BE3010 Final Project
Rohan Chhaya, Victoria Curry, Elena Grajales, Kalen Truong
12 December 2022

Request For Application to Monitor Blood Glucose Levels to Prevent Nocturnal Hypoglycemia

INTRODUCTION

The NIH has requested a device that keeps diabetic patients' blood glucose levels above hypoglycemic rates, which is typically around 0.7 g/L, while they sleep. Given that the CDC estimates that over 37 million Americans, or 1 in 10, have diabetes, and 96 million Americans, or 1 in 3, have prediabetes, it is of utmost importance to develop a device that can help diabetic patients maintain homeostasis. Furthermore, there is immense need for monitoring and regulation during sleep specifically given that over half of severe episodes of hypoglycemia occur during sleep.

The device ought to be stable, since an unstable system could result in an unbounded increase in glucose provided and significantly hurt patients. It should also help patients maintain a proper, non-hypoglycemic blood glucose levels and reach a given concentration (for example, 2 g/L). The device should also be self-adaptable to a variety of patients, since diabetic patients often have different backgrounds and lifestyles that may affect the risk of hypoglycemia and vary the initial conditions (ex. alcohol consumption, exercising, skipping meals, etc).

Throughout the development of this system, all group members worked together directly to determine relationships, plan and troubleshoot code, plan approaches, address limitations, research options, and complete algebra/calculus. Elena and Kalen directly typed the majority of the planned code as discussions occurred. Rohan and Victoria used the pictures generated from the code output to create the report itself. Kalen was responsible for proofreading and source citation generations.

ANALYSIS

$$H(s) = Y(S)/Q(S) = (s + \boldsymbol{\alpha})/(s^2 + \boldsymbol{\alpha}s + \boldsymbol{\eta}s + \boldsymbol{\eta\alpha} + \boldsymbol{\gamma\beta}) \quad (1)$$

Equation 1 is the transfer function for a sleeping-state–and thus assumed zero-state–patient with the input of glucose injection, q(t) (g / L*hr), and output of blood glucose level, y(t) (g/L). This was determined through the given differential equations and hard coded with symbolic variables into MATLAB in line 20 (Supplementary Materials).

For part b, a ‘for’ loop was used to index through the number of patients, replacing the symbolic variables with the corresponding patient values, which were given and pre-established in code lines 11-14. Each consequent equation was then plotted on subplots with axis values set to encompass the major transfer function peaks.

These subplots were generated using a continuous-time transfer function model from MATLAB. The ‘pzplot’ and ‘pzoptions’ from code lines 45 and 52 are part of MATLAB’s tf

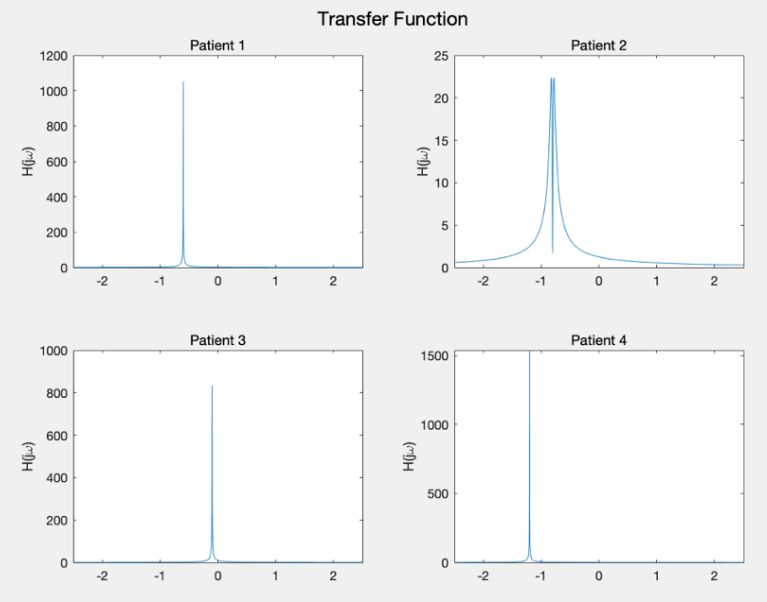

Figure 1: Part b. Transfer function plots for all patients.

stability analysis function collection, and they coded the finding and plotting of the zeros, poles, and ROC for equation 1.8:

This situation from figure 3 continues to have the patients sleeping, which means we were able to use zero-state assumptions when solving. For development of the plots, we relayed the given differential equations to the code, took their Laplace Transforms, and created preset symbolic variables for alpha, beta, gamma, and eta in lines 68-78. Following this, we substituted in the actual given patient values for the symbolic variables using the subs function in MATLAB. We were then able to create subplots of y(t), the levels of glucose in the patients' blood, over 8 hours.

Given the basic differential equations that model the insulin and glucose level in the body, as well as pumped changes to these levels, our team used Laplacian techniques to find the transfer function of the system and ultimately model blood glucose using MATLAB. After spending time researching ways to create the desired control system, we learned about and decided to use a modified version of the proportional-integral-derivative (PID) controller. PID controllers calculate a "delta" between the desired value and the measured value, and then use proportional, derivative, and integral terms to drive the system toward that desired value. Tuning these terms allows the controller to respond in a wide range of systems. For this application, the integral term was dropped by weighing it to zero, and a "PD2" controller with only

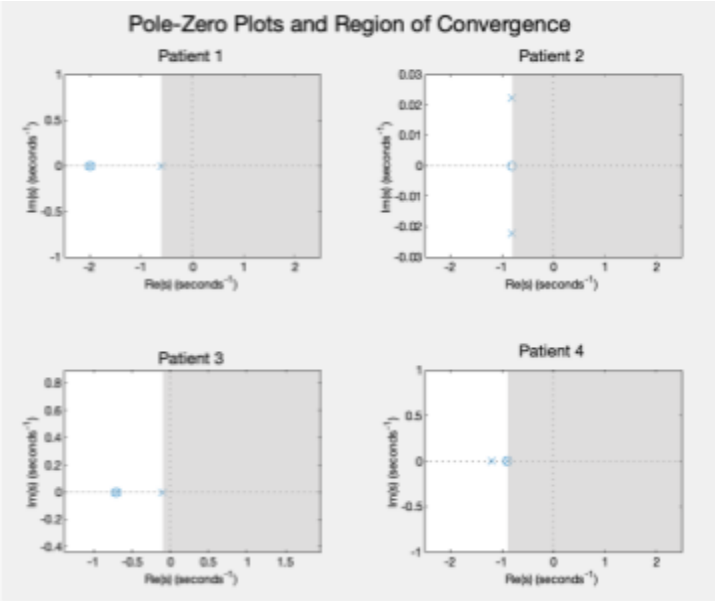

Figure 2: Part c. Plots of the transfer functions for the four patients in terms of poles, zeroes, and ROCs.

proportionality and derivative constants (Kp and Kd, respectively) was used. This was due to the need only for proportional amplification and the approach of a steady-state limit. Kp gives the proportional, constant gain needed to increase glucose concentration to above hypoglycemic levels, and Kd eliminates the oscillation about the desired glucose concentration observed in MATLAB plots for higher values of Kp by acting as a dampener on overall system change. This elimination of oscillation allowed for system stabilization approximating the step function, with a horizontal asymptote at the goal glucose concentration.

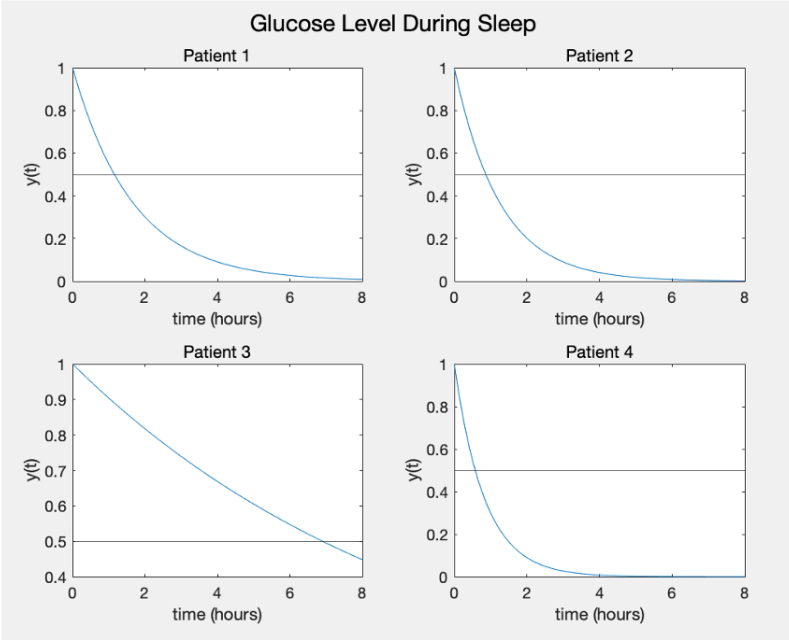

Figure 3: Part d. Glucose levels during sleep for all four patients.

For implementation of the PD controller and completion of parts e and f specifically, it was essential to determine set inputs and outputs. After some struggles with overcomplicating the system, we realized that we could set our feedback system with input of our goal glucose level, which we set to be over .7 g/L at 2 g/L because is was within the .5 to 3.5 g/L range listed as possible initial glucose levels and was well above the level for hypoglycemia. The system then had a summation with the goal glucose level as the positive input and current glucose level as the negative input. The absolute value of this subtraction was taken and set as the time dependent error level, or the amount of glucose that would need to be injected. This was inputted to the PD system and the system output considered q(t). In Laplace space,

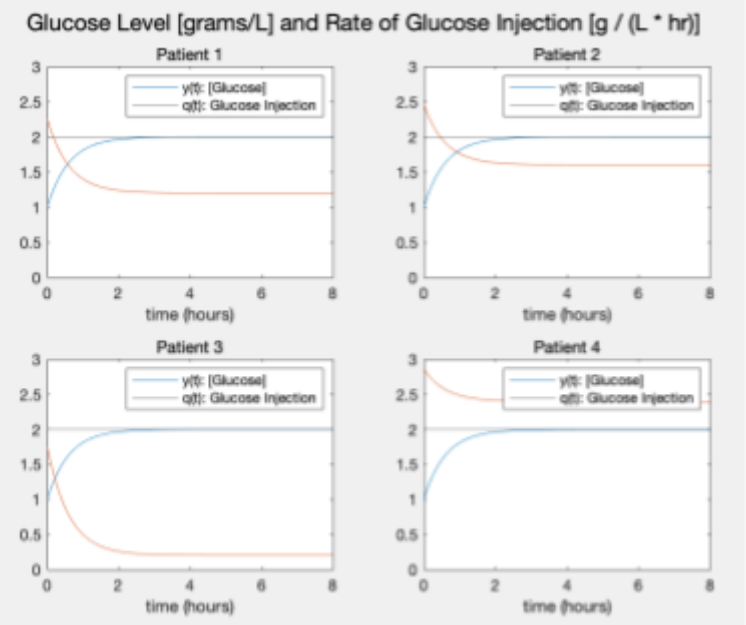

Figure 5: Part e. Patient's glucose levels and glucose pump outputs over 8 hours of sleeping time,with the implementation of the control system to mandate patient glucose levels between .5 and 3.5 g/L.

goal Y(S) was the input, and the absolute value of Yg(S) (goal value) minus Y(S) was multiplied by the Laplace Transform of the PD constants equation to give an overall output equation that could be solved for Q(S) and Y(S). These were both then Inverse Laplace Transformed and plotted with patient variables inputted. In the code, PD constant values (selected through plotting), the variable for the initial glucose level, and the goal glucose level were established in lines 116-121. Following this, a plot over 8 hours was initiated, and the determined Y(S) and Q(S) were hard coded. In lines 131 and 132, their inverse Laplace Transforms were taken and code to plot these MATLAB-determined expressions follows.

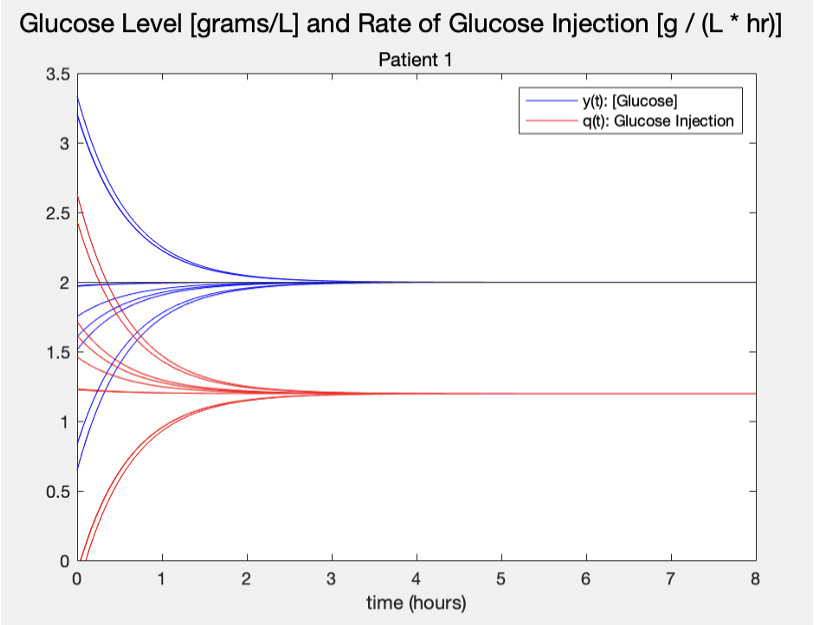

Figure 6: Part f. Patient 1's blood glucose levels during sleep with the control system applied, given a random initial glucose level between .5 and 3.5 g/L.

Part f differs from part e in that it adds a random initial glucose value. To approach this, we ran multiple simulations which were used to create 10 different initial glucose concentrations between .5 and 3.5 g/L (code lines 151-153). Following this adjustment, the code approach remained the same as it was in part e, though 10 trials were plotted with variables pre-set for only patient 1, rather than singular trials of multiple patients. This lack of necessary changes indicated that our controller was already able to adapt to varying initial glucose levels with high levels of success in achieving eventual goal levels of glucose. Though, this could not have been said during our first iterations of system development. We struggled a bit with how to set this up, and we ultimately removed our assumption of zero-state. Thus, we made the system set to vary in part e and just entered the known values as patients were set to be sleeping through that portion. Though we initially solved through all of the algebra considering only the sleep/zero state assumptions, changing this and re-solving helped in the long run due to the new system's being usable in part e and useful for f and g.

While we got a valid control system that keeps patients alive, it wasn't all smooth sailing. Given only 2 differential equations and the desire to isolate x(t), y(t), p(t), and q(t) meant falling into loops where we ended up plugging an equation into itself and getting the same output. This is when we had to take a step back, draw out the system to fully understand the mechanisms, and

figure out which terms are true inputs and which are not. Overall, though, the application of controllers and feedback for biomonitoring was really interesting, especially given how complex a glucose-insulin relationship can be. The concepts we discussed in class, like drawing feedback systems, solving differential equations, completing Laplace and Inverse Laplace Transforms, and using various transfer functions based on the control mechanism proved to be essential.

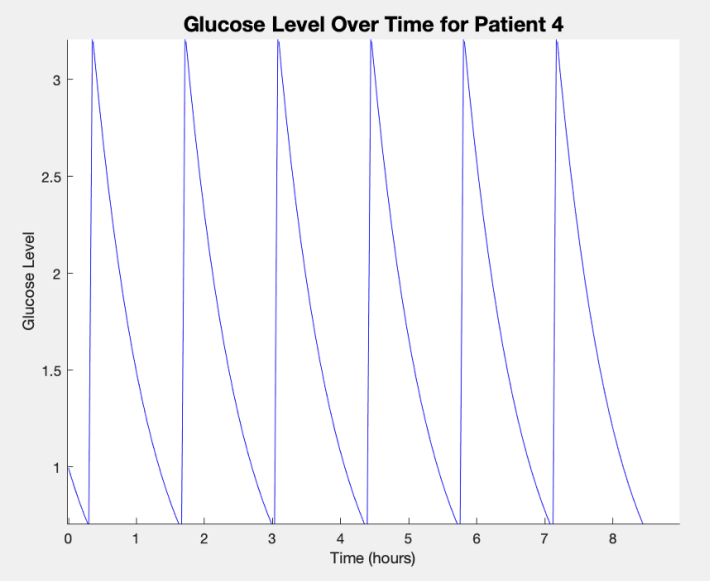

Figure 7: Part h. Patients 4 blood glucose level during sleep using a simplified glucose pump

In part H, we adapt our model to a less responsive glucose pump that either be ON (at 50 gL/h) or OFF (0 gL/h). Ultimately, the crude nature of this pump means it will have to be turned ON if glucose drops too low and OFF if glucose runs too high, leading to oscillations in a set window. This window *is* within the acceptable gluose range for patients, meaning our system will not result in hypoglycemia. However, the glucose level does not settle into a single value and changes over the course of the night, which may cause issues depending on sleep time, whether or not they wake up in the night, and other medical factors. In some patients, Ceriello et al (2008) explains how oscillatory glucose can be more damaging in Type II diabetics than a constant hyperglycemic/hypoglycemic level, meaning while our system fulfills the criteria of keeping pateints within the range, the use of this less expensive pump may not be ideal in the field

In part I, p(t) changes from p(t) = 0 to p(t) = 2.5. This alters our Y(S) and corresponding y(t) equations we found and used in parts f and g. The Laplace transform gives us P(S) = 2.5 / s, which can be fed into our Y(S). The resultant system does reach stability and the patient glucose level settles at the desired amount after ~2 hours, indicating the controller has the ability to satisfy these conditions . The glucose injection required is higher than in previous analyses, but this makes sense since insulin injection rate (p(t)) is nonzero and requires more glucose to "counteract."

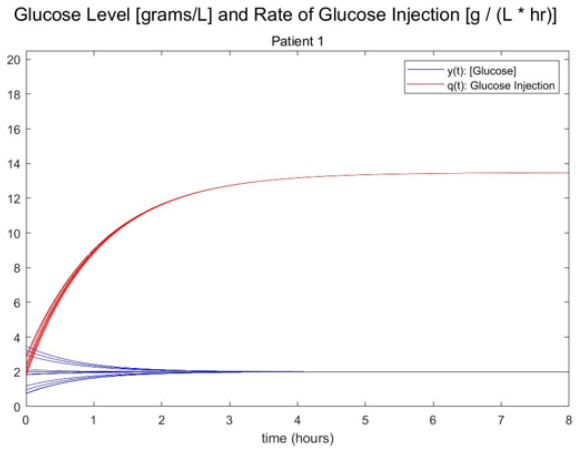

Figure 8: Part j. Visualizing patient 1's glucose levels during sleep with the control system and nonzero p(t) applied with varying initial glucose concentrations from 0.5 to 3.5

CONCLUSION

The creation of a control system to monitor glucose levels proved to be a more difficult task than initially expected and required the use of Laplace transforms, algebraic analysis, and PID control systems. By applying transformations to time-based equations into the complex frequency domain, differential equations that model human bodily systems can be controlled in a way that is stable, reaches a desired output in a variety of situations (ex. with and without an insulin pump), and allows for a variety of patients to be assisted with this device. This plays into a larger theme where complex mathematical processes can be applied for signal and system analysis in the context of biological signals and systems, including blood sugar, neurological systems, cardiac systems, and much more.

Future work may involve modifying the system to allow for the inexpensive pump in part h to create stable conditions, work more directly with insulin concentrations, and find the limits of patient parameters that the system applies to in order to prevent health issues down the line.

Works Cited

Centers for Disease Control and Prevention. (2022, January 24). *The facts, stats, and impacts of diabetes*. Centers for Disease Control and Prevention. Retrieved December 12, 2022, from https://www.cdc.gov/diabetes/library/spotlights/diabetes-facts-stats.html#:~:text=37.3%20million%20Americans%E2%80%94about%201,t%20know%20they%20have%20it.

Ceriello, A., Esposito, K., Piconi, L., Ihnat, M. A., Thorpe, J. E., Testa, R., Boemi, M., & Giugliano, D. (2008). Oscillating glucose is more deleterious to endothelial function and oxidative stress than mean glucose in normal and type 2 diabetic patients. *Diabetes*, *57*(5), 1349–1354. https://doi.org/10.2337/db08-0063

Johns Hopkins University. (2021, August 8). *Hypoglycemia: Nocturnal*. Hypoglycemia: Nocturnal | Johns Hopkins Medicine. Retrieved December 12, 2022, from https://www.hopkinsmedicine.org/health/conditions-and-diseases/diabetes/hypoglycemia-nocturnal#:~:text=When%20blood%20glucose%20levels%20fall,occur%20at%20night%20during%20sleep.

Pacheco, D. (2022, April 1). *Sleep & glucose: How blood sugar can affect rest*. Sleep Foundation. Retrieved December 12, 2022, from https://www.sleepfoundation.org/physical-health/sleep-and-blood-glucose-levels

The PID Controller & Theory explained. National Instruments. (n.d.). Retrieved December 12, 2022, from https://www.ni.com/en-us/innovations/white-papers/06/pid-theory-explained.html

Supplementary Material: MATLAB Code

```
%Final Project.m

clear; clc; close all;

%Part A

%All patients

numPatients = 4; %number of patients

a = [2 0.8 0.7 0.9]; %Alpha
b = [0.00 0.01 0.00 0.00]; %Beta
g = [0.10 0.05 0.15 0.4]; %Gamma
e = [0.6 0.8 0.1 1.2]; %Eta

syms p(t) x(t) y(t) q(t) s alpha beta gamma eta %initiate symbolic
variables

assume([p(t) x(t) y(t) q(t) t] > 0) %assume all values are positive

h_LT = (s + alpha)/(s^2 + alpha*s + eta*s + eta*alpha + gamma*beta);
%H(s) derivation

%% Part B
for i = 1:numPatients

    w = linspace(-2.5,2.5,2000); %relavent frequency ranges

    h_s = (s + a(i))/(s^2 + a(i)*s + e(i)*s + e(i)*a(i) +
g(i)*b(i));%H(s) with patient constants added

    %create plots
    subplot(2,2,i);
    plot(w,abs(subs(h_s,s,w)));
    xlim([-2.5 2.5]);
    sgtitle('Transfer Function');
    subtitle(sprintf('Patient %d', i));
    ylabel('H(j\omega)');

end

%% Part C
for i = 1:numPatients

```

```
42.       sys = tf([1 a(i)], [1 (a(i)+e(i)) (e(i)*a(i) + g(i)*b(i))]);
      %transfer function representation
43.
44.       %pole-zero plots
45.       opt = pzoptions;
46.       opt.Title.String = (sprintf('Patient %d', i));
47.       opt.Title.FontSize = 11;
48.       opt.XLabel.String = 'Re(s)';
49.       opt.YLabel.String = 'Im(s)';
50.       opt.Xlim = [-2.5 2.5];
51.       sgtitle('Pole-Zero Plots and Region of Convergence');
52.       subplot(2,2,i);
53.       graph = pzplot(sys, opt);
54.
55.   end
56.
57.   %% Part D
58.
59.   %Initial conditions:
60.   pval = 0;
61.   xval = 0.01; %U/L
62.   yval = 1; %g/L
63.
64.   %Part d.
65.   %Plot y(t) from t = 0, 8 grams/L
66.   %8 Hours, (t = 8)
67.
68.   %Declare differential equations.
69.   dx = diff(x, t);
70.   dy = diff(y, t);
71.
72.   eqn1 = dx == p - alpha*x + beta*y;
73.   eqn2 = dy == q - gamma*x - eta*y;
74.
75.   eqn1LT = laplace(eqn1, t, s);
76.   eqn2LT = laplace(eqn2, t, s);
77.
78.   syms x_LT p_LT q_LT y_LT h_LT
79.
80.
81.   eqn1LT = subs(eqn1LT, [laplace(x, t, s), laplace(p,t,s), laplace(y,
      t, s), x(0)], ...
82.       [x_LT pval y_LT xval]);
83.
84.   eqn2LT = subs(eqn2LT, [laplace(y, t, s), laplace(q,t,s), laplace(x,
      t, s), y(0) ], ...
85.       [y_LT 0 x_LT yval]);
```

```
 86.
 87.   eqns = [eqn1LT eqn2LT];
 88.   vars = [x_LT y_LT];
 89.   [x_LT, y_LT] = solve(eqns, vars);
 90.
 91.   yt = ilaplace(y_LT);
 92.   ytsol = simplify(yt);
 93.
 94.   numPatients = 4;
 95.
 96.   vars = [alpha beta gamma eta];
 97.   time = linspace(0, 8, 1000);
 98.
 99.   for i = 1:numPatients
100.       values = [a(i) b(i) g(i) e(i)];
101.       y = subs(ytsol,vars,values);
102.       subplot(2,2,i);
103.       graph = plot(time, subs(y, t, time));
104.       yline(.5);
105.       ylabel('y(t)')
106.       xlabel('time (hours)')
107.       sgtitle('Glucose Level During Sleep');
108.       subtitle(sprintf('Patient %d', i));
109.
110.   end
111.
112.   %% Part E
113.
114.   numPatients = 4; %number of patients
115.
116.   ygoal = 2; %goal glucose level
117.   Kp = 500; %proportionality constant
118.   Ki = 0; %integral constant
119.   Kd = 300; %derivative constant
120.   Gc = pid(Kp, Ki, Kd); %initiate pid controller
121.   y_i = 1; %initial y value
122.
123.   for i = 1:numPatients
124.
125.       %create plots
126.       w = linspace(0, 8 ,2000); %relevant frequency ranges
127.       time = linspace(0, 8, 1000); %time range
128.
129.       y_s =  ((Kp*ygoal)/s + Kd*y_i - ((g(i)*0.01)/(s + a(i))) +
      y_i)... / (e(i) + s + Kp + s*Kd + (g(i) * b(i))/(s + a(i)));
      %derived Y(s)
130.       q_s = (Kp*ygoal)/s - y_s*(Kp + s*Kd) + Kd*y_i;%derived Q(s)
```

```
131.        q_t = ilaplace(q_s); %inverse laplace Q(s)
132.        y_t = ilaplace(y_s); %inverse laplace Y(s)
133.
134.        %plot
135.        subplot(2,2,i);
136.        graph = plot(time, subs(y_t, t, time));
137.        yline(ygoal);
138.        ylim([0 3.0]);
139.        hold on;
140.        plot(time, subs(q_t, t, time));
141.        hold off;
142.        xlabel('time (hours)')
143.        sgtitle('Glucose Level [grams/L] and Rate of Glucose Injection
        [g / (L * hr)]');
144.        subtitle(sprintf('Patient %d', i));
145.        legend("y(t): [Glucose] ", "q(t): Glucose Injection")
146.    end
147.
148.
149.    %% Part F
150.
151.    numSimulations = 10;
152.
153.    y_i = 0.5 + (3.5-0.5) .* rand(numSimulations,1);
154.
155.    ygoal = 2; %goal glucose level
156.    Kp = 500; %proportionality constant
157.    Ki = 0; %integral constant
158.    Kd = 300; %derivative constant
159.    Gc = pid(Kp, Ki, Kd); %initiate pid controller
160.
161.    for i = 1:numSimulations
162.
163.        w = linspace(0, 8 ,2000); %relavent frequency ranges
164.        time = linspace(0, 8, 1000); %time range
165.
166.        y_s =  ((Kp*ygoal)/s + Kd*y_i(i) - ((g(1)*0.01)/(s + a(1))) +
        y_i(i))...
167.            / (e(1) + s + Kp + s*Kd + (g(1) * b(1))/(s + a(1)));
        %derived Y(s)
168.        q_s = (Kp*ygoal)/s - y_s*(Kp + s*Kd) + Kd*y_i;%derived Q(s)
169.        q_t = ilaplace(q_s); %inverse laplace Q(s)
170.        y_t = ilaplace(y_s); %inverse laplace Y(s)
171.
172.        %plot
173.        subplot(1,1,1);
174.        plot(time, subs(y_t, t, time),'b-');
```

```
175.        hold on;
176.        plot(time, subs(q_t, t, time),'r-');
177.        yline(ygoal);
178.        ylim([0 3.5]);
179.        xlabel('time (hours)')
180.        sgtitle('Glucose Level [grams/L] and Rate of Glucose Injection
       [g / (L * hr)]');
181.        subtitle(sprintf('Patient 1'));
182.        legend("y(t): [Glucose] ", "q(t): Glucose Injection")
183.
184.    end
185.
186.    hold off;
187.    %% Part G
188.
189.    % Demo: Input values for alpha, beta, gamma, eta, and y_i
190.
191.    % Inputs
192.    y_i = 3;
193.    a = 1.0;
194.    b = 1.0;
195.    g = 1.0;
196.    e = 1.0;
197.
198.    ygoal = 2; %goal glucose level
199.    Kp = 500; %proportionality constant
200.    Ki = 0; %integral constant
201.    Kd = 300; %derivative constant
202.    Gc = pid(Kp, Ki, Kd); %initiate pid controller
203.
204.
205.    w = linspace(0, 8 ,2000); %relavent frequency ranges
206.    time = linspace(0, 8, 1000); %time range
207.
208.    y_s =  ((Kp*ygoal)/s + Kd*y_i - ((g*0.01)/(s + a)) + y_i)...
209.        / (e + s + Kp + s*Kd + (g * b)/(s + a)); %derived Y(s)
210.    q_s = (Kp*ygoal)/s - y_s*(Kp + s*Kd) + Kd*y_i;%derived Q(s)
211.    q_t = ilaplace(q_s); %inverse laplace Q(s)
212.    y_t = ilaplace(y_s); %inverse laplace Y(s)
213.
214.    %plot
215.    subplot(1,1,1);
216.    plot(time, subs(y_t, t, time),'b-');
217.    hold on;
218.    plot(time, subs(q_t, t, time),'r-');
219.    yline(ygoal);
220.    ylim([0 3.5]);
```

```
221.  xlabel('time (hours)')
222.  sgtitle('Glucose Level [grams/L] and Rate of Glucose Injection [g /
      (L * hr)]');
223.  subtitle(sprintf('Patient 1'));
224.  legend("y(t): [Glucose] ", "q(t): Glucose Injection")
225.
226.  %% Part H
227.  syms s t
228.  a = [2 0.8 0.7 0.9]; %Alpha
229.  b = [0.00 0.01 0.00 0.00]; %Beta
230.  g = [0.10 0.05 0.15 0.4]; %Gamma
231.  e = [0.6 0.8 0.1 1.2]; %Eta
232.  % Initial x_i values
233.  x_i1 = 0.3;
234.  x_i2 = x_i1 + 0.064; %0.064 = time to oscillate in the given window
235.  x_i4 = x_i3 + 0.064; %even indexing because oscillations = every other
236.  x_i6 = x_i5 + 0.064;
237.  x_i8 = x_i7 + 0.064;
238.  x_i10 = x_i9 + 0.064;
239.  x_i12 = x_i11 + 0.064;
240.  % laplace and plot
241.  for i = 4
242.     y_s = (1 - (0.01*g(i))) * ((s + a(i))/((g(i)*b(i))+
      ((s+e(i))*(s+a(i)))));
243.     y_t = ilaplace(y_s,s,t);
244.
245.     y_s2 = (3.2 - (0.01*g(i))) * ((s + a(i))/((g(i)*b(i))+
      ((s+e(i))*(s+a(i)))));
246.     y_t2 = ilaplace(y_s2,s,(t-x_i2));
247.
248.     y_s3 = (3.2 - (0.01*g(i))) * (s + a(i))/((g(i)*b(i))+
      ((s+e(i))*(s+a(i))));
249.     y_t3 = ilaplace(y_s3,s,(t-x_i4));
250.
251.     y_s4 = (3.2 - (0.01*g(i))) * ((s + a(i))/((g(i)*b(i))+
      ((s+e(i))*(s+a(i)))));
252.     y_t4 = ilaplace(y_s4,s,(t-x_i6));
253.
254.     y_s5 = (3.2 - (0.01*g(i))) * ((s + a(i))/((g(i)*b(i))+
      ((s+e(i))*(s+a(i)))));
255.     y_t5 = ilaplace(y_s5,s,(t-x_i8));
256.
257.     y_s6 = (3.2 - (0.01*g(i))) * ((s + a(i))/((g(i)*b(i))+
      ((s+e(i))*(s+a(i)))));
258.     y_t6 = ilaplace(y_s6,s,(t-x_i10));
259.
```

```
260.      y_s7 = (3.2 - (0.01*g(i))) * ((s + a(i))/((g(i)*b(i))+
      ((s+e(i))*(s+a(i)))));
261.      y_t7 = ilaplace(y_s7,s,(t-x_i12));
262.      subplot(2,2,i);
263.      figure()
264.      hold on
265.      fplot(t,y_t,[0,x_i1],'b-')
266.      fplot(t,50*(t-x_i1) + 0.7,[x_i1, x_i2],'b-');
267.      fplot(t,y_t2,[x_i2, x_i3],'b-');
268.      fplot(t,50*(t-x_i3) + 0.7,[x_i3, x_i4],'b-');
269.      fplot(t,y_t3,[x_i4, x_i5],'b-');
270.      fplot(t,50*(t-x_i5) + 0.7,[x_i5, x_i6],'b-');
271.      fplot(t,y_t4,[x_i6, x_i7],'b-');
272.      fplot(t,50*(t-x_i7) + 0.7,[x_i7, x_i8],'b-');
273.      fplot(t,y_t5,[x_i8, x_i8+1.3],'b-');
274.      fplot(t,50*(t-x_i9) + 0.7,[x_i9, x_i10],'b-');
275.      fplot(t,y_t6,[x_i10, x_i10+1.3],'b-');
276.      fplot(t,50*(t-x_i11) + 0.7,[x_i11, x_i12],'b-');
277.      fplot(t,y_t7,[x_i12, x_i12+1.3],'b-');
278.      ylim([0.7 3.2]);
279.      title('Glucose Level Over Time for Patient 4','FontSize',15);
280.      xlabel('Time (hours)');
281.      ylabel('Glucose Level')
282.      hold off
283.   end
284.
285.   %% PART I
286.   hold off;
287.
288.   numSimulations = 10;
289.   y_i = 0.5 + (3.5-0.5) .* rand(numSimulations,1);
290.   ygoal = 2; %goal glucose level
291.   Kp = 1000; %proportionality constant
292.   Ki = 0; %integral constant
293.   Kd = 800; %derivative constant
294.   Gc = pid(Kp, Ki, Kd); %initiate pid controller
295.   for i = 1:numSimulations
296.      w = linspace(0, 8 ,2000); %relavent frequency ranges
297.      time = linspace(0, 8, 1000); %time range
298.
299.      y_s_p =  ((Kp*ygoal)/s + Kd*y_i - ((g(i) * (2.5/s) + g(i)*0.01)/(s +
      a(i))) + y_i) / (e(i) + s + Kp + s*Kd + (g(i) * b(i))/(s + a(i)));
      %derived Y(s) with a valid P(S)
300.      q_s_p = (Kp*ygoal)/s - y_s_p*(Kp + s*Kd) + Kd*y_i;%derived Q(s)
301.      q_t_p = ilaplace(q_s); %inverse laplace Q(s)
302.      y_t_p = ilaplace(y_s_p); %inverse laplace Y(s)
303.      %plot
```

```
304.      subplot(1,1,1);
305.      plot(time, subs(y_t_p, t, time),'b-');
306.      hold on;
307.      plot(time, subs(q_t_p, t, time),'r-');
308.      yline(ygoal);
309.      ylim([0 20.5]);
310.      xlabel('time (hours)')
311.      sgtitle('Glucose Level [grams/L] and Rate of Glucose Injection [g / (L
       * hr)]');
312.      subtitle(sprintf('Patient 1'));
313.      legend("y(t): [Glucose] ", "q(t): Glucose Injection")
314.   end
315.   hold off;
316.
```