Danmarks Tekniske Universitet

Project 2 - 02526 Mathematical Modeling

23. February 2020

Authors:	***Study No:***
Mads Esben Hansen	s174434
Marcus Lenler Garsdal	s174440
Nicolaj Hans Nielsen	s184335

Contents

1 Introduction

Diabetes is a serious chronic disease and a major cause cause of blindness, kidney failure, heart attacks, stroke and lower limb amputation. In 2016, an estimated 1.6 million deaths were directly caused by diabetes. Another 2.2 million deaths were attributable to high blood glucose in 2012. In 1980 it affected 108 million people, in 2014 that number rose to 422 million, all according to WHO [2]. Much is done to prevent more people from suffering from this condition, however there is still a great need for good, efficient treatment for those who do have to suffer. We will in this report focus on type I diabetes, which is a condition where very little or no insulin is produced by the pancreas. Insulin being a hormone required for the body to use blood sugar [4]. We will try to shed light on how to maintain a steady blood sugar level for type 1 diabetics using and PID and the Medtronic Virtual Patient (MVP) Model [1].

2 Methodology

2.1 MVP

The MVP model is a way to model blood sugar level for a diabetic using a system of differential equations. The model is presented quite extensively in the exercise material, we wil therefore only briefly go over the key concepts here. The idea is that we can find the blood sugar level (glucose level), $G[mg/dL]$, at a given time, t, as the solution to a system of differential equations, $G(t)$. The solution will be only on the carbohydrate uptake $d(t)[gCHO/min]$ (food), and the insulin infusion $u(t)[mU/min]$, and some parameter p. The system of differential equations is:

$$
\begin{gathered}
I'_{sc}(t) = \frac{u(t)}{\tau_1 C_I} - \frac{I_{sc}}{\tau_1} \\
I'_p(t) = \frac{I_{sc}(t) - I_p(t)}{\tau_2} \\
I'_{eff}(t) = -p_2 I_{eff}(t) + p_2 S_I I_p(t) \\
G'(t) = -(GEZI + I_{eff}(t))G(t) + EGP_0 + \frac{1000 D_2(t)}{V_G \tau_m} \\
D'_1(t) = d(t) - \frac{D_1(t)}{\tau_m} \\
D'_2(t) = \frac{D_1(t) - D_2(t)}{\tau_m}
\end{gathered}
$$

and the parameters, p:

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

In order to maintain elegant notation, we define $x(t)$ such that:

$$
\begin{gathered}
x'(t) = f(x(t), u(t), d(t), p) \\
x(t) = [I_{sc} \quad I_p \quad I_{eff} \quad G \quad D_1 \quad D_2]
\end{gathered}
$$

2.2 PID

Proportional-Integral-Derivative (PID) controller is a controller that is widely used industial control system and other applications that requires continuously modulated control e.g. cruise control in a car [3]. The controller works by continuously assessing the error value $e(t)$, and from this computing the desired output, $u(t)$ given by the standard form:

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

Where K_p, T_i, and T_d are tuning parameter for the proportional, integral and derivative part respectively. There is a whole science to best finding these turning parameter, we will therefore limit ourselves to the following 3 facts:
1: The proportional term is used to modulate the response time but can cause overshoot.
2: The integral term is used to modulate steady state error, but can also cause overshoot.
3: The derivative term is a dampening term, that is used to decrease overshoot, but it can also decrease responsiveness.
We will only work with discrete time steps we will therefore find our terms the following way:

$$e_k = y_k - \bar{y}$$
$$P_k = K_p e_k$$
$$I_k = I_{k-1} + \frac{K_p T_s}{T_i} e_k$$
$$D_k = \frac{K_p T_d}{T_s} e_k$$

Where T_d is the time between each sample.

2.3 Bolus

The PID-control works well for small changes in the blood glucose level. However, it cannot cope with sudden big disturbances, such as large meals full of carbs. A common way of dealing with that problem is to inject a single large amount of insulin, an insulin bolus. However, it is difficult to estimate the exact amount of insulin needed. A blood sugar that is too low, hypoglycemia, is worse than one to high, hyperglycemia - of cause neither is desirable. In order to prevent hypoglycemia we introduce a Glucose Penalty Function (GPF), given by:

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

Where $\kappa = 10^6$, $\bar{G} = 108\frac{mg}{dL}$, and $G_{min} = 70\frac{mg}{dL}$, are penalizing parameter, glucose target, and glucose threshold, respectively. We can use this penalizing function to evaluate the performance of the blood sugar level over some time period. We do this by:

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

We note that ϕ can be found as a function of $x(t)$, $u(t)$, $d(t)$, and p.

$$\phi = \phi(x(t), u(t), d(t), p)$$

This can be used to find the optimal bolus size, by finding the argmin to this function, where $x(t)$, $d(t)$, and p are known.

$$u_{optimal} = arg\min_u \phi(x_0, u, d_0, p_0)$$

In our discrete domain calculus is not an option as such, therefore we will find the discrete integral by:

$$\phi = \sum_{\tau=t_0}^{t_0+T} \rho(G(\tau))$$

3 Results

The MVP model is initially used to model the steady state glucose levels in the blood using parameters p described in methodology, $d(t) = 0$, $u(t) = 25.04\frac{mU}{min}$ and initial conditions $x0 = [0, 0, 0, 0, 0, 0]$. The glucose concentration is shown in figure 1, and steady state glucose concentration is found to be $108.1995\frac{mg}{dL}$.

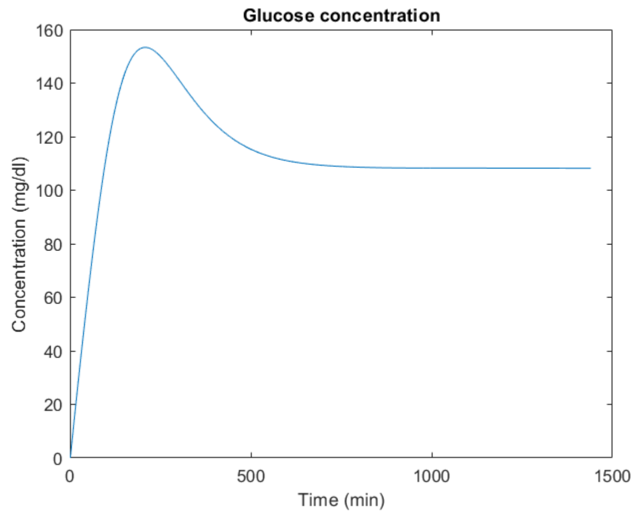
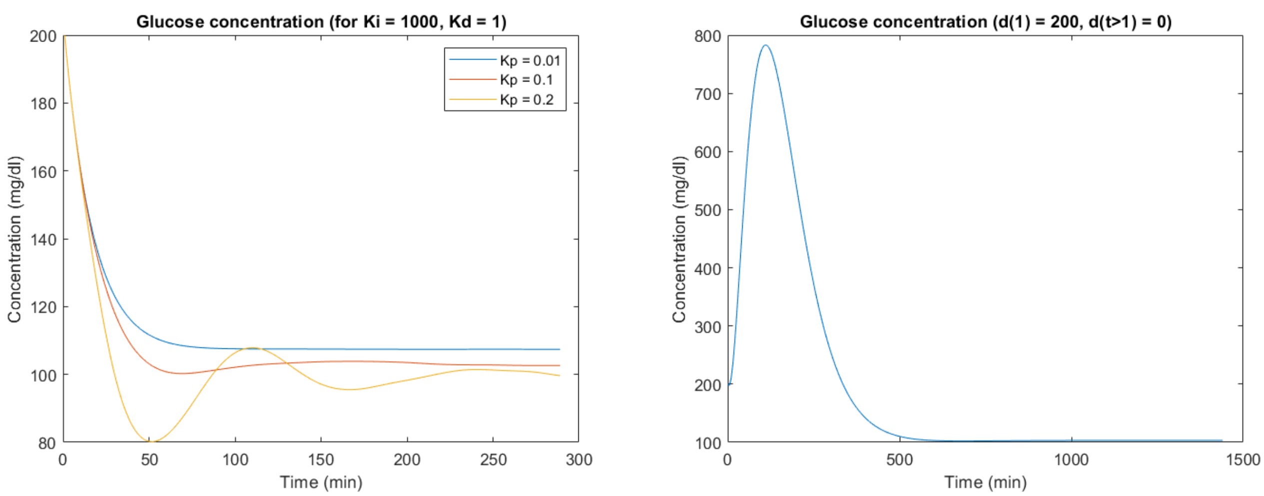


Figure 1: The MVP model simulated under steady state conditions.

The MVP model is expanded to use a PID controller to determine insulin rates to stabilize glucose levels in the blood. To test the PID controller the simulations is run with initial conditions $x_0 = [1.2458,\ 1.2458,\ 0.0101,\ 200,\ 0,\ 0]$. The PID controller was tuned to the parameters $K_p = 0.01,\ K_i = 1000,\ K_d = 1$, and the influence of K_p can be seen in figure 2a.



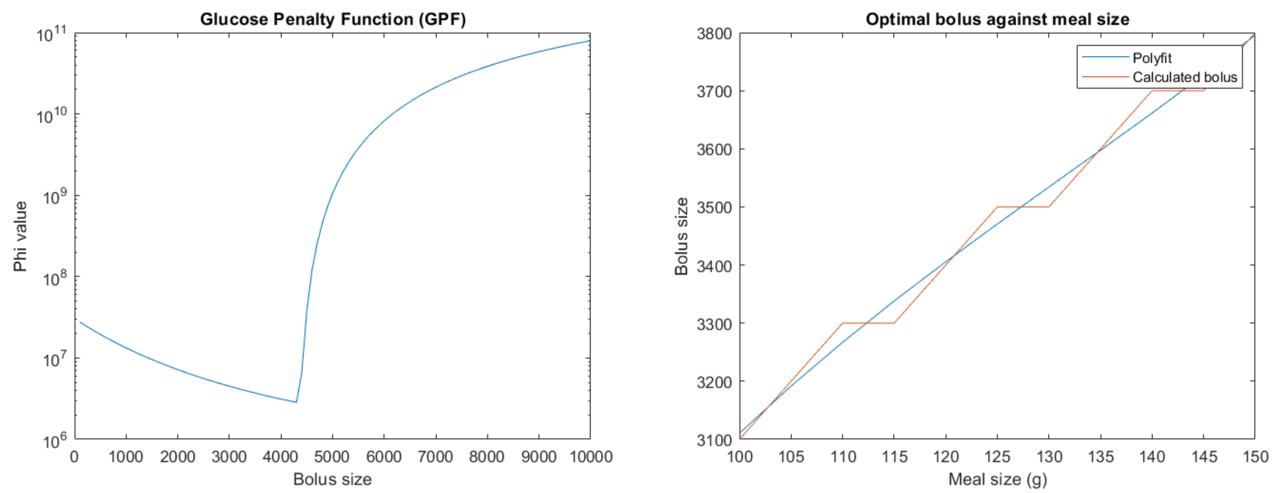
(a) The influence of the K_p parameter in PID tuning.

(b) The influence of a 200g CHO meal.

Figure 2: Influences on the PID controller

The PID controller was also tested when gaussian noise was added to the measurements, and it was found that the PID converges to a slightly lower glucose concentration.

From figure 2b it can be seen how high the glucose concentration in the blood reaches due to a large meal. This indicates that the PID controller does not perform well under large disturbances in the blood glucose level. A solution to this could then be to implement an insulin bolus into our model.
Implementing the glucose penalty function (GPF) we can determine the optimal bolus size for a given meal size. In figure 3a the GPF function is shown for $d(1) = 200, d(t > 1) = 0$. Here the bolus size is in mU and the optimal bolus size can be found at the minimum. We found the optimum to be around 4300 mU.



(a) GPF as a function of bolus size.

(b) Interpolation of optimal bolus size as function of meal size.

Figure 3: Influences on the PID controller

In figure 3b the same analysis is carried out for different meal sizes in the range $100 - 150g$ where $d(1) = CHO, d(t > 1) = 0$ and bolus sizes in mU. This results in a relation between meal size and the optimal bolus. Interpolating the values in figure 3b yields a fourth degree polynomial describing optimal bolus size as a function of meal size with the coefficients in descending order $1.0e + 08 \cdot [-1.3986, 0.7133, -0.1358, 0.0116, -0.0003]$.

4 Discussion

In the steady state solution of the MVP model, our formulated mathematical model was tested under the assumption that the patient was not eating. The steady state blood level was found to be $108.1995 \frac{mg}{dL}$. Normal blood sugar levels during fasting should be below $100 \frac{mg}{dL}$, and before a meal can be expected in the range $70 - 130 \frac{mg}{dL}$ [1]. This tells us that our model is operating in the correct range, and can be used to run simulations to analyze blood level behaviors. Implementing a PID controller allows us to regulate insulin levels around our determined steady state blood sugar level. For a diabetic blood sugar levels after a meal are expected to reach at max $180 \frac{mg}{dL}$. As shown in figure 2b, when we allow the PID controller to determine insulin rates the blood sugar level spikes to values between $700 - 800 \frac{mg}{dL}$ after a meal. This indicates that a PID controller is not a suitable method for determining insulin rate in the case where a patient is eating. To handle the large spike in blood sugar after a meal, an insulin bolus is suggested and implemented in the model.

5 Conclusion

We wanted to shed light on how to maintain a steady blood sugar level for type 1 diabetics using and PID and the Medtronic Virtual Patient Model. We found that the model reached a steady state at $108.2 \frac{mg}{dL}$ with an insulin

[1]https://www.webmd.com/diabetes/guide/normal-blood-sugar-levels-chart-adults

infusion rate at $25.04 \frac{mU}{min}$. We used a PID to moderate the infusion rate, and we manually found good tuning parameters as $K_p = 0.01,\ K_i = 1000,\ K_d = 1$. We plotted the influence of $200g$ CHO meal and found that the PID alone was not able to deal with such events. We were in turn forced to look at a bolus approach. We implemented a penalizing function and used this to find the optimal bolus size for a given CHO intake, e.g. we found the optimal bolus size of 4300 mU for a meal size of $100g$. We can use this fit to predict the size of the bolus a person needs for a given meal. This together with the PID should be able to maintain a stable glucose level and thus prevent numerous risks related to this.

6 References

References

[1] Dimitri Boiroux et al. "Model Identification using Continuous Glucose Monitoring Data for Type 1 Diabetes". In: *IFAC-PapersOnLine* 49.7 (2016). 11th IFAC Symposium on Dynamics and Control of Process SystemsIncluding Biosystems DYCOPS-CAB 2016, pp. 759–764. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2016.07.279. URL: http://www.sciencedirect.com/science/article/pii/S2405896316304864.

[2] WHO. *Fact-sheet regarding diabetes*. URL: https://www.who.int/news-room/fact-sheets/detail/diabetes. (accessed: 27.02.2020).

[3] Wikipedia contributors. *PID controller — Wikipedia, The Free Encyclopedia*. [Online; accessed 27-February-2020]. 2020. URL: https://en.wikipedia.org/w/index.php?title=PID_controller&oldid=942657155.

[4] Wikipedia contributors. *Type 1 diabetes — Wikipedia, The Free Encyclopedia*. [Online; accessed 27-February-2020]. 2020. URL: https://en.wikipedia.org/w/index.php?title=Type_1_diabetes&oldid=940227030.

Appendices

A APPENDIX 1

A.1 Appendix 1.1: Matlab implementations

```

%% Clear all
clear all

%% Set up ODE's (no PID, no eating)
% [Problem 1 & 2] %

% Inits %
parm = [49, 47, 20.1, 0.0106, 0.0081, 0.0022, 1.33, 253, 47];
x0 = [0,0,0,0,0,0];
t = linspace(0,24*60, 24*60);
us = 25.04;
d = 0;

% Solve ODE %
dxdt = @mvpfun;
% Simulate %
[T,X] = ode15s(dxdt,t,x0,[],us,d, parm);

%% Plot for ODE solution
% [Problem 1 & 2] %

```

```
% We want to plot in hours and previously we have units in minutes %
% It takes him approx 10 hours to reach ¬108 %
% This shows that we can simulate for all 24 hours %

figure
plot(X(:,4))
title('Glucose concentration')
xlabel('Time (min)')
ylabel('Concentration (mg/dl)')

%% Exercises with PID Implementation
% [Problem 4 5] %

% We want to change the insulin concentration every 5 min %
% Measure glucose conc -> compute u -> simulate ∆(t) ahead in time %

% PID parameters %
Kp = 0.01;
Ti = 1000;
Td = 1;

% Initializations %
x0 = [1.2458 1.2458 0.0101 200 0 0];
r = 108;
us = 25.04;
Ts = 5;
I = 0;

% Results init %
vecT = [0:5:24*60];
uu = zeros(length(vecT),1);
Xi = zeros(length(vecT),6);
Xi(1,:) = x0;
yprev = Xi(1,4);

% Test with meal %
d0 = zeros(length(vecT),1);
% 50 is the size of the meal we divided by 1000 because TA told us
d0(1) = 200/1000;

for i=1:(length(vecT)-1)
    % Set y % we add some noise, the PID hits lower than expected
    y = Xi(i,4) + abs(normrnd(0,10));

    % Calc PID
    [u,I] = PID(I, r, y, yprev, us, Kp, Ti, Td, Ts);
    if u<0
        u = 0;
    end
    uu(i) = u;

    % Solve ODE
    dxdt = @mvpfun;
    [T,X] = ode15s(dxdt, [vecT(i) vecT(i+1)], Xi(i,:),[], u, d0(i)/Ts, parm);

    % Assign previous y
    if (i > 1)
        yprev = Xi(i-1,4);
    else
        yprev = Xi(i,4);
    end

    % Save results
    Xi(i+1,:) = X(end,:);
end
```

```

% Plot concentrations
figure
plot(vecT, Xi(:,4))
title('Glucose concentration (d(1) = 200, d(t>1) = 0)')
xlabel('Time (min)')
ylabel('Concentration (mg/dl)')

% For this part we still get very high concentrations %

%% Used for PID plot
G = cell(3,1)
%% Used for PID plot
G{3} = Xi(:,4)

figure
plot(G{1})
hold on
plot(G{2})
hold on
plot(G{3})
title('Glucose concentration (for Ki = 1000, Kd = 1)')
xlabel('Time (min)')
ylabel('Concentration (mg/dl)')
legend('Kp = 0.01', 'Kp = 0.1', 'Kp = 0.2')

%% Minimize phi as a function of bolus
% [Problem 7] %

x0 = [1.2458 1.2458 0.0101 108.2115 0 0];
u0 = zeros(length(vecT),1);
d0 = zeros(length(vecT),1);
d0(1) = 200/1000;
phi = zeros(100,1);
bol_size = [1:1:100]*100;

% test phi function over bol sizes %
for i = bol_size/100
    u0(1) = i/10;
    phi(i) = phifun(x0, u0, d0, parm);
end

% we determine minimum and optimal bolus size %
phi_min = phi(find(phi == min(phi)));
optimal_bol = find(phi == min(phi))*100;

figure
semilogy(bol_size, phi)
xlabel('Bolus size')
ylabel('Phi value')
title('Glucose Penalty Function (GPF)')

%% Optimal bolus size for different meals (takes long time to run)
% [Problem 8 9] %

x0 = [1.2458 1.2458 0.0101 108.2115 0 0];
u0 = zeros(length(vecT),1);
d0 = zeros(length(vecT),1);
phi_min = [];
optimal_bol = [];
bol_size = [1:1:100]*100;
meal_size = [100:5:150]/1000;


% loop over meal sizes %
```

```
for j = meal_size*1000

d0(1) = j/1000;
phi = zeros(100,1);

    % test phi function %
    for i = bol_size/100
        u0(1) = i/10;
        phi(i) = phifun(x0, u0, d0, parm);
    end

% we determine minimum and optimal bolus size %
phi_min = [phi_min phi(find(phi == min(phi)))];
optimal_bol = [optimal_bol find(phi == min(phi))*100];

end

%% Interpolation

pval = polyfit(meal_size, optimal_bol, 4)

%% Function for PID
% [Problem 3] %
function [u,I_new] = PID(I,r,y,y_prev, us, Kp, Ti, Td, Ts)

e = y - r;

Pk = Kp*e;
Ki = (Kp*Ts)/Ti;
Kd = (Kp*Td)/Ts;
Dk = Kd*(y-y_prev);

% I_new is the integral we supply %
I_new = I+Ki*e;
u = us + Pk + I + Dk;

end

%% Function for MVP
% [Problem 1 2] %
function dxdt = mvpfun(t,x,u,d,parm)

dxdt = [u/(parm(1)*parm(3)) - x(1)/parm(1);
        (x(1)-x(2))/parm(2);
        -parm(4)*x(3)+parm(4)*parm(5)*x(2);
        -(parm(6)+x(3))*x(4)+parm(7)+(1000*x(6))/parm(8)*parm(9);
        d-x(5)/parm(9);
        (x(5)-x(6))/parm(9)];

end

%% Function for Rho
% [Problem 6] %

function rho = rhofun(G, G_bar, G_min, Kappa)

temp1 = G - G_bar;
temp2 = max(G_min - G,0);
rho = 0.5*temp1.^2 + 0.5*Kappa*temp2.^2;

end

%% Function for Integral
% [Problem 6] %

```

```
function Int = EulerInt(T, Y)

dT = T(2:end)-T(1:end-1);
Int = sum(Y(1:end-1).*dT);

end

%% Phi Function
% [Problem 6] %

function [phi, rho, G] = phifun(x0, u0, dmeal, parm)

% PID parameters %
Kp = 0.01;
Ti = 1000;
Td = 1;

% Initializations %
%x0 = [1.2458 1.2458 0.0101 200 0 0];
r = 108;
us = 25.04;
u = us;
Ts = 5;
I = 0;

% Results init %
vecT = [0:5:24*60];
uu = zeros(length(vecT),1);
Xi = zeros(length(vecT),6);
Xi(1,:) = x0;
yprev = Xi(1,4);

for i=1:(length(vecT)-1)
    % We only give meals on i = 1 in this project
    % For i = 1 we only give in bolus 'u0' and not PID 'u'
    if (i == 1)
        % Set y % we add some noise, the PID hits lower than expected
        y = Xi(i,4) + abs(normrnd(0,10));

        % Solve ODE
        dxdt = @mvpfun;
        [T,X] = ode15s(dxdt, [vecT(i) vecT(i+1)], Xi(i,:),[], u0(i)*1000, dmeal(i)/Ts, parm);

        % Assign previous y
        if (i > 1)
            yprev = Xi(i-1,4);
        else
            yprev = Xi(i,4);
        end

        % Save results
        Xi(i+1,:) = X(end,:);
    else
        % Set y % we add some noise, the PID hits lower than expected
        y = Xi(i,4) + abs(normrnd(0,10));

        % Calc PID
        [u,I] = PID(I, r, y, yprev, us, Kp, Ti, Td, Ts);
        if u<0
            u = 0;
        end
        uu(i) = u;

        % Solve ODE
        dxdt = @mvpfun;
```

```
        [T,X] = ode15s(dxdt, [vecT(i) vecT(i+1)], Xi(i,:),[], u, dmeal(i)/Ts, parm);

        % Assign previous y
        if (i > 1)
            yprev = Xi(i-1,4);
        else
            yprev = Xi(i,4);
        end

        % Save results
        Xi(i+1,:) = X(end,:);
    end
end

% Calculate penalty function %
G = Xi(:,4);
G_bar = 108;
G_min = 70;
Kappa = 10^6;

rho = rhofun(G, G_bar, G_min, Kappa);

% Integrate to find phi %

phi = EulerInt(vecT', rho); %vecT or T from ODE

end
```