```python
from scipy.integrate import ode

# Demo of "scitools pyreport", a modification of the original
# pyreport code to work with both matplotlib and scitools.easyviz.

# Comment lines starting with #! employ reStructuredText syntax,
# which has primitive math support but works well in HTML,
# while comment lines starting with #
```

Demonstrations

Run with `-l -e` options to turn on LaTeX support. `-t pdf` gives PDF output while `-t html` specifies HTML output. The name stem of the output file is specified by the `-o` option.

```python
"""
scitools pyreport -l -e -t pdf  -o report_mpl -a '0.05 9' demo_pyreport.py
scitools pyreport -l -e -t html -o report_mpl -a '0.05 9' demo_pyreport.py
scitools pyreport -l -e -t pdf  -o report_evz -a '0.05 9 easyviz' demo_pyreport.py
scitools pyreport -l -e -t html -o report_evz -a '0.05 9 easyviz' demo_pyreport.py
"""
```

Solving the logistic equation

The (scaled) logistic equation reads

$$u'(t) = u(1-u),$$

for $t \in (0,T]$ and with initial condition $u(0) = \alpha$.

```python

# LaTeX version:
```

The (scaled) logistic equation reads

$$u'(t) = u(1-u), \quad t \in (0,T]$$

with initial condition $u(0) = \alpha$.

```python


def f(t, u):
    return u*(1-u)  # logistic equation


try:
    alpha = float(sys.argv[1])
except IndexError:
    alpha = 0.1  # default


try:
    T = float(sys.argv[2])
except IndexError:
    T = 8
```

We solve the logistic equation using the `dopri5` solver in `scipy.integrate.ode`. This is a Dormand-Prince adaptive Runge-Kutta method of order 4-5.

```python

solver = ode(f)
solver.set_integrator('dopri5', atol=1E-6, rtol=1E-4)
solver.set_initial_value(alpha, 0)
```

The solution is computed at equally spaced time steps $t_i = i\Delta t$, with $\Delta t = 0.2$, $i = 1, 2, \ldots$. The integration between t_i and t_{i+1} applies smaller, adaptive time steps, adjusted to meet the prescribed tolerances of the solution.

```

# LaTeX version:
```

The solution is computed at equally spaced time steps $t_i=iDelta t$, with $Delta t=0.2$, $i=1,2,3,ldots$. The integration between t_i and t_{i+1} applies smaller, adaptive time steps, adjusted to meet the prescribed tolerances of the solution.

```

dt = 0.2          # time step

u = [];  t = []   # store solution and times

while solver.successful() and solver.t < T:
    solver.integrate(solver.t + dt)
    # current time is in solver.t, current solution in solver.y
    u.append(solver.y);  t.append(solver.t)

# Demonstrate plotting with matplotlib or scitools.easyviz

plot = 'matplotlib'
try:
    if sys.argv[3] == 'easyviz':
        plot = 'easyviz'
except IndexError:
    pass

if plot == 'matplotlib':
    import matplotlib.pyplot as plt
else:
    import scitools.std as plt

plt.plot(t, u)
plt.xlabel('t')
plt.ylabel('u')
plt.axis([t[0], t[-1], 0, 1.1])
plt.show()
```