

# **Obtención de código mediante inteligencia artificial generativa: resolución de varios problemas matemáticos de interés**

Autores: Miguel Escribano Ródenas [mescr@emp.ucm.es](mailto:mescr@emp.ucm.es)  
Fernando Ricote Gil [fricote@ccee.ucm.es](mailto:fricote@ccee.ucm.es)  
Mercedes Elices López [meelices@ucm.es](mailto:meelices@ucm.es)

*Departamento de Economía Financiera, Actuarial y Estadística*  
Universidad Complutense de Madrid

## **RESUMEN**

Este trabajo se centra en el uso de la Inteligencia Artificial Generativa (IAG) para la resolución de problemas en el área de economía. El objetivo consiste en la generación de código que permita la resolución de un problema en esta área. Para ello se pretende utilizar como metodología la ingeniería de prompts (*prompt engineering*) que permite definir el problema desde un punto de vista de la especificación de requisitos del software que se desea desarrollar.

En la introducción se describe qué objetivos tiene la IAG para describir el contexto en el que se sitúa este trabajo. Se ha hecho referencia a diferentes trabajos en el estado del arte para centrar este estudio. En la metodología se describe el enfoque aplicado para la generación de un prompt que será la entrada a la IAG. A continuación se describen distintos ejemplos en el ámbito de la investigación operativa, y las matemáticas financieras, en los que se presenta: el prompt que se ha utilizado, el código que se ha generado y la ejecución de dicho código para ilustrar la resolución de cada ejemplo planteado.

Los resultados muestran que los problemas planteados, pueden llevarse a cabo mediante este enfoque y que permite la obtención de un modelo con el que poder validar la solución presentada.

Este enfoque tecnológico y metodológico permite dar respuesta a problemas en el ámbito financiero y matemático con resultados muy prometedores.

## ABSTRACT

This work focuses on the use of Generative Artificial Intelligence (GAI) for problem-solving in the field of economics. The objective is to generate code that enables the resolution of a specific problem within this domain. To achieve this, prompt engineering will be employed as the methodological approach, allowing the problem to be specified from the perspective of software requirements specification for the intended development.

The introduction outlines the goals of GAI in order to contextualize the present study. A review of related works in the state of the art is included to frame and position this research. The methodology section describes the approach used to construct a prompt that will serve as input to the GAI system. Subsequently, several examples from the fields of operations research and financial mathematics are presented. For each case, the prompt used, the code generated, and its execution are described to illustrate the resolution of the proposed problems.

The results demonstrate that the proposed problems can be effectively addressed using this approach, enabling the development of models through which the generated solutions can be validated.

This technological and methodological approach proves to be a promising means of addressing problems in the financial and mathematical domains, yielding highly encouraging outcomes.

**Palabras claves:** Inteligencia artificial generativa, generación de código, ingeniería de prompts, ingeniería del software

**Área temática:** Metodología y docencia.

## 1. INTRODUCCIÓN

La Inteligencia Artificial Generativa (IAG) es una rama de la inteligencia artificial que se centra en el desarrollo de modelos capaces de generar datos nuevos que imitan las características estadísticas del conjunto de datos en el que fueron entrenados. A diferencia de los sistemas discriminativos, que se limitan a clasificar o predecir con base a patrones existentes, los modelos generativos pueden producir contenido completamente nuevo, incluyendo texto, imágenes, audio, código, o estructuras tridimensionales.

Desde una perspectiva técnica, los modelos generativos, como las redes generativas adversativas (GANs), los modelos autoregresivos (por ejemplo, ChatGPT), y los modelos de difusión, aprenden la distribución subyacente de los datos de entrenamiento y, a partir de ella, generan muestras que resultan coherentes y plausibles dentro del contexto definido.

En contextos aplicados, la IAG tiene un amplio rango de usos, incluyendo la generación de contenido creativo, la simulación de escenarios económicos, el diseño automatizado, la personalización de experiencias y la asistencia en tareas cognitivas complejas, posicionándose como una herramienta transformadora en diversas disciplinas científicas y sectores industriales.

Un modelo simplificado de cómo funciona la IAG se muestra a continuación:

1. [Usuario]  
↓
2. [Prompt] → [Tokenización]  
↓
3. [Modelo Generativo]  
↓
4. [Postprocesamiento]  
↓
5. [Respuesta generada]

Todo el proceso comienza con el usuario [1] que escribe una especificación en modo texto de lo que se desea obtener. El resultado de este texto obtenido es lo que se conoce como *prompt*. Una vez obtenido el prompt la IAG, realiza un proceso de tokenización [2] consistente en fragmentar el texto en partes manejables. Además se alimenta de datos al

modelo de IAG y se limpian y transforman los datos. Los datos obtenidos son la entrada al modelo generativo [3] y los más representativos incluyen *GPT* (para texto), *Stable Diffusion/DALL-E* (para imágenes), *MusicLM/ Jukebox* (para música) y *CodeGen/Codex* (para generación de código). Los datos que se generan pasan por una capa de posprocesamiento [4] dando lugar a la respuesta generada [5].

Es la generación de código mediante IAG en la que se centra este trabajo.

Para hacer uso de la IAG es clave la ingeniería de prompts. Esta disciplina se centra en el diseño, formulación y optimización sistemática de entradas lingüísticas (prompts) para modelos de lenguaje de gran escala (LLMs, *Large Language Models*), con el fin de guiar su comportamiento y maximizar la precisión, coherencia, relevancia y utilidad de sus salidas. Esta práctica combina principios de lingüística computacional, interacción humano-computador y aprendizaje automático, y se fundamenta en el entendimiento profundo de las arquitecturas subyacentes de los modelos generativos, sus sesgos y limitaciones. La ingeniería de prompts se emplea tanto en contextos de uso general como en tareas especializadas, incluyendo generación de texto, razonamiento lógico, traducción automática, recuperación de información, programación asistida, generación de código, entre otros.

(Ashish *et al.*, 2025) establece que los prompts deben ser: concisos, lógicos, explícitos, adaptativos y reflexivos. Un prompt debe formularse con claridad y centrarse en los elementos esenciales de la tarea, a fin de orientar a la inteligencia artificial hacia respuestas pertinentes y precisas. Es indispensable que los prompts presenten una secuencia lógica de ideas, lo cual facilita la comprensión del contexto y de las relaciones entre conceptos, promoviendo así la coherencia en las respuestas generadas. Asimismo, es fundamental que el prompt defina con precisión el formato, el contenido y el alcance esperados de la respuesta, con el objetivo de evitar resultados ambiguos o poco relevantes. Debe existir, además, un margen de flexibilidad en la elaboración de prompts, que permita experimentar con diversas estructuras y configuraciones, logrando así un equilibrio adecuado entre la creatividad y la especificidad. Por último, la evaluación constante y la mejora continua de los prompts resultan cruciales; aprovechar los aprendizajes derivados de respuestas anteriores permite optimizar de manera progresiva la calidad de las futuras interacciones con la inteligencia artificial.

## 2. ANÁLISIS DEL ESTADO DEL ARTE

El análisis del estado del arte se ha llevado a cabo en tres fases: [1] publicaciones que tratan la ingeniería de prompts, [2] publicaciones sobre la IAG para la generación del código y [3] publicaciones sobre la generación de código aplicado a la educación y docencia.

En relación las publicaciones sobre la ingeniería de prompts cabe destacar (ANAND, N. *et al.*, 2025) que ofrece una guía completa sobre la ingeniería de prompts para modelos de lenguaje de gran escala (LLMs). Cubre desde conceptos fundamentales hasta técnicas avanzadas, incluyendo consideraciones éticas y estudios de caso prácticos. (NAZIR, A. *et al.*, 2023) ofrece una visión general de ChatGPT, explorando su evolución, aplicaciones y desafíos actuales. Aunque no se centra exclusivamente en la ingeniería de prompts, proporciona información valiosa sobre cómo las técnicas de *prompting* influyen en el rendimiento y la aplicabilidad de los modelos de lenguaje. (PERES, R. *et al.* 2023) analiza el impacto de la IAG en la investigación académica, destacando la importancia de la ingeniería de prompts como habilidad esencial. Se discuten estrategias para enseñar y aprender a formular prompts efectivos, así como métodos para evaluar críticamente las respuestas generadas por la inteligencia artificial. (KHALIFA, M., *et al.*, 2024) analiza el papel transformador de la inteligencia artificial como una herramienta esencial para mejorar la productividad en la redacción académica y la investigación científica. La inteligencia artificial puede automatizar tareas repetitivas, mejorar la calidad de los manuscritos científicos y acelerar el proceso de investigación. (GUPTA, A., *et al.*, 2024) examina cómo los usuarios comparten y discuten estrategias de prompting en redes sociales, ofreciendo una perspectiva sobre las prácticas emergentes y la alfabetización crítica en el uso de la IAG.

Por otro lado, en relación con las publicaciones de IAG aplicada a la generación de código cabe reseñar que (FOGGIA, A., *et al.*, 2024) propone un método automatizado llamado ACCA para evaluar la corrección del código generado por inteligencia artificial en contextos de seguridad. ACCA utiliza técnicas de análisis estático y dinámico para identificar vulnerabilidades y errores en el código generado, proporcionando una evaluación integral de su calidad y seguridad. (PARK, A. *et al.*, 2025) investiga el papel de los *chatbots* de la IAG, como ChatGPT, en la mejora de la educación en programación

a nivel universitario. Analiza cómo estas herramientas pueden proporcionar sugerencias de código y explicaciones que facilitan el aprendizaje y la comprensión de conceptos de programación por parte de los estudiantes. (LIGUORI, P. *et al.*, 2023) analiza un conjunto amplio de métricas automáticas utilizadas para evaluar la calidad del código generado por modelos de traducción automática neuronal. Se centra en cómo estas métricas pueden aplicarse para medir la similitud y la funcionalidad del código generado, proporcionando una visión crítica sobre su efectividad y limitaciones.

En último lugar sobre las publicaciones relacionadas con generación de código aplicado a la educación y docencia cabe mencionar (RAMAZAN, Y., *et al.*, 2024) que examina cómo las herramientas basadas en IAG pueden ayudar a los estudiantes en la escritura de código, proporcionando sugerencias, detección de errores y generación automática de código. Se discute cómo estas herramientas pueden mejorar la eficiencia y la calidad del aprendizaje en programación. (LIM, W. *et al.*, 2023) analiza el impacto de la IAG en la educación, abordando las paradojas que presenta su integración en entornos educativos. Propone perspectivas para reconciliar los debates sobre su uso en la enseñanza y el aprendizaje. (KIZILCEC, R. *et al.*, 2024) presenta una encuesta internacional a educadores y estudiantes de educación superior para comprender y comparar sus perspectivas sobre el impacto de la IAG. Se discuten las implicaciones de la IAG en la evaluación y cómo puede afectar la enseñanza de la programación. (KOHNKE, L. *et al.*, 2023) explora las actitudes de los instructores de inglés en educación superior hacia las herramientas de IAG, sus intenciones de uso y las implicaciones para la enseñanza. Aunque se centra en la enseñanza del inglés, ofrece aspectos relevantes sobre la preparación y disposición de los educadores para integrar la IAG en sus prácticas pedagógicas.

Como conclusión del análisis bibliográfico efectuado cabe destacar que la ingeniería de prompts es una tarea clave en la generación de código mediante la IAG. En el ámbito de la docencia y la educación se utiliza como un vector fundamental en el proceso de enseñanza y aprendizaje. Es reseñable que incluso en el área de la programación (ingeniería del software) este enfoque para la generación automática de código, es utilizado en el ámbito de la docencia como herramienta clave para el alumnado que se está formando en esta disciplina.

### 3. METODOLOGÍA

La metodología empleada consiste en el ámbito de este trabajo consiste en lo siguiente:

1. La especificación del prompt para la IAG

Para ello se realiza una especificación de requisitos, en donde se define el problema que se desea resolver con detalle. Es clave el vocabulario del dominio que se maneja, para evitar respuestas ambiguas no deseadas así como la definición del objetivo que se persigue: un cálculo, una visualización, un gráfico, una tabla, entre otros.

2. Observación del código generado

La IAG generativa nos va a dar un código cuyo funcionamiento se detalla en la salida que se ofrece. Incluso el propio código va a estar comentado para facilitar la comprensión del mismo. En esta fase es habitual proceder a la interpretación del código que facilitará el conocimiento de los resultados que se obtienen tras su ejecución y las depuraciones futuras del mismo.

3. Ejecución del código

Para llevar a cabo la ejecución del código hay varias posibilidades:

1. Hacer uso de un IDE (*Integrated Development Environment*) que consiste en un entorno para el desarrollo y depuración del software. Este entorno ha de tener las correspondientes librerías a que hace referencia el código generado.
2. Hacer uso de un entorno de computación en la nube que permite elevar el nivel de abstracción para centrarnos en la ejecución del propio código.
3. Tener un entorno mixto entre [1] y [2] que permita hacer uso tanto del IDE como de la computación en la nube.

4. Validación del resultado obtenido

En esta fase se determina si el software obtenido es válido en términos de la calidad de las salidas que ofrece. Para ello es necesario conocer en el dominio que se maneja, si es realista la solución y por tanto disponer de problemas (pruebas unitarias) de los que se conoce una solución previa, permite validar la solución obtenida y refinarla si fuere necesario.

## 4. EJEMPLOS PRÁCTICOS DE GENERACIÓN DE CÓDIGO

En este apartado se describe un conjunto de problemas matemáticos y económicos con un enfoque basado en el uso de la tecnología IAG. En cada uno de los problemas llevados a cabo se ha planteado:

1. La especificación del prompt para la IAG
2. La observación del código generado
3. La explicación por parte la IAG del código obtenido
4. La ejecución del código obtenido mediante el uso de un IDE
5. El análisis de las salidas ofrecidas por la aplicación

Los ejemplos que se desarrollan se han llevado a cabo en Python, ya que se ha considerado que este lenguaje presenta muchas facilidades en cuanto a la facilidad de comprensión del código generado. En cualquier caso, se entiende que estos resultados son extrapolables a otros lenguajes de programación de propósito general.

La especificación del prompt se ha llevado a cabo en lengua inglesa con el propósito de poder acceder a mayor cantidad de información dentro de la base de conocimiento de las herramientas de IAG.

Los ejemplos que se presentan en este artículo se han validado mediante el uso del IDE Visual Studio Code de Microsoft para ejecutar el código generado y la IAG que se ha utilizado ha sido ChatGPT 4.

### 4.1. Problema de optimización con restricción de igualdad

Se trata de llevar a cabo el cálculo de extremos de una función de dos variables con una restricción de igualdad. Es conocido que este problema clásico de optimización se resuelve mediante el teorema de los multiplicadores de Lagrange.

Para ello se ha utilizado el siguiente prompt:

*Optimize  $f(x,y)=x^2-y^2$  subject to  $x^2+y^2=1$ . Plot contour lines (heatmap) for  $f(x,y)$  as well as the constraint, to provide de minimum and maximum plotted in the chart. Describe lagrangian function in the solution to have a mathematical method for solving this problem. Could you provide a python code that provides solution to this optimization problem? Python code should be able to visualize the contour lines as well as the solution. Can you visualize a three-dimension visualization for the model with this python code?*

La IAG proporciona la solución del problema de optimización como se ve en la figura 1.

To solve the constrained optimization problem:

$$\max / \min f(x, y) = x^2 - y^2$$

subject to the constraint

$$g(x, y) = x^2 + y^2 - 1 = 0$$

**Lagrangian Function**

The Lagrangian function for this problem is:

$$\mathcal{L}(x, y, \lambda) = x^2 - y^2 + \lambda(1 - x^2 - y^2)$$

Taking partial derivatives and setting them to zero:

$$\frac{\partial \mathcal{L}}{\partial x} = 2x - 2\lambda x = 0$$
$$\frac{\partial \mathcal{L}}{\partial y} = -2y - 2\lambda y = 0$$
$$\frac{\partial \mathcal{L}}{\partial \lambda} = 1 - x^2 - y^2 = 0$$

From  $\frac{\partial \mathcal{L}}{\partial x} = 0$ , we get:

$$x(1 - \lambda) = 0 \Rightarrow x = 0 \text{ or } \lambda = 1$$

From  $\frac{\partial \mathcal{L}}{\partial y} = 0$ , we get:

$$y(-1 - \lambda) = 0 \Rightarrow y = 0 \text{ or } \lambda = -1$$

Solving for  $(x, y)$ :

1.  $x = 0 \Rightarrow y^2 = 1 \Rightarrow y = \pm 1$ , giving  $f(0, \pm 1) = -1$  (Minimum).
2.  $y = 0 \Rightarrow x^2 = 1 \Rightarrow x = \pm 1$ , giving  $f(\pm 1, 0) = 1$  (Maximum).

**Python Code for Visualization**

The following Python code plots:

- Contour lines of  $f(x, y)$  (heatmap).
- The constraint  $x^2 + y^2 = 1$  (circle).
- The critical points  $(\pm 1, 0)$  (maxima) and  $(0, \pm 1)$  (minima).

Figura1. Solución analítica proporcionada al problema.

El código que genera la IAG se muestra en la Figura 2:

```
import numpy as np
import matplotlib.pyplot as plt

# Define function f(x, y)
def f(x, y):
    return x**2 - y**2

# Create a grid of points
x = np.linspace(-1.5, 1.5, 400)
y = np.linspace(-1.5, 1.5, 400)
X, Y = np.meshgrid(x, y)
Z = f(X, Y)

# Create figure and plot contours
fig, ax = plt.subplots(figsize=(6,6))
contour = ax.contourf(X, Y, Z, levels=20, cmap='coolwarm', alpha=0.75)
ax.contour(X, Y, Z, levels=20, colors='black', linewidths=0.5)
plt.colorbar(contour, label='f(x, y)')
```

```

# Plot constraint circle
theta = np.linspace(0, 2*np.pi, 300)
circle_x = np.cos(theta)
circle_y = np.sin(theta)
ax.plot(circle_x, circle_y, 'g', linewidth=2, label='$x^2 + y^2 = 1$')

# Mark max/min points
critical_points = [(1,0), (-1,0), (0,1), (0,-1)]
for (px, py) in critical_points:
    ax.scatter(px, py, color='red', s=100, edgecolors='black', label='Optimum' if (px, py) ==
(1,0) else "")

ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_title('Contour plot with constraint')
ax.legend()
plt.grid(True)
plt.show()

# 3D Plot
fig = plt.figure(figsize=(8,6))
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, Z, cmap='coolwarm', edgecolor='none', alpha=0.8)
ax.plot(circle_x, circle_y, f(circle_x, circle_y), 'g', linewidth=2, label="Constraint")
ax.scatter([1, -1, 0, 0], [0, 0, 1, -1], [1, 1, -1, -1], color='red', s=50, edgecolors='black')

ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('f(x, y)')
ax.set_title('3D Surface Plot with Constraint')
plt.legend()
plt.show()

```

Figura 2. Código generado por la IAG para la solución del problema

Se hace uso de las librerías *matplotlib* y *numpy* de Python. El código se encarga de la visualización mediante una malla de puntos en tres dimensiones y en dos dimensiones, tanto para la restricción como para la gráfica de la función. La ejecución de este código muestra las curvas de nivel y la superficie en tres dimensiones como se muestra en las figuras 3 y 4, así como los extremos.

Como podemos observar a la vista del código Python:

1. Se determinan las curvas de nivel de  $f(x,y)$  con una malla de puntos como un mapa de calor

2. Se superpone la restricción en dos dimensiones que define el círculo de radio unidad y se marcan los extremos en rojo
3. Se determina la gráfica de la función tres dimensiones  $z=f(x,y)$  con una malla de puntos
4. Se muestra la restricción correspondiente al círculo en tres dimensiones
5. Los extremos se marcan en rojo

Como se observa la visualización proporciona una comprensión intuitiva del problema de optimización en las figuras 3 y 4.

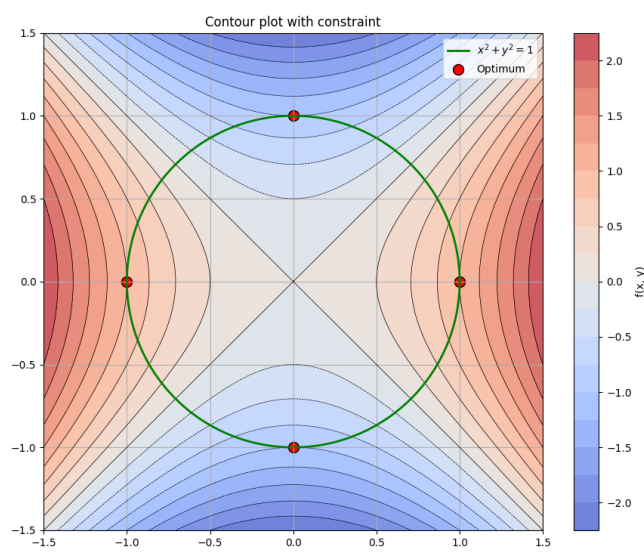


Figura 3. Curvas de nivel proporcionadas por el software.

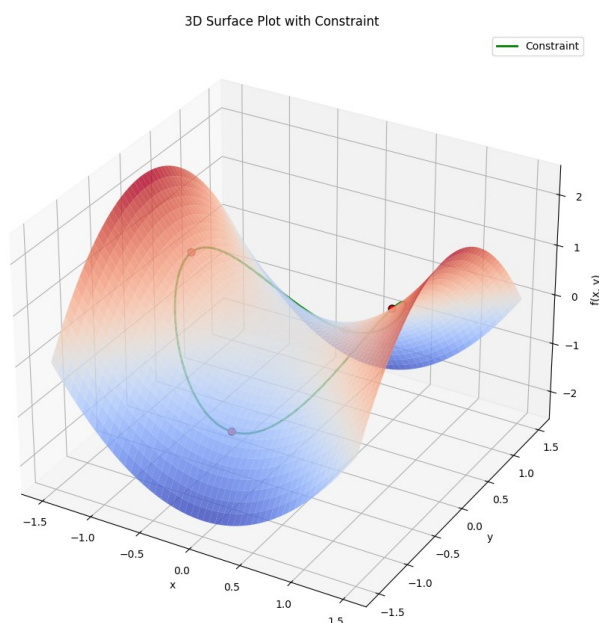


Figura 4. Visualización en tres dimensiones de la solución del problema

## 4.2. Problema del cálculo de un préstamo de tipo francés

Se trata de llevar a cabo el cálculo del cuadro de amortización de un préstamo de tipo francés en el que hay un periodo en el que se cancela el préstamo. Se pretende conocer la evolución del capital vivo (deuda pendiente) desde el comienzo de la operación hasta que se cancela el préstamo.

Para ello se ha utilizado el siguiente prompt:

*one customer contracts a loan with a bank for a principal of 100.000 dollars with monthly payments during 20 years with an monthly interest rate of 8%, corresponding to an scheme of fixed payments. At the end fo 15th year, the loan is canceled. Can you provide as a python code for the loan shedule, in which you must provide: monthly period, interest element, capital element, payment, capital outstanding after payment as a table?. Can you plot the remaining balance per each period as a line plot?*

La IAG nos proporciona como se formula el préstamo tal y como se ve en la figura 5.

To calculate the loan schedule, we use the formula for **fixed monthly payments** in an amortizing loan:

$$A = P \frac{r(1+r)^n}{(1+r)^n - 1}$$

Where:

- $P = 100,000$  (principal)
- $r = 0.08$  (monthly interest rate of 8%)
- $n = 20 \times 12 = 240$  (total number of payments)
- $A$  is the fixed monthly payment.

After computing the monthly payment, we generate the **loan amortization schedule**, which includes:

- Period (Month)
- Interest Payment
- Principal Payment
- Total Payment (Fixed)
- Remaining Balance

Figura 5. Cálculo del préstamo según establece la IAG

El código que genera la IAG se muestra en la Figura 6:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Loan Parameters
principal = 100000 # Loan amount ($)
monthly_interest_rate = 3 / 100 # 8% per month (0.08)
loan_term_months = 20 * 12 # 20 years = 240 months
```

```
early_repayment_month = 15 * 12 # Loan paid off after 15 years (180 months)

# Compute Fixed Monthly Payment (EMI)
emi = (principal * monthly_interest_rate * (1 + monthly_interest_rate)**loan_term_months) / \
      ((1 + monthly_interest_rate)**loan_term_months - 1)

# Initialize variables
remaining_balance = principal
schedule = []

# Loan Amortization Schedule Calculation
for month in range(1, loan_term_months + 1):
    interest_payment = remaining_balance * monthly_interest_rate # Interest for the month
    principal_payment = emi - interest_payment # Principal portion
    remaining_balance -= principal_payment # Update remaining balance

    # Store schedule details
    schedule.append([month, round(interest_payment, 2), round(principal_payment, 2),
                    round(emi, 2), round(max(0, remaining_balance), 2)])

    # Stop at early repayment month
    if month == early_repayment_month:
        final_payment = remaining_balance # Amount to pay off at month 180
        remaining_balance = 0
        break

# Convert to DataFrame
columns = ["Month", "Interest Paid", "Principal Paid", "Total Payment", "Remaining Balance"]
df_schedule = pd.DataFrame(schedule, columns=columns)

# Display first few rows of the schedule
print(df_schedule.head(10))
print(f"\nOutstanding balance at month {early_repayment_month}: ${final_payment:,.2f}")

# Plot Remaining Balance Over Time
plt.figure(figsize=(10,5))
plt.plot(df_schedule["Month"], df_schedule["Remaining Balance"], label="Remaining Balance", color='b')
plt.axvline(x=early_repayment_month, color='r', linestyle='--', label="Loan Canceled (Month 180)")
plt.xlabel("Month")
plt.ylabel("Remaining Loan Balance ($)")
plt.title("Loan Balance Over Time")
plt.legend()
plt.grid()
```

Figura 6. Código generado por la IAG para la solución del problema.

Se puede observar que se hace uso de las librerías *numpy*, *pandas* y *matplotlib*. El cuadro de amortización se construye de forma iterativa y se almacena en un *dataframe*, una estructura de datos de la librería *pandas*. Esta estructura de datos permite manejar la información de forma matricial y es muy apta para el cálculo desde un punto de vista algorítmico. Finalmente, este *dataframe* se utiliza para visualizar la evolución del capital vivo.

A continuación, se muestra el resultado de la *ejecución* del código en la figura 7.

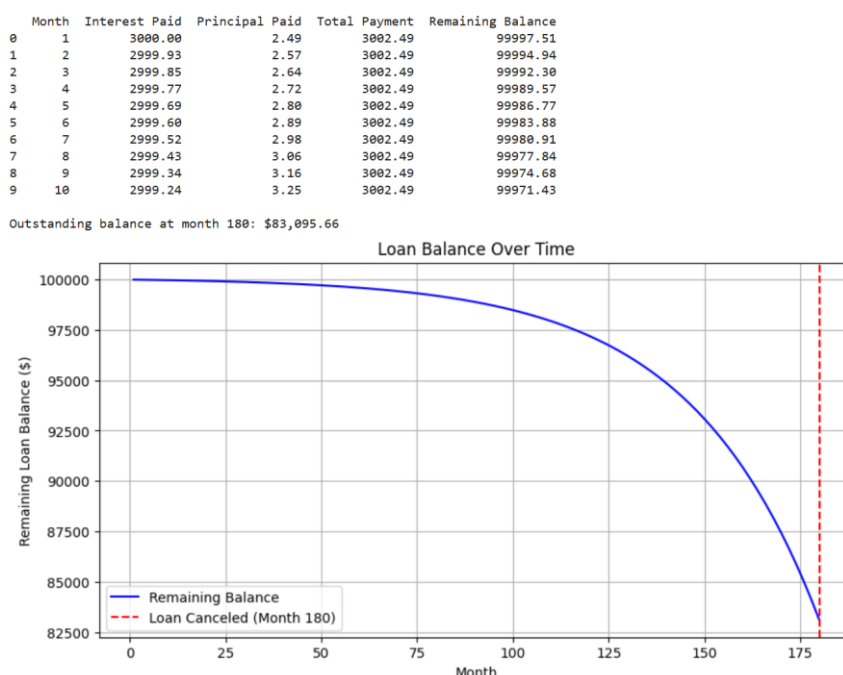


Figura 7. Resultado de la ejecución del código para el problema de operación de amortización.

Como podemos observar a la vista del código Python:

- [1] Se calcula el término amortizativo.
- [2] Se itera sobre cada mes para el cálculo de la cuota de interés, de la cuota de amortización, del capital vivo, los valores en forma de cuadro de amortización. Se para en el mes 180 (décimoquinto año).
- [3] Se visualiza el cuadro de amortización y el capital vivo se grafica hasta el momento en el que se produce la cancelación del préstamo.

### 4.3. Problema de la programación de generación para cubrir la demanda de un sistema eléctrico (*Unit Commitment*)

Se trata de la programación de los grupos de generación en un sistema eléctrico para cubrir la demanda en cada hora, optimizando los costes de generación de arranque y de parada y teniendo en cuenta que hay restricciones tales como los límites de generación, de rampas de subida y bajada, horas de encendido y de apagado entre otros.

Para ello se ha utilizado el siguiente prompt.

*Can you provide me a python code that solves the unit committment for 5 generators with next data: hourly demand in MW for 24 hours [550, 572, 539, 517, 495, 484, 506, 550, 605, 660, 825, 880, 825, 781, 759, 737, 704, 671, 638, 616, 594, 583, 572, 561] ; minimum power of the generators in MW: [50, 50, 50, 50, 50]; maximum power of the generators in MW: [200, 150, 500, 500, 400], start-up cost for each of the generators: [100, 50, 20, 20, 10] in \$/MW, shut-down cost for each of the generators: [50, 45, 20, 20, 10] in \$/MW, fuel cost for each of the generators [2, 1, 3, 3, 1] in \$/MW, maximum ramp-up for each of the generators: [20, 20, 70, 70, 60] in MW/hour, minimum ramp-down for each of the generators [20, 20, 60, 60, 60] in MW/hour, minimum up-time hours for each generator: [2, 2, 1, 1, 1] in hours; minimum down-time hours for each of the generator [2, 2, 1, 1, 1] in hours? Can you show the generation schedule for each of the generators ? Can you plot generation time series that is fully adjusted for the demand? Can you formulate the problem?*

La formulación del problema del *Unit Commitment* consiste en establecer la programación del status de los generadores que minimizan el coste total operacional mientras se satisfacen las restricciones operativas. Como variables de decisión se considera el status de los grupos, la potencia generada, el arranque del grupo y el apagado del grupo. Como función objetivo se establece minimizar la suma de costes de generación, arranque y apagado. Se establecen restricciones de cubrir la demanda, límites de generación, rampas de subida y de bajada, restricciones de arranque y apagado, y restricciones de tiempo de encendido y de apagado.

El código generado se muestra en la Figura 8:

```
import numpy as np
import matplotlib.pyplot as plt
import pulp
```

```
# Problem definition
time_horizon = 24 # 24-hour scheduling
num_generators = 5

demand = [550, 572, 539, 517, 495, 484,\
          506, 550, 605, 660, 825, 880,\
          825, 781, 759, 737, 704, 671,\
          638, 616, 594, 583, 572, 561] # Hourly demand
spinning_reserve = 0.0 # 10% of demand

# Generator parameters
P_min = [50, 50, 50, 50, 50] # Min output of generators
P_max = [200, 150, 500, 500, 400] # Max output of generators
startup_cost = [100, 50, 20, 20, 10] # Start-up costs
shutdown_cost = [50, 45, 20, 20, 10] # Shut-down costs
fuel_cost = [2, 1, 3, 3, 1] # Linear fuel cost ($/MWh)
ramp_up = [20, 20, 70, 70, 60] # Max ramp-up
ramp_down = [20, 20, 60, 60, 60] # Max ramp-down
min_up_time = [2, 2, 1, 1, 1] # Min up-time (hours)
min_down_time = [2, 2, 1, 1, 1] # Min down-time (hours)

# Define problem
prob = pulp.LpProblem("Unit_Commitment", pulp.LpMinimize)

# Decision variables
commitment = pulp.LpVariable.dicts("Commit", ((g, t) for g in range(num_generators)\
                                                for t in range(time_horizon)), cat='Binary')
startup = pulp.LpVariable.dicts("Startup", ((g, t) for g in range(num_generators)\
                                                for t in range(time_horizon)), cat='Binary')
shutdown = pulp.LpVariable.dicts("Shutdown", ((g, t) for g in range(num_generators)\
                                                for t in range(time_horizon)), cat='Binary')
power = pulp.LpVariable.dicts("Power", ((g, t) for g in range(num_generators)\
                                                for t in range(time_horizon)), lowBound=0, cat='Continuous')

# Objective function (minimize total cost)
prob += pulp.lpSum(fuel_cost[g] * power[g, t] +\
                  startup_cost[g] * startup[g, t] \
                  for g in range(num_generators) for t in range(time_horizon))

# Constraints
for t in range(time_horizon):
    # Demand + spinning reserve constraint
    prob += pulp.lpSum(power[g, t] for g in range(num_generators)) >= demand[t] + spinning_reserve *
    demand[t]
    prob += pulp.lpSum(power[g, t] for g in range(num_generators)) <= sum(P_max)

    for g in range(num_generators):
        # Power limits
```

```
prob += power[g, t] >= P_min[g] * commitment[g, t]
prob += power[g, t] <= P_max[g] * commitment[g, t]
if t > 0:
    # Ramp rate constraints
    prob += power[g, t] - power[g, t-1] <= ramp_up[g]
    prob += power[g, t-1] - power[g, t] <= ramp_down[g]

    # Startup/shutdown constraints
    prob += startup[g, t] >= commitment[g, t] - commitment[g, t-1]
    prob += shutdown[g, t] >= commitment[g, t-1] - commitment[g, t]
# Solve the problem
prob.solve()
# Extract results
generation = np.zeros((num_generators, time_horizon))
commitment_schedule = np.zeros((num_generators, time_horizon))
for g in range(num_generators):
    for t in range(time_horizon):
        generation[g, t] = power[g, t].varValue
        commitment_schedule[g, t] = commitment[g, t].varValue
times = range(time_horizon)
# Print generation schedule for each generator
print("\nGeneration Schedule (MW):")
print("Time\t" + "\t".join([f"Gen {g+1}" for g in range(num_generators)]))
for t in range(time_horizon):
    print(f"{t}\t" + "\t".join([f"{generation[g, t]:.2f}" for g in range(num_generators)]))

def plot_results():
    plt.figure(figsize=(12, 6))
    plt.plot(times, demand, label='Demand', linestyle='--', marker='o', color='black')
    plt.stackplot(times, generation, labels=[f'Gen {i+1}' for i in range(num_generators)], alpha=0.7)
    plt.xlabel("Time (hours)")
    plt.ylabel("Power (MW)")
    plt.title("Unit Commitment Schedule")
    plt.legend()
    plt.show()
plot_results()
```

Figura 8. Código generado para la solución del problema del *Unit Commitment*.

A la vista del código generado de la Figura 8, se observa que se hace uso de las librerías *numpy*, *matplotlib* y *pulp*. La librería *pulp* es una librería de optimización para modelar y resolver problema MILP (*mixed-integer linear programming*).

Cada generador tiene:

- $P_{min}/P_{max}$ : generación mínima y máxima

- *startup\_cost/shutdown\_cost*: costes fijos para el encendido y apagado de la generación
- *fuel\_cost*: coste variable por MWh producido
- *ramp\_up/ramp\_down*: máximo cambio horario de potencia permitido

El código establece como función objetivo la minimización del coste total. Se definen como variables de decisión:

*commitment*[*g,t*]: Binaria (0/1), 1 si el generador *g* está encendido a la hora *t*

*startup*[*g,t*]: Binaria, si el generador *g* arranca a la hora *t*

*shutdown*[*g,t*]: Binaria, 1 si se apaga el generador *g* a la hora *t*

*power*[*g,t*]: Potencia generada que el generador *g* produce a la hora *t*

Las restricciones contemplan: generación dentro de los límites, restricciones de rampas de subida y de bajada, restricciones sobre la generación comprometida para que el arranque se realice o se realice el apagado

La ejecución del código permite obtener la programación horaria de los grupos de generación como se ve en la tabla de la figura 9 y un gráfico en el que se muestra que la solución obtenida cubre la demanda como se muestra en la figura 10.

| Generation Schedule (MW): |        |        |       |        |        |
|---------------------------|--------|--------|-------|--------|--------|
| Time                      | Gen 1  | Gen 2  | Gen 3 | Gen 4  | Gen 5  |
| 0                         | 50.00  | 142.00 | 0.00  | 0.00   | 358.00 |
| 1                         | 50.00  | 122.00 | 0.00  | 0.00   | 400.00 |
| 2                         | 50.00  | 102.00 | 0.00  | 0.00   | 387.00 |
| 3                         | 50.00  | 82.00  | 0.00  | 0.00   | 385.00 |
| 4                         | 50.00  | 62.00  | 0.00  | 0.00   | 383.00 |
| 5                         | 50.00  | 82.00  | 0.00  | 0.00   | 352.00 |
| 6                         | 55.00  | 90.00  | 0.00  | 0.00   | 361.00 |
| 7                         | 75.00  | 110.00 | 0.00  | 0.00   | 365.00 |
| 8                         | 95.00  | 130.00 | 0.00  | 0.00   | 380.00 |
| 9                         | 115.00 | 150.00 | 0.00  | 0.00   | 395.00 |
| 10                        | 135.00 | 150.00 | 70.00 | 70.00  | 400.00 |
| 11                        | 155.00 | 150.00 | 65.00 | 110.00 | 400.00 |
| 12                        | 175.00 | 150.00 | 50.00 | 50.00  | 400.00 |
| 13                        | 179.00 | 150.00 | 52.00 | 0.00   | 400.00 |
| 14                        | 159.00 | 150.00 | 50.00 | 0.00   | 400.00 |
| 15                        | 139.00 | 148.00 | 50.00 | 0.00   | 400.00 |
| 16                        | 154.00 | 150.00 | 0.00  | 0.00   | 400.00 |
| 17                        | 134.00 | 144.00 | 0.00  | 0.00   | 393.00 |
| 18                        | 114.00 | 124.00 | 0.00  | 0.00   | 400.00 |
| 19                        | 94.00  | 130.00 | 0.00  | 0.00   | 392.00 |
| 20                        | 74.00  | 150.00 | 0.00  | 0.00   | 370.00 |
| 21                        | 54.00  | 150.00 | 0.00  | 0.00   | 379.00 |
| 22                        | 50.00  | 130.00 | 0.00  | 0.00   | 392.00 |
| 23                        | 50.00  | 111.00 | 0.00  | 0.00   | 400.00 |

Figura 9. Programación de los cinco grupos una vez resuelto el problema del *Unit Commitment* en forma tabular.

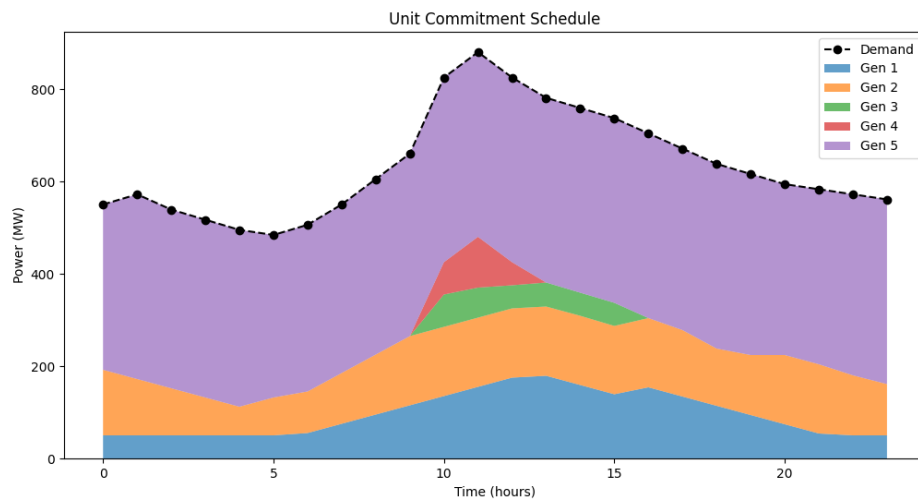


Figura 10. Programación de los cinco grupos una vez resuelto el problema del *Unit Commitment* de forma gráfica

## 5. CONCLUSIONES DEL TRABAJO REALIZADO

- El uso de la IAG se presenta en este trabajo como una técnica interesante para abordar problemas matemáticos y económicos.
- Se ha llevado a cabo el análisis de diferentes ejemplos de problemas matemáticos y de economía aplicada mediante el uso de la IAG. La metodología llevada a cabo permite mediante la creación de un prompt, y el uso de un IDE podemos obtener un software que permite resolver el problema.
- Todos los prompts generados han sido fruto de una elaboración detallada para la especificación de cada uno de los problemas presentados.
- Se ha utilizado un IDE para la ejecución del software generado y para validar los resultados obtenidos
- El esfuerzo fundamental en el marco de este trabajo se centra en la elaboración del prompt y la ejecución del software en un entorno de desarrollo que permite depurar el software, analizar los posibles errores y optimizar el desarrollo software
- Los conocimientos necesarios para hacer uso de la tecnología IAG, radican en la ingeniería del software en lo que se refiere a los IDEs y al

conocimiento del lenguaje de programación, así como del dominio que se trata para poder validar la solución ofrecida.

## 6. REFERENCIAS BIBLIOGRÁFICAS

- ASHISH, K., NISUMBA, S., RAMKUMAR, R. (2025) “Enhancing data analysis and programming skills through structured prompt training: The impact of generative AI in engineering education”. *Computers and Education: Artificial Intelligence: Volume 8*.
- FOGGIA, A., IMPROTA, C., LIGUORI P. (2024) “Automating the Correctness Assessment of AI-Generated Code for Security Purposes”. *Journal of Systems and Software. Volumen 216*
- NAZIR, A., WANG, Z. (2023) “A comprehensive survey of ChatGPT: Advancements, applications, prospects, and challenges”. *Meta-Radiology. Volumen 1 Issue 2*
- PARK, A., TAEKYUNG, K. (2025) “Code suggestions and explanations in programming learning: Use of ChatGPT and performance”. *The International Journal of Management Education. Volumen 23, Issue 2*.
- LIGUORI, P., IMPROTA, C., NATELLA, R., CUKIC B (2023). “Who evaluates the evaluators? On automatic metrics for assessing AI-based offensive code generators”. *Expert Systems with Applications. Volumen 225*
- PERES, R., SCHREIR, M., SWEIDEL, D., SORESCU, A. “On ChatGPT and beyond: How generative artificial intelligence may affect research, teaching, and practice”. *International Journal of Research in Marketing. Volumen 40. Issue 2*
- RAMAZAN, Y., FATMA G (2024).”The effect of generative artificial intelligence (AI)-based tool use on students' computational thinking skills, programming self-efficacy and motivation”. *Computers and Education: Artificial Intelligence. Volumen 4*

- LIM, W., GUNASEKARA, A., PALLANT, J.L., PALLANT, J.I., PACHENKINA, E. (2023). “Generative AI and the future of education: Ragnarok or reformation? A paradoxical perspective from management educators”. *International Journal of Management Education*. Volume 21. Issue 2
- KIZILCEC, R., HUBER, E. y otros (2024). “Perceived impact of generative AI on assessments: Comparing educator and student perspectives in Australia, Cyprus, and the United States. *Computers and Education: Artificial Intelligence*. Volumen 7
- KOHNKE, L., MOORHOUSE, B., ZOU, D. (2023) Exploring generative artificial intelligence preparedness among university language instructors: A case study. *Computers and Education: Artificial Intelligence*. Volumen 5.
- ANAND, N., AJANTHA, D., KULDEEP, K. (2025). “Mastering Prompt Engineering. Deep Insights for Optimizing Large Language Models (LLMs)”. Elsevier. ISBN: 9780443339042
- KHALIFA, M., ALBADAWY, M. (2024).” Using artificial intelligence in academic writing and research: An essential productivity tool”. *Computer Methods and Programs in Biomedicine Update*. Volumen 5.
- GUPTA, A., SHIVERS-MCNAIR, A.(2024) “ “Wayfinding” through the AI wilderness: Mapping rhetorics of ChatGPT prompt writing on X (formerly Twitter) to promote critical AI literacies”. *Computers and Composition*. Volumen 74.