## **Runge-Kutta 8th Order with Gauss-Lobatto Quadrature Points and Sectional Time Span (RK8)**

**Purpose:** **RK8 (Runge-Kutta 8th Order)** method is a higher-order numerical technique for solving ordinary differential equations (ODEs), It offers greater accuracy per step compared to lower-order methods like RK4, making it suitable for applications that require high precision, such as satellite orbit propagation. This analysis extends the RK8 method to handle multiple **Gauss-Lobatto quadrature sections** of a longer time span, which is crucial for applications like satellite orbit propagation around Earth.

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 RK8 method solves ODEs using a set of intermediate stages

(k1, k2, …, k13) derived from the **Butcher table coefficients** of the Dormand-Prince 8 method (DP8). These stages are combined to achieve an eighth-order accurate solution, providing high precision with a larger number of intermediate evaluations.

This implementation uses **Gauss-Lobatto points** to define time intervals over which the integration is performed, adapting to the characteristics of the solution. This approach ensures that each step size is suited to the complexity of the dynamics being modeled, making it particularly effective for precise calculations like satellite motion.

### Mathematical Formulas and Coefficients Table

**Formulas:**



**Update Formula**

**General Formulation**

​: 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** **13**.

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

final solution.

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

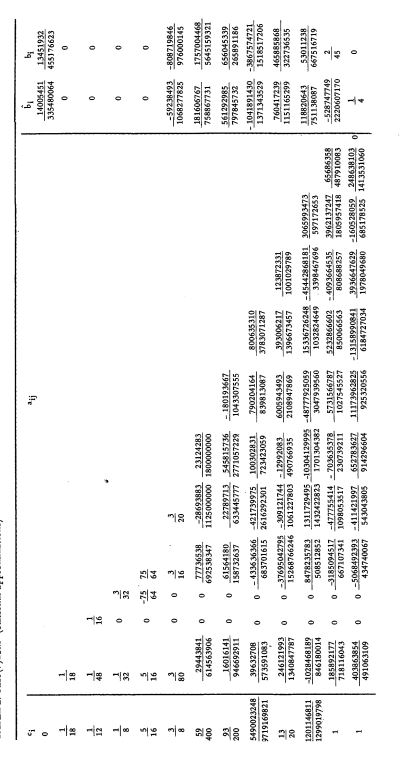
points.

: 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 RK8:**

### Special Case

In cases where **only one Gauss-Lobatto point** is provided for a section, the RK8 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 RK8 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 RK8 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 RK8 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 RK8(f, t\_gauss\_lobatto, Y0):**

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

**If length of t\_gauss\_lobatto == 1:**

**# Expand the single Gauss-Lobatto point into a small interval**

t\_gauss\_lobatto = [t\_gauss\_lobatto[0] - 0.99 \* t\_gauss\_lobatto[0], t\_gauss\_lobatto[0] + small\_offset]

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

tout = t\_gauss\_lobatto

yout = array of zeros with size (length of tout, length of Y0)

**# Set initial condition for the solution**

y = Y0

yout[0] = y

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

**For i = 1 to length of tout - 1:**

**# Calculate step size**

h\_step = tout[i] - tout[i - 1]

**# Initialize k values based on the Butcher table for RK8**

k = array of zeros with size (number of stages from Butcher table, length of Y0)

**# Step 4: Compute k values using Butcher table coefficients**

**For j = 0 to length of Butcher\_table\_DP8['c'] - 1:**

**If j == 0:**

y\_temp = y

**Else:**

y\_temp = y + h\_step \* sum(Butcher\_table\_DP8['a'][j][l] \* k[l] for l in range(0, j))

**# Compute k[j] using function f at the appropriate time step**

k[j] = f(tout[i - 1] + Butcher\_table\_DP8['c'][j] \* h\_step, y\_temp)

**# Step 5: Update the state vector y using Butcher table b coefficients**

y = y + h\_step \* sum(Butcher\_table\_DP8['b'][j] \* k[j] for j in range(0, length of k))

**# Store the updated solution in yout**

yout[i] = y

**# Step 6: Return Time Points and Solution**

**Return tout, 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 13 intermediate slopes (k1, k2, …, k13) and update the solution. Since the total number of iterations is n, the time complexity grows linearly with the number of Gauss-Lobatto points in the section. Each time step requires the same number of computations for evaluating the slopes and updating the solution.

### 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, …, k13) 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, RK8 is not ideal for stiff ODEs due to its explicit nature, making implicit methods a better choice for such systems.

**Conclusion:** The RK8 method with Gauss-Lobatto points is a highly accurate and computationally efficient method for solving ODEs, especially in scenarios requiring precise calculations, such as satellite motion. It provides a good balance between high accuracy and computational cost per step, making it ideal for long-duration simulations. However, for cases like stiff equations or when adaptive step sizing is needed, alternative methods may be more suitable.