**Glucose Control Optimization Challenge**

The blood glucose levels of healthy individuals are maintained between 70 mg/dL to 180 mg/dL. When diabetes develops, the body is no longer able to create or respond to insulin and as a result, glucose levels become chronically elevated leading to hyperglycemia which can lead to long-term micro- and macro-vascular complications. To drive down these elevated glucose levels, insulin must be injected into the body; Injecting too much, however, can lead to dangerously low glucose levels.

Diabetic patients on multiple dose injection (MDI) therapy typically take 3 or more insulin/bolus injections per day to maintain the glucose levels. The optimal amount of bolus a patient injects depends on the current blood glucose reading (mg/dL), the amount of carbohydrate (grams) the patient plans to eat, the patient’s insulin-sensitivity (drop in glucose caused by insulin).

Accurately predicting a diabetic patient’s future glucose level is an important insight for insulin and meal management. In addition, recommending when and how much insulin to inject to offset a meal or get oneself out of hyperglycemia gives a patient a deeper intuition about their own personal sugar-insulin interactions in the body.

**The Challenge: Insulin Dosage optimization for Multiple Daily Injections (MDI) Therapy**

In this challenge you will be asked to provide the timing and amount of insulin required to compensate each meal in a 24 hour period so that each patient maximizes their time in the ideal range (between 70mg/dL-180mg/dL). Each patient will be modelled by a black-box simulator with personalized parameters hidden from the challenger. You will be provided the meal timings and the carbohydrates (CHO) consumed in each meal along with a range of total insulin boluses for each user. Each day will consist of at least 3 meals. The solution should be optimized to:

1. maximize time in range,
2. minimize the compute time in which a solution is produced. \*The compute time should be understood as “wall-time,” or the time it takes to execute all the steps in the optimization or training of a model. If using a pretrained model, the inference time will be used. All submissions will run on the same computer. GPUs will not be available.
3. minimize the amount of insulin used in the regimen.

The figures below illustrate instances in which the glucose levels are simulated. Figure 1a shows a simulation of 3 meals and no insulin boluses, where glucose levels rise without control. Figure 1b shows a simulation with the same 3 meals, but also including a regimen of insulin injections that keeps the patient in range for 99% of the day. Note, in some cases it will not be possible to achieve 100% time-in-range (or even 80%), such as when glucose starts out of range, or when certain types of meals are consumed.

A close up of a map

Description automatically generatedA close up of a map

Description automatically generated

|  |  |
| --- | --- |
| a. | b. |

Figure 1. A simulated glucose trace for a patient who has consumed 3 meals and no boluses (a) and a glucose trace corrected by insulin boluses for identical meals (b).

**The Data**

The subject profiles along with the daily meal data and pre-requisites are provided in the subject\_profiles csv file.

|  |  |
| --- | --- |
| **Column Names** | **Details** |
| **day\_id** | Index of the day; a unique value that is passed to the simulator and retrieves parameters specific to the patient and meals consumed that day |
| **start\_sg** | Blood glucose reading (mg/dL) at time zero: two hours before the first meal |
| **daily\_meal\_count** | The total number of meals in that day |
| **daily\_bolus\_count\_allowed** | The maximum number of bolus doses that are allowed for that day |
| **meal\_N\_time** | Timing of the Nth (where N is an integer from 1 to 5) meal (number of minutes elapsed from start T=0) |
| **meal\_N\_carb** | Carbohydrate content of meal (grams) at meal\_N\_time |
| **min\_daily\_bolus\_allowed** | Minimum total daily bolus (used only for reference; no penalty will be added if the solution is below this value). |
| **max\_daily\_bolus\_allowed** | Maximum total daily bolus (used only for reference; no penalty will be added if the solution is above this value). |

**System requirements:**

1. Python 3.7 (64bit-version). It is recommended to use anaconda python. Python libraries: pandas (version 1.12.0 or higher) and numpy (version 1.17.0 or higher).
2. Find the version of the simulator module compiled for your operating system in the **simulator directory**:
   1. (Mac) *simulator\_wrapper.cpython-37m-darwin.so*
   2. (Windows) *simulator\_wrapper.cp37-win\_amd64.pyd*
   3. (Linux) *simulator\_wrapper.cpython-37m-x86\_64-linux-gnu.so*

**Challenge Constraints**

Each day provided may have 3-5 meals and varies by each virtual subject. The maximum number of bolus that can be injected in a day is provided in the ‘daily\_bolus\_count\_allowed’ column.

**Rules for optimization:**

1. Meal timings and amount provided are fixed
2. Total number of insulin boluses allowed per day = Number of Meals + 1

**Code snippet for running the Simulator**

The simulator function is contained within the provided python module and can be imported and called with the following code snippet.

**import numpy as np**

**from** simulator\_wrapper **import** \*

#the second dimension of the bolus matrix should not

#exceed the number of meals +1 consumed that day and should be

#passed as a numpy array

day = 1 #the day to be simulated (0-39)

arr = [[500, 900, 1080], #time

[ 5, 10, 8]] #bolus amount

bolus\_matrix = np.array(arr)

#the output of the simulator is an array of the 1-minute time steps (0-1440, an array of the glucose trace and the score

t, g, score = run\_simulator\_day(day, bolus\_matrix)

**Simulator arguments:**

1. day\_id: the day to run. an integer from 0 to 39. The day contains metabolic parameters specific to a given subject as well as the meals consumed that day.
2. The insulin treatment as a 2 x N numpy array. The first row in the array should contain values of the times at which to bolus and the second row should contain values of the amounts of insulin in each bolus. N denotes the number of boluses prescribed and should not exceed the number of meals + 1. An example of the bolus array is presented here.

arr = [[500, 900, 1080], #time

[ 5, 10, 8]] #bolus

bolus\_matrix = np.array(arr)

**Simulator output:**

1. An array of length 1440 of 1 minute time steps from 0 to 1440.
2. An array of length 1440 containing the corresponding simulated glucoses value for each time step.
3. An integer value reflecting the evaluation metric for the simulated glucose trace. This metric is calculated in the following way:
   1. Count all points inside the accepted glucose range (70-180 mg/dL).
   2. Subtract all points in hypoglycemia (below 70 mg/dL).
   3. Divide by length of the trace (1440)
   4. Multiply by 100.

**How to win the competition:**

Each submission will be evaluated on its performance over the whole hold-out day simulations. A similar subject\_profiles.csv file will be available for the hold-out set. The winner of the competition will be determined by the following criteria:

1. Highest total percent time in range, including hypo penalties, when rounded down to the nearest whole number.
2. All ties at the highest full percentage point will be broken with the lowest runtime rounded to the nearest second. If second place rounds up to the same percentage point as first place, it will be considered a tie.
3. All ties at the lowest runtime will be broken by lowest bolus dose.

**How to submit your optimizer to the leaderboard**

We will be running your code on a Linux EC2 instance, so if you are developing on Windows or Mac, please make sure to write your code in a system-independent fashion. The Linux EC2 instance will have 4 CPUs and 7GB of RAM. Your code will be able to run for an hour after which the instance will be turned off. If your code does not run to completion during that time, your results will not be posted to the leaderboard nor will it be submitted for final evaluation.

1. **Update rename.this.file.to.submission.py** script with the contents of your own submission, and rename the file so that it reads **submission.py.**
   1. It should contain a function called bolus\_optimizer() which takes the day\_id as an argument; and should return the day\_id as well as a single bolus matrix in the form of a 2 x N numpy array.
2. Make sure to **update requirements.txt** include all the libraries needed to run submission.py. Also be sure to upload any other dependencies to the repository e.g. pre-built models, python scripts, etc.
3. The evaluate.py script will call your bolus\_optimizer and run the evaluations. Make sure your submission.py scripts runs with evaluate.py. Please see the example submission.py for reference.
4. Once you have verified that your submission script runs, push the change to the git repository. Execute the following commands in the command line:

git add .

git commit -m “submission for evaluation”

git push origin

1. You will receive an email with the leaderboard. Your most recent submission will be appended to the list.

**How to submit your optimizer for final evaluation**

1. Push your code, all dependencies and updated requirements.txt file by September 10th. All submissions will be run on a Linux EC2 with specifications stated above. For final evaluation, submissions will run against 23 days unseen in the development set.