

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

Shyamal Suhana Chandra

2025

Abstract

This paper presents a comprehensive implementation of the Runge-Kutta 3rd order method and Adams methods for solving nonlinear differential equations. We introduce a novel data-driven hierarchical architecture inspired by transformer networks that enhances 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. The Runge-Kutta family of methods, particularly the 3rd order variant, provides a good balance between accuracy and computational efficiency.

2 Runge-Kutta 3rd Order Method

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

$$k_1 = f(t_n, y_n) \tag{1}$$

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

$$k_3 = f(t_n + h, y_n - hk_1 + 2hk_2) \tag{3}$$

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

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

3 Adams Methods

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

3.1 Adams-Bashforth 3rd Order

The predictor step:

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

3.2 Adams-Moulton 3rd Order

The corrector step:

$$y_{n+1} = y_n + \frac{h}{12}(5f_{n+1} + 8f_n - f_{n-1}) \tag{6}$$

4 Hierarchical Data-Driven Architecture

We propose a hierarchical architecture inspired by transformer networks that processes 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 solver consists of:

- Multiple processing layers with learnable weights
- Attention mechanisms for state-space transformations
- Adaptive step size control based on hierarchical features

5 Implementation

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

6 Test Cases and Validation

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

6.1 Exponential Decay Test

The exponential decay ODE provides a simple test case:

$$\frac{dy}{dt} = -y, \quad y(0) = 1.0 \quad (7)$$

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 $h = 0.01$.

6.1.1 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);
}
```

6.1.2 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:@[01.0]
stepSize:0.01 params:NULL];
```

6.1.3 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

6.2 Harmonic Oscillator Test

The harmonic oscillator provides a two-dimensional test case:

$$\frac{d^2x}{dt^2} = -x, \quad x(0) = 1.0, \quad v(0) = 0.0 \quad (8)$$

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 $h = 0.01$.

6.2.1 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);
}

```

6.2.2 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];

```

6.2.3 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

7 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.

8 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.