Group:A/B/C	No (in list)	Student ID	Full Name (Surname, Name)	HW
B		e20200447	Nang Sreynich	07

Use the indicated method to approximate the solutions to the initial-value problems

$$y' = t^{-2}(\sin 2t - 2ty),\ 1 \le t \le 2,\ y(1) = 2,\ h = 0.1;$$

actual solution

$$y(t) = \frac{1}{2}t^{-2}(4 + \cos 2 - \cos 2t),$$

and compare the results to the actual values.

1. Runge-Kutta third-order method.
2. Heun's third-order method.
3. Ralston's third-order method.
4. Third-order Strong Stability Preserving Runge-Kutta.

	t	exact	y_Midpoint3	e_Midpoint3	y_Heun3	e_Heun3	y_Ralston3	e_Ralston3	y_SSPRK3	e_SSPRK3
0	1	1.881251608	2	0.118748392	2	0.118748392	2	0.118748392	2	0.118748392
1	1.1	1.625974172	1.723999664	0.098025492	1.723941286	0.097967114	1.723999664	0.098025492	1.724064	0.098089828
2	1.2	1.417968783	1.500271762	0.082302979	1.500189935	0.082221152	1.500271762	0.082302979	1.50036299	0.082394208
3	1.3	1.243563649	1.313651633	0.070087984	1.313562824	0.069999175	1.313651633	0.070087984	1.313751978	0.070188329
4	1.4	1.094025184	1.154432629	0.060407446	1.154344647	0.060319463	1.154432629	0.060407446	1.154533547	0.060508363
5	1.5	0.963633083	1.01623761	0.052604526	1.016154038	0.052520954	1.01623761	0.052604526	1.016335062	0.052701979
6	1.6	0.848564679	0.894787254	0.046222575	0.894709561	0.046144883	0.894787254	0.046222575	0.89487946	0.046314782
7	1.7	0.746220514	0.787156513	0.040935999	0.7870851	0.040864586	0.787156513	0.040935999	0.787242836	0.041022322
8	1.8	0.654801666	0.69130918	0.036507514	0.691243911	0.036442246	0.69130918	0.036507514	0.691389558	0.036587892
9	1.9	0.573036578	0.605797428	0.03276085	0.605737913	0.032701335	0.605797428	0.03276085	0.605872081	0.032835504
10	2	0.5	0.529562825	0.029562825	0.529508565	0.029508565	0.529562825	0.029562825	0.5296321	0.0296321

```python
from collections.abc import Callable
from typing import Literal
from typing import Optional
import pandas as pd
import numpy as np

def RungeKutta3(f: Callable[[float, float], float],
                t_span: list,
                y_init: float,
                n: int,
                method: Optional[Literal['Midpoint3', 'Heun3', 'Ralston3', 'SSPR3']] = 'Heun3'
                ) -> pd.DataFrame:
    (a, b) = t_span
    h = (b-a) / n
    t = np.linspace(start=a, stop=b, num=n+1, dtype=np.float64)
    y = np.full_like(a=t, fill_value=np.nan, dtype=np.float64)
    y[0] = y_init

    if method == 'Heun3':
        c = np.array(object=[0, 1/3, 2/3], dtype=np.float64)
        a = np.array(object=[[0, 0, 0],
                             [1/3, 0, 0],
                             [0, 2/3, 0]],
                             dtype=np.float64)
        b = np.array(object=[1/4, 0, 3/4], dtype=np.float64)
    elif method == 'Kutta3':
        c = np.array(object=[0, 1/2, 1], dtype=np.float64)
        a = np.array(object=[[0, 0, 0],
                             [1/2, 0, 0],
                             [-1, 2, 0]],
                             dtype=np.float64)
        b = np.array(object=[1/6, 2/3, 1/6], dtype=np.float64)
    elif method == 'SSPRK3':
        c = np.array([0, 1/2, 1], dtype=np.float64)
        a = np.array([[0, 0, 0], [1/2, 0, 0], [-1, 2, 0]], dtype=np.float64)
        b = np.array([1/6, 2/3, 1/6], dtype=np.float64)

    else:
        c = np.array(object=[0, 1/2, 3/4], dtype=np.float64)
        a = np.array(object=[[0, 0, 0],
                             [1/2, 0, 0],
                             [0, 3/4, 0]],
                             dtype=np.float64)
        b = np.array(object=[2/9, 1/3, 4/9], dtype=np.float64)

    for i in range(0, n):
        k1 = f(t[i], y[i])
        k2 = f(t[i] + c[1]*h , y[i] + h*(a[1][0]*k1))
        k3 = f(t[i] + c[2]*h , y[i] + h*(a[2][0]*k1 + a[2][1]*k2))
        y[i+1] = y[i] + h*(b[0]*k1 + b[1]*k2 + b[2]*k3)

    df = pd.DataFrame(data={'t': t,'y': y})
    return df
```

```python
if __name__ == '__main__':
    from math import sin, cos
    def f(t:float, y: float) -> float:
        return t**(-2)*(sin(2*t) - 2*t*y)
    def y(t: float) -> float:
        return 0.5*t**(-2)*(4 + cos(4) - cos(2*t))

    t_span = [1.0, 2.0]
    y_init = 2.0
    n = 10
    methods = ['Midpoint3', 'Heun3', 'Ralston3', 'SSPRK3']

    t = np.linspace(start=t_span[0], stop=t_span[1], num=n+1, dtype=np.float64)
    df = pd.DataFrame(data={'t': t})
    df['exact'] = df['t'].apply(func=y)

    for method in methods:
        dfi = RungeKutta3(f=f, t_span=t_span, y_init=y_init, n=n, method=method)
        df[f'y_{method}'] = dfi['y']
        df[f'e_{method}'] = abs(df['exact'] - df[f'y_{method}'])

    print(df)
    df.to_excel('HW07.xlsx')
```

Use the Runge-Kutta-Fehlberg method with tolerance $TOL = 10^{-6}$, $hmax = 0.5$, and $hmin = 0.05$ to approximate the solutions to the following initial-value problems. Compare the results to the actua lvalues.

1. $y' = y/t - (y/t)^2,\ 1 \le t \le 4,\ y(1) = 1;$

   actual solution $y(t) = 1/(1 + \ln t)$.

2. $y' = (2 + 2t^3)y^3 - ty,\ 0 \le t \le 2,\ y(0) = 1/3;$

   actual solution $y(t) = (3 + 2t^2 + 6e^{t^2})^{-1/2}$.

1. $y' = y/t - (y/t)^2, 1 \le t \le 4, y(1) = 1;$

   actual solution $y(t) = t/(1 + \ln t)$.

t	Approx	Actual	Error
1	1	1	0
1.1120017	1.005279551	1.005279497	5.3985E-08
1.1751459	1.011842386	1.011842333	5.3076E-08
1.2432463	1.020957449	1.020957397	5.222E-08
1.3218061	1.033469233	1.033469181	5.1684E-08
1.4141522	1.050219495	1.050219444	5.1572E-08
1.5241774	1.072265833	1.072265781	5.1986E-08
1.6571461	1.101022907	1.101022854	5.3041E-08
1.8204535	1.138434074	1.138434019	5.4845E-08
2.0247792	1.187233082	1.187233025	5.7435E-08
2.2860293	1.251373401	1.25137334	6.0583E-08
2.6288072	1.336774452	1.336774389	6.326E-08
3.0930705	1.452715827	1.452715765	6.2112E-08
3.5930705	1.576594747	1.576594685	6.2286E-08
4	1.6762392	1.676239137	6.3625E-08

2. $y' = (2 + 2t^3)y^3 - ty, 0 \le t \le 2, y(0) = 1/3;$

   actual solution $y(t) = (3 + 2t^2 + 6e^{t^2})^{-1/2}$.

t	Approx	Actual	Error
0	0.333333333	0.333333333	0
0.193430327	0.342039189	0.327849911	0.014189277
0.335273277	0.340902366	0.317202547	0.023699819
0.495442667	0.33212877	0.299339378	0.032789392
0.859809707	0.288399382	0.242215144	0.046184239
1.17000644	0.234896549	0.184665576	0.050230973
1.435737915	0.183644851	0.135754099	0.047890752
1.687862313	0.134861249	0.094361353	0.040499896
1.849032492	0.105884576	0.07197103	0.033913546
1.995549252	0.082328456	0.054817835	0.027510621
2	0.081662984	0.054345507	0.027317478

```python
from _collections_abc import Callable
import pandas as pd
import numpy as np
def RungekuttaFehlberg (f: Callable [[float, float], float],
                        t_span: list or tuple,
                        y_init: float,
                        TOL: float,
                        hmax: float,
                        hmin: float) -> pd.DataFrame:
    (a, b) = t_span
    t = a
    y = y_init
    h = hmax
    t_list = []
    y_list = []
    h_list = []
    err_list = []
    def runge_kutta_step(t, y, h):
        k1= h*f(t, y)
        k2= h* f(t + h / 4, y + k1 / 4)
        k3= h*f(t + 3 * h / 8, y + 3* k1 / 32 +9 * k2 / 32)
        k4= h* f(t + 12* h/13, y + 1932 *k1 / 2197 -7200 * k2/2197 + 7296* k3/2197)
        k5= h * f(t + h, y + 439 *k1 / 216 - 8*k2 + 3680* k3 / 513 - 845 *k4 / 4104)
        k6= h * f(t + h / 2, y - 8 * k1/ 27 + 2 * k2 - 3544 * k3 / 2565 + 1859*k4 / 4104 - 11* k5 / 40)
        y_new = y + 25 *k1 / 216 + 1408* k3 / 2565 +2197 *k4 / 4104 - k5 / 5
        y_hat = y + 16 *k1 / 135 + 6656 * k3 / 12825 +28561*k4 / 56430- 9 * k5 / 50 + 2 * k6 / 55
        return y_new, y_hat
    delta = 0
    while t < b:
        if t + h > b:
            h = b - t
        y_new, y_hat= runge_kutta_step(t, y, h)
        R = abs(y_hat- y_new)
        if R <= TOL:
            t_list.append(t)
            y_list.append(y_new)
            h_list.append(h)
            err_list.append(R)
            t = t + h
            y = y_new
            delta = 0.84* (TOL / R) *0.25
            delta= max(min(delta, 4), 0.1)
        if R == 0.0:
            h=hmax
        else:
            h = delta * h
            h = max(min(h, hmax), hmin)
    df = pd.DataFrame (data={"t": t_list, "y": y_list, "error": err_list, "h": h_list})
    return df
```

```python
if __name__=="__main__":
    import math
    def f(t, y):  return  (2+2*(t(3)))*y**(3) - t*y
    t_span= [0, 2]
    y_init = 1/3
    TOL = 1e-6
    hmax = 0.5
    hmin = 0.05
    df = RungekuttaFehlberg (f=f, t_span=t_span, y_init=y_init, TOL=TOL, hmax=hmax, hmin=hmin)
    def y(t: float) -> float:
        return (3 + 2 * t**(2 )+ 6 * math.exp(t**2)) ** (-1 / 2)
    df ["exact"]= df ["t"].apply(func=y)
    df ["error"]= abs(df["exact"] - df ["y"])
    print(df)
```