CAAM 419/519, Homework #2

wnl1

September 30, 2022

1 Verification of The Correctness of The Forward Euler and Explicit Midpoint ODE Solver

Figures 1 and 2 show the initial and final position of 100 particles over a 1 second time span in a velocity field given by Equation 1 below. Figure 1 used the Forward Euler method while Figure 2 used the Explicit Midpoint method to solve for the final position.

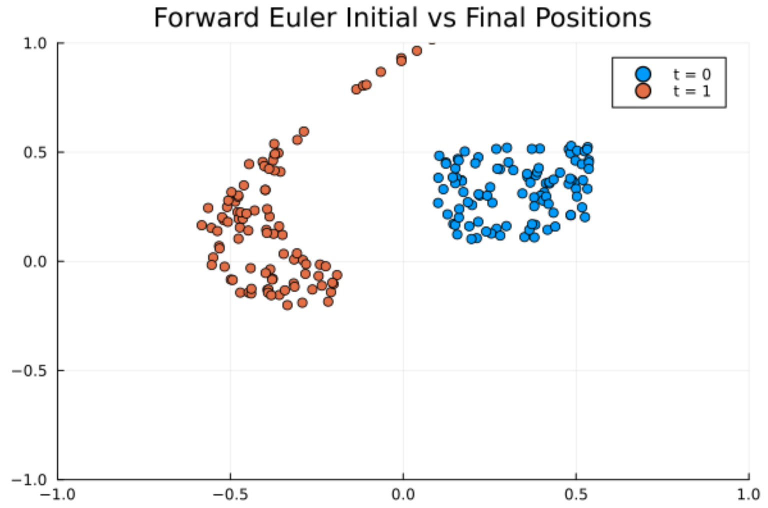

Figure 1: Initial and final particle positions for 1 second time frame using Forward Euler solver.

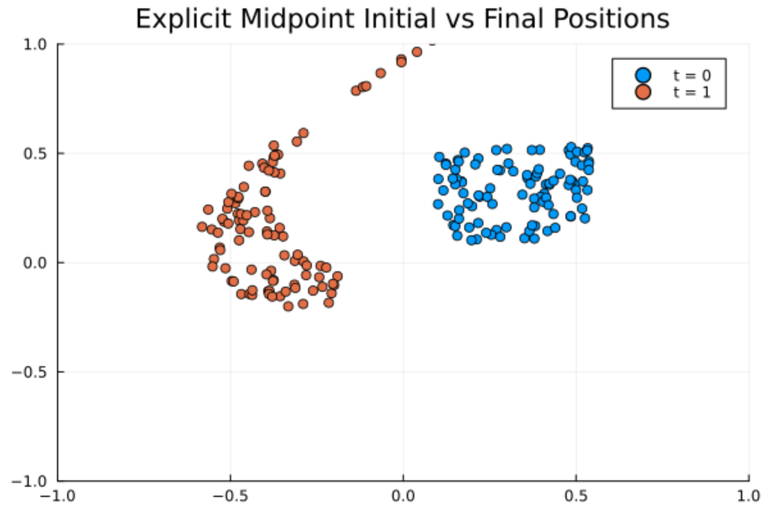

Figure 2: Initial and final particle positions for 1 second time frame using Explicit Midpoint solver.

Figure 3 displays the error of both methods at different time steps. The error is defined as the difference in position at time $= 5$ from the particle positions at time $= 0$. With the velocities given in Equation 1 below, the particle positions at time $= 0$, should be the same at time $= 5$. First, take note that as the time step size decreases, the error of both methods decrease. Additionally, it is seen that the Explicit Midpoint method produces results with lower errors.

$$v_x(x, y, t) = -\sin(\pi y)\cos(\pi t/C)$$
$$v_y(x, y, t) = \sin(\pi x)\cos(\pi t/C)$$

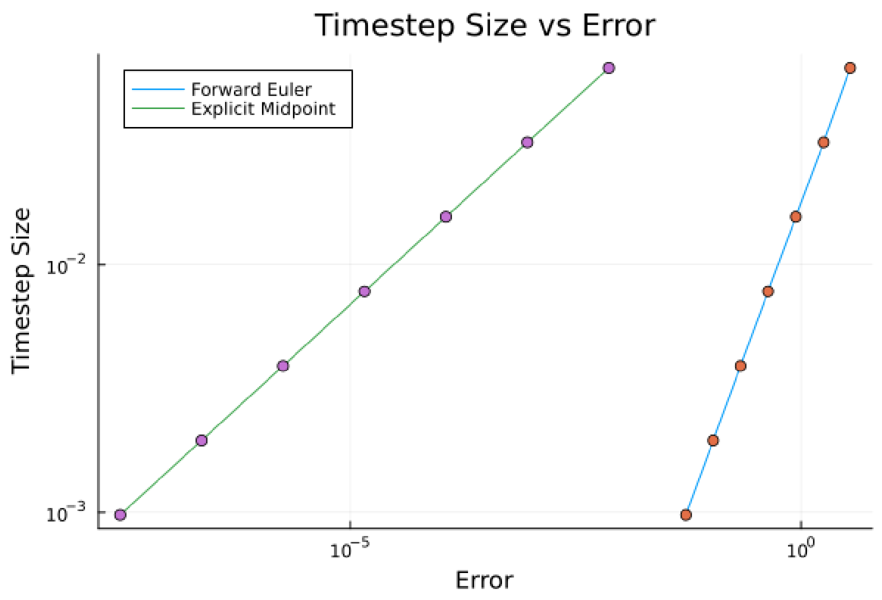

Figure 3: Solution error at varying time-steps.

2 Analysis of The Efficiency of The "rhs!" Function

Listing 1 shows the command to check the type stability of the `rhs!` function using `@code_warntype`. As shown in Listing 2, `rhs!` is type stable.

Listing 1: Type stable check command.

```
num_particles = 100 # number of particles used in simulation
du = zeros(2, num_particles) # array for storing velocity of
   particles, initialized to zero
u = 0.1 .+ 0.45 * rand(2, num_particles) # intitial position of
   particles
parameters = [5] # parameter for specified velocity field
t = 1.324 # randomly chosen point in time
@code_warntype rhs!(du, u, parameters, t) # testing type stability
   with a representative set of inputs defined above
```

Listing 2: Type stable check output.

```
MethodInstance for rhs!(::Matrix{Float64}, ::Matrix{Float64}, ::
   Vector{Int64}, ::Float64)
```

```
  from rhs!(du, u, parameters, t) in Main at c:\Users\lamki\OneDrive
      - Rice University\1st Semester\CAAM 519\caam-419-519-
     submissions\homework-2\part_2.jl:1
Arguments
  #self#::Core.Const(rhs!)
  du::Matrix{Float64}
  u::Matrix{Float64}
  parameters::Vector{Int64}
  t::Float64
Locals
  @_6::Union{Nothing, Tuple{Int64, Int64}}
  N::Int64
  C::Int64
  i::Int64
```

Listing 3 shows the command to check the speed of the `rhs!` function using `@btime`. This command also checks how many allocations the function uses. As shown in Listing 4, `rhs!` is allocation-free.

Listing 3: Type stable check command.

```
num_particles = 100 # number of particles used in simulation
du = zeros(2, num_particles) # array for storing velocity of
   particles, initialized to zero
u = 0.1 .+ 0.45 * rand(2, num_particles) # intitial position of
   particles
parameters = [5] # parameter for specified velocity field
t = 1.324 # randomly chosen point in time
@btime rhs!($du, $u, $parameters, $t)
```

Listing 4: Type stable check output.

```
3.125 μs (0 allocations: 0 bytes)
```

3 Analysis of The Efficiency of The Solver Functions

Listing 5 shows the command to check the type stability of the `solve` function using `@code_warntype`. As shown in Listing 6, `solve` is type stable.

Listing 5: Type stable check command.

```
num_particles = 100 # number of particles used in simulation
u = 0.1 .+ 0.45 * rand(2, num_particles) # intitial position of
   particles
tspan = (0, 5) # randomly chosen time span
dt = 1/512 # randomly chosen time step
```

```
parameters = [5] # parameter for specified velocity field
@code_warntype solve(ForwardEuler(), u, rhs!, tspan, dt, parameters)
```

Listing 6: Type stable check output.

```
MethodInstance for solve(::ForwardEuler, ::Matrix{Float64}, ::typeof
   (rhs!), ::Tuple{Int64, Int64}, ::Float64, ::Vector{Int64})
  from solve(method::ForwardEuler, u0, rhs!, tspan, dt, parameters;
     num_saved_steps) in Main at c:\Users\lamki\OneDrive - Rice
     University\1st Semester\CAAM 519\caam-419-519-submissions\
     homework-2\part_1.jl:3
Arguments
  #self#::Core.Const(solve)
  method::Core.Const(ForwardEuler())
  u0::Matrix{Float64}
  rhs!::Core.Const(rhs!)
  tspan::Tuple{Int64, Int64}
  dt::Float64
  parameters::Vector{Int64}
```

Figure 4 displays the error of both methods at different number of `rhs!` evaluations. The error is defined the same way as mentioned previously. The number of `rhs!` evaluations is found by the number of times the solve function calls `rhs!` during its execution. For a given `dt`, the Explicit Midpoint methods calls the `rhs!` function double the amount the Forward Euler method calls it. Looking at Figure 4, note that as the number of `rhs!` evaluations increases, the error of both methods decrease. Additionally, it is seen that the Explicit Midpoint method produces results with lower errors, similar to what was seen with Figure 3.

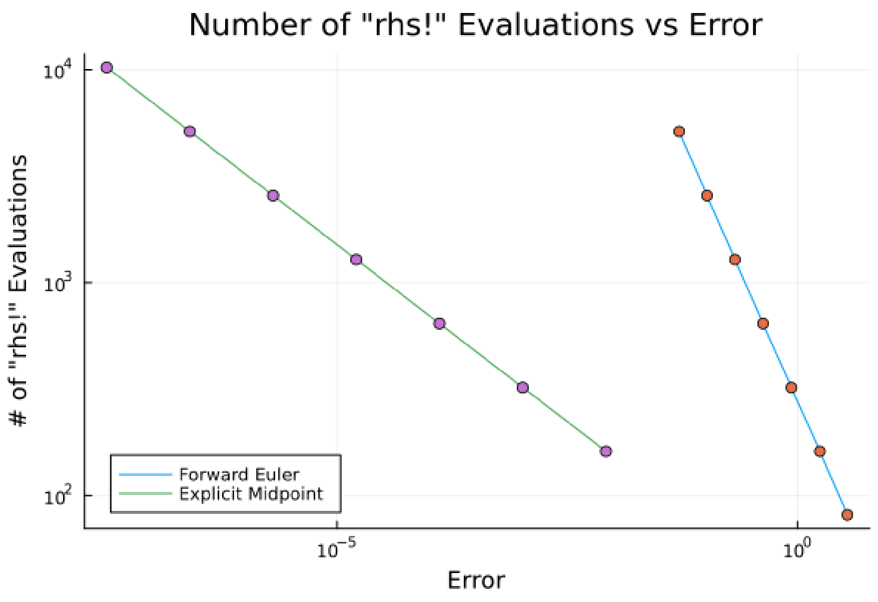

Figure 4: Solution error at varying number of rhs! evaluations.