CS**5643**

04 ODEs and procedural turbulence

Steve Marschner
Cornell University
Spring 2023

Towards higher order

Let's try expanding around $t = (t_k + t_{k+1})/2$; call this time t_m for "midpoint"

- $\mathbf{y}(t) = \mathbf{y}(t_m) + \dot{\mathbf{y}}(t_m)(t - t_m) + \frac{1}{2}\ddot{\mathbf{y}}(t_m)(t - t_m)^2 + O((t - t_m)^3)$

evaluate at t_k and t_{k+1} to compute the step increment

- $\mathbf{y}(t_{k+1}) - \mathbf{y}(t_k) = h\dot{\mathbf{y}}(t_m) + O(h^3)$ (try it yourself to see the canceling h^2 term)

…if only we knew $\mathbf{y}(t_m)$! But we can use Forward Euler to estimate it

- $\mathbf{y}(t_m) = \mathbf{y}(t_k) + \frac{h}{2}\dot{\mathbf{y}}(t_k) + O(h^2)$, so let $\mathbf{y}_m = \mathbf{y}_k + \frac{h}{2}\mathbf{f}(\mathbf{y}_k)$
- $\mathbf{f}(\mathbf{y} + O(h^2)) = \mathbf{f}(\mathbf{y}) + \mathbf{f}'(\mathbf{y})O(h^2) + O(h^2) = \mathbf{f}(\mathbf{y}) + O(h^2)$, so $\mathbf{f}(\mathbf{y}_m) = \dot{\mathbf{y}}(t_m) + O(h^2)$
- then $\mathbf{y}(t_{k+1}) = \mathbf{y}(t_k) + h(\mathbf{f}(\mathbf{y}_\mathbf{m}) + O(h^2)) + O(h^3) = \mathbf{y}(t_k) + h\mathbf{f}(\mathbf{y}_\mathbf{m}) + O(h^3)$
- so let $\mathbf{y}_{k+1} = \mathbf{y}_k + h\mathbf{f}(\mathbf{y}_m)$ and $\mathbf{y}_{k+1}$ is a second-order estimate of $\mathbf{y}(t_{k+1})$

Midpoint method

Timestep equations

$$\mathbf{y}_m = \mathbf{y}_k + \frac{h}{2}\mathbf{f}(\mathbf{y}_k)$$

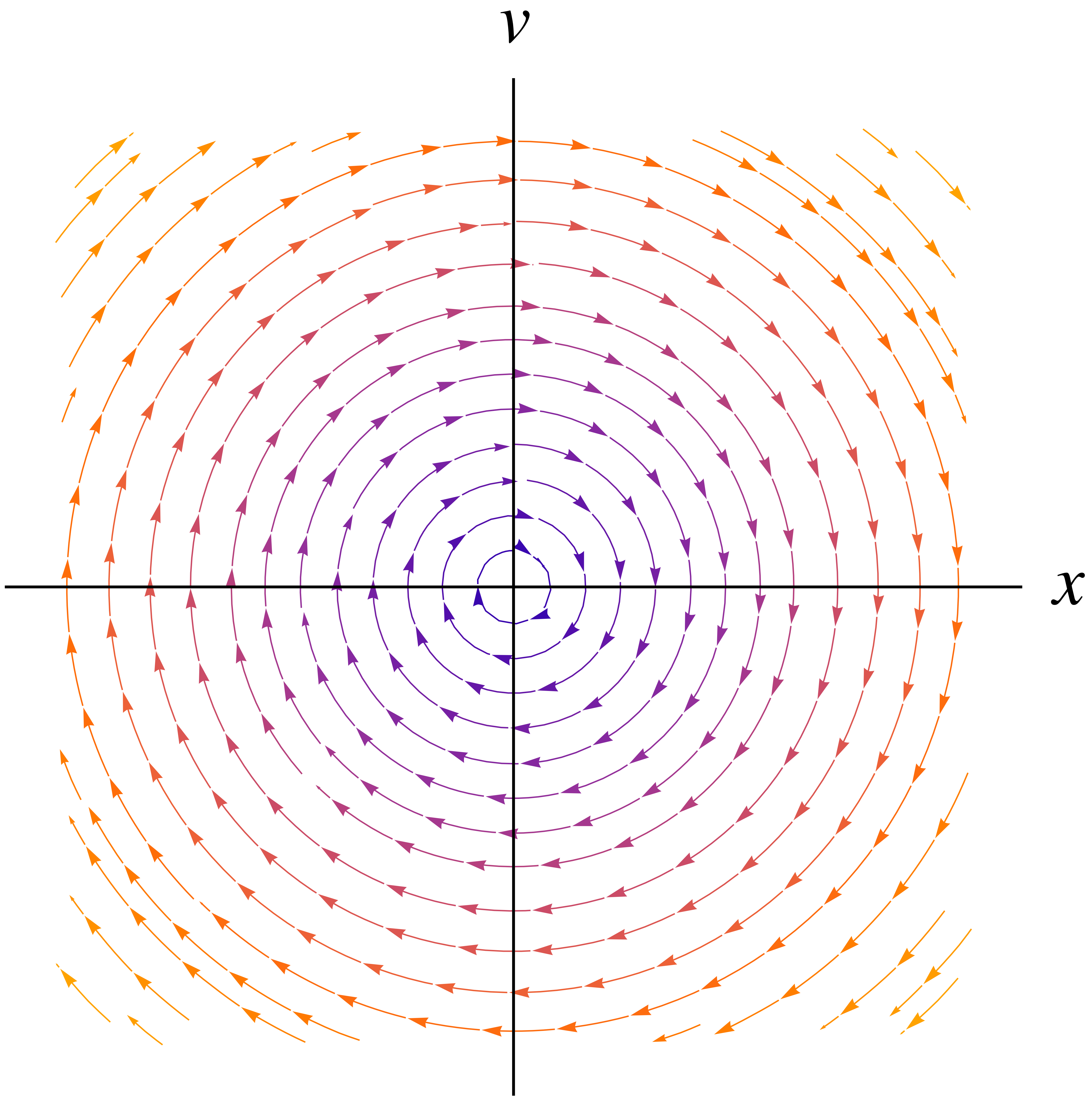
$$\mathbf{y}_{k+1} = \mathbf{y}_k + h\mathbf{f}(\mathbf{y}_m)$$

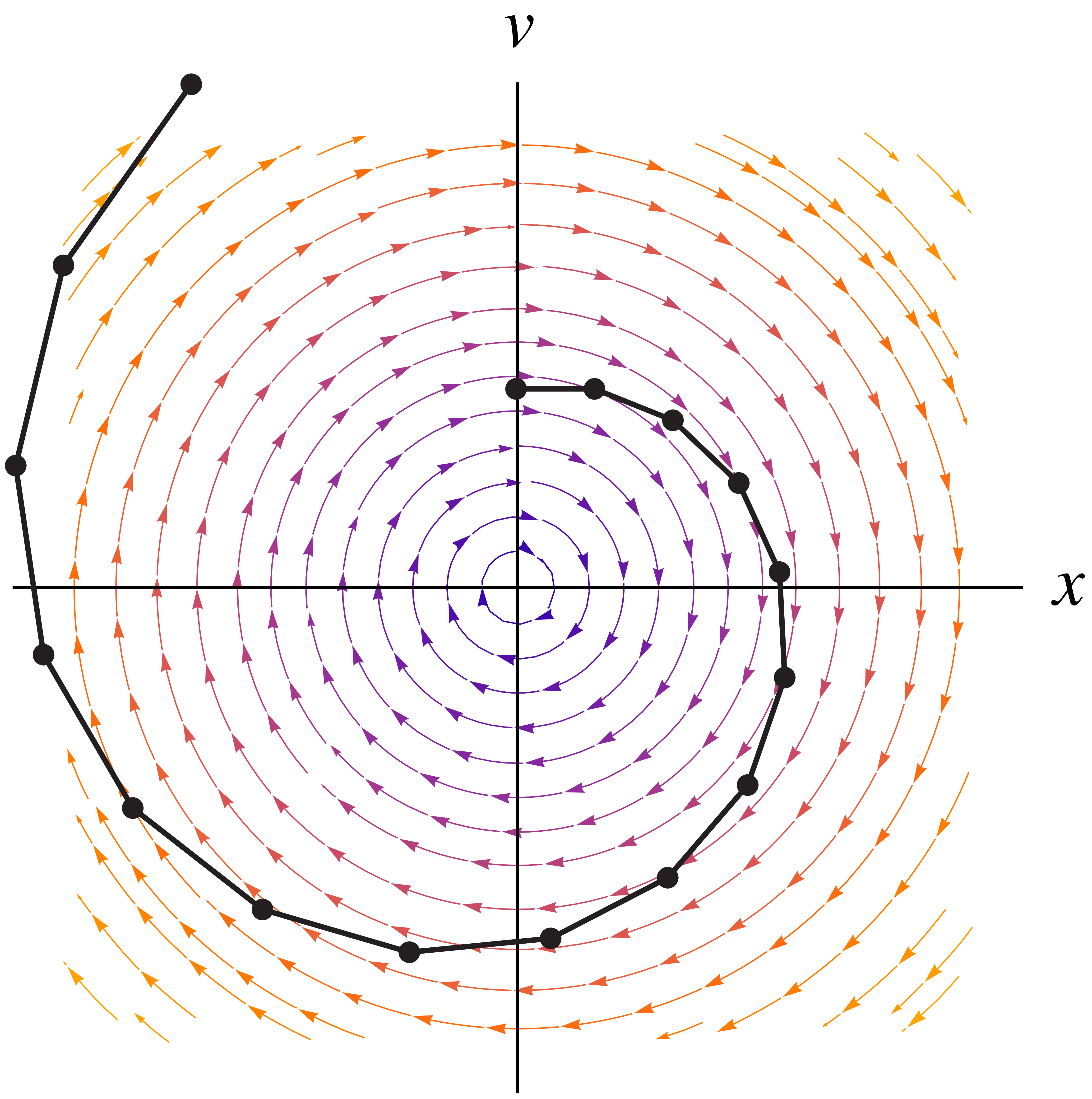
This is

- an explicit integrator
- a *two-step* integrator (requires two evaluations of $\mathbf{f}$)
- accurate to second order

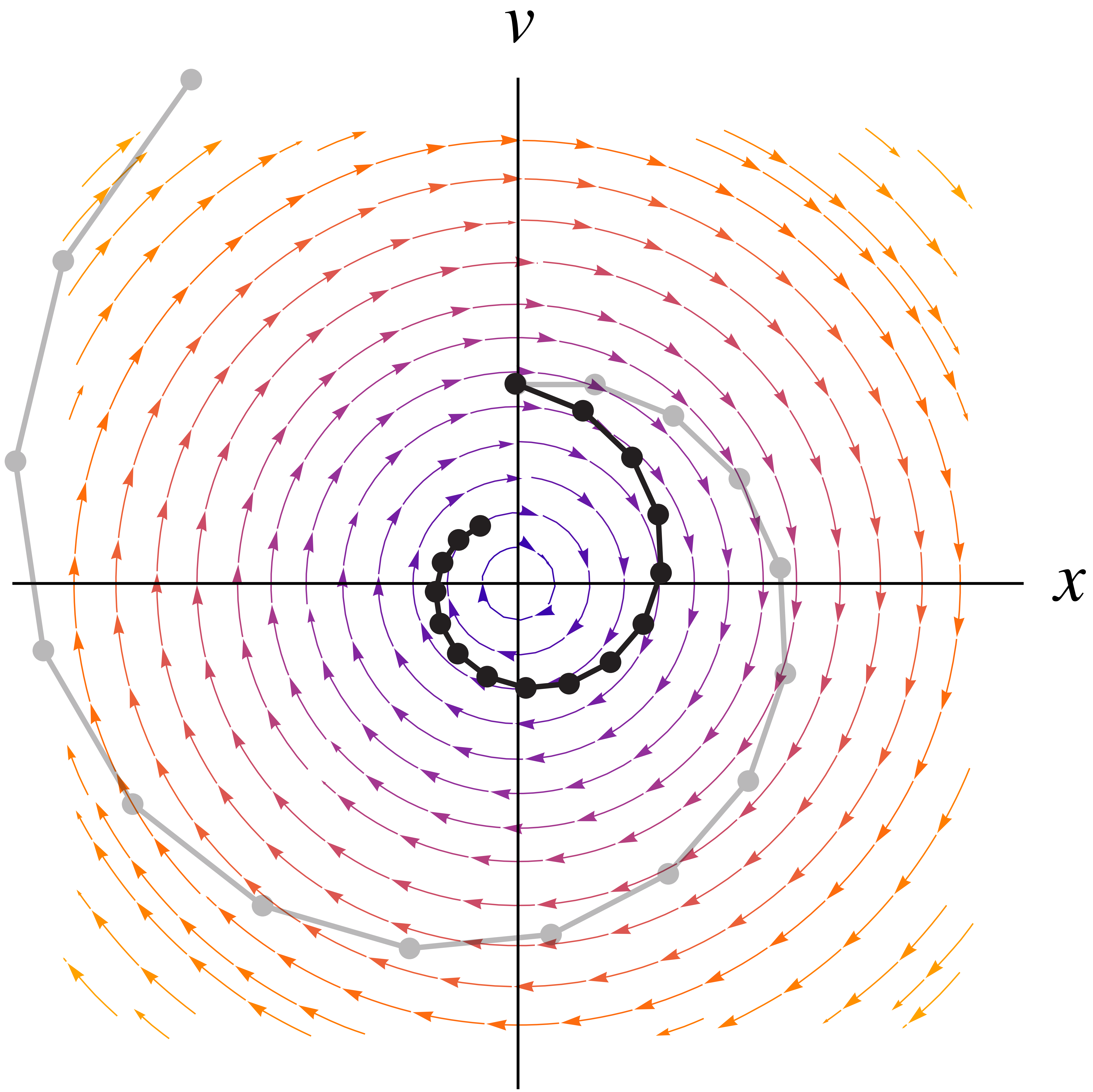
It’s also the first in a family of higher order integrators

- Runge-Kutta methods achieve order p accuracy with at least p function evaluations
- RK4 is a popular fourth-order scheme, good for smooth problems requiring high accuracy
- animation = not-so-smooth problems requiring low accuracy, hence we rarely go past second order

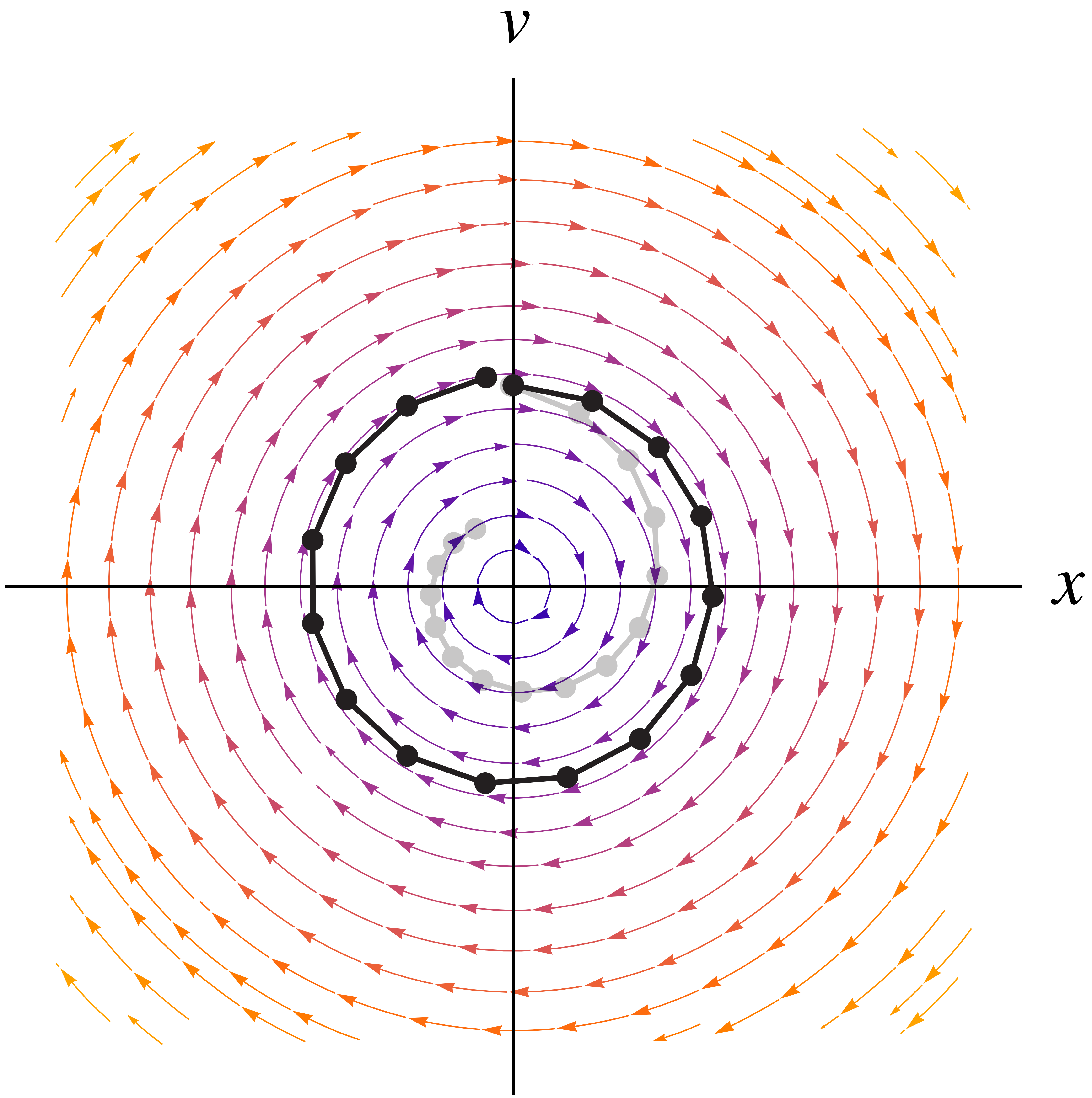

v
x



forward Euler



backward Euler



midpoint method

Demo!

accuracy of integration along circular paths

- Euler vs. midpoint

Integrators for second-order systems

Many useful systems have the form $\ddot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t))$

- note this equation skips over $\dot{\mathbf{x}}$; acceleration does not depend on velocity, only position.

Look at what the second step of the midpoint method does

- $\mathbf{y}_{k+1} = \mathbf{y}_k + h\mathbf{f}(\mathbf{y}_m)$ translates to (naming $\mathbf{y}_m$ as $\mathbf{y}_{k+0.5}$)

$$\mathbf{x}_{k+1} = \mathbf{x}_k + h\mathbf{v}_{k+0.5}$$
$$\mathbf{v}_{k+1} = \mathbf{v}_k + \mathbf{f}(\mathbf{x}_{k+0.5})$$

⟵ updating $\mathbf{x}$ only requires $\mathbf{v}_{k+0.5}$, and updating $\mathbf{v}$ only requires $\mathbf{x}_{k+0.5}$

- if we stagger the grids then we can have these values already!

$$\mathbf{x}_{k+1} = \mathbf{x}_k + h\mathbf{v}_{k+0.5}$$
$$\mathbf{v}_{k+1.5} = \mathbf{v}_{k+0.5} + h\mathbf{f}(\mathbf{x}_{k+1})$$

- this is an explicit method, and it's second order accurate for both position and velocity
- known as the Leapfrog integrator — elegant but prohibits velocity dependent forces

Symplectic Euler's method (aka. semi-implicit)

Leapfrog is nice but doesn't work for $\ddot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{v})$

- practical problem: can't evaluate $\mathbf{f}$ without knowing $\mathbf{x}$ and $\mathbf{v}$ at the same time
- a practical solution: give up the interleaved steps but keep the timestep equations

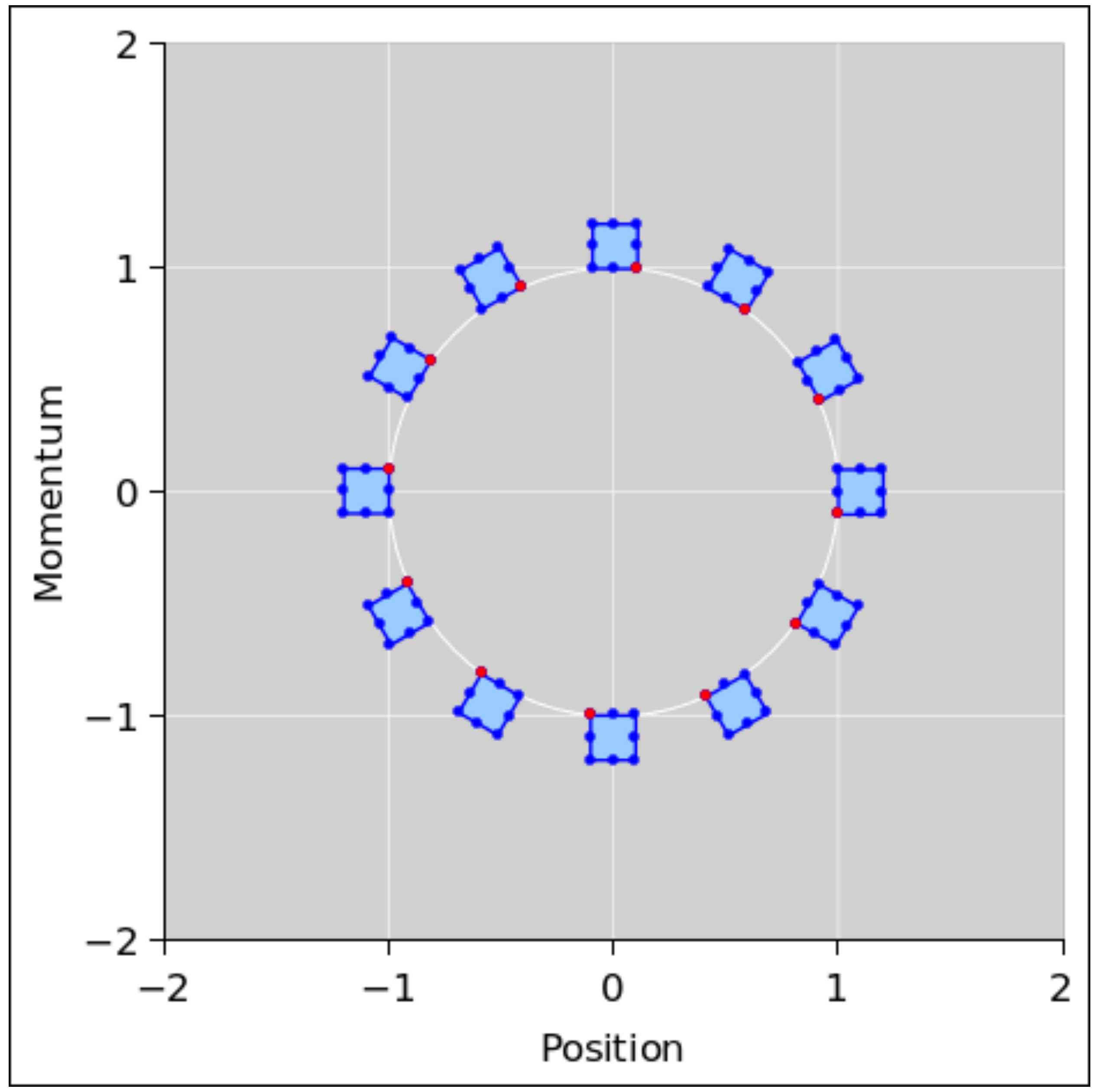
$$\mathbf{x}_{k+1} = \mathbf{x}_k + h\mathbf{v}_k$$
$$\mathbf{v}_{k+1} = \mathbf{v}_k + h\mathbf{f}(\mathbf{x}_{k+1})$$

this looks just like Forward Euler except for the last +1

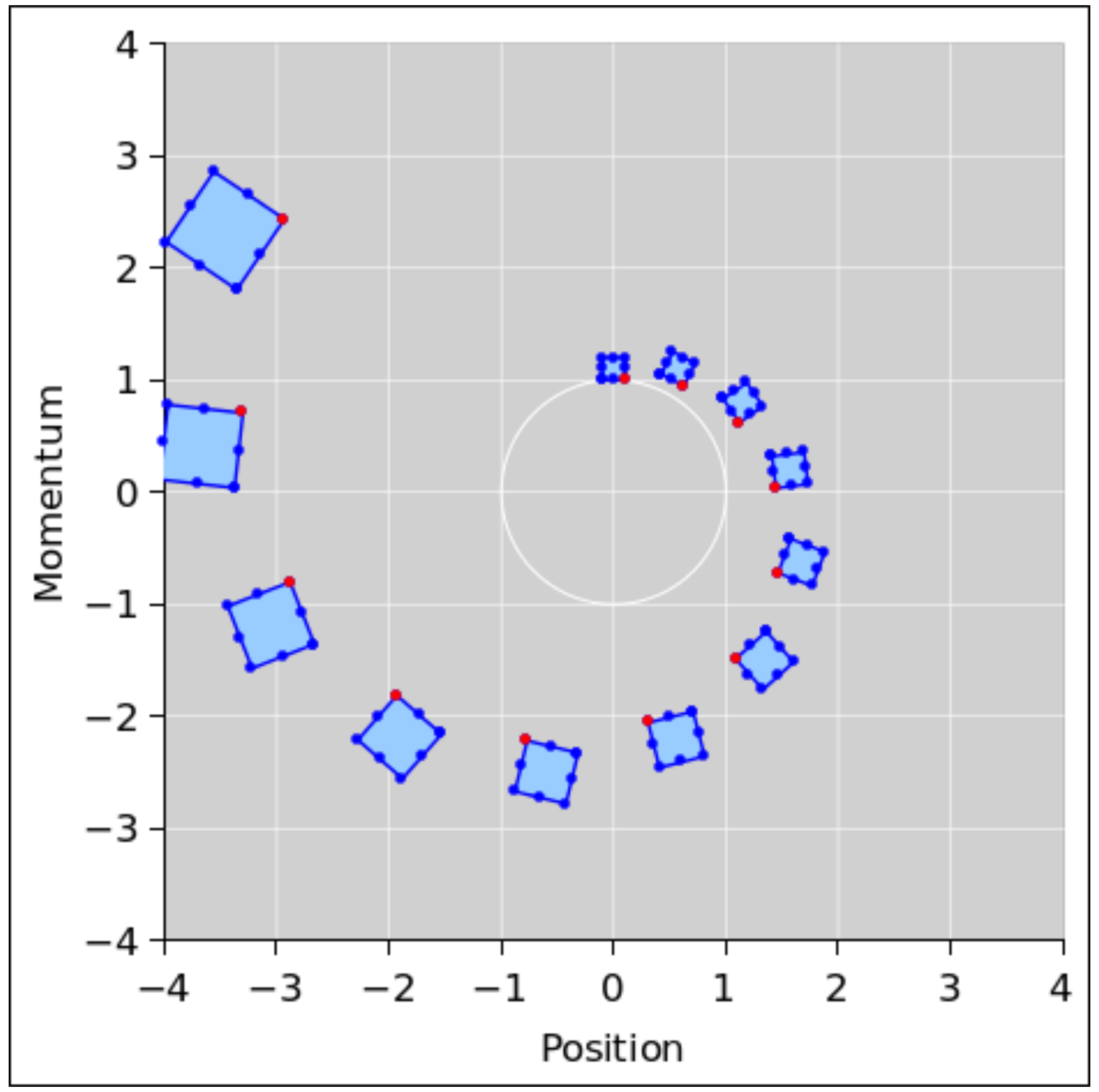
- or: use the position update from Forward Euler and the velocity update from Backward Euler
- this integrator shares a very nice property with Leapfrog: each timestep preserves area in the $(\mathbf{x}, \mathbf{v})$ picture (really in position–momentum space)

$$\begin{bmatrix}\mathbf{x}_{k+1} \\ \mathbf{v}_{k+1}\end{bmatrix} = \underbrace{\begin{bmatrix}1 & h \\ -h & 1-h^2\end{bmatrix}}_{\det = 1} \begin{bmatrix}\mathbf{x}_k \\ \mathbf{v}_k\end{bmatrix}$$

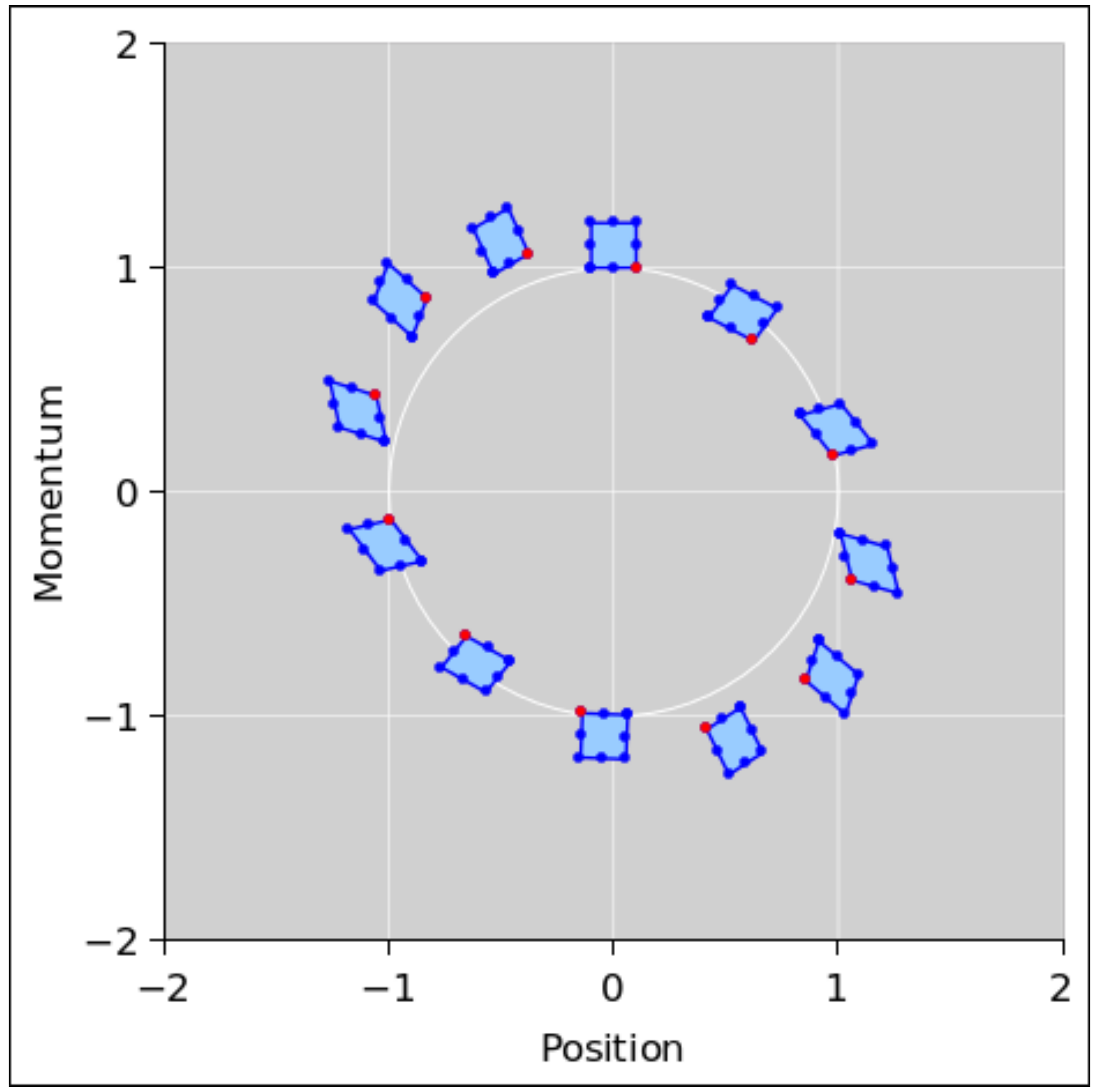
- this property holds for any Hamiltonian (roughly, energy conserving) system



exact

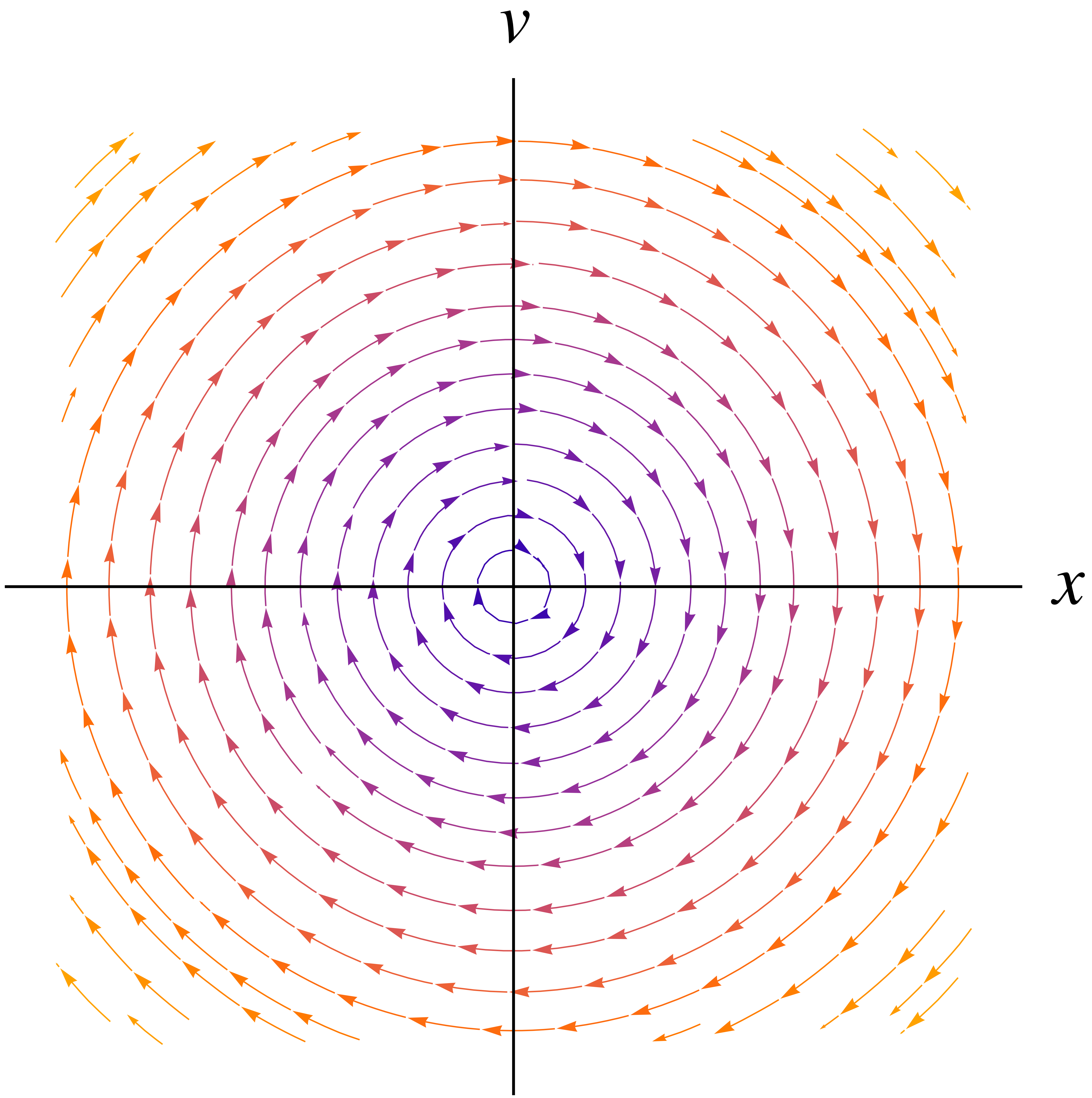


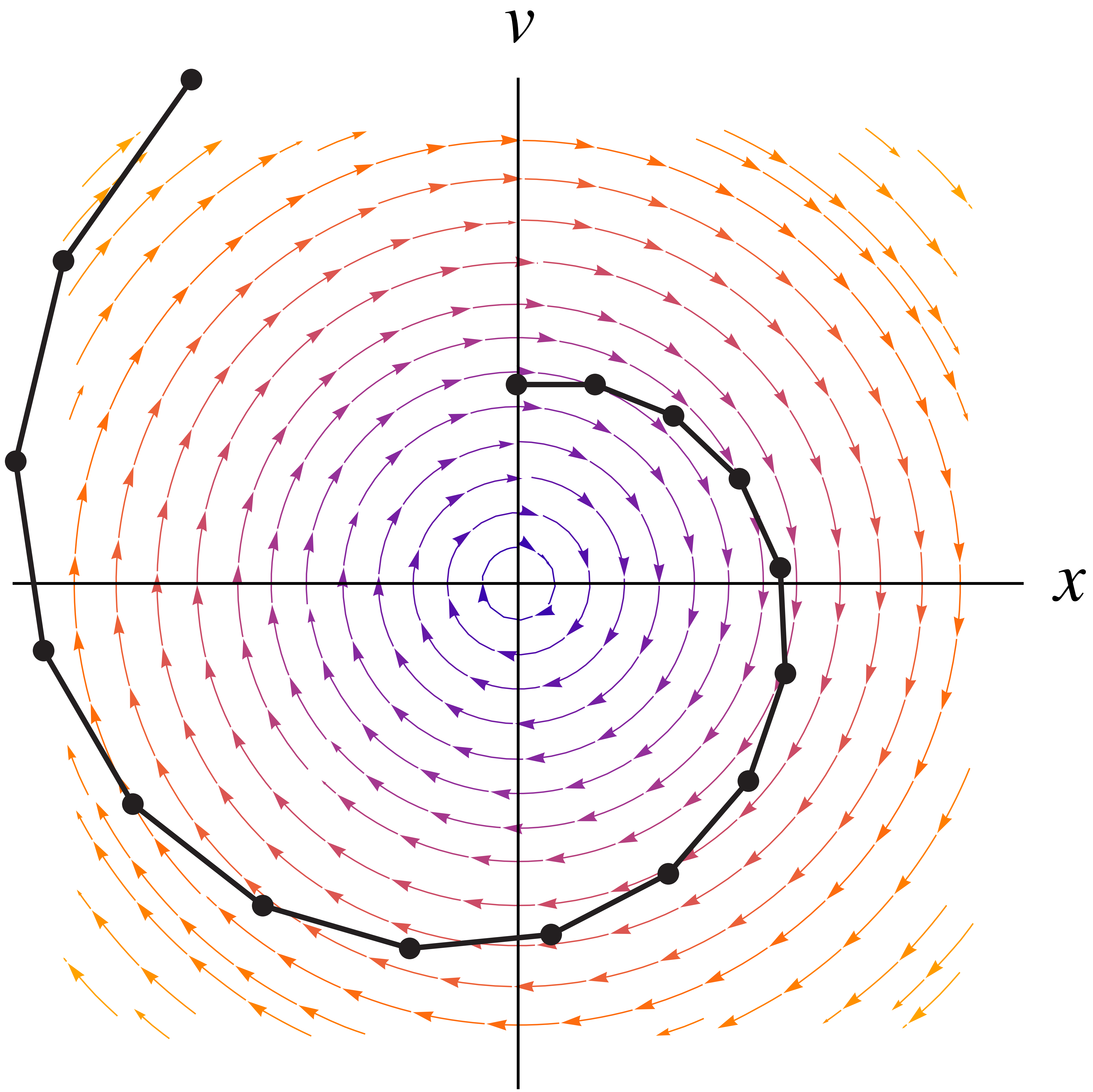
forward Euler



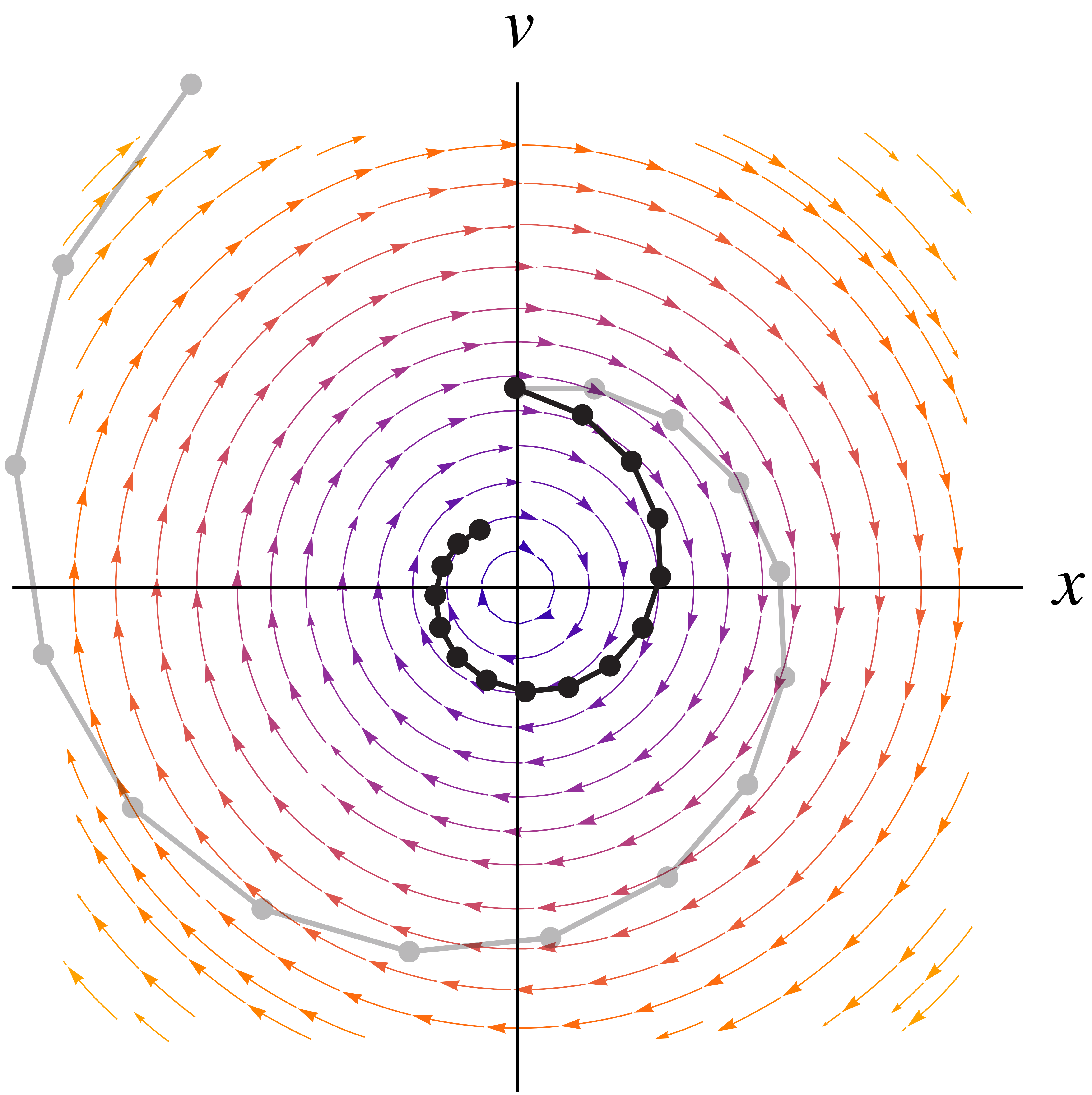
symplectic Euler

https://www.av8n.com/physics/symplectic-integrator.htm

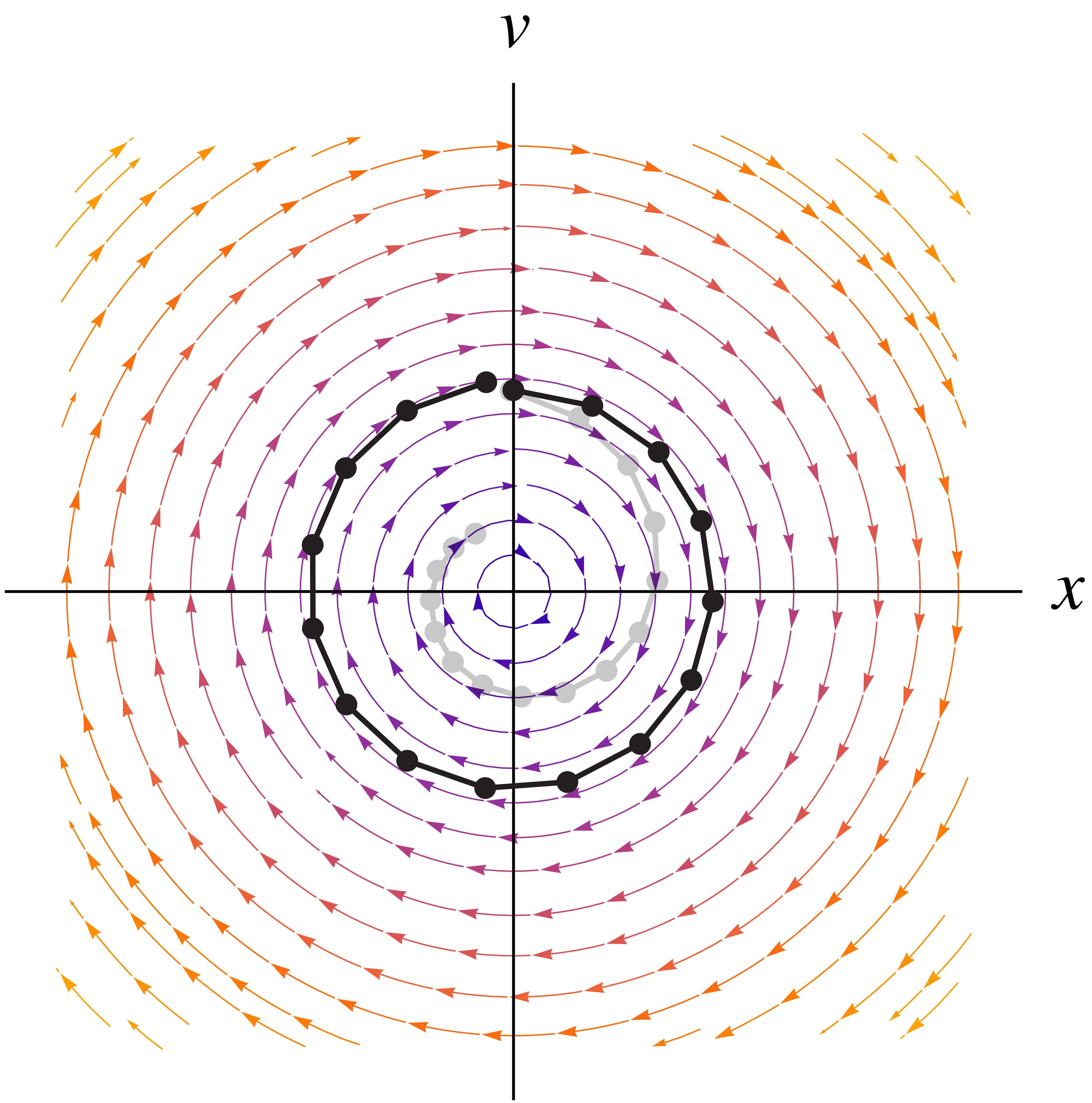

v
x



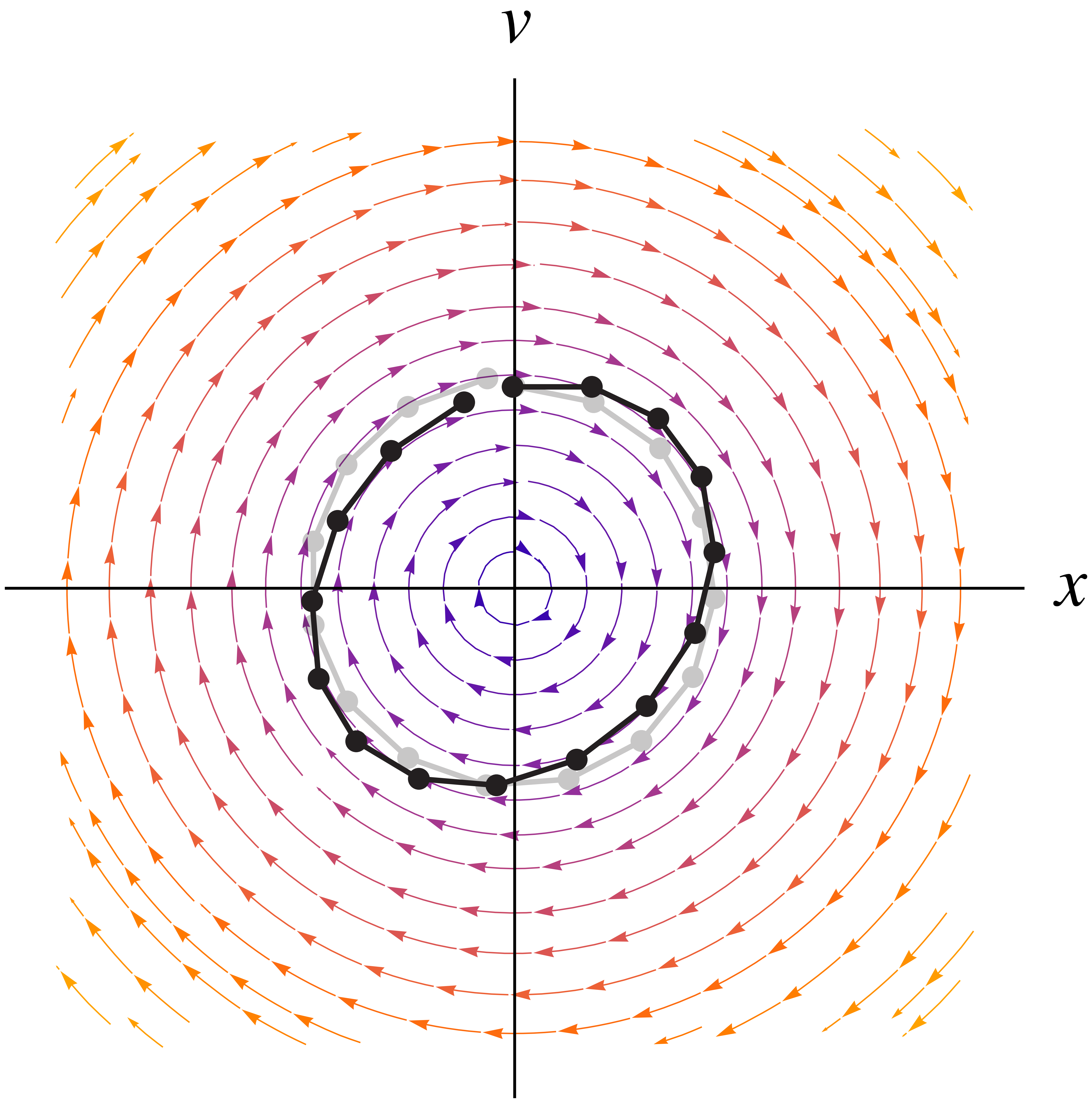
forward Euler



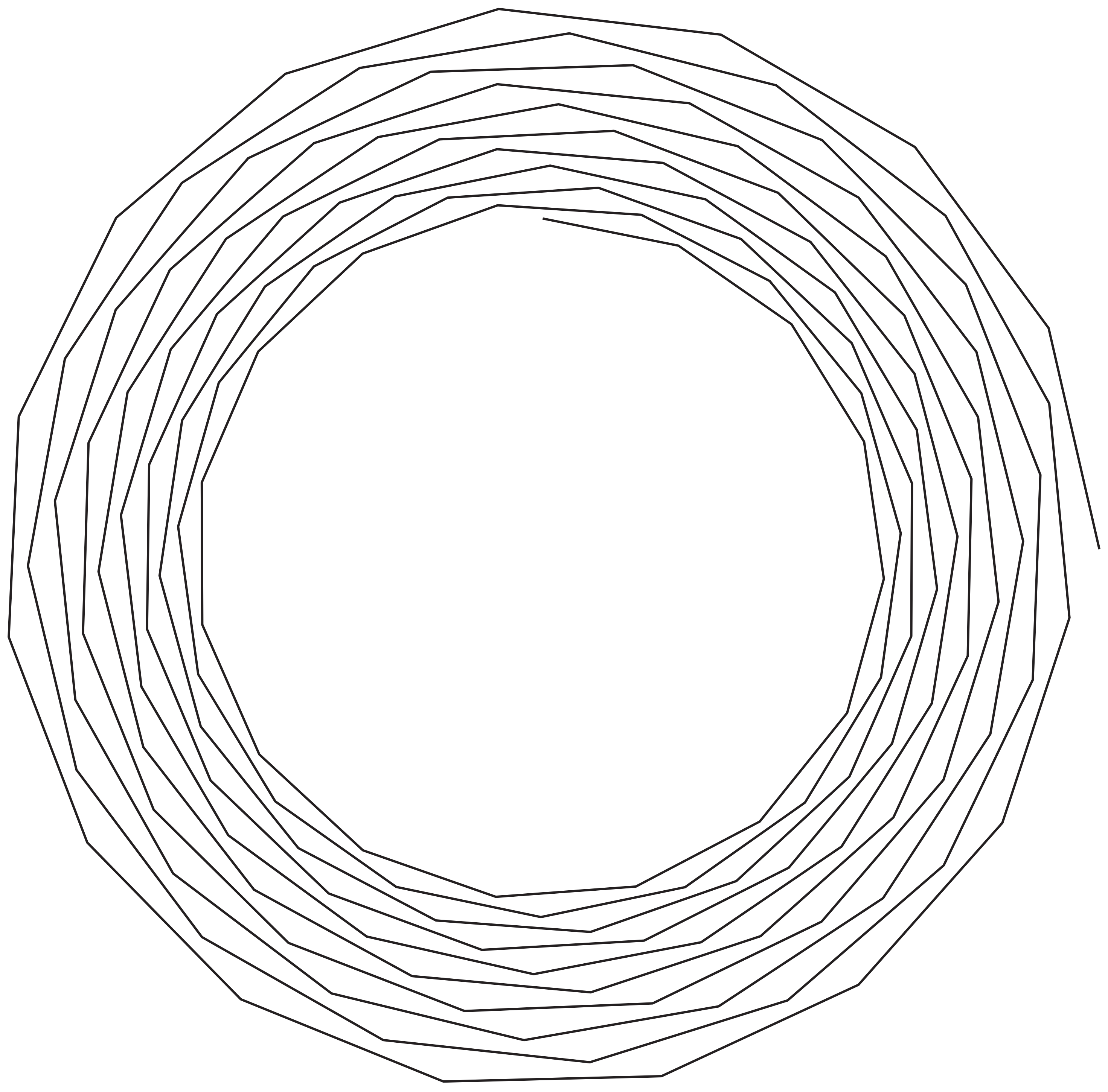
backward Euler



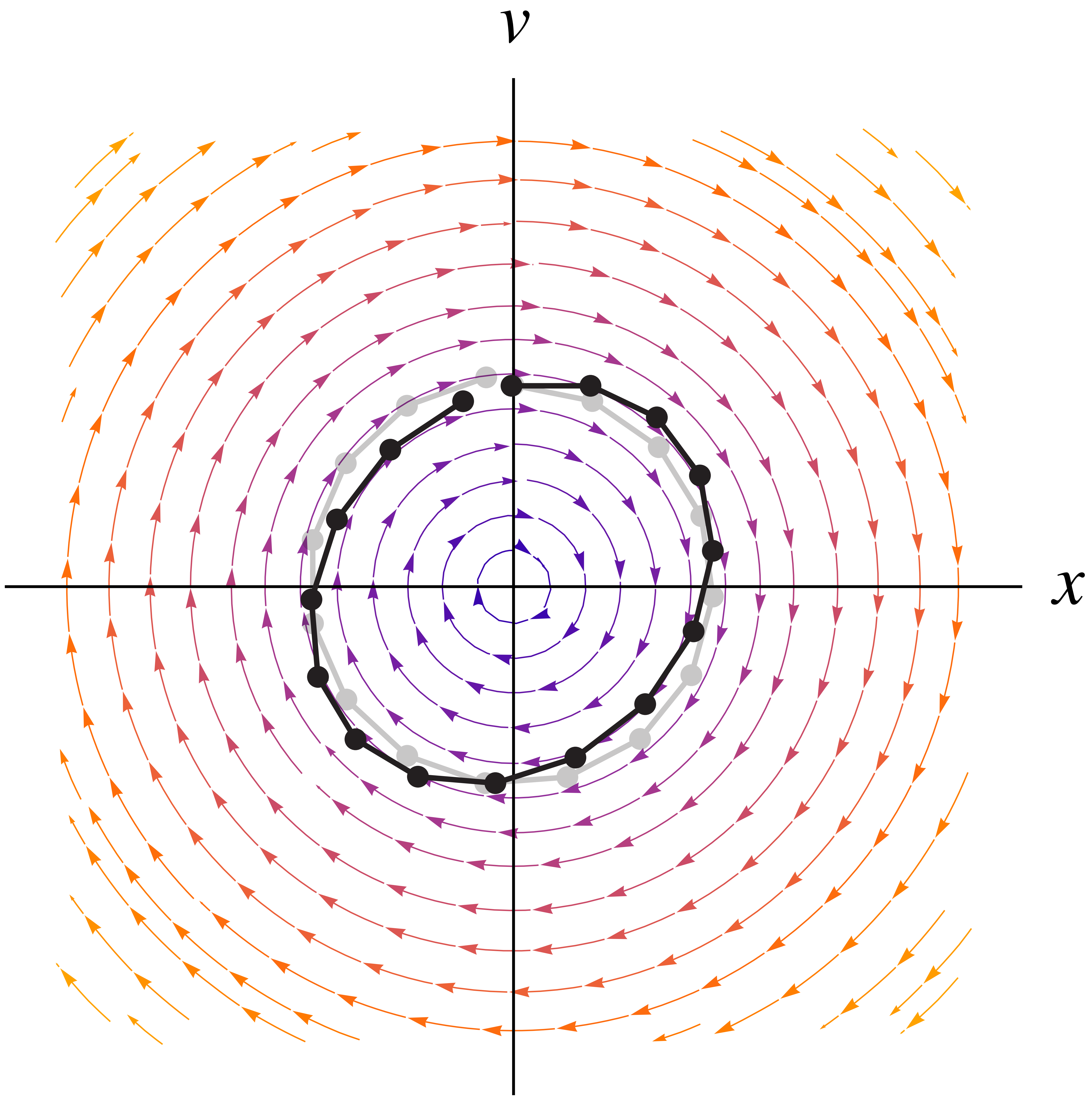
midpoint method



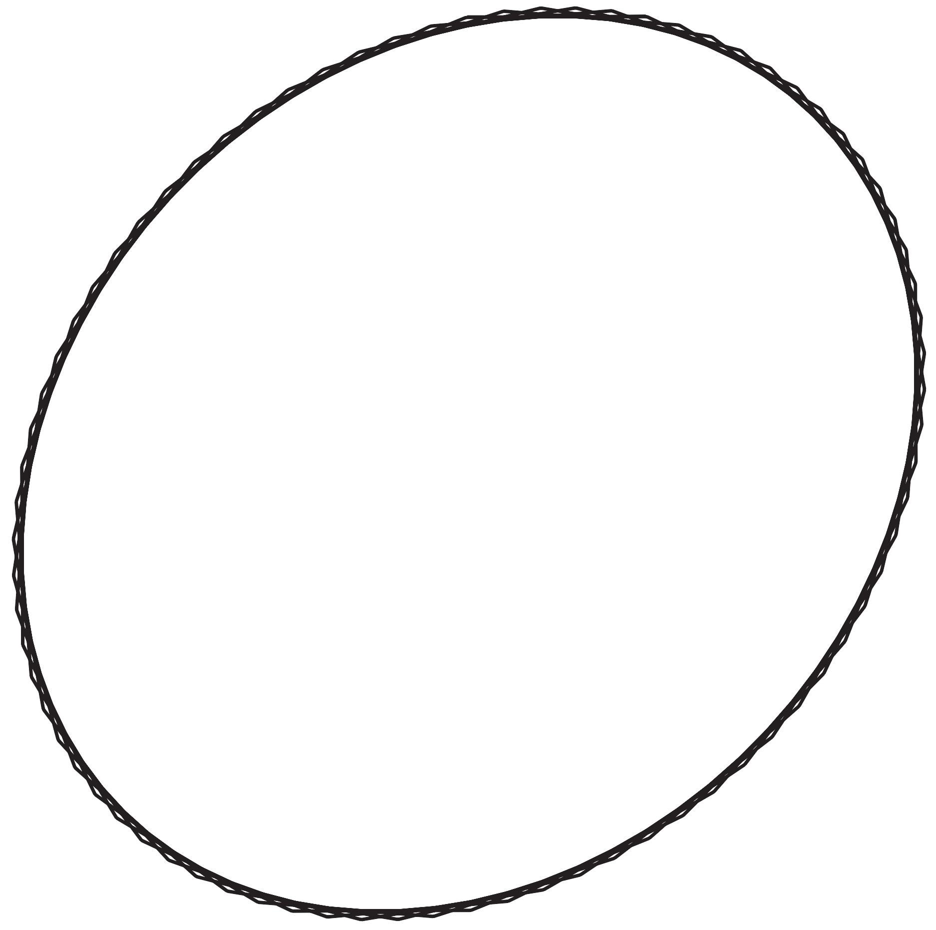
symplectic Euler



midpoint
for 10 laps



symplectic Euler



symplectic Euler
for 10 laps

Procedural noise for animation

for moving particles around we want irregular flow fields

graphics has a long tradition of defining nice looking "random" functions

- this is procedural noise

groundbreaking 1985 work of Ken Perlin established the basic approach:

- start with random values on a grid
- interpolate to get functions that are smooth locally but vary at the grid scale
- combine noise functions of different scales to get nice results

(newer methods make slightly better results)

this kind of noise can be leveraged into fake turbulence fields

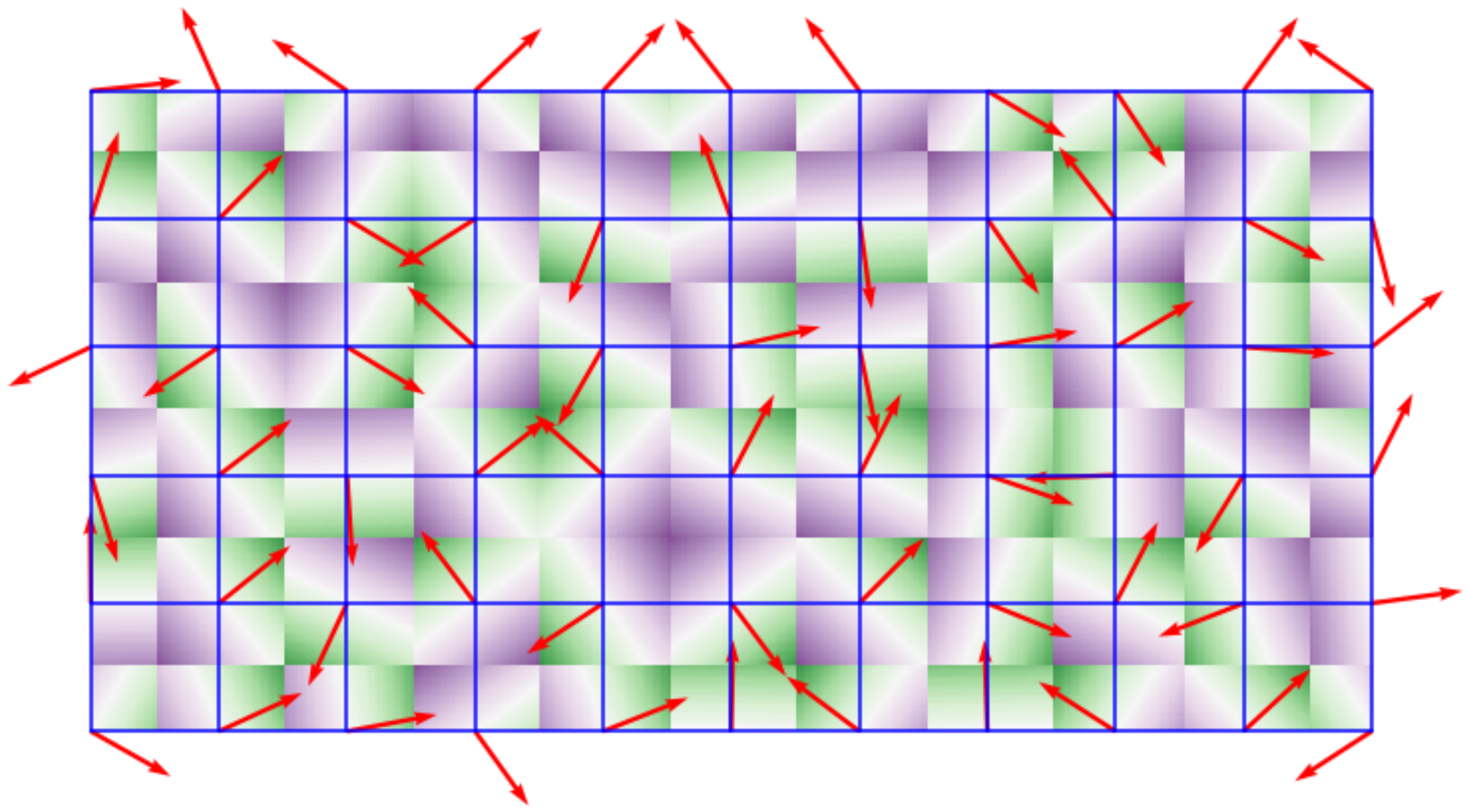
Wavelet Turbulence (Kim et al. SIGGRAPH 2008; Technical Oscar 2012)

Wavelet Turbulence (Kim et al. SIGGRAPH 2008; Technical Oscar 2012)

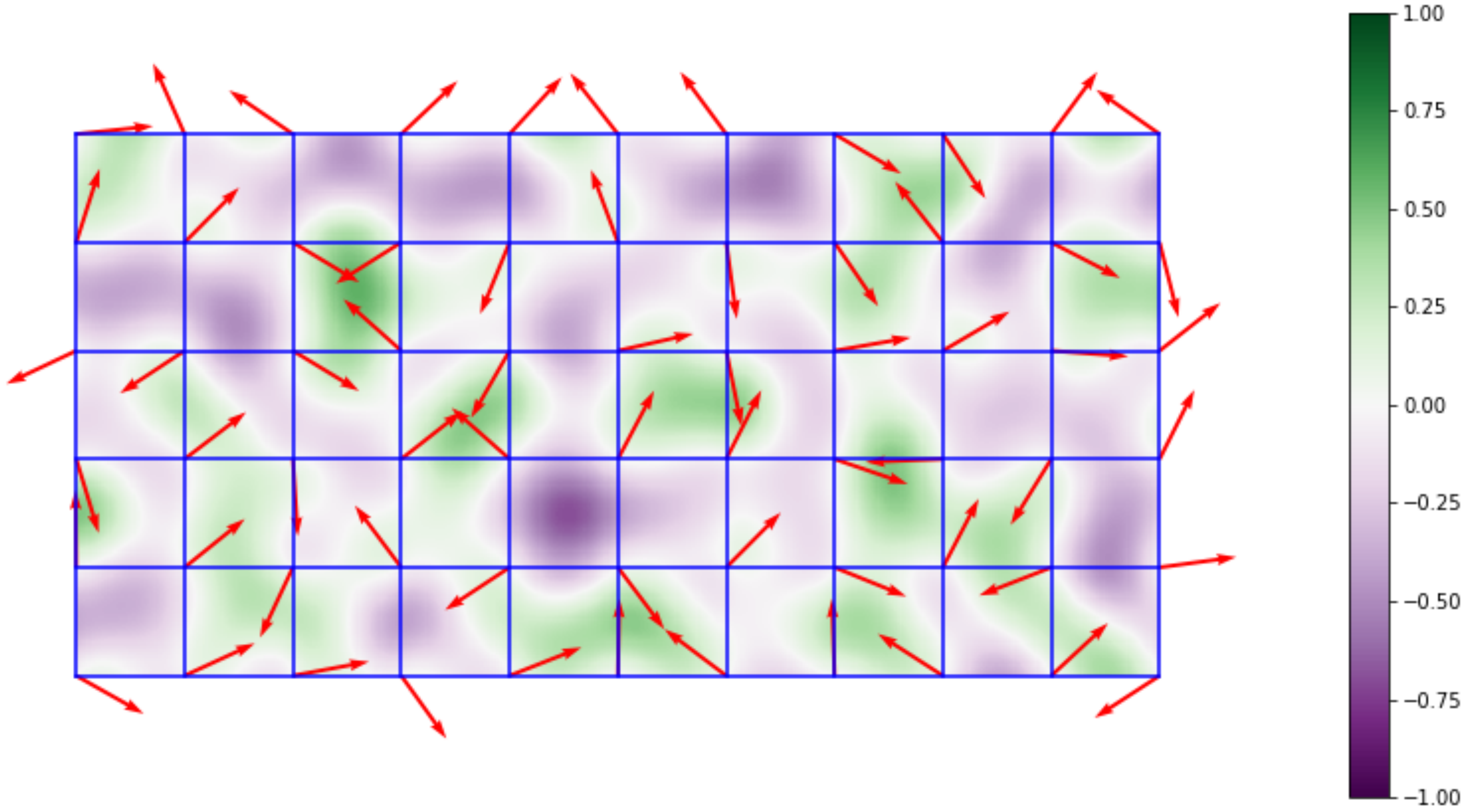
Constructing Perlin noise in 2D

1. define randomly oriented unit-slope gradients at integer points

2. interpolate between points using cubic smoothstep function $3u^2 - 2u^3$



nearest-neighbor linear gradients



interpolated with smoothstep

$\times 1$
+
$\times \frac{1}{2}$
+
$\times \frac{1}{4}$
+
$\times \frac{1}{8}$
+
$\times \frac{1}{16}$
=

Curl noise

can use Perlin or other noise to make vector fields

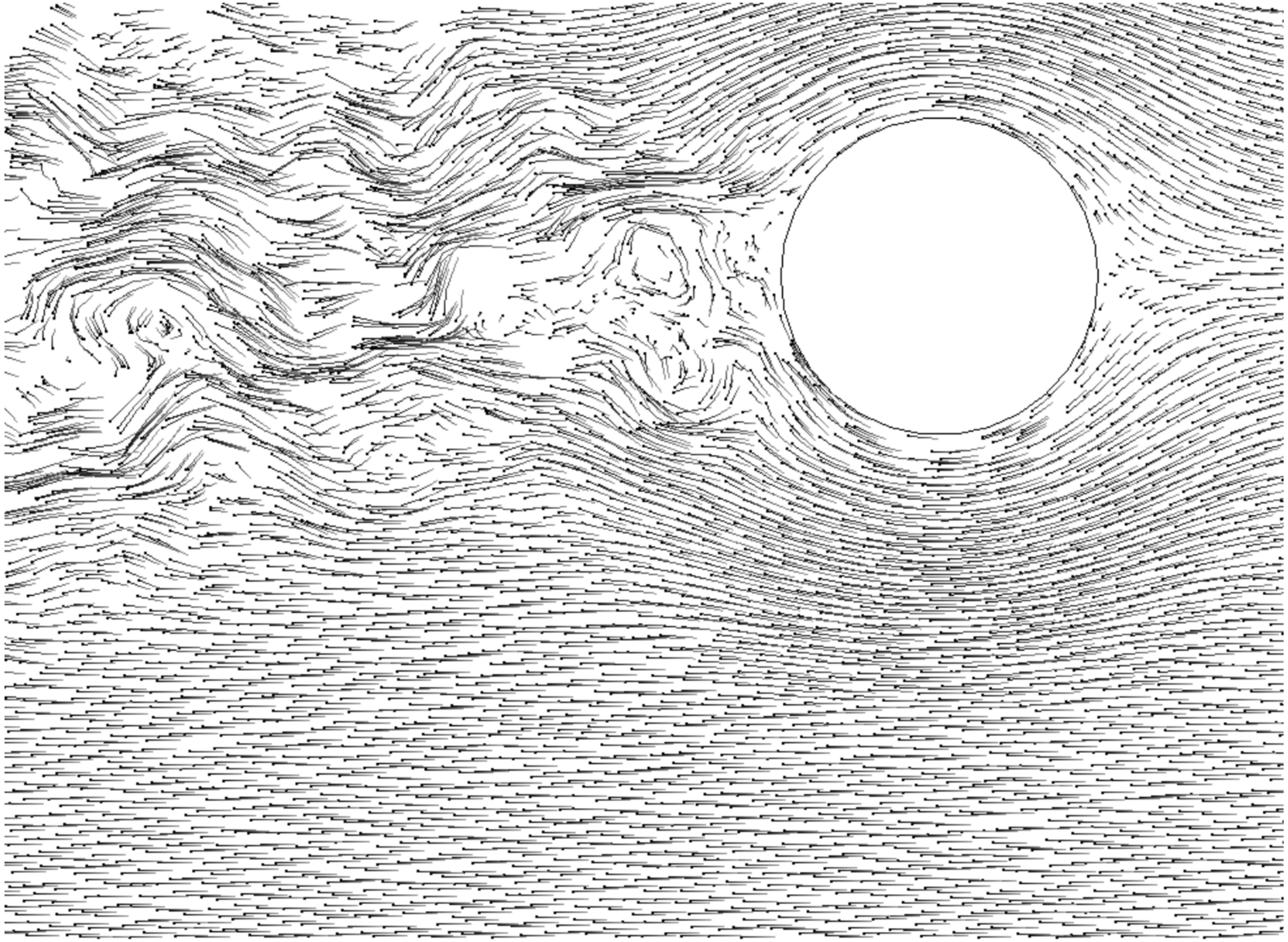
- but try advecting particles through them — doesn't work so well
- want divergence-free fields
- define them as the curl of a vector potential!

add up multiple bands just like with regular Perlin noise

- the "Kolmogorov spectrum" is a result about low-viscosity turbulent fluids: velocity at frequency f is proportional to $f^{-\frac{5}{6}}$
- close enough to f^{-1} that we might not worry about it…

additional ways to control the flow

- spatially varying weights for the different bands to make stronger turbulence in some areas
- spatial modulation of the potential to make velocities avoid obstacles



Bridson et al. 2007

Demos!

procedural noise

- Perlin noise

line integral convolution

- quick and easy way to visualize 2D vector fields

particle advection in fake turbulence fields

- first order advection
- random vector fields using curl noise