# Numerical Solution of Initial-Value Problems for Ordinary Differential Equations using Taylor Series with Recursively Defined Coefficients

by

Raymond E. Flanery Jr.

Submitted in Partial Fulfillment of the Requirements
for the Degree of
Master of Science
in the
Mathematics
Program

John S. Bruno 8/13/86
Advisor Date

Sally M. Hotchkiss August 27 1986
Dean of the Graduate School -- Date

YOUNGSTOWN STATE UNIVERSITY

August, 1986

## ABSTRACT

**Numerical Solution of Initial-Value Problems for Ordinary Differential Equations using Taylor Series with Recursively Defined Coefficients**

**Raymond E. Flanery Jr.**

**Master of Science**

**Youngstown State University, 1986**

The Taylor series method has been neglected as a way to solve numerical initial-value problems since obtaining and evaluating the derivatives of a function is a complex and time consuming process when one uses the traditional methods to obtain the derivatives. If one instead uses recurrence relations, obtained from the mathematical operations which make up an equation, to evaluate the derivatives at specific points, then the Taylor series method is efficient and can be used to obtain information about the behavior of a solution in the complex plane about its singularities.

Software packages are available that utilize the Taylor method-, In particular, the ATOMCC toolbox,-written by Y.F. Chang, is a powerful package capable of handling almost any initial-value problem or system with extreme accuracy, even those that are stiff.

## TABLE OF CONTENTS

## LIST OF TABLES

## LIST OF SYMBOLS

| SYMBOL | DEFINITION |
|---|---|
| $\frac{d^n y}{dx^n}$ or $y^{(n)}(x)$ | The $n^{th}$ derivative of y with respect to x. |
| $\frac{\partial^{(n)} y}{\partial x^{(n)}}$ | The $n^{th}$ partial derivative of y with respect to x. |
| $n!$ | n factorial = $n(n-1)(n-2)\cdots 2.1$. |
| | Denotes the summation of the terms that follow the symbol. |
| $\alpha, \beta, \delta, \xi$ | Denote real numbers. |
| $\epsilon$ | Denotes 'element of'. |
| $\subset$ | Denotes 'subset of'. |
| $R^2$ | Two-dimensional space. |

# CHAPTER I

## FINITE TAYLOR SERIES APPROXIMATIONS TO FIRST ORDER INITIAL-VALUE PROBLEMS IN ORDINARY DIFFERENTIAL EQUATIONS

### Introduction

Finding the solution $y(t)$ to the problem

$$y'(t) = f(t,y),\ a \leq t \leq b,\ y(a) = \alpha, \tag{1}$$

where $y(a) = \alpha$ is referred to as the initial condition, is called an initial-value problem. Numerical initial-value problems are like (1), except that one is interested in finding approximations of the solution $y(t)$ at the points $t_1, t_2, \cdots, t_N$, where $t_i = a + ih$, $h = (b-a)/N$, and $i = 0, 1, \cdots, N$. Here $h$ is a fixed step size for the problem and $t_0, t_1, \ldots, t_N$ are the mesh points. Any step size used in this paper is assumed to be fixed, unless otherwise stated.

Every student of Numerical Analysis has seen problems like that in (1), along with several different methods for finding numerical approximations to the solution $y(t)$. Treatment of initial-value problems usually begins with Euler's method, Taylor series methods of order n, and then Runge-Kutta methods [1;205-207]. These methods are all referred to as one-step methods. This means that they derive their approximation at any given mesh point using only information available from the previous mesh point.

The Taylor methods have been used previously for deriving lower order numerical methods, such as Euler's and the Runge-Kutta methods [5;26], rather than for approximating solutions to initial-value problems. This is because most methods to find the Taylor series coefficients involve some form of symbolic differentiation, which produces the algebraic form of the coefficients. These formulas become complex and require extensive computation time to evaluate [5;26].

A process for deriving the necessary Taylor coefficients through the use of recurrence relations can be used instead of symbolic differentiation to make the use of Taylor methods efficient [6;24]. Taylor methods utilizing these recurrence relations are the basis for software packages, including the ATOMCC toolbox [3;215], which will be discussed later.

In the following sections the recurrence relations used to derive Taylor series coefficients are developed, and a simple, but general, algorithm for applying these relations to solve a wide range of initial-value problems is presented.

## Recursive Definition of Taylor Series Coefficients

In order to define the recurrence relations necessary to evaluate the Taylor series coefficients of a function $y(t)$, the derivative of the function must be composed of rational and/or elementary functions. It is also

necessary that $y(t)$ be analytic (infinitely differentiable) in some neighborhood of the point $t = t_0$. If these conditions are satisfied, then one may define

$$(y)_k = \frac{1}{k!}\left[\frac{d^k}{dt^k}\, y(t_0)\right], \tag{2}$$

and, in particular, one can see that

$$(y)_0 = y(t_0) \quad \text{and} \quad (y)_1 = y'(t_0). \tag{3}$$

The finite Taylor series expansion for $y(t)$ about $t = t_0$ is

$$y(t) = \sum_{k=0}^{n} y^{(k)}(t_0)\frac{(t - t_0)^k}{k!} + R_n,$$

where

$$R_n = y^{(n+1)}(\xi)\frac{(t - t_0)^{n+1}}{(n+1)!}, \text{ for some } \xi \in (t_0, t).$$

This can be rewritten as

$$y(t) = \sum_{k=0}^{n} (y)_k\,(t - t_0)^k + R_n,$$

where $(y)_k$, as defined in (2), is the $k^{th}$ Taylor coefficient.

From equations (2) and (3) one has the recurrence relation

$$(y)_k = \frac{1}{k!}\left[\frac{d^k}{dt^k}\, y(t_0)\right]$$

$$= \frac{1}{k}\left[\frac{1}{(k-1)!}\left[\frac{d^{k-1}}{dt^{k-1}}\left[\frac{d}{dt}\, y(t_0)\right]\right]\right]$$

$$= \frac{1}{k}\left[\frac{1}{(-1)}\left[\frac{d^{k-1}}{dt^{k-1}}\,(y)_1\right]\right]$$

$$= \frac{1}{k}\,((y)_1)_{k-1}. \qquad (4)$$

This relation is used in the derivation of the recurrence relations of the different rational and elementary functions.

Let $u(t)$ and $v(t)$ be arbitrary analytic functions and $\beta$ some real constant, then one has the following recurrence relations [6;26], which will be derived later.

$$(u+v)_k = (u)_k + (v)_k, \qquad (5)$$

$$(u-v)_k = (u)_k - (v)_k, \qquad (6)$$

$$(uv)_k = \sum_{j=0}^{k} (u)_j (v)_{k-j}, \qquad (7)$$

$$\left(\frac{u}{v}\right)_k = \frac{1}{(v)_0}\left[(u)_k - \sum_{j=1}^{k} (v)_k \left(\frac{u}{v}\right)_{k-j}\right], \qquad (8)$$

$$(u^\beta)_k = \frac{1}{(u)_0}\left[\sum_{j=0}^{k-1} \left(\beta - \frac{j(\beta+1)}{k}\right) (u)_{k-j} (u^\beta)_j\right], \qquad (9)$$

$$(e^u)_k = \sum_{j=0}^{k-1} \left(1 - \frac{j}{k}\right) (u)_{k-j} (e^u)_j, \qquad (10)$$

$$(\log_e u)_k = \frac{1}{(u)_0}\left[ (u)_k - \sum_{j=1}^{k-1} \left(1 - \frac{j}{k}\right)(u)_j (\log_e u)_{k-j} \right], \qquad (11)$$

$$(\sin t)_k = \frac{1}{k} \sum_{j=0}^{k-1} (j+1)\,(u)_{j+1} (\cos u)_{k-1-j}, \qquad (12)$$

$$(\cos u)_k = -\frac{1}{k} \sum_{j=0}^{k-1} (j+1)\,(u)_{j+1} (\sin u)_{k-1-j}, \qquad (13)$$

In addition to these recurrence relations, one also has

$$(\sinh u)_k = \sum_{j=0}^{k-1} \left(1 - \frac{j}{k}\right)(u)_{k-j} (\cosh u)_j, \qquad (14)$$

and

$$(\cosh u)_k = \sum_{j=0}^{k-1} \left(1 - \frac{j}{k}\right)(u)_{k-j} (\sinh u)_j. \qquad (15)$$

From these recurrence relations it is a simple matter to develop the relations for $(\tan u)_k$ and $(\tanh u)_k$.

If t is the independent variable in an initial-value problem and c is any constant, then there are the following simple, but important relations:

$$(t)_0 = t,$$

$$(t)_1 = 1,$$

$$(t)_k = 0 \quad \text{if } k > 1, \qquad (16)$$

and

$$(c)_0 = c,$$

$$(c)_k = 0 \text{ if } k > 0,$$

$$(cu)_k = c(u)_k \text{ for all } k.$$

## Derivation of the Recurrence Relations

The recurrence relations, equations (5) to (15), in the previous section can all be derived using the relations in (4), some calculus, and Leibniz's rule for the $k^{th}$ derivative of the product of two functions:

$$\frac{d^k}{dt^k}(uv)(t_0)=\sum_{j=0}^{k}\left(\frac{k!}{j!(k-j)!}\right)\left[\frac{d^j}{dt^j}u(t_0)\right]\left[\frac{d^{k-j}}{dt^{k-j}}v(t_0)\right]. \quad (17)$$

The derivation of these recurrence relations is as follows:

1. For equations (5) and (6) one has

$$(u\pm v)_k = \frac{1}{k!}\left[\frac{d^k}{dt^k}(u\pm v)(t_0)\right]$$

$$= \frac{1}{k!}\left[\frac{d^k}{dt^k}u(t_0) \pm \frac{d^k}{dt^k}v(t_0)\right]$$

$$= \frac{1}{k!}\left[\frac{d^k}{dt^k}u(t_0)\right] \pm \frac{1}{k!}\left[\frac{d^k}{dt^k}v(t_0)\right]$$

$$= (u)_k \pm (v)_k.$$

2. For equation (7) one has

$$(uv)_k = \frac{1}{k!}\left[\frac{d^k}{dt^k}(uv)(t_0)\right] .$$

Applying (17) to the right side this equation becomes

$$(uv)_k = \frac{1}{k!} \sum_{j=0}^{k} \frac{k!}{j!(k-j)!} \left[ \frac{d^j}{dt^j} u(t_0) \right] \left[ \frac{d^{k-j}}{dt^{k-j}} v(t_0) \right]$$

$$= \sum_{j=0}^{k} \frac{1}{k-j)!} \left[ \frac{d^j}{dt^j} u(t_0) \right] \left[ \frac{d^{k-j}}{dt^{k-j}} v(t_0) \right]$$

$$= \sum_{j=0}^{k} \frac{1}{j!} \left[ \frac{d^j}{dt^j} u(t_0) \right] \frac{1}{(k-j)!} \left[ \frac{d^{k-j}}{dt^{k-j}} v(t_0) \right]$$

$$= \sum_{j=0}^{k} (u)_j (v)_{k-j}.$$

3. For equation (8) if one lets $f(t) = \frac{u(t)}{v(t)}$, then

$v(t)f(t) = u(t)$; hence,

$$(vf)_k = (u)_k. \tag{18}$$

Now, if equation (7) is applied to the left side of (18), one has

$$\sum_{j=0}^{k} (v)_j (f)_{k-j} = (u)_k . \tag{19}$$

Notice that if j=0, then $(f)_{k-j} = (f)_k = (u/v)_k$. Thus, by separating the first term from the rest of the sum in (19) and replacing f by u/v one has

$$(v)_0 \left( \frac{u}{v} \right)_k + \sum_{j=1}^{k} (v)_j \left( \frac{u}{v} \right)_{k-j} = (u)_k.$$

Solving this equation for $(u/v)_k$ produces:

$$\left( \frac{u}{v} \right)_k = \frac{1}{(v)_0} \left[ (u)_k - \sum_{j=1}^{k} (v)_j \left( \frac{u}{v} \right)_{k-j} \right].$$

4. For equation (9) notice that

$$(u^{\beta})_1 = \beta(u^{\beta-1})_0(u)_1 = \frac{\beta(u^{\beta})_0(u)_1}{(u)_0}$$

Also, by reordering terms in the summation, one has

$$\sum_{j=1}^{k-1}(u)_j((v)_1)_{k-1-j} = \sum_{j=1}^{k-1}(u)_{k-j}((v)_1)_{j-1}. \qquad (20)$$

Now, from equation (4),

$$(u^{\beta})_k = \frac{1}{k}\left[\,(u^{\beta})_1\,\right]_{k-1}$$

$$= \frac{1}{k}\left[\frac{\beta(u^{\beta})_0(u)_1}{(u)_0}\right]_{k-1}.$$

Now apply equation (8) to the quantity on the right. Then

$$(u^{\beta})_k = \frac{1}{k}\left[\frac{1}{(u)_0}\left[\,(\beta(u^{\beta})_0(u)_1)_{k-1} - \sum_{j=1}^{k-1}(u)_j(\beta(u^{\beta})_0(u)_1)_{k-1-j}\,\right]\right],$$

and applying equation (7) to the term before the summation the equation becomes

$$(u^{\beta})_k = \frac{1}{k}\left[\frac{1}{(u)_0}\left[\sum_{j=0}^{k-1}\beta(u^{\beta})_j\,((u)_1)_{k-1-j} - \sum_{j=1}^{k-1}(u)_j((u^{\beta})_1)_{k-1-j}\right]\right].$$

Applying equation (20) to the second summation transforms this equation into

$$(u^{\beta})_k = \frac{1}{k}\left[\frac{1}{(u)_0}\left[\sum_{j=0}^{k-1}\beta(u^{\beta})_j\,((u)_1)_{k-1-j} - \sum_{j=1}^{k-1}(u)_{k-j}((u^{\beta})_1)_{j-1}\right]\right].$$

One final transformation is required. Apply equation (4) to the terms in both summations to produce the equation

$$(u^{\beta})_k = \frac{1}{k}\left[\frac{1}{(u)_0}\left[\sum_{j=0}^{k-1}\beta(u^{\beta})_j(k-j)(u)_{k-j} - \sum_{j=1}^{k-1}(u)_{k-j}(j)(u^{\beta})_j\right]\right]$$

$$= \frac{1}{(u)_0}\left[\sum_{j=0}^{k-1}\frac{\beta(k-j)}{k}(u^{\beta})_j(u)_{k-j} - \sum_{j=1}^{k-1}\frac{j}{k}(u)_{k-j}(u^{\beta})_j\right].$$

Notice that in the second summation, adding a term for $j = 0$ adds only zero to the sum. Hence

$$(u^{\beta})_k = \frac{1}{(u)_0}\left[\sum_{j=0}^{k-1}\frac{\beta(k-j)}{k}(u^{\beta})_j(u)_{k-j} - \sum_{j=0}^{k-1}\frac{j}{k}(u)_{k-j}(u^{\beta})_j\right]$$

and bringing the two summations together then produces the equation

$$(u^{\beta})_k = \frac{1}{(u)_0} \sum_{j=0}^{k-1} \frac{\beta(k-j)-j}{k} (u)_{k-j}(u^{\beta})_j$$

$$= \frac{1}{(u)_0} \sum_{j=0}^{k-1} \left(\beta - \frac{j(\beta+1)}{k}\right) (u)_{k-j}(u^{\beta})_j \ .$$

5. For equation (10) one can use the results of equation (4) and the fact that $(e^u)_1 = (e^u)_0(u)_1$ to obtain

$$(e^u)_k = \frac{1}{k} ((e^u)_1)_{k-1}$$

$$= \frac{1}{k} ((e^u)_0)(u)_1)_{k-1}.$$

Now, if equation (7) is applied to the right side, this becomes

$$(e^u)_k = \frac{1}{k} \sum_{j=0}^{k-1} (e^u)_j((u)_1)_{k-1-j},$$

and by applying equation (4) to the terms of the summation this equation becomes

$$(e^u)_k = \frac{1}{k} \sum_{j=0}^{k-1} (e^u)_j(k-j)(u)_{k-j} \ .$$

Hence,

$$(e^u)_k = \sum_{j=0}^{k-1} \frac{k-j}{k} (e^u)_j(u)_{k-j}$$

$$= \sum_{j=0}^{k-1} \left(1 - \frac{j}{k}\right) (e^u)_j(u)_{k-j} \ .$$

6. For equation (11) one should note that

$$(\log_e u)_1 = \frac{1}{(u)_0} (u)_1 \ . \qquad (21)$$

By application of equation (4) one obtains

$$(\log_e u)_k = \frac{1}{k} ((\log_e u)_1)_{k-1}$$

$$= \frac{1}{k} \left[ \frac{(u)_1}{(u)_0} \right]_{k-1} \ .$$

Using equation (8) this now becomes

$$(\log_e u)_k = \frac{1}{k} \left[ \frac{1}{(u)_0} \left[ ((u)_1)_{k-1} \right. \right.$$

$$\left. \left. - \sum_{j=1}^{k-1} (u)_j \left[ \frac{(u)_1}{(u)_0} \right]_{k-1-j} \right] \right] \ .$$

If equations (4) and (21) are applied to the terms of the summation, then the above equation becomes

$$(\log_e u)_k = \frac{1}{k} \left[ \frac{1}{(u)_0} \left[ ((u)_1)_{k-1} \right. \right.$$

$$\left. \left. - \sum_{j=1}^{k-1} (u)_j (k-j) (\log_e u)_{k-j} \right] \right] \ .$$

Now, distributing 1/k through the parenthesized term and applying equation (4) to the term before the summation produces the equation

$$(\log_e u)_k = \frac{1}{(u)_0}\left[\frac{k}{k}(u)_k - \sum_{j=1}^{k-1}\frac{k-j}{k}(u)_j(\log_e u)_{k-j}\right]$$

$$= \frac{1}{(u)_0}\left[(u)_k - \sum_{j=1}^{k-1}\left(1-\frac{j}{k}\right)(u)_j(\log_e u)_{k-j}\right] .$$

7. For equation (12) one has

$$(\sin u)_k = \frac{1}{k}((\sin u)_1)_{k-1}$$

$$= \frac{1}{k}((\cos u)_0(u)_1)_{k-1} ;$$

hence, from equation (7) this becomes

$$(\sin u)_k = \frac{1}{k}\sum_{j=0}^{k-1}((u)_1)_j(\cos u)_{k-1-j}$$

$$= \frac{1}{k}\sum_{j=0}^{k-1}(j+1)(u)_{j+1}(\cos u)_{k-1-j} .$$

8. For equation (13) one has

$$(\cos u)_k = \frac{1}{k}((\cos u)_1)_{k-1}$$

$$= -\frac{1}{k}((\sin u)_0(u)_1)_{k-1} .$$

Hence, from equation (7) this becomes

$$(\cos u)_k = -\frac{1}{k}\sum_{j=0}^{k-1}((u)_1)_j(\sin u)_{k-1-j}$$

$$= -\frac{1}{k}\sum_{j=0}^{k-1}(j+1)(u)_{j+1}(\sin u)_{k-1-j} .$$

9. For equation (14), by definition

$$(\sinh u)_k = \left[ \frac{e^u - e^{-u}}{2} \right]_k$$

$$= \frac{1}{2} (e^u - e^{-u})_k ,$$

so equation (6) implies that

$$(\sinh u)_k = \frac{1}{2} ((e^u)_k - (e^{-u})_k) .$$

Applying equation (10) to the terms in parenthesis produces

$$(\sinh u)_k = \frac{1}{2} \left[ \sum_{j=0}^{k-1} (1 - \frac{3}{k})(e^u)_j (u)_{k-j} \right.$$

$$\left. - \sum_{j=0}^{k-1} (1 - \frac{j}{k})(e^{-u})_j (-u)_{k-j} \right]$$

$$= \frac{1}{2} \left[ \sum_{j=0}^{k-1} (1 - \frac{j}{k})(e^u)_j (u)_{k-j} \right.$$

$$\left. + \sum_{j=0}^{k-1} (1 - \frac{j}{k})(e^{-u})_j (u)_{k-j} \right] .$$

Bringing the summations together gives the equation

$$(\sinh u)_k = \frac{1}{2}\left[\sum_{j=0}^{k-1}\left(1 - \frac{j}{k}\right)(u)_{k-j}\left((e^u)_j + (e^{-u})_j\right)\right]$$

$$= \sum_{j=0}^{k-1}\left(1 - \frac{j}{k}\right)(u)_{k-j}\left[\frac{e^u + e^{-u}}{2}\right]_j$$

$$= \sum_{j=0}^{k-1}\left(1 - \frac{j}{k}\right)(u)_{k-j}(\cosh u)_j \ .$$

10. For equation (15)

$$(\cosh u)_k = \left[\frac{e^u + e^{-u}}{2}\right]_k$$

$$= \frac{1}{2}(e^u + e^{-u})_k \ ,$$

so equation (5) implies that

$$(\cosh u)_k = \frac{1}{2}\left((e^u)_k + (e^{-u})_k\right) \ .$$

Applying equation (10) to the terms in parenthesis produces

$$(\cosh u)_k = \frac{1}{2}\left[\sum_{j=0}^{k-1}\left(1 - \frac{j}{k}\right)(e^u)_j(u)_{k-j} + \sum_{j=0}^{k-1}\left(1 - \frac{j}{k}\right)(e^{-u})_j(-u)_{k-j}\right]$$

$$= \frac{1}{2}\left[\sum_{j=0}^{k-1}\left(1 - \frac{j}{k}\right)(e^u)_j(u)_{k-j} - \sum_{j=0}^{k-1}\left(1 - \frac{j}{k}\right)(e^{-u})_j(u)_{k-j}\right] \ ,$$

Bringing the summations together gives the equation

$$(\cosh u)_k = \frac{1}{2}\left[\sum_{j=0}^{k-1}\left(1 - \frac{j}{k}\right)(u)_{k-j}\left((e^u)_j - (e^{-u})_j\right)\right]$$

$$= \sum_{j=0}^{k-1}\left(1 - \frac{j}{k}\right)(u)_{k-j}\left[\frac{e^u - e^{-u}}{2}\right]_j$$

$$= \sum_{j=0}^{k-1}\left(1 - \frac{j}{k}\right)(u)_{k-j}(\sinh u)_j .$$

## A General Algorithm for Evaluation of Taylor Series Coefficients Using the Recurrence Relations

In the previous sections recurrence relations were introduced which enable one to evaluate Taylor series coefficients for the arithmetic operators, trigonometric functions, the logarithmic function, and the exponential function. What is needed now is a way to combine these recurrence relations to evaluate the coefficients of a Taylor series which approximates the solution $y(t)$ to an initial-value problem, where $y(t)$ is composed of a finite combination of these functions and operators.

Consider the finite Taylor expansion

$$y(t) \cong \sum_{k=0}^{n}(y)_k(t - t_0)^k, \qquad (22)$$

which is an approximation of the solution to the initial-value problem (1). The Taylor coefficients $(y)_k$, $k=1,2,\cdots,n$, and the approximations at each mesh point can be obtained using the following algorithm [6;26]. An example

of applying the algorithm can be found at the end of the section.

**ALGORITHM 1**

A. Initialize $(y)_0 = a$.

B. Transform f(t,y), from problem (1), into postfix notation.

C. Generate a list of auxiliary variables $\{T_i\}$ in the following manner:

1) Let m = the number of operators in f(t,y), this includes the arithmetic operators and the elementary functions.

2) Set $T_1$ = $operator_1$ with its operands,
Set $T_2$ = $operator_2$ with its operands,
...
Set $T_m$ = $operator_a$ with its operands,
where $operator_1$ through operator, are the operators in f(t,y) in order of evaluation. These auxiliary variables are generated as if one were evaluating the postfix string.

3) If any of the above operators has recurrence relations requiring the use of other operators, such as the sine function, and if these operators are not present in the list just created, then one must generate additional auxiliary variables to handle these new operators:

Set $T_{m+1}$ = operator$_{m+1}$ with operands,

...

Set $T_{m+p}$ = operator$_{m+p}$ with operands.

D. Set $(y)_1 = T_m$.

E. For each i=1, ...,m+p, generate the code for $(T_i)_k$ from the recurrence relation corresponding to the operator in each $T_i$.

F. For each k=1, ... , n-1, set $(y)_{k+1} = \frac{1}{k+1} (T_m)_k$.

G. For each mesh point $t_j$, j=1, ... , N, one can now obtain the approximation of $y(t_j)$ by:

1) For each i=1, ... , m+p, evaluate $(T_i)_0$ at the point $t_{j-1}$.

2) Evaluate $(y)_1$.

3) For each k=1, ... , n-1

a) For each i=1, ... , m+p, evaluate $(T_i)_k$,

b) Evaluate $(y)_{k+1}$.

4) Evaluate equation (22) with $t=t_j$ and $t_0=t_{j-1}$.

EXAMPLE 1

Consider the initial-value problem

$Y' = f(t,y) = \sin(t)+\exp(-t)$, $0 \leq t \leq 1$, $y(0) = 0$.

Let n = 5, N = 10, then h = (1-0)/10 = 0.1. To find the approximation at t = 0.1 will require one step of the algorithm, which will be sufficient to see how to apply it.

A. Set $(y)_0 = 0$.

B. The postfix form of f(t,y) is:

(t,sin,t,◊,exp,+) ,

where ◊ represents the unary minus operation.

C. 1) Set m = 4, the number of operators in f(t,y).

2) Set $T_1$ = sin(t), the postfix string is now:

$(T_1,t,◊,exp,+)$ .

Set $T_2$ = -t, the postfix string is now:

$(T_1,T_2,exp,+)$ .

Set $T_3$ = exp($T_2$), the postfix string is now:

$(T_1,T_3,+)$ .

Set $T_4 = T_1 + T_3$.

3) Since $T_1$ = sin(t), add an auxiliary variable for cos(t):

Set $T_5$ = cos(t), the variable p is set to 1.

D. Set $(y)_1 = T_4$.

E. 1) Set $(T_1)_k = \frac{1}{k} \sum_{j=0}^{k-1} (j+1)(t)_{j+1}(T_5)_{k-1-j}$, (from (12)).

By application of equation (16) this becomes

$$(T_1)_k = \frac{1}{k} (T_5)_{k-1}.$$

2) Set $(T_2)_k = -(t)_k$.

By application of equation (16) this becomes

$$(T_2)_1 = -1,$$

$$(T_2)_k = 0, \text{ for all } k > 1.$$

3) Set $(T_3)_k = \sum_{j=0}^{k-1} (1 - \frac{j}{k})(T_2)_{k-j}(T_3)_j$.

By the definition in E.2 above, for j=0,···,k-2 $(T_2)_{k-j} = 0$ and for j = k-1 $(T_2)_{k-j} = -1$. Thus

$$(T_3)_k = -\frac{1}{k}(T_3)_{k-1}.$$

4) Set $(T_4)_k = (T_1)_k + (T_3)_k$.

5) Set $(T_5)_k = -\frac{1}{k}\sum_{j=0}^{k-1}(j+1)(t)_{j+1}(T_1)_{k-1-j}$.

By application of equation (16) this becomes

$$(T_5)_k = -\frac{1}{k}(T_1)_{k-1}.$$

F. Set $(y)_{k+1} = \frac{1}{k+1}(T_4)_k$.

G. For j=1, $t_0$=0.0, and $t_1$=0.1:

1) $(T_1)_0 = \sin(t_0) = \sin(0) = 0$,

$(T_2)_0 = -t_0 = 0$,

$(T_3)_0 = \exp((T_2)_0) = \exp(0) = 1$,

$(T_4)_0 = (T_1)_0 + (T_3)_0 = 0 + 1 = 1$,

$(T_5)_0 = \cos(t_0) = \cos(0) = 1$.

2) $(y)_1 = (T_4)_0 = 1$.

3) For k=1:

a) $(T_1)_1 = (T_5)_0 = 1$,

$(T_2)_1 = -1$,

$(T_3)_1 = -(T_3)_0 = -1$,

$(T_4)_1 = (T_1)_1 + (T_3)_1 = 1 - 1 = 0$,

$(T_5)_1 = -(T_1)_0 = 0$.

b) $(y)_2 = \frac{1}{2}(T_4)_1 = 0$.

For k=2:

a) $(T_1)_2 = \frac{1}{2}(T_5)_1 = 0,$

$(T_2)_2 = 0,$

$(T_3)_2 = -\frac{1}{2}(T_3)_1 = \frac{1}{2},$

$(T_4)_2 = (T_1)_2 + (T_3)_2 = \frac{1}{2},$

$(T_5)_2 = -\frac{1}{2}(T_1)_1 = -\frac{1}{2}.$

b) $(y)_3 = \frac{1}{3}(T_4)_2 = \frac{1}{6}.$

For k=3:

a) $(T_1)_3 = \frac{1}{3}(T_5)_2 = -\frac{1}{6},$

$(T_2)_3 = 0,$

$(T_3)_3 = -\frac{1}{3}(T_3)_2 = -\frac{1}{6},$

$(T_4)_3 = (T_1)_3 + (T_3)_3 = -\frac{1}{3},$

$(T_5)_3 = -\frac{1}{3}(T_1)_2 = 0.$

b) $(y)_4 = \frac{1}{4}(T_4)_3 = -\frac{1}{12}.$

For k=4:

a) $(T_1)_4 = \frac{1}{4}(T_5)_3 = 0,$

$$(T_2)_4 = 0 ,$$

$$(T_3)_4 = -\frac{1}{4}(T_3)_3 = \frac{1}{24} ,$$

$$(T_4)_4 = (T_1)_4 + (T_3)_4 = \frac{1}{24} ,$$

$$(T_5)_4 = -\frac{1}{4}(T_1)_3 = \frac{1}{24} .$$

b) $$(y)_5 = \frac{1}{5}(T_4)_4 = \frac{1}{120} .$$

4) Now to approximate y(0.1) using the above values one has

$$y = \sum_{k=0}^{5} (y)_k (0.1)^k$$

$$= 0 \cdot 1 + 1 \cdot \frac{1}{10} + 0 \cdot \frac{1}{100} + \frac{1}{6} \cdot \frac{1}{1000}$$

$$- \frac{1}{24} \cdot \frac{1}{10,000} + \frac{1}{120} \cdot \frac{1}{100,000}$$

= 0.10017167, to 8 significant digits.

The actual solution for y(0.1) to 8 significant digits is 0.10015842.

# CHAPTER II

## TAYLOR SERIES VS. RUNGE-KUTTA

### Introduction

One basis for comparing the efficiency of approximating methods for IVPs is to compare the number of functional evaluations required at each step [1;226]. Another method for comparing the efficiency of numerical methods is to determine the total arithmetic operations and elementary function evaluations required. The Taylor method requires only one evaluation of the function f(t,y) in problem (1), but the recurrence relations used to evaluate the coefficients produce many arithmetic operations. Thus, the latter comparison method will be used to evaluate the performance of the Taylor method relative to some other numerical method, in this case the Runge-Kutta Order Four method.

The operations to be counted are additions/subtractions, multiplications/divisions, and elementary function evaluations. The operations are grouped this way because of their similar computational times. To compare the Runge-Kutta Order Four to the Taylor series method, it is not realistic to consider only the total operations at each step. One must instead consider the total operations required by each method over the entire interval of

solution. The total number of operations then becomes dependent on the number of steps required to reach an acceptable approximation, one within a given error tolerance, as well as the number of operations required at each step of the solution. The total operations required by the Taylor series is determined by the length of the series being used, the type of operations present in the initial-value problem, and the number of steps involved in reaching the final approximation. The total operation count for the Runge-Kutta method is determined by the initial-value problem being approximated and the number of steps involved in the approximation.

## Runge-Kutta Order Four and its Operation Counts

The Runge-Kutta Order Four method is the most commonly used of the Runge-Kutta methods, its development can be found in [1;225]. The following is a slight modification of the procedure given in [1]. It is designed to solve the initial-value problem in (1), where $N_r$ is the partition size of the interval [a,b].

ALGORITHM 2

Set $h = (b-a)/N_r$

$t_0 = a$

$w_0 = \alpha$

For $j = 1$ to $N_r$ do

Set $t^{(1)} = t_{j-1} + h/2$

$K_1 = hf(t_{j-1}, w_{j-1})$

$$K_2 = hf(t^{(1)}, w_{j-1} + K_1/2)$$
$$K_3 = hf(t^{(1)}, w_{j-1} + K_2/2)$$
$$K_4 = hf(t_{j-1} + h, w_{j-1} + K_3)$$
$$w_j = wj-1 + (K_1 + 2(K_2 + K_3) + K_4)/6$$
$$t_j = a + jh$$

Here $w_j$ is the approximation of $y(t_j)$. Now, notice that there are 10 additions/subtractions and 9 multiplications/divisions required for each step, excluding operations required to evaluate f(t,w).

Let $A_r \equiv$ the number of additions/subtractions in f(t,y),

$M_r \equiv$ the number of multiplications/divisions in f(t,y),

and

$E_r \equiv$ the number of elementary functions in f(t,y).

Then the total number of operations required for the Runge-Kutta method to approximate y(b) is as follows:

Total add/sub = $(10 + 4A_r)N_r$,

Total mult/div = $(9 + 4M_r)N_r$,

Total elem. func. evaluations = $4E_rN_r$.

From these formulas it is obvious that, if the initial-value problem involves a fair amount of operations, a large number of steps (a very small step size) will necessarily produce high operation counts.

## Taylor Series Method Operation Counts

To compute the number of operations required by the Taylor series method one must first answer these three questions:

1. What degree Taylor polynomial will be used?
2. What recurrence relations will be required to find each of the Taylor coefficients?
3. How many steps will be required to approximate the solution to the problem?

The answers to these questions determine the total operations required by the method.

The degree of the Taylor polynomial determines the number of additions/subtractions and multiplications/ divisions required to evaluate the polynomial. A Taylor polynomial of degree n contributes n additions/subtractions and n multiplications/ divisions to the total operation count of the method at each step of the solution.

Each recurrence relation requires different types of operations to be performed, as well as different numbers of these operations. To determine the number of operations required to evaluate each of the Taylor coefficients, one must consider the number of operations each different type of recurrence relation contributes to this total.

The first coefficient, $(y)_0$, does not require any operations, its value is assigned. To evaluate the second coefficient, $(y)_1$, each recurrence relation contributes operations as follows:

1. Equations (5) and (6) contribute

   1 add/sub.

2. Equations (7) and (8) contribute

   1 mult/div.

3. Equations (9) through (15) contribute

1 functional evaluation.

To evaluate the other n-1 coefficients, $(y)_k$, k=2, ..., n, each recurrence relation contributes the following number of operations:

1. Equations (5) and (6) contribute

   1 add/sub,

   for each of the coefficients.

2. Equations (7) and (8) contribute

   k add/sub,

   k+1 mult/div.

   The recurrence relation in equation (8) requires

   1 mult/div for each of the k terms in the summation,

   k-1 add/sub to sum the terms,

   1 add/sub to finish evaluation of the brackets,

   and

   1 division by $(v)_0$.

3. Equation (9) contributes

   2k add/sub,

   3k+2 mult/div,

   to evaluate the recurrence relation.

   Since $\beta$ and k are given initially, one can treat $\frac{\beta+1}{k}$ as a constant, to be evaluated before the summation. This will require

   1 add/sub

and

1 mult/div.

To evaluate the summation requires

1 add/sub

and

3 mult/div for each of the k terms,

k–1 add/sub to sum the terms, and

1 division by $(u)_0$.

4. Equations (10), (14), and (15) have the same basic structure. Thus, they each contribute

   2k-1 add/sub,

   3k mult/div.

   If one considers the recurrence relation in equation (10), then it is easy to see that it requires

   1 add/sub

   and

   3 mult/div for each of the k terms and

   k–1 add/sub to sum these terms.

5. Equation (11) contributes

   2k-2 add/sub,

   3k-2 mult/div.

   These totals come from the

   1 add/sub

   and

   3 mult/div for each of the k–1 terms,

   k-2 add/sub to sum the terms,

   1 add/sub to finish the evaluation in the brackets,

and

1 division by $(u)_0$.

6. Equations (12) and (13) are basically the same except for the additional multiplication by -1 required in (13). Their contribution is

   2k-1 add/sub,

   2k+1 mult/div, equation (12),

   2k+2 mult/div, equation (13).

Since only worst case operation counts are considered, the recurrence relations are grouped into three types, according to the types of operations that they produce.

Type I - These are equations (5) and (6), which contribute only add/sub to the counts.

Type II - These are equations (7) and (8), which contribute both add/sub and mult/div to the counts.

Type III - These are equations (9) through (15), which contribute add/sub, mult/div, and functional evaluations to the count.

To determine the total number of operations for all n+l coefficients in the Taylor polynomial contributed from each of these three types one has the following:

Type I - The second coefficient contributes

1 add/sub.

The remaining n-1 coefficients contribute

$$\sum_{k=2}^{n} 1 = n-1 \text{ add/sub.}$$

Thus, operations in f(t,y) using recurrence relations of Type I contribute

n add/sub

to the total operation count.

Type II - The second coefficient contributes

1 mult/div,

and the remaining n-1 coefficients contribute

$$\sum_{k=2}^{n} k = \frac{n^2+n-2}{2} \text{ add/sub,}$$

and

$$\sum_{k=2}^{n} k+1 = \frac{n^2+3n-4}{2} \text{ mult/div.}$$

Thus, operations in f(t,y) using recurrence relations of Type II contribute

$$\frac{n^2+n-2}{2} \text{ add/sub,}$$

and

$$\frac{n^2+3n-2}{2} \text{ mult/div}$$

to the total operation count.

Type III - For these equations, the worst case operation counts come from equation (9), which requires more operations than any of the other Type III equations. The second coefficient contributes

1 functional evaluation,

and the remaining n-1 coefficients contribute

$$\sum_{k=2}^{n} 2k = n^2+n-2 \text{ add/sub},$$

and

$$\sum_{k=2}^{n} 3k+2 = \frac{3n^2+7n-10}{2} \text{ mult/div}.$$

Thus, operations in f(t,y) using recurrence relations of Type III contribute

$n^2+n-2$ add/sub,

$\frac{3n^2+7n-10}{2}$ mult/div,

and

1 functional evaluation

to the total operation count.

All three types of equations require an additional mult/div to solve for the coefficients $(y)_2$, $(y)_3$, ... , $(y)_n$. This is from step F of Algorithm 1.

In a particular problem being solved it is likely that the actual number of operations required will-be much smaller than the worst case counts just given. As an example, consider the recurrence relation for equation (8). If the analytic function v(t) is replaced by the independent variable t, then one has the recurrence relation

$$\left(\frac{u}{t}\right)_k = \frac{1}{t}\left[(u)_k - \sum_{j=1}^{k}(t)_j\left(\frac{u}{t}\right)_{k-j}\right] .$$

If one now applies the results of (16) to the terms in the summation, then

$$\left(\frac{u}{t}\right)_k = \frac{1}{t}\left[(u)_k - \left(\frac{u}{t}\right)_{k-1}\right] .$$

This new relation requires only 1 add/sub and 1 mult/div for each coefficient after the second, which still requires only 1 mult/div. Thus, for an $n^{th}$ order Taylor polynomial. the relation contributes a total of n-1 add/sub and n mult/div, a definite savings over the operation counts of the original relation.

To establish the operation counts for the Taylor method, one will need the following variables:

$A_t \equiv$ the number of elements of f(t,y) of Type I,

$M_t \equiv$ the number of elements of f(t,y) of Type II,

$E_t \equiv$ the number of elements of f(t,y) of Type III,

and

$N_t \equiv$ the number of steps required to obtain the solution.

The worst case total operation counts are then

1. Total additions/subtractions:

   For each of the $N_t$ steps there are

   a) n add/sub for evaluation of the Taylor polynomial,

   b) $A_t n$ add/sub from Type I equations,

   c) $M_t\left(\frac{n^2+n-2}{2}\right)$ add/sub from Type II equations,

and

d) $E_t(n^2+n-2)$ add/sub from Type III equations.

Thus,

total add/sub

$$= [n + A_t n + M_t\left(\frac{n^2+n-2}{2}\right) + E_t(n^2+n-2)]N_t$$

$$= [n(1 + A_t + \frac{M_t}{2} + E_t) + n^2(\frac{M_t}{2} + E_t) - M_t - 2E_t]N_t$$

2. Total multiplications/divisions:

For each of the $N_t$ steps there are

a) n mult/div for evaluation of the Taylor polynomial,

b) n-1 mult/div for evaluation of the coefficients $(y)_2, (y)_3, \cdots, (y)_n$,

c) $M_t\left(\frac{n^2+3n-2}{2}\right)$ mult/div from Type II equations,

and

d) $E_t\left(\frac{3n^2+7n-10}{2}\right)$ mult/div from Type III equations.

Thus,

total mult/div

$$= [n - 1 + n + M_t\left(\frac{n^2+3n-2}{2}\right) + E_t\left(\frac{3n^2+7n-10}{2}\right)]N_t$$

$$= [n(2 + \frac{3M_t}{2} + \frac{7E_t}{2}) + n^2(\frac{M_t}{2} + \frac{3E_t}{2}) - M_t - 5E_t - 1]N_t.$$

3. Total elementary function evaluations

For each of the $N_t$ steps there will be $E_t$ elementary

function evaluations. Thus,

total func. eval. = $E_t N_t$.

## Examples

In the examples that follow, all results were obtained on an IBM PC-AT in double-precision arithmetic, using a Pascal Turbo-87 compiler, which utilized the 80287 mathematics coprocessor chip. Both the Runge-Kutta and the Taylor series method were run using step sizes of $(1/2)^i$, for $i=1,\cdots,14$. For both methods, the relative error of the approximation was used to decide if the approximation was within the given error tolerance. If $y^*$ is the actual solution and y the approximation to that solution, then the relative error is

$$\frac{| y^* - y |}{| y^* |}$$

The Taylor series method was able to reach an acceptable approximation for most of the step sizes used in each of the problems. The table values were chosen from these different results on the basis of which step size produced the approximation most efficiently in respect to the worst case operation counts.

While looking over the examples and comparing the timing analysis, one should keep in mind that the run times listed are for the routines in the Appendix, which were designed to eliminate most of the unnecessary operations, while the operation counts given are worst case. For this

reason one will sometimes find a time that is less for one example than for another example having smaller operation counts.

**EXAMPLE 2**

Consider the initial-value problem

$$y' = -y^2(2t + 1),\ 0 \leq t \leq 1,\ y(0) = 4,$$

which has actual solution

$$y = \frac{1}{(t + 1/2)^2}.$$

Using Algorithm 1, the following list for $f(t,y) = y'$ can be constructed:

$$T_1 = y \cdot y, \qquad (T_1)_k = \sum_{j=0}^{k} (y)_j (y)_{k-j},$$

$$T_2 = -T_1, \qquad (T_2)_k = -(T_1)_k,$$

$$T_3 = 2t, \qquad (T_3)_k = 2(t)_k,$$

$$T_4 = T_3 + 1, \qquad (T_4)_k = (T_3)_k + (1)_k,$$

$$T_5 = T_2 T_4, \qquad (T_5)_k = \sum_{j=0}^{k} (T_2)_j (T_4)_{k-j},$$

$$(y)_1 = T_5, \qquad (y)_{k+1} = \frac{1}{k+1} (T_5)_k.$$

From this list one can see that the operation count parameters for the Taylor method are

$$A_t = 2,\ M_t = 3, \text{ and } E_t = 0,$$

and from $f(t,y)$ one has the parameters for the Runge-Kutta method

$$A_r = 2,\ M_r = 3, \text{ and } E_r = 0.$$

This example is non-linear in the dependent variable y and has a singularity of order two in the solution at the point t = -1/2. This singularity gives the Taylor method problems at the larger step sizes, but after the step size is within the radius of convergence for the Taylor series an accurate approximation is quickly obtained.

The results of the Taylor method and the Runge-Kutta method applied to this example can be found in TABLE 1. In this example the Runge-Kutta method is superior to the Taylor method for the larger tolerance, which is usually the case, but the Taylor method is superior when more accuracy is required in the approximation.

TABLE 1

OPERATION COUNTS AND TIMING ANALYSIS FOR THE TAYLOR AND RUNGE-KUTTA METHODS FOR EXAMPLE 2. THE UNIT OF TIME IS SECONDS.

| method | tol limit | # of steps | # of terms | add/ sub | mult/ div | time |
|---|---|---|---|---|---|---|
| TS | $5x10^{-4}$ | 8 | 6 | 624 | 664 | 0.17 |
| RK | $5x10^{-4}$ | 8 | -- | 144 | 168 | 0.05 |
| TS | $5x10^{-8}$ | 16 | 8 | 2,064 | 2,176 | 0.55 |
| RK | $5x10^{-8}$ | 128 | -- | 2,304 | 2,688 | 0.76 |

EXAMPLE 3

Consider the initial-value problem

$$y' = - y + t^2 + 1, \quad 0 \leq t \leq 1, \quad y(0) = 1,$$

which has actual solution

$$y = - 2e^{-t} + t^2 .$$

This example is linear in the dependent variable y so the solution has no singularities which could cause possible problems for the Taylor method. TABLE 2 shows that the Taylor method requires only a small number of steps and terms to reach an accurate solution, even at the smaller tolerance. At the larger tolerance the performance of both methods is excellent, but for the smaller tolerance the Taylor method is superior.

The operation count parameters are

$$A_t = A_r = 3,\ M_t = M_r = 1, \text{ and } E_t = E_r = 0.$$

TABLE 2

OPERATION COUNTS AND TIMING ANALYSIS FOR THE TAYLOR AND RUNGE-KUTTA METHODS FOR EXAMPLE 3. THE UNIT OF TIME IS SECONDS.

| method | tol limit | # of steps | # of terms | add/ sub | mult/ div | time |
|---|---|---|---|---|---|---|
| TS | $5x10^{-4}$ | 2 | 4 | 50 | 32 | 0.01 |
| RK | $5x10^{-4}$ | 2 | -- | 44 | 26 | 0.01 |
| TS | $5x10^{-8}$ | 2 | 8 | 134 | 100 | 0.03 |
| RK | $5x10^{-8}$ | 32 | -- | 704 | 416 | 0.17 |

EXAMPLE 4

Consider the initial-value problem

$$y' = \sin(t)+e^{-t},\ 0 \leq t \leq 1,\ y(0) = 0,$$

which has actual solution

$$y = -\cos t - e^{-t} + 2.$$

This example was chosen to show how the presence of elementary functions affects the two methods. In TABLE 3

the worst case operation counts for the Taylor method are misleading. The actual operation counts are much lower since both of the elementary functions involve only the independent variable t. The reduced recurrence relations can be found in EXAMPLE 1. TABLE 3 does show that the Runge-Kutta method is slightly better at the larger tolerance, but the Taylor method is definitely superior for the smaller tolerance. This is principally due to the number of functional evaluations required by the Runge-Kutta method.

TABLE 3

OPERATION COUNTS AND TIMING ANALYSIS FOR THE TAYLOR AND RUNGE-KUTTA METHODS FOR EXAMPLE 4. THE UNIT OF TIME IS SECONDS.

| method | tol limit | # of steps | # of terms | add/ sub | mult/ div | func. eval. | time |
|---|---|---|---|---|---|---|---|
| TS | $5x10^{-4}$ | 2 | 5 | 198 | 308 | 6 | 0.02 |
| RK | $5x10^{-4}$ | 2 | -- | 36 | 18 | 16 | 0.01 |
| TS | $5x10^{-8}$ | 2 | 8 | 468 | 728 | 6 | 0.03 |
| RK | $5x10^{-8}$ | 16 | -- | 288 | 144 | 128 | 0.13 |

EXAMPLE 5

For this last example, the initial-value problem is from [1;284], where it is used as a test problem to observe how a method handles stiff differential equations. The problem is

$$y' = -30y, \quad 0 \leq t \leq 1, \quad y(0) = 1/3.$$

The actual solution

$$y = \frac{1}{3} e^{-30t}$$

has a moderately large negative exponent, which causes the solution to decrease at a rapid rate. In TABLE 4 it is obvious that the Runge-Kutta method has a very difficult time with this problem even at the larger tolerance. At the smaller tolerance the Runge-Kutta requires an extremely small step size. Thus, the time required to reach an acceptable approximation is unreasonable. The Taylor method requires relatively little computational time at either tolerance to reach an acceptable approximation.

TABLE 4

OPERATION COUNTS AND TIMING ANALYSIS FOR THE TAYLOR AND RUNGE-KUTTA METHODS FOR EXAMPLE 5. THE UNIT OF TIME IS SECONDS.

| method | tol limit | # of steps | # of terms | add/ sub | mult/ div | time |
|---|---|---|---|---|---|---|
| TS | $5x10^{-4}$ | 16 | 11 | 1,392 | 1,376 | 0.14 |
| RK | $5x10^{-4}$ | 256 | -- | 3,584 | 3,328 | 0.42 |
| TS | $5x10^{-8}$ | 8 | 23 | 2,568 | 2,560 | 0.14 |
| RK | $5x10^{-8}$ | 2,048 | -- | 28,672 | 26,624 | 8.21 |

## Summary

**The examples in the previous section show that the Taylor method is capable of producing results superior to those of the Runge-Kutta method, especially if one is interested in highly accurate approximations. The main problem in applying the Taylor method efficiently is the need for a way to choose an appropriate step size and order for the method. The choice of these two parameters will be investigated in CHAPTER 4.**

**Assuming that one can choose the ideal step size and order for the method, the examples show that the Taylor method is most useful when one needs high accuracy in the approximation, or when the solution to the problem decays, or grows, rapidly.**

# CHAPTER III

# ERROR ANALYSIS OF TAYLOR SERIES

## Introduction

Using the Taylor series method to approximate (1) will involve a certain amount of error. The amount of error will depend upon two different types of error – round-off error and truncation error.

Round-off error is the result of finite-digit arithmetic and is a cause of error for any computations performed on a computer. Truncation error is the result of using a finite number of the Taylor series terms to approximate the value of the infinite expansion.

## Round-off Error

In [1;10-16] one will find a complete discussion of round-off error. Since this type of error is unavoidable, one should be familiar with some ways to reduce its effects on computer calculations. Two of the ways to reduce this error are:

1. Reformulation of the problem to be solved,

and

2. Reduction of the number of operations that must be performed.

Reformulation of the problem is used to avoid the subtraction of nearly equal numbers, the division by numbers with small magnitude, or the multiplication of numbers with large magnitude. In the following example [1;18] one can see the result of reformulating a problem to avoid catastrophic subtractions.

**EXAMPLE *6***

Consider the problem of approximating $e^{-5}$ using the Taylor polynomial of degree 9 with the formula:

$$e^{-5} \cong \sum_{k=0}^{9} \frac{(-1)^k 5^k}{k!} \tag{23}$$

$$= 1- 5(1- \frac{5}{2}(1- \frac{5}{3}(1- \frac{5}{4}(1- \frac{5}{5}(1- \frac{5}{6}(1- \frac{5}{7}(1- \frac{5}{8}(1- \frac{5}{9}))))))))$$

$= -1.827,$ to 4 significant digits,

or with the formula

$$e^{-5} = \frac{1}{e^5} \cong \frac{1}{\sum_{k=0}^{9} \frac{5^k}{k!}} \tag{24}$$

$$= \frac{1}{1+ 5(1+ \frac{5}{2}(1+ \frac{5}{3}(1+ \frac{5}{4}(1+ \frac{5}{5}(1+ \frac{5}{6}(1+ \frac{5}{7}(1+ \frac{5}{8}(1+ \frac{5}{9}))))))))}$$

$= 6.959 \times 10^{-3}$, to 4 significant digits.

The actual solution, to 4 significant digits, is $6.738 \times 10^{-3}$. The reason that (24) is more accurate than (23) is that the equation for (24) does not involve any subtractions.

Concerning the number of operations, one way to reduce them is to place polynomials into nested form before they are evaluated.

EXAMPLE 7

Consider the $n^{th}$ degree Taylor polynomial for $y(t)$ with $h=t-t_0$,

$$y(t) \cong \sum_{k=0}^{n} (y)_k h^k .$$

To evaluate this polynomial directly one could use the following Pascal code:

```
y := yt[0];
h1 := 1;
for k := 1 to n do
  begin
    hl := hl * h;
    y := y + yt[k] * hl
  end;
```

The final value for y is the approximation to $y(t)$, and yt[k] represents the $k^{th}$ Taylor coefficient. This algorithm requires n additions and 2n multiplications. However, suppose one used instead the Pascal code:

```
y := yt[n];
for k := n-1 downto 0 do
  y := y * h + yt[k];
```

which evaluates the polynomial in nested form. Then evaluation of the Taylor polynomial requires only n additions and n multiplications.

## Global and Local Truncation Error

Suppose $y(t)$ is the solution to the initial-value problem in (1) and $y \in C^{n+1}[a,b]$, where $C^{n+1}[a,b]$ denotes the class of functions that are n+1 times continuously differentiable on $[a,b]$. Expand $y(t)$ about the point $t=t_i$ to obtain the $n^{th}$ degree Taylor polynomial:

$$y(t_{i+1}) = y(t_i) + hy'(t_i) + \cdots + \frac{h^n}{n!} y^{(n)}(t_i) + \frac{h^{n+1}}{(n+1)!} y^{(n+1)}(\xi_i), \qquad (25)$$

where $t_i < \xi_i < t_{i+1}$. Since $y(t)$ is the solution to (1), $y^{(k)}(t) = f^{(k-1)}(t,y(t))$, for each $k=1,2, \cdots ,n+1$. Substituting for the appropriate derivatives in (25) gives

$$y(t_{i+1}) = y(t_i) + hf(t,y(t_i)) + \cdots + \frac{h^n}{n!} f^{(n-1)}(t_i,y(t_i)) + \frac{h^{n+1}}{(n+1)!} f^{(n)}(\xi_i,y(\xi_i)). \qquad (26)$$

By neglecting the term involving $f^{(n)}$ one can form the Taylor method of order n [1;216]:

Set $w_0 = \alpha$

and

$$\text{Set } w_{i+1} = w_i + hf(t_i,w_i) + \cdots + \frac{h^n}{n!} f^{(n-1)}(t_i,w_i). \qquad (27)$$

The global error associated with the Taylor method of order n is the difference between the actual solution $y(t_i)$ and the approximation $w_i$. This is the error accumulated from each of the steps taken prior to, and including, the $i^{th}$

step.

To obtain a bound on this error it is necessary to present the following three lemmas.

Lemma 1

For all $x \geq -1$ and any positive m,

$0 \leq (1+x)^m \leq e^{mx}$.

The proof of this lemma can be found in [1;208]. The following lemma is a generalization [1;208]:

Lemma 2

If s and t are positive real numbers, n is a positive integer, and $\{a_i\}_{i=0}^{k}$ is a sequence satisfying

$$a_0 \geq -\frac{t}{s}, \text{ and } a_{i+1} \leq (1+s)^n a_i + t \text{ for } i=0,1,\cdots,k,$$

then

$$a_{i+1} \leq e^{sn(i+1)} \left[ \frac{t}{s} + a_0 \right] - \frac{t}{s}.$$

Proof

If i is fixed, then

$$\begin{aligned} a_{i+1} &\leq (1+s)^n a_i + t \\ &\leq (1+s)^n \left[ (1+s)^n a_{i-1} + t \right] \\ &\cdots \\ &\leq (1+s)^n \left[(1+s)^n[\ldots[(1+s)^n a_0 + t]+\cdots+ t]+ t\right] + t \\ &= (1+s)^{n(i+1)} a_0 + [1 + (1+s)^n +\cdots+ (1+s)^{in}]t. \end{aligned}$$

Now,

$$1 + (1+s)^n + (1+s)^{2n} +\cdots+ (1+s)^{in} = \sum_{j=0}^{i} (1+s)^{jn},$$

which is a geometric series with ratio $(1+s)^n$ and sums to

$$\frac{1-(1+s)^{n(i+1)}}{1-(1+s)^n} = \frac{(1+s)^{n(i+1)}-1}{(1+s)^n-1}$$

Thus,

$$\begin{aligned} a_{i+1} &\leq (1+s)^{n(i+1)}a_0 + \frac{(1+s)^{n(i+1)}-1}{(1+s)^n-1}\, t \\ &\leq (1+s)^{n(i+1)}a_0 + \frac{(1+s)^{n(i+1)}-1}{s}\, t \\ &= (1+s)^{n(i+1)}a_0 + \frac{t}{s}(1+s)^{n(i+1)} - \frac{t}{s} \\ &= (1+s)^{n(i+1)}\left[a_0 + \frac{t}{s}\right] - \frac{t}{s}\,; \end{aligned}$$

therefore, by Lemma 1,

$$a_{i+1} \leq e^{sn(i+1)}\left[\frac{t}{s} + a_0\right] - \frac{t}{s}$$

Lemma 3

Let $y \in C^1[a,b]$ be a solution to the initial-value problem in (1), where f is defined on $D \subset R^2$,

$$D = \{(t,y) \mid a \leq t \leq b, -\infty < y < \infty\},$$

and f has continuous partial derivatives of all orders less than or equal to n. Then for each $k=0,1,\cdots,n-1$ there exists a non-negative real constant $L_k$ such that

$$|f^{(k)}(t,y_2) - f^{(k)}(t,y_1)| \leq L_k|y_2 - y_1|$$

whenever $(t,y_1),(t,y_2) \in D$.

**Proof**

It follows immediately that

$$\left| \frac{\partial f^{(k)}}{\partial y}(t,y) \right| \leq L_k ,$$

for some constant $L_k > 0$ and for all $k=0,1,\cdots,n-1$. With t held fixed, each $f^{(k)}(t,y)$ is a function of the single variable y and thus, applying the Mean Value Theorem, there exists a number $\xi$, $y_1 < \xi < y_2$, such that

$$\frac{\partial f^{(k)}}{\partial y}(t,\xi) = \frac{f^{(k)}(t,y_2) - f^{(k)}(t,y_1)}{y_2 - y_1}$$

whenever $(t,y_1),(t,y_2) \in D$.

This implies that

$$|f^{(k)}(t,y_2) - f^{(k)}(t,y_1)| = \left|\frac{\partial f^{(k)}}{\partial y}(t,\xi)\right| |y_2 - y_1|$$

$$\leq L_k |y_2 - y_1| \quad \blacksquare$$

A definition is required before the following theorem [1;209] can be used to derive the desired bound for the global error of the Taylor method.

**Definition 1**

A function $f(t,y)$ is said to satisfy a Lipschitz condition in the variable y on a set $D \subset R^2$, provided a constant $L > 0$ exists with the property that

$$|f(t,y_1) - f(t,y_2)| \leq L|y_1 - y_2|$$

whenever $(t,y_1),(t,y_2) \in D$. The constant L is called a Lipschitz constant for f. [1;201]

**Theorem 1**

Let $y(t)$ denote the unique solution to the initial-value problem

$$y'(t) = f(t,y(t)),\ a \leq t \leq b,\ y(a) = \alpha,$$

and $w_0, w_1, \cdots, w_N$ be the approximations generated by the Taylor method of order n for some positive integer N. If in addition y satisfies the hypotheses of Lemma 3, then there exists non-negative constants M and L such that

$$|y^{(n+1)}(t)| \leq M, \text{ for all } t \in [a,b]$$

and

$$|y_i - w_i| \leq \frac{h^n M}{L(n+1)!}\,[e^{nL(t_i-a)}-1], \text{ for } i=0,1,\cdots,N. \quad (28)$$

**Proof**

By assumption , $y^{(n+1)}$ is continuous, so there exists $M \geq 0$ such that

$$|y^{(n+1)}(t)| \leq M, \text{ for all } t \in [a,b].$$

When $i=0$, $y(t_0) = w_0 = a$, so inequality (28) is true for $i=0$.

From (25), for $i=0,\cdots,N-1$,

$$y(t_{i+1}) = y(t_i) + hf(t,y(t_i)) + \cdots + \frac{h^n}{n!} f^{(n-1)}(t_i,y(t_i)) + \frac{h^{n+1}}{(n+1)!} y^{(n+1)}(\xi_i),$$

and from the equations in (27),

$$w_{i+1} = w_i + hf(t_i,w_i) + \cdots + \frac{h^n}{n!} f^{(n-1)}(t_i,w_i).$$

Using the notation $y_i = y(t_i)$ one finds that

$$y_{i+1} - w_{i+1} = y_i - w_i + h[f(t_i,y_i) - f(t_i,w_i)] + \cdots + \frac{h^n}{n!}[f^{(n-1)}(t_i,y_i) - f^{(n-1)}(t_i,w_i)] + \frac{h^{n+1}}{(n+1)!}y^{(n+1)}(\xi_i).$$

Hence,

$$|y_{i+1} - w_{i+1}| \leq |y_i - w_i| + h|f(t_i,y_i) - f(t_i,w_i)| + \cdots + \frac{h^n}{n!}|f^{(n-1)}(t_i,y_i) - f^{(n-1)}(t_i,w_i)| + \frac{h^{n+1}}{(n+1)!}|y^{(n+1)}(\xi_i)| \qquad (29)$$

Applying Lemma 3 and using the fact that $|y^{(n+1)}(t)| \leq M$, equation (29) becomes

$$|y_{i+1} - w_{i+1}| \leq |y_i - w_i| + hL_0|y_i - w_i| + \cdots + \frac{h^n}{n!}L_{n-1}|y_i - w_i| + \frac{h^{n+1}M}{(n+1)!}$$

$$= |y_i - w_i|\left(1 + hL_0 + \frac{h^2}{2!}L_1 + \cdots + \frac{h^n}{n!}L_{n-1}\right) + \frac{h^{n+1}M}{(n+1)!}.$$

Let $L = \max\{ L_0, L_1, \ldots, L_{n-1}, 1 \}$, then

$$|y_{i+1} - w_{i+1}| \leq |y_i - w_i|\left(1 + hL + \frac{(hL)^2}{2!} + \cdots + \frac{(hL)^n}{n!}\right) + \frac{h^{n+1}M}{(n+1)!}$$

$$\leq |y_i - w_i|\,(1 + hL)^n + \frac{h^{n+1}M}{(n+1)!}.$$

Applying Lemma 2 with $a_i = |y_i - w_i|$ for each $i=0,\cdots,N$, $s = hL$, and $t = \frac{h^{n+1}M}{(n+1)!}$, one has

$$|y_{i+1} - w_{i+1}| \leq e^{hLn(i+1)}\left(\frac{h^{n+1}M}{hL(n+1)!} + |y_0 - w_0|\right) - \frac{h^{n+1}M}{hL(n+1)!},$$

and since $|y_0 - w_0| = 0$, and $h(i+1)=(t_{i+1}-t_0)=(t_{i+1}-a)$,

$$|y_{i+1} - w_{i+1}| \leq \frac{h^nM}{L(n+1)!}\left(e^{nL(t_{i+1} - a)} - 1\right)$$

for each $i=0,\cdots,N-1$ ∎

The bound given in Theorem 1 shows that, neglecting round-off error, the global error for the Taylor method of order n is $O(h^n)$.

The local truncation error for the Taylor method of order n [1;218] is the difference between the exact solution $y(t_i)$ and the approximation at the $i^{th}$ step, assuming that the value from the previous step is exact. Under the same assumptions as in Theorem 1, the local truncation error, $\tau_{i+1}$, at the $(i+1)^{th}$ step is defined as

$$\tau_{i+1} = \frac{y_{i+1} - y_i}{h} - T^{(n)}(t_i, y_i) = \frac{h^n}{(n+1)!} f^{(n)}(\xi_i, y(\xi_i))$$

for each i=0,···,N-1, where

$$T^{(n)}(t_i, y_i) = f(t_i, y_i) + \frac{h}{2} f'(t_i, y_i) + \cdots + \frac{h^{n-1}}{n!} f^{(n-1)}(t_i, y_i). \qquad (30)$$

Thus, the local truncation error for this method is such that $\tau_i = O(h^n)$.

## Stability and Convergence

In this section the following definition is required.

**Definition 2**

The Taylor method of order n is said to be convergent with respect to the differential equation it approximates if

$$\lim_{h \to 0} \max_{1 \leq i \leq N} |y_i - w_i| = 0,$$

where $y_i$ and $w_i$ are the same as in Theorem 1. [1;271]

To see that the Taylor series method is convergent under the hypothesis of Theorem 1 one requires inequality (28), which gives

$$\max_{1 \leq i \leq N} |y_i - w_i| \leq \frac{h^n M}{L(n+1)!} \left( e^{nL(b-a)} - 1 \right).$$

Since h is the only non-constant, this tends to zero with h. Hence the Taylor method of order n is convergent as long as

the differential equation being solved satisfies the conditions of Theorem 1.

**Definition 3**

A method is stable if small changes in the initial conditions of an initial-value problem produce correspondingly small changes in the approximations of the problem. [1;272]

The following theorem [1;272], whose proof is not given therein, can be used to establish stability.

**Theorem 2**

Suppose the initial-value problem

$$y' = f(t,y),\ a \leq t \leq b,\ y(a) = a$$

is approximated by a one-step method in the form

$$w_0 = \alpha$$

$$w_{i+1} = w_i + h\Phi(t_i,w_i,h). \tag{31}$$

If a number $h_0 > 0$ exists and $\Phi(t,w,h)$ is continuous and satisfies a Lipschitz condition in the variable w on the set

$$D = \{(t,w,h) \mid a \leq t \leq b,\ -\infty < w < \infty,\ 0 \leq h \leq h_0\},$$

then the method is stable.

**Proof**

Let $\{u_i\}_{i=1}^N$ and $\{v_i\}_{i=1}^N$ satisfy (31) and let i be fixed, then upon subtraction one obtains

$$|u_{i+1} - v_{i+1}| \leq |u_i - v_i| + h|\Phi(t,u_i,h) - \Phi(t,v_i,h)|. \tag{32}$$

Since $\Phi(t,w,h)$ satisfies a Lipschitz condition in the variable w, there exists a positive constant L such that

$$|\Phi(t,u_i,h) - \Phi(t,v_i,h)| \leq L|u_i - v_i|.$$

So inequality (32) becomes

$$\begin{aligned} |u_{i+1} - v_{i+1}| &\leq |u_i - v_i| + hL|u_i - v_i| \\ &\leq |u_i - v_i| + h_0L|u_i - v_i| \\ &= |u_i - v_i| \ ( \ 1 + h_0L \ ) \\ &\leq |u_{i-1} - v_{i-1}| \ ( \ 1 + h_0L \ )^2 \\ &\cdots \\ &\leq |u_0 - v_0| \ ( \ 1 + h_0L \ )^{i+1}. \end{aligned}$$

Now let $K = ( \ 1 + h_0L \ )^{i+1}$. Then

$$|u_{i+1} - v_{i+1}| \leq K|u_0 - v_0|.$$

Thus, any small changes in the initial conditions $u_0$ and $v_0$ produce correspondingly small changes in the approximations $u_{i+1}$ and $v_{i+1}$. ■

The Taylor method of order n defined in (27) is stable when the initial-value problem being solved satisfies the hypothesis of Lemma 3. For this method define

$$\Phi(t,w,h) = T^{(n)}(t,w),$$

where $T^{(n)}$ is defined in (30). Then for any $h_0 > 0$, $\Phi$ is continuous on

$$D = \{(t,w,h) \mid a \leq t \leq b, \ -\infty < w < \infty, \ 0 \leq h \leq h_0 \}.$$

Since the method does satisfy the hypothesis of Lemma 3 and since

$$\begin{aligned} |\Phi(t,w_1,h) - \Phi(t,w_2,h)| &= |f(t,w_1) - f(t,w_2)| \\ &\quad + \frac{h}{2} |f'(t,w_1) - f'(t,w_2)| + \cdots \\ &\quad + \frac{h^{n-1}}{n!} |f^{(n-1)}(t,w_1) - f^{(n-1)}(t,w_2)|, \end{aligned}$$

the results of Lemma 3 lead to

$$|\Phi(t,w_1,h) - \Phi(t,w_2,h)| \leq L_0|w_1 - w_2| + \frac{h}{2} L_1|w_1 - w_2|$$

$$+\cdots+ \frac{h^{n-1}}{n!} L_{n-1}|w_1 - w_2|$$

$$= |w_1 - w_2| \left(L_0 + \frac{h}{2} L_1 +\cdots+ \frac{h^{n-1}}{n!} L_{n-1}\right)$$

$$\leq |w_1 - w_2| \left(L_0 + \frac{h_0}{2} L_1 +\cdots+ \frac{h_0^{n-1}}{n!} L_{n-1}\right).$$

Thus $\Phi$ satisfies a Lipschitz condition in the variable w on the set D for any $h_0 > 0$ with Lipschitz constant

$$L = \left(L_0 + \frac{h_0}{2} L_1 +\cdots+ \frac{h_0^{n-1}}{n!} L_{n-1}\right).$$

So Theorem 2 implies that the Taylor method of order n is stable.

# CHAPTER IV

## ATOMCC: UTILIZING THE TAYLOR SERIES METHOD

### Introduction

Y.F. Chang [2;80-138] has derived methods used in the ATOMCC toolbox [3;215] to locate the position and order of non-essential singularities in the solution of an ordinary differential equation (ODE). He also discusses some heuristic approaches to finding the optimum step size in the case that the solution of the ODE, or system of ODEs, is either an entire function or possesses an essential singularity.

ATOMCC uses a thirty term series, unless the user specifies some other number of terms or ATOMCC discovers that the ODE, or system of ODEs, is stiff, in which case the number of terms used in the series is reduced to fifteen. A thirty term series allows ATOMCC to use a very large step size, relative to most numerical methods, which decreases the chance of computer round-off error. This long series length also enables ATOMCC to accurately estimate the radius of convergence for the series at each step, thus allowing for accurate control of the local truncation error [2;141].

ATOMCC allows one to solve ODEs in the complex plane, thus enabling one to obtain information about the behavior of the system near singularities other than just

along the real axis [3;222].

The input for ATOMCC is structured so as to be easily used, and all inputs are in the form of FORTRAN code. There are several input blocks to allow one to

1. specify the ODE or system of ODEs,
2. specify the initial conditions or input statements to be inserted into the FORTRAN source code to be generated,
3. redefine some of the default parameters of the package, such as the error limit or the number of terms in the series,

and

4. structure the output of information from the package.

After the input to the system has been specified, the ATOMCC program generator is used to produce a FORTRAN source program to be compiled and linked with certain subroutine libraries in the ATOMCC toolbox.

The program generated by ATOMCC contains most of the code necessary for solving the ODEs, the recurrence relations, structure of the system, etc. There are two external program calls to the routines RDCV and RSET. RDCV contains the routines to estimate the radius of convergence of the truncated Taylor series. RSET contains the routines necessary to choose the optimum step size for a particular expansion based upon the radius of convergence, the length of the Taylor series being used, and the error tolerance specified for the problem.

The RDCV routines estimate the radius of convergence if the problem has a single singularity on the circle of convergence, a conjugate pair of singularities on the circle of convergence, an essential singularity on the circle of convergence, or if the solution is an entire function.

## Locating Non-Essential Singularities and Their Order

When a Taylor polynomial is used to approximate the solution to an initial-value problem it is effected by singularities occuring in the solution of the problem. These singularities, if the solution is real valued on the real axis, occur only on the real axis or in conjugate pairs [4;122-123].

In the methods to be discussed, it is assumed that only the primary singularities, those on the circle of convergence, have any significant effect on the terms of the problem's Taylor series. This assumption is accurate if a sufficiently long series is used [4;123].

Only two of Chang's methods are described here, the two-term analysis [2;89] and the three-term analysis [2;91-92). Both of these methods are able to find the radius of convergence for a series when the solution has only a single singularity on the circle of convergence. In addition to the radius of convergence, the three-term analysis also gives the order of the singularity, whereas the two-term analysis requires that this order be known beforehand to obtain the radius of convergence. These methods are both

derived through analysis of the model problem

$$y(t) = (t-a)^{-s}. \tag{33}$$

This equation adequately approximates the solution to the initial-value problem being solved as long as there is a single singularity on the circle of convergence and the series Is evaluated at a point near the singularity [2;83-84]. For a complete analysis of Chang's methods one should consult [2;80-138].

The two term analysis is derived by considering the recurrence relations one obtains for the Taylor coefficients of (33). The derivatives of (33) have the form

$$y' = -s(t-a)^{-s-1},$$

$$y'' = -s(-s-1)(t-a)^{-s-2}$$

$$= -s(t-a)^{-s-1}(-s-1)/(t-a)$$

$$= y'[(-s-1)/(t-a)]$$

...

$$y^{(n)} = y^{(n-1)}[(-s-n+1)/(t-a)],$$

so that the general recurrence relation for the Taylor coefficients, evaluated at $t = t_0$, is

$$(y)_n = \frac{y^{(n)}(t_0)}{n!}$$

$$= \frac{y^{(n-1)}(t_0)}{(n-1)!} \cdot \frac{s+n-1}{n(a-t_0)}$$

$$= \frac{(y)_{n-1}}{a-t_0} \cdot \frac{s+n-1}{n}. \tag{34}$$

Solving for $d = a - t_0$ this equation becomes

$$d = \frac{(y)_{n-1}}{(y)_n} \cdot \frac{n+s-1}{n}$$

Since the radius of convergence, p, is $|a-t_0|$ one has

$$\rho = |d| = \left| \frac{(y)_{n-1}}{(y)_n} \cdot \frac{n+s-1}{n} \right| . \qquad (35)$$

This is the formula for the two-term analysis.

When one does not possess a priori knowledge of the singularity's order, then two copies of equation (34) can be solved simultaneously to yield the radius of convergence, p, and the order of the singularity, s. The two equations to be solved are

$$(y)_n = \frac{n+s-1}{nd} (y)_{n-1}$$

and

$$(y)_{n-1} = \frac{n+s-2}{(n-1)d} (y)_{n-2} .$$

Multiplying the equations by $n/(y)_{n-1}$ and $(n-1)/(y)_{n-2}$, respectively, produces the pair of equations

$$n \frac{(y)_n}{(y)_{n-1}} = \frac{n+s-1}{d}$$

and

$$(n-1) \frac{(y)_{n-1}}{(y)_{n-2}} = \frac{n+s-2}{d}$$

Subtracting the second equation from the first produces

$$n \frac{(y)_n}{(y)_{n-1}} - (n-1) \frac{(y)_{n-1}}{(y)_{n-2}} = \frac{n+s-1-n-s+2}{d} = \frac{1}{d}$$

Thus,

$$\rho = |d| = \frac{1}{\left| n \frac{(y)_n}{(y)_{n-1}} - (n-1) \frac{(y)_{n-1}}{(y)_{n-2}} \right|} \qquad (36)$$

Now, by solving equation (34) for s, one has the equation

$$s = \frac{n\,(y)_n}{(y)_{n-1}}\, d - n + 1 \ ,$$

where d is obtained from equation (36).

EXAMPLE 8

Consider the initial-value problem of Example 3 (Chapter 2). The solution has a singularity of order 2 at the point t = -1/2. Tables 5,6, and 7 show the results of applying the three term analysis to the Taylor coefficients for this problem using series lengths of 6, 11, 16,···, 41 (ie. n=5, 10, 15,···, 40 in equations (36) and (37)).

TABLE 5

THREE-TERM ANALYSIS ESTIMATES FOR THE RADIUS OF CONVERGENCE AND ORDER OF THE SINGULARITY FROM THE TAYLOR SERIES EXPANSION ABOUT THE POINT t = 0.0 FOR THE INITIAL-VALUE PROBLEM:

$$y' = -(2t+1)y^2, \ 0 \leq t \leq 1, \ y(0)=4$$

| n | radius of convergence estimate | order of singularity-- estimate |
|---|---|---|
| 5 | 5.00000000000000E-001 | 2.00000000000000E+000 |
| 10 | 5.00000000000000E-001 | 2.00000000000000E+000 |
| 15 | 5.00000000000000E-001 | 2.00000000000000E+000 |
| 20 | 5.00000000000000E-001 | 2.00000000000000E+000 |
| 25 | 5.00000000000000E-001 | 2.00000000000000E+000 |
| 30 | 5.00000000000000E-001 | 2.00000000000000E+000 |
| 35 | 5.00000000000000E-001 | 2.00000000000000E+000 |
| 40 | 5.00000000000000E-001 | 2.00000000000000E+000 |

TABLE 6

THREE-TERM ANALYSIS ESTIMATES FOR THE RADIUS OF CONVERGENCE AND ORDER OF THE SINGULARITY FROM THE TAYLOR SERIES EXPANSION ABOUT THE POINT **t = 0.4** FOR THE INITIAL-VALUE PROBLEM:

$$y' = -(2t+1)y^2,\ 0 \leq t \leq 1,\ y(0)=4$$

| n | radius of convergence estimate | order of singularity estimate |
|---|---|---|
| 5 | 9.00000000000001E-001 | 2.00000000000001E+000 |
| 10 | 9.00000000000006E-001 | 2.00000000000007E+000 |
| 15 | 9.00000000000041E-001 | 2.00000000000069E+000 |
| 20 | 8.99999999999914E-001 | 1.99999999999804E+000 |
| 25 | 8.99999999999966E-001 | 1.99999999999898E+000 |
| 30 | 9.00000000000308E-001 | 2.00000000001044E+000 |
| 35 | 8.99999999999094E-001 | 1.99999999996437E+000 |
| 40 | 8.99999999999888E-001 | 1.99999999999505E+000 |

**One can see from the tables that the approximations for the radius of convergence and order of the singularity by the three-term analysis is extremely good, even when taken from points not very close to the singularity.**

TABLE 7

THREE-TERM ANALYSIS ESTIMATES FOR THE RADIUS OF CONVERGENCE AND ORDER OF THE SINGULARITY FROM THE TAYLOR SERIES EXPANSION ABOUT THE POINT **t = 0.9** FOR THE INITIAL-VALUE PROBLEM:

$$y' = -(2t+1)y^2,\ 0 \leq t \leq 1,\ y(0)=4$$

| n | radius of convergence estimate | order of singularity estimate |
|---|---|---|
| 5 | 1.40000000000000E+000 | 1.99999999999999E+000 |
| 10 | 1.40000000000000E+000 | 2.00000000000002E+000 |
| 15 | 1.39999999999998E+000 | 1.99999999999977E+000 |
| 20 | 1.40000000000002E+000 | 2.00000000000031E+000 |
| 25 | 1.39999999999987E+000 | 1.99999999999757E+000 |
| 30 | 1.40000000000013E+000 | 2.00000000000271E+000 |
| 35 | 1.39999999999952E+000 | 1.99999999998769E+000 |
| 40 | 1.40000000000014E+000 | 2.00000000000395E+000 |

If the three-term analysis fails (i.e. two different estimates do not agree), then it is assumed that there is a conjugate pair, or more complex structure, of singularities on the circle of convergence. Chang derives the four-term analysis [2;104] and the six-term analysis [2;108] to handle this situation. In a manner similar, but more complex, to the development of the two and three-term analysis, these methods are derived from the model problem

$$y(t) = (t - a)^{-s} (t - \bar{a})^{-s},$$

where a is the complex conjugate of a. Chang also derives a method for finding the radius of convergence if the problem contains an essential singularity [2;135].

Once the radius of convergence for a particular step has been estimated, the optimum step size can be computed. If $\delta$ is the error tolerance specified at the start of the problem and $\rho$ is the radius of convergence found for the problem, then Chang [2;141-143] shows that

$$h = \rho(\delta)^{1/n}$$

is a good estimate of the optimum step size for the problem, where n+1 is the number of terms in the Taylor series being used.

EXAMPLE 9

This example considers using the ATOMCC toolbox to solve Example 3. The error tolerances used are 1.0E-4, 1.0E-8, and 1.0E-12. The ATOMCC results are in TABLE 8. Using the information above to find the optimum step size, and knowing that the solution to the problem has a

singularity at t = -0.5, the initial step size used for each of the tolerances should be

1. $h = 0.5(1.0E\text{-}4)^{1/29} = .363947$ ,
2. $h = 0.5(1.0E\text{-}8)^{1/29} = .264915$ ,

and

3. $h = 0.5(1.0E\text{-}12)^{1/29} = .192831$ .

TABLE 8

RESULTS OF ATOMCC APPLIED TO THE INITIAL-VALUE PROBLEM

$y' = -(2t + 1)y^2$ , $0 \leq t \leq 1$, $y(0) = 4$

| error tolerance | number of steps | initial step size | relative global error |
|---|---|---|---|
| 1.0E-4 | 3 | 0.36 | 1.1E-3 |
| 1.0E-8 | 3 | 0.26 | 4.1E-8 |
| 1.0E-12 | 4 | 0.192 | 5.5E-12 |

TABLE 8 shows that the ATOMCC package can solve an initial-value problem in very few steps while maintaining tight control over the global error.

# CHAPTER V

## Summary

As seen in the first two chapters, the recurrence relations used to obtain the Taylor coefficients are not difficult to understand or to implement. These relations make it possible to efficiently use the Taylor method to solve initial-value problems. Using long Taylor series to solve the initial-value problems allows for very good error control and also allows one to obtain information about the behavior of the system near singularities in the solution's complex plane.

Y.F. Chang's ATOMCC toolbox utilizes the Taylor method to do much more than the standard software packages available for solving initial-value problems and is efficient in terms of computer time.

# APPENDIX

## Pascal Routines for Comparison of the Taylor Series Method and the Runge-Kutta Order Four Method

```
function actual1(alpha,lend,rend:real):real;
begin
  actual := alpha * exp(300 * rend)
end;

function actual2(alpha,lend,rend:real):real;
begin
  actual := 1 / (rend * rend + rend + 1/alpha)
end;

function actual3(alpha,lend,rend:real):real;
begin
  actual:= -2*exp(-rend) + rend*rend - 2*rend + 3
end;

function actual4(alpha,lend,rend:real):real;
begin
  actual := 1 - exp(-rend)
end;

function actual5(alpha,lend,rend:real):real;
begin
  actual := rend + exp(-rend)
end;

function actual6(alpha,lend,rend:real):real;
begin
  actual := rend/(1-ln(rend))
end;

function actual7(alpha,lend,rend:real):real;
begin
  actual := -cos(rend)-exp(-rend)+2
end;

function actual8(alpha,lend,rend:real):real;
begin
  actual := rend*rend*(exp(rend)+exp(1)) - 2*exp(1)
end;

function actual9(alpha,lend,rend:real):real;
begin
  actual := alpha * exp(-30 * rend)
end;

function actual10(alpha,lend,rend:real):real;
begin
  actual := alpha * exp(-300 * rend)
end;
```

```
function f1(t,y:real):real;
begin { f }
  f := 300 * y
end; { f }

function f2(t,y:real):real;
begin { f }
  f := - (2 * t + 1) * y * y
end; { f }

function f3(t,y:real):real;
begin { f }
  f := -y+t*t+1
end; { f }

function f4(t,y:real):real;
begin { f }
  f := -y+1
end; { f }

function f5(t,y:real):real;
begin { f }
  f := -y + t + 1
end; { f }

function f6(t,y:real):real;
begin { f }
  f := (y/t)*(y/t) + y/t
end; { f }

function f7(t,y:real):real;
begin { f }
  f := sin(t) + exp(-t)
end; { f }

function f8(t,y:real):real;
begin { f }
  f := 2*(y+2*exp(1))/t + t*t * exp(t)
end; { f )

function f9(t,y:real):real;
begin { f }
  f := -30 * y
end; { f }

function f10(t,y:real):real;
begin { f }
  f := -300 * y
end; { f )
```

```
procedure yt1(k:integer);

{ probl: y' = 300*y,   0 <= t <= 1,  y(0)=1 }

begin { yt }
  if k = 0 then
     ayt[0] := y
  else
     ayt[k] := 300 * ayt[k-1]/k
end; { yt }
```

```
   at1,at2,at3,at4,att:array [0 .. 34] of real;
procedure yt2(k:integer);

{ prob2: y' = -(2*t + 1) * y*y,   0 <= t <= 1,  y(0)=4 }

procedure tt(k:integer);
begin
  if k = 0 then
     att[0]:= t
  else
     if k = 1 then
        att[1] := 1
     else
        att[k] := 0
end;
procedure t1(k:integer);
begin
  at1[k] := 2*att[k];
end;
procedure t2(k:integer);
begin
  if k = 0 then
     at2[k] := at1[k] + 1
  else
     at2[k] := at1[k]
end;
procedure t3(k:integer);
var
   j:integer;
begin
   at3[k] := 0;
   for j := 0 to k do
     at3[k] := at3[k] + ayt[j]*ayt[k-j]
end;
procedure t4(k:integer);
var j:integer;
begin
  at4[k] := 0;
  for j := 0 to k do
    at4[k] := at4[k] + at2[j]*at3[k-j]
end;
begin { main - yt )
  if k = 0 then
     ayt[k] := y
  else
     begin
       tt(k-1);
       t1(k-1);
       t2(k-1);
       t3(k-1);
       t4(k-1);
       ayt[k] := -at4[k-1]/k
     end
end; { yt }
```

```
   at1,at2,at3,att:array [0 .. 34] of real;
procedure yt3(k:integer);

{ prob3: y' = -y + t*t + 1,   0<=t<=1,   y(0)=1 }

procedure tt(k:integer);
begin
  if k = 0 then
     att[0]:= t
  else
     if k = 1 then
        att[1] := 1
     else
        att[k] := 0
end;
procedure t1(k:integer);
var j,m:integer;
begin
  if k > 2 then
     m := 2
  else
     m := k;
  at1[k] := 0;
  for j := 0 to m do
    at1[k] := at1[k] + att[j]*att[k-j]
end;
procedure t2(k:integer);
begin
  at2[k] := at1[k] - ayt[k]
end;
procedure t3(k:integer);
begin
  if k = 0 then
     at3[k] := at2[k] + 1
  else
     at3[k] := at2[k]
end;
begin { main - yt }
  if k = 0 then
     ayt[k] := y
  else
     begin
       tt(k-1);
       t1(k-1);
       t2(k-1);
       t3(k-1);
       ayt[k] := at3[k-1]/k
     end
end; { yt }
```

```
   at1,at2,att:array [0 .. 34] of real;
procedure yt4(k:integer);

{ prob4: y' = -y + 1,   0 <= t <= 1,  y(0)=0 }

procedure tt(k:integer);
begin
  if k = 0 then
     att[0]:= t
  else
     if k = 1 then
        att[1] := 1
     else
        att[k] := 0
end;
procedure t1(k:integer);
begin
  at1[k] := -ayt[k];
end;
procedure t2(k:integer);
begin
  if k = 0 then
     at2[k] := at1[k] + 1
  else
     at2[k] := at1[k]
end;
begin { main - yt }
  if k = 0 then
     ayt[k] := y
  else
     begin
       tt(k-1);
       t1(k-1);
       t2(k-1);
       ayt[k] := at2[k-1]/k
     end
end; { yt }
```

```
   at1,at2,at3,att:array [0 .. 34] of real;
procedure yt5(k:integer);

{ prob5: y' = -y + t + 1,  0 <= t <= 1,   y(0)=1 }

procedure tt(k:integer);
begin
  if k = 0 then
     att[0]:= t
  else
     if k = 1 then
        att[1] := 1
     else
        att[k] := 0
end;
procedure t1(k:integer);
begin
  at1[k] := -ayt[k];
end;
procedure t2(k:integer);
begin
  at2[k] := at1[k] + att[k]
end;
procedure t3(k:integer);
begin
  if k = 0 then
     at3[k] := at2[k] + 1
  else
     at3[k] := at2[k]
end;
begin { main - yt )
  if k = 0 then
     ayt[k] := y
  else
     begin
       tt(k-1);
       t1(k-1);
       t2(k-1);
       t3(k-1);
       ayt[k] := at3[k-1]/k
     end
end; { yt }
```

```
   at1,at2,at3,att:array [0 .. 34] of real;
procedure yt6(k:integer);

{ prob6: y' = y/t * y/t + y/t,  1 <= t <= 1.2,  y(1)=1  }

procedure tt(k:integer);
begin
  if k = 0 then
     att[0]:= t
  else
     if k = 1 then
        att[1] := 1
     else
        att[k] := 0
end;
procedure t1(k:integer);
begin
  if k = 0 then
     at1[k] := ayt[k]/att[k]
  else
     at1[k] := (ayt[k] - at1[k-1])/att[0]
end;
procedure t2(k:integer);
var j:integer;
begin
  at2[k] := 0;
  for j := 0 to k do
    at2[k] := at2[k] + at1[j]*at1[k-j]
end;
procedure t3(k:integer);
begin
  at3[k] := at2[k] + at1[k]
end;
begin { main - yt }
  if k = 0 then
     ayt[k] := y
  else
     begin
       tt(k-1);
       t1(k-1);
       t2(k-1);
       t3(k-1);
       ayt[k] := at3[k-1]/k
     end
end; { yt }
```

```
   at1,at2,at3,at4,att:array [0 .. 34] of real;
procedure yt7(k:integer);

{ prob?: y' = sint + exp(-t),  0 <= t <=1,  y(0)=0 }

procedure tt(k:integer);
begin
  if k = 0 then
     att[0]:= t
  else
     if k = 1 then
        att[1] := 1
     else
        att[k] := 0
end;
procedure t1(k:integer);
begin
  if k = 0 then
     at1[k] := sin(att[k])
  else
     at1[k] := at4[k-1]/k
end;
procedure t2(k:integer);
begin
  if k = 0 then
     at2[k] := exp(-att[k])
  else
     at2[k] := -at2[k-1]/k
end;
procedure t3(k:integer);
begin
  at3[k] := at1[k] + at2[k]
end;
procedure t4(k:integer);
begin
  if k = 0 then
     at4[k] := cos(att[k])
  else
     at4[k] := -at1[k-1]/k
end;
begin { main - yt }
  if k = 0 then
     ayt[k] := y
  else
     begin
       tt(k-1);
       t1(k-1);
       t2(k-1);
       t3(k-1);
       t4(k-1);
       ayt[k] := at3[k-1]/k
     end
end; { yt }
```

```
   at1,at2,at3,at4,at5,att:array [0 .. 34] of real;
procedure yt8(k:integer);

{ prob8: y' = 2*(y+2*exp(1))/t + t*t*exp(t),   1 <= t <= 2, y(1)=0

procedure tt(k:integer);
begin
  if k = 0 then
     att[0]:= t
  else
     if k = 1 then
        att[1] := 1
     else
        att[k] := 0
end;
procedure t1(k:integer);
begin
  if k = 0 then
     at1[0] := 2*(ayt[0]+2*exp(1))/att[0]
  else
     at1[k] := (2*ayt[k] - at1[k-1])/att[0]
end;
procedure t2(k:integer);
var j,m:integer;
begin
  if k > 2 then
     m := 2
  else
     m := k;
  at2[k] := 0;
  for j := 0 to m do
    at2[k] := at2[k] + att[j]*att[k-j]
end;
procedure t3(k:integer);
begin
   if k = 0 then
      at3[k] := exp(att[k])
   else
      at3[k] := at3[k-1]/k
end;
procedure t4(k:integer);
var j:integer;
begin
  at4[k] := 0;
  for j := 0 to k do
    at4[k] := at4[k] + at2[j]*at3[k-j]
end;
procedure t5(k:integer);
begin
  at5[k] := at4[k] + at1[k]
end;
begin { main - yt }
  if k = 0 then
     ayt[k] := y
```

```
  else
     begin
       tt(k-1);
       t1(k-1);
       t2(k-1);
       t3(k-1);
       t4(k-1);
       t5(k-1);
       ayt[k] := at5[k-1]/k
     end
end; { yt }
```

```
procedure yt(k:integer);

{ prob9: y' = -30*y,   0 <= t <= 1,  y(0)=1/3 }
begin { yt )
  if k = 0 then
     ayt[0] := y
  else
     ayt[k] := -30 * ayt[k-1]/k
end; { yt }



procedure yt10(k:integer);

{ prob10: y' = -300*y,  0 <= t <= 1,  y(0)=1 )
begin { yt }
  if k = 0 then
     ayt[0] := y
  else
     ayt[k] := -300 * ayt[k-1]/k
end; { yt }
```

```
program ivpt(input,output);
const
   formfeed=#12;
type
   vector=array [0 .. 20] of integer;
var
   taylorarray:array [4 .. 35,1 .. 14,0 .. 2] of real;
   rk4array:array [1 .. 14,0 .. 2] of real;
   lowerlimit,value,a,b,alpha:real;
   partition,psize,i,j,k,accuracy,maxdepth,p:integer;
   maxpartition:integer;
   printflag,rkflag,tyflag,tyflag1,rkflag1:boolean;
   answer:char;
   workfile,rktime,tytime:text;
   hour:vector;
   min:vector;
   sec:vector;
   frac:vector;
   time,count,hour1,sec1,min1,frac1:integer;

{$I actual.pas}

procedure timer(var hour,min,sec,frac:integer);
type
  regpack = record
              ax,bx,cx,dx,bp,si,di,ds,es,flags:integer;
            end;
var
  regs: regpack;

begin
  with regs do
    begin
      ax := $2c00;
      msdos(regs);
      hour := hi(cx);
      min := lo(cx);
      sec := hi(dx);
      frac := lo(dx);
    end
end;

procedure rk4(lend,rend,alpha:real; psize:integer);
var
   i:integer;
   h,t,t1,w,k1,k2,k3,k4:real;

{$I f.pas}

begin { rk4 }
  writeln('rk4 ',psize);
  for count := 1 to 10 do
    begin
      timer(hour[count],min[count],sec[count],frac[count]);
```

```
      h := (rend - lend)/ psize;
      t := lend;
      w := alpha;
      for i := 1 to psize do
        begin
          kl := h * f(t,w);
          tl := t + h/2;
          k2 := h * f(t1,w + k1/2);
          k3 := h * f(t1,w + k2/2);
          k4 := h * f(t + h,w + k3);
          w := w + (kl + 2 * (k2 + k3) + k4)/6;
          t := t + h
        end;
      timer(hour[count+10],min[count+10],sec[count+10],
           frac[count+10])
    end;
rk4array[partition,0] := abs(value - w)/abs(value);
rk4array[partition,1] := abs(value - w);
rk4array[partition,2] := w;
if (rk4array[partition,0] <= lowerlimit) or
   (partition = maxpartition) then
   begin
     hour1 := 0;
     minl := 0;
     secl := 0;
     fracl := 0;
     for count := 1 to 10 do
       begin
         hour[count-1] := hour[count+10] - hour[count];
         if min[count+10] < min[count] then
            begin
              hour[count-1] := hour[count-1] - 1;
              min[count+10] := min[count+10] + 60
            end;
         min[count-1] := min[count+10] - min[count];
         if sec[count+10] < sec[count] then
            begin
              min[count-1] := min[count-1] - 1;
              sec[count+10] := sec[count+10] + 60
            end;
         sec[count-1] := sec[count+10] - sec[count];
         if frac[count+10] < frac[count] then
            begin
              sec[count-1] := sec[count-1] - 1;
              frac[count+10] := frac[count+10] + 100
            end;
         frac[count-1] := frac[count+10] - frac[count];
         minl := minl + min[count-1];
         secl := secl + sec[count-1];
         fracl := fracl + frac[count-1]
       end;
     min[0] := minl div 10;
     sec[0] := secl div 10;
     frac[0] := fracl div 10;
```

```
      writeln(rktime,psize,'  ',min[0],':',sec[0],':',
              frac[0])
    end
end; { rk4 }

procedure taylor(lend,rend,alpha:real; psize:integer);
var
   ayt:array [0 .. 35] of real;
   depth,term,j:integer;
   h1,error,app,t,y,h:real;
{$I yt.pas}

begin { taylor }
  writeln('taylor ',psize);
  depth := 4;
  repeat
    for count := 1 to 10 do
      begin
        timer(hour[count],min[count],sec[count],
              frac[count]);
        t := lend;
        h := (rend-lend)/psize;
        y := alpha;
        for j := 1 to psize do
          begin
            app := 0;
            for term := 0 to depth do
              yt(term);
            for term := depth downto 1 do
              app := (app + ayt[term])*h;
            y := app + ayt[0];
            t := t + h
          end;
        timer(hour[count+10],min[count+10],sec[count+10],
              frac[count+10])
      end;
    taylorarray[depth,partition,0] := abs(value-y)/
                                      abs(value);
    taylorarray[depth,partition,1] := abs(value-y);
    taylorarray[depth,partition,2] := y;
    depth := depth + 1
  until (taylorarray[depth-1,partition,0] <= lowerlimit) or
        (depth > maxdepth);
  hour1 := 0;
  minl := 0;
  secl := 0;
  fracl := 0;
  for count := 1 to 10 do
    begin
      hour[count-1] := hour[count+10] - hour[count];
      if min[count+10] < min[count] then
         begin
           hour[count-1] := hour[count-1] - 1;
           min[count+10] := min[count+10] + 60
```

```
        end;
      min[count-1] := min[count+10] - min[count];
      if sec[count+10] < sec[count] then
         begin
           min[count-1] := min[count-1] - 1;
           sec[count+10] := sec[count+10] + 60
         end;
      sec[count-1] := sec[count+10] - sec[count];
      if frac[count+10] < frac[count] then
         begin
           sec[count-1] := sec[count-1] - 1;
           frac[count+10] := frac[count+10] + 100
         end;
      frac[count-1] := frac[count+10] - frac[count];
      minl := minl + min[count-1];
      secl := secl + sec[count-1];
      fracl := fracl + frac[count-1]
    end;
  min[0] := minl div 10;
  sec[0] := secl div 10;
  frac[0] := fracl div 10;
  writeln(tytime,psize,'  ',depth,'  ',min[0],':',sec[0],
          ':',frac[0])
end; { taylor }

begin { main - ivp }
  assign(rktime,'rktime');
  rewrite(rktime);
  assign(tytime,'tytime');
  rewrite(tytime);
  assign(workfile,'ivpout');
  rewrite(workfile);
  write('Runge Kutta (y/n): ');
  readln(answer);
  if answer = 'y' then
     rkflag := true
  else
     rkflag := false;
  write('Taylor series (y/n): ');
  readln(answer);
  if answer = 'y' then
     tyflag := true
  else
     tyflag := false;
  if tyflag then
     begin
       write('enter maximum series length: ');
       readln(maxdepth);
       if maxdepth < 4 then
          maxdepth := 4;
       if maxdepth > 35 then
          maxdepth := 35
     end
  else
```

```
   maxdepth := 3;
write('enter accuracy: ');
readln(accuracy);
lowerlimit := 5*exp(accuracy * ln(10));
writeln(workfile,'lowerlimit ',lowerlimit);
write('enter maximum # of subdivisions for partition',
      ' size: ');
readln(maxpartition);
if maxpartition < 1 then
   maxpartition := 1;
if maxpartition > 14 then
   maxpartition := 14;
write('enter endpoints: ');
readln(a,b);
write('enter initial condition: ');
readln(alpha);
writeln(workfile,a,b,alpha);
writeln(workfile);
value := actual(alpha,a,b);
writeln(workfile,'actual ',value);
for j := 1 to maxpartition do
  begin
    for i := 4 to maxdepth do
      begin
        taylorarray[i,j,0] := 0;
        taylorarray[i,j,1] := 0;
        taylorarray[i,j,2] := 0
      end;
    rk4array[j,0] := 0;
    rk4array[j,1] := 0;
    rk4array[j,2] := 0
  end;
psize := 1;
tyflagl := tyflag;
rkflagl := rkflag;
partition := 1;
while (partition <= maxpartition) and
      (rkflagl or tyflagl) do
  begin
    psize := psize * 2;
    if rkflagl then
       rk4(a,b,alpha,psize);
    if tyflagl then
       taylor(a,b,alpha,psize);
    writeln(workfile,psize:5,'      absolute error',
       ' ':7,'  relative error',' ':7,'  approximation');
    for k := 3 to maxdepth do
      begin
        printflag := true;
        if (k = 3) and (rk4array[partition,0]<>0) then
           writeln(workfile,'RK  4':5,
                   rk4array[partition,1],
                   rk4array[partition,0],
                   rk4array[partition,2]);
```

```
            if (k>3) and (taylorarray[k,partition,0]<>0) then
               writeln(workfile,'TY':2,k:3,
                       taylorarray[k,partition,1],
                       taylorarray[k,partition,0],
                       taylorarray[k,partition,2])
          end;
        writeln(workfile);
        writeln(workfile);
        writeln(workfile);
        if taylorarray[4,partition,0] <= lowerlimit then
           tyflagl := false;
        if rk4array[partition,0] <= lowerlimit then
           rkflagl := false;
        partition := partition + 1
      end;
    close(workfile);
    close(rktime);
    close(tytime);
end. { main - ivp }
```

```
program taylor(input,output);
type
   a1=array[0 .. 40] of real;
const
   formfeed=#12;
var
   taylorarray:real;
   psize:integer;
   lowerlimit,value,a,b,alpha:real;
   error:real;
   outfile,workfile,termfile:text;

{$I actual.pas}

procedure rc(t:real;term:a1);
var
  rm1,rm2:real;
  i:integer;
  rc,order:array[3 .. 40] of real;
begin {rc.pas}
  for i := 3 to 40 do
    begin
      rml := term[i]/term[i-1];
      rm2 := term[i-1]/term[i-2];
      rc[i] := 1/( (i-1) * rml - (i-2) * rm2 );
      order[i] := (i-2) * rc[i] * rm2 - i + 3
    end;
  writeln(outfile);
  writeln(outfile);
  writeln(outfile,'radius of converge and order',
          'estimates from expansion point t=',t);
  writeln(outfile);
  for i := 3 to 40 do
    writeln(outfile,i,' rc= ',abs(rc[i]),'  order= ',
                    order[i])
end; {rc.pas}

procedure taylor(lend,rend,alpha:real; psize:integer);
var
   ayt:a1;
   k,depth,term,j:integer;
   h1,app,t,y,h:real;

{$I yt.pas}

begin { taylor }
  depth := 40;
    t := lend;
    h := (rend-lend)/psize;
    y := alpha;
    for j := 1 to psize do
      begin
        hl := 1;
        app := 0;
```

```
        for term := 0 to depth do
          yt(term);
        for term := depth downto 1 do
          app := (app + ayt[term])*h;
        y := app + ayt[0];
        if j<4 then
        begin
          for k := 0 to 40 do
              writeln(termfile,ayt[k]:40:15);
          writeln(termfile)
        end;
        rc(t,ayt);
        t := t + h
      end;
   taylorarray := y;
   error := abs(value - y)/abs(value)
end; { taylor }

begin { main }
  assign(outfile,'order.rc');
  rewrite(outfile);
  assign(termfile,'termfil');
  rewrite(termfile);
  assign(workfile,'ivpout');
  rewrite(workfile);
  write('enter number of partions: ');
  readln(psize);
  write('enter endpoints: ');
  readln(a,b);
  write('enter initial condition: ');
  readln(alpha);
  writeln(workfile,a,b,alpha);
  writeln(workfile);
  value := actual(alpha,a,b);
  taylor(a,b,alpha,psize);
  writeln(workfile,'appr: ',taylorarray,'  error: ',error);
  close(workfile);
  close(termfile);
  close(outfile)
end. { main }
```

## BIBLIOGRAPHY


1. Burden, Richard L. and Faires, J. Douglas. Numerical Analysis. Third Edition. Boston: Prindle, Weber, and Schmidt, 1985.

2. Chang, Y.F. The Automatic Taylor Series (ATS) Method of Analysis. Unpublished, Claremont McKenna College, Claremont, California, 1984.

3. Chang, Y.F. "The ATOMCC Toolbox," BYTE, Vol. 11, No. 4, April 1986, 215-224.

4. Corliss, George and Chang, Y.F. "Solving Ordinary Differential Equations Using Taylor Series," ACM Transactions on Mathematical Software, Vol. 8, No. 2, June 1982, 114-144.

5. Gear, C. William. Numerical Initial Value Problems in Ordinary Differential Equations. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1971.

6. Moore, Ramon E. Methods and Applications of Interval Analysis. Philadelphia: SIAM Studies in Applied Mathematics, 1979.