serialNetabc

August 21, 2023

[1]:
```
# Import packages
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
```

[2]:
```
N = 1
I0, R0 = 0.01*N, 0
S0 = N - I0 - R0
beta, gamma = 0.2, 1./10
t = np.linspace(0, 1600, 1600)
```

[3]:
```
def deriv(y, t, N, beta, gamma) :
    S, I, R = y
    dSdt = -beta * S* I / N
    dIdt = beta * S * I / N - gamma * I
    dRdt = gamma * I
    return dSdt, dIdt, dRdt
```

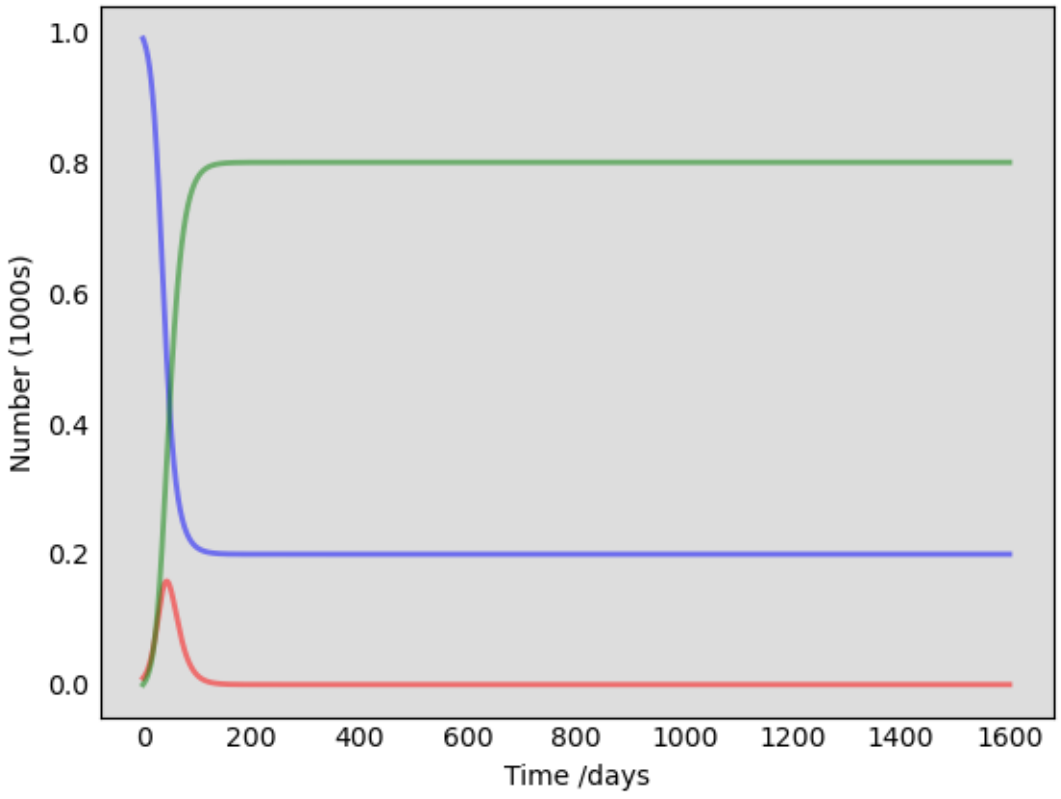
[4]:
```
y0 = S0, I0, R0
ret = odeint(deriv, y0, t, args=(N, beta, gamma))
S, I, R = ret.T
```

[5]:
```
fig = plt.figure(facecolor='w')
ax = fig.add_subplot(111, facecolor='#dddddd', axisbelow=True)
ax.plot(t, S, 'b', alpha=0.5, lw=2, label='Susceptible')
ax.plot(t, I, 'r', alpha=0.5, lw=2, label='Infected')
ax.plot(t, R, 'g', alpha=0.5, lw=2, label='Recovered with immunity')
ax.set_xlabel('Time /days' )
ax.set_ylabel('Number (1000s)')
#ax.set_ylim(0, 1.2)
ax.yaxis.set_tick_params(length=0)
ax.xaxis.set_tick_params(length=0)
ax.grid(b=True, which='major', c='w', lw=2, ls='-')
legend = ax.legend()
legend.get_frame().set_alpha(0.5)
for spine in ('top', 'right', 'bottom', 'left') :
    ax.spines[spine].set_visible(False)
```

```
    plt.show()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[5], line 11
      9 ax.yaxis.set_tick_params(length=0)
     10 ax.xaxis.set_tick_params(length=0)
---> 11 ax.grid(b=True, which='major', c='w', lw=2, ls='-')
     12 legend = ax.legend()
     13 legend.get_frame().set_alpha(0.5)

File /opt/conda/envs/anaconda-panel-2023.05-py310/lib/python3.11/site-packages/
 ↪matplotlib/axes/_base.py:3194, in _AxesBase.grid(self, visible, which, axis,␣
 ↪**kwargs)
   3192 _api.check_in_list(['x', 'y', 'both'], axis=axis)
   3193 if axis in ['x', 'both']:
-> 3194     self.xaxis.grid(visible, which=which, **kwargs)
   3195 if axis in ['y', 'both']:
   3196     self.yaxis.grid(visible, which=which, **kwargs)

File /opt/conda/envs/anaconda-panel-2023.05-py310/lib/python3.11/site-packages/
 ↪matplotlib/axis.py:1660, in Axis.grid(self, visible, which, **kwargs)
   1657 if which in ['major', 'both']:
   1658     gridkw['gridOn'] = (not self._major_tick_kw['gridOn']
   1659                         if visible is None else visible)
-> 1660     self.set_tick_params(which='major', **gridkw)
   1661 self.stale = True

File /opt/conda/envs/anaconda-panel-2023.05-py310/lib/python3.11/site-packages/
 ↪matplotlib/axis.py:932, in Axis.set_tick_params(self, which, reset, **kwargs)
    919 """
    920 Set appearance parameters for ticks, ticklabels, and gridlines.
    921
   (…)
    929     gridlines.
    930 """
    931 _api.check_in_list(['major', 'minor', 'both'], which=which)
--> 932 kwtrans = self._translate_tick_params(kwargs)
    934 # the kwargs are stored in self._major/minor_tick_kw so that any
    935 # future new ticks will automatically get them
    936 if reset:

File /opt/conda/envs/anaconda-panel-2023.05-py310/lib/python3.11/site-packages/
 ↪matplotlib/axis.py:1076, in Axis._translate_tick_params(kw, reverse)
   1074 for key in kw_:
   1075     if key not in allowed_keys:
-> 1076         raise ValueError(
```

```
   1077             "keyword %s is not recognized; valid keywords are %s"
   1078             % (key, allowed_keys))
   1079 kwtrans.update(kw_)
   1080 return kwtrans

ValueError: keyword grid_b is not recognized; valid keywords are ['size',␣
 ↪'width', 'color', 'tickdir', 'pad', 'labelsize', 'labelcolor', 'zorder',␣
 ↪'gridOn', 'tick1On', 'tick2On', 'label1On', 'label2On', 'length', 'direction' ␣
 ↪'left', 'bottom', 'right', 'top', 'labelleft', 'labelbottom', 'labelright',␣
 ↪'labeltop', 'labelrotation', 'grid_agg_filter', 'grid_alpha', 'grid_animated' ␣
 ↪'grid_antialiased', 'grid_clip_box', 'grid_clip_on', 'grid_clip_path',␣
 ↪'grid_color', 'grid_dash_capstyle', 'grid_dash_joinstyle', 'grid_dashes',␣
 ↪'grid_data', 'grid_drawstyle', 'grid_figure', 'grid_fillstyle',␣
 ↪'grid_gapcolor', 'grid_gid', 'grid_in_layout', 'grid_label', 'grid_linestyle' ␣
 ↪'grid_linewidth', 'grid_marker', 'grid_markeredgecolor',␣
 ↪'grid_markeredgewidth', 'grid_markerfacecolor', 'grid_markerfacecoloralt',␣
 ↪'grid_markersize', 'grid_markevery', 'grid_mouseover', 'grid_path_effects',␣
 ↪'grid_picker', 'grid_pickradius', 'grid_rasterized', 'grid_sketch_params',␣
 ↪'grid_snap', 'grid_solid_capstyle', 'grid_solid_joinstyle', 'grid_transform',
 ↪'grid_url', 'grid_visible', 'grid_xdata', 'grid_ydata', 'grid_zorder',␣
 ↪'grid_aa', 'grid_c', 'grid_ds', 'grid_ls', 'grid_lw', 'grid_mec', 'grid_mew',
 ↪'grid_mfc', 'grid_mfcalt', 'grid_ms']
```



[6]:
```
R[-1]
```

[6]: 0.8002039459242528

[7]:
```
def calculate_R0(beta, gamma):
    R0 = beta / gamma
    return R0

# Given values
beta = 0.5   # Replace with the actual value of beta
gamma = 0.2  # Replace with the actual value of gamma

# Calculate R0
R0 = calculate_R0(beta, gamma)
print("R0:", R0)
```

```
R0: 2.5
```

[8]:
```
def calculate_S_0(s_p_n, I_0):
    S_0 = 1 - s_p_n - I_0
    return S_0

# Given values
s_p_n = 0.2   # Replace with the actual value of s_p_n
I_0 = 0.01    # Replace with the actual value of I(0)

# Calculate S(0)
S_0 = calculate_S_0(s_p_n, I_0)
print("S(0):", S_0)
```

```
S(0): 0.79
```

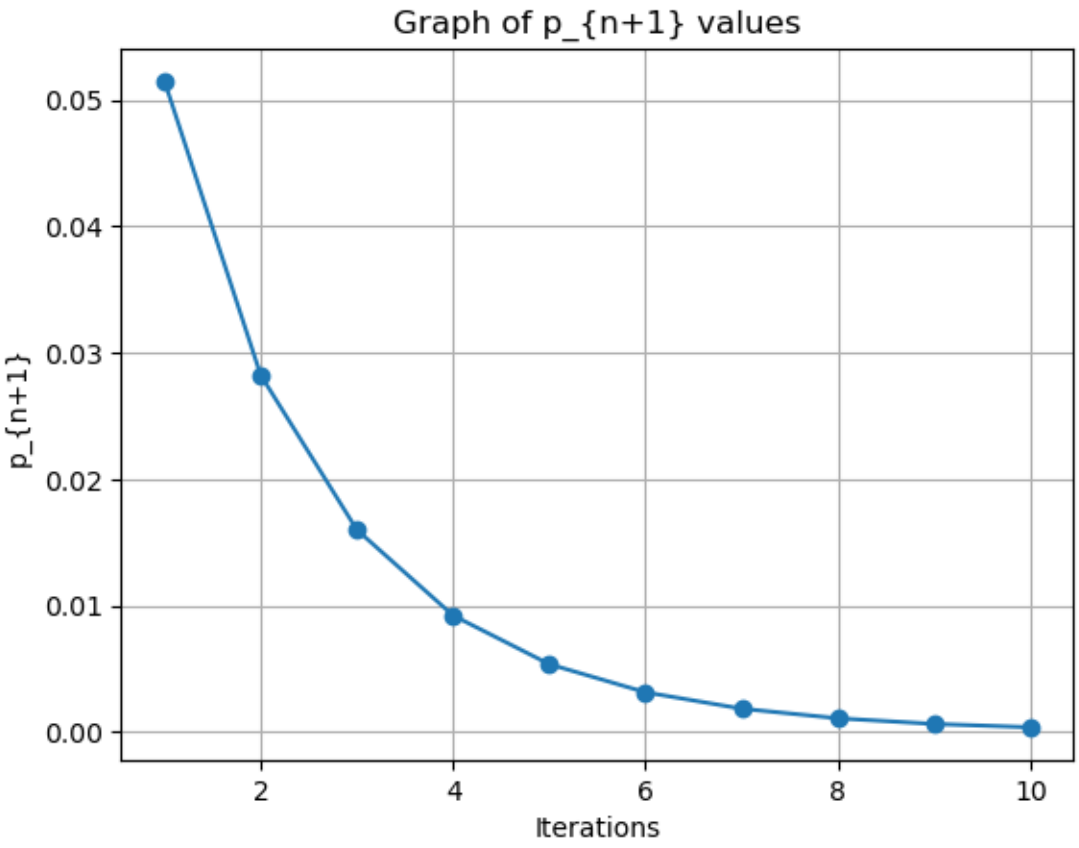
[9]:
```
def calculate_p_crit(R0, s):
    p_crit = 1 / s * (1 - 1 / R0)
    return p_crit

# Given values
R0 = 2.0  # Replace with the actual value of R0
s = 0.5   # Replace with the actual value of s

# Calculate p_crit
p_crit = calculate_p_crit(R0, s)
print("p_crit:", p_crit)
```

```
p_crit: 1.0
```

[10]:
```
import math
import matplotlib.pyplot as plt
def calculate_p_next(p_n, s, r, R0):
    phi = (1 - p_n) * (1 - math.exp(-R0 * p_n))
    p_next = phi * (1 - s) + (1 - phi) * (1 - r) * p_n
    return p_next
```

```
# Given values
R0 = 1.4
r = 0.55
s = 0.9
p_n = 0.1  # Initial value

# Calculate p_{n+1}
p_next_values = []
iterations = 10  # Number of iterations for the graph
for _ in range(iterations):
    p_next = calculate_p_next(p_n, s, r, R0)
    p_next_values.append(p_next)
    p_n = p_next



# Print the calculated values
for i, p in enumerate(p_next_values):
    print(f"p_{i+1}: {p}")

# Plot the graph
plt.plot(range(1, iterations+1), p_next_values, marker='o')
plt.xlabel('Iterations')
plt.ylabel('p_{n+1}')
plt.title('Graph of p_{n+1} values')
plt.grid(True)
plt.show()
```

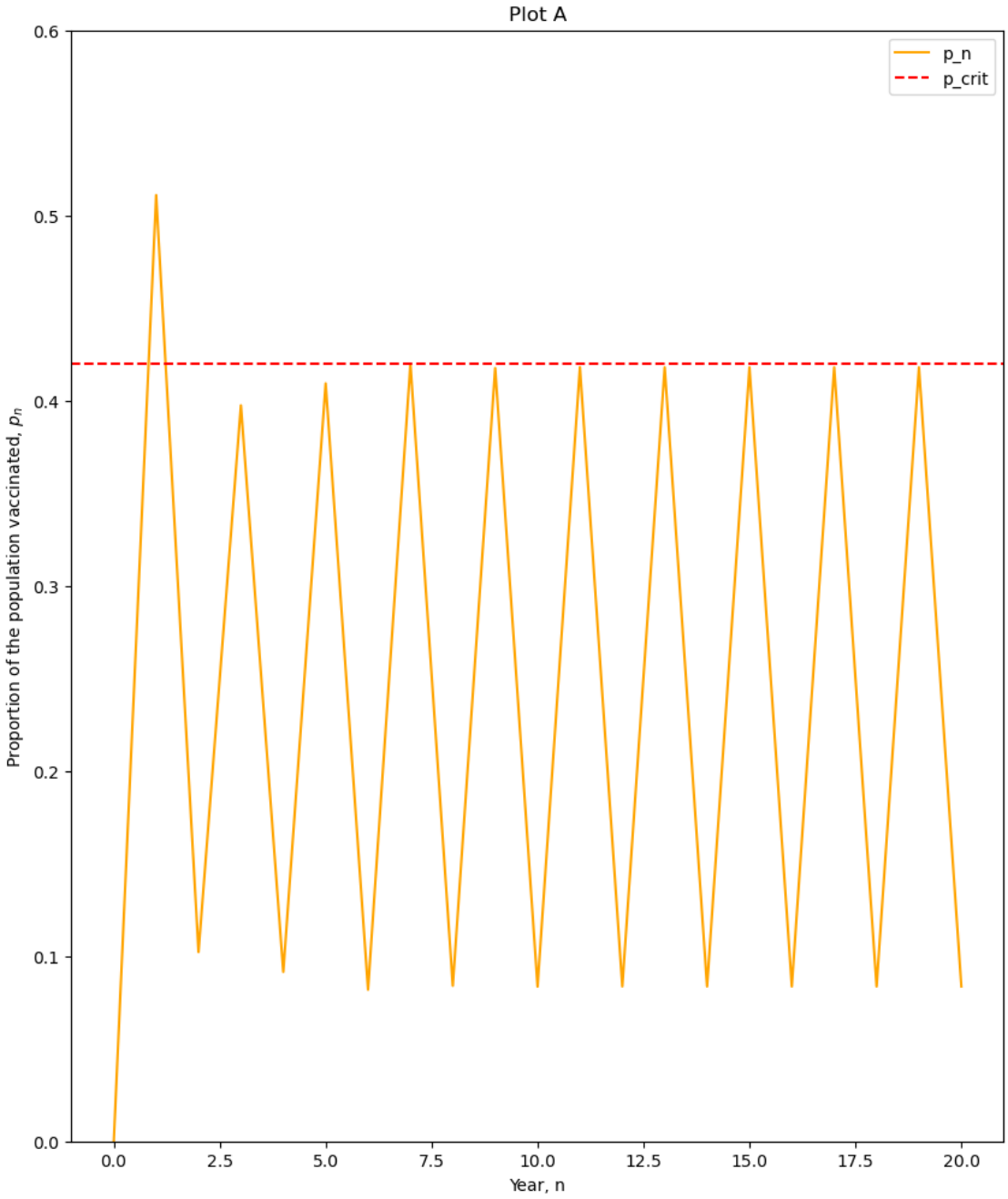
```
p_1: 0.0514667673477591
p_2: 0.02822694501071405
p_3: 0.01598918449342742
p_4: 0.009216630626377443
p_5: 0.005365024642663747
p_6: 0.003140567468802263
p_7: 0.001844411138878772
p_8: 0.0010852573775597706
p_9: 0.0006392810883043664
p_10: 0.0003768214248354778
```



[11]:
```
import numpy as np
from scipy.special import lambertw
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import patches

def phi(p, R0):
    return 1 - p + 1/R0 * lambertw(-(1-p)*R0*np.exp(-(1-p)*R0))

def p_crit(R0, s):
    return 1/s * (1 - 1/R0)

def f(p, R0, r, s):
    return (1-r)*s*p + (1-phi(s*p, R0)/(1-s*p))*(1-r)*(1-s)*p + (phi(s*p, R0)/
 ↪(1-s*p))*(1-p)

def iterate_f(R0, r, s, p0, N):
    p = np.zeros(N)
    p[0] = p0
```

```
    for i in range(1, N):
        p[i] = f(p[i-1], R0, r, s)

    return p


# Given values
R0 = 1.4
r = 0.8
s = 0.7
p0 = 0
N = 20
p_crit= 1
nudgep_crit = 0.02
adjusted_y = p_crit * (1 + nudgep_crit)


# Data for graph
n_values = np.arange(0, N+1)
p_n_values = iterate_f(R0, r, s, p0, N+1)  # Replace with actual function call


plt.figure(figsize=(10, 12))
plt.plot(n_values, p_n_values, color='orange', label='p_n')

# Sample values for demonstration
p_crit = 0.42  # Replace with your actual value

# Create a plot
#plt.plot([1, 2, 3], [0.4, 0.6, 0.8], color='orange', label='p_n')

# Draw a dashed horizontal line at p_crit
plt.axhline(y=p_crit, linestyle='dashed', color='red', label='p_crit')


plt.annotate(f'{p_crit}', xy=(2, adjusted_y), color='red')
plt.xlabel('Year, n')
plt.ylabel('Proportion of the population vaccinated, $p_n$')
plt.ylim(0,0.6)
plt.legend()
plt.title('Plot A')
plt.show()
```

```
/tmp/ipykernel_401/3885814496.py:21: ComplexWarning: Casting complex values to
real discards the imaginary part
  p[i] = f(p[i-1], R0, r, s)
```


Plot A
p_n
p_crit
Proportion of the population vaccinated, p_n
Year, n
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

[ ]:

[ ]:

[ ]: