MEEN 644 - Homework 1

Logan Harbour
January 31, 2019

Problem statement

Consider an oscillation of a simple pendulum of length l and mass m. The equation of motion is described by

$$m\frac{d^2\theta}{dt^2} + \frac{mg\sin\theta}{\ell} = 0\,,$$

where $m = 1$ kg and $\ell = 1$ m.

The pendulum is initially displaced to $20°$, and let go from the rest position to oscillate. Assume that there is no friction and the oscillation is not damped.

(a) (50 points) Starting with $\Delta t = 0.5$ sec as the time step of integration, calculate displacement (radians) and angular velocity in rad/s using the Runge-Kutta Second Order method. Stop your calculation at the end of one cycle.

(b) (20 points) Continue calculations made in (a) by systematically decreasing the step size Δt by half. Declare grid independence when the relative change in displacement and angular velocity with successive time steps vary less than 10^{-4}. Plot your results.

(c) (30 points) Use the Δt corresponding to the grid independent solution (b). Calculate displacement and angular velocity for one cycle using Euler explicit and Euler implicit methods.

Preliminaries

Problem reduction

We define a new variable $\omega = \frac{d\theta}{dt}$ to obtain the reduced system

$$\begin{cases} \frac{d\theta}{dt} = \omega = f_1(t,\mathbf{y})\,, \\ \frac{d\omega}{dt} = -\frac{g}{\ell}\sin\theta = f_2(t,\mathbf{y})\,, \\ \theta(0) = \theta_0 = 20° = \frac{\pi}{9}\,, \\ \omega(0) = \omega_0 = 0\,, \end{cases} \tag{1}$$

where $\mathbf{y} = [\theta,\omega]^T$, $\mathbf{y}_0 = [\theta_0,\omega_0]^T$, $\mathbf{f}(t,\mathbf{y}) = [f_1(t,\mathbf{y}), f_2(t,\mathbf{y})]^T$, and $g = 9.81$.

Discretization

We first define a step size Δt with $t_i = i\Delta t$ and $\mathbf{y}_i = \mathbf{y}(t_i)$. A full period (computed using the small-angle approximation) is $T = 2\pi\sqrt{\frac{\ell}{g}} \approx 2.006$ s, therefore a final time was chosen as 2.0 s to allow for only four timesteps with the intial timestep size of $\Delta t = 0.5$ s.

The discretized equations are as follows.

- Runge-Kutta second order:

$$\mathbf{k}_{1,n} = \mathbf{f}(t_n, \mathbf{y}_n)\,, \tag{2a}$$

$$\mathbf{k}_{2,n} = \mathbf{f}(t_{n+1}, \mathbf{y}_n + \Delta t \mathbf{k}_{1,n})\,, \tag{2b}$$

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \frac{\Delta t}{2}(\mathbf{k}_{1,n} + \mathbf{k}_{2,n}) \tag{2c}$$

- Explicit Euler:

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \Delta t \mathbf{f}(t_n, \mathbf{y}_n)\,, \tag{3}$$

- Implicit Euler:

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \Delta t \mathbf{f}(t_{n+1}, \mathbf{y}_{n+1})\,, \tag{4}$$

  which was solved for $\mathbf{y}_{n+1}$ using a non-linear solver; in specific, the Newton solver `fsolve` that is packaged in `SciPy` with an initial guess of $\mathbf{y}_n$.

Grid independence

First, we define a relative change between successive timestep sizes as

$$\text{relative change} = \frac{|y_{i,n}^{l+1} - y_{i,n}^{l}|}{|y_{i,n}^{l}|}\,,$$

which compares the solution of variable i at time t_n between evaluations where the timestep was at l and $l+1$. For example, for l corresponding to a timestep of $\Delta t = 0.5$ s, $l+1$ would correspond to a timestep of $\Delta t = 0.25$ s. For a timestep with index $l+1$, this value is computed for every variable and every solution time that is common with the solution times of timestep index l. This is therefore an appropriate representation of the change of the solution between successive timesteps as requested, as it takes into account the convergence of the solution over the entire time $t = [0, 2]$ s. We then stop computation when the maximum of all of the relative changes is $< 10^{-4}$.

Implementation

The above we implemented in Python 3 using `NumPy` and `Matplotlib`. The general solver is located in the function `solveODEs` as follows. All of the initial conditions and functions required for this specific problem are located as optional inputs to `solveODEs`.

```python
import numpy as np
from scipy.optimize import fsolve

def solveODEs(N, f=lambda t, y: np.array([y[1], -9.81 * np.sin(y[0])]), tf=2,
              y0=[np.pi / 9, 0], method='RK', tol=None):
    # Initalize timestep size and solution storage
    dt = tf / N
    y = np.zeros((N + 1, len(y0)))
    y[0] = y0
    t = np.linspace(0, dt * N, num=N + 1)

    # Integrate over each timestep
    for n in range(N):
        if method == 'RK':
            k1 = f(t[n], y[n])
            k2 = f(t[n + 1], y[n] + dt * k1)
            y[n + 1] = y[n] + dt * (k1 + k2) / 2
        elif method == 'explicit':
            y[n + 1] = y[n] + dt * f(t[n], y[n])
        elif method == 'implicit':
            func = lambda x: y[n] + dt * f(t[n + 1], x) - x
            y[n + 1] = fsolve(func, y[n])

    # Tolerance given, need to converge
    if tol is not None:
            print('Refining in time...)
        dts, max_changes, last_changes = [], [], []
        y_old = np.copy(y)
        # Refine in time and solve again until converged
        while True:
            N *= 2
            t, y = solvePendulum(N, method=method)

            # Relative change at each one-step-coarser point for each variable
            change = np.abs((y_old[1:] - y[::2][1:]) / (y_old[1:]))
            print('  dt {:.2e} s: max change = {:.2e}'.format(tf / N, np.max(change)))

            # Append for plotting
            dts.append(tf / N)
            max_changes.append(np.max(change, axis=0))
            last_changes.append(change[-1])

            # Max is less than tolerance: done
            if np.max(change) < tol:
                return t, y, N, dts, np.array(max_changes), np.array(last_changes)

            # Didn't converge, copy previous solution
            y_old = np.copy(y)

    # Standard run without convergence, return just t and y
    return t, y
```

Results

Part a: RK-2, small time step ($\Delta t = 0.5$ s)

The results for part a follow below in Table 1 and Figure 1

Table 1: The solution at each time step for part a.

t	θ	ω
sec	rad	rad / s
0.0	0.34907	0.00000
0.5	−0.07034	−1.67760
1.0	−0.82296	0.42971
1.5	0.29094	3.62894
2.0	1.75366	0.81515

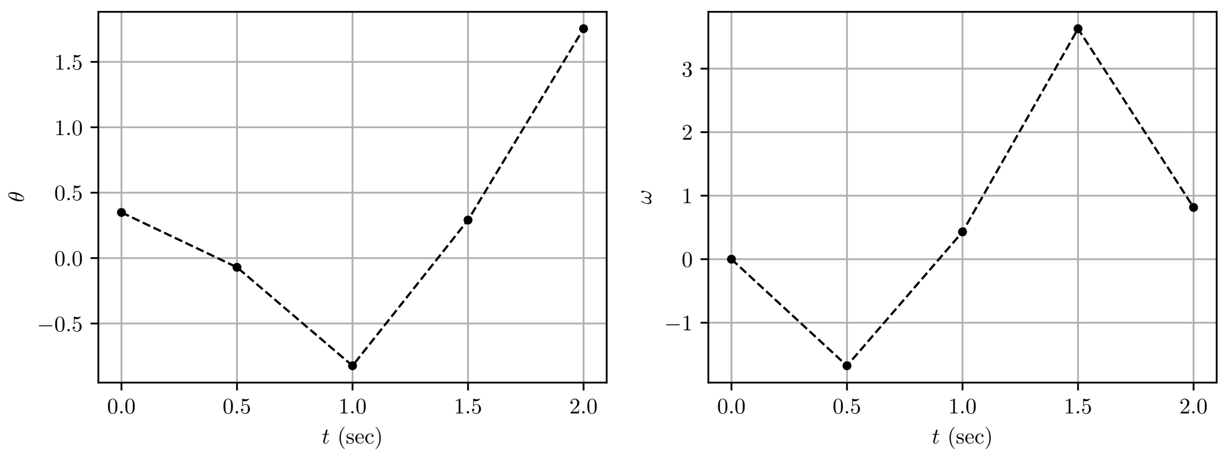

Figure 1: The plotted solution for part a.

Part b: Grid independence with RK-2

The plot of the relative changes for part b follow in Figure 2. In this plot, "all coarse timesteps" is the maximum relative change over all coarse timesteps for a given variable, and "$t = 2$ s" is the relative change in the solution at the final time for a given variable.

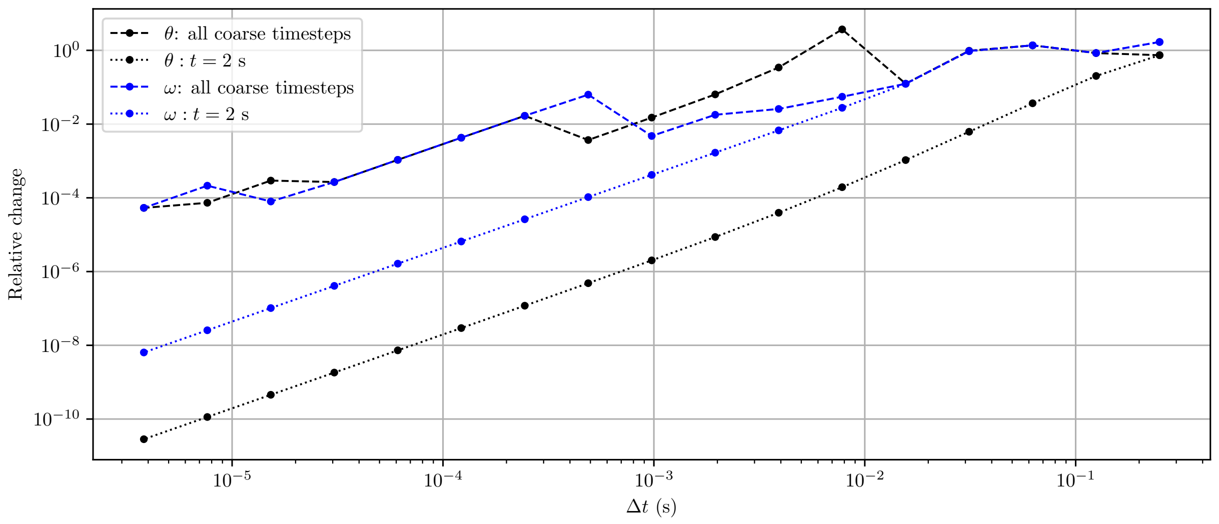

Figure 2: The plotted relative changes used to determine grid independence for part b.

The convergence for a single point in time behaves significantly better than taking the maximum over each solution time. We expect this because the order of accuracy should decrease by one from a point value to a maximum.

Part c: All methods

The explicit and implicit Euler methods were also utilized at the previously determined grid independent timestep size ($\Delta t = 3.8 \times 10^{-6}$ s) for part c. The resulting solutions are plotted below in Figure 3.

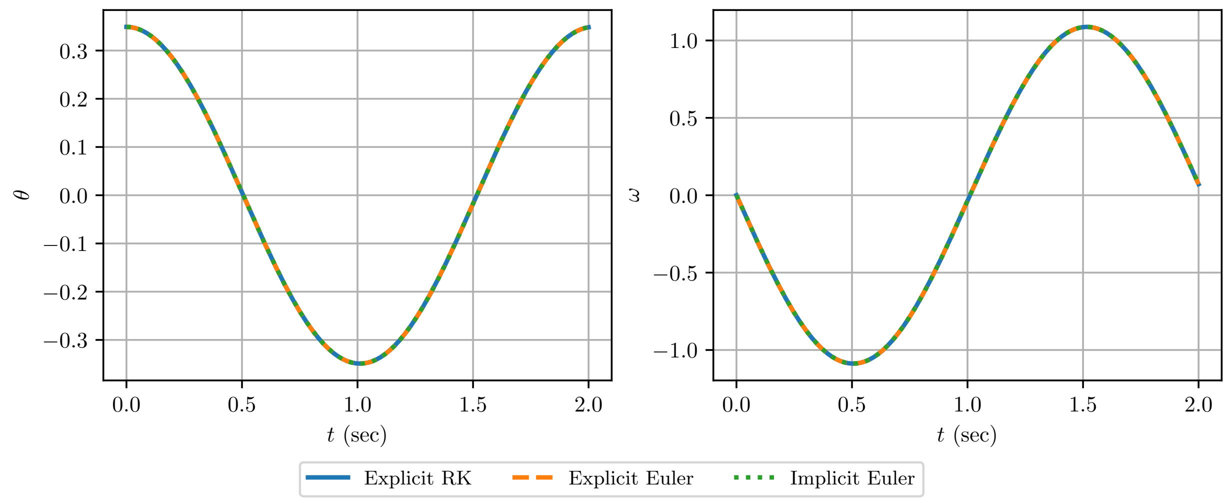

Figure 3: The plotted solutions with $\Delta t = 3.8 \times 10^{-6}$ s (grid-independent) compared among all methods for part c.