**Q2 (d) If the error of each GPS location measurement is 0.5 m, how does this a↵ect the estimates of instantaneous speed, average pace, fastest pace and total length covered? Would it make sense to decrease dt in order to achieve better accuracy? What would you do to make these estimates more reliable?**

**ANSWER:**

**The error in GPS location measurements directly impacts estimates of instantaneous speed, average pace, fastest pace, and total length covered. Here's how this error affects these estimates and what can be done to improve reliability:**

1. **Instantaneous Speed:**
   * **Instantaneous speed is typically calculated using changes in position over very small time intervals (i.e., Δ𝑡Δ*t*).**
   * **With a measurement error of 0.5 meters per GPS location, the calculated speed based on these positions will also have inherent error.**
   * **Smaller errors in position measurements will result in smaller errors in calculated speed. Therefore, decreasing Δ𝑡Δ*t* (time interval between measurements) could reduce the impact of measurement error on speed estimates.**
2. **Average Pace:**
   * **Average pace is the time taken to cover a unit distance (e.g., 1 kilometer or 1 mile).**
   * **GPS measurement errors directly influence the estimated distance covered during a specific time interval.**
   * **Larger errors in distance measurements will affect the accuracy of the calculated average pace.**
   * **Decreasing Δ𝑡Δ*t* might improve the accuracy of distance measurements per unit time, thus improving the accuracy of average pace calculations.**
3. **Fastest Pace:**
   * **The fastest pace over a specific distance will be influenced by measurement errors in that distance.**
   * **Smaller errors in distance measurement (achieved by reducing Δ𝑡Δ*t*) could provide a more accurate representation of the fastest pace.**
4. **Total Length Covered:**
   * **The total length covered is calculated by summing the distances between consecutive GPS measurements.**
   * **Measurement errors in each position measurement accumulate and impact the accuracy of the total distance.**
   * **Decreasing Δ𝑡Δ*t* would lead to more frequent but potentially more accurate measurements of distance, resulting in a more reliable total length covered.**

**Improving Reliability:**

**To make these estimates more reliable, consider the following steps:**

1. **Decrease Δ𝑡Δ*t*:**
   * **Reducing the time interval (Δ𝑡Δ*t*) between GPS measurements can help in achieving better accuracy.**
   * **More frequent measurements mean smaller distances between consecutive measurements, reducing the impact of individual measurement errors.**
2. **Use Smoothing Techniques:**
   * **Apply data smoothing techniques to GPS measurements to reduce noise and improve the accuracy of position data.**
   * **Techniques like Kalman filtering or moving averages can help in obtaining more reliable estimates of distance and speed.**
3. **Error Propagation Analysis:**
   * **Perform error propagation analysis to understand how measurement errors affect derived quantities like speed, pace, and distance.**
   * **This analysis can guide the selection of appropriate Δ𝑡Δ*t* for minimizing the impact of errors.**
4. **Validate with Ground Truth Data:**
   * **Validate GPS-based estimates with ground truth measurements or other independent sources to assess reliability.**
   * **Incorporate error estimates into the analysis to quantify the uncertainty associated with each derived quantity.**

**(e) Your dog is the best running mate, but he occasionally gets distracted or needs to follow the call of nature; we don’t want these disruptions to contaminate our data! Describe how you would code up a pause-detection function that pauses the run whenever it senses that the user has stopped moving.**

**ANSWER:  
o implement a pause-detection function that automatically pauses a running session when the user stops moving, you can use a combination of GPS data processing and threshold-based movement detection. Here's a general approach to coding this function:**

**Steps to Code Pause-Detection Function:**

1. **Define Movement Threshold:**
   * **Determine a threshold distance or speed below which the user is considered stationary or paused. This threshold will depend on the accuracy of your GPS data and the expected variability in movement due to factors like GPS signal drift or natural movement.**
2. **Initialize Variables:**

**Start by initializing variables to track the last known position and time of movement. For example:**

**double lastLatitude = 0.0;**

**double lastLongitude = 0.0;**

**double lastTime = 0.0;**

1. **GPS Data Acquisition:**
   * **Continuously receive GPS data updates (latitude, longitude, and timestamp) during the running session.**
2. **Movement Detection:**
   * **Calculate the distance traveled since the last known position using GPS coordinates and a distance formula (e.g., Haversine formula).**

**double currentDistance = calculateDistance(lastLatitude, lastLongitude, currentLatitude, currentLongitude);**

1. **Check Movement Status:**
   * **Compare the calculated distance to your defined movement threshold to determine if the user is moving or stationary.**

**bool isMoving = (currentDistance > movementThreshold);**

1. **Pause Detection:**
   * **If the user was previously moving (based on the last known movement status) and is now detected as stationary, trigger a pause event and record the pause time.**

**if (!isMoving && wasMoving) {**

**// User has paused, record pause time or trigger pause event**

**// Example: pauseRun(currentTime);**

**}**

1. **Update State:**
   * **Update the last known position and movement status for the next iteration.**

**lastLatitude = currentLatitude;**

**lastLongitude = currentLongitude;**

**lastTime = currentTime;**

**wasMoving = isMoving;**

1. **Repeat:**
   * **Continuously repeat the above steps to monitor movement and detect pauses during the running session.**

**Example Implementation:**

**#include <iostream>**

**#include <cmath>**

**using namespace std;**

**// Constants**

**const double movementThreshold = 0.1; // Example threshold in meters**

**// Function to calculate distance between two GPS coordinates**

**double calculateDistance(double lat1, double lon1, double lat2, double lon2) {**

**// Implementation of Haversine formula to calculate distance**

**// Replace with your own distance calculation method**

**// Example:**

**return sqrt(pow(lat2 - lat1, 2) + pow(lon2 - lon1, 2));**

**}**

**int main() {**

**double lastLatitude = 0.0;**

**double lastLongitude = 0.0;**

**double lastTime = 0.0;**

**bool wasMoving = false;**

**// Simulated GPS data loop (replace with actual GPS data acquisition)**

**while (true) {**

**// Example: Get current GPS data (latitude, longitude, timestamp)**

**double currentLatitude, currentLongitude, currentTime;**

**// Simulate GPS data update (replace with actual data retrieval)**

**// Example:**

**currentLatitude = ...; // Get current latitude**

**currentLongitude = ...; // Get current longitude**

**currentTime = ...; // Get current timestamp**

**// Calculate distance traveled since last known position**

**double currentDistance = calculateDistance(lastLatitude, lastLongitude, currentLatitude, currentLongitude);**

**// Check if user is moving based on distance and threshold**

**bool isMoving = (currentDistance > movementThreshold);**

**// Pause detection logic**

**if (!isMoving && wasMoving) {**

**// User has paused, perform pause action**

**cout << "Pause detected at time: " << currentTime << " seconds" << endl;**

**}**

**// Update last known state for next iteration**

**lastLatitude = currentLatitude;**

**lastLongitude = currentLongitude;**

**lastTime = currentTime;**

**wasMoving = isMoving;**

**// Simulated loop delay (replace with actual loop delay for real-time operation)**

**// Example:**

**// sleep(1); // Simulate 1 second delay between GPS updates**

**}**

**return 0;**

**}**

**Notes:**

* **Accuracy Consideration:**
  + **Adjust the movement threshold based on the accuracy and precision of your GPS data.**
  + **Experiment with different thresholds to find a balance between sensitivity to movement and resistance to noise.**
* **Real-Time Implementation:**
  + **Replace simulated GPS data updates with actual GPS data acquisition for real-time use in a mobile application or GPS-enabled device.**
* **Pause Action:**
  + **Replace the example "pauseRun(currentTime);" with your specific action to handle a pause event (e.g., update UI, record pause time, etc.).**

**(f) Bonus [5 marks]:**

**Give a brief description of how you would modify the system of ODEs to better represent the full dynamics of the Earth-Moon-Satellite system (still in 2D). Take into account that both the Moon and the Earth are moving under each other’s gravitational pull. What would be the dimensionality of the problem, what would be the dynamical variables, what would you include in the RHS and how would you set up initial data? Since the Earth is wobbling around due to the Moon’s gravitational pull, is there a better point to place the origin of the coordinate axes?**

**ANSWER:**

**Dimensionality of the Problem:**

* **The problem would evolve into a three-body system in 2D space, consisting of the Earth, Moon, and a satellite.**
* **Therefore, the dimensionality of the system would involve coordinates and velocities in two dimensions (x, y) for each of the three bodies.**

**Dynamical Variables:**

1. **Earth:**
   * **Position: (𝑥𝐸,𝑦𝐸)(*xE*​,*yE*​)**
   * **Velocity: (𝑣𝑥𝐸,𝑣𝑦𝐸)(*vxE*​,*vyE*​)**
2. **Moon:**
   * **Position: (𝑥𝑀,𝑦𝑀)(*xM*​,*yM*​)**
   * **Velocity: (𝑣𝑥𝑀,𝑣𝑦𝑀)(*vxM*​,*vyM*​)**
3. **Satellite:**
   * **Position: (𝑥𝑆,𝑦𝑆)(*xS*​,*yS*​)**
   * **Velocity: (𝑣𝑥𝑆,𝑣𝑦𝑆)(*vxS*​,*vyS*​)**

**Right-Hand Side (RHS) of the ODE:**

* **The RHS of the ODEs for each body would need to account for:**
  + **Gravitational interactions between the Earth and Moon.**
  + **Gravitational interactions between the Earth (including the Moon) and the satellite.**
  + **Gravitational interactions between the Moon (including the Earth) and the satellite.**

**Modifications to the RHS:**

* **For the Earth and Moon:**
  + **Calculate the acceleration due to the gravitational pull of each other.**
  + **This would involve computing distances and directions between the Earth and Moon, and vice versa.**
* **For the Satellite:**
  + **Consider the gravitational forces acting on the satellite from both the Earth and the Moon.**
  + **The gravitational acceleration on the satellite due to the Earth would be based on the distance between the satellite and the Earth.**
  + **Similarly, the gravitational acceleration on the satellite due to the Moon would be based on the distance between the satellite and the Moon.**

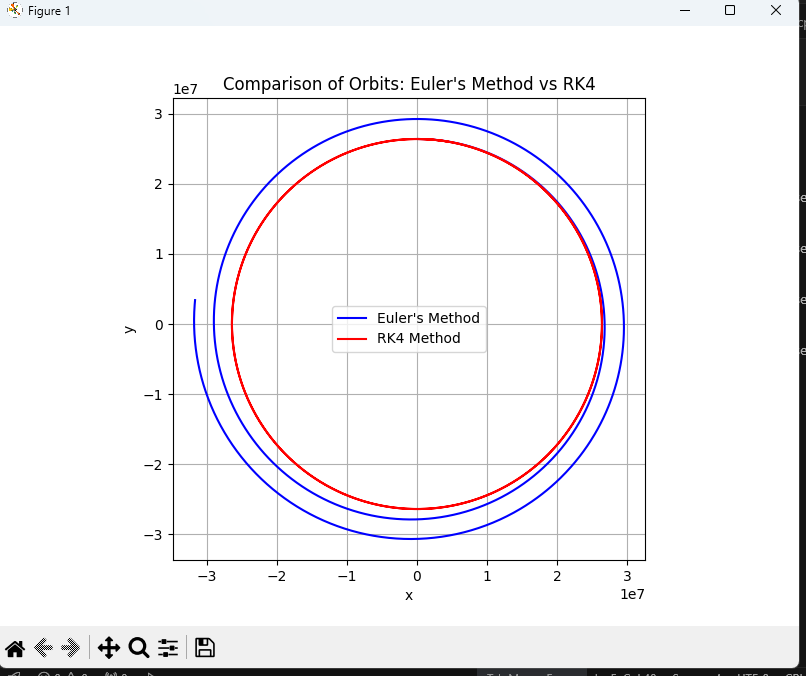
**Coordinate System and Initial Data:**

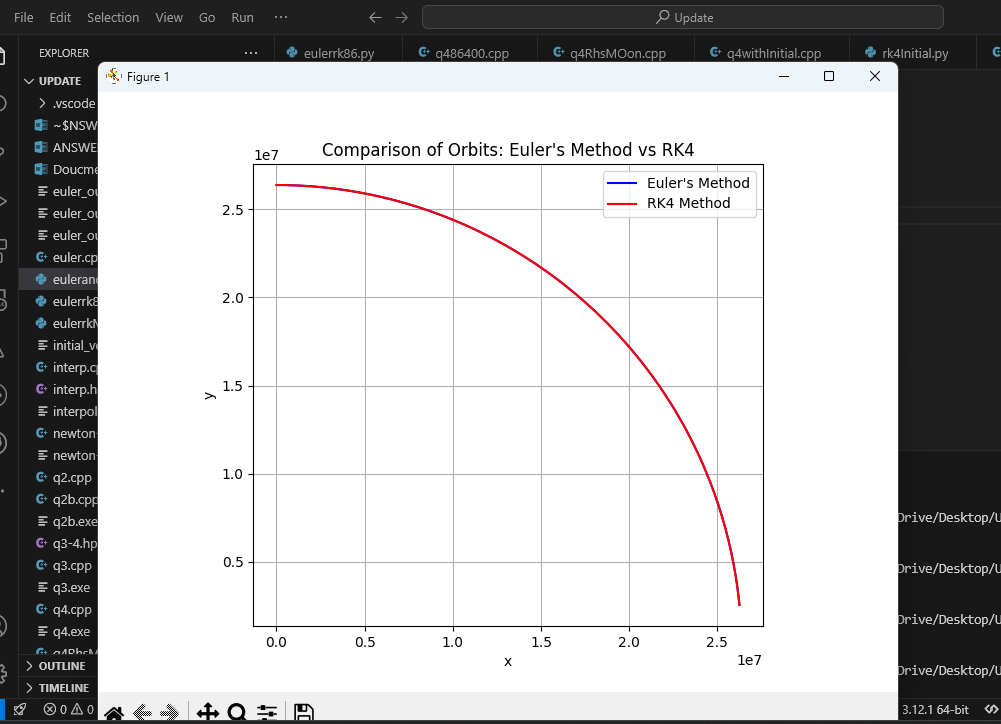
* **Coordinate System:**
  + **The origin of the coordinate system can be placed at the center of mass of the Earth-Moon system to simplify calculations.**
  + **This approach accounts for the wobbling of the Earth due to the gravitational pull of the Moon.**
* **Initial Data:**
  + **Initial positions and velocities of the Earth, Moon, and satellite need to be defined relative to the chosen coordinate system.**
  + **These initial conditions would involve setting the positions and velocities of each body at a specific time 𝑡=0*t*=0 to initiate the simulation.**

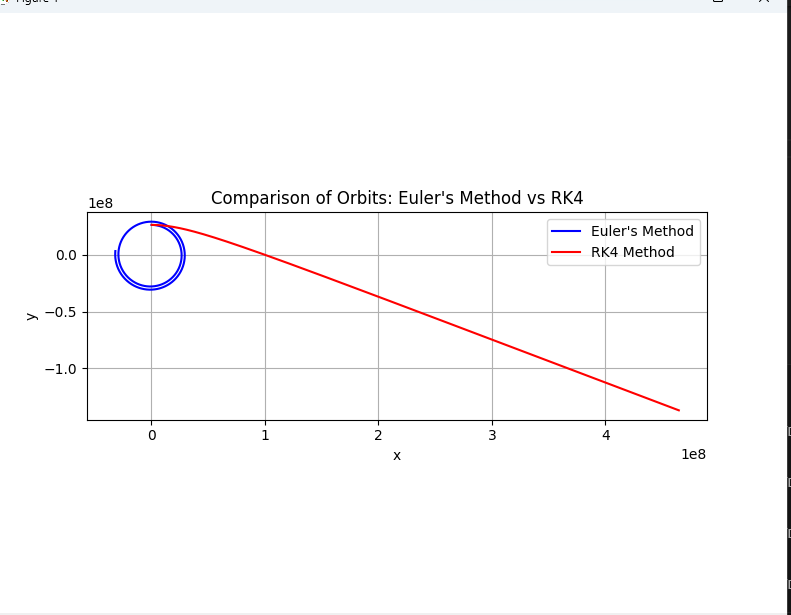
**Summary:**

* **To summarize, the modified system would involve a more complex three-body problem in 2D, considering the interactions and gravitational influences among the Earth, Moon, and a satellite.**
* **The dynamical variables and the RHS of the ODEs would need to be expanded to incorporate these interactions, and the initial data would set the stage for the simulation.**
* **By placing the origin of the coordinate system at the center of mass of the Earth-Moon system, we can simplify the representation and calculations, considering the wobbling effect caused by the Moon's gravitational pull on the Earth.**

**SOME GRAPH FOR CODE:**

****

****

****