Trabajo de fin de grado

Grado en Ingeniería en Tecnologías Industriales

Desarrollo de herramientas de animación para la realización de vídeos didácticos de matemáticas para la ingeniería

MEMORIA

Autor: Marina Puyuelo Puyuelo
Director: Antonio Susín Sánchez
Codirector: Rafael Ramírez Ros

Convocatoria: junio 2021

Escuela Técnica Superior
De Ingeniería Industrial de Barcelona

Resumen

La importancia de crear contenidos didácticos para la enseñanza online y la motivación de la estudiante por aportar algo a la docencia de la escuela han impulsado este trabajo, que tiene por objetivo realizar vídeos explicativos de los contenidos de matemáticas en las carreras de ingeniería

El resultado obtenido son vídeos de alta calidad que combinan imagen con sonido. La imagen ha sido generada con MATLAB y el sonido se ha superpuesto en la posproducción. Para la programación de los vídeos en MATLAB se han creado diversas funciones que facilitan la creación de elementos animados tales como texto, funciones, formas geométricas o líneas.

En lo que al contenido de los vídeos se refiere, este proyecto se ha centrado en aquellos temas vistos durante las prácticas de asignaturas de matemáticas dentro del grado de Ingeniería en Tecnologías Industriales en la ETSEIB. No obstante, otro de los objetivos principales es que este documento pueda servir como guía a otros futuros trabajos que quieran seguir con la labor aquí iniciada, la de crear contenido audiovisual de refuerzo.

Índice

1. Glosario

Este documento contiene cierto vocabulario que puede resultar desconocido para el lector por lo que se va a proceder a explicarlo en primera instancia para la correcta comprensión del trabajo.

Código de colores **RGB**: proviene de las siglas en inglés «*Red*» «*Green*» and «*Blue*» y está basado en la composición del color a partir de estos tres colores primarios de la luz. Para especificar un color distinto a los 8 predeterminados ofrecidos por MATLAB (amarillo, magenta, azul cian, rojo, verde, azul, blanco y negro), se ha de rellenar un vector de dimensiones 1 x 3 cuyos tres valores corresponden al tanto por uno de cada uno de los tres colores primarios que componen el color final deseado. Así pues, el color blanco, por ejemplo, se codifica como [1 1 1] y el negro como [0 0 0] (ausencia total de color).

Lenguaje **LaTeX**: este lenguaje desarrollado por Leslie Lamport en 1984 es un sistema de composición de textos orientado a la creación de documentos escritos que presenten una alta calidad tipográfica y usado en artículos y libros científicos que incluyen expresiones matemáticas, como es el caso.

Struct (matriz de estructura): se trata de un tipo de datos de MATLAB agrupados mediante contenedores denominados campos. Su utilidad reside en que cada campo puede contener cualquier tipo de datos, desde una matriz o un vector hasta un string. También resulta atractivo su uso ya que la accesibilidad a estos campos es muy sencilla, basta con usar la notación de puntos de la forma NombreEstructura.NombreCampo. Véase un ejemplo de creación de una “struct”:

```
t1 = struct('Text',['The Game of Life  is a zero-player game,'],...
    'Loc',[n/2 XYCENTER(2)+m/12],'FS',25,'Style','Heat','Frames',30);
```

En el ejemplo anterior, los campos son: “Text”, “Loc”, “FS”, “Style” y “Frames”. Sus valores son los datos escritos inmediatamente después a ellos.

String: tipo de datos de MATLAB que recoge una cadena de caracteres, normalmente texto. Se declara mediante comillas simples o dobles.

Por último, otro tema de vocabulario importante es el uso recurrente que se hace de la palabra “animación”. Se desea poner de relieve que cada vez que se haga alusión a ella, se trata de una animación científica, de contenido didáctico y basada en unas fórmulas concretas. No debe confundirse con una animación cinematográfica de contenido artístico o de entretenimiento que tiene una gran componente manual.

2. Prefacio

2.1. Origen del proyecto

Este proyecto tiene su origen en el interés que despierta el software de programación MATLAB en la estudiante tras cursar la asignatura de la escuela Métodos Numéricos. Es por ello que mientras pensaba sobre el tema para el TFG se puso en contacto con el profesor que había impartido dicha asignatura, Toni Susín, es aquí donde nace este proyecto. Toni viene desde hace unos años creando material para la página web «Numerical Factory» por lo que pensó que sería interesante crear material audiovisual para la misma. Era una idea que ya tenía en mente y que desde el primer momento encajó como proyecto TFG.

El programa usado es MATLAB, como se ha mencionado, pues no solo es una herramienta previamente conocida por la estudiante, sino que es lo suficientemente potente y versátil como para desarrollar lo que se tiene en mente. Además, es uno de los métodos docentes más emergentes en la escuela, por lo que este trabajo destaca por su enfoque didáctico.

2.2. Motivación

La motivación principal de este trabajo ha sido el crear animaciones que puedan servir al alumnado futuro de la escuela. Es decir, la idea de que los vídeos aquí desarrollados sean mostrados y queden presentes en las prácticas de las asignaturas es lo que en mayor proporción ha impulsado su desarrollo.

Por otra parte, este trabajo también dedica una buena parte al diseño, ámbito en el cual está interesada la estudiante, por lo que su interés en él es todavía mayor. Como más tarde se verá, ciertos principios como el aspecto de la pantalla, la gama de colores o la disposición de los elementos, juegan un papel fundamental en las artes audiovisuales.

En tercer y último lugar, otra de las motivaciones del trabajo ha sido el profundizar las bases sobre el funcionamiento de este software tan útil hoy en día en muchos ámbitos.

En cierta manera el hecho de que los vídeos puedan perdurar en el tiempo es el fin máximo puesto que la estudiante siente que, tras cuatro años de grado, finalmente ha sido ella quien ha aportado algo a la docencia de la ETSEIB.

2.3. Requerimientos previos

Los requerimientos previos de este trabajo son fundamentalmente dos: informáticos y matemáticos. En primer lugar, informáticos porque las herramientas principales a utilizar son MATLAB y un editor de vídeos.

MATLAB (abreviatura de MATrix LABoratory, «laboratorio de matrices») es un software matemático de cómputo numérico que ofrece un entorno de desarrollo integrado con un lenguaje de programación propio (lenguaje M). Como paso previo a la programación, se debe leer una guía de estilo de MATLAB [1]. En ella, se habla de las convenciones para los nombres de las variables, las constantes, las estructuras y las funciones; se habla de la declaración de variables, constantes, variables globales, bucles y condiciones; y también se habla del diseño y los comentarios. Este documento es la base del estilo de los códigos creados.

Por su parte, un editor de vídeos es aquel programa que permite realizar todo tipo de variaciones sobre un vídeo y/o su montaje, en este caso, posibilita unir en un mismo archivo varios vídeos y superponer a ellos una «voz en off».

Los requerimientos matemáticos son imprescindibles puesto que el contenido del soporte audiovisual aquí creado es matemático. No tendría sentido entonces crear un material que no se entiende. Así pues, no es necesario estudiar, pero sí entender cada uno de los pasos que se está siguiendo y cada una de las explicaciones que se está dando.

En cuanto a cuál es el contenido matemático específico, se trata de una mezcla entre diversas asignaturas del grado en Ingeniería en Tecnologías Industriales tales como Álgebra Lineal, Cálculo I o Ecuaciones Diferenciales.

3. Introducción

En la actualidad se ha puesto de relieve la importancia de crear contenidos didácticos para la enseñanza online. En el ámbito de las matemáticas se recurre en la mayoría de casos a crear vídeos con cámaras estáticas y a escribir sobre un papel o pizarra. El proyecto consiste en crear, mediante la programación en MATLAB, las funciones y herramientas que permitan animar desde fórmulas a textos y dibujos matemáticos para así poder realizar vídeos explicativos de alta calidad sobre los contenidos de las matemáticas en las carreras de ingeniería.

3.1. Objetivos del proyecto

Los objetivos de este proyecto son fundamentalmente dos. El primero de ellos es diseñar las herramientas necesarias para la creación de vídeos y, el segundo de ellos es, a partir de estas, crear algunos vídeos como ejemplo.

La intención es que estos vídeos sean utilizados como soporte audiovisual de las prácticas del grado en Ingeniería en Tecnologías Industriales. Por lo tanto, el contenido de los vídeos es tal que tiene como objetivo, a su vez, servir como una ayuda, un apoyo a los estudiantes para entender mejor ciertos temas relacionados con las matemáticas.

3.2. Alcance del proyecto

El alcance de este proyecto es sobre la propia escuela, la ETSEIB, y, lo más importante, sobre sus alumnos, pues ellos son los beneficiarios fundamentales. Más concretamente el impacto se produce sobre los estudiantes de los primeros cursos ya que son ellos quienes probablemente necesiten este soporte audiovisual extra para un mejor asentamiento de lo aprendido o para una mejor comprensión.

Por otra parte, este trabajo también se concibe como guía para la futura creación de un mayor número de vídeos así que su alcance podría ser incluyo mayor, es decir, este trabajo podría ser la base de otros proyectos que continúen con la tarea aquí iniciada.

4. Metodología

Con tal de poder realizar las animaciones de una manera sencilla, se ha seguido una misma metodología en cada una de ellas.

4.1. Formato «.mp4»

El primer paso de la metodología es elegir el formato en el que se van a crear los vídeos. MATLAB, por defecto, crea vídeos con la extensión «.avi» pero permite generar y almacenar vídeos con otros formatos tales como «.mp4» o «.gif» gracias simplemente a la implementación de un par de líneas de código.

El proceso de creación de un vídeo implica esencialmente tres pasos: crear el archivo de vídeo, escoger los fotogramas de la animación y finalmente, grabarlos en el archivo. Para el primer paso MATLAB ofrece la función «VideoWriter», encargada de asignar el nombre al fichero y la extensión a utilizar.

Véanse a continuación las líneas de código a ejecutar para crear los ficheros de vídeo en los diferentes formatos. Por ejemplo, para crear un archivo de nombre «helix» en formato «.avi» se procede de la siguiente manera:

```
myVideo = VideoWriter('helix.avi'); %name for the video file
```

En cambio, si se desea crear un archivo del mismo nombre, pero en formato «.mp4» se procede así:

```
myVideo = VideoWriter('helix','MPEG-4');
```

Por último, en el caso de querer obtener el vídeo en formato «.gif»:

```
gif ('helix.gif');
```

El segundo y tercer paso recordemos son escoger los fotogramas de la animación y grabarlos dentro del fichero. Para ello, las instrucciones son «getframe» y «writeVideo» respectivamente, tanto para el caso de «.mp4» como para el caso de «.avi»:

```
frame = getframe; %get the present plot
writeVideo(myVideo,frame); %save it to the video
```

Pero en el caso de la extensión «.gif», la instrucción es diferente y algo más directa:

```
gif;
```

Dada la variedad de formatos que se pueden obtener, una de las primeras decisiones de este trabajo ha sido la elección entre las diferentes extensiones. Finalmente se ha optado por la «.mp4» puesto que, en comparación con las extensiones tipo «.avi» o tipo «.gif», ocupa un espacio inmensamente menor para una calidad cualitativamente similar. A continuación, se muestran las características de los diferentes formatos obtenidos para un vídeo generado con el mismo código.

Formato	Calidad	Tamaño del archivo
«.mp4»	Algo inferior (inapreciable cualitativamente)	157 KB
«.avi»	Superior (inapreciable cualitativamente)	2,86 MB
«.gif»	Alta	1,37 MB

Tabla 1. Comparación de los diferentes formatos de vídeo.

4.2. Apariencia unificada y parámetros predeterminados

Se ha aplicado el principio de que las animaciones deben tener una apariencia similar, por ello, en todas ellas se observa un mismo color de fondo, gris oscuro, y una misma paleta de colores, tonos oscuros para el fondo y claros para el texto y el resto de elementos. La manera de llevar a cabo esto semiautomáticamente ha sido crear un fichero Matlab en el cual se predefinen una serie de parámetros. Luego, este fichero se importa en las primeras líneas de cada uno de los códigos, haciendo posible que una gran cantidad de información no se tenga que copiar y pegar en cada uno de los códigos. El fichero se denomina «*parametros*» y su código se muestra a continuación.

```
%Global Variables regarding final video info
global myVideo
global saveVideo

saveVideo=1; %this value can take only two values: 0 (not to save the video) or 1

global BGCOLOR
global FGCOLOR
global XYCENTER
global FONTSIZE
global TEXTSPEED
global LINEWIDTH
global ANIMSPEED
global ANIMSTYLE
```

```
%Default values
BGCOLOR = [0.25 0.25 0.25]; %Backgroud color
FGCOLOR = [1 1 1];          %Foreground color
FONTSIZE = 20;
TEXTSPEED = 0.01;
LINEWIDTH = 2;
ANIMSPEED = 0.01;
ANIMSTYLE = 'Heat';         %Objects appear gradually
ASPECTRATIO = 16/9;
```

Las variables globales se declaran para no tener que inicializarlas en cada uno de los códigos donde se alude a ellas. Es una buena herramienta, aunque no se debe abusar de ella. Seguidamente, se procede a definir el significado de las variables globales creadas en el código superior.

- «myVideo»: variable a la que posteriormente en cada vídeo se le asigna un objeto de tipo «VideoWriter» sobre el que guardar el vídeo. Véase a continuación un ejemplo de esta instrucción:

  ```
  myVideo=VideoWriter('Transformations F2 modul3','MPEG-4');
  ```

- «saveVideo»: variable numérica que puede tomar únicamente dos valores. Si está establecida en 0, el vídeo mostrado en MATLAB no se guarda como archivo externo mientras que, si está establecida en 1, el vídeo sí se guarda.

- «BGCOLOR»: vector de dimensión 1x3 que contiene el color del fondo (“background” en inglés, de ahí su abreviación BGC) en código RGB. Por defecto está establecido en [0.25 0.25 0.25], un gris oscuro.

- «FGCOLOR»: vector de dimensión 1x3 que contiene el color del texto o de otros elementos en código RGB. La abreviación FGC proviene de la palabra en inglés “foreground”, que hace referencia al primer plano de una imagen. Por defecto está establecido en [1 1 1], blanco.

- «XYCENTER»: vector de dimensión 1x2 que contiene las coordenadas del centro geométrico de la ventana. Se trata de una variable muy útil a la hora de posicionar todos los elementos tales como texto, líneas o puntos. No tiene valor por defecto porque su valor va a depender de cada vídeo.

- «FONTSIZE»: tamaño del texto. Su valor ha de ser un número real positivo y por defecto es 20. Cabe mencionar el pequeño inconveniente que presenta este parámetro en MATLAB y es que es un valor relativo, como una constante de proporcionalidad entre el tamaño de un píxel y el tamaño del texto. Por lo tanto, un mismo texto no se ve del mismo tamaño en todos los ordenadores, depende de la resolución de cada monitor. De todas maneras, una vez ya se ha grabado en formato «.mp4», el tamaño ya no va a variar, simplemente es al ejecutar el código y visualizar la animación directamente en MATLAB que las dimensiones oscilan.

- «TEXTSPEED»: variable usada en funciones como «animaText» para indicar cuánto tiempo de pausa ha de haber entre fotogramas consecutivos al escribirse un texto (solo válida para la visualización en MATLAB). Se indica con un valor

numérico que hace referencia al tiempo en segundos. Por defecto es 0.01 segundos.

- «LINEWIDTH»: valor numérico positivo que indica el grosor de las líneas. Por defecto está establecido en 2. Un valor superior indica un mayor grosor y viceversa.

- «ANIMSPEED»: variable usada en funciones como «animaText» para indicar cuánto tiempo (en segundos) de pausa ha de haber entre fotogramas consecutivos al animarse algún elemento general (solo válida para la visualización en MATLAB). Ha de ser un valor numérico. Por defecto es 0.01 segundos.

- «ANIMSTYLE»: string que hace referencia al modo en el que se va a animar un elemento. Por defecto es 'Heat' porque esta opción la tienen todas las funciones que "animan" elementos, pero puede tomar otras opciones. Estas se explicarán más adelante.

En la última línea del fichero de «parametros» se declara la variable, no global, «ASPECTRATIO». Su valor es numérico e igual a 16/9, que es la proporción del marco del vídeo que se quiere mantener en todo momento. Por tanto, no es tanto un valor por defecto sino un valor único y válido para todas y cada una de las animaciones.

4.3. Organización modular

Una vez se hubo iniciado el proceso de crear vídeos y, en consecuencia, de crear código, se entendió la necesidad de separar el mismo en diferentes ficheros, a los cuales se ha denominado módulos.

Esta necesidad viene impulsada por la complicación que implican códigos tan extensos como los que se han podido crear en este trabajo. Un código largo impide muchas veces localizar las líneas que se desea hallar y/o modificar, pese a la existencia de la herramienta buscar («Ctrl+F»); también dificulta una rápida detección de errores, esto es, si se pretende hacer alguna comprobación del código mediante su propia ejecución—visualización en este caso—, cuanto más largo sea el mismo, más se tardará en comprobar la parte final. Por el contrario, si el código se encuentra dividido en módulos, cada uno de los cuales es un fichero distinto, los posibles errores son más fácilmente localizables.

Hubiera cabido la posibilidad de crear un único fichero por vídeo pues MATLAB ofrece la funcionalidad de generar subsecciones dentro de un mismo fichero, pero esta opción queda descartada porque igualmente implica códigos extremadamente largos.

Por ejemplo, en el caso de la animación «*Function Transformations*», los tres módulos en los que queda dividida corresponden a una introducción, a la explicación de las transformaciones sobre la variable x y a la explicación de las transformaciones sobre la

variable y. De esta manera se obtienen tres ficheros de 151, 288 y 301 líneas de código respectivamente en lugar del correspondiente fichero unificado de 740 líneas.

4.4. Funciones creadas

Durante las primeras semanas de trabajo en este TFG, se constató la importancia de crear funciones que permitieran automatizar y animar tareas como las de dibujar una función o escribir un texto, ya que iban a ser acciones recurrentes. Por ello, se han concebido una serie de funciones que generan movimiento a partir de funcionalidades previamente disponibles en la biblioteca de MATLAB.

4.4.1. Función «animaText»

Para animar la aparición de texto, se ha desarrollado la función «animaText». Está basada en la función de MATLAB «*text*», que se ha introducido dentro de bucles «*for*». Estos, para un número determinado de veces («*Frames*»), van cambiando gradualmente las características del texto, bien sea su posición, su color o su tamaño, consiguiendo el efecto final de animación.

El argumento de entrada a esta función es solo uno: una variable de tipo «struct». Esta variable puede contener distintos campos:

- «Text»: es el campo más importante puesto que contiene el texto a animar. Ha de ser un string o un vector de strings.

  Es aquí donde se aprovecha la potencia del procesador de texto “LaTeX” introducido en el glosario y que sirve para expresar fórmulas matemáticas. En términos generales, cada vez que se vaya a hacer uso de este lenguaje, se ha de escribir un símbolo del dólar ($), seguido del símbolo barra inclinada hacia la izquierda (\) y, a continuación, lo que queramos expresar. Todo ello cerrado finalmente con un símbolo del dólar de nuevo. Véanse algunos ejemplos de notación en MATLAB y del resultado visual final:

`'Reference Function: $a \sin(w_1x) + b \cos(w_2x)$'`	Reference Function: $a\sin(w_1x) + b\cos(w_2x)$
`'$y = f(x)\pm \alpha$, where $\alpha > 0$'`	$y = f(x) \pm \alpha$, where $\alpha > 0$
`'$y = \frac{f(x)}{\beta}$'`	$y = \frac{f(x)}{\beta}$

Tabla 2. Ejemplos de uso del lenguaje LaTeX y su visualización final.

- «Loc»: es la posición donde se va a localizar el texto. Ha de ser un vector de dimensión 1x2 cuyos elementos son respectivamente la coordenada de abscisas y la coordenada de ordenadas.

- «HA»: es el tipo de alineamiento horizontal que se desea. Dependiendo de si la posición de referencia («Loc») ha de estar centrada, ha de quedar a la izquierda del texto o ha de quedar a la derecha del texto, se indica «center», «left» o «right» respectivamente como valor de este campo.

- «FS»: es el tamaño de la fuente. Ha de ser un número real positivo y en su valor por defecto llama a la variable global «FONTSIZE».

- «TC»: es el color del texto (la abreviación procede de las siglas en inglés “Text Color”). Ha de ser un vector en código RGB, es decir, un vector 1x3, cuyos tres elementos han de estar comprendidos entre 0 y 1.

- «Frames»: valor numérico entero referente al número de secuencias/fotogramas que se han de capturar mediante el comando «writeVideo» para ser incluidas en el archivo de vídeo. Juega uno de los papeles más importantes pues de él depende la duración del mismo.

- «Speed»: valor numérico que indica los segundos que dura el comando «pause» allí donde se incluya. Solo es válido para la visualización en MATLAB, lo cual quiere decir que esta pausa no se produce en el vídeo grabado en formato «.mp4».

- «Style»: es la variable de selección del estilo. Por defecto llama a la función global «ANIMSTYLE». Si se quiere escoger el estilo, este ha de ser uno de los siguientes strings:

    - ‘Heat’: el texto cambia su color gradualmente desde el indicado en «Param» hasta el indiciado en «TC». Para ello se hace uso de la interpolación lineal.

    - ‘Zoom’: el texto incrementa su tamaño de manera gradual, desde el tamaño indicado por el campo «Param», que por defecto es 1 por lo que casi no se aprecia el texto, hasta el tamaño indicado por la variable «FS». Se aplica el concepto de interpolación lineal para cambiar sutilmente el tamaño.

    - ‘Slide’: el texto se desliza para entrar en la figura desde la orientación indicada por «Param» hasta la posición final indicada en «Loc». De nuevo se hace uso de la interpolación lineal para crear efecto de movimiento.

- «Param»: es una variable de la cual hacen uso los estilos explicados con

anterioridad y tiene un significado diferente en cada uno de ellos. Es un vector 1x3 en código RGB para el caso de 'Heat', cuyo valor por defecto llama a la variable global «BGCOLOR»; es un valor numérico (tamaño) en el caso de 'Zoom', cuyo valor por defecto es 1; y es un vector de coordenadas 1x2 en el caso de 'Slide', cuyo valor por defecto llama a la variable global «XYCENTER».

A continuación, se muestra una parte del código, en la cual se han omitido la importación inicial del fichero «parametros», varias líneas de comentarios respecto a la definición de las variables y la declaración de los valores por defecto de los campos de la "struct".

```
function h = animaText(s)
switch s.Style
    case 'Heat'
        h = text(s.Loc(1),s.Loc(2),s.Text,'interpreter','LaTeX',...
            'FontSize',s.FS,'HorizontalAlignment',s.HA,'Color',s.Param);
        for iFrame = 1:s.Frames
            iColor = s.Param + (s.TC-s.Param)*iFrame/s.Frames;
            delete(h);
            h = text(s.Loc(1),s.Loc(2),s.Text,'interpreter','LaTeX',...
            'FontSize',s.FS,'HorizontalAlignment',s.HA,'Color',iColor);
            pause(TEXTSPEED)
            if (saveVideo ==1)
                frame = getframe; %get the present plot
                writeVideo(myVideo,frame);
            end
        end
    case 'Slide'
        XY0 = s.Param;
        dXY = (s.Loc-s.Param)/s.Frames;
        h = text(XY0(:,1),XY0(:,2),s.Text,'interpreter','LaTeX',...
            'FontSize',s.FS,'HorizontalAlignment',s.HA,'Color',s.TC);
        for iFrame = 1:s.Frames
            iXY = XY0 + dXY*iFrame;
            delete(h);
            h = text(iXY(:,1),iXY(:,2),s.Text,'interpreter','LaTeX',...
            'FontSize',s.FS,'HorizontalAlignment',s.HA,'Color',s.TC);
            pause(s.Speed)
            if (saveVideo ==1)
                frame = getframe; %get the present plot
                writeVideo(myVideo,frame);
            end
        end
        case 'Zoom'
        h = text(s.Loc(1),s.Loc(2),s.Text,'interpreter','LaTeX',...
            'FontSize',s.Param,'HorizontalAlignment',s.HA,'Color',s.TC);
        for iFrame = 1:s.Frames
            fsi = s.FS*iFrame/s.Frames;
            delete(h);
            h = text(s.Loc(1),s.Loc(2),s.Text,'interpreter','LaTeX',...
            'FontSize',fsi,'HorizontalAlignment',s.HA,'Color',s.TC);
            pause(s.Speed)
            if (saveVideo ==1)
                frame = getframe; %get the present plot
                writeVideo(myVideo,frame);
            end
        end
end
```

4.4.2. Función «animaFunction»

Para animar el dibujo de una función, se ha desarrollado la función «animaFunction». Está basada en la función de MATLAB «*plot*», que se ha introducido dentro de bucles «*for*». A diferencia de la función descrita en el apartado 4.4.1, en esta no se emplea el concepto de «*Frames*», sino que el número de iteraciones del bucle «*for*» depende de la longitud del vector x, el cual sí es un argumento de entrada.

Los argumentos de entrada a esta función son tres: el argumento “x”, el argumento “y”, y una variable de tipo «struct».

El argumento “x” es un vector de la forma [1 x n] donde n es el número de columnas, que corresponde en este caso al número de puntos que componen la función. Cuanto mayor sea este, más tardará MATLAB en dibujar la función debido a que debe procesar una mayor cantidad de datos, pero también más preciso y definido será el dibujo de la función. Podría pasar que un número insuficiente de puntos diera como resultado una gráfica con forma poligonal y no suavizada.

El argumento “y” también es un vector de la forma [1 x n] de las mismas características. Como es lógico, ambos deben tener el mismo número de columnas, ya que “x” e “y” están haciendo referencia a las coordenadas 2D de un conjunto de puntos.

Respecto al argumento de entrada que es una «struct», en ella se pueden definir los siguientes campos:

- «LT»: es el tipo de línea (la abreviatura LT proviene de las siglas en inglés “Line Type”). Puede ser uno de los siguientes strings: ‘- ‘, ‘--‘, ‘:’ o ‘-.’, por defecto es ‘-‘.

- «LW»: es el grosor de la línea (la abreviatura LW procede de las siglas inglesas “Line Width”). Ha de ser un número real positivo, que por defecto es el asignado a la variable global «LINEWIDTH».

- «LC»: es el color de la línea y sus siglas provienen de las siglas en inglés “Line Color”. Ha de ser un vector 1x3 en código RGB y su valor predeterminado es el asignado a la variable global «FGCOLOR».

- «Speed»: al igual que en la función “animaText”, es el valor numérico del comando «pause» y que indica los segundos de pausa allí donde se incluya (solo válido para la visualización en MATLAB). Por defecto se le asigna el valor que tenga la variable global «ANIMSPEED».

- «Style»: es la variable de selección del estilo. Por defecto llama a la función global

«ANIMSTYLE», establecida a su vez por defecto en 'Heat'. Si se quiere escoger otro estilo, se ha de especificar. En el caso de esta función, se dispone de dos estilos:

  - 'Heat': la función aparece dibujada en su totalidad, pero gradualmente puesto que su color comienza siendo el elegido en el campo "Param" (por defecto corresponde a la variable global "BGCOLOR") y va cambiando hasta ser del color escogido en el campo «LC».

  - 'Normal': la función se va a ir dibujando desde su punto inicial hasta su punto final. La velocidad a la que es graficada depende del número de puntos por los que está compuesta la función, cuanto mayor sea el número de puntos, menor velocidad.

- «Param»: es una variable de la cual hacen uso los estilos explicados en el campo 'Style' y tiene un significado diferente en cada uno de ellos. Para el efecto 'Heat', este parámetro tiene el mismo significado que el explicado en la función "animaText". Para el efecto 'Normal', no es necesario este parámetro extra.

A continuación, se muestra el código. Téngase en cuenta que se ha omitido parte de él, concretamente las líneas donde se especifican los valores por defecto de los campos del argumento «struct» (s), las líneas con comentarios y las líneas donde se importa el fichero «parametros», que contiene la declaración de las variables globales.

```
function h=animaFunction(x,y,s)
switch s.Style
    case 'Normal'
        h = plot(0,0,s.LT,'LineWidth',s.LW,'color',s.LC);
        for i=1:length(x)
            set(h, 'Xdata',x(1:i), 'Ydata', y(1:i));
            pause(s.Speed)
            if (saveVideo ==1)
                frame = getframe; %get the present plot
                writeVideo(myVideo,frame);
            end
        end
        % to recover the complete function
        set(h, 'Xdata',x(1:end), 'Ydata', y(1:end));
        if (saveVideo ==1)
            frame = getframe; %get the present plot
            writeVideo(myVideo,frame);
        end

    case 'Heat'
        XY = [x',y'];
        h = plot(XY(:,1),XY(:,2),s.LT,'LineWidth',s.LW,'Color',s.Param);
        for iFrame = 1:length(x)
            iColor = s.Param + (s.LC-s.Param)*iFrame/length(x);
            delete(h);
            h = plot(XY(:,1),XY(:,2),s.LT,'LineWidth',s.LW,'Color',iColor);
            pause(s.Speed)
            if (saveVideo ==1)
                frame = getframe; %get the present plot
```

```
                writeVideo(myVideo,frame);
            end
        end
end
```

4.4.3. Función «animaRectangle»

La obligación de programar esta función nace cuando se inicia la creación de la animación «Juego de la vida». El código proporcionado para esta práctica de la asignatura de Álgebra Lineal[3], hace uso del comando «*spy(matrix)*» para dibujar la matriz introducida, pero no conviene utilizarlo puesto que sus opciones son limitadas. Son limitadas ya que, en primer lugar, este comando abre una figura de MATLAB y la matriz coloreada se distribuye a lo largo de toda esta ventana por lo que no hay lugar para el posicionamiento de texto u otros elementos necesarios en nuestras animaciones, es decir, no se puede modificar la disposición de la ventana una vez abierta. En segundo lugar, se desea controlar fácilmente las instrucciones de tamaño y posición de los elementos de la matriz y este comando no es muy flexible ya que cada disposición de celdas requiere un llamamiento al comando «*spy(matrix)*» y, en consecuencia, requiere una nueva matriz. Por el contrario, la nueva función sí es capaz de modificar fácilmente la distribución de celdas pues puede acceder a cada una de ellas. Por último, para conseguir el dinamismo deseado en una animación, con el comando «*spy(matrix)*» sería necesario recurrir a algún bucle mientras que, si se programa una nueva función, esta ya lo incluye y solo es necesario "llamarla".

Debido a lo explicado, se crea "animaRectangle", que anima el dibujo de un rectángulo. Está basada en la función de MATLAB «*rectangle*», introducida dentro de bucles «*for*». De este modo, mientras se recorren los "for", programados para iterar el número de veces indicado por el campo de la "struct" «*Frames*», el rectángulo va cambiando gradualmente el color de relleno y de bordes, de manera que se va intensificando. Esta función no dispone de múltiples efectos, únicamente sigue el comportamiento explicado.

El argumento de entrada a esta función es uno: una variable de tipo «struct» (s), en la cual se pueden definir los siguientes campos:

- «coordIni»: es la coordenada del eje de abscisas de la esquina inferior izquierda del rectángulo. Ha de ser un valor numérico contenido dentro de los límites establecidos para este eje, de no ser así, se omitiría parte del dibujo del rectángulo. Su valor por defecto es 0.

- «coordFin»: es la coordenada del eje de ordenadas de la esquina inferior izquierda del rectángulo. De igual manera que para la «coordIni», ha de ser un valor numérico comprendido dentro de los límites establecidos para este eje. Su valor por defecto es 0.

- «finalColor»: color final que adquiere el relleno del rectángulo. Ha de ser un vector 1x3 representado en código RGB. Su valor por defecto es [0.9 0.25 0.25].

- «initialColor»: color de partida del relleno del rectángulo. También ha de ser un vector en código RGB. Su valor por defecto llama a la variable global «BGCOLOR».

- «eColor»: color final del borde del rectángulo. Ha de ser un vector en código RGB. Su valor por defecto llama a una variable denominada «axisXcolor», que viene importada desde otro fichero.

- «Frames»: valor numérico entero referente al número de secuencias/fotogramas que se han de capturar mediante el comando «writeVideo» para ser incluidas en el archivo de vídeo. Su valor por defecto es 20.

- «Speed»: valor numérico positivo cuya función es la misma que la citada para las funciones “animaText” y “animaFunction”. Su valor por defecto es 0.05.

- «xLon»: longitud horizontal del rectángulo. Ha de ser un valor numérico positivo. Su valor por defecto es 1.

- «yLon»: longitud vertical del rectángulo. Ha de ser un valor numérico positivo. Su valor por defecto es 1.

A continuación, se muestra una parte del código, en la cual se han omitido la importación inicial del fichero «parametros», varias líneas de comentarios respecto a la definición de las variables y la declaración de los valores por defecto de los campos de la “struct”.

```
function h = animaRectangle(s)

h = rectangle('Position',[s.coordIni s.coordFin s.xLon s.yLon],'FaceColor',...
    BGCOLOR,'EdgeColor',s.eColor);
for iFrame = 1:s.Frames
    iColor = s.initialColor + (s.finalColor-s.initialColor)*iFrame/s.Frames;
    delete(h);
    ieColor = s.initialColor + (s.eColor-s.initialColor)*iFrame/s.Frames;
    h = rectangle('Position',[s.coordIni s.coordFin s.xLon s.yLon],'FaceColor',…
        iColor,'EdgeColor',ieColor);
    pause(s.Speed);

    if (saveVideo ==1)
        frame = getframe; %get the present plot
        writeVideo(myVideo,frame);
    end
end
```

4.4.4. Función «animaSegment»

En cuarto lugar, se crea la función “animaSegment” para poder generar líneas allí donde se desee, pues es uno de los elementos más versátiles. Las líneas se pueden usar a modo de flechas, a modo de ejes, para esquematizar e incluso para crear las divisiones de una cuadrícula.

Está basada en la función de MATLAB «*plot*», introducida dentro de bucles «*for*». Viene a ser un caso concreto de la función “animaSegment”, ya que una línea es la gráfica de una función lineal, pero se ha creído necesaria su implementación para una mayor agilidad. Además, se ha dotado a esta función de una mayor gama de estilos que a la función “animaFunction”. De igual manera que en el caso de las funciones “animaText” y “animaRectangle”, se utilizan los «Frames» para recorrer los bucles «for» tantas veces como este campo indique.

El argumento de entrada a esta función es uno: una variable de tipo «struct» (s), en la cual se pueden definir los siguientes campos:

- «From»: es un vector de coordenadas 1x2 que representa el punto de inicio del segmento y su valor predeterminado es la variable global «XYCENTER».
- «To»: es un vector de coordenadas 1x2 que representa el punto final del segmento y su valor predeterminado es XYCENTER + [1.,0.]
- «LT»: misma descripción que para el caso “animaFunction”.
- «LW»: misma descripción que para el caso “animaFunction”.
- «LC»: misma descripción que para el caso “animaFunction”.
- «Frames»: misma descripción que para el caso “animaText”.
- «Speed»: misma descripción que para el caso “animaFunction”.
- «Style»: la variable de estilo puede escogerse de entre los siguientes strings:
    - ‘Heat’: el segmento aparece dibujado en su totalidad, pero gradualmente puesto que su color comienza siendo el elegido en el campo “Param” (por defecto corresponde a la variable global “BGCOLOR”) y va cambiando hasta ser del color dictado por el campo «LC». Esto se logra gracias a la interpolación lineal.
    - ‘Rotate’: el segmento aparece dibujado en su totalidad y gira un determinado

número de radianes, el indicado por «Param».

- ‘Zoom’: el segmento va aumentando su longitud de manera gradual, el efecto es como si se estirara. Este estiramiento se produce desde el punto indicado por «Param»—pudiendo ser este un punto dentro del segmento o uno exterior, por defecto es el punto medio—hasta que el segmento ocupa su posición final. Para ello se utiliza la interpolación lineal.

- ‘Slide’: el segmento se desliza para entrar en la figura desde la orientación indicada por «Param» (por defecto el centro de la figura), hasta la su posición final. De nuevo se hace uso de la interpolación lineal para crear efecto de movimiento.

- ‘Stroke’: el segmento se va a ir dibujando desde su punto inicial hasta su punto final. La velocidad a la que este se grafica depende del valor de «Frames».

- «Param»: es una variable de la cual hacen uso los diferentes estilos (campo ‘Style’) y tiene un significado diferente en cada uno de ellos. Para el efecto ‘Heat’, este parámetro tiene el mismo significado que el explicado en la función “animaText”. Para el efecto ‘Rotate’ es un valor numérico expresado en radianes que representa cuánto ha de girar el segmento. Para el efecto ‘Zoom’ este parámetro es un vector de coordenadas 1x2 que representa el punto desde donde se empieza a agrandar el segmento. Para el efecto ‘Slide’ es un vector de coordenadas 1x2 que representa el punto desde donde aparece y comienza a trasladarse el segmento y, por último, para el efecto ‘Stroke’, no es necesario este parámetro extra.

A continuación, se muestra una parte del código,

```
function h = animaSegment(s)
switch s.Style
    case 'Heat'
        XY = [s.From; s.To];
        h = plot(XY(:,1),XY(:,2),s.LT,'LineWidth',s.LW,'Color',s.Param);
        for iFrame = 1:s.Frames
            iColor = s.Param + (s.LC-s.Param)*iFrame/s.Frames;
            delete(h);
            h = plot(XY(:,1),XY(:,2),s.LT,'LineWidth',s.LW,'Color',iColor);
            pause(s.Speed)
            if (saveVideo ==1)
                frame = getframe; %get the present plot
                writeVideo(myVideo,frame);
            end
        end
    case 'Stroke'
        XY = [s.From; s.From];
        h = plot(XY(:,1),XY(:,2),s.LT,'LineWidth',s.LW,'Color',s.LC);
        for iFrame = 1:s.Frames
            XY(2,:) = s.From + (s.To-s.From)*iFrame/s.Frames;
```

```
            delete(h);
            h = plot(XY(:,1),XY(:,2),s.LT,'LineWidth',s.LW,...
                'Color',s.LC);
            pause(s.Speed)
            if (saveVideo ==1)
                frame = getframe; %get the present plot
                writeVideo(myVideo,frame);
            end
        end
    case 'Zoom'
        XY = [s.Param; s.Param];
        h = plot(XY(:,1),XY(:,2),s.LT,'LineWidth',s.LW,'Color',s.LC);
        for iFrame = 1:s.Frames
            XY(1,:) = s.Param + (s.From-s.Param)*iFrame/s.Frames;
            XY(2,:) = s.Param + (s.To-s.Param)*iFrame/s.Frames;
            delete(h);
            h = plot(XY(:,1),XY(:,2),s.LT,'LineWidth',s.LW,...
                'Color',s.LC);
            pause(s.Speed)
            if (saveVideo ==1)
                frame = getframe; %get the present plot
                writeVideo(myVideo,frame);
            end
        end
    case 'Slide'
        XY0 = [s.Param; s.Param - s.From + s.To];
        h = plot(XY0(:,1),XY0(:,2),'LineWidth',s.LW,'Color',s.LC);
        for iFrame = 1:s.Frames
            XY = XY0 + (s.From-s.Param)*iFrame/s.Frames;
            delete(h);
            h = plot(XY(:,1),XY(:,2),'LineWidth',s.LW,'Color',s.LC);
            pause(s.Speed)
            if (saveVideo ==1)
                frame = getframe; %get the present plot
                writeVideo(myVideo,frame);
            end
        end
    case 'Rotate'
        theta = linspace(s.Param,0,s.Frames);
        vect = (s.To - s.From)';
        R = @(th) [cos(th), sin(th); -sin(th), cos(th)];
        vect0 = R(s.Param)*vect;
        XY = [s.From; s.From+vect0'];
        h = plot(XY(:,1),XY(:,2),'LineWidth',s.LW,'Color',s.LC);
        for iFrame = 1:s.Frames
            XY(2,:) = s.From + (R(theta(iFrame))*vect)';
            delete(h);
            h = plot(XY(:,1),XY(:,2),'LineWidth',s.LW,'Color',s.LC);
            pause(s.Speed)
            if (saveVideo ==1)
                frame = getframe; %get the present plot
                writeVideo(myVideo,frame);
            end
        end
end
```

4.5. Interpolación lineal

Se ha explicado cómo, con las diferentes funciones creadas, se pueden lograr diferentes efectos tales como hacer aparecer con zoom un texto o que una línea se vaya dibujando poco a poco, pues bien, esto es resultado de la interpolación lineal, uno de los principios básicos de las matemáticas y que se ha aplicado en numerosos casos aquí. A modo de resumen, se muestra en la *Tabla 3* cuáles son las diferentes variables sobre las que se aplica este concepto en cada uno de los efectos.

Efecto	**Heat**	**Zoom**	**Slide**	**Stroke/Normal**
Variable que interpola	Color del elemento	Tamaño del elemento	Posición del elemento	Porción del segmento

Tabla 3. Variables sobre las que se aplica la interpolación lineal en los diferentes efectos.

El primer ejemplo es el efecto 'Heat', aplicable a las funciones "animaText", "animaSegment" y "animaFunction". Consiste en hacer aparecer gradualmente el elemento en cuestión (un texto, un segmento o una función respectivamente) gracias la intensificación de su color. Para ello se asemeja la función del color del elemento ("iColor") a una función lineal. Comparando la fórmula de una función lineal general con la aplicada en el código se tiene:

Fórmula general función lineal	$y = n + mx$
Fórmula empleada en MATLAB	`for iFrame = 1:s.Frames` `iColor = s.Param + (s.TC-s.Param)*iFrame/s.Frames;`

Tabla 4. Comparación entre la fórmula general de una función lineal y el código que la implementa el cambio de color en MATLAB.

Observando la *Tabla 4*, se deduce que el argumento "y" equivale a la variable "iColor", el argumento "x" equivale a la variable "iFrame", la ordenada en el origen "n" equivale a la variable "s.Param" y la pendiente "m" equivale a la expresión "(s.TC - s.Param)/s.Frames". De esta manera, se consigue que, para cada una de las veces que itera el bucle—tantas veces como "s.Frames" indique—, la variable "i.Frame" tome un valor diferente, variando así el valor de "iColor", que representa el color del elemento. El resultado de "iColor" es un vector 1x3 en código RGB.

Téngase en cuenta que el código de la *Tabla 4* es el código concreto de la función

"animaText". Para el caso de "animaFunction" sería diferente ya que no existe un campo "Frames", esta variable queda sustituida por el número de puntos de puntos de la función: "length(x)".

El segundo ejemplo es el efecto 'Zoom', aplicable a las funciones "animaText" y "animaSegment". Consiste en hacer aparecer el elemento en cuestión (un texto o un segmento) mientras se va haciendo cada vez más grande, es decir, va aumentando de tamaño desde que prácticamente no se aprecia hasta su tamaño final. Para ello se asemeja la función del tamaño del elemento ("fsi") a una función lineal. Comparando la fórmula de una función lineal general con la aplicada en el código se tiene:

Fórmula general función lineal	$y = n + mx$
Fórmula empleada en MATLAB	`for iFrame = 1:s.Frames` `  fsi = s.FS*iFrame/s.Frames;`

Tabla 5. Comparación entre la fórmula general de una función lineal y el código que la implementa el cambio de tamaño en MATLAB.

Observando la *Tabla 5*, se deduce que el argumento "y" equivale a la variable "fsi", el argumento "x" equivale a la variable "iFrame", la ordenada en el origen "n" vale 0 y la pendiente "m" equivale a la expresión "(s.FS/s.Frames)". De esta manera, se consigue que, para cada una de las veces que itera el bucle—tantas veces como "s.Frames" indique—, la variable "fsi", que representa el tamaño del texto, vaya aumentando su valor. El resultado de "fsi" es un número real positivo ya que los dos valores que determinan la pendiente de la recta, "s.FS" y "s.Frames", también lo son.

4.6. Grabación del discurso oral

En los inicios de este trabajo, el objetivo final se debatía entre el hacer un mayor número de animaciones o hacer menos, pero de mayor calidad, lo cual conlleva crear vídeos con sonido, es decir, con una explicación de viva voz. Al final, se optó por lo segundo así que una vez se dispone de los diferentes módulos que conforman un vídeo en formato «.mp4», es momento de crear el discurso que se va a oír sobre el vídeo.

No se trata de una tarea fácil pues se ha de intentar compactar lo máximo posible la información para que no resulte pesado, pero a la vez se han de dar las suficientes explicaciones para que el mensaje sea transmitido exitosamente. Además, tampoco conviene extenderse mucho puesto que entonces quizás sea necesario alargar el vídeo.

El primer paso es subdividir los módulos en partes más pequeñas. El segundo paso es escribir un discurso parcial para cada una de las partes de modo que se siga el orden de lo mostrado en el vídeo, haciendo alguna puntualización respecto a lo que se enseña si cabe. El tercer paso es comparar las duraciones de los discursos parciales hablados con sus respectivas duraciones reales en el vídeo. Si ambas duraciones coinciden o son relativamente similares, no hay que hacer ningún cambio, pero si hay discordancia, se plantean dos opciones: se adapta (acorta o alarga) el discurso o, en caso de no ser posible porque ya contiene la información lo más compactada posible, se adapta el vídeo (acorta o alarga). En términos generales esta adaptación tiende a ser al alza, o sea, a “estirar” el vídeo por lo que se crea una función con tal utilidad.

La función se denomina «*stop_animation*» y consigue aumentar la duración al extender unos segundos una imagen, produciendo el efecto de “imagen congelada”. Esta función tiene dos argumentos de entrada: «sec» y «FR». El primero de ellos es el número de segundos que se desea alargar el vídeo y el segundo, los «*frames*» (fotogramas) por segundo, un parámetro que en todas las animaciones es *«myVideo.FrameRate».* Posee un código muy sencillo que se muestra a continuación:

```
function stop_animation(FR,sec)%FR is the frame rate
parametros;
nFrames = FR*sec;  %Number of frames in sec seconds
for i=1:nFrames
    frame = getframe;
    writeVideo(myVideo,frame);
end
```

El cuarto y último paso, una vez ejecutados los posibles cambios al vídeo, es grabar todos los discursos parciales por separado, pues si se graba todo el discurso de vez es muy probable que se cometan fallos. Uno de estos errores puede ser hablar a una velocidad demasiado rápida, con lo cual, el vídeo queda descuadrado respecto de la voz y se

producen silencios; otro error es, por el contrario, hablar demasiado lento, con lo cual, los cambios efectuados para compenetrar voz y vídeo son en vano; por último, no vocalizar o no entonar adecuadamente son otros de los errores más frecuentes. Es por esto que es fundamental crear discursos parciales y grabarlos también de manera separada.

4.7. Montaje final

El montaje final es una de las partes menos laboriosas pues si las anteriores se han completado eficazmente, no queda nada sino superponer el audio al vídeo. Para llevar a cabo dicha tarea, este proyecto se ha servido de un software abierto y gratuito, disponible en internet y llamado “CapCut” [2]. Es una aplicación para dispositivos móviles muy intuitiva y que permite el acople de vídeos y la posterior superposición de una “voz en off”, que es grabada por el teléfono en la propia aplicación.

Esta aplicación guarda el vídeo final montado en formato «.mov», pese a que los vídeos que se han acoplado sean «.mp4».

5. Resultados

El resultado de este TFG son vídeos explicativos sobre el contenido de ciertas prácticas realizadas con MATLAB, con lo que se pretende dotarlas de un soporte audiovisual. Todas las mencionadas prácticas son del grado en Ingeniería en Tecnologías Industriales de la ETSEIB.

En cuanto a sobre qué prácticas se han realizado las animaciones, se ha de mencionar la intención de abarcar tantas asignaturas de matemáticas como fuera posible. De este modo, tanto la animación "Function Transformations" como "Bisection Method", forman parte del contenido didáctico de "´Cálculo I"; "Juego de la vida" corresponde a una práctica de "Álgebra lineal" y, por su parte, la animación "Predator Prey" pertenece a la asignatura de "Ecuaciones diferenciales".

5.1. Retos técnicos

Uno de los principales retos a los que se enfrentaba este trabajo era el ajuste de los tiempos y las transiciones. Cuando se ejecuta un código, el resultado visto desde MATLAB no es para nada igual a la animación que se visualiza posteriormente en el vídeo «.mp4» creado.

En primer lugar, cabe comentar que uno de los parámetros del archivo «parametros» es «*saveVideo*», el cual hace referencia al deseo de generar el archivo «.mp4» o no. En consecuencia, si se ajusta el valor de esta variable a 0, el vídeo no se genera y, por tanto, MATLAB va a tener menor carga de trabajo y la animación va a ser más rápida. Por el contrario, si se ajusta esta variable a su valor 1, el vídeo sí se genera y MATLAB va a reproducir la animación de manera considerablemente más lenta.

En segundo lugar, ciertos comandos tales como el «*pause*» o el «*drawnow*» ayudan a dar formato de animación a una figura creada con MATLAB, es decir, conceden un cierto tiempo entre secuencias consecutivas y consiguen que estas se vean de manera continua, sin saltos. No obstante, esto solo ocurre cuando la animación es visualizada desde el propio programa, no si se visualiza el vídeo «.mp4». Para conseguir esta continuidad también en el vídeo «.mp4», se ha de recurrir al comando «*getFrame*», el cual se podría decir que hace una captura de pantalla del fotograma de ese instante por lo que, encadenando múltiples veces este comando, se consigue la apariencia de una animación.

Otra de las dificultades que se han encontrado a lo largo del proyecto, es el ajuste de la dimensión de los píxeles de una imagen. Se trata de uno de los aspectos más delicados durante la creación de vídeos ya que para grabar varios «frames» en un mismo fichero,

estos han de ser exactamente del mismo tamaño en píxeles. El problema aparece, por ejemplo, cuando se desea hacer zoom a la pantalla o cuando se cambian las dimensiones de una cuadrícula ya que, a pesar de que se mantiene en todo momento la proporción 16:9, en ciertas ocasiones MATLAB cambia las dimensiones de los píxeles de los "frames" consecutivos, que varían dentro de un rango muy pequeño—siempre manteniendo la proporción—por tanto, se debe ser muy meticuloso.

5.2. Animación "Transformación de funciones"

En este vídeo se quiere mostrar cómo cambia una función cuando modificamos sus variables mediante la suma o el producto de un parámetro [4].

5.2.1. Base teórica

Partiendo de una función cualquiera, tómese $y = f(x) = a\cdot\sin(\omega_1\cdot x) + b\cdot\cos(\omega_2\cdot x)$ como referencia (ver *Figura 1*), el vídeo pretende aclarar visualmente los cambios que esta sufre al aplicársele modificaciones, bien sean sobre la variable "*y"* o sobre la variable *"x"*, es decir, el vídeo trata sobre las distintas transformaciones que se pueden llevar a cabo sobre una función.

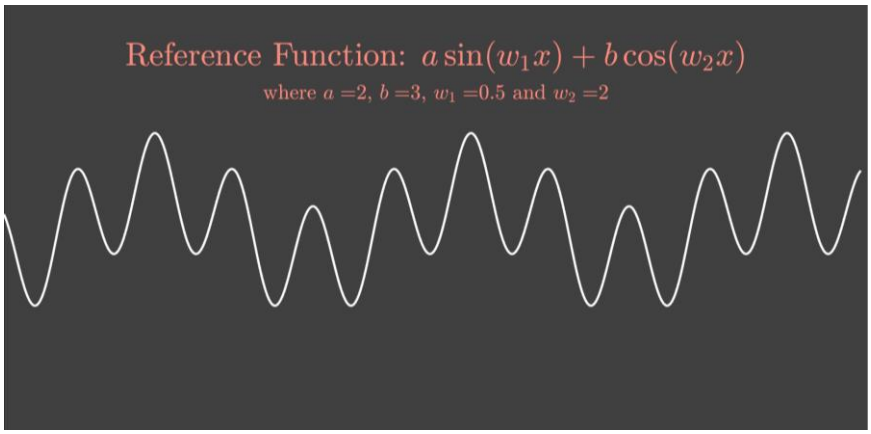

Figura 1. Función de referencia.

Se ha elegido esta función porque, estéticamente hablando queda bien. Es decir, es más llamativa que, por ejemplo, una recta y, lo más importante, abarca una parte considerable de la pantalla por lo que la animación no resulta "pobre", vacía o rígida.

5.2.2. Transformaciones sobre la "y"

Las transformaciones que afectan a la variable "*y"* producen una modificación en el eje vertical. Se detallan a continuación, son 5:

$y = f(x) + \alpha$ Ecuación 1

$y = f(x) - \alpha$ *Ecuación 2*

$y = \beta f(x)$ *Ecuación 3*

$y = \frac{f(x)}{\beta}$ *Ecuación 4*

$y = -f(x)$ *Ecuación 5*

La transformación de la

$y = f(x) + \alpha$ *Ecuación 1* produce un desplazamiento vertical positivo de la función inicial. Es positivo puesto que se ha considerado que la constante α también lo es. De este modo, cuanto mayor sea la constante α, más amplio será el desplazamiento. En el fotograma de la *Figura 2* se observa un ejemplo para un valor de α igual a 5.

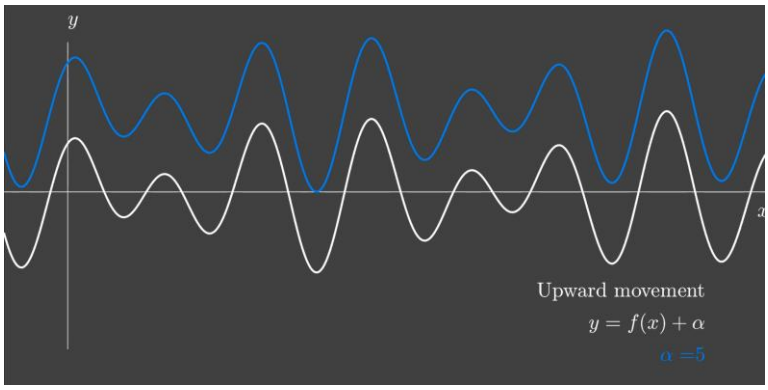

Figura 2. Transformación sobre la "y". Desplazamiento vertical positivo.

Por su parte, cuando la transformación es restando la constante en lugar de sumando ($y = f(x) - \alpha$ *Ecuación 2*), el desplazamiento resultante es en sentido opuesto, o sea, vertical negativo. Véase un ejemplo en el fotograma de la *Figura 3* para el mismo valor de α que anteriormente.

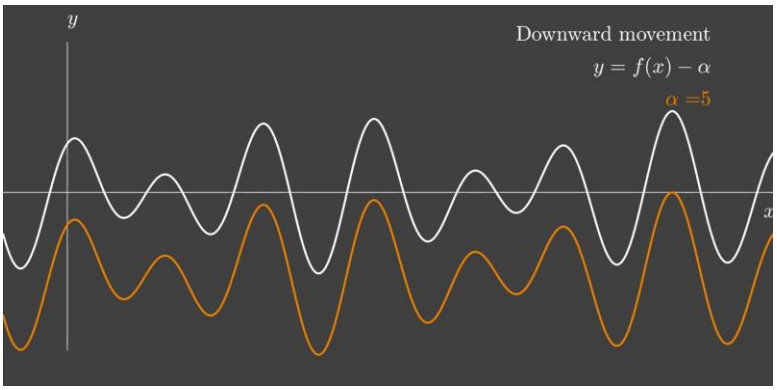

Figura 3. Transformación sobre la "y". Desplazamiento vertical negativo.

En tercer y cuarto lugar, se presentan las transformaciones $y = \beta f(x)$ *Ecuación 3* e $y = \frac{f(x)}{\beta}$ *Ecuación 4*. La primera de ellas conlleva un estiramiento de la función de partida en dirección vertical, puesto que como se ha dicho las transformaciones sobre la variable del eje de ordenadas tienen efecto sobre este mismo eje. La segunda de ellas, propicia que la función de referencia, y = f(x), se contraiga en dirección vertical. En ambos casos se ha considerado que el parámetro β es positivo y mayor que la unidad. En los fotogramas de la *Figura 4* y de la *Figura 5* se muestra respectivamente el resultado de estas transformaciones para un valor del parámetro β igual a 2.

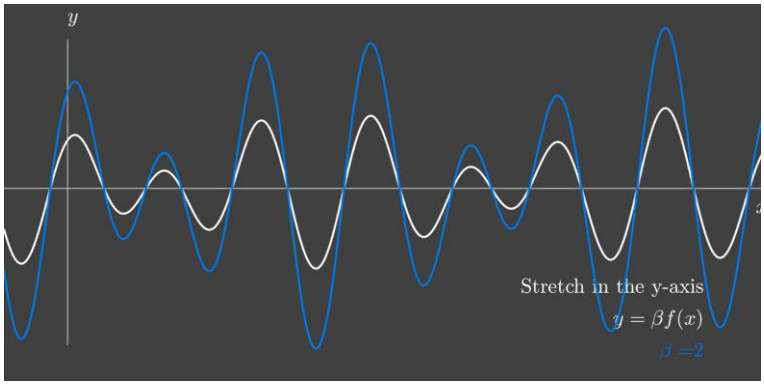

Figura 4. Transformación sobre la "y". Expansión en el eje vertical.

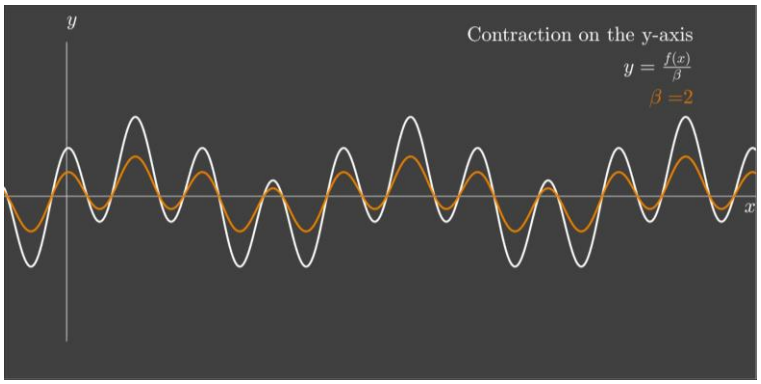

Figura 5. Transformación sobre la "y". Contracción en el eje vertical.

Por último, se trata el caso de $y = -f(x)$ *Ecuación 5*, que no es sino un caso particular de la transformación número 3 en la que el parámetro β toma un valor negativo e igual a la unidad, lo cual efectúa sobre la función inicial una reflexión vertical, es decir, una simetría respecto al eje de abscisas, obsérvese este efecto en la *Figura 6*.

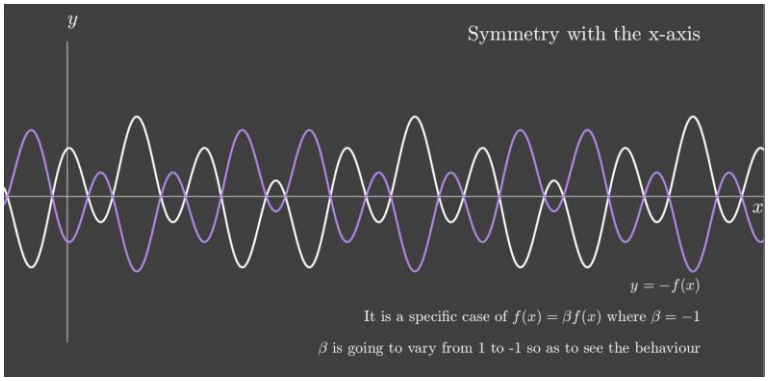

Figura 6. Transformación sobre la "y". Reflexión respecto al eje de abscisas.

5.2.3. Transformaciones sobre la "x"

Una vez vistas las transformaciones que afectan a la variable "*y*", se procede a explicar las mismas, pero para la variable "*x*", las cuales ocasionan una modificación en el eje horizontal. Son 5 y se detallan a continuación:

$y = f(x+\alpha)$ *Ecuación 6*

$y = f(x-\alpha)$ *Ecuación 7*

$y = f(\beta \cdot x)$ *Ecuación 8*

$$y = f\left(\frac{x}{\beta}\right)$$ *Ecuación 9*

$$y = f(-x)$$ *Ecuación 10*

La transformación de la $y = f(x+\alpha)$ *Ecuación 6*, al contrario de lo que podría parecer debido a que la constante α está sumándose, provoca un desplazamiento no en sentido positivo sino negativo. Por supuesto, como ya se ha mencionado, este movimiento es en la dirección del eje horizontal. Así pues, cuanto mayor sea el valor del parámetro α, mayor desplazamiento habrá. En el fotograma de la *Figura 7* se observa un ejemplo para un valor de α igual a 5.

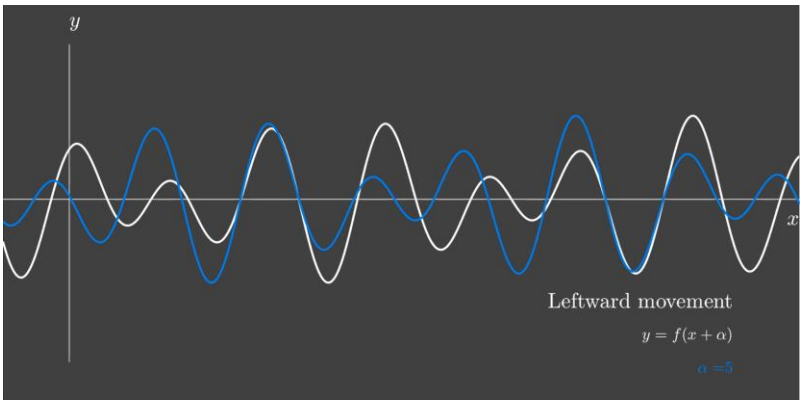

Figura 7. Transformación sobre la "x". Desplazamiento horizontal hacia la izquierda.

Por su parte, cuando la transformación es la misma, pero restando ($y = f(x-\alpha)$ *Ecuación 7*), el desplazamiento resultante es en sentido opuesto, o sea, horizontal positivo. Véase un ejemplo en el fotograma de la *Figura 8* para el mismo valor de α que anteriormente.

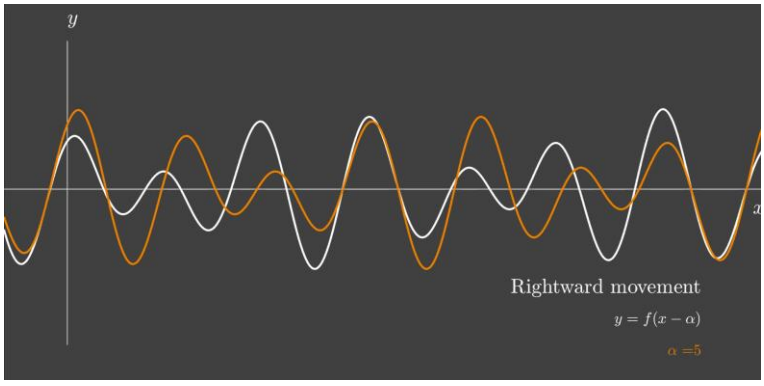

Figura 8. Transformación sobre la "x". Desplazamiento horizontal hacia la derecha.

En tercer y cuarto lugar, se presentan las transformaciones $y = f(\beta \cdot x)$ *Ecuación 8* y $y = f\left(\frac{x}{\beta}\right)$ *Ecuación 9*. La primera de ellas conlleva una contracción en la dirección horizontal, cuanto mayor es β, más "chafada" queda la función respecto a la función de referencia. Téngase en cuenta que se han considerado valores positivos y superiores a 1 para el parámetro β. La segunda transformación es similar solo que esta vez el parámetro β se encuentra dividiendo. Se dice similar porque dividir para números positivos y mayores que uno no es sino multiplicar por números decimales inferiores a 1. El resultado provocado entonces es la expansión de la función a lo largo del eje de abscisas; si se visualizara la animación, el efecto es el de una cuerda que se va estirando hasta llegar casi a ser una línea horizontal. En los fotogramas de la *Figura 9* y de la *Figura 10* se muestra respectivamente el resultado de estas transformaciones para un valor de β igual a 2.

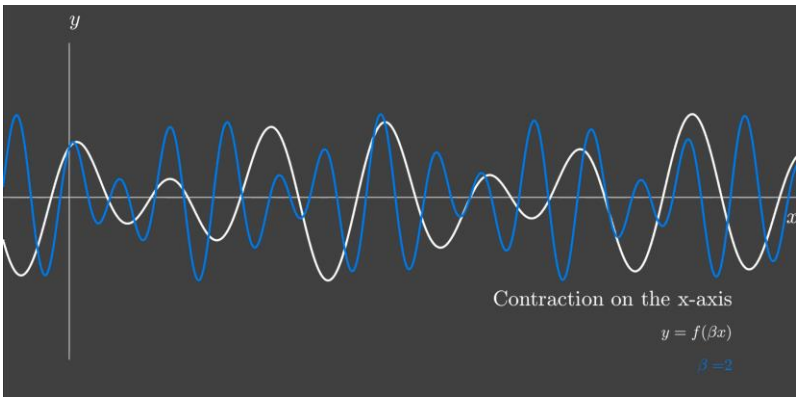

Figura 9. Transformación sobre la "x". Contracción en el eje horizontal.

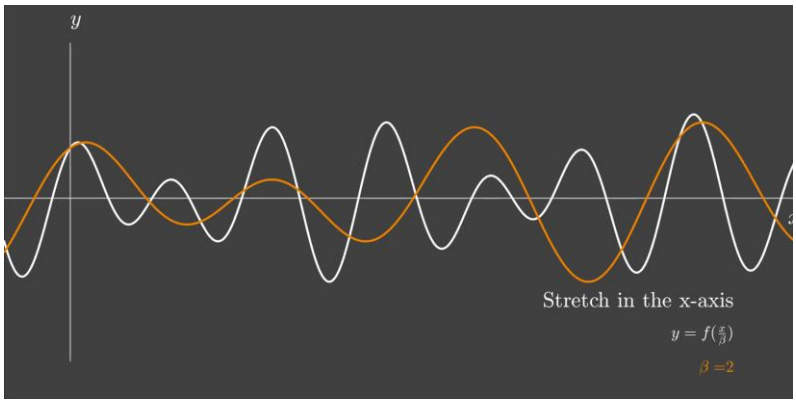

Figura 10. Transformación sobre la "x". Extensión en el eje horizontal.

En última instancia, se tiene el caso de $y = f(-x)$ *Ecuación 10*. De igual manera que para la transformación sobre la "*y*", se trata de un caso concreto de la transformación número 8, en la cual β vale -1. Así pues, aplicando esta última transformación a la función de referencia

lo que se obtiene es su simétrica respecto del eje de ordenadas. Esto puede verse en la *Figura 11*.

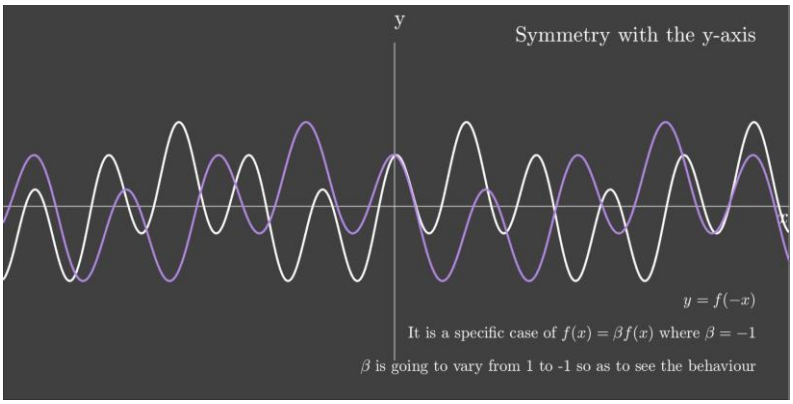

Figura 11. Transformación sobre la "x". Reflexión respecto al eje vertical.

5.3. Animación "Juego de la vida"

5.3.1. Base teórica

El «Juego de la Vida» [5] es un juego autómata de cero jugadores que tiene lugar en una cuadrícula 2D infinita. Esta cuadrícula está dividida en celdas cuadradas, cada una de las cuales puede encontrarse en dos estados: viva o muerta. Su estado va a depender del estado de sus celdas adyacentes, denominadas vecinas.

5.3.2. Celdas adyacentes

Cada celda tiene un total de 8 celdas vecinas, todas las que están a su alrededor formando un cuadrado más grande. Esto posee una particularidad, y es que las celdas vecinas de las situadas en los bordes o las esquinas pueden no ser intuitivas. El universo original concebido por el creador es infinito pero un ordenador no puede simularlo. Por tanto, nos autolimitamos a un universo finito, pero lo suponemos toroidal porque entonces no tiene "fronteras" pese a ser finito. Así pues, en los fotogramas de la *Figura 12*, *Figura 13* y *Figura 14* se ejemplifican dichas situaciones, es decir, se muestran cuáles son las 8 celdas vecinas en estos casos concretos.

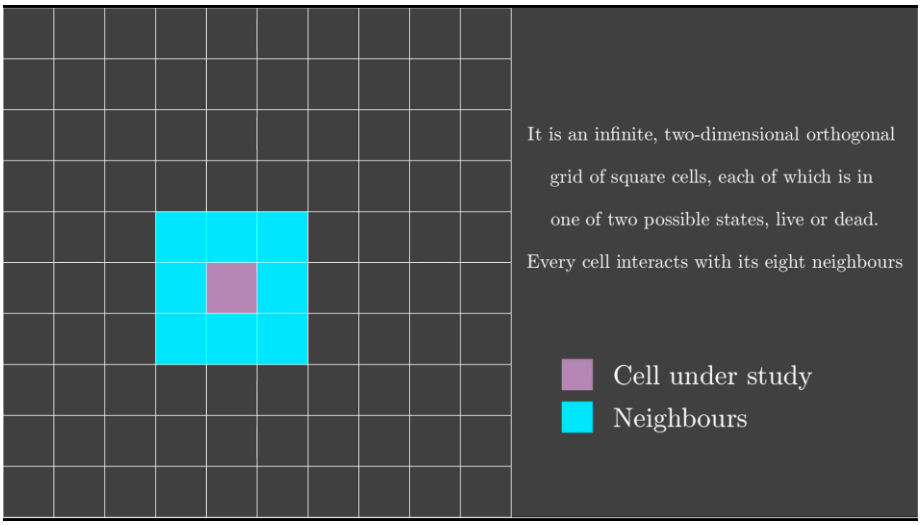

Figura 12. Celdas vecinas. Caso general.

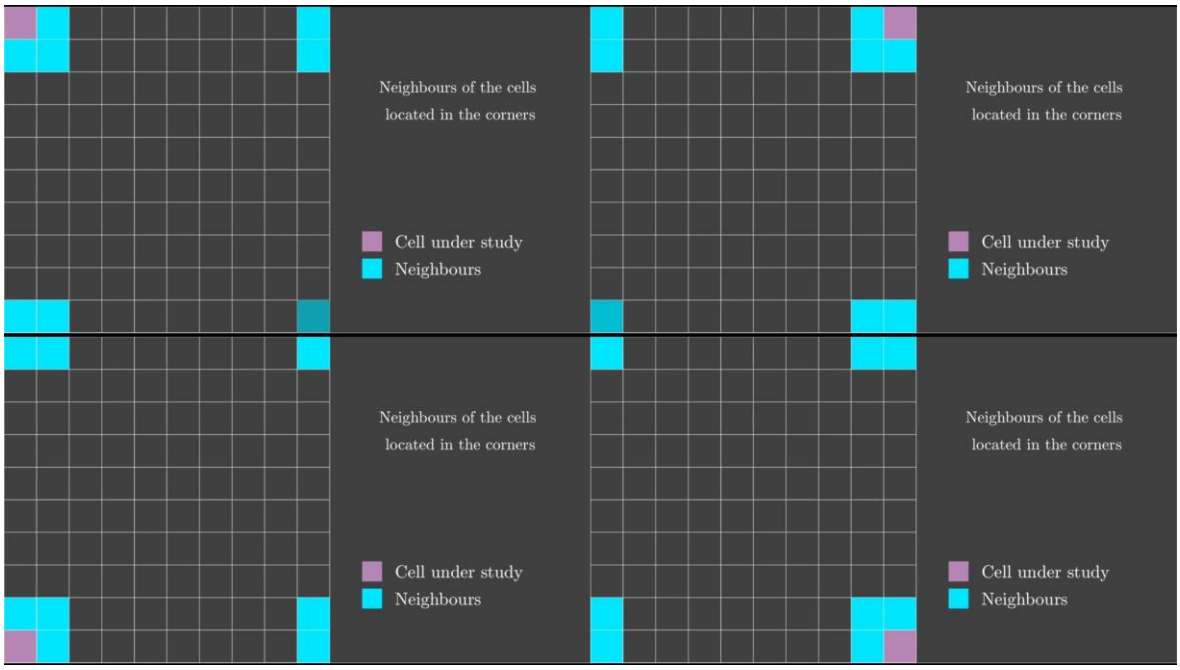

Figura 13. Celdas vecinas. Caso esquina superior izquierda, superior derecha, inferior izquierda e inferior derecha.

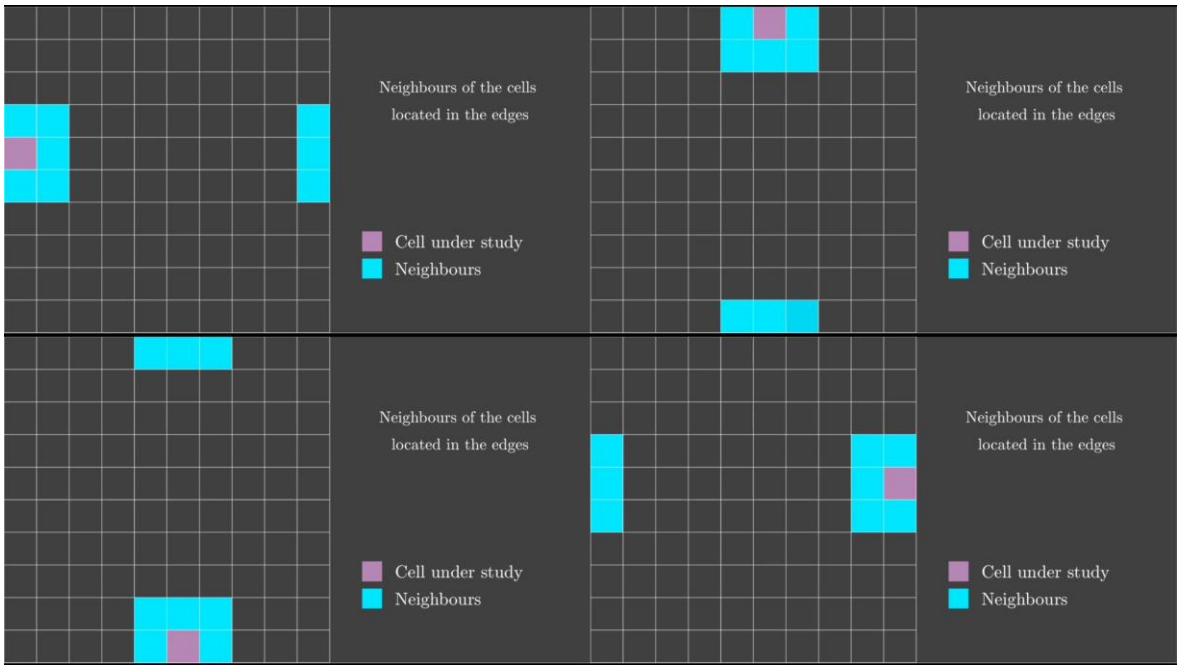

Figura 14. Celdas vecinas. Caso general borde izquierdo, borde superior, borde inferior y borde derecho.

5.3.3. Reglas del juego

El funcionamiento del juego es el siguiente: partiendo de un estado inicial de celdas vivas y muertas, que puede ser generado aleatoriamente o no, como se verá más adelante, se van sucediendo las generaciones y el aspecto de la cuadrícula evoluciona infinitamente de acuerdo a una serie de leyes. Esta evolución puede seguir un patrón, periódico o no, o puede tener un comportamiento completamente impredecible en su defecto.

En MATLAB, hablar de un estado inicial de celdas vivas o muertas equivale a hablar de una matriz de determinadas dimensiones—según el tamaño de la cuadrícula—cuyos elementos pueden tomar el valor 1 (celda viva) o el valor 0 (celda muerta).

Para entender de qué manera las celdas cambian de estado se procede a explicar las normas que rigen su comportamiento, son 4:

1) Una celda viva en el instante (t-1) con menos de dos celdas vecinas vivas en el instante (t-1) pasará a estar muerta en el instante t.

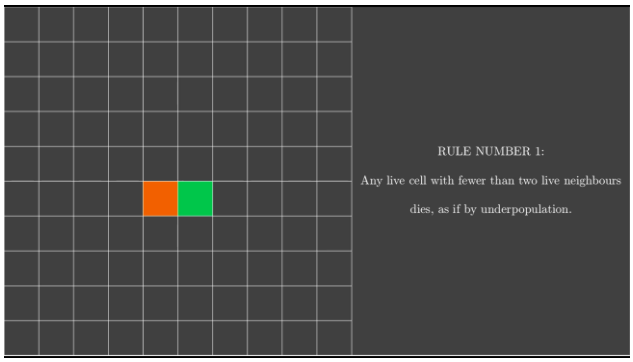

Figura 15. Representación de la regla número 1.

2) Una celda viva en el instante (t-1) con dos o tres celdas vecinas vivas en el instante (t-1), permanecerá viva también en el instante t.

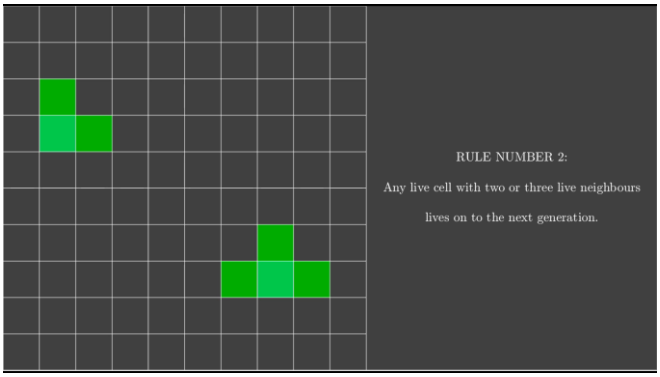

Figura 16. Representación de la regla número 2.

3) Una celda viva en el instante (t-1) con más de tres celdas adyacentes vivas en el instante (t-1) pasará a estar muerta en el instante t.

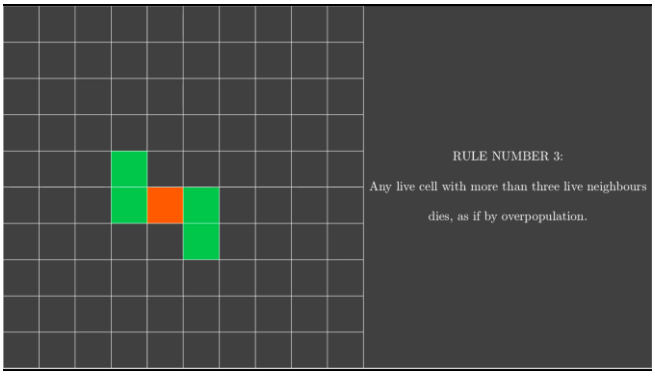

Figura 17. Representación de la regla número 3.

4) Una celda muerta en el instante (t-1) con exactamente tres celdas vecinas vivas en el instante (t-1) pasará a estar viva en el instante t.

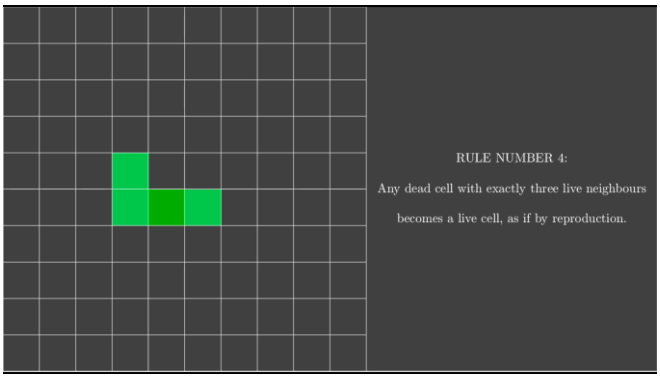

Figura 18. Representación de la regla número 4.

Cabe resaltar que el juego se desarrolla durante un tiempo discretizado, aunque en la animación no se habla de tiempo sino generaciones. Estas aparecen a modo de contador en la zona derecha de la pantalla y se especifican por números enteros positivos, de tal modo que si el instante (t-1) está representado por la generación número 23, por ejemplo, el instante t es la generación número 24.

Las leyes arriba descritas están concebidas y pensadas para describir los procesos básicos de la vida y la muerte. Es por ello que la regla número 1 representa la “muerte por despoblación”, la regla número 2 representa la “vida sostenible”, la regla número 3 representa la “muerte por sobrepoblación” y la regla número 4 representa el “nacimiento”.

5.3.4. Estados iniciales especiales

Como se ha mencionado en el apartado 5.3.3, el estado inicial desde el cual el juego se desarrolla de manera autónoma se puede generar aleatoriamente o no. Entre los patrones comunes se incluyen los estáticos, los oscilantes o las naves espaciales, que se trasladan cruzando toda la cuadrícula y dando muestra de la naturaleza toroidal del juego. En este apartado pues se enseñan las múltiples disposiciones iniciales que se han tratado en el vídeo y que evolucionan de manera particular.

1. Ejemplo *"glider*": este patrón da muestra del comportamiento toroidal programado puesto que el elemento que cruza la cuadrícula diagonalmente llega al límite lateral derecho y reaparece por el izquierdo.

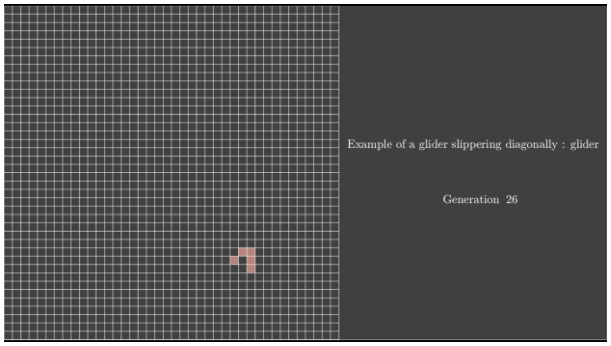

Figura 19. Respresentación del estado inicial "glider".

2. Ejemplo "*gun*": este patrón pertenece al grupo de las "naves espaciales" y debe su nombre, que en inglés significa "arma", a que dispara una serie de elementos.

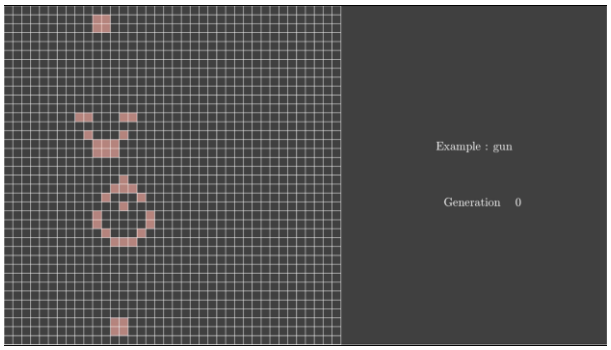

Figura 20. Representación del estado inicial "gun".

3. Ejemplo "*oscillatory with period 2*": este patrón pertenece al grupo de los oscilantes ya que tiene un periodo de 2, esto es, cada 2 generaciones se vuelve a repetir la disposición inicial.

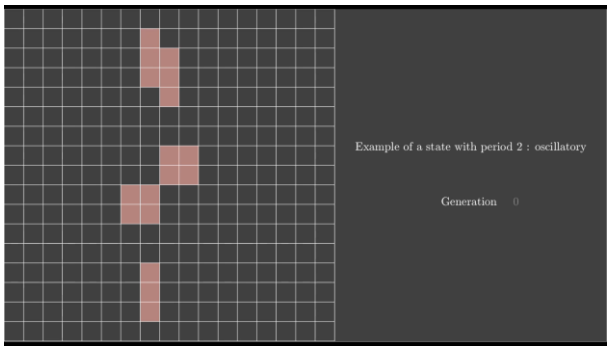

Figura 21. Representación del estado inicial "oscillatory with period 2".

4. Ejemplo "*pulsar with period 3*": este patrón pertenece al grupo de los oscilantes ya que tiene un periodo de 3 generaciones.

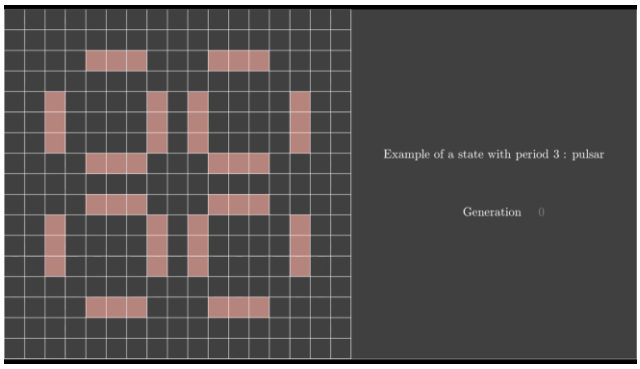

Figura 22. Representación del estado inicial "oscillatory with period 3".

5. Ejemplo "*pentadecathlon with period 15*": este patrón pertenece al grupo de los oscilantes ya que tiene un periodo de 15 generaciones.

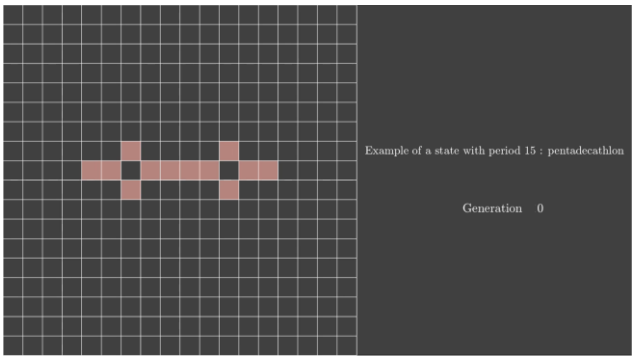

Figura 23. Representación del estado inicial "pentadecathlon".

6. Ejemplo "steady": este patrón corresponde a un patrón estático en el cual no importa por cuanto tiempo o cuantas generaciones se desarrolle el juego ya que la disposición se va a mantener igual permanentemente.

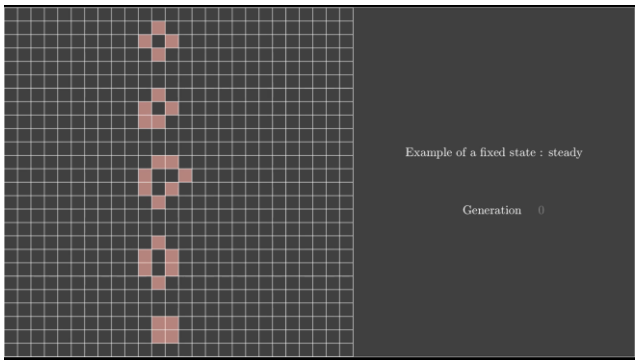

Figura 24. Representación del estado inicial "steady".

5.4. Animación “Depredador presa”

5.4.1. Base teórica

El modelo depredador-presa [6], [7], es un modelo general de ecuaciones diferenciales que describe la dinámica de sistemas biológicos como puede ser la interacción entre dos poblaciones de especies diferentes, la de depredadores y la de presas. Este trabajo en concreto se sirve de las ecuaciones Lotka-Volterra, que representan el modelo más simple.

El modelo Lotka-Volterra, que constituye uno de los primeros modelos matemáticos sobre la ecología, se compone de dos ecuaciones diferenciales no lineales y de primer orden que describen la evolución a lo largo del tiempo en las poblaciones de ambas especies. Por tanto, considerando dos poblaciones cuyas medidas para un determinado instante de tiempo t son y(t) y x(t) se tiene:

$$\begin{cases} \frac{dx}{dt} = \alpha x - \beta xy \\ \frac{dy}{dx} = \delta xy - \gamma y \end{cases}$$ *Ecuación 11*

Donde:

t ≡ tiempo

x ≡ población de presas, considerada una función continua.

y ≡ población de depredadores, considerada una función continua.

dx/dt ≡ tasa de crecimiento instantáneo de la población de presas.

dy/dt ≡ tasa de crecimiento instantáneo de la población de depredadores.

α ≡ tasa de crecimiento de la especie x (presas) en ausencia de interacción con la especie y (depredadores).

β ≡ tasa de impacto de la depredación en las presas x.

γ ≡ tasa de muerte o emigración de la especie y (depredadores) en ausencia de interacción con la especie x.

δ ≡ tasa neta de crecimiento o inmigración de la población de depredadores como respuesta al tamaño de la población de presas.

α, β, γ, δ ≥ 0 y ϵ R

Se trata de un modelo teórico que realiza una serie de suposiciones sobre el ecosistema y la evolución de ambas poblaciones que pueden no ser totalmente verídicas en la naturaleza:

1. La población de presas dispone de abundante comida incesantemente.
2. Los depredadores se nutren únicamente de las presas, o sea, el abastecimiento de comida de estos únicamente depende del tamaño de la población de presas.
3. Los ritmos de cambio de las poblaciones son directamente proporcionales a sus tamaños.
4. El ecosistema no varía a favor de una de las dos especies a lo largo del proceso.
5. Los depredadores tienen un apetito infinito.

Una vez explicados los parámetros y las suposiciones, se puede entender mejor el significado de cada una de las ecuaciones diferenciales. De este modo, $\frac{dx}{dt} = \alpha x - \beta xy$ viene a decir que el ritmo de cambio de la población de presas queda determinado por su propia tasa de crecimiento menos la tasa que sufre por ser la presa. A su vez, $\frac{dy}{dx} = \delta xy - \gamma y$, establece que el ritmo de cambio de la población de depredadores depende de la tasa a la que consume presas—o lo que es lo mismo, depende del tamaño de la población de presas—menos su tasa propia de mortalidad.

Las soluciones a estas ecuaciones diferenciales presentan periodicidad y a continuación se muestra el ejemplo tratado en el vídeo, con unos valores de α = 0.9, β = 1, γ = 1.1 y δ = 1.

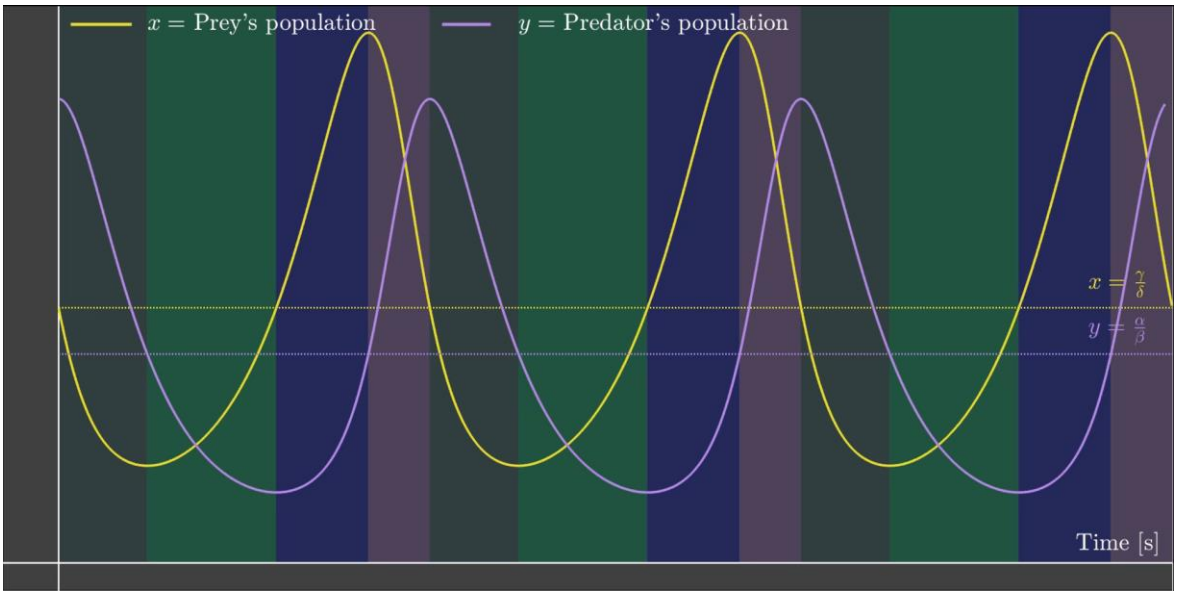

Figura 25. Series temporales del número de depredadores y presas.

Las líneas horizontales representan uno de los dos puntos de equilibrio que posee el sistema, allí donde ninguno de los dos niveles de población cambia, es decir, cuando ambos valores de las derivadas, x' e y', son iguales a 0. Además del punto de equilibrio mostrado en el fotograma y en la *Ecuación 12*, existe un segundo punto de equilibrio que no aparece dibujado y es el mostrado por la *Ecuación 13*.

$$\left\{ y = \frac{\alpha}{\beta}, x = \frac{\gamma}{\delta} \right\}$$ *Ecuación 12*

$$\{ y = 0, x = 0 \}$$ *Ecuación 13*

La primera ecuación representa un nivel fijo al cual las dos poblaciones mantienen su número de individuos (diferente de cero) indefinidamente mientras que la segunda ecuación representa la extinción de ambas especies y es lógica puesto que, si ambas especies se encuentran a un nivel nulo, van a seguir estando a este nivel.

La solución ya mostrada es la de las series temporales, pero también se puede representar el diagrama del espacio de fase.

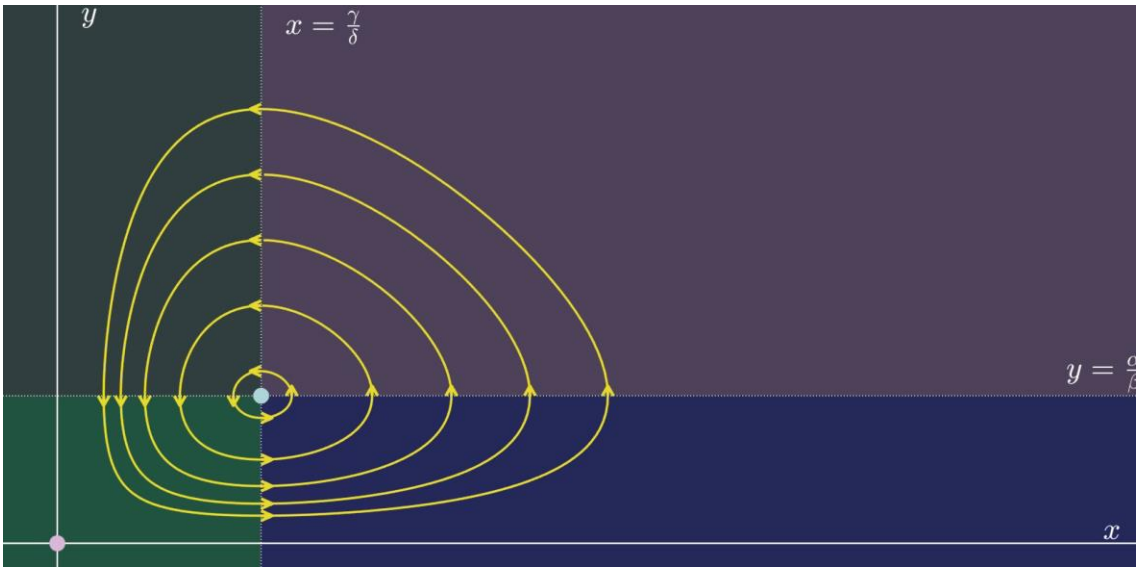

Figura 26. Retrato de fase del sistema depredador-presa

La *Figura 26* muestra la evolución de las poblaciones de ambas especies para diferentes condiciones iniciales, o sea, para diferentes poblaciones iniciales tanto de presas como de depredadores. La dinámica general es que los depredadores prosperan cuando hay abundantes presas, pero como tienen un apetito infinito, finalmente su demanda/consumo acaba superando la oferta de alimento (presas) y entonces la población de depredadores disminuye. Cuando la población de depredadores es baja, la población de presas vuelve a aumentar pues ya no hay tantos individuos por encima de ellas en la cadena alimenticia y el ciclo comienza de nuevo. Así pues, esta dinámica se repite sucesivamente propiciando etapas de crecimiento y declive en ambas especies

Cada uno de los cuadrantes, noroeste, suroeste, sureste y noreste representan las diferentes etapas de un ciclo.

- Cuadrante noroeste: ambas especies disminuyen su población
- Cuadrante suroeste: la población de depredadores sigue en declive mientras que la de presas comienza a aumentar.
- Cuadrante sureste: la población de presas continúa al alza y la de depredadores también se empieza a ver incrementada.
- Cuadrante noreste: la población de depredadores aumenta mientras que la de presas inicia su descenso.

5.5. Animación “Método de bisección”

5.5.1. Base teórica

El método de bisección [8],[9], es uno de los métodos empleados en matemáticas para hallar las raíces de una función. Solo es aplicable a funciones que cumplen determinadas condiciones tales como ser continua en el intervalo de estudio y tener dos valores conocidos y de signo opuesto. El método se basa en el Teorema de Bolzano, que asegura que siendo f una función continua en un intervalo cerrado [a, b] y que toma valores de signo contrario en los extremos, entonces existe, al menos, un valor c ϵ [a, b] tal que f(c)=0.

Así pues, el método consiste en dividir en dos repetidamente el intervalo definido por estos dos valores conocidos y de signo contrario, según se muestra en la *Ecuación 14*, y entonces elegir el subintervalo en el cual la función cambia de signo y que, por tanto, contendrá una de las raíces de la función. Por ejemplo, tras la iteración de la *Figura 27,* el intervalo escogido es el [c, b] ya que son los valores de signo opuesto. Las iteraciones necesarias para que la solución converja dependen de la tolerancia que se desee obtener. Normalmente la tolerancia se calcula como lo indicado en la *Ecuación 15* pero, en general, es más preciso utilizar la *Ecuación 16.*

$$c = \frac{(a+b)}{2} \qquad \textit{Ecuación 14}$$

$$\left| f(x) \right| < tol \qquad \textit{Ecuación 15}$$

$$\left| x_{n+1} - x_n \right| < tol \quad \textit{Ecuación 16}$$

Es un método sencillo y robusto, pero también bastante lento. Tanto es así que, en términos generales, se usa como herramienta para obtener una aproximación a la solución que sirva de punto de partida para otros métodos de convergencia más rápida.

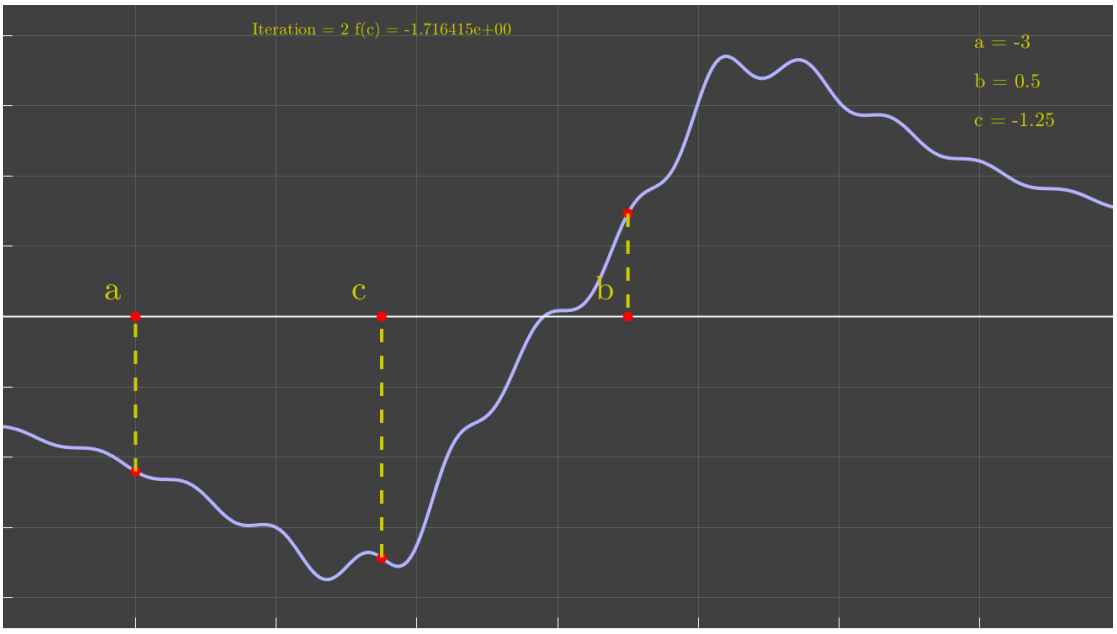

Figura 27. Iteración del método de bisección.

6. Gestión económica

En este apartado se analiza el impacto económico del proyecto. El presupuesto tiene en cuenta los diferentes recursos empleados a lo largo de su desarrollo, desde los recursos informáticos de software y hardware hasta los recursos humanos necesarios, además de otros costes indirectos. No se contemplan otras partidas económicas habituales en un presupuesto tales como el coste de producción o de comercialización puesto que el contenido creado no ha presentado dichas etapas.

6.1. Recursos utilizados

En cuanto a los recursos informáticos, las herramientas hardware utilizadas son un ordenador portátil Acer Swift SF314-51 (procesador Intel ® Core ™ i5-7200U CPU @ 2.50GHz 2.70GHz) y un dispositivo móvil Apple Iphone SE 128 GB. La vida útil de estos dispositivos es de 5 y 3 años respectivamente por lo que sus costes para el cálculo del presupuesto en la *Tabla 6* se verán divididos entre estos periodos y multiplicados por los meses que ha durado el proyecto.

Por su parte, las herramientas software utilizadas son MATLAB versión R2020b, *Google Meet*, *Google Drive*, *Microsoft office Word* y la aplicación móvil gratuita de edición de vídeo "*CapCut*" [2]. Estas aplicaciones/programas de software no han supuesto ningún coste dado que la UPC ofrece sus licencias gratuitamente a los estudiantes—de todas las mencionadas excepto "CapCut", que de todas maneras es gratuita—.

6.1.1. Costes

Para el cálculo de las horas dedicadas por parte de la estudiante se ha tenido en cuenta la equivalencia de 1 crédito ECTS en horas: 1 ECTS = 25 horas. De este modo, si el TFG tiene una asignación de 12 ECTS, le corresponden 300 horas, añadiendo a ello 25 horas extra que se cree se han dedicado, resultan 325 horas en total dedicadas a la programación, búsqueda de información, montaje de los vídeos y desarrollo de la memoria.

Para el cálculo de las horas de los tutores se han tenido en cuenta las horas de las reuniones llevadas a cabo semanalmente vía *Google Meet*, las horas invertidas en la visualización y corrección de los vídeos, las horas empleadas en ayudar a programar y resolver dudas por mail y, finalmente, las horas de revisión de la memoria. Todas ellas se han estimado en 16,25 horas (13 semanas · 1,25 horas / (reunión · semana)), 8 horas, 10 horas y 6 horas respectivamente.

Además de los costes de recursos hardware, software y de recursos humanos, se han producido otros indirectamente: se trata de los costes derivados del consumo energético y de la conexión a internet.

Realicemos una estimación del consumo eléctrico. Teniendo en cuenta que el consumo medio de un ordenador portátil es 180 W, que se ha utilizado durante un total de unas 320 horas (se desprecia el consumo eléctrico del dispositivo móvil) y que el precio medio del kWh en España es de 0,0943 €:

$$320h \cdot 180W = 576000Wh = 57,6kWh \rightarrow 57kWh \cdot 0.0943 \frac{€}{kWh} = 5,43168€$$

El coste indirecto por consumo eléctrico del ordenador asciende a 5,43 €. A esto se le debe sumar el coste mensual de acceso a internet (30 €) durante los aproximadamente 4 meses que se ha extendido la realización del proyecto.

Concepto	Precio unitario	Cantidad	Total
Salario proyectista	12 €/h	325 horas	3.900,00 €
Salario tutores	35 €/h	40,25 horas	1.408,75 €
Total recursos humanos			**5.308,75 €**
Ordenador Acer Swift 3	700 €	1 unidad	47 €
Iphone SE 128 GB	539 €	1 unidad	60 €
Total recursos hardware			**106,56 €**
MATLAB 2020b	0 €	1 unidad	0 €
Capcut	0 €	1 unidad	0 €
Total recursos Software			**0 €**
Consumo eléctrico	0,0943 €/kWh	57,6 kW	5,43 €
Conexión a internet	30 €/mes	4 meses	120 €
Total costes indirectos			**125,43 €**
Total			**5.540,74 €**

Tabla 6. Presupuesto total del proyecto.

Así pues, tal y como se extrae de la *Tabla 6*, este proyecto ha tenido un coste total de 5540,74 €. Como se observa, la mayor parte de este presupuesto proviene de los costes en recursos humanos.

Se trata de un coste orientativo pues el consumo eléctrico es una estimación y tampoco se ha llevado un control exhaustivo de las horas de dedicación ya que el objetivo principal era desarrollar un buen trabajo y no cumplir con unas especificaciones de tiempo/dinero.

7. Impacto ambiental

Podría decirse que este proyecto ha tenido un impacto ambiental prácticamente nulo. Al tratarse de un proyecto cuya principal herramienta de trabajo es un software matemático, el único impacto que ello tiene es el consumo de luz por parte del hardware que lo contiene.

Conclusiones

Gracias a la realización de este trabajo se han adquirido nuevos conocimientos: conocimientos básicos de “LaTeX” y “CapCut” y conocimientos más avanzados de MATLAB.

En cuanto a los objetivos que tanto estudiante como tutores habían fijado, se puede decir que se han completado satisfactoriamente. En primer lugar, se han desarrollado herramientas que permiten crear animaciones científicas mediante funciones escritas en MATLAB. En segundo lugar, se han mostrado las capacidades de esas herramientas mediante la generación de tres vídeos completos de carácter didáctico y temáticas diferentes: Transformaciones de Funciones, El juego de la Vida y el modelo Depredador-Presa. Estos tres vídeos tratan materias de varias asignaturas del GETI, concretamente, Cálculo I, Álgebra Lineal y Ecuaciones Diferenciales. A su vez, se ha avanzado en la generación de otros vídeos, como el del Método de Bisección.

Tras la finalización de este proyecto se extrae una conclusión muy clara y es el gran potencial que tiene MATLAB. Ha quedado patente cómo mediante unas pocas funciones se pueden desarrollar proyectos mucho mayores, como son en este caso animaciones científicas de contenido matemático.

Además de potente, se trata de un programa muy completo pues ¿quién hubiera imaginado que con este software se pueden crear vídeos? Por tanto, este trabajo es innovador en tanto que demuestra que no solo es un programa de cómputo numérico, sino que ofrece otras muchas posibilidades que, quizás no se habían enseñado anteriormente.

También ha quedado patente el gran salto cualitativo que dan las animaciones al dotarlas de una explicación oral durante el proceso de posproducción, es decir, al añadir una “voz en off”. De esta manera, se facilita la comprensión de los vídeos.

Quisiera añadir también la gran aportación al desarrollo personal que en segundo plano lleva a cabo MATLAB pues, tras cada error de ejecución en los códigos—que no son pocos—te obliga a ser perseverante y no dejar de buscar posibles soluciones, ya sea programando de otra manera o buscando en foros.

En conclusión, este proyecto, que mezcla nociones de programación y aspectos de diseño, no solo ha cumplido con sus objetivos, sino que ha sentado las bases para una posible revisión de este mismo proyecto o para la realización de otros posteriores que sigan con la tarea de crear contenido visual de matemáticas en las carreras de ingeniería.

Agradecimientos

En primer lugar, quisiera dar las gracias a mis tutores, Rafa y Toni, por hacer posible este trabajo. Ellos han sido quienes me han guiado en todo momento, quienes han sabido establecer un objetivo desde el inicio y, sobre todo, quienes le han puesto imaginación. Sus ganas de llevar a cabo este proyecto y sus reacciones al ver cómo iba evolucionando han sido mi motivación en la práctica. Me gustaría especialmente agradecer a ambos el entusiasmo que demuestran por la docencia, intentando esforzarse cada día más por hacer los contenidos más interactivos y, en consecuencia, más didácticos.

Mis agradecimientos también van para mis compañeras y amigas del grado, ambas Martas, que me han enseñado a no compararme con los demás ni con sus proyectos pues cada estudiante es un mundo.

Por último, no quisiera acabar sin mencionar a mi familia, que es la que siempre me apoya y me da ánimos en los días menos productivos.

Bibliografía

Referencias bibliográficas

[1] JOHNSON, RICHARD, *Matlab Style Guidelines 2.0*. MATLAB central File Exchange. Marzo 2014.

[2] BYTEDANCE PTE. LTD, *CapCut – Video Editor.* 2021.

[https://apps.apple.com/us/app/capcut-video-editor/id1500855883]

[3] *Automat Cel·lular: El joc de la Vida.* Numerical Factory 2019.

[https://numfactory.upc.edu/web/Algebra/P2JuegoDeLaVida/html/JdV.html].

[4] FERNÁNDEZ, JOSÉ L., *Transformación de funciones.* Fisicalab.

[https://www.fisicalab.com/apartado/transformacion-funciones].

[5] IZHIKEVICH EUGENE M., CONWAY JOHN H., SETH ANIL, *Game of life.* Scholarpedia, 2015. [http://www.scholarpedia.org/article/Game_of_Life].

[6] HOPPENSTEADT, DR. FRANK, *Predator-Prey Model.* Scholarpedia, 2006. [http://www.scholarpedia.org/article/Predator-prey].

[7] *Lotka-Volterra equations.* Wikipedia, junio 2021

[https://en.wikipedia.org/wiki/Lotka%E2%80%93Volterra_equations].

[8] *Càlcul dels zeros d'una funció.* Numerical Factory 2020.

[https://numfactory.upc.edu/web/Calculo1/P2CerosFunciones/html/CalculZeros.html].

[9] *Bisection method.* Wikipedia, marzo 2021.

[https://en.wikipedia.org/wiki/Bisection_method].