Data-Driven Hierarchical Runge-Kutta and Adams Methods for Nonlinear Dynamical Systems

Shyamal Suhana Chandra

2025

Abstract

This paper presents a comprehensive implementation of numerical methods for solving nonlinear differential equations, including Euler's Method, Data-Driven Euler's Method, Runge-Kutta 3rd order method, Data-Driven Runge-Kutta, Adams Methods, and Data-Driven Adams Methods. We introduce novel data-driven hierarchical architectures inspired by transformer networks that enhance traditional numerical integration methods. The framework is implemented in C/C++ with Objective-C visualization capabilities, making it suitable for macOS and VisionOS platforms.

1 Introduction

Numerical methods for solving ordinary differential equations (ODEs) are fundamental tools in scientific computing. We present a comprehensive framework including Euler's Method, Data-Driven Euler's Method, Runge-Kutta 3rd order, Data-Driven Runge-Kutta, Adams Methods, and Data-Driven Adams Methods.

2 Euler's Method

Euler's Method is the simplest numerical method for solving ODEs. It is a first-order explicit method:

$$y_{n+1} = y_n + h \cdot f(t_n, y_n) \tag{1}$$

where h is the step size, f is the ODE function, and y_n is the state at time t_n. The local truncation error is $O(h^2)$, making it a first-order method.

2.1 Data-Driven Euler's Method

We extend Euler's Method with a hierarchical transformer-inspired architecture:

$$y_{n+1} = y_n + h \cdot f(t_n, y_n) + h \cdot \alpha \cdot \text{Attention}(y_n) \tag{2}$$

where α is a learning rate and $\text{Attention}(y_n)$ is a hierarchical attention mechanism that refines the Euler step using multiple transformer layers.

3 Runge-Kutta 3rd Order Method

The Runge-Kutta 3rd order method (RK3) is defined by the following stages:

$$k_1 = f(t_n, y_n) \tag{3}$$

$$k_2 = f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1) \tag{4}$$

$$k_3 = f(t_n + h, y_n - hk_1 + 2hk_2) \tag{5}$$

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 4k_2 + k_3) \tag{6}$$

where h is the step size, f is the ODE function, and y_n is the state at time t_n.

4 Adams Methods

Adams-Bashforth and Adams-Moulton methods are multi-step methods that use information from previous steps.

4.1 Adams-Bashforth 3rd Order

The predictor step:

$$y_{n+1} = y_n + \frac{h}{12}(23f_n - 16f_{n-1} + 5f_{n-2}) \tag{7}$$

4.2 Adams-Moulton 3rd Order

The corrector step:

$$y_{n+1} = y_n + \frac{h}{12}(5f_{n+1} + 8f_n - f_{n-1}) \tag{8}$$

5 Parallel, Distributed, and Concurrent Execution

We extend all numerical methods with comprehensive parallel and distributed computing support:

5.1 Parallel Execution Modes

- **OpenMP**: Shared-memory multi-threading for single-node parallelization
- **POSIX Threads (pthreads)**: Fine-grained thread control
- **MPI**: Distributed computing across multiple nodes
- **Hybrid**: Combined MPI + OpenMP for hierarchical parallelism

5.2 Concurrent Execution

Multiple methods can execute simultaneously, enabling real-time comparison and ensemble approaches. The concurrent execution framework manages resource allocation and synchronization across parallel method instances.

5.3 Real-Time, Online, and Dynamic Methods

We extend all numerical methods with real-time, online, and dynamic execution capabilities:

5.3.1 Real-Time Methods

Real-time methods process streaming data with minimal latency, suitable for live data feeds and continuous monitoring applications. They feature:

- Streaming data buffers for continuous processing
- Callback mechanisms for immediate result delivery
- Low-latency execution optimized for real-time constraints

5.3.2 Online Methods

Online methods adapt to incoming data with incremental learning, adjusting parameters based on observed errors:

- Adaptive step size control based on error estimates
- Learning rate mechanisms for parameter adjustment
- History tracking for adaptive refinement

5.3.3 Dynamic Methods

Dynamic methods provide fully adaptive execution with dynamic step sizes and parameter adaptation:

- Real-time error and stability estimation
- Dynamic step size adjustment
- Parameter history tracking
- Adaptive mode switching

6 Hierarchical and Stacked Architecture

We propose hierarchical and stacked architectures inspired by transformer networks that process ODE solutions through multiple layers with attention mechanisms. Each layer applies transformations to the state space, enabling adaptive refinement of the numerical solution.

The hierarchical/stacked solver consists of:

- Multiple processing layers with learnable weights
- Attention mechanisms for state-space transformations
- Residual connections for gradient flow
- Adaptive step size control based on hierarchical features
- Stacked configurations for deep hierarchical processing

6.1 Stacked Configurations

Stacked methods process solutions through multiple hierarchical layers:

$$y^{(l+1)} = \text{Attention}(y^{(l)}) + \text{Residual}(y^{(l)}) \tag{9}$$

where l denotes the layer index and the attention mechanism applies transformer-like transformations.

7 Implementation

The framework is implemented in C/C++ for core numerical methods, with Objective-C wrappers for visualization and integration with Apple platforms.

8 Test Cases and Validation

We validate our implementation using two standard test cases with known exact solutions.

8.1 Exponential Decay Test

The exponential decay ODE provides a simple test case:

$$\frac{dy}{dt} = -y, \quad y(0) = 1.0 \tag{10}$$

The exact solution is $y(t) = y_0 \exp(-t)$. We test all four methods (RK3, DDRK3, AM, DDAM) over the interval $t \in [0, 2.0]$ with step size $h = 0.01$.

8.1.1 C/C++ Implementation

The test is implemented in `test_exponential_decay.c`:

```
void exponential_ode(double t, const double* y,
                     double* dydt, void* params) {
    dydt[0] = -y[0];
}

double exact_exponential(double t, double y0) {
    return y0 * exp(-t);
}
```

8.1.2 Objective-C Implementation

The Objective-C test uses the DDRKAM framework:

```
DDRKAMSolver* solver = [[DDRKAMSolver alloc]
                        initWithDimension:1];
NSDictionary* result = [solver solveWithFunction:^(
    double t, const double* y, double* dydt, void* params) {
    dydt[0] = -y[0];
} startTime:0.0 endTime:2.0 initialState:@[@1.0]
stepSize:0.01 params:NULL];
```

8.1.3 Validated Results

All methods achieve high accuracy:

- RK3: 99.999992% accuracy, 201 steps
- DDRK3: 99.999992% accuracy, 201 steps
- AM: 99.999991% accuracy, 201 steps
- DDAM: 99.999991% accuracy, 201 steps

8.2 Harmonic Oscillator Test

The harmonic oscillator provides a two-dimensional test case:

$$\frac{d^2x}{dt^2} = -x, \quad x(0) = 1.0, \quad v(0) = 0.0 \tag{11}$$

In first-order form: $dx/dt = v$, $dv/dt = -x$. The exact solution is $x(t) = \cos(t)$, $v(t) = -\sin(t)$. We test over one full period $t \in [0, 2\pi]$ with $h = 0.01$.

8.2.1 C/C++ Implementation

The test is implemented in `test_harmonic_oscillator.c`:

```
void oscillator_ode(double t, const double* y,
                    double* dydt, void* params) {
    dydt[0] = y[1];   // dx/dt = v
    dydt[1] = -y[0];  // dv/dt = -x
}

void exact_oscillator(double t, double x0, double v0,
                      double* x, double* v) {
    *x = x0 * cos(t) - v0 * sin(t);
    *v = -x0 * sin(t) - v0 * cos(t);
}
```

8.2.2 Objective-C Implementation

```
DDRKAMSolver* solver = [[DDRKAMSolver alloc]
                        initWithDimension:2];
NSDictionary* result = [solver solveWithFunction:^(
    double t, const double* y, double* dydt, void* params) {
    dydt[0] = y[1];
    dydt[1] = -y[0];
} startTime:0.0 endTime:2*M_PI
initialState:@[@1.0, @0.0] stepSize:0.01 params:NULL];
```

8.2.3 Validated Results

All methods demonstrate excellent accuracy:

- RK3: 99.682004% accuracy, 629 steps
- DDRK3: 99.682003% accuracy, 629 steps
- AM: 99.320833% accuracy, 630 steps
- DDAM: 99.320914% accuracy, 630 steps

9 Results

Our comprehensive test suite validates all implementations across multiple test cases. The exponential decay test demonstrates exceptional accuracy (99.99999%) for all methods, while the harmonic oscillator test shows excellent performance (99.3-99.7%) over a full period.

10 Conclusion

We have presented a comprehensive framework for solving nonlinear ODEs using traditional and data-driven hierarchical methods, suitable for deployment on Apple platforms.

References

[1] Butcher, J. C. (2008). *Numerical Methods for Ordinary Differential Equations*. Wiley.

[2] Gear, C. W. (1971). *Numerical Initial Value Problems in Ordinary Differential Equations*. Prentice-Hall.