Inverted Pendulum Simulation in Swift

Conversation by *Brian Abraham DiZio*

Last updated: 2025-01-07

User:

Hi Claude, can you write me some Swift code for an existing Xcode project I have that simulates a driven inverted pendulum according to attached Matlab simulation, 'myiptype8.m', summary of a pendulum, and that uses the Runge-Kutta ODE solver in the attached file, 'NumericalODESolvers? Further, can you implement a call to the simulation in the attached file, 'GameSceneMainMaze.swift', adding visualization of the driven pendulum, and the ability to balance it incrementally with buttons? Lastly, can you import that data from the input file, where these are the conditions for the simulation, and the parameters are 'InputPendulumSim.csv'? The input to the integrator are th0(1)=IPPos, th0(2)= IPVel. The output of the integrator is th(1)=IPPos, th(2)=IPVel. At t=0 (initial conditions of simulation) th0(1)=.05, th0(2)=0. At each iteration; th0(1)t= th(1)t-1, th0(2)t= th(2)t-1 + Command*kj/tinc. The first row is mass, the second is point mass, the third is pendulum radius, the fourth is gravitational constant, the fifth is stiffness, the sixth is damping, the seventh is integration interval, the eighth is fall boundary in degrees at 90 and the ninth is fall boundary at 90 deg increments. the The output of the simulation that I'd like to test the Swift code against is in OutputPendulumSim.csv, with the first column time samples, the second column pendulum positions, and the third column pendulum velocities.

Claude:

I'll help you create a Swift implementation of the inverted pendulum simulation. Let's break this down into steps:

1. First, let's create a struct to handle the pendulum simulation using the Runge-Kutta solver

Inverted Pendulum Simulation

```swift
import Foundation

struct InvertedPendulum {
    // System parameters
    let ma: Double      // Mass
    let Iz: Double      // Moment of inertia
    let ln: Double      // Length
    let gr: Double      // Gravity
    let ksp: Double     // Spring constant
    let bv: Double      // Damping coefficient
    let kj: Double      // Joy constant
    let tinc: Double    // Time increment
    let fallBoundary: Double // Fall boundary in radians
```

```
    // State variables
    private var position: Double
    private var velocity: Double

    init(parameters: [Double], initialPosition: Double = 0.05,
initialVelocity: Double = 0.0) {
        self.ma = parameters[0]
        self.Iz = parameters[1]
        self.ln = parameters[2]
        self.gr = parameters[3]
        self.ksp = parameters[4]
        self.bv = parameters[5]
        self.tinc = parameters[6]
        self.fallBoundary = parameters[7].degreesToRadians

        self.position = initialPosition
        self.velocity = initialVelocity
    }

    // Convert the system into first-order differential equations
    private func derivatives(_ t: Double, _ state: [Double], command:
Double) -&gt; [(Double, [Double]) -&gt; Double] {
        return [
            // θ' = velocity
            { (_, _) in state[1] },

            // θ'' = ka*sin(θ) - ks*θ - kb*θ' + kj*command
            { (_, _) in
                let ka = (ma * ln * gr) / (ma * ln * ln + Iz)
                let ks = ksp / (ma * ln * ln + Iz)
                let kb = bv / (ma * ln * ln + Iz)

                return ka * sin(state[0]) - ks * state[0] - kb *
state[1] + kj * command
            }
        ]
    }

    mutating func step(command: Double) -&gt; (position: Double,
velocity: Double) {
        let currentState = [position, velocity]

        // Use RK4 to integrate one time step
        let nextState = rK4(tinc, 0.0, currentState, functions:
derivatives(0.0, currentState, command: command))

        // Update state
        position = nextState[0]
        velocity = nextState[1]

        // Check fall boundary
```

```
            if abs(position) &gt; fallBoundary {
                position = position &gt; 0 ? fallBoundary : -fallBoundary
                velocity = 0
            }

            return (position, velocity)
        }
    }

    // Helper extension
    extension Double {
        var degreesToRadians: Double {
            return self * .pi / 180.0
        }

        var radiansToDegrees: Double {
            return self * 180.0 / .pi
        }
    }
```

Now let's create a visualization component for the SpriteKit scene:

Pendulum SpriteKit Node

```
    import SpriteKit

    class PendulumNode: SKNode {
        private let rodNode: SKShapeNode
        private let massNode: SKShapeNode
        private let length: CGFloat

        init(length: CGFloat, massRadius: CGFloat) {
            self.length = length

            // Create rod
            rodNode = SKShapeNode(rectOf: CGSize(width: 4, height: length))
            rodNode.fillColor = .white
            rodNode.strokeColor = .white
            rodNode.position = CGPoint(x: 0, y: -length/2)

            // Create mass
            massNode = SKShapeNode(circleOfRadius: massRadius)
            massNode.fillColor = .red
            massNode.strokeColor = .red
            massNode.position = CGPoint(x: 0, y: -length)

            super.init()

            addChild(rodNode)
            addChild(massNode)
```

```
    }

    required init?(coder aDecoder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }

    func updateAngle(_ angle: Double) {
        zRotation = CGFloat(angle)
    }
}
```

Now let's add the implementation to GameScene:

Game Scene Implementation

```
import SpriteKit

class GameScene: SKScene {
    private var pendulum: InvertedPendulum?
    private var pendulumNode: PendulumNode?
    private var leftButton: SKShapeNode?
    private var rightButton: SKShapeNode?

    private var lastUpdateTime: TimeInterval = 0
    private var command: Double = 0

    override func didMove(to view: SKView) {
        setupSimulation()
        setupPendulum()
        setupControls()
    }

    private func setupSimulation() {
        // Load parameters from CSV
        if let path = Bundle.main.path(forResource: "InputPendulumSim",
ofType: "csv") {
            do {
                let content = try String(contentsOfFile: path, encoding:
.utf8)
                let rows = content.components(separatedBy: .newlines)
                let parameters = rows.compactMap { row in
                    let components = row.components(separatedBy: ",")
                    return Double(components[0])
                }

                pendulum = InvertedPendulum(parameters: parameters)
            } catch {
                print("Error loading simulation parameters: \(error)")
            }
        }
```

```
    }

    private func setupPendulum() {
        let screenHeight = frame.size.height
        let pendulumLength: CGFloat = screenHeight * 0.3
        let massRadius: CGFloat = 20

        pendulumNode = PendulumNode(length: pendulumLength, massRadius:
massRadius)
        if let pendulumNode = pendulumNode {
            pendulumNode.position = CGPoint(x: frame.midX, y: frame.midY
+ pendulumLength/2)
            addChild(pendulumNode)
        }
    }

    private func setupControls() {
        let buttonSize = CGSize(width: 100, height: 50)
        let buttonY = frame.minY + 100

        // Left button
        leftButton = SKShapeNode(rectOf: buttonSize)
        if let leftButton = leftButton {
            leftButton.position = CGPoint(x: frame.midX - 120, y:
buttonY)
            leftButton.fillColor = .blue
            leftButton.name = "leftButton"
            addChild(leftButton)

            let leftLabel = SKLabelNode(text: "←")
            leftLabel.fontSize = 24
            leftLabel.position = CGPoint(x: 0, y: -8)
            leftButton.addChild(leftLabel)
        }

        // Right button
        rightButton = SKShapeNode(rectOf: buttonSize)
        if let rightButton = rightButton {
            rightButton.position = CGPoint(x: frame.midX + 120, y:
buttonY)
            rightButton.fillColor = .blue
            rightButton.name = "rightButton"
            addChild(rightButton)

            let rightLabel = SKLabelNode(text: "→")
            rightLabel.fontSize = 24
            rightLabel.position = CGPoint(x: 0, y: -8)
            rightButton.addChild(rightLabel)
        }
    }
```

```
    override func touchesBegan(_ touches: Set<uitouch>, with event:
UIEvent?) {
        for touch in touches {
            let location = touch.location(in: self)
            let touchedNode = nodes(at: location).first

            switch touchedNode?.name {
            case "leftButton":
                command = -1.0
            case "rightButton":
                command = 1.0
            default:
                break
            }
        }
    }

    override func touchesEnded(_ touches: Set<uitouch>, with event:
UIEvent?) {
        command = 0
    }

    override func update(_ currentTime: TimeInterval) {
        if lastUpdateTime == 0 {
            lastUpdateTime = currentTime
            return
        }

        guard let pendulum = pendulum else { return }

        // Step simulation
        let (position, _) = pendulum.step(command: command)

        // Update visualization
        pendulumNode?.updateAngle(position)
    }
}
```

This implementation provides:

1. A complete inverted pendulum simulation using the RK4 solver
2. Loading of parameters from the CSV file
3. Visual representation of the pendulum using SpriteKit
4. Interactive control using left/right buttons
5. The same mathematical model as the MATLAB implementation

To use this:

1. Add the code files to your Xcode project
2. Make sure InputPendulumSim.csv is included in your project bundle
3. Update your GameScene scene file to use the new GameScene class
4. Build and run

The simulation will load the parameters from the CSV file, create a visual pendulum that follows the physics simulation, and allow control via the buttons. The pendulum will fall if it exceeds the fall boundary angle.

Would you like me to explain any part of the implementation in more detail or help with testing against the output data in OutputPendulumSim.csv?

User:

Can you help with testing output against the data in OutputPendulumSim.csv? I am using the following input file (InputPendulumSim.csv), and the code compiles, but the pendulum does not move when I click on the buttons.

Claude:

I'll help you test and debug the implementation. First, let's write code to verify the input parameters and add testing against OutputPendulumSim.csv.

Pendulum Testing Utilities

```
import Foundation

class PendulumTester {
    static func validateSimulation() {
        // Load and parse input parameters
        guard let inputPath = Bundle.main.path(forResource:
"InputPendulumSim", ofType: "csv") else {
            print("Error: Could not find InputPendulumSim.csv")
            return
        }

        do {
            let inputContent = try String(contentsOfFile: inputPath,
encoding: .utf8)
            let rows = inputContent.components(separatedBy: .newlines)
                                   .filter { !$0.isEmpty }

            // Parse input parameters
            let parameters = rows.compactMap { row -&gt; Double? in
                let components = row.components(separatedBy: ",")
                guard let firstValue = components.first else { return
nil }
                return Double(firstValue)
            }

            print("Loaded parameters:", parameters)

            // Create pendulum simulation
            var pendulum = InvertedPendulum(parameters: parameters)

            // Load output data for comparison
```

```
            guard let outputPath = Bundle.main.path(forResource:
"OutputPendulumSim", ofType: "csv") else {
                print("Error: Could not find OutputPendulumSim.csv")
                return
            }

            let outputContent = try String(contentsOfFile: outputPath,
encoding: .utf8)
            let outputRows = outputContent.components(separatedBy:
.newlines)
                                          .filter { !$0.isEmpty }
                                          .dropFirst() // Skip header

            // Compare simulation results with reference data
            var maxError = 0.0
            var totalError = 0.0
            var sampleCount = 0

            for row in outputRows {
                let values = row.components(separatedBy: ",")
                guard values.count &gt;= 4,
                      let time = Double(values[0]),
                      let refPosition = Double(values[1]),
                      let refVelocity = Double(values[2]),
                      let command = Double(values[3]) else {
                    continue
                }

                // Run simulation step
                let (simPosition, simVelocity) = pendulum.step(command:
command)

                // Calculate error
                let posError = abs(simPosition - refPosition)
                let velError = abs(simVelocity - refVelocity)

                maxError = max(maxError, max(posError, velError))
                totalError += posError + velError
                sampleCount += 1

                // Print periodic comparison
                if sampleCount % 50 == 0 {
                    print("Time: \(time)")
                    print("Reference - Pos: \(refPosition), Vel: \
(refVelocity)")
                    print("Simulation - Pos: \(simPosition), Vel: \
(simVelocity)")
                    print("Error - Pos: \(posError), Vel: \(velError)")
                    print("---")
                }
            }
```

```
            print("Validation complete:")
            print("Max error: \(maxError)")
            print("Average error: \(totalError / Double(sampleCount *
2))")

        } catch {
            print("Error reading files: \(error)")
        }
    }
}
```

Now let's modify the GameScene to add debugging and fix the button control issues:

Game Scene Implementation

```
import SpriteKit

class GameScene: SKScene {
    private var pendulum: InvertedPendulum?
    private var pendulumNode: PendulumNode?
    private var leftButton: SKShapeNode?
    private var rightButton: SKShapeNode?
    private var debugLabel: SKLabelNode?

    private var lastUpdateTime: TimeInterval = 0
    private var command: Double = 0

    override func didMove(to view: SKView) {
        backgroundColor = .black
        setupSimulation()
        setupPendulum()
        setupControls()
        setupDebugLabel()

        // Run validation on startup
        PendulumTester.validateSimulation()
    }

    private func setupDebugLabel() {
        debugLabel = SKLabelNode(text: "State: Initializing")
        if let debugLabel = debugLabel {
            debugLabel.fontSize = 14
            debugLabel.fontColor = .white
            debugLabel.position = CGPoint(x: frame.midX, y: frame.maxY -
50)
            addChild(debugLabel)
        }
    }
```

```
    private func setupSimulation() {
        guard let path = Bundle.main.path(forResource:
"InputPendulumSim", ofType: "csv") else {
            print("Error: Could not find input file")
            return
        }

        do {
            let content = try String(contentsOfFile: path, encoding:
.utf8)
            let rows = content.components(separatedBy: .newlines)
                        .filter { !$0.isEmpty }

            let parameters = rows.compactMap { row -&gt; Double? in
                let components = row.components(separatedBy: ",")
                guard let firstValue = components.first else { return
nil }
                return Double(firstValue)
            }

            print("Loaded parameters:", parameters)
            pendulum = InvertedPendulum(parameters: parameters)
            debugLabel?.text = "State: Simulation initialized"

        } catch {
            print("Error loading simulation parameters: \(error)")
            debugLabel?.text = "State: Error loading parameters"
        }
    }

    private func setupPendulum() {
        let screenHeight = frame.size.height
        let pendulumLength: CGFloat = screenHeight * 0.3
        let massRadius: CGFloat = 20

        pendulumNode = PendulumNode(length: pendulumLength, massRadius:
massRadius)
        if let pendulumNode = pendulumNode {
            pendulumNode.position = CGPoint(x: frame.midX, y: frame.midY
+ pendulumLength/2)
            addChild(pendulumNode)
        }
    }

    private func setupControls() {
        let buttonSize = CGSize(width: 100, height: 50)
        let buttonY = frame.minY + 100

        // Left button
        leftButton = SKShapeNode(rectOf: buttonSize)
        if let leftButton = leftButton {
```

```
            leftButton.position = CGPoint(x: frame.midX - 120, y:
buttonY)
            leftButton.fillColor = .blue
            leftButton.strokeColor = .white
            leftButton.name = "leftButton"
            leftButton.zPosition = 100  // Ensure buttons are above
other nodes
            addChild(leftButton)

            let leftLabel = SKLabelNode(text: "←")
            leftLabel.fontSize = 24
            leftLabel.fontColor = .white
            leftLabel.position = CGPoint(x: 0, y: -8)
            leftButton.addChild(leftLabel)
        }

        // Right button
        rightButton = SKShapeNode(rectOf: buttonSize)
        if let rightButton = rightButton {
            rightButton.position = CGPoint(x: frame.midX + 120, y:
buttonY)
            rightButton.fillColor = .blue
            rightButton.strokeColor = .white
            rightButton.name = "rightButton"
            rightButton.zPosition = 100  // Ensure buttons are above
other nodes
            addChild(rightButton)

            let rightLabel = SKLabelNode(text: "→")
            rightLabel.fontSize = 24
            rightLabel.fontColor = .white
            rightLabel.position = CGPoint(x: 0, y: -8)
            rightButton.addChild(rightLabel)
        }
    }

    override func touchesBegan(_ touches: Set<uitouch>, with event:
UIEvent?) {
        for touch in touches {
            let location = touch.location(in: self)
            let touchedNodes = nodes(at: location)

            for node in touchedNodes {
                switch node.name {
                case "leftButton":
                    command = -1.0
                    debugLabel?.text = "Command: Left"
                    print("Left button pressed")
                    return

                case "rightButton":
```

```
                    command = 1.0
                    debugLabel?.text = "Command: Right"
                    print("Right button pressed")
                    return

                default:
                    break
                }
            }
        }
    }

    override func touchesEnded(_ touches: Set<uitouch>, with event:
UIEvent?) {
        command = 0
        debugLabel?.text = "Command: None"
        print("Touch ended")
    }

    override func update(_ currentTime: TimeInterval) {
        guard let pendulum = pendulum else { return }

        if lastUpdateTime == 0 {
            lastUpdateTime = currentTime
            return
        }

        // Step simulation
        let (position, velocity) = pendulum.step(command: command)

        // Update visualization
        pendulumNode?.updateAngle(position)

        // Update debug info
        debugLabel?.text = String(format: "Pos: %.3f, Vel: %.3f, Cmd:
%.1f",
                                  position, velocity, command)

        lastUpdateTime = currentTime
    }
}
```

The changes include:

1. Added a debug label to show the current state of the simulation
2. Added print statements to verify button presses
3. Added validation testing against OutputPendulumSim.csv
4. Improved button visibility and interaction
5. Added error checking for parameter loading
6. Added detailed state logging

To test this:

1. Add the new PendulumTester class to your project
2. Replace your existing GameScene implementation
3. Make sure both CSV files are included in your project bundle
4. Run the app

The debug label will show you:

- If the simulation initialized properly
- Current pendulum position and velocity
- Current command value

If the buttons still don't respond:

1. Check if you see the print statements in the console when pressing buttons
2. Verify the buttons are visible and their positions are correct
3. Verify the CSV files are properly loaded (check console output)
4. Make sure the pendulum parameters are within reasonable ranges

Let me know what you observe with these changes and I can help further debug any issues.

Would you also like me to add visualization of the pendulum's phase space (position vs. velocity) to help debug the dynamics?

User:

Can you give some example pendulum parameters that will allow me to play with the joystick, trying to balance it after first moving it?

Claude:

I'll help you set up parameters that will create an interesting balancing challenge. Looking at the physics from the lab document and MATLAB code, we need parameters that create an unstable equilibrium that's controllable with input.

Here's a set of reasonable parameters for the InputPendulumSim.csv file:

```
70.0        # ma (mass in kg)
2.0         # Iz (moment of inertia around CG)
1.0         # ln (length in meters)
9.81        # gr (gravitational acceleration in m/s^2)
700.0       # ksp (torsional spring constant in N·m/rad)
5.0         # bv (viscous damping coefficient)
0.01        # tinc (integration time step in seconds)
90.0        # fall boundary in degrees
```

With these parameters:

- The mass (70 kg) and length (1 m) are chosen to give reasonable inertia
- The spring constant (700 N·m/rad) is strong enough to help stabilize but not too stiff
- The damping (5.0) provides enough friction to make it controllable

- The time step (0.01s) gives smooth simulation
- The 90-degree fall boundary lets you recover from fairly large angles

To use these:

1. The pendulum will start slightly offset from vertical (0.05 radians)
2. It will begin to fall due to gravity
3. Use the left/right buttons to apply corrective forces
4. Try to keep it balanced near vertical
5. If it falls beyond 90 degrees, it will stop at the boundary

The key to balancing is to:

1. Make small, quick corrections
2. Counter any developing angular velocity early
3. Try to keep the pendulum within ±30 degrees where control is easier

Would you like me to add visualization of the control input magnitude or additional feedback to help with the balancing task?

User:

Can you tell me where in the code I could set an initial position of 0.05 radians and 0 velocity? Then it would match the simulation output in OutputPendulumSim.csv with the previous inputs.

Claude:

Looking at the current code, we can modify the InvertedPendulum initializer where the initial conditions are set. Let's update that part:

Inverted Pendulum Simulation

```
import Foundation

struct InvertedPendulum {
    // System parameters
    let ma: Double      // Mass
    let Iz: Double      // Moment of inertia
    let ln: Double      // Length
    let gr: Double      // Gravity
    let ksp: Double     // Spring constant
    let bv: Double      // Damping coefficient
    let kj: Double      // Joy constant
    let tinc: Double    // Time increment
    let fallBoundary: Double // Fall boundary in radians

    // State variables
    private var position: Double
    private var velocity: Double

    init(parameters: [Double]) {
        self.ma = parameters[0]
```

```
        self.Iz = parameters[1]
        self.ln = parameters[2]
        self.gr = parameters[3]
        self.ksp = parameters[4]
        self.bv = parameters[5]
        self.tinc = parameters[6]
        self.fallBoundary = parameters[7].degreesToRadians

        self.position = initialPosition
        self.velocity = initialVelocity
    }

    // Convert the system into first-order differential equations
    private func derivatives(_ t: Double, _ state: [Double], command:
Double) -&gt; [(Double, [Double]) -&gt; Double] {
        return [
            // θ' = velocity
            { (_, _) in state[1] },

            // θ'' = ka*sin(θ) - ks*θ - kb*θ' + kj*command
            { (_, _) in
                let ka = (ma * ln * gr) / (ma * ln * ln + Iz)
                let ks = ksp / (ma * ln * ln + Iz)
                let kb = bv / (ma * ln * ln + Iz)

                return ka * sin(state[0]) - ks * state[0] - kb *
state[1] + kj * command
            }
        ]
    }

    mutating func step(command: Double) -&gt; (position: Double,
velocity: Double) {
        let currentState = [position, velocity]

        // Use RK4 to integrate one time step
        let nextState = rK4(tinc, 0.0, currentState, functions:
derivatives(0.0, currentState, command: command))

        // Update state
        position = nextState[0]
        velocity = nextState[1]

        // Check fall boundary
        if abs(position) &gt; fallBoundary {
            position = position &gt; 0 ? fallBoundary : -fallBoundary
            velocity = 0
        }

        return (position, velocity)
    }
```

```
}

// Helper extension
extension Double {
    var degreesToRadians: Double {
        return self * .pi / 180.0
    }

    var radiansToDegrees: Double {
        return self * 180.0 / .pi
    }
}
```

Inverted Pendulum Simulation

```
import Foundation

struct InvertedPendulum {
    // System parameters
    let ma: Double      // Mass
    let Iz: Double      // Moment of inertia
    let ln: Double      // Length
    let gr: Double      // Gravity
    let ksp: Double     // Spring constant
    let bv: Double      // Damping coefficient
    let kj: Double      // Joy constant
    let tinc: Double    // Time increment
    let fallBoundary: Double // Fall boundary in radians

    // State variables
    private var position: Double
    private var velocity: Double

    init(parameters: [Double]) {
        self.ma = parameters[0]
        self.Iz = parameters[1]
        self.ln = parameters[2]
        self.gr = parameters[3]
        self.ksp = parameters[4]
        self.bv = parameters[5]
        self.tinc = parameters[6]
        self.fallBoundary = parameters[7].degreesToRadians

        // Set initial conditions to match MATLAB simulation
        self.position = 0.05  // Initial position of 0.05 radians
        self.velocity = 0.0   // Initial velocity of 0 rad/s
    }

    // Convert the system into first-order differential equations
    private func derivatives(_ t: Double, _ state: [Double], command:
```

```
Double) -&gt; [(Double, [Double]) -&gt; Double] {
        return [
            // θ' = velocity
            { (_, _) in state[1] },

            // θ'' = ka*sin(θ) - ks*θ - kb*θ' + kj*command
            { (_, _) in
                let ka = (ma * ln * gr) / (ma * ln * ln + Iz)
                let ks = ksp / (ma * ln * ln + Iz)
                let kb = bv / (ma * ln * ln + Iz)

                return ka * sin(state[0]) - ks * state[0] - kb *
state[1] + kj * command
            }
        ]
    }

    mutating func step(command: Double) -&gt; (position: Double,
velocity: Double) {
        let currentState = [position, velocity]

        // Use RK4 to integrate one time step
        let nextState = rK4(tinc, 0.0, currentState, functions:
derivatives(0.0, currentState, command: command))

        // Update state
        position = nextState[0]
        velocity = nextState[1]

        // Check fall boundary
        if abs(position) &gt; fallBoundary {
            position = position &gt; 0 ? fallBoundary : -fallBoundary
            velocity = 0
        }

        return (position, velocity)
    }
}

// Helper extension
extension Double {
    var degreesToRadians: Double {
        return self * .pi / 180.0
    }

    var radiansToDegrees: Double {
        return self * 180.0 / .pi
    }
}
```

Now when the pendulum is initialized in GameScene, it will automatically start at 0.05 radians (about 2.86 degrees) with zero velocity, matching the initial conditions in the MATLAB simulation.

This means you don't need to specify these values when creating the pendulum in GameScene - they're now built into the InvertedPendulum struct. The simulation should now match the OutputPendulumSim.csv more closely since it starts from the same initial conditions.

Would you like me to add some code to verify that the initial conditions are being set correctly? We could add debug output or visual indicators to confirm the starting position.

User:

Your computations were correct -- the simulation matched the Swift code. But, the pendulum did not move, is there a reason for that?

Claude:

Let me help debug why the pendulum isn't moving. There could be a few reasons:

1. First, let's check if the time step integration is actually happening. Let's add some debug prints in the step function:

Inverted Pendulum Simulation

```swift
import Foundation

struct InvertedPendulum {
    // System parameters
    let ma: Double      // Mass
    let Iz: Double      // Moment of inertia
    let ln: Double      // Length
    let gr: Double      // Gravity
    let ksp: Double     // Spring constant
    let bv: Double      // Damping coefficient
    let kj: Double      // Joy constant
    let tinc: Double    // Time increment
    let fallBoundary: Double // Fall boundary in radians

    // State variables
    private var position: Double
    private var velocity: Double

    init(parameters: [Double]) {
        self.ma = parameters[0]
        self.Iz = parameters[1]
        self.ln = parameters[2]
        self.gr = parameters[3]
        self.ksp = parameters[4]
        self.bv = parameters[5]
        self.tinc = parameters[6]
        self.fallBoundary = parameters[7].degreesToRadians
```

```
        // Set initial conditions to match MATLAB simulation
        self.position = 0.05  // Initial position of 0.05 radians
        self.velocity = 0.0   // Initial velocity of 0 rad/s
    }

    // Convert the system into first-order differential equations
    private func derivatives(_ t: Double, _ state: [Double], command:
Double) -&gt; [(Double, [Double]) -&gt; Double] {
        return [
            // θ' = velocity
            { (_, _) in state[1] },

            // θ'' = ka*sin(θ) - ks*θ - kb*θ' + kj*command
            { (_, _) in
                let ka = (ma * ln * gr) / (ma * ln * ln + Iz)
                let ks = ksp / (ma * ln * ln + Iz)
                let kb = bv / (ma * ln * ln + Iz)

                return ka * sin(state[0]) - ks * state[0] - kb *
state[1] + kj * command
            }
        ]
    }

    mutating func step(command: Double) -&gt; (position: Double,
velocity: Double) {
        print("Step called - Current state - Position: \(position),
Velocity: \(velocity)")
        let currentState = [position, velocity]

        // Use RK4 to integrate one time step
        let nextState = rK4(tinc, 0.0, currentState, functions:
derivatives(0.0, currentState, command: command))

        // Update state
        position = nextState[0]
        velocity = nextState[1]

        // Check fall boundary
        if abs(position) &gt; fallBoundary {
            position = position &gt; 0 ? fallBoundary : -fallBoundary
            velocity = 0
        }

        return (position, velocity)
    }
}

// Helper extension
extension Double {
    var degreesToRadians: Double {
```

```
        return self * .pi / 180.0
    }

    var radiansToDegrees: Double {
        return self * 180.0 / .pi
    }
}
```

2. Next, let's modify the GameScene update function to ensure it's being called at the correct frequency:

Game Scene Implementation

```
import SpriteKit

class GameScene: SKScene {
    private var pendulum: InvertedPendulum?
    private var pendulumNode: PendulumNode?
    private var leftButton: SKShapeNode?
    private var rightButton: SKShapeNode?
    private var debugLabel: SKLabelNode?

    private var lastUpdateTime: TimeInterval = 0
    private var command: Double = 0

    override func didMove(to view: SKView) {
        backgroundColor = .black
        setupSimulation()
        setupPendulum()
        setupControls()
        setupDebugLabel()

        // Run validation on startup
        PendulumTester.validateSimulation()
    }

    private func setupDebugLabel() {
        debugLabel = SKLabelNode(text: "State: Initializing")
        if let debugLabel = debugLabel {
            debugLabel.fontSize = 14
            debugLabel.fontColor = .white
            debugLabel.position = CGPoint(x: frame.midX, y: frame.maxY -
50)
            addChild(debugLabel)
        }
    }

    private func setupSimulation() {
        guard let path = Bundle.main.path(forResource:
"InputPendulumSim", ofType: "csv") else {
```

```
            print("Error: Could not find input file")
            return
        }

        do {
            let content = try String(contentsOfFile: path, encoding:
.utf8)
            let rows = content.components(separatedBy: .newlines)
                        .filter { !$0.isEmpty }

            let parameters = rows.compactMap { row -&gt; Double? in
                let components = row.components(separatedBy: ",")
                guard let firstValue = components.first else { return
nil }
                return Double(firstValue)
            }

            print("Loaded parameters:", parameters)
            pendulum = InvertedPendulum(parameters: parameters)
            debugLabel?.text = "State: Simulation initialized"

        } catch {
            print("Error loading simulation parameters: \(error)")
            debugLabel?.text = "State: Error loading parameters"
        }
    }

    private func setupPendulum() {
        let screenHeight = frame.size.height
        let pendulumLength: CGFloat = screenHeight * 0.3
        let massRadius: CGFloat = 20

        pendulumNode = PendulumNode(length: pendulumLength, massRadius:
massRadius)
        if let pendulumNode = pendulumNode {
            pendulumNode.position = CGPoint(x: frame.midX, y: frame.midY
+ pendulumLength/2)
            addChild(pendulumNode)
        }
    }

    private func setupControls() {
        let buttonSize = CGSize(width: 100, height: 50)
        let buttonY = frame.minY + 100

        // Left button
        leftButton = SKShapeNode(rectOf: buttonSize)
        if let leftButton = leftButton {
            leftButton.position = CGPoint(x: frame.midX - 120, y:
buttonY)
            leftButton.fillColor = .blue
```

```
            leftButton.strokeColor = .white
            leftButton.name = "leftButton"
            leftButton.zPosition = 100  // Ensure buttons are above
other nodes
            addChild(leftButton)

            let leftLabel = SKLabelNode(text: "←")
            leftLabel.fontSize = 24
            leftLabel.fontColor = .white
            leftLabel.position = CGPoint(x: 0, y: -8)
            leftButton.addChild(leftLabel)
        }

        // Right button
        rightButton = SKShapeNode(rectOf: buttonSize)
        if let rightButton = rightButton {
            rightButton.position = CGPoint(x: frame.midX + 120, y:
buttonY)
            rightButton.fillColor = .blue
            rightButton.strokeColor = .white
            rightButton.name = "rightButton"
            rightButton.zPosition = 100  // Ensure buttons are above
other nodes
            addChild(rightButton)

            let rightLabel = SKLabelNode(text: "→")
            rightLabel.fontSize = 24
            rightLabel.fontColor = .white
            rightLabel.position = CGPoint(x: 0, y: -8)
            rightButton.addChild(rightLabel)
        }
    }

    override func touchesBegan(_ touches: Set<uitouch>, with event:
UIEvent?) {
        for touch in touches {
            let location = touch.location(in: self)
            let touchedNodes = nodes(at: location)

            for node in touchedNodes {
                switch node.name {
                case "leftButton":
                    command = -1.0
                    debugLabel?.text = "Command: Left"
                    print("Left button pressed")
                    return

                case "rightButton":
                    command = 1.0
                    debugLabel?.text = "Command: Right"
                    print("Right button pressed")
```

```
                    return

                default:
                    break
                }
            }
        }
    }

    override func touchesEnded(_ touches: Set<uitouch>, with event:
UIEvent?) {
        command = 0
        debugLabel?.text = "Command: None"
        print("Touch ended")
    }

    override func update(_ currentTime: TimeInterval) {
        guard let pendulum = pendulum else {
            print("Pendulum not initialized")
            return
        }

        if lastUpdateTime == 0 {
            lastUpdateTime = currentTime
            return
        }

        let dt = currentTime - lastUpdateTime
        print("Update called - dt: \(dt)")

        // Step simulation
        let (position, velocity) = pendulum.step(command: command)

        // Update visualization
        pendulumNode?.updateAngle(position)

        // Update debug info
        debugLabel?.text = String(format: "Pos: %.3f, Vel: %.3f, Cmd:
%.1f",
                                  position, velocity, command)

        lastUpdateTime = currentTime
    }
}
```

3. The pendulum visualization might not be updating correctly. Let's modify PendulumNode:

Pendulum SpriteKit Node

```
import SpriteKit

class PendulumNode: SKNode {
    private let rodNode: SKShapeNode
    private let massNode: SKShapeNode
    private let length: CGFloat

    init(length: CGFloat, massRadius: CGFloat) {
        self.length = length

        // Create rod
        rodNode = SKShapeNode(rectOf: CGSize(width: 4, height: length))
        rodNode.fillColor = .white
        rodNode.strokeColor = .white
        rodNode.position = CGPoint(x: 0, y: -length/2)

        // Create mass
        massNode = SKShapeNode(circleOfRadius: massRadius)
        massNode.fillColor = .red
        massNode.strokeColor = .red
        massNode.position = CGPoint(x: 0, y: -length)

        super.init()

        addChild(rodNode)
        addChild(massNode)
    }

    required init?(coder aDecoder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }

    func updateAngle(_ angle: Double) {
        zRotation = CGFloat(angle)
        print("Updating pendulum visual angle to: \(angle)")
    }
}
```

4. The most likely issue might be that we need to configure the SpriteKit scene properly. Let's add this to GameScene's didMove:

Game Scene Implementation

```
import SpriteKit

class GameScene: SKScene {
    private var pendulum: InvertedPendulum?
    private var pendulumNode: PendulumNode?
    private var leftButton: SKShapeNode?
```

```
    private var rightButton: SKShapeNode?
    private var debugLabel: SKLabelNode?

    private var lastUpdateTime: TimeInterval = 0
    private var command: Double = 0

    override func didMove(to view: SKView) {
        backgroundColor = .black

        // Configure physics world
        physicsWorld.gravity = CGVector(dx: 0, dy: -9.8)

        // Configure scene update rate
        view.preferredFramesPerSecond = 60
        view.ignoresSiblingOrder = true
        view.showsFPS = true
        view.showsNodeCount = true

        setupSimulation()
        setupPendulum()
        setupControls()
        setupDebugLabel()

        // Run validation on startup
        PendulumTester.validateSimulation()
    }

    private func setupDebugLabel() {
        debugLabel = SKLabelNode(text: "State: Initializing")
        if let debugLabel = debugLabel {
            debugLabel.fontSize = 14
            debugLabel.fontColor = .white
            debugLabel.position = CGPoint(x: frame.midX, y: frame.maxY -
50)
            addChild(debugLabel)
        }
    }

    private func setupSimulation() {
        guard let path = Bundle.main.path(forResource:
"InputPendulumSim", ofType: "csv") else {
            print("Error: Could not find input file")
            return
        }

        do {
            let content = try String(contentsOfFile: path, encoding:
.utf8)
            let rows = content.components(separatedBy: .newlines)
                        .filter { !$0.isEmpty }
```

```
            let parameters = rows.compactMap { row -&gt; Double? in
                let components = row.components(separatedBy: ",")
                guard let firstValue = components.first else { return
nil }
                return Double(firstValue)
            }

            print("Loaded parameters:", parameters)
            pendulum = InvertedPendulum(parameters: parameters)
            debugLabel?.text = "State: Simulation initialized"

        } catch {
            print("Error loading simulation parameters: \(error)")
            debugLabel?.text = "State: Error loading parameters"
        }
    }

    private func setupPendulum() {
        let screenHeight = frame.size.height
        let pendulumLength: CGFloat = screenHeight * 0.3
        let massRadius: CGFloat = 20

        pendulumNode = PendulumNode(length: pendulumLength, massRadius:
massRadius)
        if let pendulumNode = pendulumNode {
            pendulumNode.position = CGPoint(x: frame.midX, y: frame.midY
+ pendulumLength/2)
            addChild(pendulumNode)
        }
    }

    private func setupControls() {
        let buttonSize = CGSize(width: 100, height: 50)
        let buttonY = frame.minY + 100

        // Left button
        leftButton = SKShapeNode(rectOf: buttonSize)
        if let leftButton = leftButton {
            leftButton.position = CGPoint(x: frame.midX - 120, y:
buttonY)
            leftButton.fillColor = .blue
            leftButton.strokeColor = .white
            leftButton.name = "leftButton"
            leftButton.zPosition = 100  // Ensure buttons are above
other nodes
            addChild(leftButton)

            let leftLabel = SKLabelNode(text: "←")
            leftLabel.fontSize = 24
            leftLabel.fontColor = .white
            leftLabel.position = CGPoint(x: 0, y: -8)
```

```
            leftButton.addChild(leftLabel)
        }

        // Right button
        rightButton = SKShapeNode(rectOf: buttonSize)
        if let rightButton = rightButton {
            rightButton.position = CGPoint(x: frame.midX + 120, y:
buttonY)
            rightButton.fillColor = .blue
            rightButton.strokeColor = .white
            rightButton.name = "rightButton"
            rightButton.zPosition = 100  // Ensure buttons are above
other nodes
            addChild(rightButton)

            let rightLabel = SKLabelNode(text: "→")
            rightLabel.fontSize = 24
            rightLabel.fontColor = .white
            rightLabel.position = CGPoint(x: 0, y: -8)
            rightButton.addChild(rightLabel)
        }
    }

    override func touchesBegan(_ touches: Set<uitouch>, with event:
UIEvent?) {
        for touch in touches {
            let location = touch.location(in: self)
            let touchedNodes = nodes(at: location)

            for node in touchedNodes {
                switch node.name {
                case "leftButton":
                    command = -1.0
                    debugLabel?.text = "Command: Left"
                    print("Left button pressed")
                    return

                case "rightButton":
                    command = 1.0
                    debugLabel?.text = "Command: Right"
                    print("Right button pressed")
                    return

                default:
                    break
                }
            }
        }
    }

    override func touchesEnded(_ touches: Set<uitouch>, with event:
```

```
UIEvent?) {
        command = 0
        debugLabel?.text = "Command: None"
        print("Touch ended")
    }

    override func update(_ currentTime: TimeInterval) {
        guard let pendulum = pendulum else {
            print("Pendulum not initialized")
            return
        }

        if lastUpdateTime == 0 {
            lastUpdateTime = currentTime
            return
        }

        let dt = currentTime - lastUpdateTime
        print("Update called - dt: \(dt)")

        // Step simulation
        let (position, velocity) = pendulum.step(command: command)

        // Update visualization
        pendulumNode?.updateAngle(position)

        // Update debug info
        debugLabel?.text = String(format: "Pos: %.3f, Vel: %.3f, Cmd:
%.1f",
                                  position, velocity, command)

        lastUpdateTime = currentTime
    }
}
```

After adding these debug statements, please run the app and let me know what you see in the console output. This will help us determine:

1. If the simulation step function is being called
2. If the time steps are occurring at the correct intervals
3. If the pendulum visualization is being updated
4. If the scene is properly configured

Additionally, one critical thing to check: Did you modify your main view controller or scene configuration to use the GameScene class? In your project's ViewControllerMain or main storyboard, make sure you're properly loading the GameScene:

```
if let view = self.view as! SKView? {
    let scene = GameScene(size: view.bounds.size)
    scene.scaleMode = .aspectFill
```

```
        view.presentScene(scene)
    }
```

Let me know what debug output you see and we can identify where exactly the simulation is getting stuck.

User:

It is giving the error 'Cannot use mutating member on immutable value: 'pendulum' is a 'let' constant' at line 169 of game-scene.swift.

Claude:

Ah, I see the issue! We need to make the pendulum variable mutable in GameScene since the step function is marked as mutating. Let's fix that:

Game Scene Implementation

```
import SpriteKit

class GameScene: SKScene {
    private var pendulum: InvertedPendulum!
    private var pendulumNode: PendulumNode?
    private var leftButton: SKShapeNode?
    private var rightButton: SKShapeNode?
    private var debugLabel: SKLabelNode?

    private var lastUpdateTime: TimeInterval = 0
    private var command: Double = 0

    override func didMove(to view: SKView) {
        backgroundColor = .black

        // Configure physics world
        physicsWorld.gravity = CGVector(dx: 0, dy: -9.8)

        // Configure scene update rate
        view.preferredFramesPerSecond = 60
        view.ignoresSiblingOrder = true
        view.showsFPS = true
        view.showsNodeCount = true

        setupSimulation()
        setupPendulum()
        setupControls()
        setupDebugLabel()

        // Run validation on startup
        PendulumTester.validateSimulation()
    }
```

```
    private func setupDebugLabel() {
        debugLabel = SKLabelNode(text: "State: Initializing")
        if let debugLabel = debugLabel {
            debugLabel.fontSize = 14
            debugLabel.fontColor = .white
            debugLabel.position = CGPoint(x: frame.midX, y: frame.maxY -
50)
            addChild(debugLabel)
        }
    }

    private func setupSimulation() {
        guard let path = Bundle.main.path(forResource:
"InputPendulumSim", ofType: "csv") else {
            print("Error: Could not find input file")
            return
        }

        do {
            let content = try String(contentsOfFile: path, encoding:
.utf8)
            let rows = content.components(separatedBy: .newlines)
                              .filter { !$0.isEmpty }

            let parameters = rows.compactMap { row -&gt; Double? in
                let components = row.components(separatedBy: ",")
                guard let firstValue = components.first else { return
nil }
                return Double(firstValue)
            }

            print("Loaded parameters:", parameters)
            pendulum = InvertedPendulum(parameters: parameters)
            debugLabel?.text = "State: Simulation initialized"

        } catch {
            print("Error loading simulation parameters: \(error)")
            debugLabel?.text = "State: Error loading parameters"
        }
    }

    private func setupPendulum() {
        let screenHeight = frame.size.height
        let pendulumLength: CGFloat = screenHeight * 0.3
        let massRadius: CGFloat = 20

        pendulumNode = PendulumNode(length: pendulumLength, massRadius:
massRadius)
        if let pendulumNode = pendulumNode {
            pendulumNode.position = CGPoint(x: frame.midX, y: frame.midY
```

```
    + pendulumLength/2)
                addChild(pendulumNode)
            }
        }

        private func setupControls() {
            let buttonSize = CGSize(width: 100, height: 50)
            let buttonY = frame.minY + 100

            // Left button
            leftButton = SKShapeNode(rectOf: buttonSize)
            if let leftButton = leftButton {
                leftButton.position = CGPoint(x: frame.midX - 120, y:
    buttonY)
                leftButton.fillColor = .blue
                leftButton.strokeColor = .white
                leftButton.name = "leftButton"
                leftButton.zPosition = 100  // Ensure buttons are above
    other nodes
                addChild(leftButton)

                let leftLabel = SKLabelNode(text: "←")
                leftLabel.fontSize = 24
                leftLabel.fontColor = .white
                leftLabel.position = CGPoint(x: 0, y: -8)
                leftButton.addChild(leftLabel)
            }

            // Right button
            rightButton = SKShapeNode(rectOf: buttonSize)
            if let rightButton = rightButton {
                rightButton.position = CGPoint(x: frame.midX + 120, y:
    buttonY)
                rightButton.fillColor = .blue
                rightButton.strokeColor = .white
                rightButton.name = "rightButton"
                rightButton.zPosition = 100  // Ensure buttons are above
    other nodes
                addChild(rightButton)

                let rightLabel = SKLabelNode(text: "→")
                rightLabel.fontSize = 24
                rightLabel.fontColor = .white
                rightLabel.position = CGPoint(x: 0, y: -8)
                rightButton.addChild(rightLabel)
            }
        }

        override func touchesBegan(_ touches: Set<uitouch>, with event:
    UIEvent?) {
            for touch in touches {
```

```
            let location = touch.location(in: self)
            let touchedNodes = nodes(at: location)

            for node in touchedNodes {
                switch node.name {
                case "leftButton":
                    command = -1.0
                    debugLabel?.text = "Command: Left"
                    print("Left button pressed")
                    return

                case "rightButton":
                    command = 1.0
                    debugLabel?.text = "Command: Right"
                    print("Right button pressed")
                    return

                default:
                    break
                }
            }
        }
    }

    override func touchesEnded(_ touches: Set<uitouch>, with event:
UIEvent?) {
        command = 0
        debugLabel?.text = "Command: None"
        print("Touch ended")
    }

    override func update(_ currentTime: TimeInterval) {
        guard let pendulum = pendulum else {
            print("Pendulum not initialized")
            return
        }

        if lastUpdateTime == 0 {
            lastUpdateTime = currentTime
            return
        }

        let dt = currentTime - lastUpdateTime
        print("Update called - dt: \(dt)")

        // Step simulation
        let (position, velocity) = pendulum.step(command: command)

        // Update visualization
        pendulumNode?.updateAngle(position)
```

```
        // Update debug info
        debugLabel?.text = String(format: "Pos: %.3f, Vel: %.3f, Cmd:
%.1f",
                                  position, velocity, command)

        lastUpdateTime = currentTime
    }
}
```

Then in update:

Game Scene Implementation

```
import SpriteKit

class GameScene: SKScene {
    private var pendulum: InvertedPendulum!
    private var pendulumNode: PendulumNode?
    private var leftButton: SKShapeNode?
    private var rightButton: SKShapeNode?
    private var debugLabel: SKLabelNode?

    private var lastUpdateTime: TimeInterval = 0
    private var command: Double = 0

    override func didMove(to view: SKView) {
        backgroundColor = .black

        // Configure physics world
        physicsWorld.gravity = CGVector(dx: 0, dy: -9.8)

        // Configure scene update rate
        view.preferredFramesPerSecond = 60
        view.ignoresSiblingOrder = true
        view.showsFPS = true
        view.showsNodeCount = true

        setupSimulation()
        setupPendulum()
        setupControls()
        setupDebugLabel()

        // Run validation on startup
        PendulumTester.validateSimulation()
    }

    private func setupDebugLabel() {
        debugLabel = SKLabelNode(text: "State: Initializing")
        if let debugLabel = debugLabel {
            debugLabel.fontSize = 14
```

```
            debugLabel.fontColor = .white
            debugLabel.position = CGPoint(x: frame.midX, y: frame.maxY -
50)
            addChild(debugLabel)
        }
    }

    private func setupSimulation() {
        guard let path = Bundle.main.path(forResource:
"InputPendulumSim", ofType: "csv") else {
            print("Error: Could not find input file")
            return
        }

        do {
            let content = try String(contentsOfFile: path, encoding:
.utf8)
            let rows = content.components(separatedBy: .newlines)
                              .filter { !$0.isEmpty }

            let parameters = rows.compactMap { row -&gt; Double? in
                let components = row.components(separatedBy: ",")
                guard let firstValue = components.first else { return
nil }
                return Double(firstValue)
            }

            print("Loaded parameters:", parameters)
            pendulum = InvertedPendulum(parameters: parameters)
            debugLabel?.text = "State: Simulation initialized"

        } catch {
            print("Error loading simulation parameters: \(error)")
            debugLabel?.text = "State: Error loading parameters"
        }
    }

    private func setupPendulum() {
        let screenHeight = frame.size.height
        let pendulumLength: CGFloat = screenHeight * 0.3
        let massRadius: CGFloat = 20

        pendulumNode = PendulumNode(length: pendulumLength, massRadius:
massRadius)
        if let pendulumNode = pendulumNode {
            pendulumNode.position = CGPoint(x: frame.midX, y: frame.midY
+ pendulumLength/2)
            addChild(pendulumNode)
        }
    }
```

```
    private func setupControls() {
        let buttonSize = CGSize(width: 100, height: 50)
        let buttonY = frame.minY + 100

        // Left button
        leftButton = SKShapeNode(rectOf: buttonSize)
        if let leftButton = leftButton {
            leftButton.position = CGPoint(x: frame.midX - 120, y:
buttonY)
            leftButton.fillColor = .blue
            leftButton.strokeColor = .white
            leftButton.name = "leftButton"
            leftButton.zPosition = 100  // Ensure buttons are above
other nodes
            addChild(leftButton)

            let leftLabel = SKLabelNode(text: "←")
            leftLabel.fontSize = 24
            leftLabel.fontColor = .white
            leftLabel.position = CGPoint(x: 0, y: -8)
            leftButton.addChild(leftLabel)
        }

        // Right button
        rightButton = SKShapeNode(rectOf: buttonSize)
        if let rightButton = rightButton {
            rightButton.position = CGPoint(x: frame.midX + 120, y:
buttonY)
            rightButton.fillColor = .blue
            rightButton.strokeColor = .white
            rightButton.name = "rightButton"
            rightButton.zPosition = 100  // Ensure buttons are above
other nodes
            addChild(rightButton)

            let rightLabel = SKLabelNode(text: "→")
            rightLabel.fontSize = 24
            rightLabel.fontColor = .white
            rightLabel.position = CGPoint(x: 0, y: -8)
            rightButton.addChild(rightLabel)
        }
    }

    override func touchesBegan(_ touches: Set<uitouch>, with event:
UIEvent?) {
        for touch in touches {
            let location = touch.location(in: self)
            let touchedNodes = nodes(at: location)

            for node in touchedNodes {
                switch node.name {
```

```
                case "leftButton":
                    command = -1.0
                    debugLabel?.text = "Command: Left"
                    print("Left button pressed")
                    return

                case "rightButton":
                    command = 1.0
                    debugLabel?.text = "Command: Right"
                    print("Right button pressed")
                    return

                default:
                    break
                }
            }
        }
    }

    override func touchesEnded(_ touches: Set<uitouch>, with event:
UIEvent?) {
        command = 0
        debugLabel?.text = "Command: None"
        print("Touch ended")
    }

    override func update(_ currentTime: TimeInterval) {
        // No need for guard let since we're using force unwrap
        let stepResult = pendulum.step(command: command)

        if lastUpdateTime == 0 {
            lastUpdateTime = currentTime
            return
        }

        let dt = currentTime - lastUpdateTime
        print("Update called - dt: \(dt)")

        // Step simulation
        let (position, velocity) = pendulum.step(command: command)

        // Update visualization
        pendulumNode?.updateAngle(position)

        // Update debug info
        debugLabel?.text = String(format: "Pos: %.3f, Vel: %.3f, Cmd:
%.1f",
                                  position, velocity, command)

        lastUpdateTime = currentTime
```

```
        }
    }
```

And update the output assignment:

Game Scene Implementation

```
import SpriteKit

class GameScene: SKScene {
    private var pendulum: InvertedPendulum!
    private var pendulumNode: PendulumNode?
    private var leftButton: SKShapeNode?
    private var rightButton: SKShapeNode?
    private var debugLabel: SKLabelNode?

    private var lastUpdateTime: TimeInterval = 0
    private var command: Double = 0

    override func didMove(to view: SKView) {
        backgroundColor = .black

        // Configure physics world
        physicsWorld.gravity = CGVector(dx: 0, dy: -9.8)

        // Configure scene update rate
        view.preferredFramesPerSecond = 60
        view.ignoresSiblingOrder = true
        view.showsFPS = true
        view.showsNodeCount = true

        setupSimulation()
        setupPendulum()
        setupControls()
        setupDebugLabel()

        // Run validation on startup
        PendulumTester.validateSimulation()
    }

    private func setupDebugLabel() {
        debugLabel = SKLabelNode(text: "State: Initializing")
        if let debugLabel = debugLabel {
            debugLabel.fontSize = 14
            debugLabel.fontColor = .white
            debugLabel.position = CGPoint(x: frame.midX, y: frame.maxY -
50)
            addChild(debugLabel)
        }
    }
```

```
    private func setupSimulation() {
        guard let path = Bundle.main.path(forResource:
"InputPendulumSim", ofType: "csv") else {
            print("Error: Could not find input file")
            return
        }

        do {
            let content = try String(contentsOfFile: path, encoding:
.utf8)
            let rows = content.components(separatedBy: .newlines)
                              .filter { !$0.isEmpty }

            let parameters = rows.compactMap { row -&gt; Double? in
                let components = row.components(separatedBy: ",")
                guard let firstValue = components.first else { return
nil }
                return Double(firstValue)
            }

            print("Loaded parameters:", parameters)
            pendulum = InvertedPendulum(parameters: parameters)
            debugLabel?.text = "State: Simulation initialized"

        } catch {
            print("Error loading simulation parameters: \(error)")
            debugLabel?.text = "State: Error loading parameters"
        }
    }

    private func setupPendulum() {
        let screenHeight = frame.size.height
        let pendulumLength: CGFloat = screenHeight * 0.3
        let massRadius: CGFloat = 20

        pendulumNode = PendulumNode(length: pendulumLength, massRadius:
massRadius)
        if let pendulumNode = pendulumNode {
            pendulumNode.position = CGPoint(x: frame.midX, y: frame.midY
+ pendulumLength/2)
            addChild(pendulumNode)
        }
    }

    private func setupControls() {
        let buttonSize = CGSize(width: 100, height: 50)
        let buttonY = frame.minY + 100

        // Left button
        leftButton = SKShapeNode(rectOf: buttonSize)
```

```
        if let leftButton = leftButton {
            leftButton.position = CGPoint(x: frame.midX - 120, y:
buttonY)
            leftButton.fillColor = .blue
            leftButton.strokeColor = .white
            leftButton.name = "leftButton"
            leftButton.zPosition = 100  // Ensure buttons are above
other nodes
            addChild(leftButton)

            let leftLabel = SKLabelNode(text: "←")
            leftLabel.fontSize = 24
            leftLabel.fontColor = .white
            leftLabel.position = CGPoint(x: 0, y: -8)
            leftButton.addChild(leftLabel)
        }

        // Right button
        rightButton = SKShapeNode(rectOf: buttonSize)
        if let rightButton = rightButton {
            rightButton.position = CGPoint(x: frame.midX + 120, y:
buttonY)
            rightButton.fillColor = .blue
            rightButton.strokeColor = .white
            rightButton.name = "rightButton"
            rightButton.zPosition = 100  // Ensure buttons are above
other nodes
            addChild(rightButton)

            let rightLabel = SKLabelNode(text: "→")
            rightLabel.fontSize = 24
            rightLabel.fontColor = .white
            rightLabel.position = CGPoint(x: 0, y: -8)
            rightButton.addChild(rightLabel)
        }
    }

    override func touchesBegan(_ touches: Set<uitouch>, with event:
UIEvent?) {
        for touch in touches {
            let location = touch.location(in: self)
            let touchedNodes = nodes(at: location)

            for node in touchedNodes {
                switch node.name {
                case "leftButton":
                    command = -1.0
                    debugLabel?.text = "Command: Left"
                    print("Left button pressed")
                    return
```

```
                case "rightButton":
                    command = 1.0
                    debugLabel?.text = "Command: Right"
                    print("Right button pressed")
                    return

                default:
                    break
                }
            }
        }
    }

    override func touchesEnded(_ touches: Set<uitouch>, with event:
UIEvent?) {
        command = 0
        debugLabel?.text = "Command: None"
        print("Touch ended")
    }

    override func update(_ currentTime: TimeInterval) {
        // No need for guard let since we're using force unwrap
        let stepResult = pendulum.step(command: command)

        if lastUpdateTime == 0 {
            lastUpdateTime = currentTime
            return
        }

        let dt = currentTime - lastUpdateTime
        print("Update called - dt: \(dt)")

        // Step simulation and update visualization
        let (position, velocity) = stepResult
        pendulumNode?.updateAngle(position)

        // Update debug info
        debugLabel?.text = String(format: "Pos: %.3f, Vel: %.3f, Cmd:
%.1f",
                                  position, velocity, command)

        lastUpdateTime = currentTime
    }
}
```

The error occurred because we were trying to call a mutating function on a copy of the optional pendulum variable. By making it non-optional with an implicitly unwrapped optional (!) and accessing it directly, we can modify it in place.

This should resolve the compilation error. Try running it now and let me know if you see the pendulum moving. The debug output we added earlier should help us verify if the simulation is

running correctly.

User:

You got it to work! It falls and the output matches the Swift solver. Can you modify the code so the last input, the 9th row in InputPendulumSim.csv is the joystick constant, and apply it to the model if it isn't already?

Claude:

I'll help modify the code to properly use the joystick constant (kj) from the input file.

Inverted Pendulum Simulation

```
import Foundation

struct InvertedPendulum {
    // System parameters
    let ma: Double      // Mass
    let Iz: Double      // Moment of inertia
    let ln: Double      // Length
    let gr: Double      // Gravity
    let ksp: Double     // Spring constant
    let bv: Double      // Damping coefficient
    let kj: Double      // Joy constant
    let tinc: Double    // Time increment
    let fallBoundary: Double // Fall boundary in radians

    // State variables
    private var position: Double
    private var velocity: Double

    init(parameters: [Double]) {
        self.ma = parameters[0]
        self.Iz = parameters[1]
        self.ln = parameters[2]
        self.gr = parameters[3]
        self.ksp = parameters[4]
        self.bv = parameters[5]
        self.tinc = parameters[6]
        self.fallBoundary = parameters[7].degreesToRadians
        self.kj = parameters[8]  // 9th row is joystick constant

        // Set initial conditions to match MATLAB simulation
        self.position = 0.05  // Initial position of 0.05 radians
        self.velocity = 0.0   // Initial velocity of 0 rad/s
    }

    // Convert the system into first-order differential equations
    private func derivatives(_ t: Double, _ state: [Double], command:
Double) -&gt; [(Double, [Double]) -&gt; Double] {
```

```
            return [
                // θ' = velocity
                { (_, _) in state[1] },

                // θ'' = ka*sin(θ) - ks*θ - kb*θ' + kj*command
                { (_, _) in
                    let ka = (ma * ln * gr) / (ma * ln * ln + Iz)
                    let ks = ksp / (ma * ln * ln + Iz)
                    let kb = bv / (ma * ln * ln + Iz)

                    return ka * sin(state[0]) - ks * state[0] - kb *
state[1] + kj * command
                }
            ]
        }

        mutating func step(command: Double) -&gt; (position: Double,
velocity: Double) {
            print("Step called - Current state - Position: \(position),
Velocity: \(velocity)")
            let currentState = [position, velocity]

            // Use RK4 to integrate one time step
            let nextState = rK4(tinc, 0.0, currentState, functions:
derivatives(0.0, currentState, command: command))

            // Update state
            position = nextState[0]
            velocity = nextState[1]

            // Check fall boundary
            if abs(position) &gt; fallBoundary {
                position = position &gt; 0 ? fallBoundary : -fallBoundary
                velocity = 0
            }

            return (position, velocity)
        }
    }

    // Helper extension
    extension Double {
        var degreesToRadians: Double {
            return self * .pi / 180.0
        }

        var radiansToDegrees: Double {
            return self * 180.0 / .pi
        }
    }
```

Let's also add some debug output to verify the joystick constant is being properly applied:

Inverted Pendulum Simulation

```
import Foundation

struct InvertedPendulum {
    // System parameters
    let ma: Double      // Mass
    let Iz: Double      // Moment of inertia
    let ln: Double      // Length
    let gr: Double      // Gravity
    let ksp: Double     // Spring constant
    let bv: Double      // Damping coefficient
    let kj: Double      // Joy constant
    let tinc: Double    // Time increment
    let fallBoundary: Double // Fall boundary in radians

    // State variables
    private var position: Double
    private var velocity: Double

    init(parameters: [Double]) {
        self.ma = parameters[0]
        self.Iz = parameters[1]
        self.ln = parameters[2]
        self.gr = parameters[3]
        self.ksp = parameters[4]
        self.bv = parameters[5]
        self.tinc = parameters[6]
        self.fallBoundary = parameters[7].degreesToRadians
        self.kj = parameters[8]  // 9th row is joystick constant

        // Set initial conditions to match MATLAB simulation
        self.position = 0.05  // Initial position of 0.05 radians
        self.velocity = 0.0   // Initial velocity of 0 rad/s
    }

    // Convert the system into first-order differential equations
    private func derivatives(_ t: Double, _ state: [Double], command:
Double) -&gt; [(Double, [Double]) -&gt; Double] {
        return [
            // θ' = velocity
            { (_, _) in state[1] },

            // θ'' = ka*sin(θ) - ks*θ - kb*θ' + kj*command
            { (_, _) in
                let ka = (ma * ln * gr) / (ma * ln * ln + Iz)
                let ks = ksp / (ma * ln * ln + Iz)
                let kb = bv / (ma * ln * ln + Iz)
```

```
                return ka * sin(state[0]) - ks * state[0] - kb *
state[1] + kj * command
            }
        ]
    }

    mutating func step(command: Double) -&gt; (position: Double,
velocity: Double) {
        let appliedForce = command * kj
        print("Step called - Position: \(position), Velocity: \
(velocity), Command: \(command), Applied Force: \(appliedForce)")
        let currentState = [position, velocity]

        // Use RK4 to integrate one time step
        let nextState = rK4(tinc, 0.0, currentState, functions:
derivatives(0.0, currentState, command: command))

        // Update state
        position = nextState[0]
        velocity = nextState[1]

        // Check fall boundary
        if abs(position) &gt; fallBoundary {
            position = position &gt; 0 ? fallBoundary : -fallBoundary
            velocity = 0
        }

        return (position, velocity)
    }
}

// Helper extension
extension Double {
    var degreesToRadians: Double {
        return self * .pi / 180.0
    }

    var radiansToDegrees: Double {
        return self * 180.0 / .pi
    }
}
```

The joystick constant (kj) is now being properly initialized from the input file and applied to the command in the step function. When you press the left/right buttons:

- command = -1.0 or 1.0
- This gets multiplied by kj to produce the actual force applied to the pendulum
- The debug output will show you the magnitude of the force being applied

You should now be able to better control the pendulum since the joystick force scaling is properly calibrated. Let me know if you'd like me to adjust the scaling or add more visual feedback about the

control forces being applied.

User:

Can you give some example inputs that would allow me to move the pendulum when I click on the buttons?

Claude:

Based on the physics described in the inverted pendulum lab document, I'll suggest some parameters that should make the pendulum controllable.

Here's a set of parameters for InputPendulumSim.csv that should give noticeable movement with button presses:

```
70.0        # ma (mass in kg)
2.0         # Iz (moment of inertia)
1.0         # ln (length in meters)
9.81        # gr (gravity)
700.0       # ksp (spring constant)
5.0         # bv (damping)
0.01        # tinc (time step)
90.0        # fall boundary in degrees
50.0        # kj (joystick constant) - increased this significantly
```

The key changes I made:

1. Increased kj (joystick constant) to 50.0 - this will make the button inputs have a more noticeable effect
2. Kept reasonable damping (bv=5.0) so the pendulum doesn't oscillate too wildly
3. Used a standard spring constant (ksp=700.0) that provides some restoring force

When you press the buttons:

- Left button: command = -1.0 * 50.0 = -50.0 torque
- Right button: command = 1.0 * 50.0 = 50.0 torque

These forces should be large enough to visibly move the pendulum while still allowing you to control it.

Would you like me to adjust any of these parameters or add visualization of the applied forces to help with the control?
