Update solid concentration.
- PetscErrorCode UpdateIsotherm (DMDALocalInfo *info, PetscReal *W, AppCtx *user)

  Update value of adsorption isotherm.

- PetscReal eval_S_i_j (PetscReal f_j, PetscReal f_jm1, PetscReal exp_mdT, PetscReal S_jm1, PetscReal dT)
  evaluate value of solid concentration
- PetscReal eval_dS_i_j (PetscReal df_i_j, PetscReal dT)
  evaluates gradient in solid concentration wrt fluid
- PetscReal theta_rule (PetscReal val_i, PetscReal val_im1, PetscReal theta)
  does theta rule for discretization in space
- PetscErrorCode FormFunctionLocal (DMDALocalInfo *info, PetscReal *W_j, PetscReal *FF, AppCtx *user)
  residuals of nonlinear equations
- PetscErrorCode FormJacobianLocal (DMDALocalInfo *info, PetscReal *W_j, Mat J, Mat P, AppCtx *user)
  jacobian of nonlinear equations
- PetscErrorCode CalculateOmega (DMDALocalInfo *info, PetscReal *W, AppCtx *user)
  calculate parameter to determine if at steady state

7.3.1 Detailed Description

Functions and structs for solving PDE.

Author

Robert F. DeJaco

7.3.2 Function Documentation

7.3.2.1 CalculateOmega()

```
PetscErrorCode CalculateOmega (
            DMDALocalInfo * info,
            PetscReal * W,
            AppCtx * user )
```

calculate parameter to determine if at steady state

Parameter used to determine if simulation is at steady state

Returns

void

Parameters

	info	grid object
in	*W*	solution
in,out	*user*	stores attribute omega

7.3.2.2 DestroyVecAndArray()

```
PetscErrorCode DestroyVecAndArray (
            DM da,
            VecAndArray * x )
```

Destroy Struct containing both vector and array.

See also

VecAndArray

Parameters

in	*da*	manages an abstract grid object and its interactions with the algebraic solvers
	x	Vector and array to destroy

7.3.2.3 eval_dS_i_j()

```
PetscReal eval_dS_i_j (
            PetscReal df_i_j,
            PetscReal dT )
```

evaluates gradient in solid concentration wrt fluid

returns gradient, $\Delta\tau F'/2/\varepsilon$

Parameters

in	*df_i_j*	gradient in isotherm
in	*dT*	scaled time step, likely $\Delta\tau/\varepsilon$

7.3.2.4 eval_S_i_j()

```
PetscReal eval_S_i_j (
            PetscReal f_j,
            PetscReal f_jm1,
            PetscReal exp_mdT,
            PetscReal S_jm1,
            PetscReal dT )
```

evaluate value of solid concentration

Returns

value of solid concentration S_i^j

Parameters

f_j	value of adsorption isotherm at i and j
f_jm1	value of adsorption isotherm at i but j - 1
exp_mdT	value for $e^{-\Delta\tau/\varepsilon}$
S_jm1	Value for solid concentration at i but j - 1
dT	value for $\Delta\tau/\varepsilon$

7.3.2.5 FormFunctionLocal()

```
PetscErrorCode FormFunctionLocal (
            DMDALocalInfo * info,
            PetscReal * W_j,
            PetscReal * FF,
            AppCtx * user )
```

residuals of nonlinear equations

Returns

void

Parameters

info	Abstract object with grid info
W_j	Solution, used to evaluate residuals
FF	residuals or form function
user	application context

7.3.2.6 FormJacobianLocal()

```
PetscErrorCode FormJacobianLocal (
            DMDALocalInfo * info,
            PetscReal * W_j,
            Mat J,
            Mat P,
            AppCtx * user )
```

jacobian of nonlinear equations

Returns

void

Parameters

info	Abstract object with grid info
W_j	Solution, used to evaluate residuals
J	not used?
P	jacobian
user	application context

7.3.2.7 InitializeVecAndArray()

```
PetscErrorCode InitializeVecAndArray (
            DM da,
            VecAndArray * x )
```

Initialize Struct containing both vector and array.

See also

VecAndArray

Parameters

in	*da*	manages an abstract grid object and its interactions with the algebraic solvers
	x	Vector and array to initialize

7.3.2.8 one_time_step()

```
PetscErrorCode one_time_step (
            SNES snes,
            DMDALocalInfo * p_info,
            AppCtx * p_user,
            Variables_j * p_vars )
```

Perform one time step.

Parameters

in	*snes*	Solver
in	*p_info*	Information about some parameters needed
in,out	*p_user*	Application context
out	*p_vars*	Variables at time j, updated for next time step

7.3.2.9 run()

```
PetscErrorCode run (
            int argc,
            char ** args,
            int m,
            double eps,
            double tau_star,
            double theta,
            ISO * isotherm,
            bool compute_analytical,
            double * btc,
            bool make_movie,
            bool print_header )
```

Main function called.

Parameters

in	*argc*	number of command arguments
in	*args*	command arguments
in	*m*	number of time steps
in	*eps*	value of epsilon ε
in	*tau_star*	total simulation time $\tau_\star$
in	*theta*	fraction of bfd/ffd
in	*isotherm*	adsorption isotherm struct. Needs to be allocated/initialized beforehand
in	*compute_analytical*	whether or not to compute analytical solution
in,out	*btc*	values of break-through curve
in	*make_movie*	whether or not to make movie
in	*print_header*	whether or not to print header

7.3.2.10 simulate()

```
PetscErrorCode simulate (
            DM da,
            SNES snes,
            DMDALocalInfo * p_info,
            AppCtx * p_user )
```

Perform calculations at all time steps.

Parameters

da	! Abstract grid object
snes	! solver
p_info	! local info on grid
p_user	User Application context

7.3.2.11 theta_rule()

```
PetscReal theta_rule (
            PetscReal val_i,
            PetscReal val_im1,
            PetscReal theta )
```

does theta rule for discretization in space

Returns

value of expression averaged wrt theta, $v_i(1-\theta)+v_{i-1}\theta$ for some v_i, v_{i-1}

Note

that in paper theta=0.5 is always used

Parameters

val_i	value at spatial index i
val_im1	value at spatial index i - 1
theta	value for $0 \leq \theta \leq 1$

7.3.2.12 UpdateAnalytical()

```
PetscErrorCode UpdateAnalytical (
            DMDALocalInfo * info,
            PetscReal * a_j,
            PetscReal * a_jm1,
            AppCtx * user )
```

Updates analytical solution.

Updates analytical solution for time j based off of info for $j-1$.

Parameters

in	*info*	information about local grid
in,out	*a_j*	new solution computed
in	*a_jm1*	old solution
in	*user*	Application context

7.3.2.13 UpdateIsotherm()

```
PetscErrorCode UpdateIsotherm (
            DMDALocalInfo * info,
```

```
            PetscReal * W,
            AppCtx * user )
```

Update value of adsorption isotherm.

Parameters

in	*info*	local information about grid
in	*W*	Solution of PDE at some value of time
in,out	*user*	application context. new values of isotherm stored here

7.3.2.14 UpdateSolid()

```
PetscErrorCode UpdateSolid (
            DMDALocalInfo * info,
            PetscReal * W_j,
            AppCtx * user )
```

Update solid concentration.

Parameters

in	*info*	information about grid
in	*W_j*	Numerical solution of fluid concentration
in	*user*	application context

Index