Hopf Bifurcations: Modeling More Mathematically Complex Behavior to Stimulate a Healthy Natural Cycle

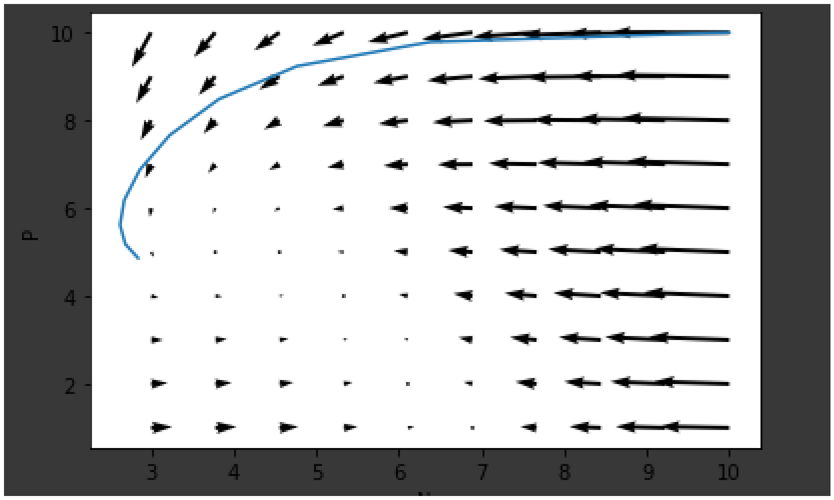
David Tran

August 2020

Problems

1. Simulating the given conditions with a vector field, I created the following models with the code:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint

N,P = np.meshgrid(np.linspace(3,10,10), np.linspace(1,10,10))
t = np.linspace(3,10,10)

r1 = 1
r2 = 0.1
k = 7
d = 1
j = 1
w = 0.4

def HTmodel(x,t):
  N = x[0]
  P = x[1]
  Np = r1 * N * (1 - (N/k)) - (w * N)/(d + N) * P
  Pp = r2 * P * (1 - (j * P) / N)
  z = [Np, Pp]
  return z

y0 = [10,10]
sol = odeint(HTmodel,y0,t)

Np = r1 * N * (1 - (N/k)) - (w * N)/(d + N) * P
Pp = r2 * P * (1 - (j * P) / N)

plt.quiver(N,P,Np,Pp)
plt.plot(sol[:,0],sol[:,1])
plt.xlabel("N")
plt.ylabel("P")
plt.show()
```



2. Using Python's **solve** function, I was able to obtain the system's equilibria points in terms of w. I obtained the following point along with multiple complicated equations:

$$(7.0, 0.0)$$

To accomplish this, I ran the following code:

```
import numpy as np
import matplotlib.pyplot as plt
from sympy.solvers import solve
from sympy import Symbol, init_printing
init_printing(use_unicode=True)

r1 = 1
r2 = 0.1
k = 7
d = 1
j = 1

N = Symbol('N')
P = Symbol('P')
w = Symbol('w')

Np = r1 * N * (1 - (N/k)) - (w * N)/(d + N) * P
Pp = r2 * P * (1 - (j * P) / N)

eqsoln = solve([Np,Pp],(N,P))
eqsoln
```

3. Pulling out the biologically relevant equilibrium point, I obtained the following result:

$$-3.5w + 0.5\sqrt{49.0w^2 - 84.0w + 64.0} + 3.0$$

To extract this, I wrote

```
bioEqN = eqsoln[1][0]
bioEqP = eqsoln[1][1]
bioEqN
bioEqP
```

4. Using the **jacobian** function, I determined the system's Jacobian to be:

$$\begin{bmatrix} \frac{NPw}{(N+1)^2} - \frac{2N}{7} - \frac{Pw}{N+1} + 1 & \frac{-Nw}{N+1} \\ \frac{0.1P^2}{N^2} & 0.1 - \frac{0.2P}{N} \end{bmatrix}$$

This was accomplished with the following code:

```
from sympy import Symbol, init_printing, Matrix
init_printing(use_unicode=True)

mat = Matrix([Np,Pp])
vars = Matrix([N,P])
jacb = mat.jacobian(vars)
jacb
```

5. The equilibrium point found from 2 was then replaced into the Jacobian and assigned to a variable.

```
eqn = jacb.subs([(N,bioEqN), (P,bioEqP)]);
eqn
```

6. The eigenvalues of this matrix were then found using

```
evals = eqn.eigenvals()
extract = list(evals.items())
eval1 = extract[0]
eval1
```

7. Substituting w with values of 0.4, 0.7, and 1.0, I obtained the following eigenvalues: $-0.249 - 0.073\sqrt{2}i$, $-0.0776 - 0.163\sqrt{2}i$, $0.0292 - 0.161\sqrt{2}i$. I ran the following code:

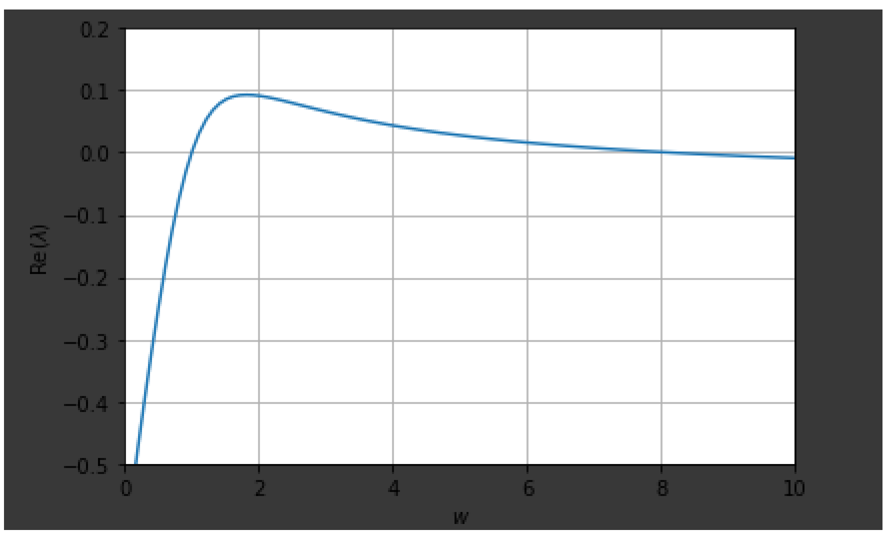
```
w = Symbol('w')

list([eval1.subs(w,0.4),eval1.subs(w,0.7),eval1.subs(w,1.0)])
```

8. In order to obtain the real part of the eigenvalues, the formula $\frac{1}{2}(a+d)$ will be utilized, where $a+d$ is the trace of the matrix. The code to obtain this is the following:

```
eqntrace = 0.5 * eqn.trace()
eqntrace
```

9. As a function of w, the graph of Re(λ) vs. w in the domains of [0,10] and [0.4,1.1] was created using the following code:

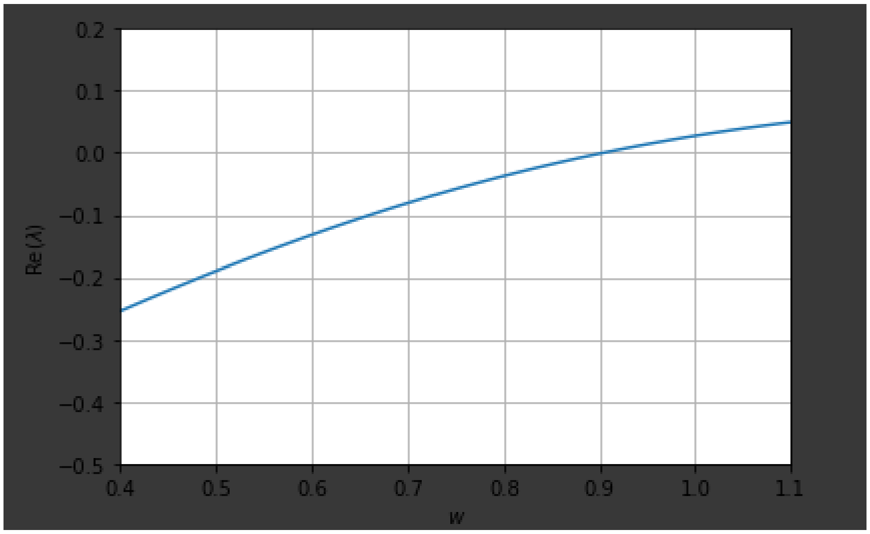
```
x = np.linspace(0,10,100)
arr = np.zeros(100)

for k in range(100):
    arr[k] = eqntrace.subs(w,(k-1)/9.9)
    #I used the formula x(k) = a + (k-1) * h, where h = (b-a)/(n
        -1). a and b are the endpoints of linspace and n is the 3
        rd argument of np.linspace

init_printing()
plt.xlabel('$w$')
plt.ylabel('Re($\lambda$)')
plt.xlim(0,10)
plt.ylim(-0.5,0.2)
plt.plot(x,arr)
plt.grid()
plt.show()
```



For [0.4,1.1], the code and graph are the following:

```
x = np.linspace(0.4,1.1,100)
arr = np.zeros(100)

for k in range(100):
    arr[k] = eqntrace.subs(w,0.4+(k-1)*(.7/(99)))

init_printing()
```

```
plt.xlabel('$w$')
plt.ylabel('Re($\lambda$)')
plt.xlim(0.4,1.1)
plt.ylim(-0.5,0.2)
plt.plot(x,arr)
plt.grid()
plt.show()
```



10. Upon first glance, it seems that the Hopf bifurcation occurs approximately at a value of $w = 0.9$. Using the **solve** function to obtain a more accurate value, the exact values are $w = 0.8968, 8.01$.

```
hopf = solve(eqntrace,w)
hopf
```