ENGG1811 Assignment 2: Simulation and Design of Passive Suspensions

- **Due date:** 5pm, Monday Week 11 (12/August/2019) .
- **Late Penalty:** Late submissions will be penalised at the rate of **10% per day** (including weekends). The penalty applies to the maximum available mark. For example, if you submit 2 days late, maximum available marks is 80% of the assignment marks. Submissions will **not** be accepted after 5pm Wednesday of Week-11.
- **Individual Assignment:** this is an individual assignment.

Change log:

- **[06:40pm 05/Aug]:** After relaxing TOL to 10e-4, some students are now passing the tests for Task-1 (because of the relaxed TOL of 10e-4), however they **fail the test for Task-2** (difference of approx 32). This is because small changes in `vs` results in a larger difference for Task-2 (discomfort value, difference of approx. 32).
  - I have now changed the argument of the function `calc_discomfort` to `vs_ref` (the reference value, not your value), this should address the issues and you should be able to pass tests for both Task-1 and Task-2.
  - You can **download the revised test_task1_task2.py from here.**
- [06:40pm 01/Aug]: Change TOL in the file `test_task1_task2.py` to 10e-4 (earlier it was 10e-6) or download the file test_task1_task2.py from here.
- [06:40pm 01/Aug]: You can test "Task 1" using the file `test_task1_task2.py` (available in assign2.zip) or available here.
- [07:10am 30/Jul]: Please read "Hints" in every section, and also before the Task-1. More explanations added to the Task3 and 4.

Introduction

This assignment gives you an opportunity to work on a small-scale engineering design problem in python. The engineering system that you will be working on is a passive suspension, which is used in vehicles to reduce the amount of vertical vibration. A passive suspension is made of multiple components and the parameter of each component must be chosen correctly so that the passengers get a comfortable ride. The engineering design problem is to choose the right parameters and you will use python programming to solve this problem. The first step is to write a program to simulate the motion of a vehicle with suspension. You will then use the simulation result to evaluate the level of comfort for different choices of suspension parameters.

Learning objectives

- Applying programming to solve a simple engineering design problem
- Writing a python program to simulate an engineering system
- Applying a number of python features, which include vectorisation, built-in functions and others
- Applying good software engineering practices, proper documentation, program style

Assignment overview

This assignment is divided in 4 main tasks:

- **Task-1 and Task-2 (Simulation):** Simulation of a vehicle with a passive suspension.
- **Task-3 and Task-4: (Design)** Evaluating the comfort level of different suspension designs, and choose the designs that give the most and least comfortable rides. (Note: The normal practice is to look at the best few designs and don't care about the worst. As an exercise, we look at the best and the worst so that you can see the contrast.)

In the following, we will first give an introduction to passive suspension and vehicle modelling. The introduction is meant to give you some intuition on the design problem. After that we will tell you what you need to do for each task.

Passive suspensions

Passive suspensions are used to reduce the vibration experienced by passengers, but they can also be used to improve the tyre grip and other performance measures. For this assignment, we will only be concerned about vibration reduction or comfort.

There are many designs for passive suspension. For this assignment, we consider a passive suspension consisting of a spring, a damper and an inerter in parallel, as depicted in Figure 1.

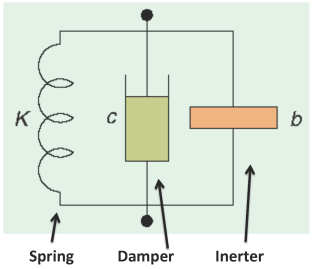


Figure 1. A passive suspension. Figure modified from [1].

The classical method to reduce vibration is to use a spring and a damper where the damper is used to slow down the motion of the car body. A new method is to use a new mechanical device called an *inerter* [1]. An inerter can store kinetic energy temporarily. Intuitively, a shaking car has excessive kinetic energy in the vertical axis, so the inerter can absorb and store some of this energy temporarily. The energy in the inerter can be released later in an orderly manner without the passengers feeling sudden movements. The first ever deployment of inerter was in the 2005 Spanish Grand Prix by the MaLaren Formula 1 racing team, which also happened to have won that race. There are a few interesting tidbits surrounding the use of inerters, including McLaren invented the decoy name J-damper so that its rivals would not know that it was actually an inerter and a spy scandal, see [1] if you are interested.

The passive suspension has three parameters:

- Spring stiffness *k*. (A large *k* means a stiff spring which is hard to stretched.) (python variable `k`)
- Damping coefficient *c*. (A larger *c* means a larger resistance to movement.) (python variable `c`)
- Inertance *b*. (A large *b* means the interter can store more kinetic energy.) (python variable `b`)

The design problem is to choose values of *k*, *c* and *b* so that the ride is comfortable. A passive suspension can only be designed together with a vehicle, so we need to look at the vehicle model now.

The vehicle model

For this assignment, we will use *a quarter car* model for the vehicle. The quarter car model consists of one wheel/tyre and a quarter of the car body. It is commonly used to evaluate suspension designs at the initial stage. Of course, a full car model will be used for the final design but it is too complicated for this assignment.

Engineers very often have to make simplified models in order to derive mathematical models for real-life engineering systems. You will learn how to do this in later years. Figure 2 shows a simplified quarter car model.

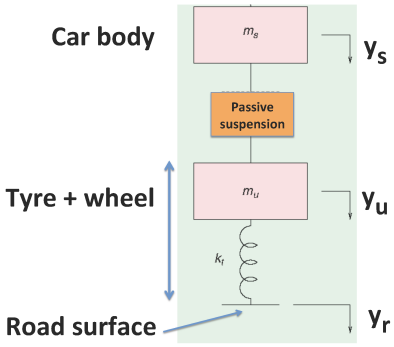


Figure 2. A quarter car. Figure modified from [1].

The quarter car model consists of two lumps of masses. The mass m_s (python variable name `ms`), which is on top, is the mass of 1/4 of the car body. (Note: The technical name is the sprung mass, hence the subscript *s*.) The lower mass m_u (variable name `mu`) is the mass of the tyre and wheel. (Note: Technically this is the unsprung mass, hence subscript *u*.) The tyre is a bit elastic and is represented by a spring with stiffness k_t (variable name `kt`). The passive suspension sits between the car body and the wheel/tyre.

There are three vertical displacements (see Figure 2) that we are interested in:

- y_r is the height of the road surface from a reference level (variable name `yRoad`)

- y_u is the vertical displacement of the center of wheel/tyre from a reference level (variable name `yu`)
- y_s is the vertical displacement of the center of the quarter car body from a reference level (variable name `ys`)

As the quarter car moves, y_r, y_u and y_s change. That means these displacements are functions of time and we should write them as $y_r(t)$, $y_u(t)$ and $y_s(t)$. The corresponding python variables `yRoad`, `yu` and `ys` are therefore arrays.

We also need two velocities for the quarter car model:

- v_u is the vertical velocity of the wheel/tyre (variable name `vu`)
- v_s is the vertical velocity of the quarter car body (variable name `vs`)

These velocities should be functions of time and we write them as $v_u(t)$ and $v_s(t)$. The corresponding python variables `vu` and `vs` are arrays.

The aim of the mathematical model for the quarter car is to determine $y_s(t)$, $y_u(t)$, $v_s(t)$ and $v_u(t)$. The mathematical model assumes the following are given:

- The height of the road $y_r(t)$ over time. We call this the road profile.
- The parameters: k, c and b (from the passive suspension) and m_s, m_u and k_t (from the quarter car)

We have placed the mathematical model for the quarter car on a separate page. We believe it is best for you to understand what you need to do for this assignment first before dwelling into the mathematical model. You should be able to understand what you need to do for the assignment without going into the mathematical model at this stage. (The model is here and you can read it later.) We will now describe what you need to do for the three tasks.

Overview of tasks

We have divided the work into a number of tasks.

- Task 1 and Task 2: are on simulation
- Task 3 and Task 4: are on engineering design

Supplied files

The supplied files are in assign2.zip (click here)

> Hint: You are strongly encouraged to read (or re-read!) the lecture notes, the code examples and the labs on numpy. In particular, look for numpy functions relevant to the following tasks. See the code examples on numpy from week-06 (click), week-07 (click) and week-08 (click) .

Task 1: Simulation of the quarter car

The aim of this task is to write a python function `simulate_qc` (which should be in a file with name `simulate_qc.py`) to simulate the quarter car. You can find a template for this function in `simulate_qc_template.py` (in assign2.zip). You should rename it as `simulate_qc.py` before you start. The declaration of the function `simulate_qc` is:

```
def simulate_qc(time_array, y_road, ms, mu, kt, k, b, c) :
```

The above function **returns four arrays** . These four arrays contain the following simulation outputs:

```
Outputs:
ys     the verticle displacement of the center of the car body
       from a reference level (array of floats)
yu     the verticle displacement of the center of the wheel/tyre
       from a reference level (array of floats)
vs     the verticle velocity of the quarter car body  (array of floats)
vu     the verticle velocity of the wheel/tyre (array of floats)
```

The inputs are:

```
  Inputs:
  time   time_array
```

```
y_road  an array of road heights
ms      the mass of 1/4 of the car body
mu      the mass of the tyre and wheel
kt      tyre stiffness
k       spring stiffness
b       inertance
c       damping coefficient
```

The implementation of `simulate_qc` requires the mathematical model for the quarter car. The **model is here (click here)**, you need to use equations 7 to 12. You can use the python simulation programs from the lab.

> **Hint:** You can use the python simulation program `para_ODE_ext_lib.py and para_speed_height_by_ODE.py` (code from Week 7's lecture) or the material from "Lab 08: Simulation and its applications" as a starting point to develop the function for this task.

You can assume the following **initial conditions**: vs(1) = vu(1) = ys(1) = yu(1) = 0.

Testing: You can test "Task 1" using the file `test_task1_task2.py` (available in assign2.zip).

Task 2: A function to calculate discomfort

The aim of Task 2 is to determine the `discomfort` level for a given set of suspension parameters. Intuitively, a comfortable ride means the passengers are not experiencing much vibration. We can express this quantitatively by calculating how much acceleration the car body experiences. The higher or longer the acceleration is, the more uncomfortable the ride is. (Note: An important part of using computers to perform engineering design is to express the design objective quantitatively. You will learn that in later years but this assignment will show you how to do that.)

Since the function `simulate_qc` gives us the velocity of the car body, we can use it to determine the acceleration and subsequently the discomfort level. For this task, you are asked to write a python function

```
def calc_discomfort(vs , dt):
```

The above function should be in a file `calc_discomfort.py` .

The inputs and output values are:

```
Purpose:
Determining the discomfort level for a given set
of suspension parameters

Inputs:
 vs    the verticle velocity of the quarter car body
 dt    time increment

Output:
 discomfort: a scalar representing the discomfort level
 for the given vehicle and suspension parameters
```

Let us assume that the array `vs` has `n` elements and let us use `dt` to denote the time increment used in the array `time`. We can use `vs` to calculate the acceleration at `(n-1)` time instances:

```
a[i] = ( vs[i+1] - vs[i] ) / dt
```
where $i = 0, 1, \ldots, n-3, n-2$

where `vs[i]` is the *i*-th element of the array vs and `a[i]` is the *i*-th element of the acceleration array `a`. The discomfort level is then given by

$$\text{discomfort} = a[0]^2 + a[1]^2 + \ldots + a[n-3]^2 + a[n-2]^2$$

This should be the output of the function `calc_discomfort`. Intuitively, this calculation says the discomfort is higher if the acceleration is higher.

> **Hint:** Consider the python and numpy functions like `numpy.sum`, and `numpy.diff`, they may prove useful here.

Testing: You can use the python program `test_task1_task2.py` (a file in **assign2.zip**) to test whether your `calc_discomfort` function is working correctly. If the reported error is small, i.e. less than 10^{-4}, then it should be fine.

Important requirement on implementation: The calculation of the discomfort level from the array `vs` can be done without using any loops. You will only receive full marks for this part if the calculation is done **without** using loops, otherwise you will receive a **reduced** mark if loops are used.

Task 3: Calculating discomfort level for many pairs of (inertance, damping coefficient)

The function `calc_discomfort` allows you to determine the discomfort level for each set of suspension parameters: spring stiffness *k*, damping coefficient *c* and inertance *b*. For simplicity, we will not change the value of *k*. We will calculate the discomfort level for many different pairs of (inertance,damping coefficient) or (*b*,*c*) values.

```
def explore_qc(time_array, y_road, ms, mu, kt, k, inerter_array,
                damping_coefficient_array):
```

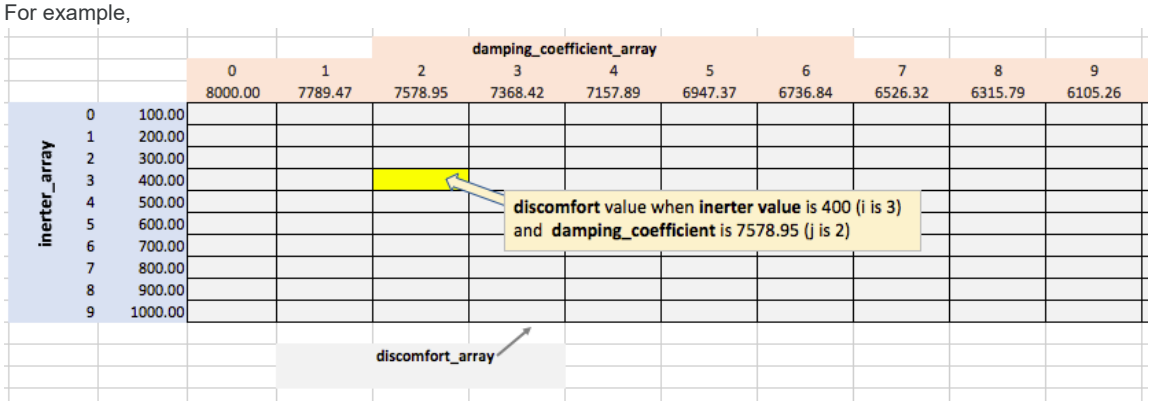
The above function should be in a file `explore_qc.py` .

```
Purpose:
Determining the discomfort levels for a given damper values and inerter values

Inputs:

  time    time_array
  y_road  an array of road heights
  ms      the mass of 1/4 of the car body
  mu      the mass of the tyre and wheel
  kt      tyre stiffness
  k       spring stiffness
  inerter_values              inertance values (array of type float)
  damping_coefficient_values  damping coefficient values (array of type float)

Output:
  discomfort_array   2-dimentional numpy array with discomfort values for
                     given damper values and inerter values (read the specs)
```

The steps for this Task are:

1. Create a 2-dimentional zero array `discomfort_array` with `len(inerter_array)` rows and `len(damping_coefficient_array)` columns.
2. The `(i, j)` element of the 2-dimentional array `discomfort_array`, i.e. `discomfort_array(i,j)`, should be assigned the discomfort level of a suspension when `inerter_values[i]` and `damping_coefficient_values[j]` are used.

   For example,



   You can assume all other parameters are as specified.

You **can use loops** to complete **this task**.

Testing: You can use the file `test_task3_task4.py` to check whether you have calculated the array **discomfort_array** correctly or not.

> Hint: please read the examples in the file `numpy_2d_examples.py`

Task 4: Determining the (inertance, damping coefficient) pairs that give, respectively, the best and worst comfort

The engineering design problem is to choose good design parameters to meet our design requirements. In our case, a design has two design parameters `inertance` and `damping coefficient`.

By using the `discomfort_array`, determine the (inertance, damping coefficient) pair that gives the **best** comfort. For example, the pair that gives the **smallest** value of discomfort level in the array `discomfort_array`.

For comparison purpose, we will also determine a poor design which we define as the design that maximises the level of discomfort, that is less than or equal to `discomfort_upper_limit`. Determine the (inertance, damping coefficient) pair that gives the **worst** comfort, that is less than or equal to `discomfort_upper_limit`. For example, the pair that gives the **largest** value of discomfort level in the array `discomfort_array`, that is less than or equal to `discomfort_upper_limit`.

Once you have obtained the best design and the poor design, you need to return these four values from the following function you need to implement for his task.

```
def optimise_qc(discomfort_array, inerter_array, damping_coefficient_array,
                        discomfort_upper_limit):
```

The input and output values are:

```
Inputs:
  discomfort_array   2-dimentional numpy array with discomfort values for
                     given inerter_values and damping_coefficient_values (read the specs)
  inerter_values     inertance values (array of type float)
  damping_coefficient_values  damping coefficient values (array of type float)
  discomfort_upper_limit      maximum discomfort value to calculate worst comfort
                              (i.e. 'max_inerter' and 'max_damping_coefficient' values)

Output:
  min_inerter and min_damping_coefficient  the pair that gives the smallest value of discomfort
  max_inerter and max_damping_coefficient  the pair that gives the worst value of discomfort, that
                                           is less than or equal to a given 'discomfort_upper_limit'
```

You should complete this task using the `min` and `max` functions, **without** using any loops. You can only get full marks if your solution does **not** use loops. If your solution requires loops, then you can only get a **reduced** mark.

A requirement for Task 4 is that you should complete this task **without** using any loops. You can only get full marks if your solution does **not** use loops. If your solution requires a loop(s) for this task, then you can only get a **reduced** mark.

> Hint: You can easily implement this function WITHOUT using a loop structure. Please read (or re-read!) lecture notes, the code examples and the labs on numpy. In particular, look for numpy functions that may help you to solve problems related to Task-4. For example, min, max, argmin, argmax, boolean indexing, boolean masking (Boolean addressing for assignment), etc. See the code examples on numpy from week-06 (click), week-07 (click) and week-08 (click) . You may find the following example useful, week-08 lecture example (click) .

Testing: You can use the file `test_task3_task4.py` to check your answers for this task.

Remark: We have used exhaustive search here to determine the best and worst suspension parameters. This is certainly not the most efficient algorithm but you will learn better optimization methods in later years.

Style

You should make sure that all your files are properly documented with appropriate comments. Variables that you use should have well chosen names and their meaning explained. Appropriate style should be used.

Assessment

The following table shows the maximum possible marks for the tasks. Note that there are specific requirements for some tasks and the maximum is reduced is those requirements are not met.

Marks	Feature/Assessable Item
7	Task 1 (Simulation of the quarter car).
3	Task 2 (A function to calculate discomfort). Reduced maximum (for loop usage): 1.5
6	Task 3 (Calculating discomfort level for many pairs of (inertance, damping coefficient)).
4	Task 4 (Determining the best and worst suspension), Reduced maximum (for loop usage): 1.5
5	Style analysis
25	Total mark (rescaled to 10% of overall assessment)

Submission

Later instructions on how to submit this assignment will be available here.

Originality of Assignments

As with all material submitted for assessment, this must be substantially your own work. It's OK to discuss approaches to solutions with other students, and to get help from tutors and consultants, but you must write the Basic code yourself. Sophisticated software is used to identify submissions that are unreasonably similar, and marks will be reduced or removed in such cases.

Further Information

Use the forum to ask general questions about the assignment, and keep an eye on it for updates and responses.

Reference

[1] M.Z.Q. Chen et al. The missing mechanical circuit element. IEEE Circuits and Systems Magazine (First Quarter, 2009), pp. 10-26, 2009. URL: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=4799284. (Note: As a UNSW student, you can download this research article for free. You get direct access if you are on the university computer network, otherwise you need to access through the library if you are outside of the university.)
