BIEN 3200: Lab 6

Laboratory Exercise #7

Peter Dobbs
Lab Section 401
Nicholas Heugel

6 December 2016

Abstract:

In order to improve the efficient monitoring of diabetic patients, numerical methods must be explored. Through this laboratory exercise, the fundamentals of numerical methods used for classification of diabetic patients and their treatment are introduced. The glucose/insulin model of normal patients and type I and type II diabetics is used for understanding. To solve the differential equations that represent the glucose and insulin systems, the Runge-Kutta method is employed. Through adjustments of the various parameters for the model, type I and type II diabetes can be simulated and treated.

Introduction:

Algorithms currently exist that satisfy the needs of diabetics, who require glucose monitoring and possibly insulin infusion or injection. Mathematical modeling and numerical methods can be used to simulate the kinetics of blood glucose/insulin metabolism under physiologic and pathophysiologic conditions. Using these new models, care can be personalized to patients on an individual basis, saving expense and concern.

Methods:

Equations:

For $G \leq G_0$

$$\frac{dG}{dt} = u_1 - a_1 GS + a_2(G_0 - G) - G_b$$

$$\frac{dS}{dt} = u_2 - a_4 S$$

For $G > G_0$

$$\frac{dG}{dt} = u_1 - a_1 GS - G_b$$

$$\frac{dS}{dt} = u_2 - a_4 S + a_3(G - G_0)$$

Initial Conditions

$$G(0) = 85\ mg/dL$$

$$S(0) = 0\ mg/dL$$

Where G is the blood glucose concentration (mg/dL), S is the blood insulin concentration, u_1 is the input rate of exogenous glucose (mg/(dL.hr)), u_2 is the input rate of exogenous insulin (mg/(dL.hr)), G_0 is the reference blood glucose concentration (mg/dL), and G_b is the rate of insulin-independent glucose utilization(mg/(dL.hr)). The various a's are rate constants for endogenous glucose or insulin production or utilization. These equations are both linear and coupled.

The Runge-Kutta method was implemented in order to solve this system of Ordinary Differential Equations (ODE). It was selected because the formulas are equivalent in accuracy to higher order Taylor series and do not involve higher order derivatives. For a given time interval ($t_i \leq t \leq t_{i+1}$), the Runge-Kutta methods use a weighted average of values of slopes taken at

different points within an interval ($t_i \leq t \leq t_i + h$). For a given step size, s, in time, the error of the fourth order Runge-Kutta method is of order s^4. This means that reducing the step size reduces the error significantly greater than that for the Euler method.

Using C programming, a function can be made that loops through 25 hours of time, with a step size of 0.005 hours, to solve the ODEs. Start by calculating the values of u_1 and u_2 based on the equations in the lab handout. Begin the looping process by determining the four slopes at a given time instance based on the equations listed above. With the resulting eight slopes, since four slopes per ODE, solve the ODE at that given time point based on the equation: $\hat{C}(t) = C_0 + \frac{1}{6}(k_{i1} + 2k_{i2} + 2k_{i3} + k_{i4})$. Then continue the loop through all given time values.

To simulate type I diabetes, the parameters have to be altered for very low blood insulin concentration and very high, and increasing, levels of blood glucose concentration. The conditions require a_1, a_2, a_3, a_4, and G_0 to equal 0.01, 10.0, 0.1, 64.0, and 85 respectively. In order to properly treat the patient, one would need to apply a value of roughly 500 to the parameter A.

Describe how a rate of exogenous insulin input was determined to simulate treatment of type I diabetes

In order to simulate type II diabetes, the parameters had to be altered to create conditions where blood insulin concentration and blood glucose concentration both are abnormally high. To achieve these conditions, the values for a_1, G_0, and G_b must all be reduced to 0.025, 85, and 0 respectively. To treat that condition, a B value must be reduced from 205 to 120.

Results:

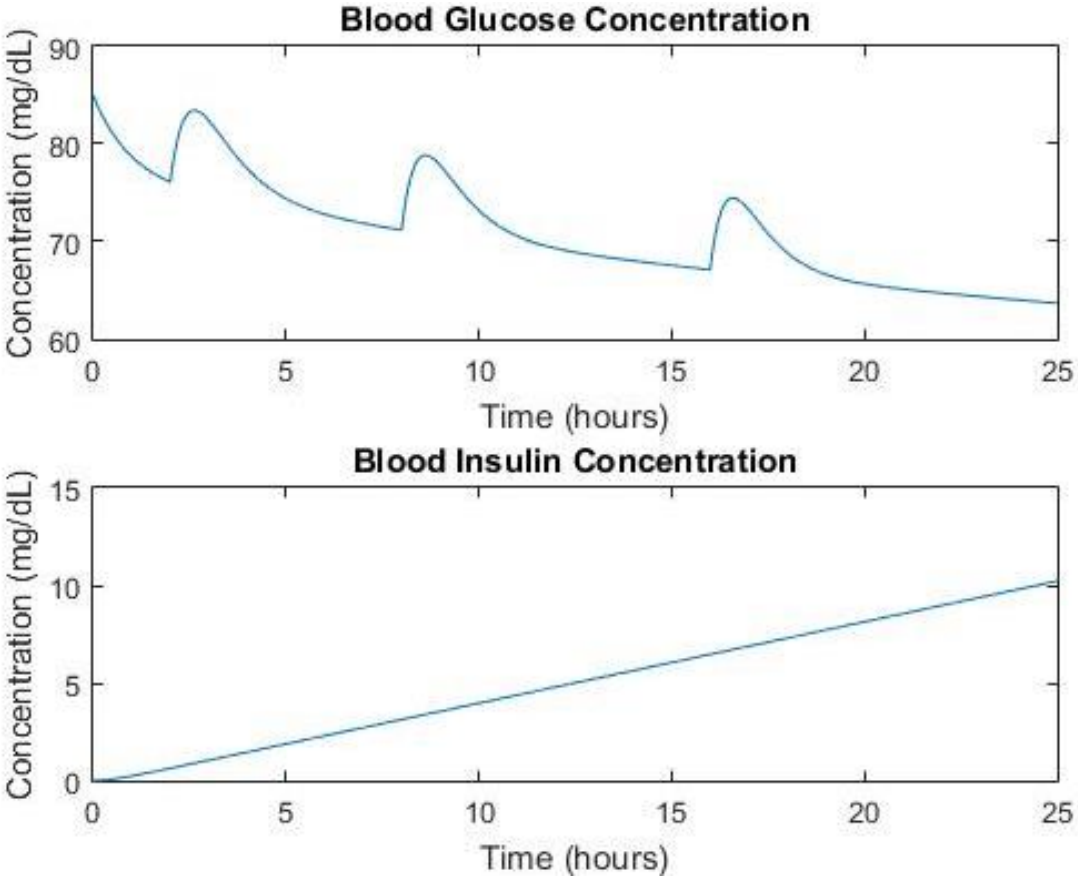


Figure 1: Plot of the concentration of glucose and insulin in the bloodstream of a healthy patient

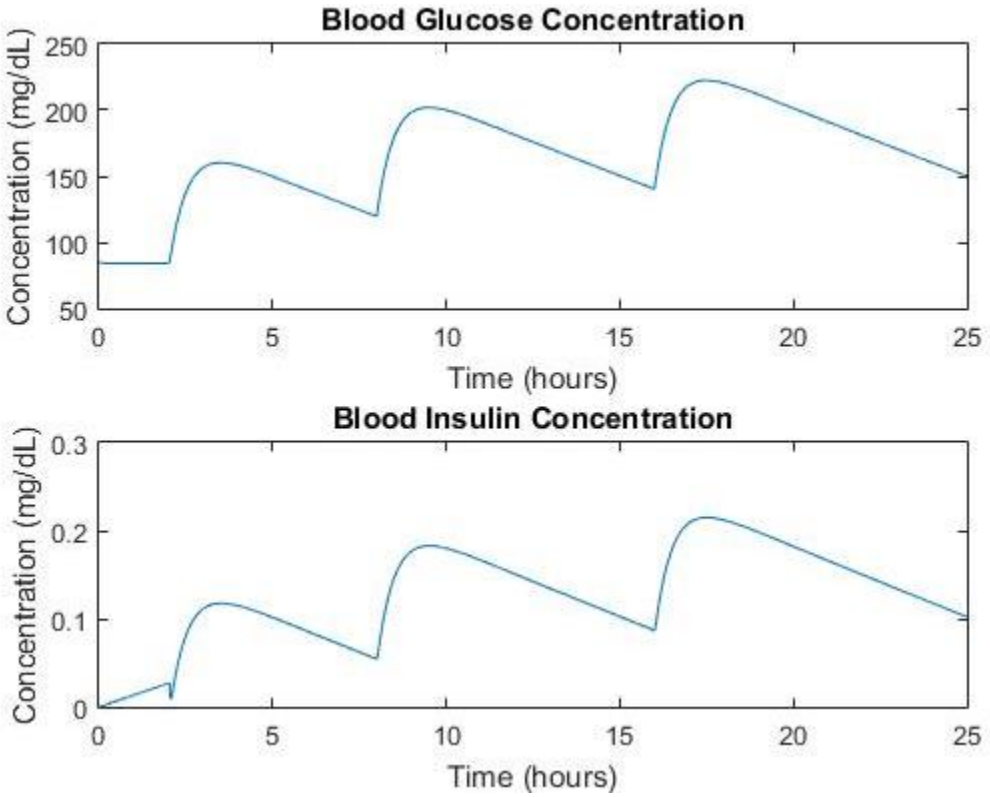


Figure 2: Plot of the concentration of glucose and insulin in the bloodstream of an untreated type I diabetic patient

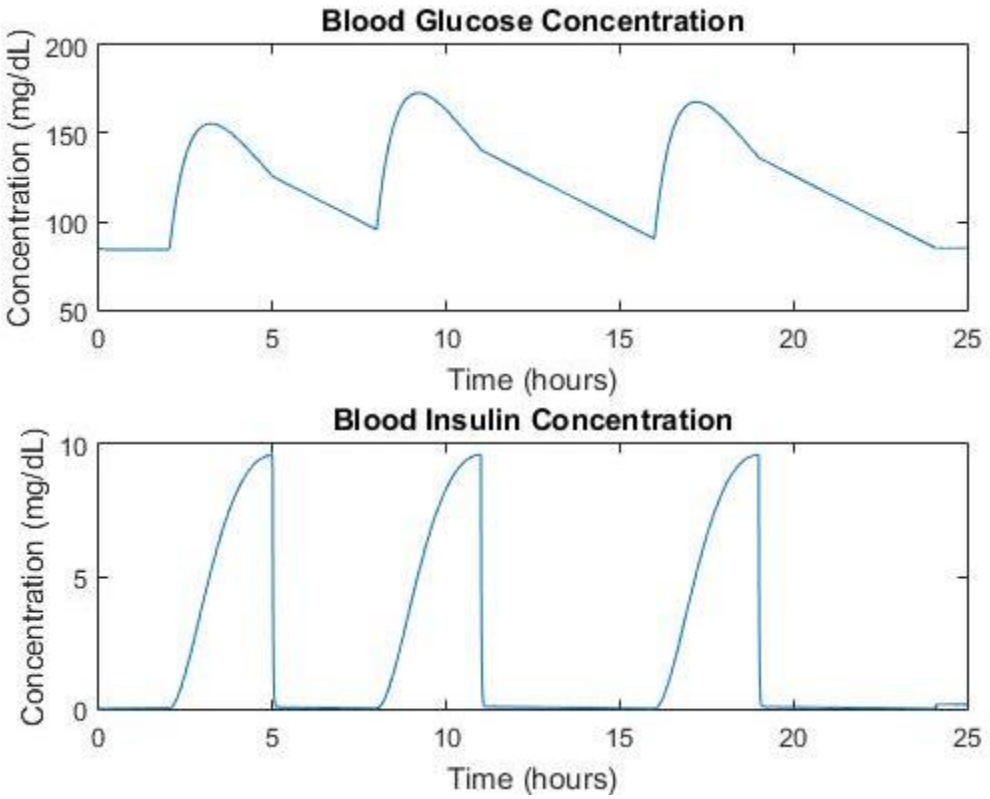


Figure 3: Plot of the concentration of glucose and insulin in the bloodstream of a treated type I diabetic patient. *A = 500*

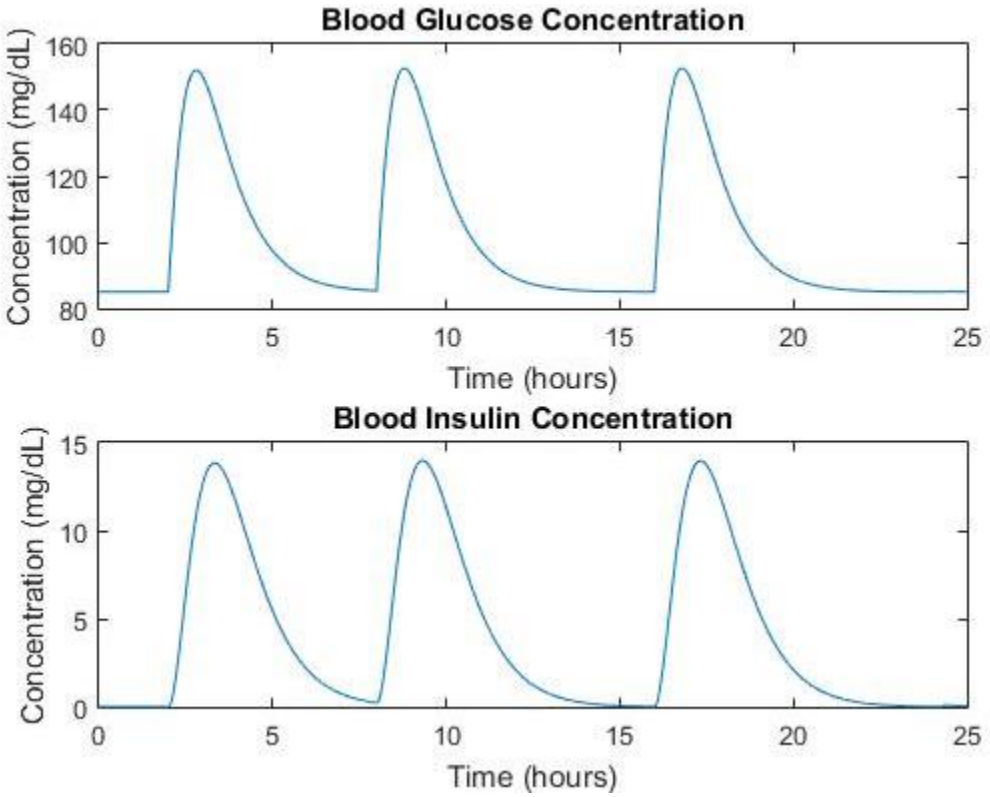


Figure 4: Plot of the concentration of glucose and insulin in the bloodstream of an untreated type II diabetic patient

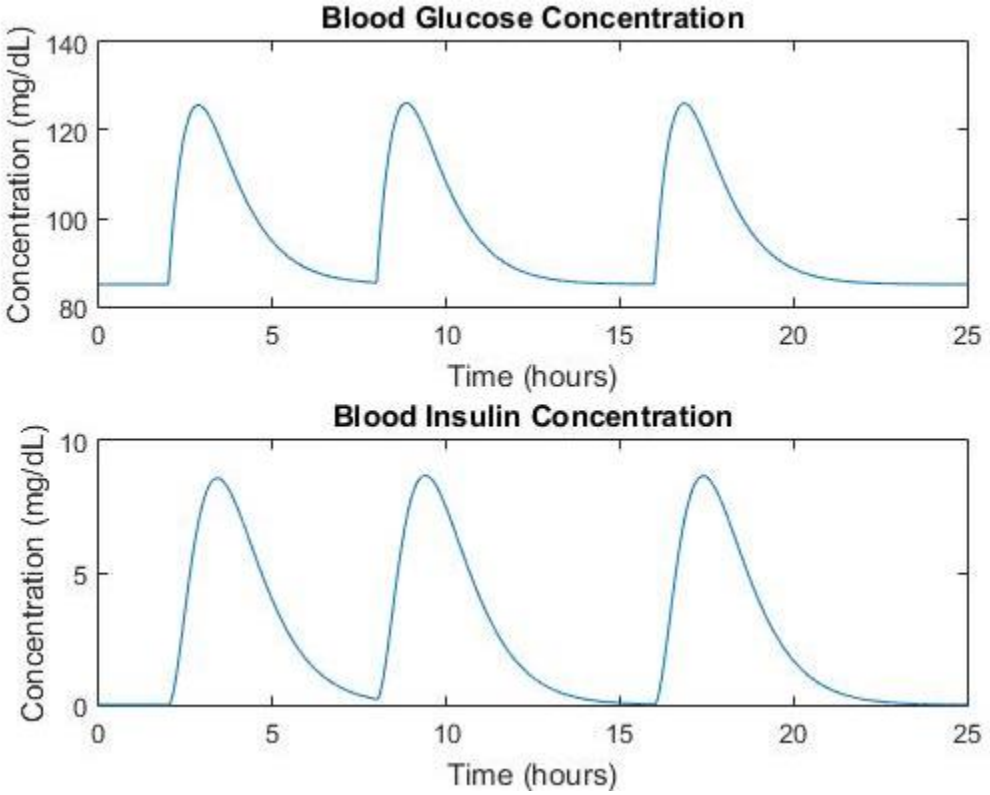


Figure 5: Plot of the concentration of glucose and insulin in the bloodstream of a treated type II diabetic patient. *B = 120*

Discussion:

The use of computers allows for improvements in the implementation of numerical methods and compartmental models in biosignal processing. Programming in a low-level language like C speeds up the process of executing these numerical methods. Instead of manually solving the ordinary differential equation, users are able to plug and chug without actually having to plug and chug!

The most common therapies for type II diabetes include diet, exercise, medication, and injected or infused insulin. Type I and type II diabetes were able to be simulated by adjusting the parameters to get the plots to look characteristic of the condition. Treatments were simulated by adjusting the A value for type I and B value for type II. Estimating the correct values for A and B was very difficult because of the magnitude of the number. The difference between 120 and 125 is so slight that it is almost an approximation that can be eye-balled. An increase in the rate of exogenous insulin input (A value) brings down the levels of blood glucose concentration. An increase in the amount of carbohydrates consumed (B value) decreases the concentration of insulin in the blood stream.

This model is limited because it assumes that diabetic patients will stick to regularly spaced meals three times a day. Not every diabetic is capable of keeping to those restrictions. Often times it is better for the diabetic patient to learn how to gauge how much insulin to administer or how much food to eat.

Conclusion:

The Runge-Kutta method works very well in the application of solving ODEs. It allowed the programmer to simulate the conditions of type I and type II diabetes and how to treat those. This model makes type II diabetes appear easier to treat, since it just requires that you eat properly and that the levels look similar to normal patients when treatment works. However, with type I diabetics, their waveforms look distorted with treatment. Perhaps that is an issue with simulation or treatment method.

C Code:

```
/* Solve_ODE.cpp : Defines the entry point for the console application.
              Peter Dobbs
              BIEN 3200 - Section 401
              7 December 2016
              Lab 7

              Program Description:
                     Solves the model ODEs for the blood glucose and insulin
concentrations in a ‘’’normal subject’’’ using the 4th order Runge-Kutta method
*/

#include "stdafx.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include "engine.h"
#include "mex.h"

float a1, a2, a3, a4; //Rate constant for endogenous glucose/insulin production or
degradation/utilization
float G0; //Reference blood glucose concentration (mg/dl)
float Gb; // Rate of insulin-independent glucose utilization (mg/(dl*hr))

/*
@params
float* u1 - array for storing values of exogenous insulin
float stepsize - constant value for step size
float B - (205 mg/(dl*hr)) is proportional to injected amount of glucose
*/
void calcExogenousGlucose(float* u1, float stepsize, float B) {
       for (float t = 0; t <= 25.0; t += stepsize) {
              if (t >= 2 && t < 5) {
                     u1[(int)(t / stepsize)] = (B)*exp(-2 * (t - 2));
              } else if (t >= 8 && t < 11) {
                     u1[(int)(t / stepsize)] = (B)*exp(-2 * (t - 8));
              } else if (t >= 16 && t < 19) {
                     u1[(int)(t / stepsize)] = (B)*exp(-2 * (t - 16));
              } else {
                     u1[(int)(t / stepsize)] = 0;
              }
       }
}

/*
@params
float* u2 - array for storing values of exogenous insulin
float stepsize - constant value for step size
float A - parameter (in mg/(dl*hr)) is proportional to injected dose of insulin
*/
void calcExogenousInsulin(float* u2, float stepsize, float A) {
       for (float t = 0; t <= 25.0; t += stepsize) {
              if (t >= 2 && t < 5) {
                     u2[(int)(t / stepsize)] = (A)*pow(t - 2, 2.0)*exp(-1 * (t - 2) /
3.0);
```

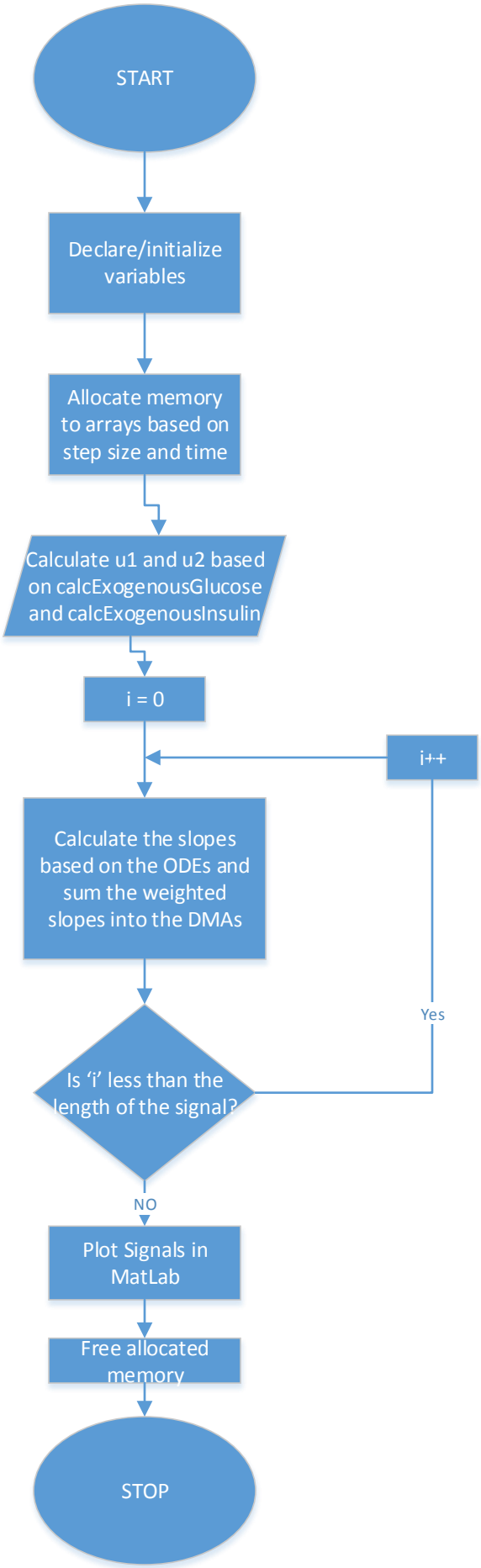
```
			} else if (t >= 8 && t < 11) {
				u2[(int)(t / stepsize)] = (A)*pow(t - 8, 2.0)*exp(-2 * (t - 8) /
3.0);
			} else if (t >= 16 && t < 19) {
				u2[(int)(t / stepsize)] = (A)*pow(t - 16, 2.0)*exp(-2 * (t - 16) /
3.0);
			} else {
				u2[(int)(t / stepsize)] = 0;
			}
		}
}

/*
For G<=G0
dG/dt = u1 - a1*G*S + a2*(G0 - G) - Gb
*/
float slope1(float u1, float G, float S) {
		return (u1 - (a1*G*S) + a2*(G0 - G) - Gb);
}

/*
For G<=G0
dS/dt = u2 - a4*S
*/
float slope2(float u2, float S) {
		return (u2 - a4*S);
}

/*
For G>G0
dG/dt = u1 - a1*G*S - Gb
*/
float slope3(float u1, float G, float S) {
		return (u1 - (a1*G*S) - Gb);
}

/*
For G>G0
dS/dt = u2 - a4*S + a3*(G-G0)
*/
float slope4(float u2, float G, float S) {
		return (u2 - a4*S + a3*(G - G0));
}

int main() {
		printf("Start of main program: Lab 7 ODE_Solver\n");
		float* G; //Blood glucose concentration (mg/dl)
		float* S; //Insulin glucose concentration (mg/dl)
		float *u1; //Input rate of exogenous glucose (mg/(dl*hr))
		float *u2; //Input rate of exogenous insulin (mg/(dl*hr))

		float t0, tf, ti, stepsize;
		float A, B; //coefficients on functions for exogenous insulin/glucose
		float k11, k12, k13, k14, q11, q12, q13, q14; //slopes

		int N;

		/*Model Parameters Under Physiologic Conditions (normal subject)*/
```

```
	a1 = 0.05;	// normal: 0.05
	a2 = 1.0;	// normal: 1.0
	a3 = 0.5;	// normal: 0.5
	a4 = 2.0;	// normal: 2.0
	G0 = 100;	// normal: 100
	Gb = 0;		// normal: 10
	A = 0;		// normal: 0
	B = 205;	// normal: 205

	/*Setup*/
	t0 = 0;
	tf = 25.0;
	stepsize = 0.005;
	N = (int)((tf - t0) / stepsize);

	/*Allocate memory to arrays*/
	G = (float*)calloc(sizeof(float), N);
	S = (float*)calloc(sizeof(float), N);
	u1 = (float*)calloc(sizeof(float), N);
	u2 = (float*)calloc(sizeof(float), N);

	/*Set Initial Conditions*/
	G[0] = 85;
	S[0] = 0;

	/*Determine values for u1 and u2*/
	calcExogenousGlucose(u1, stepsize, B);
	calcExogenousInsulin(u2, stepsize, A);

	/*char sim_file[100];
	FILE *out_file;
	printf("Enter the name of the output file.\n");
	scanf("%s", &sim_file);
	out_file = fopen(sim_file, "w");
	fprintf(out_file, "%f \t %f \t \n", 0.0, G[0], S[0]);*/

	for (int i = 0; i < N - 1; ++i) {
		ti = (i + 1)*stepsize;

		if (G[i] <= G0) {
			k11 = stepsize*slope1(u1[i], G[i], S[i]);
			q11 = stepsize*slope2(u2[i], S[i]);

			k12 = stepsize*slope1(ti + stepsize / 2, G[i] + k11 / 2, S[i] + q11
/ 2);
			q12 = stepsize*slope2(ti + stepsize / 2, S[i] + q11 / 2);

			k13 = stepsize*slope1(ti + stepsize / 2, G[i] + k12 / 2, S[i] + q12
/ 2);
			q13 = stepsize*slope2(ti + stepsize / 2, S[i] + q12 / 2);

			k14 = stepsize*slope1(ti + stepsize, G[i] + k13, S[i] + q13);
			q14 = stepsize*slope2(ti + stepsize, S[i] + q13);
		} else {
			k11 = stepsize*slope3(u1[i], G[i], S[i]);
			q11 = stepsize*slope4(u2[i], G[i], S[i]);
```

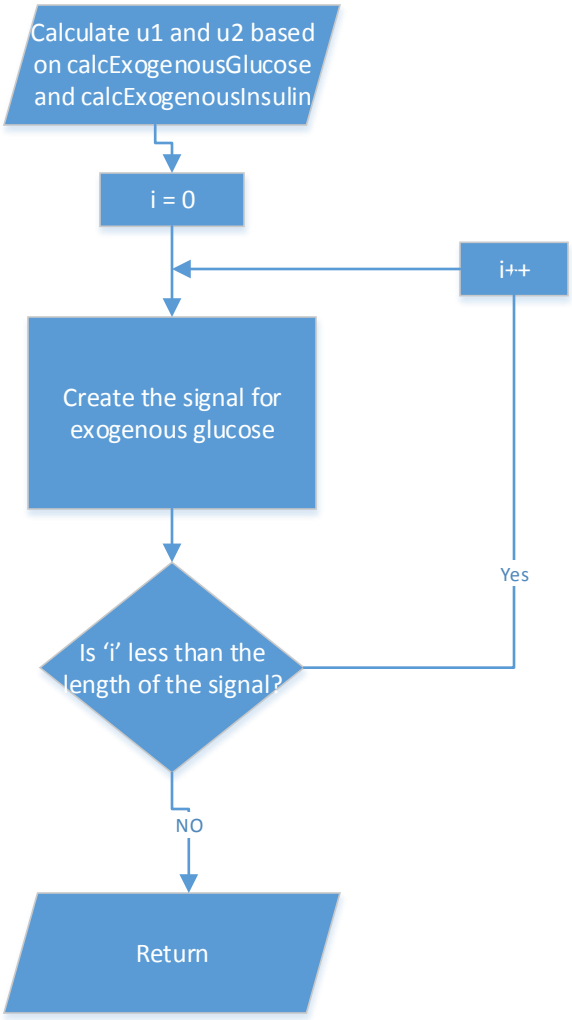
```
                        k12 = stepsize*slope3(u1[i] + stepsize / 2, G[i] + k11 / 2, S[i] +
q11 / 2);
                        q12 = stepsize*slope4(u2[i] + stepsize / 2, G[i] + k11 / 2, S[i] +
q11 / 2);

                        k13 = stepsize*slope3(u1[i] + stepsize / 2, G[i] + k12 / 2, S[i] +
q12 / 2);
                        q13 = stepsize*slope4(u2[i] + stepsize / 2, G[i] + k12 / 2, S[i] +
q12 / 2);

                        k14 = stepsize*slope3(u1[i] + stepsize, G[i] + k13, S[i] + q13);
                        q14 = stepsize*slope4(u2[i] + stepsize, G[i] + k13, S[i] + q13);
                }
                G[i + 1] = G[i] + (1.0 / 6)*(k11 + 2 * k12 + 2 * k13 + k14);
                S[i + 1] = S[i] + (1.0 / 6)*(q11 + 2 * q12 + 2 * q13 + q14);

                // send values to output file
                //fprintf(out_file, "%f \t %f \t %f \t \n", ti, G[i + 1], S[i + 1]);
        }
        //fclose(out_file);

        /*Plotting in MatLab*/
        printf("Start of Plotting\n");
        //////////////====MATLAB====/////////////////
        Engine *ep;
        ep = engOpen(NULL);

        mxArray* matG = mxCreateNumericMatrix(N, 1, mxSINGLE_CLASS, mxREAL);
        mxArray* matS = mxCreateNumericMatrix(N, 1, mxSINGLE_CLASS, mxREAL);

        memcpy((void*)mxGetPr(matG), (void*)G, sizeof(float) * N);
        memcpy((void*)mxGetPr(matS), (void*)S, sizeof(float) * N);

        engPutVariable(ep, "G", matG);
        engPutVariable(ep, "S", matS);

        engEvalString(ep, "t=0:0.005:24.995;"
                "figure; subplot(2,1,1),plot(t,G);"
                "title ('Blood Glucose Concentration');"
                "xlabel('Time (hours)');"
                "ylabel('Concentration (mg/dL)');"
                "subplot(2,1,2),plot(t,S);"
                "title ('Blood Insulin Concentration');"
                "xlabel('Time (hours)');"
                "ylabel('Concentration (mg/dL)');");

        system("pause"); //Pauses figure
        engClose(ep);
        //////////////======C======/////////////////
        printf("End of plotting\n");

        /*Free the allocated memory*/
        free(G);
        free(S);
        free(u1);
        free(u2);

        /*End Of Program*/
```

```
        printf("That's all folks!\n");
        system("PAUSE");
        return 0;
}
```

SolveODE (main)



SolveODE
(calcExogenousGlucose)



SolveODE
(calcExogenousInsulin)