# FrictionTest

Generated by Doxygen 1.9.6

# Chapter 1

# Friction Module and Test Model

## 1.1 Overview

This project simulates the motion of a marble falling through a viscous liquid and collects data at regular time intervals. The goal is to estimate the $*$friction rate ($\alpha$)$*$ using the recorded trajectory data.

The project consists of:

- A **friction module** that calculates the friction rate from velocity data.
- A **test model** that generates synthetic trajectory data.
- A **data analysis pipeline** that estimates the friction rate based on the collected data.

## 1.2 Physical Model

The marble's velocity and position evolve according to the following equations:

[ v(t) = (v_0 + \frac{g}{\alpha}) e^{-\alpha t} - \frac{g}{\alpha} ]

[ z(t) = z_0 + \frac{1}{\alpha} (v_0 + \frac{g}{\alpha}) (1 - e^{-\alpha t}) - \frac{g}{\alpha} t ]

where:

- $ z_0 $ is the initial height (m),
- $ v_0 $ is the initial velocity (m/s),
- $ g $ is gravitational acceleration (m/s$^2$),
- $ \alpha $ is the friction rate (1/s).

To estimate $ \alpha $, we compute the numerical derivative of the recorded position data to obtain velocity values, then use an averaging method based on acceleration ratios:

[ \alpha = \frac{1}{\Delta t} \ln \left[ \frac{1}{n - 2} \sum_{i=1}^{n-2} \frac{v_{i+1} - v_i}{v_{i+2} - v_{i+1}} \right] ]

## 1.3 Code Structure

The project consists of the following components:

### 1.3.1 **1. Friction Module (<tt>friction.h</tt>, <tt>friction.cpp</tt>)**

- `double frictionrate(double dt, const rvector<double>& v);`
  Computes the friction rate given a set of velocity samples.
- `rvector<double> numdiff(double dt, const rvector<double>& z);`
  Numerically differentiates position data to estimate velocities.

### 1.3.2 **2. Test Model (<tt>testmodel.cpp</tt>)**

This program generates synthetic trajectory data based on the physical model. The data is stored in a file and can be used for analysis.

### 1.3.3 **3. Model Implementation (<tt>model.h</tt>, <tt>model.cpp</tt>)**

Contains the equations that describe the marble's motion.

### 1.3.4 **4. Data Analysis (<tt>analyze.cpp</tt>)**

Performs numerical differentiation and estimates the friction rate.

## 1.4 How to Build and Run

### 1.4.1 Compiling the Code

To compile the project, simply run:

make all make analysis

# Chapter 2

# Class Index

## 2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

# Chapter 3

# File Index

## 3.1 File List

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

# Chapter 4

# Class Documentation

## 4.1 ModelParameters Struct Reference

### Public Attributes

- double **alpha**
- double **g**
- double **v0**
- double **z0**

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

- model.h

# Chapter 5

# File Documentation

## 5.1 friction.h File Reference

Header file for the friction module.

```
#include <rarray>
```

### Functions

- double frictionrate (double dt, const rvector< double > &v)
  *Computes friction force given velocity and coefficient.*
- rvector< double > numdiff (double dt, const rvector< double > &z)
  *Estimates the velocities using finite differences of the position sample.*

### 5.1.1 Detailed Description

Header file for the friction module.

**Author**

Patrick Deng

**Date**

February 12, 2025 This module computes the friction rate based on velocity samples v taken a time dt apart.

### 5.1.2 Function Documentation

#### 5.1.2.1 frictionrate()

```
double frictionrate (
            double dt,
            const rvector< double > & v )
```

Computes friction force given velocity and coefficient.

**Parameters**

| | |
|---|---|
| *dt* | the timstep size (s). |
| *v* | the velcoity gradient, output from numdiff |

**Returns**

Computed friction rate

#### 5.1.2.2 numdiff()

```
rvector< double > numdiff (
            double dt,
            const rvector< double > & z )
```

Estimates the velocities using finite differences of the position sample.

**Parameters**

| | |
|---|---|
| *dt* | the timstep size (s). |
| *z* | the position of the marble |

**Returns**

Computed velocity gradient

## 5.2 friction.h

Go to the documentation of this file.

```
00001
00006 #ifndef FRICTIONH
00007 #define FRICTIONH
00008
00009 #include <rarray>
00014 double frictionrate(double dt, const rvector<double>& v);
00015
00020 rvector<double> numdiff(double dt, const rvector<double>& z);
00021
00022 #endif
```

## 5.3 model.h

```
00001 #ifndef MODELH
00002 #define MODELH
00003
00004 #include <rarray>
00005
00006 struct ModelParameters
00007 {
00008     double alpha;  // friction constant
00009     double g;      // gravitation accelleration
00010     double v0;     // initial (vertical) velocity
00011     double z0;     // initial height
```

```
00012 };
00013
00014 double z(double t, const ModelParameters& p);
00015
00016 double v(double t, const ModelParameters& p);
00017
00018 rvector<double> compute_model_v(double t1, double t2, double dt,
00019                                 const ModelParameters& p);
00020
00021 rvector<double> compute_model_z(double t1, double t2, double dt,
00022                                 const ModelParameters& p);
00023
00024 #endif
```

## 5.4 testmodel.cpp File Reference

Generates synthetic data for a falling marble in a viscous fluid.

```
#include "model.h"
#include <fstream>
#include <iostream>
#include <string>
#include <boost/program_options.hpp>
#include <boost/lexical_cast.hpp>
```

### Functions

- void generate_data (const std::string &filename, double t1, double t2, double dt, const ModelParameters &p)

  *Generates trajectory data and writes it to a file. This function computes the vertical position of the marble at different time steps using the provided model parameters and saves the data to a specified file.*
- int read_command_line (int argc, char ∗argv[ ], std::string &filename, double &t1, double &t2, double &dt, ModelParameters &p)

  *Parses command-line arguments to configure the test model. This function reads parameters from the command line and assigns them to variables for simulation. If no parameters are provided, default values are used.*
- int main (int argc, char ∗argv[ ])

  *Main function to generate test model data. Reads command-line arguments, sets up simulation parameters, and generates a dataset.*

### 5.4.1 Detailed Description

Generates synthetic data for a falling marble in a viscous fluid.

**Author**

Patrick Deng

**Date**

February 12, 2025 This program simulates and records the trajectory of a marble falling in a viscous liquid by computing its vertical position over time using a given physical model. The data is saved in a file for further analysis.

The program allows customization of parameters such as the friction coefficient (alpha), gravitational acceleration (g), initial velocity (v0), initial height (z0), and tims step (dt).

### 5.4.2 Function Documentation

#### 5.4.2.1 generate_data()

```
void generate_data (
            const std::string & filename,
            double t1,
            double t2,
            double dt,
            const ModelParameters & p )
```

Generates trajectory data and writes it to a file. This function computes the vertical position of the marble at different time steps using the provided model parameters and saves the data to a specified file.

**Parameters**

| | |
|---|---|
| *filename* | Name of the output file where data is written. |
| *t1* | Initial time (s). |
| *t2* | Final time (s). |
| *dt* | Time step interval (s). |
| *p* | Model parameters: friction coefficient, gravity, initial velocity, and initial position. |

#### 5.4.2.2 main()

```
int main (
            int argc,
            char * argv[] )
```

Main function to generate test model data. Reads command-line arguments, sets up simulation parameters, and generates a dataset.

**Parameters**

| | |
|---|---|
| *argc* | Number of command-line arguments. |
| *argv* | Array of command-line arguments. |

**Returns**

Returns 0 on success, or an error code from `read_command_line()`.

#### 5.4.2.3 read_command_line()

```
int read_command_line (
            int argc,
```

```
char * argv[],
std::string & filename,
double & t1,
double & t2,
double & dt,
ModelParameters & p )
```

Parses command-line arguments to configure the test model. This function reads parameters from the command line and assigns them to variables for simulation. If no parameters are provided, default values are used.

**Parameters**

| | |
|---|---|
| *argc* | Number of command-line arguments. |
| *argv* | Array of command-line arguments. |
| *filename* | Name of the output file where data will be stored. |
| *t1* | Reference to the variable storing the initial time. |
| *t2* | Reference to the variable storing the final time. |
| *dt* | Reference to the variable storing the time step. |
| *p* | Reference to the `ModelParameters` structure storing physical parameters. |

**Returns**

Returns 0 on success, 1 if help is requested, and 2 for errors.

# Index