TFY4510 Specialization Project in Physics

Journal for Arne Magnus Tveita Løken, fall 2017

1 Weeks 34–36

1.1 Preliminaries

I've spent the first few weeks of the project work studying the dynamics, i.e., the transport properties, of an analytically known velocity field. The velocity field is a simplified dynamical model of a periodic, two dimensional double gyre system, first described by Shadden et al. (2005). It is defined on the domain $[0,2]\times[0,1]$, and described mathematically by the Lagrange stream function

$$\psi(\mathbf{x},t) = A\sin\big(\pi f(x,t)\big)\sin(\pi y) \tag{1}$$

where

$$f(x,t) = a(t)x^2 + b(t)x \tag{2a}$$
$$a(t) = \epsilon\sin(\omega t) \tag{2b}$$
$$b(t) = 1 - 2\epsilon\sin(\omega t) \tag{2c}$$

and the parameters A, ϵ and ω adjust the properties of the system.

The stream function description only applies to incompressible, i.e., divergence-free, two-dimensional flows. Per definition, the velocity field relates to the stream function in the following way:

$$\mathbf{V} = \begin{bmatrix} u \\ v \end{bmatrix} \tag{3a}$$
$$u = -\frac{\partial\psi}{\partial y} \tag{3b}$$
$$v = \frac{\partial\psi}{\partial x} \tag{3c}$$

which, for the stream function given by eqn. (1), yields

$$u = -\pi A\sin\big(\pi f(x,t)\big)\cos(\pi y) \tag{4a}$$
$$v = \pi A\cos\big(\pi f(x,t)\big)\sin(\pi y)\frac{\partial f(x,t)}{\partial x} \tag{4b}$$

with $f(x,t)$ is given by eqn. (2).

1.2 Considerations regarding simulation of the system dynamics

To begin with, i.e., at time $t = 0$, I covered the domain $[0, 2] \times [0, 1]$ with a mesh of equidistant fluid elements, approximated as point masses on the grid points. Because the velocity field is known analytically, simulating how the fluid elements are transported in time amounted to solving a set of independent ordinary differential eqn.s (ODEs) in time. A myriad of numerical solution techniques for ODE systems exist, which is reflected in my project problem theme, namely "Sensitivity to numerical integration scheme in calculation of transport barriers". I simulated the transport of the fluid elements in a set amount of time using a variety of solution schemes, with intention to estimate how the precision of the fluid elements' trajectories is affected by the choice of numerical scheme.

Put simply, all numerical integration schemes have their strengths and weaknesses. Simple schemes involve few computational steps, at the cost of precision. Similarly, schemes with a high degree of precision are more complex, and typically require a larger number of intermediary calculations in order to arrive at a solution. Basically, my project work is centered around the tradeoff between the precision of numerical solutions of dynamic systems, and the time we have to wait for the calculations to finish.

Naturally, devising a way to estimate the degree of precision of the numerical solutions produced by the different integration schemes is crucial. One glaring hindrance lies in the fact that the fluid elements' trajectories can be expressed analytically, but only in terms of integrals that can only be evaluated numerically. Thus, simply using a solution from a higher order numerical scheme – using a small time step (for fixed step size integrators) or low tolerance levels (for adaptive step size integrators) – as reference strikes me as the most practical approach.

1.3 Numerical integrators implemented so far

So far, I've implemented the four most common single-step explicit Runge-Kutta methods. They are summarized in table 1.

Table 1: The most common explicit Runge-Kutta integrators, and their precision, here indicated by their leading error terms.

(Explicit) integrator	Leading error term
Euler's method	$\mathcal{O}(\Delta t)$
Heun's method	$\mathcal{O}(\Delta t^2)$
Kutta's method	$\mathcal{O}(\Delta t^3)$
RK4 method	$\mathcal{O}(\Delta t^4)$

Høyere ordens numerisk metode må brukes som referanse, ettersom posisjonen til enhver partikkel ikke kan uttrykkes analytisk (på lukket form). Adaptiv-tidssteg-metoder krever en viss finesse, ettersom hver enkelt bane vil trenge sitt eget tidssteg osv.

1.4 Error estimation, various integrators

Using the RK4 method with timestep $\Delta t = 0.001$ as reference, I have simulated the fluid flow until $t = 5$ with the integration schemes named in table 1 for time step sizes $\Delta t \in \{0.1, 0.01, 0.001\}$ and computed the error in the FTLE field. The errors of each direct solver are presented in tables 2–5.

Introduce the concept of a (Finite-Time) Lyapunov Exponent field in an introductory chapter.
Explicitly provide the Runge-Kutta methods, e.g. Euler, Heun, Kutta and RK4, in the theory section. State the leading error term.
Investigate array slicing and utilization of adaptive timestep integrators as reference solution.
Compare numerical error of the different integrators with the reference solution – e.g. logarithmic plot of the error as function of the time step, with reference slopes $\sim h^n$.

(Dormand and Prince, 1986)

Table 2: Euler, $t = 5$

Euler \ Δt	0.1	0.01	0.001
Avg. abs. err.	$5 \cdot 10^{-3}$	$4 \cdot 10^{-4}$	$4 \cdot 10^{-5}$
Max. abs. err.	$7 \cdot 10^{-2}$	$1 \cdot 10^{-2}$	$1 \cdot 10^{-3}$

Table 3: Heun, $t = 5$

Heun \ Δt	0.1	0.01	0.001
Avg. abs. err.	$1 \cdot 10^{-4}$	$1 \cdot 10^{-6}$	$1 \cdot 10^{-8}$
Max. abs. err.	$5 \cdot 10^{-3}$	$5 \cdot 10^{-5}$	$5 \cdot 10^{-7}$

Table 4: Kutta, $t = 5$

Kutta \ Δt	0.1	0.01	0.001
Avg. abs. err.	$4 \cdot 10^{-6}$	$4 \cdot 10^{-9}$	$4 \cdot 10^{-12}$
Max. abs. err.	$2 \cdot 10^{-4}$	$2 \cdot 10^{-7}$	$3 \cdot 10^{-10}$

2 Week 37

...was mostly a writeoff, due to job interview preparations.

3 Weeks 38–39

4 Week 40

Several trial runs for integrating trajectories in parallel using adaptive timestep integrators suggest that attempting steps with all particles each time is generally more efficient in terms of computation time than the alternative of only stepping forwards in time with the trajectories which are yet to reach the end point. Although there is a benefit of approximately 20% for first and second order accurate adaptive integrators, there is no significant difference for third order methods, whereas higher order methods require about 30% *more* computation time when only stepping forwards with trajectories which are not at the end point. Alas, I see no real benefit to applying masks to the integrated trajectories, taking into account the more complex and less comprehensible code it requires.

Table 5: RK4, $t = 5$

RK4 \ Δt	0.1	0.01
Avg. abs. err.	$7 \cdot 10^{-8}$	$7 \cdot 10^{-12}$
Max. abs. err.	$4 \cdot 10^{-6}$	$4 \cdot 10^{-10}$

Table 6: General Butcher Tableau representation for Runge-Kutta methods

$$
\begin{array}{c|ccccc}
0 & & & & & \\
c_2 & a_{21} & & & & \\
c_3 & a_{31} & a_{32} & & & \\
\vdots & \vdots & & \ddots & & \\
c_s & a_{s1} & a_{s2} & \dots & a_{s,s-1} & \\
\hline
 & b_1 & b_2 & \dots & b_{s-1} & b_s
\end{array}
$$

Table 7: Butcher tableau for the Dormand-Prince 5(4) adaptive timestep integrator. The first row of b-coefficients yield the 5th order solution, while the second row yields the 4th order interpolant. See Prince and Dormand (1981).

$$
\begin{array}{c|ccccccc}
0 & & & & & & & \\
\frac{1}{5} & \frac{1}{5} & & & & & & \\
\frac{3}{10} & \frac{3}{40} & \frac{9}{40} & & & & & \\
\frac{4}{5} & \frac{44}{45} & -\frac{56}{15} & \frac{32}{9} & & & & \\
\frac{8}{9} & \frac{19372}{6561} & -\frac{25360}{2187} & \frac{64448}{6561} & -\frac{212}{769} & & & \\
1 & \frac{9017}{3168} & -\frac{355}{33} & \frac{46732}{5247} & \frac{49}{176} & -\frac{5103}{18656} & & \\
1 & \frac{35}{384} & 0 & \frac{500}{1113} & \frac{125}{192} & -\frac{2187}{6784} & \frac{11}{84} & \\
\hline
 & \frac{35}{384} & 0 & \frac{500}{1113} & \frac{125}{192} & -\frac{2187}{6784} & \frac{11}{84} & \\
 & \frac{5179}{57600} & 0 & \frac{7571}{16695} & \frac{393}{640} & -\frac{92097}{339200} & \frac{187}{2100} & \frac{1}{40}
\end{array}
$$

Second order forward difference approximation of first derivative:

$$
\frac{\mathrm{d}f(x)}{\mathrm{d}x} \approx \frac{-f(x+2h) + 4f(x+h) - 3f(x)}{2h} \tag{5}
$$

Second order backward difference approximation of first derivative:

$$\frac{\mathrm{d}f(x)}{\mathrm{d}x} \approx \frac{3f(x) - 4f(x-h) + f(x-2h)}{2h} \tag{6}$$

Second order forward difference approximation of second derivative:

$$\frac{\mathrm{d}^2 f(x)}{\mathrm{d}x^2} \approx \frac{2f(x) - 5f(x+h) + 4f(x+2h) - f(x+3h)}{h^2} \tag{7}$$

Second order backward difference approximation of second derivative:

$$\frac{\mathrm{d}^2 f(x)}{\mathrm{d}x^2} \approx \frac{2f(x) - 5f(x-h) + 4f(x-2h) - f(x-3h)}{h^2} \tag{8}$$

5 Week 41

When using NumPy or SciPy interpolation, e.g. by use of RectBivariateSpline, the indexing is not what we expect from Python. This is due to how the NumPy meshgrids share indexing pattern with Matlab. In particular, this means that, when generating the interpolation object, the correct call structure is as follows:

```python
from scipy.interpolate import RectBivariateSpline as RBSpline
import numpy as np

# ----------------------------------------------------- #
# Function and variables defined somewhere in the above #
# ----------------------------------------------------- #

x = np.linspace(dx/2, Nx-dx/2, Nx)
y = np.linspace(dy/2, Ny-dy/2, Ny)

z = foo(x, y)

spline = RBSpline(y, x, z)
```

rather than

```python
spline = RBSpline(x, y, z) # <-- Incorrect
```

Similarly, the correct call structure when evaluating the interpolated object is

```python
x2 = np.linspace(x_min, x_max, NI)
y2 = np.linspace(y_min, y_max, NJ)

z2 = spline.ev(y2, x2)
```

rather than

```python
z2 = spline.ev(x2, y2) # <-- Incorrect
```

References

Dormand, J. and Prince, P. (1986). A reconsideration of some embedded Runge-Kutta formulae. *Journal of Computational and Applied Mathematics*, 15(2):203–211.

Prince, P. and Dormand, J. (1981). High order embedded Runge-Kutta formulae. *Journal of Computational and Applied Mathematics*, 7(1):67–75.

Shadden, S. C., Lekien, F., and Marsden, J. E. (2005). Definition and properties of Lagrangian coherent structures from finite-time Lyapunov exponents in two-dimensional aperiodic flows. *Physica D: Nonlinear Phenomena*, 212(3):271–304.