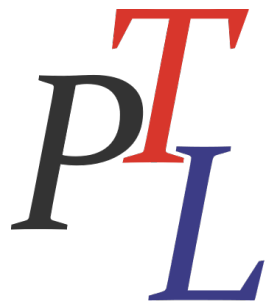
Computer-based methods for solving ordinary differential equations

Modelling and Computation of Electric and Magnetic Fields

Arturo Popoli

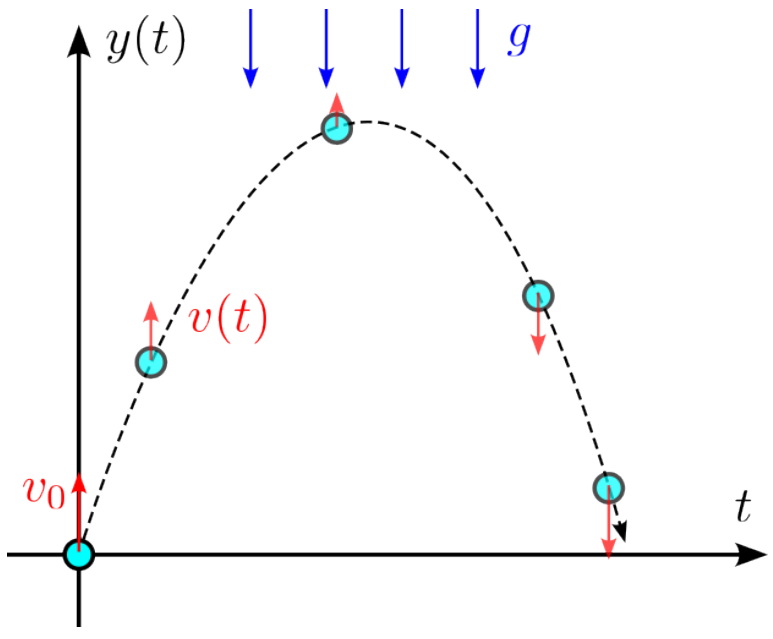


December 18, 2025

Introduction

- The goal of this short seminar is to provide an introduction on computer methods to solve ordinary differential equations (ODEs)
- Most of the examples will be based on **equations of motion** for particles, but the developed methods are nonetheless of general applicability

We start by considering a particle launched upward with initial velocity v_0; it decelerates until it reaches its apex, then accelerates downward under gravity.

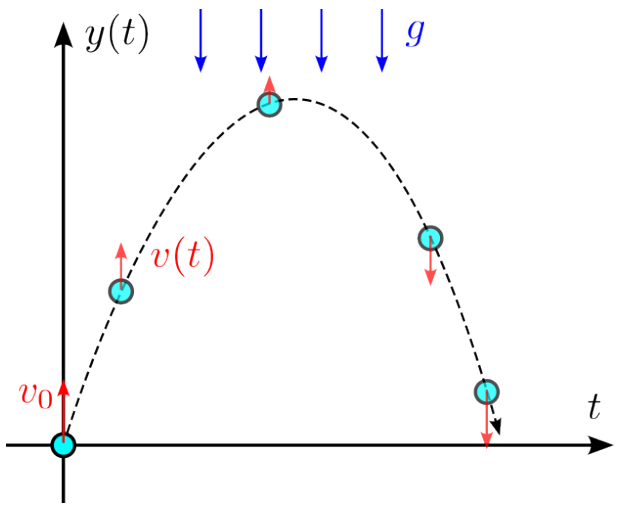


Particle in gravitational field – analytical solution

Goal: compute $y-$component of particle trajectory, $y(t)$ – we want an analytical solution to compare our solvers

- Governing equation (2^{nd}-order ODE)

$$\frac{\mathrm{d}^2 y}{\mathrm{d}t^2} = F/m \qquad (1)$$



- Cast into system of 1^{st}-order ODEs

$$\begin{cases} \dfrac{\mathrm{d}v}{\mathrm{d}t} = \dfrac{F}{m} & (2a) \\ \dfrac{\mathrm{d}y}{\mathrm{d}t} = v & (2b) \end{cases}$$

Particle in gravitational field – analytical solution

Starting from (2b), F/m is known (gravity acceleration)

$$g = 9.81\,\mathrm{m/s^2}$$

Substituting $\frac{F}{m} = -g$

$$\begin{cases} \dfrac{\mathrm{d}v}{\mathrm{d}t} = -g & \text{(3a)} \\ \dfrac{\mathrm{d}y}{\mathrm{d}t} = v & \text{(3b)} \end{cases}$$

Direct integration of Eq. (3a)

$$dv = -g dt$$

Particle in gravitational field – analytical solution

$$\int_{v_0}^{v} dv = \int_{t_0}^{t} -g \; dt$$

$$v - v_0 = -g(t - t_0)$$

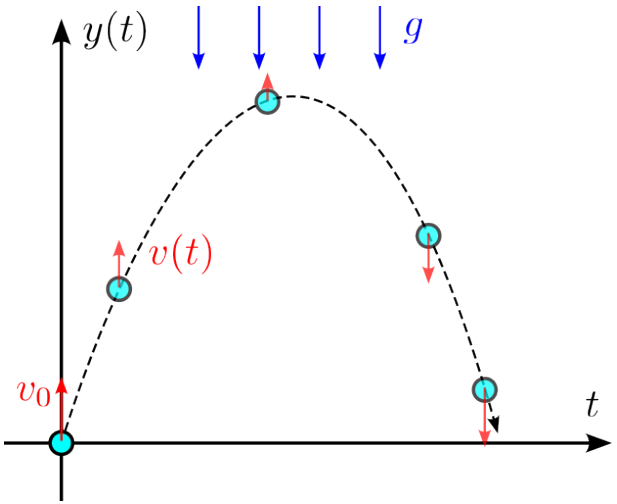
- Assuming $t_0 = 0$

$$v = v_0 - gt \tag{4}$$

- We obtained an expression for $v(t)$
- Now repeat for (3b) by substituting (4)

$$\frac{\mathrm{d}y}{\mathrm{d}t} = v_0 - gt$$

$$dy = (v_0 - gt)dt$$



Particle in gravitational field – analytical solution

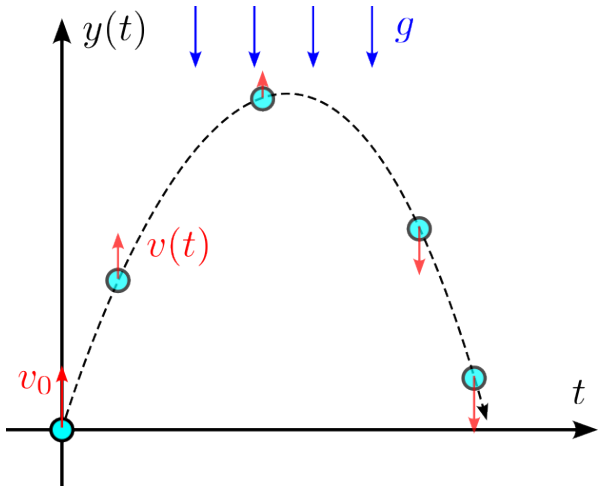
- Integrating

$$\int_{y_0}^{y} dy = \int_{0}^{t} (v_0 - gt)\, dt$$

- We finally get the analytical expression for the y-coordinate

$$y = y_0 + v_0 t - g\frac{t^2}{2} \qquad (5)$$

- We can use (5) to learn how to check the accuracy of our ODE integrators



Now we can *discretize* the governing equations and compare the numerical and analytical solution.

From continuous time to discrete time

We start from Taylor expansion around time t^k:

- Position:

$$y^{k+1} = y^k + \left.\frac{\mathrm{d}y}{\mathrm{d}t}\right|_k \Delta t + \frac{1}{2}\left.\frac{\mathrm{d}^2 y}{\mathrm{d}t^2}\right|_k \Delta t^2 + \mathcal{O}(\Delta t^3)$$

$$y^{k+1} = y^k + v^k \Delta t + \frac{1}{2} a^k \Delta t^2 + \mathcal{O}(\Delta t^3) \tag{6}$$

- Velocity:

$$v^{k+1} = v^k + a^k \Delta t + \mathcal{O}(\Delta t^2) \tag{7}$$

Our numerical schemes differ in how many terms of these expansions we keep.

Forward Euler as a first-order truncation

If we neglect terms of order $\mathcal{O}(\Delta t^2)$ and higher:

- Velocity update:

$$v^{k+1} = v^k + a^k \Delta t \tag{8}$$

- Position update:

$$y^{k+1} = y^k + v^k \Delta t \tag{9}$$

Note: completely neglects the quadratic term $\frac{1}{2} a^k \Delta t^2$

This is the **Forward Euler** method: first-order accurate in time.

Improving the position update

For the same velocity update...

$$v^{k+1} = v^k + a^k \Delta t \tag{10}$$

- **Euler–Cromer**: use the velocity at the end of the step

$$y^{k+1} = y^k + v^{k+1} \Delta t \tag{11}$$

- **Midpoint method**: approximate the average velocity

$$y^{k+1} = y^k + \frac{1}{2}\left(v^k + v^{k+1}\right) \Delta t \tag{12}$$

The midpoint update partially accounts for the $\frac{1}{2} a^k \Delta t^2$ term of the Taylor expansion.

Particle in gravitational field – discretization approaches

Now try to code the falling particle with the three methods:

- Forward Euler

$$\begin{cases} v^{k+1} = v^k + a^k \Delta t \\ y^{k+1} = y^k + v^k \Delta t \end{cases} \tag{13}$$

- Euler-Cromer

$$\begin{cases} v^{k+1} = v^k + a^k \Delta t \\ y^{k+1} = y^k + v^{k+1} \Delta t \end{cases} \tag{14}$$

- Midpoint Algorithm

$$\begin{cases} v^{k+1} = v^k + a^k \Delta t \\ y^{k+1} = y^k + \frac{1}{2}(v^k + v^{k+1}) \Delta t \end{cases} \tag{15}$$

Use the following parameters:

- $a = -g$, $g = 9.81\,\mathrm{m\,s^{-2}}$
- $v_0 = 10\,\mathrm{m\,s^{-1}}$
- $y_0 = 0$
- Start with $dt = 0.1$ and experiment to see how the accuracy is affected

Falling particle – pseudocode

- Pseudocode for falling particle with forward Euler

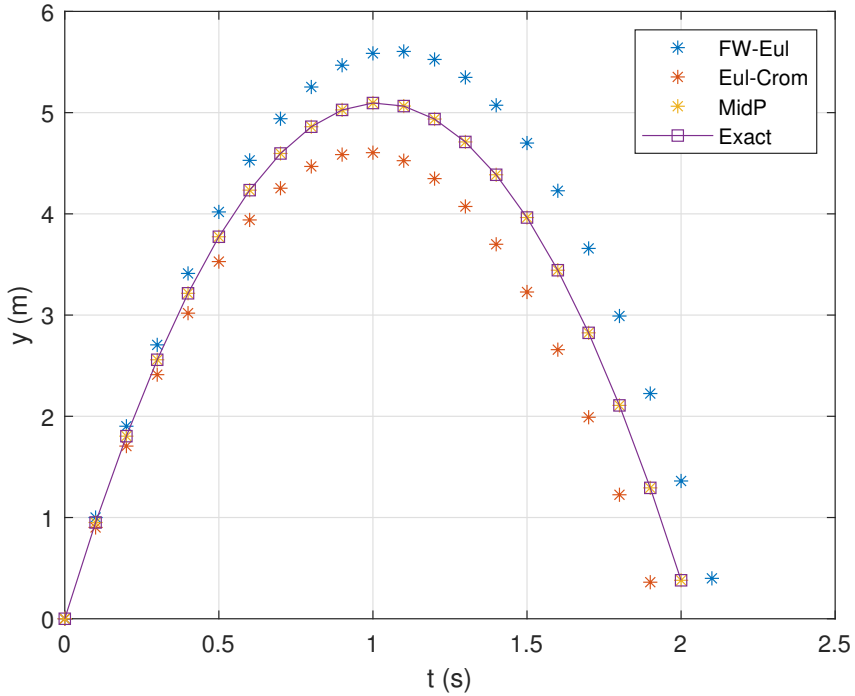
```
% time discretization
t = 0:dt:T

% allocate state variables
y(1) = y0          % position
v(1) = v0          % velocity

% time-stepping loop
start loop
        % update velocity (Newton's law)
        v(k+1) = ...

        % update position (select method)
        y(k+1) = ...
end loop
```

Expected results

You should get something like this...

- FW tends to *overshoot* the solution
- EC tends to *undershoot* the solution
- CE is much better – second-order!

- What about stability?



Circular Orbit

- The previous example was instructive, now we know that we need to look for **second order particle integrators**
- However, working with a constant acceleration is too easy – surely not the case in many real problems (such as in plasmas!)
- We need a more realistic test: Earth orbiting the Sun $\longrightarrow$ Kepler's law of motion
- **Assumption**: mass of the Earth (m) is insignificant compared to the mass of the Sun (M) $\longrightarrow$ circular orbit
- Gravitational force

$$\mathbf{F} = -\frac{GmM}{r^3}\mathbf{r},$$

where G is the gravitational constant, and $\mathbf{r}$ the position (origin or reference frame is the sun)

Circular Orbit

- For Earth, this must be balanced by mass (m) times acceleration

$$-\frac{GmM}{r^3}\mathbf{r} = m\mathbf{a}$$

- Therefore we have that acceleration depends only on position and the solar mass

$$\mathbf{a} = -\frac{GM}{r^3}\mathbf{r}$$

- Now we have our governing equation

$$\frac{\mathrm{d}^2\mathbf{r}}{\mathrm{d}t^2} = -\frac{GM}{r^3}\mathbf{r}$$

Circular Orbit – astronomical units

- Consider the acceleration magnitude $a = \frac{GM}{r^2}$
- We can simplify the expression using **astronomical units**
- The motion is circular, hence

$$a = \frac{v^2}{r} = \frac{\omega^2 r^2}{r} = \frac{4\pi^2}{T^2} r \tag{16}$$

- Substituting, we find

$$\frac{4\pi^2}{T^2} r = \frac{GM}{r^2} \tag{17}$$

- We know that if $r = 1\,\mathrm{au}$, $T = 1\,\mathrm{yr}$
- If we set that $M = 1$

$$G = 4\pi^2 \tag{18}$$

- The governing equation now becomes

$$\frac{\mathrm{d}^2\mathbf{r}}{\mathrm{d}t^2} = -\frac{4\pi^2}{r^3}\mathbf{r} \tag{19}$$

Circular Orbit – implementation

- Decompose the motion in the x and y directions

$$\begin{cases} \frac{\mathrm{d}^2 x}{\mathrm{d}t^2} = -\frac{4\pi^2}{r^3} x \\ \frac{\mathrm{d}^2 y}{\mathrm{d}t^2} = -\frac{4\pi^2}{r^3} y \end{cases} \tag{20}$$

- Then perform again the same reasoning used for falling particle
  - Split each second-order IVP in 2 coupled first-order IVPs
  - Example for x component

$$\begin{cases} \frac{\mathrm{d}v_x}{\mathrm{d}t} = -\frac{4\pi^2}{r^3} x \\ \frac{\mathrm{d}x}{\mathrm{d}t} = v_x \end{cases} \tag{21}$$

  - Discretize using the three methods we have derived, see (13), (14), (15)

Circular Orbit – implementation

Suggested settings:

```
T = 1 % 1 year
dt = T/50; % time-step
x_old = 0; % initial position
y_old = 1;
vx_old = -sqrt(4*pi^2/y_old); % initial velocity
vy_old = 0;
```

Circular Orbit – implementation

- You can use an m function or an anonymous function to compute the distance r

```
r = @(x,y) sqrt(x^2+y^2); % distance
```

- ... the two acceleration components

```
ax = @(x,y) -4*pi^2*r(x,y)^3 * x; % F/m - x
ay = @(x,y) -4*pi^2*r(x,y)^3 * y; % F/m - y
```

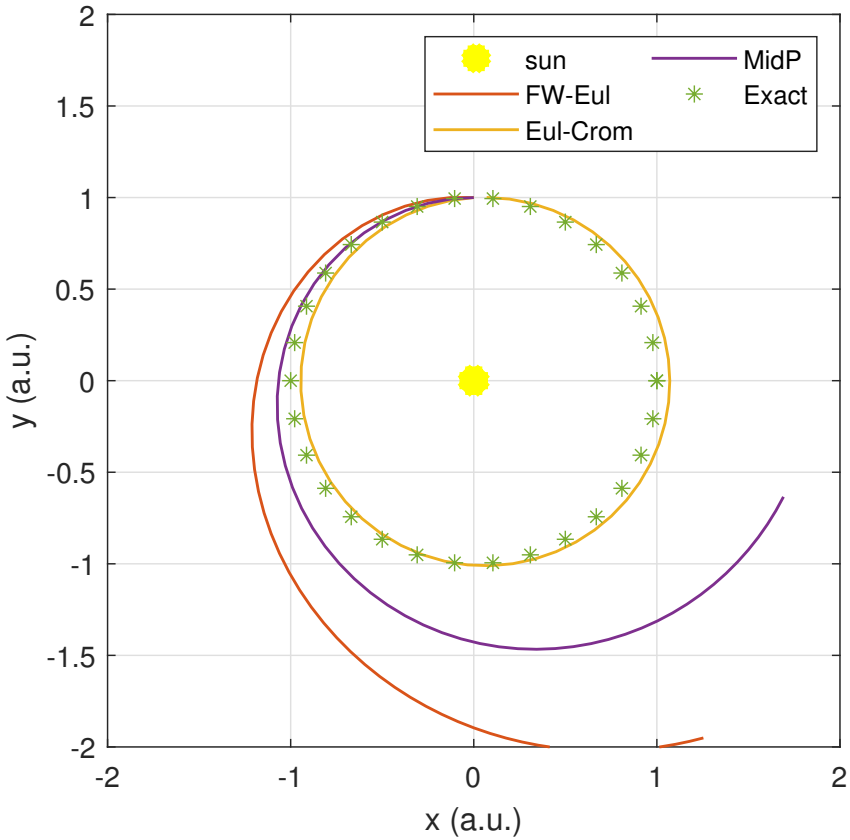
- We expect a circular orbit with unit radius, we can check it!

```
r = 1;
th = 0:pi/15:2*pi;
xunit = r * cos(th);
yunit = r * sin(th);
plot(xunit, yunit,'*');
axis equal
```

Circular Orbit – results

You should get something like this...

- Forward Euler fails spectacularly!
- Midpoint is clearly better than forward Euler, but still unstable... (it's like a poor man's Crank-Nicolson), since the velocity update is still explicit
- Real surprise is Euler-Cromer – only first-order accurate but stable!



What about accuracy? We would expect midpoint to be better than Euler-Cromer...

- Try to use `dt = T/500` and zoom on the initial part of trajectory

From midpoint to half-step velocities

In the midpoint method, the position update uses the *average velocity*:

$$\begin{cases} v^{k+1} = v^k + a^k \Delta t \\ y^{k+1} = y^k + \frac{1}{2}(v^k + v^{k+1})\Delta t \end{cases}$$

The key idea for its improvement is to *interpret* this average as the velocity at the midpoint in time between instants k and $k+1$:

$$v^{k+\frac{1}{2}} \equiv \frac{1}{2}\left(v^k + v^{k+1}\right)$$

The position update becomes

$$y^{k+1} = y^k + v^{k+\frac{1}{2}} \Delta t$$

Central difference for the velocity

The acceleration is known at integer time steps:

$$a^k = a\left(y^k, t^k\right)$$

. . . but, instead of just updating v^k to v^{k+1} with:

$$v^{k+1} = v^k + a^k \Delta t$$

We update the velocity using a **central difference** in time:

$$v^{k+\frac{1}{2}} = v^{k-\frac{1}{2}} + a^k \Delta t$$

This advances the velocity from one half-step to the next

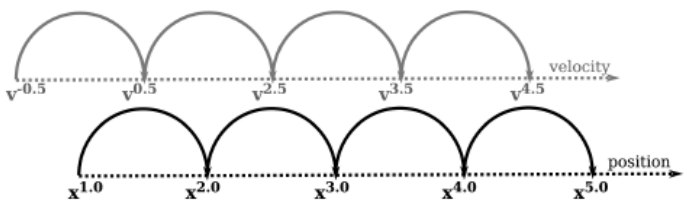
Velocity is now treated **symmetrically in time**

Leapfrog method

. . . so now velocity and position are **offset** by half a time step:

- positions are known at t^k
- velocities are known at $t^{k+\frac{1}{2}}$

$$\begin{cases} v^{k+\frac{1}{2}} = v^{k-\frac{1}{2}} + a^k \Delta t \\ y^{k+1} = y^k + v^{k+\frac{1}{2}} \Delta t \end{cases}$$



Velocity and position *leap over* each other in time.

Both updates use central differences $\Rightarrow$ second-order accurate in time.

Leapfrog startup step

Problem: at $t = 0$ we know:

- initial position y^0
- initial velocity v^0

But the leapfrog scheme requires $v^{\frac{1}{2}}$.

We compute it using a backward half-step via forward Euler

$$v^{-\frac{1}{2}} = v^0 - \frac{1}{2} a^0 \Delta t$$

After this initialization, the leapfrog scheme proceeds normally.

Pseudocode for leapfrog integrator

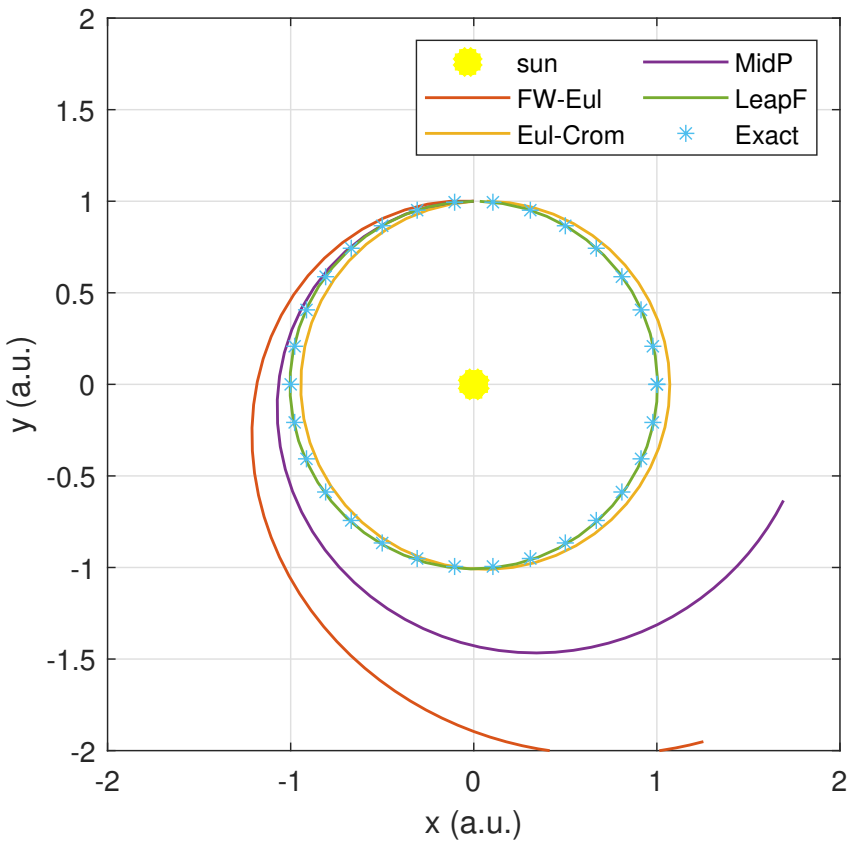
Pseudo-code for Leapfrog scheme

```
% initialization (t = 0)
compute acceleration a
set v_old = v0 - 0.5*a*dt % vel rewind, v0: initial vel

% main loop
start loop
        compute acceleration a
        integrate velocity, v = v_old + a*dt
        integrate position, x = x_old + v*dt
end loop
```

Circular Orbit – results

You should get something like this...

- Leapfrog is – by far – the best option
  - Second-order
  - Almost same computational cost of a first-order method
  - Symplectic
- Euler-Cromer is also symplectic
  - Also known as *symplectic Euler* or *semi-implicit Euler*
- What about higher-order schemes? They would allow for larger time-steps...
  - Not necessarily symplectic...

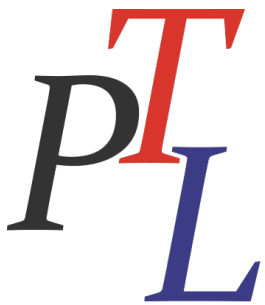


Conclusions / Thank You!

Thank you for your attention!

That's all for today.

arturo.popoli@unibo.it



Link to PTL website
Link to PTL LinkedIn page

References

- Tutorial on computational astrophysics
- Numerical Integration of Newton's Equation of Motion
- Programming for Computations (Python version) (hplgit.github.io)