2019

Modelos & Simulación

Trabajo Práctico Final

Tomás Gañan

CONTENIDO

Trabajo Práctico Nº1: Desarrollo de Modelos Simples

Contenido conceptual:

Teoría de sistemas. Teoría de modelos. Etapas de desarrollo a partir de sistemas reales. Modelos lineales y no lineales. Elementos de los modelos.

Objetivos:

- Comprensión de las etapas del modelado de sistemas reales.
- Identificación de los elementos de un modelo.
- Desarrollo y validez de modelos de sistemas reales lineales y no lineales.

Ejercicios:

En los siguientes ejemplos se estudiará y practicará las etapas en el análisis y elaboración de modelos de sistemas. A partir del modelo verbal, realizar el diagrama de efectos, luego el diagrama de simulación, y posteriormente el algoritmo necesario para llevar a cabo la simulación del modelo.

EJERCICIO 1: INFLUENCIA DE LA ESTRUCTURA Y PARÁMETROS DE UN SISTEMA.

Un sistema puede describirse con pocos elementos y pocas conexiones, pero no por ello su comportamiento dinámico será sencillo. Muchas veces un conocimiento a priori o intuitivo del sistema no alcanza para vislumbrar las diferentes posibilidades. El siguiente ejemplo, tiene por objeto demostrar la formalización de un modelo y su comportamiento dinámico, pero no el de describir acabadamente la realidad. Por ejemplo, se analizan las relaciones socioeconómicas de una sociedad y su carga sobre el medio ambiente.

Modelo Verbal: Se investigan las posibles relaciones entre el consumo de recursos naturales, la carga sobre el ambiente, el consumo específico de recursos por habitante, y la acción del estado. Manteniendo todos los demás parámetros constantes, a modo de simplificación, podemos afirmar (premisas):

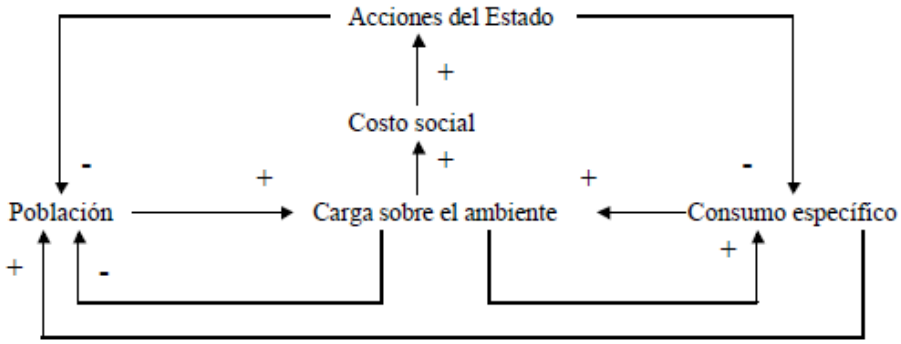
- Un aumento de la población conduce a un aumento de carga sobre el ambiente.
- Un aumento del consumo específico conduce a un aumento de carga sobre el ambiente.
- Un aumento de la carga ambiental produce un aumento del consumo específico para obtener mayores recursos con menor incidencia ambiental.
- Un aumento en la carga ambiental produce un aumento en el riesgo de la salud de la población (mortalidad y morbilidad).
- Un aumento en el consumo específico conduce, en general, a un aumento en el estándar de vida, y por ende a mejorar la salud y a menor mortalidad infantil. Por ende generando un aumento en la población.
- El Estado intenta, por su parte, a través de investigación y desarrollo reducir el consumo específico por habitante como medida para reducir la carga ambiental.

- Otras medidas del Estado pueden incluir influir en la población para estabilizar el crecimiento poblacional (educación, organización, etc.) de manera de producir una menor incidencia ambiental.

Dado el modelo verbal podemos determinar las siguientes variables:

- **Consumo**
- **Población**
- **Acciones**
- **Carga sobre el ambiente**
- **Costo social**

Diagrama de Efectos

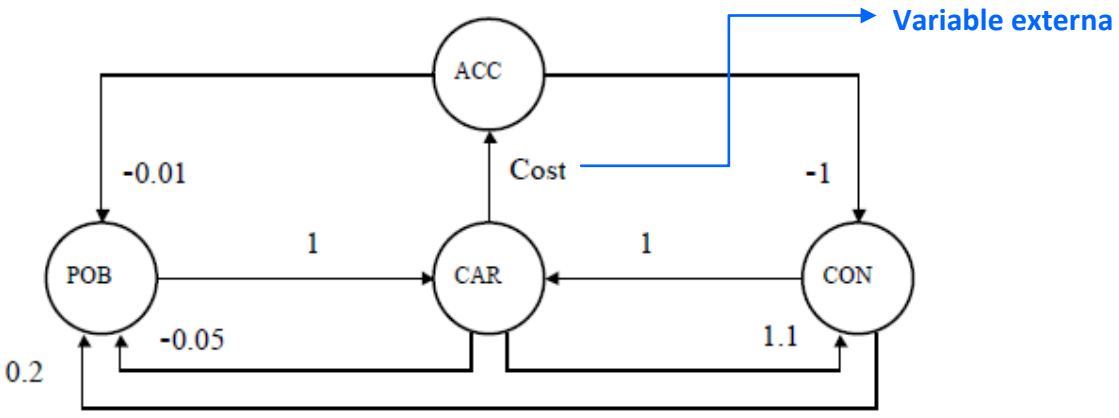


Cuantificación

Como primera aproximación, se dirá que si la relación es positiva y de la misma magnitud, será 1 (o -1 si fuera negativa). Si la influencia tiende a amplificar entonces se le asigna un valor mayor que 1, y si la influencia es débil, menor que 1.

Diagrama de Simulación

A partir del diagrama de efectos, se reemplaza por nombres de variables y se le asignan los coeficientes a las relaciones funcionales.

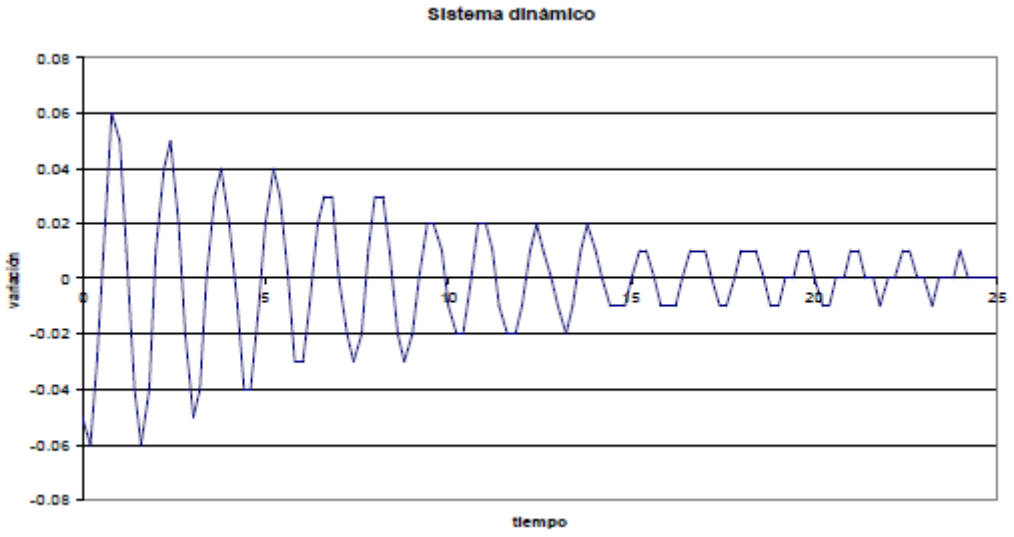


Programación: A partir de las relaciones del diagrama de efectos podemos transcribirlo en cualquier lenguaje de programación:

```
POB = 0.2 * CON - 0.01 * ACC - 0.05 * CAR

CAR = POB + CON

CON = 1.1 * CAR - ACC

ACC = COST * CAR
```

A través de las anteriores expresiones se podría calcular los sucesivos estados del sistema en un lazo determinado.

A continuación se ilustran los datos de salida (carga ambiental) para un costo del 80%:



EJERCICIO 2: LAS VARIABLES Y SU TASA DE CAMBIO

Las variables físicas en un sistema real se relacionan la mayoría de las veces con flujo de materia, energía o información. Estos flujos modifican el estado (o el valor actual) de las variables, produciéndose un aumento o una disminución del mismo. Estos cambios se asocian con la tasa del flujo (cambio de la variable por unidad de tiempo). Por lo tanto si durante un período de tiempo determinado, esta tasa no cambia, se puede calcular el estado de la variable a partir del valor actual más la tasa de cambio por el intervalo de tiempo transcurrido. Matemáticamente este concepto se asocia con el cálculo diferencial o integral de la variable. Se verá en el siguiente ejemplo, el uso de tasa de cambios, usando el cálculo numérico de una ecuación integral.

Modelo Verbal: El sistema consiste en una población homogénea de un país dado. En esta población no se considerarán los movimientos migratorios (de entrada o salida) de ese país. Por lo tanto la población queda determinada por su situación actual más las tasa de nacimiento y mortalidad. La tasa de natalidad se estima teniendo en cuenta la cantidad de padres potenciales en edad de procrear. Este valor es da en cantidad de nacimientos por cada 1000 habitantes por año. En cambio la tasa de mortalidad se expresa en cantidad de muertes por año para la población completa. La población actual, en cada paso del modelo, se determina como la población actual más la diferencia entre ambas tasas multiplicada por el intervalo de tiempo usado.

Dado el modelo verbal podemos determinar las siguientes variables:

- **Tasa de Natalidad**
- **Tasa de Mortalidad**
- **Población Inicial**

Diagrama de Efectos y Cuantificación: Reconocemos de éste dos lazos de realimentación, uno positivo, mientras más población, más nacimientos, y otro negativo, a mayor población mayor mortalidad y por ende menor población. Dependiendo cuál de estos dos lazos predomine, la población, crecerá exponencialmente, decrecerá exponencialmente o se mantendrá constante (si ambas tasas son iguales).

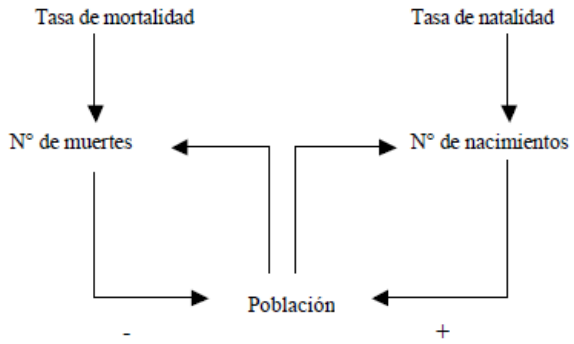
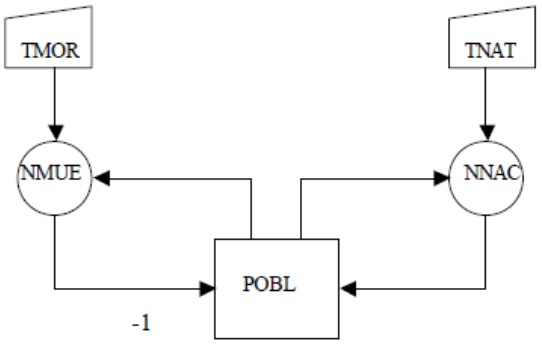


Diagrama de Simulación

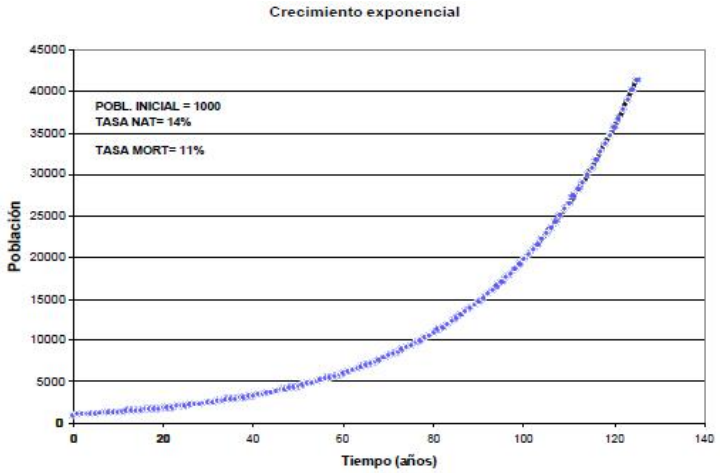
A partir del diagrama de efectos, se reemplaza por nombres de variables para construir el diagrama de simulación.



Programación: A partir de las relaciones del diagrama de efectos podemos transcribirlo en cualquier lenguaje de programación:

```
N_NAC = T_NAT * POB

N_MUE = T_MUE * POB

POB = POB + N_NAC - N_MUE
```

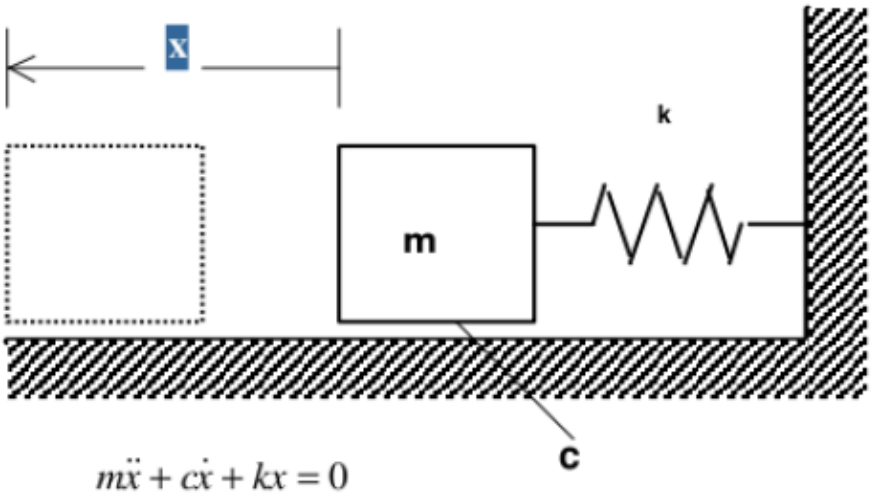
A continuación se ilustran los datos de salida correspondiente a la variable *población*:



EJERCICIO 3: SISTEMA DE SEGUNDO ORDEN. UN OSCILADOR LINEAL

Se estudiará el desarrollo de un modelo equivalente para un sistema de segundo orden representado por un oscilador lineal formado por una masa, un resorte y una amortiguación, en forma de rozamiento, como se ve en la figura siguiente.

Sistema de 2º Orden:



Modelo Verbal: La masa m está unida en un punto a una pared a través de un resorte. Si desplazamos inicialmente la masa una elongación x, y luego se deja libre, la masa retrocederá acelerándose hasta comprimir nuevamente el resorte una cierta distancia. Una vez finalizada la compresión del resorte, éste reaccionará impulsando la masa en sentido contrario, y por lo tanto estirando el resorte. Si el piso se supone suficientemente liso, la masa podrá desplazarse, pero el rozamiento de esta con el piso genera una fricción o amortiguación. Si se observa el movimiento que desarrolla la masa, la elongación de la misma se irá atenuando a medida que el impulso inicial se va amortiguando a causa del rozamiento. Esta fuerza de rozamiento dependerá de una constante c y de la velocidad de desplazamiento del cuerpo. A medida que la masa se desplaza cambia su velocidad y aceleración. La fuerza de atracción o repulsión del resorte dependerá de una constante elástica k y de su elongación o estiramiento. Finalmente la fuerza actuante que acelera la masa será la suma, instante a instante de la fuerza de elongación del resorte y la de rozamiento con el piso.

La frecuencia de oscilación dependerá de la relación constante de elasticidad del resorte y masa, pero también del valor de la amortiguación:

$$FrecOscNat = \frac{1}{\pi}\sqrt{\frac{k}{m}}$$

$$FrecOscAmort = \frac{1}{2\pi}\sqrt{\frac{k}{m}}\sqrt{1-\zeta} \quad \text{donde} \quad \zeta = \frac{c^2}{4.m.k} \left(donde\ \zeta\ es\ el\ coef.deamortiguación\right)$$

El valor del coeficiente de amortiguación determina los estados posibles de oscilación:

- $\zeta = 0$ Sin amortiguación.
- $\zeta^2 < 1$ Subamortiguado.
- $\zeta^2 = 1$ Amortiguación critica.
- $\zeta^2 > 1$ Sobreamortiguado.

Diagrama de Efectos y Cuantificación

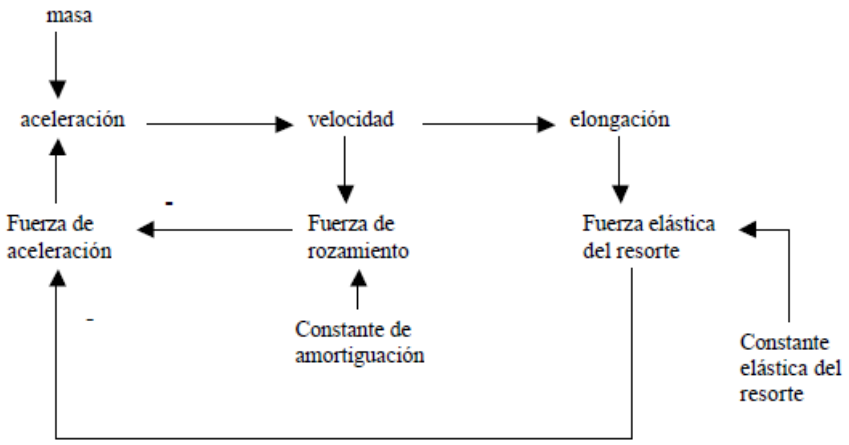
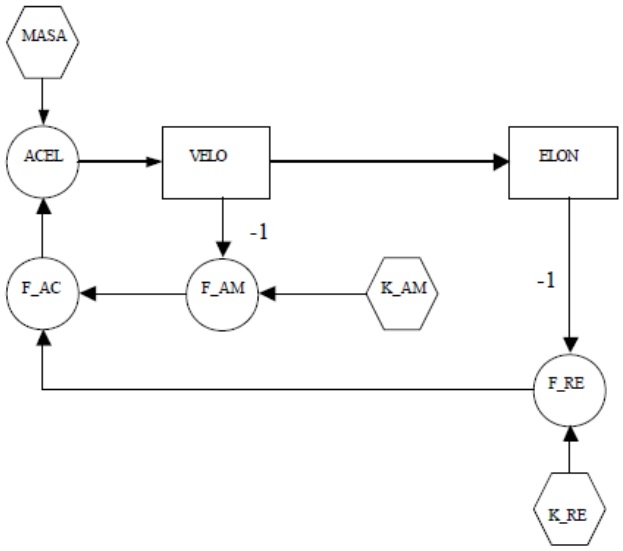


Diagrama de Simulación

A partir del diagrama de efectos, se reemplaza por nombres de variables para construir el diagrama de simulación.



En este ejemplo se ha asumido un oscilador de tipo lineal, es decir las constantes de elasticidad y amortiguación actúan linealmente sobre la elongación y velocidad respectivamente. Y así podemos considerar la suma de las fuerzas actuantes iguales a cero, una vez que desaparecen la fuerza exterior inicial de elongación. En estas condiciones definimos como parámetros y variables externas a MASA m (kg), constante de elasticidad K-RE (N/m) (con hexágonos) y la constante de amortiguación K-AM (N/m/s); y como variables la aceleración ACEL (m/s2), la velocidad VELO (m/s) y la elongación ELON (m). Como este sistema es de segundo orden se pueden definir dos variables de estado, que se comportan como integradores, esta son la velocidad y la elongación (en bloques cuadrados).

Programación: A partir de las relaciones del diagrama de efectos podemos transcribirlo en cualquier lenguaje de programación:

```
F_RE = K_RE * ELON

F_AM = - K_AM * VELOC

F_AC = F_RE + F_AM

ACEL = F_AC / MASA

VELOC = VELO + ΔT * ACEL
                                } Variables de estado
ELON = ELON + ΔT * VELOC
```

Diagrama de Elongación

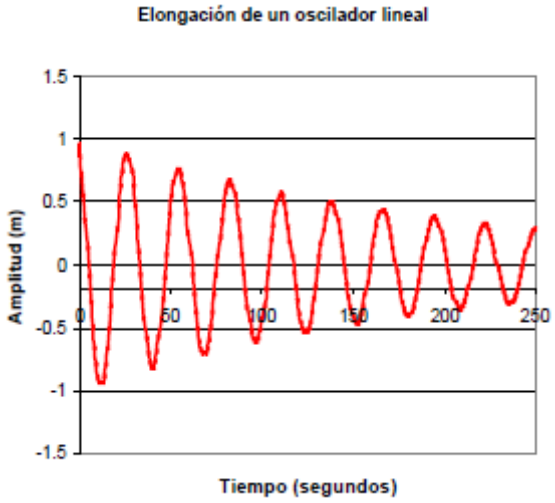
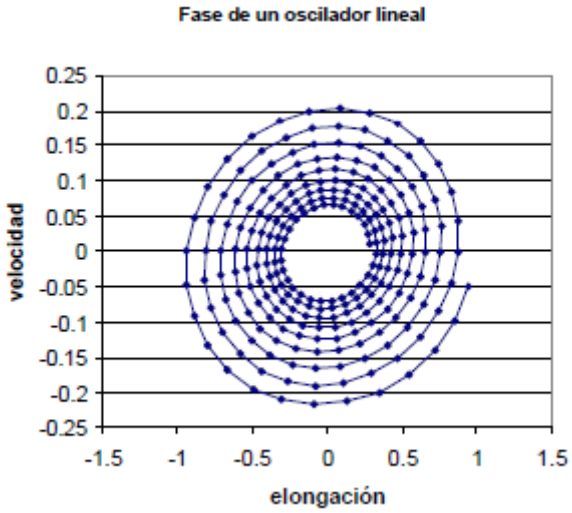


Diagrama de fase



La frecuencia de oscilación dependerá de la relación constante de elasticidad del resorte y masa, pero también del valor de la amortiguación.

EJERCICIO 4: SISTEMA NO LINEAL SIMPLE: EL CASO PRESA-PREDADOR

En un sistema lineal como el visto en el caso anterior, las principales variables del sistema, como la frecuencia de oscilación o la constante de amortiguación, no dependen de las condiciones iniciales, mientras que la amplitud de la oscilación y la velocidad, sí dependen de las condiciones iniciales. En todos los casos lineales, la posición y la velocidad del sistema pueden determinarse con precisión, instante a instante.

En cambio en los sistemas no lineales, todas las variables dependen fuertemente de las condiciones iniciales. Esto trae como consecuencia, que una leve variación o incertidumbre en el conocimiento de estas condiciones iniciales, modificará la posición final del sistema. Esta no linealidad se manifestará como un producto, cociente o exponenciación de dos variables del sistema. Veremos a continuación un ejemplo sencillo de no linealidad.

Modelo Verbal: Supongamos que tenemos dos poblaciones, una de zorros y otra de liebres en un campo cerrado natural, donde no intervienen otros animales ni predadores. Si no existiesen zorros, las liebres se multiplicarían, en tanto el campo les ofrezca suficiente pastura para alimentarse. Superado este límite las liebres se mueren de hambre y la población se estabiliza. Sin liebres, los zorros no tendrían alimentos, podrían sobrevivir un tiempo, pero luego morirán de hambre. Si coexisten ambas poblaciones, los zorros se alimentan de las liebres y se multiplicarán. Pero hasta un punto en que si la población de zorros es muy numerosa no alcanzarán las liebres para todos los zorros y comenzarán a morir de hambre. Con esto la población de zorros se estabilizará. A pesar de ser un sistema simple, no es tan sencillo de imaginar las variaciones y oscilaciones que puede sufrir este sistema. El comportamiento de este sistema no lineal, dependerá fuertemente de la cantidad de zorros y liebres iniciales que existan en el campo, como así también de las tasa de natalidad y mortalidad de cada especie.

Dado el modelo verbal podemos determinar lo siguiente:

- **Población 1: Zorros**
- **Población 2: Liebres**
- **Capacidad del terreno**
- **Las liebres "crecen" hasta completar la capacidad del terreno.**
- **Los zorros por defecto mueren salvo que puedan cazar liebres.**

Diagrama de Efectos y Cuantificación

Del modelo verbal nos damos cuenta que el sistema tiene varias realimentaciones, que dependiendo del peso de cada una hará que la balanza se incline a uno u otro lado (de cada especie).

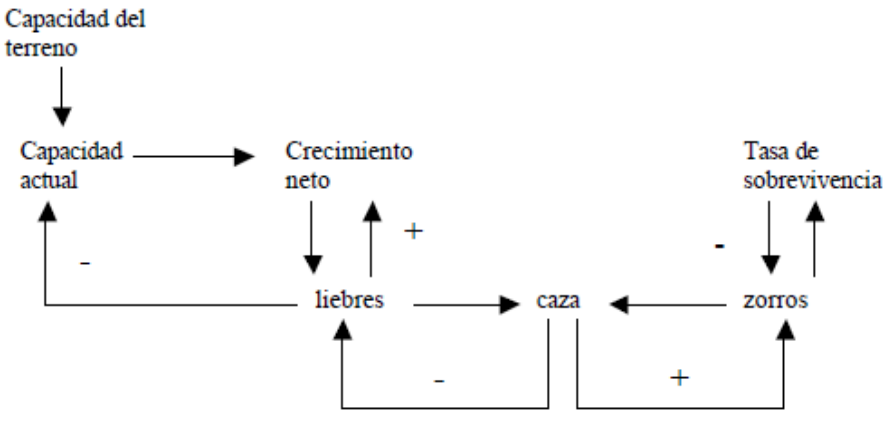


Diagrama de Simulación

A fin de convertir el diagrama de efectos en un diagrama de simulación, es preciso cuantificar las relaciones arriba expuestas. La variable tiempo puede expresarse en "semanas". Suponemos que la tasa de sobrevivencia de los zorros (sólo por respirar) es del orden del 20% de su biomasa. De allí que el coeficiente de zorros a tasa de sobrev. es de 0.2 y de –1 de tasa a zorros. El crecimiento de las liebres es del orden del 8% por semana, en condiciones ideales, es decir sin límite de alimentación provista por el campo.

Entonces la relación liebres a crecimiento neto es 0.08. Sin embargo este crecimiento está limitado a la capacidad actual del terreno, de allí que debemos relacionarla con esta variable y con la capacidad máxima del terreno. Las poblaciones de las liebres también están reguladas por la población de los zorros. Esta variable reguladora entre los zorros y las liebres la llamamos caza. A fin de evaluar los coeficientes de la caza, debemos estimar que los zorros puedan subsistir y compensar su gasto en respiración (energía interna). En una población normal de 500 liebres y 10 zorros, los zorros apenas pueden sobrevivir; la relación de caza entre zorros y liebres es el producto 500 libres * 10 zorros = 5000 encuentros. Las pérdidas de zorros es 0.2*10 = 2; de allí obtenemos que por cada liebre cazada sobreviven 2/5000 =0.0004 zorros.

Las pérdidas en la población de liebres por cada caza es 10 z. /5000 enc. = -0.002.

En resumen, podemos determinar lo siguiente:

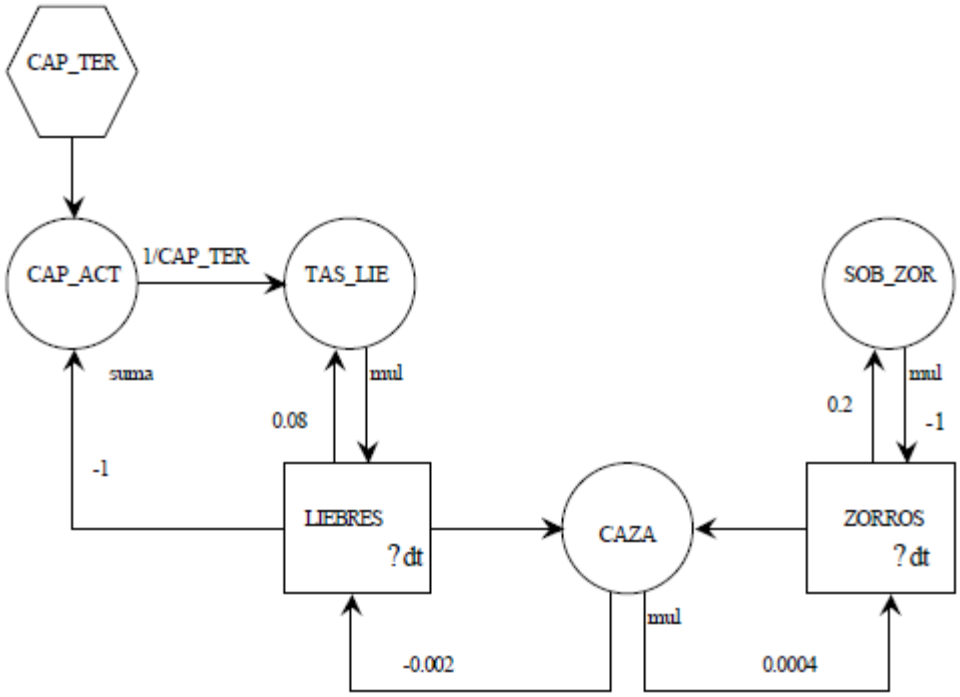
- **Tasa mortalidad de zorros = 0.2**
- **Tasa natalidad de liebres = 0.08**
- **Caza = Cantidad de encuentros posibles entre zorros y liebres**
- **Población Normal = 500 Liebres y 10 Zorros**
- **Caza = Zorros * Liebres = 5000 encuentros**

Zorros:

- **Sin liebres, muere el 20% de los zorros de un ciclo al otro.**
- **Por las condiciones iniciales mueren 2 zorros en el 1º ciclo donde de allí obtenemos que por cada libre cazada sobreviven -> 2 zorros / 5000 encuentros = 0.0004 zorros por encuentro posible.**

Liebres:

- **En cada encuentro entre zorro y libre (caza) -> 10 liebres / 5000 encuentros = 0.002 liebres mueren por encuentro posible.**



Programación: A partir de las relaciones del diagrama de efectos podemos transcribirlo en cualquier lenguaje de programación:

```
CAP_TER = 1500;   LIEBRES = 500;   ZORROS= 10;

T_LIE = 0.08;

T_ZOR = 0.2;

CAP_ACT = CAP_TER – LIEBRES

INC_LIE = T_LIE * LIEBRES * CAP_ACT / CAP_TER

DIS_ZOR = T_ZOR * ZORROS

CAZA = LIEBRES * ZORROS

LIEBRES = LIEBRES + ΔT * (INC_LIE – 0.002 * CAZA)
ZORROS = ZORROS + ΔT * (0.0004 * CAZA – DIS_ZOR)
```

Variables de estado

Diagrama de Elongación: En la siguiente ilustración se muestra la variación de la población

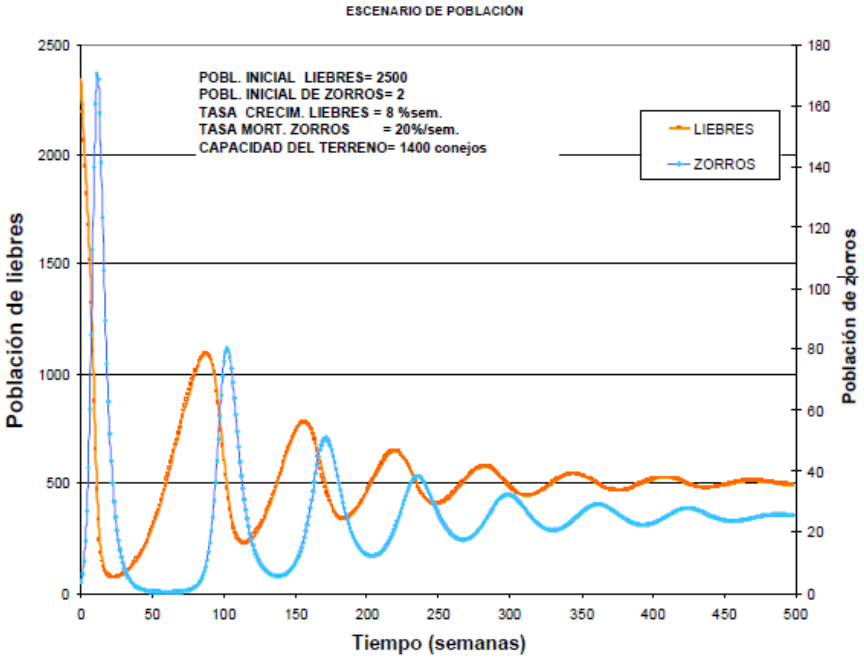
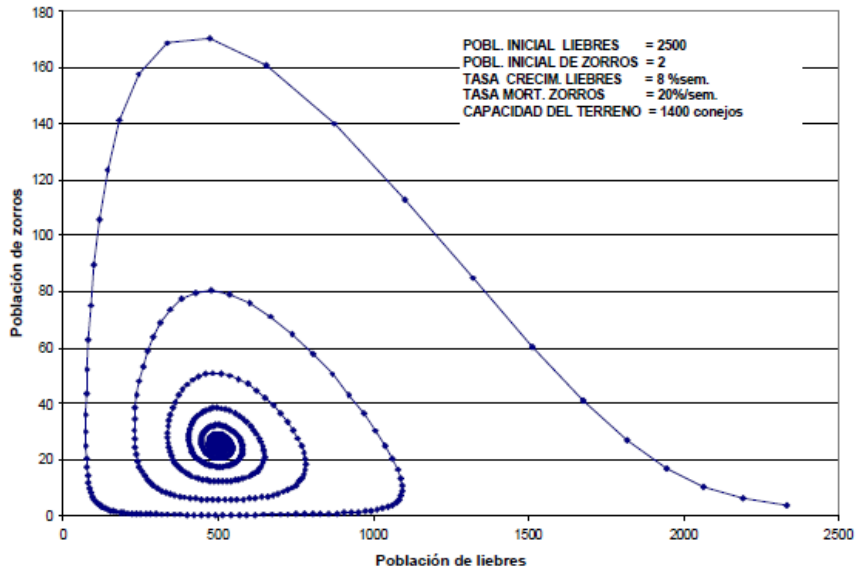


Diagrama de Fase



Trabajo Práctico Nº2: Resolución de casos de simulación por computadora

Contenido conceptual:

Etapas de desarrollo a partir de sistemas reales. Modelos lineales y no lineales. Control de aptitud de modelos. Análisis de incertidumbre y sensibilidad.

Objetivos:

- Desarrollo de software de computadora que simule sistemas reales.
- Análisis de los datos generados por software de computadora.
- Análisis de incertidumbre y sensibilidad de las variables de un modelo simulado.

EJERCICIO 1: SISTEMA NO LINEAL SIMPLE: EL CASO PRESA-PREDADOR

Realizar un programa interactivo del caso presa-predador visto en prácticas anteriores, que permita estudiar las salidas del programa cuando se cambien las condiciones de contornos y parámetros de entrada. Graficar en pantalla los diagramas poblacionales y el diagrama de fase.

Deberá presentar un informe que contenga:

- Una explicación de los algoritmos principales utilizados en el programa.
- Pruebas de simulación:
  - Una simulación con los valores por defecto: 500 liebres y 10 zorros.
  - Dos simulaciones distintas cambiando las condiciones iniciales.

Para cada caso realizar una interpretación de los gráficos de comportamiento de las poblaciones, incluyendo capturas de los gráficos poblacionales y de diagrama de fase.

Para dar comienzo a esta explicación comenzamos diciendo que para el siguiente algoritmo se utilizaron las ecuaciones de Lotka-Volterra las cuales son un modelo biomatemático que predice la dinámica de las poblaciones de las presas y de los predadores. Dicho modelo se trata de un sistema de dos ecuaciones diferenciales de primer orden no lineales:

$$\frac{dx}{dt} = (\alpha x - rx^2) - \beta xy$$

$$\frac{dy}{dt} = -\gamma y + \delta yx$$

X = Presas.
Y = Predadores.

α = Tasa de natalidad de Presas.
β = Liebres que mueren por encuentro posible.
γ = Tasa de mortalidad de Predadores.
δ = Predadores por encuentro posible.

Para el desarrollo de este ejercicio se ha utilizado Python 3.7.3 como lenguaje de programación en donde se ha desarrollo en Visual Studio Code 1.35.0 y compilado en un entorno virtual sobre la terminal macOs.

Una vez importadas las librerías y módulos a utilizar se procede a cargar el modelo matemático en nuestro programa:

```
# Definicion generica de sistema no lineal

# dx/dt = (alpha x - r x^2) - beta * x * y
# dy/dt = - gamma * y + delta * y * x

def df_dt(x, t, a, b, c, d, r):

    dx = a * x[0] - r * x[0]**2 - b * x[0] * x[1]
    dy = - c * x[1] + d * x[0] * x[1]

    return np.array([dx, dy])
```

Teniendo el modelo matemático planteado, procedemos a definir las condiciones iniciales las cuales se detallan a continuación:

```
# Condiciones iniciales

x0 = 500        #Liebres (Presa)
y0 = 10         #Zorros (Predador)

cond_ini = np.array([x0, y0])

a = 0.08       #Tasa de natalidad de Liebres (Presas)
b = 0.02       #Liebres que mueren por encuentro posible (Exito en la caza del depredador)
c = 0.2        #Tasa de mortalidad de Zorros (Predadores)
d = 0.01       #Zorros por encuentro posible (Exito en la caza)
r = 0.001

tf = 1000       #Tiempo final
nc = 100        #Cantidad de muestras/ciclos
```

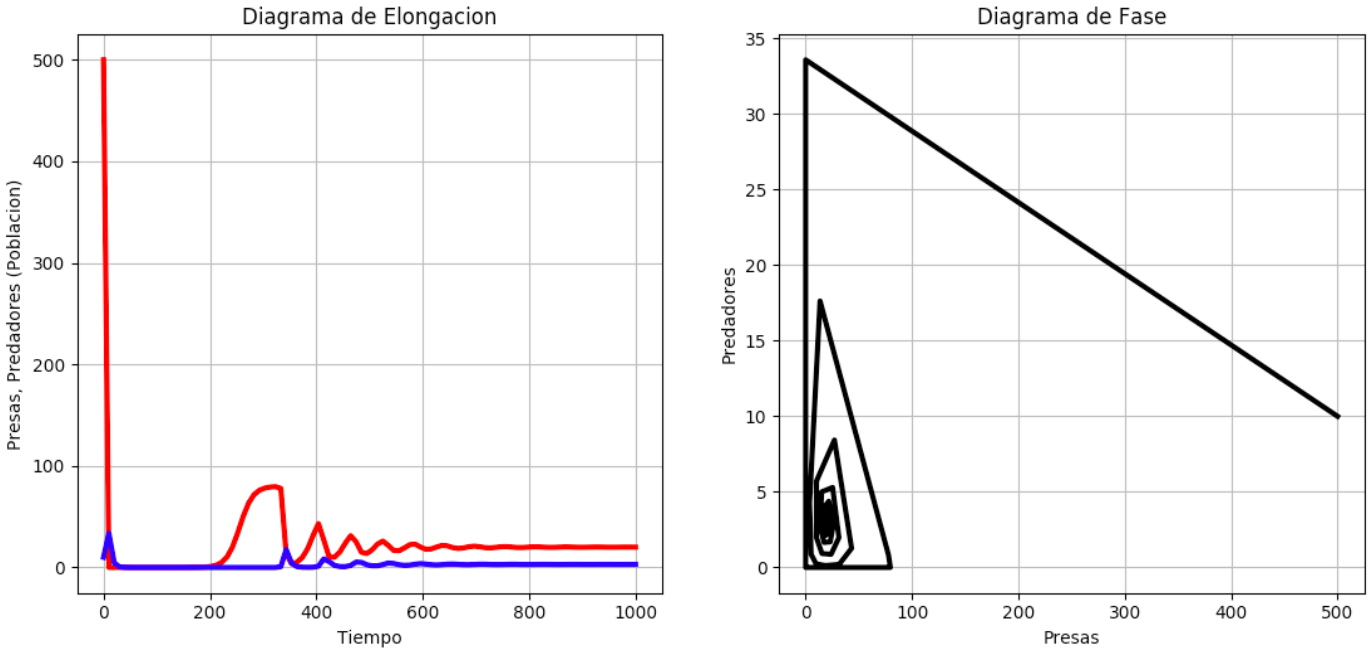
Una vez definidas las condiciones, pasamos a generar un vector e integrar con la función odeint de la librería o biblioteca libre SciPy.

```
t = np.linspace(0, tf, nc) #La funcion linspace nos devuelve un vector con 100 ciclos espaciados entre 0 y 1000

# Integramos nuestro sistema

sol = odeint(df_dt, cond_ini, t, args=(a, b, c, d, r))
```

Por último, procedemos a representar gráficamente nuestros diagramas haciendo uso de Matplotlib, biblioteca para generación de gráficos a partir de datos contenidos en listas o arrays.

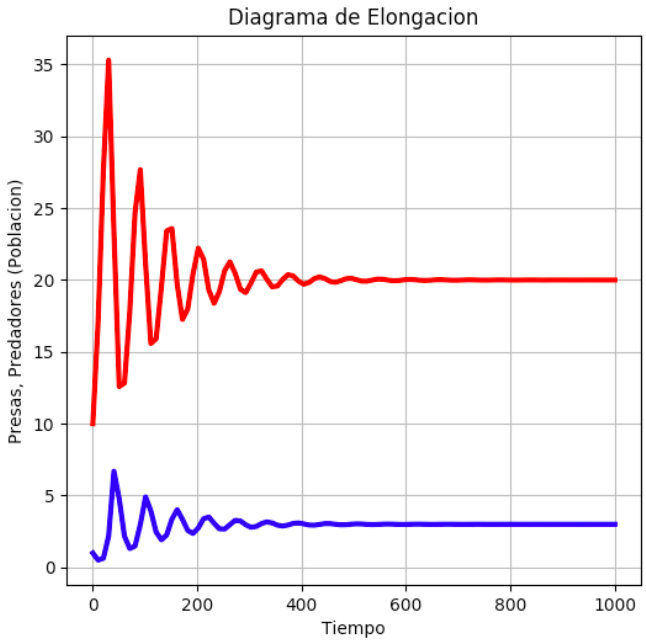
```
# Diagrama de elongacion

pyplot.subplot(121)
pyplot.plot(t, sol[:, 0], label='Presa', color='red', linewidth=3)
pyplot.plot(t, sol[:, 1], label='Depredador', color='blue', linewidth=3)
pyplot.grid(True)
pyplot.title('Diagrama de Elongacion')
pyplot.ylabel('Presas, Predadores (Poblacion)')
pyplot.xlabel('Tiempo')

# Diagrama de fase

pyplot.subplot(122)
pyplot.plot(sol[:, 0], sol[:, 1], color='black', linewidth=3,)
pyplot.grid(True)
pyplot.title('Diagrama de Fase')
pyplot.ylabel('Predadores')
pyplot.xlabel('Presas')

pyplot.show() # Nos imprime ambos diagramas
```

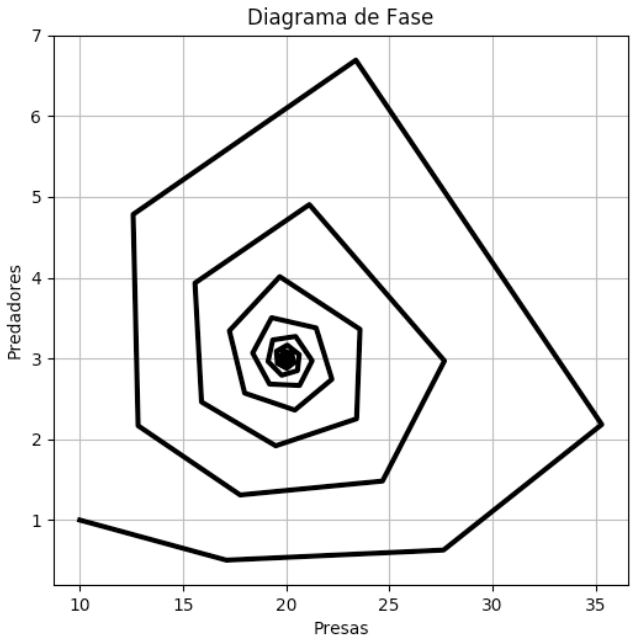
Simulación para 500 liebres y 10 zorros:



Se puede observar que ambas poblaciones tardan varios ciclos en estabilizarse ya que hay 10 predadores (zorros) en el terreno, los cuales son bastantes significativos para la cantidad de presas (liebres) que hay.

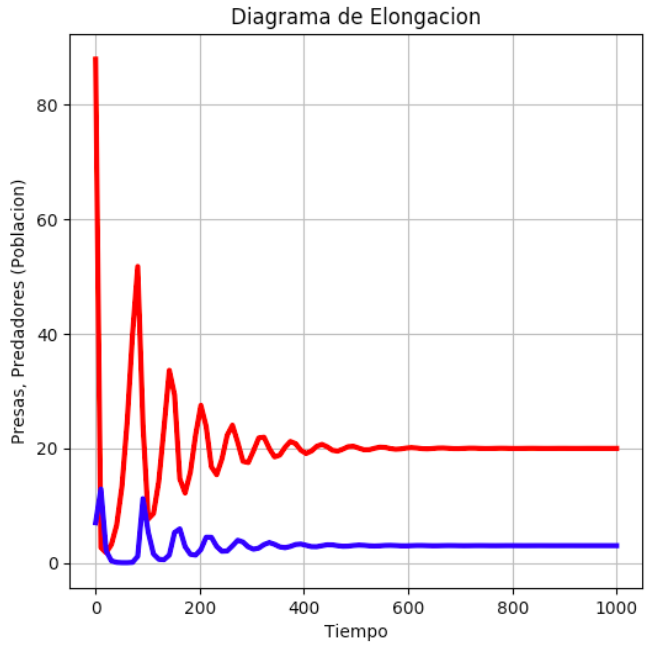
Simulación para 10 liebres y 1 zorro:





Al reducir de forma más representativa las presas notamos que las poblaciones se estabilizan rápidamente, en un corto lapso de tiempo y pocos ciclos.

Simulación para 88 liebres y 7 zorros:



De las 3 representaciones esta sería el punto medio, aunque cuenta con una representativa cantidad de predadores en función de presas que hay en la población.

EJERCICIO 2: EL JUEGO DE LA VIDA

El "juego" consiste en desarrollar un patrón de figuras que evolucionen de acuerdo a reglas predeterminadas, a partir de una configuración inicial y un conjunto de reglas. La competencia consiste en descubrir nuevas formas originales y calcular cuántas generaciones evoluciona el sistema antes de que se repitan o desaparezcan. Esto se realiza en una grilla de celdas que se estira al infinito en todas las direcciones. Este efecto lo lograremos considerando que la última columna de la derecha tenga por vecina a la primera columna de la izquierda, y lo mismo con la fila de abajo y de arriba. Una celda viva se marcará con un 1 o se pintará de un color gris oscuro, mientras que una celda muerta se marcará con un 0 o se pintará con un color blanco.

Las reglas son:

1. "Una celda viviente sobrevive únicamente si tiene 2 o 3 celdas vecinas vivas"
2. "El nacimiento de una nueva celda se da si esta tiene exactamente 3 celdas vivas vecinas"

Estas simples reglas tienen un potencial asombroso de generar patrones complejos, dependiendo del patrón inicial. En todo momento el programa debe mostrar la evolución de la grilla e ir mostrando el estado del sistema, es decir, la cantidad de celdas vivas y muertas, además del número de generaciones que han pasado hasta el momento. Se debe permitir el ingreso de celdas vivas en cualquier posición de la grilla antes de comenzar el juego. La grilla por defecto deberá tener 10 x 10. El programa se debe detener en caso de que se hayan muerto todas las celdas.

Para el desarrollo de este ejercicio se ha utilizado Python 3.7.3 como lenguaje de programación en donde se ha desarrollo en Visual Studio Code 1.35.0 y compilado en un entorno virtual sobre la terminal macOs.

El siguiente ejercicio cuenta con desarrollo orientado a objetos (POO) el cual cuenta con una única clase que contiene diversas funciones las cuales llaman a objetos.

El ejercicio utiliza las siguientes librerías:

```
# Importacion de librerias/modulos

import copy
import random
import itertools
import time
import os
```

Copy = Operaciones de copiado de bajo y alto nivel.
Random = Generación de números randoms.
Itertools = Funciones que crean iteradores, entre otras cosas.
Time = Funciones relacionadas con el tiempo.
Os = Manipulación del sistema operativo.

```
class juego_vida(object):

    def __init__(self, filas, colum):
        self.filas = filas
        self.colum = colum
        fila_viva = lambda: [random.randint(0, 1) for n in range(self.colum)]
        self.juego = [fila_viva() for n in range(self.filas)]
        self.viva = 1
        self.muerta = 1

    def __str__(self):
        tabla = ''

        for fila in self.juego:
            for celda in fila:
                tabla += ' @ ' if celda else ' . '
            tabla += '\n'
        tabla += "\n Celda Viva: {0} \n Celda Muerta: {1}".format(self.viva, self.muerta)

        return tabla

    def valuar(self, fila, col):
        espacio = list(set(itertools.permutations([-1, -1, 1, 1, 0], 2)))
        intro = lambda x, y: (x in range(self.filas) and y in range(self.colum))
        total = 0

        for r, c in espacio:
            if intro(r + fila, c + col):
                total += self.juego[r + fila][c + col]

        return total

    def test(self):
        juego_t = copy.deepcopy(self.juego)
        self.viva = 0
        self.muerta = 0

        for r in range(self.filas):
            for c in range(self.colum):
                total = self.valuar(r, c)

                if (total < 2 or total > 3) and juego_t[r][c]:
                    juego_t[r][c] = 0
                    self.muerta += 1
                elif total == 3 and not juego_t[r][c]:
                    juego_t[r][c] = 1
                    self.viva += 1

        self.juego = copy.deepcopy(juego_t)
```

```
filas, colum = int(input("Filas: ")), int(input("Columnas: "))
juego = juego_vida(filas, colum)
iteraciones = 0

while juego.viva > 0 or juego.muerta > 0:
    try:
        clear() #Liberacion de memoria
        juego.test() #Ejecucion
        print(juego)
        time.sleep(1) #Supension de la ejecucion
        iteraciones += 1 #Contador
    except KeyboardInterrupt:
        break
print("\n Total de generaciones: ", iteraciones)
```

Ejemplo de salida en una matriz de 10x10:

En primer lugar compilamos y ejecutamos nuestro programa el cual nos pedirá ingresar el numero de filas y el numero de columnas que deseemos:

```
(entorno) Air-de-Tomas:TP2 tomasganan$ python3 ej2.py
Filas: 10
Columnas: 10
```

Para este ejemplo ingresamos 10 filas y 10 columnas. A continuación se ilustra una captura del programa una vez finalizados sus ciclos:

```
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .

Celda Viva: 0
Celda Muerta: 0

Total de generaciones:  62
```

Como se puede observar, el programa pudo registrar que pasaron 62 generaciones desde que comenzó hasta que finalizó.

EJERCICIO 3: EL MÉTODO DE MONTECARLO

Una compañía de productos alimenticios necesita tomar decisiones estratégicas de ventas futuras, y uno de los datos que requiere es la estimación de visitas a su sitio web para los próximos 100 días.

El sysadmin de la empresa nos informa que el servidor web que aloja el sitio es propio, y que además cuenta con una consola de monitoreo, desde hace 250 días, que puede brindarnos información histórica de las visitas al sitio.

Se le solicita que desarrolle un software que, tomando los 250 datos datos históricos de visitas diarias al sitio web, y sabiendo que los mismos se distribuyen normalmente, permita estimar los datos futuros para los próximos 100 días.

Deberá presentar:

- El código fuente del software desarrollado en algún lenguaje de programación de su elección.
- La explicación del algoritmo principal.
- Los cálculos de las medidas estadísticas (media y varianza), tanto de los datos históricos proporcionados por la consola de monitoreo, como de los datos generados por su software.

Para el desarrollo de este ejercicio se ha utilizado Python 3.7.3 como lenguaje de programación en donde se ha desarrollo en Visual Studio Code 1.35.0 y compilado en un entorno virtual sobre la terminal macOs.

Una vez importadas las librerías y módulos a utilizar se procede a cargar los 250 datos históricos correspondientes a las visitas diarias del sitio web:

```
# Sistema

visitas = [33,18,27,36,25,24,30,23,29,38,23,22,19,21,18,28,12,20,22,22,22,34,32,15,14,16,32,17,22,37,11,22,37,15,29,11,36,24,18,19,23,
           12,21,7,30,6,26,31,24,31,25,21,25,21,24,30,29,18,19,16,17,20,24,18,19,43,9,37,32,26,25,18,16,14,23,37,31,34,21,22,26,16,24,
           14,25,32,11,30,30,17,18,36,14,31,24,35,12,19,17,21,28,14,22,39,27,18,33,39,30,32,26,32,19,35,26,20,23,23,23,23,11,20,15,19,
           33,23,40,25,18,16,20,23,20,31,23,20,38,30,19,15,6,23,23,21,17,27,19,23,44,19,23,20,24,22,18,23,21,16,10,23,26,27,29,21,13,32,
           22,27,28,35,24,16,14,25,32,22,32,24,33,32,32,25,24,28,23,16,27,27,10,32,13,29,27,22,17,25,33,46,17,35,17,20,22,30,31,27,21,32,
           18,19,25,26,20,33,19,30,11,34,34,32,24,30,26,28,29,19,21,21,30,26,25,18,22,21,37,24,25,35,18,16,29,25,20,26,28,32,16,15,42,29]
```

Luego se procede a calcular la media, varianza y desviación estándar de los datos históricos. También se calcula el mínimo y máximo del total de las visitas:

```
# Calculos Estadisticos Datos Historicos

media = np.mean(visitas)
varianza = np.var(visitas)
desviacion = np.std(visitas)

print("Datos Historicos")
print("Media: %.3f " % media + "| Varianza: %.3f " % varianza + "| Desviacion: %.3f " % desviacion)

# Min/Max

vis_min = min(visitas)
vis_max = max(visitas)
```

Si bien existen funciones de Python para calcular tanto la distribución normal como la distribución acumulada, decidí calcularlo de forma manual, es decir, cargando la siguiente expresión:

$$f(x) = -\frac{1}{\sigma \cdot \sqrt{2\pi}} \cdot e^{-\frac{1}{2} \cdot \left(\frac{X-\mu}{\sigma}\right)^2}$$

Luego, para la distribución acumulada realicé un contador para que me sumara ítem por ítem los elementos de la lista de la distribución normal. Por último genere 100 randoms que varian entre 0 y 1 con 3 decimales.

```
# Distribucion Normal

distr_norm = []
for values in range(vis_min,vis_max):
        den = desviacion * math.sqrt(2*3.14159)
        miem = pow(2.71828,-0.5*pow((values-media)/desviacion,2))
        distr_norm.append(round((1/den)*miem,3)) #Calculo de Distr. Norm manualmente

# Distribucion Acumulada

acum = 0
distr_acum = []

for ac in range(0,40):
        acum = acum + distr_norm[ac]
        distr_acum.append(round(acum,3))

# Generacion de 100 randoms

list_rand = []
for ran in range(100): # 100 randoms entre 0-1
        list_rand.append(round(random.uniform(0,1),3)) # Funcion round para truncar decimales
```

Como explicación a porque hice todo el proceso manual (distribución normal y acumulada) es porque los 100 randoms necesitaba compararlos ítem por ítem con la lista de la distribución acumulada para estimar las visitas.

```
comparacion = [item for item in list_rand if item in distr_acum]

if len(comparacion) > 0:
        print('|-------------------------------------------------------|')
        print ('Ambas listas contienen estos elementos: ')
        for item in comparacion: print(item)
        print('|-------------------------------------------------------|')
        print ('Respecto a su posicion, sus visitas estimadas son: ')
        for item in comparacion: print(comparacion.index(item)+6)
        print('|-------------------------------------------------------|')
else:
        print('No existe ningun elemento igual en las listas')
```

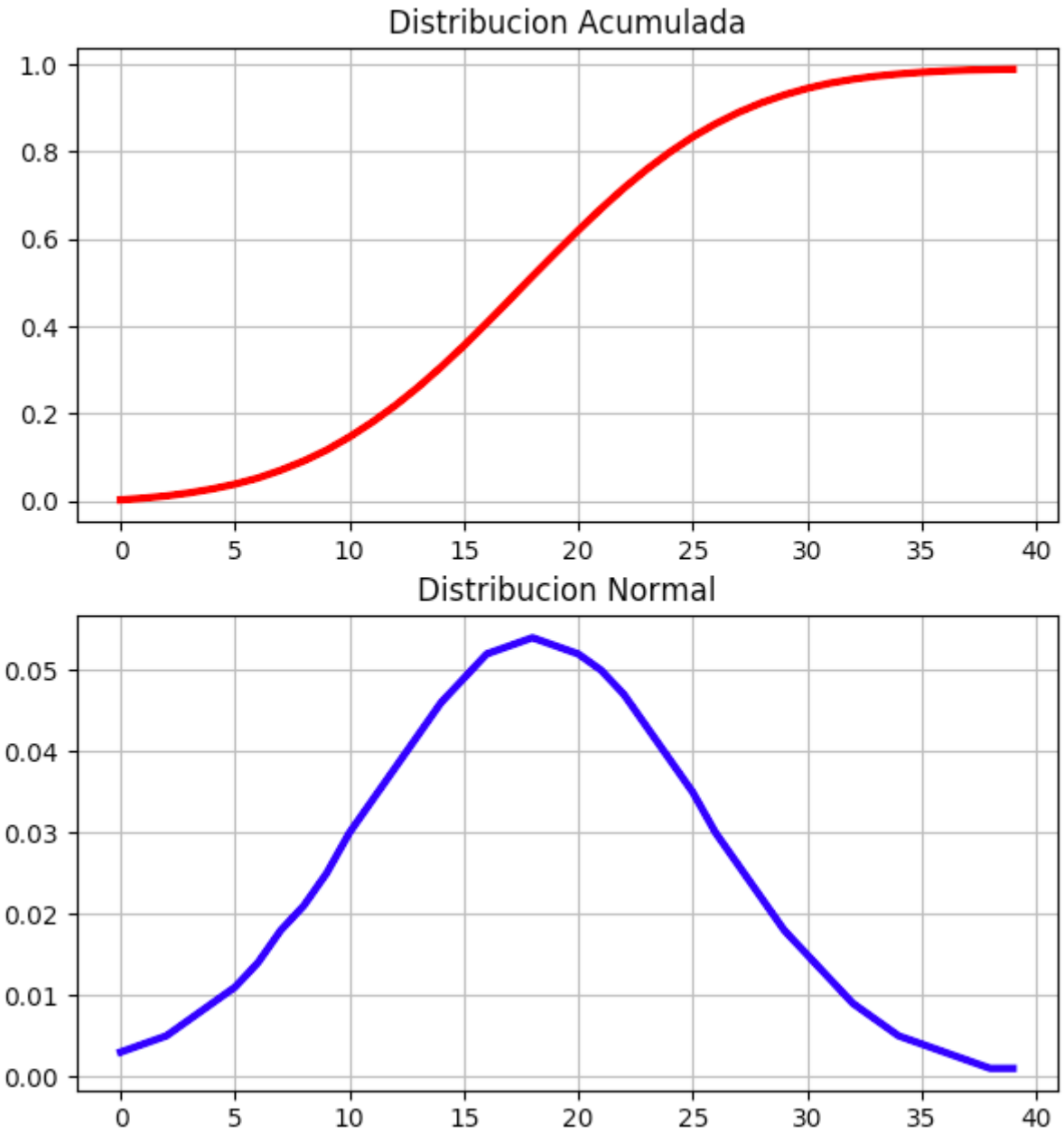
Por último calculo la media, varianza y desviación estándar de los datos generados y grafico todo el sistema, el cual nos devuelve lo siguiente:

```
# Calculos Estadisticos Datos Generados

mediaGen = np.mean(list_rand)
varianzaGen = np.var(list_rand)
desviacionGen = np.std(list_rand)

print("Datos Generados")
print("Media: %.3f " % mediaGen + "| Varianza: %.3f " % varianzaGen + "| Desviacion: %.3f " % desviacionGen)

# Distribucion Normal

pyplot.subplot(212)
pyplot.plot(distr_norm, color='blue', linewidth=3)
pyplot.grid(True)
pyplot.title('Distribucion Normal')

# Distribucion Acumulada

pyplot.subplot(211)
pyplot.plot(distr_acum, color='red', linewidth=3)
pyplot.grid(True)
pyplot.title('Distribucion Acumulada')

pyplot.show()
```

Salida por consola:

```
Datos Historicos
Media: 24.092 | Varianza: 55.132 | Desviacion: 7.425
|-----------------------------------------------------|
Ambas listas contienen estos elementos:
0.019
0.67
0.261
|-----------------------------------------------------|
Respecto a su posicion, sus visitas estimadas son:
6
7
8
|-----------------------------------------------------|
Datos Generados
Media: 0.469 | Varianza: 0.087 | Desviacion: 0.295
```

Distribución Normal y Distribución Acumulada:



Como conclusión final a este ejercicio, reconozco que no pude dar con la parte final de la consiga de este caso práctico ya que no puede estimar las visitas para 100 días, sino que comparé los randoms con la tabla de distribución acumulada y los valores que coincidían le sume '6' a su posición (ya que nuestro intervalo comenzaba en 6 y terminaba en 46) para estimar las visitas en esos valores que 'coincidieron'.