

# Lecture 05 - Time integration

## SSP and symplectic methods

Georgii Oblapenko

### 1 Objectives

In this lecture, we will learn about advanced methods for solving time-dependent PDEs and ODEs. In particular, we will be discussing

1. Stability
2. Monotonicity
3. “Structure” preservation

### 2 ODEs: why?

Want to solve ODE

$$d\frac{u}{dt} = f(u, t), \quad u \in \mathbb{R}^n, \quad t > 0, \quad u(0) = u^0.$$

ODEs come from

- direct description of system (i.e. biology, chemical reactors)
- semi-discretization of system (i.e.  $u$  — vector of cell values from a FVM discretization, RHS are the fluxes + sources)

#### **i** Note

**Method of lines:** discretize in space, solve in time. Take PDE

$$\partial_t u(x, t) + \sum_{\alpha} \partial_{\alpha} a_{\alpha} u(x, t) = f(u(x, t), t),$$

discretize in space (FD/FVM/FEM/DG):  $u(x_i, t) = u_i(t)$ , discretize operators in terms of  $\{u_j\}$ , obtain ODEs

$$\partial_t \mathbf{u} + \mathbf{A} \mathbf{u}(t) = \mathbf{f}(t).$$

## 2.1 PDE discretization - remark

! Alternatives to Method of Lines

Are all discretizations of time-dependent PDEs in time done via Method of Lines?

Lax-Wendroff: couples time and space discretization. For linear advection

$$\partial_t u + a_0 \partial_x u = 0, \quad C = \frac{a_0 \Delta t}{\Delta x}$$

$$u_j^{n+1} = u_j^n + \frac{C}{2}(1+C)u_{j-1}^n + (1-C^2)u_j^n - \frac{C}{2}(1+C)u_{j+1}^n$$

## 2.2 ODE solvers: computational cost

## 2.3 ODE solvers: accuracy

**Example 1:** you want to model a system of chemical reactions. Which method would you choose?

1. Method with  $\mathcal{O}(\Delta t^4)$  error, but still might lead to negative densities
2. Method with  $\mathcal{O}(\Delta t^2)$  error, but preserves non-negativity of densities

**Example 2:** you want to model a system of particles. Which method would you choose?

1. Method with  $\mathcal{O}(\Delta t^4)$  error, but does not conserve energy
2. Method with  $\mathcal{O}(\Delta t^2)$  error, but conserves energy

We therefore will focus on different (not all!) notions of error/accuracy and talk about some problem-specific methods.

## 3 Recap of some standard methods

Let's recall some textbook methods first.

$$t_n = n\Delta t, \quad u^n = u(t_n).$$

### 3.1 Forward Euler

Simplest approach:

$$u^{n+1} = u^n + \Delta t f(u^n, t^n)$$

💡 Properties?

$\mathcal{O}(\Delta t)$  error; what else?

### 3.2 RK4

$$k_1 = \Delta t f(u^n, t^n),$$

$$k_2 = \Delta t f(u^n + \frac{1}{2}k_1, t^n + \frac{1}{2}\Delta t),$$

$$k_3 = \Delta t f(u^n + \frac{1}{2}k_2, t^n + \frac{1}{2}\Delta t),$$

$$k_4 = \Delta t f(u^n + k_3, t^n + \Delta t),$$

$$u^{n+1} = u^n + \frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4$$

💡 Properties?

$\mathcal{O}((\Delta t)^4)$  error; what else?

## 4 Stability, monotonicity

Let's discuss stability and monotonicity. These are not identical properties!

### 4.1 Stability

Linear analysis:

$$u' = -\alpha u, \alpha \in \mathbb{C}, \operatorname{Re}(\alpha) > 0 \Rightarrow u(t) = u_0 e^{-\alpha t}.$$

Expect numerically for  $\alpha : \operatorname{Re}(\alpha) < 0$ :

$$\lim_{n \rightarrow \infty} u^n = 0$$

I.e. solution does not “blow up”.

**i** Stability

For forward Euler:  $\Delta t < \frac{2}{\alpha}$

### 4.2 Monotonicity

$$u(t) = u_0 e^{-\alpha t} : t_2 > t_1 \Rightarrow |u(t_1)| > |u(t_2)|$$

Expect numerically:

$$|u^n| \leq |u^{n-1}|$$

### **i** Monotonicity

For forward Euler:  $\Delta t < \frac{1}{\alpha}$

Example where monotonicity may be desirable:

$$\mathbf{u} = (\rho, \rho v_1, \rho v_2, \rho e),$$

if  $\rho$  or  $\rho e$  become negative - solver might crash or produce meaningless results!

### **i** Generalized stability/monotonicity

Consider  $G(u^n)$  instead of  $|u^n|$ , where  $G$  is a 1) norm 2) semi-norm or 3) convex functional

## 4.3 Numerical example

$$u' = -\alpha u, \alpha > 0, u(0) = u_0$$

### **i** Note

For forward Euler: Monotonicity:  $\Delta t < \frac{1}{\alpha}$ , stability:  $\Delta t < \frac{2}{\alpha}$ .

```
1 import numpy as np
2 from matplotlib import pyplot as plt
3
4
5 def forward_euler(alpha, u0, times):
6     u = np.zeros(len(times))
7     u[0] = u0
8     for (i, t) in enumerate(times[1:]):
9         u[i+1] = u[i] + (times[i+1] - times[i]) * (-alpha * u[i])
10    return u
11
12
13 fig, ax = plt.subplots(figsize=(6.5, 4.2))
14
15 u0 = 2.0
16
17 alpha = 10.0
18 times = np.linspace(0, 1, 100)
19
20 times_fe1 = np.linspace(0, 1, 5)
21 dt1 = times_fe1[1] - times_fe1[0]
22
23 times_fe2 = np.linspace(0, 1, 10)
24 dt2 = times_fe2[1] - times_fe2[0]
25
26 times_fe3 = np.linspace(0, 1, 12)
27 dt3 = times_fe3[1] - times_fe3[0]
```

```

28
29 dts = r"$\Delta t$"
30 oas = r"$1/\alpha$"
31 tas = r"$2/\alpha$"
32
33
34 ax.plot(times, u0 * np.exp(-alpha * times),
35         'k-', label=f"Analytical")
36 ax.plot(times_fe1, forward_euler(alpha, u0, times_fe1),
37         color='tab:red', label=f"F.E. ({dts} = {dt1:.3f}; {tas} = {2.0/alpha:.2f})")
38 ax.plot(times_fe2, forward_euler(alpha, u0, times_fe2),
39         color='tab:purple', label=f"F.E. ({dts} = {dt2:.3f}; {oas} = {1.0/alpha:.2f})")
40 ax.plot(times_fe3, forward_euler(alpha, u0, times_fe3),
41         color='tab:blue',
42         label=f"F.E. ({dts} = {dt3:.3f})")
43
44 ax.legend(fontsize=12)
45 ax.set_xlabel("t", fontsize=10)
46 ax.set_ylabel("u", fontsize=10)
47 ax.tick_params(axis='both', labelsize=8)
48
49 ax.set_yscale("log")
50 plt.show()

```

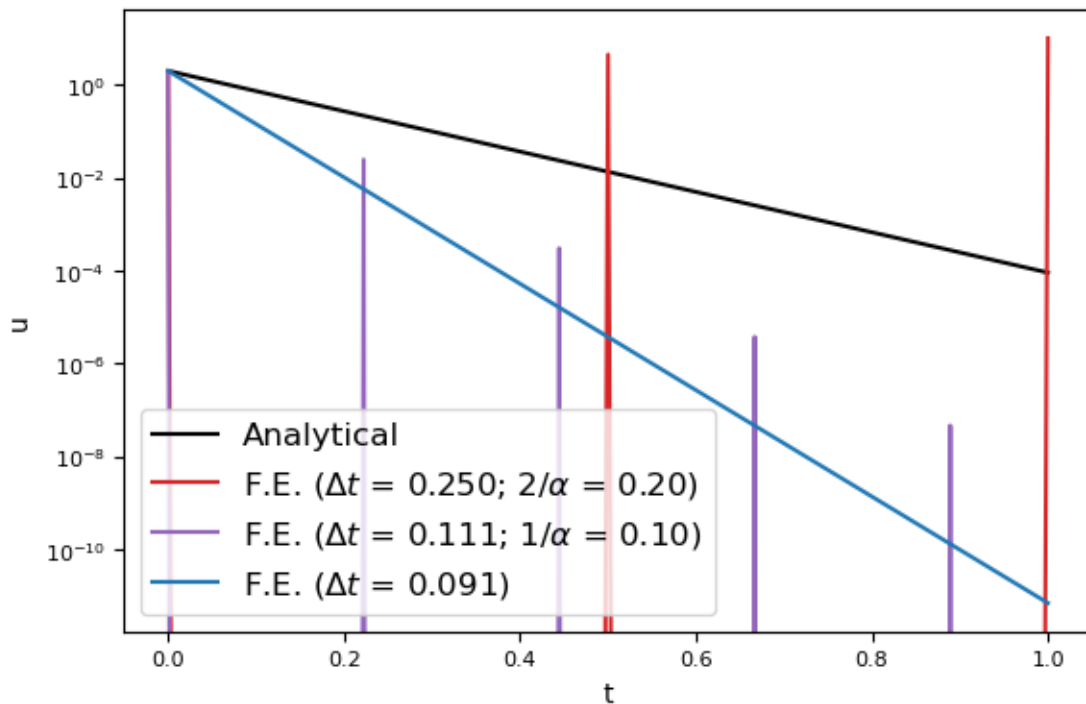


Figure 1: Euler's method for  $du/dt = -au$

#### 4.4 Region of absolute stability

Re-write discretization of ODE ( $-\alpha \rightarrow \lambda$ )

$$u'(t) = \lambda u, \quad u^n = R(\lambda \Delta t) u^{n-1},$$

$R(z), z \in \mathbb{C}$  is called the stability function.

Region of absolute stability:

$$\mathcal{S} = \{z \in \mathbb{C} \mid |R(z)| < 1\}$$

## 4.5 Region of absolute stability: examples

Forward Euler:

$$R(z) = 1 + z$$

Backward (implicit Euler):

$$u^{n+1} = u^n + \Delta \lambda t u^{n+1} \implies R(z) = \frac{1}{1 + z}$$

RK4:

$$R(z) = 1 + z + \frac{1}{2!}z^2 + \frac{1}{3!}z^3 + \frac{1}{4!}z^4$$

## 5 SSPRK methods

Strong stability-preserving (Total Variation Diminishing) Runge-Kutta methods:

- ensure strong stability/monotonicity of solutions **and** have higher order than forward Euler:  $\|u^{n+1}\| \leq \|u^n\|$  (or a generalized  $G(u)$ ) for all  $n$
- *usually* written as convex combination of forward Euler steps

### 5.1 SSPRK methods: formulation

An  $m$ -stage method for  $u' = f(u)$  is written as (*Shu-Osher form*)

$$u^{(0)} = u^n,$$

$$u^{(i)} = \sum_{j=0}^{i-1} (\alpha_{ij} u^{(j)} + \Delta t \beta_{ij} f(u^{(j)})), \quad i = 1, \dots, m; \quad \alpha_{ij} \geq 0,$$

$$u^{n+1} = u^{(m)}.$$

## 5.2 Theorem

### i Theorem

If  $\|u^{n+1}\| \leq \|u^n\|$  in the forward Euler method with  $\Delta t \leq \Delta_{FE}$ , then the SSPRK method is stable for  $\Delta t \leq c_{SSP} \Delta_{FE}$ .

- $c_{SSP}$  is called the **SSP coefficient**,  $c_{SSP} = \min \frac{\alpha_{ij}}{\beta_{ij}}$
- Most research focused on finding SSPRK methods with largest  $c_{SSP}$
- $\Delta_{FE}$  - depends on spatial discretization,  $c_{SSP}$  - on time-stepping scheme

## 5.3 SSPRK examples

SSPRK2:

$$\begin{aligned}u^{(1)} &= u^n + \Delta t f(u^n), \\u^{(n+1)} &= \frac{1}{2}u^n + \frac{1}{2}u^{(1)} + \frac{1}{2}\Delta t f(u^{(1)}).\end{aligned}$$

SSPRK3:

$$\begin{aligned}u^{(1)} &= u^n + \Delta t f(u^n), \\u^{(2)} &= \frac{3}{4}u^n + \frac{1}{4}u^{(1)} + \frac{1}{4}\Delta t f(u^{(1)}), \\u^{(n+1)} &= \frac{1}{3}u^n + \frac{2}{3}u^{(2)} + \frac{2}{3}\Delta t f(u^{(2)}).\end{aligned}$$

## 5.4 SSP coefficient, effective CFL

### i Question

What does  $c_{SSP} = 1$  (for example, SSPRK3) mean?

Makes more sense to talk about **effective CFL**:  $c_{eff} = c_{SSP}/k$ ,  $k$  - number of evaluations of RHS. For example, SSPRK3 has  $c_{eff} = 1/3$  (but third-order convergence!).

### ! Stability vs. monotonicity vs. accuracy

SSPRK methods may have same restrictions on  $\Delta t$  to achieve monotonicity, but

- are more accurate
- have a larger stability region (monotone  $\neq$  stable!)

## 6 Stability for oscillatory problems

Consider harmonic oscillator

$$\frac{d}{dt} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} u_2 \\ -\omega^2 u_1 \end{pmatrix}$$

Analytical solution:

$$\begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = c_1 \begin{pmatrix} \cos(\omega t) \\ -\omega \sin(\omega t) \end{pmatrix} + c_2 \begin{pmatrix} \frac{1}{\omega} \sin(\omega t) \\ \cos(\omega t) \end{pmatrix}$$

### 6.1 Forward Euler

Energy is given by

$$E = (\omega^2 u_1^2 + u_2^2)/2$$

and is conserved for the analytical solution.

#### **i** Energy blow-up

For forward Euler we can show that  $E^{n+1} = \frac{1}{2}(1 + \omega^2 \Delta t^2) (\omega^2 (u_1^n)^2 + (u_2^n)^2)$ . Eigenvalues of harmonic oscillator system are purely imaginary ( $\pm i\omega$ ) and do not lie in stability region of forward Euler.

### 6.2 Forward Euler: analysis

Let's write the forward Euler scheme as a matrix equation

$$\begin{pmatrix} u_1^{n+1} \\ u_2^{n+1} \end{pmatrix} = \begin{pmatrix} 1 & \Delta t \\ -\omega^2 \Delta t & 1 \end{pmatrix} \begin{pmatrix} u_1^n \\ u_2^n \end{pmatrix} =: J \mathbf{u}$$

#### **i** Determinant of $J$

Note that  $\det(J) = (1 + \omega^2 \Delta t^2)$  is exactly the energy growth factor!

Re-write integration step via **growth factor**  $\gamma$ :

$$u_1^{n+1} = \gamma u_1^n, u_2^{n+1} = \gamma u_2^n \implies \begin{pmatrix} \gamma - 1 & -\Delta t \\ \omega^2 \Delta t & \gamma - 1 \end{pmatrix} \begin{pmatrix} u_1^n \\ u_2^n \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

#### **i** Exercise

Non-trivial solution  $\implies \det(LHS) = 0$ ; can show that  $|\gamma| > 1$ , i.e. method is unstable.

### 6.3 Example - harmonic oscillator

```

1 def forward_euler_generic(rhs, u0, times, omega):
2     u = np.zeros((len(times), len(u0)))
3     u[0,:] = u0
4     for (i, t) in enumerate(times[1:]):
5         u[i+1,:] = u[i,:] + (times[i+1] - times[i]) * rhs(u[i,:], omega)
6     return u
7
8 def ssprk22_generic(rhs, u0, times, omega):
9     u = np.zeros((len(times), len(u0)))
10    u[0,:] = u0
11    for (i, t) in enumerate(times[1:]):
12        u_1 = u[i,:] + (times[i+1] - times[i]) * rhs(u[i,:], omega)
13        u[i+1,:] = 0.5 * u[i,:] + 0.5 * u_1 + 0.5 * (times[i+1] - times[i]) * rhs(u_1, omega)
14    return u
15
16 def rk2_generic(rhs, u0, times, omega):
17     # Ralston's method
18     u = np.zeros((len(times), len(u0)))
19     u[0,:] = u0
20     for (i, t) in enumerate(times[1:]):
21         dt = (times[i+1] - times[i])
22         k1 = rhs(u[i,:], omega)
23         k2 = rhs(u[i,:] + 0.75 * k1 * dt, omega)
24         u[i+1,:] = u[i,:] + dt * (0.25 * k1 + 0.75 * k2)
25     return u
26
27 def harmonic_rhs(u, omega):
28     return np.array([u[1], -omega**2 * u[0]])
29
30 def energy(u, omega):
31     return 0.5 * (u[:,1]**2 + omega**2 * u[:,0]**2)
32
33 fig, axes = plt.subplots(figsize=(12.0, 4.2), ncols=2)
34
35 ax = axes[0]
36 ax_e = axes[1]
37
38 t_max = 5.0
39 omega = 3.0
40 times = np.linspace(0, t_max, 100)
41
42 u0 = [0.5, 0.5]
43
44 analytical = np.array([u0[0] * np.cos(omega * times) + (u0[0]/omega) * np.sin(omega * times),
45                        -u0[0] * omega * np.sin(omega * times) + u0[0] * np.cos(omega * times)])
46
47
48
49 times_fe1 = np.linspace(0, t_max, 100)
50 dt1 = times_fe1[1] - times_fe1[0]

```

```

51 sol_fe1 = forward_euler_generic(harmonic_rhs, u0, times_fe1, omega)
52
53 times_fe2 = np.linspace(0, t_max, 400)
54 dt2 = times_fe2[1] - times_fe2[0]
55 sol_fe2 = forward_euler_generic(harmonic_rhs, u0, times_fe2, omega)
56
57 times_ssprk2 = np.linspace(0, t_max, 80)
58 dt3 = times_ssprk2[1] - times_ssprk2[0]
59 sol_ssprk2 = ssprk22_generic(harmonic_rhs, u0, times_ssprk2, omega)
60
61 times_rk2 = np.linspace(0, t_max, 80)
62 dt4 = times_rk2[1] - times_rk2[0]
63 sol_rk2 = rk2_generic(harmonic_rhs, u0, times_rk2, omega)
64
65
66 ax.plot(sol_fe1[:,0], sol_fe1[:,1],
67         color='tab:red', label=f"F.E. dt = {dt1:.3f}")
68
69 ax.plot(sol_fe2[:,0], sol_fe2[:,1],
70         color='tab:purple', label=f"F.E. dt = {dt2:.3f}")
71
72 ax.plot(sol_ssprk2[:,0], sol_ssprk2[:,1],
73         color='tab:blue', label=f"SSPRK2 dt = {dt3:.3f}")
74
75 ax.plot(sol_rk2[:,0], sol_rk2[:,1],
76         color='tab:green', label=f"RK2 dt = {dt4:.3f}")
77
78
79 ax.plot(analytical[:,0], analytical[:,1],
80         'k-', label=f"Analytical")
81
82 ax_e.plot(times, energy(analytical, omega), 'k')
83 ax_e.plot(times_fe1, energy(sol_fe1, omega), 'tab:red')
84 ax_e.plot(times_fe2, energy(sol_fe2, omega), 'tab:purple')
85 ax_e.plot(times_ssprk2, energy(sol_ssprk2, omega), 'tab:blue')
86 ax_e.plot(times_rk2, energy(sol_rk2, omega), 'tab:green')
87
88 ax.legend(fontsize=12)
89 ax.set_xlabel("x(t)", fontsize=10)
90 ax.set_ylabel("v(t)", fontsize=10)
91 ax.tick_params(axis='both', labelsize=8)
92
93
94 print(f"dE F.E. (dt={dt1:.3f}) at t={t_max}: {(energy(analytical, omega)[-1] - energy(sol_fe1,
95 print(f"dE F.E. (dt={dt2:.3f}) at t={t_max}: {(energy(analytical, omega)[-1] - energy(sol_fe2,
96 print(f"dE SSPRK2 (dt={dt3:.3f}) at t={t_max}: {(energy(analytical, omega)[-1] - energy(sol_ss
97 print(f"dE RK2 (dt={dt4:.3f}) at t={t_max}: {(energy(analytical, omega)[-1] - energy(sol_rk2,
98
99 ax_e.set_yscale("log")
100 plt.show()

```

dE F.E. (dt=0.051) at t=5.0: -1.057e+01  
dE F.E. (dt=0.013) at t=5.0: -9.460e-01  
dE SSPRK2 (dt=0.063) at t=5.0: -3.250e-02  
dE RK2 (dt=0.063) at t=5.0: 3.461e-01

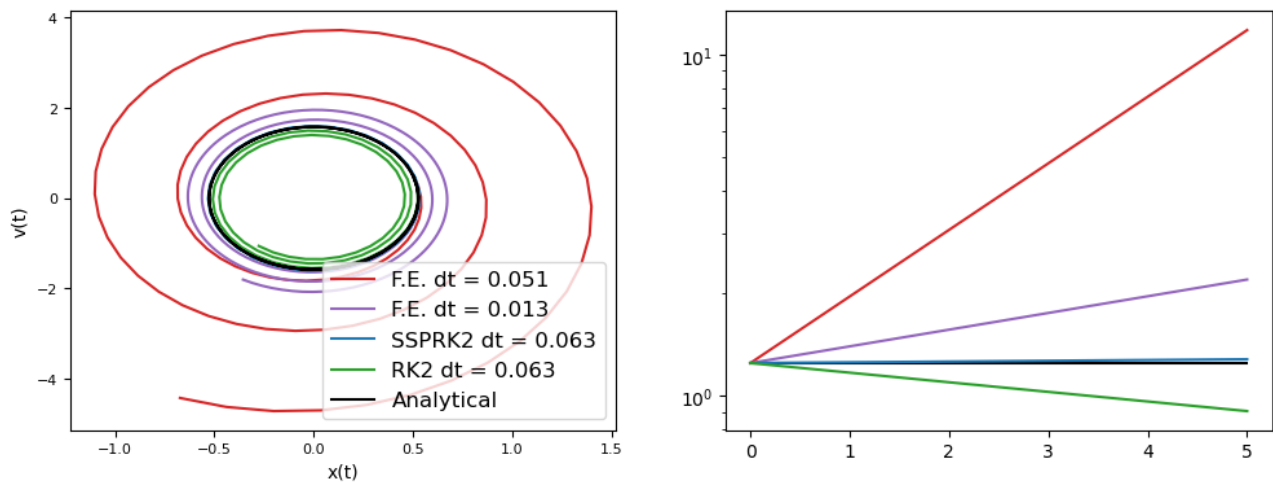


Figure 2: Harmonic oscillator phase space (left) and energy (right),  $\omega=3$

### 💡 Question

Why does SSPRK2 still cause a growth of energy?

## 7 Stiffness

We don't go into detail about stiff ODEs in this lecture, but still mention some definitions.

- **Definition 1** Step size required for stability is much smaller than the step size required for accuracy
- **Definition 2** Computational cost of an explicit method becomes larger than that of an implicit one

Arise in biology, chemistry, low-Mach number compressible flows.

## 8 Symplectic integrators

Consider ODE system given by

$$\frac{dq_i}{dt} = \frac{\partial H}{\partial p_i}, \quad \frac{dp_i}{dt} = -\frac{\partial H}{\partial q_i},$$

where  $H$  is the **Hamiltonian** of the system.

- Why do we are interested in specific methods for such systems?
- What can these systems describe?

### i Naming

$q_i$  are called “generalized coordinates”,  $p_i$  - “generalized momenta”

## 8.1 Example: interacting particles

$$H(p, q) = \frac{1}{2} \frac{1}{m} \sum_i p_i^2 + \sum_{i < j} V(|q_i - q_j|),$$

$V$  - *interaction potential*. Examples: gravity, electrostatic potential (both  $V(r)$  are proportional to  $1/r$ ). Potential gives force:

$$F = -\nabla V.$$

### 💡 Hamiltonian of interacting particles?

What does Hamiltonian resemble?

Hamiltonian is kinetic + potential energy! Time integration of particle movement should conserve total energy.

## 8.2 First integrals

Consider the system

$$\frac{dy}{dt} = f(y).$$

$I(y)$  is called a **first integral** of a system, if

$$\frac{dI}{dt} = \frac{dI}{dy} f(y) = 0.$$

### i H as first integral

The Hamiltonian is a first integral of a Hamiltonian system!

## 8.3 Symplectic integrators

Define **mapping**  $\psi$ :

$$\psi : (\mathbf{q}(t), \mathbf{p}(t)) \rightarrow (\mathbf{q}(t + \Delta t), \mathbf{p}(t + \Delta t))$$

**Theorem** Symplectic iff.

$$(D\psi)^T \begin{pmatrix} 0 & I \\ -I & 0 \end{pmatrix} (D\psi) = \begin{pmatrix} 0 & I \\ -I & 0 \end{pmatrix},$$

$D\psi$  is the Jacobi matrix of the mapping ( $\psi$  - “canonical transformation”).

## 8.4 Phase space volume-preserving integrators

A weaker condition:  $\mu(\psi(\mathcal{S})) \equiv \mu(\mathcal{S})$ , where  $\mu$  is a measure (volume). This implies that the Jacobi matrix  $D\psi$  has determinant 1.

What does this give?

- Ensures no spurious sources/sinks/attractors
- Ensures that time-averaging remains equivalent to ensemble averaging for a certain class of systems (*ergodic systems*): important for particle methods!

**i** Difference between symplectic and volume-preserving mappings:

The [non-squeezing theorem](#) explains the difference nicely and shows that symplecticity is a stronger condition.

## 8.5 Leapfrog integrator

If  $H(\mathbf{p}, \mathbf{q}) = T(\mathbf{p}) + V(\mathbf{q})$  (*separable Hamiltonian*), then we can integrate with **leapfrog**:

$$\begin{aligned}\mathbf{p}^{n+1/2} &= \mathbf{p}^{n-1/2} - \Delta t \nabla V(\mathbf{q}^n), \\ \mathbf{q}^{n+1} &= \mathbf{q}^n + \Delta t \nabla T(\mathbf{p}^{n+1/2}),\end{aligned}$$

Leapfrog is **symplectic** and has error  $\mathcal{O}((\Delta t)^2)$ .

**i** Harmonic oscillator

One can show that stability for harmonic oscillator requires  $\Delta t \leq 2/\omega$ ; this can be done using the growth factor analysis (keeping in mind that a step of  $\Delta t/2$  corresponds to a growth factor of  $\sqrt{\gamma}$  and re-writing  $u_2^{n+1/2}$  and  $u_2^{n-1/2}$  in terms of  $u_2^n$ ).

## 8.6 Verlet integrator

What if we need  $\mathbf{q}, \mathbf{p}$  at the same point in time?

$$\begin{aligned}\mathbf{p}^{n+1/2} &= \mathbf{p}^n - \frac{\Delta t}{2} \nabla V(\mathbf{q}^n), \\ \mathbf{q}^{n+1} &= \mathbf{q}^n + \Delta t \nabla T(\mathbf{p}^{n+1/2}), \\ \mathbf{p}^{n+1} &= \mathbf{p}^{n+1/2} - \frac{\Delta t}{2} \nabla V(\mathbf{q}^{n+1}),\end{aligned}$$

**i** Naming

Often this is called “Leapfrog in kick-drift-kick” form or “Störmer-Verlet”.

## 8.7 Example - harmonic oscillator revisited

```
1 def verlet_generic(rhs, u0, times, omega):
2     u = np.zeros((len(times), len(u0)))
3     u[0,:] = u0
4     for (i, t) in enumerate(times[1:]):
5         dt = (times[i+1] - times[i])
6         p_hs = u[i,1] + 0.5 * dt * rhs(u[i,:], omega)[1]
7         u[i+1,0] = u[i,0] + dt * p_hs
8         u[i+1,1] = p_hs + 0.5 * dt * rhs(u[i+1,:], omega)[1]
9     return u
10
11 fig, axes = plt.subplots(figsize=(12.0, 4.2), ncols=2)
12
13 ax = axes[0]
14 ax_e = axes[1]
15
16
17 times_verlet = np.linspace(0, t_max, 20)
18 dt5 = times_verlet[1] - times_verlet[0]
19 sol_verlet = verlet_generic(harmonic_rhs, u0, times_verlet, omega)
20 sol_ssprk2_new = ssprk2_generic(harmonic_rhs, u0, times_verlet, omega)
21
22 ax.plot(sol_fe2[:,0], sol_fe2[:,1],
23         color='tab:purple', label=f"F.E. dt = {dt2:.3f}")
24
25 ax.plot(sol_ssprk2[:,0], sol_ssprk2[:,1],
26         color='tab:blue', label=f"SSPRK2 dt = {dt3:.3f}")
27
28 # ax.plot(sol_ssprk2_new[:,0], sol_ssprk2_new[:,1],
29 #         color='tab:orange', label=f"SSPRK2 dt = {dt5:.3f}")
30
31 ax.plot(sol_verlet[:,0], sol_verlet[:,1],
32         color='tab:red', label=f"Verlet dt = {dt5:.3f}; 2/omega={2/omega:.3f}")
33
34 ax.plot(analytical[:,0], analytical[:,1],
35         'k-', label=f"Analytical")
36
37 ax_e.plot(times, energy(analytical, omega), 'k')
38 ax_e.plot(times_fe2, energy(sol_fe2, omega), 'tab:purple')
39 ax_e.plot(times_ssprk2, energy(sol_ssprk2, omega), 'tab:blue')
40 # ax_e.plot(times_verlet, energy(sol_ssprk2_new, omega), 'tab:orange')
41 ax_e.plot(times_verlet, energy(sol_verlet, omega), 'tab:red')
42
43 ax.legend(fontsize=12, loc="upper right")
44 ax.set_xlabel("x(t)", fontsize=10)
45 ax.set_ylabel("v(t)", fontsize=10)
46 ax.tick_params(axis='both', labelsize=8)
47
48
```

```

49 print(f"dE F.E. (dt={dt2:.3f}) at t={t_max}: {(energy(analytical, omega)[-1] - energy(sol_fe2,
50 print(f"dE SSPRK2 (dt={dt3:.3f}) at t={t_max}: {(energy(analytical, omega)[-1] - energy(sol_ss
51 print(f"dE SSPRK2 (dt={dt5:.3f}) at t={t_max}: {(energy(analytical, omega)[-1] - energy(sol_ss
52 print(f"dE Verlet (dt={dt5:.3f}) at t={t_max}: {(energy(analytical, omega)[-1] - energy(sol_ve
53
54 # ax_e.set_yscale("log")
55 plt.show()

```

```

dE F.E. (dt=0.013) at t=5.0: -9.460e-01
dE SSPRK2 (dt=0.063) at t=5.0: -3.250e-02
dE SSPRK2 (dt=0.263) at t=5.0: -6.023e+00
dE Verlet (dt=0.263) at t=5.0: 4.697e-02

```

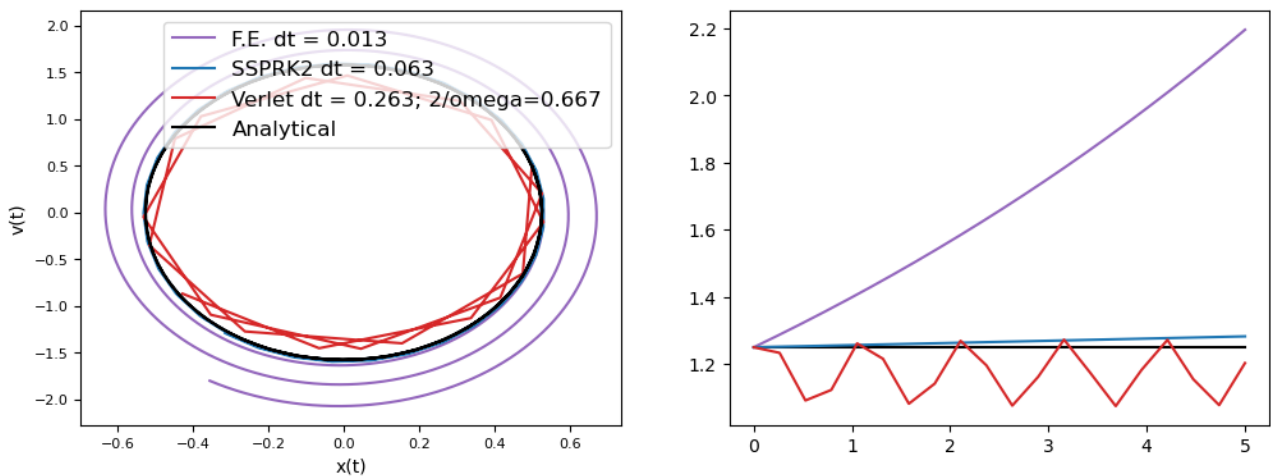


Figure 3: Harmonic oscillator phase space (left) and energy (right),  $\omega=3$

## 9 Code

- [DifferentialEquations.jl](#) - huge amount of methods available
- [Comparison of various methods \(2019!!!\)](#) (see also [summary table](#))

## 10 Summary

Methods not covered in this lecture but also useful/actively developed:

- IMEX (IMPLICIT/EXPLICIT): separate RHS into stiff and non-stiff parts, treat them implicitly and explicitly, respectively.
  - “Implicit-explicit Runge-Kutta methods for time-dependent partial differential equations”, Ascher, Ruth et al.
  - “Additive Runge-Kutta schemes for convection-diffusion-reaction equations”, Kennedy & Carpenter
- MPRK (Modified Patankar Runge-Kutta) for production-destruction systems. Ensure non-negativity of densities.

- “On order conditions for modified Patankar–Runge–Kutta schemes”, Kopecz & Meister
- Implementation available in [PositiveIntegrators.jl](#)
- Explicit integrators for particles with velocity-dependent force terms (for example, Lorentz force)
  - “On the Boris solver in particle-in-cell simulation”, Zenitani & Umeda

## 10.1 Exercises

1. Derive the stability polynomial  $R(z)$  for the RK4 method
2. Prove the SSP property of a method in Shu-Osher form (hint: use the TVD property of forward Euler, the triangle inequality and look at the definition of  $c_{SSP}$ )
3. Derive energy growth rate for SSPRK2 for the harmonic oscillator
4. Show that forward Euler is **always** unstable for the harmonic oscillator via growth factor analysis
5. Show that the forward Euler method is not a symplectic method

## 10.2 Questions

1. What is the difference between monotonicity and stability of a method?
2. Why are implicit methods oftentimes more computationally expensive?
3. What is the difference between a symplectic and a phase space volume-preserving integrator? Which condition is stronger? Why?
4. What two formulations of explicit symplectic integrators do you know and what is the difference between them?

## 11 References

1. S. Gottlieb, On High Order Strong Stability Preserving Runge–Kutta and Multi Step Time Discretizations
2. Kubatko, Yeager et al., Optimal Strong-Stability-Preserving Runge–Kutta Time Discretizations for Discontinuous Galerkin Methods
3. A. Giorgilli, Notes on Hamiltonian Dynamical Systems