國立政治大學 應用數學系

碩士 學位論文

混合週期因子的生長曲線建模

The Modeling of Logistic Curve by Mixing the Periodic Factor

碩士班學生：林昱　撰
指導教授：曾正男　博士

中華民國 106 年 5 月 11 日

國 立 政 治 大 學 應 用 數 學 系

林昱君所撰之碩士學位論文

混合週期因子的生長曲線建模
The Modeling of Logistic Curve by Mixing the Periodic Factor

業經本委員會審議通過
論文考試委員會委員：

指導教授：

系主任：

中華民國 106 年 5 月 11 日

致謝

耶! 終於把論文打完囉!! 真是太開動了!! 這些日子要感謝曾正男老師對我的協助與指導，讓我能夠在寫這篇論文時，能夠十分得心應手。同時也要感謝幫我口試的薛老師和李老師，對我的論文提出許多修改的建議。

在碩班的這幾年，真是過得多采多姿。非常幸運的在這段期間遇到了我的女朋友辰兒，一起在啦啦隊裡同甘共苦，參加了大大小小的比賽跟表演。在系上，也交到了一群白痴朋友阿宗、治安、沛倫跟小潤，一起度過了實變的激戰還有各種夜晚的促膝長談。除此之外，這段期間也很開心加入了政大啦啦隊，能跟隊上的大家一起在場上揮灑青春的汗水，應該是我碩士班生涯中美好的一段回憶。最後，也很高興能跟屁孩一起踏入重訓的這個領域，希望未來能一起練成超強巨巨。

能完成碩士學位算是人生中重要的一個里程碑。接下來的日子希望能將所學做出對社會有貢獻的事。

林昱　謹誌于
國立政治大學應用數學所
中華民國一百零六年七月

中文摘要

在這篇論文中，我們引用“海岸綠提-水筆仔”的資料。我們的目的是要找到一個合適的函數去擬合這些資料，以及找到一個合適的微分方程式，使得該方程式的解是能夠擬合這些資料的。這樣的函數以及微分方程將能幫助我們更了解這筆資料的性質，並對預測資料未來走向更有幫助。

我們首先藉由生長曲線來建構我們的數學函數，並對這個數學函數的模型加上週期項來改良。藉由 Matlab curve fittig tool，我們找到了這個函數的一組參數來擬合原始資料。最後得到的結果如我們的預期，加了週期函數做出來的建模較原本生長曲線的建模更為貼近原始資料。

在尋找微分方程系統的建模上，我們參考了文獻方法，並加以改良。然而這個新的微分方程式在加上週期項來改良之後卻沒辦法找到解析解，所以我們利用基因演算的方法來去尋找適合這個微分方程系統的參數，並搭配 Heun’s method，RK2 method 和 RK4 method 求出一些數值解。最後得到了一個比原參考文章更好的結果。

關鍵字：曲線擬合、數學模型、生長曲線、基因演算法

Abstract

In this paper, we focus on the data from the website 'Seacoast Green Bank-Kandelia'. There are two things we want to do for these data. First, we want to find a function which graph fits these data. Second, we want to find a differential equation such that its solution fits these data well. By exploring the function and the differential equation, we can understand more properties of these data.

We first build our mathematical function from Logistic curve and improve it by adding a periodic factor. By using Matlab curve fitting tools, we find parameters of this function which fit these data well. The final result is the same as our expectation. The model by adding a periodic factor fits the data better than the model of Logistic function.

To look for a differential equation, we follow the method in [7] and improve it. However, there is no analytical solution after adding a periodic factor into the model. Thus we use the method of a genetic algorithm to find suitable parameters of this differential equation. Moreover, we find the numerical solution by using Heun's method, RK2 method and RK4 method. Finally, we get a better result than one in Ren-fa, Chen's paper.

Keywords: Curve fitting, Mathematical model, Logistic curve, Genetic algorithm

Contents

口試委員會審定書 i

致謝 ii

中文摘要 iii

Abstract iv

Contents v

List of Figures vii

List of Tables ix

1 Introductin 1

2 Source of Data 4

2.1 Sources 4

2.2 Time and Height 7

3 Construct Model Function 9

3.1 Fit by Logistic Curve 9

3.2 Model 1 10

3.3 Model 2 11

3.4 Genetic Algorithm 13

4 Construct Differential Equation Model **15**

4.1 Survey from Ren-fa's Paper . . . 15

4.2 First Order Differential of the Data . . . 20

4.3 Model 3 . . . 22

4.3.1 Using Regression . . . 23

4.3.2 Using Finite Difference Method . . . 25

4.3.3 Using Analytical Solution . . . 29

4.4 Model 4 . . . 32

4.4.1 Using Finite Difference Method . . . 32

4.4.2 Using Genetic Algorithm . . . 35

4.4.3 Using Matlab Curve Fitting Tool . . . 39

5 Conclusion **42**

A Code use in paper **44**

A.1 Code 01 . . . 44

A.2 Code 02 . . . 55

A.3 Code 03 . . . 65

A.4 Code 04 . . . 76

A.5 Code 05 . . . 90

Bibliography **102**

List of Figures

2.1 Some pictures of process of growth in different month 7

2.2 Graph of time and height 7

3.1 Graph of data and equation 3.2.1 11

3.2 Graph of data and (3.3.1) 12

3.3 Predict of Kandelia's height by equation 3.3.1 13

4.1 Graph of Table 2.3 17

4.2 Graph of Table 4.2 and equation4.1.3 18

4.3 Graph of $E(t)$ 19

4.4 Numerical result of equation 4.1.5 20

4.5 Fit the data by regression 24

4.6 Best result 25

4.7 Best result 29

4.8 Graph of data and (4.3.1) 30

4.9 Graph of data and (4.3.5) 31

4.10 Best result 35

4.11 Fit the data by genetic algorithm 36

4.12 Fit the data by genetic algorithm with parameters adjusted 37

4.13 Best result 38

4.14 Predict of Kandelia's height by equation 4.4.1 39

4.15 Graph of data and equation(4.4.1) 40

4.16 Best result 41

List of Tables

2.1 Germination under different environments 5

2.2 Process of growth of Kandelia all the year round 6

2.3 Relation of time and height 8

4.1 Relation between velocity of growth and height of hypocotyl of Kandelia . . . 16

4.2 Relation between acceleration of growth and height of hypocotyl of Kandelia . 17

4.3 Relation between velocity of growth and height of hypocotyl of Kandelia . . . 21

5.1 Conclusion 43

Chapter 1

Introductin

In this thesis, we focus on the data from the website 'Seacoast Green Bank-Kandelia' [1]. There are two things we want to do with these data. First, we want to find a function which graph fits these data. Second, we want to find a differential equation such that its solution fits these data well. By exploring the function and the differential equation, we can understand more properties of these data.

We begin our paper in chapter 2 by referring to the website [1]. In this website, we observe flower of Kandelia. Also, we record the process of its anthesis and seed-bearing. Furthermore, we are interested in the growth of hypocotyl. Thus, we sort out height of Kandelia in different month. We then notice that these data perform quite similar to logistic curve. Therefore, our modeling starts by using it. Here is a brief introduction to the logistic curve according to the website [4]. Further detail will be shown in chapter 2.

A logistic function or logistic curve is a common ”S”shape (sigmoid) curve, using the equation below:

$$Q(t) = \frac{B}{1 + Ae^{-Bkt}}$$

Here, e is the natural logarithm (also known as Euler's number). A , B and k are constants.

Next, in chapter 3, we build our mathematical function based on the logistic curve. By using

the Matlab curve fitting tool, we find suitable parameters. We want the function with these parameters fit original data well. The model is below:

Let $Q(t)$ be the height of Kandelia, A , B and k are the parameters.

$$Q(t) = \frac{B}{1 + Ae^{-Bkt}}$$

We then fit our data to this model. After that, we notice that the curve seems to be different from the original data by adding a periodic factor. Thus we adjust our model as follows:

Again, let $Q(t)$ be the height of Kandelia. A , B, k, a, b, and c are the parameters we want to find.

$$Q(t) = \frac{B}{1 + Ae^{-Bkt}} + asin(bt + c)$$

The final result is the same as our expectation. By adding a periodic factor, the model fits the data better than the model of the logistic function. However, the parameters suitable for this model were found by using Matlab curve fitting tool. The problem is, we are not familiar with the operation inside the Matlab. Thus, at the end of Chapter 3, we give an introduction to Genetic Algorithm [6]. This is a clearer way of getting these parameters.

Next in chapter 4, to look for a differential equation model, we follow R-f's paper [7]. We then improve it by estimating the velocity of growth of the Kandelia in a better way. That is, instead of using the forward difference method [5] R-f's paper does, we use the central difference method. This will make estimation much better.

Note that the main difference between the models in chapter 3 and chapter 4 is that, in chapter 3 we find a function to fit the data directly. On the other hand, in chapter 4 we alternate the form of the logistic curve into a first order differential equation. We want to find parameters which fit the differential equation. Thus we construct the following model:

$$Q'(t) = p_1 Q^2(t) + p_2 Q(t)$$

Here, $Q'(t)$ is the growth rate of Kadelia. $Q(t)$ is the height of Kandelia. Also, p_1 and p_2 are the parameters we hope to find.

Under this model, we use the relationship between height and velocity. With regression, we can find the suitable parameters. Next, by using Heun's method, the RK2 method and the RK4 method, we then find the numerical solution of this model. Furthermore, since this equation is the form of Bernoulli equation [2], we can find its analytical solution. We then compare this solution to the numerical result.

Similar to Chapter 3, we notice that the curve seems to be different from the original data after adding a periodic factor. Thus we adjust our model as following:

$$Q'(t) = p_1 Q^2(t) + p_2 Q(t) + asin(bQ(t) + c)$$

$Q'(t)$ is the growth rate of Kadelia. $Q(t)$ is the height of Kandelia. Here, a, b, c, p_1 and p_2 are the parameters we hope to find.

However, there is no analytical solution to the model. Thus we use genetic algorithm [6] to find suitable parameters for this differential equation. Moreover, we find the numerical solution by using Heun's method, the RK2 method and the RK4 method. Finally, we get a better result than the survey paper [7].

In chapter 5, we give a review to all the methods we used in this paper. Also, we draw a table to show all the results.

Chapter 2

Source of Data

In this chapter, we will talk about the data we use in this paper. In Section 2.1, we will give the website [1] an introduction. It is a website recording behaviors of Kandelia. Next, in Section 2.2, we will sort out the data of Kandelia from time to time.

2.1 Sources

We focus on two observations of Kadelia in the website [1]. First, we observe hypocotyl of Kandelia in four different environment. We then recored its phenomenon of germination in six weeks.

Second, we observs flower of Kandelia. Also, we record the process of its anthesis and seed-bearing. Furthermore, we are interested in the growth of hypocotyl. Detailed observations are shown in Table 2.1 and Table 2.2.

We also put some pictures of process of growth in different months. From left to right namely: “small bud in March”, “most of flower blossom in July”, “seed in September”, “fruit grow viviparous seedlings in October”, and “viviparous seedlings grow about twenty centimeters February next year”. The pictures are shown in Figure 2.1.

Table 2.1: Germination under different environments

	First week	Second week	Third week	Fourth week	Sixth week
Soil of Intertidal zone and water of Intertidal zone	Viviparous seedlings grow about 0.5 to 0.7 centimeters of root.	Viviparous seedlings about 1.5 to 2 centimeters of root,end of bud begins to grow.	There are three viviparous seedlings to sprout of phenomena.	There are two viviparous seedlings start to grow leaves.	There are four viviparous seedlings grow a pair of leaves.
Soil of Intertidal zone and general water	Viviparous seedlings grow about 0.5 to 0.6 centimeters of root.	Viviparous seedlings about 1.5 to 2 centimeters of root,end of bud begins to grow.	There are four viviparous seedlings to sprout of phenomena.	There are four viviparous seedlings start to grow leaves.	There are five viviparous seedlings grow a pair of leaves.
General soil and general water	Viviparous seedlings grow about 0.4 to 0.6 centimeters of root.	Viviparous seedlings about 1.5 to 2 centimeters of root,end of bud begins to grow.	There are eight viviparous seedlings to sprout of phenomena.	There are eight viviparous seedlings start to grow leaves.	There are nine viviparous seedlings grow a pair of leaves.
Insert into water	Viviparous seedlings grow small roots at the end.	Viviparous seedlings grow small roots at the end, bud end almost no growth.	Viviparous seedlings grow small roots at the end, bud end almost no growth.	Viviparous seedlings grow small roots at the end, bud end almost no growth.	Viviparous seedlings grow small roots at the end, bud end begins to grow a little.
Sink into water	Viviparous seedlings grow small roots at the end.	Viviparous seedlings grow about 0.3 centimeters of small roots at the end.	Viviparous seedlings grow about 1 centimeters of small roots at the end.	Viviparous seedlings grow about 1.5 to 2 centimeters of small roots at the end but a few number of roots.	Viviparous seedlings grow about 1.5 to 2 centimeters of small roots at the end but a few number of roots.

Table 2.2: Process of growth of Kandelia all the year round

Month	Observed Results	Note
Last year of February	Hypocotyls of Kandelia grow about eighteen to twenty-five centimeters, at the end of February a little part of Kandelia began to fall. Kandelia grow new small bud like small rod.	Dropping length of hypocotyl short than the average hypocotyl about four centimeters.
March	Hypocotyls of Kandelia grow about eighteen to twenty-five centimeters, at the end of March most Kandelia began to fall and start defoliating. Small buds of Kandelia grow in a shape resembling a matchstick.	At the end of March almost all of hypocotyls of Kandelia have fallen
April	1. Previously falling Kandelia viviparous seedlings sprout almost together at the middle of April, we found Hypocotyl sprout has nothing to do with length,the main factor is to be fixed hypocotyl about 5-10 cm deep in sediment, hypocotyl not to be washed away or moved with tide and river. 2. Hypocotyls of Kandelia grow about twenty to twenty seven centimeters. Kandelia almost fall at the end of April. Kandelia and small rod small bud begin to grow into the shape of a small fire wood stick.	Kandelia (small rod of small bud) begin to grow into the shape of a small fire wood stick.
May	Kandelia viviparous seedlings grow root in two weeks and grow new leaves small rod small bud about 1 cm in two months.	A small part of bud will blossom at the end of May.
June	One part of Kandelia blossom and the other part of Kandelia is also bud.	The end of June is the peak of flowering.
July	All Kandelia blossom, star-shaped flowers with a fragrance and attract a large number of foraging.	Blossom ends in mid-July.
August	Brown seeds grow.	Size as peanuts.
September	Small fruits grow one centimeter in width and two centimeter in length, at the end of September all the seeds will develop a small hypocotyl.	Length of Viviparous seedlings vary.
October	Hypocotyl of Kandelia grow about one to eight centimeters.	Length of Viviparous seedlings vary.
November	Hypocotyl of Kandelia grow about eight to fifteen centimeters.	Length of Viviparous seedlings vary.
December	Hypocotyl of Kandelia grow about ten to twenty centimeters.	Length of Viviparous seedlings vary.
January	Hypocotyl of Kandelia grow about fifteen to twenty-two centimeters.	Length of Viviparous seedlings vary.
February	Hypocotyl of Kandelia grow about eighteen to twenty-five centimeters.	Length of Viviparous seedlings vary.

Figure 2.1: Some pictures of process of growth in different month

2.2 Time and Height

Accroding to Section 2.1, we know the growth of hypocotyl of Kandelia weekly and monthly. We will reasonably assume the height of growth each weeks. In addition, we give restriction to height of growth each months from September to February. Further data is in Table 2.3 and relations are shown in Figure 2.2.

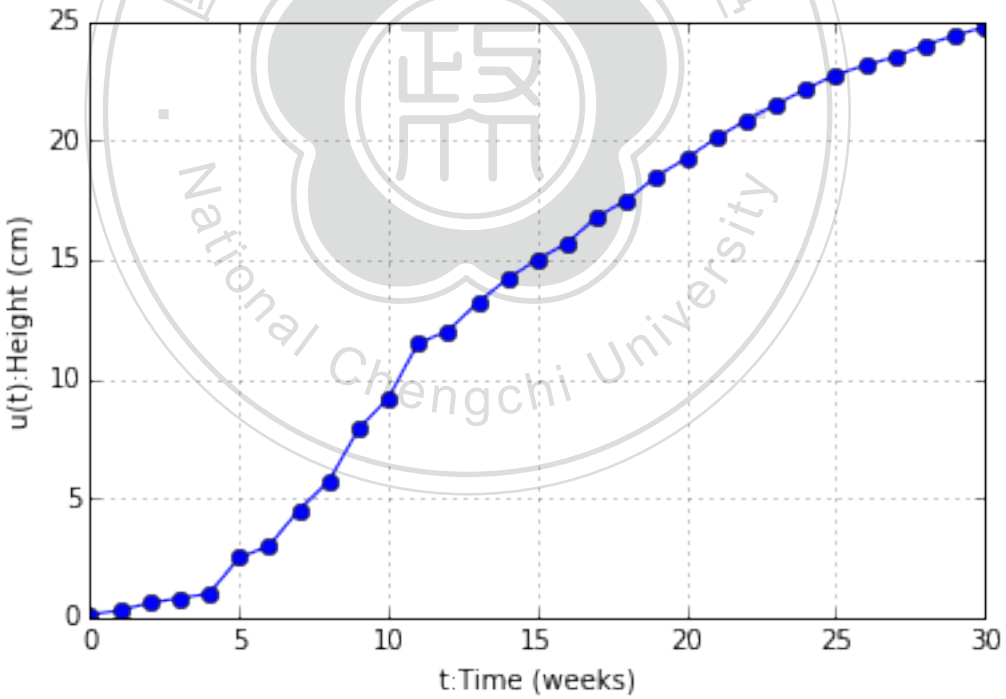


Figure 2.2: Graph of time and height

Table 2.3: Relation of time and height

Time	Period	Height	Time	Period	Height
	t	$u(t)$		t	$u(t)$
Initial	0	0	January first week	17	15.7
September first week	1	0.1	January second week	18	16.8
September second week	2	0.3	January third week	19	17.5
September third week	3	0.6	January fourth week	20	18.5
September fourth week	4	0.8	February first week	21	19.25
October first week	5	1	February second week	22	20.1
October second week	6	2.5	February third week	23	20.85
October third week	7	3	February fourth week	24	21.5
October fourth week	8	4.5	March first week	25	22.15
November first week	9	5.7	March second week	26	22.75
November second week	10	7.9	March third week	27	23.15
November third week	11	9.2	March fourth week	28	23.5
November fourth week	12	11.5	April first week	29	24
December first week	13	12	April second week	30	24.4
December second week	14	13.2	April third week	31	24.75
December third week	15	14.2	April fourth week	32	25
December fourth week	16	15			

Chapter 3

Construct Model Function

In this chapter, we first introduce logistic curve in section 3.1. Then we construct a model function by logistic curve in section 3.2. Also,we improve this model in section 3.3. Whether we use logistic model or the improved model, there are many unknown parameters we need to find. Therefore, in section 3.4 we will introduce Genetic algorithm.

3.1 Fit by Logistic Curve

We begin by observing the data in Figure 2.2. Notice that these data is increasing under time. The increasing rate grows slow from beginning. Then, it becomes faster in the middle. After that, it slow down again. The performs of these data is quite similar to logistic curve [4].

Thus we give logistic curve an introduction here:

A logistic function or logistic curve is a common "S" shape (sigmoid curve), with equation 3.2.1

$$Q(t) = \frac{B}{1 + Ae^{-Bkt}}$$

where

- *A , B and k are constants*
- *e = the natural logarithm base (also known as Euler's number)*

The function was named in 1844–1845 by Pierre François Verhulst, who studied it in relation to population growth. The initial stage of growth is approximately exponential; then, as saturation begins, the growth slows, and at maturity, growth stops.

The logistic function finds applications in a range of fields, including artificial neural networks, biology (especially ecology), biomathematics, chemistry, demography, economics, geoscience, mathematical psychology, probability, sociology, political science, linguistics, and statistics.

3.2 Model 1

We begin our modeling by using logistic curve mentioned in section 3.1. That is, we are going to use the data in Table 2.3 as the value of $Q(t)$. We want to find parameters A, B, and k which fit the following equation:

$$Q(t) = \frac{B}{1 + Ae^{-Bkt}} \tag{3.2.1}$$

By using Matlab curve fitting tool, we then find A , B and k as follow:

- $A = 25.8$
- $B = 24.21$
- $k = 0.009621$

Substituted the above parameters into equation 3.2.1, we have result shown in Figure 3.1.

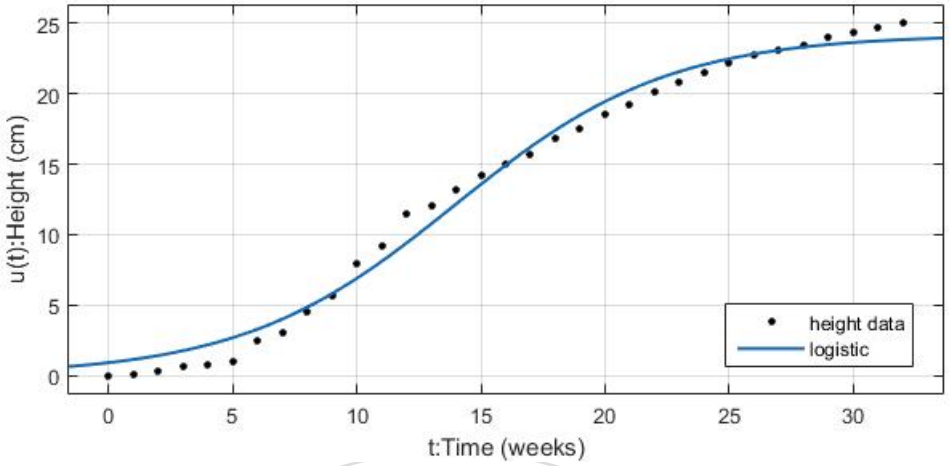


Figure 3.1: Graph of data and equation 3.2.1

By this method, the sum of square error (SSE) is "5.38952958". Overall, this is not the ideal result we want. Thus, a further improvement will be done in next section.

3.3 Model 2

We now observe the curve in Figure 3.1. The value of model function is higher then the data from week 0 to week 8. Then from week 9 to week 16, its value is lower then the data. Next, from week 17 to week 24 it goes higher again. After that, it goes lower from week 25 to week 32. The behavior of this model function seems to be different from the original data by adding a periodic factor. Thus, in order to decrease the SSE, we now consider equation 3.3.1 as our new model:

$$Q(t) = \frac{B}{1 + Ae^{-Bkt}} + asin(bt + c) \tag{3.3.1}$$

Similar to section 3.2, we use the data in Table 2.3 as the value of $Q(t)$ and find parameters a, b, c, A, B, and k which fit the following equation well. By using curve fitting tool again, we have the parameters as below:

- $A = 29.83$
- $B = 23.79$
- $k = 0.01033$

- $a = 1.233$
- $b = 0.3468$
- $c = 8.948$

We substitute the parameters above into equation 3.3.1. The result is shown in Figure 3.2.

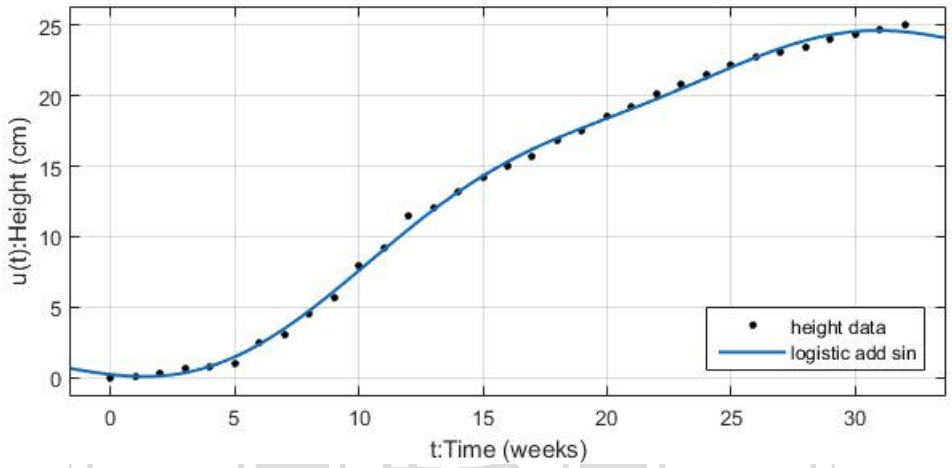


Figure 3.2: Graph of data and (3.3.1)

By this method, the SSE is "4.92591951". That is a pretty good result. Also, compare this result to the one in section 3.2, we have:

$$1 - \frac{4.92591951}{5.38952958} \approx 0.08602$$

which means we reduced about 8.6% of sum of square error.

The model we built here seems to fit pretty well. However, its actually over fit our data. To be more specific, we take a closer look at equation 3.3.1 again. As the time keeps going, we plot the graph of prediction:

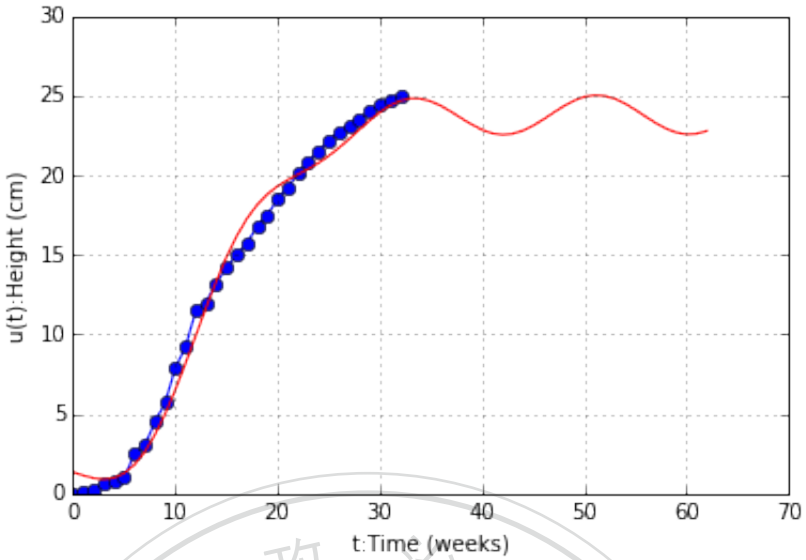


Figure 3.3: Predict of Kandelia's height by equation 3.3.1

In Figure 3.3. We see the curve remains oscillate after $t = 30$. Since this is a data from height of Kandelia, this kind of behavior is not reasonable. In the real world, the height is more likely to keep growing. Also, it will end up approaching to a stable value. Therefore, we will try improving this part in model later.

3.4 Genetic Algorithm

We are curious about how Matlab curve fitting tools find the parameters of models in section 3.2 and section 3.3. With some survey, we find it works similar to the process of genetic algorithm [6]. Thus, we will give an introduction to genetic algorithm here. This algorithm will be used frequently in the follow up chapter.

> *In computer science and operations research, a genetic algorithm (GA) is a metaheuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA). Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on bio-inspired operators such as mutation, crossover and selection.*

Now given data $(x_i, y_i)_{i=0}^{n}$. We construct a model function $y = f(a_1, a_2, ..., a_m, x)$, where $a_1, ..., a_m$ are the parameters and $x \in [x_0, x_n]$. Our goal is to find a set of $a_1, ..., a_m$, such that the function with these parameters has a relative small sum of square error with respect to the data.

Give an initial set $a_1^{(0)}, ..., a_m^{(0)}$. With these parameters, we can compute the sum of square error $SSE^{(0)}$. Let $a_i = a_i^{(0)}$, $i = 1...m$, and $SSE = SSE^{(0)}$

Next, we give each a_i ,$i = 1...m$, some perturbation and get a set of new parameters $a_1^{(1)}, ..., a_m^{(1)}$. Again, we can compute another sum of square error $SSE^{(1)}$. Compare SSE and $SSE^{(1)}$:

- case1:$SSE > SSE^{(1)}$

  In this case, we replace each a_i by $a_i^{(1)}$ and replace SSE by $SSE^{(1)}$. Denote by:

  $a_i = a_i^{(1)}$ for $i = 1...m$

  $SSE = SSE^{(1)}$

- case2:$SSE \leq SSE^{(1)}$

  Here we do nothing

Again we give our new $a_1, ..., a_m$ perturbation and get $a_1^{(2)}, ..., a_m^{(2)}$. As before, we find the corresponding sum of square error $SSE^{(2)}$. Similar to the former two cases, we renew a_i for $i = 1...m$ and SSE.

Finally, repeating above steps, we can find a set of parameters $a_1, ..., a_m$ which corresponding SSE is relative small.

Chapter 4

Construct Differential Equation Model

In this chapter, we want to look for a differential equation as our model. A reference of R-f's paper is shown in section 4.1. Then, in section 4.2, we will improve his method of estimating Kandelia's growth velocity. Next, in section 4.3, we construct a model by changing Logistic function into form of first order differential equation. After that, in section 4.4, we improve this model by adding a periodic factor.

4.1 Survey from Ren-fa's Paper

We want to find a differential equation model. To do that, we follow R-f's paper [7]. Here is the idea. First, we find growth velocity of Kandelia from the data of its height. Next, we use Kandelia's growth velocity to find acceleration rate of it. Thus, we have the acceleration rate of Kandelia in different height. In other word, we can plot a graph with Kandelia's height as x axis and Kandelia's acceleration rate as y axis. Our goal is to find an equation that fits this graph. We give a summary of R-f's work:

In this paper, we first calculate the first order differential value of the data in Table 2.3 by using forward difference approximation. These data will represent the growth velocity of Kandelia in different height:

$$h'(t_i) \approx \frac{h(t_{i+1}) - h(t_i)}{t_{i+1} - t_i} \tag{4.1.1}$$

Here, $h'(t_i)$ is the approximation of Kandelia's growth velocity at time t_i, and $h(t_i)$ is the height of Kandelia at time t_i. The result is shown in Table 4.1:

Table 4.1: Relation between velocity of growth and height of hypocotyl of Kandelia

Period	Height	Velocity of growth	Period	Height	Velocity of growth
0	0	0.1	17	15.7	1.1
1	0.1	0.2	18	16.8	0.7
2	0.3	0.3	19	17.5	1.0
3	0.6	0.2	20	18.5	0.75
4	0.8	0.2	21	19.25	0.85
5	1	1.5	22	20.1	0.75
6	2.5	0.5	23	20.85	0.65
7	3	1.5	24	21.5	0.55
8	4.5	1.2	25	22.15	0.6
9	5.7	2.2	26	22.75	0.15
10	7.9	1.3	27	23.15	0.4
11	9.2	2.3	28	23.5	0.35
12	11.5	0.5	29	24	0.4
13	12	1.2	30	24.4	0.35
14	13.2	1.0	31	24.75	0.25
15	14.2	0.8	32	25	
16	15	0.7			

Next, we use forward difference approximation on Table 4.1. Thus, we can calculate the second order differential value:

$$h''(t_i) \approx \frac{h'(t_{i+1}) - h'(t_i)}{t_{i+1} - t_i} \tag{4.1.2}$$

Here, $h''(t_i)$ is the growth acceleration of Kandelia at time t_i, and $h'(t_i)$ is the growth velocity of Kandelia at time t_i. The result is shown in Table 4.2.

Table 4.2: Relation between acceleration of growth and height of hypocotyl of Kandelia

Period	Height	Acceleration of growth	Period	Height	Acceleration of growth
0	0	0.1	17	15.7	-0.4
1	0.1	0.1	18	16.8	0.3
2	0.3	-0.1	19	17.5	-0.25
3	0.6	0	20	18.5	0.1
4	0.8	1.3	21	19.25	-0.1
5	1	-1	22	20.1	-0.1
6	2.5	1	23	20.85	0
7	3	-0.3	24	21.5	-0.05
8	4.5	1	25	22.15	-0.2
9	5.7	-0.9	26	22.75	-0.05
10	7.9	1	27	23.15	0.15
11	9.2	-1.8	28	23.5	-0.1
12	11.5	0.7	29	24	-0.05
13	12	-0.2	30	24.4	-0.1
14	13.2	-0.2	31	24.75	
15	14.2	-0.1	32	25	
16	15	0.4			

Once we have the data in Table 4.2 and Table 2.3, we get the relation between second order differential data and original data. We then have a graph with Kandelia's height as x axis and its acceleration rate as y axis. It is shown as in Figure 4.1. We want to find a curve to fit it.

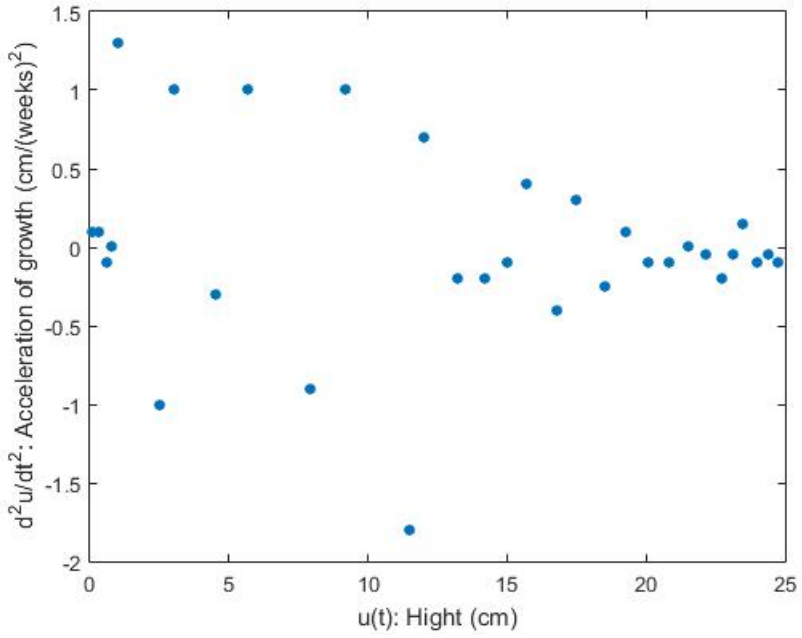


Figure 4.1: Graph of Table 2.3

In R-f's paper, we conclude that equation 4.1.3 is the best mathematical model fitting Figure 4.1:

$$h''(t) = a_1 sin(b_1 h(t) + c_1) + a_2 sin(b_2 h(t) + c_2) + a_3 sin(b_3 h(t) + c_3) \tag{4.1.3}$$

It is a second order differential equation with sum of *sin* functions. We want to find suitable parameters: a_i, b_i, and c_i $(i = 1, 2, 3)$. By using matlab curve fitting tool, we find result as below:

- $a_1 = 0.4573$
- $b_1 = 3.497$
- $c_1 = -1.402$
- $a_2 = 0.3126$
- $b_2 = 2.662$
- $c_2 = 1.053$
- $a_3 = 0.3544$
- $b_3 = 2.424$
- $c_3 = 2.24$

We then substitute above parameters into equation 4.1.3. The result is shown in Figure 4.2.

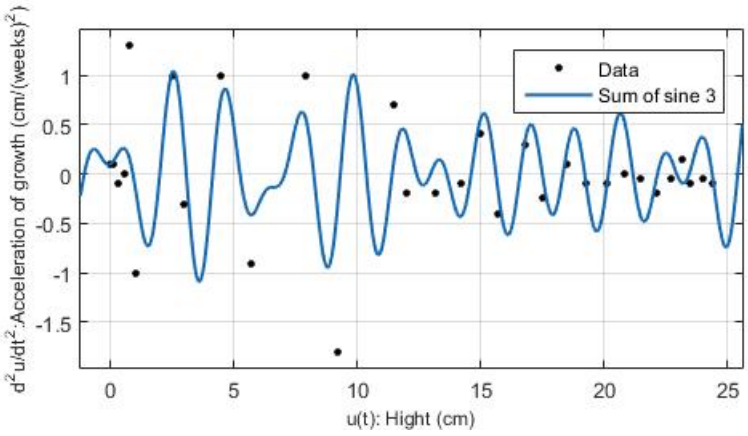


Figure 4.2: Graph of Table 4.2 and equation4.1.3

Next, we try to find an analytical solution of equation 4.1.3. The result is as follow:

If $h(t)$ is the solution of $h''(t) = \sum_{i=1}^{3} a_i \sin\left(b_i h(t) + c_i\right)$ and $t_0 = 0$, $h(t_0) > 0$, then we have :

1.

$$\frac{1}{2}h'(t)^2 + \sum_{i=1}^{3} \frac{a_i}{b_i} \cos\left(b_i h(t) + c_i\right) = E(t). \tag{4.1.4}$$

2.

$$h'(t) = \sqrt{2\left(E - \sum_{i=1}^{3} \frac{a_i}{b_i} \cos\left(b_i h(t) + c_i\right)\right)}, \tag{4.1.5}$$

Here, $E = E(t)$ is a constant. Also, a_i, b_i and c_i are constants. We find out R-f's paper end up here. However, we are curious about the solution of equation 4.1.5. Therefore, we use numerical method to find a solution of it. Here is how it works:

Since it has been proved that $E(t)$ is a constant. Our first step is to find the value of it. We use data in Table 4.1. Also, we substitute Kandelia's velocity into $h'(t)$ and its corresponding height into $h(t)$. Thus, we have a value of $E(t)$ in equation 4.1.4 as follow:

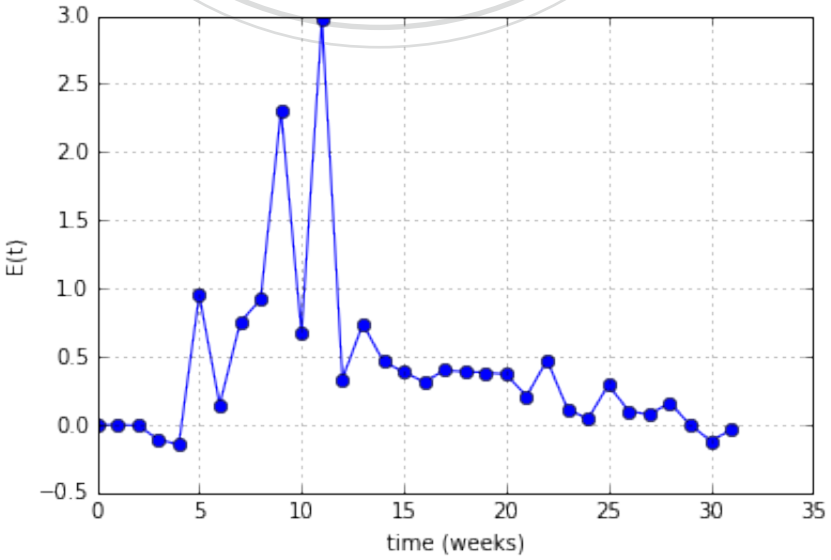


Figure 4.3: Graph of $E(t)$

We take a closer look at Figure 4.3. We see that $E(t)$ is stable at $t > 12$. Thus, we take average of them and we have E = ”0.266860291408”. Next, we use RK4 method to solve equation 4.1.5. The result is shown below:

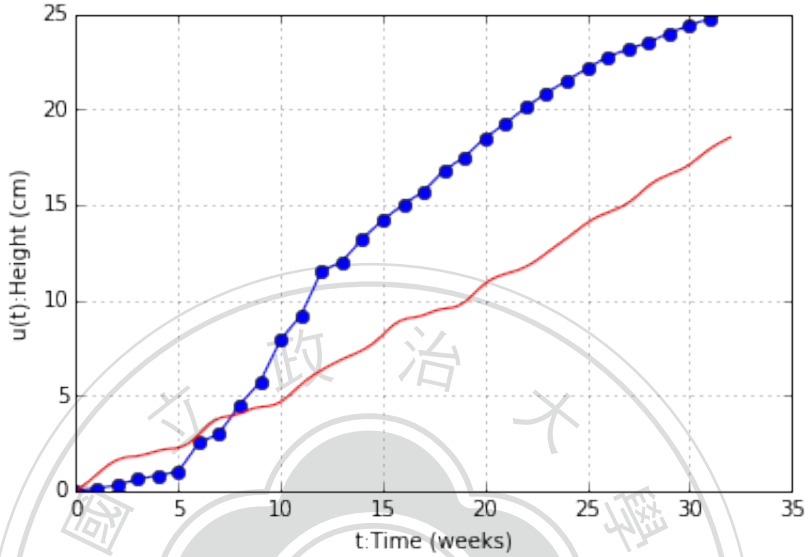


Figure 4.4: Numerical result of equation 4.1.5

The blue line in Figure 4.4 is the original data of Kandelia's height, and the red line is our fitting curve. We compute the sum of square error and find SSE = “74.3052894517”.

4.2 First Order Differential of the Data

Recall that in R-f's paper, we try to find growth velocity of Kendelia by the height of it. The method we used there is forward difference approximation [5]. That is, Given function $f \in C^1 : \mathbb{R} \to \mathbb{R}$, $x \in \mathbb{R}$ and t be a constant. Then, we can approximate the differential value of f at point x with the step size t as follow:

$$D_f(x,t) = \frac{f(x+t) - f(x)}{t}$$

Notice that, if $f'(x)$ is the real value. Then, this approximation will have first order big O error. That is, we have: $D_f(x,t) = f'(x) + O(t)$.

We want to decrease the error. To do that, we generate the data by central difference approx-

imation [5]. Again, given function $f \in C^1 : \mathbb{R} \to \mathbb{R}$, $x \in \mathbb{R}$ and t be a constant. Then we can approximate the differential value of f at point x with the step size t as follow:

$$D_f(x,t) = \frac{f(x+t) - f(x-t)}{2t}$$

Now, if $f'(x)$ is the real value. Then this approximation will have second order big O error. That is, we have: $D_f(x,t) = f'(x) + O^2(t)$. This is better than forward difference approximation.

Now, we let $h(t_i)$ be Kandelia's height at the time t_i. We will use central difference approximation to generate Kandelia's growth velocity:

$$h'(t_i) = \frac{h(t_{i+1}) - h(t_{i-1})}{t_{i+1} - t_{i-1}} \tag{4.2.1}$$

Here, $h'(t_i)$ is Kandelia's growth velocity we approximate at the time t_i. Result is shown in Table 4.3. We will use these data in follow up sections to construct our mathematical models.

Table 4.3: Relation between velocity of growth and height of hypocotyl of Kandelia

Period	Height	Velocity of growth	Period	Height	Velocity of growth
0	0		17	15.7	0.9
1	0.1	0.1 5	18	16.8	0.9
2	0.3	0.25	19	17.5	0.85
3	0.6	0.25	20	18.5	0.875
4	0.8	0.2	21	19.25	0.8
5	1	0.85	22	20.1	0.8
6	2.5	1.0	23	20.85	0.7
7	3	1.0	24	21.5	0.65
8	4.5	1.35	25	22.15	0.625
9	5.7	1.7	26	22.75	0.5
10	7.9	1.75	27	23.15	0.375
11	9.2	1.8	28	23.5	0.425
12	11.5	1.4	29	24	0.45
13	12	0.85	30	24.4	0.375
14	13.2	1.1	31	24.75	0.3
15	14.2	0.9	32	25	
16	15	0.75			

4.3 Model 3

We will construct a differential equation model here. Different from what R-f did in his paper, we will do the curve fitting by using the data in Table 4.3 and Table 2.3. That is, we plot a graph with Kandelia's height as x axis and Kandelia's growth velocity as y axis. The goal is to find a model that fits this graph well. Here is how we do it. Recall the Logistic curve in equation 3.2.1:

$$Q(t) = \frac{B}{1 + Ae^{-Bkt}}$$

Here, $Q(t)$ will be the function of height of Kandelia, A, B and k are parameters. Next, we differentiate both sides of this equation. The left hand side will be kept as the growth velocity of Kandelia. On the right hand side, we will try to substitute it with $Q(t)$. It works as below:

$$\begin{aligned} Q'(t) &= \frac{AB^2ke^{-Bkt}}{(1 + Ae^{-Bkt})^2} \\ &= Q^2(t)(\frac{kB}{Q(t)} - k) \\ &= BkQ(t) - kQ^2(t) \end{aligned} \tag{4.3.1}$$

Since B and k are both parameters. Thus, we let $p_1 = Bk$ and $p_2 = -k$. Our goal is to find p_1 and p_2 which make equation 4.3.2 fits Table 4.3 the best.

$$Q'(t) = p_1Q^2(t) + p_2Q(t) \tag{4.3.2}$$

To do this we will use method by regression in subsection 4.3.1. Next, in subsection 4.3.2, we will use the finite difference method. Also, we will improve this method by method of genetic algorithm. At last, in subsection 4.3.3, we will use an analytical approach to find the solution. This will give us a standard to compare with.

4.3.1 Using Regression

In this subsection, we use regression [3] as the key method. This is a method to find the suitable parameters of a target function. Here, we use equation 4.3.2 as our target function:

$$Q'(t) = p_1 Q^2(t) + p_2 Q(t)$$

Our goal is to find suitable p_1 and p_2. Notice that this is a first order differential equation. So, we need the data of Kandelia's height versus Kendelia's velocity. We can get this data from Table 4.3. Here is how we do it. We will start from our target function:

Given data $\big(Q'(t_i), Q(t_i)\big), i = 0, \cdots, n$, we have:

$$Q'(t_i) = p_1 Q^2(t_i) + p_2 Q(t_i)$$

Here, $Q'(t_i)$ is Kandelia's growth velocity at time t_i and $Q(t_i)$ is Kandelia's height as time t_i. To find p_1 and p_2, we will construct a linear system from it:

$$\begin{pmatrix} Q^2(t_0) & Q(t_0) \\ Q^2(t_1) & Q(t_1) \\ \vdots & \vdots \\ Q^2(t_n) & Q(t_n) \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} = \begin{pmatrix} Q'(t_0) \\ Q'(t_1) \\ \vdots \\ Q(t_n) \end{pmatrix}$$

We denote the above linear system by $Ax = b$. Using regression, we product A^T to both side of the equation. That is, $A^T Ax = A^T b$:

$$\begin{pmatrix} Q^2(t_0) & Q^2(t_0) & \cdots & Q^2(t_n) \\ Q(t_0) & Q(t_0) & \cdots & Q(t_n) \end{pmatrix} \begin{pmatrix} Q^2(t_0) & Q(t_0) \\ Q^2(t_1) & Q(t_1) \\ \vdots & \vdots \\ Q^2(t_n) & Q(t_n) \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \end{pmatrix}$$

$$= \begin{pmatrix} Q^2(t_0) & Q^2(t_1) & \cdots & Q^2(t_n) \\ Q(t_0) & Q(t_1) & \cdots & Q(t_n) \end{pmatrix} \begin{pmatrix} Q(t_0) \\ Q(t_1) \\ \vdots \\ Q(t_n) \end{pmatrix}$$

We substitute $\big(Q'(t_i), Q(t_i)\big), i = 0, \cdots, n$ by the value in Table 4.3. The solution to p_1 and p_2 is:

- $p_1 = -0.00863605$
- $p_2 = 0.21215756$

We substitute above p_1 and p_2 into equation 4.3.2. Then, we can plot a graph with Kandelia's height as its x axis and Kandelia's growth velocity as its y axis. The result is shown in Figure 4.5. The blue line is our original data, and the red line is our fitting curve.

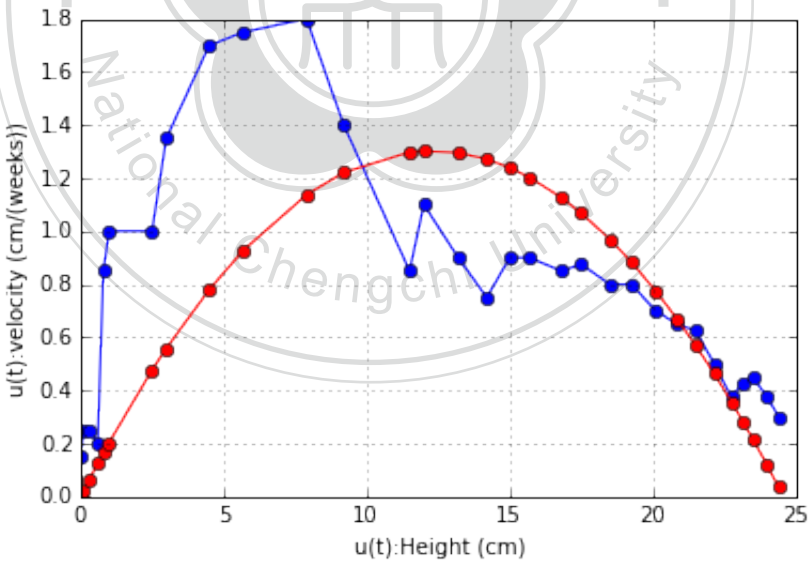


Figure 4.5: Fit the data by regression

Using the above p_1 and p_2 together with the initial value in Table 2.3, we find the differential equation which fits the data of Kandelia's velocity. We want to know whether the solution of it fit Kandelia's height or not. Thus, using Heun's method, RK2 method and RK4 method, we have the following result:

- SSE solved by Heun's method = 6.3916710696
- SSE solved by RK2 method = 6.39166483931
- SSE solved by RK4 method = 6.39165419709

The best result solved by this method is shown in Figure 4.6. The blue line is the data of Kandelia's height, and the red line is our fitting curve.

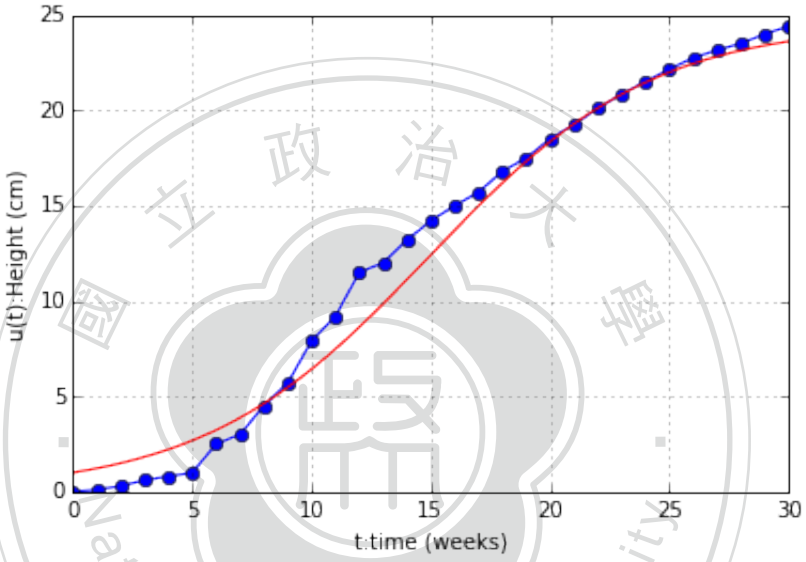


Figure 4.6: Best result

Notice that the best result we find in section 3.3 is $SSE =$ "4.92591951". It is better than the best result here: $SSE =$ "6.3916". Thus, we will improve our model in the follow up subsection.

4.3.2 Using Finite Difference Method

In this subsection, we will use three steps to find the suitable parameters of equation 4.3.2:

$$Q'(t_i) = p_1 Q^2(t_i) + p_2 Q(t_i)$$

First of all, we construct the target function from equation 4.3.2. But different from section 4.3.1, we will use the finite difference method here. By this method, we can substitute Kandelia's growth velocity into terms of Kandelia's height. This gives us a new model with

time and Kandelia's height only. Thus, we use regression with the data in Table 2.3. We then find suitable parameters for this model.

Secondly, we use genetic algorithm to find suitable initial value. That is, with the parameters above, we give the initial value some perturbation. Comparing the outcome via different initial value, we find a better result.

Finally, we use genetic algorithm on the above model again. But this time, we use it not only on the initial value, but also on the parameters of the model. To be more precise, we first find an initial value and a set of parameters. Then we give the initial value some perturbation and get a new initial value. With this new initial value, we give parameters perturbation. Repeating this process, we will have a better outcome. The follow up is how it all works:

From Table 2.3, we have data $\{(Q(t_i), t_i)\}_{i=1}^{n}$, where $Q(t_i)$ is the hight of Kandelia at time t_i. Our model start from equation 4.3.2:

$$Q'(t_i) = p_1 Q^2(t_i) + p_2 Q(t_i)$$

Where $Q'(t)$ is Kandelia's growth velocity. Now, instead of using data in Table 4.3 on $Q'(t)$, we use finite difference method. In other word, we let $h_k = t_{k+1} - t_{k-1}$, $(k = 1, ..., n-1)$ and substituted $Q'(t_i)$ by $\frac{Q(t_{i+1}) - Q(t_{i-1})}{t_{i+1} - t_{i-1}}$. Thus, we have:

$$\begin{aligned} \frac{Q(t_{k+1}) - Q(t_{k-1})}{h_k} &= p_1 Q(t_k)^2 + p_2 Q(t_k) \\ \Rightarrow Q(t_{k+1}) &= Q(t_{k-1}) + h_k(p_1 Q(t_k)^2 + p_2 Q(t_k)) \end{aligned} \tag{4.3.3}$$

The equation 4.3.3 is our target function. Notice that, with p_1 and p_2 be the changing vari-

able, we can get the following linear system:

$$\begin{pmatrix} h_1Q(t_1)^2 & h_1Q(t_1) \\ h_2Q(t_2)^2 & h_2Q(t_2) \\ \vdots & \vdots \\ h_{n-1}Q(t_{n-1})^2 & h_{n-1}Q(t_{n-1}) \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} = \begin{pmatrix} Q(t_2) - Q(t_0) \\ Q(t_3) - Q(t_1) \\ \vdots \\ Q(t_n) - Q(t_{n-2}) \end{pmatrix}$$

Denote the above linear system by $Ax = b$, and we will use regression to find suitable p_1 and p_2. That is, we product A^T to both side of the equation. The solution of the linear system $A^TAx = A^Tb$ is as below:

$$\begin{pmatrix} h_1Q(t_1)^2 & h_2Q(t_2)^2 & \cdots & h_nQ(t_{n-1})^2 \\ h_1Q(t_1) & h_2Q(t_2) & \cdots & h_nQ(t_{n-1}) \end{pmatrix} \begin{pmatrix} h_1Q(t_1)^2 & h_1Q(t_1) \\ h_2Q(t_2)^2 & h_2Q(t_2) \\ \vdots & \vdots \\ h_nQ(t_{n-1})^2 & h_nQ(t_{n-1}) \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \end{pmatrix}$$

$$= \begin{pmatrix} h_1Q(t_1)^2 & h_2Q(t_2)^2 & \cdots & h_nQ(t_{n-1})^2 \\ h_1Q(t_1) & h_2Q(t_2) & \cdots & h_nQ(t_{n-1}) \end{pmatrix} \begin{pmatrix} Q(t_2) - Q(t_0) \\ Q(t_3) - Q(t_1) \\ \vdots \\ Q(t_n) - Q(t_{n-2}) \end{pmatrix}$$

Using the data in Table 2.3 ,and the code in appendix A.1, we can compute p_1 and p_2 which fit equation 4.3.3 well.

- $p_1 = -0.00844749$
- $p_2 = 0.21174495$

With the parameters above, we can solve equation 4.3.2 by Heun's method, RK2 method and RK4 method. The python code is also in appendix A.1.

- SSE solved by Heun's method = 55.7242174456
- SSE solved by RK2 method = 55.66572951

- SSE solved by RK4 method = 55.3013480532

The result above is not very good. In order to lower the SSE, we pull in the idea of genetic algorithm. That is, we add some perturbation to the initial value. Then we can get different curves by using equation 4.3.3 within different initial value. Further detail is written in appendix A.1. Overall, we will find a curve fits Table 2.3 relatively good. With the initial value of this curve and p_1 and p_2 above, we have the result:

- initial value = (0.18824931457, 0.369491784562)
- SSE solved by Heun's method = 30.2074082818
- SSE solved by RK2 method = 30.1259883441
- SSE solved by RK4 method = 29.8589927079

We have decrease the SSE from "55.3" to "29.8". But it is still not good enough. Thus we will now add some perturbation to p_1 and p_2 to lower the SSE. After that, we add some perturbation to the initial value as well. Repeating these two steps, we find an initial value and a set of parameters which are relatively good. We can understand this part better by checking out the Python code in appendix A.1. Result is as follow:

- adjusted p_1 = -0.00959911519495
- adjusted p_2 = 0.232405855098
- initial value = (0.990787303889, 1.07654767318)
- SSE solved by Heun's method = 5.6053053617
- SSE solved by RK2 method = 5.58681233538
- SSE solved by RK4 method = 5.533732902

The best result solved by this method is shown in Figure 4.7. The blue line is the original data of Kandelia's Height, and the red line is our fitting curve.

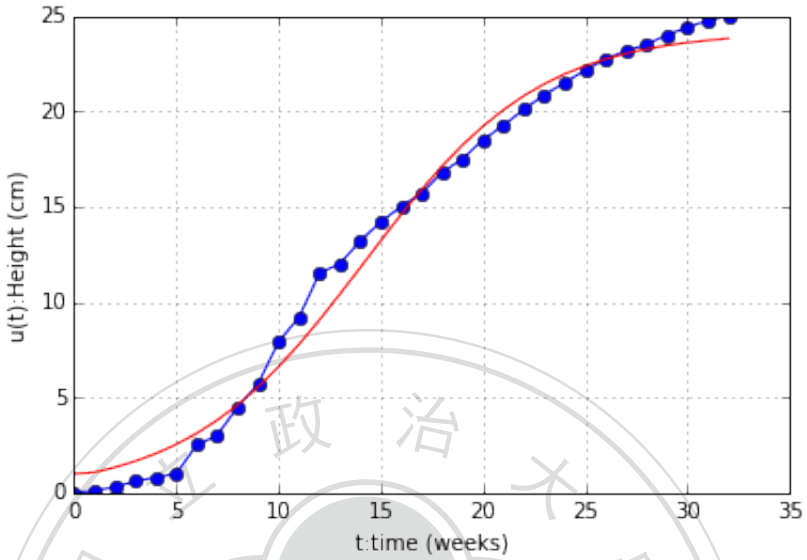


Figure 4.7: Best result

In conclusion, we have decrease the final SSE to "5.533". Compare to the result in former subsection $SSE = 6.3916$, we decrease the SSE by:

$$1 - \frac{5.533}{6.63916} \approx 0.16666$$

Which means we reduced about 16% of sum of square error. However we recall that in section 3.3, we build a model with $SSE =$ "4.92591951". It is still the best result until now. Thus we need to figure out some other method to decrease SSE.

4.3.3 Using Analytical Solution

Before moving on to the next section, we look at equation 4.3.2 again:

$$Q'(t) = p_1 Q^2(t) + p_2 Q(t)$$

We notice that this is the form of Bernoulli differential equation [2]. Thus, in this subsection, we will first use Matlab curve fitting tool to find suitable p_1 and p_2. Then, we will find

analytical solution of this equation. Here is how it works:

Similar to subsection 4.3.1, we are going to fit the graph with Kandelia's height as x axis and Kandelia's growth velocity as y axis. Using the model below, we then find p_1 and p_2 by Matlab curve fitting tool.

$$Q'(t_i) = p_1 Q^2(t_i) + p_2 Q(t_i)$$

Here, $(Q'(t_i), Q(t_i)), i = 0, \cdots, n$ are the data of Kandelia's velocity and Kandelia's height at time t_i. Suitable parameters we found are as follow. Also, we plot the result in Figure 4.8:

- p_1 = -0.008447
- p_2 = 0.2117

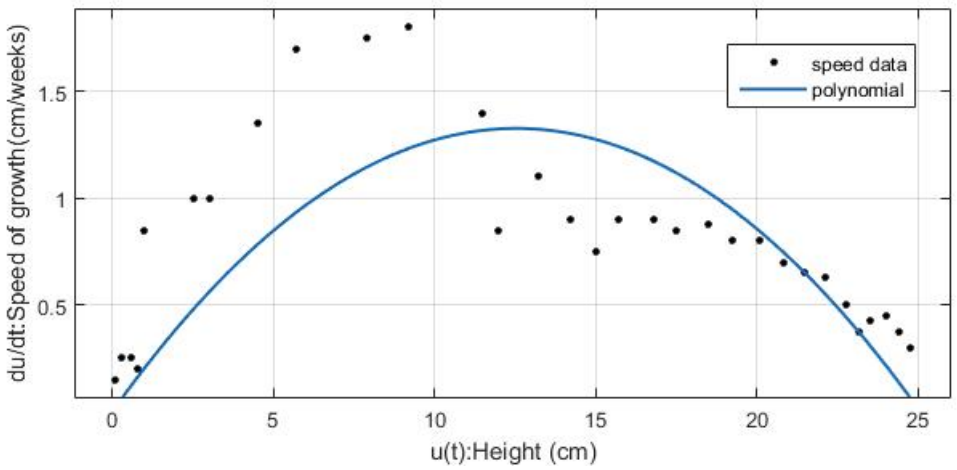


Figure 4.8: Graph of data and (4.3.1)

We will now find the analytical solution of equation 4.3.2:

$$\begin{aligned} Q'(t) &= p_1 Q^2(t) + p_2 Q(t) \\ \Rightarrow Q'(t) + (-p_2)Q(t) &= p_1 Q^2(t) \\ \Rightarrow Q^{-2}(t)Q'(t) + (-p_2)Q^{-1}(t) &= p_1 \end{aligned}$$

Here, we let $w(t) = Q^{-1}(t)$, then we have $w'(t) = -Q^{-2}(t)Q'(t)$. Substitute this into the above equation, we have the follow:

$$\begin{aligned} -w'(t) + (-p_2)w(t) &= p_1 \\ \Rightarrow w'(t) + p_2 w(t) &= -p_1 \end{aligned} \tag{4.3.4}$$

Equation 4.3.4 turned out to be a first order linear differential equation. Thus we solve it and substitute $Q(t)$ back into the solution. We then have the following result:

$$Q(t) = \frac{p_2 \times 10^6 \times e^{p_2 t}}{e^{(c_1 \times 10^6 \times p_2)} - p_1 \times 10^6 \times e^{p_2 t}} \tag{4.3.5}$$

Since p_1 and p_2 are already found. We now want to find suitable c_1 in equation 4.3.5. Again, using Matlab curve fitting tool, we have $c_1 = 5.717 \times 10^{-5}$. The result is shown in Figure 4.9

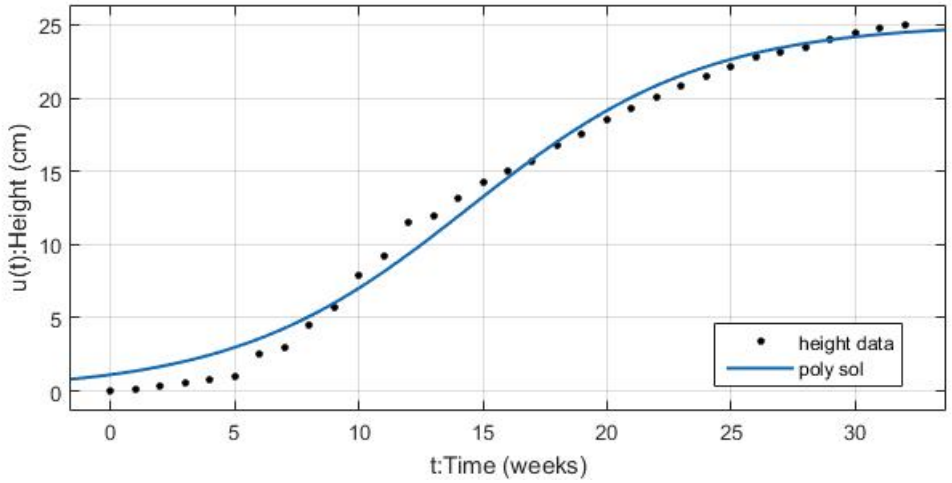


Figure 4.9: Graph of data and (4.3.5)

Overall, we have SSE = "5.646539" here. The result is almost the same as previous subsection SSE = "5.533733". This shows that the solution found by genetic algorithm is similar to the analytical solution.

4.4 Model 4

Until now, the best result in section 4.3 is SSE = '5.533733". However, we recall the result in section 3.3. There, SSE = "4.92591", which is still better. Therefore, in this section, we give equation 4.3.2 a furthermore correction.

We now give a closer examination on Figure 4.8. Starting from height in range 0 to 9, the data is higher than the fitting function. Next , in the range 10 to 20, the data is lower. After that, form range 21 to 25, it goes higher again. This difference seems like the behavior of a periodic factor. Thus, we build our new model by adding a *sin* term to equation 4.3.2. It is shown as follow:

$$Q'(t) = p_1Q^2(t) + p_2Q(t) + asin(bQ(t) + c) \tag{4.4.1}$$

Again, our goal is to find suitable p_1, p_2, a, b and c of equation 4.4.1. In subsection 4.4.1, we will use finite difference method. This substitutes term of $Q'(t)$ by $Q(t)$ in our model function. Then we can fit it by Table 2.3. Next, in subsection 4.4.2, we will fit equation 4.4.1 by genetic algorithm directly. Notice that Table 4.3 is used here to fit our model. At last, in subsection 4.4.3, we will use Matlab curve fitting tool to do the job. We will see the result is similar to subsection 4.4.2.

4.4.1 Using Finite Difference Method

In this subsection, our goal is to find suitable parameters of equation 4.4.1:

$$Q'(t) = p_1Q^2(t) + p_2Q(t) + asin(bQ(t) + c)$$

Here is how we do it. We first use the central difference method as our key ideal. That is, we will substitutes term of $Q'(t)$ by $Q(t)$ in the above model equation. The equation will turn out to be a function with $Q(t)$ and t. Thus, with some initial parameters, we can fit it by data of Kandelia's height.

Secondly, we use genetic algorithm to improve our model. That is, we give some perturbation to the initial value. Then, we compare the outcome depending on different initial value.

After that, we choose the best one.

Finally, we use genetic algorithm again. This time, we give some perturbation on parameters. With these new parameters, we give initial value some perturbation. Repeating these two process, we will at last get a set of parameters depend on an initial value.We will now give a further detail of how this all works:

Let $Q(t_i)$ be the data of Kandelia's height at time t_i , $i = 1, \cdots, n-1$. First, we substitute $Q'(t)$ in equation 4.4.1 by $\frac{Q(t_{i+1})-Q(t_{i-1})}{t_{i+1}-t_{i-1}}$

$$\begin{aligned} \frac{Q(t_{i+1}) - Q(t_{i-1})}{t_{i+1} - t_{i-1}} &= p_1 Q^2(t_i) + p_2 Q(t_i) + a\sin(bQ(t_i) + c) \\ \Rightarrow Q(t_{i+1}) &= Q(t_{i-1}) + (t_{i+1} - t_{i-1})(p_1 Q^2(t_i) + p_2 Q(t_i) + a\sin(bQ(t_i) + c) \end{aligned} \tag{4.4.2}$$

We first give a set of initial coefficient by referring the result in subsection 4.3.2:

- $p_1 = -0.00844749$
- $p_2 = 0.21174495$
- $a = 0.0$
- $b = 0.0$
- $c = 0.0$

Notice that these parameters can be chosen randomly. However, if the parameter is not chosen properly, the method by genetic algorithm may not work well. We will give a further discussion of this problem later. Anyway, by using parameters above we can solve equation 4.4.1 by Heun's method, RK2 method and RK4 method.

- SSE solved by Heun's method = 55.7242229592
- SSE solved by RK2 method = 55.6657350223
- SSE solved by RK4 method = 55.3013536756

From our experience before, the result is always not so good at the first step. In order to lower SSE, we add some perturbation to the initial value. Thus we can get different data by using equation 4.4.2 within different initial value. We then use our new initial value to solve equation 4.4.2. The result is as follow :

- adjusted initial value = (1.20182153439, 1.32129224963)
- SSE solved by Heun's method = 5.79278605472
- SSE solved by RK2 method = 5.7761112128
- SSE solved by RK4 method = 5.7418495602

The above result is pretty good. To lower SSE even more, we will now add some perturbation to p_1, p_2, a, b and c. After that, we add some perturbation to the initial value as well. Repeating these two steps, we hope to find best parameters and the initial value they depends on. We have the following result :

- adjusted p_1 = -0.00951510827755
- adjusted p_2 = 0.230182300243
- adjusted a = -0.0262259316043
- adjusted b = -0.0147144948651
- adjusted c = -0.00726659819372
- initial value = (1.01627076858, 1.10278819877)
- SSE solved by Heun's method = 5.58823184026
- SSE solved by RK2 method = 5.56985208097
- SSE solved by RK4 method = 5.51999849112

The best result solved by this method is shown in Figure 4.10. The blue line is our original data of Kandelia's height, and the red line is our fitting curve. Further detail of how this get from Python is in appendix A.3

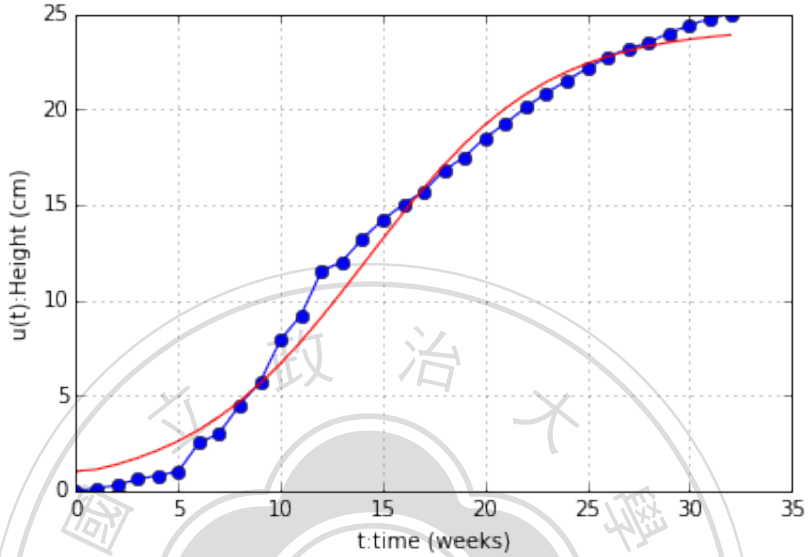


Figure 4.10: Best result

Until now our best result in this section is SSE = "5.519998". It slightly better than the result in section 4.3. There, SSE = "5.533733". Compare these two result:

$$1 - \frac{5.519998}{5.533733} \approx 0.002482$$

This means, we only decrease about 0.24% of SSE. Also, recall the result in section 3.3,we have SSE = "4.9259". It is still the best model until now. Therefore, we will try to improve it by using different method.

4.4.2 Using Genetic Algorithm

In this subsection, we again try to find parameters that fits the equation below:

$$Q'(t) = p_1 Q^2(t) + p_2 Q(t) + asin(bQ(t) + c)$$

Different from the previous subsection, we will not change the term of $Q'(t)$ here. Instead,

we will use the data of Kandelia's growth velocity as $Q'(t)$. Therefore, using Table 4.3, we can plot a graph with Kandelia's height as x axis and Kandelia's growth velocity as y axis. We then use the method of genetic algorithm to find suitable parameters. Here is how we do it:

First, we give a set of initial parameters. Notice that these parameters are chosen in purpose. We can see that in Figure 4.11 the equation generated by them won't fit so badly.

- $p_1 = -0.01$
- $p_2 = 0.2$
- $a = -0.5$
- $b = -0.3$
- $c = -0.1$

Here, the blue line represents the data of Kandelia's growth rate, and the red line is our fitting curve.

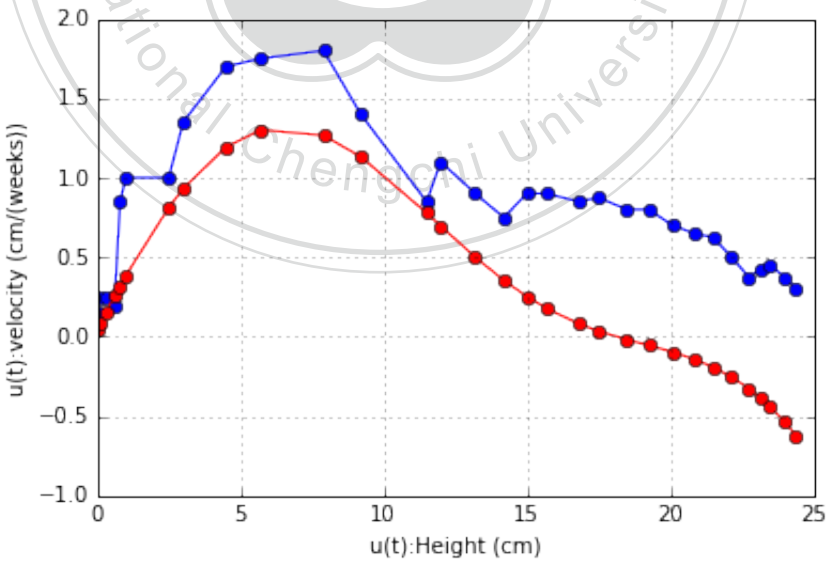


Figure 4.11: Fit the data by genetic algorithm

Next, in order to fit our data better, we add some perturbation to these parameters. Then, we compare the outcome of equation generated by different parameters. At last, we get the best result as follow. Also, we plot it out in Figure 4.12

- adjusted p_1 = -0.0114876114475
- adjusted p_2 = 0.262698012377
- adjusted a = -0.53344134757
- adjusted b = -0.318721070024
- adjusted c = -0.120528934405

Again, the blue line represents the data of Kandelia's growth rate, and the red line is our fitting curve.

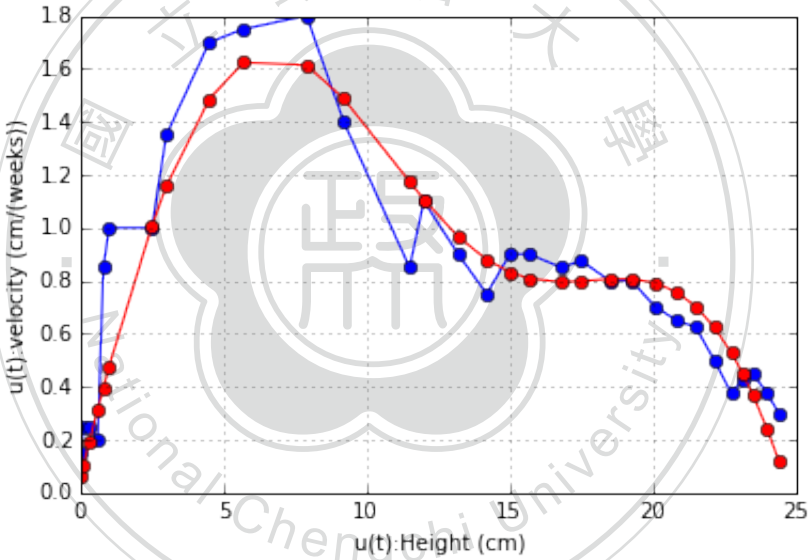


Figure 4.12: Fit the data by genetic algorithm with parameters adjusted

Finally, we solve equation 4.4.1 by Heun's method, RK2 method and RK4 method. After that, we give initial value some perturbation as well. This can help us fit the original data better. A further detail of python code used in this subsection is in appendix A.4.

- initial value for Heun's method= 0.0489081076939
- initial value for RK2 method= 0.0489704328867
- initial value for RK4 method= 0.0486656124813
- SSE solved by Heun's method = 1.68479162933

- SSE solved by RK2 method = 1.68477818706
- SSE solved by RK4 method = 1.68491878659

Final result is in Figure 4.13. Here, the blue line represents the data of Kandelia's height, and the red line is our fitting curve.

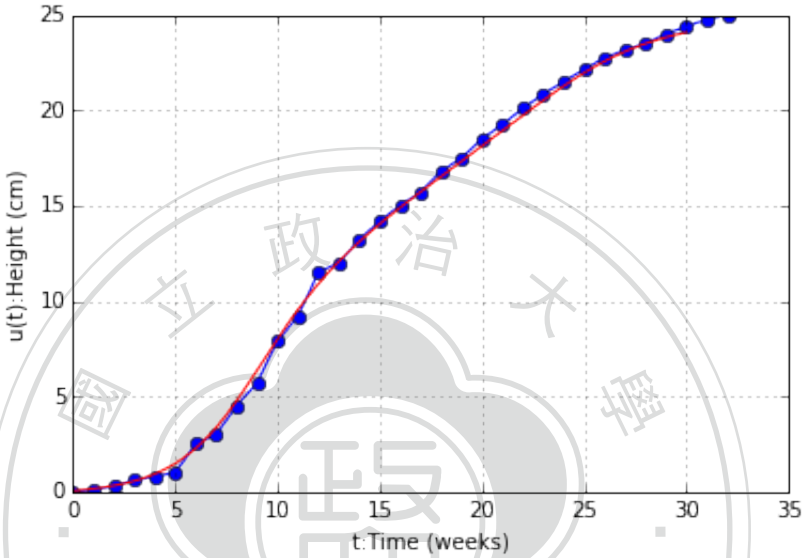


Figure 4.13: Best result

In conclusion, we have our best SSE = "1.68477818706". This is the best result until now. Thus, we compare this to SSE in section 3.3:

$$1 - \frac{1.684778}{4.9259} \approx 0.657975$$

This means, we decrease about 65% of SSE. Although we are satisfied with the SSE we get, we still have to check this model does not over fit the data. Thus, we solve equation 4.4.1 with RK4 method. But this time, we doubled the range of time and double the grids as well. The result is as follow:

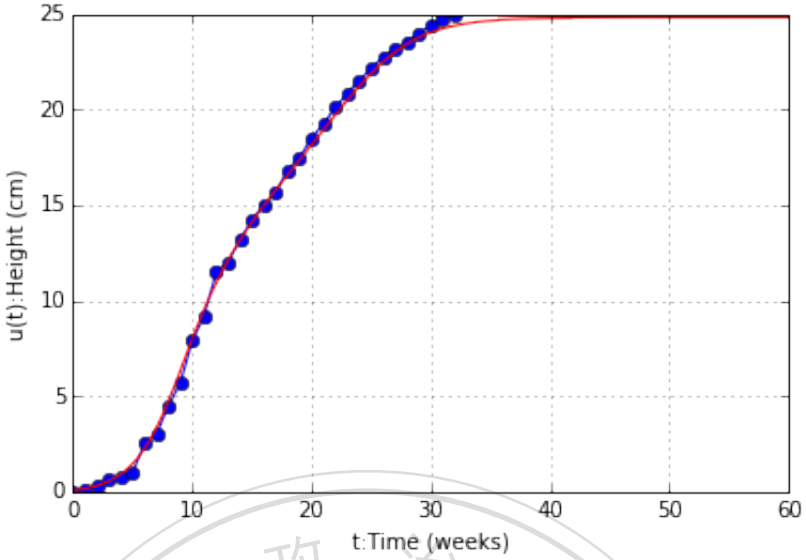


Figure 4.14: Predict of Kandelia's height by equation 4.4.1

In Figure 4.14, the blue line represents the original data of Kandelia's height, and the red line is our fitting curve. While $t > 30$, the fitting curve grows slowly and approaches stable. This is more reasonable to the real world. Therefore, we take this model as our final result.

4.4.3 Using Matlab Curve Fitting Tool

In this section, we will compare the result from genetic algorithm to the result using Matlab curve fitting tool. We want to show that they are similar. Thus, we again set equation 4.4.1 as our model:

$$Q'(t) = p_1 Q^2(t) + p_2 Q(t) + asin(bQ(t) + c)$$

Again, we use the data in Table 4.3 to construct a graph. Our goal is to find suitable parameters by Matlab curve fitting tool.

The corresponding parameters are as follow. Also, we have our result shown in Figure 4.15:

- p_1 = -0.011
- p_2 = 0.2601
- a = -0.5341
- b = -0.3017

- c = -0.1563

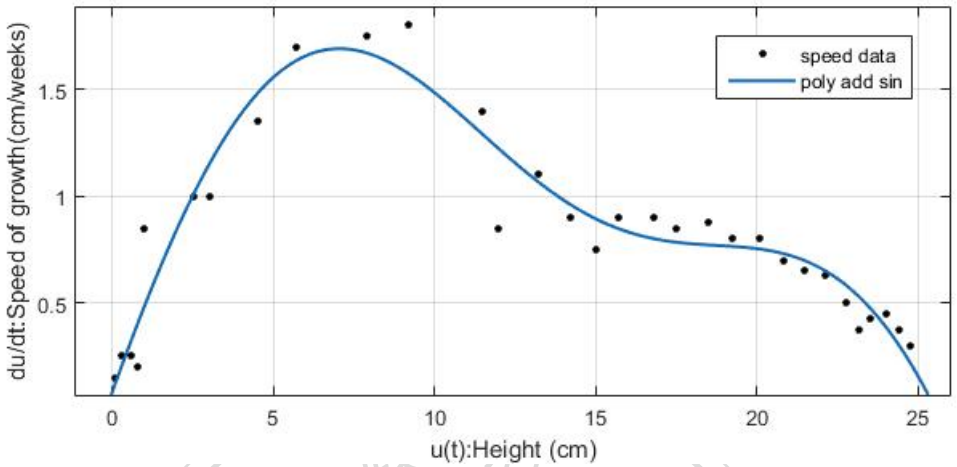


Figure 4.15: Graph of data and equation(4.4.1)

Using the parameters above, we can find a numerical solution of equation4.4.1 via Heun's method, RK2 method and RK4 method. Also we give the initial value some perturbation in order to decrease the SSE. Results are as below:

- initial value for Heun's method= -0.0151605021498
- initial value for RK2 method= -0.0153907948845
- initial value for RK4 method= -0.0152950386194
- SSE solved by Heun's method = 2.12606450927
- SSE solved by RK2 method = 2.12599915696
- SSE solved by RK4 method = 2.12601296975

The best result solved by this method is shown in Figure 4.16. Here, the blue line represents the data of Kandelia's height, and the red line is our fitting curve.

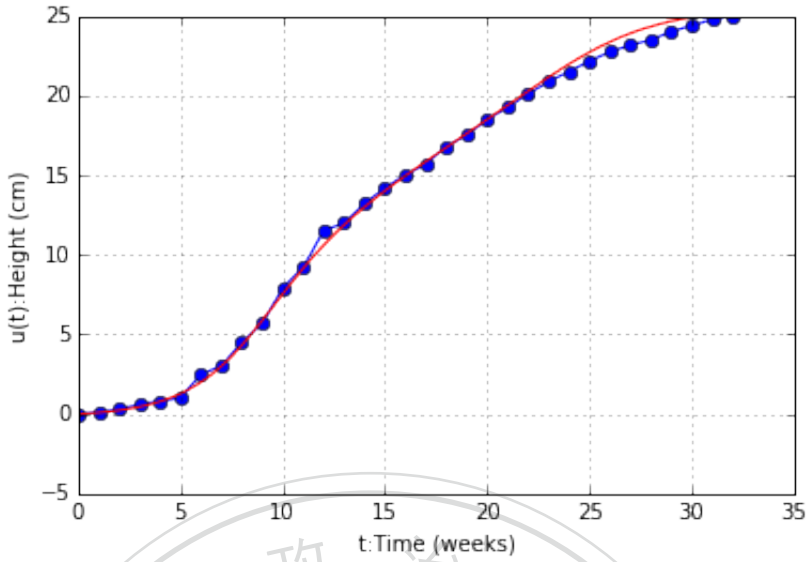


Figure 4.16: Best result

Overall, we have SSE = "2.215999" . It is slightly worse than the result in previous subsection (SSE = "1.68477818706"). However, the result here is the second best in this paper. Also, we notice that parameters we found by Matlab curve fitting tool is similar to those we found in subsection 4.4.2.

Chapter 5

Conclusion

We sort out all the mathematical models in table 5.1. Here is a quick review of what we have done in this paper:

In chapter 3, we built two model functions. In section 3.2 the model is constructed by logistic curve. There, SSE = "5.38952958". Then, in section 3.3 we improve this model by adding a *sin* function. By doing so, we reduced SSE to "4.92591951". It means, we decrease about 8.6% of first SSE.

Next in chapter 4, we built two model of differential equation. In section 4.3, we construct a model from differential form of logistic curve. There, we have outcome SSE = "5.533733", which is not a good one. However, in section 4.4, we improve the model equation by adding a *sin* function. Then, we reduced SSE to "1.68477819". That is, we decrease about 65% of SSE in section 3.3.

In conclusion, we find out the result is the best with following three condition:

- fit data: $(Q'(t), Q(t))$
- fit model: $Q'(t) = p_1Q^2(t) + p_2Q(t) + asin(bQ(t_i) + c)$
- genetic algorithm as our fit method

However, this method also has its disadvantages. Like the Matlab curve fitting tool, if the model function with initial parameters has a large SSE with original data. Then the parameters

may probably not converge. Therefore, how to find a suitable initial parameters may be a good question for further research.

Table 5.1: Conclusion

Section	Fit model	Fit Method		SSE
Section 3.2	$Q(t) = \frac{B}{1+Ae^{-Bkt}}$	Matalb curve fitting tool		5.38952958
Section 3.3	$Q(t) = \frac{B}{1+Ae^{-Bkt}} + asin(bt + c)$	Matalb curve fitting tool		4.92591951
Section 4.3.1	$Q'(t) = p_1Q^2(t) + p_2Q(t)$	Regression	Heun's	6.39167107
			RK2	6.39166484
			RK4	6.39165420
Section 4.3.2	$\frac{Q(t_{i+1})-Q(t_{i-1})}{t_{i+1}-t_{i-1}} = p_1Q^2(t_i) + p_2Q(t_i)$	Finite difference	Heun's	55.7242174
			RK2	55.6657295
			RK4	55.3013481
		Initial value adjust	Heun's	30.2074083
			RK2	30.1259883
			RK4	29.8589927
		Coefficient adjust	Heun's	5.605305
			RK2	5.586812
			RK4	5.533733
Section 4.3.3	$Q'(t) = p_1Q^2(t) + p_2Q(t)$	Analytical Solution		5.646539
Section 4.4.1	$\frac{Q(t_{i+1})-Q(t_{i-1})}{t_{i+1}-t_{i-1}} = p_1Q^2(t_i) + p_2Q(t_i) + asin(bQ(t_i) + c)$	Finite difference	Heun's	55.724230
			RK2	55.665735
			RK4	55.301354
		Initial value adjust	Heun's	5.7927861
			RK2	5.7761112
			RK4	5.7418496
		Coefficient adjust	Heun's	5.58823184
			RK2	5.56985208
			RK4	5.51999849
Section 4.4.2	$Q'(t) = p_1Q^2(t) + p_2Q(t) + asin(bQ(t_i) + c)$	Genetic algorithm	Heun's	1.68479163
			RK2	1.68477819
			RK4	1.68491879
Section 4.4.3	$Q'(t) = p_1Q^2(t) + p_2Q(t) + asin(bQ(t) + c)$	Curve fitting tool	Heun's	2.12606451
			RK2	2.12599916
			RK4	2.12601297

Appendix A

Code use in paper

A.1 Code 01

In[1]:

```
%pylab inline
```

Some Tools

Heun's Method

In[2]:

```
def ode_Heun(f,t0,tf,y0,y1,n):
    t = np.linspace(t0,tf,n)
    y = list([y0,y1])
    for i in range(1,n-1):
        h = t[i+1]-t[i]
        fk = f(t[i],y[-1])
        fk1 = f(t[i+1],y[-1]+h*fk)
        y.append(y[-1]+h*(fk+fk1)/2.0)
    return y
```

RK2 Method

In[3]:

```
def ode_RK2(f,t0,tf,y0,y1,n):
    t = linspace(t0,tf,n)
    y = list([y0,y1])
    for i in range(1,n-1):
        h = t[i+1]-t[i]
        k1 = h*f(t[i],y[-1])
        k2 = h*f(t[i]+h/2.0,y[-1]+k1/2.0)
        y.append(y[-1]+k2)
    return y
```

RK4 Method

In[4]:

```
def ode_RK4(f,t0,tf,y0,y1,n):
    t = linspace(t0,tf,n)
    y = list([y0,y1])
    for i in range(1,n-1):
        h = t[i+1]-t[i]
        k1 = h*f(t[i],y[-1])
        k2 = h*f(t[i]+h/2.0,y[-1]+k1/2.0)
        k3 = h*f(t[i]+h/2.0,y[-1]+k2/2.0)
        k4 = h*f(t[i]+h,y[-1]+k3)
        y.append(y[-1]+(k1+2*k2+2*k3+k4)/6)
    return y
```

SSE

In[5]:

```
def SSE_finitedifference(y_estimate,y_real):
    SSE_Temp = 0
    for i in range(len(y_real)):
```

```
        SSE_Temp = SSE_Temp + (y_real[i]-y_estimate[i])**2
    SSE_Temp = sqrt(SSE_Temp)
    return SSE_Temp
```

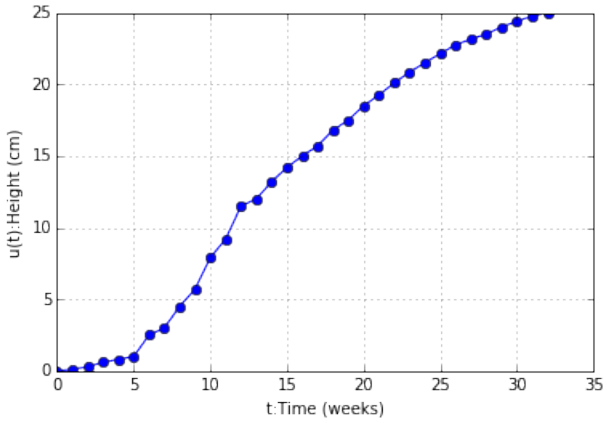
Data

In[6]:

```
y =
   [0.0,0.1,0.3,0.6,0.8,1,2.5,3,4.5,5.7,7.9,9.2,11.5,12,13.2,14.2,15,5.7,1


cdy =
   [0.15,0.25,0.25,0.2,0.85,1.0,1.0,1.35,1.7,1.75,1.8,1.4,0.85,1.1,0.9,0.7


n = len(y)
n_cdy = len(cdy)
t = range(n)
```

In[7]:

```
xlabel('t:Time (weeks)')
ylabel('u(t):Height (cm)')
plot(t,y,marker = 'o')
```



1. Finite Difference Method 求 $y' = \alpha y^2 + \beta y$ 的最佳 α 和 β

利用 central difference method

$$\frac{y_{k+1}-y_{k-1}}{t_{k+1}-t_{k-1}} = \alpha y_k^2 + \beta y_k$$

$$\rightarrow y_{k+2} = y_k + 2\alpha y_{k+1}^2 + 2\beta y_{k+1}$$

Step1. 利用最小平方法求出最佳 α 和 β

In[8]:

```
A = zeros((n-2,2))
b = zeros(n-2)

for i in range(n-2):
    A[i,0] = y[i+1]**2
    A[i,1] = y[i+1]
    b[i] = 0.5*(y[i+2] - y[i])

A = mat(A)
b = mat(b)

z = linalg.solve(A.T*A,A.T*b.T)

alpha = float(z[0])
beta = float(z[1])
z
```

matrix([[-0.00844749],[0.21174495]])

Step.2 接下來利用上面求出的 α 和 β，搭配上數值偏微分方法來得到一組數據逼近原始資料

也就是說 Given $\{(y_i, t_i)\}_{i=1}^n$，我們要用 Heun's Method、RK2 Method 和 RK4 Method

去找到一組資料來逼近原始資料

Settintg up

Data 如上面給定的

In[9]:

```
t0 = t[0]
tf = t[-1]
y0 = 0.0
y1 = 0.1

#要解的微分方程

def cdy_fcn(t,y):
    return alpha*y**2+beta*y
```

Heun's Method

In[10]:

```
y_Heun = ode_Heun(lambda x,y: cdy_fcn(x,y),t0,tf,y0,y1,n)
SSE_Heun0 = SSE_finitedifference(y_Heun,y)
print SSE_Heun0
```

55.7242174456

RK2 Method

In[11]:

```
y_RK2 = ode_RK2(lambda x,y: cdy_fcn(x,y),t0,tf,y0,y1,n)
SSE_RK2a = SSE_finitedifference(y_RK2,y)
print SSE_RK2a
```

55.66572951

RK4 Method

In[12]:

```
y_RK4 = ode_RK4(lambda x,y: cdy_fcn(x,y),t0,tf,y0,y1,n)
SSE_RK4a = SSE_finitedifference(y_RK4,y)
print SSE_RK4a
```

55.3013480532

Step3.** 對 y_0，y_1 做擾動，希望能降低 SSE

$$y_{k+2} = y_k + 2\alpha y_{k+1}^2 + 2\beta y_{k+1}$$

In[13]:

```
y_1 = [0.0 , 0.1]
for i in range(n-2):
    y_1.append(y_1[-2]+2*alpha*y_1[-1]**2+2*beta*y_1[-1])

#最小平方差
SSE_0 = SSE_finitedifference(y,y_1)
print SSE_0
```

In[14]:

```
count = 0
SSE = SSE_0
while count <1000:
    y_temp = [0.0, 0.0]

    #對起始點增加擾動
    e1 = random.uniform(-0.1,0.1)
    e2 = random.uniform(-0.1,0.1)
```

```
    y_temp[0] = y_1[0]+e1
    y_temp[1] = y_1[0]+e2

    #生成新的Data
    for i in range(n-2):
        y_temp.append(y_temp[-2]+2*alpha*y_temp[-1]**2+2*beta*y_temp
           [-1])

    #計算新的最小平方差
    SSE_temp = SSE_finitedifference(y,y_temp)

    #比較最小平方差
    if SSE_temp < SSE:
        SSE = SSE_temp
        y_1 = y_temp

    count += 1

print SSE
print alpha,beta
print y_1[0], y_1[1]
```

5.659286856

-0.0084474869693 0.211744951986

1.16903645355 1.32198681127

Heun's Method

In[15]:

```
y_Heun = ode_Heun(lambda x,y: cdy_fcn(x,y),t0,tf,y_1[0],y_1[1],n)
```

```
SSE_Heun0 = SSE_finitedifference(y_Heun,y)
print SSE_Heun0
```

30.2074082818

RK2 Method

In[16]:

```
y_RK2 = ode_RK2(lambda x,y: cdy_fcn(x,y),t0,tf,y_1[0],y_1[1],n)
SSE_RK2a = SSE_finitedifference(y_RK2,y)
print SSE_RK2a
```

30.1259883441

RK4 Method

In[17]:

```
y_RK4 = ode_RK4(lambda x,y: cdy_fcn(x,y),t0,tf,y_1[0],y_1[1],n)
SSE_RK4a = SSE_finitedifference(y_RK4,y)
print SSE_RK4a
```

29.8589927079

Step.4 對 α、β 和 y_0 做擾動，降低 SSE

有了 Step.3 修正過的 y_0 我們接著來修正 α 和 β

In[18]:

```
flag = 0
while flag < 100:
    count = 0
    SSE = SSE_0
    while count <1000:
        y_temp = [y_1[0], y_1[1]]
```

```
        #對起始點增加擾動
        e1 = random.uniform(-0.001,0.001)
        e2 = random.uniform(-0.001,0.001)
        alpha_temp = alpha + e1
        beta_temp = beta + e2

        #生成新的Data
        for i in range(n-2):
            y_temp.append(y_temp[-2]+2*alpha_temp*y_temp[-1]**2+2*
               beta_temp*y_temp[-1])
        #計算新的最小平方差
        SSE_temp = SSE_finitedifference(y,y_temp)

        #比較最小平方差
        if SSE_temp < SSE:
            SSE = SSE_temp
            y_1 = y_temp
            alpha = alpha_temp
            beta = beta_temp

        count += 1

    count = 0
    while count <1000:
        y_temp = [0.0, 0.0]

        #對起始點增加擾動
```

```
        e1 = random.uniform(-0.1,0.1)
        e2 = random.uniform(-0.1,0.1)
        y_temp[0] = y_1[0]+e1
        y_temp[1] = y_1[0]+e2

        #生成新的Data
        for i in range(n-2):
            y_temp.append(y_temp[-2]+2*alpha*y_temp[-1]**2+2*beta*
               y_temp[-1])

        #計算新的最小平方差
        SSE_temp = SSE_finitedifference(y,y_temp)

        #比較最小平方差
        if SSE_temp < SSE:
            SSE = SSE_temp
            y_1 = y_temp

        count += 1
    flag+=1
print SSE
print alpha,beta
print y_1[0], y_1[1]
```

5.39888930477

-0.00959911519495 0.232405855098

0.990787303889 1.07654767318

In[19]:

```
t0 = t[0]
tf = t[-1]
y0 = y_1[0]
y1 = y_1[1]
#要解的微分方程
def cdy_fcn(t,y):
    return alpha*y**2+beta*y
```

Heun's Method

n[20]:

```
y_Heun = ode_Heun(lambda x,y: cdy_fcn(x,y),t0,tf,y0,y1,n)
SSE_Heun0 = SSE_finitedifference(y_Heun,y)
print SSE_Heun0
```

5.6053053617

RK2's Method

In[21]:

```
y_RK2 = ode_RK2(lambda x,y: cdy_fcn(x,y),t0,tf,y0,y1,n)
SSE_RK2a = SSE_finitedifference(y_RK2,y)
print SSE_RK2a
```

5.58681233538

RK4 Method

In[22]:

```
y_RK4 = ode_RK4(lambda x,y: cdy_fcn(x,y),t0,tf,y0,y1,n)
SSE_RK4a = SSE_finitedifference(y_RK4,y)
print SSE_RK4a
```

5.533732902

A.2 Code 02

In[1]:

```
%pylab inline
```

Some Tools

Heun's Method

In[2]:

```
def ode_Heun(f,t0,tf,y0,n):
    t = np.linspace(t0,tf,n)
    y = list([y0])
    for i in range(n-1):
        h = t[i+1]-t[i]
        fk = f(t[i],y[-1])
        fk1 = f(t[i+1],y[-1]+h*fk)
        y.append(y[-1]+h*(fk+fk1)/2.0)

    return y
```

RK2 Method

In[3]:

```
def ode_RK2(f,t0,tf,y0,n):
    t = linspace(t0,tf,n)
    y = list([y0])
    for i in range(n-1):
        h = t[i+1]-t[i]
        k1 = h*f(t[i],y[-1])
```

```
        k2 = h*f(t[i]+h/2.0,y[-1]+k1/2.0)
        y.append(y[-1]+k2)
    return y
```

RK4 Method

In[4]:

```
def ode_RK4(f,t0,tf,y0,n):
    t = linspace(t0,tf,n)
    y = list([y0])
    for i in range(n-1):
        h = t[i+1]-t[i]
        k1 = h*f(t[i],y[-1])
        k2 = h*f(t[i]+h/2.0,y[-1]+k1/2.0)
        k3 = h*f(t[i]+h/2.0,y[-1]+k2/2.0)
        k4 = h*f(t[i]+h,y[-1]+k3)
        y.append(y[-1]+(k1+2*k2+2*k3+k4)/6)
    return y
```

SSE

In[5]:

```
def SSE_cdy(y_estimate,y_real):
    SSE_Temp = 0
    for j in range(len(y_real)):
        SSE_Temp = SSE_Temp + (y_estimate[j]-y_real[j])**2
    SSE_Temp = sqrt(SSE_Temp)
    return SSE_Temp
```

In[6]:

```
def SSE_numerical(y_estimate,y_real,gap):
    y_T = []
```

```
    SSE_Temp = 0.0
    for i in range(len(y_real)):
        y_T.append(y_estimate[i*gap])
        SSE_Temp = SSE_Temp+(y_T[i]-y_real[i])**2
    SSE_Temp = sqrt(SSE_Temp)
    return SSE_Temp
```

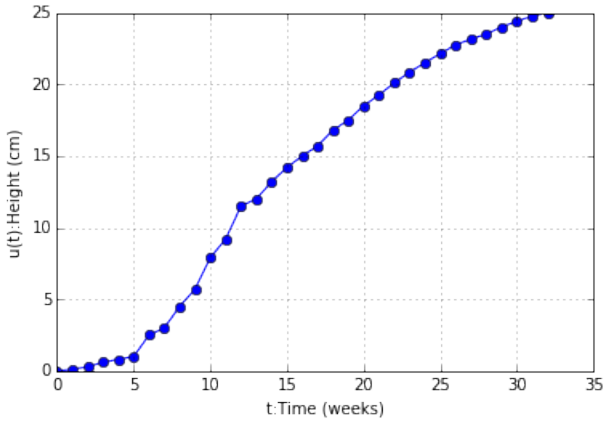
Data

In[7]:

```
y =
   [0.0,0.1,0.3,0.6,0.8,1,2.5,3,4.5,5.7,7.9,9.2,11.5,12,13.2,14.2,15,15.7,


cdy =
   [0.15,0.25,0.25,0.2,0.85,1.0,1.0,1.35,1.7,1.75,1.8,1.4,0.85,1.1,0.9,0.7


n = len(y)
n_cdy = len(cdy)
t = range(n)
```

In[8]:

```
xlabel('t:Time (weeks)')
ylabel('u(t):Height (cm)')
plot(t,y,marker = 'o')
```

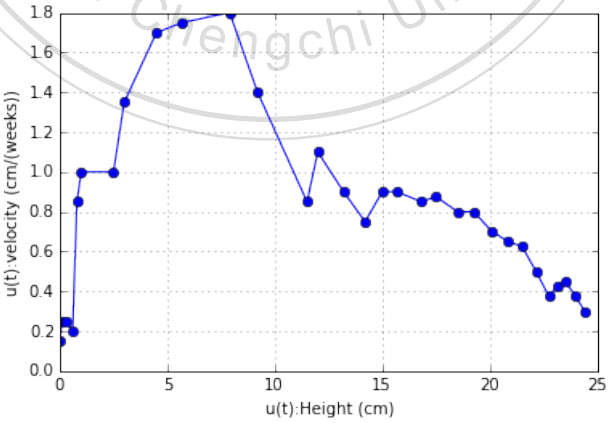


2. 利用 (y, cdy) 來找出 $y' = \alpha y^2 + \beta y$ 的最佳 α 和 β

Step1. 利用最小平方法求出最佳 α 和 β

In[9]:

```
y_5 =
   [0.0,0.1,0.3,0.6,0.8,1,2.5,3,4.5,5.7,7.9,9.2,11.5,12,13.2,14.2,15,15.7,


xlabel('u(t):Height (cm)')
ylabel('u(t):velocity (cm/(weeks))')
plot(y_5,cdy,marker = 'o',color = 'b',)
```



In[10]:

```
#利用最小平方法求出 alpha 和 beta
A = zeros((n_cdy,2))
b = zeros(n_cdy)
```

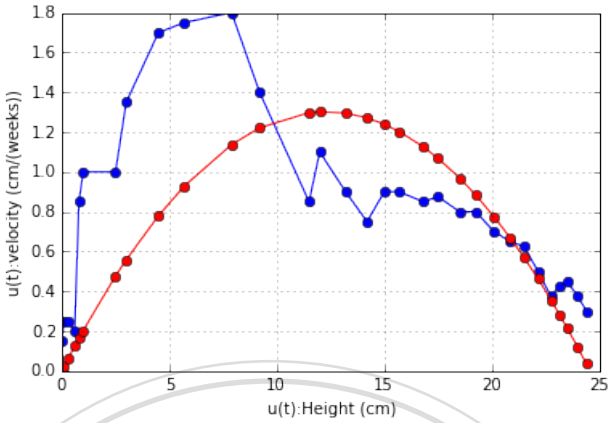
```

for i in range(n_cdy):
    A[i,0] = y[i]**2
    A[i,1] = y[i]
    b[i] = cdy[i]

A = mat(A)
b = mat(b)

z = linalg.solve(A.T*A,A.T*b.T)

alpha = float(z[0])
beta = float(z[1])
z
```

matrix([[-0.00863605],[0.21215756]])

In[11]:

```

def cdy_fcn(t,y):
    return alpha*y**2+beta*y

cdy_estimate = []
for i in range(n_cdy):
    cdy_estimate.append(cdy_fcn(t,y_5[i]))

xlabel('u(t):Height (cm)')
ylabel('u(t):velocity (cm/(weeks))')
plot(y_5,cdy,marker = 'o',color = 'b',)
```

```
plot(y_5,cdy_estimate,marker = 'o',color = 'r',)
```



In[12]:

```
SSE_cdy0 = SSE_cdy(cdy_estimate,cdy)
print SSE_cdy0
print alpha, beta
```

2.31538926492

-0.00863604893773 0.212157561965

Step2. 利用求得的 α 和 β，搭配數值微分方程的方法來找到一組函數資料來逼近原本 Data

Setting up

In[13]:

```
t0 = 0
tf = 30
y0 = 1

#這裡的gap必須要是整數，代表每格資料間細分的次數
gap = 100
n = 30*gap+1
t = linspace(t0,tf,n)
```

Heun's method

In[14]:

```
y0_Heun = y0
y_Heun = ode_Heun(lambda x,y: cdy_fcn(x,y),t0,tf,y0_Heun,n)
SSE_Heun0 = SSE_numerical(y_Heun,y_5,gap)
print SSE_Heun0
```

6.3916710696

RK2 Method

In[15]:

```
y0_RK2 = y0
y_RK2 = ode_RK2(lambda x,y: cdy_fcn(x,y),t0,tf,y0_RK2,n)
SSE_RK2a = SSE_numerical(y_RK2,y_5,gap)
print SSE_RK2a
```

6.39166483931

RK4 Method

In[16]:

```
y0_RK4 = y0
y_RK4 = ode_RK4(lambda x,y: cdy_fcn(x,y),t0,tf,y0_RK4,n)
SSE_RK4a = SSE_numerical(y_RK4,y_5,gap)
print SSE_RK4a
```

6.39165419709

Step.3 對 y_0 做擾動，希望能降低誤差

Heun's Method

In[17]:

```
count = 0
SSE_Heun = SSE_Heun0
while count <1000:

    #對起始點增加擾動
    e1 = random.uniform(-0.1,0.1)
    y0_temp = y0_Heun+e1

    #生成新的Data
    y_temp = ode_Heun(lambda x,y: cdy_fcn(x,y),t0,tf,y0_temp,n)

    #計算新的最小平方差
    SSE_temp = SSE_numerical(y_temp,y_5,gap)

    #比較最小平方差
    if SSE_temp < SSE_Heun:
        SSE_Heun = SSE_temp
        y0_Heun = y0_temp

    count += 1

print SSE_Heun
print y0_Heun
```

5.49586952118

1.15292974301

RK2 Method

In[18]:

```
count = 0
SSE_RK2 = SSE_RK2a
while count <1000:

    #對起始點增加擾動
    e1 = random.uniform(-0.1,0.1)
    y0_temp = y0_RK2+e1

    #生成新的Data
    y_temp = ode_Heun(lambda x,y: cdy_fcn(x,y),t0,tf,y0_temp,n)

    #計算新的最小平方差
    SSE_temp = SSE_numerical(y_temp,y_5,gap)

    #比較最小平方差
    if SSE_temp < SSE_RK2:
        SSE_RK2 = SSE_temp
        y0_RK2 = y0_temp

    count += 1

print SSE_RK2
print y0_RK2
```

5.49586689845

1.15320990593

RK4 Method

In[19]:

```
count = 0
SSE_RK4 = SSE_RK4a
while count <1000:

    #對起始點增加擾動
    e1 = random.uniform(-0.1,0.1)
    y0_temp = y0_RK4+e1

    #生成新的Data
    y_temp = ode_Heun(lambda x,y: cdy_fcn(x,y),t0,tf,y0_temp,n)

    #計算新的最小平方差
    SSE_temp = SSE_numerical(y_temp,y_5,gap)

    #比較最小平方差
    if SSE_temp < SSE_RK4:
        SSE_RK4 = SSE_temp
        y0_RK4 = y0_temp

    count += 1

print SSE_RK4
print y0_RK4
```

5.49586689702

1.15320789793

A.3 Code 03

In[1]:

```
%pylab inline
```

Some Tools

Heun's Method

In[2]:

```
def ode_Heun(f,t0,tf,y0,y1,n):
    t = np.linspace(t0,tf,n)
    y = list([y0,y1])
    for i in range(1,n-1):
        h = t[i+1]-t[i]
        fk = f(t[i],y[-1])
        fk1 = f(t[i+1],y[-1]+h*fk)
        y.append(y[-1]+h*(fk+fk1)/2.0)

    return y
```

RK2 Method

In[3]:

```
def ode_RK2(f,t0,tf,y0,y1,n):
    t = linspace(t0,tf,n)
    y = list([y0,y1])
    for i in range(1,n-1):
        h = t[i+1]-t[i]
        k1 = h*f(t[i],y[-1])
        k2 = h*f(t[i]+h/2.0,y[-1]+k1/2.0)
        y.append(y[-1]+k2)
    return y
```

RK4 Method

In[4]:

```
def ode_RK4(f,t0,tf,y0,y1,n):
    t = linspace(t0,tf,n)
    y = list([y0,y1])
    for i in range(1,n-1):
        h = t[i+1]-t[i]
        k1 = h*f(t[i],y[-1])
        k2 = h*f(t[i]+h/2.0,y[-1]+k1/2.0)
        k3 = h*f(t[i]+h/2.0,y[-1]+k2/2.0)
        k4 = h*f(t[i]+h,y[-1]+k3)
        y.append(y[-1]+(k1+2*k2+2*k3+k4)/6)
    return y
```

SSE

In[5]:

```
def SSE_finitedifference(y_estimate,y_real):
    SSE_Temp = 0
    for i in range(len(y_real)):
        SSE_Temp = SSE_Temp + (y_real[i]-y_estimate[i])**2
    SSE_Temp = sqrt(SSE_Temp)
    return SSE_Temp
```

Data

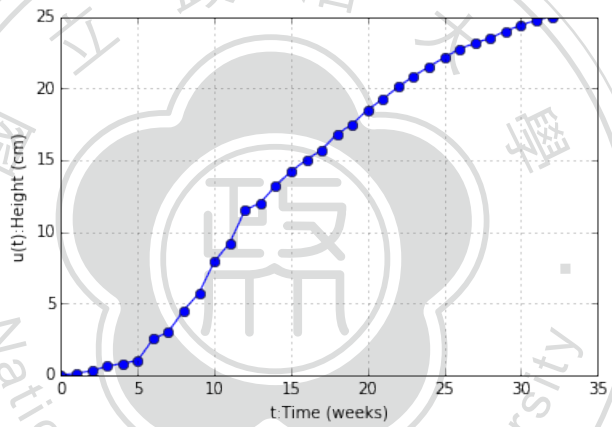
In[6]:

```
y =
   [0.0,0.1,0.3,0.6,0.8,1,2.5,3,4.5,5.7,7.9,9.2,11.5,12,13.2,14.2,15,15.7,


cdy =
   [0.15,0.25,0.25,0.2,0.85,1.0,1.0,1.35,1.7,1.75,1.8,1.4,0.85,1.1,0.9,0.7
```

```
n = len(y)
n_cdy = len(cdy)
t = range(n)
```

In[7]:

```
xlabel('t:Time (weeks)')
ylabel('u(t):Height (cm)')
plot(t,y,marker = 'o')
```



3. Finite Difference Method 求 $y' = \alpha y^2 + \beta y + asin(by + c)$ 的最佳 α、β、a、b 和 c

利用 central difference method

$$\frac{y_{k+1} - y_{k-1}}{t_{k+1} - t_{k-1}} = \alpha y_k^2 + \beta y_k + asin(by_k + c)$$

$$\rightarrow y_{k+2} = y_k + 2\alpha y_{k+1}^2 + 2\beta y_{k+1} + asin(by_{k+1} + c)$$

Step1. 給定一組 α、β、a、b 和 c

In[8]:

```
alpha = -0.00844749
beta = 0.21174495
a = 0.0
```

```
b = 0.0
c = 0.0
```

Step.2 接下來利用上面求出的 α、β、a、b 和 c，搭配上數值偏微分方法來得到一組數據逼近原始資料

也就是說 Given $\{(y_i, t_i)\}_{i=1}^{n}$，我們要用 Heun's Method、RK2 Method 和 RK4 Method 去找到一組資料來逼近原始資料

Settintg up

Data 如上面給定的

In[9]:

```
t0 = t[0]
tf = t[-1]
y0 = 0.0
y1 = 0.1

#要解的微分方程
def cdy_fcn(t,y):
    return alpha*y**2+beta*y +a*sin(b*y+c)
```

Heun's Method

In[10]:

```
y_Heun = ode_Heun(lambda x,y: cdy_fcn(x,y),t0,tf,y0,y1,n)
SSE_Heun0 = SSE_finitedifference(y_Heun,y)
print SSE_Heun0
```

55.7242229592

RK2 Method

In[11]:

```
y_RK2 = ode_RK2(lambda x,y: cdy_fcn(x,y),t0,tf,y0,y1,n)
```

```
SSE_RK2a = SSE_finitedifference(y_RK2,y)
print SSE_RK2a
```

55.6657350223

RK4 Method

In[12]:

```
y_RK4 = ode_RK4(lambda x,y: cdy_fcn(x,y),t0,tf,y0,y1,n)
SSE_RK4a = SSE_finitedifference(y_RK4,y)
print SSE_RK4a
```

55.3013536756

Step3. 對 y_0，y_1 做擾動，希望能降低 SSE

$$y_{k+2} = y_k + 2\alpha y_{k+1}^2 + 2\beta y_{k+1} + 2asin(by_{k+1} + c)$$

In[13]:

```
y_1 = [0.0 , 0.1]
for i in range(n-2):
    y_1.append(y_1[-2]+2*alpha*y_1[-1]**2+2*beta*y_1[-1]+2*a*sin(b*y_1
       [-1]+c))

#最小平方差
SSE_0 = SSE_finitedifference(y,y_1)
print SSE_0
```

64.1919298183

In[14]:

```
count = 0
```

```
SSE = SSE_0
while count <1000:
    y_temp = [0.0, 0.0]

    #對起始點增加擾動
    e1 = random.uniform(-0.1,0.1)
    e2 = random.uniform(-0.1,0.1)
    y_temp[0] = y_1[0]+e1
    y_temp[1] = y_1[0]+e2

    #生成新的Data
    for i in range(n-2):
        y_temp.append(y_temp[-2]+2*alpha*y_temp[-1]**2+2*beta*y_temp
            [-1]+2*a*sin(b*y_temp[-1]+c))

    #計算新的最小平方差
    SSE_temp = SSE_finitedifference(y,y_temp)

    #比較最小平方差
    if SSE_temp < SSE:
        SSE = SSE_temp
        y_1 = y_temp

    count += 1

print SSE
print alpha,beta,a,b,c
print y_1[0], y_1[1]
```

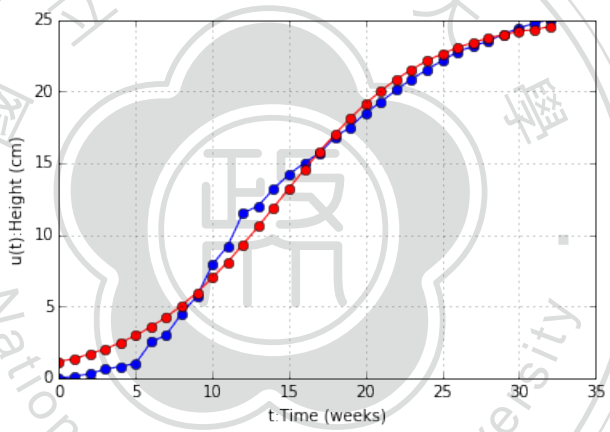
5.65827878047

-0.00844749 0.21174495 0.0 0.0 0.0

1.20182153439 1.32129224963

In[15]:

```
xlabel('t:Time (weeks)')
ylabel('u(t):Height (cm)')
plot(t,y,marker = 'o',color = 'b',)
plot(t,y_1,marker = 'o',color = 'r',)
```



Heun's Method

In[16]:

```
y_Heun = ode_Heun(lambda x,y: cdy_fcn(x,y),t0,tf,y_1[0],y_1[1],n)
SSE_Heun0 = SSE_finitedifference(y_Heun,y)
print SSE_Heun0
```

5.79278605472

RK2 Method

In[17]:

```
y_RK2 = ode_RK2(lambda x,y: cdy_fcn(x,y),t0,tf,y_1[0],y_1[1],n)
SSE_RK2a = SSE_finitedifference(y_RK2,y)
```

```
print SSE_RK2a
```

5.7761112128

RK4 Method

In[18]:

```
y_RK4 = ode_RK4(lambda x,y: cdy_fcn(x,y),t0,tf,y_1[0],y_1[1],n)
SSE_RK4a = SSE_finitedifference(y_RK4,y)
print SSE_RK4a
```

5.7418495602

Step.4 對 α、β、a、b、c 和 y_0 做擾動，降低 SSE

有了 Step.3 修正過的 y_0 我們接著來修正 α、β、a、b 和 c

In[19]:

```
flag = 0
while flag < 100:
    count = 0
    SSE = SSE_0
    while count <100:
        y_temp = [y_1[0], y_1[1]]

        #對起始點增加擾動
        e1 = random.uniform(-0.001,0.001)
        e2 = random.uniform(-0.001,0.001)
        e3 = random.uniform(-0.001,0.001)
        e4 = random.uniform(-0.001,0.001)
        e5 = random.uniform(-0.001,0.001)

        alpha_temp = alpha + e1
```

```
        beta_temp = beta + e2
        a_temp = a + e3
        b_temp = b + e4
        c_temp = c + e5


        #生成新的Data
        for i in range(n-2):
            y_temp.append(y_temp[-2]+2*alpha_temp*y_temp[-1]**2+2*
               beta_temp*y_temp[-1]+a_temp*sin(b_temp*y_temp[-1]+c_temp
               ))
        #計算新的最小平方差
        SSE_temp = SSE_finitedifference(y,y_temp)

        #比較最小平方差
        if SSE_temp < SSE:
            SSE = SSE_temp
            y_1 = y_temp
            alpha = alpha_temp
            beta = beta_temp
            a = a_temp
            b = b_temp
            c = c_temp

        count += 1

    count = 0
    while count <100:
```

```
        y_temp = [0.0, 0.0]

        #對起始點增加擾動
        e1 = random.uniform(-0.1,0.1)
        e2 = random.uniform(-0.1,0.1)
        y_temp[0] = y_1[0]+e1
        y_temp[1] = y_1[1]+e2

        #生成新的Data
        for i in range(n-2):
            y_temp.append(y_temp[-2]+2*alpha*y_temp[-1]**2+2*beta*
               y_temp[-1]+a*sin(b*y_temp[-1]+c))

        #計算新的最小平方差
        SSE_temp = SSE_finitedifference(y,y_temp)

        #比較最小平方差
        if SSE_temp < SSE:
            SSE = SSE_temp
            y_1 = y_temp

        count += 1
    flag+=1
print SSE
print alpha,beta,a,b,c
print y_1[0], y_1[1]
```

5.40260286876

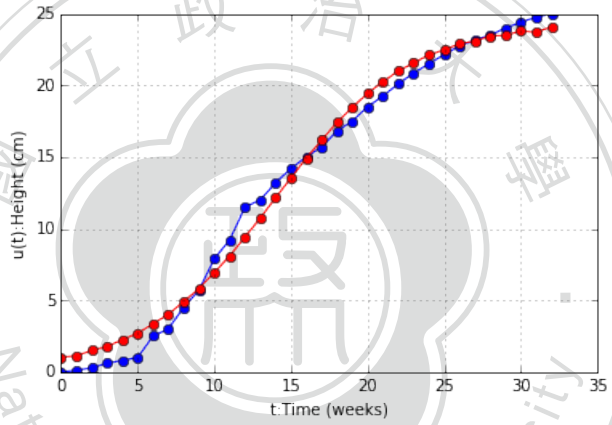
-0.00951510827755 0.230182300243 -0.0262259316043

-0.0147144948651 -0.00726659819372

1.01627076858 1.10278819877

In[20]:

```
xlabel('t:Time (weeks)')
ylabel('u(t):Height (cm)')
plot(t,y,marker = 'o',color = 'b',)
plot(t,y_1,marker = 'o',color = 'r',)
```



In[21]:

```
t0 = t[0]
tf = t[-1]
y0 = y_1[0]
y1 = y_1[1]
#要解的微分方程
def cdy_fcn(t,y):
    return alpha*y**2+beta*y +a*sin(b*y +c)
```

Heun's Method

In[22]:

```
y_Heun = ode_Heun(lambda x,y: cdy_fcn(x,y),t0,tf,y0,y1,n)
SSE_Heun0 = SSE_finitedifference(y_Heun,y)
```

```
print SSE_Heun0
```

5.58823184026

RK2's Method

In[23]:

```
y_RK2 = ode_RK2(lambda x,y: cdy_fcn(x,y),t0,tf,y0,y1,n)
SSE_RK2a = SSE_finitedifference(y_RK2,y)
print SSE_RK2a
```

5.56985208097

RK4 Method

In[24]:

```
y_RK4 = ode_RK4(lambda x,y: cdy_fcn(x,y),t0,tf,y0,y1,n)
SSE_RK4a = SSE_finitedifference(y_RK4,y)
print SSE_RK4a
```

5.51999849112

A.4 Code 04

In[1]:

```
%pylab inline
```

Some Tools

Heun's Method

In[2]:

```
def ode_Heun(f,t0,tf,y0,n):
    t = np.linspace(t0,tf,n)
```

```
    y = list([y0])
    for i in range(n-1):
        h = t[i+1]-t[i]
        fk = f(t[i],y[-1])
        fk1 = f(t[i+1],y[-1]+h*fk)
        y.append(y[-1]+h*(fk+fk1)/2.0)

    return y
```

RK2 Method

In[3]:

```
def ode_RK2(f,t0,tf,y0,n):
    t = linspace(t0,tf,n)
    y = list([y0])
    for i in range(n-1):
        h = t[i+1]-t[i]
        k1 = h*f(t[i],y[-1])
        k2 = h*f(t[i]+h/2.0,y[-1]+k1/2.0)
        y.append(y[-1]+k2)
    return y
```

RK4 Method

In[4]:

```
def ode_RK4(f,t0,tf,y0,n):
    t = linspace(t0,tf,n)
    y = list([y0])
    for i in range(n-1):
        h = t[i+1]-t[i]
        k1 = h*f(t[i],y[-1])
        k2 = h*f(t[i]+h/2.0,y[-1]+k1/2.0)
```

```
        k3 = h*f(t[i]+h/2.0,y[-1]+k2/2.0)
        k4 = h*f(t[i]+h,y[-1]+k3)
        y.append(y[-1]+(k1+2*k2+2*k3+k4)/6)
    return y
```

SSE

In[5]:

```
def SSE_cdy(y_estimate,y_real):
    SSE_Temp = 0
    for j in range(len(y_real)):
        SSE_Temp = SSE_Temp + (y_estimate[j]-y_real[j])**2
    SSE_Temp = sqrt(SSE_Temp)
    return SSE_Temp
```

In[6]:

```
def SSE_numerical(y_estimate,y_real,gap):
    y_T = []
    SSE_Temp = 0.0
    for i in range(len(y_real)):
        y_T.append(y_estimate[i*gap])
        SSE_Temp = SSE_Temp+(y_T[i]-y_real[i])**2
    SSE_Temp = sqrt(SSE_Temp)
    return SSE_Temp
```

Data

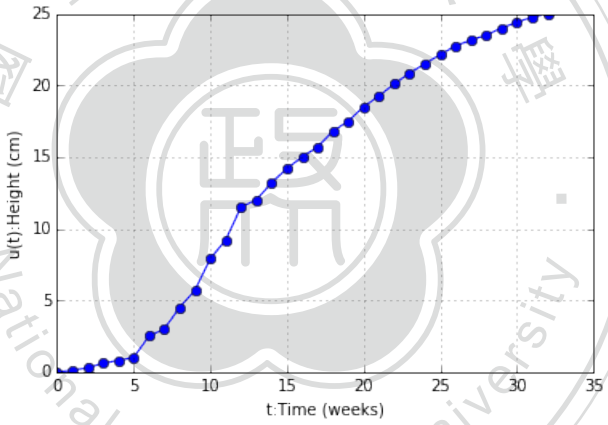
In[7]:

```
y =
   [0.0,0.1,0.3,0.6,0.8,1,2.5,3,4.5,5.7,7.9,9.2,11.5,12,13.2,14.2,15,15.7,


cdy =
```

```
   [0.15,0.25,0.25,0.2,0.85,1.0,1.0,1.35,1.7,1.75,1.8,1.4,0.85,1.1,0.9,0.7
n = len(y)
n_cdy = len(cdy)
t_0 = range(n)
```

In[8]:

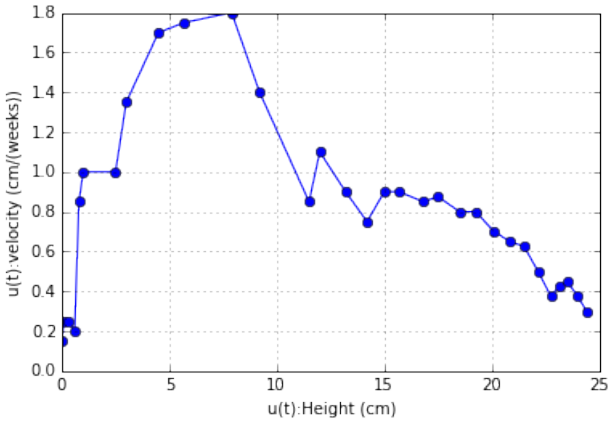
```
xlabel('t:Time (weeks)')
ylabel('u(t):Height (cm)')
plot(t_0,y,marker = 'o')
```



4. 利用 (y, cdy) 來找出 $y' = \alpha y^2 + \beta y + asin(by + c)$ 的最佳 α、β、a、b 和 c

In[9]:

```
y_5 =
   [0.0,0.1,0.3,0.6,0.8,1,2.5,3,4.5,5.7,7.9,9.2,11.5,12,13.2,14.2,15,15.7,
xlabel('u(t):Height (cm)')
ylabel('u(t):velocity (cm/(weeks))')
plot(y_5,cdy,marker = 'o',color = 'b',)
```



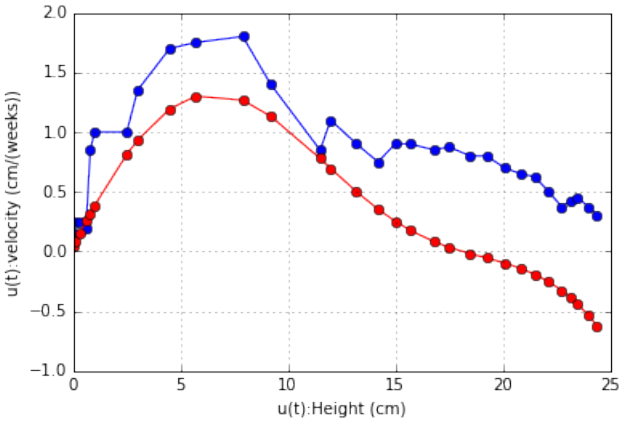
Step1. 給定一組 α、β、a、b 和 c

In[10]:

```
alpha = -0.01
beta = 0.2
a = -0.5
b = -0.3
c = -0.1
```

In[11]:

```
def cdy_fcn(t,y):
    return alpha*y**2+beta*y+a*sin(b*y+c)

cdy_estimate = []
for i in range(n_cdy):
    cdy_estimate.append(cdy_fcn(t_0,y_5[i]))

xlabel('u(t):Height (cm)')
ylabel('u(t):velocity (cm/(weeks))')
plot(y_5,cdy,marker = 'o',color = 'b',)
plot(y_5,cdy_estimate,marker = 'o',color = 'r',)
```



In[12]:

```
SSE_cdy0 = SSE_cdy(cdy_estimate,cdy)
print SSE_cdy0
print alpha, beta, a, b, c
```

3.45827066306

-0.01 0.2 -0.5 -0.3 -0.1

Step2. 利用求得的α、β、a、b和c，搭配數值微分方程的方法來找到一組函數資料來逼近原本 Data

Setting up

In[13]:

```
t0 = 0
tf = 30
y0 = 1

#這裡的gap必須要是整數，代表每格資料間細分的次數
gap = 100
n = 30*gap+1
t = linspace(t0,tf,n)
```

Heun's method

In[14]:

```
y0_Heun = y0
y_Heun = ode_Heun(lambda x,y: cdy_fcn(x,y),t0,tf,y0_Heun,n)
SSE_Heun0 = SSE_numerical(y_Heun,y_5,gap)
print SSE_Heun0
```

22.7715170995

RK2 Method

In[15]:

```
y0_RK2 = y0
y_RK2 = ode_RK2(lambda x,y: cdy_fcn(x,y),t0,tf,y0_RK2,n)
SSE_RK2a = SSE_numerical(y_RK2,y_5,gap)
print SSE_RK2a
```

22.7715218758

RK4 Method

In[16]:

```
y0_RK4 = y0
y_RK4 = ode_RK4(lambda x,y: cdy_fcn(x,y),t0,tf,y0_RK4,n)
SSE_RK4a = SSE_numerical(y_RK4,y_5,gap)
print SSE_RK4a
```

22.7715246995

Step.3 對 α、β、a、b 和 c 做擾動，降低二階函數估計值跟實際值

In[17]:

```
count = 0
SSE_0 = SSE_cdy0
```

```
while count <10000:

    #對起始點增加擾動
    e1 = random.uniform(-0.001,0.001)
    e2 = random.uniform(-0.001,0.001)
    e3 = random.uniform(-0.001,0.001)
    e4 = random.uniform(-0.001,0.001)
    e5 = random.uniform(-0.001,0.001)
    alpha_temp = alpha + e1
    beta_temp = beta + e2
    a_temp = a + e3
    b_temp = b + e4
    c_temp = c + e5

    #定義新的函數
    def cdy_fcn_temp(t,y):
        return alpha_temp*y**2+beta_temp*y+a_temp*sin(b_temp*y+c_temp)

    #生成新的一次微分估計函數Data
    cdy_estimate_temp = []
    for i in range(n_cdy):
        cdy_estimate_temp.append(cdy_fcn_temp(t,y_5[i]))

    #計算新的最小平方差
    SSE_temp = SSE_cdy(cdy_estimate_temp,cdy)

    #比較最小平方差
    if SSE_temp < SSE_0:
```

```
        SSE_0 = SSE_temp
        alpha = alpha_temp
        beta = beta_temp
        a = a_temp
        b = b_temp
        c = c_temp


    count += 1

print SSE_0
print alpha, beta, a, b, c
```

0.968597241154

-0.0114876114475 0.262698012377 -0.53344134757 -0.318721070024

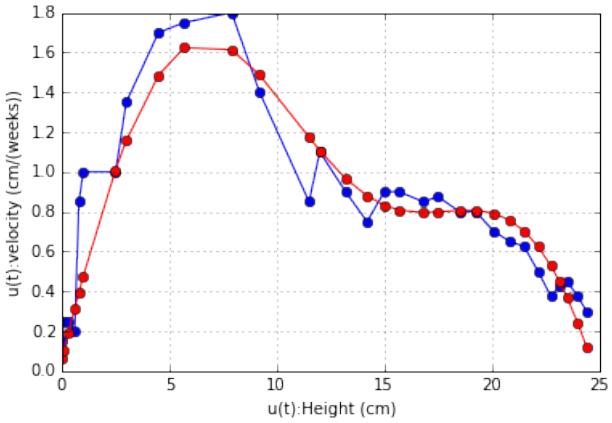
-0.120528934405

In[18]:

```
def cdy_fcn(t,y):
    return alpha*y**2+beta*y+a*sin(b*y+c)
```

In[19]:

```
cdy_estimate = []
for i in range(n_cdy):
    cdy_estimate.append(cdy_fcn(t,y_5[i]))

xlabel('u(t):Height (cm)')
ylabel('u(t):velocity (cm/(weeks))')
plot(y_5,cdy,marker = 'o',color = 'b',)
plot(y_5,cdy_estimate,marker = 'o',color = 'r',)
```



In[20]:

```
t0 = 0
tf = 30
y0 = 1

#這裡的gap必須要是整數，代表每格資料間細分的次數
gap = 100
n = 30*gap+1
t = linspace(t0,tf,n)
```

In[21]:

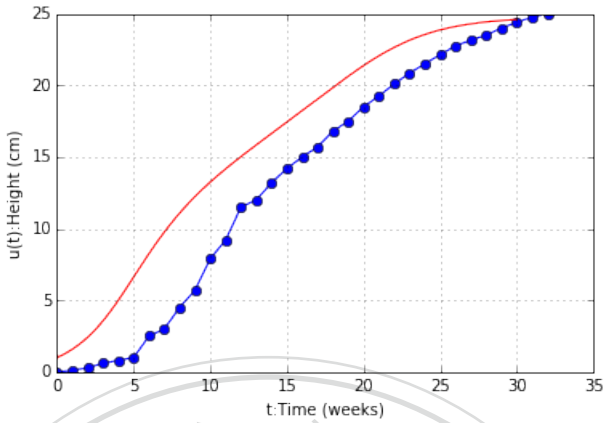
```
y0_Heun = y0
y_Heun = ode_Heun(lambda x,y: cdy_fcn(x,y),t0,tf,y0_Heun,n)
SSE_Heun0 = SSE_numerical(y_Heun,y_5,gap)
print SSE_Heun0
```

20.2251287211

In[22]:

```
xlabel('t:Time (weeks)')
ylabel('u(t):Height (cm)')
plot(t_0,y,marker = 'o',color = 'blue')
```

```
plot(t,y_Heun,color = 'red')
```



Step.4 對 y_0 做擾動，希望能降低誤差

Heun's Method

In[23]:

```
count = 0
SSE_Heun = SSE_Heun0
while count <1000:

    #對起始點增加擾動
    e1 = random.uniform(-0.1,0.1)
    y0_temp = y0_Heun+e1

    #生成新的Data
    y_temp = ode_Heun(lambda x,y: cdy_fcn(x,y),t0,tf,y0_temp,n)

    #計算新的最小平方差
    SSE_temp = SSE_numerical(y_temp,y_5,gap)

    #比較最小平方差
    if SSE_temp < SSE_Heun:
```

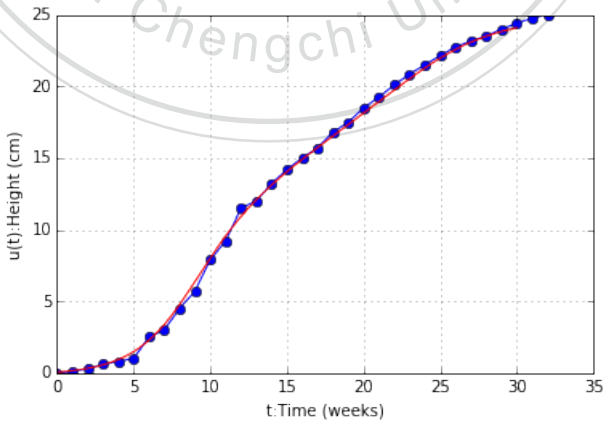
```
        SSE_Heun = SSE_temp
        y0_Heun = y0_temp

    count += 1

print SSE_Heun
print y0_Heun
```

1.68479162933

0.0489081076939

In[24]:

```
y_Heun = ode_Heun(lambda x,y: cdy_fcn(x,y),t0,tf,y0_Heun,n)
xlabel('t:Time (weeks)')
ylabel('u(t):Height (cm)')
plot(t_0,y,marker = 'o',color = 'blue')
plot(t,y_Heun,color = 'red')
```



RK2 Method

In[25]:

```
count = 0
SSE_RK2 = SSE_RK2a
```

```
while count <1000:

    #對起始點增加擾動
    e1 = random.uniform(-0.1,0.1)
    y0_temp = y0_RK2+e1

    #生成新的Data
    y_temp = ode_Heun(lambda x,y: cdy_fcn(x,y),t0,tf,y0_temp,n)

    #計算新的最小平方差
    SSE_temp = SSE_numerical(y_temp,y_5,gap)

    #比較最小平方差
    if SSE_temp < SSE_RK2:
        SSE_RK2 = SSE_temp
        y0_RK2 = y0_temp

    count += 1

print SSE_RK2
print y0_RK2
```

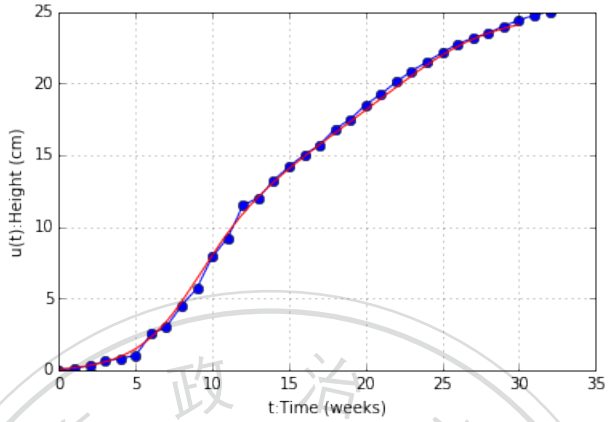
1.68477818706

0.0489704328867

In[26]:

```
y_RK2 = ode_RK2(lambda x,y: cdy_fcn(x,y),t0,tf,y0_RK2,n)
xlabel('t:Time (weeks)')
ylabel('u(t):Height (cm)')
```

```
plot(t_0,y,marker = 'o',color = 'blue')
plot(t,y_RK2,color = 'red')
```



RK4 Method

In[27]:

```
count = 0
SSE_RK4 = SSE_RK4a
while count <1000:

    #對起始點增加擾動
    e1 = random.uniform(-0.1,0.1)
    y0_temp = y0_RK4+e1

    #生成新的Data
    y_temp = ode_Heun(lambda x,y: cdy_fcn(x,y),t0,tf,y0_temp,n)

    #計算新的最小平方差
    SSE_temp = SSE_numerical(y_temp,y_5,gap)

    #比較最小平方差
    if SSE_temp < SSE_RK4:
```

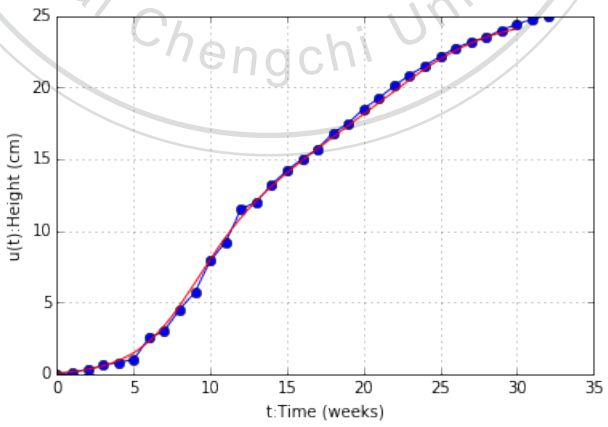
```
        SSE_RK4 = SSE_temp
        y0_RK4 = y0_temp

    count += 1

print SSE_RK4
print y0_RK4
```

1.68491878659

0.0486656124813

In[28]:

```
y_RK4 = ode_RK4(lambda x,y: cdy_fcn(x,y),t0,tf,y0_RK4,n)
xlabel('t:Time (weeks)')
ylabel('u(t):Height (cm)')
plot(t_0,y,marker = 'o',color = 'blue')
plot(t,y_RK4,color = 'red')
```



A.5 Code 05

In[1]:

```
%pylab inline
```

Some Tools

Heun's Method

In[2]:

```
def ode_Heun(f,t0,tf,y0,n):
    t = np.linspace(t0,tf,n)
    y = list([y0])
    for i in range(n-1):
        h = t[i+1]-t[i]
        fk = f(t[i],y[-1])
        fk1 = f(t[i+1],y[-1]+h*fk)
        y.append(y[-1]+h*(fk+fk1)/2.0)

    return y
```

RK2 Method

In[3]:

```
def ode_RK2(f,t0,tf,y0,n):
    t = linspace(t0,tf,n)
    y = list([y0])
    for i in range(n-1):
        h = t[i+1]-t[i]
        k1 = h*f(t[i],y[-1])
        k2 = h*f(t[i]+h/2.0,y[-1]+k1/2.0)
        y.append(y[-1]+k2)
    return y
```

RK4 Method

In[4]:

```
def ode_RK4(f,t0,tf,y0,n):
    t = linspace(t0,tf,n)
```

```
    y = list([y0])
    for i in range(n-1):
        h = t[i+1]-t[i]
        k1 = h*f(t[i],y[-1])
        k2 = h*f(t[i]+h/2.0,y[-1]+k1/2.0)
        k3 = h*f(t[i]+h/2.0,y[-1]+k2/2.0)
        k4 = h*f(t[i]+h,y[-1]+k3)
        y.append(y[-1]+(k1+2*k2+2*k3+k4)/6)
    return y
```

SSE

In[5]:

```
def SSE_cdy(y_estimate,y_real):
    SSE_Temp = 0
    for j in range(len(y_real)):
        SSE_Temp = SSE_Temp + (y_estimate[j]-y_real[j])**2
    SSE_Temp = sqrt(SSE_Temp)
    return SSE_Temp
```

In[6]:

```
def SSE_numerical(y_estimate,y_real,gap):
    y_T = []
    SSE_Temp = 0.0
    for i in range(len(y_real)):
        y_T.append(y_estimate[i*gap])
        SSE_Temp = SSE_Temp+(y_T[i]-y_real[i])**2
    SSE_Temp = sqrt(SSE_Temp)
    return SSE_Temp
```

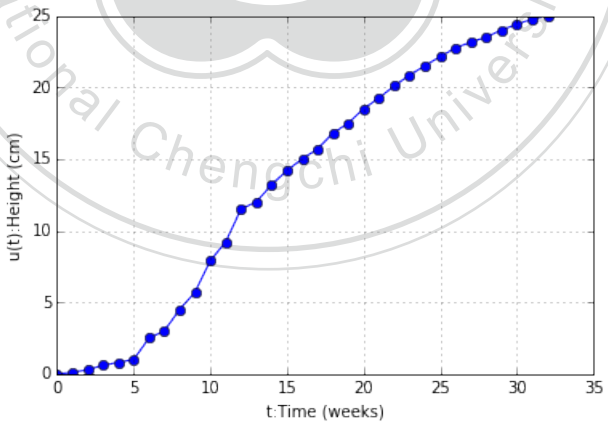
Data

In[7]:

```
y =
   [0.0,0.1,0.3,0.6,0.8,1,2.5,3,4.5,5.7,7.9,9.2,11.5,12,13.2,14.2,15,15.7,

cdy =
   [0.15,0.25,0.25,0.2,0.85,1.0,1.0,1.35,1.7,1.75,1.8,1.4,0.85,1.1,0.9,0.7

n = len(y)
n_cdy = len(cdy)
t_0 = range(n)
```

In[8]:

```
xlabel('t:Time (weeks)')
ylabel('u(t):Height (cm)')
plot(t_0,y,marker = 'o')
```

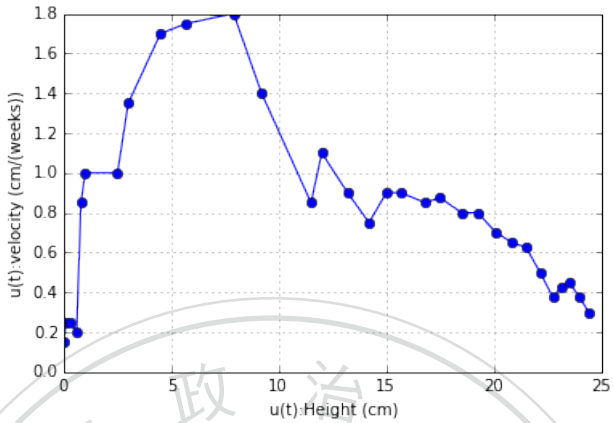


5. 利用 (y, cdy) 來找出 $y' = \alpha y^2 + \beta y + asin(by + c)$ 的最佳 α、β、a、b 和 c

In[9]:

```
y_5 =
   [0.0,0.1,0.3,0.6,0.8,1,2.5,3,4.5,5.7,7.9,9.2,11.5,12,13.2,14.2,15,15.7,

xlabel('u(t):Height (cm)')
```

```
ylabel('u(t):velocity (cm/(weeks))')
plot(y_5,cdy,marker = 'o',color = 'b',)
```



Step1. 利用 Matlab Curve Fitting Tool 找到一組 α、β、a、b 和 c

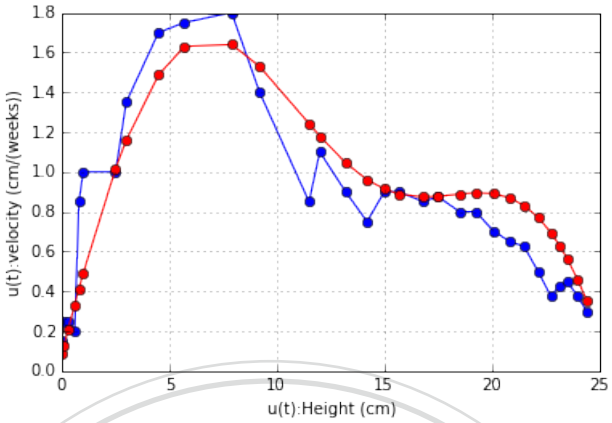
In[10]:

```
alpha = -0.011
beta = 0.2612
a = -0.5341
b = -0.3127
c = -0.1563
```

In[11]:

```
def cdy_fcn(t,y):
    return alpha*y**2+beta*y+a*sin(b*y+c)

cdy_estimate = []
for i in range(n_cdy):
    cdy_estimate.append(cdy_fcn(t_0,y_5[i]))

xlabel('u(t):Height (cm)')
ylabel('u(t):velocity (cm/(weeks))')
plot(y_5,cdy,marker = 'o',color = 'b',)
```

```
plot(y_5,cdy_estimate,marker = 'o',color = 'r',)
```



In[12]:

```
SSE_cdy0 = SSE_cdy(cdy_estimate,cdy)
print SSE_cdy0
print alpha, beta, a, b, c
```

1.11321312693

-0.011 0.2612 -0.5341 -0.3127 -0.1563

Step2. 利用上述的 α、β、a、b 和 c，搭配數值微分方程的方法來找到一組函數資料來逼近原本 Data

Setting up

In[13]:

```
t0 = 0
tf = 30
y0 = 1

#這裡的gap必須要是整數，代表每格資料間細分的次數
gap = 100
n = 30*gap+1
t = linspace(t0,tf,n)
```

Heun's method

In[14]:

```
y0_Heun = y0
y_Heun = ode_Heun(lambda x,y: cdy_fcn(x,y),t0,tf,y0_Heun,n)
SSE_Heun0 = SSE_numerical(y_Heun,y_5,gap)
print SSE_Heun0
```

22.60031044

RK2 Method

In[15]:

```
y0_RK2 = y0
y_RK2 = ode_RK2(lambda x,y: cdy_fcn(x,y),t0,tf,y0_RK2,n)
SSE_RK2a = SSE_numerical(y_RK2,y_5,gap)
print SSE_RK2a
```

22.6003363704

RK4 Method

In[16]:

```
y0_RK4 = y0
y_RK4 = ode_RK4(lambda x,y: cdy_fcn(x,y),t0,tf,y0_RK4,n)
SSE_RK4a = SSE_numerical(y_RK4,y_5,gap)
print SSE_RK4a
```

22.6003603238

Step.4 對 y_0 做擾動，希望能降低誤差

Heun's Method

In[17]:

```
count = 0
SSE_Heun = SSE_Heun0
while count <1000:

    #對起始點增加擾動
    e1 = random.uniform(-0.1,0.1)
    y0_temp = y0_Heun+e1

    #生成新的Data
    y_temp = ode_Heun(lambda x,y: cdy_fcn(x,y),t0,tf,y0_temp,n)

    #計算新的最小平方差
    SSE_temp = SSE_numerical(y_temp,y_5,gap)

    #比較最小平方差
    if SSE_temp < SSE_Heun:
        SSE_Heun = SSE_temp
        y0_Heun = y0_temp

    count += 1

print SSE_Heun
print y0_Heun
```

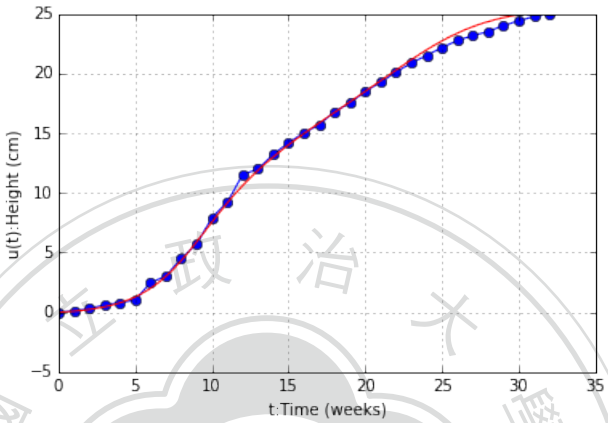
2.12606450927

-0.0151605021498

In[18]:

```
y_Heun = ode_Heun(lambda x,y: cdy_fcn(x,y),t0,tf,y0_Heun,n)
```

```
xlabel('t:Time (weeks)')
ylabel('u(t):Height (cm)')
plot(t_0,y,marker = 'o',color = 'blue')
plot(t,y_Heun,color = 'red')
```



RK2 Method

In[19]:

```
count = 0
SSE_RK2 = SSE_RK2a
while count <1000:

    #對起始點增加擾動
    e1 = random.uniform(-0.1,0.1)
    y0_temp = y0_RK2+e1

    #生成新的Data
    y_temp = ode_Heun(lambda x,y: cdy_fcn(x,y),t0,tf,y0_temp,n)

    #計算新的最小平方差
    SSE_temp = SSE_numerical(y_temp,y_5,gap)

```

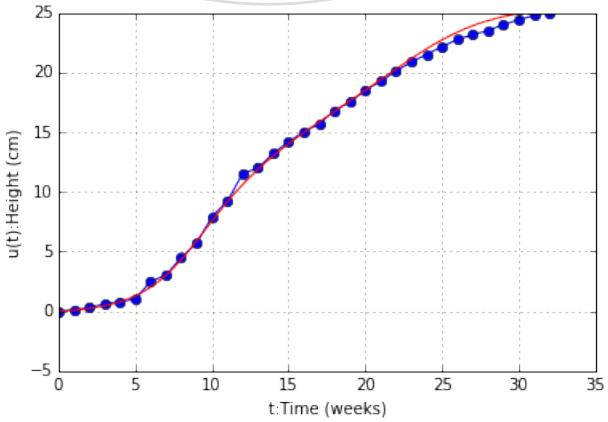
```
    #比較最小平方差
    if SSE_temp < SSE_RK2:
        SSE_RK2 = SSE_temp
        y0_RK2 = y0_temp

    count += 1

print SSE_RK2
print y0_RK2
```

2.12599915696

-0.0153907948845

In[20]:

```
y_RK2 = ode_RK2(lambda x,y: cdy_fcn(x,y),t0,tf,y0_RK2,n)
xlabel('t:Time (weeks)')
ylabel('u(t):Height (cm)')
plot(t_0,y,marker = 'o',color = 'blue')
plot(t,y_RK2,color = 'red')
```



RK4 Method

In[21]:

```
count = 0
SSE_RK4 = SSE_RK4a
while count <1000:

    #對起始點增加擾動
    e1 = random.uniform(-0.1,0.1)
    y0_temp = y0_RK4+e1

    #生成新的Data
    y_temp = ode_Heun(lambda x,y: cdy_fcn(x,y),t0,tf,y0_temp,n)

    #計算新的最小平方差
    SSE_temp = SSE_numerical(y_temp,y_5,gap)

    #比較最小平方差
    if SSE_temp < SSE_RK4:
        SSE_RK4 = SSE_temp
        y0_RK4 = y0_temp

    count += 1

print SSE_RK4
print y0_RK4
```

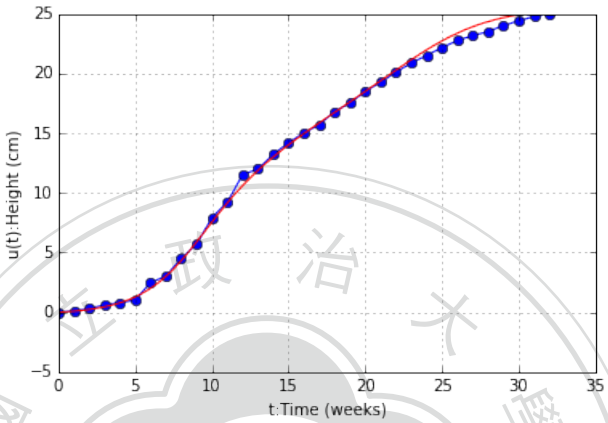
2.12601296975

-0.0152950386194

In[22]:

```
y_RK4 = ode_RK4(lambda x,y: cdy_fcn(x,y),t0,tf,y0_RK4,n)
```

```
xlabel('t:Time (weeks)')
ylabel('u(t):Height (cm)')
plot(t_0,y,marker = 'o',color = 'blue')
plot(t,y_RK4,color = 'red')
```



Bibliography

[1] 海岸綠提—水筆仔 http://163.20.52.80/stu635/cwpspage/mang/study/index.htm.

[2] William E Boyce. *Elementary differential equations and boundary value problems*. Wiley, 9th edition, 2010.

[3] Brian Bradie. *A friendly introduction to nummerical analysis*. Pearson Education, Inc., 2006.

[4] Laurence D Hoffmann. *Applied calculus for bussiness, economics, and the social and life sciences*. McGraw-Hill, eleventh edition, 2014.

[5] Tzeng Jeng-nan. 數值微分 http://glophy.com/index.php/2014-02-07-01-06-58/2014-02-07-01-07-46/79-2014-02-05-07-28-44.

[6] David Kincaid. *Numerical analysis: Mathematics of scientific computing*. Brooks/Cole, third edition, 2002.

[7] Chen Ren-fa. Nonlinear differential equation of second order and its applications. 2015.