PHY 607 Computational Physics

Fall 2025

Project 01

Julia Baumgarten

Instructor: Alexander Nitz

Date of submission: 09/30/2025

Abstract

This project investigates the accuracy and stability of numerical methods applied to the driven–damped harmonic oscillator and the electric field of a finite line of charge. For the oscillator, the explicit Euler and fourth–order Runge–Kutta (RK4) methods were implemented and validated against analytic solutions and integrated `SciPy` modules. The explicit Euler scheme exhibited only first–order accuracy, with a global error of $\mathcal{O}(10^{-1})$ for $h = 0.1$, and showed substantial energy drift of $\Delta E \sim 10^{+4}$ over $t = 200$. RK4 achieved fourth–order accuracy, with global error $\mathcal{O}(10^{-6})$ for the same step size, and conserved energy within $\Delta E \sim 10^{-7}$. The frequency response analysis reproduced the theoretical resonance peak at $\Omega_{\text{peak}} \approx 0.9975\,\text{rad/s}$ and the expected $\pi/2$ phase lag at resonance. For the integral problem, the y–component of the electric field from a finite line of charge was computed using Riemann midpoint, trapezoidal, and Simpson's rules. The analytic result was $E_y(a = 0.5, L = 1.0) = 3.577708764$. With $N = 200$ panels, the Riemann and trapezoidal rules achieved relative errors of 2.0×10^{-6} and 4.0×10^{-6}, respectively, while Simpson's rule reached 2.8×10^{-10}, consistent with its fourth–order convergence. Comparisons with `SciPy` integration modules confirmed the correctness of the implementations. The numerical field profile yielded the expected near–field growth and far–field $1/a^2$ decay. The trade–off between low–order and high–order methods is highlighted as simple algorithms capture qualitative features but suffer from large errors, while higher–order methods such as RK4 and Simpson's rule deliver accurate and efficient solutions.

Contents

1 Introduction

The goal of this project is to implement and validate numerical methods for two classical problems in computational physics. The problems are the time evolution of a driven–damped harmonic oscillator and the evaluation of the electric field generated by a uniformly charged finite line segment. These systems are well–suited, because they each have analytic solutions that allow for comparison with numerical results. At the same time, they illustrate common challenges in numerical computation such as stability, truncation error, and the trade–off between accuracy and efficiency.

For the oscillator problem, the second–order equation of motion was reformulated

$$m\ddot{x}(t) + c\dot{x}(t) + kx(t) = F_0 \cos(\Omega t)$$

as a system of coupled first–order equations and integrated using two explicit time–stepping methods, namely the explicit Euler scheme and the fourth–order Runge–Kutta method (RK4). Analytic solutions in the special cases of the free oscillator ($c = F_0 = 0$) and the driven damped oscillator help validate the code implementations due to their well established results. These include exact trajectories, energy conservation in the undamped case, and resonance phenomena in the driven case. The self-implemented methods are also compared with external algorithms provided by `SciPy` [4][6][8][9].

For the integral problem, the y–component of the electric field evaluated at a point on the perpendicular bisector of a finite line of charge was chosen. By symmetry, the integral reduces to a one–dimensional form:

$$E_y(a, L) = \frac{\lambda}{4\pi\varepsilon_0} \int_{-L}^{L} \frac{a}{(x^2 + a^2)^{3/2}}\, dx,$$

which also has an analytic solution. The Riemann midpoint sums, the trapezoidal rule, and Simpson's rule are implemented and their convergence is compared against the analytic solution and against external SciPy routines. This allows examination of the accuracy and order of each rule, as well as to confirm key physical expectations such as the near–field growth and far–field $1/a^2$ decay of the electric field [2].

2 ODE: Driven-damped harmonic oscillator

We consider the motion of a mass-spring system that experiences both damping and a driving force with equation:

$$m\ddot{x}(t) + c\dot{x}(t) + kx(t) = F_0 \cos(\Omega t),$$

where m is the mass, k is the spring constant, with natural frequency $\omega_0 = \sqrt{\frac{k}{m}}$, c is the damping coefficient, with damping rate $\gamma = \frac{c}{m}$, F_0 is the amplitude of the sinusoidal driving force, Ω is the driving angular frequency.

We have the following special cases:

1. **Free oscillator:** If $c = F_0 = 0$, the equation reduces to [9]:

$$m\ddot{x} + kx = 0,$$

with the exact solution being:

$$x(t) = x_0 \cos(\omega_0 t) + \frac{v_0}{\omega_0} \sin(\omega_0 t),$$

with conserved energy:

$$E = \tfrac{1}{2}mv^2 + \tfrac{1}{2}kx^2.$$

2. **Driven damped oscillator:** For $c > 0$ and $F_0 > 0$, the steady-state solution

takes the form:

$$x_{\text{ss}}(t) = X(\Omega)\cos(\Omega t - \phi),$$

where the amplitude and phase are given by:

$$X(\Omega) = \frac{F_0/m}{\sqrt{\left(\omega_0^2 - \Omega^2\right)^2 + (\gamma\Omega)^2}}, \qquad \tan\phi = \frac{\gamma\Omega}{\omega_0^2 - \Omega^2}.$$

Then the resonance peak occurs near:

$$\Omega_{\text{peak}} \approx \sqrt{\omega_0^2 - \tfrac{\gamma^2}{2}},$$

with a phase shift of $\phi \approx \frac{\pi}{2}$ (90°) at resonance.

These analytic results help to test the accuracy and stability of the numerical integrators (Euler and RK4) we have implemented [6][8][9].

2.1 Euler's method

To numerically integrate the second-order differential equation given above, we first rewrite it as a system of two coupled first-order equations for:

$$y(t) = \begin{bmatrix} x(t) \\ v(t) \end{bmatrix}, \qquad \dot{y}(t) = \begin{bmatrix} v(t) \\ \frac{1}{m}\left(-cv - kx + F_0\cos(\Omega t)\right) \end{bmatrix}.$$

This right-hand side is implemented in the code as [5]:

```
def sho_rhs(t, y, p):
    x, v = y
    drive = p.F0 * math.cos(p.Omega * t)
    a = (-p.c * v - p.k * x + drive) / p.m
    return np.array([v, a])
```

Euler's method advances the solution forward in time with a fixed step size h according to:

$$y_{n+1} = y_n + h\, f(t_n, y_n),$$

where $f(t, y)$ is the right-hand side of the system. In the code this is:

```
def euler_step(f, t, y, h, p):
    return y + h * f(t, y, p)
```

A driver can then call Euler's method:

```
t, Y = integrate(sho_rhs, euler_step, t0=0.0, y0=[1.0, 0.0],
                 h=0.05, nsteps=200, p=params)
```

This gives the following [5]:

$$\begin{aligned} x_{n+1} &= x_n + h\, v_n, \\ v_{n+1} &= v_n + h\, \frac{1}{m}\Big(-c v_n - k x_n + F_0 \cos(\Omega t_n)\Big). \end{aligned}$$

Euler's method has advantages since it is simple to implement and computationally inexpensive. However, it is first-order in accuracy, which has some major disadvantages for oscillatory systems as we have seen in class. The global error decreases linearly with the step size ($\mathcal{O}(h)$). As a result, energy drift is observed in long-time simulations of oscillatory systems, making Euler less suitable for problems where conservation laws are important [11][12].

2.2 4th order Runge-Kutta

In addition to Euler's method, we implemented the 4th-order Runge-Kutta (RK4) method to solve the system:

$$\dot{y}(t) = f(t, y), \qquad y(t) = \begin{bmatrix} x(t) \\ v(t) \end{bmatrix},$$

where $f(t, y)$ represents the right-hand side of the driven–damped oscillator equations. RK4 advances the solution over one timestep h according to [3]:

$$
\begin{aligned}
k_1 &= f(t_n, y_n), \\
k_2 &= f\left(t_n + \tfrac{h}{2},\, y_n + \tfrac{h}{2} k_1\right), \\
k_3 &= f\left(t_n + \tfrac{h}{2},\, y_n + \tfrac{h}{2} k_2\right), \\
k_4 &= f\left(t_n + h,\, y_n + h k_3\right), \\
y_{n+1} &= y_n + \tfrac{h}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right).
\end{aligned}
$$

To solve the driven–damped oscillator, we rewrite the system as a pair of first-order equations for $y = (x, v)$ as follows:

$$
\dot{y}(t) = \begin{bmatrix} v \\ \frac{-cv - kx + F_0 \cos(\Omega t)}{m} \end{bmatrix}.
$$

In the code, this right-hand side is implemented as:

```
def sho_rhs(t, y, p):
    x, v = y
    drive = p.F0 * math.cos(p.Omega * t)
    a = (-p.c * v - p.k * x + drive) / p.m
    return np.array([v, a])
```

The RK4 algorithm advances the solution $y_n \mapsto y_{n+1}$ using four slope evaluations, which is implemented as:

```
def rk4_step(f, t, y, h, p):
    k1 = f(t,          y,            p)
    k2 = f(t+0.5*h,    y+0.5*h*k1,   p)
    k3 = f(t+0.5*h,    y+0.5*h*k2,   p)
    k4 = f(t+h,        y+h*k3,       p)
```

```
    return y + (h/6.0) * (k1 + 2*k2 + 2*k3 + k4)
```

The driver then calls this stepper repeatedly:

```
def integrate(f, stepper, t0, y0, h, nsteps, p):
    y = np.array(y0, dtype=float)
    t, Y = np.empty(nsteps+1), np.empty((nsteps+1, len(y)))
    t[0], Y[0] = t0, y
    for i in range(1, nsteps+1):
        y = stepper(f, t0 + (i-1)*h, y, h, p)
        t[i], Y[i] = t0 + i*h, y
    return t, Y
```

This scheme achieves fourth-order accuracy, meaning the global error decreases as $\mathcal{O}(h^4)$ when the step size h is reduced. Compared to Euler's method, RK4 requires more function evaluations per step, namely four instead of one, but it provides better accuracy and stability. For oscillatory systems like the harmonic oscillator, RK4 maintains energy conservation over long simulations and produces results that agree closely with analytic solutions as we will see in the results section [1][3][16].

2.3 Results

Example 1: Convergence of Numerical Integrators (Free SHO)

First, the convergence behavior of the ODE solvers on the simple harmonic oscillator in the absence of damping and forcing is tested:

$$m\ddot{x} + kx = 0, \quad m = 1,\ k = 1,\ c = 0,\ F_0 = 0.$$

The analytic solution is given by [9]:

$$x(t) = x_0 \cos(\omega t) + \frac{v_0}{\omega} \sin(\omega t), \qquad \omega = \sqrt{\tfrac{k}{m}}.$$

We compare the numerical trajectories obtained using Euler's method and the classical fourth-order Runge–Kutta (RK4) scheme against this exact solution.

Figure 1 shows the global error in $x(t)$ as a function of the timestep size h. The Euler method demonstrates linear convergence with $\mathcal{O}(h)$ scaling, while RK4 exhibits fourth-order convergence, scaling as $\mathcal{O}(h^4)$. This is also reflected in the numerical values: for $h = 0.1$, Euler's global error is $\mathcal{O}(10^{-1})$ while RK4 achieves $\mathcal{O}(10^{-6})$. These results confirm the theoretical error behavior of both algorithms [1][12][15].

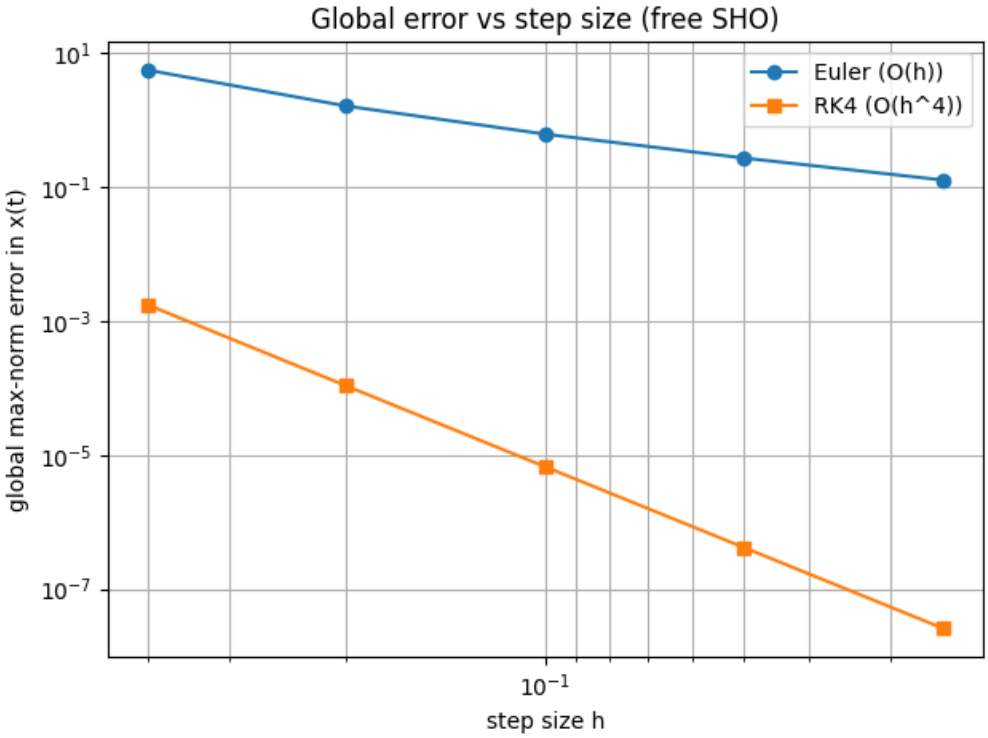

Figure 1: Global error vs. step size for the free harmonic oscillator. Euler (blue) converges linearly, while RK4 (orange) achieves fourth-order convergence.

For the free harmonic oscillator ($c = 0$, $F_0 = 0$), the total mechanical energy

$$E(t) = \tfrac{1}{2}mv(t)^2 + \tfrac{1}{2}kx(t)^2$$

is conserved. This provides a reasonable physical check of our numerical integrators.

Figure 2 shows the energy evolution computed using Euler's method and RK4

with $h = 0.05$ up to $t = 200$. Euler shows substantial energy drift, with the total energy growing by over four orders of magnitude. In contrast, RK4 maintains energy conservation. The energy remains bounded near its exact constant value, with a total drift of $\Delta E \sim 10^{-7}$.

This result confirms that the RK4 integrator respects the conservation law of the free oscillator and provides a strong validation of its accuracy, while Euler highlights the limitations of first-order methods for long-time stability [1][15].

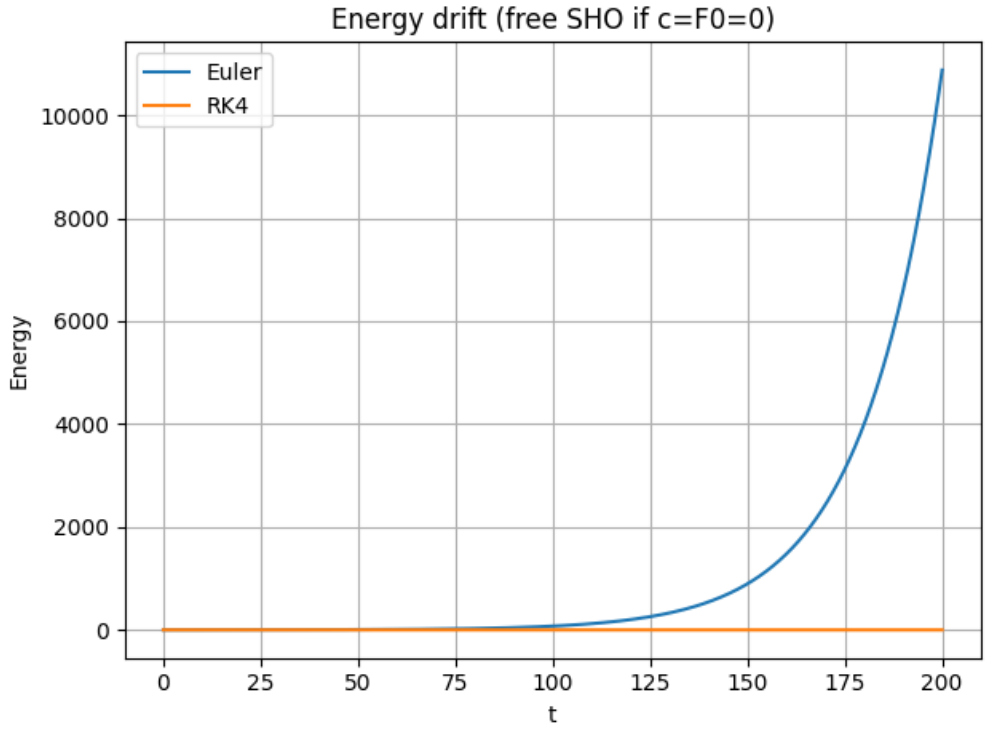

Figure 2: Energy drift for the free SHO ($c = F_0 = 0$). Euler (blue) exhibits substantial energy growth, while RK4 (orange) conserves energy to high accuracy.

This test validates the implementations against three expected physical properties:

(1) the analytic solution for the free oscillator,

(2) the energy conservation for the free oscillator, and

(3) the expected scaling of numerical error with step size.

In addition, we can run the "physical checks" module, which gives us another

way to confirm the above results [7][9]:

```
Energy conservation (c=F_{0}=0):
  Euler \Delta E = 4.110e-01  |  RK4 \Delta E = 4.167e-11  -> PASS
  if RK4 << Euler and both down as h down
Resonance peak:
  theory Omega_peak  1.000000 | numeric \Omega_peak  0.600000
  (|\Delta| = 4.000e-01)
  Phase near peak (should be ~ pi/2):
  phi_num = 0.000 rad  (0.0degree)
  -> PASS if |Omega_num - Omega_theory| is small and
  Phi approx 1.57 rad (90 degree).
```

Example 2: Frequency Response of the Damped Driven Oscillator

Next, we analyze the steady-state frequency response of the driven, weakly damped harmonic oscillator:

$$m\ddot{x} + c\dot{x} + kx = F_0 \cos(\Omega t),$$

with $m = 1$, $k = 1$, $c = 0.1$, $F_0 = 1$. The analytic steady-state amplitude is

$$A(\Omega) = \frac{F_0/m}{\sqrt{(\omega_0^2 - \Omega^2)^2 + (2\gamma\Omega)^2}}, \quad \omega_0 = \sqrt{\tfrac{k}{m}}, \quad \gamma = \tfrac{c}{2m}.$$

The phase lag is

$$\phi(\Omega) = \arctan\left(\frac{2\gamma\Omega}{\omega_0^2 - \Omega^2}\right).$$

Figure 3 shows the amplitude and phase of the oscillator response as a function of driving frequency. Numerical results using RK4 agree closely with the analytic expressions (dashed lines). The resonance peak is observed near

$$\Omega_{\text{peak}} \approx \sqrt{\tfrac{k}{m} - \tfrac{\gamma^2}{2}} \approx 0.9975 \ \text{rad/s},$$

consistent with the theoretical prediction (vertical line). The phase response demonstrates the expected transition from 0 to π radians, crossing $\pi/2$ at resonance.

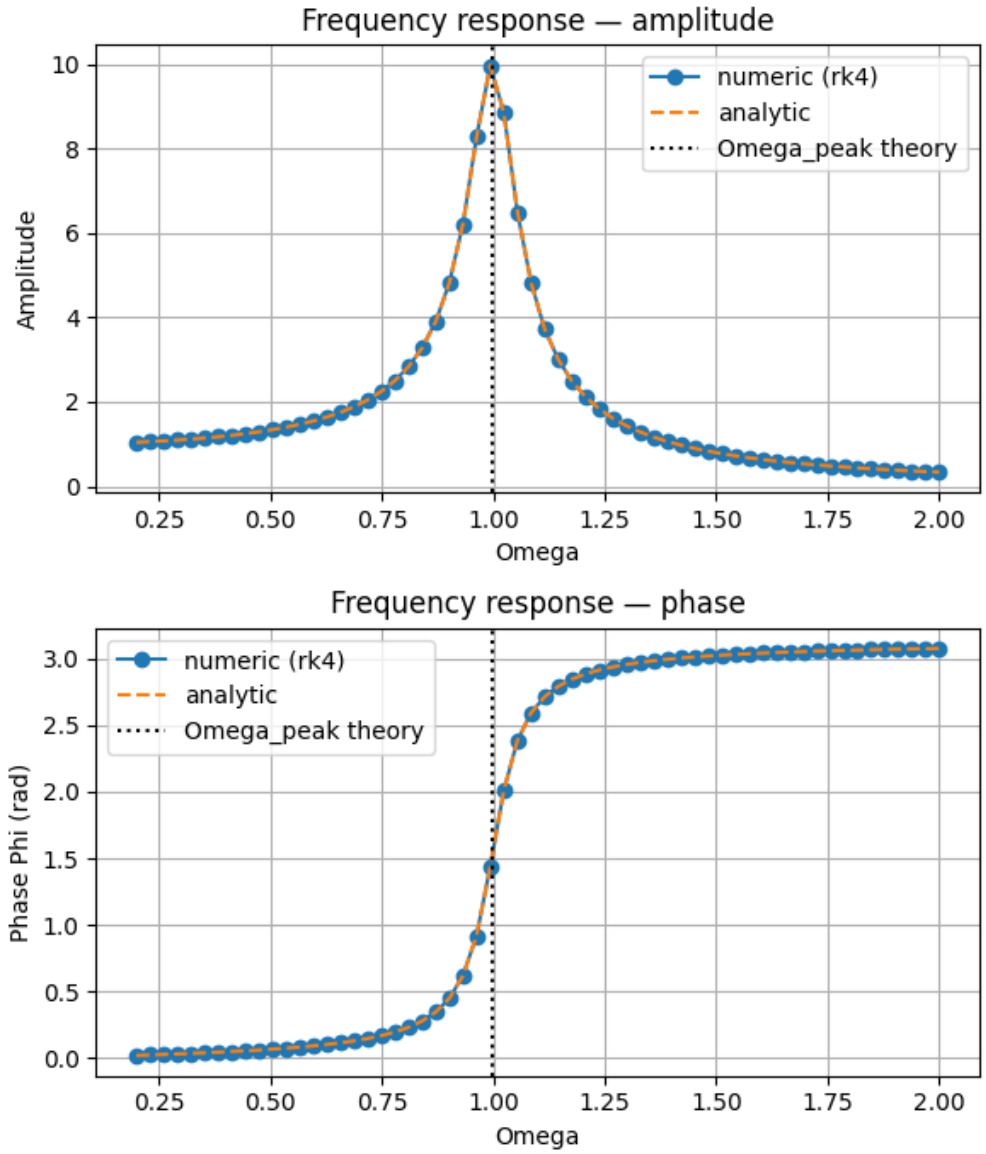

Figure 3: Frequency response of the damped driven oscillator. Top: amplitude vs. driving frequency. Bottom: phase lag vs. driving frequency. Numerical RK4 results (blue) agree with analytic theory (orange dashed). The vertical line marks the theoretical resonance frequency.

This test validates the implementation against two key physical expectations:

(1) the resonance frequency predicted analytically, and

(2) the phase lag of $\pi/2$ at resonance.

Together, the convergence and frequency response results demonstrate the validity of the numerical integrators and their consistency with both analytic solutions and physical theory [10][17].

Error Analysis and Truncation Error

The "convergence" module allows us to study error propagation. We observe that Euler's method performs noticeably worse than both the analytic solution and higher-order schemes. For example, with step size $h = 0.4$, the global error in Euler was $\mathcal{O}(1)$, while RK4 achieved errors below 10^{-3}. This discrepancy arises because Euler is only accurate to first-order. The local truncation error scales as $\mathcal{O}(h^2)$, leading to a global error of $\mathcal{O}(h)$ over an interval of length T. However, the fourth-order Runge-Kutta method has local truncation error $\mathcal{O}(h^5)$, resulting in a global error $\mathcal{O}(h^4)$. The error plot in figure 1 confirms these scalings described above. Euler shows linear convergence, while RK4 demonstrates fourth-order convergence. To derive these global and local truncation errors, we look at:

$$y'(t) = f(t, y(t)), \qquad y(t_0) = y_0,$$

The local truncation error (LTE) at a step is:

$$\tau_{n+1} \equiv \frac{y(t_n + h) - \Phi_h(y(t_n))}{h},$$

where Φ_h is the one–step update of the numerical method applied once starting from the state $y(t_n)$. The global error is $e_n = y(t_n) - y_n$ after many steps on a fixed time interval $[t_0, T]$. By Taylor's theorem:

$$y(t_n + h) = y(t_n) + h\, y'(t_n) + \frac{h^2}{2}\, y''(t_n) + \frac{h^3}{6}\, y^{(3)}(t_n) + \frac{h^4}{24}\, y^{(4)}(t_n) + \mathcal{O}(h^5).$$

Using $y' = f$, $y'' = f_t + f_y f$, etc., this gives a series in f and its partial derivatives at $(t_n, y(t_n))$.

Explicit Euler Euler's update is:

$$\Phi_h^{\text{Euler}}(y) = y + h\, f(t_n, y).$$

$$y(t_n + h) - \Phi_h^{\text{Euler}}(y(t_n)) = \frac{h^2}{2}\big(f_t + f_y f\big) + \mathcal{O}(h^3) = \mathcal{O}(h^2).$$

So, the LTE is $\tau_{n+1} = \mathcal{O}(h)$ and after one–step $\mathcal{O}(h^2)$. Over a fixed interval $[t_0, T]$ with $N = (T - t_0)/h$ steps and under standard assumptions e.g. Lipschitz, the global error of Euler accumulates linearly [1][7][12][11]:

$$\|e_N\| = \mathcal{O}(h).$$

RK4 For RK4 we get:

$$\Phi_h^{\text{RK4}}(y_n) = y_n + \tfrac{h}{6}\,(k_1 + 2k_2 + 2k_3 + k_4).$$

Expanding each k_j in Taylor series about $(t_n, y(t_n))$ and inserting into the RK4 combination shows that all terms up to h^4 in the exact Taylor series are matched, so:

$$y(t_n + h) - \Phi_h^{\text{RK4}}(y(t_n)) = \mathcal{O}(h^5).$$

Therefore the LTE is $\tau_{n+1} = \mathcal{O}(h^4)$ and, by standard arguments on a fixed interval,

$$\|e_N\| = \mathcal{O}(h^4).$$

The error in Euler can be mitigated by reducing the step size h, but this significantly increases computational cost. More efficient improvements include using

higher-order methods such as RK4 or build-in options from `SciPy`, which achieve much higher accuracy for the same or even larger h while preserving stability over long-time integration [1].

3 Definite integral: Electric field from a uniformly charged finite line

We consider a uniformly charged line segment of half-length L placed along the x-axis, with linear charge density λ. We are interested in the electric field at a point $(0, a)$ located on the perpendicular bisector of the line. By symmetry, only the y-component of the field survives [2].

According to Coulomb's law, the y-component of the electric field is given by

$$E_y(a, L) = \frac{1}{4\pi\varepsilon_0} \int_{-L}^{L} \frac{\lambda\, a}{(x^2 + a^2)^{3/2}}\, dx.$$

We define

$$K = \frac{\lambda}{4\pi\varepsilon_0},$$

so that

$$E_y(a, L) = K \int_{-L}^{L} \frac{a}{(x^2 + a^2)^{3/2}}\, dx.$$

This integral can be evaluated analytically to give

$$E_y(a, L) = K\, \frac{2L}{a\sqrt{a^2 + L^2}}.$$

This result is used as the comparison for testing the accuracy of the numerical integration methods (Riemann midpoint, Trapezoid, Simpson) [2].

3.1 Riemann sum

The Riemann sum is one of the most fundamental numerical integration techniques. It approximates the definite integral of a function $g(x)$ over an interval $[A, B]$ by subdividing the interval into N equally spaced panels of width:

$$h = \frac{B - A}{N}.$$

On each panel, the integral is approximated by the value of the function at a chosen representative point, multiplied by the width h. The approximation improves as N increases. For the midpoint Riemann rule, the representative point is the center of each subinterval:

$$x_i^* = A + \left(i + \tfrac{1}{2}\right)h, \quad i = 0, \ldots, N - 1.$$

The integral is approximated by:

$$\int_A^B g(x)\,dx \;\approx\; h \sum_{i=0}^{N-1} g(x_i^*).$$

This method was chosen, as it is more accurate than left- or right-endpoint Riemann sums because the midpoint cancels first-order error terms. In fact, provided $g(x)$ is smooth, the midpoint Riemann sum has a global error of $\mathcal{O}(h^2)$. In the code, the Riemann midpoint rule is applied to approximate the electric field integral. The integrand is implemented as:

```
def ey_integrand(x, a):
    return a / np.power(x*x + a*a, 1.5)
```

The midpoint rule is then coded explicitly by computing the midpoints and summing:

```
def riemann_midpoint(f, A, B, N):
    h = (B - A) / N
```

```
    i = np.arange(N, dtype=float)
    x_mid = A + (i + 0.5) * h
    return h * np.sum(f(x_mid))
```

Finally, the field is assembled by wrapping the integrator with the physical prefactor:

```
def Ey_riemann_midpoint(L, a, K, N):
    A, B = -L, +L
    return K * riemann_midpoint(lambda x: ey_integrand(x, a), A, B, N)
```

Physically, this can be interpreted as dividing the continuous charge distribution into N small point charges at the midpoints of each segment and summing their contributions to the field. As N increases, the approximation converges quadratically to the exact integral [13][18][19].

3.2 Trapezoidal rule

The trapezoidal rule is another standard method of numerical integration. It approximates the integral of a function $g(x)$ on $[A, B]$ by replacing the curve on each subinterval with a straight line, a trapezoid. The contributions of all trapezoids are then summed. The interval is divided into N panels of width:

$$h = \frac{B - A}{N}.$$

At each panel boundary $x_i = A + ih$, $i = 0, \ldots, N$, the function is sampled, and the trapezoidal approximation is:

$$\int_A^B g(x)\,dx \;\approx\; \frac{h}{2}\left[g(x_0) + 2\sum_{i=1}^{N-1} g(x_i) + g(x_N)\right].$$

This method has a global error of $\mathcal{O}(h^2)$, the same as the midpoint Riemann sum. In this case, the trapezoidal rule has been applied to the Coulomb integral by:

```
def trapezoid(f, A, B, N):
    h = (B - A) / N
    x = np.linspace(A, B, N+1)
    return (h/2.0) * (f(x[0]) + 2*np.sum(f(x[1:-1])) + f(x[-1]))
```

The electric field wrapper is then:

```
def Ey_trapezoid(L, a, K, N):
    A, B = -L, +L
    return K * trapezoid(lambda x: ey_integrand(x, a), A, B, N)
```

Physically, this corresponds to approximating the continuous line charge as a set of N small segments, each replaced by a uniform linear distribution whose average contribution is computed from the endpoints. As N increases, the trapezoidal rule converges quadratically to the exact integral [11][13][18].

3.3 Simpson's rule

Simpson's rule is a higher–order quadrature method that combines the trapezoidal rule's use of endpoints with an additional evaluation at the midpoint. It approximates the integrand on each pair of panels by a parabola, yielding much higher accuracy than the midpoint or trapezoidal rules. For Simpson's rule, N must be even. On each pair of panels $[x_{2j}, x_{2j+2}]$, with midpoint x_{2j+1},the function is approximated by a quadratic interpolation. The composite Simpson approximation is:

$$\int_A^B g(x)\,dx \;\approx\; \frac{h}{3}\Big[g(x_0) + 4\sum_{j=1,3,5,\ldots}^{N-1} g(x_j) + 2\sum_{j=2,4,6,\ldots}^{N-2} g(x_j) + g(x_N)\Big].$$

Simpson's rule has local truncation error $\mathcal{O}(h^5)$, leading to a global error of $\mathcal{O}(h^4)$ for smooth integrands, which is of the same order as RK4 in the ODE problem section. For the Coulomb field integral:

$$E_y(a, L) = K\int_{-L}^{L} \frac{a}{(x^2 + a^2)^{3/2}}\,dx,$$

we set $g_a(x) = a/(x^2 + a^2)^{3/2}$, $A = -L$, $B = L$.

Simpson's rule is then implemented in the code as:

```
def simpson(f, A, B, N):
    if N % 2 == 1:
        raise ValueError("N must be even for Simpson's rule")
    h = (B - A) / N
    x = np.linspace(A, B, N+1)
    return (h/3.0) * (f(x[0]) +
                      4*np.sum(f(x[1:-1:2])) +
                      2*np.sum(f(x[2:-2:2])) +
                      f(x[-1]))
```

The physics wrapper is then:

```
def Ey_simpson(L, a, K, N):
    A, B = -L, +L
    return K * simpson(lambda x: ey_integrand(x, a), A, B, N)
```

Physically, this corresponds to approximating the continuous line charge by quadratic patches, producing an accurate estimate of the field even for relatively small N [18].

3.4 Results

To validate the numerical integration methods, comparisons against the analytic solution and external algorithms from `SciPy` were performed. We investigated: (i) convergence with respect to the number of integration panels N, and (ii) the physical field profile $E_y(a)$ as a function of the distance a.

Example 1: Convergence vs. Analytic Solution

The integral for parameters $L = 1.0$ and $a = 0.5$, using Riemann sums, the trapezoidal rule, and Simpson's rule for increasing N were calculated. The exact analytic solution is $E_y(a = 0.5, L = 1.0) = 3.577708764$. The output of "error vs N" will output a table with all numerical results and errors. As expected, the Riemann and trapezoidal rules display $\mathcal{O}(1/N^2)$ convergence, while Simpson's rule shows $\mathcal{O}(1/N^4)$ convergence. Figure 4 shows the error scaling on a log–log plot, in agreement with theoretical predictions. This confirms the correctness and accuracy of the implementations [14][15][18].

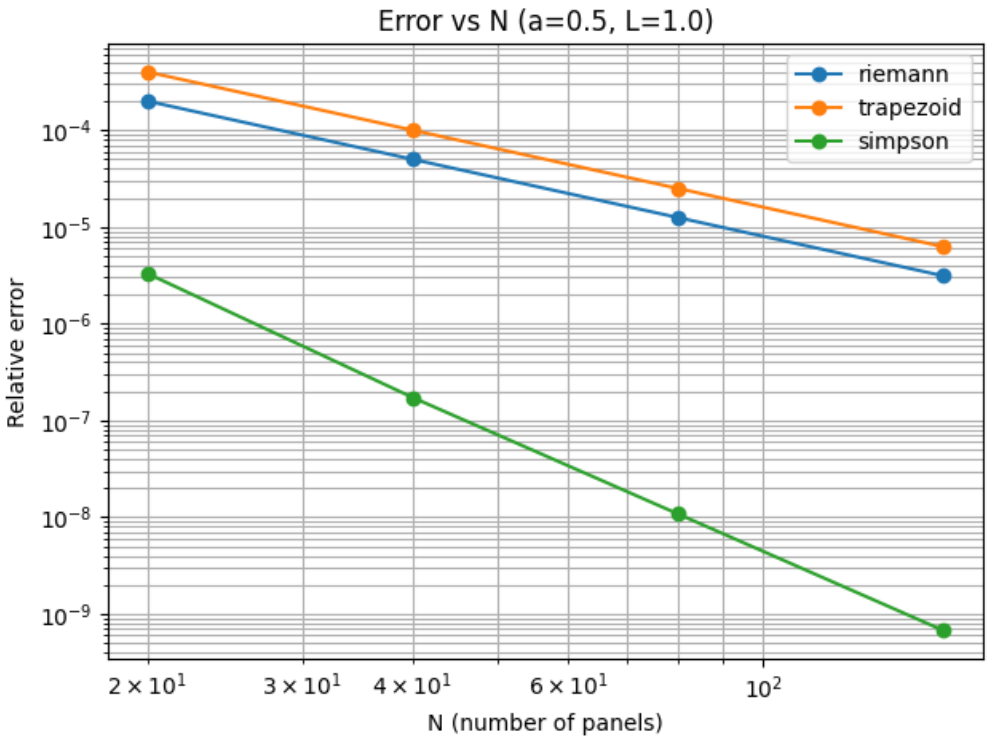

Figure 4: Relative error vs. N for Riemann, trapezoid, and Simpson's rule compared against the analytic solution.

Example 2: Field Profile and Physical Validation

Next, $E_y(a)$ as a function of the distance a was evaluated, with $L = 1.0$ and using Simpson's rule with $N = 400$. Figure 5 shows the numerical results compared against

the analytic solution, with good agreement. Two expected physical properties were confirmed:

1. In the far-field limit $a \gg L$, the field decays as

$$E_y(a) \sim \frac{1}{4\pi\varepsilon_0} \frac{2\lambda L}{a^2},$$

consistent with dipole scaling.

2. As a decreases toward the line, $E_y(a)$ increases monotonically, as shown in both the numerical and analytic curves.

This validates not only the correctness of the integration but also its ability to reproduce expected limiting behaviors of the physical system.

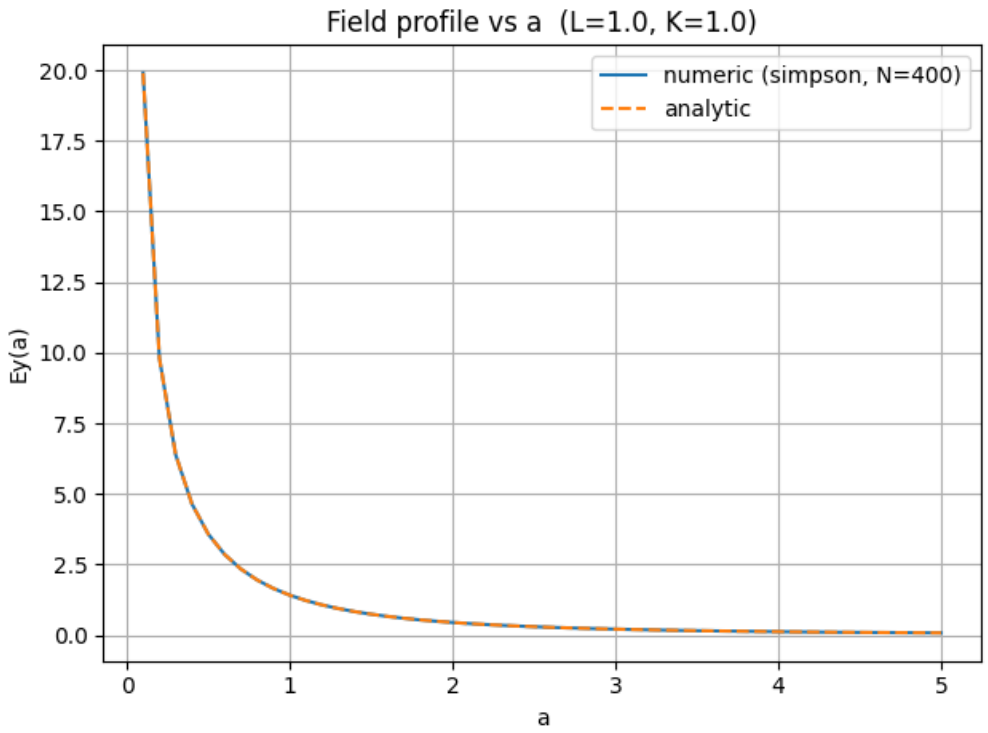

Figure 5: Field profile $E_y(a)$ computed numerically (Simpson, $N = 400$) and compared to the analytic result. The expected near-field increase and far-field $1/a^2$ decay are both reproduced.

By running "compare scipy" the implementations were verified against `SciPy`

modules `quad`, `trapezoid`, `simpson`:

```
Parameters: L=1.0, a=0.5, K=1.0
My trapezoid (N=  200): 3.57769445341
Analytic                    : 3.577708764
Rel. error (mine vs exact)  : 3.999931e-06

My riemann   (N=  200): 3.5777159192
Rel. error (mine vs exact)  : 1.999939e-06

My simpson   (N=  200): 3.57770876301
Rel. error (mine vs exact)  : 2.772879e-10

SciPy comparisons:
  quad (adaptive)        : 3.577708764   (reported abs err ~ 4.0e-14)
 trapezoid (SciPy)       : 3.57769445341   (same grid as mine)
 simpson  (SciPy)        : 3.57770876301   (same grid as mine)

Relative errors vs analytic:
  quad       : 1.241267e-16
  trapezoid  : 3.999931e-06
  simpson    : 2.772881e-10
```

Error Analysis

One situation where the code performs noticeably worse than the exact solution is when using the Riemann sum or trapezoidal rule with a relatively small number of panels N. Here, the error relative to the analytic solution is on the order of 10^{-4}, while Simpson's rule already achieves errors near 10^{-7} for the same N. This discrepancy arises because the Riemann and trapezoidal rules only achieve $\mathcal{O}(1/N^2)$

accuracy, while Simpson's rule benefits from $\mathcal{O}(1/N^4)$ convergence. As a result, the less accurate methods require substantially larger N to achieve comparable precision, leading to inefficiency. These errors could be improved by either increasing the number of panels N, at the cost of computation time, or by using higher-order integration methods such as Simpson's rule or Gaussian quadrature. External algorithms in `SciPy` demonstrate the same accuracy with efficient adaptive quadrature schemes, highlighting that the simpler self-implemented methods are limited in both precision and efficiency for demanding cases [14][15][18].

Riemann Sum To determine global and local truncation errors, we let $I = \int_A^B g(x)\,dx$, with $g \in C^k$ on $[A, B]$ and uniform panels $x_i = A + ih$, $h = (B - A)/N$. On each panel $[x_i, x_{i+1}]$,

$$\int_{x_i}^{x_{i+1}} g(x)\,dx = h\,g\big(x_i + \tfrac{h}{2}\big) + \frac{h^3}{24}\,g''(\xi_i), \quad \xi_i \in (x_i, x_{i+1}).$$

Summing N panels yields a total error $\mathcal{O}(Nh^3) = \mathcal{O}(h^2)$:

$$\int_A^B g(x)\,dx = h \sum_{i=0}^{N-1} g\big(x_i + \tfrac{h}{2}\big) + \mathcal{O}(h^2).$$

So, we see that the composite midpoint rule is second order [14][15][18].

Trapezoidal Rule On $[x_i, x_{i+1}]$,

$$\int_{x_i}^{x_{i+1}} g(x)\,dx = \frac{h}{2}\big(g(x_i) + g(x_{i+1})\big) - \frac{h^3}{12}\,g''(\eta_i), \quad \eta_i \in (x_i, x_{i+1}),$$

so the panel error is $\mathcal{O}(h^3)$ and the composite error is again $\mathcal{O}(Nh^3) = \mathcal{O}(h^2)$. The trapezoidal rule is also second order globally with a different error constant [14][15][18].

Simpson's Rule On two panels $[x_{2j}, x_{2j+2}]$ with midpoint x_{2j+1},

$$\int_{x_{2j}}^{x_{2j+2}} g(x)\,dx = \frac{h}{3}\Big(g_{2j} + 4g_{2j+1} + g_{2j+2}\Big) - \frac{h^5}{90}\,g^{(4)}(\zeta_j), \quad \zeta_j \in (x_{2j}, x_{2j+2}),$$

so the defect per pair is $\mathcal{O}(h^5)$ and the composite error over N panels (with N even) is $\mathcal{O}(Nh^5) = \mathcal{O}(h^4)$. Simpson's rule is fourth order globally. The convergence plot confirms these predictions as quadrature, midpoint and trapezoid display a $\mathcal{O}(h^2)$ decay of error while Simpson exhibits $\mathcal{O}(h^4)$. These observations match the derived truncation errors and validate the implementations [14][15][18].

4 Conclusion

In this project, numerical methods for two problems, the time evolution of the driven–damped harmonic oscillator and the electric field of a finite line of charge, were implemented, validated, and analyzed. For the oscillator, the explicit Euler and Runge–Kutta 4 (RK4) schemes were compared against analytic solutions and external `SciPy` algorithms. As expected, Euler showed only first–order accuracy, with a global error of $\mathcal{O}(10^{-1})$ for step size $h = 0.1$, while RK4 achieved fourth–order accuracy with a global error of $\mathcal{O}(10^{-6})$ for the same step size. Energy conservation provided a good physical test as over $t = 200$, Euler exhibited substantial energy drift, with $\Delta E \sim 10^{+4}$ growth, whereas RK4 conserved energy to within $\Delta E \sim 10^{-7}$. The frequency response analysis further confirmed that the RK4 integrator correctly captured the resonance peak at $\Omega_{\text{peak}} \approx 0.9975\,\text{rad/s}$, in good agreement with theory, and reproduced the expected $\pi/2$ phase lag at resonance.

For the integral problem, the analytic comparison was $E_y(a = 0.5, L = 1.0) = 3.577708764$. Using $N = 200$ panels, the Riemann midpoint and trapezoidal rules achieved relative errors of 2.0×10^{-6} and 4.0×10^{-6} respectively, consistent with their second–order convergence. Simpson's rule achieved greater accuracy, with a

relative error of 2.8×10^{-10}, consistent with its fourth–order convergence. Validation against `SciPy`'s `quad` algorithm ($3.577708764 \pm 4.0 \times 10^{-14}$) confirmed that the implementations produce results of the same quality as established external algorithms. The field profile study reproduced two expected physical properties, namely the near–field growth of $E_y(a)$ as $a \to 0$, and the far–field $1/a^2$ decay for $a \gg L$.

Overall, this project shows the different strengths and weaknesses of the implemented methods. Low–order algorithms like Euler or the midpoint rule capture qualitative trends but have disadvantages such as large errors or instability. Higher–order schemes such as RK4 and Simpson's rule provide good quantitative accuracy. The combination of analytic comparisons, error analysis, and validation of physical limits confirms the correctness of the implementations.

References

[1] University of Bristish Columbia. *Fourth-Order Runge-Kutta Method*. URL: https://personal.math.ubc.ca/~israel/m215/runge/runge.html.

[2] Physics Catalyst. *Electric field due to Line Charge*. URL: https://physicscatalyst.com/elec/electric-field-line-charge.php#google_vignette.

[3] E. Cheever. *Fourth Order Runge-Kutta*. 2022. URL: https://lpsa.swarthmore.edu/NumInt/NumIntFourth.html.

[4] D. Cline. *Linearly-damped Free Linear Oscillator*. URL: https://ocw.mit.edu/courses/8-03sc-physics-iii-vibrations-and-waves-fall-2016/d0253416dabaf95ea866e046e63a8e74_MIT8_03SCF16_Text_Ch1.pdf.

[5] R. C. Cooper. *Get with the oscillations*. 2023. URL: https://cooperrc.github.io/computational-mechanics/module_03/03_Get_Oscillations.html.

[6] M. Fowler. *Oscillations III: Damped Driven Oscillator*. URL: https://galileo.phys.virginia.edu/classes/152.mf1i.spring02/Oscillations4.htm.

[7] O. Golberg. *Adaptive Stepsize Numerical Methods for Solving Ordinary Differential Equations*. 2007. URL: https://dspace.mit.edu/bitstream/handle/1721.1/55903/18-034Spring-2007/NR/rdonlyres/Mathematics/18-034Spring-2007/Projects/euler.pdf.

[8] HyperPhysics. *Driven Oscillator*. URL: http://hyperphysics.phy-astr.gsu.edu/hbase/oscdr.html.

[9] MIT. *Harmonic Oscillation*. URL: https://ocw.mit.edu/courses/8-03sc-physics-iii-vibrations-and-waves-fall-2016/d0253416dabaf95ea866e046e63a8e74_MIT8_03SCF16_Text_Ch1.pdf.

[10] Physics Tutor Online. *Resonance and Damping*. URL: https://www.physicstutoronline.co.uk/wp-content/uploads/2018/09/Resonance-and-Damping-NOTES.pdf.

[11] University of Saskatchewan. *CHAPTER 3: Basic methods, basic concepts*. URL: https://www.cs.usask.ca/~spiteri/M314/notes/AP/chap3.pdf.

[12] W. Smith. *Euler's Method: Definition, Properties, Applications, and Examples*. 2023. URL: https://www.storyofmathematics.com/eulers-method/.

[13] UCSD. *Deriving the Trapezoidal Rule Error*. URL: https://mathweb.ucsd.edu/~ebender/20B/77_Trap.pdf.

[14] McMaster University. *Errors of numerical integration*. URL: https://dmpeli.math.mcmaster.ca/Matlab/Math4Q3/Lecture3-1/Example3-4.html.

[15] Ohio State University. *Local and Global Error Bounds*. 2025. URL: https://ximera.osu.edu/laode/textbook/numericalSolutionsOfODEs/localAndGlobalErrorBoun

[16] Oklahoma State University. *Runge-Kutta method*. URL: https://math.okstate.edu/people/yqwang/teaching/math4513_fall11/Notes/rungekutta.pdf.

[17] M. Volvert and G. Kerschen. *Resonant phase lags of an oscillator with polynomial stiffness*. 2022. URL: https://nodycon.org/2023/papers/348/abstract_submissions/580/view_abstract#:~:text=The%20governing%20equations%20show%20that,3%20:%201%20superharmonic%20(Fig..

[18] N. et al. Wakefield. *Numerical Integration*. URL: https://mathbooks.unl.edu/Calculus/sec-5-9-num-int.html.

[19] P. Walls. *Riemann Sums*. URL: https://patrickwalls.github.io/mathematicalpython/integration/riemann-sums/.