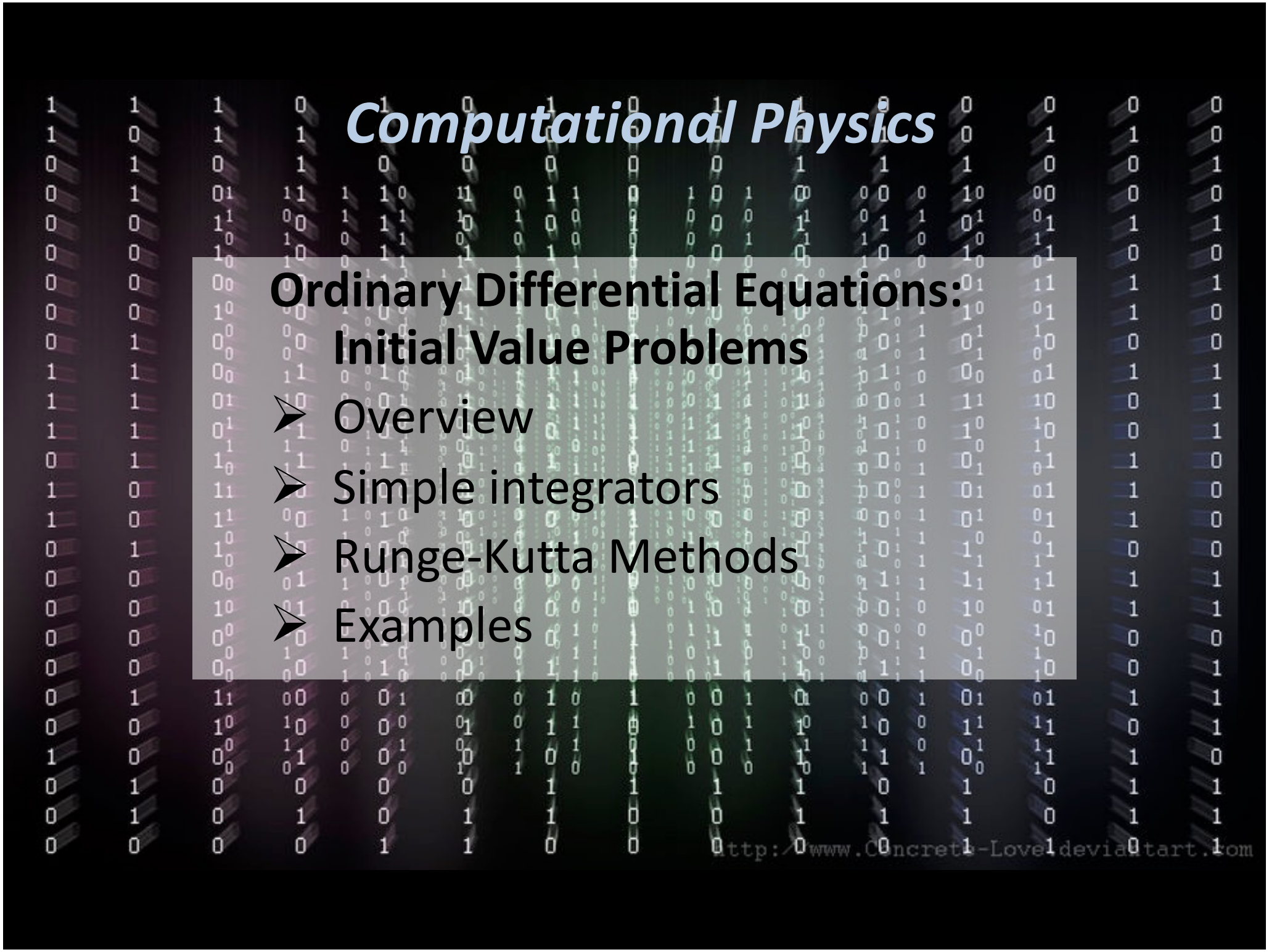

Computational Physics
Ordinary Differential Equations:
Initial Value Problems
Overview
Simple integrators
Runge-Kutta Methods
Examples

Initial value problem

We already introduced explicit, implicit, and midpoint Euler methods. Now we introduce a more general context.

- Initial value problem:

$$\begin{cases} \dot{y}(t) = f(y,t) \ , \\ y(0) = y_0 \ , \end{cases} \qquad \dot{y} \equiv \frac{dy}{dt}$$

assumes an explicit form, meaning in G(.) is analytically invertible.

$$\begin{cases} G(\dot{y}) = f(y,t) \\ y(0) = y_0 \ , \end{cases}$$

E.g. differential equations of form $\dot{y} + \log(\dot{y}) = 1$ are not explicit.

Higher order differential equations

- the above Eq. is a **first order** differential equation in y
- However, every explicit differential equation of order n can be decomposed in n coupled first order differential equations:

$$y^{(n)} = f(t; y, \dot{y}, \ddot{y}, \ldots, y^{(n-1)})$$

$$\Leftrightarrow \qquad \begin{aligned} \dot{y}_1 &= y_2 \ , \\ \dot{y}_2 &= y_3 \ , \\ &\vdots \quad \vdots \\ \dot{y}_{n-1} &= y_n \ , \\ \dot{y}_n &= f(t, y_1, y_2, \ldots, y_n) \end{aligned}$$

i.e. the differential equation is explicit if explicit in $y^{(n)}$

Numerical solutions

Even seemingly simple differential equations cannot be solved analytically and need numerical treatment, e.g.:

$$\dot{y} = t^2 + y^2$$

is difficult to solve and not well posed: no initial condition given
But: **Numerical methods are no excuse for poor analysis**
(Sometimes numerical solutions are used even when an analytical solution is possible.)

Next, we discretize t again as $t_n=t_0+n\Delta t$ and $y_n=y(t_n)$, i.e.

$$\dot{y}_n = f(y_n, t_n)$$

Simple Integrators

Integration the above differential equation over a Δt interval:

$$y_{n+1} = y_n + \int_{t_n}^{t_{n+1}} \mathrm{d}t' f[y(t'), t']$$

Note: This is still exact!

Approximations for the integral:

- **rectangular rule** gives:

$$y_{n+1} = y_n + f(y_n, t_n)\Delta t + \mathcal{O}(\Delta t^2)$$

Explicit or forward Euler method

Note, $\mathcal{O}(\Delta t^2)$ means the following terms in this Taylor series are of order Δt^2 or higher.

compare to Big- $\mathcal{O}$ -notation

Informal usage in Computer Science *(and not in the mathematical sense):*

An algorithm can be said to exhibit a growth rate on the order of a mathematical function if beyond a certain input size n, the function f(n) times a positive constant provides an upper bound or limit for the run-time of that algorithm.
In other words, for a given input size n greater than some n_0 and a constant c, the running time of that algorithm will never be larger than c × f(n). This concept is frequently expressed using Big- $\mathcal{O}$ -notation. For example, since the run-time of *insertion sort* grows quadratically as its input size increases, *insertion sort* can be said to be of order $\mathcal{O}(n^2)$.

Big- $\mathcal{O}$ -notation is a convenient way to express the worst-case scenario for a given algorithm, although it can also be used to express the average-case — for example, the worst-case scenario for *quicksort* is $\mathcal{O}(n^2)$, but the average-case run-time is $\mathcal{O}(n \log n)$.

See also ***Computational complexity theory***

more simple integrators

- the backward rectangular rule gives:

$$y_{n+1} = y_n + f(y_{n+1}, t_{n+1})\Delta t + \mathcal{O}(\Delta t^2)$$

implicit or backward Euler method/scheme

→ needs solving for y_{n+1}

- central rectangular rule

$$y_{n+1} = y_n + f(y_{n+\frac{1}{2}}, t_{n+\frac{1}{2}})\Delta t + \mathcal{O}(\Delta t^3)$$

or $$y_{n+1} = y_{n-1} + 2f(y_n, t_n)\Delta t + \mathcal{O}(\Delta t^3)$$

using two time steps

leap-frog routine or **Störmer-Verlet** method

using $$y_{n+\frac{1}{2}} \approx \frac{y_n + y_{n+1}}{2}$$ gives the **implicit midpoint rule**

...

Using the trapezoidal rule gives:

$$y_{n+1} = y_n + \frac{\Delta t}{2}\left[f(y_n, t_n) + f(y_{n+1}, t_{n+1})\right] + \mathcal{O}(\Delta t^3)$$

Crank-Nicolson or trapezoidal method (implicit)

- The leap-frog method is a multi-step method, since it involves values at three different times to evolve the solution in time
- All others are single-step methods (need only t_n and t_{n+1}).

Improvements of the simple methods

- **Taylor series methods**: Use more terms in the Taylor expansion of y_{n+1}.
- **Linear Multi-Step methods**: Use data from previous time steps y_k, k<n in order to cancel terms in the truncation error.
- **Runge-Kutta** method: Use intermediate points within one time step.

Taylor series methods

Taylor series of y_{n+1} around y_n:

$$y_{n+1} = y_n + \Delta t\, \dot{y}_n + \frac{\Delta t^2}{2}\ddot{y}_n + \mathcal{O}(\Delta t^3)$$

which gives:

$$y_{n+1} = y_n + \Delta t f(y_n, t_n) + \frac{\Delta t^2}{2}\ddot{y}_n + \mathcal{O}(\Delta t^3)$$

having still error Δt^2, but with

$$\ddot{y}_n = \frac{\mathrm{d}}{\mathrm{d}t} f(y_n, t_n) = \dot{f}(y_n, t_n) + f'(y_n, t_n)\dot{y}_n = \dot{f}(y_n, t_n) + f'(y_n, t_n) f(y_n, t_n)$$

we get

$$y_{n+1} = y_n + \Delta t f(y_n, t_n) + \frac{\Delta t^2}{2}\left[\dot{f}(y_n, t_n) + f'(y_n, t_n) f(y_n, t_n)\right] + \mathcal{O}(\Delta t^3)$$

with error Δt^3

$$f' \equiv \frac{df(y,t)}{dy}$$

Linear multi-step methods

A *k*-th order linear multi-step method is defined by the approximation

$$y_{n+1} = \sum_{j=0}^{k} a_j y_{n-j} + \Delta t \sum_{j=0}^{k+1} b_j f(y_{n+1-j}, t_{n+1-j})$$

coefficients a_j and b_j have to be determined in such a way that the local truncation error is reduced

For example, the **Adams-Bashford** method

$$y_{n+1} = y_n + \frac{\Delta t}{2}\left[3f(y_n, t_n) - f(y_{n-1}, t_{n-1})\right]$$

or the **second order rule** (backward differentiation formula)

$$y_{n+1} = \frac{1}{3}\left[4y_n - y_{n-1} + \frac{\Delta t}{2} f(y_{n+1}, t_{n+1})\right]$$

- multi-step methods are often based on the interpolation of previously computed values y_k by Lagrange polynomials

...

Note $$y_{n+1} = \sum_{j=0}^{k} a_j y_{n-j} + \Delta t \sum_{j=0}^{k+1} b_j f(y_{n+1-j}, t_{n+1-j})$$

is explicit for $b_0=0$ and implicit for $b_0\neq 0$.

In many numerical realizations one combines implicit and explicit multi-step methods in such a way that the explicit result with $b_0=0$ is used as a guess to solve the implicit equation with $b_0\neq 0$.

→ the explicit method predicts the value y_{n+1} and the implicit method corrects it.

Such methods yield very good results and are commonly referred to as ***predictor–corrector methods***.

Runge-Kutta methods

- In contrast to multi-step methods, Runge-Kutta methods improve the accuracy by calculating intermediate grid-points within the interval $[t_n, t_{n+1}]$.
- Note, the central rectangular rule approximation results in such a method, since the function value $y_{n+1/2}$ at the grid-point $t_{n+1/2}=t_n+\Delta t/2$ is used:

$$y_{n+1} = y_n + f(y_{n+\frac{1}{2}}, t_{n+\frac{1}{2}})\Delta t + \mathcal{O}(\Delta t^3)$$

- Next, find approximations for $y_{n+1/2}$. Using explicit Euler gives:

$$y_{n+\frac{1}{2}} = y_n + \frac{\Delta t}{2}\dot{y}_n = y_n + \frac{\Delta t}{2} f(y_n, t_n)$$

- resulting in:

$$y_{n+1} = y_n + f\left[y_n + \frac{\Delta t}{2} f(y_n, t_n), t_n + \frac{\Delta t}{2}\right]\Delta t + \mathcal{O}(\Delta t^2)$$

explicit midpoint rule

...

Using instead the average approximation, results in the **implicit midpoint rule**

$$y_{n+1} = y_n + f\left(\frac{y_n + y_{n+1}}{2}, t_n + \frac{\Delta t}{2}\right)\Delta t + \mathcal{O}(\Delta t^2)$$

both have error $\mathcal{O}(\Delta t^2)$

Algorithmic form

Introduce variables Y_i

→ Explicit Euler:

$$Y_1 = y_n \ ,$$

$$y_{n+1} = y_n + f(Y_1, t_n)\Delta t$$

solve for Y_1

→ Implicit Euler:

$$Y_1 = y_n + f(Y_1, t_{n+1})\Delta t$$

$$y_{n+1} = y_n + f(Y_1, t_{n+1})\Delta t$$

→ Crank-Nicolson:

$$Y_1 = y_n \ ,$$

$$Y_2 = y_n + \frac{\Delta t}{2}\left[f(Y_1, t_n) + f(Y_2, t_{n+1})\right]$$

solve for Y_2

$$y_{n+1} = y_n + \frac{\Delta t}{2}\left[f(Y_1, t_n) + f(Y_2, t_{n+1})\right]$$

...

- explicit midpoint rule

$$Y_1 = y_n \ ,$$

$$Y_2 = y_n + \frac{\Delta t}{2} f\left(Y_1, t_n + \frac{\Delta t}{2}\right)$$

$$y_{n+1} = y_n + \frac{\Delta t}{2} f\left(Y_2, t_n + \frac{\Delta t}{2}\right)$$

- implicit midpoint rule

$$Y_1 = y_n + \frac{\Delta t}{2} f\left(Y_1, t_n + \frac{\Delta t}{2}\right)$$

$$y_{n+1} = y_n + \Delta t f\left(Y_1, t_n + \frac{\Delta t}{2}\right)$$

These are examples of Kunge-Kutta methods

general *d*-stage Runge-Kutta (RK) method

$$Y_i = y_n + \Delta t \sum_{j=1}^{d} a_{ij} f\left(Y_j, t_n + c_j \Delta t\right) , \qquad i = 1, \ldots, d$$

$$y_{n+1} = y_n + \Delta t \sum_{j=1}^{d} b_j f\left(Y_j, t_n + c_j \Delta t\right) .$$

Fully determined by coefficients a_{ij}, b_j, c_j,
where **A**=$\{a_{ij}\}$ is a dxd matrix, **b**=$\{b_j\}$ and **c**=$\{c_j\}$ d-dimensional vectors.
Useful way to describe RK methods by Butcher tableaus:

$$\begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \ldots & a_{1d} \\ c_2 & a_{21} & a_{22} & \ldots & a_{2d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_d & a_{d1} & a_{d2} & \ldots & a_{dd} \\ \hline & b_1 & b_2 & \ldots & b_d \end{array}$$

Butcher tableaus

explicit methods methods have zeros on and above the diagonal of **A**

Explicit Euler

$$\begin{array}{c|c} 0 & 0 \\ \hline & 1 \end{array}$$

Implicit Euler

$$\begin{array}{c|c} 1 & 1 \\ \hline & 1 \end{array}$$

Crank-Nicolson

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & \frac{1}{2} & \frac{1}{2} \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$

Explicit Midpoint

$$\begin{array}{c|cc} 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$

Implicit Midpoint

$$\begin{array}{c|c} \frac{1}{2} & \frac{1}{2} \\ \hline & 1 \end{array}$$

RK4 method

Using the general RK scheme one can develop arbitrarily precise methods

Most famous:

- four stage RK: RK4 or e-RK-4
- explicit

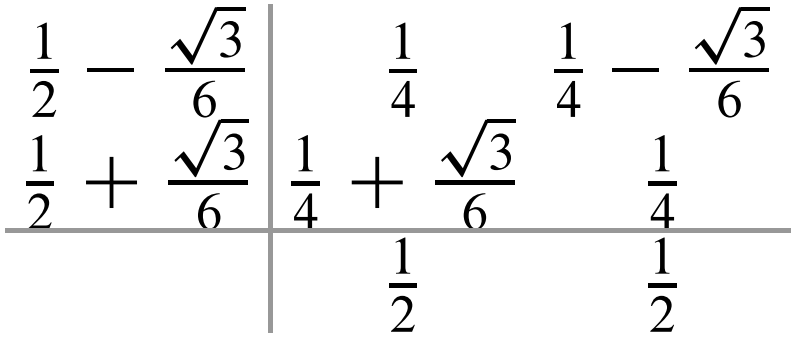
$$
\begin{aligned}
Y_1 &= y_n, \\
Y_2 &= y_n + \frac{\Delta t}{2} f(Y_1, t_n) \ , \\
Y_3 &= y_n + \frac{\Delta t}{2} f\left(Y_2, t_n + \frac{\Delta t}{2}\right) \ , \\
Y_4 &= y_n + \Delta t f\left(Y_3, t_n + \frac{\Delta t}{2}\right) \ , \\
y_{n+1} &= y_n + \frac{\Delta t}{6}\left[f(Y_1, t_n) + 2f\left(Y_2, t_n + \frac{\Delta t}{2}\right) \right. \\
&\qquad \left. + 2f\left(Y_3, t_n + \frac{\Delta t}{2}\right) + f(Y_4, t_n) \right] \ .
\end{aligned}
$$

0	0	0	0	0
$\frac{1}{2}$	$\frac{1}{2}$	0	0	0
$\frac{1}{2}$	0	$\frac{1}{2}$	0	0
1	0	0	1	0
	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{6}$

is analog to Simpson rule for integration

...

Another popular method is defined by



is related to Gauss-Legendre 2-point method.

book by Press et al:
For many scientific users, fourth-order Runge-Kutta is not just the first word on ODE integrators, but the last word as well. In fact, you can get pretty far on this old workhorse, especially if you combine it with an adaptive step-size algorithm. Keep in mind, however, that the old workhorse's last trip may well take you to the poorhouse: Bulirsch-Stoer or predictor-corrector methods can be very much more efficient for problems where high accuracy is a requirement. Those methods are the high-strung racehorses. Runge-Kutta is for ploughing the fields.

The Kepler Problem, revisited

Before, we did not use real equations of motion. In fact these were first order equations, based on energy considerations. Still produce correct trajectories with correct initial conditions.

Here we start with the Hamilton function:

$$H(p,q) = \frac{1}{2}\left(p_1^2 + p_2^2\right) - \frac{1}{\sqrt{q_1^2 + q_2^2}}$$

resulting in Hamilton's equations of motion:

$$\dot{p}_1 = -\nabla_{q_1} H(p,q) = -\frac{q_1}{(q_1^2 + q_2^2)^{\frac{3}{2}}}$$

$$\dot{p}_2 = -\nabla_{q_2} H(p,q) = -\frac{q_2}{(q_1^2 + q_2^2)^{\frac{3}{2}}}$$

$$\dot{q}_1 = \nabla_{p_1} H(p,q) = p_1 \ ,$$

$$\dot{q}_2 = \nabla_{p_2} H(p,q) = p_2 \ .$$

discretization: $q_i^n \equiv q_i(t_n)$

$$p_i^n \equiv p_i(t_n) \text{ for } i = 1, 2$$

...

Explicit Euler

$$p_1^{n+1} = p_1^n - \frac{q_1^n \Delta t}{[(q_1^n)^2 + (q_2^n)^2]^{\frac{3}{2}}}$$

$$p_2^{n+1} = p_2^n - \frac{q_2^n \Delta t}{[(q_1^n)^2 + (q_2^n)^2]^{\frac{3}{2}}}$$

$$q_1^{n+1} = q_1^n + p_1^n \Delta t \ ,$$

$$q_2^{n+1} = q_2^n + p_2^n \Delta t \ .$$

Implicit Euler

$$p_1^{n+1} = p_1^n - \frac{q_1^{n+1} \Delta t}{[(q_1^{n+1})^2 + (q_2^{n+1})^2]^{\frac{3}{2}}}$$

$$p_2^{n+1} = p_2^n - \frac{q_2^{n+1} \Delta t}{[(q_1^{n+1})^2 + (q_2^{n+1})^2]^{\frac{3}{2}}}$$

$$q_1^{n+1} = q_1^n + p_1^{n+1} \Delta t \ ,$$

$$q_2^{n+1} = q_2^n + p_2^{n+1} \Delta t \ .$$

...

Symplectic Euler

$$p_1^{n+1} = p_1^n - \frac{q_1^n \Delta t}{[(q_1^n)^2 + (q_2^n)^2]^{\frac{3}{2}}} ,$$

$$p_2^{n+1} = p_2^n - \frac{q_2^n \Delta t}{[(q_1^n)^2 + (q_2^n)^2]^{\frac{3}{2}}} ,$$

$$q_1^{n+1} = q_1^n + p_1^n \Delta t - \frac{q_1^n \Delta t^2}{[(q_1^n)^2 + (q_2^n)^2]^{\frac{3}{2}}}$$

$$q_2^{n+1} = q_2^n + p_2^n \Delta t - \frac{q_2^n \Delta t^2}{[(q_1^n)^2 + (q_2^n)^2]^{\frac{3}{2}}}$$

Numerical solutions

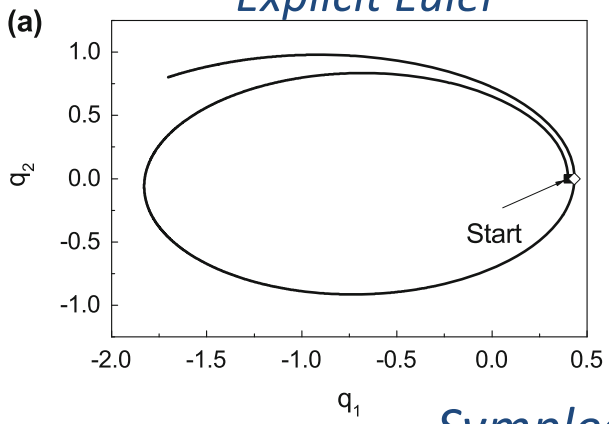
Initial conditions:

$$p_1(0) = 0, \qquad q_1(0) = 1 - e$$

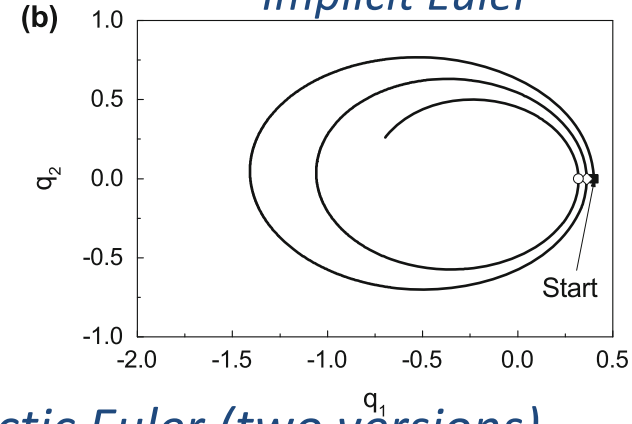
$$p_2(0) = \sqrt{\frac{1+e}{1-e}}, \qquad q_2(0) = 0$$

Parameters: e=0.6, Δt=0.01 for Symplectic Euler, Δt=0.005 else

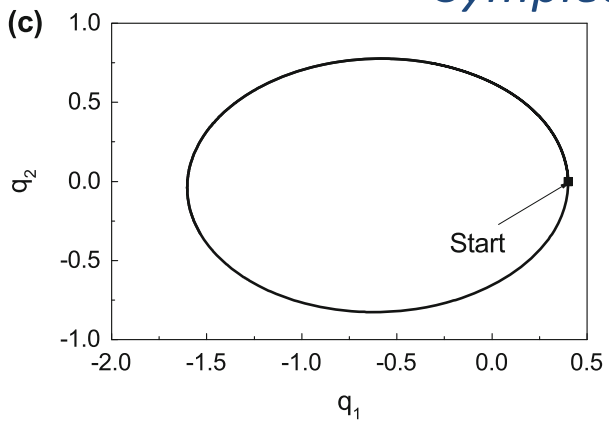
Explicit Euler

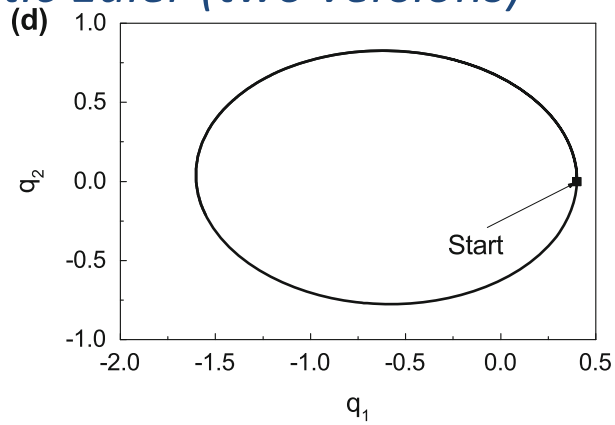


Implicit Euler

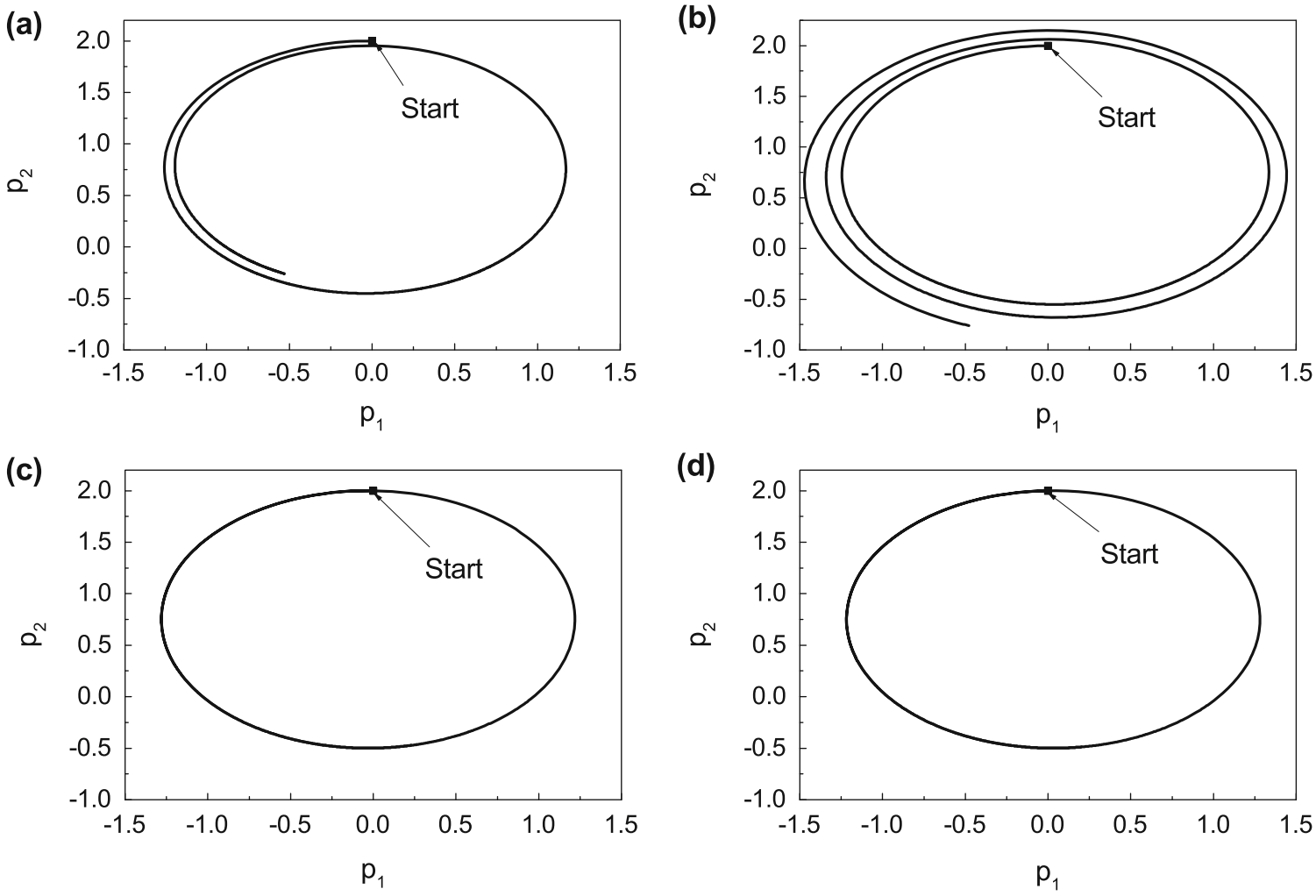


Symplectic Euler (two versions)

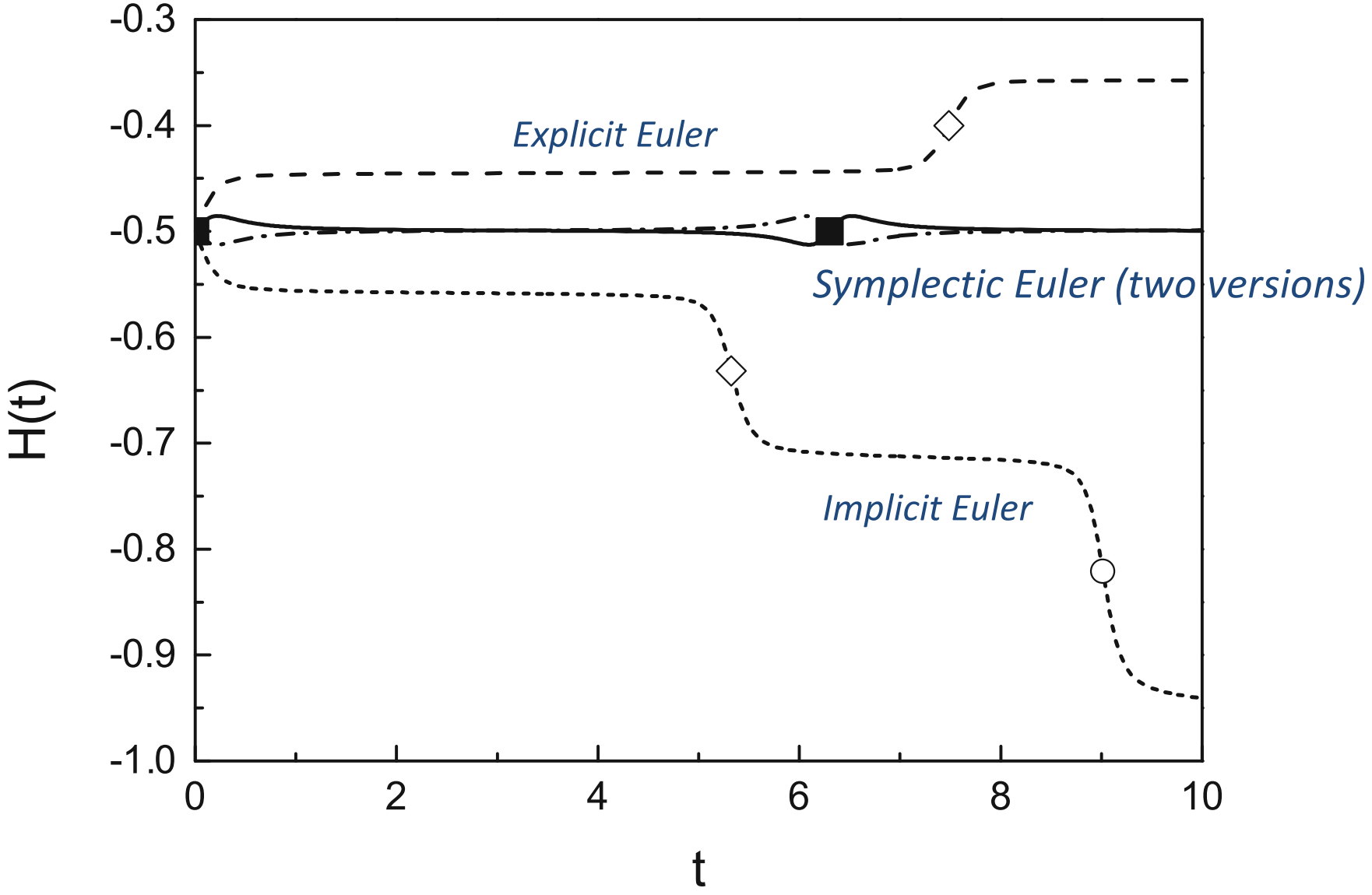




… (p space)



Energy


-0.3
-0.4
-0.5
-0.6
-0.7
-0.8
-0.9
-1.0
H(t)
0
2
4
6
8
10
t
Explicit Euler
Symplectic Euler (two versions)
Implicit Euler

Lab (today &) Thursday

Implement explicit methods for the Hamilton equations. Reproduce Figs. (a) & (c).

Homework:

- Problems: Chapter 5: 1,2,4
- Extra credit: Solve implicit Euler using Newton's method (appendix B)

Read chapter 5.4 for *Symplectic* integrators used for Hamiltonian systems to preserve energy better.

Possible final project: Solve the full two-body problem for different masses & initial conditions using appropriate Hamilton equations with a symplectic method (in 3D, with animation?)