LABORATORIO N° 06 - Simulación

Movimiento Armónico Simple (M.A.S.)

Apellidos y Nombres

Código: 182739
Curso: Física II - Unidad 3

DOCENTE: HUALLPA AIMITUMA Josué David

Abstract

En este laboratorio virtual se estudia el Movimiento Armónico Simple (M.A.S.) mediante la implementación paso a paso de las ecuaciones del movimiento en una simulación construida con Manim. Los estudiantes analizarán la relación entre las condiciones iniciales, la frecuencia natural, la amplitud, el desfase y la posición de equilibrio. Además, implementarán el modelo matemático $x(t) = A\cos(\omega_0 t - \delta)$ y explorarán gráficamente cómo las variaciones en los parámetros físicos afectan el comportamiento del sistema masa-resorte.

Contacto: *autor@unamba.edu.pe*
DOI: 10.1234/lab06
Recibido: 12 Nov 2025 **Aceptado:** 15 Nov 2025 **Publicado:** 20 Nov 2025

1. OBJETIVOS

1. Comprender las ecuaciones que describen el movimiento armónico simple (M.A.S.) a partir de las leyes de Newton.
2. Identificar la relación entre las condiciones iniciales y los parámetros característicos del sistema (A, ω_0, δ).
3. Implementar una simulación en Manim que represente el movimiento oscilatorio de un bloque unido a un resorte.
4. Analizar la correspondencia entre el modelo matemático y la animación obtenida.

2. FUNDAMENTO TEÓRICO

El movimiento armónico simple (M.A.S.) describe el movimiento de una partícula o cuerpo que oscila alrededor de una posición de equilibrio bajo una fuerza restauradora proporcional al desplazamiento.

2.1. Ecuación fundamental

Partiendo de la segunda ley de Newton:

$$F = ma = -kx \tag{1}$$

se obtiene la ecuación diferencial del movimiento:

$$m\frac{d^2x}{dt^2} + kx = 0 \tag{2}$$

La solución general tiene la forma:

$$x(t) = A\cos(\omega_0 t - \delta) \tag{3}$$

donde:

- A: amplitud de oscilación (máximo desplazamiento)
- $\omega_0 = \sqrt{\frac{k}{m}}$: frecuencia angular natural
- δ: desfase determinado por las condiciones iniciales

A partir de las condiciones iniciales $x(0) = x_0$ y $v(0) = v_0$, se tiene:

$$A = \sqrt{x_0^2 + \frac{v_0^2}{\omega_0^2}}, \qquad \delta = \tan^{-1}\left(\frac{v_0}{\omega_0 x_0}\right) \tag{4}$$

La energía mecánica del sistema se conserva:

$$E = \frac{1}{2}kA^2 = \frac{1}{2}kx^2 + \frac{1}{2}mv^2 \tag{5}$$

3. MATERIALES Y HERRAMIENTAS

- Entorno de programación **Python 3**.
- Librería **ManimCE** (Community Edition).
- Calculadora científica o simbólica (para derivar y comprobar ecuaciones).
- Bloc de notas para registro de resultados y observaciones.

Instalación y configuración de ManimCE

Para la implementación de la simulación del Movimiento Armónico Simple (M.A.S.) es necesario instalar y configurar un entorno de trabajo compatible con **Manim Community Edition**. A continuación se detallan los pasos recomendados para usuarios de Windows (también aplicables a Linux o macOS con ligeras variaciones):

Instalación de Miniforge (gestor de entornos Python)

Miniforge es una alternativa ligera a Anaconda y permite gestionar entornos aislados para proyectos científicos. Para descargarlo:

- Buscar en la web: ``Download Miniforge GitHub''.
- Seleccionar la versión: `Miniforge3-Windows-x86_64.exe`.
- Instalar aceptando las opciones por defecto.

Creación del entorno virtual

Abrir **Miniforge Prompt** y crear un entorno dedicado para ManimCE, considerar que ManimCE no siempre esta disponible para las versiones mas recientes de Python, a la fecha la versión mas reciente de Python es 3.14, pero esta es incompatible con la libreria actual de ManimCE, revisar la compatibilidad en la seción de instalación de la documentacion:

```
conda create -n manim-ce python=3.12
```

sigue las instruciones durante la creacion de el entorno virtual **manim-ce**, una vez finalizada activar el entorno con:

```
conda activate manim-ce
```

Es importante verificar que el nombre del entorno aparezca al inicio de la terminal, por ejemplo:

```
(manim-ce) C:\Users\usuario>
```

Instalación de ManimCE e IPython

Una vez activado el entorno virtual **manim-ce**, instalar la libreria ManimCE desde la misma terminal usadno el repositorio de conda-forge:

```
conda install -c conda-forge manim
```

Para habilitar el modo interactivo con OpenGL (recomendado para pruebas rápidas), se obligatorio tener instalado IPython, la cual es una shell interactiva que permite la comunicacion entre el lienzo de OpenGL y Python de forma directa:

```
conda install -c conda-forge ipython
```

Instalación de Visual Studio Code (VSCode)

Visual studio code es un editor de código que permitirá la programacion de la simulación de forma local desde el ordenador. Descargar VSCode desde: `https://code.visualstudio.com/`

En VSCode:

- Abrir el panel de extensiones.
- Instalar la extensión **Python** (Microsoft).
- Abrir la paleta de comandos: `Ctrl + Shift + P`.
- Seleccionar: `Python: Select Interpreter`.
- Elegir el intérprete correspondiente al entorno `manim-ce`.

Organización del proyecto

Para comenzar, es fundamental establecer una estructura de carpetas clara. Se recomienda crear el directorio de trabajo principal en el Escritorio (`Desktop`) o Documentos (`Documents`).Siga los pasos a continuación en la terminal de Miniforge Prompt para configurar la estructura de directorios:

1. **Verificar ubicación**: Utilice el comando `dir` para listar los archivos y carpetas del directorio actual.
2. **Navegar al Escritorio**: Ingrese al directorio del Escritorio. Deberá verificar el nombre correcto de la carpeta (generalmente `Desktop` o `Escritorio`) ejecutando previamente `dir`.
3. **Crear la Carpeta Principal**: Una vez ubicado en el Escritorio, cree la carpeta para el proyecto de Física II.
4. **Acceder a la Carpeta Principal**: Ingrese al directorio de `Fisica-II`.
5. **Crear el Subdirectorio de Manim**: Dentro de la carpeta `Fisica-II`, cree el subdirectorio para los archivos de Manim.
6. **Acceder al Directorio de Trabajo**: Ingrese al directorio `manim`.

```
dir
cd Desktop
mkdir Fisica-II
cd Fisica-II
mkdir manim
cd manim
```

Luego abrir dicha carpeta con VSCode:

```
code .
```

Crear un archivo llamado:

```
mas.py
```

y colocarlo dentro de la carpeta `manim`.

Prueba de funcionamiento

Para verificar que ManimCE con soporte OpenGL está instalado correctamente, crear un archivo llamado `mas.py` con el siguiente contenido:

```
from manim import *
from manim.opengl import *

class ManimOpenGL(Scene):
    def construct(self):
        # Crear un texto usando ManimCE
        t = Text("Hola desde ManimCE")

        # Animar el texto con Write
        self.play(Write(t))

        # Entrar al modo interactivo (requiere OpenGL)
        self.interactive_embed()
```

Este script contiene la estructura mínima para un proyecto con OpenGL:

- `from manim import *` importa las herramientas principales de ManimCE.
- `from manim.opengl import *` habilita el renderizado y objetos basados en OpenGL.
- La clase `ManimOpenGL(Scene)` define una escena que será procesada por Manim.
- El método `construct()` es el punto donde se construyen las animaciones.
- `self.play(Write(t))` muestra el texto mediante la animación de escritura.
- `self.interactive_embed()` activa el modo interactivo en IPython, disponible solo si OpenGL funciona correctamente.

Luego ejecutar en la terminal:

```
manim -qm -p --renderer=opengl mas.py ManimOpenGL
```

Si durante la ejecución aparece una ventana de OpenGL con la animación “Hola desde ManimCE”, y después se abre automáticamente un intérprete IPython en la terminal (Miniforge Prompt), entonces la instalación de ManimCE con soporte OpenGL se ha completado correctamente.

A veces es necesario decidir si se desea renderizar la animación como un video o simplemente visualizarla en una ventana mediante OpenGL. La elección depende del propósito:

- **Ventana OpenGL (previsualización interactiva)**: Es útil durante el desarrollo de una escena, cuando se desea observar rápidamente el comportamiento de los objetos, las transformaciones o la sincronización de las animaciones. Este modo no genera archivos de video; simplemente abre una ventana con la animación, permitiendo iterar y depurar más rápido.
- **Renderizado como video (resultado final)**: Se usa cuando la escena ya está terminada y se necesita producir un video reproducible, estable y con una calidad determinada (`-ql`, `-qm`, `-qh`). Este archivo puede luego incluirse en informes, presentaciones o material de clase. El render en video garantiza consistencia visual y mejor iluminación que la previsualización OpenGL.

Para generar un video en calidad media y abrirlo automáticamente, solo es necesario omitir OpenGL:

```
manim -qm -p mas.py ManimOpenGL
```

Recomendaciones adicionales

- Mantener el entorno `manim-ce` exclusivamente para proyectos gráficos.
- Evitar mezclar versiones de Python fuera del entorno virtual.
- Utilizar el comando: `conda info --envs` para verificar los entornos instalados.
- En caso de errores gráficos en Windows, instalar o actualizar drivers OpenGL/DirectX.

4. PROCEDIMIENTO EXPERIMENTAL

En esta sección se explica detalladamente cómo implementar, probar y depurar la simulación del Movimiento Armónico Simple (M.A.S.) usando **ManimCE** con **OpenGL** y una shell interactiva (`IPython`). El objetivo es que el estudiante pueda testear pequeñas porciones de código desde la sesión interactiva y, una vez verificado su comportamiento, integrarlas en el archivo final `mas.py`.

Flujo de trabajo recomendado

1. Abrir **Miniforge Prompt** y activar el entorno `manim-ce`:

   ```
   conda activate manim-ce
   ```

2. Abrir un editor (VSCode) en la carpeta del proyecto y crear `mas.py`.
3. Ejecutar la escena en modo OpenGL con IPython para permitir inspección y modificación en caliente:

   ```
   manim -qm -p --renderer=opengl mas.py MAS
   ```

   Cuando la escena alcanza `self.interactive_embed()` se abrirá un intérprete `IPython` conectado a la escena (variable `scene` disponible). Desde allí se pueden ejecutar comandos para crear, modificar y eliminar *mobjects* sin recompilar todo el archivo.

Estructura general del código (explicación por secciones)

A continuación se detalla cada bloque de la clase `MAS` y se explica cómo probarlo interactivamente. El código que seguiremos es el siguiente:

```
class MAS(Scene):
    def construct(self):
        # ... (todo el código que se presenta más abajo por secciones)
        self.interactive_embed()
```

Listing 1: Clase completa (referencia)

Trabajaremos sección por sección: variables, condiciones iniciales, interfaz (texto y fórmulas), ejes/plot, masa–resorte (geometría) y animadores (updaters).

1. Variables globales y temporizador

Definición y propósito:

```
# VARIABLES GLOBALES
tiempo_interno = ValueTracker(0)    # variable que la escena animará; leer con .get_value()
ESCALA = 0.5                        # escala visual entre unidades físicas y la pantalla
tiempo_simulado = 18                # duración total de la simulación (s)
```

Probar en IPython:

- En IPython escribir: `tiempo_interno.get_value()` para ver el valor.
- Para avanzar manualmente durante pruebas: `tiempo_interno.set_value(2.5)`.

2. Parámetros físicos y función $x(t)$

Definición, significado y pruebas:

```
# Condiciones iniciales y parámetros físicos
x0 = 1.5
v0 = 3
k = 10
m = 3
g = 9.81

# parámetros calculados
w0 = np.sqrt(k / m)
A = np.sqrt(x0**2 + (v0**2) / (w0**2))
delta = np.arctan2(v0, w0 * x0)
x_equilibrio = (m * g) / k

# función x(t)
def x(t):
    return A * np.cos(w0 * t - delta)
```

Probar en IPython:

- Evaluar valores numéricos: `A, w0, delta, x_equilibrio`.
- Evaluar la función en un tiempo: `x(0), x(0.5), x(3.14)`.
- Si las expresiones dan `nan` o `inf`, revisar entradas (ej. x_0=0).

3. Interfaz: Título y ecuaciones en pantalla

Cómo crear y probar textos/MathTex:

```
# Título
Titulo = Text("LAB 06: M.A.S.", font_size=24)
Titulo.move_to(UP * 3)
self.play(Write(Titulo))

# Ecuaciones
x_tex = MathTex(r"x(t)=A\cos(\omega_0 t - \delta)", font_size=24)
x_tex.to_corner(UL)
A_tex = MathTex(r"A=\sqrt{x_0^2+\frac{\dot{x}_0^2}{\omega_0^2}}", font_size=24)
A_tex.next_to(x_tex, DOWN)
delta_tex = MathTex(r"\delta = \tan^{-1}\!\left(\frac{\dot{x}_0}{\omega_0 x_0}\right)", font_size=24)
delta_tex.next_to(A_tex, DOWN)
omega_tex = MathTex(r"\omega_0 = \sqrt{\frac{k}{m}}", font_size=24)
omega_tex.next_to(delta_tex, DOWN)

self.play(Write(x_tex), Write(A_tex), Write(delta_tex), Write(omega_tex))
```

Probar en IPython:

- Una vez en el intérprete interactivo, crear una instancia de `Text` o `MathTex` y añadirla:

```
Titulo = Text("Prueba"); scene.add(Titulo); scene.ren
```

- Para borrar: `scene.remove(Titulo); scene.renderer.force_`

4. Ejes y gráfica de $x(t)$

Crear ejes y comprobar rangos:

```
ejes = Axes(
    x_range=[0, tiempo_simulado, 1],
    y_range=[-A, A, 1],
    x_length=tiempo_simulado * ESCALA,
    y_length=2 * A * ESCALA,
    axis_config={
        "include_tip": False,
        "include_ticks": True,
        "include_numbers": True,
        "font_size": 24,
    }
)
ejes.to_corner(DL)
self.play(Create(ejes))
```

Probar en IPython:

- `ejes.c2p(1, 0)` convierte coordenadas (t, x) a coordenadas de pantalla;
- comprobar que `ejes.c2p(0,0)` coincide con la línea de equilibrio visual.

5. Construcción del sistema masa–resorte

El siguiente código construye el sistme masa resorte de la simulacion

```
# techo
techo = Dot(color=GREY)
self.add(techo)

# linea de equilibrio
linea_equilibrio = DashedLine(start=[-0.5,0,0], end=[0.5,0,0])
linea_equilibrio.shift(DOWN * x_equilibrio)

# bloque de masa
bloque_masa = Square(0.5, color=BLUE)
bloque_masa.set_fill(BLUE, opacity=0.7)
bloque_masa.move_to(linea_equilibrio.get_center())

# centro de masa
cm = Dot(color=RED)
cm.move_to(bloque_masa.get_center())

# resorte
resorte = Line(color=YELLOW, start=techo.get_center(), end=cm.get_center())

# etiquetas y líneas auxiliares (+A, -A, x0)
etiqueta_equilibrio = MathTex(r"x=0", font_size=20)
etiqueta_equilibrio.next_to(linea_equilibrio, RIGHT)
linea_A_mas = DashedLine(start=[-0.5,0,0], end=[0.5,0,0])
linea_A_mas.shift(linea_equilibrio.get_center() + A * ESCALA * UP)
linea_A_menos = DashedLine(start=[-0.5,0,0], end=[0.5,0,0])
linea_A_menos.shift(linea_equilibrio.get_center() - A * ESCALA * UP)
etiqueta_A_mas = MathTex(f"A={A:.2f}", font_size=20)
etiqueta_A_mas.next_to(linea_A_mas, RIGHT)
etiqueta_A_menos = MathTex(f"A={-A:.2f}", font_size=20)
etiqueta_A_menos.next_to(linea_A_menos, RIGHT)
linea_x0 = DashedLine(start=[-0.5,0,0], end=[0.5,0,0])
linea_x0.shift(linea_equilibrio.get_center() + x0 * ESCALA * UP)

sistema_masa_resorte = VGroup(
    techo, linea_equilibrio, bloque_masa, cm, resorte,
    etiqueta_equilibrio, linea_A_mas, linea_A_menos,
    etiqueta_A_mas, etiqueta_A_menos, linea_x0
)
sistema_masa_resorte.next_to(ejes, RIGHT, buff=1)
```

Probar en IPython:

- Crear cada objeto por separado y añadirlo: `scene.add(bloque_masa)`.

- Mover el bloque con `bloque_masa.move_to([x,y,0])` y forzar redraw: `scene.renderer.force_redraw(scene)`.

6. Updaters y animación (vincular $x(t)$ al objeto)

Explicación de updaters y su uso con `ValueTracker`:

```
# updaters que hacen que los objetos sigan la física
bloque_masa.add_updater(
    lambda m: m.move_to(linea_equilibrio.get_center() +
    ESCALA * x(tiempo_interno.get_value()) * UP)
)
cm.add_updater(lambda m:
    m.move_to(bloque_masa.get_center()))
resorte.add_updater(lambda m:
    m.put_start_and_end_on(techo.get_center(),
    cm.get_center()))

# punto en la grafica y su traza
punto_grafica = always_redraw(lambda: Dot(
    ejes.c2p(tiempo_interno.get_value(),
    x(tiempo_interno.get_value())),
    color=RED, radius=0.05
))
trayectoria = TracedPath(punto_grafica.get_center,
    stroke_color=RED, stroke_width=2)
self.add(punto_grafica, trayectoria)

# ejecutar la animación temporal
self.play(
    tiempo_interno.animate.set_value(tiempo_simulado),
    run_time=tiempo_simulado,
    rate_func=linear
)
```

Qué prueba cada línea:

- `add_updater(lambda m: ...)`: cada frame la lambda se ejecuta y mueve el mobject.
- `always_redraw(...)` crea un mobject que se recomputa cada frame (útil para puntos).
- `tiempo_interno.animate.set_value(...)` anima el ValueTracker y activa los updaters.

Probar interactivamente: ejemplos de comandos útiles en IPython

Mientras la escena está detenida en `self.interactive_embed()` la variable `scene` está disponible. Algunos comandos útiles:

```
# Añadir objetos rápidamente
scene.add(Text("Prueba Rápida").move_to(LEFT))

# Eliminar un objeto (suponiendo que su variable es t)
scene.remove(t)
scene.renderer.force_redraw(scene)

# Cambiar el valor del time tracker para ver efecto
    inmediato
tiempo_interno.set_value(2.5)
scene.renderer.force_redraw(scene)

# Borrar updaters (útil si el objeto queda "pegado")
bloque_masa.clear_updaters()
scene.renderer.force_redraw(scene)
```

Cómo limpiar OpenGL y el estado entre pruebas

A veces, después de múltiples pruebas interactivas, el estado gráfico queda en memoria o la ventana OpenGL queda abierta. Recomendaciones ordenadas:

1. **Cerrar la ventana OpenGL**: normalmente cerrando la ventana con el ratón o teclado (tecla de cierre) es suficiente, de no ser asi, desde la terminal de IPython ejecutar `exit`.
2. **Detener el proceso**: si la ejecución quedó colgada, en la terminal presionar `Ctrl+C` para forzar la interrupción.
3. **Limpiar variables en IPython**: en el prompt interactivo ejecutar:

```
%reset -f
```

   Esto borra todas las variables definidas durante la sesión.
4. **Reiniciar el kernel/IPython**: si persisten problemas, cerrar el intérprete y volver a lanzar el comando `manim ...` desde la terminal.
5. **Eliminar updaters y forzar redraw** (si trabajas dentro del intérprete y no quieres reiniciar):

```
bloque_masa.clear_updaters()
scene.remove_all_mobjects()  # si el helper está
    disponible
scene.renderer.force_redraw(scene)
```

   Nota: `scene.remove_all_mobjects()` no es API oficial en todas las versiones; si no existe, remover explícitamente los mobjects que conozcas con `scene.remove(obj)`.

Migrar código probado al archivo `mas.py`

1. Una vez que una porción de código (por ejemplo: definición de ejes, o el updater del bloque) funciona en IPython, copiar ese bloque en `mas.py` dentro del método `construct()` en el orden correcto:
   a) definir variables y funciones matemáticas,
   b) crear interfaz (Text, MathTex),
   c) construir ejes,
   d) crear elementos del sistema masa–resorte,
   e) adjuntar updaters,
   f) ejecutar la animación con `self.play(...)`.
2. Guardar `mas.py` y ejecutar fuera del intérprete para generar el video final:

```
manim -qm -p mas.py MAS
```

Qué hacer si aparece un error

Si a pesar de las instrucciones aparece un error, copia todo el **traceback** (de la terminal) y pégalo en ChatGPT junto al fragmento de código que estabas probando. Para obtener ayuda rápida, se sugiere a los estudiantes usar el siguiente prompt (copiar/pegar):

```
# Prompt sugerido para ChatGPT/ayuda:
"Estoy probando una escena en ManimCE con OpenGL.
Adjunto el traceback completo (incluye las últimas
    20-50 líneas) y el
fragmento de código que causa el error.
Por favor, indícame: 1) posible causa, 2) 2-3
    soluciones prácticas,
3) si necesitas más información, qué exactamente debo
    pegar."
```

Consejos al pegar errores:

- Incluye las primeras y últimas 30-50 líneas del traceback (no solo la última línea).
- Indica la versión de Python (`python --version`) y la versión de Manim (`manim --version`).
- Si el error ocurre al renderizar con OpenGL, pega también el comando exacto que ejecutaste.

4.1. Preguntas de Análisis

1. ¿Qué ocurre con la frecuencia natural si la masa del bloque se duplica?
2. ¿Qué significa físicamente el desfase δ?
3. Compare la amplitud teórica con la amplitud visual observada.
4. ¿Cómo se comportaría el sistema si se introdujera rozamiento?
5. Si $k = 10$ N/m y $m = 2$ kg, calcule el período de oscilación.

5. CONCLUSIONES

- El modelo matemático del M.A.S. describe con precisión la trayectoria oscilatoria de un sistema masa-resorte.
- La simulación en Manim permite visualizar la relación entre los parámetros físicos y la forma del movimiento.
- La frecuencia natural depende únicamente de la masa y la constante elástica, mientras que la amplitud depende de las condiciones iniciales.

6. REFERENCIAS

1. Serway, R. & Jewett, J. (2019). *Física para ciencias e ingeniería*, Vol. I. Cengage Learning.
2. Young, H. & Freedman, R. (2020). *Física Universitaria*, Pearson, 14a Edición.

3. Manim Community. (2025). *Documentación oficial de ManimCE*. Disponible en: `https://docs.manim.community/`

$$x(t) = A\cos\omega_0 t - \delta$$

$$x(2.2) = A\cos\omega_0 2.2 - \delta$$