## **Runge-Kutta 4th Order with Gauss-Lobatto Quadrature Points and Sectional Time Span (RK4)**

**Purpose:** **RK4 (Runge-Kutta 4th Order)** method is a widely-used technique for solving ordinary differential equations (ODEs), providing a balance between accuracy and computational efficiency. This analysis extends the RK4 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 RK4 method computes the solution of an ODE at discrete time steps using four intermediate slope evaluations (k1, k2, k3, k4) at each step. These slopes are combined in a weighted manner to produce an accurate estimate of the solution at the next time step.

In this version, the total time span ​ (e.g., the time it takes for a satellite to orbit Earth) is divided into multiple **sections**, each of which is integrated using Gauss-Lobatto quadrature points to define the time steps within that section. This allows for efficient integration over a large time span without excessive computational cost or error accumulation.

.

### 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** **4**.

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

final solution.

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

points.

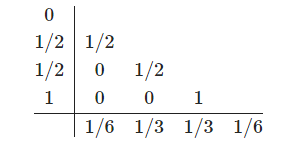
: The current time point.

​: 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 RK4:**



### Special Case

In cases where **only one Gauss-Lobatto point** is provided for a section, the RK4 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 RK4 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 RK4 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 RK4 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 RK4(odefun, gauss\_lobatto\_points, y0):**

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

**If length of gauss\_lobatto\_points == 1:**

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

gauss\_lobatto\_points = [gauss\_lobatto\_points[0] - small\_offset, gauss\_lobatto\_points[0] + small\_offset]

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

tout = gauss\_lobatto\_points

yout = array of zeros with size (length of tout, length of y0)

**# Set the 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 = tout[i] - tout[i - 1]

**# Compute the four Runge-Kutta increments**

k1 = h \* odefun(tout[i - 1], y)

k2 = h \* odefun(tout[i - 1] + h / 2, y + k1 / 2)

k3 = h \* odefun(tout[i - 1] + h / 2, y + k2 / 2)

k4 = h \* odefun(tout[i - 1] + h, y + k3)

**# Update the solution y at the next time step**

y = y + (k1 + 2 \* k2 + 2 \* k3 + k4) / 6

**# Store the updated solution in yout**

yout[i] = y

**# Step 4: 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 four slopes (k1, K2, k3, k4) 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. This is because 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, k3, k4) 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

If only one Gauss-Lobatto point is provided, the code expands the time span to create a small interval for meaningful integration. Large step sizes can lead to inaccuracies, but dividing the total time span into sections helps reduce this by enabling smaller, more manageable steps. Small step sizes, while improving accuracy, increase computation time, but the use of Gauss-Lobatto points allows for non-uniform time steps that adapt to the complexity of the solution in each section.

**Conclusion:** The RK4 method with Gauss-Lobatto points is a robust and efficient approach for solving ordinary differential equations, particularly in scenarios like satellite motion where non-uniform time steps are beneficial. It offers a good balance between accuracy and computational cost, especially when applied over smaller sections of a larger time span. However, for cases where larger step sizes could introduce inaccuracies, or when dealing with stiff equations.