Exercise 2 – Diabetes Vs. 1.1

The goal with this exercise is to use mathematical modelling to solve a practical problem: Regulation of insulin infusion to diabetic patients. You should:

- Identify significant issues
- Formulate a mathematical model, which solves these issues
- Implement the model in a computer program
- Use the implemented model for analysis of data
- Report the results from the analysis and discuss the validity of the model

The report for this exercise must be 5 pages at most, including graphs, tables, and images (excluding frontpage and appendices). In this exercise a number of questions are asked. The report, however, should not be a list of answers, but instead be a coherent documentation and discussion of the analysis performed.

The exercise consists of several parts. The first is the implementation of the Medtronic Virtual Patient Model. This part is necessary to solve the subsequent parts. We then implement a so called PID controller to regulate the infusion of insulin and further investigate the need for bolus injections when the patient consumes a large meal.

To solve the exercise, you can use either MATLAB or PYTHON. It is up to you which programming language you use.

Part 1 - Medtronic Virtual Patient Model

An illustration of the model is shown in figure 1. Based on the carbohydrate uptake $d(t)$ [g CHO/min] (food), and the insulin infusion $u(t)$ [mU/min] we can model the blood glucose level $G(t)$ [mg/dL]. If the the blood glucose is too low - hypoglycemia - it can lead to unconsciousness, coma and death. If too high - hyperglycemia - it will often present as hunger, thirst and frequent urination. It is not as acutely dangerous as hypoglycemia, but is certainly dangerous, especially over time. The goal is thus to maintain a stable blood glucose level.

The Medtronic Virtual Patient (MVP) model of a person with type 1 diabetes is

$$\dot{I}_{sc}(t) = \frac{u(t)}{\tau_1 C_I} - \frac{I_{sc}(t)}{\tau_1} \tag{1}$$

$$\dot{I}_{p}(t) = \frac{I_{sc}(t) - I_p(t)}{\tau_2} \tag{2}$$

$$\dot{I}_{eff}(t) = -p_2 I_{eff}(t) + p_2 S_I I_p(t) \tag{3}$$

$$\dot{G}(t) = -\left(GEZI + I_{eff}(t)\right) G(t) + EGP_0 + \frac{1000 \cdot D_2(t)}{V_G \tau_m} \tag{4}$$

$$\dot{D}_1(t) = d(t) - \frac{D_1(t)}{\tau_m} \tag{5}$$

$$\dot{D}_2(t) = \frac{D_1(t) - D_2(t)}{\tau_m} \tag{6}$$

This can be represented as

$$\dot{x}(t) = f(x(t), u(t), d(t), p) \tag{7}$$

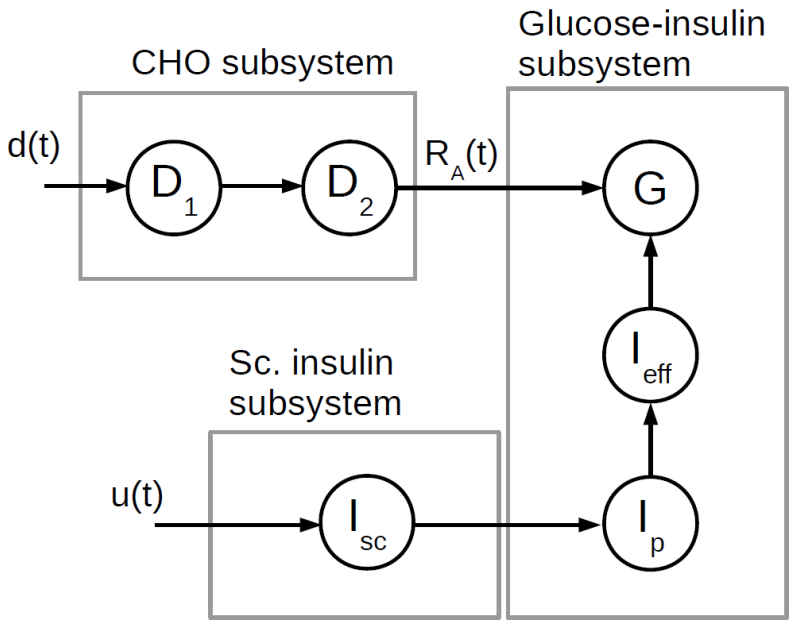

Figure 1: Components in the Medtronic Virtual Patient Model. $d(t)$ is meal intake and $u(t)$ is insulin.

where $x(t)$ contains all the variables in the model $x = \begin{bmatrix} I_{sc} & I_p & I_{eff} & G & D_1 & D_2 \end{bmatrix}$. $u(t)$ is the infusion of insulin and $d(t)$ is the carbohydrate uptake. Finally, we have p, which is all the parameters in the model - listed below.

$$\begin{aligned} \tau_1 &= 49 \text{ min} \\ \tau_2 &= 47 \text{ min} \\ C_I &= 20.1 \text{ dL/min} \\ p_2 &= 0.0106 \text{ min}^{-1} \\ S_I &= 0.0081 \text{ (dL/mU)/min} \\ GEZI &= 0.0022 \text{ min}^{-1} \\ EGP_0 &= 1.33 \text{ mg/dL/min} \\ V_G &= 253 \text{ dL} \\ \tau_m &= 47 \text{ min} \end{aligned}$$

For conceptual definitions of variables and parameters: [1]

- $d(t)$ is meal intake
- $u(t)$ is insulin infusion rate
- $R_A(t)$ is rate of appearance of contribution from meals
- I_{SC} is subcutaneous insulin concentration
- I_P is plasma insulin concentration
- I_{eff} is effect of insulin
- G is glucose concentration
- D_1, D_2 is a two-compartment model of CHO absorption
- C_I is insulin clearance rate

- p_2 is an inverse time constant
- S_I is insulin sensitivity
- $GEZI$ is Glucose Effectiveness at Zero Insulin
- EGP_0 is Endogenous Glucose Production
- V_G is glucose distribution volume
- τ_1, τ_2, τ_m are time constants

Problem 1 - MVP Implement a function that can compute f for the MVP model. This function can be in the form
MATLAB: `function dx = MVPModel(t,x,u,d,parm)`
PYTHON: `def MVPModel(x, t, parm, u, d):`

The function should return a vector with the derivatives of x. Note that even though t is an input parameter to the function, it is not explicitly used in the function.

Problem 2 - Simulation using MVP We now have a function that returns the derivatives of x when given $x(t), u(t), d(t), p$. Using so called ODE solvers, we can use this to simulate the system.
MATLAB:
```
[T,X] = ode15s(@MVPModel,tspan,x0,[],u,d,parm);
tspan = [0 50000];
```
PYTHON:
```
xs = scipy.integrate.odeint(MVPModel, x0, tspan, args=(parm, u, d))
tspan = np.linspace(0,50000,50000)
```

Use $x0 = [0, 0, 0, 0, 0, 0]$ as the inital condition.

Assume that the insulin injection rate is us = 25.04 mU/min and that no meal is given ($d(t) = 0$). What is the blood glucose concentration at steady state?

Optional: You may compare the results to your own implementation of the explicit Euler method.

Part 2 - PID Controller

We now have a way to simulate the blood glucose under different condition using the MVP model. We now turn to the problem of regulating the blood glucose level, through the insulin infusion. As we know, both too high and too low blood glucose values are bad, but very low levels are acutely dangerous.

As a first approach we will look at the Proportional-Integral-Derivative (PID) controller. The basic idea is to look at an error value $e(t)$, i.e. the difference between a set-point and a measured value. The PID will then apply a correction based on a Proportional, Integral and Derivative term of the error value $e(t)$. See figure 2.

The standard form of the equation is:

$$u(t) = K_p \left(e(t) + \frac{1}{T_i} \int_0^t e(\tau) d\tau + T_d \frac{de(t)}{dt} \right), \tag{8}$$

where $e(t)$ is the error value and $u(t)$ is the output. K_p is a tuning parameter for the proportional part, T_i for the integral part, and T_d for the derivative part.

More info can be found here here.

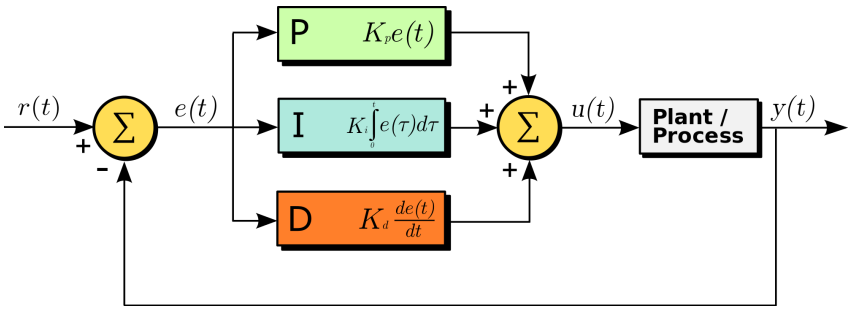

Figure 2: The PID controller. Source: Arturo Urquizo / CC BY-SA (https://creativecommons.org/licenses/by-sa/3.0)

The controller for insulin is given by

$$e_k = y_k - \overline{y} \tag{9}$$
$$P_k = K_P e_k \tag{10}$$
$$I_{k+1} = I_k + K_I e_k \quad K_I = K_P T_s / T_I \tag{11}$$
$$D_k = K_D \left(y_k - y_{k-1}\right) \quad K_D = K_P T_D / T_s \tag{12}$$
$$u_k = \overline{u} + P_k + I_k + D_k \tag{13}$$
$$\tag{14}$$

and returns u_k and I_{k+1}, where

- $\bar{y} = 108\text{mg/dL}$ is the glucose concentration target,
- y_k is the current blood glucose concentration,
- y_{k-1}, is the previous blood glucose concentration,
- us is the insulin steady state,
- K_P, T_I and T_D are the tuning parameters of the PID (to be determined - you may consider KP in the range 0-0.5, Ti in the range 100-1000 minutes, and TD in the range 0-30 minutes)
- $T_S = 5\text{min}$ is the sampling time
- I_{k+1} is an integral term
- u_k is the insulin infusion rate

Problem 3 Formulate the above PID controller as a function:
MATLAB: `function [u,i] = PIDControl(i,r,y,y_prev,us,Kp,Ti,Td,Ts)`
PYTHON: `def PIDControl(i,r,y,y_prev,us,Kp,Ti,Td,Ts):`

where `r`: $\bar{y}$, `y`: y_k, `y_prev`: y_{k-1}, `us`: us, `Kp, Ti, Td`: K_P, T_I, T_D, `Ts`: T_S, `i`: I_{k+1}, `u`: u_k, `us`: $\bar{u}$.

Problem 4 Run closed loop simulation by iterating over a e.g. 500 time steps. In each step initialise the insulin rate by the PID controller and feed the resulting rate into the the MVP model. The `tspan` for the MVP is then only the size of the current iteration, e.g. 5 min. Note, that the PID might output negative insulin rates, if that is the case set $u = 0$ before feeding it to the MVP. Further, assume stable blood sugar `y_prev` = `y` at $t = 0$, and initialise with $i = 0$.

Start with a high initial blood glucose value G, e.g. 200 mg/dL. Use $x_0 = [1.2458, 1.2458, 0.0101, G, 0, 0]$ as the initial conditions in the MVP. Use the output G of the MVP as the initial condition in the next iteration.

Problem 5 Now, tune the 3 PID parameters (hint: there are several ways to do this). One approach is the so called *manual tuning*:

> "If the system must remain online, one tuning method is to first setK_i and K_d values to zero. Increase the K_p until the output of the loop oscillates, then the K_p should be set to approximately half of that value for a "quarter amplitude decay" type response. Then increase K_i until any offset is corrected in sufficient time for the process. However, too much K_i will cause instability. Finally, increase K_d, if required, until the loop is acceptably quick to reach its reference after a load disturbance. However, too much K_d will cause excessive response and overshoot. A fast PID loop tuning usually overshoots slightly to reach the setpoint more quickly; however, some systems cannot accept overshoot, in which case an overdamped closed-loop system is required, which will require a K_p setting significantly less than half that of the K_p setting that was causing oscillation." Source

The questions we want answered are:

- How do you fastest get to a stable blood glucose level?
- How do you quickly get to a stable blood glucose level, while minimising over- and undershoot of the target value?
- Add Gaussian noise to your blood glucose values, to simulate real life measurement errors. How does that affect the performance?
- What is the behaviour if the patient have a large meal, 200 g CHO ($d(1) = 200,\ d(t > 1) = 0$)?
- Discuss the importance of the three parameters

Part 3 - Bolus Calculator

The PID controller works well for smaller disturbances in the blood glucose level. However, if we have a large disturbance e.g. a large meal or lots of sugar, we need something else. One option is to give a insulin bolus, i.e. a single large injection of insulin in connection with the meal. However, low blood sugar is very dangerous, and we need to make sure we do not inject too much. For that we will use a Glucose Penalty Function

$$\rho(G(t)) = \frac{1}{2}(G - \bar{G})^2 + \frac{\kappa}{2}\max\left((G_{\min} - G(t)), 0\right)^2 \tag{15}$$

where $\kappa = 10^6$ is a weight parameter penalizing hypoglycemia, $\bar{G} = 108$ mg/dL is the target value, and $G_{\min} = 70$ mg/dL is the threshold for hypoglycemia. By integrating over a blood glucose curve, we get a measure for how good or bad the blood sugar has been. For all of the following, we will continue the infusion of insulin at `us` = 25.04 mU/min, after the bolus injection.

Problem 6 Make a function that can compute $\phi = \phi\left(x_0, u_0, d_0, p\right)$

$$\phi = \int_{t_0}^{t_0+T} \rho(G(t))dt \tag{16}$$

e.g. by using the left endpoint rule. x_0 is the initial state. u_0 is the insulin bolus given with the meal. d_0 is the meal size. p are the model parameters.

Problem 7 Make a script that minimises the function $\phi = \phi\left(x_0, u_0, d_0, p\right)$ for a given meal size. You can e.g. evaluate the the function for different bolus sizes $u_0 = 0, 0.1, \ldots, 15$ U, and take the bolus size that minimises ϕ. Use $x_0 = [1.2458, 1.2458, 0.0101, 108.2115, 0, 0]$.

Problem 8 Plot the optimal bolus size for different meal sizes in the range 100-150 g of carbohydrates (CHO) ($d(1) = CHO,\ d(t > 1) = 0$).

Problem 9 Try to fit the optimal bolus curve for different meal sizes. Consider using a linear or quadratic fit, and plot the fits along with the optimal bolus sizes previously computed. Discuss the results.
MATLAB `polyfit` and `polyval`
PYTHON `numpy.polyfit` and `numpy.polyval`

Reporting Contents of the report

1. Describe the problem and the background – what is modelled and why?
2. Describe data and experiments
3. Describe the mathematical model - how and why?
4. Describe results
5. Discuss results – how good are the results? To what degree do they reflect the true problem?
6. Conclude – what is the contribution of the analysis?

Hand in The report is uploaded to DTU Inside and to peergrade.io. The code must *also* be included in an appendix in the report.

John Bagterp, Dimitri Boiroux, Anders Nymark Christensen, Harald Løvenskjold Mortensen 2018
Revised Anders Nymark Christensen 2019

References

[1] Dimitri Boiroux and John Bagterp Jørgensen. A nonlinear model predictive control strategy for glucose control in people with type 1 diabetes. IFAC-PapersOnLine, 51(27):192 – 197, 2018. 10th IFAC Symposium on Biological and Medical Systems BMS 2018.