# `MoULDyS` User Guide

## Introduction

We will discuss how to perform the following:

- **Introduction to Logs.** This section will provide a brief introduction to the types of logs, with examples, that are currently supported by `MoULDyS`. We currently support logs represented as intervals and zonotopes.
- **Offline Monitoring.** Here, we provide a step wise guide on how to encode a dynamics and perform offline monitoring of a given log.
- **Online Monitoring.** Next, we will discuss the steps to perform online monitoring, given the actual behavior of the system.
- **Compare Online and Offline Monitoring.** In this section we will discuss steps to compare online and offline monitoring.
- **Generating Logs (Optional)**. This optional section will provide a step wise guide to synthesize random logs of a system from a given initial set.

## Introduction to Logs

The logs are required to be stored at `/my/location/MoULDyS/data/`. We currently support logs represented as intervals and zonotopes. The log file must use the extension `.mlog`.

### Logs as Intervals

The format of the log is as follows:

```
<time_step>: <interval>
```

Following is an an example log file:

```
4: [[1,2],[2,3]]
5: [[2,3],[5,6]]
```

Each line in the log file represents the log at a given time step. For instance, `4: [[1,2],[2,3]]` implies the behavior of the system is given by the interval `[[1,2],[2,3]]` at time step `4`.

### Logs as Zonotopes

The format of the log is as follows:

```
<time_step>: <center_of_zonotope>; <generator_of_zonotope>
```

Following is an an example log file:

```
4: [0,0]; [[1,0],[0,1]]
5: [0,0]; [[1.2,0.1],[0.2,1.2]]
```

Each line in the log file represents the log at a given time step. For instance, `4: [0,0]; [[1,0],[0,1]]` implies the behavior of the system is given by the zonotope with center `[0,0]` and generator `[[1,0],[0,1]]`.

# Offline Monitoring

Consider the following toy dynamics for which we want to perform offline monitoring:

$$x[t+1] = Ax[t] + Bu[t];$$

where:

$$A = \begin{bmatrix} 1 & c \\ 0 & 1 \end{bmatrix},$$

$$B = \begin{bmatrix} d \\ 0.01 \end{bmatrix},$$

$c \in 0.1 \pm 1\%$, $d \in 0.01 \pm 1\%$, and $\forall_t u[t] \in [-0.1, 0.1]$.

### Step 0: Set the project path, and import MoULDyS engine.

```
import os,sys
PROJECT_ROOT = os.environ['MNTR_ROOT_DIR']
sys.path.append(PROJECT_ROOT) # Set the project path

from lib.MoULDySEngine import * # Importing all the functionalities of MoULDyS.
```

### Step 1: Encode the dynamics.

```
def toyEgOffline():
    A=np.array(
    [
    [1, 0.1],
    [0, 1]
    ]
    )
    B=np.array(
    [
    [0.01],
    [0.01]
    ]
    )
```

### Step 2: Define the mode of the dynamics.

- Use `.` for continuous time systems (Note: `MoULDyS` will discretize).
- Use `+` for discrete time systems.

```
mode='+'
```

### Optional: Set discretization parameter, if the mode is continuous.

```
h=0.01 # This step is not needed for this example. However, having this, wouldn't have any impact in this example.
```

### Step 3: Encode the uncertainties in the dynamics.

The uncertainties in the dynamics are as follows:

- $c \in 0.1 \pm 1\%$. If matrices $A$ and $B$ are augmented together, this uncertainty would be at cell $(0,1)$ of the augmented matrix. To encode an uncertainty of 1%, we use the following syntax: $[0.99, 1.01]$.
- $d \in 0.1 \pm 1\%$. If matrices $A$ and $B$ are augmented together, this uncertainty would be at cell $(0,2)$ of the augmented matrix.
  - The augmented dynamics is given below. Note that `MoULDyS` augments the dynamics as follows, internally, before performing monitoring.
  - $$augment(A,B) = \Lambda = \begin{bmatrix} 1 & c & d \\ 0 & 1 & 0.01 \\ 0 & 0 & 1 \end{bmatrix}.$$
  - Therefore, the dynamics $x[t+1] = Ax[t] + Bu[t]$ can be rewritten as follows:
    - $z[t+1] = \Lambda z[t]$
    - Note that the input $u[t] \in \mathcal{U}$, for all $t$, is treated as a state variable in the augmented dynamics.

### Step 4: Encode the unsafe set.

Let the unsafe behavior be as follows:

- `state_variable_0 >=200` `AND`
- `state_variable_0 <=-200`

Unsafe sets can be encoded as intervals

```
unsafe1=[(-np.inf,-200),(-np.inf,np.inf),(-np.inf,np.inf)]
unsafe2=[(200,np.inf),(-np.inf,np.inf),(-np.inf,np.inf)]
unsafeList=[unsafe1,unsafe2]
```

### Step 5: Instantiate the `MoULDyS` engine.

```
mEngine=MoULDyS(A,B,Er,mode,unsafeList,h) # Note: In this example, h is optional.
```

### Step 6: Perform offline monitoring

Let the log be given in the file `/my/location/MoULDyS/data/toyEg_1_interval` (Note: Don't use `.mlog` extension).

```
logFname='toyEg_1_interval'
tp='interval' # Use tp='zonotope', if the log file is represented in zonotope.
reachSets=mEngine.offlineMonitorLogFile(logFname,tp)
```

### Step 7: Visualize the results of monitoring

Following is the color coding of the generated figures:

- Blue: Reachable sets from offline monitoring.
- Black: Logs generated by the offline monitoring.

```
T=2000 # Time step upto which we want to visualize
th1=0 # State variable that is to be visualized
vizCov=5 # Percentage of reachable sets to be visualized. Note: Visualizing all reachable sets is expensive.
#Note: Visualization takes time!
mEngine.vizMonitorLogFile(reachSets,logFname,tp,T,th1,"toyEg_monitor",vizCoverage=vizCov)
```

### Final Code Snippet

```
import os,sys
PROJECT_ROOT = os.environ['MNTR_ROOT_DIR']
sys.path.append(PROJECT_ROOT) # Set the project path

from lib.MoULDySEngine import * # Importing all the functionalities of MoULDyS.

def toyEgOffline():
    ######### Step 1 ########
    A=np.array(
    [
    [1, 0.1],
    [0, 1]
    ]
    )
    B=np.array(
    [
    [0.01],
    [0.01]
    ]
    )
    ######### Step 2 ########
    mode='+'

    ######### Optional ########
    h=0.01

    ######### Step 3 ########
    Er={
    (0,1): [0.99,1.01], # Encoding c
    (0,2): [0.99,1.01], # Encoding d
    }

    ######### Step 4 ########
    unsafe1=[(-np.inf,-200),(-np.inf,np.inf),(-np.inf,np.inf)]
    unsafe2=[(200,np.inf),(-np.inf,np.inf),(-np.inf,np.inf)]
    unsafeList=[unsafe1,unsafe2]
```

```
    ######### Step 5 ########
    mEngine=MoULDyS(A,B,Er,mode,unsafeList,h) # Note: In this example, h is
optional.

    ######### Step 6 ########
    logFname='toyEg_1_interval'
    tp='interval' # Use tp='zonotope', if the log file is represented in
zonotope.
    reachSets=mEngine.offlineMonitorLogFile(logFname,tp)

    ######### Step 7 ########

    T=2000 # Time step upto which we want to visualize
    th1=0 # State variable that is to be visualized
    vizCov=5 # Percentage of reachable sets to be visualized.
    #Note: Visualization takes time!

 mEngine.vizMonitorLogFile(reachSets,logFname,tp,T,th1,"toyEg_monitor",vizCoverag
e=vizCov)


toyEgOffline()
```

This snippet can be found in `/my/location/MoULDyS/src/tutorial/TutorialOfflineMonitoring.py`.

# Online Monitoring

Similar to offline monitoring, we provide the steps to perform online monitoring of the toy example dynamics, given its actual behavior.

### Step 1: Perform steps 0-3 as offline monitoring.

```
import os,sys
PROJECT_ROOT = os.environ['MNTR_ROOT_DIR']
sys.path.append(PROJECT_ROOT) # Set the project path

from lib.MoULDySEngine import * # Importing all the functionalities of MoULDyS.

def toyEgOffline():
    ######### Step 1 ########
    A=np.array(
    [
    [1, 0.1],
    [0, 1]
    ]
    )
    B=np.array(
    [
    [0.01],
    [0.01]
    ]
    )
    ######### Step 2 ########
```

```
    mode='+'

    ######### Optional ########
    h=0.01

    ######### Step 3 ########
    Er={
    (0,1): [0.99,1.01], # Encoding c
    (0,2): [0.99,1.01], # Encoding d
    }
```

### Step 4: Encode the unsafe set (let's use a different one from offline).

Let the unsafe behavior be as follows:

- `state_variable_0 >=20` `AND`
- `state_variable_0 <=-20`

```
unsafe1=[(-np.inf,-20),(-np.inf,np.inf),(-np.inf,np.inf)]
unsafe2=[(20,np.inf),(-np.inf,np.inf),(-np.inf,np.inf)]
unsafeList=[unsafe1,unsafe2]
```

### Step 5: Instantiate the `MoULDyS` engine.

```
mEngine=MoULDyS(A,B,Er,mode,unsafeList,h) # Note: In this example, h is optional.
```

### Step 6: Perform online monitoring.

- Let the actual behavior be given in file `/my/location/MoULDyS/data/toyEg_1_interval` (Note: Don't use `.mbeh` extension)
- The actual behavior type can either be interval or zonotope.

```
logFname='toyEg_1_interval'
tp='interval' # Use tp='zonotope', if the log file is represented in zonotope.
(reachSets,logs)=mEngine.onlineMonitorBehFile(logFname,tp)
```

### Step 7: Visualize the results of monitoring.

Following is the color coding of the generated figures:

- Black: Represents logs.
- Blue: Represents reachable sets.
- Red: Unsafe region.

```
T=2000 # Time step upto which we want to visualize
th1=0 # State variable that is to be visualized
vizCov=5 # Percentage of reachable sets to be visualized. Note: Visualizing all reachable sets is expensive.
#Note: Visualization takes time!
mEngine.vizMonitor(reachSets,logs,tp,T,th1,"toyEg_monitor",vizCoverage=vizCov)
```

### Final Code Snippet

```
import os,sys
PROJECT_ROOT = os.environ['MNTR_ROOT_DIR']
sys.path.append(PROJECT_ROOT) # Set the project path

from lib.MoULDySEngine import * # Importing all the functionalities of MoULDyS.

def toyEgOnline():
    ######### Step 1-3 ########
    A=np.array(
    [
    [1, 0.1],
    [0, 1]
    ]
    )
    B=np.array(
    [
    [0.01],
    [0.01]
    ]
    )
    mode='+'
    h=0.01 # This step is not needed for this example.

    ######### Step 3 ########
    '''
    Step 3: Encode the uncertainties in the dynamics (i.e, c and d)
    '''
    Er={
    (0,1): [0.99,1.01], # Encoding c
    (0,2): [0.99,1.01], # Encoding d
    }

    ######### Step 4 ########
    '''
    Step 4: Encode the unsafe set.
    Let the unsafe behavior be as follows:
    * state_variable_0 >=20 AND
    * state_variable_0 <=-20
    Unsafe sets can be encoded as intervals
    '''
    unsafe1=[(-np.inf,-20),(-np.inf,np.inf),(-np.inf,np.inf)]
    unsafe2=[(20,np.inf),(-np.inf,np.inf),(-np.inf,np.inf)]
    unsafeList=[unsafe1,unsafe2]

    ######### Step 5 ########
    mEngine=MoULDyS(A,B,Er,mode,unsafeList,h) # Note: In this example, h is
optional.

    ######### Step 6 ########
    logFname='toyEg_1_interval'
    tp='interval' # Use tp='zonotope', if the log file is represented in
zonotope.
    (reachSets,logs)=mEngine.onlineMonitorBehFile(logFname,tp)
```

```
    ######### Step 7 ########
    T=2000 # Time step upto which we want to visualize
    th1=0 # State variable that is to be visualized
    vizCov=5 # Percentage of reachable sets to be visualized. Note: Visualizing
all reachable sets is expensive.
    #Note: Visualization takes time!

 mEngine.vizMonitor(reachSets,logs,tp,T,th1,"toyEg_monitor",vizCoverage=vizCov)

toyEgOnline()
```

This snippet can be found in `/my/location/MoULDyS/src/tutorial/TutorialOnlineMonitoring.py`.

## Compare Online and Offline Monitoring

### Step 1: Perform steps 0-5 as online monitoring.

```
import os,sys
PROJECT_ROOT = os.environ['MNTR_ROOT_DIR']
sys.path.append(PROJECT_ROOT) # Set the project path

from lib.MoULDySEngine import * # Importing all the functionalities of MoULDyS.


def toyEgOnline():
    ######### Step 1-3 ########
    A=np.array(
    [
    [1, 0.1],
    [0, 1]
    ]
    )
    B=np.array(
    [
    [0.01],
    [0.01]
    ]
    )
    mode='+'
    h=0.01 # This step is not needed for this example.

    ######### Step 3 ########
    '''
    Step 3: Encode the uncertainties in the dynamics (i.e, c and d)
    '''
    Er={
    (0,1): [0.99,1.01], # Encoding c
    (0,2): [0.99,1.01], # Encoding d
    }

    ######### Step 4 ########
    '''
```

```
    Step 4: Encode the unsafe set.
    Let the unsafe behavior be as follows:
    * state_variable_0 >=20 AND
    * state_variable_0 <=-20
    Unsafe sets can be encoded as intervals
    '''
    unsafe1=[(-np.inf,-20),(-np.inf,np.inf),(-np.inf,np.inf)]
    unsafe2=[(20,np.inf),(-np.inf,np.inf),(-np.inf,np.inf)]
    unsafeList=[unsafe1,unsafe2]

    ######### Step 5 ########
    mEngine=MoULDyS(A,B,Er,mode,unsafeList,h) # Note: In this example, h is
optional.
```

### Step 2: Perform both offline and online monitoring.

```
logFname='toyEg_1_interval'
tp='interval' # Use tp='zonotope', if the log file is represented in zonotope.
reachSets=mEngine.offlineMonitorLogFile(logFname,tp)
(reachSetsOnline,logsOnline)=mEngine.onlineMonitorBehFile(logFname,tp)
```

### Step 3: Visualize both the results from online and offline monitoring.

Following is the color coding of the generated figures:

- Blue: Reachable sets from online monitoring.
- Black: Logs generated by the online monitoring.
- Green: Reachable sets from offline monitoring.
- Magenta: Logs provided to the offline monitoring.
- Red: Unsafe region.

```
T=2000 # Time step upto which we want to visualize
th1=0 # State variable that is to be visualized
vizCov=5 # Percentage of reachable sets to be visualized. Note: Visualizing all
reachable sets is expensive.
mEngine.vizCompMonitorLogFile(reachSets,logFname,reachSetsOnline,logsOnline,tp,T,
th1,"viz_test",vizCov)
```

### Final Code Snippet

```
import os,sys
PROJECT_ROOT = os.environ['MNTR_ROOT_DIR']
sys.path.append(PROJECT_ROOT) # Set the project path

from lib.MoULDySEngine import * # Importing all the functionalities of MoULDyS.


def toyEgCompare():
    ######### Step 1-2 ########
    A=np.array(
    [
    [1, 0.1],
    [0, 1]
    ]
```

```
    )
    B=np.array(
    [
    [0.01],
    [0.01]
    ]
    )
    mode='+'
    h=0.01 # This step is not needed for this example.

    ######### Step 3 ########
    '''
    Step 3: Encode the uncertainties in the dynamics (i.e, c and d)
    '''
    Er={
    (0,1): [0.99,1.01], # Encoding c
    (0,2): [0.99,1.01], # Encoding d
    }

    ######### Step 4 ########
    '''
    Step 4: Encode the unsafe set.
    Let the unsafe behavior be as follows:
    * state_variable_0 >=20 AND
    * state_variable_0 <=-20
    Unsafe sets can be encoded as intervals
    '''
    unsafe1=[(-np.inf,-20),(-np.inf,np.inf),(-np.inf,np.inf)]
    unsafe2=[(20,np.inf),(-np.inf,np.inf),(-np.inf,np.inf)]
    unsafeList=[unsafe1,unsafe2]

    ######### Step 5 ########
    mEngine=MoULDyS(A,B,Er,mode,unsafeList,h) # Note: In this example, h is
optional.

    ######### Step 6 ########
    logFname='toyEg_1_interval'
    tp='interval' # Use tp='zonotope', if the log file is represented in
zonotope.
    reachSets=mEngine.offlineMonitorLogFile(logFname,tp)
    (reachSetsOnline,logsOnline)=mEngine.onlineMonitorBehFile(logFname,tp)

    ######### Step 7 ########
    T=2000 # Time step upto which we want to visualize
    th1=0 # State variable that is to be visualized
    vizCov=5 # Percentage of reachable sets to be visualized. Note: Visualizing
all reachable sets is expensive.

 mEngine.vizCompMonitorLogFile(reachSets,logFname,reachSetsOnline,logsOnline,tp,T
,th1,"viz_test",vizCov)
```

This snippet can be found in [/my/location/MoULDyS/src/tutorial/TutorialCompareMonitoring.py](/my/location/MoULDyS/src/tutorial/TutorialCompareMonitoring.py).

# Generate Logs (Optional)

This optional section discusses how to generate logs.

### Step 1: Perform steps 0-3 as offline monitoring

```
import os,sys
PROJECT_ROOT = os.environ['MNTR_ROOT_DIR']
sys.path.append(PROJECT_ROOT) # Set the project path

from lib.MoULDySEngine import * # Importing all the functionalities of MoULDyS.

def toyEgOffline():
    ######### Step 1 ########
    A=np.array(
    [
    [1, 0.1],
    [0, 1]
    ]
    )
    B=np.array(
    [
    [0.01],
    [0.01]
    ]
    )
    ######### Step 2 ########
    mode='+'

    ######### Optional ########
    h=0.01

    ######### Step 3 ########
    Er={
    (0,1): [0.99,1.01], # Encoding c
    (0,2): [0.99,1.01], # Encoding d
    }
```

### Step 2: Instantiate the `MoULDyS` engine.

```
mEngine=MoULDyS(A,B,Er,mode,unsafeList,h) # Note: In this example, h is optional.
```

### Step 3: Generate random logs

```
initialSet=[(0,0),(0,0),(-0.1,0.1)] # Initial Set, represented as an interval
pr=1 # Probability of logging
fname='toyEg'
T=2000 # Time upto which logs are generated
tp='interval' # Use tp='zonotope' for creating logs with zonotopes
(log,actualBehavior)=mEngine.genLogFile(initialSet,T,fname,tp,pr)
```

## Final Code Snippet

```
import os,sys
PROJECT_ROOT = os.environ['MNTR_ROOT_DIR']
sys.path.append(PROJECT_ROOT) # Set the project path

from lib.MoULDySEngine import * # Importing all the functionalities of MoULDyS.

def toyEgOffline():
    ######### Step 1 ########
    A=np.array(
    [
    [1, 0.1],
    [0, 1]
    ]
    )
    B=np.array(
    [
    [0.01],
    [0.01]
    ]
    )
    ######### Step 2 ########
    mode='+'

    ######### Optional ########
    h=0.01

    ######### Step 3 ########
    Er={
    (0,1): [0.99,1.01], # Encoding c
    (0,2): [0.99,1.01], # Encoding d
    }
    mEngine=MoULDyS(A,B,Er,mode,unsafeList,h) # Note: In this example, h is
optional.
    initialSet=[(0,0),(0,0),(-0.1,0.1)] # Initial Set, represented as an interval
    pr=1 # Probability of logging
    fname='toyEg'
    T=2000 # Time upto which logs are generated
    tp='interval' # Use tp='zonotope' for creating logs with zonotopes
    (log,actualBehavior)=mEngine.genLogFile(initialSet,T,fname,tp,pr)
```

This will generate logs file (extension `.mlog`) and behavior file (extension `.mbeh`) in `/my/location/data/`.