DDRKAM Reference Manual
Data-Driven Runge-Kutta and Adams Methods

Shyamal Suhana Chandra

2025

Contents

1 Introduction

This manual provides comprehensive documentation for the DDRKAM (Data-Driven Runge-Kutta and Adams Methods) framework. The framework implements numerical methods for solving ordinary differential equations (ODEs) with support for traditional and hierarchical data-driven approaches.

The framework includes:

- Euler's Method (1st order)
- Data-Driven Euler's Method (DDEuler)
- Runge-Kutta 3rd Order Method (RK3)
- Data-Driven Runge-Kutta 3rd Order (DDRK3)
- Adams Methods (AM)
- Data-Driven Adams Methods (DDAM)

2 Euler's Method

2.1 Overview

Euler's Method is the simplest numerical method for solving ODEs. It is a first-order explicit method with local truncation error $O(h^2)$.

2.2 Algorithm

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

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

2.3 API Reference

2.3.1 euler_step

Performs a single integration step using Euler's method.

```
double euler_step(ODEFunction f, double t0, double* y0,
                  size_t n, double h, void* params);
```

Parameters:

- `f`: Function pointer to the ODE system
- `t0`: Current time
- `y0`: Current state vector (modified in-place)
- `n`: Dimension of the system
- `h`: Step size
- `params`: User-defined parameters

Returns: New time value (t0 + h)

2.3.2 euler_solve

Solves an ODE system over a time interval using Euler's method.

```
size_t euler_solve(ODEFunction f, double t0, double t_end,
                   const double* y0, size_t n, double h,
                   void* params, double* t_out, double* y_out);
```

3 Data-Driven Euler's Method

3.1 Overview

Data-Driven Euler's Method (DDEuler) extends standard Euler's method with a hierarchical transformer-inspired architecture that applies adaptive corrections to improve accuracy.

3.2 Algorithm

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

where α is a learning rate and $\text{Attention}(y_n)$ is computed through hierarchical transformer layers.

3.3 API Reference

3.3.1 hierarchical_euler_init

Initializes a Data-Driven Euler solver.

```
int hierarchical_euler_init(HierarchicalEulerSolver* solver,
                            size_t num_layers, size_t state_dim,
                            size_t hidden_dim);
```

3.3.2 hierarchical_euler_step

Performs a single integration step using Data-Driven Euler.

```
double hierarchical_euler_step(HierarchicalEulerSolver* solver,
                               ODEFunction f, double t, double* y,
                               double h, void* params);
```

3.3.3 hierarchical_euler_solve

Solves an ODE system using Data-Driven Euler over a time interval.

```
size_t hierarchical_euler_solve(HierarchicalEulerSolver* solver,
                                ODEFunction f, double t0, double t_end,
                                const double* y0, double h, void* params,
                                double* t_out, double* y_out);
```

4 Parallel and Distributed Methods

4.1 Overview

All methods support parallel, distributed, concurrent, hierarchical, and stacked execution modes. This enables:

- Multi-threaded execution (OpenMP, pthreads)
- Distributed computing (MPI)
- Concurrent execution of multiple methods
- Hierarchical/stacked architectures
- Enhanced performance and scalability

4.2 Parallel Runge-Kutta

4.2.1 parallel_rk_init

Initialize parallel RK3 solver.

```
int parallel_rk_init(ParallelRKSolver* solver, size_t state_dim,
                      size_t num_workers, ParallelMode mode,
                      StackedConfig* stacked);
```

4.2.2 parallel_rk_step

Perform parallel RK3 step.

```
double parallel_rk_step(ParallelRKSolver* solver, ODEFunction f,
                         double t, double* y, double h, void* params);
```

4.2.3 stacked_rk_step

Perform stacked/hierarchical RK3 step.

```
double stacked_rk_step(ParallelRKSolver* solver, ODEFunction f,
                      double t, double* y, double h, void* params);
```

4.2.4 concurrent_rk_execute

Execute multiple RK3 instances concurrently.

```
int concurrent_rk_execute(ParallelRKSolver* solvers[], size_t num_solvers,
                         ODEFunction f, double t, const double* y, double h,
                         void* params, double** results);
```

5 Runge-Kutta 3rd Order Method

5.1 Overview

The Runge-Kutta 3rd order method provides a good balance between accuracy and computational efficiency for solving ODEs.

5.2 API Reference

5.2.1 rk3_step

Performs a single integration step using RK3.

```
double rk3_step(ODEFunction f, double t0, double* y0,
               size_t n, double h, void* params);
```

Parameters:

- `f`: Function pointer to the ODE system
- `t0`: Current time

- `y0`: Current state vector (modified in-place)
- `n`: Dimension of the system
- `h`: Step size
- `params`: User-defined parameters

Returns: New time value (t0 + h)

5.2.2 rk3_solve

Solves an ODE system over a time interval.

```
size_t rk3_solve(ODEFunction f, double t0, double t_end,
                    const double* y0, size_t n, double h,
                    void* params, double* t_out, double*
                       y_out);
```

Parameters:

- `f`: Function pointer to the ODE system
- `t0`: Initial time
- `t_end`: Final time
- `y0`: Initial state vector
- `n`: Dimension of the system
- `h`: Step size
- `params`: User-defined parameters
- `t_out`: Output time array (allocated by caller)
- `y_out`: Output state array (n $\times$ num_steps, allocated by caller)

Returns: Number of steps taken

5.3 Example

```
void lorenz(double t, const double* y, double* dydt, void* params) {
    double* p = (double*)params;
    double sigma = p[0], rho = p[1], beta = p[2];
    dydt[0] = sigma * (y[1] - y[0]);
    dydt[1] = y[0] * (rho - y[2]) - y[1];
    dydt[2] = y[0] * y[1] - beta * y[2];
}

double params[3] = {10.0, 28.0, 8.0/3.0};
double y0[3] = {1.0, 1.0, 1.0};
double t_out[100];
double y_out[300];
size_t steps = rk3_solve(lorenz, 0.0, 1.0, y0, 3, 0.01,
                         params, t_out, y_out);
```

6 Adams Methods

6.1 Adams-Bashforth 3rd Order

Predictor step for multi-step integration.

```
void adams_bashforth3(ODEFunction f, const double* t,
                      const double* y, size_t n, double h,
                      void* params, double* y_pred);
```

6.2 Adams-Moulton 3rd Order

Corrector step for multi-step integration.

```
void adams_moulton3(ODEFunction f, const double* t,
                    const double* y, size_t n, double h,
                    void* params, const double* y_pred,
                    double* y_corr);
```

7 Hierarchical Runge-Kutta Method

7.1 Overview

The hierarchical RK method uses a transformer-like architecture with multiple processing layers and attention mechanisms.

7.2 API Reference

7.2.1 hierarchical_rk_init

Initializes a hierarchical RK solver.

```c
int hierarchical_rk_init(HierarchicalRKSolver* solver,
                         size_t num_layers, size_t state_dim,
                         size_t hidden_dim);
```

Returns: 0 on success, -1 on failure

7.2.2 hierarchical_rk_free

Frees resources allocated by the solver.

```c
void hierarchical_rk_free(HierarchicalRKSolver* solver);
```

7.2.3 hierarchical_rk_solve

Solves an ODE using the hierarchical method.

```c
size_t hierarchical_rk_solve(HierarchicalRKSolver* solver,
                             ODEFunction f, double t0, double t_end,
                             const double* y0, double h, void* params,
                             double* t_out, double* y_out);
```

8 Objective-C Framework

8.1 DDRKAMSolver

Main solver class for Objective-C applications.

```
DDRKAMSolver* solver = [[DDRKAMSolver alloc]
                         initWithDimension:3];
NSDictionary* result = [solver solveWithFunction:^(double t,
                                                     const double* y,
                                                     double* dydt,
                                                     void* params) {
    // ODE definition
} startTime:0.0 endTime:1.0
initialState:@[@1.0, @1.0, @1.0]
stepSize:0.01 params:NULL];
```

8.2 DDRKAMVisualizer

Visualization component for plotting solutions.

```
DDRKAMVisualizer* viz = [[DDRKAMVisualizer alloc] init];
NSView* view = [viz createVisualizationViewWithTime:timeArray
```

```
                                                        state:
                                                           stateArray

                                                    dimension:3];
[viz exportToCSV:@"/path/to/output.csv"
            time:timeArray
           state:stateArray];
```

8.3 DDRKAMHierarchicalSolver

Hierarchical solver for Objective-C.

```
DDRKAMHierarchicalSolver* solver =
    [[DDRKAMHierarchicalSolver alloc]
     initWithDimension:3 numLayers:4 hiddenDim:32];
```

9 Platform Support

- macOS 10.13+
- iOS 11.0+
- visionOS 1.0+

10 Copyright