Instruction Manual for Using the Numerical Solvers

Peter Gillich
260803469

January 5, 2022

The solvers are intended to solve IVP's of the form

$$\begin{cases} y'(t) = f\left(y(t), \frac{a^j}{\Gamma(j)} \int_{-\infty}^{t} y(s) e^{-a(t-s)} (t-s)^{j-1} ds\right); \quad t_0 \leq t \leq t_f;\ a > 0, j > 0 \\ y(t) = \phi(t);\ t < t_0; \end{cases}$$

0.1 Fixed Stepsize Solvers

The signature of these solvers is:

```
solverName(F, phi, tau, j, left, right, h)
```

The numerical solvers I have worked on all function the same way. They take as input:

- A function handle representing the right-hand-side function f, written in terms of $y(t)$ and the delay term $I(t)$; that is, in MATLAB, `F = @(x, d) ...` , where `x` stands for $y(t)$ and `d` stands for $I(t) = \frac{a^j}{\Gamma(j)} \int_{-\infty}^{t} y(s) g_a^j(t-s) ds.$;
- A function handle representing the history function $\phi(t)$; in MATLAB, `phi = @(t) ...`
- `left, right` are the endpoints of the interval $[t_0, t_f]$ over which we solve the IVP above;
- The parameters j and τ, where $\tau = \frac{j}{a}$ is the so-called mean delay;
- Finally, the mesh size of the FCRK, h.

A desired step size is entered by the user, but since not every choice of step size will evenly divide the interval over which we solve the IVP, the solvers find a step size slightly smaller than the one entered by the user, which *does* evenly divide the IVP interval. The process through which this is done is given in the implementation section of the final report for this project.

The function gives the following output:

- a structure `sol` with fields
  - `sol.x`, a column vector of the evenly-spaced mesh points t_i, where the nth row contains the mesh point t_{n-1}
  - `sol.y`, a matrix containing the computed values u_i, as well as the coefficients of the interpolant defining the continuous solution $\eta_i(t)$ on the interval $[t_i, t_{i+1}]$. The nth row of `sol.y` contains, from the leftmost column to the rightmost,

    * u_{n-1};
    * the coefficients of the interpolant over the interval $[t_{n-1}, t_n]$, in increasing order of degree.
  - `sol.phi`, which is the anonymous function entered by the user as a history function. Its use will be explained later.
  - `sol.interpolantOrder`, which records the local order of the interpolant used between the u_n. For example, a linear interpolant is of order 2, cubic Hermite of order 4, etc.

  The size of `sol.y` depends on the order of the method. Methods of order ≤ 2 employ a linear interpolant between the points u_n, while methods of order > 2 use a cubic Hermite interpolant.

The fixed stepsize solvers available now are:

- `ddef1`, of order 1;
- `ddef2`, of order 2;
- `ddef3`, of order 3;
- `ddef4`, of order 4.

0.2 Variable Stepsize Solvers

The signature of these solvers is:

```
solverName(F, phi, tau, j, left, right, errTol),
```

where the only change from the above inputs is the use of `errTol`, which is the user-entered local truncation error tolerance.

the function gives the following output:

- a structure `sol` with fields
  - `sol.x`, a column vector of the mesh points t_i, this time not evenly spaced because the step size is varied.
  - `sol.y`, a matrix containing the computed values u_i, as well as the coefficients of the interpolant defining the continuous solution $\eta_i(t)$ on the interval $[t_i, t_{i+1}]$, in the same manner as above.
  - `sol.phi`
  - `sol.successfulSteps`, the number of steps the solver used;
  - `sol.rejectedSteps`, the number of steps resulting in local truncation error exceeding `errTol`.
  - `sol.interpolantOrder`.

The variable stepsize solvers available now are:

- `ddev21`, variable stepsize method with $p = 2$ (See final report, section on variable stepsize methods), stepping with the lower order method;

- `ddev12` variable stepsize method with $p = 2$, stepping with the higher order method;
- `ddev32` variable stepsize method with $p = 3$, stepping with the lower order method;
- `ddev23` variable stepsize method with $p = 3$, stepping with the higher order method;
- `ddev43` variable stepsize method with $p = 4$, stepping with the lower order method;
- `ddev34` variable stepsize method with $p = 4$, stepping with the higher order method;

Currently, the variable time stepping methods do not work properly.

0.3 Auxiliary Functions

These solvers above require the following auxiliary files in order to run:

- `linTerp.m`, for any fixed stepsize solver of order ≤ 2 and any variable stepsize solver with $p \leq 2$;
- `herm3terp.m`, for higher order solvers, and for variable stepsize solvers with $p > 2$.

It is best to just include both files in the same folder containing the solvers, and not worry about which one specifically is needed where.

Finally, in order to evaluate solution structures anywhere in the interval $[t_0, t_f]$, we require also the file `evalSol.m`, containing the function `evalSol`, with signature

```
evalSol(sol, pointsToEval).
```

The function takes as input a solution structure `sol` from one of the solvers mentioned in the previous sections, along with a column vector of points `pointsToEval` that the user wishes to evaluate the solution on. The points must be listed in increasing order, and cannot lie outside the interval $[t_0, t_f]$.

`evalSol` returns a vector whose ith entry corresponds to the numerical solution's value at the ith entry of `pointsToEval`.