Integro-Differential Equation Solver

Russell J. Phelan

University of Massachusetts Amherst

February 13, 2017

1 Overview

1.1 Program Structure

integro-differential-solver

- solver.m
  - INITIAL CONDITIONS AND GLOBAL PARAMTERS
    - initialization of all matrix types
    - extrapolate IC for pre-calculated derivatives
  - CLASSICAL ODE SOLVE LOOP
    - calc next Runge-Kutta step, with integral term = 0
    - calc 1st, 2nd, 3rd derivatives
    - various error checks and debug print-outs, then loop again
  - INTEGRAL TERM CALC, AND QUANTUM IDE SOLVE LOOP
    - calculate next point for all R functions
    - integrate quantum terms fully from t_0 to this loop's t, using all past R function data
    - calculate and store resulting 'total area'
    - calculate next quantum-corrected Runge-Kutta step, then loop again
  - CALC BASEM'S ANALYTICAL FIRST APPROX FOR TESTING AND DEBUGGING
  - VARIOUS PLOTS
- **equations**
  - equations.m
  - r_funcs.m
- **ode-solver**
  - runge_step.m
- **integration**
  - causal_nonlocal_int.m
- **helpers**
- **tests**
- **notes**

1.2 What it Does

This documentation is written so that anyone with basic Matlab experience can understand that choices made in the design of this integro-differential solver, and how to adapt it to solve novel problems.

The particular equation that the code is set up to solve is from physics. It results from quantum loop gravity theory, and governs the change in the "scale factor", a quantity representing the expansion/contraction of the universe. It should be easily adaptable to any equation of a similar form.

The general form that the program solves is as follows:

$$\dot{a}(t) = P(a, \dot{a}, \ddot{a}, \cdots, t) \int \mathrm{d}t' R(a, \dot{a}, \ddot{a}, \cdots, t') \frac{1}{t - t'} \tag{1}$$

Here, t' is a dummy variable, integrated from t_0 to the t in current iteration.

As is, the system is also set up to accommodate for behavior **(see section 1.3)** that truncates the integral early, and adds a logarithmic term in order to prevent a non-physical divergence. This behavior can easily be removed if it isn't needed. **Be careful not to run the program as is without removing this truncation behavior if you do not explicitly need it.**

1.3 Numerical Methods Used

The differential equation solver is a standard Runge-Kutta type, the workings of which can be found just about anywhere. The basic idea is that at the current t, $\dot{a}(t)$ is determined from the differential equation. Then, several linear approximations are taken, some of which are overestimates, others underestimates. The slopes of these approximations are averaged. The program uses the average slope to approximate $a(t)$ over the next small interval $\mathrm{d}t$, and stores its value for $a(t + \mathrm{d}t)$. It then uses this new value to get $\dot{a}(t + \mathrm{d}t)$, and the iteration repeats.

In order to find $\dot{a}(t)$ at the next point, the program needs to evaluate an integral, thus 'integro-differential equaton'. This integral is evaluated numerically using the trapezoid rule. The specific formula used follows, where N is the number of grid points, $x_1 = a$, and $x_N = b$:

$$\int_a^b f(x)\mathrm{d}x = \frac{b-a}{2N}(f(x_1) + 2f(x_2) + 2f(x_3) + \cdot + 2f(x_{N-1}) + f(x_N)) \tag{2}$$

Note that there is one less "trapezoid" than there are grid points.

1.4 Implementing the Causal Non-Local Function

The causal non-local function is given as follows:

$$\mathcal{L}(t - t') = \lim_{\epsilon \to 0} \left[\frac{\theta(t - t' - \epsilon)}{t - t'} + \delta(t - t') \log(\mu_R \epsilon) \right] \tag{3}$$

This function multiplies the integrand in the quantum loop gravity equation currently configured, as follows:

$$\dot{a}(t) = P(a, \dot{a}, \ddot{a}, \cdots, t) \int \mathrm{d}t' R(a, \dot{a}, \ddot{a}, \cdots, t') \mathcal{L}(t - t') \tag{4}$$

Examining the equation (3) shows that $\mathcal{L}(t - t'$ contains the $\frac{1}{t-t'}$ behavior, but also has the effect of "shutting off" the integral ϵ away from the upper bound (Heaviside function). The delta function has the effect of tacking the logarithmic term onto the final sum at the end. This is exactly how this behavior has been implemented in the program: terminate integral early, tack logarithmic term on the end.

It should be noted that

1.5 Discontinuities Due to Discretization

Since the function values are discrete, there is no way to truly represent a continuous function using our function model. When working with equations that are less stable, small discontinuities can propagate into serious errors. For this reason, a sort of smoothness interpolating method has been implemented. This section explains the choices made, and mechanics behind this method.

2 The Function Structure

Functions are represented using Matlab's Array/Matrix data type. Here is a quick tutorial for Matlab array manipulations.

In math, a function is defined as a set of tuples, $(x, f(x))$. Functions in this program work similarly. We use $2 \times n$ arrays, where the indices and function values are organized as in the following diagram:

	1	2	3	
1	$f(x_1)$	$f(x_2)$	$f(x_3)$	$\ldots$
2	x_1	x_2	x_3	$\ldots$

(5)

The spacing between values is determined by the `step` parameter, found under `%INITIAL CONDITIONS/PARAMETERS` in `solver.m`.