Ejercicios de Simulación Tema 1

Bubble Shooter

Hacer un **Bubble Shooter** sencillo (lanzador de bolas con velocidad constante).

Resolución

Para realizar la velocidad constante en todo el intervalo me he basado en las siguientes fórmulas:

$$v_{media} = \frac{\Delta x}{\Delta t} \tag{1}$$

$$x \mathrel{+}= v_{media} \cdot \Delta t \tag{2}$$

Primero se ha creado un *punto A* fijo, y un *punto B* con la posición del ratón. Después se ha creado un vector entre ambos *dir* que llevará la dirección de la partícula *P*.

```
void puntoA(){
  stroke(0);
  fill(#f4a261);
  ellipse(a.x,a.y,80,80);
}
void puntoB(){
  stroke(#edede9);
  fill(#edede9);
  ellipse(mouseX, mouseY, 5, 5);
}
```

Luego con un evento de ratón he implementado el lanzamiento de la partícula en base al vector dirección (*dir*).

```
if (mousePressed) {
    // 1. Crear una nueva bola en la posición inicial cuando se hace clic.
    p = new PVector(a.x, a.y);
    // 2. Calcular la dirección entre la posición de la bola inicial y la posición del mouse.
    dir = PVector.sub(new PVector(mouseX, mouseY), p).normalize().mult(75);
  }
    // 3. Mover la bola en esa dirección.
  p.add(PVector.mult(dir, dt)); // Mueve la partícula en la dirección calculada
```

Siendo **p** la **posición de la partícula** que se desplaza y *dir* siendo la velocidad media y dirección en el tramo.

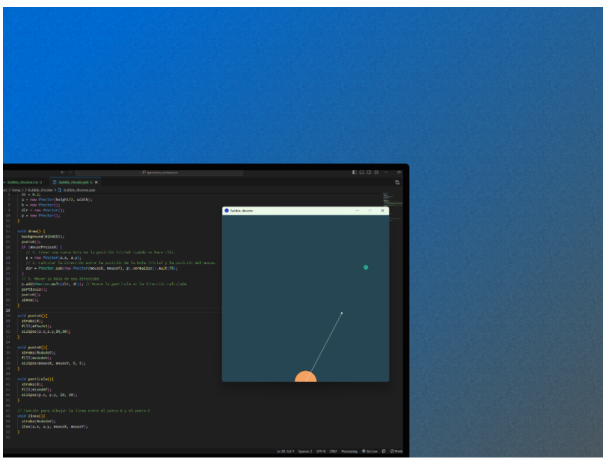


Figure 1: Video

Montaña Rusa 🎢

Simular el movimiento de una partícula que se mueve a tramos de velocidad (ej. con pendientes distintas en cada tramo y velocidades en función de las pendientes). Además añadir aceleración en los tramos.

Resolución

Para realizar la velocidad constante en todo el intervalo me he basado en las siguientes fórmulas:

$$v_{media} = \frac{\Delta x}{\Delta t} \tag{1}$$

$$x\ + = v_{media} \cdot \Delta t \tag{2}$$

Se ha implementado de la siguiente manera en el código:

```
1   if (p.dist(b) <= 1) {
2     velAB = velBC; // Cuando alcanza B cambia de direccion hacia C
3   }
4   p.add(PVector.mult(velAB,dt)); // Para velocidad constante
```

Siendo **p** la **posición de la partícula** que se desplaza por el circuito y **velAB** la <u>velocidad media</u> en el primer tramo. La línea 4 del código es la fórmula 2.

Para la aceleración constante en el intervalo me he basado en la fórmula de la velocidad:

$$v = v_0 + a \cdot \Delta t \tag{3}$$

Se ha aplicado de la siguiente manera en el código:

```
// Actualizar la velocidad y posición según la aceleración en el tramo AB
  if (p.x < b.x) {
    velAB.add(PVector.mult(velAB.copy().normalize(), accAB * dt));
    p.add(PVector.mult(velAB, dt));
  }

  // Actualizar la velocidad y posición según la aceleración en el tramo BC
  if (p.x >= b.x && p.x < c.x) {
    velBC.add(PVector.mult(velBC.copy().normalize(), accBC * dt));
    p.add(PVector.mult(velBC, dt));
  }
```

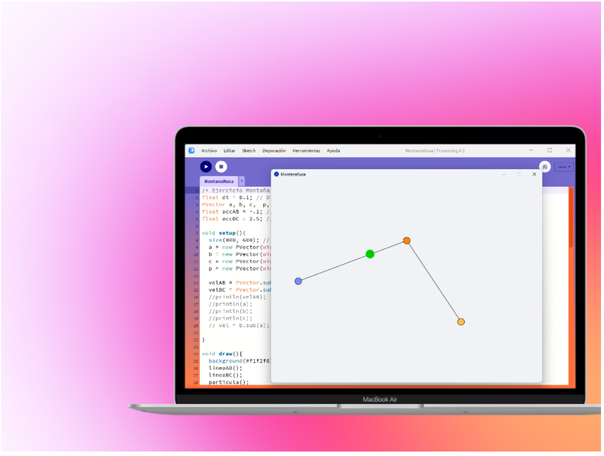


Figure 2: Video

Movimiento Circular ⚾

Simular el movimiento de una bola alrededor de un punto situado a una distancia r de la bola. Dará una vuelta por segundo.

Resolución

Para resolverlo me he basado en la fórmula de la velocidad angular (w):

$$w = \frac{2\pi}{T} \tag{1}$$

Siendo $\boldsymbol{T}$ el tiempo que tarda en recorrer una onda completa en función periódica (Periodo). Para actualizar la posición he usado:

$$x = r \cdot \cos(w \cdot t) \tag{2}$$
$$y = r \cdot \sin(w \cdot t) \tag{3}$$

Se ha implementado de la siguiente manera:

```
t = 1; // Periodo -> Tiempo en recorrer una onda completa
w = (2 * PI) / t; // Fórmula de la velocidad angular
...
// Calcular las coordenadas x y y de la bola utilizando funciones trigonométricas
x = width / 2 + r * cos(w * millis() / 1000);
// millis() devuelve el tiempo transcurrido en milisegundos
y = height / 2 + r * sin(w * millis() / 1000);
```

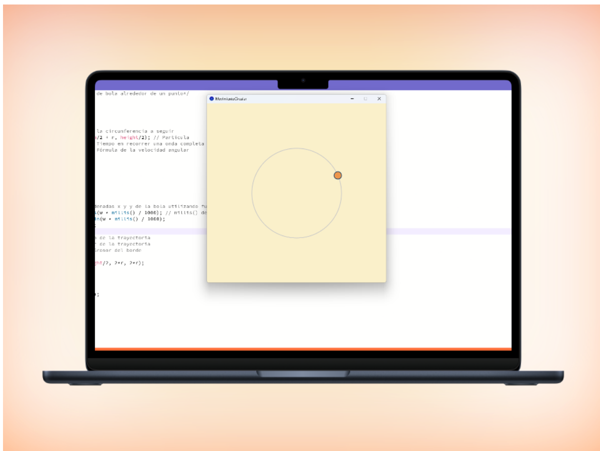


Figure 3: [Video]

Movimiento Oscilatorio

Animar el movimiento de una partícula a velocidad v sobre las dos funciones osciladoras.

$$y = 0.5 \cdot \sin(3x) + 0.5 \cdot \sin(3.5x)$$

Resolución

Para resolverlo he tenido que multiplicar por 50 la función para que se apreciase mejor el cambio en la onda. Además se le suma *height* / 2 para que empiece centrada en la pantalla.

```
// Calcular las coordenadas x y y de la bola utilizando funciones trigonométricas
 x += v; // Incrementa la posición en x con la velocidad

 y = height/2 + 50* (0.5 * sin(3*x) + 0.5 * sin(3.5*x)); // Mantiene la misma amplitud en y
```

A parte de esto, he añadido la visualización del trazado para que se observe de mejor manera la oscilación.

```
ArrayList<PVector> trayectoria; // Arraylist para almacenar las coordenadas de la trayectoria
...
trayectoria.add(new PVector(x, y)); // Agregar las coordenadas a la trayectoria
...
void dibujarTrayectoria() {
  noFill();
  stroke(0,0,0); // Color rojo para la trayectoria
  beginShape();
  for (PVector punto : trayectoria) {
    vertex(punto.x, punto.y);
  }
  endShape();
}
```

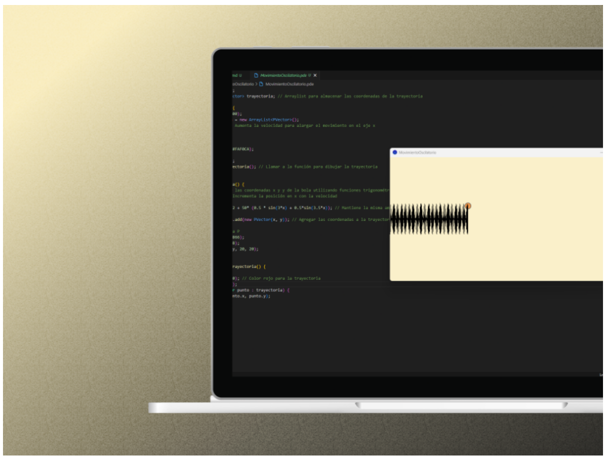
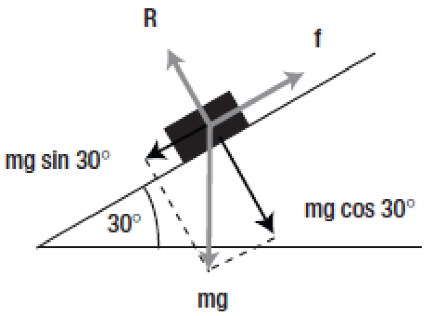


Figure 4: Video

Descomposición de fuerzas, Plano inclinado

El peso se descompone en normal y tangencial al plano de movimiento (en este caso inclinado).



- El Rozamiento (fricción): $f = -kv$. Se opone al movimiento (velocidad negada).
- La normal se anula con R.
- $F = mg\sin(30°) - f$.

Resolución

Para resolverlo nos tenemos que basar en la segunda ley de newton para calcular la aceleración:

$$\sum F = m \cdot a$$
$$a = \frac{\sum F}{m} \tag{1}$$

Donde tendremos que obtener la fuera de rozamiento y el peso, pero al estar en un plano inclinado tenemos que tener en cuenta el ángulo de este, al que hemos llamado *Unitario*.

```
PVector calculateAcceleration(PVector s, PVector v)
{
    PVector Unitario = new PVector(cos(theta), sin(theta));
    PVector Froz  = PVector.mult(Unitario, PVector.mult(v,-K).mag());
    PVector Fpeso = PVector.mult(Unitario, PVector.mult(G,M).mag());
    PVector SumF  = PVector.add(Froz, Fpeso);
    PVector a = SumF.div(M);
    return a;
}
```

Además para calcular la posición y velocidad se ha empleado Euler Simpléctico.

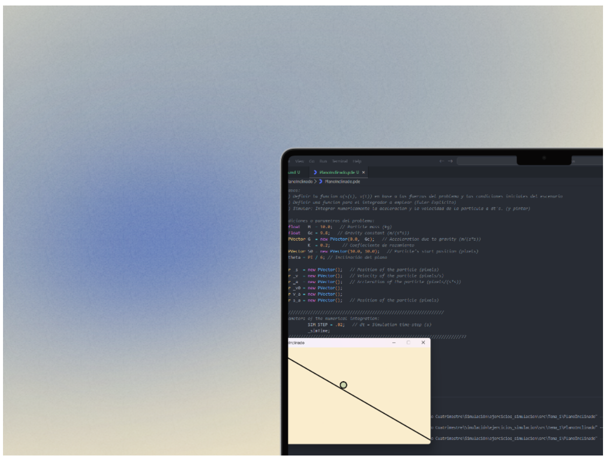
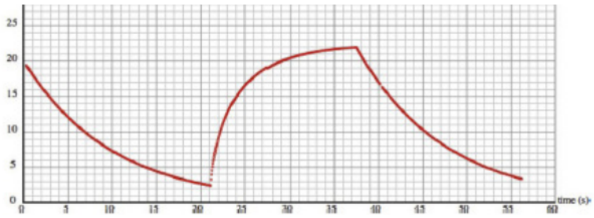


Figure 5: Video

Simulador de coche🚗

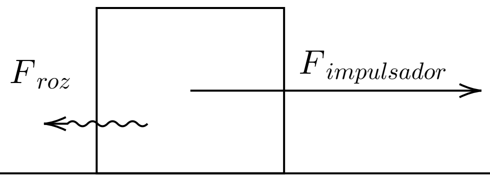
Simularemos el acelerador de un coche.

- Clase coche $(masa,\ vel,\ E_c)$.
- $AplicarPotencia()$, actualizar $E_c = P\Delta t$ al pulsar una tecla.
- Pérdida de energía (rozamiento $F = -kv$) entonces $P = Fv = -kv^2$.
- $updateVelo()$, actualiza la velocidad $v = \sqrt{2E_c \,/\, m}$.



Obtener la gráfica de velocidad y simular el movimiento de un coche con acelerador (trayectoria libre).

Resolución



Para resolverlo nos tenemos que basar en la segunda ley de newton para calcular las fuerzas:

$$a = \frac{\sum F}{m} \tag{1}$$

Lo que he hecho ha sido crear una fuerza de rozamiento $F_{roz} = -vk$ que se aplica cuando el coche tiene velocidad.

```
PVector calculateAcceleration(PVector s, PVector v)
{
    PVector Froz = new PVector(0.0, 0.0);
    if(v.mag() > 0.01) { // Umbral de velocidad
        Froz = PVector.mult(v,-K);
    }
    PVector SumF  = PVector.add(Froz, Fimpulsador);

    PVector a = SumF.div(M);

    return a;
}
```

Lo que he hecho para el desplazamiento es cuando se pulse la tecla *a* aplica una fuerza llamada *Fimpulsador* que aplica en la dirección (1,0) una constante *impulsador*.

```
void AplicarPotencia() {
    PVector normal_aceleracor = new PVector(1.0,0.0);
    normal_aceleracor.normalize();
    Fimpulsador = PVector.mult(normal_aceleracor, impulsador);
}
```

Y cuando se deja de pulsar esta tecla, al igual que un coche deja de ejercer fuerza, por lo que se pone la fuerza del impulsador a 0:

```
void keyReleased()
{
    if(key == 'a' || key == 'A')
    {
        Fimpulsador.set(0.0, 0.0);
    }
}
```

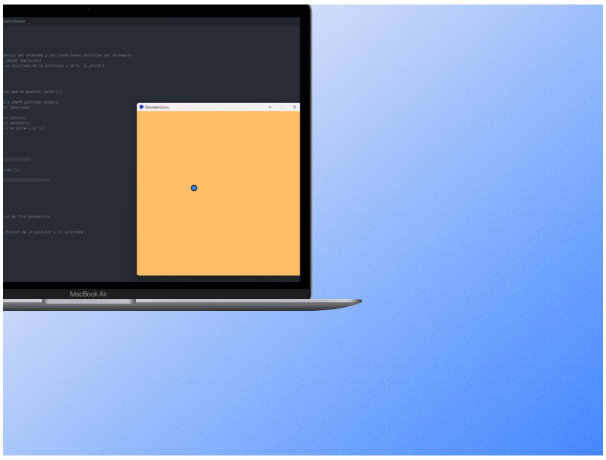


Figure 6: [Video]

Ejercicio Paracaidista con Euler Explícito

1 Ecuación diferencial

La velocidad de caída de un paracaidista en función del tiempo viene dada por la siguiente ecuación:

$$\frac{dv}{dt} = g - \frac{c}{m}v$$

Siendo g la constante gravitacional, m la masa y c el coeficiente de arrastre.

2 Análisis y dibujo

Analizar ecuación (módulo) y dibujar los vectores principales del escenario. Incluir las condiciones iniciales (p.ej. una persona de 80 Kg de peso que saltó de una cabina a 36 Km de altura, dió un pequeño paso adelante, vel (1,0) m/s).

3 Iteración con método de Euler

Iterar con el método de Euler para ir obteniendo la velocidad en los 10 primeros segundos ($dt = 1$).

Parámetros	**Ecuación diferencial (aceleración)**
dt	0.1
g	9.8
m	60
c (coef. arrastre)	80

4 Aceleración del paracaidista

Para sacar la **aceleración** del paracaidista hay que tener en cuenta que

$$\vec{a}_{paracaidista} = \frac{d\vec{v}}{dt} = \vec{g} - \frac{c}{m} \times \vec{v}$$

5 Velocidad de Euler

Para sacar la **velocidad de Euler** se obtiene con

$$\vec{v}_{i+1} = \vec{v}_i + \vec{a}_i \times dt$$

6 Velocidad analítica

Para sacar la **velocidad analítica** se obtiene con

$$v = \frac{gm}{c}(1 - e^{-(c/m)t})$$

7 Error de Euler

Y para obtener el **Error de Euler** se hace el valor absoluto de la diferencia entre el resultado de Euler y el analítico.

$$|v_{euler} - v_{analitica}|$$

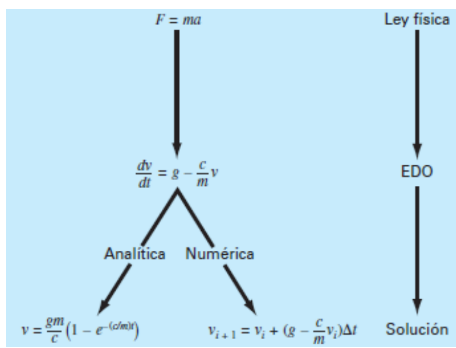


Figure 1: Gráfico de la solución analítica y numérica.

8 Resultados

Los resultados en una tabla quedarían así:

t	$A = \frac{dv}{dt}$	V_{Euler}	$V_{analitica}$	Error(Euler_exp)
0,00	9,80	0,00	0,00	0,06
0,10	8,49	0,98	0,92	0,11
0,20	7,36	1,83	1,72	0,14
0,30	6,38	2,57	2,42	0,17
0,40	5,53	3,20	3,04	0,18
0,50	4,79	3,76	3,58	0,19
0,60	4,15	4,24	4,05	0,19
0,70	3,60	4,65	4,46	0,19
0,80	3,12	5,01	4,82	0,19
0,90	2,70	5,32	5,14	0,18
1,00	2,34	5,59	5,41	0,17
1,10	2,03	5,83	5,65	0,16
1,20	1,76	6,03	5,87	0,15
1,30	1,53	6,21	6,05	0,15
1,40	1,32	6,36	6,21	0,00

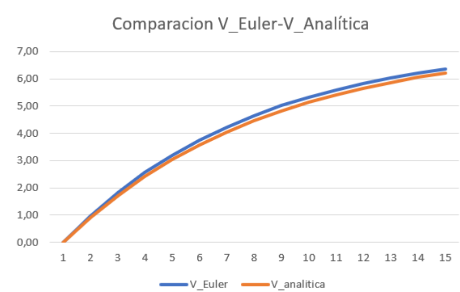


Figure 2: Comparación entre la solución numérica (Euler) y analítica.