# FMI – Version 1.0
# FMI for Model Exchange の
# モデルについて

オープンCAE勉強会＠関東（流体など）

finback

# FMI-1.0 FMI for Model Exchangeの概要

(Functional Mock-up Interface) https://www.fmi-standard.org/start

**様々なダイナミックシステムのシミュレーションツール間でモデルを交換するための規格である。**

https://www.fmi-standard.org/tools (の一部)

Total numbers | Filter | ⇄ ModelExchange | CoSimulation

| Tools supporting FMI | FMI Version | Export | Import | Slave | Master | Notes |
|---|---|---|---|---|---|---|
| OpenModelica | FMI_2.0 | Available | Available | Planned | Planned | Open source Modelica environment from OSMC |
| | FMI_1.0 | Available 3 | Available | Planned | Available | |
| Ptolemy II | FMI_1.0 | | | | Planned | Software environment for design and analysis of heterogeneous systems. |
| PyFMI | FMI_2.0 | | Available 23 | | Available 32 | For Python via the open source package PyFMI from Modelon. Also available as part of the JModelica.org platform. |
| | FMI_1.0 | | Available 63 | | Available 59 | |
| RecurDyn | FMI_1.0 | Planned | Planned | Planned | Available | High End Multi Flexible Body Dynamcis Software from FunctionBay |
| Reference FMUs | FMI_1.0 | Planned | | Planned | | Reference FMUs supplied by enthusiasts and volunteers to show case specific FMU features |
| Scilab/Xcos FMU wrapper | FMI_2.0 | Planned | Planned | Planned | Planned | FMI support for Scilab / Xcos a free and open source software for numerical computation. |
| | FMI_1.0 | Available | Available | Available | Available | |

どんな形で交換するのか？

どんな概念のモデルが
交換できるのか？

仕様書とFMU SDK を調べる

どのように使うのか？

- OpenModelica
- Scilab/Xcos FMU wrapper
- PyFMI

で使ってみる。

**仕様書**　https://www.fmi-standard.org/downloads

FMI 1.0 FMI for Model Exchange  2010-07-26 MODELISAR
FMI 1.0 FMI for Co-Simulation  2010-10-12 MODELISAR
FMI 2.0 FMI for Model Exchange & Co-Simulation 2014-07-25  Modelica Association

# FMI 1.0 for Model Exchangeのモデル概要

## FMU (Functional Mock-up Unit)

モデル交換の形

FMIの規格のモデルは、**XMLファイル**、**Cのソースコード**、**ダイナミックリンクライブラリ**などをまとめたZIPファイル（拡張子fmu)の形で配布され、交換される。これをFMUと呼ぶ。

**FMUの例（FMUSDKの bouncingBall.fmu の内容）**


**modelDescription.xml**

モデルの変数や機能などについて記述されたXMLファイル

**ダイナミックリンクライブラリ**

ソルバー（シミュレーションツール）によってロードされる。ソルバーとモデルの間のインターフェース関数が実装されている。いろんなプラットフォームのものを同梱できる。

**ダイナミックリンクライブラリのソースコード**

ソースコードの同梱は必須ではない。

# FMI 1.0 for Model Exchangeのモデル概要

仕様書より

**FMU（FMI規格のモデル)のインスタンス**

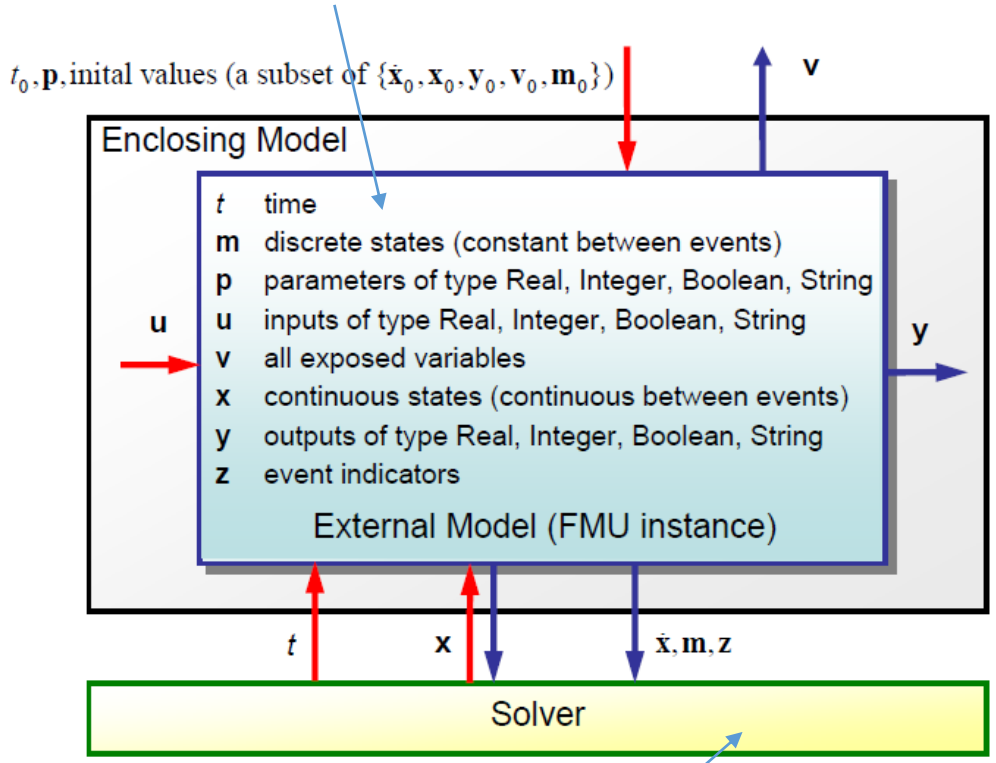



**Figure 1**: Data flow between the components, for details see section 2.1.
Blue arrows: Information provided by the FMU.
Red arrows: information provided to the FMU.

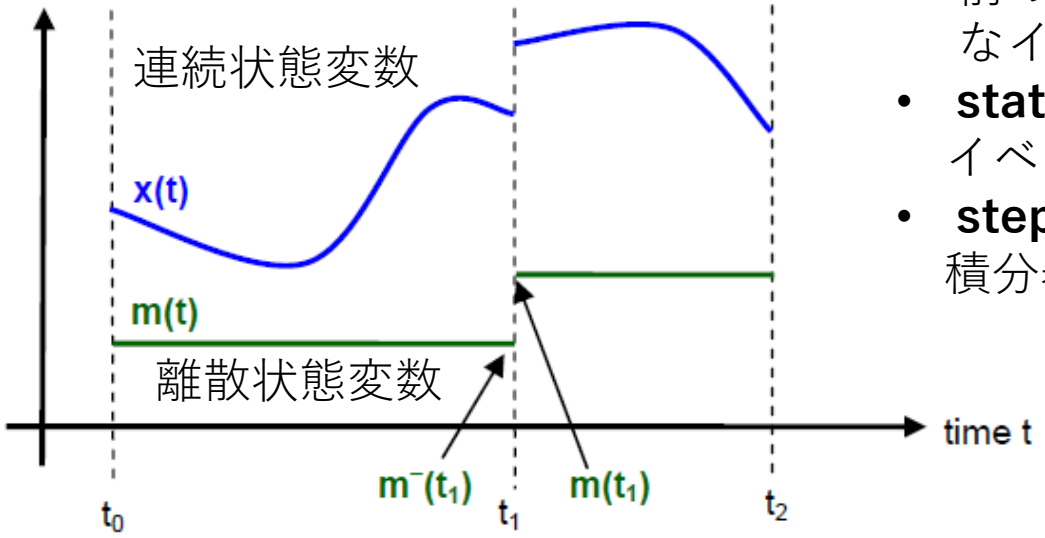
(仕様書 P.6)

- **時間**
  $t$
- **モデルの状態を表す変数**
  - $\mathbf{x}$ 連続的状態変数、モデルは $\dot{\mathbf{x}}$ を計算する機能を持つ
  - $\mathbf{m}$ 離散的状態変数、モデルは次のイベント時の値を計算する機能を持つ
- **パラメータ**
  - $\mathbf{p}$ シミュレーション中に変化しない変数
- **入出力変数**
  - $\mathbf{u}$ 入力変数、モデルの外部から与えられる変数
  - $\mathbf{y}$ 出力変数、モデル外部へ出力される変数
- **イベントインジケータ**
  - $\mathbf{z}$ モデルの何等かの状態を表す連続変数で符号が変化するときにイベントが発生する。

**OpenModelica, PyFMIなど** 仕様書ではsimulator, simulation tool, solver, integrator がほぼ同義で使われている。

# FMI 1.0 for Model Exchangeのモデル概要

仕様書より



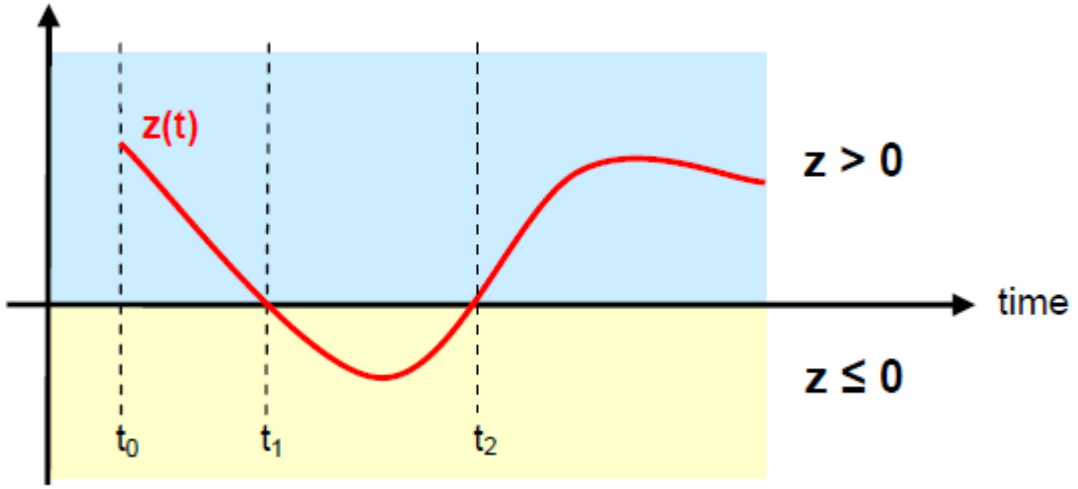

$t = t_0, t_1, t_2, \cdots, t_i, \cdots, t_n$　**イベント**

2 つのイベント間は状態変数 $\mathbf{x}(t), \mathbf{m}(t)$ は連続

（図は仕様書 p.10より）

**イベントの種類**

- **time event**
  前のイベント時刻に予め次のイベント時刻が決定されるようなイベント
- **state event**
  イベントインジケータの符号の変化によって生じるイベント
- **step event**
  積分器(ソルバー)のステップの計算の完了に関係するイベント




**イベントインジケータ**

# FMU SDK 2.0.4でモデルの具体例を見る

**FMU SDK**（https://www.qtronic.de/jp/fmusdk.html）

- FMUの基本的な使い方を示すためのフリーのSDKである。
- FMI 1.0 と FMI 2.0 の両方に対応している。
- QTronic より、BSDライセンスで提供されている。
- 比較的簡単なサンプルFMUとシミュレーションツールのソースコードが含まれている。
- 開発環境としてMicrosoft Visual Studio 2005, 2008, 2010, 2012 のいずれかを必要とする。
- Linux 用に移植されたものも提供されている。

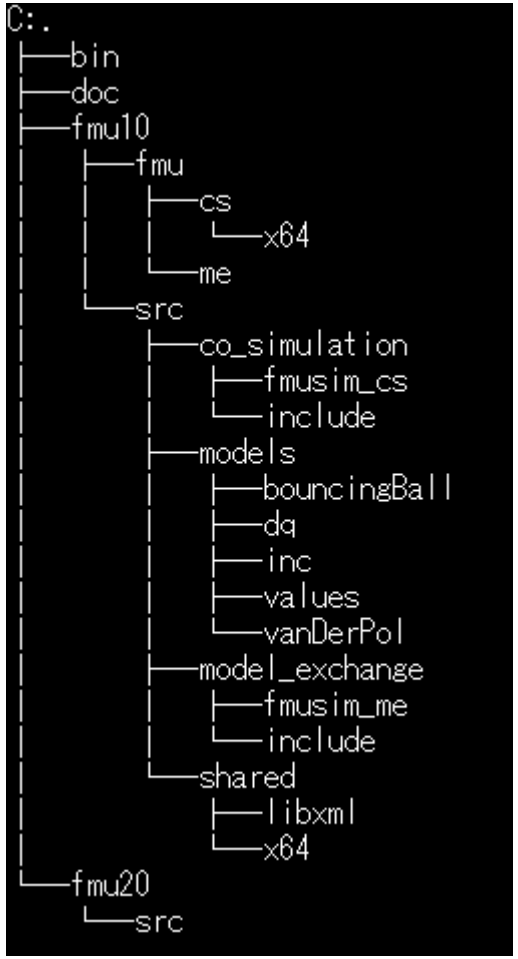
**サンプルFMU**（https://resources.qtronic.de/fmusdk/FmuSdk_reference.html）

- dq　Dehlquist test function x = -k der(x)
- inc　1秒毎に更新される整数カウンタのモデル
- values　すべてのスカラーFMUデータ型の使用法を示すモデル
- vanDerPol　２つの連続状態変数を持つODEのモデル
- bouncingBall　イベントが定義された跳ね返るボールのモデル

おそらく
バッチファイルを
書き換えれば
もっと新しい
バージョンでも
大丈夫だと思う。

# FMU SDK - FMUの作成方法

FMUSDKのディレクトリツリー




FMI1.0のサンプルモデルのソースコードのディレクトリ

① fmusdk.zip をダウンロードして解凍する。
② fmu10¥src¥models にモデル名のサブディレクトリを作成し、以下のファイルを作成する。

- モデル名.c　　ソースファイル
- modelDescription.xml　XMLファイル
  （最後の行</fmiModelDescription> を除く）

③ その他、オプションとして説明用のhtmlファイル、アイコン用ファイルなどを置く。
④ fmu10¥src¥modelsで以下のようにバッチファイルを実行する

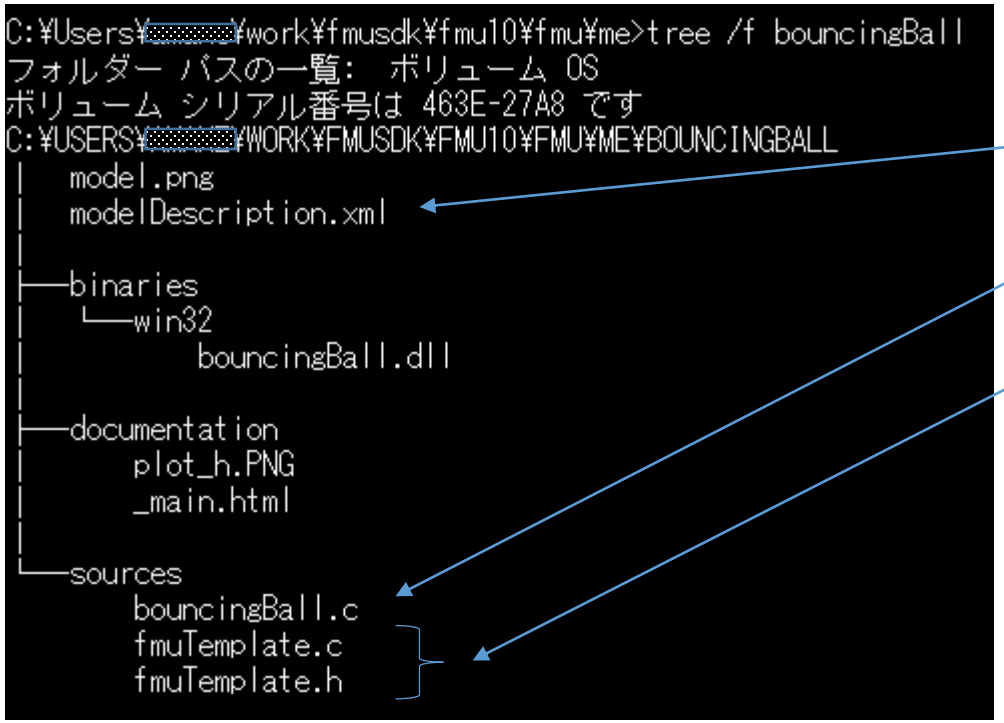
**build_fmu　タイプ　モデルのディレクトリ名　[-win64]**

- タイプは cs（Co-Simulation）または me（Model Exchange）
- -win64 をつけると64bit のDLLが生成される。
  つけなければ 32bit のDLLが生成される。

⑤ fmu10¥fmu¥cs または fmu10¥fmu¥me に、モデル名.fmu が生成される。

# FMU SDK – bouncingBall.fmu

fmu10¥src¥modelsで build_fmu me bouncingBall を実行して、fmu10¥fmu¥meにFMUを生成する。
生成された bouncingBall.fmu をコピーして bouncingBall.zip を作成し解凍する。




内容を見る

**XMLファイル**

**bouncingBallモデル固有のソースコード(100行程度)**

**他のモデルと共通な部分のソースコード**

これらと、FMI 1.0 for Model Exchange とともに配布されている、ヘッダファイル

- **fmiModelTypes.h**
- **fmiModelFunctions.h**

を加えて全てのソースコードとなる。

# FMU SDK – bouncingBall.c

**モデルの概要（最初のコメント部分）**

```
/* ---------------------------------------------------------------------------*
* Sample implementation of an FMU - a bouncing ball.
* This demonstrates the use of state events and reinit of states.
* Equations:
*  der(h) = v;
*  der(v) = -g;
*  when h<0 then v := -e * v;
*  where
*    h      height [m], used as state, start = 1
*    v      velocity of ball [m/s], used as state
*    der(h) velocity of ball [m/s]
*    der(v) acceleration of ball [m/s2]
*    g      acceleration of gravity [m/s2], a parameter, start = 9.81
*    e      a dimensionless parameter, start = 0.7
*
* Copyright QTronic GmbH. All rights reserved.
* ---------------------------------------------------------------------------*/
```

**運動方程式**

$$\frac{d^2h}{dt^2} = -g \Rightarrow \begin{cases} \frac{dh}{dt} = v \\ \frac{dv}{dt} = -g \end{cases}$$

**イベント　地面との衝突**

$h < 0$　なら　$v := -e * v$

**変数**

イベントインジケータ 1 個

| | | |
|---|---|---|
| $h$ | ボールの高さ[m] | 連続状態変数 2個 |
| $v$ | ボールの速度[m/s] | |
| $der(h)$ | 連続状態変数の時間微分 | |
| $der(v)$ | | |
| $g$ | 重力加速度[m/s2] | パラメータ 2 個 |
| $e$ | 無次元パラメータ | |

実数変数 6 個

# FMU SDK – bouncingBall.c

- **モデル識別情報**
- **モデル変数の数**

```
// define class name and unique id
#define MODEL_IDENTIFIER bouncingBall
#define MODEL_GUID "{8c4e810f-3df3-4a00-8276-176fa3c9f003}"

// define model size
#define NUMBER_OF_REALS 6
#define NUMBER_OF_INTEGERS 0
#define NUMBER_OF_BOOLEANS 0
#define NUMBER_OF_STRINGS 0
#define NUMBER_OF_STATES 2
#define NUMBER_OF_EVENT_INDICATORS 1

// include fmu header files, typedefs and macros
#include "fmuTemplate.h"
```

モデル識別情報、XMLファイルにも記述される

実数変数 6 個

状態変数2個

イベントインジケータ 1 個

# FMU SDK – bouncingBall.c

- **6個の実数変数とその変数参照番号(value reference)の定義**
- **状態変数の定義**

```
// define all model variables and their value references
// conventions used here:
// - if x is a variable, then macro x_ is its variable reference
// - the vr of a variable is its index in array  r, i, b or s
// - if k is the vr of a real state, then k+1 is the vr of its derivative
#define h_     0
#define der_h_  1
#define v_     2
#define der_v_  3
#define g_     4
#define e_     5

// define initial state vector as vector of value references
#define STATES { h_, v_ }
```

FMU SDK では状態変数とその微分は連番にする必要がありそう！

- **実数変数を参照する関数の定義**

```
// called by fmiGetReal, fmiGetContinuousStates and fmiGetDerivatives
fmiReal getReal(ModelInstance* comp, fmiValueReference vr){
  switch (vr) {
    case h_    : return r(h_);
    case der_h_ : return r(v_);
    case v_    : return r(v_);
    case der_v_ : return r(der_v_);
    case g_    : return r(g_);
    case e_    : return r(e_);
    default: return 0;
  }
}
```

$\frac{dh}{dt} = v$ を返す。

$\frac{dv}{dt} = der(v) = -g$ を返す。

ここで方程式が実装されている。 $der(v) = -g$ は初期化処理

r(vr)は実際はcompという構造体の要素として格納される

```
// macros used to define variables
#define  r(vr) comp->r[vr]
#define  i(vr) comp->i[vr]
#define  b(vr) comp->b[vr]
#define  s(vr) comp->s[vr]
#define pos(z) comp->isPositive[z]
#define copy(vr, value) setString(comp, vr, value)
```

(fmuTemplate.hより)

# FMU SDK – bouncingBall.c

- **変数の初期化**

```
// called by fmiInstantiateModel
// Set values for all variables that define a start value
// Settings used unless changed by fmiSetX before fmiInitialize
void setStartValues(ModelInstance *comp) {
    r(h_)    = 1;
    r(v_)    = 0;
    r(der_v_) = -9.81;
    r(g_)    = 9.81;
    r(e_)    = 0.7;
    pos(0) = r(h_) > 0;
}
```

1個目のイベントインジケータの正負を表す。

```
// macros used to define variables
#define  r(vr) comp->r[vr]
#define  i(vr) comp->i[vr]
#define  b(vr) comp->b[vr]
#define  s(vr) comp->s[vr]
#define pos(z) comp->isPositive[z]
#define copy(vr, value) setString(comp, vr, value)
```

(fmuTemplate.hより)

- **初期化**

```
// called by fmiInitialize() after setting eventInfo to defaults
// Used to set the first time event, if any.
void initialize(ModelInstance* comp, fmiEventInfo* eventInfo) {
    r(der_v_) = -r(g_);
}
```

速度の微分として重力加速度が設定されている。
(パラメータ)

- **イベントインジケータの状態を表す関数の定義**

```
// offset for event indicator, adds hysteresis and prevents z=0 at restart
#define EPS_INDICATORS 1e-14

fmiReal getEventIndicator(ModelInstance* comp, int z) {
    switch (z) {
        case 0 : return r(h_) + (pos(0) ? EPS_INDICATORS : -EPS_INDICATORS);
        default: return 0;
    }
}
```

z=0 のとき1個目のインジケータの状態を返す。
絶対値が$1 \times 10^{-14}$より小さくならないように
処理されている。

# FMU SDK – bouncingBall.c

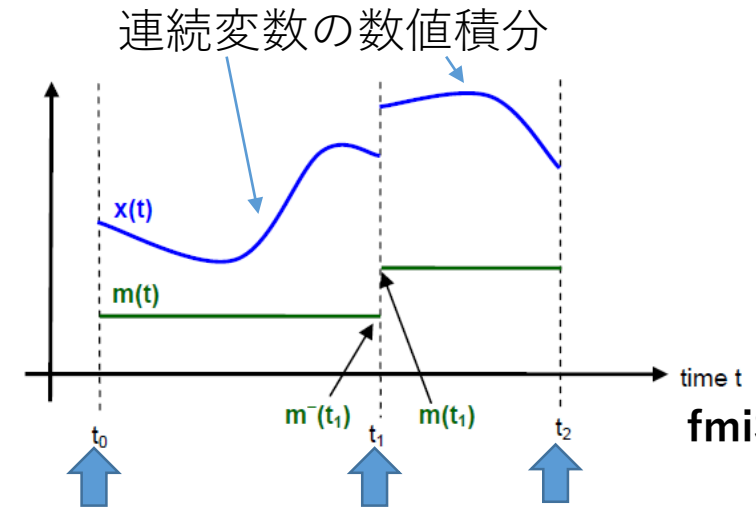
- **イベントの処理**

```
// Used to set the next time event, if any.
void eventUpdate(ModelInstance* comp, fmiEventInfo* eventInfo) {
  if (pos(0)) {
    r(v_) = - r(e_) * r(v_);
  }
  pos(0) = r(h_) > 0;
  eventInfo->iterationConverged  = fmiTrue;
  eventInfo->stateValueReferencesChanged = fmiFalse;
  eventInfo->stateValuesChanged  = fmiTrue;
  eventInfo->terminateSimulation = fmiFalse;
  eventInfo->upcomingTimeEvent   = fmiFalse;
}

// include code that implements the FMI based on the above definitions
#include "fmuTemplate.c"
```

$v := -e * v$

状態変数　$v$ が変化したことを表す。

# FMU SDK - イベント処理




## step event について

連続状態変数は２つのイベントの間で連続的に変化する。一般的には、イベント間隔よりも小さな時間ステップでソルバーが時間積分を行っている。ステップ毎の時間積分が完了するとソルバーから関数

**fmiStatus fmiCompletedIntegratorStep(fmiComponent c, fmiBoolean* callEventUpdate)**

がコールされる。このときモデルが、第２引数で
**callEventUpdate = fmiTrue** を返したら、**step event** が発生する。

```
fmiStatus fmiCompletedIntegratorStep(fmiComponent c, fmiBoolean* callEventUpdate){
    ModelInstance* comp = (ModelInstance *)c;
    if (invalidState(comp, "fmiCompletedIntegratorStep", modelInitialized))
        return fmiError;
    if (nullPointer(comp, "fmiCompletedIntegratorStep", "callEventUpdate", callEventUpdate))
        return fmiError;
    if (comp->loggingOn) comp->functions.logger(c, comp->instanceName, fmiOK, "log",
        "fmiCompletedIntegratorStep");
    *callEventUpdate = fmiFalse;
    return fmiOK;
}
```

FMU SDK のコードでは step event は発生しないようである。

(fmuTemplate.cより)

# FMU SDK - イベント処理

time event, state event, step event のいずれかが発生するとソルバーは関数

**fmiStatus fmiEventUpdate(fmiComponent c, fmiBoolean intermediateResults, fmiEventInfo* eventInfo)**

をコールし、モデルは第３引数の**fmiEventInfo**型の構造体のポインタを返す。

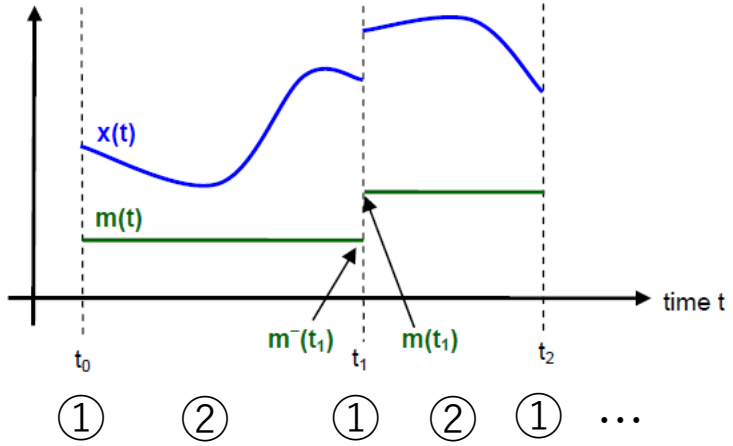
**fmiEventInfo 構造体**

```
typedef struct {
    fmiBoolean iterationConverged;
    fmiBoolean stateValueReferencesChanged;
    fmiBoolean stateValuesChanged;
    fmiBoolean terminateSimulation;
    fmiBoolean upcomingTimeEvent;
    fmiReal    nextEventTime;
} fmiEventInfo;
```

（fmiModelFunctions.hより）

- **iterationConverged = fmiFalse**の場合、これが fmiTrueとなるまで繰り返し fmiEventUpdateがコールされる。これにより、モデル内で繰返し計算が必要な処理を行うことができる。
- **stateValueReferenceChanged = fmiTrue** はモデルの状態変数の組み合わせを別の変数に変える場合である。
- **stateValueChanged=fmiTrue** は、状態変数が不連続に変化する場合など状態変数の更新が必要な場合である。
- **terminateSimulation=fmiTrue** は、シミュレーションを中断する場合である。
- **upcommingTimeEvent=fmiTrue** は、次の time event の時刻が**nextEventTime**にセットされていることを示す。

# FMU SDK – シミュレーションの進行




結局、

① $t = t_i, i = 0,1,2,\cdots,n$ **イベント処理ループ**

② $t_i < t < t_{i+1}$ **ソルバーによる状態変数の時間積分**

の繰り返しによってシミュレーションが進行すると考えられる。

①では、何らかの物理法則が成立するまでイタレーションを行うような処理も可能である。

仕様書より

**モデルの状態遷移**

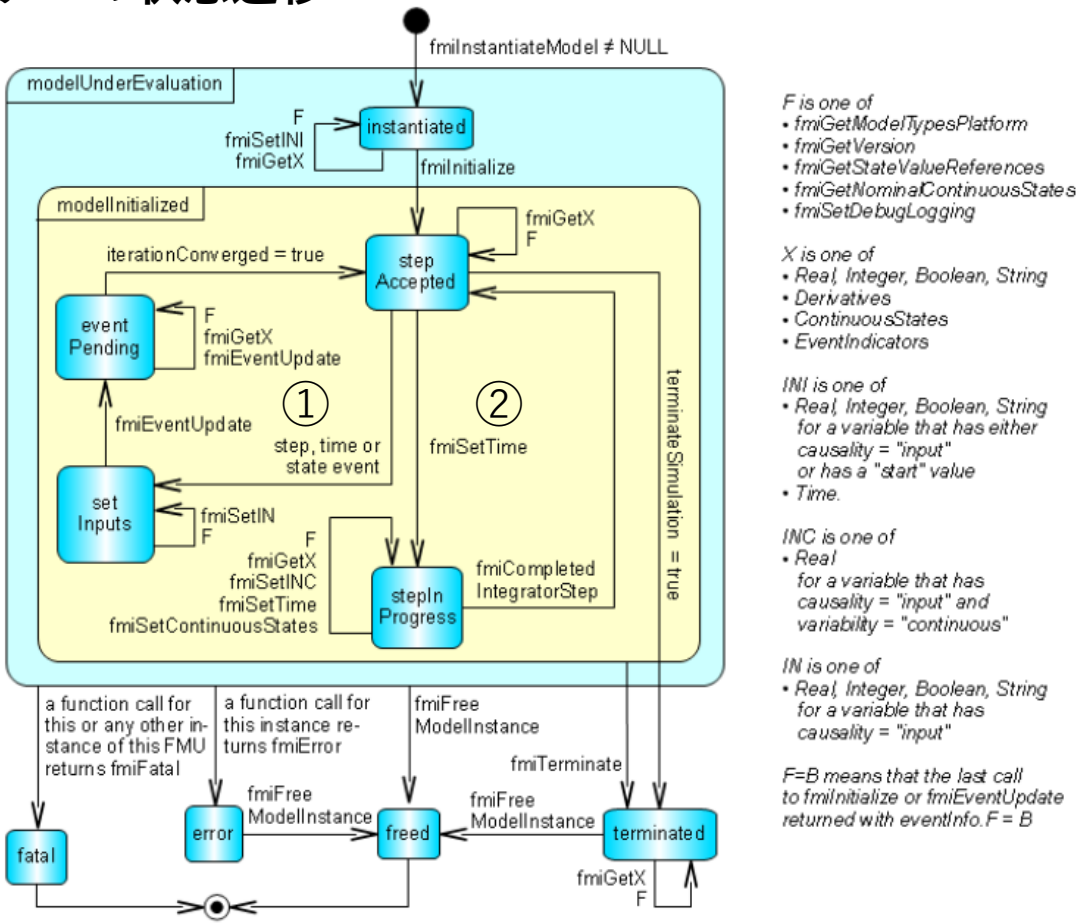



**Figure 4:** Calling sequence of Model Exchange C-functions in form of an UML 2.0 state machine.

(仕様書 P.22より)

# FMU SDK – modelDescription.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<fmiModelDescription
 fmiVersion="1.0"
 modelName="bouncingBall"
 modelIdentifier="bouncingBall"
 guid="{8c4e810f-3df3-4a00-8276-176fa3c9f003}"
 numberOfContinuousStates="2"
 numberOfEventIndicators="1">
<ModelVariables>
 <ScalarVariable name="h" valueReference="0" description="height, used as state">
   <Real start="1" fixed="true"/>
 </ScalarVariable>
 <ScalarVariable name="der(h)" valueReference="1" description="velocity of ball">
   <Real/>
 </ScalarVariable>
 <ScalarVariable name="v" valueReference="2" description="velocity of ball, used as state">
   <Real/>
 </ScalarVariable>
 <ScalarVariable name="der(v)" valueReference="3" description="acceleration of ball">
   <Real/>
 </ScalarVariable>
 <ScalarVariable name="g" valueReference="4" description="acceleration of gravity"
                 variability="parameter">
   <Real start="9.81" fixed="true"/>
 </ScalarVariable>
 <ScalarVariable name="e" valueReference="5" description="dimensionless parameter"
                 variability="parameter">
   <Real start="0.7" fixed="true"/>
 </ScalarVariable>
</ModelVariables>

</fmiModelDescription>
```

**モデル識別情報**

**連続状態変数** 2個

**イベントインジケータ** 1個

**各変数に対する情報**

変数名(name)
変数参照番号(valueReference)
概要(description)
種類(variability など)
型や初期値(start=…) など

# FMU SDK – モデルのインスタンス化

FMU SDK では、モデル変数等は
次のような構造体で表現される。

```
typedef struct {
   fmiReal    *r;
   fmiInteger *i;
   fmiBoolean *b;
   fmiString  *s;
   fmiBoolean *isPositive;
   fmiReal time;
   fmiString instanceName;
   fmiString GUID;
   fmiCallbackFunctions functions;
   fmiBoolean loggingOn;
   ModelState state;
#ifdef FMI_COSIMULATION
   fmiEventInfo eventInfo;
#endif
} ModelInstance;
```

(fmuTemplate.hより)

関数 **fmiInstantiateModel** がコールされると
次のような処理によってメモリが確保される。

```
comp = (ModelInstance *)functions.allocateMemory(1, sizeof(ModelInstance));
if (comp) {
   comp->r = (fmiReal *)functions.allocateMemory(NUMBER_OF_REALS,    sizeof(fmiReal));
   comp->i = (fmiInteger *)functions.allocateMemory(NUMBER_OF_INTEGERS, sizeof(fmiInteger));
   comp->b = (fmiBoolean *)functions.allocateMemory(NUMBER_OF_BOOLEANS, sizeof(fmiBoolean));
   comp->s = (fmiString *)functions.allocateMemory(NUMBER_OF_STRINGS,  sizeof(fmiString));
   comp->isPositive = (fmiBoolean *)functions.allocateMemory(NUMBER_OF_EVENT_INDICATORS, sizeof(fmiBoolean));
   comp->instanceName = (char *)functions.allocateMemory(1 + strlen(instanceName), sizeof(char));
   comp->GUID = (char *)functions.allocateMemory(1 + strlen(GUID), sizeof(char));
}
if (!comp || !comp->r || !comp->i || !comp->b || !comp->s || !comp->isPositive
   || !comp->instanceName || !comp->GUID) {
   functions.logger(NULL, instanceName, fmiError, "error",
         "%s: Out of memory.", fname);
   return NULL;
```

(fmuTemplate.cより)

# FMI 1.0 for Model Exchange の関数

FMUのダイナミックリンクライブラリに実装される関数
シミュレーションツールからコールされる。

**FMI 1.0 for Model Exchange と FMI 1.0 for Co-Simulation で共通の関数**

FMIのバージョンを返す
const char* fmiGetVersion()

変数に値を設定する
fmiStatus fmiSetDebugLogging(fmiComponent c, fmiBoolean loggingOn)
fmiStatus fmiSetReal(fmiComponent c, const fmiValueReference vr[], size_t nvr, const fmiReal value[])
fmiStatus fmiSetInteger(fmiComponent c, const fmiValueReference vr[], size_t nvr, const fmiInteger value[])
fmiStatus fmiSetBoolean(fmiComponent c, const fmiValueReference vr[], size_t nvr, const fmiBoolean value[])
fmiStatus fmiSetString(fmiComponent c, const fmiValueReference vr[], size_t nvr, const fmiString value[])

変数の値を取得する
fmiStatus fmiGetReal(fmiComponent c, const fmiValueReference vr[], size_t nvr, fmiReal value[])
fmiStatus fmiGetInteger(fmiComponent c, const fmiValueReference vr[], size_t nvr, fmiInteger value[])
fmiStatus fmiGetBoolean(fmiComponent c, const fmiValueReference vr[], size_t nvr, fmiBoolean value[])
fmiStatus fmiGetString(fmiComponent c, const fmiValueReference vr[], size_t nvr, fmiString  value[])

# FMI 1.0 for Model Exchange の関数

**FMI 1.0 for Model Exchange のみの関数(1)**

モデルの変数のタイプを返す。
const char* fmiGetModelTypesPlatform()

モデルのインスタンス化、シミュレーションの初期化
fmiComponent fmiInstantiateModel(fmiString instanceName, fmiString GUID, fmiCallbackFunctions functions, fmiBoolean loggingOn)
fmiStatus fmiInitialize(fmiComponent c, fmiBoolean toleranceControlled, fmiReal relativeTolerance, fmiEventInfo* eventInfo)

シミュレーションの中止、インスタンスのメモリ解放
fmiStatus fmiTerminate(fmiComponent c)
void fmiFreeModelInstance(fmiComponent c)

連続変数の状態の設定と取得、時間微分の取得
fmiStatus fmiSetContinuousStates(fmiComponent c, const fmiReal x[], size_t nx)
fmiStatus fmiGetContinuousStates(fmiComponent c, fmiReal states[], size_t nx)
fmiStatus fmiGetNominalContinuousStates(fmiComponent c, fmiReal x_nominal[], size_t nx)
fmiStatus fmiGetDerivatives(fmiComponent c, fmiReal derivatives[], size_t nx)
fmiStatus fmiGetStateValueReferences(fmiComponent c, fmiValueReference vrx[], size_t nx)

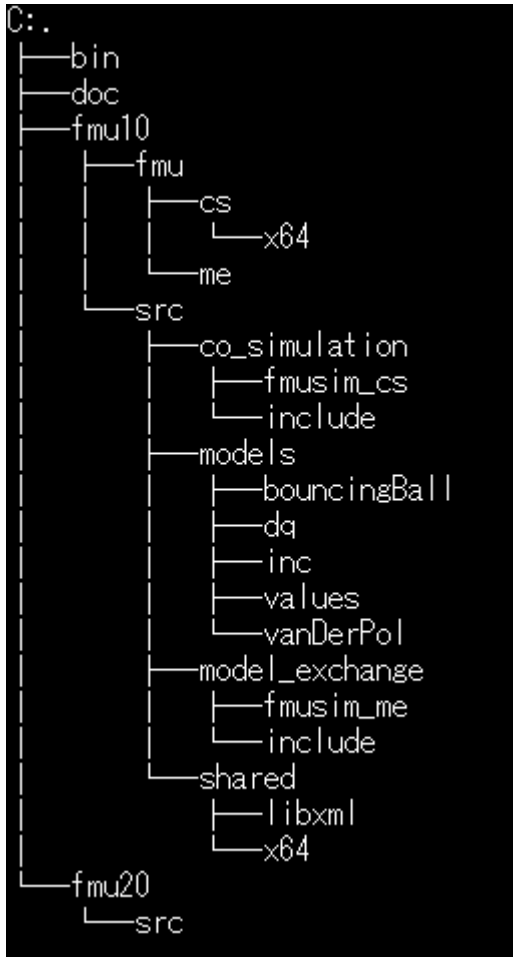
# FMI 1.0 for Model Exchange の関数

**FMI 1.0 for Model Exchange のみの関数(2)**

イベント処理
fmiStatus fmiCompletedIntegratorStep(fmiComponent c, fmiBoolean* callEventUpdate)
fmiStatus fmiGetEventIndicators(fmiComponent c, fmiReal eventIndicators[], size_t ni)
fmiStatus fmiEventUpdate(fmiComponent c, fmiBoolean intermediateResults, fmiEventInfo* eventInfo)

その他
fmiStatus fmiSetTime(fmiComponent c, fmiReal time)

# FMU SDK によるシミュレーション実行




- **シミュレータのビルド**
  ディレクトリ fmu10¥src で、 build_fmusim_me.bat を実行する。
  ディレクトリ bin に fmu10sim_me.exe が生成される。

- **シミュレーションの実行方法**
  fmusdk.zip を解凍したディレクトリで以下のように入力する

**fmusim  simulator model.fmu tEnd h loggingOn csvSeparator [-win64]**

- simulator:  cs10, cs20, me10, me20 のいずれか(シミュレータをビルドしたもの)
- model.fmu: FMUファイル
- tEnd: シミュレーション終了時間
- h: タイムステップ
- h: タイムステップ
- loggingOn: 1か０。ログファイルを出力スイッチ
- csvSeparator: c か s　CSV結果ファイルのセパレータ。カンマかセミコロン
- -win64: 64bit版を使用する場合のスイッチ

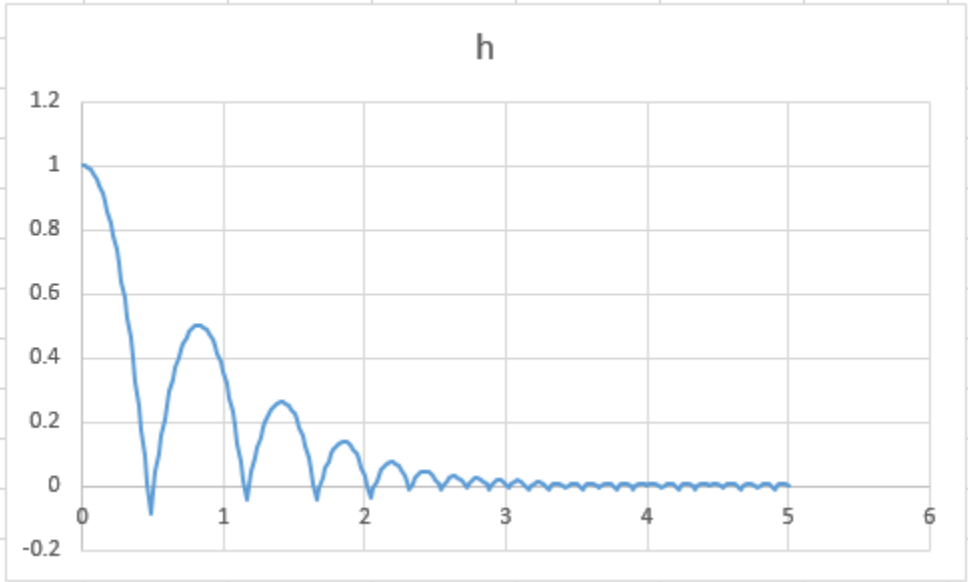
実際のコマンドラインには以下を入力した。

**fmusim me10 fmu10¥fmu¥me¥bouncingBall.fmu 5 0.02 0 c**

# FMU SDK によるシミュレーション実行

CSV結果ファイル result.csv をExcelで読み込んで作図した結果

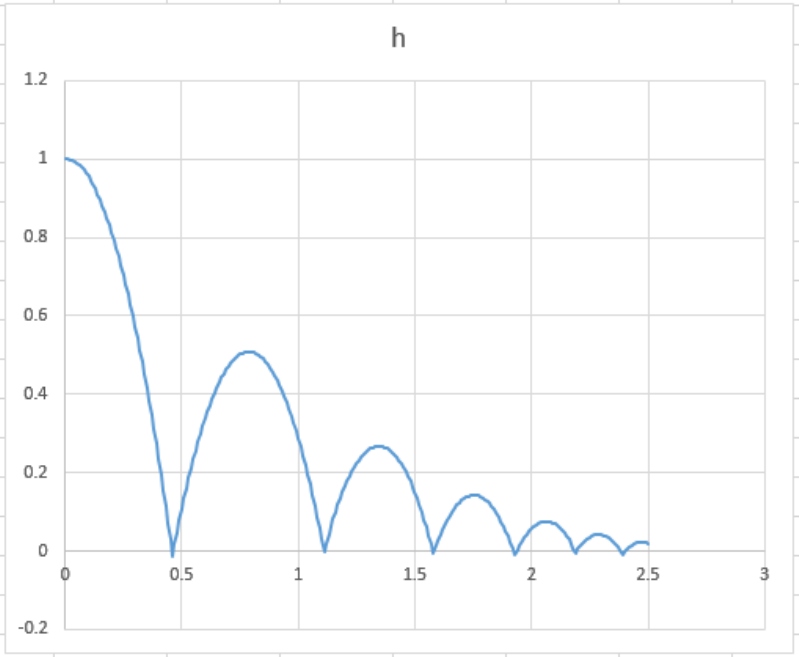
| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| 1 | time | h | der(h) | v | der(v) | g | e |
| 2 | 0 | 1 | 0 | 0 | -9.81 | 9.81 | 0.7 |
| 3 | 0.02 | 1 | -0.1962 | -0.1962 | -9.81 | 9.81 | 0.7 |
| 4 | 0.04 | 0.996076 | -0.3924 | -0.3924 | -9.81 | 9.81 | 0.7 |
| 5 | 0.06 | 0.988228 | -0.5886 | -0.5886 | -9.81 | 9.81 | 0.7 |
| 6 | 0.08 | 0.976456 | -0.7848 | -0.7848 | -9.81 | 9.81 | 0.7 |
| 7 | 0.1 | 0.96076 | -0.981 | -0.981 | -9.81 | 9.81 | 0.7 |
| 8 | 0.12 | 0.94114 | -1.1772 | -1.1772 | -9.81 | 9.81 | 0.7 |
| 9 | 0.14 | 0.917596 | -1.3734 | -1.3734 | -9.81 | 9.81 | 0.7 |
| 10 | 0.16 | 0.890128 | -1.5696 | -1.5696 | -9.81 | 9.81 | 0.7 |
| 11 | 0.18 | 0.858736 | -1.7658 | -1.7658 | -9.81 | 9.81 | 0.7 |
| 12 | 0.2 | 0.82342 | -1.962 | -1.962 | -9.81 | 9.81 | 0.7 |



# FMU SDK によるシミュレーション実行

tEnd=2.5, h(タイムステップ)=0.01とした場合の計算結果

| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| 1 | time | h | der(h) | v | der(v) | g | e |
| 2 | 0 | 1 | 0 | 0 | -9.81 | 9.81 | 0.7 |
| 3 | 0.01 | 1 | -0.0981 | -0.0981 | -9.81 | 9.81 | 0.7 |
| 4 | 0.02 | 0.999019 | -0.1962 | -0.1962 | -9.81 | 9.81 | 0.7 |
| 5 | 0.03 | 0.997057 | -0.2943 | -0.2943 | -9.81 | 9.81 | 0.7 |
| 6 | 0.04 | 0.994114 | -0.3924 | -0.3924 | -9.81 | 9.81 | 0.7 |
| 7 | 0.05 | 0.99019 | -0.4905 | -0.4905 | -9.81 | 9.81 | 0.7 |
| 8 | 0.06 | 0.985285 | -0.5886 | -0.5886 | -9.81 | 9.81 | 0.7 |
| 9 | 0.07 | 0.979399 | -0.6867 | -0.6867 | -9.81 | 9.81 | 0.7 |
| 10 | 0.08 | 0.972532 | -0.7848 | -0.7848 | -9.81 | 9.81 | 0.7 |
| 11 | 0.09 | 0.964684 | -0.8829 | -0.8829 | -9.81 | 9.81 | 0.7 |
| 12 | 0.1 | 0.955855 | -0.981 | -0.981 | -9.81 | 9.81 | 0.7 |
| 13 | 0.11 | 0.946045 | -1.0791 | -1.0791 | -9.81 | 9.81 | 0.7 |
| 14 | 0.12 | 0.935254 | -1.1772 | -1.1772 | -9.81 | 9.81 | 0.7 |
| 15 | 0.13 | 0.923482 | -1.2753 | -1.2753 | -9.81 | 9.81 | 0.7 |
| 16 | 0.14 | 0.910729 | -1.3734 | -1.3734 | -9.81 | 9.81 | 0.7 |
| 17 | 0.15 | 0.896995 | -1.4715 | -1.4715 | -9.81 | 9.81 | 0.7 |
| 18 | 0.16 | 0.88228 | -1.5696 | -1.5696 | -9.81 | 9.81 | 0.7 |

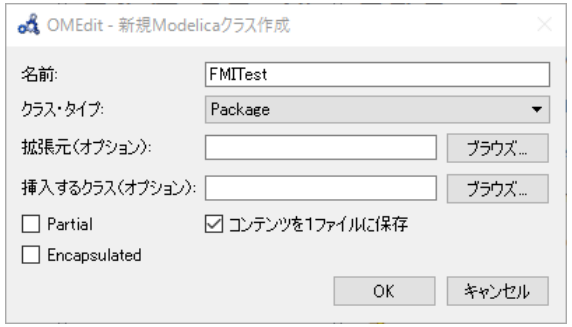


# OpenModelicaによるシミュレーション実行

OpenModelica 1.6.9 Windows版　https://openmodelica.org/download/download-windows

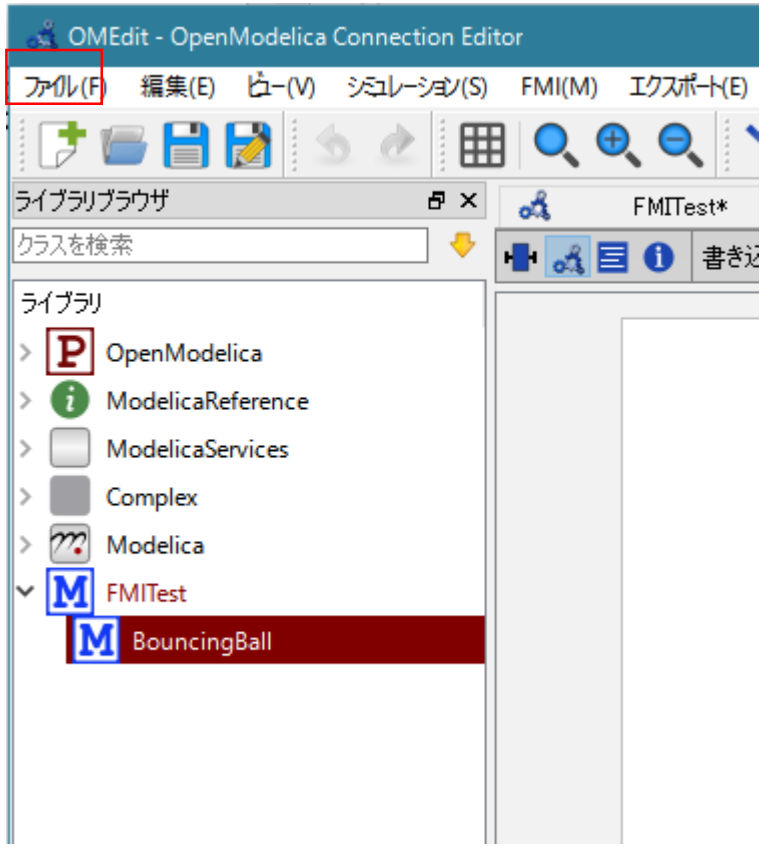
**OpenModelica Connection Editor (OMEdit)** を使用する。

①ファイル(F)＞Modelicaクラス新規作成

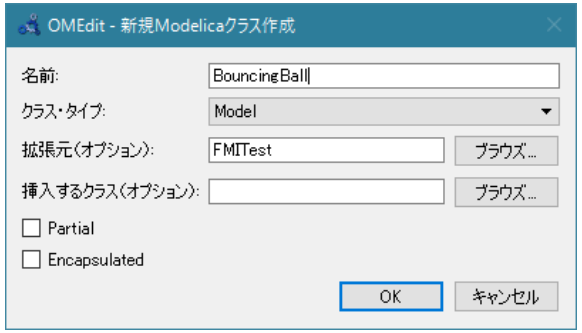



名前: FMITest

クラスタイプ: Package




②FMITestを右クリック＞Modelicaクラス新規作成

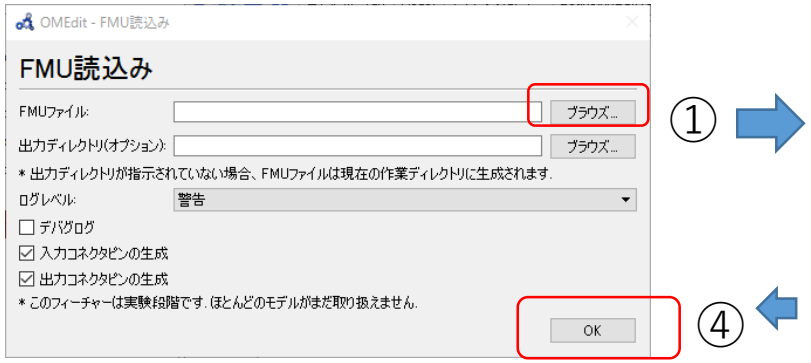



名前: BouncingBall

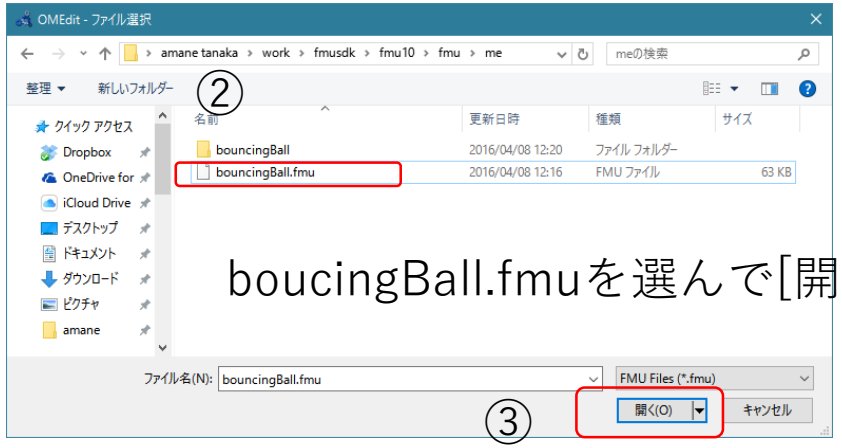
クラス・タイプ: Model

# OpenModelicaによるシミュレーション実行

③ FMI(M)＞FMI読込み

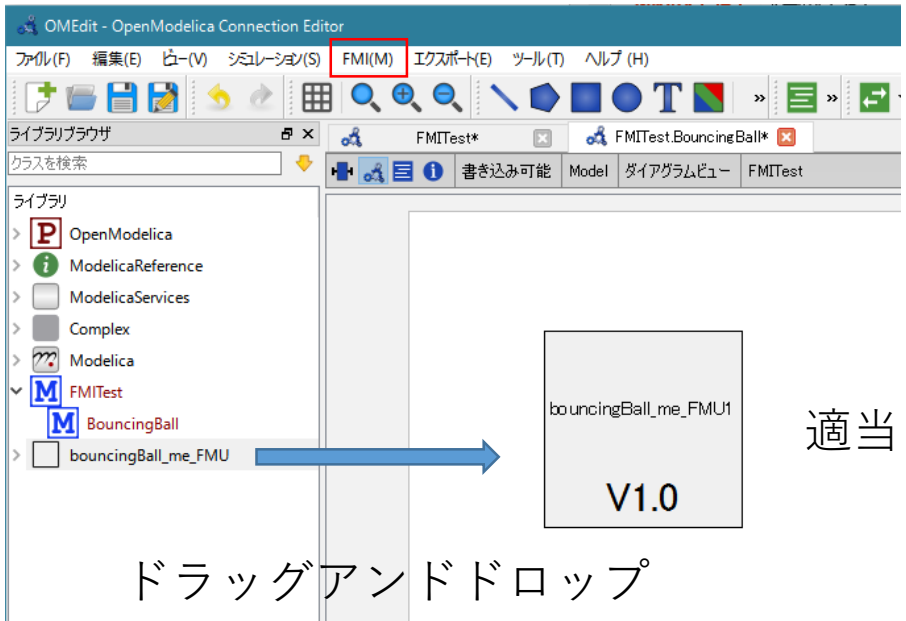



①




④

boucingBall.fmuを選んで[開く]



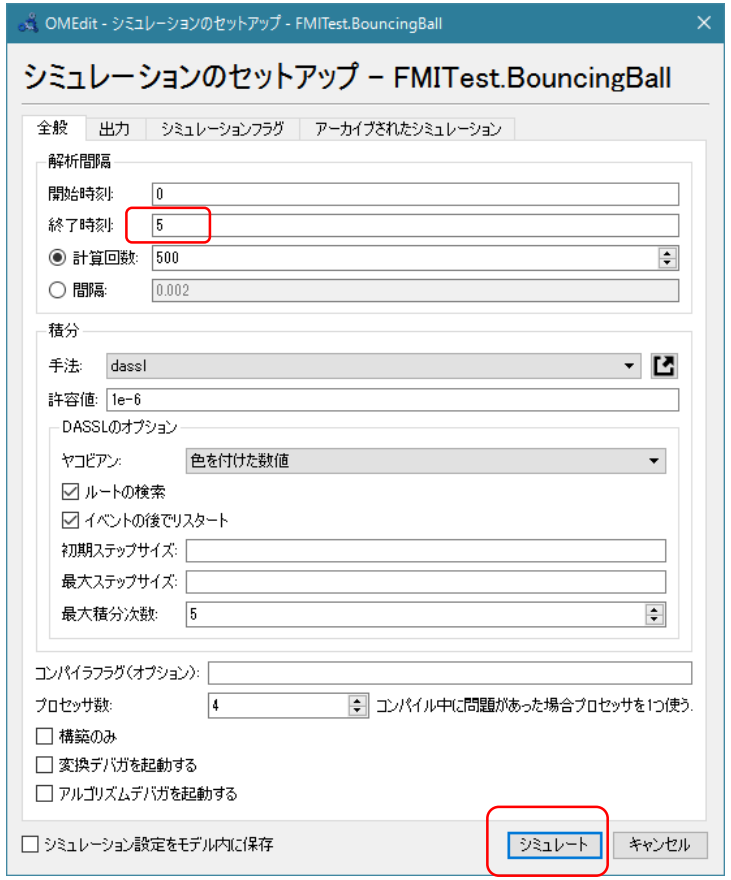

適当に拡大する

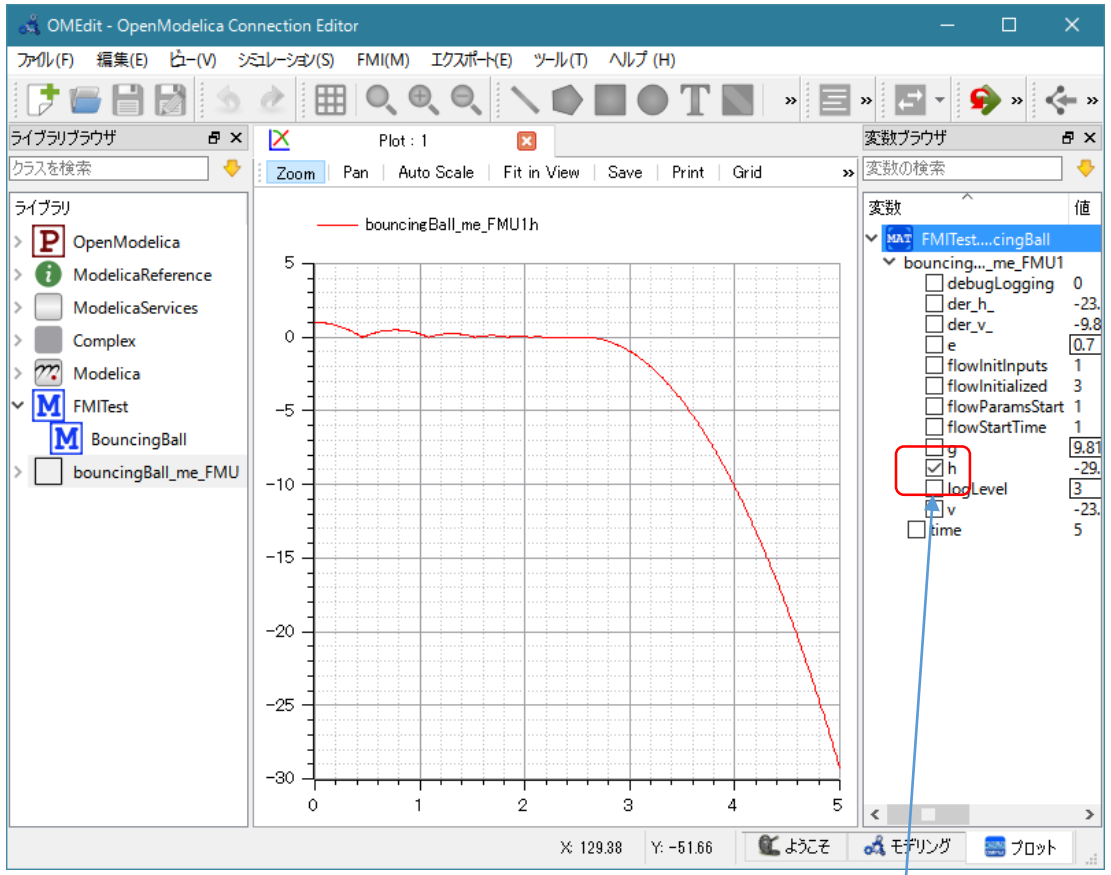
ドラッグアンドドロップ

# OpenModelicaによるシミュレーション実行

④シミュレーション＞シミュレーションのセットアップ

2.6秒ぐらいからの結果が変！！



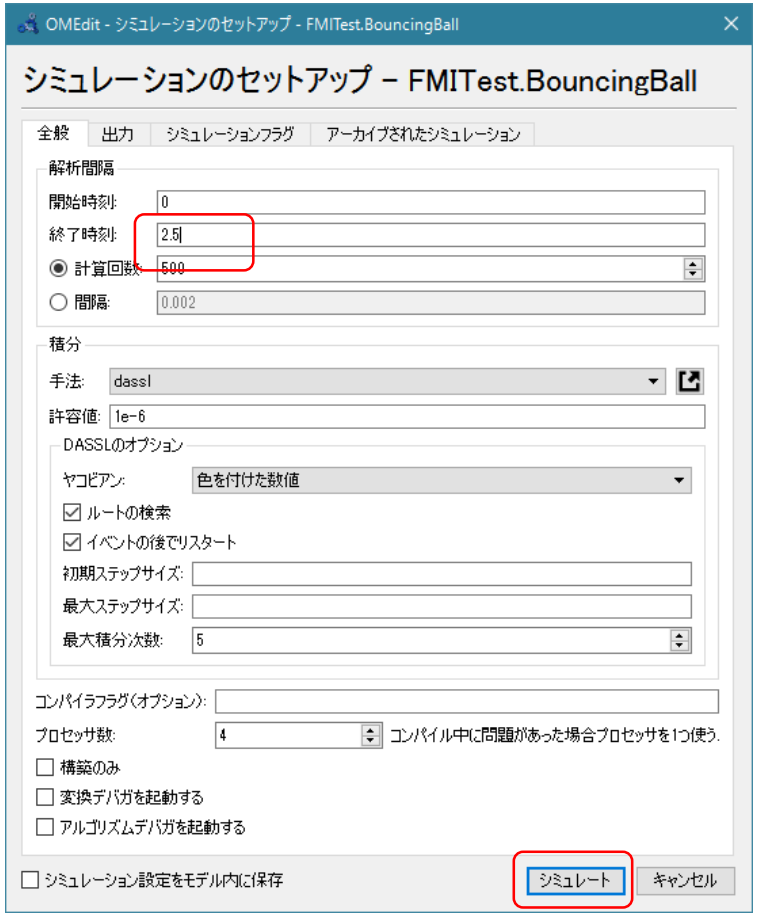



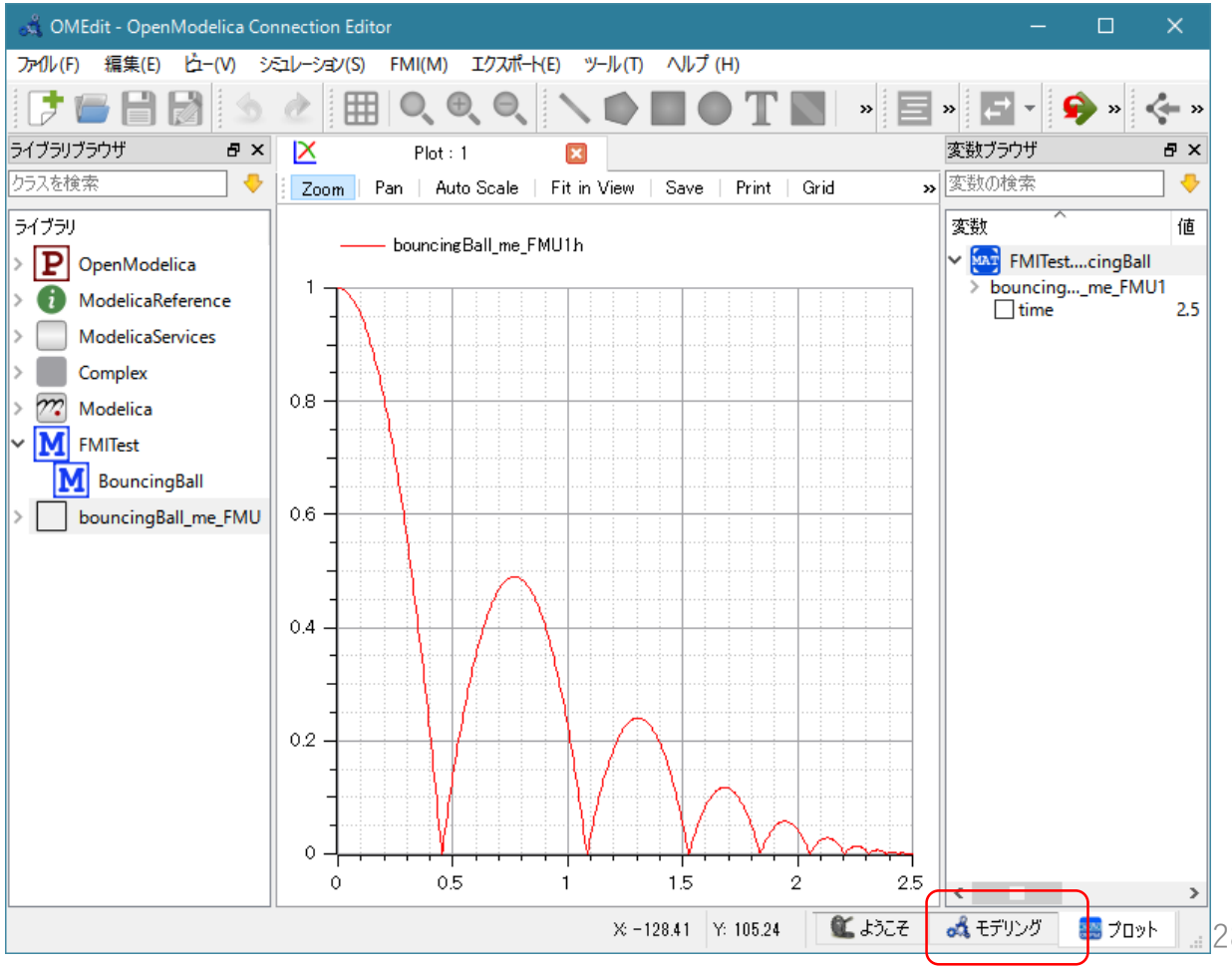

終了時刻を５秒にして[シミュレート]をクリック

変数ｈをチェックしてプロットする。

# OpenModelicaによるシミュレーション実行

[モデリング]タブに戻してから再びシミュレーションのセットアップで終了時刻を2.5秒にしてシミュレート。



# 入力変数と出力変数の設定

**FMU SDK**

bouncingBall.fmu

**無次元パラメータ $e$（反発係数)を入力変数、ボールの高さ $h$ を出力変数に変更してみる。**

これにより他のモデルと変数のやりとりができる。

① fmu10¥src¥models¥bouncingBall¥modelDescription.xml を編集する。

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<fmiModelDescription
 fmiVersion="1.0"
 modelName="bouncingBall"
 modelIdentifier="bouncingBall"
 guid="{8c4e810f-3df3-4a00-8276-176fa3c9f003}"
 numberOfContinuousStates="2"
 numberOfEventIndicators="1">
<ModelVariables>
  <ScalarVariable name="h" valueReference="0" description="height, used as state" causality="output">
    <Real start="1" fixed="true"/>
  </ScalarVariable>
  <ScalarVariable name="der(h)" valueReference="1" description="velocity of ball">
    <Real/>
  </ScalarVariable>
  <ScalarVariable name="v" valueReference="2" description="velocity of ball, used as state">
    <Real/>
  </ScalarVariable>
  <ScalarVariable name="der(v)" valueReference="3" description="acceleration of ball">
    <Real/>
  </ScalarVariable>
  <ScalarVariable name="g" valueReference="4" description="acceleration of gravity"
                  variability="parameter">
    <Real start="9.81" fixed="true"/>
  </ScalarVariable>
  <ScalarVariable name="e" valueReference="5" description="dimensionless parameter" causality="input">
    <Real start="0.7" fixed="true"/>
  </ScalarVariable>
</ModelVariables>
```

```
<ScalarVariable name="h" valueReference="0" description="height, used as state" causality="output">
  <Real start="1" fixed="true"/>
</ScalarVariable>
```

```
<ScalarVariable name="e" valueReference="5" description="dimensionless parameter" causality="input">
  <Real start="0.7" fixed="true"/>
</ScalarVariable>
```

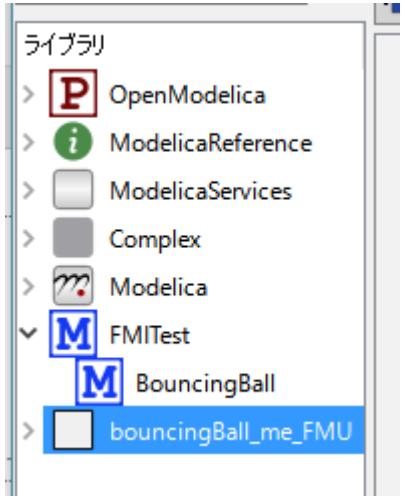
② ディレクトリ fmu10¥src¥models で

**build_fmu me bouncingBall**

を実行して bouncingBall.fmu を再構築する。

# 入力変数と出力変数の設定

**OpenModelica Connection Editor**

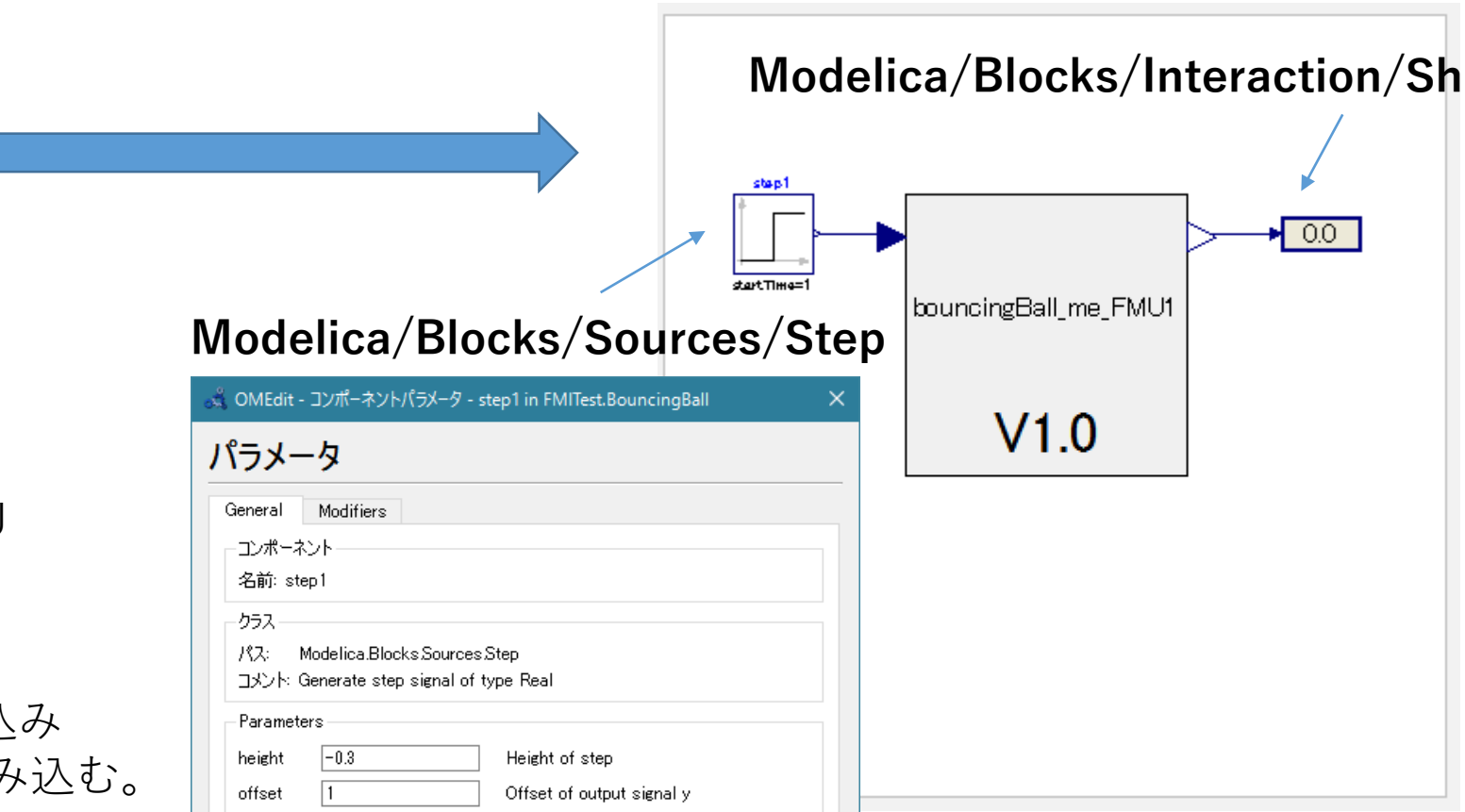



bouncingBall_me_FMU
を右クリックして
アンロードを選択する。

再び、FMI(M)＞FMIの読込み
で bouncingBall.fmu を読み込む。

**入出力コネクタに**
**Modelica標準ライブラリのモデルを配置して接続する。**

**Modelica/Blocks/Interaction/Show/RealValue**



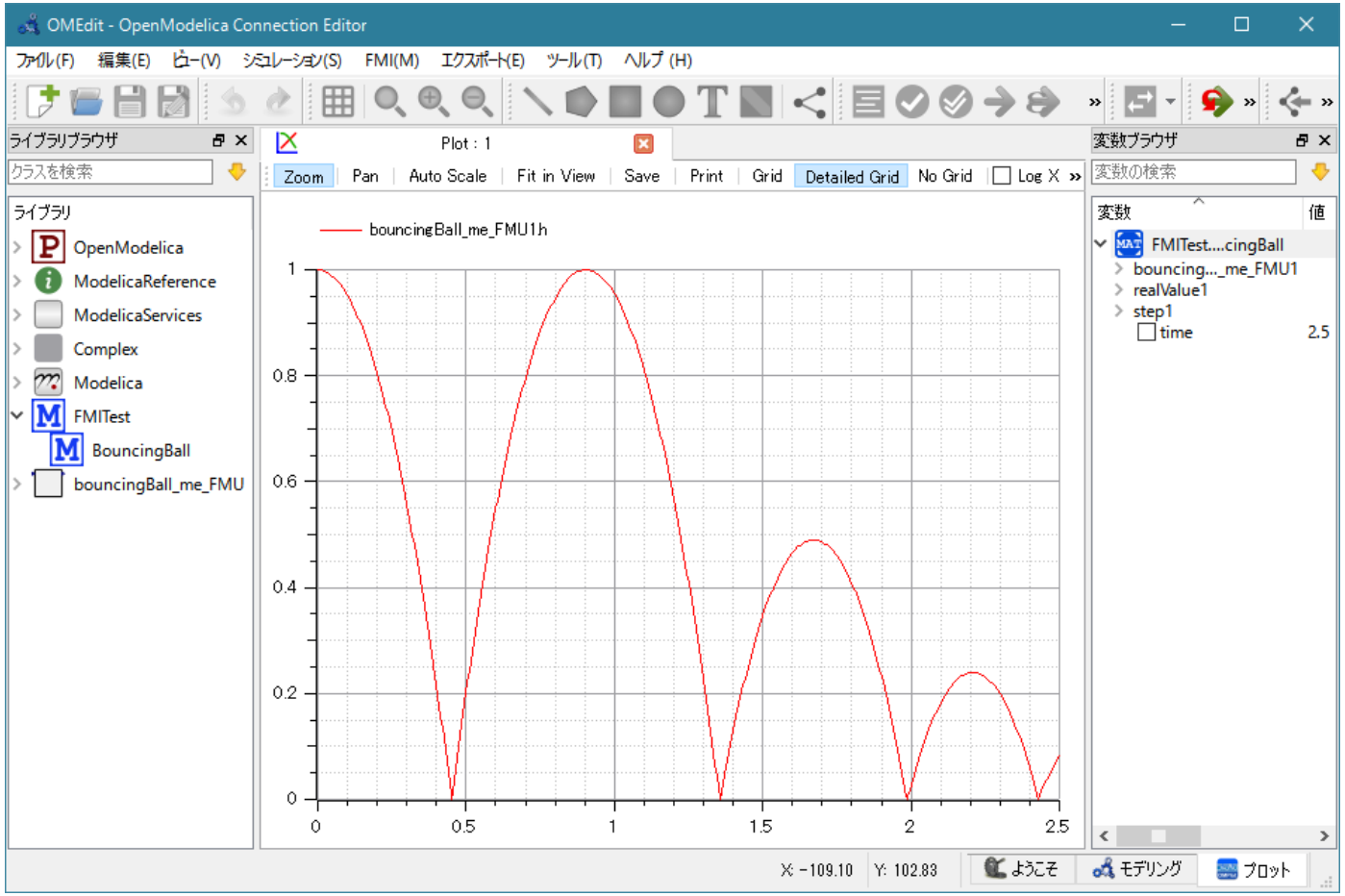

**Modelica/Blocks/Sources/Step**




1秒までe=1、その後 e=0.7

# 入力変数と出力変数

再びシミュレーションを実行する。




１秒までは
反発係数　$e = 1$
一秒以後は
反発係数　$e = 0.7$

# Scilab/Xcos によるシミュレーション実行

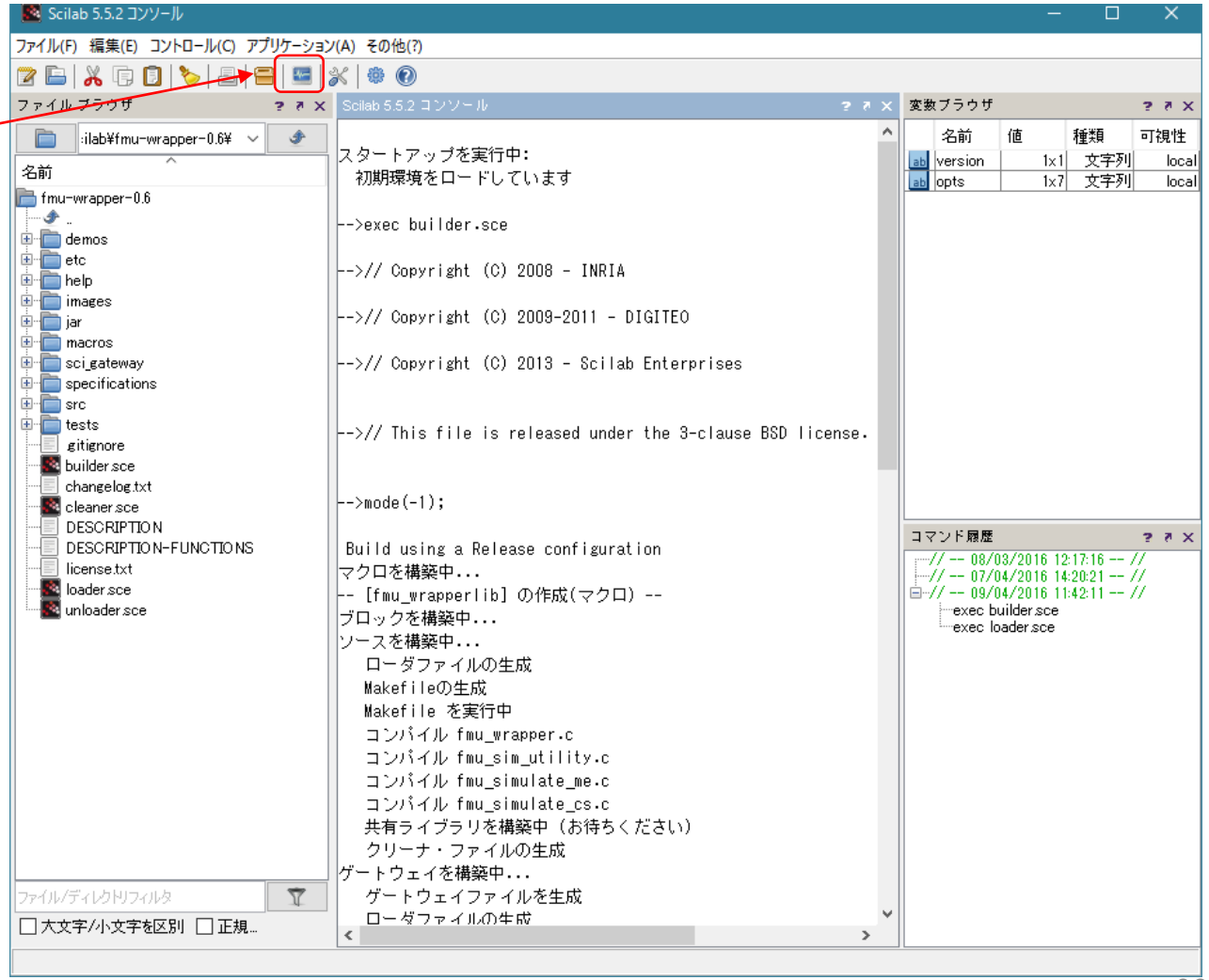
**SCILAB 5.5.2**(Windows 64bit版)を使用

http://www.scilab.org/

**Xcos FMU wrapper**

https://forge.scilab.org/index.php/p/fmu-wrapper/
public domain
fmu-wrapper-0.6.zip

[Xcos]ボタン




① Scilab を起動し、ファイルブラウザで fmu-wrappwr-0.6を解凍したフォルダに移動して、以下を実行する。

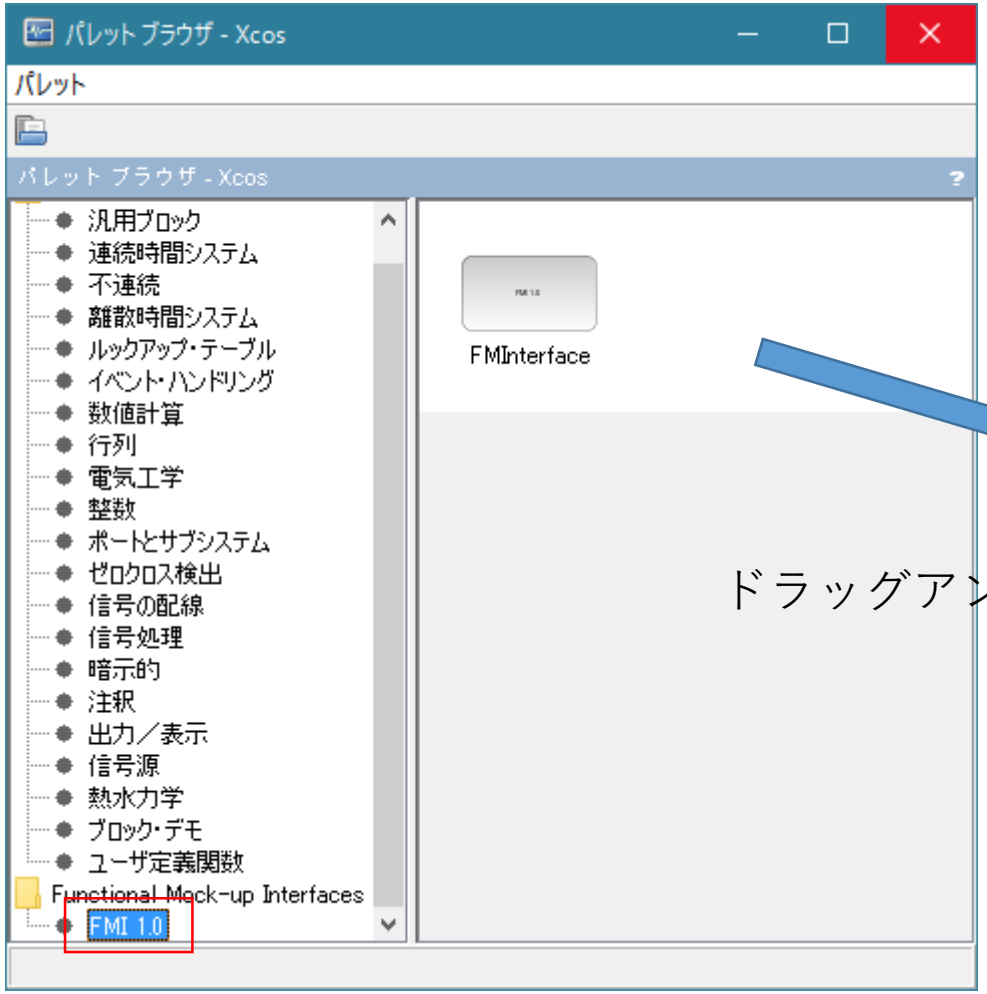
**exec builder.sce**
**exec loader.sce**

② [Xcos] ボタンで Xcosを起動する。

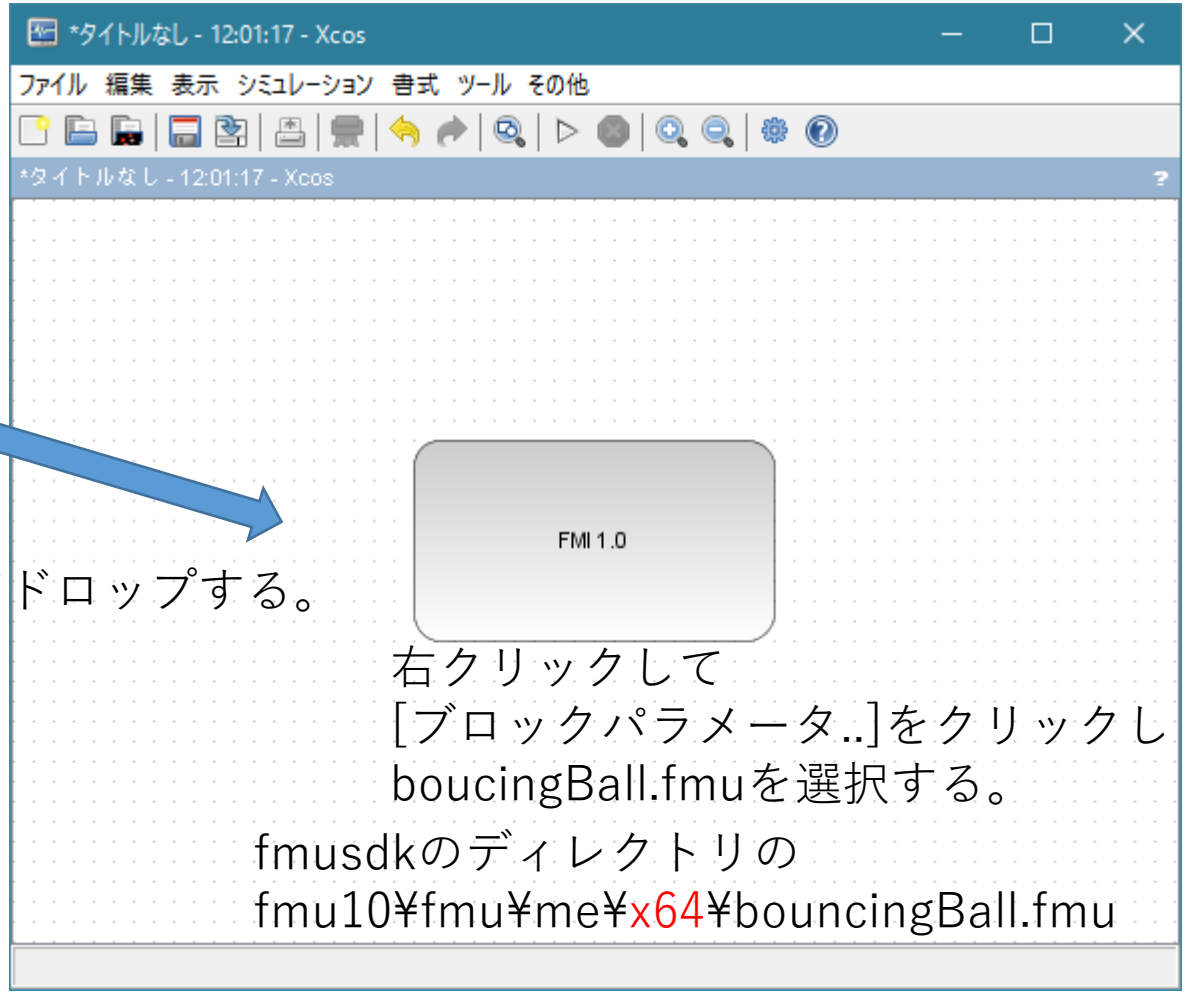
**注意** ディレクトリ fmu10¥src¥models で
build_fmu me bouncingBall –win64
と入力し64bit版のDLLを含むFMUを用意する。

# Scilab/Xcos によるシミュレーション実行

## Xcos によるシミュレーション



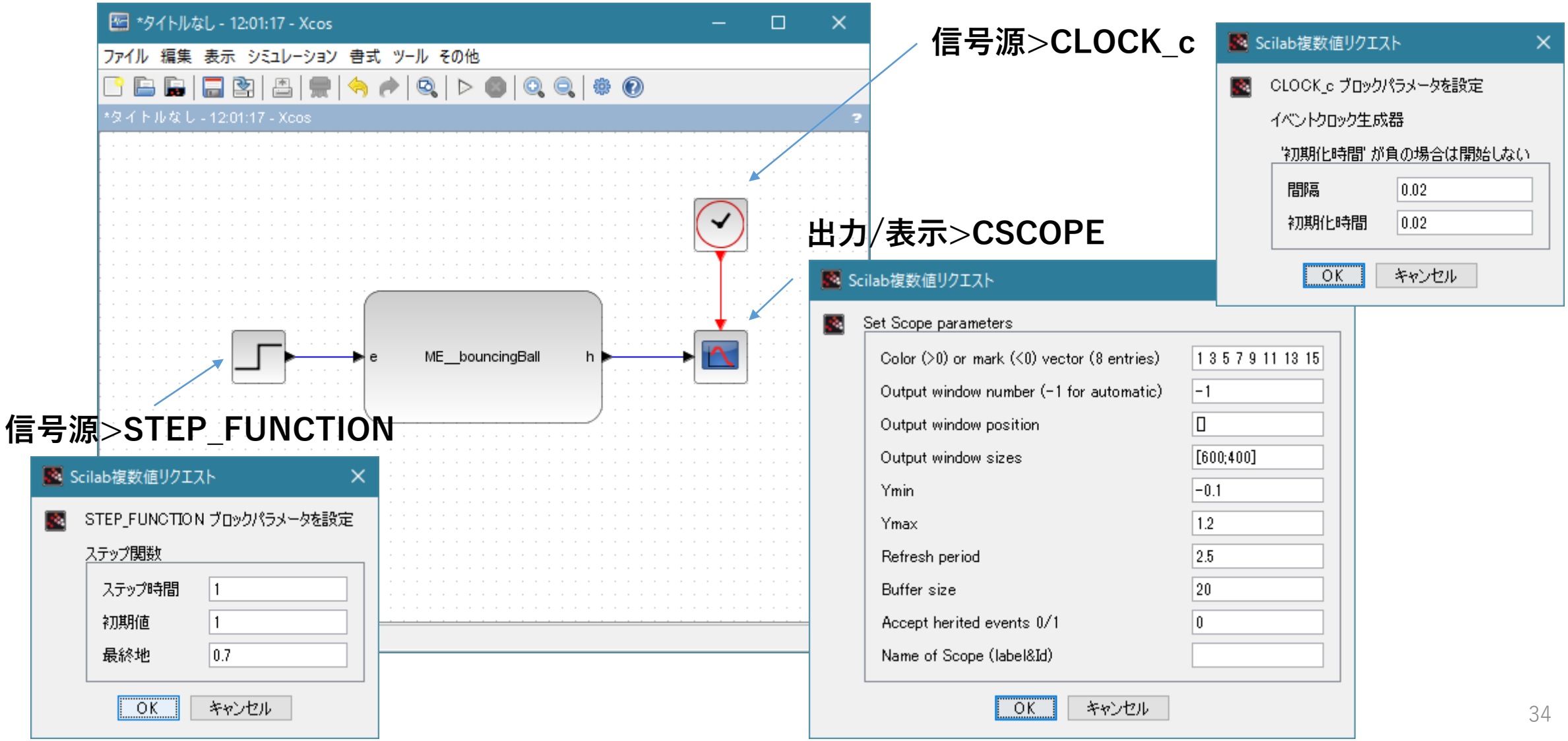

ドラッグアンドドロップする。



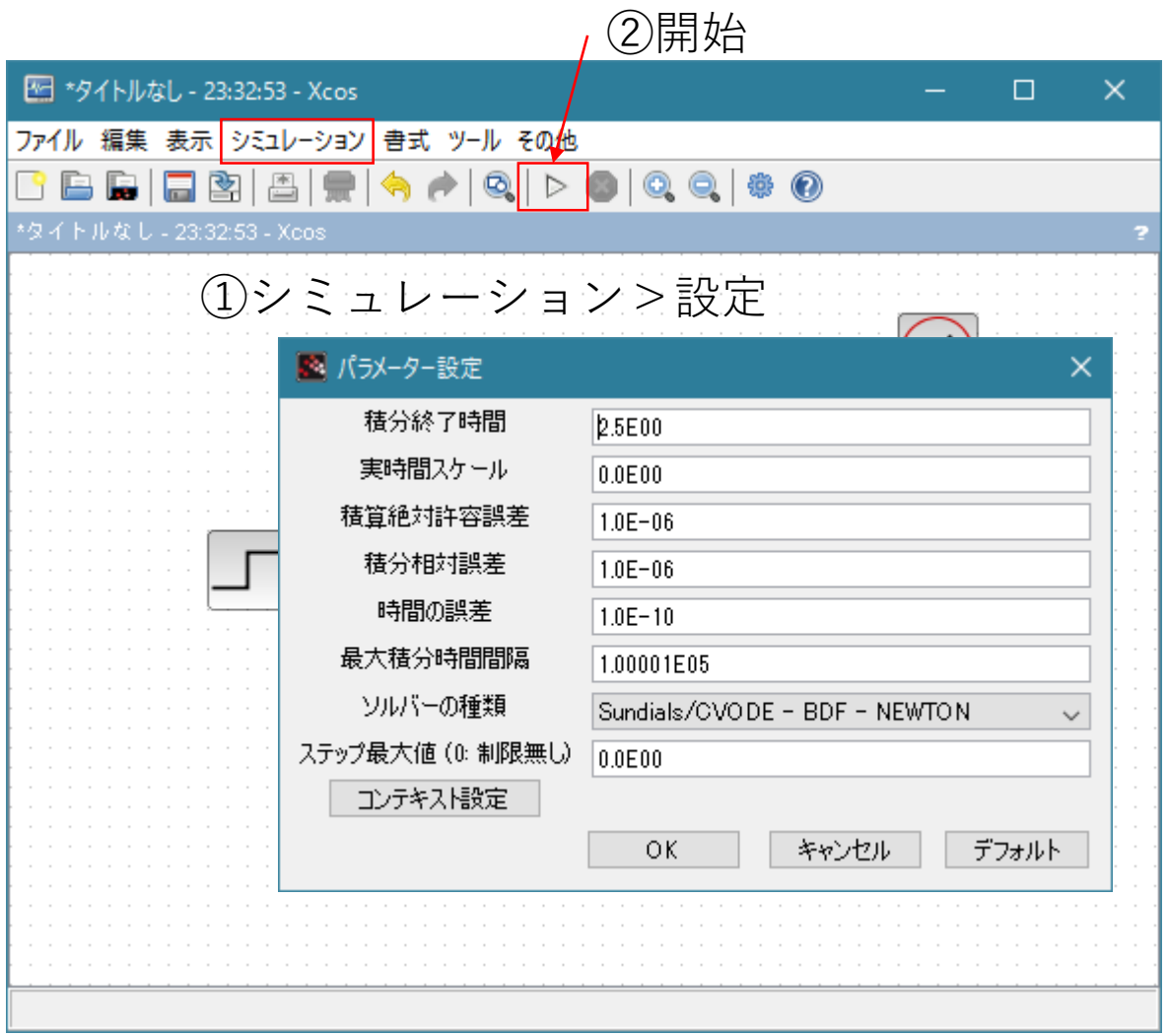

右クリックして
[ブロックパラメータ..]をクリックし
boucingBall.fmuを選択する。

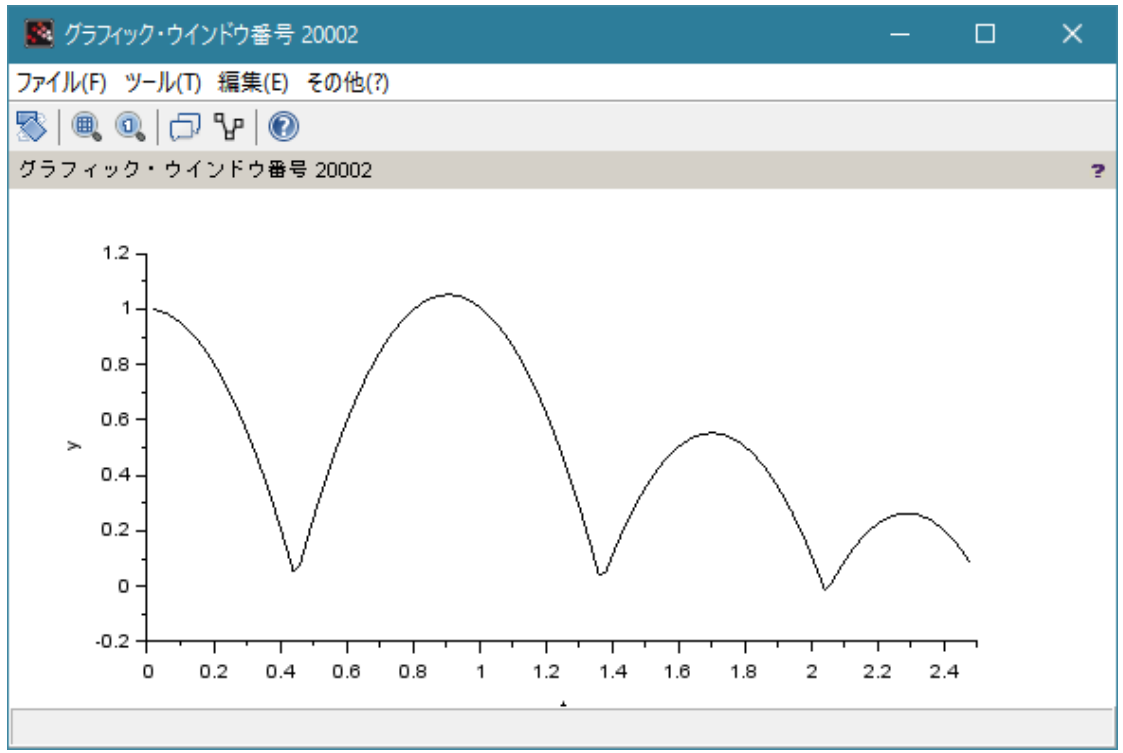
fmusdkのディレクトリの
fmu10¥fmu¥me¥x64¥bouncingBall.fmu

# Scilab/Xcos によるシミュレーション実行



# Scilab/Xcos によるシミュレーション実行



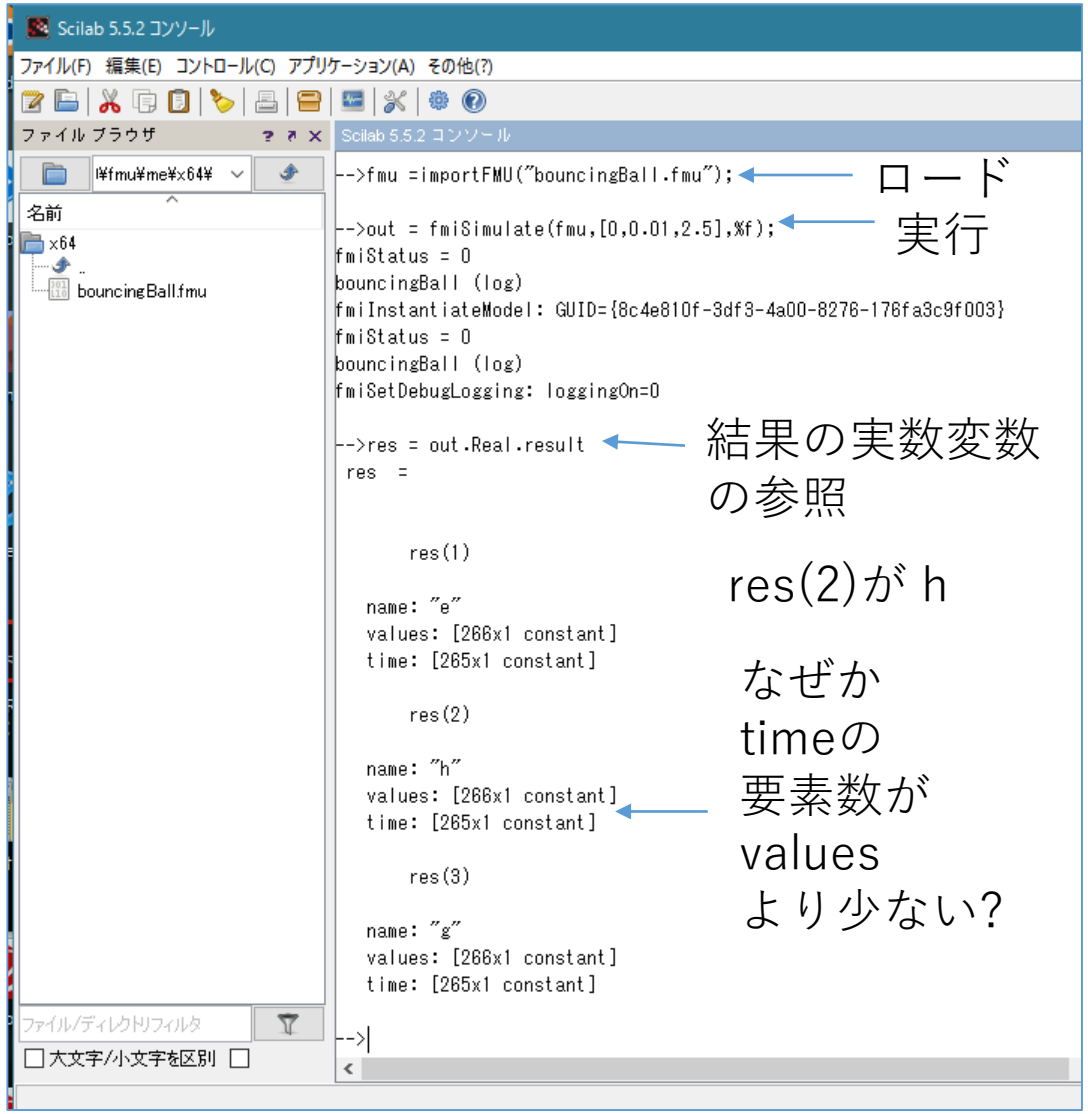
# Scilab/Xcos によるシミュレーション実行

**Silabによる入力変数を使用しない**
**FMU単体の実行**

① Silabを起動し、ファイルブラウザを fmu-wrapper-0.6 を解凍したフォルダに移動して
exec loader.sce
を実行する。

② フォルダをfmi10¥fmu10¥fmu¥me¥x64に移動して以下を実行する。

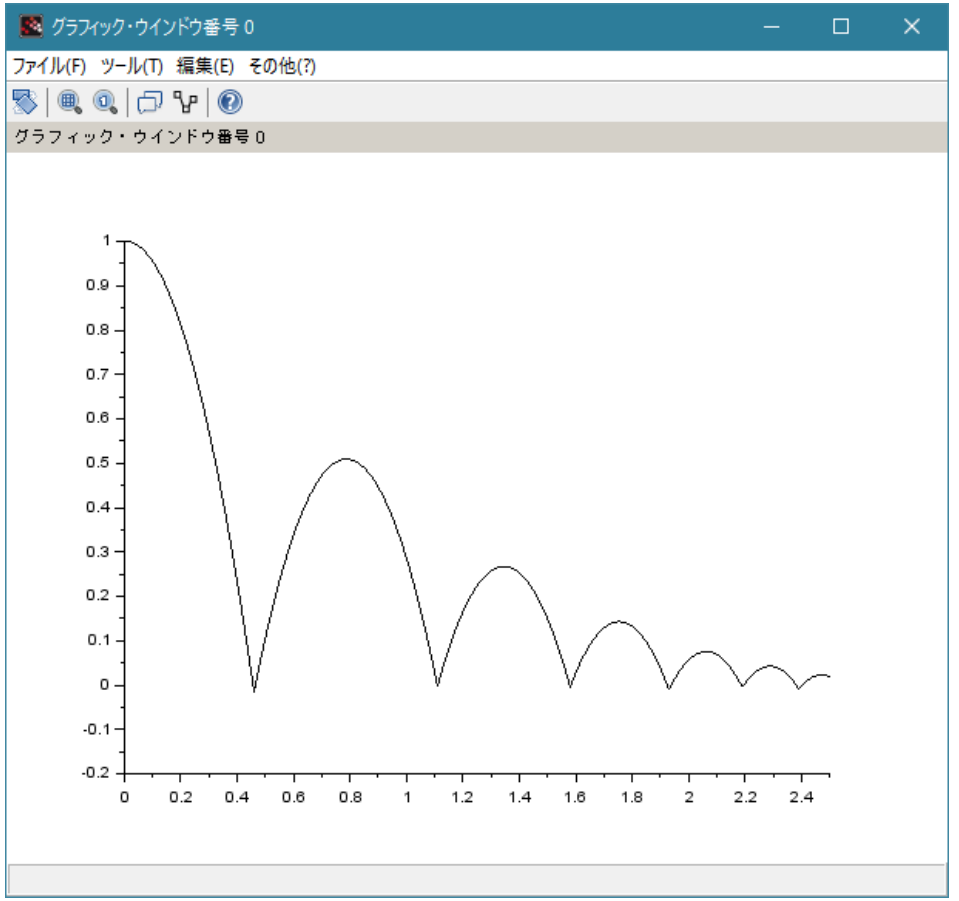
```
fmu = importFMU("bouncingBall.fmu");
out = fmiSimulate(fmu,[0,0.01,2.5],%f);
res = out.Real.result
```



# Scilab/Xcos によるシミュレーション実行

timeのデータが１個少ないので
1個加えて、プロットする。

```
-->t = res(2).time;

-->h = res(2).values;

-->t(265)
 ans  =

    2.5

-->t(266)=2.5;

-->plot2d(t,h)

-->
```



# PyFMIによるシミュレーション実行

**PyFMI**は Python でFMUのロードや操作などが行えるパッケージで Modelon によって開発、配布されている。

http://www.jmodelica.org/assimulo_home/pyfmi_1.0/index.html#

**PyFMIのインストール方法**

以下のようなインストール方法がある。ここでは1の方法でインストールした Windows 32bit版を使用した。

1．AnacondaやMinicondaを使ってPythonをインストールした環境で以下を実行する。

**conda install -c https://conda.binstar.org/chria pyfmi**

2. 依存するライブラリやパッケージを手でインストールして setup.pyを使ってインストールする。

http://www.jmodelica.org/assimulo_home/pyfmi_1.0/installation.html#

3．JModelica.org をインストールする。

PyFMIは JModelia.org の１部分となっているので、JModelca.org をインストールすることでもインストールすることができる。

http://www.jmodelica.org/

オペレーションに**IPython**, グラフの作成に**Matplotlib** も使用した。

cmd - ipython

```
In [1]: import matplotlib.pyplot as plt

In [2]: from pyfmi import load_fmu

In [3]: model = load_fmu('bouncingBall.fmu')

In [4]: opts = model.simulate_options()

In [5]: opts["ncp"]=250

In [6]: res = model.simulate(final_time=2.5, options=opts)
Final Run Statistics: bouncingBall

 Number of steps                                 : 158
 Number of function evaluations                  : 278
 Number of Jacobian evaluations                  : 23
 Number of function eval. due to Jacobian eval.  : 46
 Number of error test failures                   : 6
 Number of nonlinear iterations                  : 186
 Number of nonlinear convergence failures        : 0
 Number of state function evaluations            : 370
 Number of state events                          : 22

Solver options:

 Solver                     : CVode
 Linear multistep method    : BDF
 Nonlinear solver           : Newton
 Linear solver type         : DENSE
 Maximal order              : 5
 Tolerances (absolute)      : 1e-06
 Tolerances (relative)      : 0.0001

Simulation interval    : 0.0 - 2.5 seconds.
Elapsed simulation time: 0.0241726871286 seconds.

In [7]:
```

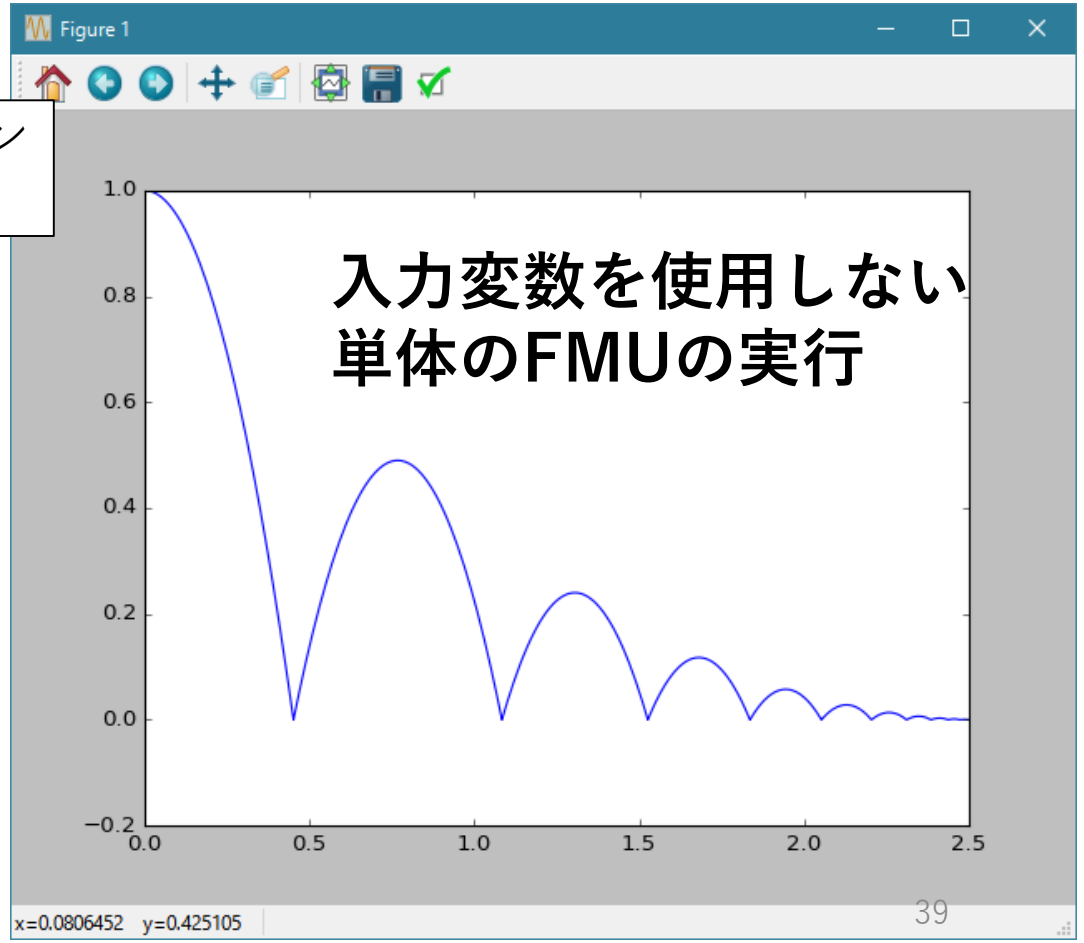
FMUロード

オプション
出力250個

シミュレーション
実行

```
In [7]: h=res['h']

In [8]: t=res['time']

In [9]: plt.plot(t,h)
Out[9]: [<matplotlib.lines.Line2D at 0xde656f0>]

In [10]: plt.show()
```

プロット処理




**入力変数を使用しない**
**単体のFMUの実行**

# PyFMIによるシミュレーション実行

## 入力変数がある場合の実行方法

左のスクリプトを作成し、
コマンドラインから次のように実行する。

**python bouncingBall.py**

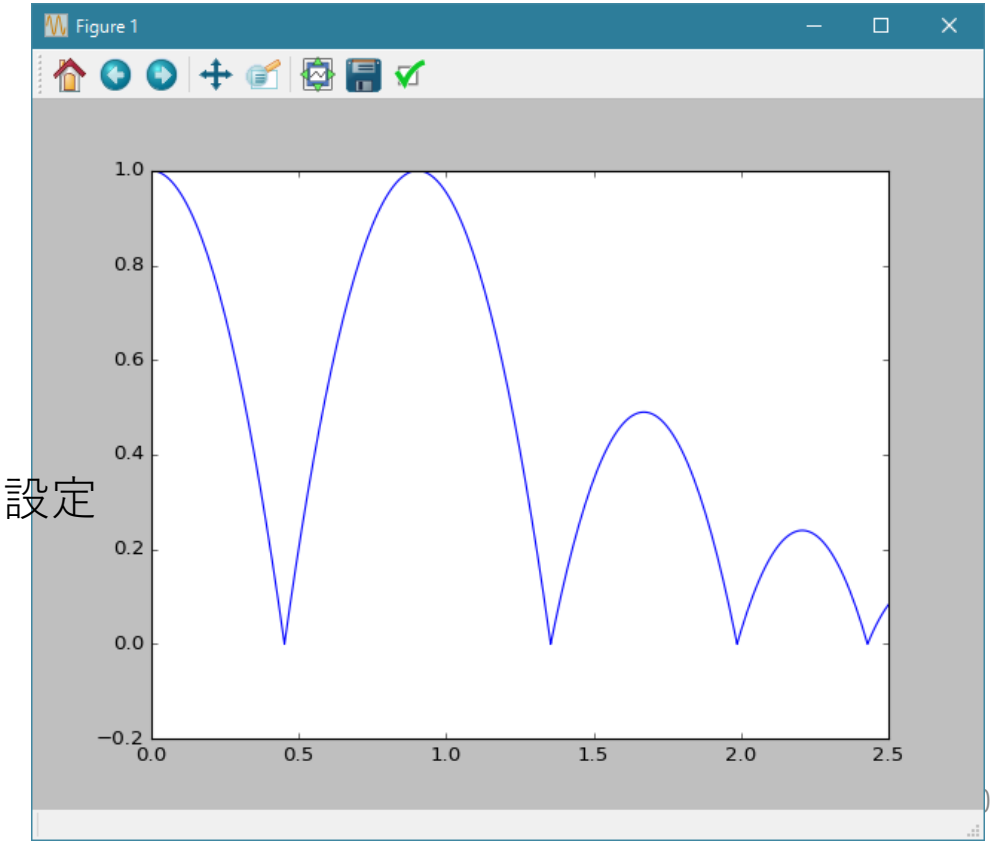
bouncaingBall.py

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Sun Apr 10 22:38:08 2016
4
5  @author: finback
6  """
7
8  import pylab as P
9  import numpy as N
10 from pyfmi import load_fmu
11
12 t = N.linspace(0.,2.5,250)
13 e = N.array([])
14 for tt in t:
15     if tt < 1.0:
16         e = N.append(e,[1.0])
17     else:
18         e = N.append(e,[0.7])
19 e_traj = N.transpose(N.vstack((t,e)))
20 input_object=('e',e_traj)
21
22 model = load_fmu('bouncingBall.fmu')
23 model.set('e',e[0])
24 res = model.simulate(final_time=2.5, input=input_object, options={'ncp':250})
25
26 t = res['time']
27 h = res['h']
28 P.plot(t,h)
29 P.show()
```

入力変数のオブジェクトの作成

FMUロード
入力変数のオブジェクト設定
シミュレーション実行

プロット処理



# まとめ

- FMI 1.0 for Model Exchange のモデルの概念について仕様書とFMU SDKを用いて調査した。
- サンプルモデルを使用してシミュレーションツールとしてOpenModelica, Scilab/Xcos FMU wrapper, PyFMI を用いたシミュレーションの実行方法を調べた。