Numerical Methods For Ordinary Differential Equations

References: > 1. Chasnov, J. R. (2012). Numerical methods. Hong Kong University of Science and Technology. https://www.math.hkust.edu.hk/~machas/numerical-methods.pdf > 2. Liu, J., Spiegel, M. R. (1999). Mathematical Handbook of Formulas and Tables. United Kingdom: McGraw-Hill.

Here we give methods to solve the following initial-value problem of an ordinary differential equation:

$$
\begin{cases} \dfrac{dx}{dt} = f(t,x) \\ x(t_0) = x_0 \end{cases} \tag{1}
$$

The methods will use a computational grid:

$$
t_n = t_0 + nh \tag{2}
$$

where h is the grid size.

Example

Consider $f(t,x) = rx(K-x)$, with the following parameters:

```
sage vscode={"languageId": "sage"} r = 0.15 K = 100 t0 = 0 tf = 1
x0 = 1 n = 10 step_size =  (tf-t0)/n step_size
```

“‘sage vscode={“languageId”: “sage”} t = var(‘t’)

x = function(‘x’)(t)

ode = x.diff(t) == rx(K-x)

solution = desolve(ode, dvar=x, ivar=t, ics=[t0,x0])

solution

````
```sage vscode={"languageId": "sage"}
u = var('u')
solution = solution.subs({x: u})
solution
````

```
sage vscode={"languageId": "sage"} solution.solve(u, to_poly_solve=True)
```

```
sage vscode={"languageId": "sage"} x_exact = solution.solve(u,
to_poly_solve=True)[0].rhs() x_exact
```

```
sage vscode={"languageId": "sage"} custom_grid = [t0 + i*step_size
for i in range(n)] exact_solution = [[_t.n(), x_exact(t=_t).n()]
for _t in custom_grid]
```
```

```
sage vscode={"languageId": "sage"} t,x = var('t x') numerical_solution
= desolve_rk4( r*x*(K-x), x, ivar = t, ics=[t0,x0], end_points=tf,
step=step_size) sage_solution = list_plot(exact_solution, plotjoined=True,
axes_labels=['$t$', '$x(t)$'], gridlines=True, alpha=0.5, color="green")
sage_solution += list_plot(numerical_solution, plotjoined=True,
alpha=0.5, color="red", linestyle=":") sage_solution.show()
```

Euler Method: Manual Reproduction of the Code

We consider the ODE:

$$\frac{dx}{dt} = 0.15\,x(100 - x), \quad x(0) = 1$$

with parameters:

- $r = 0.15$
- $K = 100$
- $t_0 = 0$, $t_f = 1$
- $x_0 = 1$
- $n = 10$, $h = \frac{1-0}{10} = 0.1$

Euler's method iterates:

$$x_{n+1} = x_n + h \cdot f(t_n, x_n) \quad \text{where} \quad f(t, x) = 0.15\,x(100 - x)$$

Iteration Table

Step n	t_n	x_n	$f(t_n, x_n)$	$x_{n+1} = x_n + hf(t_n, x_n)$
0	0.0	1.000	14.850	2.485
1	0.1	2.485	36.3076	6.116
2	0.2	6.116	85.9874	14.714
3	0.3	14.714	188.442	33.558
4	0.4	33.558	335.419	67.100
5	0.5	67.100	329.207	100.021
6	0.6	100.021	-0.315	99.990
7	0.7	99.990	0.149	100.005
8	0.8	100.005	-0.075	99.998
9	0.9	99.998	0.003	100.000

"'sage vscode={"languageId": "sage"} def euler_method(f, x0, t0, t_end, h): """
Implements the Euler method for solving x' = f(t, x)

```
Parameters:
    f: a function of (t, x)
    x0: initial value x(t0)
```

```
    t0: initial time
    t_end: final time
    h: step size

Returns:
    A list of (t, x) points
"""
steps = int((t_end - t0) / h)
t_vals = [t0]
x_vals = [x0]

t = t0
x = x0

for _ in range(steps):
    x = x + h * f(t, x)
    t = t + h
    t_vals.append(t)
    x_vals.append(x)

return list(zip(t_vals, x_vals))
```

````
```sage vscode={"languageId": "sage"}
t, x = var('t x')
f(t, x) = r * x * (K - x)  # logistic equation

solution = euler_method(f, x0=x0, t0=t0, t_end=tf, h=step_size)

# Plot
euler_solution = list_plot(solution, plotjoined=True, axes_labels=['$t$', '$x(t)$'], gridlin
compare_euler = sage_solution + euler_solution
compare_euler.plot()
````

“‘sage vscode={“languageId”: “sage”} def modified_euler_method(f, x0, t0, t_end, h): """ Implements the Modified Euler method (Heun’s Method) for solving x’ = f(t, x)

```
Parameters:
    f: a function of (t, x)
    x0: initial value x(t0)
    t0: initial time
    t_end: final time
    h: step size

Returns:
    A list of (t, x) points
```
```

```
"""
steps = int((t_end - t0) / h)
t_vals = [t0]
x_vals = [x0]

t = t0
x = x0

for _ in range(steps):
    k1 = h*f(t, x)
    k2 = h*f(t + h, x + k1)
    x = x + 1/2 * (k1 + k2)
    t = t + h
    t_vals.append(t)
    x_vals.append(x)

return list(zip(t_vals, x_vals))
```

```
```sage vscode={"languageId": "sage"}
solution = modified_euler_method(f, x0=x0, t0=t0, t_end=tf, h=step_size)

# Plot the numerical approximation
heun_solution = list_plot(solution, plotjoined=True, axes_labels=['$t$', '$x(t)$'], gridline
compare_heun = sage_solution + heun_solution
compare_heun.plot()
```

"'sage vscode={"languageId": "sage"} def rk2_general_form(f, x0, t0, t_end, h=0.01, alpha=1/2, beta=1/2, a=0, b=1): """ General second-order Runge-Kutta method using parameters alpha, beta, a, b.

```
Solves x' = f(t, x) over [t0, t_end] with initial condition x(t0) = x0.

Parameters:
    f: right-hand side function f(t, x)
    x0: initial condition x(t0)
    t0: initial time
    t_end: final time
    h: time step size
    alpha: time increment coefficient for k2
    beta: slope coefficient for k2
    a, b: weights for k1 and k2 in the update

Returns:
    List of (t, x) points approximating the solution
"""
steps = int((t_end - t0) / h)
t_vals = [t0]
```
```

````
x_vals = [x0]

t = t0
x = x0

for _ in range(steps):
    k1 = h * f(t, x)
    k2 = h * f(t + alpha * h, x + beta * k1)
    x = x + a * k1 + b * k2
    t = t + h
    t_vals.append(t)
    x_vals.append(x)

return list(zip(t_vals, x_vals))
```sage vscode={"languageId": "sage"}
sol = rk2_general_form(f, x0=x0, t0=t0, t_end=tf, h=step_size)

heun_solution  = list_plot(sol, plotjoined=True, axes_labels=['$t$', '$x(t)$'], gridlines=Tr
compare_heun = heun_solution  + sage_solution
compare_heun.plot()
````

“‘sage vscode={“languageId”: “sage”} sol = rk2_general_form(f, x0=x0, t0=t0, t_end=tf, h=step_size, alpha=3/4, beta=3/4, a=1/3, b=2/3)

ralston_solution = list_plot(sol, plotjoined=True, axes_labels=[‘t’, ’$x(t)$’], gridlines=True, linestyle=“-.”) compare_heun = ralston_solution + sage_solution compare_heun.plot()

````
```sage vscode={"languageId": "sage"}
def rk4(f, x0, t0, t_end, h=0.01):
    """
    Fourth-order Runge-Kutta method as specified in the image.

    Solves x' = f(t, x) over [t0, t_end] with initial condition x(t0) = x0.

    Parameters:
        f: right-hand side function f(t, x)
        x0: initial condition x(t0)
        t0: initial time
        t_end: final time
        h: time step size

    Returns:
        List of (t, x) points approximating the solution
    """
    steps = int((t_end - t0) / h)
````

```
    t_vals = [t0]
    x_vals = [x0]

    t = t0
    x = x0

    for _ in range(steps):
        # Calculate the four k values according to the formulas
        k1 = h * f(t, x)
        k2 = h * f(t + 0.5 * h, x + 0.5 * k1)
        k3 = h * f(t + 0.5 * h, x + 0.5 * k2)
        k4 = h * f(t + h, x + k3)

        # Update x using the weighted average
        x = x + (1/6) * (k1 + 2*k2 + 2*k3 + k4)

        # Increment time
        t = t + h

        # Store results
        t_vals.append(t)
        x_vals.append(x)

    return list(zip(t_vals, x_vals))
```

"'sage vscode={"languageId": "sage"} sol = rk4(f, x0=x0, t0=t0, t_end=tf, h=step_size)

ralston_solution = list_plot(sol, plotjoined=True, axes_labels=['t', '$x(t)$'], gridlines=True, linestyle="-.") compare_heun = ralston_solution + sage_solution compare_heun.plot() "'