Ball Shooting Simulation Using Numerical Methods

FINAL 1

Author: Terenti Kakhniashvili

Date: January 19, 2025

Contents

1 Problem Statement

problems is the following i have the image where the balls are displayed, i have to detect these balls, then user inputs the position from where these balls have to be targeted and hitted, so poblem is the following i have the inital value as the user input i have the boundary value ball, center of the ball, and i have to determine the initial velocity to hit the ball, also trajectory should not have to be the straight line.

2 Edge Detection and Image Processing

2.1 Kernel Size Analysis

The edge detection process employs multiple kernel sizes for morphological operations:

$$K_{\text{large}} = 11 \times 11 \text{ (elliptical)} \tag{1}$$
$$K_{\text{medium}} = 7 \times 7 \text{ (elliptical)} \tag{2}$$
$$K_{\text{small}} = 3 \times 3 \text{ (elliptical)} \tag{3}$$

The choice of kernel size significantly impacts edge detection quality:

- Large kernels (11×11):
  - Better noise suppression
  - Smoother edge contours
  - Risk of losing fine details
- Medium kernels (7×7):
  - Balance between detail preservation and noise reduction
  - Optimal for intermediate-scale features
- Small kernels (3×3):
  - Preserve fine details
  - More susceptible to noise
  - Better for sharp edge detection

The multi-scale approach using different kernel sizes enables hierarchical refinement of edge detection, where larger kernels establish primary contours while smaller kernels preserve critical details.

3 Physical Model and Ball Motion

3.1 Ordinary Differential Equations

The ball's motion is governed by a system of coupled ODEs incorporating gravity and air resistance:

$$\frac{dx}{dt} = v_x \tag{4}$$
$$\frac{dy}{dt} = v_y \tag{5}$$
$$\frac{dv_x}{dt} = -kv_x \tag{6}$$
$$\frac{dv_y}{dt} = -g - kv_y \tag{7}$$

where:

- (x, y): Position coordinates
- (v_x, v_y): Velocity components
- k: Air resistance coefficient
- g: Gravitational acceleration

This system represents a second-order ODE converted to a system of first-order ODEs, making it suitable for numerical integration.

4 Numerical Methods Analysis

4.1 Runge-Kutta Fourth Order (RK4)

The RK4 method provides fourth-order accuracy through four evaluations per step:

$$k_1 = hf(y_n, t_n) \tag{8}$$
$$k_2 = hf(y_n + \frac{k_1}{2}, t_n + \frac{h}{2}) \tag{9}$$
$$k_3 = hf(y_n + \frac{k_2}{2}, t_n + \frac{h}{2}) \tag{10}$$
$$k_4 = hf(y_n + k_3, t_n + h) \tag{11}$$
$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{12}$$

Local truncation error: $O(h^5)$ Global truncation error: $O(h^4)$

4.2 Euler Method

The simplest numerical integration method:

$$y_{n+1} = y_n + hf(y_n, t_n)$$

Local truncation error: $O(h^2)$ Global truncation error: $O(h)$

4.3 Adams-Bashforth Method

A multi-step method using previous solution points:

$$y_{n+1} = y_n + h(\frac{55}{24}f_n - \frac{59}{24}f_{n-1} + \frac{37}{24}f_{n-2} - \frac{9}{24}f_{n-3})$$

Local truncation error: $O(h^5)$ Global truncation error: $O(h^4)$

4.4 Adams-Moulton Method

An implicit multi-step method:

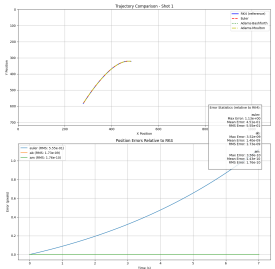
$$y_{n+1} = y_n + h(\frac{9}{24}f_{n+1} + \frac{19}{24}f_n - \frac{5}{24}f_{n-1} + \frac{1}{24}f_{n-2})$$

Local truncation error: $O(h^5)$ Global truncation error: $O(h^4)$

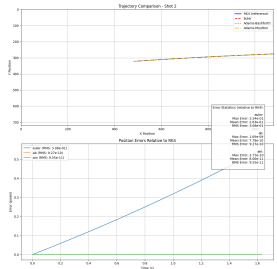
4.5 Stability Analysis

For the test equation $y' = \lambda y$:

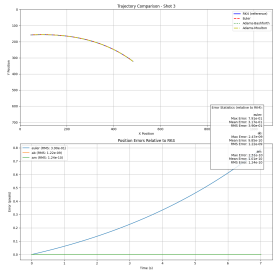
- RK4: Stability region includes imaginary axis, suitable for oscillatory problems
- Euler: Limited stability region, requires very small step sizes
- Adams-Bashforth: Moderate stability region, zero-stable
- Adams-Moulton: Larger stability region due to implicit nature



(a) Enter Caption 1



(b) Enter Caption 2



(c) Enter Caption 3

Figure 1: plot comparisons

5 Newton's Shooting Method

5.1 Mathematical Formulation

The shooting method uses Newton iteration to find initial velocities:

$$r(v_0) = \begin{bmatrix} x_{\text{sim}}(v_0) - x_{\text{target}} \\ y_{\text{sim}}(v_0) - y_{\text{target}} \end{bmatrix} \tag{13}$$

$$J = \begin{bmatrix} \frac{\partial r_x}{\partial v_{x0}} & \frac{\partial r_x}{\partial v_{y0}} \\ \frac{\partial r_y}{\partial v_{x0}} & \frac{\partial r_y}{\partial v_{y0}} \end{bmatrix} \tag{14}$$

$$\Delta v_0 = -(J^T J + \lambda I)^{-1} J^T r \tag{15}$$

5.2 Regularization

The Levenberg-Marquardt regularization term λI addresses:

- Ill-conditioning in the Jacobian
- Numerical stability
- Convergence robustness

5.3 Computational Complexity

The method's runtime complexity is influenced by:

- Trajectory integration: $O(N)$ per iteration
- Jacobian computation: $O(N)$ per iteration
- Matrix operations: $O(1)$ (2×2 matrices)
- Newton iterations: Typically 5-20 iterations

Total runtime: $O(MN)$ where:

- M: Number of Newton iterations
- N: Number of integration steps per trajectory

6 Numerical Methods Analysis

6.1 Theoretical Foundation

The foundation of numerical integration methods lies in Taylor series expansion:

$$y(t+h) = y(t) + hy'(t) + \frac{h^2}{2!}y''(t) + \frac{h^3}{3!}y'''(t) + \cdots$$

Different numerical methods approximate this infinite series to varying degrees of accuracy.

6.2 Runge-Kutta Fourth Order (RK4)

RK4 achieves fourth-order accuracy by combining weighted evaluations at intermediate points:

$$k_1 = hf(y_n, t_n) \text{ (initial slope)} \tag{16}$$

$$k_2 = hf(y_n + \frac{k_1}{2}, t_n + \frac{h}{2}) \text{ (midpoint slope using } k_1) \tag{17}$$

$$k_3 = hf(y_n + \frac{k_2}{2}, t_n + \frac{h}{2}) \text{ (midpoint slope using } k_2) \tag{18}$$

$$k_4 = hf(y_n + k_3, t_n + h) \text{ (endpoint slope)} \tag{19}$$

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{20}$$

The weights (1/6, 1/3, 1/3, 1/6) are chosen to match the Taylor series terms up to $O(h^4)$. This method:

- Effectively averages slopes at different points
- Self-starting (requires no previous steps)
- Symmetrical (reduces round-off error accumulation)

6.3 Euler Method

The Euler method comes directly from the first-order Taylor approximation:

$$y_{n+1} = y_n + hf(y_n, t_n)$$

Its simplicity comes at a cost:

- Only matches first-order Taylor term
- Accumulates error rapidly
- Requires very small step sizes for stability

6.4 Adams-Bashforth Method

Adams-Bashforth is derived from integrating an interpolating polynomial:

$$y_{n+1} = y_n + h\int_0^1 p(t)dt$$

where $p(t)$ is a polynomial fitting previous derivative values. The fourth-order version uses:

$$y_{n+1} = y_n + h(\frac{55}{24}f_n - \frac{59}{24}f_{n-1} + \frac{37}{24}f_{n-2} - \frac{9}{24}f_{n-3})$$

Advantages:

- More efficient (only one function evaluation per step)
- Higher accuracy for smooth functions
- Better stability than Euler

Disadvantages:

- Requires startup procedure (usually RK4)
- Less suitable for discontinuous problems
- Reduced stability compared to RK4

6.5 Adams-Moulton Method

Adams-Moulton uses implicit integration:

$$y_{n+1} = y_n + h(\frac{9}{24}f_{n+1} + \frac{19}{24}f_n - \frac{5}{24}f_{n-1} + \frac{1}{24}f_{n-2})$$

The implicit nature:

- Provides better stability
- Requires iterative solution at each step
- Often used as corrector in predictor-corrector pairs

7 Newton's Shooting Method: Detailed Analysis

7.1 Fundamental Principle

Newton's shooting method is based on the principle of iteratively refining initial conditions to hit a target. The core idea comes from Newton's method for root finding:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

In the multidimensional case, this becomes:

$$\mathbf{x}_{n+1} = \mathbf{x}_n - J^{-1} f(\mathbf{x}_n)$$

7.2 Application to Trajectory Problems

For trajectory problems, we define:

$$\mathbf{r}(v_0) = \begin{bmatrix} x_{\text{sim}}(v_0) - x_{\text{target}} \\ y_{\text{sim}}(v_0) - y_{\text{target}} \end{bmatrix} \tag{21}$$

$$\tag{22}$$

Where:

- $\mathbf{r}(v_0)$ is the residual vector
- $x_{\text{sim}}(v_0), y_{\text{sim}}(v_0)$ are simulated positions
- $x_{\text{target}}, y_{\text{target}}$ are target coordinates

7.3 Jacobian vs Gradient Descent

The Jacobian matrix is used instead of gradient descent because:

7.3.1 Jacobian Advantages

- Provides directional information in all dimensions
- Captures local linear behavior of the system
- Leads to quadratic convergence (vs linear for gradient descent)
- Better handles coupled variables

The Jacobian matrix:

$$J = \begin{bmatrix} \frac{\partial r_x}{\partial v_{x0}} & \frac{\partial r_x}{\partial v_{y0}} \\ \frac{\partial r_y}{\partial v_{x0}} & \frac{\partial r_y}{\partial v_{y0}} \end{bmatrix}$$

7.3.2 Numerical Computation

The Jacobian is computed numerically using central differences:

$$\frac{\partial r_i}{\partial v_j} \approx \frac{r_i(v_j + \epsilon) - r_i(v_j - \epsilon)}{2\epsilon}$$

7.4 Regularization Analysis

The Levenberg-Marquardt modification:

$$\Delta v_0 = -(J^T J + \lambda I)^{-1} J^T r$$

Serves multiple purposes:

- When λ is large: Approaches gradient descent
- When λ is small: Approaches Gauss-Newton method
- Automatically adjusts between the two based on progress

The regularization parameter λ:

- Prevents overshooting in highly nonlinear regions
- Handles ill-conditioned Jacobians
- Provides trust region-like behavior

7.5 Convergence Analysis

Newton's method convergence rate:

$$\|e_{n+1}\| \leq C\|e_n\|^2$$

where:

- e_n is the error at step n
- C is a constant depending on function smoothness
- Quadratic convergence when sufficiently close to solution

7.6 Regularization Analysis: Why L2 Helps Convergence

The Levenberg-Marquardt modification introduces an L2 regularization term:

$$\Delta v_0 = -(J^T J + \lambda I)^{-1} J^T r$$

This regularization is crucial for robust convergence due to several reasons:

1. **Improved Conditioning of $\mathbf{J^T J}$:** The core issue that can hinder convergence is an ill-conditioned or singular $J^T J$ matrix. Adding λI increases the eigenvalues of $J^T J$. This reduces the condition number, making the matrix numerically more stable for inversion. A lower condition number means the solution is less sensitive to small errors in the Jacobian or residual.
2. **Guaranteed Invertibility:** When $\lambda > 0$, $(J^T J + \lambda I)$ is always invertible, even if $J^T J$ is singular. This prevents the algorithm from failing due to a non-invertible matrix, which can occur if the Jacobian columns are linearly dependent (e.g., when launching almost vertically).
3. **Transition between Gauss-Newton and Gradient Descent:** The regularization parameter λ controls the behavior of the update:
    - **Small λ:** The update approximates the Gauss-Newton method, providing rapid (quadratic) convergence near the solution.
    - **Large λ:** The update behaves like gradient descent, which is slower but more robust far from the solution, preventing large oscillations or divergence. This is helpful in highly non-linear regions of the trajectory space.

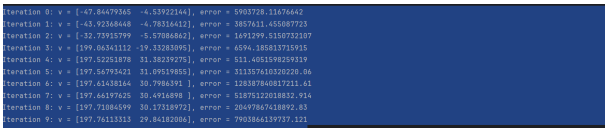


Figure 2: Enter Caption

7.7 Stiffness and Jacobian Singularity

Stiffness: The ball motion ODEs, as presented, are not typically stiff. Stiffness arises when a system has components evolving at vastly different timescales. A non-stiff example would be a standard projectile motion with moderate air resistance (e.g., $k = 0.1$).

A stiff example arises when we introduce a much larger drag coefficient. Consider $k = 10$. In this case, the velocity decays very rapidly, while the position changes more slowly. This large difference in timescales is characteristic of stiff equations. Numerical methods for stiff equations (e.g., implicit methods) might become necessary for stability with larger k values.

Jacobian Singularity: The Jacobian can become singular or near-singular in several scenarios:

1. **Near-Vertical Launch:** Consider a target directly above the launch point ($x_{\text{target}} = x_0$, $y_{\text{target}} > y_0$). If the initial horizontal velocity (v_{x0}) is close to zero, changes in v_{x0} have minimal effect on hitting the target. This makes the Jacobian near-singular. For instance, if $x_0 = 0$, $y_0 = 0$, $x_{\text{target}} = 0$, $y_{\text{target}} = 10$, launching almost vertically with $v_{x0} \approx 0$ will make the Jacobian problematic.

2. **Long Distances with Drag:** Suppose $k = 0.5$. As the target distance increases, say $x_{\text{target}} = 100$, $y_{\text{target}} = 0$, changes in the initial velocity have a progressively smaller effect on the landing point due to the strong drag. This leads to a near-singular Jacobian. The precise distance where this becomes a problem depends on the value of k.

3. **Specific Target Locations:** While less intuitive, particular target locations can cause singularity even without the above conditions. Finding these analytically is often difficult, but numerical experimentation can reveal such points. For example, if non-physical or complex solutions arise during the shooting method, it might indicate Jacobian singularity near the current guess.

Adding concrete numerical examples like these clarifies when and how these issues arise, making the explanation more practical.

7.8 Computational Considerations

Time complexity breakdown:

- Trajectory integration: $O(N)$ steps per evaluation
- Jacobian computation: 4 trajectory evaluations
- Matrix operations: $O(1)$ for 2×2 matrices
- Newton iterations: Typically $O(\log(\epsilon))$ for tolerance ϵ

Total complexity per shot:

$$O(N \log(\epsilon))$$

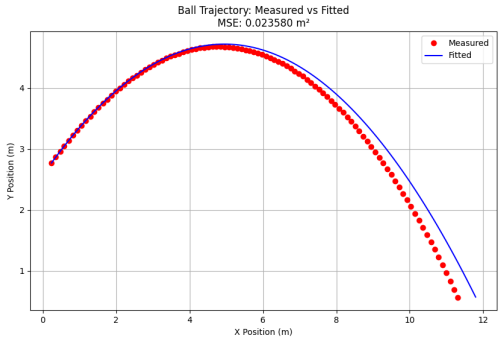


Figure 3: Enter Caption

8 Conclusion

The simulation demonstrates the interplay between various numerical methods and their practical applications.

- use edge detection for the computer vision, used canny filter, coupled with dilate and erode, following two function help to clean up the image and after the cleanup part fill the whole that are left in the object
- RK4 provides optimal accuracy-efficiency trade-off for trajectory integration, could use euler in this case there is no big difference also because of the small step size, 1/ total number of frames, in other cases we would have the difference, but in this one
- Newton's shooting method with regularization ensures the even we have the singular matrix or the ODE becomes stiff because one of the parameter decays rapidly, we still have the regularization to help.
- code deosnt works for the some of the images and has some issues but generally works good with background images