Tolerance-Constrained Approximate Solution of ODEs via Picard Iteration

Paras Balani

August 20, 2025

Field	Information
Name	Paras Balani
ID	2023B4A70738H
Subject	Ordinary Differential Equation
Course Code	Math F312
Professor	K. Bhargav Kumar

Contents

Abstract

This work implements Picard iteration to solve the ODE $u' = u,\ u(0) = 1$ to a tolerance of 10^{-2}. The algorithm computes successive approximations $\varphi_n(t)$ symbolically, evaluates them numerically, and terminates when the maximum error from the exact solution e^t is achieved. Results confirm the expected convergence of the iterative series.

1 Problem

Find an approximate solution for the initial value problem:

$$\frac{du}{dt} = u, \quad u(0) = 1$$

such that the maximum error between the approximation and exact solution is less than a specified tolerance tol $= 10^{-2}$.

2 Mathematical Background

The Picard iteration method solves the initial value problem:

$$\frac{du}{dt} = f(t, u), \quad u(t_0) = u_0$$

by converting it to the integral equation:

$$u(t) = u_0 + \int_{t_0}^{t} f(s, u(s))ds$$

and then applying the iterative scheme:

$$\varphi_{n+1}(t) = u_0 + \int_{t_0}^{t} f(s, \varphi_n(s))ds$$

with initial approximation $\varphi_0(t) = u_0$.

For our specific problem $u' = u$, $u(0) = 1$, the iteration becomes:

$$\varphi_{n+1}(t) = 1 + \int_{0}^{t} \varphi_n(s)ds$$

3 Expected Output

The program generates the following sequence of approximations:

$$\begin{aligned}
\varphi_0(t) &= 1 \\
\varphi_1(t) &= 1 + t \\
\varphi_2(t) &= 1 + t + \frac{t^2}{2} \\
\varphi_3(t) &= 1 + t + \frac{t^2}{2} + \frac{t^3}{6} \\
&\vdots \\
\varphi_n(t) &= \sum_{k=0}^{n} \frac{t^k}{k!}
\end{aligned}$$

These converge to the exact solution:

$$u(t) = e^t = \sum_{k=0}^{\infty} \frac{t^k}{k!}$$

4 Python code

```python
import numpy as np
import matplotlib.pyplot as plt
from sympy import symbols, integrate, exp, lambdify


tol = 1e-2
t_range = (0, 2)
num_points = 100

t = symbols('t')
s = symbols('s')
u_prev = 1  # Initial constant function
t_values = np.linspace(t_range[0], t_range[1], num_points)
iterates = [np.full_like(t_values, u_prev)]  # Store numerical values

print("Picard Iterates for u' = u, u(0) = 1")
print("=" * 50)
print(f"     (t) = {u_prev}")

exact = np.exp(t_values)  # Exact solution

n = 0
max_error = float('inf')

while max_error > tol:
    n += 1

    # For the first iteration, u_prev is a constant (1)
    if isinstance(u_prev, (int, float)):
        integrand = u_prev  # Constant function
    else:
        integrand = u_prev.subs(t, s)  # Substitute t with s for integration

    try:
        # Perform the integration
        integral = integrate(integrand, (s, 0, t))
        u_current = 1 + integral

        # Convert to numerical function for evaluation
        if hasattr(u_current, 'subs'):
            u_func = lambdify(t, u_current, 'numpy')
            current_values = u_func(t_values)
        else:
            current_values = np.full_like(t_values, u_current)

        iterates.append(current_values)

        # Calculate maximum error
        max_error = np.max(np.abs(current_values - exact))

        print(f" _ {n}(t) = {u_current}")
        print(f"  Max error: {max_error:.6f}")

        # Check if tolerance is met
        if max_error <= tol:
            print(f"\ n   Tolerance {tol} achieved at iteration n = {n}")
            print(f"  Maximum error: {max_error:.6f}")
            break

        # Update for next iteration
        u_prev = u_current

    except Exception as e:
        print(f"Error in iteration {n}: {e}")
        break
```

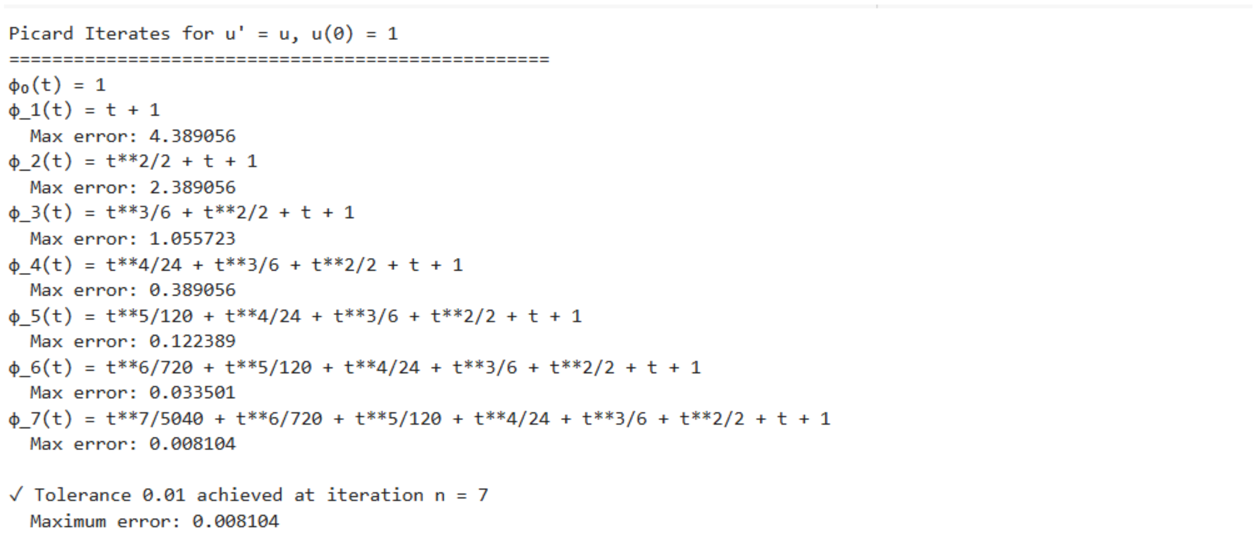
```python

# Plot the results
plt.figure(figsize=(10, 6))
plt.plot(t_values, exact, ’k-’, linewidth=2, label=’Exact: $e^t$’)

colors = [’r’, ’g’, ’b’, ’m’, ’c’]
for i, iterate in enumerate(iterates):
    if i < len(colors):
        plt.plot(t_values, iterate, ’--’, color=colors[i], linewidth=1.5,
                label=f’ _ {i}(t)’)
    else:
        plt.plot(t_values, iterate, ’--’, linewidth=1,
                label=f’ _ {i}(t)’)

plt.xlabel(’t’)
plt.ylabel(’u(t)’)
plt.title(’Picard Iteration for u\’ = u, u(0) = 1’)
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()

# Print final results
print(f"\nFinal iteration: n = {n}")
print(f"Final maximum error: {max_error:.6f}")
```



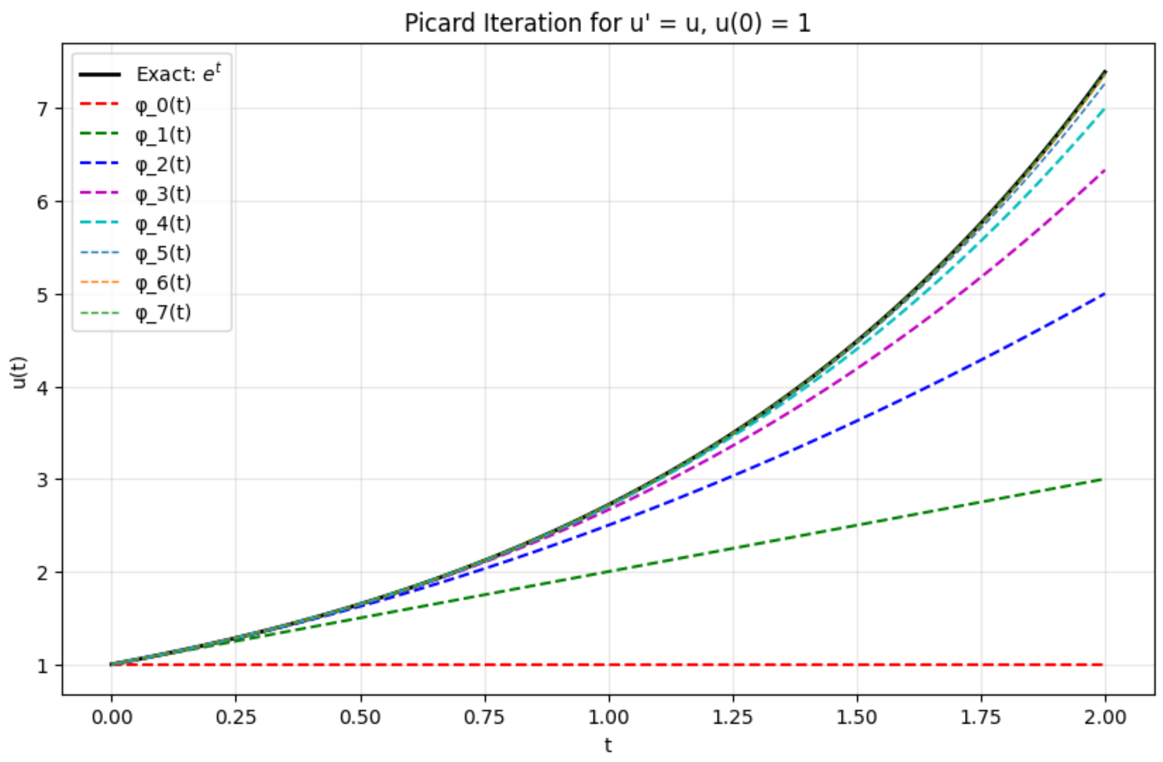
```
Picard Iterates for u' = u, u(0) = 1
==================================================
ϕ₀(t) = 1
ϕ_1(t) = t + 1
  Max error: 4.389056
ϕ_2(t) = t**2/2 + t + 1
  Max error: 2.389056
ϕ_3(t) = t**3/6 + t**2/2 + t + 1
  Max error: 1.055723
ϕ_4(t) = t**4/24 + t**3/6 + t**2/2 + t + 1
  Max error: 0.389056
ϕ_5(t) = t**5/120 + t**4/24 + t**3/6 + t**2/2 + t + 1
  Max error: 0.122389
ϕ_6(t) = t**6/720 + t**5/120 + t**4/24 + t**3/6 + t**2/2 + t + 1
  Max error: 0.033501
ϕ_7(t) = t**7/5040 + t**6/720 + t**5/120 + t**4/24 + t**3/6 + t**2/2 + t + 1
  Max error: 0.008104

✓ Tolerance 0.01 achieved at iteration n = 7
  Maximum error: 0.008104
```

5 Python Implementation Comments

Package Imports

The implementation utilizes several essential Python packages to achieve both symbolic and numerical computation. The NumPy library provides fundamental numerical calculations and efficient array operations, forming the computational backbone for handling discrete data points. For visualization purposes, matplotlib.pyplot enables comprehensive plotting capabilities to graphically represent the convergence behavior of the iterative approximations. From the SymPy symbolic mathematics library, specific functions are imported: the `symbols` function converts string representations into symbolic mathematical variables to facilitate symbolic manipulation; the `integrate` function performs both antiderivative and definite integral calculations using established calculus rules; the `exp` function provides the symbolic representation of the exponential function e^x; and finally, the `lambdify` function serves as a bridge between symbolic expressions and fast numerical functions by converting symbolic formulas into executable code for efficient numerical evaluation.



Function Parameters

Three key parameters govern the algorithm's execution and output. The tolerance parameter, `tol`, determines the convergence criterion; the algorithm terminates when the maximum absolute difference between the current approximation and the exact solution falls below this threshold value. The time interval `t_range` defines the domain over which the solution is calculated and subsequently visualized, typically specified as a tuple containing the start and end times. The parameter `num_points` controls the number of discrete evaluation points within the specified time range, directly influencing the smoothness and resolution of the resulting plots; a higher number of points yields smoother curves but increases computational expense.

Symbolic Variables

The symbolic computation is built upon two fundamental variables. The primary independent variable `t` represents time within the differential equation and serves as the variable for which the solution is constructed. A dummy variable `s` is introduced specifically for integration purposes, acting as the variable of integration within the definite integrals computed during each Picard iteration step, thereby maintaining proper mathematical notation and avoiding confusion with the time variable.

Initialization

The algorithm begins by initializing several critical components. The zeroth iterate $\varphi_0(t)$, stored as `u_prev`, is set to the constant value of 1, representing the initial guess for the iterative process as derived from the initial condition. An array `t_values` is created containing a sequence of evenly spaced time points spanning the specified interval, providing the discrete points at which all approximations will be numerically evaluated. The exact solution e^t is pre-calculated at all these time points and stored as `exact`, establishing the reference values for error computation throughout the iteration process. An iteration counter `n` is initialized to zero to track the number of completed iterations. Finally, the maximum error variable `max_error` is initialized to positive infinity, ensuring that the while loop condition is satisfied upon first evaluation and guaranteeing the loop's execution.

Picard Iteration Loop

The core computational process occurs within a while loop that continues until the maximum error satisfies the convergence criterion. With each iteration, the counter `n` is incremented to reflect the current iteration number. The integrand for the current step is prepared by substituting the integration variable `s` for the time variable `t` in the previous approximation `u_prev`, resulting in the expression $\varphi_{n-1}(s)$. This expression is then symbolically integrated with respect to `s` from the lower limit 0 to the upper limit `t`, computing the definite integral $\int_0^t \varphi_{n-1}(s)ds$. The resulting integral is added to the initial value 1 to form the new approximation according to the Picard iteration formula, yielding $\varphi_n(t) = 1 + \int_0^t \varphi_{n-1}(s)ds$, which is stored as `u_current`.

Numerical Evaluation

Following the symbolic computation of each new approximation, numerical evaluation is performed to enable error calculation and visualization. If the current approximation `u_current` remains a symbolic expression, the `lambdify` function is employed to convert this symbolic representation into a fast numerical function that can be efficiently evaluated across the array of time points. In the case where the approximation simplifies to a numeric constant, a constant array is created instead. The numerical values of the current approximation at all discrete time points are computed and stored as `current_values`, providing the data necessary for subsequent error analysis and graphical representation.

Error Calculation

The convergence metric is computed by evaluating the maximum absolute difference between the current numerical approximation and the precomputed exact solution values across the entire time domain. Mathematically, this error is defined as $\max |\varphi_n(t_i) - e^{t_i}|$ for all discrete time points t_i contained within the `t_values` array. This maximum absolute error serves as the primary termination criterion for the iterative process, quantitatively measuring how closely the current approximation matches the true solution over the specified interval.

Convergence Check

The algorithm concludes each iteration by assessing whether the convergence criterion has been met. If the computed maximum error is less than or equal to the specified tolerance, the while loop terminates, indicating that the desired accuracy has been achieved and the final approximation is deemed sufficient. Otherwise, the current approximation `u_current` is assigned to `u_prev`, thereby updating the state for the subsequent iteration, and the process repeats with this new approximation serving as the basis for the next integral computation.