DESIGN AND VALIDATION OF A NUMERICAL PROBLEM SOLVING ENVIRONMENT FOR ORDINARY DIFFERENTIAL EQUATIONS

Mordechai Shacham[1*], Neima Brauner[2], Michael B. Cutlip[3] and Michael Elly[4]

[1]Dept. Chem. Eng., Ben-Gurion University, Beer-Sheva, Israel
[2]School of Engineering, Tel-Aviv University, Tel-Aviv, Israel
[3] Dept. Chem. Eng., University of Connecticut, Storrs, CT 06269, USA
[4]Intel Corp., Qiryat Gat, Israel

Introduction

In the last decade, two distinct directions have emerged in the way Ordinary Differential Equation (ODE) solver software is being used (Enright[1]). These include Large Scale Scientific Computation and Problem Solving Environments (PSE). Most practicing engineers and scientists, as well as engineering and science students use numerical software for problem solving, while only a few very specific research applications require large scale computing. Consequently, this work is limited to PSEs.

We have carried out an extensive study of the requirements for a PSE intended for solving ODE's arising in Chemical and Biochemical engineering applications. In this study we have collected a "Library" of 100 sample problems and solved these problems which contain between one to 50 ordinary differential equations. Many of the problems were taken from the book of Cutlip and Shacham[2] and the rest from our previous publications (Brauner et al.[3], Shacham et al.[4],[5]) as well as from other sources. Mainly initial value problems were considered in this study. Polymath 6.1 (copyrighted by M. Shacham, M. B. Cutlip and M. Elly, http://www.polymath-software.com) and MATLAB (MATLAB is a trademark of The Math Works, Inc., http://www.mathworks.com) were used for solving the problems and validation the results. Stiff and non-stiff algorithms were used where needed. All the algorithms used included error estimation and step-size control in order to achieve solution of a pre-specified error tolerance.

The specific needs for a PSE of ODE's that were identified in this study are the following.

1. *Providing Approximations to the Solution at "Off-Mesh" Points* - Most ODE solvers provide approximations to the solution on a discrete, adaptively chosen mesh (determined by the step-size control algorithm). There are many applications where off-mesh values of the variables are needed. These applications include, for example, output of the results at equal intervals, plotting an oscillatory solution in a form of a continuous curve instead of "broken lines", output of the results for pre-specified values of the independent variable (as required in parameter estimation problems) and finding minimal and maximal values of variables.

* To whom correspondence should be addressed, e-mail shacham@bgumail.bgu.ac.il

2. *Enabling Event Control* - Often the value of an independent variable controls the solution process and the independent variable may reach a critical value (event) at an off-mesh point. A typical example of such a situation is in a biochemical problem, where the integration must be stopped when the amount of biomass reaches zero, as the model equations are not valid for negative biomass values.
3. *Dealing with Discontinuities* - Most numerical algorithms assume continuity of the derivatives, but often there is a need to change the derivative function during the integration.(A typical example is an exothermic reactor, which should be heated until the reaction starts, and then should be cooled). Such a change may not be properly taken care by the integration algorithm and erroneous solution may result.
4. *Implementation of Delays* - Delays that often appear in control related problems, may x historical data for selected variables which are often at off mesh points.
5. *Validation of Approximate Solutions* - Numerical solutions of ODE's are always considered "approximate", while the error estimation and step size control provide "some confidence in the numerical solution" (Shampine[6]). In many cases further validation is required to bring the confidence to an acceptable level.

The identified needs required improvements of traditional ODE solvers capabilities. Many of the requirements can be met if the solution is stored as a vector of piecewise interpolating polynomials. Such polynomials will enable the generation of off mesh points of the dependent variables both in the current integration step and in previous steps (history). It is also essential to stop and restart the integration at points of discontinuity, so that the integration algorithm has to deal only with continuous functions. The validity of the solutions can be verified by solving the same problem using several different algorithms, by carefully considering the error tolerances and by preparing "residual plots" of the results obtained by the different algorithms.

The study revealed cases where the use of state of the art software resulted in solutions with erroneous oscillatory behavior. In the rest of the paper we investigate the cause of this erroneous behavior and suggest a method to overcome it.

A Motivating Example – oxidation of ortho-xylene to phtalic anhydride in a tubular reactor

Let us consider the oxidation of ortho-xylene (A) to phthalic anhydride (B) in an ideal, continuous flow tubular reactor (Rase[7], Ingham et al.[8]). The reaction proceeds via a complex consecutive parallel reaction sequence:

$$\begin{aligned} A + O_2 &\xrightarrow{k_1} B \xrightarrow{k_2} C \\ A + O_2 &\xrightarrow{k_3} C \end{aligned} \qquad (1)$$

where C represents waste gaseous products (CO and CO_2).

The reactor model (including the numerical constants) is shown in Table 1. The model equations and the comments, shown in the Table, provide a complete documentation of the problem, thus there is no need to discuss the model any further.

Note that the model is described in a format that is consistent with the Polymath software package. Thus, the model can be copied from Table 1, pasted into the ODE solver program of the Polymath package and integrated up to the specified reactor length.

When the problem is solved using the *RKF45* algorithm of Polymath with the numerical values shown in Table 1, the results presented in Table 2 and Figures 1 and 2 are obtained. In Table 2 the initial, maximal, minimal and final values of the temperature and the flow-rates of components A, B and C are shown. The temperature rises near the entrance to the reactor (see Figure 1), reaches maximal value of 687.1 K than decreases gradually to an almost constant value of 662.2 K toward the exit from the reactor. The mol flow rate of the desired product, phtalyc anhydride reaches a maximum value of N_B = 0.00668 kmol/h at about z = 6.26 m (see Figure 2). From this point on, the flow rate of B decreases and the flow rate of the undesired product C is increases.

Solving the very same problem using the explicit *ode45* function of MATLAB yields the temperature profile shown in Figure 3. In this plot the decrease of the temperature from it maximum value is oscillatory instead of the smooth and gradual decrease indicated by Figure 1. It is very important to determine which of these solutions is the correct one, the reason for obtaining an incorrect solution and it can be prevented. These issues will be discussed in the next section.

Analysis of the Causes for the Oscillatory Temperature Profile in the Example Problem

It is easy to verify that the correct temperature profile is the smooth curve shown in Figure 1 and not the oscillatory one shown in Figure 3. This can be done by rerunning the problem with MATLAB using the same explicit *ode45* function with reduced error tolerances or using a stiff algorithm such as ode23s with the default error tolerances. However, it is very important to identify the what caused a sophisticated algorithm with error estimation and adaptive step size control to yield incorrect results.

The Polymath *RKF45* method uses the Runge-Kutta-Fehlberg algorithm (Forsythe et al.[9]) while the MATLAB *ode45* function uses a more recent Dormand-Prince[10] version of the Runge-Kutta method. Both programs include an adaptive step-size control algorithm that monitors the estimate of the integration error, and reduces or increases the step size of the integration in order to keep the error below a specified threshold. The accuracy requested is that both the relative and absolute (maximal) errors be less than the truncation error tolerance. In MATLAB, both relative (*RelTol*) and absolute (*AbsTol*) tolerances can be specified. The default values (that were used in solving the example problem) are *RelTol*= 0.001 and *AbsTol*= 10^{-6}. In Polymath only the *RelTol* can be specified with a default value of 10^{-6}. Polymath also allows to specify minimum number of reporting points (the default is *RP* = 100). The solution algorithm (the RKF algorithm, in this case) is forced to adjust the step size so that at least *RP* full steps are carried out inside the integration interval irrespective of the error tolerance used. In the *ode45* algorithm, four points (instead of one) are reported for each Runge-Kutta integration step. The additional points are calculated by cubic Hermite interpolation to the values and slopes computed at the ends of the step.

The results reported in the previous section were obtained by the *RKF45* algorithm using 100 integration steps, while the ode45 algorithm executed only 28 steps and reported the results in 112 points. If the relative error tolerance is reduced to *RelTol* = 10^{-6} (the same as the RKF45 default value), ode45 yields the same results that are shown in Figure 1 using 42 integration

steps. Thus, in this case, incorrect results were caused by inadequate error tolerance. But why the default error tolerances, which are sufficient for most problems (according to the developers' experience), are too large for this particular problem?

To further investigate this point the Jacobian matrix of the ODE system has to be calculated. Let us rewrite the system of ODEs (presented in Table 1) in the following form:

$$\begin{aligned}\frac{dT}{dz} &= f_1\left(T, X_B, X_C\right)\\ \frac{dX_B}{dz} &= f_2\left(T, X_B, X_C\right)\\ \frac{dX_C}{dz} &= f_3\left(T, X_B, X_C\right)\end{aligned} \qquad (2)$$

The matrix of partial derivatives (Jacobian matrix) can be defined:

$$\mathbf{J} = \begin{bmatrix} \frac{\partial f_1}{\partial T} & \frac{\partial f_1}{\partial X_B} & \frac{\partial f_1}{\partial X_C} \\ \frac{\partial f_2}{\partial T} & \frac{\partial f_2}{\partial X_B} & \frac{\partial f_2}{\partial X_C} \\ \frac{\partial f_3}{\partial T} & \frac{\partial f_3}{\partial X_B} & \frac{\partial f_3}{\partial X_C} \end{bmatrix} \qquad (3)$$

The eigenvalues of the Jacobian matrix can provide indication regarding the stiffness of the problem. An indicator recommended by Rice and Do[11] is the stiffness ratio, defined by equation (5):

$$SR = \frac{\max\left|\lambda_i\right|}{\min\left|\lambda_i\right|} \qquad (4)$$

where $\max\left|\lambda_i\right|$ is a negative eigenvalue of the $\mathbf{J}$ matrix with maximal absolute value and $\min\left|\lambda_i\right|$ is a negative eigenvalue of the $\mathbf{J}$ matrix with minimal absolute value. If SR>1000 the system of ODEs is considered a stiff system.

The elements of the Jacobian matrix can be also helpfulin determining the upper bounds on the errors in the calculated values of the slopes $\left(\frac{dT}{dz}, \frac{dX_B}{dz} \text{ and } \frac{dX_C}{dz}\right)$ using the error propagation formula. Let δT, δX_B and δX_C be the truncation errors in the temperature, conversion of B and conversion of C respectively. Then, the upper bound on the error of the slope dT/dz, for example is given by:

$$\delta\left(dT / dz\right) = \left|\frac{\partial f_1}{\partial T}\right|\delta T + \left|\frac{\partial f_1}{\partial X_B}\right|\delta X_B + \left|\frac{\partial f_1}{\partial X_C}\right|\delta X_C \qquad (5)$$

In Table 3 the elements of the Jacobian matrix, the eigenvalues and the stiffness ratio values are shown for the initial point (z = 0) and the final point (z =8). The problem is not considered stiff by the commonly used measures, as SR = 397.95 at the initial point and SR = 27.87 at the final point. However, the partial derivative values associated with the calculation of

the error $\delta(dT/dz)$ using equation (5) are very large. Introducing the numerical values at the final points, for example, yields:

$$\delta(dT/dz) = 10.4(\delta T) + 152(\delta X_B) + 179(\delta X_C)$$

Thus, the error in the slope is larger by an order of magnitude than the error in the temperature, and by more than two orders of magnitude than the error in X_B and X_C.

To compare the actual error in the temperature and its slope values, the integration interval was divided into 112 sections (according to the *ode45* reporting points when the default error tolerance was used). The integration was carried out separately for the 112 sections so that the variable values at the reporting points can be compared. Using the very small integration intervals ensures that the results obtained are accurate, so that the difference between the new and previously obtained results can be considered as the the error in δT or $\delta(dT/dz)$.

Figure 4 shows the errors in the calculated temperature values as function of the distance from the entrance to the reactor. The maximal deviations are approximately 0.8 K to -0.8 K, thus the relative error is 0.8/670 ~ 0.001, consistent with the specified relative error tolerance. In Figure 5 the errors in the dT/dz values are shown. Those errors are approximately ten times larger than the respective temperature value errors. This is consistent with the error propagation formula assuming that the errors δX_B and δX_C can be neglected. The temperature slope is gradually decreasing (in absolute value) when getting closer to the reactor exit. It gets the value of $dT/dz = -1$ at about $z = 3.8$ and $dT/dz = -0.227$ at about $z = 8$. Considering that the error at some point reaches a value close to 8 implies relative errors over 2000%.

In order to assess the role of the interpolation used in *ode45* for increasing the number of reported points, the calculation of the error in the temperature values was repeated after removing the points that were obtained by interpolation. The resultant plot is shown in Figure 6. Comparing Figure 6 with Figure 4 reveals that by removing the interpolated points the positive deviations were mostly eliminated, thus the interpolation is responsible to about half of the amplitude of the oscillation cycle.

Aerobic Microbial Growth Problem – A Stiff Example (Brauner et al.[3])

The system of equations representing this problem is shown in Table 4. There are three differential equations representing the concentration of the cells (*x*), the substrate (*S*) and oxygen (*O2*) in a microbial growth system. The model equations and the comments, shown in the Table, provide a complete documentation of the problem, and the format of the model is consistent with Polymath.

In this case there are substantial differences between the oxygen concentration profiles obtained with the *RKF45* algorithm of Polymath and the *ode45* function of MATLAB when the default error tolerances are used. The solution obtained by the *RKF45* algorithm (RelTol = 10^{-6}) is a smooth curve as can be seen in Figure 7. The oxygen concentration changes are very small in absolute values. At the start, the concentration decreases from the initial value of 8 mg/l to 7.9975 mg/l, then increases gradually to a maximum of 7.998778 mg/l and decreases again gradually to a minimum value of 7.99371 mg/l at time t= 6 h 25 min. Shortly after this time all the substrate is consumed and the concentration of the oxygen returns to the initial value of 8 mg/l.

The solution obtained with the *ode45* function is oscillatory, as shown in Figure 8, with deviations of up to 0.008 mg/l (or -0.008 mg/l) from the correct values. It should be pointed out that the maximal deviations are larger than the maximal change in the oxygen concentration

(0.0063 mg/l). To investigate the cause of the oscillation, the Jacobian matrix of the system of equations and its eigenvalues have been calculated. These values are shown in Table 5 at the initial point (t = 0) and at the point where all the substrate is consumed (t = 6.5 h). It can be seen that the system is stiff (the stiffness ratio SR = 9900 at t = 6.5 h, for example). To check whether the oscillations can be attributed to the stiffness, the problem was resolved using the explicit *ode45* function, after reducing the relative error tolerance to $RelTol = 10^{-6}$.
With the reduced error tolerance the correct solution, shown in Figure 7, was obtained. Thus, the explicit Runge-Kutta algorithm with adaptive step-size control yields correct solutions also to stiff systems provided adequate error tolerances are specified. Of course, an explicit algorithm is very inefficient in solving stiff problems. The *ode45* function used 1220 steps to obtain the correct solution of this problem, while the stiff *ode23s* algorithm used only 57 steps.

Using the terms of the Jacobian matrix at t = 6.5 h the propagated error in the slope of the oxygen concentration $\delta(dO2/dt)$ can be calculated:

$$\delta(dO2/dt) = 0.27658(\delta x) + 16.143(\delta S) + 400(\delta O2)$$

Thus, the upper limit of the absolute error in the slope is 400 times the error in the oxygen concentration: 3.2 mg/l-h. Considering that the value of $dO2/dt$ is in mostly in the range of $10^{-3} - 10^{-5}$, the relative errors reach huge values.

Conclusions

The cause of erroneous oscillatory behavior when solving systems of ODEs with state of the art software has been investigated. It has been demonstrated that such a behavior may result from the use of inappropriately large error tolerances. Depending on the terms of the Jacobian matrix, small error in the variable values may lead to huge relative errors in the derivative values. These errors cause the erroneous oscillations.
The oscillations can be prevented by monitoring the errors in the derivative values and restarting the integration with smaller error tolerances, if the error in the derivatives exceeds a certain limit.

For the examples considered it has been shown that the use of an explicit integration algorithm for a stiff system does not cause erroneous oscillatory behavior provided that adequate error tolerances are specified. The adaptive step change algorithm of the explicit methods enable obtaining correct solutions also for stiff systems in the expense of much lower efficiency that that achieved with stiff algorithms

References

1. Enright W.H., "The design and implementation of usable ODE software", *Numerical Algorithms* **31** (1-4): 125-137 (2002)

2. Cutlip, M. B. and Shacham, M. *Problem Solving In Chemical and Biochemical Engineering with Polymath, Excel and MATLAB.* Prentice-Hall, Upper Saddle River, New-Jersey, 2006.

3. Brauner, N., Shacham, M. and M. B. Cutlip, ``Computational Results: How Reliable Are They? A Systematic Approach to Modal Validation", *Chem. Eng. Educ*., **30** (1), 20-25 (1996).

4. Shacham, M., N. Brauner and M. Pozin, "Potential Pitfalls in Using General Purpose Software for Interactive Solution of Ordinary Differential Equations", *Acta Chimica Slovenica*, **42**(1), 119(1995)

5. Shacham, M., N. Brauner and M. B. Cutlip, "Prediction and Prevention of Chemical Reaction Hazards – Learning by Simulation", *Chem. Eng. Educ*., **35**(4), 268-273(2001)

6. Shampine L.F., "Error estimation and control for ODEs", *Journal of Scientific Computing* **25** (1): 3-16 (2005)

7. Rase, Howard F. (1977) Chemical Reactor Design For Process Plants, New York, John Wiley.

8. Ingham, John, Irving J. Dunn, Elmar Heinzle and Jiri E. Prenosil (1994), Chemical Engineering Dynamics : Modelling with PC Simulation, Weinheim,VCH

9. Forsythe, G.E., M.A. Malcolm, and C.B. Moler, Computer Methods for Mathematical Computation, Prentice-Hall, Englewood Cliffs, 1977.

10. Dormand, J. R. and P.J. Prince, "A family of Embedded Runge-Kutta Formulae, J. Comp. Appl. Math., 6(1980), pp. 19-26

11. Rice, R. G. and Do, D.D. Applied Mathematics and Modeling for Chemical Engineers, John Wiley, (1995)

Table 1. Reactor Model for the Tubular Reactor Example

No.	Equation # Comment
1	d(Temp)/d(z) = B2 * RB + B3 * RC – B4 * (Temp – Tj) # Temperature in the reactor [K] (energy balance)
2	d(XB)/d(z) = B1 * RB # Conversion to phtalic anhydride (mol balance)
3	d(XC)/d(z) = B1 * RC # Conversion to CO and CO2 (mol balance)
4	G = 4684 # Superficial mass velocity [kg/m^2-h]
5	MM = 0.02948 # Mean molecular weight [g/mol]
6	NA0 = 9.27e-3 # Inlet mole fraction of o-xylene
7	N0 = 0.208 # Mole fraction of oxygen
8	CP = 0.25 # specific heat [kcal/kg K]
9	H1 = -307 # Heat of reaction A -> B [kcal/mol]
10	H3 = -1090 # Heat of reaction A -> C [kcal/mol]
11	RHOB = 1300 # Catalyst bulk density [kg/m^3]
12	DP = 3e-3 # Catalyst particle diameter [m]
13	U = 82.7 # Heat transfer coefficient [kcal/m^2-h-K]
14	DT = 0.025 # Tube diameter [m]
15	R = 1.897 # Gas constant [cal/mol-K]
16	Tj = 660 # Cooling jacket temp. [K}
17	B1 = RHOB * MM / (G * NA0) # Combined terms in the mole balance eqs.
18	B2 = RHOB * (-H1) / (G * CP) # Combined terms in the energy balance eqs.
19	B3 = RHOB * (-H3) / (G * CP) # Combined terms in the energy balance eqs.
20	B4 = 4 * U / (G * CP * DT) # Combined terms in the energy balance eqs.
21	XA = XB + XC # Conversion of o-xylene
22	NA = NA0 * (1 – XA) # Mole fraction of o-xylene
23	NB = NA0 * XB # Mole fraction of phtalic anhydride
24	K1 = 1000 * exp(-27000 / (R * Temp) + 19.837) # Kinetic constant of the 1st reaction [kmol/kg(cat) – h]
25	K2 = 1000 * exp(-31400 / (R * Temp) + 20.86) # Kinetic constant of the 2nd reaction [kmol/kg(cat) – h]
26	K3 = 1000 * exp(-28600 / (R * Temp) + 18.97) # Kinetic constant of the 3rd reaction [kmol/kg(cat) – h]
27	RA = -(K1 + K3) * NA * N0 # Rate of reaction of o-xylene [kmol/kg(cat) – h]
28	RB = K1 * NA * N0 – K2 * NB * N0 # Rate of generation of phtalic anhydride [kmol/kg(cat) – h]
29	RC = K3 * NA * N0 + K2 * NB * N0 # Rate of generation of CO and CO2 [kmol/kg(cat) – h]
30	XB(0) = 0 # Initial conversion to phtalic anhydride
31	XC(0) = 0 # Initial conversion to CO and CO2
32	Temp(0) = 650 # Initial temp. in the reactor [K]
33	z(0) = 0
34	z(f) = 8 # Reactor length [m]

Table 2. Principal Results for the Tubular Reactor Example

	Initial Value	Minimal Value	Maximal Value	Final Value
z (m)	0	0	8	8
Temp. (K)	650	650	687.0774	662.2154
N_A (kmol/h)	0.00927	0.0002977	0.00927	0.0002977
N_B (kmol/h)	0	0	0.0066834	0.0066026
N_C (kmol/h)	0	0	0.0023697	0.0023697

Table 3. Jacobian Matrix and its Eigenvalues for the Tubular Reactor Example

	Initial point (z = 0)				Final point (z = 8)			
	Jacobian Matrix			Eigenvalues	Jacobian Matrix			Eigenvalues
1	-7.2597	-101.15	-117.9	-7.1373	-10.382	-152.35	-179.43	-10.374
2	0.0073076	-0.23394	-0.21692	-0.36324	-0.000397	-0.35528	-0.32778	-0.029479
3	0.0008886	-0.007886	-0.0249	-0.017935	0.0007814	-0.01106	-0.03856	-0.37226
SR				397.95				27.87

Table 4. Mathematical Model of the Aerobic Microbial Growth System

No.	Equation # Comment
1	d(x)/d(t) = mju * x # Cell concentration [g/liter]
2	d(S)/d(t) = If (S > 0) Then (-(mju * x) / Yxs - m * x) Else (0) # Substrate concentration [g/liter]
3	d(O2)/d(t) = KLa * (O2star - O2) - mju * x / Yxo2 - mo2 / x # Oxygen concentration [mg/liter]
4	Ks = 0.05 # Constant [g/liter]
5	mjumax = 0.6 # Constant [1/h]
6	mju = mjumax * S / (Ks + S)
7	Yxs = 0.5 # Constant [g-cells/g-glucose]
8	Yxo2 = 1 # Constant [g-cells/g-O2]
9	m = 0.08 # Constant [g-glucose/g-cells-h]
10	KLa = 400 # Constant [1/h]
11	mo2 = 0.1 # Constant [g-O2/g-cells-h]
12	O2star = 8 # Constant [mg/liter]
13	t(0) = 0
14	t(f) = 10 # Final time [h]
15	x(0) = 0.1 # Initial cell concentration [g/liter]
16	S(0) = 10 # Initial substrate concentration [g/liter]
17	O2(0) = 8 # Initial oxygen concentration [mg/l]

Table 5. Jacobian Matrix and its Eigenvalues for the Aerobic Microbial Growth System

	Initial point (t = 0)				All substrate consumed (t = 6.5)			
	Jacobian Matrix			Eigenvalues	Jacobian Matrix			Eigenvalues
1	0.59701	2.97E-05	0	-400	0.281	16.143	0	-400
2	-1.274	-5.94E-05	0	3.9805E-06	-0.64199	-32.287	0	-31.965
3	9.403	-2.97E-05	-400	0.59695	-0.27658	-16.143	-400	-0.040402
SR								9900

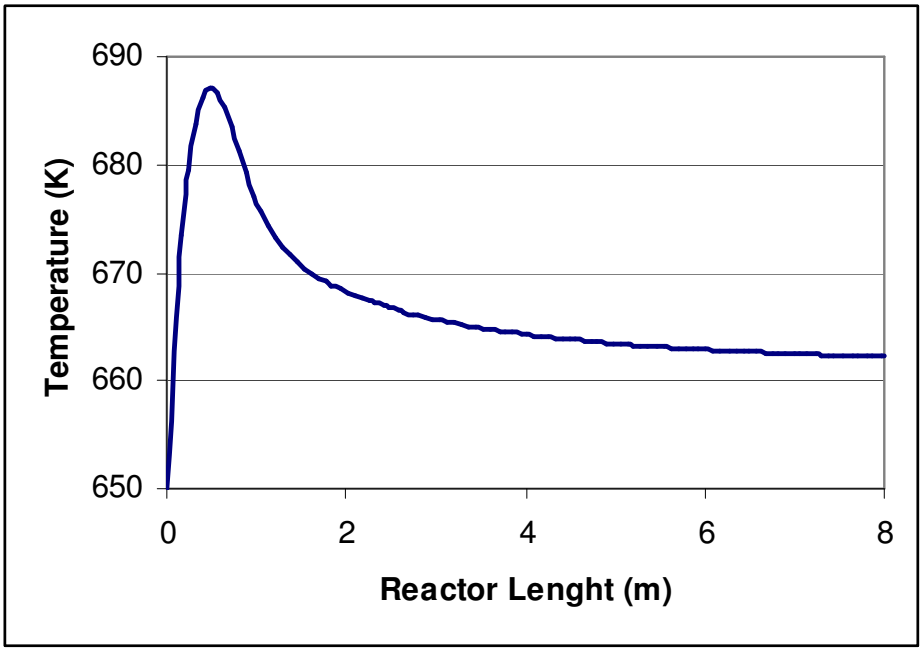

Figure 1. Temperature profile in the o-xylene oxidation reactor (obtained by the Polymath RKF45 algorithm)

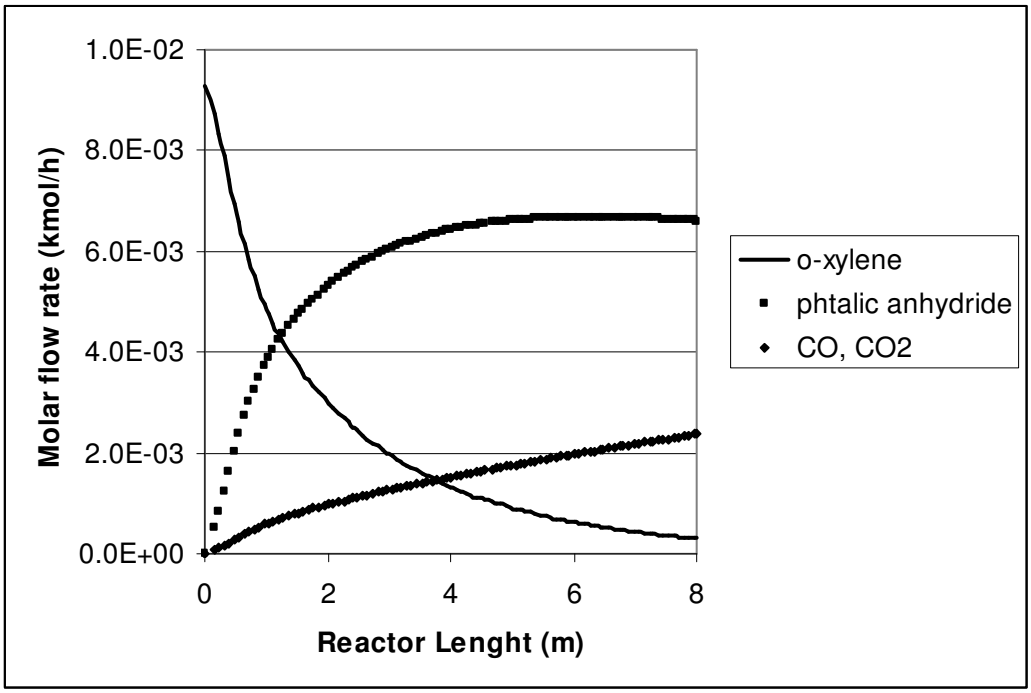

Figure 2. Flow rate profiles of the reactant and the product in the o-xylene oxidation reactor (obtained by the Polymath RKF45 algorithm)

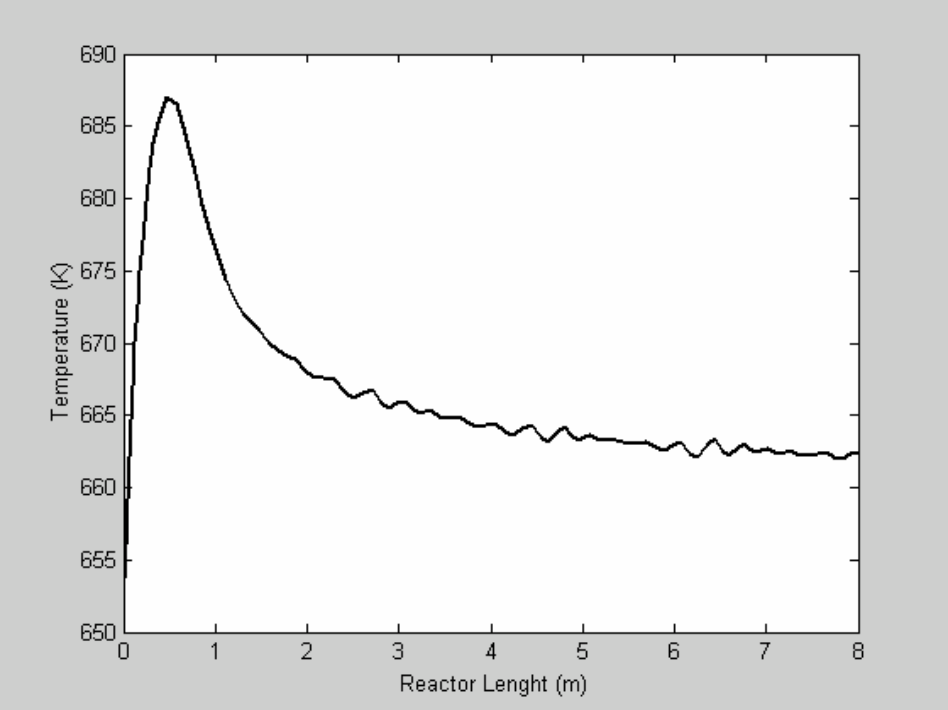

Figure 3. Temperature profile in the o-xylene oxidation reactor (obtained by the MATLAB ode45 algorithm)

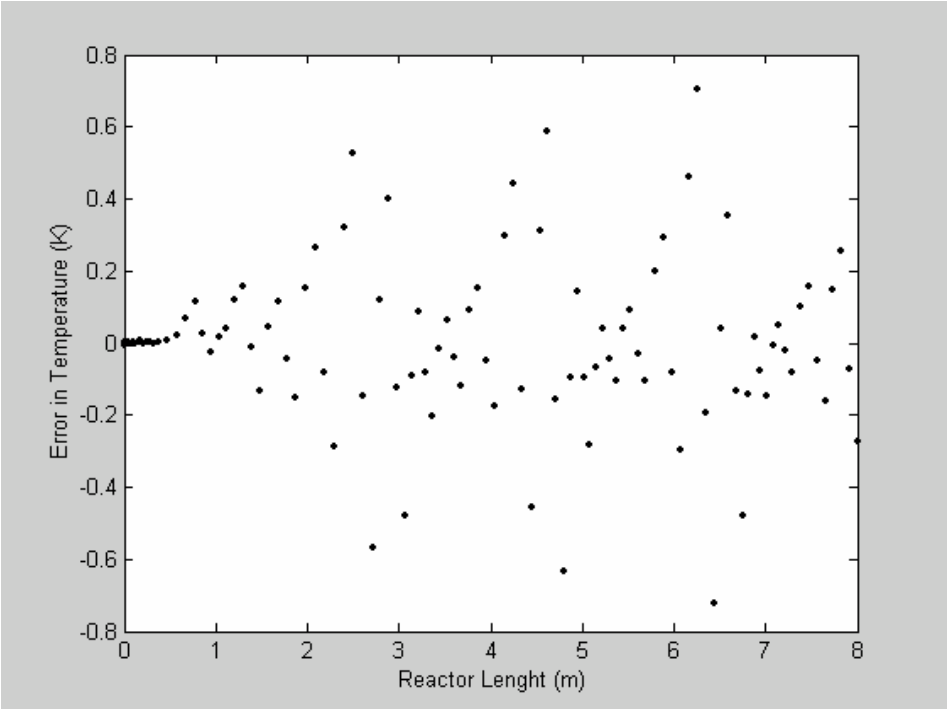

Figure 4. Error in the calculated temperature values (ode45 function, RelTol = 0.001)

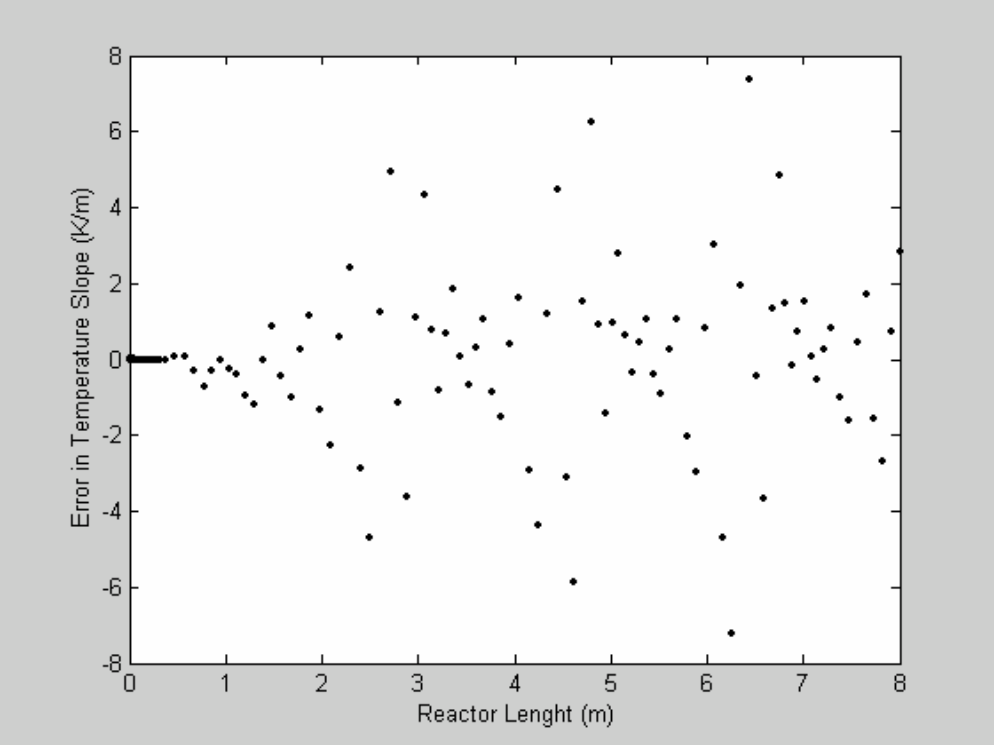

Figure 5. Error in the calculated *dT/dz* values (ode45 function, RelTol = 0.001)

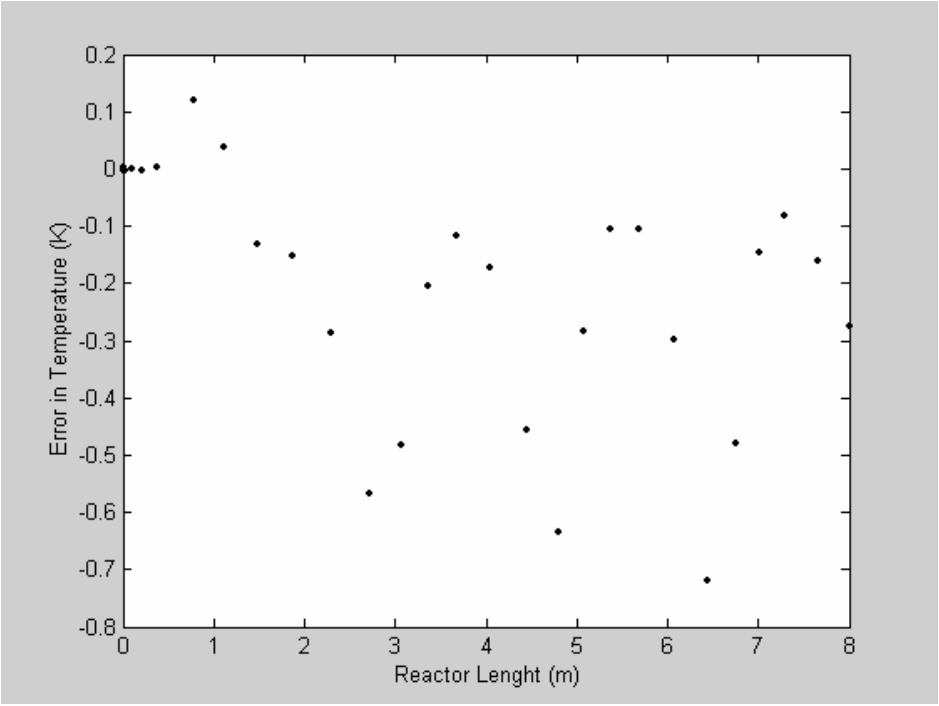

Figure 6. Error in the calculated temperature values (ode45 function, RelTol = 0.001, interpolated points removed)

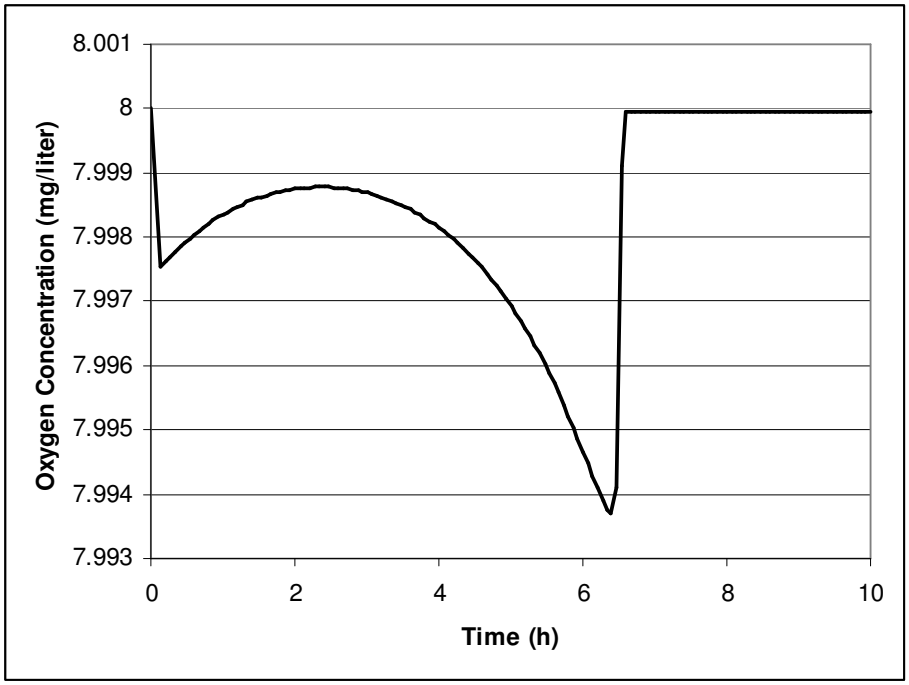

Figure 7. Oxygen concentration profile in the aerobic microbial growth system (obtained by the Polymath RKF45 algorithm)

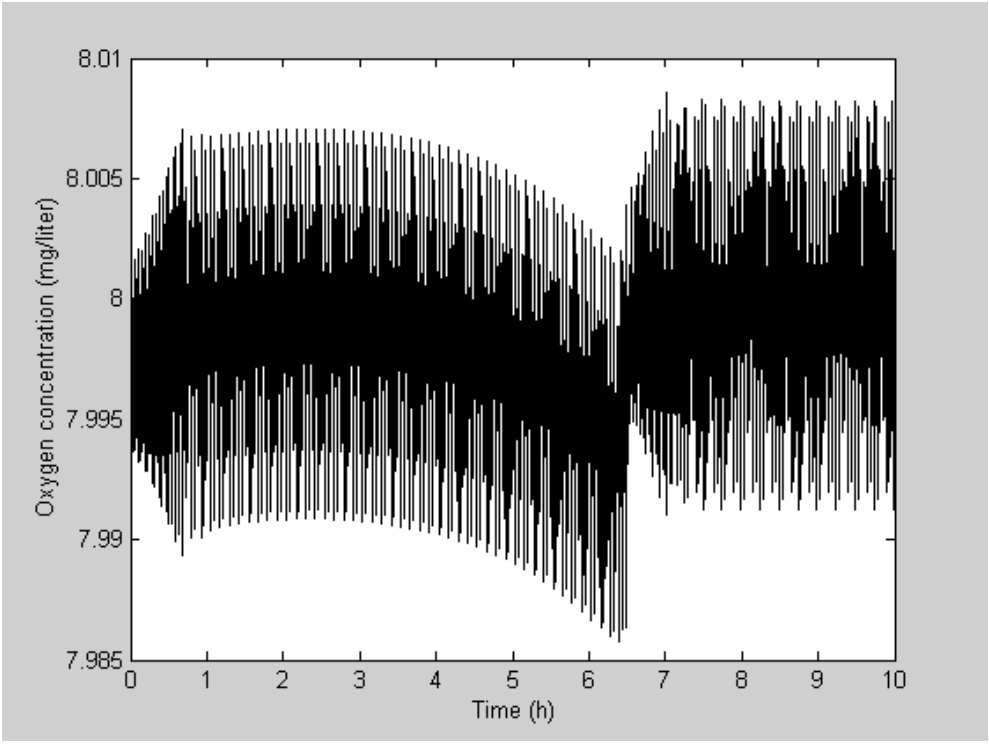

Figure 8. Oxygen concentration profile in the aerobic microbial growth system (obtained by the MATLAB ode45 algorithm)