Answers: Exercise Sheet No. 6
ODE Pt. II.

1 Basic theory and manipulations

2 asod

```python
# -*- coding: utf-8 -*-
"""
Created on Thu Feb 17 12:23:21 2022

@author: P. Maxwell
"""


import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp




def f_lorenz(t, w, sigma, beta, rho):
    """Lorenz differential equations"""
    x = w[0]
    y = w[1]
    z = w[2]
    return np.array([ sigma*(y-x), x*(rho-z) - y, x*y - beta*z ],
    dtype=np.float64 )



# Parameters in order sigma, beta, rho.
param_args = (10.0, 8/3, 28)

# Initial point
x0 = np.array([5.0, 5.0, 5.0])

# Time interval to integrate over
tspan = [0, 60]

# Set tolerances
reltol = 1e-12
abstol = 1e-14




def do_solve(f, tspan, x0, abstol, reltol, param_args):
    """Use SciPy to solve system explicitly with 8th order RK"""
    sol = solve_ivp(f, tspan, x0,
                            args=param_args,
                            method='DOP853', dense_output=True,
                            rtol=reltol, atol=abstol)
    return sol


```

```python



def plot_lorenz(sol_lorenz, t_pts):
    # Plot using a 2x2 subplot figure, with t-x, t-y, t-z, and 3d x-y-z.
    fig = plt.figure(figsize=(30, 30))
    ax1 = fig.add_subplot(221)
    ax1.plot(t_pts, sol_lorenz.sol(t_pts)[0,:], 'b-', linewidth=1.0)
    ax1.tick_params(axis='both', labelsize=22)
    ax1.set_xlabel(r'$t$', fontsize=28)
    ax1.set_ylabel(r'$x$', fontsize=28)

    ax2 = fig.add_subplot(222)
    ax2.plot(t_pts, sol_lorenz.sol(t_pts)[1,:], 'b-', linewidth=1.0)
    ax2.tick_params(axis='both', labelsize=22)
    ax2.set_xlabel(r'$t$', fontsize=28)
    ax2.set_ylabel(r'$y$', fontsize=28)

    ax3 = fig.add_subplot(223)
    ax3.plot(t_pts, sol_lorenz.sol(t_pts)[2,:], 'b-', linewidth=1.0)
    ax3.tick_params(axis='both', labelsize=22)
    ax3.set_xlabel(r'$t$', fontsize=28)
    ax3.set_ylabel(r'$z$', fontsize=28)

    ax4 = fig.add_subplot(224, projection='3d')
    ax4.plot3D(sol_lorenz.sol(t_pts)[0,:], sol_lorenz.sol(t_pts)[1,:], sol_lorenz.sol(t_pts)[2,:], 'b-', linewidth=1.0)
    ax4.tick_params(axis='both', labelsize=22)





sol_lorenz = do_solve(f_lorenz, tspan, x0, abstol, reltol, param_args)

t_pts = np.linspace(tspan[0], tspan[1], 10000)
plot_lorenz(sol_lorenz, t_pts)
```

```python
    # -*- coding: utf-8 -*-
"""
Created on Mon Feb 21 03:41:46 2022

@author: P. Maxwell
"""

import autograd.numpy as np
from autograd import jacobian
from autograd import grad
import matplotlib as mpl
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp




# Parameters in order sigma, beta, rho.
param_args = ()

```

```python
# Initial point
x0 = np.array([1.0, 0.0, 0.0])

# Time interval to integrate over
tspan = np.array([0.0, 10.0])

# Set tolerances
reltol = 1e-4
abstol = 1e-7




def f_robertson(t, w):
    """Simplified Brusselator  differential equations"""
    y1 = w[0]
    y2 = w[1]
    y3 = w[2]
    return np.array([ -0.04*y1 + 1e4*y2*y3,
                     0.04*y1 - 1e4*y2*y3 - 3e7*y2**2,
                     3e7*y2**2 ], dtype=np.float64 )



# Jacobian (used for the Newton iteration of an implicit method
    like RADAU)
jac_robertson_autograd = jacobian(f_robertson, 1)
def jac_robertson(t, x):
    return jac_robertson_autograd(t, x)




def do_solve_explicit(f, tspan, x0, abstol, reltol, param_args):
    """Use SciPy to solve system explicitly with RK45"""
    sol = solve_ivp(f, tspan, x0,
                            args=param_args,
                            method='RK45', dense_output=True,
                            rtol=reltol, atol=abstol)
    return sol




def do_solve_implicit(f, tspan, x0, abstol, reltol, param_args, jac
    ):
    """Use SciPy to solve system explicitly with Radau"""
    sol = solve_ivp(f, tspan, x0,
                            args=param_args,
                            method='Radau', dense_output=True,
                            jac=jac,
                            rtol=reltol, atol=abstol)
    return sol




def plot_robertson(sol_robertson_explicit, sol_robertson_implicit,
    t_pts):
    fig = plt.figure(figsize=(40, 40))
    ax1 = fig.add_subplot(321)
    ax1.semilogx(t_pts, sol_robertson_explicit.sol(t_pts)[0,:], 'b
```

```
-', linewidth=3)
    ax1.semilogx(sol_robertson_explicit.t, sol_robertson_explicit.y[0,:], 'ro', linewidth=3, markersize=15)
    ax1.grid()
    ax1.tick_params(axis='both', labelsize=22)
    ax1.set_title(r'Robertson $y_1$ (explicit)', fontsize=28)
    ax1.set_xlabel(r'$t$', fontsize=28)
    ax1.set_ylabel(r'$y_1$', fontsize=28)


    ax2 = fig.add_subplot(323)
    ax2.semilogx(t_pts, sol_robertson_explicit.sol(t_pts)[1,:], 'b-', linewidth=3)
    ax2.semilogx(sol_robertson_explicit.t, sol_robertson_explicit.y[1,:], 'ro', linewidth=3, markersize=15)
    ax2.grid()
    ax2.tick_params(axis='both', labelsize=22)
    ax2.set_title(r'Robertson $y_2$ (explicit)', fontsize=28)
    ax2.set_xlabel(r'$t$', fontsize=28)
    ax2.set_ylabel(r'$y_2$', fontsize=28)

    ax3 = fig.add_subplot(325)
    ax3.semilogx(t_pts, sol_robertson_explicit.sol(t_pts)[2,:], 'b-', linewidth=3)
    ax3.semilogx(sol_robertson_explicit.t, sol_robertson_explicit.y[2,:], 'ro', linewidth=3, markersize=15)
    ax3.grid()
    ax3.tick_params(axis='both', labelsize=22)
    ax3.set_title(r'Robertson $y_3$ (explicit)', fontsize=28)
    ax3.set_xlabel(r'$t$', fontsize=28)
    ax3.set_ylabel(r'$y_3$', fontsize=28)


    ax4 = fig.add_subplot(322)
    ax4.semilogx(t_pts, sol_robertson_implicit.sol(t_pts)[0,:], 'b-', linewidth=3)
    ax4.semilogx(sol_robertson_implicit.t, sol_robertson_implicit.y[0,:], 'ro', linewidth=3, markersize=15)
    ax4.grid()
    ax4.tick_params(axis='both', labelsize=22)
    ax4.set_title(r'Robertson $y_1$ (implicit)', fontsize=28)
    ax4.set_xlabel(r'$t$', fontsize=28)
    ax4.set_ylabel(r'$y_1$', fontsize=28)

    ax5 = fig.add_subplot(324)
    ax5.semilogx(t_pts, sol_robertson_implicit.sol(t_pts)[1,:], 'b-', linewidth=3)
    ax5.semilogx(sol_robertson_implicit.t, sol_robertson_implicit.y[1,:], 'ro', linewidth=3, markersize=15)
    ax5.grid()
    ax5.tick_params(axis='both', labelsize=22)
    ax5.set_title(r'Robertson $y_2$ (implicit)', fontsize=28)
    ax5.set_xlabel(r'$t$', fontsize=28)
    ax5.set_ylabel(r'$y_2$', fontsize=28)

    ax6 = fig.add_subplot(326)
    ax6.semilogx(t_pts, sol_robertson_implicit.sol(t_pts)[2,:], 'b-', linewidth=3)
    ax6.semilogx(sol_robertson_implicit.t, sol_robertson_implicit.y[2,:], 'ro', linewidth=3, markersize=15)
    ax6.grid()
    ax6.tick_params(axis='both', labelsize=22)
```

```
    ax6.set_title(r'Robertson $y_3$ (implicit)', fontsize=28)
    ax6.set_xlabel(r'$t$', fontsize=28)
    ax6.set_ylabel(r'$y_3$', fontsize=28)




sol_robertson_explicit = do_solve_explicit(f_robertson, tspan, x0, abstol, reltol, param_args)
sol_robertson_implicit = do_solve_implicit(f_robertson, tspan, x0, abstol, reltol, param_args, jac_robertson)


# t_pts = np.linspace(tspan[0], tspan[1], 1000)
t_pts = 10**np.linspace(-6, np.log10(tspan[1]), 1000)
plot_robertson(sol_robertson_explicit, sol_robertson_implicit, t_pts)
```

```
    # -*- coding: utf-8 -*-
"""
Created on Mon Feb 21 03:19:20 2022

@author: P. Maxwell
"""

import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp




# Parameters in order sigma, beta, rho.
param_args = (1.0, 3.0)

# Initial point
x0 = np.array([1.0, 3.08])

# Time interval to integrate over
tspan = [0.0, 15.0]

# Set tolerances
reltol = 1e-6
abstol = 1e-9




def f_brusselator_simple(t, w, A, B):
    """Simplified Brusselator  differential equations"""
    y1 = w[0]
    y2 = w[1]
    return np.array([ A + y2*y1**2 -(B+1)*y1, B*y1 -y2*y1**2  ], dtype=np.float64 )




def do_solve(f, tspan, x0, abstol, reltol, param_args):
    """Use SciPy to solve system explicitly with RK45"""
    sol = solve_ivp(f, tspan, x0,
                        args=param_args,
```

```
                            method='RK45', dense_output=True,
                            rtol=reltol, atol=abstol)
    return sol



def plot_brusselator_simple(sol_brusselator_simple, t_pts):
    fig, ax = plt.subplots(1, 1, figsize=(20, 20))
    ax.plot(sol_brusselator_simple.sol(t_pts)[0,:],
    sol_brusselator_simple.sol(t_pts)[1,:], linewidth=3)
    ax.grid()
    ax.tick_params(axis='both', labelsize=22)
    ax.set_title(r'Brusselator Simple', fontsize=28)
    ax.set_xlabel(r'$y_1$', fontsize=28)
    ax.set_ylabel(r'$y_2$', fontsize=28)



sol_brusselator_simple = do_solve(f_brusselator_simple, tspan, x0,
    abstol, reltol, param_args)


t_pts = np.linspace(tspan[0], tspan[1], 1000)
plot_brusselator_simple(sol_brusselator_simple, t_pts)
```

```
    # -*- coding: utf-8 -*-
"""
Created on Sun Feb 20 21:23:11 2022

@author: P. Maxwell
"""


import autograd.numpy as np
from autograd import jacobian
from autograd import grad
import matplotlib as mpl
import matplotlib.pyplot as plt
#import scipy.optimize as spO
from scipy.integrate import solve_ivp


# Remember to note that this is a continuous reformulation of a
    discrete process




N = 1000000
Ethres = N * 0.2
R0 = 4.0
gamma = 1 / 5
beta_sir = R0 * gamma
beta_seirsd = R0 * (gamma + alpha1)
alpha1 = 0.015
# alpha2 = 0.03
alpha2 = 0.03
sigma = 1/5  # Original Covid
omega = 1/180


initial_seed = 10.0
w0_sir = np.array([N-initial_seed, initial_seed, 0.0])
```

```
w0_seirsd = np.array([N-initial_seed, initial_seed, 0.0, 0.0, 0.0])



param_args_sir = np.array([N, beta_sir, gamma])
param_args_seirsd = np.array([N, alpha1, alpha2, beta_seirsd, gamma, sigma, omega, Ethres])
tspan = (0.0, 400.0)
reltol = 1e-4
abstol = 1e-7


def f_sir(t, w, N, beta, gamma):
    """Calculate derivatives in SIR model"""
    S = w[0]
    I = w[1]
    R = w[2]

    dwdt = np.zeros((3,), dtype=np.float64)
    # dS/dt
    dwdt[0] = -beta*I*S/N
    # dI/dt
    dwdt[1] = beta*I*S/N -gamma*I
    # dR/dt
    dwdt[2] = gamma*I


    if abs((S+I+R-N)) > 1.0:
        print("Consistency error!  S, I, R, N, diff:", S, I, R, N, abs((S+I+R-N)))

    return dwdt







def f_seirsd(t, w, N, alpha1, alpha2, beta, gamma, sigma, omega, Ethres):
    """Calculate derivatives in simple SEIRSD model."""
    S = w[0]
    E = w[1]
    I = w[2]
    R = w[3]
    D = w[4]
    alpha1_cntr = np.max(np.array([0.0, (alpha1*N - D)/N])) * I
    alpha2_cntr = np.max(np.array([0.0, (alpha2*N - D)/N])) * np.max(np.array([0.0, I-Ethres]))
    dwdt = np.zeros((5,), dtype=np.float64)
    # dS/dt
    dwdt[0] = -beta*I*S/N + omega*R
    # dE/dt
    dwdt[1] = beta*I*S/N -sigma*E
    # dI/dt
    dwdt[2] = sigma*E -gamma*I -alpha1_cntr -alpha2_cntr
    # dR/dt
    dwdt[3] = gamma*I -omega*R
    # dD/dt
    dwdt[4] = alpha1_cntr + alpha2_cntr

```

```python
    return dwdt






def do_solve_cm(f, tspan, w0, abstol, reltol, param_args):
    # Do the integration using explicit 8th order RK
    sol_cm = solve_ivp(f, tspan, w0,
                            args=param_args,
                            method='RK45', dense_output=True,
                            rtol=reltol, atol=abstol)
    return sol_cm








def plot_solution_sir(sol_sir, end_day):
    t_pts = np.float64(np.arange(0, end_day+1))
    fig, ax = plt.subplots(1, 1, figsize=(30, 15))
    ax.plot(t_pts, sol_sir.sol(t_pts)[0, :], label='Susceptible', linewidth=3)
    ax.plot(t_pts, sol_sir.sol(t_pts)[1, :], label='Infectious', linewidth=3)
    ax.plot(t_pts, sol_sir.sol(t_pts)[2, :], label='Recovered', linewidth=3)
    ax.grid()
    ax.tick_params(axis='both', labelsize=22)
    ax.legend(fontsize=28)
    ax.set_title(r'SIR model output', fontsize=28)
    ax.set_xlabel(r'$t$ (days)', fontsize=28)
    ax.set_ylabel(r'Individuals (millions)', fontsize=28)






def plot_solution_seirsd(sol_seirsd, end_day):
    t_pts = np.float64(np.arange(0, end_day+1))
    fig, ax = plt.subplots(1, 1, figsize=(30, 15))
    ax.plot(t_pts, sol_seirsd.sol(t_pts)[0, :], label='Susceptible', linewidth=3)
    ax.plot(t_pts, sol_seirsd.sol(t_pts)[1, :], label='Exposed', linewidth=3)
    ax.plot(t_pts, sol_seirsd.sol(t_pts)[2, :], label='Infectious', linewidth=3)
    ax.plot(t_pts, sol_seirsd.sol(t_pts)[3, :], label='Recovered', linewidth=3)
    ax.plot(t_pts, sol_seirsd.sol(t_pts)[4, :], label='Dead', linewidth=3)
    ax.grid()
    ax.tick_params(axis='both', labelsize=22)
    ax.legend(fontsize=28)
    ax.set_title(r'SEIRSD model output', fontsize=28)
    ax.set_xlabel(r'$t$ (days)', fontsize=28)
    ax.set_ylabel(r'Individuals (millions)', fontsize=28)
```

```

# sol_sir = do_solve_cm(f_sir, tspan, w0_sir, abstol, reltol,
    param_args_sir)
# plot_solution_sir(sol_sir, np.int_(tspan[1]))


sol_seirsd = do_solve_cm(f_seirsd, tspan, w0_seirsd, abstol, reltol
    , param_args_seirsd)
plot_solution_seirsd(sol_seirsd, np.int_(tspan[1]))
```