Java™ API Documentation for The Kalman Filter On-line Learning Tool

May 10 2001
Contents

1 Package kfengine
   1.1 Interfaces .......................................................... 5
      1.1.1 INTERFACE KFModelFactory ........................................ 5
      1.1.2 INTERFACE MeasurementModel ..................................... 5
      1.1.3 INTERFACE ProcessModel ........................................ 7
      1.1.4 INTERFACE Truth ................................................ 7
   1.2 Classes ............................................................ 8
      1.2.1 CLASS AngleMeasurement .......................................... 8
      1.2.2 CLASS ConstantProcess .......................................... 9
      1.2.3 CLASS ConstantTruth ........................................... 10
      1.2.4 CLASS FillingProcess ........................................... 11
      1.2.5 CLASS FillingSloshingProcess ................................... 12
      1.2.6 CLASS FillingTruth ............................................... 14
      1.2.7 CLASS KalmanFilter ............................................ 14
      1.2.8 CLASS KalmanFilterData ......................................... 15
      1.2.9 CLASS KalmanFilterFactory ....................................... 17
      1.2.10 CLASS LevelMeasurement ......................................... 18
      1.2.11 CLASS MeasurementType ......................................... 19
      1.2.12 CLASS ProcessType ............................................ 20
      1.2.13 CLASS SloshingProcess ......................................... 20
      1.2.14 CLASS SloshingTruth ........................................... 22
      1.2.15 CLASS TruthDecorator .......................................... 22
      1.2.16 CLASS TruthType .............................................. 23
      1.2.17 CLASS WaterTankModelFactory .................................. 24

2 Package gui ............................................................ 27
   2.1 Classes ............................................................ 28
      2.1.1 CLASS ColumnPanel ............................................... 28
      2.1.2 CLASS ComponentSize ........................................... 28
      2.1.3 CLASS EditDialog ............................................... 29
      2.1.4 CLASS HelpDialog ............................................... 31
      2.1.5 CLASS KalmanFilterTool ......................................... 31
      2.1.6 CLASS RowPanel .................................................. 34
      2.1.7 CLASS StepDialog ............................................... 35

3 Package data repository ................................................ 39
   3.1 Classes ............................................................ 40
      3.1.1 CLASS DataRepository ............................................ 40
4 Package plotter

4.1 Classes ................................................................. 47

4.1.1 CLASS ThreePlot .............................................. 47
Chapter 1

Package kfengine

Package Contents

<table>
<thead>
<tr>
<th>Interfaces</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>KFModelFactory</td>
<td>5</td>
</tr>
<tr>
<td>Interface for factory classes that create process, measurement, and truth models.</td>
<td></td>
</tr>
<tr>
<td>MeasurementModel</td>
<td>5</td>
</tr>
<tr>
<td>Interface that describes the measurement model used by the Kalman filter.</td>
<td></td>
</tr>
<tr>
<td>ProcessModel</td>
<td>7</td>
</tr>
<tr>
<td>Interface that describes the process model used by the Kalman filter.</td>
<td></td>
</tr>
<tr>
<td>Truth</td>
<td>7</td>
</tr>
<tr>
<td>Interface for the actual process dynamics.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Classes</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>AngleMeasurement</td>
<td>8</td>
</tr>
<tr>
<td>Implementation of the MeasurementModel interface for a non-linear measurement model that measures the angle of the float arm.</td>
<td></td>
</tr>
<tr>
<td>ConstantProcess</td>
<td>9</td>
</tr>
<tr>
<td>Implementation of the ProcessModel interface that assumes a constant water level.</td>
<td></td>
</tr>
<tr>
<td>ConstantTruth</td>
<td>10</td>
</tr>
<tr>
<td>Implementation of the Truth interface that describes a constant water level.</td>
<td></td>
</tr>
<tr>
<td>FillingProcess</td>
<td>11</td>
</tr>
<tr>
<td>Implementation of the ProcessModel interface that assumes the water level may be rising steadily.</td>
<td></td>
</tr>
<tr>
<td>FillingSloshingProcess</td>
<td>12</td>
</tr>
<tr>
<td>Implementation of the ProcessModel interface that assumes the water level may be filling and sloshing.</td>
<td></td>
</tr>
<tr>
<td>FillingTruth</td>
<td>14</td>
</tr>
<tr>
<td>Add a filling component to the existing water level dynamics.</td>
<td></td>
</tr>
<tr>
<td>KalmanFilter</td>
<td>14</td>
</tr>
<tr>
<td>General implementation of a Kalman filter.</td>
<td></td>
</tr>
<tr>
<td>KalmanFilterData</td>
<td>15</td>
</tr>
<tr>
<td>Simulation data from Kalman filter for a single time step.</td>
<td></td>
</tr>
<tr>
<td>KalmanFilterFactory</td>
<td>17</td>
</tr>
<tr>
<td>Factory class that creates Kalman filters.</td>
<td></td>
</tr>
<tr>
<td>LevelMeasurement</td>
<td>18</td>
</tr>
<tr>
<td>Implementation of the MeasurementModel interface for a linear measurement model that measures the height of the float.</td>
<td></td>
</tr>
<tr>
<td>MeasurementType</td>
<td>19</td>
</tr>
</tbody>
</table>
Symbolic constants for the available measurement models.

**ProcessType** ................................................................. 20

Symbolic constants for the available process models.

**SloshingProcess** ............................................................... 20

Implementation of the ProcessModel interface that assumes the water level may be changing as a sinusoidal function of time.

**SloshingTruth** ............................................................... 22

Add a sloshing component to the existing water level dynamics.

**TruthDecorator** ............................................................. 22

Implementation of the Truth interface that allows additional behavior to be added to an existing Truth component.

**TruthType** ................................................................. 23

Symbolic constants for the available truth models.

**WaterTankModelFactory** ................................................... 24

Implementation of the KFModelFactory for the 1-D estimation of the water level in a tank.
1.1 Interfaces

1.1.1 INTERFACE KFModelFactory

Interface for factory classes that create process, measurement, and truth models.

DECLEARATION

```
public interface KFModelFactory
```

METHODS

- `makeMeasurementModel`
  ```java
  public MeasurementModel makeMeasurementModel(
    kfengine.MeasurementType type, kfengine.Truth signal, double scaleR
  )
  ```
  - Usage
    * Creates a specific measurement model with the given signal and noise.
  - Parameters
    * type - the measurement model type
    * signal - the truth signal
    * scaleR - the scale factor for the measurement noise

- `makeProcessModel`
  ```java
  public ProcessModel makeProcessModel(
    kfengine.ProcessType type, double scaleQ
  )
  ```
  - Usage
    * Creates a specific process model.
  - Parameters
    * type - the process model type
    * scaleQ - the scale factor for the process noise

- `makeTruth`
  ```java
  public Truth makeTruth( kfengine.TruthType type )
  ```
  - Usage
    * Creates a specific truth model.
  - Parameters
    * type - the truth type

1.1.2 INTERFACE MeasurementModel

Interface that describes the measurement model used by the Kalman filter.
DECLARATION

interface MeasurementModel

METHODS

- $H$
  public Matrix $H(\text{Jama.Matrix } x)$
  
  - Usage
    * Returns the measurement matrix for this measurement model
  
  - Parameters
    * $x$ - the estimated state

- $R$
  public Matrix $R(\ )$
  
  - Usage
    * Returns the measurement noise matrix for this measurement model

- $x_{\text{True}}$
  public Matrix $x_{\text{True}}(\text{double } t)$
  
  - Usage
    * Returns the actual state at time $t$ corresponding to an exact measurement.
  
  - Parameters
    * $t$ - the elapsed time in seconds

- $z$
  public Matrix $z(\text{double } t)$
  
  - Usage
    * Returns a noisy measurement at time $t$.
  
  - Parameters
    * $t$ - the elapsed time in seconds

- $z_{\text{Pred}}$
  public Matrix $z_{\text{Pred}}(\text{Jama.Matrix } x)$
  
  - Usage
    * Returns the predicted measurement at time $t$.
  
  - Parameters
    * $x$ - the estimated state

- $z_{\text{True}}$
  public Matrix $z_{\text{True}}(\text{double } t)$
  
  - Usage
    * Returns the exact measurement at time $t$.
  
  - Parameters
    * $t$ - the elapsed time in seconds
1.1.3 INTERFACE ProcessModel

Interface that describes the process model used by the Kalman filter.

DECLARATION

interface ProcessModel

METHODS

- **A**
  
  ```java
  public Matrix A( double t, double dt )
  ```
  
  **Usage**
  
  * Returns the state transition matrix (n x n) for this process model.
  
  **Parameters**
  
  * t - the time in seconds since start of simulation
  * dt - the number of seconds between measurements

- **P0**
  
  ```java
  public Matrix P0( )
  ```
  
  **Usage**
  
  * Returns an initial guess for the error covariance matrix (n x n).

- **Q**
  
  ```java
  public Matrix Q( double t, double dt )
  ```
  
  **Usage**
  
  * Returns the process noise matrix (n x n).
  
  **Parameters**
  
  * t - the time in seconds since start of simulation
  * dt - the number of seconds between measurements

- **x0**
  
  ```java
  public Matrix x0( )
  ```
  
  **Usage**
  
  * Returns an initial guess for the process state (n x 1).

1.1.4 INTERFACE Truth

Interface for the actual process dynamics.

DECLARATION

interface Truth
**Methods**

- \( x \)
  ```java
  public Matrix x( double t )
  ```
  - **Usage**
    * Returns the actual state at the specified time.
  - **Parameters**
    * \( t \) - the time in seconds since start of simulation

**1.2 Classes**

**1.2.1 Class AngleMeasurement**

Implementation of the MeasurementModel interface for a non-linear measurement model that measures the angle of the float arm.

**Declaration**

```java
class AngleMeasurement
extends java.lang.Object
implements MeasurementModel
```

**Constructors**

- **AngleMeasurement**
  ```java
  public AngleMeasurement( kfengine.Truth signal, java.util.Random noise, double r, double db, double df, double ka )
  ```
  - **Usage**
    * Constructs an angle measurement model for the given signal and noise.
  - **Parameters**
    * \( \text{signal} \) - the actual level
    * \( \text{noise} \) - the noise generator
    * \( r \) - the standard deviation of the measurement noise [deg]
    * \( db \) - the base height of angular sensor [m]
    * \( df \) - the height of angular sensor arm [m]
    * \( ka \) - the angle scale factor [V/deg]

**Methods**

- **\( H \)**
  ```java
  public Matrix H( Jama.Matrix x )
  ```
  - **Usage**
    * Returns the measurement matrix for this measurement model
### Class ConstantProcess

Implementation of the ProcessModel interface that assumes a constant water level. The estimated state has one element: the current water level.

```java
class ConstantProcess
    extends java.lang.Object
    implements ProcessModel
```

- **Parameters**
  * x - the estimated state

- **R**
  ```java
  public Matrix R()
  ```

  - **Usage**
    * Returns the measurement noise matrix for this measurement model

- **xTrue**
  ```java
  public Matrix xTrue( double t )
  ```

  - **Usage**
    * Returns the actual state at time t corresponding to an exact measurement.
  - **Parameters**
    * t - the elapsed time in seconds

- **z**
  ```java
  public Matrix z( double t )
  ```

  - **Usage**
    * Returns a noisy measurement at time t.
  - **Parameters**
    * t - the elapsed time in seconds

- **zPred**
  ```java
  public Matrix zPred( Jama.Matrix x )
  ```

  - **Usage**
    * Returns the predicted measurement at time t.
  - **Parameters**
    * x - the estimated state

- **zTrue**
  ```java
  public Matrix zTrue( double t )
  ```

  - **Usage**
    * Returns the exact measurement at time t.
  - **Parameters**
    * t - the elapsed time in seconds
CONSTRUCTORS

- **ConstantProcess**

  - **Usage**
    * Constructs a model that assumes a constant water level.

  - **Parameters**
    * \(x_0\) - the initial guess for the state
    * \(e_0\) - the initial guess for the estimate error
    * \(q_c\) - the standard deviation of process noise

METHODS

- **A**

  - **Usage**
    * Returns the state transition matrix for this process model.

  - **Parameters**
    * \(t\) - the time in seconds since start of simulation
    * \(dt\) - the number of seconds between measurements

- **P0**

  - **Usage**
    * Returns an initial guess for the error covariance matrix.

- **Q**

  - **Usage**
    * Returns the process noise matrix.

  - **Parameters**
    * \(t\) - the time in seconds since start of simulation
    * \(dt\) - the number of seconds between measurements

- **x0**

  - **Usage**
    * Returns an initial guess for the process state.

1.2.3 CLASS ConstantTruth

Implementation of the Truth interface that describes a constant water level.
class ConstantTruth
extends java.lang.Object
implements Truth

CONSTRUCTORS

• ConstantTruth
  ConstantTruth( double c )
    – Usage
      * Constructs the actual dynamics for a constant water level.
    – Parameters
      * c - the constant

METHODS

• x
  public Matrix x( double t )
    – Usage
      * Returns the actual state at the specified time.
    – Parameters
      * t - the time in seconds since start of simulation

1.2.4 CLASS FillingProcess

Implementation of the ProcessModel interface that assumes the water level may be rising steadily. The estimated state has two elements: the current level and the fill rate.

class FillingProcess
extends java.lang.Object
implements ProcessModel

CONSTRUCTORS

• FillingProcess
  FillingProcess( double [] x0, double [] e0, double qf )
    – Usage
      * Constructs a model that assumes an increasing water level.
- **Parameters**
  * $x_0$ - the initial guess for the state
  * $e_0$ - the initial guess for the estimate error
  * $q_f$ - the standard deviation of process noise

**METHODS**

- **$A$**
  ```java
  public Matrix A( double t, double dt )
  ```
  - **Usage**
    * Returns the state transition matrix for this process model.
  - **Parameters**
    * $t$ - the time in seconds since start of simulation
    * $dt$ - the number of seconds between measurements

- **$P_0$**
  ```java
  public Matrix P0( )
  ```
  - **Usage**
    * Returns an initial guess for the error covariance matrix.

- **$Q$**
  ```java
  public Matrix Q( double t, double dt )
  ```
  - **Usage**
    * Returns the process noise matrix.
  - **Parameters**
    * $t$ - the time in seconds since start of simulation
    * $dt$ - the number of seconds between measurements

- **$x_0$**
  ```java
  public Matrix x0( )
  ```
  - **Usage**
    * Returns an initial guess for the process state.

### 1.2.5 Class FillingSloshingProcess

Implementation of the ProcessModel interface that assumes the water level may be filling and sloshing. The estimated state contains three elements: the current level, the fill rate, and the magnitude of the sloshing.

**DECLARATION**

```java
class FillingSloshingProcess
extends java.lang.Object
implements ProcessModel
```
CONSTRUCTORS

- *FillingSloshingProcess*

  \[ \text{FillingSloshingProcess}( \text{double} \ [\] \ x0, \ \text{double} \ [\] \ e0, \ \text{double} \ \text{freq}, \ \text{double} \ \text{phase}, \ \text{double} \ \text{qf}, \ \text{double} \ \text{qs} ) \]

  - **Usage**
    * Constructs a model for a filling and sloshing process.
  
  - **Parameters**
    * \text{x0} - the initial guess for the state
    * \text{e0} - the initial guess for the estimate error
    * \text{freq} - the sloshing frequency [Hz]
    * \text{phase} - the phase [deg]
    * \text{qf} - the standard deviation of filling process noise
    * \text{qs} - the standard deviation of sloshing process noise

METHODS

- \text{A}

  \[ \text{public Matrix A}( \text{double} \ \text{t}, \ \text{double} \ \text{dt} ) \]

  - **Usage**
    * Returns the state transition matrix for this process model.
  
  - **Parameters**
    * \text{t} - the time in seconds since start of simulation
    * \text{dt} - the number of seconds between measurements

- \text{P0}

  \[ \text{public Matrix P0}( ) \]

  - **Usage**
    * Returns an initial guess for the error covariance matrix.

- \text{Q}

  \[ \text{public Matrix Q}( \text{double} \ \text{t}, \ \text{double} \ \text{dt} ) \]

  - **Usage**
    * Returns the process noise matrix.
  
  - **Parameters**
    * \text{t} - the time in seconds since start of simulation
    * \text{dt} - the number of seconds between measurements

- \text{x0}

  \[ \text{public Matrix x0}( ) \]

  - **Usage**
    * Returns an initial guess for the process state.
1.2.6  **CLASS FillingTruth**

Add a filling component to the existing water level dynamics.

**DECLARATION**

```java
class FillingTruth extends kfengine.TruthDecorator
```

**CONSTRUCTORS**

- *FillingTruth*
  ```java
  FillingTruth( kfengine.Truth component, double r, double tf )
  ```

  - **Usage**
    * Constructs the actual dynamics for a water level that is increasing or decreasing at a constant rate.
  
  - **Parameters**
    * component - the current dynamics
    * r - the fill rate [m/sec]
    * tf - the time to start filling tank [sec]

**METHODS**

- *x*
  ```java
  public Matrix x( double t )
  ```

  - **Usage**
    * Returns the actual state at the specified time.
  
  - **Parameters**
    * t - the time in seconds since start of simulation

1.2.7  **CLASS KalmanFilter**

General implementation of a Kalman filter.

**DECLARATION**

```java
public class KalmanFilter extends java.lang.Object
```
CONSTRUCTORS

- **KalmanFilter**
  
  ```java
  KalmanFilter( kfengine.ProcessModel process, kfengine.MeasurementModel measure, double t, double freq )
  ```

  - Usage
    * Constructs a Kalman filter for specific process and measurement models.
  
  - Parameters
    * `process` - the process model
    * `measure` - the measurement model
    * `t` - the length of the simulation (seconds)
    * `freq` - the frequency of measurements (Hz)

METHODS

- **runSimulation**
  
  ```java
  public List runSimulation( )
  ```

  - Usage
    * Run Kalman filter simulation.
  
  - Returns - a list containing the Kalman filter data

1.2.8 CLASS KalmanFilterData

Simulation data from Kalman filter for a single time step.

DECLARATION

```java
public class KalmanFilterData
    extends java.lang.Object
```

CONSTRUCTORS

- **KalmanFilterData**
  
  ```java
  KalmanFilterData( )
  ```

  - Usage
    * Constructs a set of Kalman filter data. This constructor is only accessible from this package
      and is meant to be used by the Kalman filter only.
METHODS

- **K**
  ```java
  public Matrix K()
  ```
  - Usage
    * Returns the Kalman gain.

- **PCorr**
  ```java
  public Matrix PCorr()
  ```
  - Usage
    * Returns the corrected (a posteriori) error covariance.

- **PPred**
  ```java
  public Matrix PPred()
  ```
  - Usage
    * Returns the predicted (a priori) error covariance.

- **time**
  ```java
  public double time()
  ```
  - Usage
    * Returns the time.

- **xCorr**
  ```java
  public Matrix xCorr()
  ```
  - Usage
    * Returns the corrected (a posteriori) state.

- **xPred**
  ```java
  public Matrix xPred()
  ```
  - Usage
    * Returns the predicted (a priori) state.

- **xTrue**
  ```java
  public Matrix xTrue()
  ```
  - Usage
    * Returns the actual (primary) state.

- **zMeas**
  ```java
  public Matrix zMeas()
  ```
  - Usage
    * Returns the actual measurements.

- **zPred**
  ```java
  public Matrix zPred()
  ```
  - Usage
    * Returns the predicted (a priori) measurement.
1.2.9 **Class KalmanFilterFactory**

Factory class that creates Kalman filters.

**Declaration**

```java
public class KalmanFilterFactory
    extends java.lang.Object
```

**Fields**

- public static final double T_SIM
- public static final double M_RATE

**Constructors**

- `KalmanFilterFactory`
  ```java
  public KalmanFilterFactory( kfengine.KFModelFactory models )
  ```
  - Usage
    * Creates a factory for Kalman filters.
  - Parameters
    * models - the factory of the specific Kalman filter models

**Methods**

- `makeKalmanFilter`
  ```java
  public KalmanFilter makeKalmanFilter( kfengine.ProcessType procType,
          kfengine.TruthType truthType, kfengine.MeasurementType measType,
          double scaleQ, double scaleR )
  ```
  - Usage
    * Creates a Kalman filter for the specific conditions.
  - Parameters
    * procType - the process model type
    * truthType - the truth type
    * measType - the measurement model type
    * scaleQ - the scale factor for the process noise
    * scaleR - the scale factor for the measurement noise
### 1.2.10 Class LevelMeasurement

Implementation of the MeasurementModel interface for a linear measurement model that measures the height of the float.

#### Declaration

class LevelMeasurement
    extends java.lang.Object
    implements MeasurementModel

#### Constructors

- **LevelMeasurement**
  
  LevelMeasurement( kfengine.Truth signal, java.util.Random noise, 
  double r, double kl )
  
  - Usage
    * Constructs a level measurement model for the given signal and noise.
  
  - Parameters
    * signal - the actual level
    * noise - the noise generator
    * r - the standard deviation of the measurement noise [m]
    * kl - the level scale factor [V/m]

#### Methods

- **H**
  
  public Matrix H( Jama.Matrix x )
  
  - Usage
    * Returns the measurement matrix for this measurement model
  
  - Parameters
    * x - the estimated state

- **R**
  
  public Matrix R( )
  
  - Usage
    * Returns the measurement noise matrix for this measurement model

- **xTrue**
  
  public Matrix xTrue( double t )
  
  - Usage
    * Returns the actual state at time t corresponding to an exact measurement.
  
  - Parameters
* \( t \) - the elapsed time in seconds

- \( z \)
  public Matrix \( z(\) double \( t \) \)
  
  - **Usage**
    * Returns a noisy measurement at time \( t \).
  
  - **Parameters**
    * \( t \) - the elapsed time in seconds

- \( z_{\text{Pred}} \)
  public Matrix \( z_{\text{Pred}}(\) Jama.Matrix \( x \) \)
  
  - **Usage**
    * Returns the predicted measurement at time \( t \).
  
  - **Parameters**
    * \( x \) - the estimated state

- \( z_{\text{True}} \)
  public Matrix \( z_{\text{True}}(\) double \( t \) \)
  
  - **Usage**
    * Returns the exact measurement at time \( t \).
  
  - **Parameters**
    * \( t \) - the elapsed time in seconds

### 1.2.11 Class MeasurementType

Symbolic constants for the available measurement models.

#### Declaration

```java
public final class MeasurementType
extends java.lang.Object
```

#### Fields

- public static final MeasurementType LEVEL
  
  - Represents a LEVEL measurement.

- public static final MeasurementType ANGLE
  
  - Represents an ANGLE measurement.
METHODS

- `toString`
  ```java
  public String toString()
  ```
  - **Usage**
    * Returns a string identifying the measurement type of this object.

1.2.12 CLASS ProcessType

Symbolic constants for the available process models.

DECLARATION

```java
public final class ProcessType
  extends java.lang.Object
```

FIELDS

- public static final ProcessType CONSTANT
  - Represents a CONSTANT process.
- public static final ProcessType FILLING
  - Represents a FILLING process.
- public static final ProcessType SLOSHING
  - Represents a SLOSHING process.
- public static final ProcessType FILLING_SLOSHING
  - Represents a FILLING and SLOSHING process.

METHODS

- `toString`
  ```java
  public String toString()
  ```
  - **Usage**
    * Returns a string identifying the process type of this object.

1.2.13 CLASS SloshingProcess

Implementation of the ProcessModel interface that assumes the water level may be changing as a sinusoidal function of time. The estimated state contains two elements: the current level and the magnitude of the sloshing.
class SloshingProcess  

extends java.lang.Object  
implements ProcessModel  

**Constructors**

- *SloshingProcess*
  
  SloshingProcess( double [] x0, double [] e0, double freq, double phase, double qs )

  - **Usage**
    * Constructs a model for a sloshing process.

  - **Parameters**
    * x0 - the initial guess for the state
    * e0 - the initial guess for the estimate error
    * freq - the sloshing frequency [Hz]
    * phase - the phase [deg]
    * qs - the standard deviation of process noise

**Methods**

- *A*
  
  public Matrix A( double t, double dt )

  - **Usage**
    * Returns the state transition matrix for this process model.

  - **Parameters**
    * t - the time in seconds since start of simulation
    * dt - the number of seconds between measurements

- *P0*
  
  public Matrix P0( )

  - **Usage**
    * Returns an initial guess for the error covariance matrix.

- *Q*
  
  public Matrix Q( double t, double dt )

  - **Usage**
    * Returns the process noise matrix.

  - **Parameters**
    * t - the time in seconds since start of simulation
    * dt - the number of seconds between measurements

- *x0*
  
  public Matrix x0( )
– Usage
  * Returns an initial guess for the process state.

1.2.14 CLASS SloshingTruth

Add a sloshing component to the existing water level dynamics.

DECLARATION

```java
class SloshingTruth
extends kfengine.TruthDecorator
```

CONSTRUCTORS

- **SloshingTruth**
  ```java
  SloshingTruth( kfengine.Truth component, double freq, double phase, double ks )
  ```

  – Usage
    * Constructs the actual dynamics for a water level that is changing as a sinusoidal function of time.

  – Parameters
    * component - the current dynamics
    * freq - the sloshing frequency [Hz]
    * phase - the phase angle [deg]
    * ks - the magnitude of the sloshing [m]

METHODS

- **x**
  ```java
  public Matrix x( double t )
  ```

  – Usage
    * Returns the actual state at the specified time.

  – Parameters
    * t - the time in seconds since start of simulation

1.2.15 CLASS TruthDecorator

Implementation of the Truth interface that allows additional behavior to be added to an existing Truth component.
class TruthDecorator
extends java.lang.Object
implements Truth

CONSTRUCTORS

• TruthDecorator
  TruthDecorator( kfengine.Truth component )
  – Usage
    * Constructs a decorator that adds additional dynamics to an existing Truth component.
  – Parameters
    * component - the current dynamics

METHODS

• x
  public Matrix x( double t )
  – Usage
    * Returns the actual state at the specified time.
  – Parameters
    * t - the time in seconds since start of simulation

1.2.16 CLASS TruthType

Symbolic constants for the available truth models.

DECLARATION

public final class TruthType
extends java.lang.Object

FIELDS

• public static final TruthType CONSTANT
  – Represents CONSTANT actual dynamics.
• public static final TruthType FILLING
  – Represents FILLING actual dynamics.
• public static final TruthType SLOSHING
  – Represents SLOSHING actual dynamics.

• public static final TruthType FILLING_SLOSHING
  – Represents FILLING and SLOSHING actual dynamics.

METHODS

• toString
  public String toString()
  – Usage
    * Returns a string identifying the truth type of this object.

1.2.17 CLASS WaterTankModelFactory

Implementation of the KFModelFactory for the 1-D estimation of the water level in a tank.

DECLARATION

```java
public class WaterTankModelFactory
  extends java.lang.Object
  implements KFModelFactory
```

FIELDS

• public static final double L_MIN
  – The initial water level [m].

• public static final double L_MAX
  – The full level of the tank [m].

• public static final double QC
  – The standard deviation of the process noise for CONSTANT model.

• public static final double QF
  – The standard deviation of the process noise for FILLING model.

• public static final double QS
  – The standard deviation of the process noise for SLOSHING model.

• public static final double QFS_F
  – The standard deviation of the process noise for FILLING component of FILLING and SLOSHING model.
- public static final double QFS_S
  - The standard deviation of the process noise for SLOSHING component of FILLING and SLOSHING model.
- public static final double FILL
  - The fill rate of the tank [m/s].
- public static final double DELAY
  - The delay before filling the tank [s].
- public static final double FREQ
  - The sloshing frequency [Hz].
- public static final double PHASE
  - The sloshing phase [deg].
- public static final double KS
  - The sloshing magnitude [m].
- public static final double L_INIT
  - Guess for the initial level of water in tank [m].
- public static final double MAX_FILL
  - Guess for initial fill rate [m/s].
- public static final double MAX_SLOSH
  - Guess for initial sloshing magnitude [m].
- public static final double DB
  - Height of angular float sensor base [m].
- public static final double DF
  - Length of angular float sensor arm [m].
- public static final double KA
  - Angular float sensor scale factor [V/deg].
- public static final double KL
  - Level float sensor scale factor [V/m].
- public static final double RL
  - Standard deviation of level measurement noise [m].
- public static final double RA
  - Standard deviation of angle measurement noise [deg].
- public static final long SEED
  - Seed for random number generator used for noise.
CONSTRUCTORS

- WaterTankModelFactory
  
  public WaterTankModelFactory( )

  - Usage
    * Default constructor.

METHODS

- makeMeasurementModel
  
  public MeasurementModel makeMeasurementModel( kfengine.MeasurementType type, kfengine.Truth signal, double scaleR )

  - Usage
    * Creates a specific measurement model with the given signal and noise.

  - Parameters
    * type - the measurement model type
    * signal - the truth signal
    * scaleR - the scale factor for the measurement noise

- makeProcessModel
  
  public ProcessModel makeProcessModel( kfengine.ProcessType type, double qScale )

  - Usage
    * Creates a specific process model.

  - Parameters
    * type - the process model type
    * qScale - the scale factor for the process noise

- makeTruth
  
  public Truth makeTruth( kfengine.TruthType type )

  - Usage
    * Creates a specific truth model.

  - Parameters
    * type - the truth type
Chapter 2

Package gui

Package Contents

<table>
<thead>
<tr>
<th>Classes</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ColumnPanel</td>
<td>28</td>
</tr>
<tr>
<td>ComponentSize</td>
<td>28</td>
</tr>
<tr>
<td>EditDialog</td>
<td>29</td>
</tr>
<tr>
<td>HelpDialog</td>
<td>31</td>
</tr>
<tr>
<td>KalmanFilterTool</td>
<td>31</td>
</tr>
<tr>
<td>RowPanel</td>
<td>34</td>
</tr>
<tr>
<td>StepDialog</td>
<td>35</td>
</tr>
</tbody>
</table>

- **ColumnPanel**: Displays a column containing multiple components.
- **ComponentSize**: Manipulates sizes of components.
- **EditDialog**: Displays and edits current settings for the water level simulation.
- **HelpDialog**: Displays a help screen containing HTML text to the user.
- **KalmanFilterTool**: Displays the current settings for the Kalman Filter Tutorial as well as command buttons to plot and step through the simulation.
- **RowPanel**: Displays a row containing multiple components.
- **StepDialog**: Steps through water tank simulation and displays intermediate values for state, covariance, measurement, and Kalman filter gain.
2.1 Classes

2.1.1 Class ColumnPanel

Displays a column containing multiple components.

**Declaration**

```
public class ColumnPanel
    extends javax.swing.JPanel
```

**Constructors**

- **ColumnPanel**
  ```
  public ColumnPanel(java.awt.Component c1, java.awt.Component c2)
  ```
  - **Usage**
    * Creates a column containing two components.
  - **Parameters**
    * c1, c2 the components

- **ColumnPanel**
  ```
  public ColumnPanel(java.awt.Component c1, java.awt.Component c2, java.awt.Component c3)
  ```
  - **Usage**
    * Creates a column containing three components.
  - **Parameters**
    * c1, c2, c3 the components

2.1.2 Class ComponentSize

Manipulates sizes of components.

**Declaration**

```
public class ComponentSize
    extends java.lang.Object
```

**Constructors**

- **ComponentSize**
  ```
  public ComponentSize()
  ```
METHODS

- `createFiller`
  
  ```java
  public static Component createFiller( javax.swing.JComponent c )
  ```
  
  **Usage**
  
  * Creates filler equal in size to the given component.

  **Parameters**
  
  * c - the component

- `setEqual`
  
  ```java
  public static void setEqual( javax.swing.JComponent c1, javax.swing.JComponent c2 )
  ```
  
  **Usage**
  
  * Sets minimum, maximum, and preferred sizes of the second component equal to the sizes of the first.

  **Parameters**
  
  * c1 - the component with the defined sizes
  * c2 - the component whose size is changed

2.1.3 CLASS EditDialog

Displays and edits current settings for the water level simulation.

DECLARATION

```java
public class EditDialog
extends javax.swing.JDialog
```

SERIALIZABLE FIELDS

- private KalmanFilterTool parent
- private JCheckBox truthFill
- private JCheckBox truthSlosh
- private JCheckBox modelFill
- private JCheckBox modelSlosh
• private JRadioButton typeLevel
• private JRadioButton typeAngle
• private JRadioButton noise01
• private JRadioButton noise1
• private JRadioButton noise10
• private ButtonGroup typeGroup
• private ButtonGroup noiseGroup
• private JButton apply
• private JButton close
• private JButton help
• private HelpDialog helpDialog

FIELDS

• public static final String HELP_PAGE
  · Edit window help page (relative to KalmanFilterTool.HELP_HOME).

CONSTRUCTORS

• EditDialog
  public EditDialog( gui.KalmanFilterTool parent )
  · Usage
    * Creates a dialog for editing the settings for the water level simulation.
  · Parameters
    * parent - the parent frame
2.1.4 Class HelpDialog

Displays a help screen containing HTML text to the user.

DECLARATION

```java
public class HelpDialog
    extends javax.swing.JDialog
```

SERIALIZABLE FIELDS

- private URL current
  -
- private JEditorPane editorPane
  -
- private Stack previous
  -
- private JButton back
  -

CONSTRUCTORS

- `HelpDialog`
  ```java
  public HelpDialog( java.lang.String current )
  ```
  - Usage
    * Creates a dialog for displaying HTML help text to the user.

2.1.5 Class KalmanFilterTool

Displays the current settings for the Kalman Filter Tutorial as well as command buttons to plot and step through the simulation.

DECLARATION

```java
public class KalmanFilterTool
    extends javax.swing.JApplet
```
Serializable Fields

- private TruthType truthType
  -
- private ProcessType modelType
  -
- private MeasurementType measType
  -
- private double scaleR
  -
- private double length
  -
- private double rate
  -
- private ThreePlot plots
  -
- private StepDialog stepper
  -
- private DataRepository data
  -
- private boolean haveSettingsChanged
  -
- private JTextField truthField
  -
- private JTextField modelField
  -
- private JTextField typeField
  -
- private JTextField noiseField
  -
- private JTextField lengthField
  -
- private JTextField rateField
  -
• private JButton save
  –
• private JButton edit
  –
• private JButton plot
  –
• private JButton step
  –
• private JButton help
  –
• private boolean isApplication
  –

FIELDS

• public static final String HELP_HOME
  – URL of directory of help pages.
• public static final String ICON_HOME
  – URL of directory of button icons.
• public static final String HELP_PAGE
  – Main help page (relative to HELP_HOME).

CONSTRUCTORS

• KalmanFilterTool
  public KalmanFilterTool()

METHODS

• init
  public void init()
  – Usage
    * Creates a main window displaying the current simulation settings.

• main
  public static void main( java.lang.String [] args )
  – Usage
    * Run the Kalman Filter Tutorial as an application.
- **setApplication**
  
  ```java
  public void setApplication( boolean isApplication )
  ```

  - **Usage**
    - Set the KalmanFilterTool to run as an application or applet.
  
  - **Parameters**
    - isApplication - true if run as an application

- **setMeasurementNoise**
  
  ```java
  void setMeasurementNoise( double scaleR )
  ```

  - **Usage**
    - Sets the current measurement noise scale factor.
  
  - **Parameters**
    - scaleR - the noise scale factor

- **setParameters**
  
  ```java
  void setParameters( kfengine.TruthType truthType, kfengine.ProcessType procType, kfengine.MeasurementType measType, double scaleR )
  ```

  - **Usage**
    - Sets the current parameters.
  
  - **Parameters**
    - truthType - the actual dynamics
    - procType - the modeled dynamics
    - measType - the measurement model
    - scaleR - the noise scale factor

### 2.1.6 CLASS RowPanel

Displays a row containing multiple components.

**DECLARATION**

```java
public class RowPanel
extends javax.swing.JPanel
```

**CONSTRUCTORS**

- **RowPanel**
  
  ```java
  public RowPanel( java.awt.Component c1, java.awt.Component c2 )
  ```

  - **Usage**
    - Create a row containing two components.
  
  - **Parameters**
    - c1, c2 the components
• **RowPanel**

  ```java
  public RowPanel(java.awt.Component c1, java.awt.Component c2, java.awt.Component c3)
  ```

  **Usage**
  * Create a row containing three components.

  **Parameters**
  * c1, c2, c3 the components

**2.1.7 CLASS StepDialog**

Steps through water tank simulation and displays intermediate values for state, covariance, measurement, and Kalman filter gain.

**DECLARATION**

```java
public class StepDialog
extends javax.swing.JDialog
```

**SERIALIZABLE FIELDS**

- public ThreePlot plots
- private JButton rewind
- private JButton stepBack
- private JButton stepForward
- private JButton fastForward
- private JButton close
- private JButton help
- private int m
- private int n
- private JTextField time
  -
- private JTextField t1
  -
- private JTextField t2
  -
- private JTextField xPred1
  -
- private JTextField xPred2
  -
- private JTextField PPred1
  -
- private JTextField PPred2
  -
- private JTextField zPred1
  -
- private JTextField zPred2
  -
- private JTextField xCorr1
  -
- private JTextField xCorr2
  -
- private JTextField PCorr1
  -
- private JTextField PCorr2
  -
- private JTextField zCorr1
  -
- private JTextField zCorr2
  -
- private JTextField xTrue1
  -
- private JTextField xTrue2
• private JTextField K1
• private JTextField K2
• private DataRepository data
• private int step
• private int maxStep
• private HelpDialog helpDialog

FIELDS

• public static final String HELP_PAGE
  – Edit window help page (relative to KalmanFilterTool.HELP_HOME).
• public static final String REWIND_ICON
  – Rewind icon (relative to KalmanFilterTool.ICON_HOME).
• public static final String STEP_BACK_ICON
  – Step back icon (relative to KalmanFilterTool.ICON_HOME).
• public static final String STEP_FORWARD_ICON
  – Step forward icon (relative to KalmanFilterTool.ICON_HOME).
• public static final String FAST_FORWARD_ICON
  – Fast forward icon (relative to KalmanFilterTool.ICON_HOME).
• public static final int FIELD_WIDTH
  –
• public static final int FIELD_HEIGHT
  –
• public ThreePlot plots
  –
CONSTRUCTORS

- `StepDialog`
  public `StepDialog` ( `data.repository.DataRepository data` )
  
  - Usage
    * Creates a dialog for stepping through water level simulation.
  
  - Parameters
    * parent - the parent frame

METHODS

- `resetPlotWindow`
  public void `resetPlotWindow` ( `plotter.ThreePlot t` )
  
  - Usage
    * Set the plots to a particular instance of a ThreePlot.
Chapter 3

Package data_repository

Package Contents

<table>
<thead>
<tr>
<th>Classes</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataRepository</td>
<td>40</td>
</tr>
</tbody>
</table>

Data Repository module of the ‘Kalman Filter On-line Learning Tool’
3.1 Classes

3.1.1 Class DataRepository

Data Repository module of the ‘Kalman Filter On-line Learning Tool’

Declaration

```
public class DataRepository
extends java.lang.Object
```

Constructors

- `DataRepository`
  ```
  public DataRepository(java.util.List KF_data_list, kfengine.ProcessType pt, kfengine.TruthType tt, kfengine.MeasurementType mt, double scaleQ, double scaleR)
  ```
  - Usage
    * Constructs a repository for a set of Kalman filter data. The repository provides storage for the data and ‘dumps’ the data to a string (writing it out in tab-delimited columns).
  - Parameters
    * KF_data_list - a list of KalmanFilterData objects
    * pt - a descriptor for the process type
    * tt - a descriptor for the truth type
    * mt - a descriptor for the measurement type
    * scaleQ - a scale factor for the process noise
    * scaleR - a scale factor for the measurement noise

Methods

- `getActualMeas`
  ```
  public String getActualMeas(int timestep)
  ```
  - Usage
    * Retrieves a single ‘actual measurement’ value for the timestep provided.
  - Parameters
    * timestep - an integer representing the desired timestep
  - Returns - a String representation of the ‘actual measurement’ value

- `getActualMeas`
  ```
  public String getActualMeas(int timestep, int i)
  ```
  - Usage
    * Retrieves a component of a single ‘actual measurement’ value for the timestep provided.
  - Parameters
    * timestep - an integer representing the desired timestep
- * i - the specific component
  - **Returns** - a String representation of the 'actual measurement' value

- **getCorrCovar**
  public String getCorrCovar( int timestep )

  - **Usage**
    * Retrieves a single 'corrected covariance' value for the timestep provided.
  
  - **Parameters**
    * timestep - an integer representing the desired timestep
  
  - **Returns** - a String representation of the 'corrected covariance' value

- **getCorrCovar**
  public String getCorrCovar( int timestep, int i, int j )

  - **Usage**
    * Retrieves a component of a single 'corrected covariance' value for the timestep provided.
  
  - **Parameters**
    * timestep - an integer representing the desired timestep
    * i, j - the specific component indices
  
  - **Returns** - a String representation of the 'corrected covariance' value

- **getCorrState**
  public String getCorrState( int timestep )

  - **Usage**
    * Retrieves a single 'corrected state' value for the timestep provided.
  
  - **Parameters**
    * timestep - an integer representing the desired timestep
  
  - **Returns** - a String representation of the 'corrected state' value

- **getCorrState**
  public String getCorrState( int timestep, int i )

  - **Usage**
    * Retrieves a component of a single 'corrected state' value for the timestep provided.
  
  - **Parameters**
    * timestep - an integer representing the desired timestep
    * i - the specific component
  
  - **Returns** - a String representation of the 'corrected state' value

- **getError**
  public double getError( )

  - **Usage**
    * Retrieves an array of error values from the set of simulation data. These values represent the error in the filter’s attempt to estimate the system, where error is measured as the square root of the covariance.
  
  - **Returns** - an array of (double) error values
• getEstimate
  public double getEstimate() {
    // Implementation
  }
  
  – Usage
    * Retrieves an array of estimate values from the set of simulation data. These values
      represent the estimated state of the system.
  – Returns
    - an array of (double) estimate values

• getKalmanGain
  public String getKalmanGain(int timestep) {
    // Implementation
  }
  
  – Usage
    * Retrieves a single 'Kalman gain' value for the timestep provided.
  – Parameters
    * timestep - an integer representing the desired timestep
  – Returns
    - a String representation of the 'Kalman gain' value

• getKalmanGain
  public String getKalmanGain(int timestep, int i, int j) {
    // Implementation
  }
  
  – Usage
    * Retrieves a component of a single 'Kalman gain' value for the timestep provided.
  – Parameters
    * timestep - an integer representing the desired timestep
    * i, j - the specific component indices
  – Returns
    - a String representation of the 'Kalman gain' value

• getMaxSteps
  public int getMaxSteps() {
    // Implementation
  }
  
  – Usage
    * Returns the total number of time steps in the simulation.

• getMeasurementSize
  public int getMeasurementSize() {
    // Implementation
  }
  
  – Usage
    * Returns the number of elements in a measurement vector.

• getPredCovar
  public String getPredCovar(int timestep) {
    // Implementation
  }
  
  – Usage
    * Retrieves a single 'predicted covariance' value for the timestep provided.
  – Parameters
    * timestep - an integer representing the desired timestep
  – Returns
    - a String representation of the 'predicted state' value

• getPredCovar
  public String getPredCovar(int timestep, int i, int j) {
    // Implementation
  }
  
  – Usage
* Retrieves a component of a single ‘predicted covariance’ value for the timestep provided.
  
  **Parameters**
  * timestep - an integer representing the desired timestep
  * i, j the specific component indices
  
  **Returns** - a String representation of the ‘predicted state’ value

---

- `getPredMeas`
  
  `public String getPredMeas( int timestep )`

  **Usage**
  * Retrieves a single ‘predicted measurement’ value for the timestep provided.
  
  **Parameters**
  * timestep - an integer representing the desired timestep
  
  **Returns** - a String representation of the ‘predicted measurement’ value

---

- `getPredMeas`
  
  `public String getPredMeas( int timestep, int i )`

  **Usage**
  * Retrieves a component of a single ‘predicted measurement’ value for the timestep provided.
  
  **Parameters**
  * timestep - an integer representing the desired timestep
  * i - the specific component
  
  **Returns** - a String representation of the ‘predicted measurement’ value

---

- `getPredState`
  
  `public String getPredState( int timestep )`

  **Usage**
  * Retrieves a single ‘predicted state’ value for the timestep provided.
  
  **Parameters**
  * timestep - an integer representing the desired timestep
  
  **Returns** - a String representation of the ‘predicted state’ value

---

- `getPredState`
  
  `public String getPredState( int timestep, int i )`

  **Usage**
  * Retrieves a component of a single ‘predicted state’ value for the timestep provided.
  
  **Parameters**
  * timestep - an integer representing the desired timestep
  * i - the specific component
  
  **Returns** - a String representation of the ‘predicted state’ value

---

- `getResidual`
  
  `public double getResidual( )`

  **Usage**
  * Retrieves an array of residual values from the set of simulation data. These values represent the difference in the true state and the estimated state.
- **Returns** - an array of (double) residual values

- **getStateSize**
  public int `getStateSize`()
  - **Usage**
    * Returns the number of elements in a state vector.

- **getTime**
  public String `getTime`(int `timestep`)
  - **Usage**
    * Returns the time at a specific step.

- **getTimes**
  public double `getTimes`()
  - **Usage**
    * Retrieves an array of time-steps from the set of simulation data.
    - **Returns** - an array of (double) time-step values

- **getTrueState**
  public String `getTrueState`(int `timestep`)
  - **Usage**
    * Retrieves a single 'true state' value for the timestep provided.
    - **Parameters**
      * `timestep` - an integer representing the desired timestep
    - **Returns** - a String representation of the 'true state' value

- **getTrueState**
  public String `getTrueState`(int `timestep`, int `i`)
  - **Usage**
    * Retrieves a component of a single 'true state' value for the timestep provided.
    - **Parameters**
      * `timestep` - an integer representing the desired timestep
      * `i` - the specific component
    - **Returns** - a String representation of the 'true state' value

- **getTruth**
  public double `getTruth`()
  - **Usage**
    * Retrieves an array of ‘truth’ values from the set of simulation data. These values represent the actual state of the system.
    - **Returns** - an array of (double) ‘truth’ values

- **toString**
  public String `toString`()
  - **Usage**
    * Returns a String version of the ‘data dump’.
    - **Returns** - the ‘data dump’ String
Chapter 4

Package plotter

Package Contents

<table>
<thead>
<tr>
<th>Classes</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ThreePlot</td>
<td>47</td>
</tr>
</tbody>
</table>

Basic class that creates the plots viewed by the user (truth and model, variance, and residual plots).
4.1 Classes

4.1.1 Class ThreePlot

Basic class that creates the plots viewed by the user (truth and model, variance, and residual plots). The plots are created using the ptplot3.1 package available for download at http://ptolemy.eecs.berkeley.edu/java/ptplot/

Declaration

```java
public class ThreePlot
    extends java.awt.Frame
```

Serializable Fields

- private Plot truthAndEstimatePlot
- private Plot variancePlot
- private Plot residualPlot
- private double truthAndEstimateMax
- private double truthAndEstimateMin
- private double varianceMax
- private double varianceMin
- private double residualMax
- private double residualMin
- private double times
- private double truth
- private double estimate
- private double variance
- private double residual

**Constructors**

- `ThreePlot`  
  `public ThreePlot(double[] times, double[] t, double[] e, double[] v, double[] r, double timeStep, java.lang.String Actual, java.lang.String Model)`

  - **Usage**  
    * Contracts a frame object with three embedded plots.
  
  - **Parameters**  
    * `times` - the double array containing the time steps  
    * `t` - the double array containing the truth values  
    * `e` - the double array containing the estimated values  
    * `v` - the double array containing the variance values  
    * `r` - the double array containing the residual values  
    * `timeStep` - the current time step examined in the step window  
    * `Actual` - the actual dynamics  
    * `Model` - the modeled dynamics

**Methods**

- `resetTimeStep`  
  `public void resetTimeStep(double timeStep)`

  - **Usage**  
    * Allows a time step vertical line to be added to the plots at the specified time step. The line is printed between the min and max values of the data sets in the associated data sets.

  - **Parameters**  
    * `timeStep` - a double value indicating the position of the current time step