King Abdullah University of Science and Technology

KAUST

CS 247 – Scientific Visualization
Lecture 21: Vector Field / Flow Visualization, Pt.3

Markus Hadwiger, KAUST

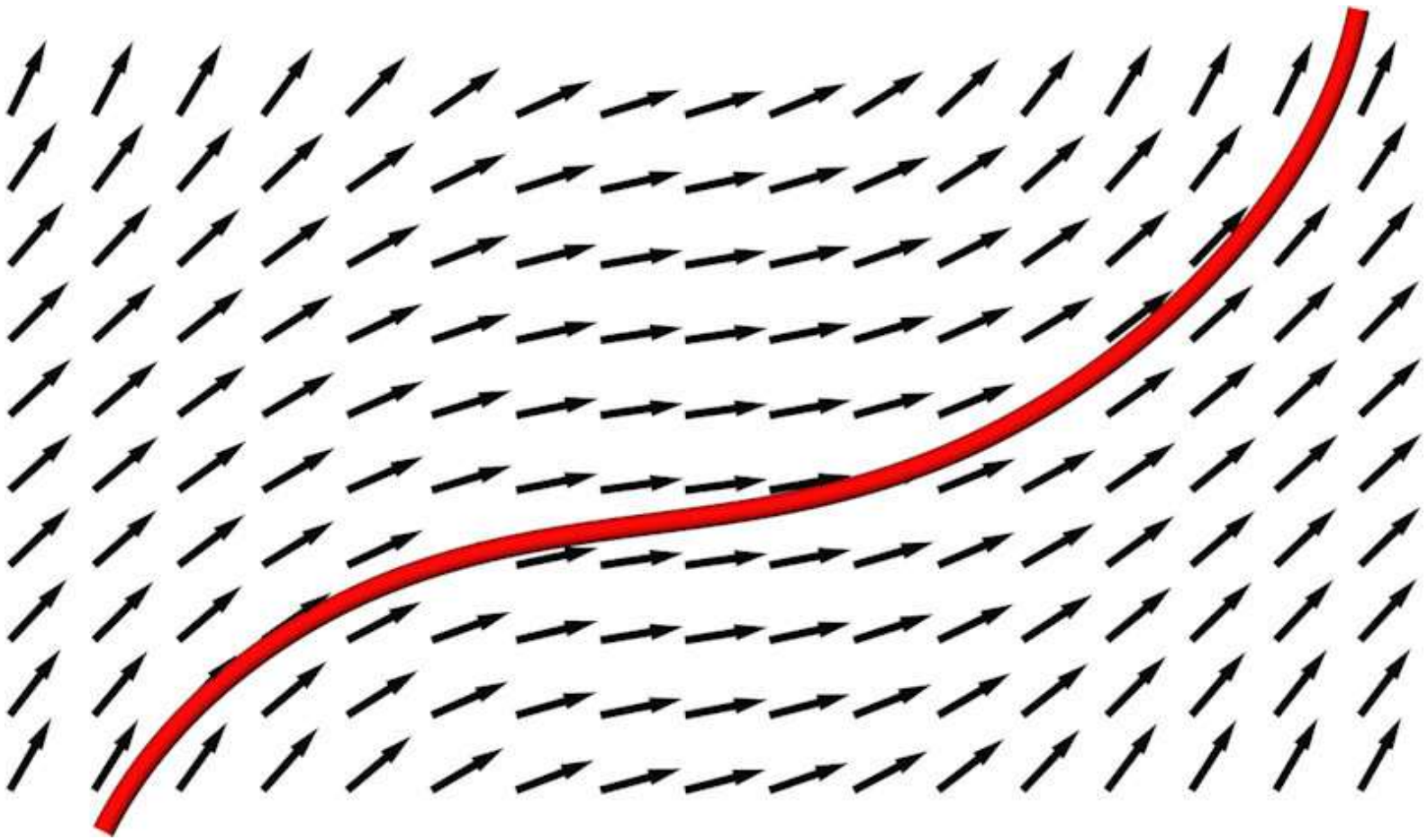
Reading Assignment #13 (until May 10)

Read (required):

- Data Visualization book
  - Chapter 6.1 (Divergence and Vorticity)
  - Chapter 6.6 (Texture-Based Vector Visualization)
- Diffeomorphisms / smooth deformations
  `https://en.wikipedia.org/wiki/Diffeomorphism`
- Learn how convolution (the convolution of two functions) works:
  `https://en.wikipedia.org/wiki/Convolution`
- B. Cabral, C. Leedom:
  Imaging Vector Fields Using Line Integral Convolution, SIGGRAPH 1993
  `http://dx.doi.org/10.1145/166117.166151`

Integral Curves / Stream Objects

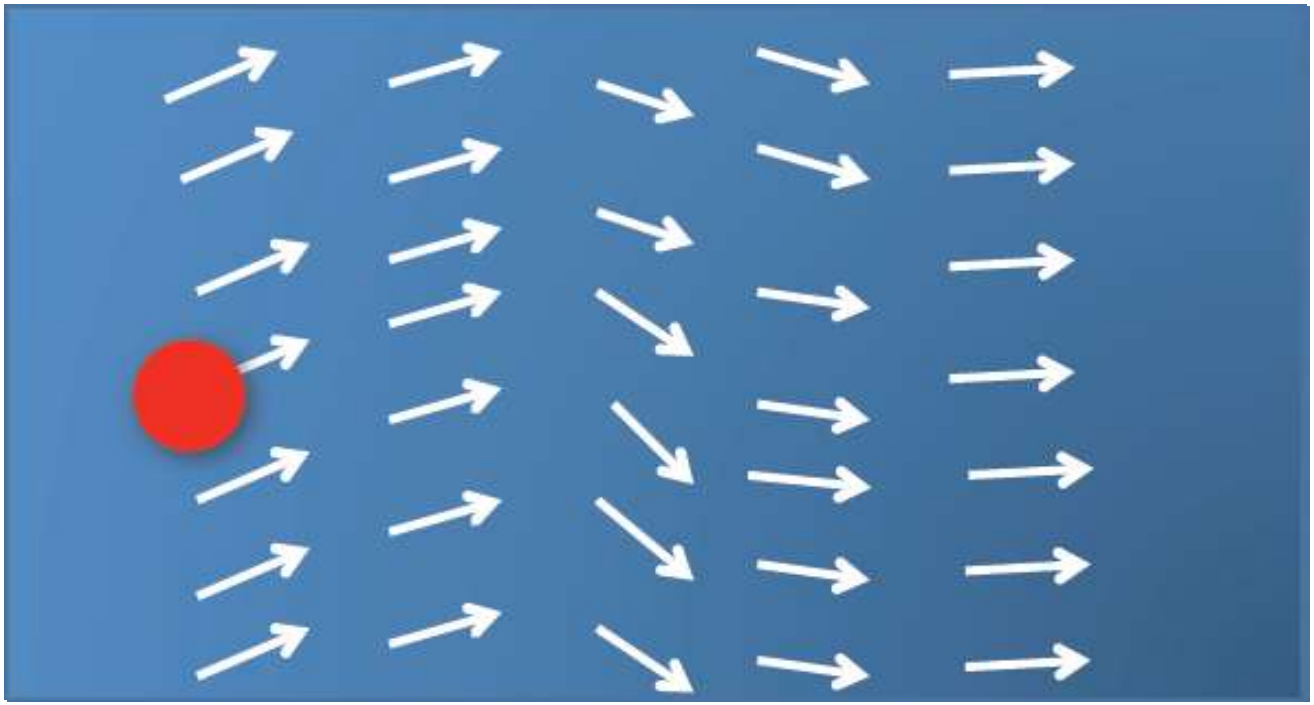
Integrating velocity over time yields spatial motion



Particle Trajectories

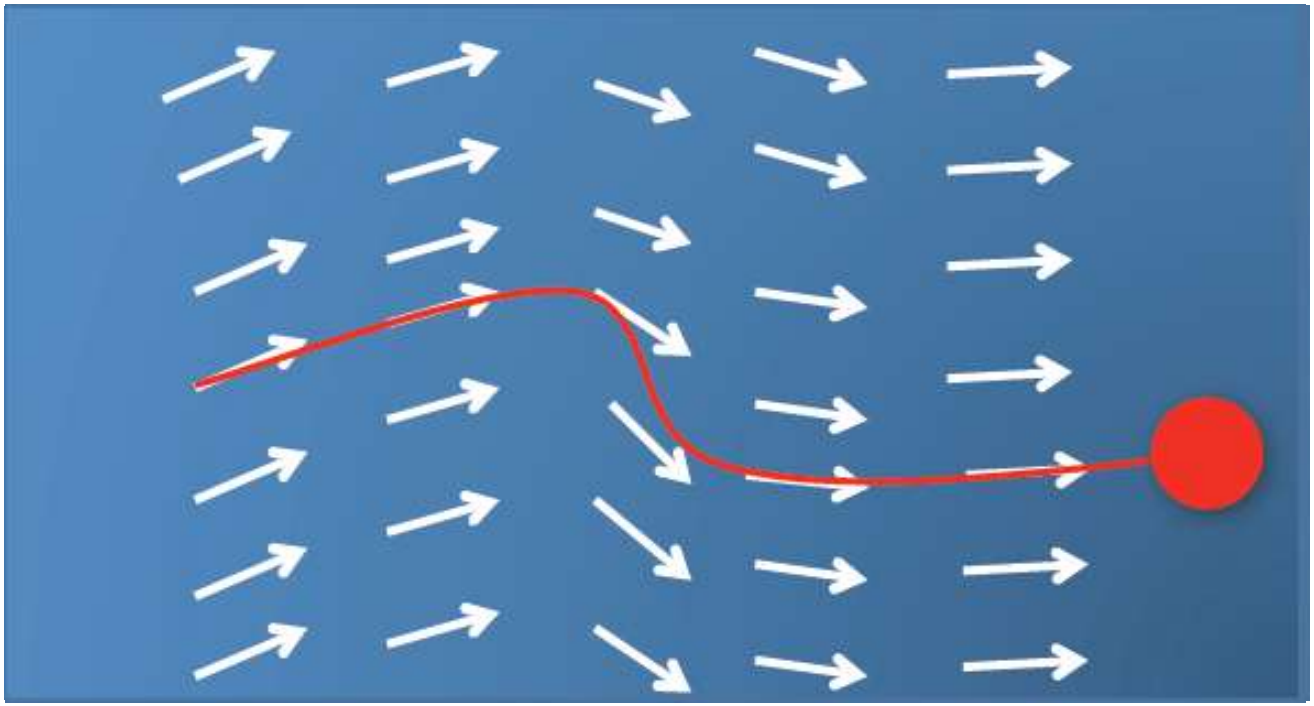
Courtesy Jens Krüger

Particle Trajectories



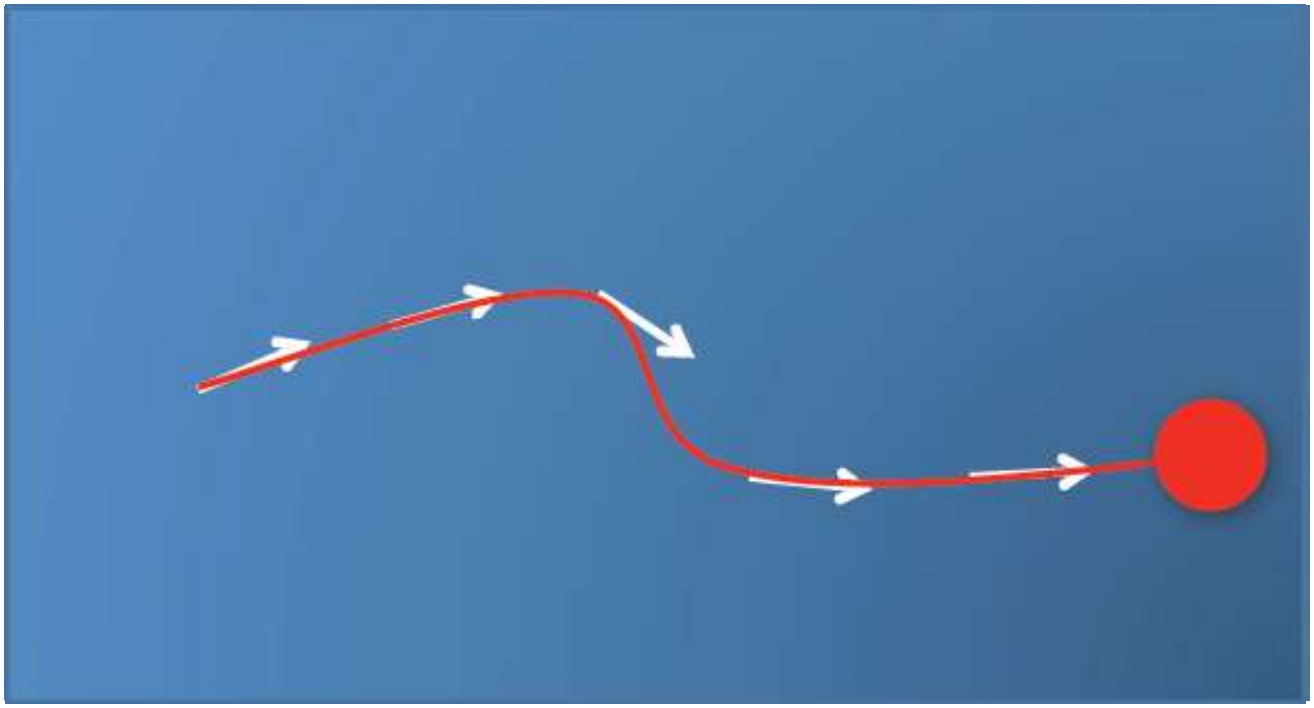
Courtesy Jens Krüger

Particle Trajectories



Courtesy Jens Krüger

Particle Trajectories



Courtesy Jens Krüger

Vector fields as ODEs

For simplicity, the vector field is now interpreted as a velocity field.

Then the field $\mathbf{v}(\mathbf{x},t)$ describes the connection between location and velocity of a (massless) particle.

It can equivalently be expressed as an ordinary differential equation

$$\dot{\mathbf{x}}(t) = \mathbf{v}(\mathbf{x}(t),t)$$

This ODE, together with an initial condition

$$\mathbf{x}(t_0) = \mathbf{x}_0 \ ,$$

is a so-called initial value problem (IVP).

Its solution is the integral curve (or trajectory)

$$\mathbf{x}(t) = \mathbf{x}_0 + \int_{t_0}^{t} \mathbf{v}(\mathbf{x}(\tau),\tau)\, d\tau$$

The integral curve is a pathline, describing the path of a massless particle which was released at time t_0 at position x_0.

Remark: $t < t_0$ is allowed.

For static fields, the ODE is autonomous:

$$\dot{\mathbf{x}}(t) = \mathbf{v}(\mathbf{x}(t))$$

and its integral curves

$$\mathbf{x}(t) = \mathbf{x}_0 + \int_{t_0}^{t} \mathbf{v}(\mathbf{x}(\tau))\, d\tau$$

are called field lines, or (in the case of velocity fields) streamlines.

In static vector fields, pathlines and streamlines are identical.

In time-dependent vector fields, instantaneous streamlines can be computed from a "snapshot" at a fixed time T (which is a static vector field)

$$\mathbf{v}_T(\mathbf{x}) = \mathbf{v}(\mathbf{x}, T)$$

In practice, time-dependent fields are often given as a dataset per time step. Each dataset is then a snapshot.

Streamline integration

Outline of algorithm for numerical streamline integration (with obvious extension to pathlines):

Inputs:

- static vector field $\mathbf{v}(\mathbf{x})$
- seed points with time of release $(\mathbf{x}_0, t_0)$
- control parameters:
  - step size (temporal, spatial, or in local coordinates)
  - step count limit, time limit, etc.
  - order of integration scheme

Output:

- streamlines as "polylines", with possible attributes (interpolated field values, time, speed, arc length, etc.)

Preprocessing:

- set up search structure for point location
- for each seed point:
  - global point location: Given a point **x**,
    find the cell containing **x** and the local coordinates (ξ, η, ζ)
    or ir the grid is structured:
    find the computational space coordinates $(i + \xi,\ j + \eta,\ k + \zeta)$
  - If **x** is not found in a cell, remove seed point

Integration loop, for each seed point **x**:

- interpolate ***v*** trilinearly to local coordinates (ξ, η, ζ)
- do an integration step, producing a new point **x'**
- incremental point location: For position **x'** find cell and local coordinates (ξ', η', ζ') making use of information (coordinates, local coordinates, cell) of old point **x**

Termination criteria:

- grid boundary reached
- step count limit reached
- optional: velocity close to zero
- optional: time limit reached
- optional: arc length limit reached

Integration step: widely used integration methods:

- Euler (used only in special speed-optimized techniques, e.g. GPU-based texture advection)

$$\mathbf{x}_{new} = \mathbf{x} + \mathbf{v}(\mathbf{x}, t) \cdot \Delta t$$

- Runge-Kutta, 2nd or 4th order

Higher order than 4th?

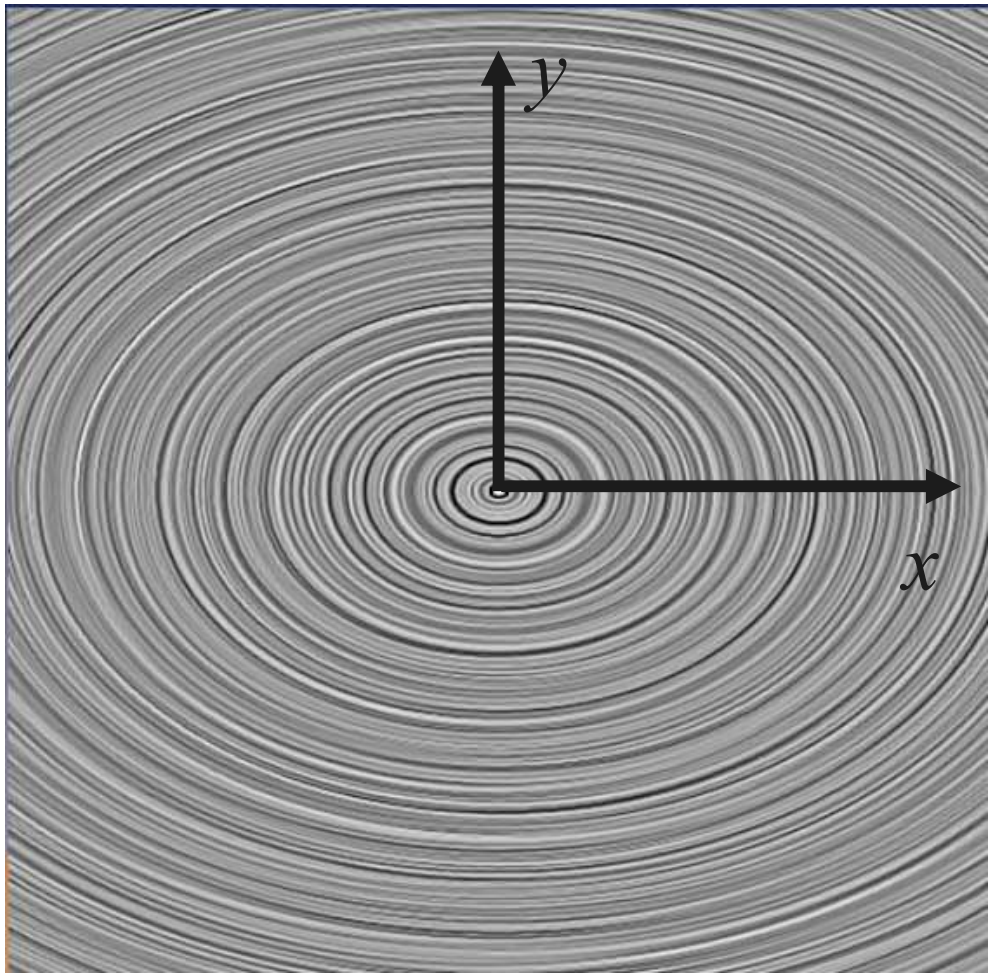
- often too slow for visualization
- study (Yeung/Pope 1987) shows that, when using standard trilinear interpolation, interpolation errors dominate integration errors.

Numerical Integration

- **Numerical integration of stream lines:**

- approximate streamline by polygon $\mathbf{x}_i$

- **Testing example:**
  - $\mathbf{v}(x,y) = (-y, x/2)^T$
  - exact solution: ellipses
  - starting integration from (0,-1)



Streamlines – Practice

- Basic approach:
  - theory: $\mathbf{s}(t) = \mathbf{s}_0 + \int_{0 \le u \le t} \mathbf{v}(\mathbf{s}(u))\,\mathrm{d}u$
  - practice: numerical integration
  - idea:
    (very) locally, the solution is (approx.) linear
  - Euler integration:
    follow the current flow vector $\mathbf{v}(\mathbf{s}_i)$ from the current streamline point $\mathbf{s}_i$ for a very small time ($\mathrm{d}t$) and therefore distance
  - Euler integration: $\mathbf{s}_{i+1} = \mathbf{s}_i + \mathrm{d}t \cdot \mathbf{v}(\mathbf{s}_i)$,
    integration of small steps ($\mathrm{d}t$ very small)

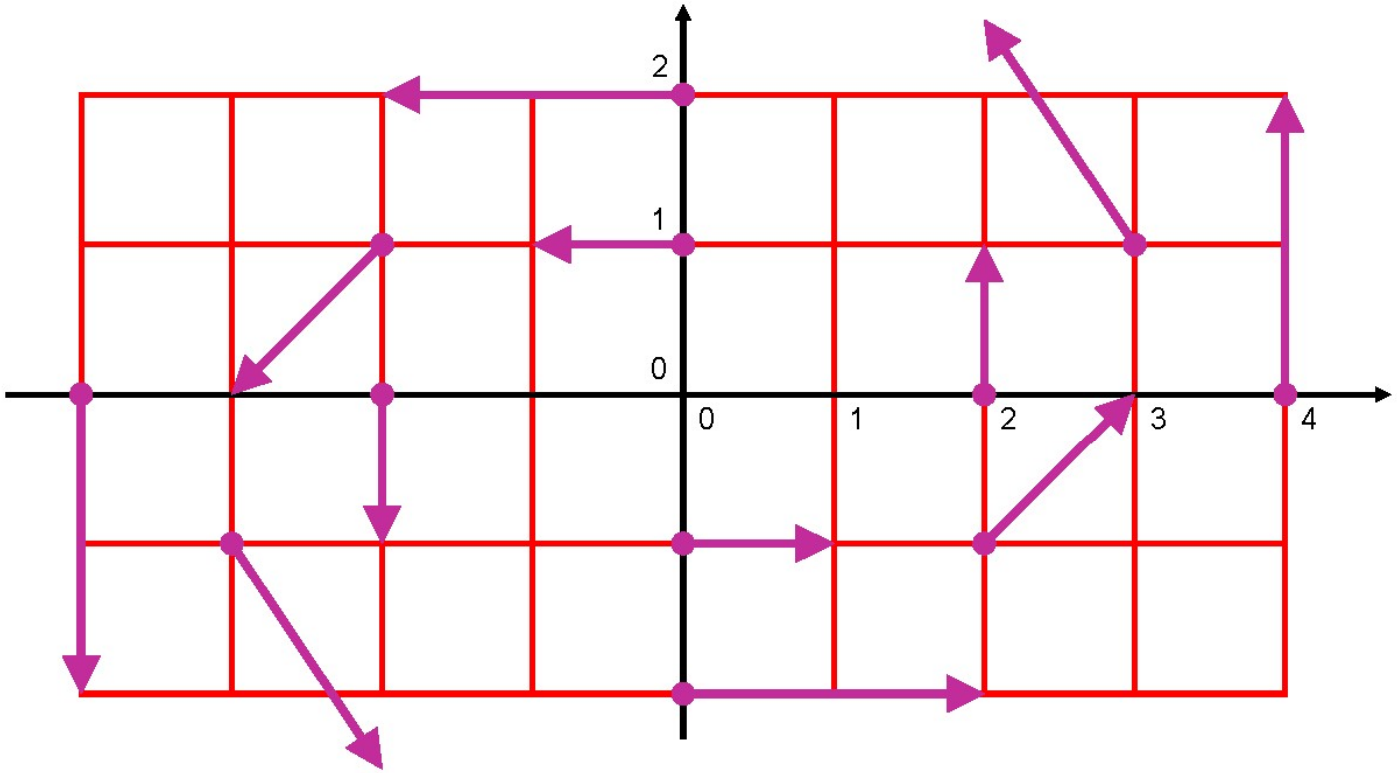
Euler Integration – Example

- 2D model data:

$$v_x = dx/dt = -y$$
$$v_y = dy/dt = x/2$$

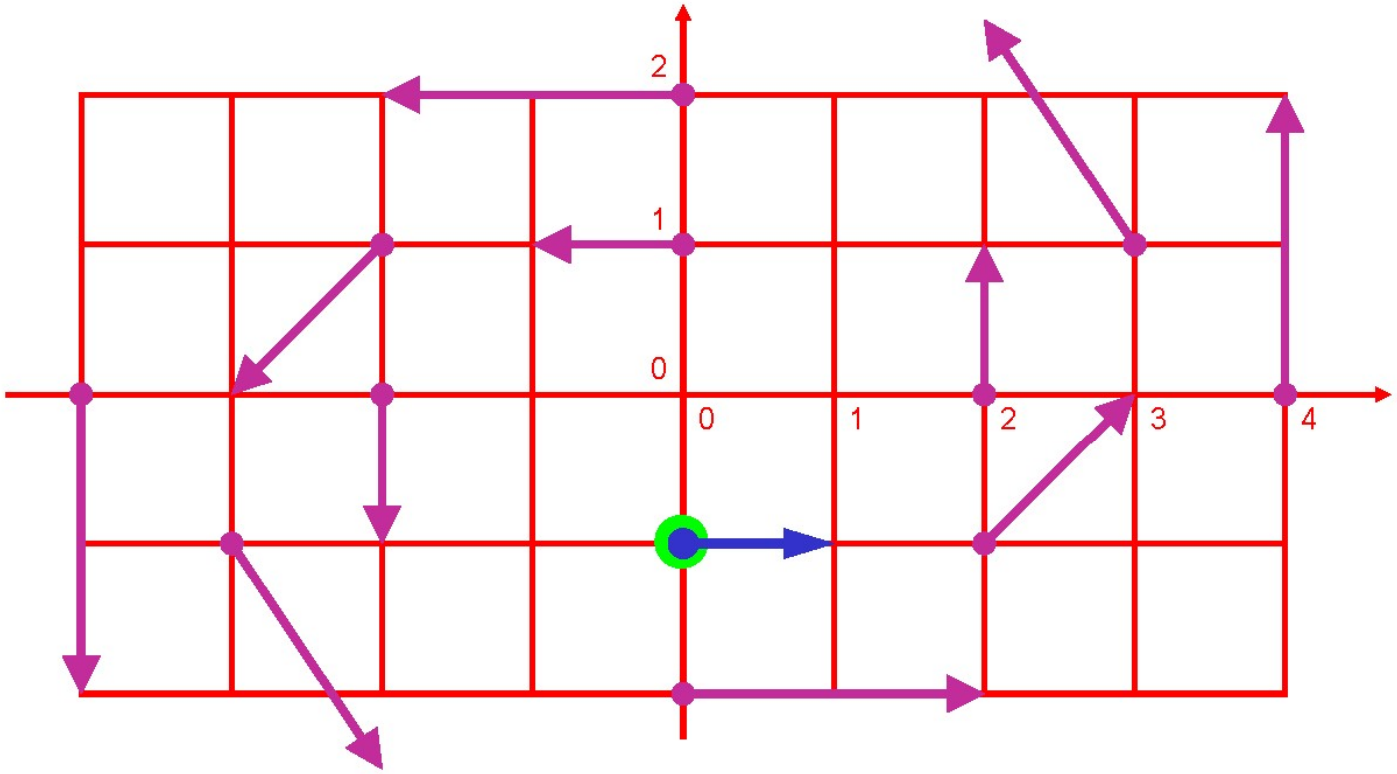
- Sample arrows:

- True solution: ellipses!



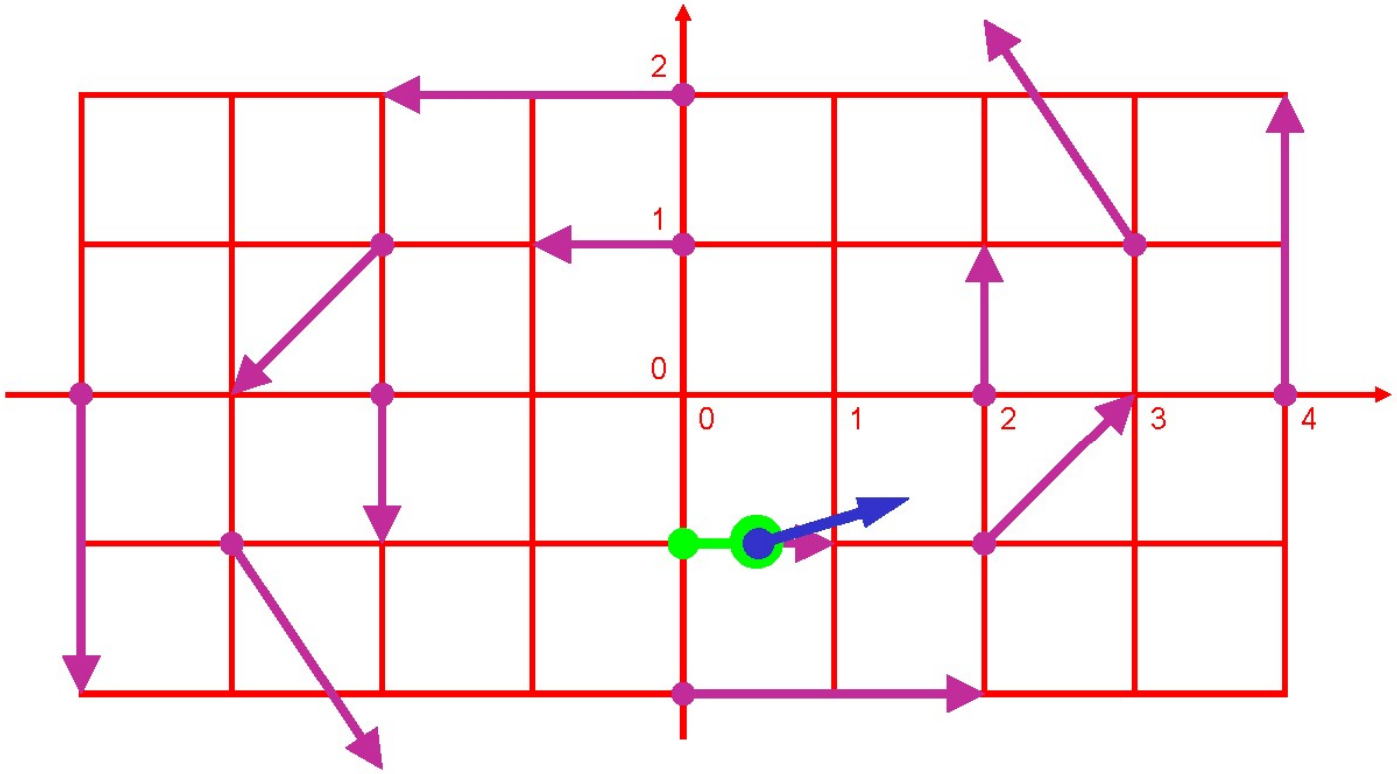
Euler Integration – Example

- Seed point $\mathbf{s}_0 = (0|-1)^T$;
  current flow vector $\mathbf{v}(\mathbf{s}_0) = (1|0)^T$;
  $dt = 1/2$



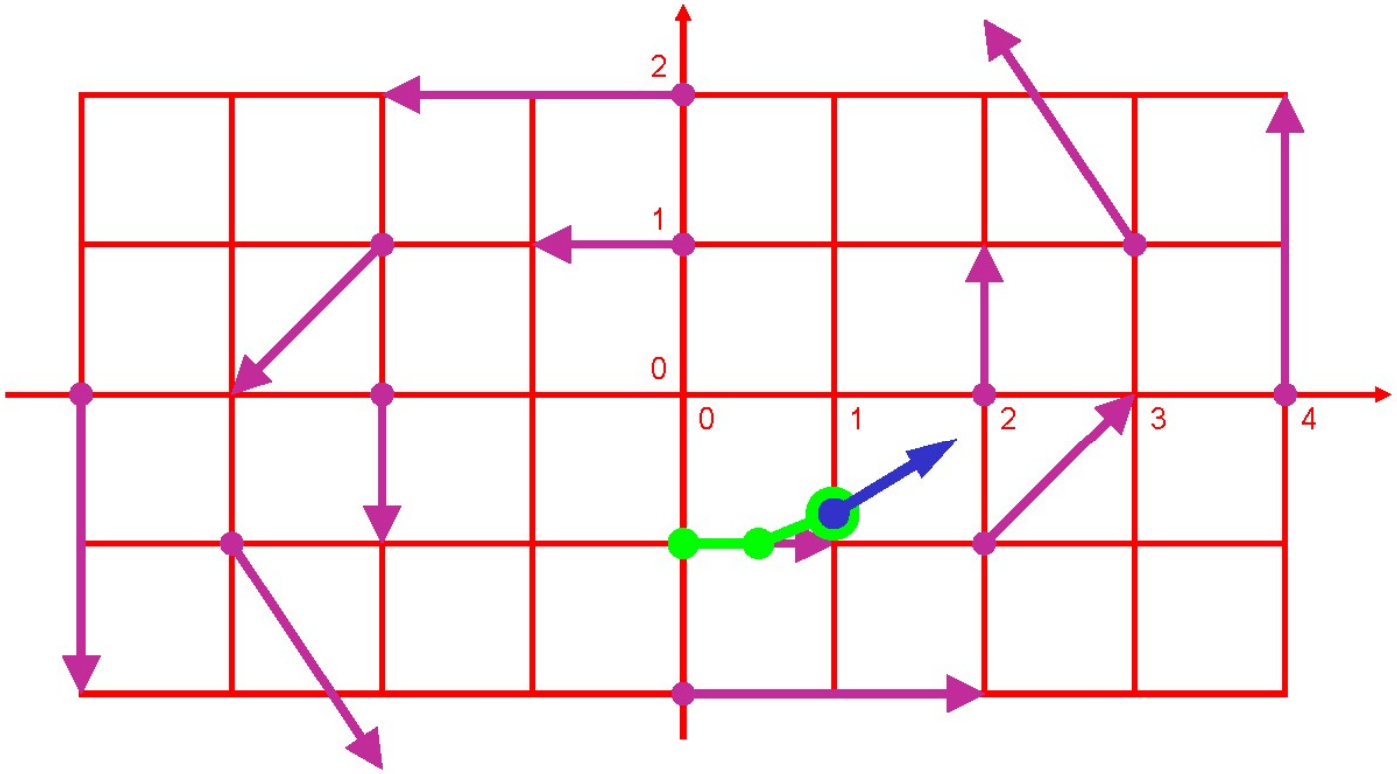
Euler Integration – Example

- New point $\mathbf{s}_1 = \mathbf{s}_0 + \mathbf{v}(\mathbf{s}_0) \cdot dt = (1/2 \,|\, -1)^T$;
  current flow vector $\mathbf{v}(\mathbf{s}_1) = (1 \,|\, 1/4)^T$;



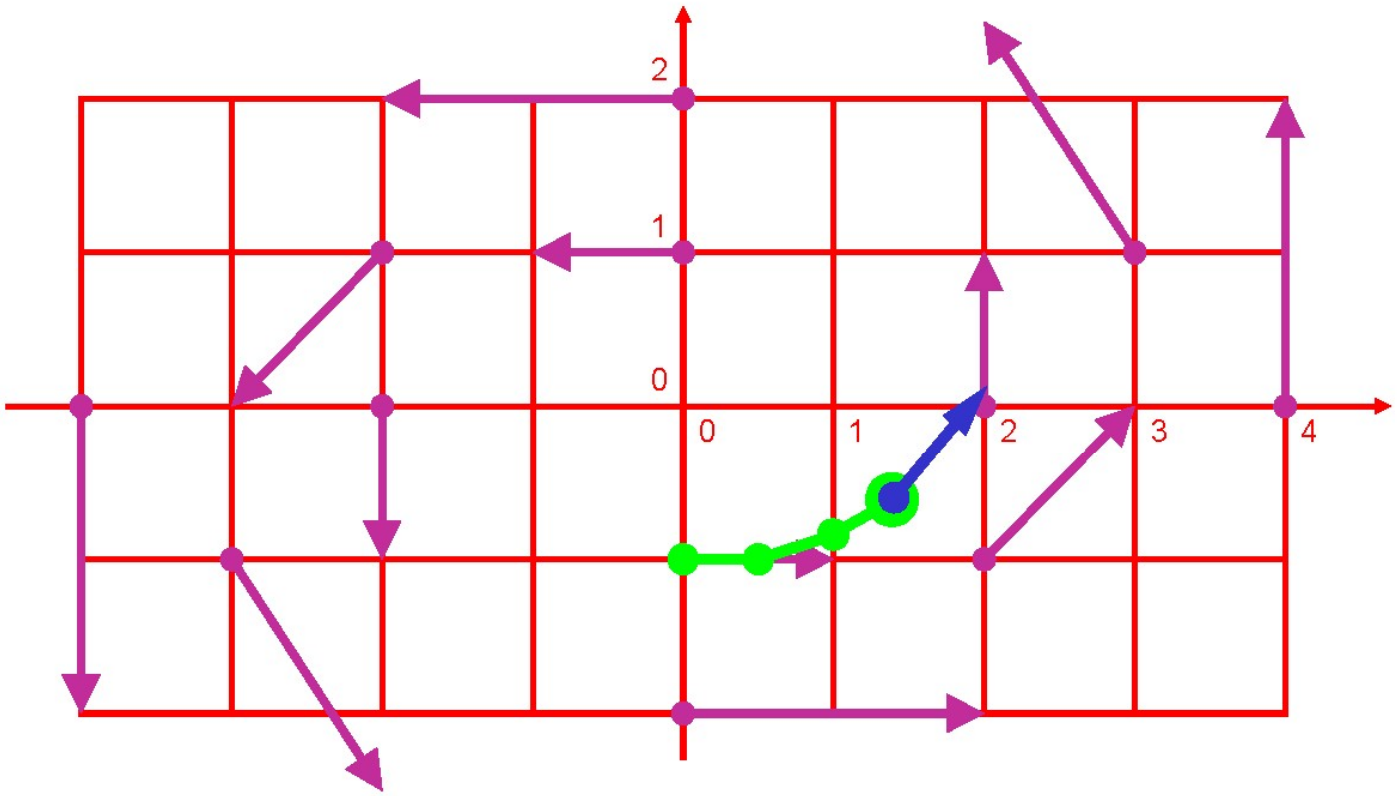
Euler Integration – Example

- New point $\mathbf{s}_2 = \mathbf{s}_1 + \mathbf{v}(\mathbf{s}_1) \cdot dt = (1 \mid -7/8)^T$;
  current flow vector $\mathbf{v}(\mathbf{s}_2) = (7/8 \mid 1/2)^T$;



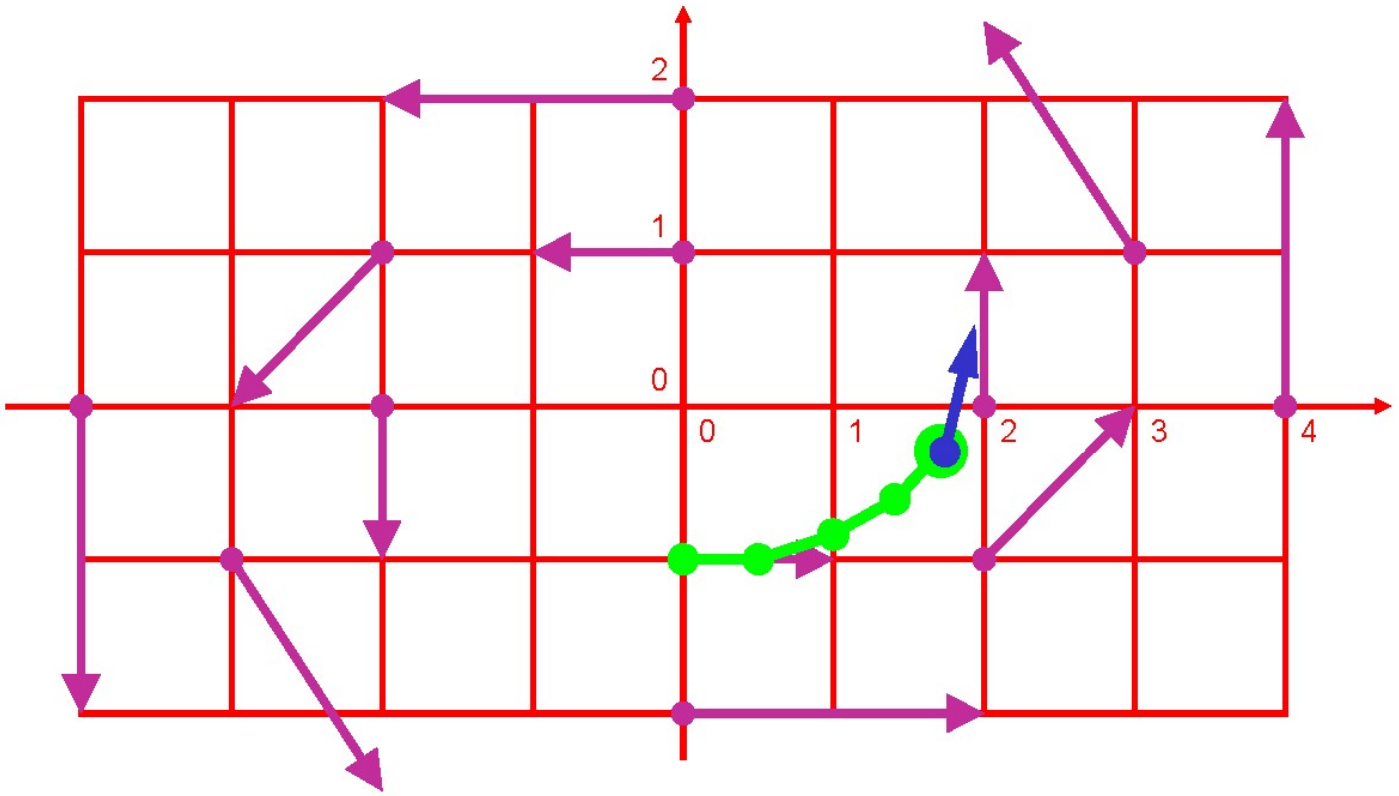
Euler Integration – Example

- $\mathbf{s}_3 = (23/16 | -5/8)^T \approx (1.44 | -0.63)^T;$
  $\mathbf{v}(\mathbf{s}_3) = (5/8 | 23/32)^T \approx (0.63 | 0.72)^T;$



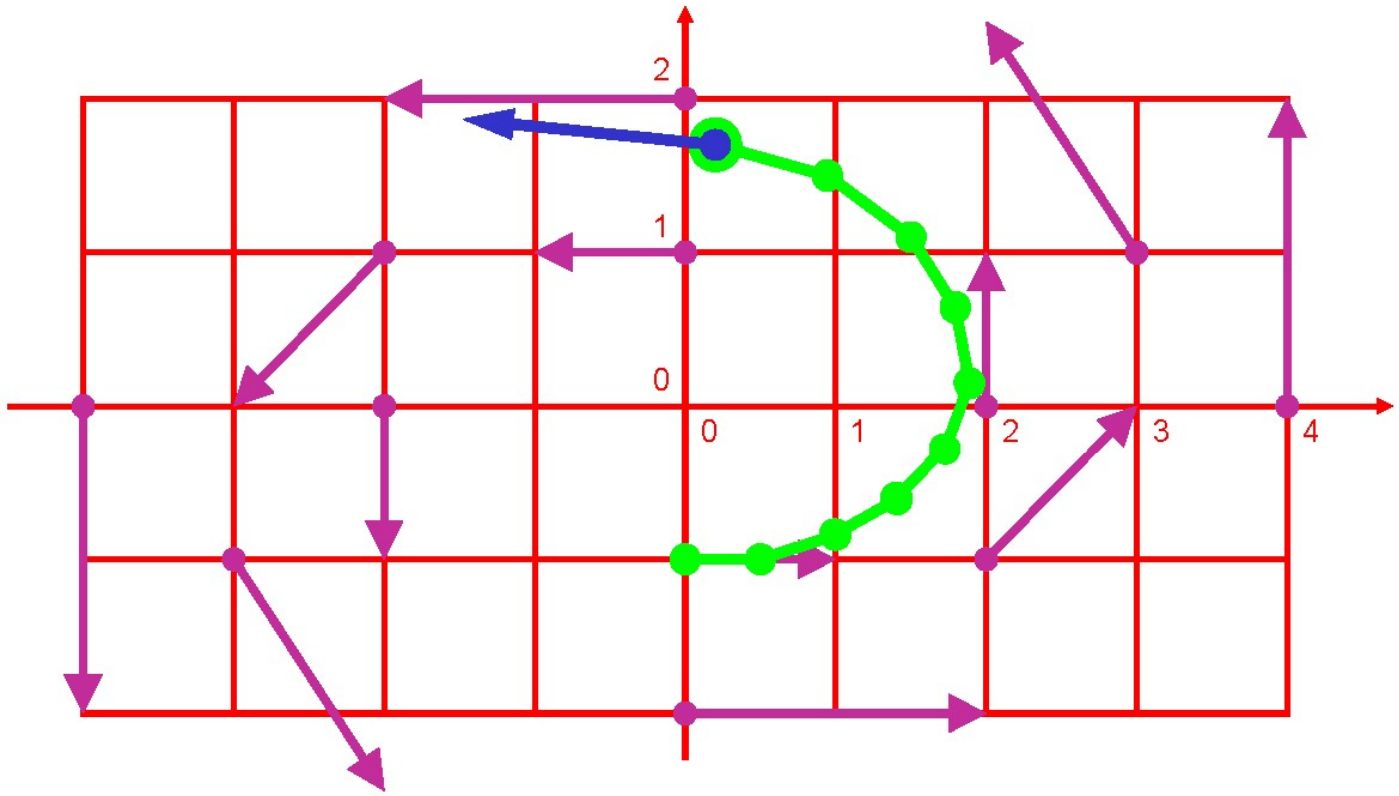
Euler Integration – Example

- $\mathbf{s}_4 = (7/4 \mid -17/64)^T \approx (1.75 \mid -0.27)^T;$
  $\mathbf{v}(\mathbf{s}_4) = (17/64 \mid 7/8)^T \approx (0.27 \mid 0.88)^T;$



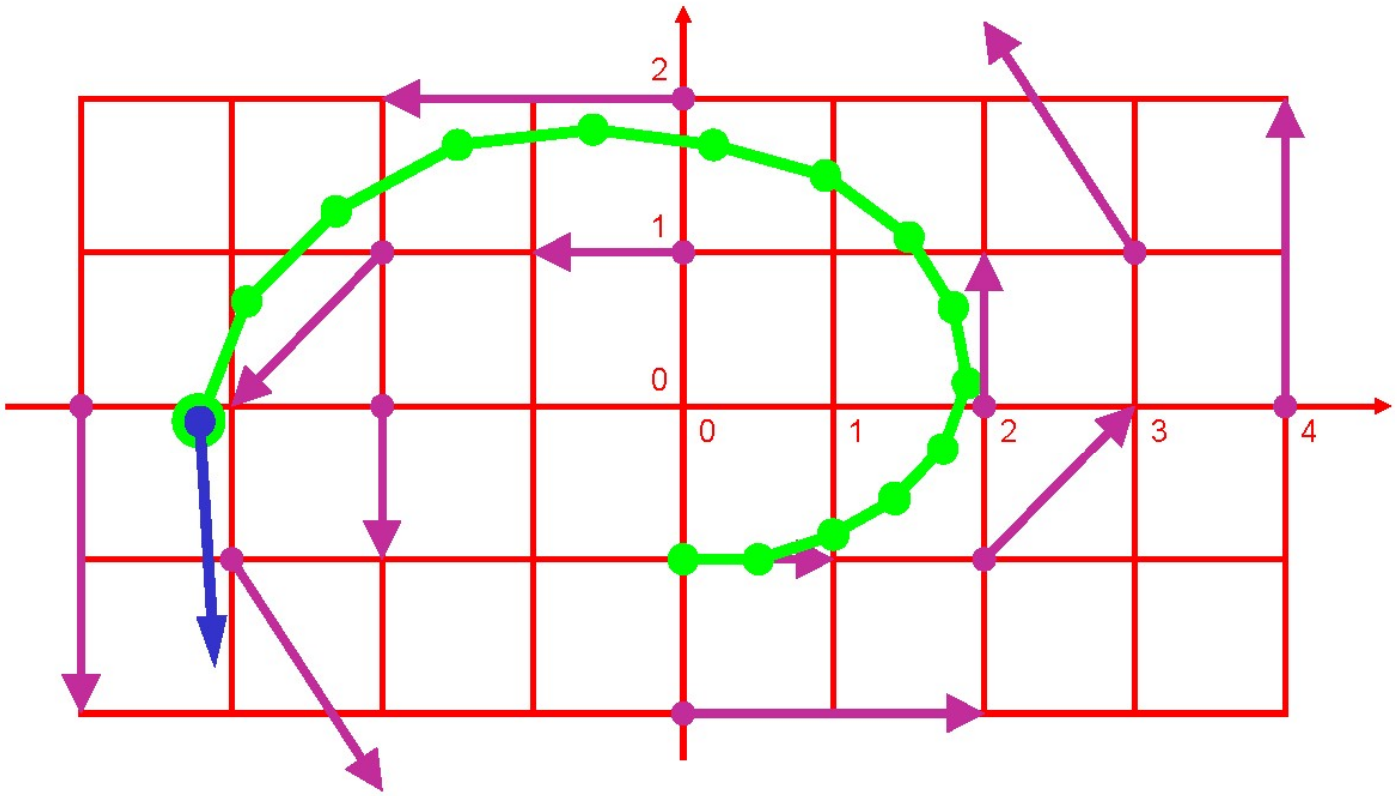
Euler Integration – Example

- $\mathbf{s}_9 \approx (0.20 | 1.69)^T$;
  $\mathbf{v}(\mathbf{s}_9) \approx (-1.69 | 0.10)^T$;



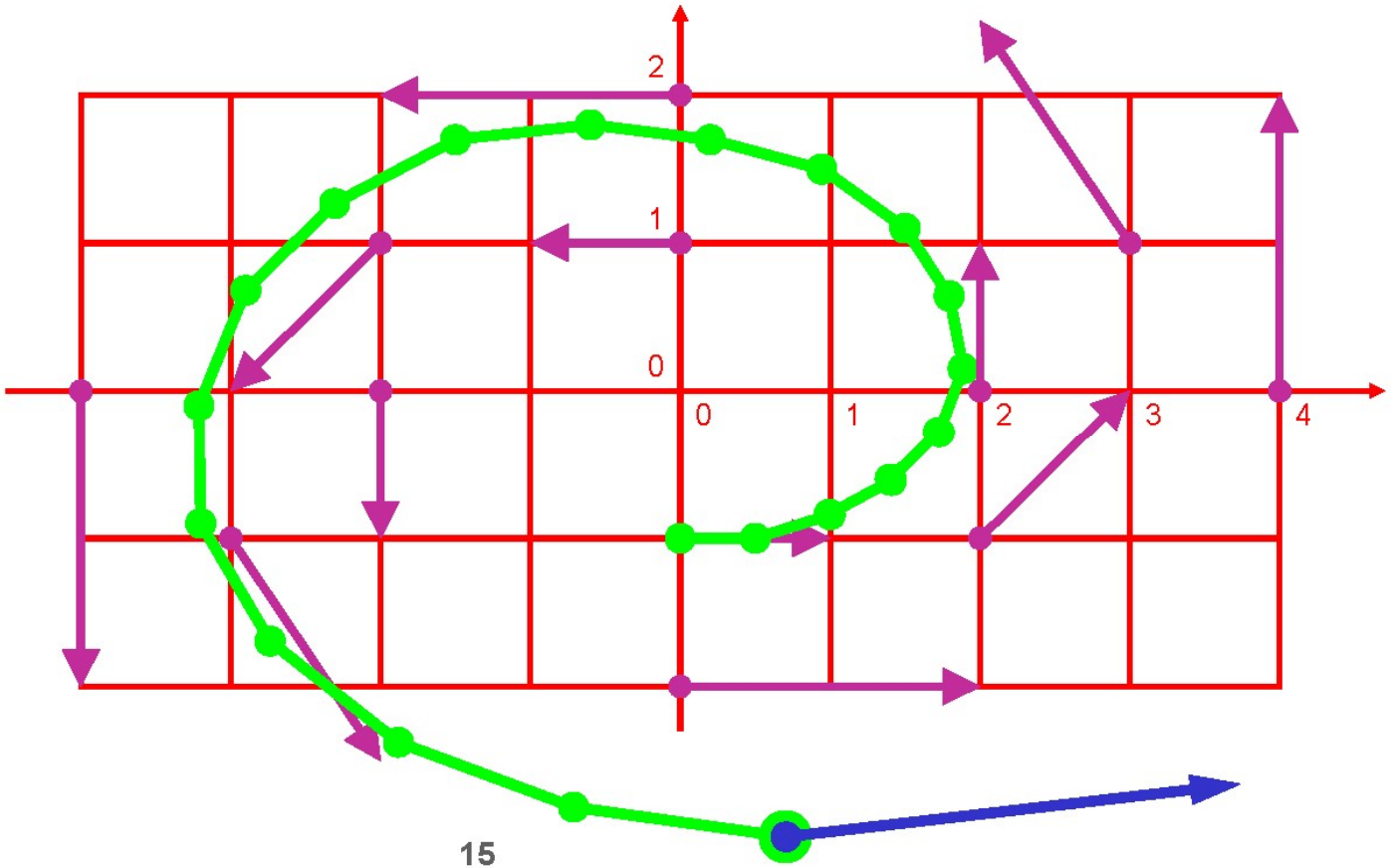
Euler Integration – Example

- $\mathbf{s}_{14} \approx (-3.22 | -0.10)^T;$
  $\mathbf{v}(\mathbf{s}_{14}) \approx (0.10 | -1.61)^T;$



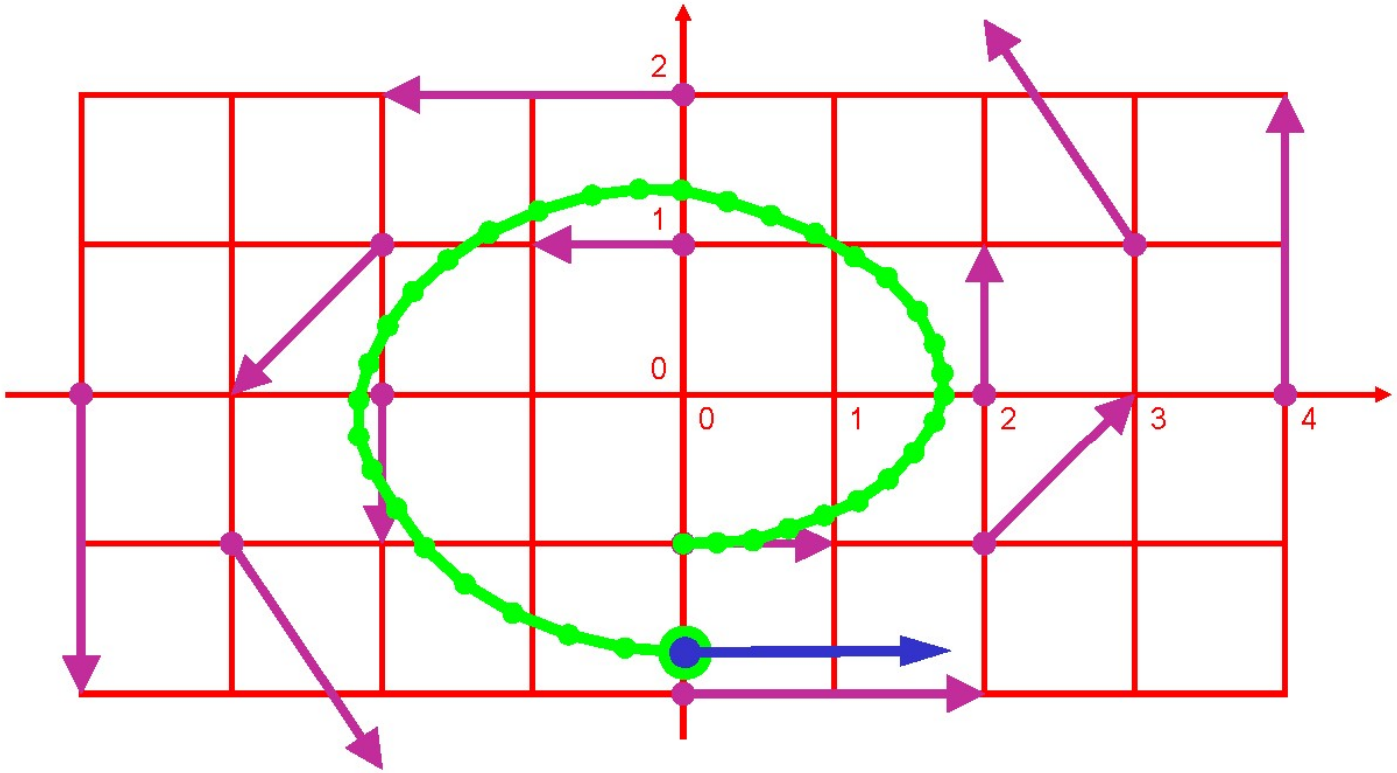
Euler Integration – Example

- $\mathbf{s}_{19} \approx (0.75|-3.02)^T$; $\mathbf{v}(\mathbf{s}_{19}) \approx (3.02|0.37)^T$;
  clearly: large integration error, d*t* too large!
  19 steps



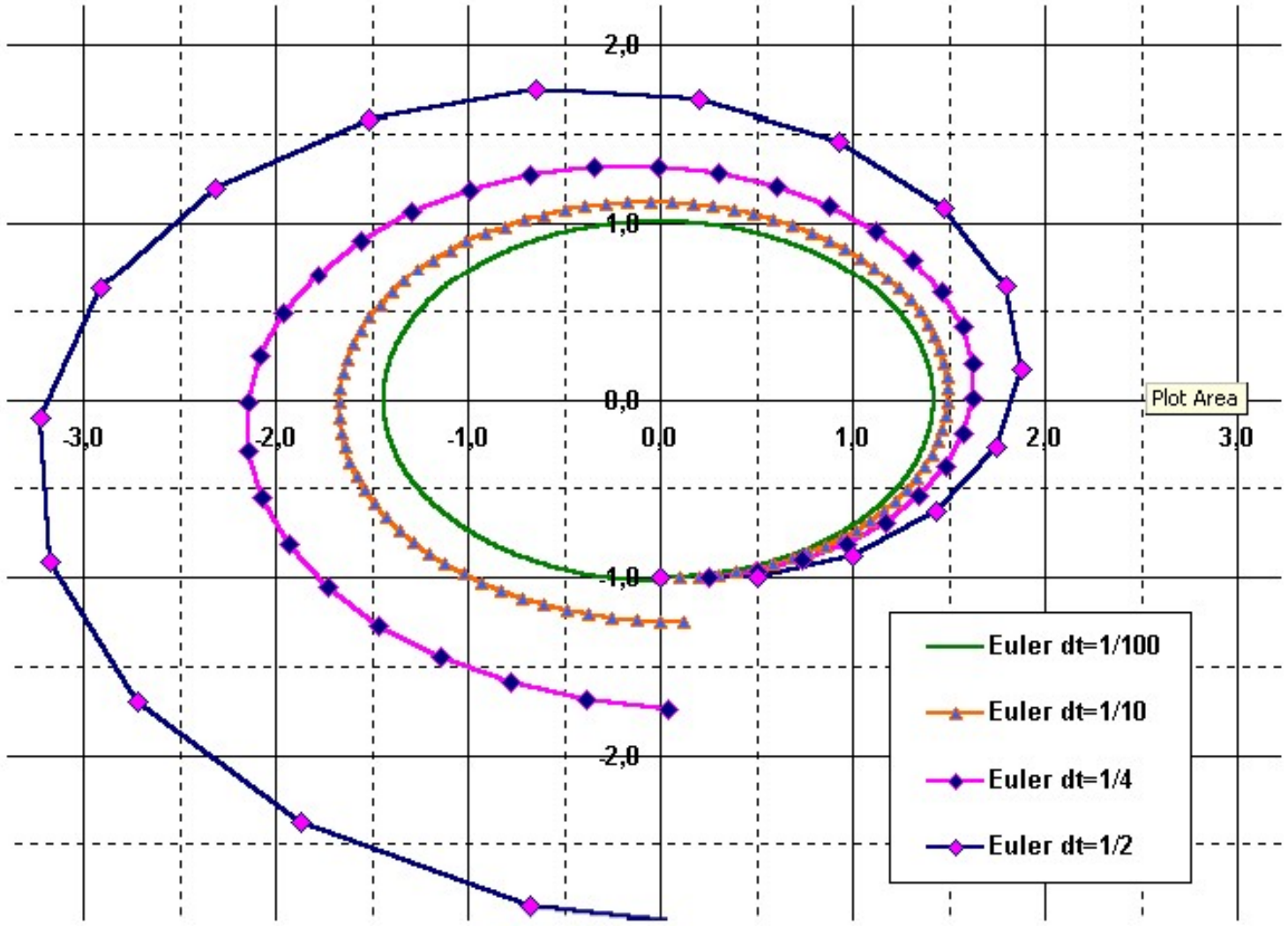
Euler Integration – Example

- d*t* smaller (1/4): more steps, more exact!
  $\mathbf{s}_{36} \approx (0.04 | -1.74)^T$; $\mathbf{v}(\mathbf{s}_{36}) \approx (1.74 | 0.02)^T$;
- 36 steps



Comparison Euler, Step Sizes

Euler is getting better proportionally to d*t*

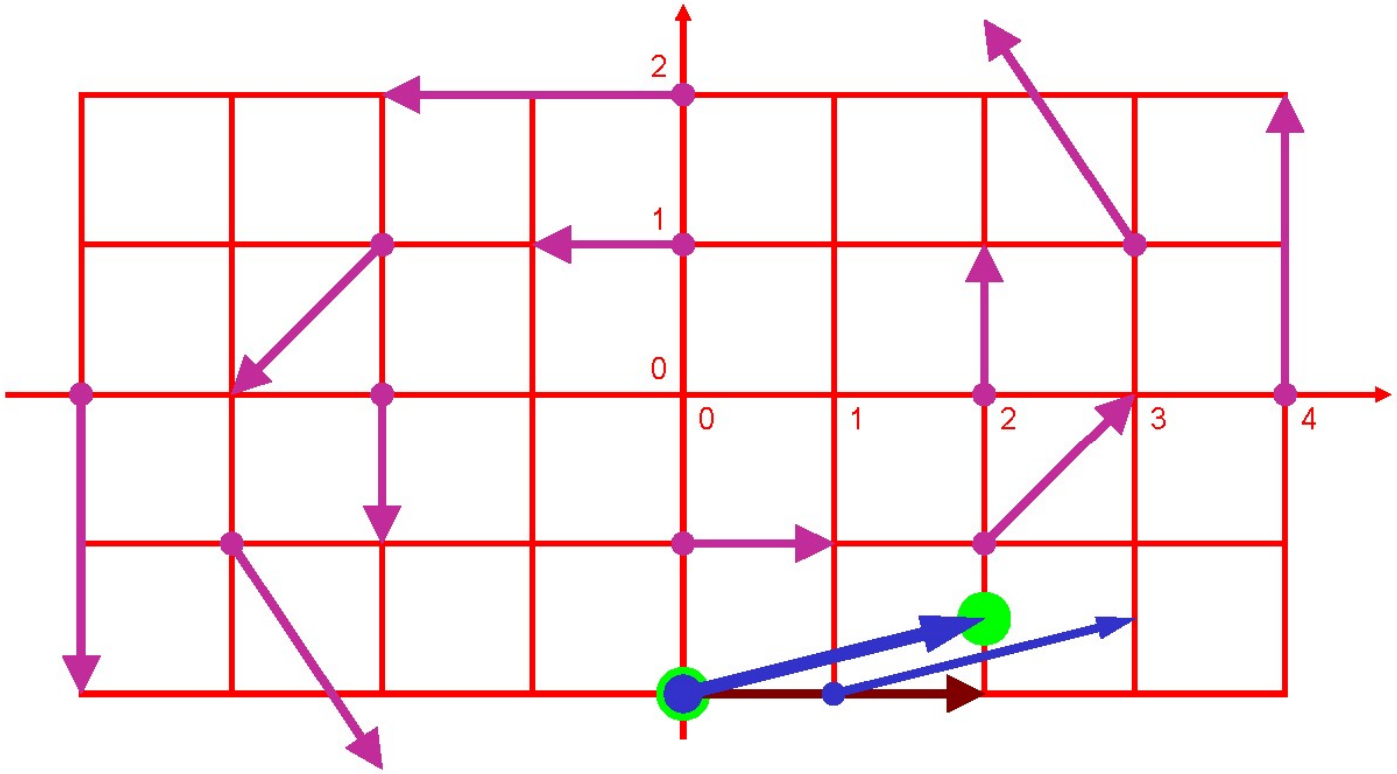


Better than Euler Integr.: RK

- Runge-Kutta Approach:
  - theory: $\mathbf{s}(t) = \mathbf{s}_0 + \int_{0 \le u \le t} \mathbf{v}(\mathbf{s}(u))\,\mathrm{d}u$
  - Euler: $\mathbf{s}_i = \mathbf{s}_0 + \sum_{0 \le u < i} \mathbf{v}(\mathbf{s}_u) \cdot \mathrm{d}t$
  - Runge-Kutta integration:
    - idea: cut short the curve arc
    - RK-2 (second order RK):
      1.: do half a Euler step
      2.: evaluate flow vector there
      3.: use it in the origin
    - RK-2 (two evaluations of **v** per step):
      $\mathbf{s}_{i+1} = \mathbf{s}_i + \mathbf{v}(\mathbf{s}_i + \mathbf{v}(\mathbf{s}_i) \cdot \mathrm{d}t/2) \cdot \mathrm{d}t$

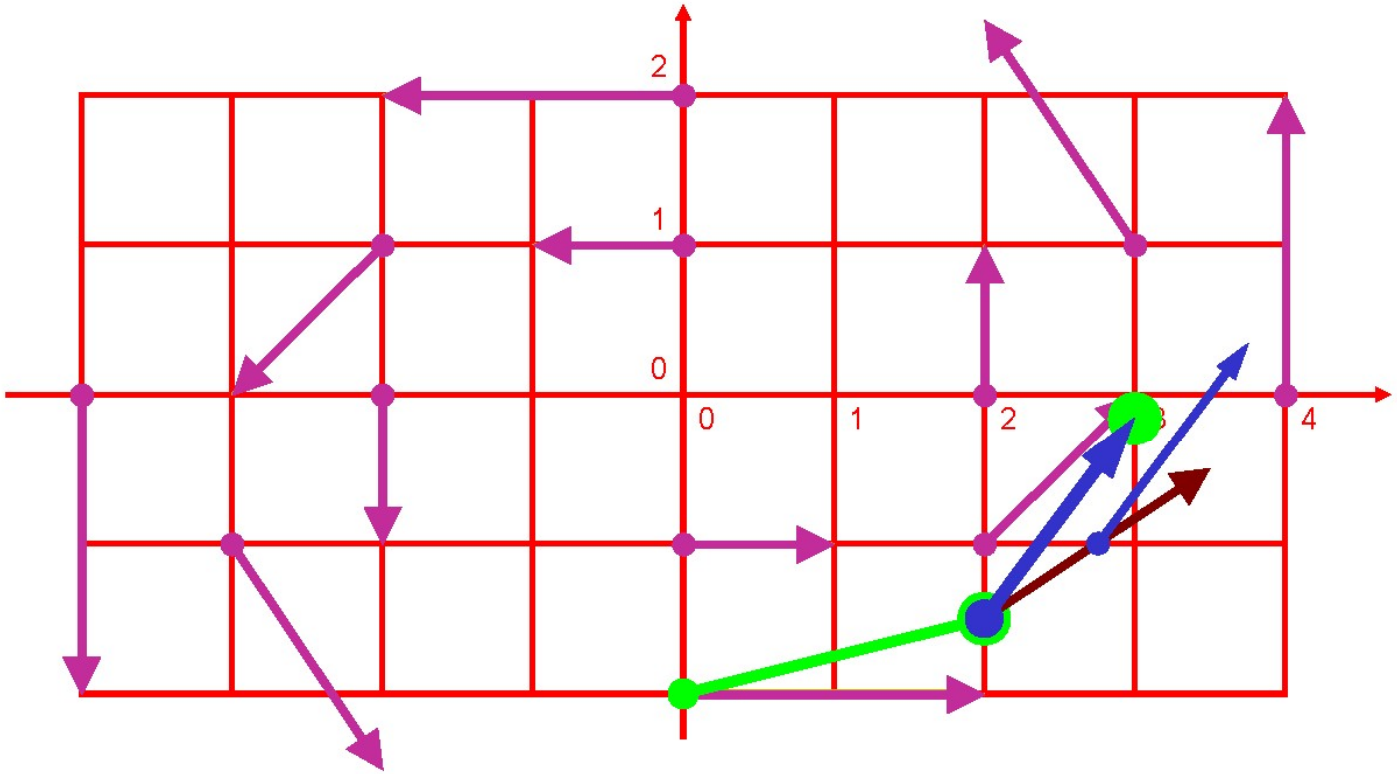
RK-2 Integration – One Step

- Seed point $\mathbf{s}_0 = (0|-2)^T$;
  current flow vector $\mathbf{v}(\mathbf{s}_0) = (2|0)^T$;
  preview vector $\mathbf{v}(\mathbf{s}_0+\mathbf{v}(\mathbf{s}_0)\cdot dt/2) = (2|0.5)^T$;
  $dt = 1$



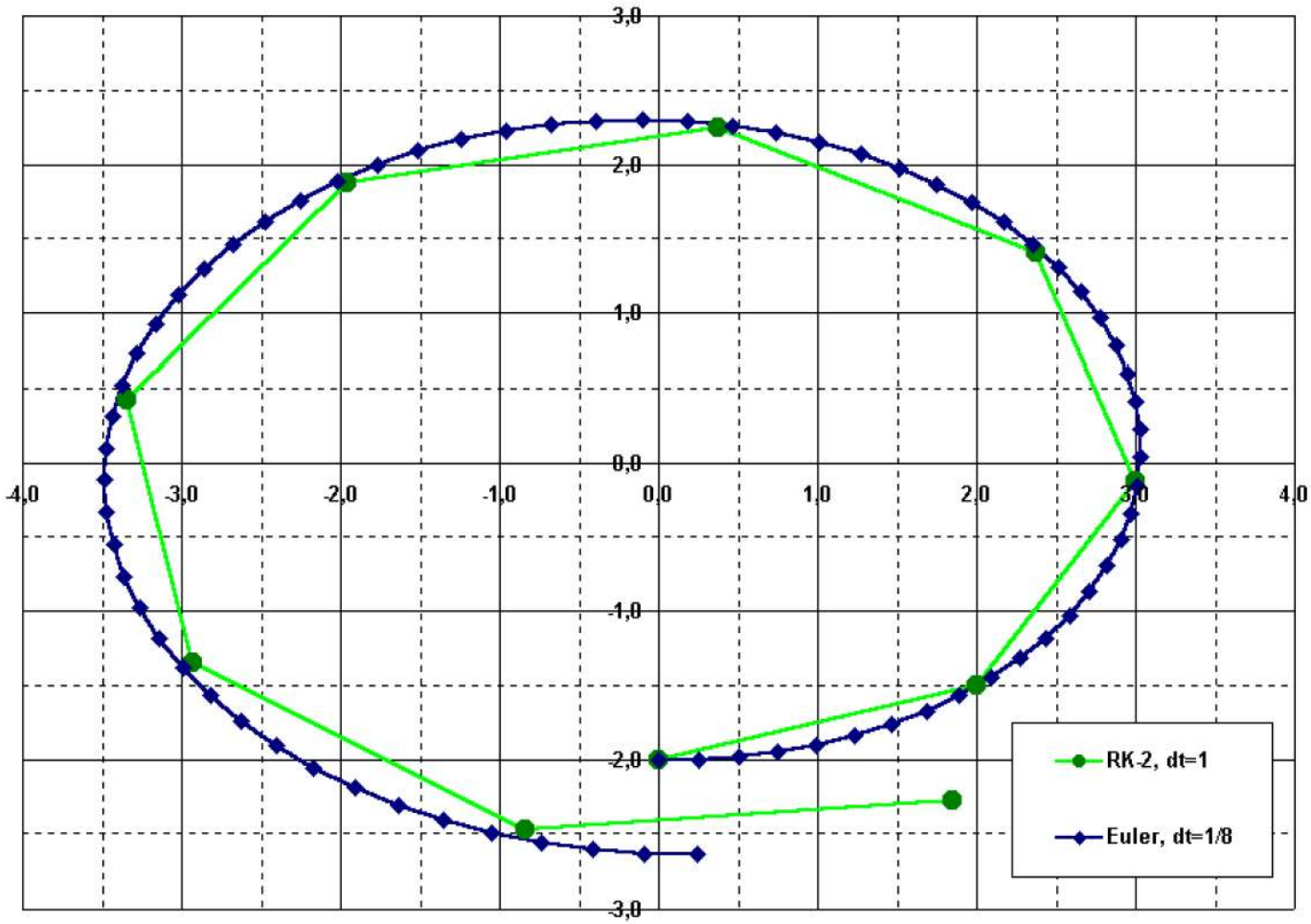
RK-2 – One more step

- Seed point $\mathbf{s}_1 = (2|-1.5)^T$;
  current flow vector $\mathbf{v}(\mathbf{s}_1) = (1.5|1)^T$;
  preview vector $\mathbf{v}(\mathbf{s}_1+\mathbf{v}(\mathbf{s}_1)\cdot dt/2) \approx (1|1.4)^T$;
  $dt = 1$



RK-2 – A Quick Round

- RK-2: even with dt=1 (9 steps) better than Euler with dt=1/8 (72 steps)

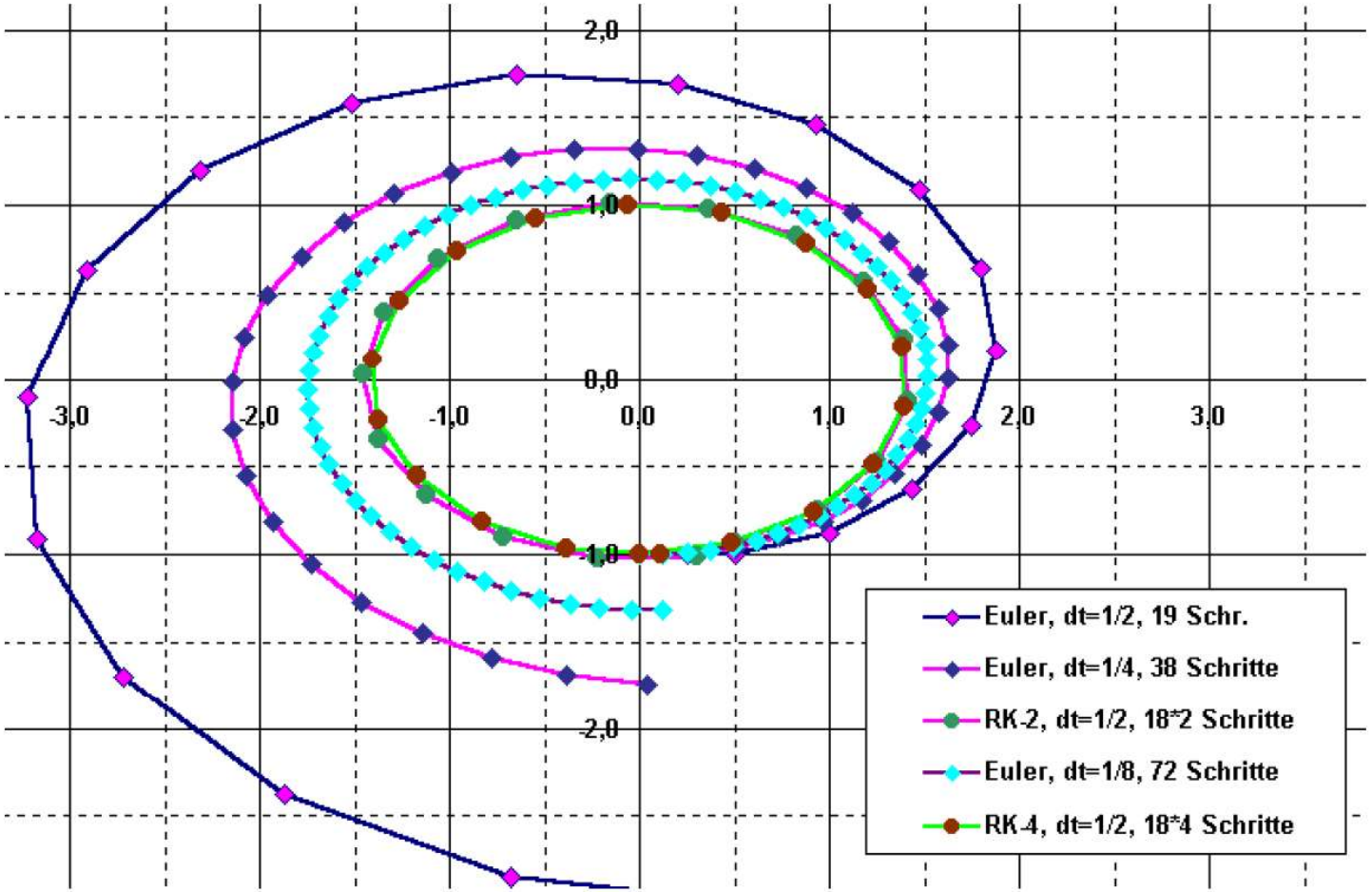


RK-4 vs. Euler, RK-2

- Even better: fourth order RK:
  - four vectors **a**, **b**, **c**, **d**
  - one step is a convex combination:
    $\mathbf{s}_{i+1} = \mathbf{s}_i + (\mathbf{a} + 2\cdot\mathbf{b} + 2\cdot\mathbf{c} + \mathbf{d})/6$
  - vectors:
    - $\mathbf{a} = dt\cdot\mathbf{v}(\mathbf{s}_i)$ … original vector
    - $\mathbf{b} = dt\cdot\mathbf{v}(\mathbf{s}_i+\mathbf{a}/2)$ … RK-2 vector
    - $\mathbf{c} = dt\cdot\mathbf{v}(\mathbf{s}_i+\mathbf{b}/2)$ … use RK-2 …
    - $\mathbf{d} = dt\cdot\mathbf{v}(\mathbf{s}_i+\mathbf{c})$ … and again!

Euler vs. Runge-Kutta

- RK-4: pays off only with complex flows
- Here approx. like RK-2



Integration, Conclusions

- Summary:
  - analytic determination of streamlines usually not possible
  - hence: numerical integration
  - several methods available (Euler, Runge-Kutta, etc.)
  - Euler: simple, imprecise, esp. with small dt
  - RK: more accurate in higher orders
  - furthermore: adaptive methods, implicit methods, etc.

Thank you.

Thanks for material

- Helwig Hauser
- Eduard Gröller
- Daniel Weiskopf
- Torsten Möller
- Ronny Peikert
- Philipp Muigg
- Christof Rezk-Salama