Simulation Engine for Computational Vector Calculus, Kinematic Tracing, and Error-Bounded Differentiation

Koen Hicswa
Independent Research

May 2026

Abstract

This paper covers the Vector Field Simulation with a Physics Integration framework for 2D vector field analysis. The program evaluates path independence, domain conservativeness, and localized vorticity distributions across continuous grids. Beyond visualization, the system implements a component-wise multi-solver kinematic particle tracking module to simulate continuous advection, computes discrete line integrals along parameterized or freehand contours using Riemann approximations, and performs asymptotic error profiling to optimize step-size thresholds in central difference approximations. Using an equation parser allows for more robust error handling and extended capabilities for complicated equations.

Keywords: Vector Calculus, Numerical Integration, Runge-Kutta, Central Differences, Computational Physics, Grid Discretization.

1 Introduction

The analysis of continuous, non-uniform vector-valued functions forms the mathematical foundation of many branches of math and science, namely electromagnets. However, analytic analysis of localized differential properties and path-dependent integrals over spatial domains remains tedious without dedicated numerical infrastructure. Traditional analytical methodologies rely heavily on algebraic manipulation, which fails to capture the topology of complex spaces.

To fix this deficiency, the simulation framework integrates high level analysis with rigorous numerical analysis algorithms. This program allows researchers and students to interactively explore symbolic force fields while maintaining analytical bounds on discretization error.

2 System Architecture and Implementation

The engine uses an architecture that separates parsing, numerical processing, and graphic rendering.

2.1 Software Stack

The development framework leverages three core libraries:

- **Dear ImGui:** Handles user input, and displays outputs to the user.
- **ExprTk:** A mathematical string parsing and evaluation engine that compiles user-defined component strings into bytecode at runtime.
- **SFML:** Manages the main render window and drawing calls.

2.2 Coordinate Transformation Mapping

To translate user interactions on the SFML render window back to mathematical space, the program converts pixel coordinates $(x_{\text{screen}}, y_{\text{screen}})$ on a viewport $W \times H$ into a coordinate space defined by the values G_x and G_y.

$$x_{\text{math}} = \left(\frac{x_{\text{screen}}}{W}\right) \cdot G_x - \frac{G_x}{2} \tag{1}$$

$$y_{\text{math}} = -\left(\frac{y_{\text{screen}} - H}{H}\right) \cdot G_y - \frac{G_y}{2} = \left(1 - \frac{y_{\text{screen}}}{H}\right) \cdot G_y - \frac{G_y}{2} \tag{2}$$

2.3 Tool Pipeline

The user must input $P(x, y)$ and $Q(x, y)$. The process follows:

1. **Validation Phase:** User strings are evaluated via the bytecode compiler. If structural or syntax validation fails, an alert popup notifies the user.
2. **Discretization Phase:** Upon validation, parameters are sampled across uniform coordinate intervals bounded by $[-G_x/2, G_x/2]$ and $[-G_y/2, G_y/2]$ to build a collection of `Vector2D` objects tracking localized positions, components, magnitude, and direction angles.
3. **Analytical Phase:** The verification module runs, solving local partial derivatives across the space to evaluate field conservativeness and curl vectors.

3 Mathematical Documentation

A field is formalized as a mapping over a subset of two-dimensional Euclidean space:

$$\vec{F}(x, y) = \langle P(x, y), Q(x, y) \rangle \tag{3}$$

For practical purposes, rendered vector orientations are normalized via a scaling and clamping function. The rendering magnitude L is calculated using a scaling scalar $\sigma = 12.0$ and clamped within a pixel range of $[10.0, 30.0]$:

$$L = \max\left(10.0, \min\left(30.0,\ \sigma\sqrt{P^2 + Q^2}\right)\right) \tag{4}$$

The screen rendering direction vector $\vec{d}_{\text{screen}}$ accounts for the inverted y-axis of screen space relative to standard Cartesian coordinate using the calculated angle $\theta = \text{atan2}(Q, P)$:

$$\vec{d}_{\text{screen}} = \langle \cos(\theta), -\sin(\theta) \rangle \tag{5}$$

Field vector colors are determined based on a range of magnitudes $\|\vec{F}\|$:

$$\text{Color}(\|\vec{F}\|) = \begin{cases} \text{Cyan} & \text{if } \|\vec{F}\| < 0.5 \\ \text{Green} & \text{if } 0.5 \leq \|\vec{F}\| < 1.5 \\ \text{Yellow} & \text{if } 1.5 \leq \|\vec{F}\| < 3.0 \\ \text{Red} & \text{if } \|\vec{F}\| \geq 3.0 \end{cases} \tag{6}$$

3.1 Vorticity and Curl

The curl of a two-dimensional vector field evaluates the net rotational density around a localized perpendicular axis:

$$\text{curl}(\vec{F}) = \frac{\partial Q}{\partial x} - \frac{\partial P}{\partial y} \tag{7}$$

If $|\omega_z| \leq \epsilon_{\text{threshold}}$ (where $\epsilon_{\text{threshold}} = 0.005$) across all sampled coordinate vectors within the grid domain, the engine identifies the field as conservative, guaranteeing the existence of a scalar potential field ϕ such that $\nabla\phi = \vec{F}$. Under these conditions, the work performed along any closed contour C satisfies:

$$\oint_C \vec{F} \cdot d\vec{r} = 0 \tag{8}$$

3.2 Asymptotic Derivative Error Profiling

To evaluate partial derivatives, the framework implements a central difference algorithm using an adjustable step size h. The engine isolates coordinate paths to evaluate the shifted components:

$$\frac{\partial Q}{\partial x} \approx \frac{Q(x+h, y) - Q(x-h, y)}{2h} \tag{9}$$

$$\frac{\partial P}{\partial y} \approx \frac{P(x, y+h) - P(x, y-h)}{2h} \tag{10}$$

The total computational error $E(h)$ is modeled as a combination of two numerical constraints:

1. **Truncation Error** (ϵ_{trunc}): Derived directly via the Taylor series expansions of a multi-variable function about point y:

$$P(x, y+h) = P(x, y) + h\frac{\partial P}{\partial y} + \frac{h^2}{2}\frac{\partial^2 P}{\partial y^2} + \frac{h^3}{6}\frac{\partial^3 P}{\partial y^3} + \mathcal{O}(h^4) \tag{11}$$

$$P(x, y-h) = P(x, y) - h\frac{\partial P}{\partial y} + \frac{h^2}{2}\frac{\partial^2 P}{\partial y^2} - \frac{h^3}{6}\frac{\partial^3 P}{\partial y^3} + \mathcal{O}(h^4) \tag{12}$$

   Subtracting equation (13) from (12) eliminates the even-order derivative terms:

$$P(x, y+h) - P(x, y-h) = 2h\frac{\partial P}{\partial y} + \frac{h^3}{3}\frac{\partial^3 P}{\partial y^3} + \mathcal{O}(h^5) \tag{13}$$

   Dividing by $2h$ isolates the derivative term, demonstrating that the truncation error scales with $\mathcal{O}(h^2)$:

$$\frac{P(x, y+h) - P(x, y-h)}{2h} = \frac{\partial P}{\partial y} + \frac{h^2}{6}\frac{\partial^3 P}{\partial y^3} + \mathcal{O}(h^4) \implies \epsilon_{\text{trunc}} = \frac{h^2}{6}\left|\frac{\partial^3 P}{\partial y^3}\right| \tag{14}$$

2. **Round-off Error** (ϵ_{round}): Double-precision floating-point arithmetic imposes a precision limit ($\epsilon_{\text{mach}} \approx 2.2 \times 10^{-16}$). Subtracting near-identical values results in a loss of significance scaling with $\mathcal{O}(\frac{\epsilon_{\text{mach}}}{h})$:

$$\epsilon_{\text{round}} \approx \frac{\epsilon_{\text{mach}} \cdot |P(x, y)|}{h} \tag{15}$$

The complete numerical error profile is defined as:

$$E(h) = C_1 h^2 + C_2 \frac{\epsilon_{\text{mach}}}{h} \tag{16}$$

Differentiating $E(h)$ with respect to h and setting the result to zero minimizes error:

$$\frac{dE}{dh} = 2C_1 h - C_2 \frac{\epsilon_{\text{mach}}}{h^2} = 0 \implies h_{\text{opt}} = \sqrt[3]{\frac{C_2 \cdot \epsilon_{\text{mach}}}{2C_1}} \approx 10^{-5} \tag{17}$$

The software includes a slider that allows users to observe different values of $\vec{\omega}$ if h is configured above 10^{-3} (truncation dominant) or below 10^{-12} (round-off dominant).

3.3 Component-Wise Kinematic Particle Solvers

To analyze vector spaces dynamically, a tracer module tracks state variables by treating the field as an implicit velocity map where $\frac{d\vec{r}}{dt} = \vec{F}(\vec{r}(t))$.

1. **Euler's Method (First-Order):** Maps immediate forward approximations:

$$x_{n+1} = x_n + P(x_n, y_n)\Delta t \tag{18}$$

$$y_{n+1} = y_n + Q(x_n, y_n)\Delta t \tag{19}$$

2. **Midpoint Method (Second-Order Runge-Kutta / RK2):** Evaluates a half-step projection state to eliminate first-order drift constraints:

$$x_{\text{mid}} = x_n + P(x_n, y_n)\frac{\Delta t}{2}, \quad y_{\text{mid}} = y_n + Q(x_n, y_n)\frac{\Delta t}{2} \tag{20}$$

$$x_{n+1} = x_n + P(x_{\text{mid}}, y_{\text{mid}})\Delta t \tag{21}$$

$$y_{n+1} = y_n + Q(x_{\text{mid}}, y_{\text{mid}})\Delta t \tag{22}$$

3. **Classical Runge-Kutta (RK4 - Fourth-Order):** Establishes stable orbital trajectories by combining four distinct weighted vector updates:

$$\begin{aligned}
k_{1x} &= P(x_n, y_n), & k_{1y} &= Q(x_n, y_n) \\
k_{2x} &= P\left(x_n + k_{1x}\frac{\Delta t}{2},\, y_n + k_{1y}\frac{\Delta t}{2}\right), & k_{2y} &= Q\left(x_n + k_{1x}\frac{\Delta t}{2},\, y_n + k_{1y}\frac{\Delta t}{2}\right) \\
k_{3x} &= P\left(x_n + k_{2x}\frac{\Delta t}{2},\, y_n + k_{2y}\frac{\Delta t}{2}\right), & k_{3y} &= Q\left(x_n + k_{2x}\frac{\Delta t}{2},\, y_n + k_{2y}\frac{\Delta t}{2}\right) \\
k_{4x} &= P\left(x_n + k_{3x}\Delta t,\, y_n + k_{3y}\Delta t\right), & k_{4y} &= Q\left(x_n + k_{3x}\Delta t,\, y_n + k_{3y}\Delta t\right)
\end{aligned} \tag{23}$$

The structural components are updated using a weighted average:

$$x_{n+1} = x_n + \frac{\Delta t}{6}\left(k_{1x} + 2k_{2x} + 2k_{3x} + k_{4x}\right) \tag{24}$$

$$y_{n+1} = y_n + \frac{\Delta t}{6}\left(k_{1y} + 2k_{2y} + 2k_{3y} + k_{4y}\right) \tag{25}$$

Tracer trails are handled via a container limited to size S_{trail}. Alpha fading channels are calculated dynamically across history indexes i:

$$\alpha(i) = \left\lfloor 255.0 \cdot \left(1.0 - \frac{i}{S_{\text{trail}}}\right) \right\rfloor \tag{26}$$

3.4 Real-Time Discrete Line Integration Loops

To evaluate path independence, users can choose between dynamic parametric string generation ($x(t), y(t)$) across domain boundaries $[t_{\text{start}}, t_{\text{end}}]$ with configured intervals Δt_{param}) or freehand path plotting on the grid canvas.

The integration module computes total work done by gathering sequential vertex tuples $C = \langle \vec{r}_0, \vec{r}_1, \ldots, \vec{r}_N \rangle$. It tracks explicit directional displacement variables ($\Delta x_i = x_{i+1} - x_i$, $\Delta y_i = y_{i+1} - y_i$) and maps them to a Midpoint Riemann Sum:

$$\int_C \vec{F} \cdot d\vec{r} \approx \sum_{i=0}^{N-1} \vec{F}\left(\frac{\vec{r}_i + \vec{r}_{i+1}}{2}\right) \cdot \Delta\vec{r}_i \tag{27}$$

This expression is expanded inside the computational loops:

$$W = \sum_{i=0}^{N-1} \left[P\left(\frac{x_i + x_{i+1}}{2}, \frac{y_i + y_{i+1}}{2}\right) \Delta x_i + Q\left(\frac{x_i + x_{i+1}}{2}, \frac{y_i + y_{i+1}}{2}\right) \Delta y_i \right] \tag{28}$$

4 Empirical Testing Suite & Results

The engine was validated across multiple baseline test fields with distinct topological properties. Table 1 tracks the profiles and outputs for the given vector-valued functions.

Table 1: **Empirical Test Field Configurations and Observed System Behaviors**

Field Components	**Topology Profile**	**Curl Vector ($\vec{\omega}_{\text{engine}}$)**	**Line Integral (W)**
$P = \sin(y)$, $Q = \sin(x)$	Confinement Cells	$\langle 0.0, 0.0, \cos(x) - \cos(y) \rangle$	Non-Zero
$P = x$, $Q = -y$	Hyperbolic Slopes	$\langle 0.0, 0.0, 0.0 \rangle$	0.00000
$P = \cos(x + y)$, $Q = \sin(x - y)$	Wave Patterns	$\langle 0.0, 0.0, \cos(x - y) + \sin(x + y) \rangle$	Non-Zero
$P = -y$, $Q = x$	Concentric Vortices	$\langle 0.0, 0.0, 2.0 \rangle$	Non-Zero
$P = \frac{x}{x^2+y^2}$, $Q = \frac{y}{x^2+y^2}$	Radial Point Source	$\langle 0.0, 0.0, 0.0 \rangle$	0.00000
$P = y^2$, $Q = x^2$	Parabolic Shear	$\langle 0.0, 0.0, 2x - 2y \rangle$	Non-Zero
$P = 2$, $Q = 3$	Uniform Flow	$\langle 0.0, 0.0, 0.0 \rangle$	0.00000
$P = y$, $Q = 0$	Linear Shear	$\langle 0.0, 0.0, -1.0 \rangle$	Non-Zero
$P = e^x \cos(y)$, $Q = -e^x \sin(y)$	Exponential Decay	$\langle 0.0, 0.0, 0.0 \rangle$	0.00000
$P = e^{-x} \sin(y)$, $Q = e^{-y} \cos(x)$	Damped Waves	$\langle 0.0, 0.0, -e^{-y} \sin(x) - e^{-x} \cos(y) \rangle$	Non-Zero
$P = 3x^2y$, $Q = x^3 + 2y$	Polynomial Saddle	$\langle 0.0, 0.0, 0.0 \rangle$	0.00000
$P = -y\sqrt{x^2+y^2}$, $Q = x\sqrt{x^2+y^2}$	Non-linear Vortex	$\langle 0.0, 0.0, 3\sqrt{x^2+y^2} \rangle$	Non-Zero
$P = x$, $Q = y$	Outward Expansion	$\langle 0.0, 0.0, 0.0 \rangle$	0.00000
$P = x - y$, $Q = x + y$	Expanding Spiral	$\langle 0.0, 0.0, 2.0 \rangle$	Non-Zero
$P = \cos(y)$, $Q = \cos(x)$	Orthogonal Bands	$\langle 0.0, 0.0, \sin(y) - \sin(x) \rangle$	Non-Zero
$P = \frac{x}{1+y^2}$, $Q = \frac{y}{1+x^2}$	Bounded Point Source	$\langle 0.0, 0.0, \frac{2xy}{(1+y^2)^2} - \frac{2xy}{(1+x^2)^2} \rangle$	Non-Zero
$P = -\frac{x}{(x^2+y^2)^{3/2}}$, $Q = -\frac{y}{(x^2+y^2)^{3/2}}$	Inverse-Square Sink	$\langle 0.0, 0.0, 0.0 \rangle$	0.00000
$P = -y^3$, $Q = x^3$	Cubic Rotator	$\langle 0.0, 0.0, 3(x^2 + y^2) \rangle$	Non-Zero
$P = -2x$, $Q = -2y$	Harmonic Attractor	$\langle 0.0, 0.0, 0.0 \rangle$	0.00000
$P = \sin(y)$, $Q = 0$	Sinusoidal Shear	$\langle 0.0, 0.0, -\cos(y) \rangle$	Non-Zero
$P = \ln(x^2 + 1)$, $Q = \ln(y^2 + 1)$	Logarithmic Drift	$\langle 0.0, 0.0, 0.0 \rangle$	0.00000
$P = e^y$, $Q = x$	Exponential Shear	$\langle 0.0, 0.0, 1 - e^y \rangle$	Non-Zero
$P = \frac{y^2-x^2}{(x^2+y^2)^2}$, $Q = \frac{-2xy}{(x^2+y^2)^2}$	Dipole Configuration	$\langle 0.0, 0.0, 0.0 \rangle$	0.00000
$P = \frac{-y}{1+x^2+y^2}$, $Q = \frac{x}{1+x^2+y^2}$	Shielded Vortex	$\langle 0.0, 0.0, \frac{2}{(1+x^2+y^2)^2} \rangle$	Non-Zero
$P = \sin(x)\cos(y)$, $Q = \cos(x)\sin(y)$	Standing Wave Potential	$\langle 0.0, 0.0, 0.0 \rangle$	0.00000
$P = x^2y$, $Q = xy^2$	Anisotropic Shear	$\langle 0.0, 0.0, y^2 - x^2 \rangle$	Non-Zero

When testing the conservative properties of Fields 2 and 5 along arbitrary user-defined closed paths, the computed line integral returned a value of 0.00000 (bounded within a numerical tolerance

constraint matching ϵ_{mach}), confirming path independence and the existence of a scalar potential. Testing rotation-dominant fields (e.g., Fields 1 and 4) over closed loops strictly yielded integrated work values scaling proportionally with the enclosed surface vorticity.

5 Conclusion

The Simulation Engine establishes a mathematically rigorous environment for computational vector calculus and dynamic field analysis. By synthesizing variable-order kinematic Runge-Kutta solvers, discrete line integration topologies, and dynamic step-size optimization for derivative error minimization. It functions as a computational tool capable of demonstrating asymptotic numerical limits, and verifying topological properties such as path independence across complex boundaries.

Future iterations of this architecture will expand computational capabilities to encompass three-dimensional spatial tensor fields, implement adaptive time-stepping algorithms (e.g., Runge-Kutta-Fehlberg) for managing differential stiffness, and introduce hardware-accelerated parallel computing to widen the scope of the computational tool.

References

[1] A. Partow, "The C++ Mathematical Expression Toolkit Library (ExprTk)," *Partow Technologies*, 2020.

[2] L. N. Trefethen and D. Bau III, *Numerical Linear Algebra*, SIAM, 1997.

[3] E. Hairer, S. P. Nørsett, and G. Wanner, *Solving Ordinary Differential Equations I: Nonstiff Problems*, Springer, 1993.

[4] R. L. Burden and J. D. Faires, *Numerical Analysis*, Cengage Learning, 2011.