P1. Buscamos primero resolver la ecuación de Fisher-KPP, la cual está dada por:

$$\frac{\partial n}{\partial t} = \gamma \frac{\partial^2 n}{\partial x^2} + \mu n - \mu n^2$$

Esta es una ecuación de reacción-difusión que busca modelar el comportamiento de una especie animal, donde:

- $n(t, x)$: densidad de la especie como función de la posición y el tiempo.
- μn: tendencia de la especie a crecer indefinidamente, bajo la suposición de recursos infinitos.
- $-\mu n^2$: competencia por los recursos.
- $\frac{\partial^2 n}{\partial x^2}$: tendencia de la especie a dispersarse para encontrar más recursos.

Para resolver esta ecuación utilizaremos los métodos de Crank–Nicolson (para la parte de difusión) y de Euler Explicito (para la parte de reacción), por lo que necesitamos primero entender en qué consisten.

- Método de Crank-Nicolson:

El Método de Crank-Nicolson discretiza una Ecuación de Derivadas Parcialas, para la resolución de ecuaciones de difusión, que es lo que nos compete, vemos el ejemplo:

$$\frac{\partial u}{\partial t} = a \frac{\partial^2 u}{\partial x^2}$$

que es la ecuación de difusión lineal.

La discretización del método de Crank-Nicolson está dada por:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = \frac{a}{2(\Delta x)^2}\left(\left(u_{i+1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}\right) + \left(u_{i+1}^n - 2u_i^n + u_{i-1}^n\right)\right)$$

O, tomando $r = \frac{a\Delta t}{2(\Delta x)^2}$

$$-ru_{i+1}^{n+1} + (1+2r)u_i^{n+1} - ru_{i-1}^{n+1} = ru_{i+1}^n + (1-2r)u_i^n + ru_{i-1}^n$$

Que corresponde a un sistema con matriz tridiagonal.

Para nuestro problema, discretizaremos la función ∅, la cual representará la variable $n(t,x)$, utilizando j como el salto espacial y n el temporal. Para la derivada espacial utilizamos el método de Crank–Nicolson y para el resto de la ecuación el método de Euler Explicito. Con lo que nos queda:

$$\frac{\emptyset_{n+1}^{j}+\emptyset_{n}^{j}}{h}=\frac{\gamma}{2}\left(\frac{\emptyset_{n+1}^{j+1}-2\emptyset_{n+1}^{j}+\emptyset_{n+1}^{j-1}}{h^2}+\frac{\emptyset_{n}^{j+1}-2\emptyset_{n}^{j}+\emptyset_{n}^{j-1}}{h^2}\right)+\mu\emptyset_{n}^{j}-\emptyset_{n}^{j}\emptyset_{n}^{j}$$

Reordenando:

$$-r\emptyset_{n+1}^{j+1}+(1+2r)\emptyset_{n+1}^{j}-r\emptyset_{n+1}^{j-1}=r\left(\emptyset_{n}^{j+1}+\emptyset_{n}^{j-1}\right)+(\epsilon\mu\left(1-\emptyset_{n}^{j}\right)+1-2r)\emptyset_{n}^{j} \qquad \text{(I)}$$

Donde $r=\frac{\gamma\epsilon}{2h^2}$

Vemos que la única diferencia con una ecuación solamente difusiva es el último término de la derecha, lo que implica que el único termino a cambiar en el método de Crank-Nicolson es la expresión a usar de β_k

La matriz entonces, para cada iteración, tendrá la forma:

$$\emptyset_k=\alpha_k\emptyset_{k+1}+\beta_k \qquad (II)$$

Para integrar la ecuación, iremos guardando arreglos que almacenaran los valores para cada iteración, resultando una matriz. La primera fila de esta matriz serán las condiciones iniciales, y las filas siguientes mostrarán los pasos temporales.

Nos damos las condiciones de borde:

- $n(t,0)=1$
- $n(t,1)=0$
- $n(0,x)=e^{-x^2/0.1}$

Para fijar esto, escribimos:

```
def poner_condiciones_iniciales(S, N, h):
  for i in range(N-1):
    x = i * h
    S[i] = np.exp(-x**2./0.1)

  S[0] = BORDE_1
  S[-1] = BORDE_2
  return S
```

Esta función recibe la matriz S, de dimensiones N, y el paso espacial h, y reemplaza su fila 0 con las condiciones iniciales, y retorna la nueva matriz.

Definimos ahora:

```
def calcula_b(b, N, r):
  for k in range(1, N - 1):
      b[k] = (S[k+1] * r + S[k-1] * r +
            S[k] * (e * MU * (1 - S[k]) + 1 - 2 * r))
```

Esta función calcula todos los elementos del lado derecho de la ecuación (I), es decir; todos los elementos asociados al tiempo anterior a la iteración correspondiente, para cada punto k.

También será necesario encontrar los coeficientes $\alpha \; y \; \beta$ de la ecuación (II), para esto definimos la función:

```
def encontrar_alfa_beta(alfa, beta, b, r,N ):
   A_mas = -1 * r
   A_menos = -1* r
   A_0 = (1 + 2 * r)
   alfa[0] = 0              # condiciones finales para alfa y beta
   beta[0] = BORDE_1
```

la cual determina $\alpha \; y \; \beta$ para cada t dado.

Finalmente, para hacer el paso temporal, definimos la función:

```
def avance_temporal(S, S_next, alfa, beta, N):
   S_next[0] = BORDE_1
   S_next[-1] = BORDE_2
   for k in range(int(N) - 2, 0, -1):
      S_next[k] = alfa[k] * S_next[k+1] + beta[k]
```

Esta función ira llenando las filas correspondientes de la matriz para cada salto temporal.

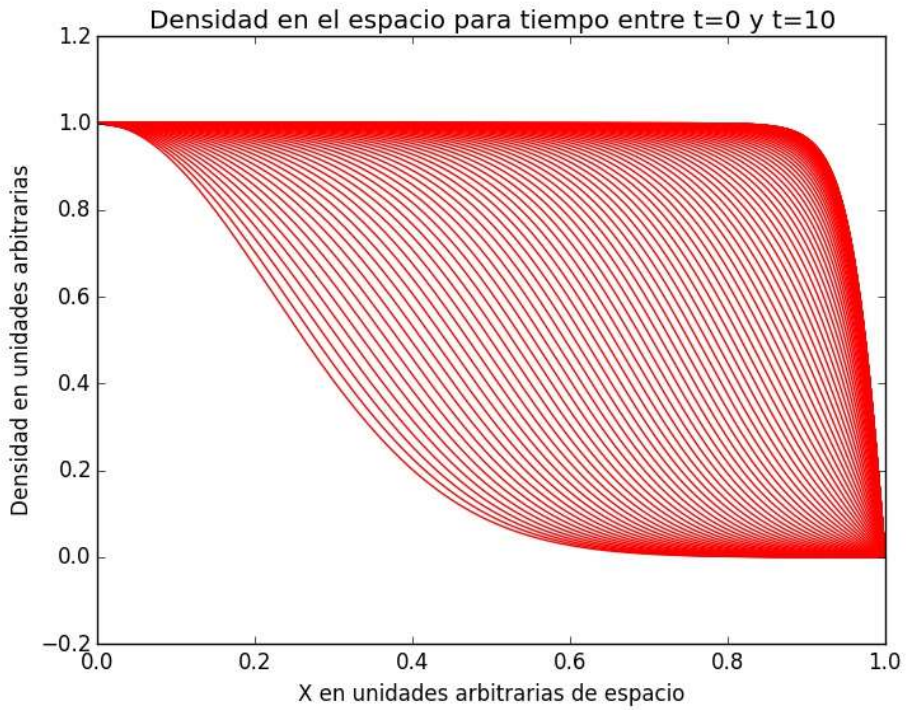
Ya que tenemos todas las funciones que necesitamos, inicializamos:

```
T_FIN = 10      # tiempo final
T_IN = 0       # tiempo inicial
X_FIN = 1      # extremo 2
X_IN = 0       # extremo 1
numero_t = 100      # cantidad de valores en t
numero_x = 500   # cantidad de valores en x
numero_pasos_t = numero_t - 1
numero_pasos_x = numero_x - 1
e = (T_FIN - T_IN) / (numero_pasos_t)
h = (X_FIN - X_IN) / (numero_pasos_x)
GAMMA = 0.001
MU = 1.5
BORDE_1 = 1     # condicion de borde 1
BORDE_2 = 0     # condicion de borde 2
r = (GAMMA * e) / (2 * h ** 2)

S = np.zeros(numero_x)
S_next = np.zeros(numero_x)

b = np.zeros(numero_x)
alfa = np.zeros(numero_x)
beta = np.zeros(numero_x)

S = poner_condiciones_iniciales(S, numero_x, h)

S_solucion = np.zeros((numero_t, numero_x))
S_solucion[0, :] = S.copy()
```

Y hacemos correr la iteración:

```
for i in range(1, numero_t):
   calcula_b(b, numero_x, r)
   encontrar_alfa_beta(alfa, beta, b, r, numero_x)
   avance_temporal(S, S_next, alfa, beta, numero_x)
   S = S_next.copy()
   S_solucion[i, :] = S.copy()
```

Al graficar para t entre 0 y 10, obtenemos:



Vemos que, mientras pasa el tiempo, la densidad poblacional tiende al valor 1 en todo el espacio. Esto se debe a que 1 es el único punto estable de la ecuación.

P2. Buscamos ahora resolver la ecuación de Newell-Whitehead-Segel, la cual está dada por:

$$\frac{\partial n}{\partial t} = \gamma \frac{\partial^2 n}{\partial x^2} + \mu(n - n^3)$$

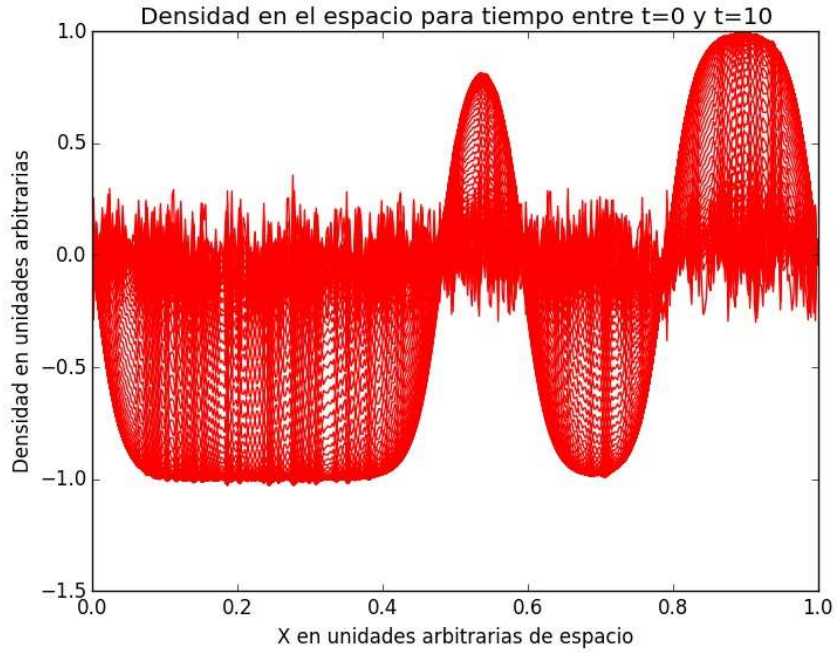
Que corresponde a una ecuacion de reacción-difusion que describe fenómenos de convección y combustión entre otros.

Nos damos esta vez las condiciones de borde:

- $n(t, 0) = 0$
- $n(t, 1) = 0$
- $n(0, x) = np.random.uniorm(low = \quad 0.3, high = 0.3, size$=Nx)

Para resolver esta pregunta utilizamos el mismo programa que para la P1, solo alteramos las condiciones de borde y el valor de β_k, que pasa a ser cubico.

Al graficar, obtenemos:



Se observa que la densidad converge a los puntos de equilibrio estables 1 y -1. En estos puntos se anulan los términos de reacción dejando la ecuación como solo de difusión.