MFiX Tutorial
Rotating drum with varying rotational speed

Sathish Sanjeevi and Jeff Dietiker

National Energy Technology Laboratory, USA

November, 2019

1 Introduction

In this tutorial, a rotating drum is simulated with varying rotational speed using user-defined functions (UDFs). The objectives of this tutorial are:

- Show how to apply a tangential velocity on a STL object to represent moving walls. It should be noted that the STL faces do not move in our simulations but rather the movement is mimicked by adding tangential velocity on the stationary STL faces.
- Use keyframe data to specify transient rotational velocity for the drum.

2 Keyframes

This section provides a general description of keyframe data usage. Keyframes are tables with numerical data to be used in simulations, and are currently implemented through UDFs. Such tables are useful to perform dynamic simulations, for example, a time dependent inflow boundary. Each keyframe is titled in the form `data_kf_<ID>.txt`. Here, ID is a four-padded integer representing the keyframe ID, which will typically be associated with the same boundary condition ID (BCIDs), although ID and BCID are independent from each other (it is the user's responsibility to decide how to use the keyframe data). The user-defined keyword `read_kf(ID)` allows reading a given keyframe file. Each keyframe file has the following structure:

```
nrows, nvar
interpolation_type
!  header
 x_1   y_{11} ...   y_{1n}
 .     .            .
 .     .            .
 x_m   y_{m1} ...   y_{mn}
```

Here, `nrows(=m)` is the number of data rows in the keyframe file, and `nvar(=n)` is the number of

dependent variables. The first column defines a list of increasing values for the independent variable `x`. In this tutorial, we are using keyframes to define transient data, so `x` correspond to time. Other columns define values of the dependent variables `y`, for example velocity at corresponding time. The data in all columns are floating point (real) scalars. For vector quantities, users need to specify the 3 components of the vector using 3 columns (for example, if two scalars and one vector are defined, then `n=5`). `interpolation_type` defines the available interpolation types, namely `linear` or `step`. Once the keyframe data is defined, users can evaluate `y` at any `x`-value between the data points. Two interpolation schemes are available: 1) Linear interpolation, where `y` is assumed to vary linearly between two consecutive points (interval $[x_i; x_{i+1}]$) , and 2) Step function which returns the constant y-value y_i when the `x` value lies within the interval $[x_i; x_{i+1}]$. The actual keyframe reading (`read_keyframe_data`) and interpolation (`interpolate_keyframe_data`) functions are located in `usr_mod.f` file.

2.1 A keyframe example

The following example explains the actual value of a variable `var1` depending on the `interpolation_type`. Consider the following keyframe file:

```
6, 1
<interpolation_type>
!x                      var1
0.0                     0.5
2.0                     0.0
4.0                     2.5
6.0                     6.0
8.0                     4.0
10.0                    3.0
```

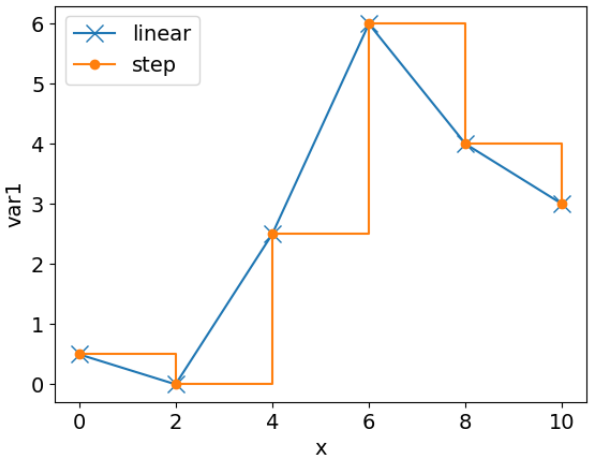


Figure 1: Profiles of `var1` when `interpolation_type = linear` or `step`.

The actual profile of `var1` is illustrated in figure 1 depending on `interpolation_type = linear` or `step`. Symbols represent the keyframe data, lines represent the interpolated data. Values of `var1` are kept constant outside the `x`-range. For example, consider the following keyframe file:

```
3, 1
<interpolation_type>
!x                      var1
4.0                     2.5
6.0                     6.0
8.0                     4.0
```

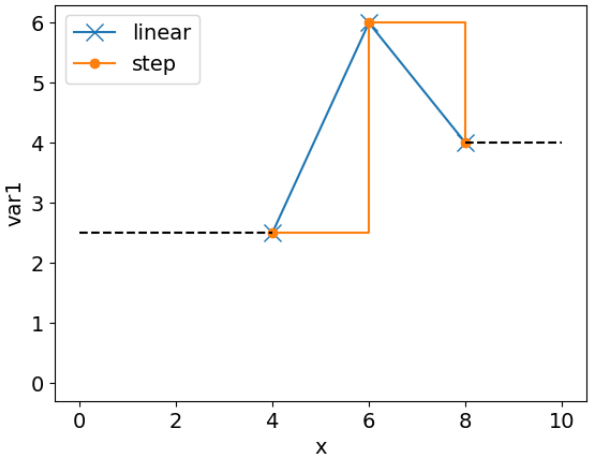


Figure 2: Value of `var1` outside the keyframe `x`-range is indicated with dashed line. Behaviour outside the specified keyframe `x`-range is independent of `interpolation_type`.

The full profile of `var1` in actual simulations is shown in figure 2. As seen, the first and last values of the keyframe values are extended outside the `x`-range (dashed line), both for `linear` and `step` interpolation schemes.

3 Problem setup

This tutorial performs a Discrete Element Method (DEM) simulation of particles in a rotating drum enclosed with side walls. This is a pure granular flow (the gas phase is ignored). In this testcase, the drum's rotational velocity is ramped from zero to 20 rad/s and subsequently to -20 rad/s about z-axis over time. The rotating drum initialized with particles is shown in figure 3.

The rotating drum has BCID=1 and accordingly reads `data_kf_0001.txt` which contains the time and the corresponding rotational velocity shown in table 1. For each simulation timestep, the velocities of the conveyor belts are linearly interpolated between keyframe values. The corresponding interpolation of the keyframe data is shown in figure 4. We briefly explain the functions of different files available for this tutorial:

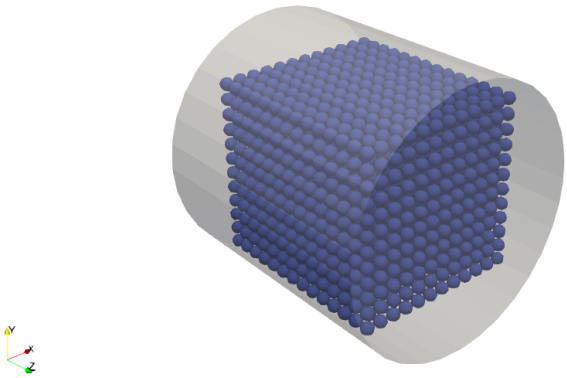


Figure 3: The rotating drum initialized with particles.

4, 1	
linear	
!time	var1
0.0	0.0
0.5	20.0
1.0	0.0
1.5	-20.0

Table 1: Contents of `data_kf_0001.txt`.

- `usr_mod.f` - contains all the necessary functions for reading and interpolating keyframe data. Also contains the functions for returning appropriate interpolated data.
- `usr0.f` - calls the function to read keyframe files and allocates required variables.
- `usr1_des.f` - utilizes the interpolated data (rotational velocity magnitude) and converts it to rotational velocity vector.
- `calc_collision_wall_mod.f` - assigns the tangential relative velocity for the wall (drum) boundaries. It should be noted that the drum (made with STL faces) does not move in our simulation. Rather, the drum rotation is mimicked by adding relative tangential velocity on the still drum.
- `des_time_march.f` - the call for `usr1_des` is moved earlier (compare with original file: `mfix/model/des/des_time_march.f`). This enables proper use of interpolated data for a given time.
- `usr_init_namelist.f` and `usrnlst.inc` - Both files let the solver and GUI known about the new keyword `read_kf` which is an addition in this tutorial. This is a Boolean flag used in the UDFs to read a given keyframe file.

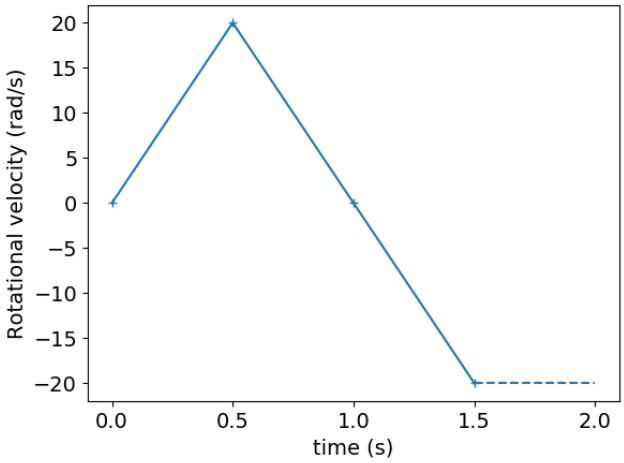
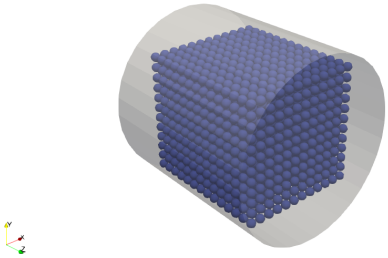
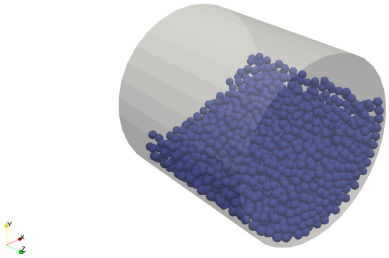


Figure 4: Rotational velocity of the drum. Markers indicate keyframe data (`data_kf_0001.txt`), solid lines indicate the linear interpolation, and dashed line indicate the extrapolation of last keyframe data within the simulation.

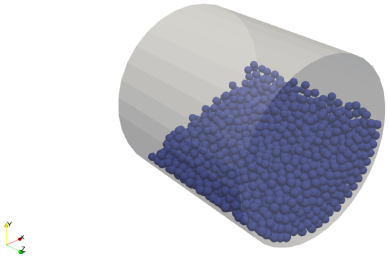
4 Results



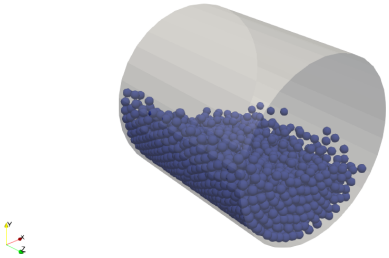
(a) $t = 0$s



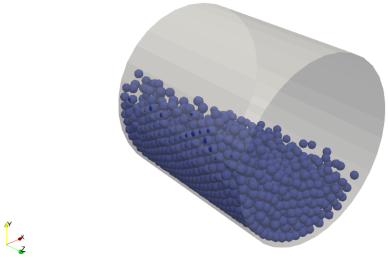
(b) $t = 0.5$s



(c) $t = 1$s



(d) $t = 1.5$s



(e) $t = 2$s

Figure 5: Snapshot of the rotating drum at different time intervals. In this case, the drum starts from from rest, rotates in one direction and then changes direction after a specific time based on the user specified keyframe data.

The dynamics of particles inside the rotating drum with varying rotational speed are shown in figure 5. The particles are initialized in a matrix form and then released to settle under gravity. Simultaneously, the drum rotates from rest with increasing speed until $t = 0.5$ s and then the rotational velocity decreases until $t = 1.5$ s. It should be noted that after $t = 1$ s, the change in drum rotation direction can be observed.

5 Notes

- Since this tutorial involves UDFs, the custom solver must be built prior to running the simulation.
- Once the custom solver is built, users can modify and save the keyframe files to adjust the belt velocities. There is no need to build the custom solver when modifying keyframe data.
- The user-defined keyword `read_kf` is accessible through the GUI Advanced pane. To completely turn off the keyframe data for the drum, set `read_kf(1)=False`. This will set the default zero rotational velocity to the drum.