## **Dormand-Price Method with Gauss-Lobatto Quadrature Points and Sectional Time Span (ODE45)**

**Purpose:** **ODE45** method is a popular adaptive step-size integrator for solving ordinary differential equations (ODEs). It uses the **Dormand-Prince method (RK45)**, which provides both fourth-order and fifth-order accurate solutions to estimate errors and adjust the step size accordingly. This method is well-suited for problems where precision and computational efficiency are critical, such as satellite orbit propagation. The analysis extends ODE45 to work with **Gauss-Lobatto quadrature points** for defining time steps, allowing for non-uniform intervals that adapt to the solution's dynamics. This approach is particularly effective for applications requiring high accuracy over irregular time spans.

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:** ODE45 computes the solution of an ODE by evaluating multiple intermediate stages (k1, k2, …, k7) at each step, using coefficients from the Dormand-Prince Butcher table. These stages yield both fourth-order and fifth-order solutions, which help estimate the local error and adapt the step size to achieve the desired accuracy. The use of **Gauss-Lobatto points** enables the definition of time steps over a specified interval, ensuring that the time steps adapt to the problem's dynamics while maintaining the desired accuracy.

### Mathematical Formulas and Coefficients Table

**Formulas:**

**Update Formula**

**General Formulation**

**Error Estimation**

The Error Estimate can be computed using different set of weights



​: The current value of the solution.

​: The next value of the solution.

: The step size, defined as the difference between consecutive time points

Here, tn​ and tn+1​ are the Gauss-Lobatto points for the current and next time steps, respectively. The difference between these time points determines the step size h, which can vary from step to step based on the distribution of the Gauss-Lobatto points.

: The number of stages in the Runge-Kutta method **in this case** **7**.

​: The weights used to combine the intermediate slopes to obtain the

final solution.

​: The intermediate slopes, calculated using the function at different

points.

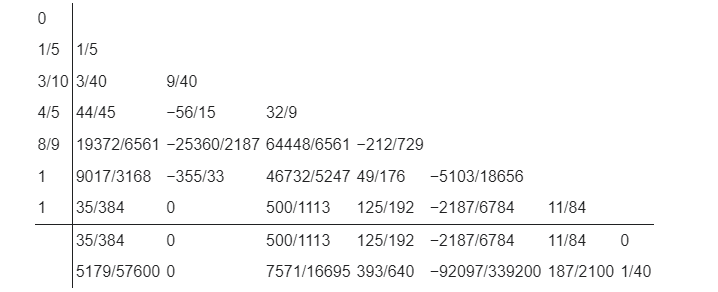
: The current time point.

​: The coefficients that determine the evaluation points within the step.

​: The coefficients that weight the contributions of the intermediate slopes

to calculate the next slope .

**Coefficient Table for ODE45:**



### Special Case

In cases where **only one Gauss-Lobatto point** is provided for a section, the ODE45 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 ODE45 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 ODE45 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 ODE45 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 rk45\_step(func, t, y, h, rtol, atol)**

**# Step 1: Initialize Butcher Tableau Coefficients**

a = [0, 1/5, 3/10, 4/5, 8/9, 1, 1]

b = matrix of values for intermediate stages

c4 = coefficients for 4th-order solution

c5 = coefficients for 5th-order solution

**# Step 2: Calculate RK45 Stages using Butcher Table Coefficients**

K1 = h \* func(t, y)

K2 = h \* func(t + a[1] \* h, y + b[1][0] \* K1)

K3 = h \* func(t + a[2] \* h, y + b[2][0] \* K1 + b[2][1] \* K2)

K4 = h \* func(t + a[3] \* h, y + b[3][0] \* K1 + b[3][1] \* K2 + b[3][2] \* K3)

K5 = h \* func(t + a[4] \* h, y + b[4][0] \* K1 + b[4][1] \* K2 + b[4][2] \* K3 + b[4][3] \* K4)

K6 = h \* func(t + a[5] \* h, y + b[5][0] \* K1 + b[5][1] \* K2 + b[5][2] \* K3 + b[5][3] \* K4 + b[5][4] \* K5)

K7 = h \* func(t + a[6] \* h, y + b[6][0] \* K1 + b[6][1] \* K2 + b[6][2] \* K3 + b[6][3] \* K4 + b[6][4] \* K5 + b[6][5] \* K6)

**# Step 3: Compute 4th and 5th Order Solutions**

y4 = y + dot product of c4 and [K1, K2, K3, K4, K5, K6, K7]

y5 = y + dot product of c5 and [K1, K2, K3, K4, K5, K6, K7]

**# Step 4: Estimate the Error**

error = norm(y5 - y4) / (atol + rtol \* max(norm(y4), norm(y5)))

**# Step 5: Adjust Step Size Based on Error Estimate**

**If error is not zero:**

h\_new = h \* min(2, max(0.1, 0.9 / error ^ 0.2))

**Else:**

h\_new = h \* 2

**# Step 6: Return the Next Time, Updated Solution, and New Step Size**

**Return t + h, y5, h\_new**

**Function ode45(func, t\_span, y0, rtol, atol)**

**# Step 1: Handle Single Gauss-Lobatto Point**

**If length of t\_span == 1:**

t\_start = t\_span[0] - 0.99 \* t\_span[0]

t\_end = t\_span[0] + small\_offset

t\_span = array of values from t\_start to t\_end with small increments

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

Initialize tout = [t\_span[0]] # List to store time points

Initialize yout = [y0] # List to store state vectors

Set t = t\_span[0] # Current time

Set y = y0 # Initial state vector

**# Step 3: Loop through Gauss-Lobatto Points**

**For each point i in t\_span from 1 to length of t\_span:**

**# Calculate step size**

h = t\_span[i] - t\_span[i-1]

**# Step 4: While the current time is less than the next Gauss-Lobatto point**

**While t < t\_span[i]:**

# Call rk45\_step to compute the next time and state

t\_next, y\_next, h = rk45\_step(func, t, y, h, rtol, atol)

**# Update t and y for the next step**

t = t\_next

y = y\_next

**# Append the current time and state to the result lists**

Append t to tout

Append y to yout

**# Step 5: Convert Output Lists to Arrays**

Convert tout and yout to arrays

**# Step 6: Return Time and Solution in Column Stack Format**

**Return column stack of tout and yout**

### Time Complexity:

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

**Explanation:** Each iteration involves a constant number of operations to compute the 7 stages (k1, k2, …, k7) and update the solution. As the total number of iterations is n, the time complexity grows linearly with the number of Gauss-Lobatto points in the section

### 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 (k1, k2, …, k7) 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 reduce accuracy, but dividing the problem into sections with smaller intervals 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. However, ODE45 is not ideal for stiff ODEs due to its explicit nature, making implicit methods a better choice for such systems.

**Conclusion:** The ODE45 method with Gauss-Lobatto points is an effective numerical integrator for solving ODEs with adaptive time steps. It offers a good trade-off between accuracy and computational efficiency, making it suitable for precise simulations like satellite motion. However, for scenarios involving stiff equations or when very fine error control is required, other methods like implicit solvers or methods with built-in stiffness handling may be more appropriate.