

# Understanding Breakthrough Curves

v0.3

Generated by Doxygen 1.9.1



<b>1 Namespace Index</b>	<b>1</b>
1.1 Namespace List	1
<b>2 Hierarchical Index</b>	<b>3</b>
2.1 Class Hierarchy	3
<b>3 Data Structure Index</b>	<b>5</b>
3.1 Data Structures	5
<b>4 File Index</b>	<b>7</b>
4.1 File List	7
<b>5 Namespace Documentation</b>	<b>9</b>
5.1 isotherms Namespace Reference	9
5.1.1 Detailed Description	9
5.2 plotting_util Namespace Reference	9
5.2.1 Detailed Description	10
5.2.2 Function Documentation	10
5.2.2.1 calculate_slope_error()	10
5.2.2.2 get_u_TW_Langmuir()	10
5.2.2.3 save_figure()	11
<b>6 Data Structure Documentation</b>	<b>13</b>
6.1 AppCtx Struct Reference	13
6.1.1 Detailed Description	14
6.1.2 Field Documentation	14
6.1.2.1 b	14
6.1.2.2 beta	14
6.1.2.3 btc	14
6.1.2.4 compute_analytical	14
6.1.2.5 dtau	15
6.1.2.6 dtau_est	15
6.1.2.7 dx	15
6.1.2.8 eps	15
6.1.2.9 errnorm_all	15
6.1.2.10 f_jm1	15
6.1.2.11 isotherm	15
6.1.2.12 j	15
6.1.2.13 K	16
6.1.2.14 kappa	16
6.1.2.15 m	16
6.1.2.16 make_movie	16
6.1.2.17 movie_step_interval	16
6.1.2.18 omega	16

6.1.2.19 print_header . . . . .	16
6.1.2.20 S_jm1 . . . . .	16
6.1.2.21 save_btc . . . . .	17
6.1.2.22 tau_star . . . . .	17
6.1.2.23 theta . . . . .	17
6.1.2.24 viewer . . . . .	17
6.2 plotting_util.AsymptoticConvergence Class Reference . . . . .	17
6.2.1 Detailed Description . . . . .	18
6.2.2 Constructor & Destructor Documentation . . . . .	18
6.2.2.1 __init__() . . . . .	18
6.2.3 Member Function Documentation . . . . .	19
6.2.3.1 get_other_theory() . . . . .	19
6.2.3.2 plot_btcs() . . . . .	19
6.2.3.3 plot_error() . . . . .	19
6.3 plotting_util.Convergence Class Reference . . . . .	20
6.3.1 Detailed Description . . . . .	20
6.3.2 Constructor & Destructor Documentation . . . . .	21
6.3.2.1 __init__() . . . . .	21
6.3.3 Member Function Documentation . . . . .	21
6.3.3.1 plot() . . . . .	21
6.4 ISO Struct Reference . . . . .	22
6.4.1 Field Documentation . . . . .	22
6.4.1.1 data . . . . .	22
6.4.1.2 dF . . . . .	22
6.4.1.3 F . . . . .	22
6.4.1.4 num_params . . . . .	22
6.4.1.5 params . . . . .	22
6.5 Isotherm Struct Reference . . . . .	23
6.5.1 Detailed Description . . . . .	23
6.6 isotherms.Langmuir Class Reference . . . . .	23
6.6.1 Detailed Description . . . . .	23
6.6.2 Constructor & Destructor Documentation . . . . .	24
6.6.2.1 __init__() . . . . .	24
6.6.3 Member Function Documentation . . . . .	24
6.6.3.1 F() . . . . .	24
6.6.3.2 F_prime() . . . . .	24
6.6.3.3 get_c_local_equilibrium() . . . . .	25
6.6.3.4 get_c_rarefaction() . . . . .	25
6.6.3.5 get_c_shock() . . . . .	25
6.6.3.6 is_rarefaction_wave() . . . . .	25
6.6.3.7 is_shock_wave() . . . . .	26
6.6.3.8 s() . . . . .	26

6.6.3.9 shock_lambda()	26
6.7 plotting_util.Spatial Class Reference	27
6.7.1 Detailed Description	27
6.7.2 Constructor & Destructor Documentation	28
6.7.2.1 __init__()	28
6.8 plotting_util.Temporal Class Reference	28
6.8.1 Detailed Description	29
6.8.2 Constructor & Destructor Documentation	29
6.8.2.1 __init__()	29
6.9 plotting_util.TWConvergence Class Reference	30
6.9.1 Detailed Description	31
6.9.2 Constructor & Destructor Documentation	31
6.9.2.1 __init__()	31
6.9.3 Member Function Documentation	32
6.9.3.1 get_other_theory()	32
6.9.3.2 plot_btcs()	32
6.9.3.3 plot_error()	32
6.10 Variables_j Struct Reference	33
6.10.1 Detailed Description	33
6.10.2 Field Documentation	33
6.10.2.1 a_j	34
6.10.2.2 a_jm1	34
6.10.2.3 error	34
6.10.2.4 W_j	34
6.11 VecAndArray Struct Reference	34
6.11.1 Detailed Description	34
6.11.2 Field Documentation	34
6.11.2.1 arr	35
6.11.2.2 vec	35
<b>7 File Documentation</b>	<b>37</b>
7.1 src/isotherm.h File Reference	37
7.1.1 Detailed Description	38
7.1.2 Typedef Documentation	38
7.1.2.1 dFun	38
7.1.2.2 Fun	39
7.1.3 Function Documentation	39
7.1.3.1 dF_langmuir_dimensionless()	39
7.1.3.2 F_langmuir_dimensionless()	39
7.1.3.3 initialize_langmuir_dimensionless()	40
7.2 src/my_funcs.hpp File Reference	40
7.2.1 Detailed Description	40

---

7.2.2 Function Documentation	40
7.2.2.1 analytical_btc()	40
7.2.2.2 exp_I0()	41
7.2.2.3 integrate()	41
7.2.2.4 update_solution()	42
7.3 src/my_petsc.h File Reference	42
7.3.1 Detailed Description	44
7.3.2 Function Documentation	44
7.3.2.1 CalculateOmega()	44
7.3.2.2 DestroyVecAndArray()	45
7.3.2.3 eval_dS_i_j()	45
7.3.2.4 eval_S_i_j()	45
7.3.2.5 FormFunctionLocal()	46
7.3.2.6 FormJacobianLocal()	46
7.3.2.7 InitializeVecAndArray()	47
7.3.2.8 one_time_step()	47
7.3.2.9 run()	48
7.3.2.10 simulate()	48
7.3.2.11 theta_rule()	49
7.3.2.12 UpdateAnalytical()	49
7.3.2.13 UpdateIsotherm()	49
7.3.2.14 UpdateSolid()	50
<b>Index</b>	<b>51</b>

# Chapter 1

## Namespace Index

### 1.1 Namespace List

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

<a href="#">isotherms</a>	Defines the <a href="#">Langmuir</a> isotherm class . . . . .	9
<a href="#">plotting_util</a>	Utilities for making plots . . . . .	9





## Chapter 2

# Hierarchical Index

### 2.1 Class Hierarchy

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

AppCtx . . . . .	13
plotting_util.AsymptoticConvergence . . . . .	17
plotting_util.TWConvergence . . . . .	30
plotting_util.Convergence . . . . .	20
plotting_util.Spatial . . . . .	27
plotting_util.Temporal . . . . .	28
ISO . . . . .	22
Isotherm . . . . .	23
isotherms.Langmuir . . . . .	23
Variables_j . . . . .	33
VecAndArray . . . . .	34



## Chapter 3

# Data Structure Index

### 3.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">AppCtx</a>	Useful parameters for solving, Application context . . . . .	13
<a href="#">plotting_util.AsymptoticConvergence</a>	Class for plotting convergence in epsilon (not N or M) . . . . .	17
<a href="#">plotting_util.Convergence</a>	Class for performing convergence analysis . . . . .	20
<a href="#">ISO</a>	. . . . .	22
<a href="#">Isotherm</a>	. . . . .	23
<a href="#">isotherms.Langmuir</a>	Class for <a href="#">Langmuir</a> isotherm . . . . .	23
<a href="#">plotting_util.Spatial</a>	Class for plotting temporal convergence . . . . .	27
<a href="#">plotting_util.Temporal</a>	Class for plotting temporal convergence . . . . .	28
<a href="#">plotting_util.TWConvergence</a>	Class for plotting traveling wave convergence in epsilon (not N or M) . . . . .	30
<a href="#">Variables_j</a>	Analytical and numerical variables at time j (and jm1 if analytical) . . . . .	33
<a href="#">VecAndArray</a>	. . . . .	34



## Chapter 4

# File Index

### 4.1 File List

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

<a href="#">src/isortherm.h</a>	Functions and structs for adsorption isotherm . . . . .	37
<a href="#">src/my_funcs.hpp</a>	For calculating analytical solutions . . . . .	40
<a href="#">src/my_petsc.h</a>	Functions and structs for solving PDE . . . . .	42



## 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

<i>x</i>	np array of x-values of plot
<i>y</i>	numpy array of y-values of plot
<i>slope</i>	float representing slope calculated
<i>intercept</i>	float representing intercept calculated
<i>conf</i>	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

<i>xi</i>	(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

<i>fig</i>	(matplotlib.pyplot.figure) instance of figure to be saved
<i>name</i>	(str) name of figure to be saved



## Chapter 6

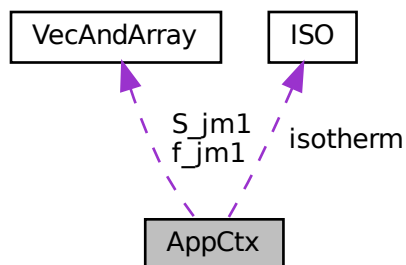
# Data Structure Documentation

### 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  $\varepsilon$

#### 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  $\tau_*$

### 6.1.2.23 theta

```
PetscReal AppCtx::theta
```

fraction of BFD/FFD

### 6.1.2.24 viewer

```
PetscViewer AppCtx::viewer
```

petsc viewer

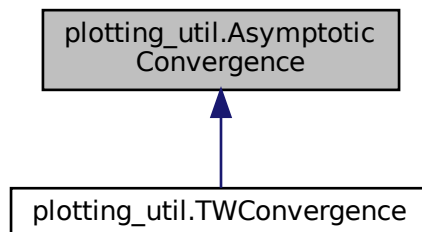
The documentation for this struct was generated from the following file:

- [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  $\varepsilon$*

## Data Fields

- **isotherm**
- **klases**
- **es**
- **colors**
- **file\_prefix**
- **tau\_star**

### 6.2.1 Detailed Description

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

#### Parameters

<i>isotherm</i>	(Langmuir) type of isotherm
<i>klases</i>	array of solutions to store
<i>es</i>	array of epsilons, hard-coded to 0.02, 0.04, 0.08, 0.16, 0.32, 0.64
<i>colors, list</i>	of colors to plot, hard-coded to plt.cm.viridis_r
<i>tau_star, total</i>	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  $\varepsilon$

```

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

```

Reimplemented in [plotting\\_util.TWConvergence](#).

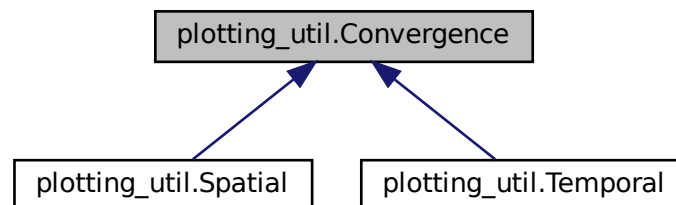
The documentation for this class was generated from the following file:

- 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

<i>kappa</i>	(float) value of Langmuir isotherm parameter $\kappa$
<i>e</i>	(float) value of $\varepsilon$
<i>ns</i>	(np.array) value of $N$ for each calculation
<i>ms</i>	(np.array) value of $M$ for each calculation
<i>tau_star</i>	(float) value of $\tau_*$
<i>du_inf</i>	(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

<i>k</i>	The value of $\kappa$ chosen
<i>name</i>	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:

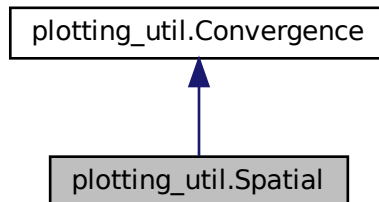
- src/isotherms.py



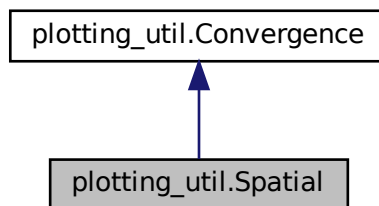
## 6.7 plotting\_util.Spatial Class Reference

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**

<i>m</i>	(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 instances of class
```

Reimplemented from [plotting\\_util.Convergence](#).

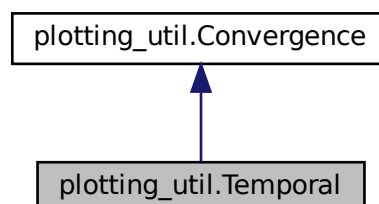
The documentation for this class was generated from the following file:

- `src/plotting_util.py`

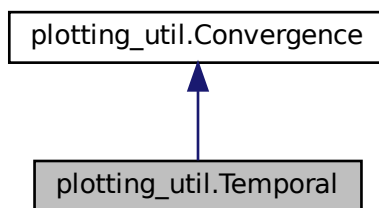
## 6.8 `plotting_util.Temporal` Class Reference

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

<code>n</code>	(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 instances of class
```

Reimplemented from [plotting\\_util.Convergence](#).

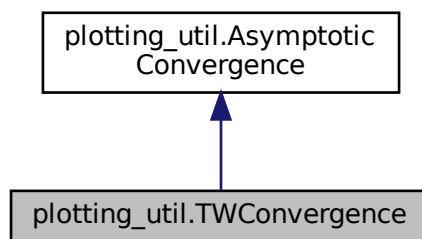
The documentation for this class was generated from the following file:

- `src/plotting_util.py`

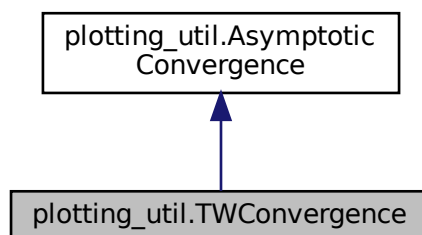
## 6.9 plotting\_util.TWConvergence Class Reference

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

<i>isotherm</i>	(Langmuir) type of isotherm
<i>klases</i>	array of solutions to store
<i>es</i>	array of epsilons, hard-coded to 0.02, 0.04, 0.08, 0.16, 0.32, 0.64
<i>colors,list</i>	of colors to plot, hard-coded to plt.cm.viridis_r
<i>tau_star,total</i>	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  $\epsilon$  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

<i>ax</i>	matplotlib axis
<i>plot_kwargs</i>	kws for plotting
<i>line_kwargs</i>	kws for line, defaults to None

**Returns**

none

Reimplemented from [plotting\\_util.AsymptoticConvergence](#).

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

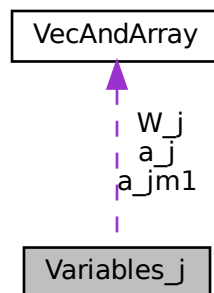
- `src/plotting_util.py`

## 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

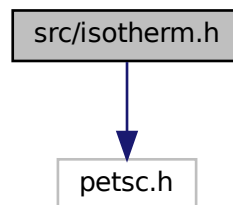
# File Documentation

### 7.1 src/isotherm.h File Reference

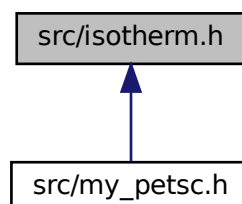
Functions and structs for adsorption isotherm.

```
#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

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

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**

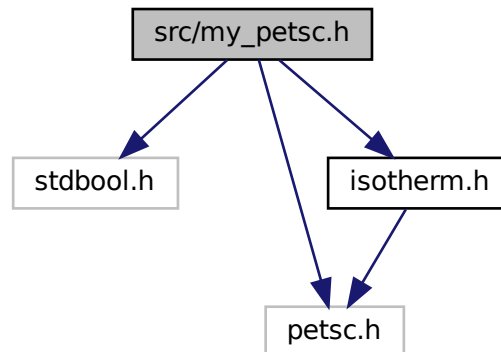
Functions and structs for solving PDE.

```
#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

	<i>info</i>	grid object
in	<i>W</i>	solution
in, out	<i>user</i>	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	<i>da</i>	manages an abstract grid object and its interactions with the algebraic solvers
	<i>x</i>	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	$df_{\leftrightarrow i_j}$	gradient in isotherm
in	<i>dT</i>	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 *  $\bar{W}_j$ ,
    PetscReal * FF,
    AppCtx * user )
```

residuals of nonlinear equations

## Returns

void

## Parameters

<i>info</i>	Abstract object with grid info
$\bar{W}_j$	Solution, used to evaluate residuals
<i>FF</i>	residuals or form function
<i>user</i>	application context

## 7.3.2.6 FormJacobianLocal()

```
PetscErrorCode FormJacobianLocal (
    DMDALocalInfo * info,
    PetscReal *  $\bar{W}_j$ ,
    Mat J,
    Mat P,
    AppCtx * user )
```

jacobian of nonlinear equations

## Returns

void

## Parameters

<i>info</i>	Abstract object with grid info
$W \leftarrow j$	Solution, used to evaluate residuals
<i>J</i>	not used?
<i>P</i>	jacobian
<i>user</i>	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	<i>da</i>	manages an abstract grid object and its interactions with the algebraic solvers
	<i>x</i>	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	<i>snes</i>	Solver
in	<i>p_info</i>	Information about some parameters needed
in, out	<i>p_user</i>	Application context
out	<i>p_vars</i>	Variables at time <i>j</i> , 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	<i>argc</i>	number of command arguments
in	<i>args</i>	command arguments
in	<i>m</i>	number of time steps
in	<i>eps</i>	value of epsilon $\varepsilon$
in	<i>tau_star</i>	total simulation time $\tau_*$
in	<i>theta</i>	fraction of bfd/ffd
in	<i>isotherm</i>	adsorption isotherm struct. Needs to be allocated/initialized beforehand
in	<i>compute_analytical</i>	whether or not to compute analytical solution
in, out	<i>btc</i>	values of break-through curve
in	<i>make_movie</i>	whether or not to make movie
in	<i>print_header</i>	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

<i>da</i>	! Abstract grid object
<i>snes</i>	! solver
<i>p_info</i>	! local info on grid
<i>p_user</i>	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

<i>val_i</i>	value at spatial index i
<i>val_im1</i>	value at spatial index i - 1
<i>theta</i>	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	<i>info</i>	information about local grid
in, out	<i>a_j</i>	new solution computed
in	<i>a_jm1</i>	old solution
in	<i>user</i>	Application context

### 7.3.2.13 UpdateIsotherm()

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

```
PetscReal *  $\bar{W}$ ,
AppCtx * user )
```

Update value of adsorption isotherm.

#### Parameters

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

#### 7.3.2.14 UpdateSolid()

```
PetscErrorCode UpdateSolid (
    DMDALocalInfo * info,
    PetscReal *  $\bar{W}_j$ ,
    AppCtx * user )
```

Update solid concentration.

#### Parameters

in	<i>info</i>	information about grid
in	$W_{\leftrightarrow j}$	Numerical solution of fluid concentration
in	<i>user</i>	application context



# Index

- `__init__`
    - `isotherms.Langmuir`, [24](#)
    - `plotting_util.AsymptoticConvergence`, [18](#)
    - `plotting_util.Convergence`, [21](#)
    - `plotting_util.Spatial`, [28](#)
    - `plotting_util.Temporal`, [29](#)
    - `plotting_util.TWConvergence`, [31](#)
- `a_j`
  - `Variables_j`, [33](#)
- `a_jm1`
  - `Variables_j`, [34](#)
- `analytical_btc`
  - `my_funcs.hpp`, [40](#)
- `AppCtx`, [13](#)
  - `b`, [14](#)
  - `beta`, [14](#)
  - `btc`, [14](#)
  - `compute_analytical`, [14](#)
  - `dtau`, [14](#)
  - `dtau_est`, [15](#)
  - `dx`, [15](#)
  - `eps`, [15](#)
  - `errnorm_all`, [15](#)
  - `f_jm1`, [15](#)
  - `isotherm`, [15](#)
  - `j`, [15](#)
  - `K`, [15](#)
  - `kappa`, [16](#)
  - `m`, [16](#)
  - `make_movie`, [16](#)
  - `movie_step_interval`, [16](#)
  - `omega`, [16](#)
  - `print_header`, [16](#)
  - `S_jm1`, [16](#)
  - `save_btc`, [16](#)
  - `tau_star`, [17](#)
  - `theta`, [17](#)
  - `viewer`, [17](#)
- `arr`
  - `VecAndArray`, [34](#)
- `b`
  - `AppCtx`, [14](#)
- `beta`
  - `AppCtx`, [14](#)
- `btc`
  - `AppCtx`, [14](#)
- `calculate_slope_error`
  - `plotting_util`, [10](#)
- `CalculateOmega`
  - `my_petsc.h`, [44](#)
- `compute_analytical`
  - `AppCtx`, [14](#)
- `data`
  - `ISO`, [22](#)
- `DestroyVecAndArray`
  - `my_petsc.h`, [44](#)
- `dF`
  - `ISO`, [22](#)
- `dF_langmuir_dimensionless`
  - `isotherm.h`, [39](#)
- `dFun`
  - `isotherm.h`, [38](#)
- `dtau`
  - `AppCtx`, [14](#)
- `dtau_est`
  - `AppCtx`, [15](#)
- `dx`
  - `AppCtx`, [15](#)
- `eps`
  - `AppCtx`, [15](#)
- `errnorm_all`
  - `AppCtx`, [15](#)
- `error`
  - `Variables_j`, [34](#)
- `eval_dS_i_j`
  - `my_petsc.h`, [45](#)
- `eval_S_i_j`
  - `my_petsc.h`, [45](#)
- `exp_l0`
  - `my_funcs.hpp`, [41](#)
- `F`
  - `ISO`, [22](#)
  - `isotherms.Langmuir`, [24](#)
- `f_jm1`
  - `AppCtx`, [15](#)
- `F_langmuir_dimensionless`
  - `isotherm.h`, [39](#)
- `F_prime`
  - `isotherms.Langmuir`, [24](#)
- `FormFunctionLocal`
  - `my_petsc.h`, [46](#)
- `FormJacobianLocal`
  - `my_petsc.h`, [46](#)
- `Fun`

- isotherm.h, 38
- get\_c\_local\_equilibrium
  - isotherms.Langmuir, 24
- get\_c\_rarefaction
  - isotherms.Langmuir, 25
- get\_c\_shock
  - isotherms.Langmuir, 25
- get\_other\_theory
  - plotting\_util.AsymptoticConvergence, 19
  - plotting\_util.TWConvergence, 32
- get\_u\_TW\_Langmuir
  - plotting\_util, 10
- initialize\_langmuir\_dimensionless
  - isotherm.h, 39
- InitializeVecAndArray
  - my\_petsc.h, 47
- integrate
  - my\_funcs.hpp, 41
- is\_rarefaction\_wave
  - isotherms.Langmuir, 25
- is\_shock\_wave
  - isotherms.Langmuir, 25
- ISO, 22
  - data, 22
  - dF, 22
  - F, 22
  - num\_params, 22
  - params, 22
- Isotherm, 23
- isotherm
  - AppCtx, 15
- isotherm.h
  - dF\_langmuir\_dimensionless, 39
  - dFun, 38
  - F\_langmuir\_dimensionless, 39
  - Fun, 38
  - initialize\_langmuir\_dimensionless, 39
- isotherms, 9
- isotherms.Langmuir, 23
  - \_\_init\_\_, 24
  - F, 24
  - F\_prime, 24
  - get\_c\_local\_equilibrium, 24
  - get\_c\_rarefaction, 25
  - get\_c\_shock, 25
  - is\_rarefaction\_wave, 25
  - is\_shock\_wave, 25
  - s, 26
  - shock\_lambda, 26
- j
  - AppCtx, 15
- K
  - AppCtx, 15
- kappa
  - AppCtx, 16
- m
  - AppCtx, 16
- make\_movie
  - AppCtx, 16
- movie\_step\_interval
  - AppCtx, 16
- my\_funcs.hpp
  - analytical\_btc, 40
  - exp\_I0, 41
  - integrate, 41
  - update\_solution, 42
- my\_petsc.h
  - CalculateOmega, 44
  - DestroyVecAndArray, 44
  - eval\_dS\_i\_j, 45
  - eval\_S\_i\_j, 45
  - FormFunctionLocal, 46
  - FormJacobianLocal, 46
  - InitializeVecAndArray, 47
  - one\_time\_step, 47
  - run, 47
  - simulate, 48
  - theta\_rule, 48
  - UpdateAnalytical, 49
  - UpdateIsotherm, 49
  - UpdateSolid, 50
- num\_params
  - ISO, 22
- omega
  - AppCtx, 16
- one\_time\_step
  - my\_petsc.h, 47
- params
  - ISO, 22
- plot
  - plotting\_util.Convergence, 21
- plot\_btcs
  - plotting\_util.AsymptoticConvergence, 19
  - plotting\_util.TWConvergence, 32
- plot\_error
  - plotting\_util.AsymptoticConvergence, 19
  - plotting\_util.TWConvergence, 32
- plotting\_util, 9
  - calculate\_slope\_error, 10
  - get\_u\_TW\_Langmuir, 10
  - save\_figure, 11
- plotting\_util.AsymptoticConvergence, 17
  - \_\_init\_\_, 18
  - get\_other\_theory, 19
  - plot\_btcs, 19
  - plot\_error, 19
- plotting\_util.Convergence, 20
  - \_\_init\_\_, 21
  - plot, 21
- plotting\_util.Spatial, 27
  - \_\_init\_\_, 28

- plotting\_util.Temporal, [28](#)
  - \_\_init\_\_, [29](#)
- plotting\_util.TWConvergence, [30](#)
  - \_\_init\_\_, [31](#)
  - get\_other\_theory, [32](#)
  - plot\_btcs, [32](#)
  - plot\_error, [32](#)
- print\_header
  - AppCtx, [16](#)
- run
  - my\_petsc.h, [47](#)
- s
  - isotherms.Langmuir, [26](#)
- S\_jm1
  - AppCtx, [16](#)
- save\_btc
  - AppCtx, [16](#)
- save\_figure
  - plotting\_util, [11](#)
- shock\_lambda
  - isotherms.Langmuir, [26](#)
- simulate
  - my\_petsc.h, [48](#)
- src/isotherm.h, [37](#)
- src/my\_funcs.hpp, [40](#)
- src/my\_petsc.h, [42](#)
- tau\_star
  - AppCtx, [17](#)
- theta
  - AppCtx, [17](#)
- theta\_rule
  - my\_petsc.h, [48](#)
- update\_solution
  - my\_funcs.hpp, [42](#)
- UpdateAnalytical
  - my\_petsc.h, [49](#)
- UpdateIsotherm
  - my\_petsc.h, [49](#)
- UpdateSolid
  - my\_petsc.h, [50](#)
- Variables\_j, [33](#)
  - a\_j, [33](#)
  - a\_jm1, [34](#)
  - error, [34](#)
  - W\_j, [34](#)
- vec
  - VecAndArray, [35](#)
- VecAndArray, [34](#)
  - arr, [34](#)
  - vec, [35](#)
- viewer
  - AppCtx, [17](#)
- W\_j
  - Variables\_j, [34](#)