IOSR Journal of Mathematics (IOSR-JM)
e-ISSN: 2278-5728, p-ISSN: 2319-765X. Volume 17, Issue 1 Ser. III (Jan. – Feb. 2021), PP 14-18
www.iosrjournals.org

Derivation and Implementation of a Numerical Scheme for Approximating First Order Ordinary Differential Equations Using a Non-Polynomial Interpolating Function

Tersoo Luga[1], OrshiAnongo[2] and Celestine Orterem[3]
[1, 2, 3] *Department of Mathematics/Statistics/Computer Science, College of Science, Federal University of Agriculture, Makurdi, Nigeria*

Abstract: *A non-polynomial interpolating function considered by Ayinde and Ibijola (2015) was modified and utilized to develop an improved numerical scheme for solving first order initial value problems in ordinary differential equations. The scheme was implemented in MATLAB and tested on four problems; all our numerical results show better approximations of the analytical solutions.*
Key words*: Non-polynomial, Single-step and Multistep.*

Date of Submission: 04-02-2021 Date of Acceptance: 19-02-2021

I. Introduction

Single-step methods are veritable tools for solving many physical problems. This is because multistep methods depend on single step methods for starting values, thus the accuracy of a multistep method is determined by the single step method used as a stating method. Numerical analysts are also concerned about developing accurate numerical methods for solving first order ordinary differential equations because nth order ordinary differential equations can be reduced to systems of first order ordinary differential equations which is easier to solve by writing a computer programme. In light of the above, accuracy and stability are the main features considered for developing single-step methods. Besides Taylor series, there are other polynomials and non-polynomial interpolating functionsthat are in recent times used for the derivation of single-step methods for solving initial value problems of the form

$$y'(t) = f(t, y(t)) \tag{1}$$

with the initial condition

$$y(t_0) = \alpha.$$

For example, [1-6] have developed some polynomial and non-polynomial interpolating methods for integrating equation (1). The motivation of this work comes from [5], they used five terms of a non-polynomial interpolating function to develop a method for approximating equation (1). We extend this work to include more terms in order to get better results. The rest of the paper is sectioned as fellows, the improved numerical scheme is developed in Section 2, while the results are provided in Section 3. Finally, discussion and conclusion are carried out in Section 4.

II. Methods

Derived Numerical Scheme by [5]

First, we briefly explain the numerical scheme developed by [5] for approximating equation (1). Consider a non-polynomial interpolating function defined by

$$y(x) = (\alpha_1 + \alpha_2)e^{-2x} + \alpha_3 x^2 + \alpha_4 x + \alpha_5 \tag{2}$$

where $\alpha_{1,}\alpha_2, \alpha_3and\alpha_4$ are undetermined coefficients and α_5 is a constant, which is assumed to be the solution of equation (1).

Finding the first, second and third derivatives of equation (2), solving for the undetermined coefficients, substituting in equation (2) and interpolating at $y(x_{n+1})$ and $y(x_n)$gives

$$\boldsymbol{y_{n+1} = y_n - \frac{1}{8}F_n^2\left(e^{-2h} - 1\right) + \frac{1}{2}\left(F_n^1 + \frac{1}{2}F_n^2\right)h^2 + \left(F_n - \frac{1}{4}F_n^2\right)h} \tag{3}$$

where h is the step size and F_n, F_n^1, F_n^2 are the first, second and third derivatives respectively obtained from$f(x_n, y_n)$ in equation (1).

Derivation of the Improved Numerical Scheme

consider the non-polynomial interpolating function by [5] redefined to include more terms and assumed to be the solution of equation (1)

$$y(x) = (\alpha_1 + \alpha_2 + \alpha_3)e^{-3x} + \alpha_4 x^3 + \alpha_5 x^2 + \alpha_6 x + \alpha_7 \tag{4}$$

where $\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5$ and α_6 are undetermined coefficients and α_7 is a constant.Taking the first, second third and fourth derivatives of (4), yields

$$y'(x_n) = -3(\alpha_1 + \alpha_2 + \alpha_3)e^{-3x_n} + 3\alpha_4 x_n^2 + 2\alpha_5 x_n + \alpha_6 = F_n \quad (5)$$

$$y''(x_n) = 9(\alpha_1 + \alpha_2 + \alpha_3)e^{-3x_n} + 6\alpha_4 x_n + 2\alpha_5 = F_n^1 \quad (6)$$

$$y'''(x_n) = -27(\alpha_1 + \alpha_2 + \alpha_3)e^{-3x_n} + 6\alpha_4 = F_n^2 \quad (7)$$

$$y'^v(x_n) = 81(\alpha_1 + \alpha_2 + \alpha_3)e^{-3x_n} = F_n^3. \quad (8)$$

Solving for $\alpha_1 + \alpha_2 + \alpha_3$ in equation (8), gives

$$\alpha_1 + \alpha_2 + \alpha_3 = \frac{1}{81} F_n^3 e^{3x_n}, \quad (9)$$

substituting equation (9) into (7) and evaluating leads to

$$\alpha_4 = \frac{1}{6} F_n^2 + \frac{1}{18} F_n^3, \quad (10)$$

substituting (9) and (10) into (6) and solving for α_5 yields,

$$\alpha_5 = \frac{1}{2}\left(\left(F_n^1 - \frac{1}{9}F_n^3\right) - \left(F_n^2 + \frac{1}{3}F_n^3\right)x_n\right), \quad (11)$$

substituting (11), (10), and (9) into (5) and simplifying gives,

$$\alpha_6 = F_n + \frac{1}{27}F_n^3 + \left(\frac{1}{2}F_n^2 + \frac{1}{3}F_n^3\right)x_n^2 - \left(F_n^1 - \frac{1}{9}F_n^3\right). \quad (12)$$

Let

$y(x_n) = y_n$ and $y(x_{n+1}) = y_{n+1}$,

since

$F(x_{n+1}) = y(x_{n+1})$ and $y(x_n) = y(x_n)$,

to derive the required numerical method, we substitute $y(x_{n+1})$ and $y(x_n)$ into equation (4) and find the difference which simplifies to

$$y_{n+1} - y_n = (\alpha_1 + \alpha_2 + \alpha_3)e^{-3x_{n+1}} + \alpha_4 x_{n+1}^3 + \alpha_5 x_{n+1}^2 + \alpha_6 x_{n+1} + \alpha_7 - \left((\alpha_1 + \alpha_2 + \alpha_3)e^{-3x_n} + \alpha_4 x_n^3 + \alpha_5 x_n^2 + \alpha_6 x_n + \alpha_7\right). \quad (13)$$

simplifying equation (13) gives

$$y_{n+1} - y_n = (\alpha_1 + \alpha_2 + \alpha_3)(e^{-3x_{n+1}} - e^{-3x_n}) + \alpha_4(x_{n+1}^3 - x_n^3) + \alpha_5(x_{n+1}^2 - x_n^2) + \alpha_6(x_{n+1} - x_n) + \alpha_7 - \alpha_7 \quad (14)$$

Recall that

$$x_n = a + nh, \quad x_{n+1} = a + (n+1)h, \quad n = 0, 1, 2, ... \quad (15)$$

Substituting (9), (10), (11) and (12) into (14) and solving, we have

$$\boldsymbol{y_{n+1} = y_n + \frac{1}{81}F_n^3(e^{-3h} - 1) + \left(\frac{1}{6}F_n^2 + \frac{1}{18}F_n^3\right)h^3 - \left(\frac{1}{18}F_n^3 - \frac{1}{2}F_n^1\right)h^2 + \left(F_n + \frac{1}{27}F_n^3\right)h} \quad (\mathbf{16})$$

where $F_n, F_n^1, F_n^2,$ and F_n^3 are the first, second third and fourth derivatives of $f(x_n, y_n)$ obtained from equation (1) and h is the step size.

Hence,equation (16) is the numerical method derived for solving first order differential equation.

III. Results

In this Section, we implement the derived numerical scheme (16). All the four examples considered in this work are obtained from the work of [5] for the purpose of comparison. All the programmes are implemented in Windows 10 operating system using Matlab 2018a. Our numerical results, the exact solutions and the results of [5] are displayed in Tables for comparison. We shall abbreviate the scheme by [5] as NS1 our numerical scheme as NS2 in the Tables and presentation.

The four examples are:

Example 1

$$y' = y, \quad y(0) = 1, \quad 0 \le x \le 1.$$

Exact solution : $y(x) = e^x$,

Example 2

$y' = x^2 + y, \quad y(0) = 1, \quad 0 \le x \le 1.$

Exact solution: $y(x) = -2 - 2x - x^2 + 3e^x$,

Example 3

$y' = 2xy, \quad y(0) = 1, \quad 0 \le x \le 1.$

Exact solution: $y(x) = e^{x^2}$,

Example 4

$y' = 2xy + 4x, \quad y(0) = 1, \quad 0 \le x \le 1$

Exact solution: $y(x) = 3e^{x^2} - 2$,

Consider Example 1
Given that

$$y' = y_n = f(x_n, y_n) = F_n$$

Taking first, second, third and fourth derivatives, we have

$$y'' = y_n = F_n^1$$
$$y''' = y_n = F_n^2$$
$$y^{iv} = y_n = F_n^3$$

F_n, F_n^1, $F_n^2$and F_n^3 are substituted into equation (16) and implemented in Matlab, the numerical results, alongside the result of [5] and the exact solution are displayed in Table 1.

Table1: Results of the Scheme by [5] (NS1) and the Improved Numerical Scheme (NS2) for Example 1

n	x_n	$NS1$	$y(x_n)$	$\lvert y(x_n) - NS1 \rvert$	$NS2$	$\lvert y(x_n) - NS2 \rvert$
0	0	1.000000000000000	1.000000000000000	0	1.000000000000000	0
1	0.1	1.105158655865252	1.105170918075648	1.22622103×10^{-5}	1.105170595317058	$3.227585896325991\times 10^{-7}$
2	0.2	1.221375654633891	1.221402758160170	71035262×10^{-5}	1.221402044753461	$7.134067090408536\times 10^{-7}$
3	0.3	1.349813876781733	1.349858807576003	4.49307942×10^{-5}	1.349857624921654	$1.182654348896861\times 10^{-6}$
4	0.4	1.491758489732366	1.491824697641270	6.62079089×10^{-5}	1.491822954927935	$1.742713335195489\times 10^{-6}$
5	0.5	1.648629807388200	1.648721270700128	9.14633119×10^{-5}	1.648718863205359	$2.407494769318674\times 10^{-6}$
6	0.6	1.821997501952533	1.822118800390509	1.21298437×10^{-4}	1.822115607559130	$3.192831379061900\times 10^{-6}$
7	0.7	2.013596310247709	2.013752707470477	1.56397222×10^{-4}	2.013748590742627	$4.116727850167479\times 10^{-6}$
8	0.8	2.225343391688589	2.225540928492467	1.97536803×10^{-4}	2.225535728849916	$5.199642551634298\times 10^{-6}$
9	0.9	2.459357511597183	2.459603111156949	2.45599559×10^{-4}	2.459596646352444	$6.464804505057486\times 10^{-6}$
10	1.0	2.717980241808854	2.718281828459045	3.01586650×10^{-4}	2.718273889889171	$7.938569874355039\times 10^{-6}$

Consider Example 2
Given that

$$y' = x_n^2 + y_n = f(x_n, y_n) = F_n$$

Taking first, second, third and fourth derivatives, we have

$$y'' = 2x_n + x_n^2 + y_n = F_n^1$$
$$y''' = 2 + 2x_n + x_n^2 + y_n = F_n^2$$
$$y^{iv} = 2 + 2x_n + x_n^2 + y_n = F_n^3$$

F_n, F_n^1, $F_n^2$and F_n^3 are substituted into equation (16) and implemented in Matlab, the numerical results, alongside the result of [5] and the exact solution are displayed in Table 2.

Table 2: Results of the Scheme by [5] (NS1) and the Improved Scheme (NS2) forExample 2

n	x_n	$NS1$	$y(x_n)$	$\lvert y(x_n) - NS1 \rvert$	$NS2$	$\lvert y(x_n) - NS2 \rvert$
0	0	1.000000000000000	1.000000000000000	0	1.000000000000000	0
1	0.1	1.105475967595757	1.105512754226943	$3.678663118\times 10^{-5}$	1.105511785951175	$9.682757684537080\times 10^{-7}$
2	0.2	1.224126963901673	1.224208274480510	$8.131057883\times 10^{-5}$	1.224206134260383	$2.140220127122561\times 10^{-6}$
3	0.3	1.359441630345201	1.359576422728009	$1.347923828\times 10^{-4}$	1.359572874764964	$3.547963045358316\times 10^{-6}$
4	0.4	1.515275469197098	1.515474092923811	$1.986237267\times 10^{-4}$	1.515468864783806	$5.228140004920334\times 10^{-6}$
5	0.5	1.695889422164601	1.696163812100385	$2.743899357\times 10^{-4}$	1.696156589616078	$7.222484306845800\times 10^{-6}$
6	0.6	1.905992505857600	1.906356401171526	$3.638953139\times 10^{-4}$	1.906346822677391	$9.578494134965254\times 10^{-6}$
7	0.7	2.150788930743127	2.151258122411430	$4.691916683\times 10^{-4}$	2.151245772227882	$1.235018354739381\times 10^{-5}$
8	0.8	2.436030175065768	2.436622785477403	$5.926104116\times 10^{-4}$	2.436607186549750	$1.559892765312654\times 10^{-5}$
9	0.9	2.768072534791549	2.768809333470848	$7.367986792\times 10^{-4}$	2.768789939057337	$1.939441351161975\times 10^{-5}$
10	1.0	3.153940725426563	3.154845485377136	$9.047599505\times 10^{-4}$	3.154821669667516	$2.381570961951240\times 10^{-5}$

Consider Example 3

Given that

$$y' = 2x_n y_n = f(x_n, y_n) = F_n$$

Taking first, second, third and fourth derivatives, we have

$$y'' = 4x_n^2 y_n + 2y_n = F_n^1$$
$$y''' = 8x_n^3 y_n + 12x_n y_n = F_n^2$$
$$y'^v = 16x_n^4 y_n + 48x_n^2 y_n + 12y_n = F_n^3$$

F_n, F_n^1, $F_n^2$and F_n^3 are substituted into equation (16) and implemented in Matlab, the numerical results, alongside the result of [5] and the exact solution are displayed in Table 3.

Table 3: Results of the Scheme by [5] (NS1) and the Improved Scheme (NS2) forExample 3

n	x_n	$NS1$	$y(x_n)$	$\lvert y(x_n) - NS1 \rvert$	$NS2$	$\lvert y(x_n) - NS2 \rvert$
0	0	1.000000000000000	1.000000000000000	0	1.000000000000000	0
1	0.1	1.010000000000000	1.010050167084168	$5.016708416\times 10^{-5}$	1.010047143804699	$3.023279468994389\times 10^{-6}$
2	0.2	1.040695572848077	1.040810774192388	$1.152013443\times 10^{-4}$	1.040803452155623	$7.322036765122775\times 10^{-6}$
3	0.3	1.093969745041759	1.094174283705210	$2.045386634\times 10^{-4}$	1.094160709607158	$1.357409805291532\times 10^{-5}$
4	0.4	1.173179095693766	1.173510870991810	$3.317752980\times 10^{-4}$	1.173487996663724	$2.287432808589784\times 10^{-5}$
5	0.5	1.283508119239332	1.284025416687741	$5.172974484\times 10^{-4}$	1.283988344377544	$3.707231019745016\times 10^{-5}$
6	0.6	1.432537005590369	1.433329414560340	$7.924089699\times 10^{-4}$	1.433269983224613	$5.943133572738901\times 10^{-5}$
7	0.7	1.631110242151853	1.632316219955379	1.20597780×10^{-3}	1.632220347419935	$9.587253544407481\times 10^{-5}$
8	0.8	1.894645561192191	1.896480879304951	$1.835318112\times 10^{-3}$	1.896323597768354	$1.572815365971092\times 10^{-4}$
9	0.9	2.245103741234345	2.247907986676471	2.80424544×10^{-3}	2.247644236586702	$2.637500897688660\times 10^{-4}$
10	1.0	2.713968432393255	2.718281828459045	$4.313396065\times 10^{-3}$	2.717829470398960	$4.523580600848121\times 10^{-4}$

Consider Example 4

Given that

$$y' = 2x_n y_n + 4x_n = f(x_n, y_n) = F_n$$

Taking first, second, third and fourth derivatives, we have

$$y'' = 4x_n^2 y_n + 2y_n + 4 = F_n^1$$
$$y''' = 8x_n^3 y_n + 12x_n y_n = F_n^2$$
$$y'^v = 16x_n^4 y_n + 48x_n^2 y_n + 12y_n = F_n^3$$

F_n, F_n^1, $F_n^2$and F_n^3 are substituted into equation (16) and implemented in Matlab, the numerical results, alongside the result of [5] and the exact solution are displayed in Table 4.

Table 4: Results of the Scheme by [5] (NS1) and the Improved Scheme (NS2) forExample 4

n	x_n	$NS1$	$y(x_n)$	$\lvert y(x_n) - NS1 \rvert$	$NS2$	$\lvert y(x_n) - NS2 \rvert$
0	0	1.000000000000000	1.000000000000000	0	1.000000000000000	0
1	0.1	1.030000000000000	1.030150501252504	$1.505012525\times 10^{-4}$	1.030094287609398	$5.621364310592902\times 10^{-5}$
2	0.2	1.121988745374320	1.122432322577165	$4.435772028\times 10^{-4}$	1.122318070997852	$1.142515793124410\times 10^{-4}$
3	0.3	1.281595111845066	1.282522851115632	$9.277392705\times 10^{-4}$	1.282342796332057	$1.800547835746347\times 10^{-4}$
4	0.4	1.518853721264559	1.520532612975431	$1.678891710\times 10^{-3}$	1.520272381473036	$2.602315023949586\times 10^{-4}$
5	0.5	1.849282462194907	1.852076250063224	$2.793787868\times 10^{-3}$	1.851713216603692	$3.630334595325913\times 10^{-4}$
6	0.6	2.295614212785162	2.299988243681021	$4.374030895\times 10^{-3}$	2.299488554395282	$4.996892857387358\times 10^{-4}$
7	0.7	2.890456371237462	2.896948659866137	$6.492288628\times 10^{-3}$	2.896262259904568	$6.863999615687177\times 10^{-4}$
8	0.8	3.680317882130108	3.689442637914854	$9.124755784\times 10^{-3}$	3.688495145927897	$9.474919869569831\times 10^{-4}$
9	0.9	4.731703935455796	4.743723960029413	$1.202002457\times 10^{-2}$	4.742403390685642	$1.320569343770117\times 10^{-3}$
10	1.0	6.140396568913641	6.154845485377134	$1.444891646\times 10^{-2}$	6.152980357913922	$1.865127463211991\times 10^{-3}$

IV. Discussion

This work is an improvement over the numerical scheme of [5]. Five terms of a non-polynomial interpolating function defined by equation (2) were used to derive a numerical scheme by [5] for the solution of first order initial value problems (IVPs) in ordinary differential equations (ODEs). Although the absolute errors between the analytical and numerical solution show good approximations, we thought if more terms of the function were considered, a better result would have been obtained. To this end, seven terms of the non-polynomial interpolating function defined by equation (3) were differentiated the number of required times and manipulated to derive the numerical scheme (16).For the purpose of comparison, all the four test problems were obtained from [5]. MATLAB programmes were written using equations (3) and (16) and were applied to approximate all the four test problems. The numerical results of [5] denoted by $NS1$, our numerical results denoted by $NS2$, the analytical results denoted by $y(x)$ and the various absolute errors are provided in Tables 1-4. We observed that for all the four test problems, our numerical scheme $NS2$ gave better approximations. Also, the results of [5] which we implemented in MATLAB are better than the result in their published work. The accuracy of these numerical schemes may be as a result of checking the round-off errors.

V. Conclusion

A numerical scheme which is an improvement of the scheme by [5] was derived using a non-polynomial interpolating function. This scheme was applied to solve some problems on first order initial value problems and compared with the scheme by [5]. We observed that our scheme provided better approximations, however the computations were also more. The implementation of these schemes is similar to that of the Taylor series method which the accuracy increases by considering more terms of the polynomial, but the task of differentiating the function repeatedly is not trivial. A numerical scheme can be constructed by Taylor expanding our scheme and manipulating it in order to get a scheme akin to the Runge-Kutta methods.

References

[1]. Kama, P. and Ibijola, E. A. (2000). On a New-Step Method for Numerical Integration of Ordinary Differential Equations. *International Journal of Computer Mathematics.* 78(3-4)

[2]. Ibijola, E. A. (1997). *New Schemes for Numerical Integration of Special Initial Value Problems in Ordinary Differential Equations*. Ph. D. Thesis, University of Benin, Nigeria.

[3]. Ogunrinde, R. B., Fadugba, S. E. and Okunlola, J. T. (2012). On Some Numerical Methods for Solving Initial Value Problems in ODEs. *IORS Journal of Mathematics*(IORSJM). 1(3): 25-31.

[4]. Ogunderi, R. B. and Olaosebikan, T. E. (2016). Development of a Numerical Scheme. *American Journal of Computational Mathematics*, 6:49-54.

[5]. Ayinde, S. O. and Ibijola, E. A. (2015). A New Numerical Method for Solving First Order Differential Equations. *American Journal of Applied Mathematics and Statistics*, 3(4):156-160.

[6]. Kamruzzaman, M. and Hassan, M. M. (2018). A New Numerical Approach for Solving Initial Value Problems of Ordinary Differential Equations. *Annals of Pure and Applied Mathematics*, 17(2):157-162.

Tersoo Luga, et. al. "Derivation and Implementation of a Numerical Scheme for Approximating First Order Ordinary Differential Equations Using a Non-Polynomial Interpolating Function." *IOSR Journal of Mathematics (IOSR-JM)*, 17(1), (2021): pp. 14-18.