```python
import numpy as np
import matplotlib.pyplot as plt
import os

# Configure figures for production
WIDTH = 495.0  # the number latex spits out
FACTOR = 1.0   # the fraction of the width you'd like the figure to occupy
fig_width_pt  = WIDTH * FACTOR

inches_per_pt = 1.0 / 72.27
golden_ratio  = (np.sqrt(5) - 1.0) / 2.0  # because it looks good

fig_width_in  = fig_width_pt * inches_per_pt  # figure width in inches
fig_height_in = fig_width_in * golden_ratio   # figure height in inches
fig_dims      = [fig_width_in, fig_height_in] # fig dims as a list


def Solver(s, t_end, show_plot=False):
    # Problem Parameters
    L = 1.            # Domain lenghth       [n.d.]
    T0 = 0.           # Initial temperature  [n.d.]
    T1 = 1.           # Boundary temperature [n.d.]
    N = 21

    # Set-up Mesh
    x = np.linspace(0, L, N)
    dx = x[1] - x[0]

    # Calculate time-step
    dt = s * dx ** 2.0

    # Initial Condition with boundary conditions
    T_initial = [T0] * N
    T_initial[0] = T1
    T_initial[N - 1] = T1

    # Explicit Numerical Solution
    T_explicit = Explicit(np.array(T_initial).copy(), t_end, dt, s)

    # Implicit Numerical Solution
    T_implicit = Implicit(np.array(T_initial).copy(), t_end, dt, s)

    # Analytical Solution
    T_analytic = np.array(T_initial).copy()
    for i in range(0, N):
        T_analytic[i] = Analytic(x[i], t_end)

    # Find the RMS
    RMS = RootMeanSquare(T_implicit, T_analytic)
    ExplicitRMS = RootMeanSquare(T_explicit, T_analytic)

    # Format our plots
    plt.figure(figsize=fig_dims)
    plt.axis([0, L, T0, T1])
    plt.xlabel('Length [nd]')
    plt.ylabel('Temperature [nd]')
    plt.title('s = ' + str(s)[:5] + ', t = ' + str(t_end)[:4])

    # ...and finally plot
    plt.plot(x, T_explicit, 'xr', markersize=9, label='Explicit Solution')
    plt.plot(x, T_implicit, '+g', markersize=9, label='Implicit Solution')
    plt.plot(x, T_analytic, 'ob', markersize=9, mfc='none', label='Analytic Solution')
    plt.legend(loc='lower right')

    # Save plots
    save_name = 'proj_1_s_' + str(s)[:5] + '_t_' + str(t_end) + '.pdf'
```

```python
    try:
        os.mkdir('figures')
    except Exception:
        pass

    plt.savefig('figures/' + save_name, bbox_inches='tight')
    if show_plot:
        plt.show()
    plt.clf()

    return RMS, ExplicitRMS


def Explicit(Told, t_end, dt, s):
    """
    This function computes the Forward-Time, Centered-Space (FTCS) explicit
    scheme for the 1D unsteady heat diffusion problem.
    """
    N = len(Told)
    time = 0.
    Tnew = Told

    while time <= t_end:
        for i in range(1, N - 1):
            Tnew[i] = s * Told[i + 1] + (1 - 2.0 * s) * Told[i] + s * Told[i - 1]

        Told = Tnew
        time += dt

    return Told


def Implicit(Told, t_end, dt, s):
    """
    This function computes the Forward-Time, Centered-Space (FTCS) implicit
    scheme for the 1D unsteady heat diffusion problem.
    """
    N = len(Told)
    time = 0.

    # Build our 'A' matrix
    a = [-s] * N
    a[0], a[-1] = 0, 0
    b = [1 + 2 * s] * N
    b[0], b[-1] = 1, 1          # hold boundary
    c = a

    while time <= t_end:
        Tnew = TDMAsolver(a, b, c, Told)

        Told = Tnew
        time += dt

    return Told


def RootMeanSquare(a, b):
    """
    This function will return the RMS between two lists (but does no checking
    to confirm that the lists are the same length).
    """
    N = len(a)

    RMS = 0.
    for i in range(0, N):
        RMS += (a[i] - b[i]) ** 2.

```

```python
    RMS = RMS ** (1. / 2.)
    RMS /= N

    return RMS


def TDMAsolver(a, b, c, d):
    """
    Tridiagonal Matrix Algorithm (a.k.a Thomas algorithm).
    """
    N = len(a)
    Tnew = d

    # Initialize arrays
    gamma = np.zeros(N)
    xi = np.zeros(N)

    # Step 1
    gamma[0] = c[0] / b[0]
    xi[0] = d[0] / b[0]

    for i in range(1, N):
        gamma[i] = c[i] / (b[i] - a[i] * gamma[i - 1])
        xi[i] = (d[i] - a[i] * xi[i - 1]) / (b[i] - a[i] * gamma[i - 1])

    # Step 2
    Tnew[N - 1] = xi[N - 1]

    for i in range(N - 2, -1, -1):
        Tnew[i] = xi[i] - gamma[i] * Tnew[i + 1]

    return Tnew


def Analytic(x, t):
    """
    The analytic answer is 1 - Sum(terms). Though there are an infinite
    number of terms, only the first few matter when we compute the answer.
    """
    result = 1
    large_number = 1E6

    for k in range(1, int(large_number) + 1):
        term = ((4. / ((2. * k - 1.) * np.pi)) *
                np.sin((2. * k - 1.) * np.pi * x) *
                np.exp(-(2. * k - 1.) ** 2. * np.pi ** 2. * t))

        # If subtracting the term from the result doesn't change the result
        # then we've hit the computational limit, else we continue.
        # print '{0} {1}, {2:.15f}'.format(k, term, result)
        if result - term == result:
            return result
        else:
            result -= term


def main():
    """
    Main function to call solver over assigned values and create some plots to
    look at the trends in RMS compared to s and t.
    """
    # Loop over requested values for s and t
    s = [1. / 6., .25, .5, .75]
    t = [0.03, 0.06, 0.09]

    RMS = []
    with open('results.dat', 'w+') as f:
```

```python
        for i, s_ in enumerate(s):
            sRMS = [0] * len(t)
            for j, t_ in enumerate(t):
                sRMS[j], ExplicitRMS = Solver(s_, t_, False)
                f.write('{0:.3f} {1:.2f} {2:.2e} {3:.2e} \n'.format(s_, t_, sRMS[j], ExplicitRMS))
                # print i, j, sRMS[j]
            RMS.append(sRMS)

    # Convert to np array to make this easier...
    RMS = np.array(RMS)

    # Check for trends in RMS vs t
    plt.figure(figsize=fig_dims)
    plt.plot(t, RMS[0], '.r', label='s = 1/6')
    plt.plot(t, RMS[1], '.g', label='s = .25')
    plt.plot(t, RMS[2], '.b', label='s = .50')
    plt.plot(t, RMS[3], '.k', label='s = .75')
    plt.xlabel('t')
    plt.ylabel('RMS')
    plt.title('RMS vs t')
    plt.legend(loc='best')

    save_name = 'proj_1_rms_vs_t.pdf'
    plt.savefig('figures/' + save_name, bbox_inches='tight')
    plt.clf()

    # Check for trends in RMS vs s
    plt.figure(figsize=fig_dims)
    plt.plot(s, RMS[:, 0], '.r', label='t = 0.03')
    plt.plot(s, RMS[:, 1], '.g', label='t = 0.06')
    plt.plot(s, RMS[:, 2], '.b', label='t = 0.09')
    plt.xlabel('s')
    plt.ylabel('RMS')
    plt.title('RMS vs s')
    plt.legend(loc='best')

    save_name = 'proj_1_rms_vs_s.pdf'
    plt.savefig('figures/' + save_name, bbox_inches='tight')
    plt.clf()

if __name__ == "__main__":
    main()
```