Journal of NAMP 70, (2025) 167-178

The Nigerian Association of Mathematical Physics

Journal homepage: https://nampjournals.org.ng

# A WEB-BASED TOOL FOR GENERATING BLOCK LINEAR MULTISTEP METHODS

* Christie Y. Ishola, Oyewole A. Oyelami, Collins C. Elebor
*Department of Mathematics, National Open University of Nigeria, University Village, Plot 91 Cadastral Zone, Nnamdi Azikiwe Express Way, Jabi, Abuja FCT, Nigeria*

ARTICLE INFO

*Article history*:
Received 30/6/2025
Revised 12/7/2025
Accepted 15/7/2025
Available online 17/7/2025

*Keywords:*
Linear Multistep Methods, Ordinary Differential Equations, Interpolation, Collocation, Web-Based Tool, Numerical Methods

***ABSTRACT***
*Linear Multistep Methods are widely used to obtain numerical solutions of ordinary differential equations. However, generating these methods manually can be computationally challenging, time-consuming, and prone to errors. Existing mathematical software's lacks the flexibility to efficiently automate the derivation of Linear Multistep Methods. This paper introduces a web-based tool designed to address these limitations by automating the generation of LMMs of first and second derivative methods using interpolation and collocation techniques. The tool enhances computational efficiency, saves time, and simplifies the process of solving ODEs. The backend of the tool is developed in Python using the FASTAPI framework. The frontend, built with React, provides a dynamic and responsive interface for user interaction. This architecture's integration of Python and React highlights the advantages of combining a powerful computational backend with an intuitive frontend. The effectiveness of the tool is demonstrated by successfully generating well-known Linear Multistep Methods.*

## 1. INTRODUCTION

Numerical methods are algorithms used for approximating solutions to mathematical problems by performing finite sequences of arithmetic operations. In the context of solving ordinary differential equations (ODEs), differential equations (ODEs), numerical methods provide discrete approximations of the solution by iteratively computing values at specific points. These methods can be broadly categorized into single-step and multi-step methods [1]. Single-step methods, such as Euler's method and Runge-Kutta methods, compute the solution at the next step using only the information from the current step. In contrast, Linear Multistep Methods (LMMs) use information from multiple previous steps to determine the next value. LMMs are derived by expressing the numerical solution as a linear combination of past solution values and, in some cases, their derivatives [2].

*Corresponding author: CHRISTIE Y. ISHOLA
*E-mail address:* oyelamioyewole@gmail.com

https://doi.org/10.60787/jnamp.vol70no.566
1118-4388© 2025 JNAMP. All rights reserved

This allows LMMs to achieve higher accuracy without requiring additional function evaluations at each step, making them computationally efficient.

LMMs can be categorized into explicit and implicit methods. Explicit methods compute the next solution value directly from known previous values, while implicit methods involve solving an equation to obtain the next value, often requiring iterative techniques such as Newton's method [3]. Common examples of LMMs include Adams-Bashforth (explicit) and Adams-Moulton (implicit) methods. Another important class of LMMs is the Backward Differentiation Formula (BDF), which is particularly useful for stiff ODEs [4].

An LMM typically takes the following form:

$$\sum_{j=1}^{k} \alpha_i \, y_{n+i} = \sum_{i=0}^{k} \beta_i \, f(t_{n+i},, y_{n+i}) \qquad (1)$$

where $\alpha_i$ and $\beta_i$ are coefficients, $h$ is the step size, $y_{n+i}$ represents solution points, and $f$ is the function defined by the ODE.

Hybrid LMMs extend traditional LMMs by incorporating off-step points (i.e., points within a single step interval) into their formulation [5]. This inclusion allows for increased accuracy and better stability properties [6]. The general form of a hybrid LMM is:

$$\sum_{j=1}^{k} \alpha_i \, y_{n+i} + \sum_{j=1}^{m} \gamma_j \, y_{n+\theta_j} = h \sum_{i=0}^{k} \beta_i \, f(t_{n+i},, y_{n+i}) + h \sum_{j=0}^{m} \gamma f\left(t_{n+\theta_j},, y_{n+\theta_j}\right) \qquad (2)$$

Where $y_{n+\theta_j}$ and $f\left(t_{n+\theta_j}, y_{n+\theta_j}\right)$ are the values and derivatives evaluated at fractional step points $\theta_j$ within the interval [1]. Hybrid LMMs provide a flexible approach that combines the efficiency of multistep methods with the enhanced accuracy often associated with single-step methods [ 2].

Block method generates independent solution at selected grid points without overlapping. It is less expensive in terms of number of function evaluation compared to predictor corrector method, moreover it possesses the properties of Runge Kutta method for being self-starting and does not require starting values. Some of the authors that proposed block method are: [7,8,9,10]

Creating hybrid LMMs in symbolic or numerical computing tools like Maple, Mathematica, and MATLAB poses several challenges:

1. Complexity in Coefficient Derivation:
   Hybrid LMMs require the derivation of additional coefficients for off-step points, increasing the algebraic complexity compared to standard LMMs.
2. Error Constant and Order derivation:
   Verifying the order and error constants for hybrid LMMs involves symbolic computations that can become cumbersome, especially for higher-order methods.
3. Customizability:
   Most symbolic tools are not tailored for hybrid LMMs, requiring manual coding or modifications to generic LMM templates, which is error-prone and time-consuming.

**Objectives**

i. Develop a tool that automatically derives Linear Multistep Methods using interpolation and collocation, thereby reducing manual computation errors and saving time.

ii. Integrate Modern Web Technologies: Combine a Python backend (built on FASTAPI) with a dynamic React frontend to create an interactive, responsive user interface for inputting parameters and viewing results.

iii. Improve Usability and Accessibility: Design the tool so that users can easily generate, download, or render the outputs in LaTeX format, making advanced numerical methods more accessible to a wide range of users.

iv. Validate the Tool's Effectiveness: Demonstrate the tool's reliability by successfully generating established Linear Multistep Methods, proving its practical value for researchers and practitioners in numerical analysis.

The significance of the proposed web-based tool lies in its ability to address critical challenges associated with the generation of Linear Multistep Methods (LMMs) for solving ordinary differential equations (ODEs). These challenges include the computational complexity, time consumption, and potential for manual errors inherent in traditional methods. The tool's contributions can be summarized as follows:

1. Automation of Numerical Method Derivation 2. Enhanced Computational Efficiency. 3. User Accessibility and Scalability 4. Reproducibility and Standardization 5. Time-Saving and Practical Utility 6. Advancement in Numerical Methods

This paper presents the development of a web-based tool for generating Linear Multistep Methods (LMMs) of first and second derivative methods. The tool features a robust Python backend built using FASTAPI and an interactive frontend implemented with the React framework. It employs interpolation and collocation techniques to derive LMMs, streamlining the process of solving ordinary differential equations (ODEs). This integration of modern web technologies and advanced numerical methods offers researchers and practitioners an efficient and user-friendly platform for automated LMM generation.

## 2. Theoretical Analysis

### 2.1 Mathematical Techniques

The mathematical technique for generating numerical methods functions utilizes interpolation and collocation of power series approximate solutions, combined with a matrix inversion approach, to derive a continuous hybrid linear multistep method. This method is subsequently evaluated at selected grid points to formulate a continuous block method within the framework of Linear Multistep Methods (LMMs). The incorporation of interpolation ensures that the approximate solution smoothly aligns with known values, while collocation guarantees that the derived method satisfies the given differential equation at specific points. The matrix inversion technique plays a crucial role in solving the system of equations that arise from these conditions, facilitating the development of a stable and accurate numerical scheme. Below is a detailed explanation of the key techniques used. A lot of contributions have been made by many prominent researchers. To mention a few [11,12, 13, 14, 15, 16], and so on. These researchers utilized collocation and interpolation technique applied on power series approximation to obtained linear multistep methods

#### 2.1.1 Interpolation

Interpolation involves approximating a continuous function by constructing a polynomial (or another type of function) that passes through a given set of discrete data points. This technique is widely used to estimate intermediate values, and its accuracy depends on the choice of interpolation nodes and the degree of the polynomial. Methods such as Lagrange and barycentric interpolation are popular because they offer straightforward implementations and can achieve high accuracy when the data is well-behaved [17].

- Inputs are used to construct a system of equations based on function values.

- Polynomial Construction: For each interpolation point $x_i$, the power series $x_i^j$ is evaluated for increasing powers j. This forms rows in the matrix where coefficients correspond to polynomial terms.
  Interpolation ensures that the polynomial satisfies the function's values at specified points [10], helping construct the continuous form of the method.

**2.1.2 Collocation:**
Collocation is a technique used to solve differential equations by enforcing that the approximate solution satisfies the equation exactly at a finite number of selected points—called collocation points. By converting a continuous problem into a discrete one, collocation methods reduce the differential equation to a system of algebraic equations. This approach is especially useful for boundary value problems and is often combined with interpolation methods to create spectral and finite element methods, ensuring both accuracy and stability [18].

When generating Linear Multistep Methods (LMMs), interpolation is used to construct an approximation of the solution and its derivatives over an interval, while collocation is employed to ensure that the differential equation holds at selected fractional points within that interval. This combination not only simplifies the derivation of LMMs but also enhances their computational efficiency and accuracy [1]. To impose conditions on derivatives of the approximating polynomial to align it with the differential equation.

- First Derivative Collocation: Points in first derivatives points enforce conditions on the first derivative using symbolic differentiation.
- Second Derivative Collocation: Points in second derivatives points enforce conditions on the second derivative.
  Collocation ensures that the polynomial's derivatives adhere to the system's behaviour, ensuring accuracy and stability.

**2.2 Implementation Workflow**

1 Input Handling: The function accepts interpolation, first derivative, and second derivative points as inputs. These points are used to construct a unified list of all evaluation points.
2 Matrix Construction: A symbolic matrix is constructed based on the power series representation of the function and its derivatives. Special cases (e.g., zero points) are handled explicitly to avoid singularities.
3 Symbolic Inversion: The matrix is inverted symbolically to compute the coefficients of the polynomial approximation.
4 Continuous Method Construction: The coefficients are combined with the power series to construct the continuous form of the method.
5 Discrete Methods Generation: The continuous method is evaluated at specific points to generate discrete methods for interpolation, first derivatives, and second derivatives.

**2.2.1 Implementation Details**
The implementation of the architecture describe is done using Python, leveraging its powerful libraries for symbolic computation, matrix operations, and efficient handling of data structures. Below are the details of the programming languages, libraries, and frameworks used [19].

**1) Python:**

- Python programming language is chosen for its versatility, readability, and extensive ecosystem of libraries.
- It provides an excellent platform for both symbolic and numerical computations, making it ideal for this architecture.

2) **Libraries and Frameworks**
   - SymPy: SymPy is a Python library for symbolic mathematics. It allows exact computation of derivatives, expansions, and inverses. It uses for Constructing and manipulating symbolic matrices. Performing symbolic differentiation to compute derivatives of the power series. Expanding and simplifying mathematical expressions [20]. Key functions used:
   - sp.diff(): For symbolic differentiation.
   - sp.expand(): For expanding symbolic expressions.
   - matrix.inv(): For computing the inverse of symbolic matrices.
   - Python Standard Library: Modules like copy are used for efficient data handling and manipulation. For example, copy.deepcopy() is utilized to create independent copies of lists.

## 2.3 Development Methodology

### 2.3.1 Workflow Steps

1) **Input Initialization**

**Interpolation Points ($ip$):**
These are the discrete points at which the function values are known or will be approximated. The interpolation points form the basis for constructing a polynomial (or other basis functions) that approximates the continuous function.

**Collocation Points for First Derivatives ($dp$):**
These points are selected such that the first derivative of the approximating function is forced to match the derivative of the target function at these locations. This ensures that the approximation not only fits the function values but also aligns with its slope, enhancing accuracy.

**Collocation Points for Second Derivatives ($gp$):**
Similarly, these points are chosen to enforce that the second derivative of the approximating function agrees with that of the target function. This additional condition is particularly important when higher-order accuracy is desired, or when the underlying differential equation involves second derivatives.

**Define variables**
$ip, dp$, and $gp$: These variables represent the interpolation points, first derivative collocation points, and second derivative collocation points, respectively. They are the primary inputs to the system, and their selection can significantly affect the accuracy and stability of the resulting numerical method. A check is performed to determine if `$gp$` (the collection of second derivative collocation points) is non-empty. If it contains points, a flag is set to indicate that second derivative conditions must be included in the method construction. This flag helps in dynamically adjusting the subsequent matrix construction and coefficient calculation steps.

2) **Matrix Construction**

**Initialize Matrix:** An empty matrix of size $n \times n$ is created, where $n$ is the total number of conditions imposed. This total is the sum of the number of interpolation points ($ip$), first derivative collocation points ($dp$), and (if applicable) second derivative collocation points ($gp$).

**Populate the Matrix:** Loop through the matrix rows: Each row corresponds to one of the imposed conditions (function value, first derivative, or second derivative).

**Interpolation Rows:** For interpolation, the matrix rows are populated with the powers of the interpolation points. For example, for a polynomial approximation, each row consists of

$1, x_{i+j}, x_{i+j}^2, \ldots, x_{i+j}^{n-1}$ where $x_{i+j}$ is an interpolation point. $i$ is the current step and $j$ is the future step.

**First Derivative Rows:** For collocation of the first derivative, the rows are filled with the derivatives of the basis functions evaluated at the first derivative collocation points.

**Second Derivative Rows:** Similarly, if second derivative conditions are required (i.e., the $gp$ list is non-empty), rows are populated with the second derivatives of the basic functions evaluated at those points.

Special cases (such as zero or near-zero values) must be handled explicitly to avoid numerical instabilities or inaccuracies in the matrix. For instance, if a collocation point is at a singularity or a point where the derivative is not well-behaved, the algorithm may need to adjust the row entries or apply a specific numerical remedy.

3) **Matrix Inversion**

Once the matrix is constructed, the next step is to compute its inverse. This inversion is critical because it provides the coefficients that link the discrete conditions (values and derivatives) to the continuous approximating function.

**Singular Matrix Check:**

Before attempting the inversion, the determinant of the matrix is checked. A zero determinant indicates that the matrix is singular (non-invertible), which means that the chosen interpolation/collocation points or basis functions do not lead to a unique solution. If the matrix is found to be non-invertible, the method should return an error message, prompting the user to re-examine the input points or the formulation.

4) **Generate Grids**

The process involves creating symbolic representations for the function $y_{i+j}$, its first derivative $f_{i+j}$ and its second derivative $g_{i+j}$. These symbols correspond to the values at the interpolation and collocation points and are crucial for constructing a general form of the solution.

**Combine into a Single Vector:** Once the symbolic representations are defined, they are combined into a single vector. This vector represents all the conditions that the continuous method must satisfy. By consolidating these symbols, one obtains a unified framework that will be used to compute the coefficients in the next step.

5) **Continuous Method Construction**

The inverse matrix computed earlier is multiplied by the grid vector. This operation yields the coefficients that will weigh the contribution of each basis function (or term in the power series) in the continuous approximation.

6) **Power Series Construction:**

**Form the Series:** Using the total number of points $n$, a power series is constructed. Each term in the series corresponds to a basis function scaled by the computed coefficient.

**Combine with Coefficients:** The final continuous method is obtained by multiplying the constructed power series by the vector of coefficients derived from the matrix inversion. This operation integrates the discrete conditions into a single continuous representation.

7) **Generation of Methods**

Interpolation Methods are generated by evaluating the continuous method at points that are in the collocation points but not present in the interpolation points.

First derivative Collocation methods are generated by differentiating the continuous method with respect to the independent variable. Then evaluating the differentiate continuous method at points that are in the second derivative collocation points and interpolation points but not present in the first derivative collocation points.

Second derivative Collocation methods are generated by differentiating the continuous method with respect to the independent variable twice. Then evaluating the differentiate continuous method at points that are in the first derivative collocation points and interpolation points but not present in the second derivative collocation points.

8) **Output Results**
Create and return results containing a message (Success or error).

9) **Flow of Execution**
Input Initialization → Point Evaluation, Matrix Construction → Matrix Inversion, Grid Generation → Continuous Method, Generate Methods (Interpolation, Derivative)

### 2.3.2 Technology Stack

**1) Frontend: React.js.**

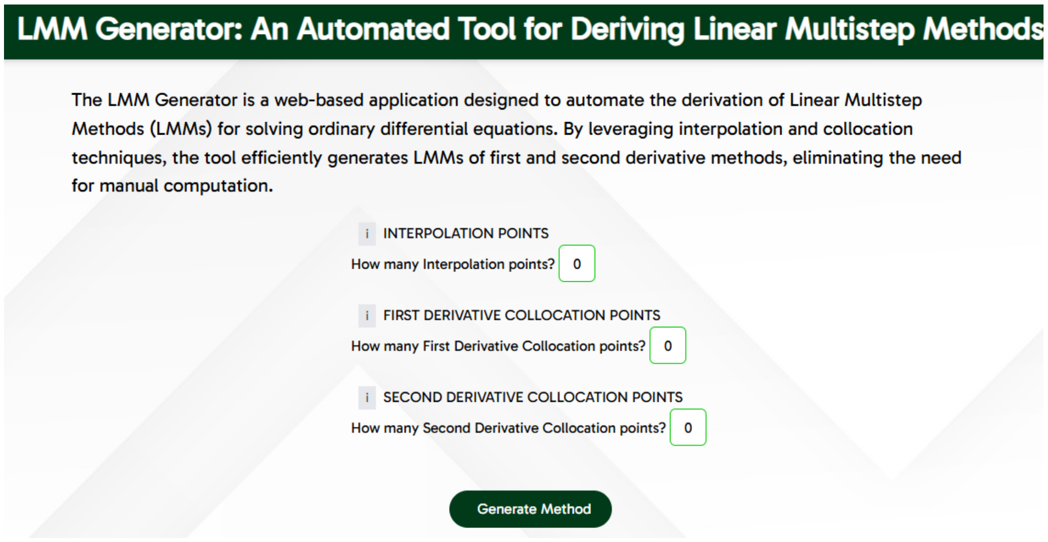
The choice of React.js for the frontend of the described system is a strategic one, aligning with the project's goals of providing a dynamic, responsive, and user-friendly interface. Here’s a list of why React.js is suitable for this tool:
Component-Based Architecture 2. Dynamic and Responsive UI 3. Integration with Backend APIs 4. Flexibility for LaTeX Rendering 5. Scalability and Performance 6. Developer Productivity 7. Future-Proofing

**2) Backend: Python**

Python is an excellent choice for implementing robust computational logic, especially for a tool designed to automate mathematical computations like deriving Linear Multistep Methods (LMMs). Here's how Python supports this:
1. Comprehensive Libraries for Computation (NumPy, SymPy, SciPy). 2. Flexibility and High-Level Abstractions 3. Robustness and Error Handling 4. High Performance with Optimization Techniques 5. Interfacing with Other Systems 6. Scalability for Mathematical Workflows
7. Integration with APIs 8. Support for Advanced Mathematical Techniques 9. Community and Ecosystem

## RESULTS AND DISCUSSION

### 3.1 Front-End Interface:



LMM Generator: An Automated Tool for Deriving Linear Multistep Methods

The LMM Generator is a web-based application designed to automate the derivation of Linear Multistep Methods (LMMs) for solving ordinary differential equations. By leveraging interpolation and collocation techniques, the tool efficiently generates LMMs of first and second derivative methods, eliminating the need for manual computation.

INTERPOLATION POINTS
How many Interpolation points? 0

FIRST DERIVATIVE COLLOCATION POINTS
How many First Derivative Collocation points? 0

SECOND DERIVATIVE COLLOCATION POINTS
How many Second Derivative Collocation points? 0

### 3.2 Test LMMs

To validate the software ability to generate numerical method. Note that '$f$' stands for the first derivative collocation points and 'g' stands for the second derivative collocation points in the methods generated. The software will used to generate 8 BLOCK LMMs found in literature.

1. Implicit Euler method 2. Trapezoidal rule 3. Simpson's ODE Solver [21] 4. Two-step Block Adams-Moulton Method [22] 5. Hybrid Linear Multistep method [23] 6. Hybrid Linear Multistep method [24] 7. Falkner Hybrid Block Methods [25] 8. Hybrid Linear Multistep method [26].

## 3.3 Results

1. In generating implicit Euler implicit method, the interpolation points are "0" while the collocation point is "1" for first derivative collocation. It was generated in 0.382 seconds.

$$\{method\colon y_{n+0} = -f_{(n+0)h} + y_{n+1}\} \tag{3}$$
$$\{method\colon f_{n+1} = f_{n+0}\} \tag{4}$$

   Eq. (3) is Euler's implicit method generated by the LMM generator.
2. In generating Trapezoidal rule, the interpolation point is "0" while the collocation points are "0,1" for first derivative collocation. It was generated in 0.539 seconds.

$$\left\{method\colon y_{n+1} = \frac{f_{(n+0)}h}{2} + \frac{f_{n+1}h}{2} + y_{n+0}\right\} \tag{5}$$

   Eq. (4) is trapezoid's method generated by the LMM generator.
3. In generating Simpson's ODE Solver, the interpolation points are "0,1" while the collocation points are "0,1,2" for first derivative collocation only. It was generated in 0.654 seconds.

$$\left\{method\colon y_{n+1} = \frac{5f_{n+0}h}{12} + \frac{2f_{(n+1)}h}{3} - \frac{f_{n+2}h}{12} + y_{n+0}\right\} \tag{6}$$
$$\left\{method\colon y_{n+2} = \frac{f_{n+0}h}{3} + \frac{4f_{(n+1)}h}{3} - \frac{f_{n+2}h}{3} + y_{n+0}\right\} \tag{7}$$

   Eq. (5) and Eq. (7) is Simpson's ODE Solver generated by the LMM generator. This result agrees with Simpson's ODE Solver Method found in literature.
4. In generating Two-step Block Adams-Moulton Method, the interpolation points are "0,1" while the collocation points are "0,1,2" for first derivative collocation only. It was generated in 0.419 seconds.

$$\left\{method\colon y_{n+0} = -\frac{5f_{n+0}h}{12} - \frac{2f_{(n+1)}h}{3} + \frac{f_{n+2}h}{12} + y_{n+1}\right\} \tag{8}$$
$$\left\{method\colon y_{n+2} = -\frac{f_{n+0}h}{12} + \frac{2f_{(n+1)}h}{3} + \frac{5f_{n+2}h}{12} + y_{n+1}\right\} \tag{9}$$

   Eq. (8) and Eq. (9) Two-step Block Adams-Moulton Method generated by the LMM generator. This result agrees with Two-step Block Adams-Moulton Method found in literature.

5. In generating hybrid Linear Multistep method [23], the interpolation points is "0,1" while the collocation points are for first derivative collocation is none and second derivative collocation is "$0, \frac{3}{2}, 1, 2$". It was generated in 2.612 seconds.

$$\left\{method: y_{n+\frac{3}{2}} = \frac{g_{(n+0)}h^2}{24} + \frac{13g_{n+1}h^2}{32} + \frac{g_{n+2}h^2}{32} - \frac{5g_{n+\frac{3}{2}}h^2}{48} - y_{n+0} + \frac{3y_{n+1}}{2}\right\} \quad (10)$$

$$\left\{method: y_{n+2} = \frac{g_{(n+0)}h^2}{12} + \frac{5g_{n+1}h^2}{6} + \frac{g_{n+2}h^2}{12} - y_{n+0} + 2y_{n+1}\right\} \quad (11)$$

$$\left\{method: f_{n+0} = -\frac{89g_{(n+0)}h}{360} - \frac{31g_{n+1}h}{60} - \frac{11g_{n+2}h^2}{120} + \frac{16g_{n+\frac{3}{2}}h}{45} - \frac{y_{n+0}}{h} + \frac{y_{n+1}}{h}\right\} \quad (12)$$

$$\left\{method: f_{n+1} = \frac{31g_{(n+0)}h}{360} + \frac{13g_{n+1}h}{20} + \frac{3g_{n+2}h}{40} - \frac{14g_{n+\frac{3}{2}}h}{45} - \frac{y_{n+0}}{h} + \frac{y_{n+1}}{h}\right\} \quad (13)$$

$$\left\{method: f_{n+\frac{3}{2}} = \frac{33g_{(n+0)}h}{2880} + \frac{427g_{n+1}h}{480} + \frac{47g_{n+2}h}{960} - \frac{7g_{n+\frac{3}{2}}h}{360} - \frac{y_{n+0}}{h} + \frac{y_{n+1}}{h}\right\} \quad (14)$$

$$\left\{method: f_{n+2} = \frac{31g_{(n+0)}h}{360} + \frac{49g_{n+1}h}{60} + \frac{29g_{n+2}h}{120} - \frac{16g_{n+\frac{3}{2}}h}{45} - \frac{y_{n+0}}{h} + \frac{y_{n+1}}{h}\right\} \quad (15)$$

Eq. (11) to Eq. (15) is the Hybrid Linear Multistep method of [23] generated by the LMM generator. This result agrees with [23] results.

6. In generating hybrid Linear Multistep method [24], the interpolation points are "0,1" while the collocation points are "0,$\frac{1}{4}$,$\frac{3}{4}$, 1" for first derivative collocation and second derivative collocation. It was generated in 8.795 seconds.

$$\left\{method: y_{n+0} = \frac{-6f_{n+0}h}{378} - \frac{61f_{n+1}h}{378} - \frac{64f_{n+\frac{1}{4}}h}{180} - \frac{64f_{n+\frac{3}{4}}h}{180} - \frac{g_{n+0}h^2}{140} + \frac{g_{n+1}h^2}{140} - \frac{g_{n+\frac{1}{4}}h^2}{315} + y_{n+1}\right\} \quad (16)$$

$$\left\{method: y_{n+\frac{1}{4}} = \frac{-411f_{n+0}h}{7168} - \frac{1125f_{n+1}h}{7168} - \frac{177f_{n+\frac{1}{4}}h}{896} - \frac{303f_{n+\frac{3}{4}}h}{896} - \frac{279g_{n+0}h^2}{71680} + \frac{489g_{n+1}h^2}{71680} - \frac{633g_{n+\frac{1}{4}}h^2}{17920} + \frac{423g_{n+\frac{3}{4}}h^2}{17920} + y_{n+1}\right\} \quad (17)$$

$$\left\{method: y_{n+\frac{3}{4}} = \frac{-857f_{n+0}h}{193536} - \frac{20135f_{n+1}h}{193536} - \frac{11f_{n+\frac{1}{4}}h}{24192} - \frac{3413f_{n+\frac{3}{4}}h}{24192} - \frac{23g_{n+0}h^2}{71680} + \frac{233g_{n+1}h^2}{71680} - \frac{289g_{n+\frac{1}{4}}h^2}{161280} - \frac{160g_{n+\frac{3}{4}}h^2}{161280} + y_{n+1}\right\} \quad (18)$$

Eq. (17) to Eq. (18) is the hybrid Linear Multistep method of [24] generated by the LMM generator. This result agrees with [24] results.

7. In generating Linear Multistep method [25], the interpolation points are "1" while the collocation points are "1" for first derivative collocation and "0,1,2"second derivative collocation. It was generated in 0.419 seconds.

$$\left\{method: y_{n+0} = -f_{n+1}h + \frac{g_{n+0}h^2}{8} + \frac{5g_{n+1}h^2}{12} - \frac{g_{n+2}h^2}{24} + y_{n+1}\right\} \quad (19)$$

$$\left\{method: y_{n+2} = f_{n+1}h - \frac{g_{n+0}h^2}{24} + \frac{5g_{n+1}h^2}{12} + \frac{g_{n+2}h^2}{8} + y_{n+1}\right\} \quad (20)$$

$$\left\{method: f_{n+0} = f_{n+1} - \frac{5g_{n+0}h}{12} - \frac{2g_{n+1}h}{3} + \frac{g_{n+2}h}{12}\right\} \quad (21)$$

$$\left\{method: f_{n+2} = f_{n+1} - \frac{g_{n+0}h}{12} + \frac{2g_{n+1}h}{3} + \frac{5g_{n+2}h}{12}\right\} \qquad (22)$$

Eq. (19) to Eq. (22) Hybrid Linear Multistep methods of [25] generated by the LMM generator. It was generated in 0.419 seconds.

8. In generating hybrid Linear Multistep method [26], the interpolation points are "0" while the collocation points are "$0, \frac{1}{4}, \frac{2}{4}, \frac{3}{4}, 1$" for first derivative collocation and second derivative collocation. It was generated in 1.109 seconds.

$$\left\{method: y_{n+\frac{1}{4}} = \frac{251f_{n+0}h}{2880} - \frac{19f_{n+1}h}{2880} - \frac{11f_{n+\frac{1}{2}}h}{120} + \frac{323f_{n+\frac{1}{4}}h}{1440} + \frac{53f_{n+\frac{3}{4}}h}{1440} + y_{n+0}\right\} \qquad (23)$$

$$\left\{method: y_{n+\frac{1}{2}} = \frac{29f_{n+0}h}{360} - \frac{f_{n+1}h}{360} - \frac{1f_{n+\frac{1}{2}}h}{15} + \frac{31f_{n+\frac{1}{4}}h}{90} + \frac{1f_{n+\frac{3}{4}}h}{90} + y_{n+0}\right\} \qquad (24)$$

$$\left\{method: y_{n+\frac{3}{4}} = \frac{27f_{n+0}h}{320} - \frac{3f_{n+1}h}{320} - \frac{9f_{n+\frac{1}{2}}h}{40} + \frac{51f_{n+\frac{1}{4}}h}{160} + \frac{21f_{n+\frac{3}{4}}h}{160} + y_{n+0}\right\} \qquad (25)$$

$$\left\{method: y_{n+1} = \frac{7f_{n+0}h}{90} - \frac{7f_{n+1}h}{90} - \frac{2f_{n+\frac{1}{2}}h}{15} + \frac{16f_{n+\frac{1}{4}}h}{45} + \frac{16f_{n+\frac{3}{4}}h}{45} + y_{n+0}\right\} \qquad (26)$$

Eq. (19) to Eq. (22) is the hybrid Linear Multistep methods of [26] generated by the LMM generator.

## 3.4 User testing

**Summarized feedback from potential users, such as students and researchers.**
**Positive Feedback:**
The tool is praised for its intuitive and user-friendly interface, making LMM derivation accessible without requiring advanced programming skills. Its automation significantly reduces the time and effort compared to manual methods. Being web-based, it offers easy accessibility without installation or licensing requirements. The LaTeX output feature is especially beneficial for researchers incorporating results into academic papers. Students find it valuable for learning and visualizing LMM derivation. Additionally, researchers appreciate its computational efficiency and scalability, supported by the Python backend and FASTAPI framework.
**Constructive Feedback:**
1. Learning Curve for New User 2. Limited Customization 3. Internet Dependency 4. Integration with Other Tools

## CONCLUSION

The web-based tool is designed to automate the generation of Linear Multistep Methods (LMMs) of first and second derivative methods for solving ordinary differential equations (ODEs), addressing the limitations of traditional mathematical software. Its purpose is to simplify the derivation process, enhance accessibility, improve computational efficiency, and support education and research. Overall, the tool offers an efficient, accessible, and user-friendly solution for automating LMM derivation, benefiting both educational and research applications. The address of the web-based tool for the generation of LMM is "https://lmmgenerator.com/GenerateMethod"

## ACKNOWLEDGEMENT

The author(s) gratefully acknowledge the financial support provided by the National Open University of Nigeria (NOUN) for this research. This funding was instrumental in facilitating the study, enabling data collection, analysis, and dissemination of findings. The support and resources provided by NOUN have significantly contributed to the successful completion of this work.

## REFERENCES

[1] Butcher, J.C. (2016) *Numerical Methods for Ordinary Differential Equations*. 3rd edn. John Wiley & Sons.

[2] Hairer, E., Nørsett, S.P. & Wanner, G. (1993) *Solving Ordinary Differential Equations I: Nonstiff Problems*. 2nd edn. Springer.

[3] Iserles, A. (2009) *A First Course in the Numerical Analysis of Differential Equations*. 2nd edn. Cambridge University Press.

[4] Hairer, E. & Wanner, G. (1996) *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*. 2nd edn. Springer.

[5] Brugnano, L. & Trigiante, D. (1998) *Solving Differential Problems by Multistep Initial and Boundary Value Methods*. Gordon and Breach.

[6] Jackiewicz, Z. (2009) *General Linear Methods for Ordinary Differential Equations*. Oxford University Press.

[7] Majid, Z.A., Sulaiman, M.B. & Omar, Z. (2006) 'Three-point implicit block method for solving ordinary differential equations', *Bulletin of the Malaysian Mathematical Sciences Society*, 29(1), pp. 23–31.

[8] Awoyemi, D.O., Adebile, E.A., Adesanya, A.O. & Anake, T.A. (2011) 'Modified block for the direct solution for the second-order ordinary differential equations', *International Journal of Applied Mathematics and Computation*, 3(3), pp. 181–188.

[9] Jator, S.N. (2007) 'A sixth-order linear multistep method for the direct solution of $y'' = f(x, y, y')$', *International Journal of Pure and Applied Mathematics*, 40(1), pp. 457–472.

[10] Fornberg, B. (2017) *A Practical Guide to Pseudospectral Methods*. Cambridge University Press.

[11] Awoyemi, D. O., Kayode, S. J., & Adoghe, L. O. (2015). A six-step continuous multistep method for the solution of general fouth order initial value problems of ordinary differential equations. *Journal of natural sciences research.* 5(5), 2224-3186.

[12] Yahaya, Y. A. (2007). A Note on the construction of numerov method through quadratic continuous polynomial for the general second order ordinary differential equation. *Journal of Advances In Education and Professionalism*, 11, 261-268.

[13] Olabode, B. T., & Omole, E. O (2015). Implicit hybrid block numerov-type method for the direct solution of fourth-order ordinary differential equations. *American Journal of computational and applied mathematics,* 2015, 2165-8935.

[14] Kayode, S. J. (2014). Symmetric implicit multi-derivative numerical integration for direct solution of fifth-order differential equations. *Thammasat International Journal of science and Technology,* 19(2), 56-61.

[15] Awari, Y. S., Chima, E. E., kamoh. N. M., & Oladele, F. L. (2014). A family of implicit uniform accurate order block integrators for the solution of $y'' = f(x, y, y')$. *International Journal Of Mathematics And Statistics Invention,* 2(1), 33-46.

[16] Yahaya, Y. A., & Badmus, A. M. (2009). A class of collocation methods for general second order differential equation. *Africa Journal of Mathematical and Computer Science Research*, 2(4), 69-71.

[17] Trefethen, L.N. (2013) *Approximation Theory and Approximation Practice*. SIAM.

[18] Canuto, C., Hussaini, M.Y., Quarteroni, A. & Zang, T.A. (2007) *Spectral Methods: Fundamentals in Single Domains*. Springer

[19] Van Rossum, G. & Drake, F.L. (2009) *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace.

[20] Meurer, A., Smith, C.P., Paprocki, M., Čertík, O., Kirpichev, S.B., Rocklin, M., Kumar, A., Ivanov, S., Moore, J.K., Singh, S., Rathnayake, T., Vig, S., Granger, B.E., Muller, R.P., Bonazzi, F., Gupta, H., Vats, S., Johansson, F., Pedregosa, F., Curry, M.J., Terrel, A.R., Roučka, Š., Saboo, A., Fernando, I., Kulal, S., Cimrman, R. & Scopatz, A. (2017) 'SymPy: Symbolic computing in Python', *PeerJ Computer Science*, 3, e103. doi: https://doi.org/10.7717/peerj-cs.103

[21] Lambert J. D. (1991). *Numerical Methods for Ordinary Differential Systems, The Initial Value Problem*. *Wiley,* Chichester, New York.

[22] Yahaya, Y. A., Odeyemi, A. O., & Audu, K. J. (2022). *Block method approach for computation of errors of some Adams class of methods*. **Nigerian Journal of Physics, 31**(2), 66–77.

[23] Mohammed, U., Oyelami, O. and Semenov, M. (2019). An Orthogonal-based Self-starting Numerical Integrator for Second Order Initial and Boundary Value Problem ODEs. *Journal of Physics, Conference Series,* 1145(2019), 012040. Doi:10.1088/1742-6596/1145/1/012040.

[24] Mohammed U., Garba J., Semenov M.E. (2021). One-step second derivative block intra-step method for stiff system of ordinary differential equations. *Journal of Nigerian Mathematical Society*, 40(1) 47-57.

[25] Okuonghae, R.I. & Ozobokeme, J.K. (2024) 'Falkner hybrid block methods for second-order IVPs: A novel approach to enhancing accuracy and stability properties', *Journal of Numerical Analysis and Approximation Theory*, 53(2), pp. 324–342.

[26] Motsa, S. S. (2021). *Spectral Block Hybrid Methods.* UK-Africa Postgraduate Study Institute in Mathematical Sciences.