## **Adams-Bashforth-Moulton Method with Gauss-Lobatto Quadrature Points and Sectional Time Span (ODE113)**

**Purpose:** **Adams-Bashforth-Moulton (ODE113)** method is a **multi-step predictor-corrector method** designed for solving ordinary differential equations (ODEs) with both efficiency and accuracy. It combines the **Adams-Bashforth method** for predicting the next time step and the **Adams-Moulton method** for correcting this prediction, providing a balance between computational speed and precision. This method is particularly suitable for problems requiring long-term integration, such as satellite orbit calculations. The analysis extends ODE113 to work with **Gauss-Lobatto quadrature points**, which define time steps over irregular intervals, allowing the integration process to adapt to the complexity of the solution within each section.

In this approach, **Gauss-Lobatto points** define time steps for integration within smaller sections of the total time span. Each section is computed one after the other, making the method suitable for handling large-scale problems where the total time span is divided into manageable chunks.

**Overview:** The ODE113 method begins with **fourth-order Runge-Kutta (RK4)** steps to initialize the solution at the first few time points, ensuring accurate starting values. After the initial steps, the method transitions to the **Adams-Bashforth-Moulton predictor-corrector scheme**. The **Adams-Bashforth step** predicts the next state using previously computed derivatives, while the **Adams-Moulton step** refines this prediction for greater accuracy. By using **Gauss-Lobatto points** to define the time steps within a section, the method adapts to varying dynamics, making it effective for scenarios where the total time span is divided into smaller, more manageable segments.

### Mathematical Formulas and Coefficients

**Adams-Bashforth Method (Predictor)**

The Adams-Bashforth method is an explicit multistep method. The general formula for the k-step Adams-Bashforth method is:

where is the step size, is the current value, and is the function representing the ODE

The coefficients depend on the number of steps k.

**Adams-Moulton Method (Corrector)**

The Adams-Multon method is an implicit multistep method. The general formula for the k- steps Adams-Multon method is:

The coefficients ​ depend on the number of steps k.

**PECE Algorithm**

In the PECE (Predict, Evaluate, Correct, Evaluate) approach, the Adams-Bashforth method is used to predict the value of ​, and the Adams-Moulton method is used to correct this prediction.

* **Predict**: Use the Adams-Bashforth method to predict
* **Evaluate**: Evaluate the function at the predicted point
* **Correct**: Use the Adams-Moulton method to correct
* **Evaluate**: Recompute the function at the corrected point if needed

### Special Case

In cases where **only one Gauss-Lobatto point** is provided for a section, the ODE113 method cannot proceed as it requires at least two time points to compute a step size h=tn​ – tn+1. In your provided code, this situation is handled by **expanding the single Gauss-Lobatto point** into a small-time interval. This is done by generating additional time points around the single Gauss-Lobatto point to create a meaningful step size for ODE113 integration.

Mathematically, if the single Gauss-Lobatto point is t0​, the expanded time interval is defined as:

tlower = t0 -

tupper =

The new time points are then generated as:

texpanded={tlower, tlower + 0.1,tlower + 0.2,…,tupper}

This ensures that the ODE113 method has at least two time points to calculate a meaningful step size h=tn – tn+1 ​, allowing the integration to proceed smoothly.

After expanding the time span, the ODE113 method can then proceed with the integration, using the newly generated time points. This ensures that even in cases where only one Gauss-Lobatto point is provided, the algorithm performs meaningful integration and provides useful results.

### Pseudocode:

**Function ODE113(f, time\_points, y0, tol, hmax, hmin):**

**# Step 1: Handle Single Time Point**

**If length of time\_points == 1:**

# Expand the single time point into a small interval

time\_points = [time\_points[0] - 0.99 \* time\_points[0], time\_points[0] + small\_offset]

**# Step 2: Initialize Time and Solution Lists**

t\_values\_list = [time\_points[0]] # List for storing time points

y\_values\_list = [y0] # List for storing solution values

**# Step 3: Use RK4 for the Initial Few Steps**

**For i = 0 to 2:**

**If i >= length of time\_points - 1:**

Break # Exit loop if out of time points

t = time\_points[i]

h = time\_points[i + 1] - t # Calculate step size

**# Perform RK4 step to get the next value**

y\_next = rk4\_step(f, t, y\_values\_list[-1], h)

**# Store the time and solution values**

t\_values\_list.append(time\_points[i + 1])

y\_values\_list.append(y\_next)

**# Step 4: Apply Adams-Bashforth-Moulton for Subsequent Steps**

**For i = 3 to length of time\_points - 2:**

t = time\_points[i]

h = time\_points[i + 1] - t # Calculate step size

**# Predictor Step (Adams-Bashforth)**

y\_pred = y\_values\_list[-1] + h / 24 \* (

55 \* f(t, y\_values\_list[-1])

- 59 \* f(time\_points[i - 1], y\_values\_list[-2])

+ 37 \* f(time\_points[i - 2], y\_values\_list[-3])

- 9 \* f(time\_points[i - 3], y\_values\_list[-4])

)

**# Corrector Step (Adams-Moulton)**

y\_correct = y\_values\_list[-1] + h / 24 \* (

9 \* f(time\_points[i + 1], y\_pred)

+ 19 \* f(t, y\_values\_list[-1])

- 5 \* f(time\_points[i - 1], y\_values\_list[-2])

+ f(time\_points[i - 2], y\_values\_list[-3])

)

**# Store the updated time and solution values**

t\_values\_list.append(time\_points[i + 1])

y\_values\_list.append(y\_correct)

**# Step 5: Return Time and Solution Values**

**Return (t\_values\_list, y\_values\_list)**

### Time Complexity:

* **Per Iteration:** O (1)
* **Total Complexity:** O (n), where 𝑛 = Number of Gauss-Lobatto points

**Explanation:** Each iteration involves a fixed number of operations to perform the predictor-corrector steps using the Adams-Bashforth and Adams-Moulton formulas. As the total number of iterations is n, the time complexity is linear with respect to the number of Gauss-Lobatto points. The initial RK4 steps also require a constant amount of work, which does not significantly affect the overall complexity.

### Space Complexity:

* **Overall:** O (), where 𝑛 = Number of Gauss-Lobatto points

**Explanation:** The method requires memory to store the time points and the solution values at each time step. This results in a space complexity of O(n × m), where *n* is the number Gauss-Lobatto points, and *m* is the dimension of the solution vector *y*, since *y* is assumed to have a constant dimension (6), the space requirement primarily scales with the number of time steps *n*. Additionally, a fixed amount of space is needed for intermediate calculations, such as the slopes predicator and corrector calculations and the current values of *y* and *t*. However, these constant space requirements do not impact the overall space complexity, which is dominated by the size of the problem.

### Edge Cases and Limitations

When only one Gauss-Lobatto point is provided, the code expands the time span into a small interval to enable meaningful integration. Large step sizes can introduce inaccuracies, but dividing the time span into smaller sections helps maintain precision. Conversely, small step sizes increase accuracy but also computational time, while Gauss-Lobatto points enable non-uniform time steps that adapt to the dynamics of the solution. ODE113 is not ideal for stiff ODEs due to its explicit nature, making implicit methods more suitable for such problems.

**Conclusion:** The Adams-Bashforth-Moulton method (ODE113) with Gauss-Lobatto points is a robust multi-step integrator for solving ODEs over long durations. Its combination of predictor-corrector steps allows for accurate and efficient calculations, making it ideal for scenarios like satellite motion where precision is critical. While the initial RK4 steps provide accurate starting values, the method's ability to adapt using Gauss-Lobatto points ensures stable time-stepping. However, for scenarios involving stiff equations or where highly adaptive step sizing is needed, alternative methods like implicit solvers may offer better stability.