Proyecto de Modelos Matemáticos Aplicados

Ana Karla Caballero Gonzales C411
Alejandro Camacho Pérez C412

1 Introduction

En una investigación anterior realizada en la facultad de Matemática y Computación de la Universidad de La Habana, MATCOM, se proponía basándose en trabajos anteriores varios modelos matemáticos para modelar y resolver problemáticas relacionadas con el acto sexual. En el mismo se plantean los siguientes 5 problemas:

1. Maximizar la duración del acto sexual.
2. Maximizar el placer del que menor placer alcance al finalizar el acto sexual.
3. Minimizar el cansancio del participante con mayor cansancio al finalizar el acto sexual.
4. Minimizar la energía inicial de todos los participantes de forma que al terminar todos hayan alcanzado el orgasmo y tengan la misma energía.
5. Maximizar el placer inicial de un participante específico, de forma tal que todos los participantes, excepto el específico, alcancen el orgasmo.

El objetivo de este trabajo, es hacer una implementación, utilizando un lenguaje de programación, de los modelos matemáticos propuestos en dicha investigación.

2 Cuestiones Generales de los problemas

La investigación realizada utilizó los siguientes parámetros:

2.1 Datos

- $J = \{1 \dots j\}$: Conjunto de participantes.
- $N = \{1 \dots n\}$: Posturas a adoptar
- A_{ij}: Energía de la persona j al terminar la postura i $\forall i = \overline{1,n}$.
- P_{ij}: Placer de la persona j al terminar la postura i. i $\forall i = \overline{1,n}$. y i $\forall j \in J$.
- $\widehat{p}_j$: Placer necesario para que la persona j alcance el orgasmo. $\forall j \in J$.
- T_i: Tiempo empleado en la postura i. $\forall i = \overline{1,n}$.
- C_{ij} : Cantidad de energía que consume la postura i al participante j por unidad de tiempo $\forall i = \overline{1,n}$ y $\forall j \in J$.
- Pa_{ij}:Cantidad de placer que otorga la postura i al participante j por unidad de tiempo. $\forall i = \overline{1,n}$ y $\forall j \in J$

2.2 Variables

- X_i: Tiempo empleado en la postura i. $\forall i = \overline{1,n}$.
- Pp_{ok}: Placer inicial del participante k, $k \in J$. (Sólo el problema 5)
- E_{0j}: Energía inicial de la persona j. $\forall j \in J$. (Sólo el problema 4)

2.3 Restricciones comunes

- Después de cada postura, la energía disminuye de manera proporcional al tiempo que se permanezca en ella:

$$A_{ij} = A_{(i-i)j} - C_{ij}X_i \ \forall i \in N \ \forall j \in J$$

- Después de cada postura, el placer de cada participante aumenta de manera proporcional al tiempo que se permanezca en ella:

$$P_{ij} = P_{(i-1)j} + P_{ij}X_i \ \forall i \in N \ \forall j \in J$$

- En todo momento la energía es mayor igual que cero

$$A_{ij} \geq 0 \ \forall i \in N \ \forall j \in J$$

- El placer después del acto sexual es mayor o igual al placer necesario para alcanzar el orgasmo

$$P_{nj} \geq \widehat{P}_j \ \forall j \in J$$

3 Sobre la implementación

La implementación del trabajo fue hecha en Python, utilizando la librería pulp para la resolución de problemas de programación lineal. El mismo consta de dos módulos principales, uno para la interfaz de usuario y la comunicación del mismo con el programa, y otro para la resolución de los problemas.

3.1 Módulo de interfaz de usuario

La interfaz de usuario ofrece la posibilidad de:

- Crea, editar y eliminar participantes.
- Crea, editar y eliminar posturas.
- Listar participantes y posturas con todos sus detalles.
- Exportar e importar los datos de los participantes y posturas a un archivo de json.
- Seleccionar y resolver uno de los problemas planteados.

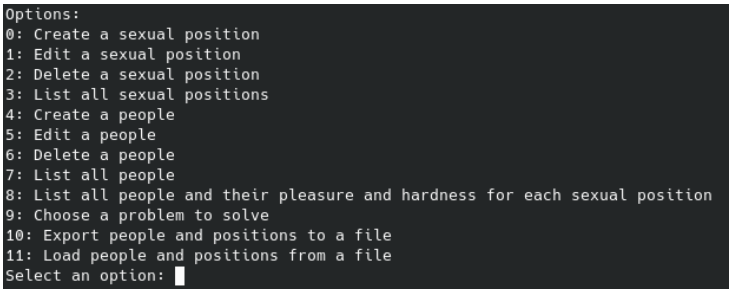


Figure 1: Interfaz de usuario

3.1.1 Uso de la interfaz de usuario

La interfaz de usuario sigue las siguientes constantes:

- Para la creación de una nueva postura es necesario sólo indicar su nombre
- Para la creación de un nuevo participante es necesario indicar su nombre, placer necesario para alcanzar el orgasmo y energía inicial. Así cómo también los valores de energía y placer para cada postura.
- Tras modificar un participante, se pedirá actualizar los valores de energía y placer para cada postura.
- Tras añadir una postura si existían participantes, se pedirá actualizar los valores de energía y placer para cada participante.
- La letra c cancela la acción que se esté realizando por completo.

Tras tener al menos un participante y una postura, se puede seleccionar uno de los problemas planteados y resolverlo. El programa entonces pasa a enviar los datos en un formato adecuado al módulo de resolución de problemas.

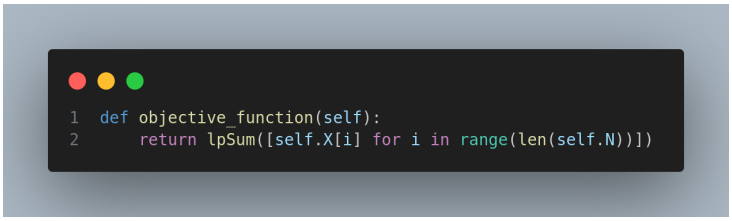
3.2 Módulo de resolución de problemas

El módulo de resolución de problemas consta de una clase abstracta, AbstractProblem, que encapsula el procedimiento para cada problema, utilizando los métodos objetive_function, constraints y solve. Cada problema

```python
class AbstractProblem:
    def __init__(self, peoples: list[People], positions: list[SexualPosition]):
        self.J = peoples
        self.N = positions
        self.P_t = [people.orgasm_pleasure for people in peoples]
        self.C = self._build_C()
        self.Pa = self._build_P()

    def _build_C(self) -> list[list[int]]:
        C = [[0 for _ in range(len(self.J))] for _ in range(len(self.N))]

        for i, position in enumerate(self.N):
            for j, people in enumerate(self.J):
                C[i][j] = people.get_hardness_of_position(position.name)

        return C

    def _build_P(self) -> list[list[int]]:
        P = [[0 for _ in range(len(self.J))] for _ in range(len(self.N))]

        for i, position in enumerate(self.N):
            for j, people in enumerate(self.J):
                P[i][j] = people.get_pleasure_of_position(position.name)

        return P

    def objective_function(self):
        """
        The objective function of the problem to solve
        """
        raise NotImplementedError("objective_function method must be implemented")

    def constraints(self):
        """
        The constraints of the problem to solve
        """
        raise NotImplementedError("constraints method must be implemented")

    def solve(self):
        """
        Solve the problem
        """
        raise NotImplementedError("solve method must be implemented")
```

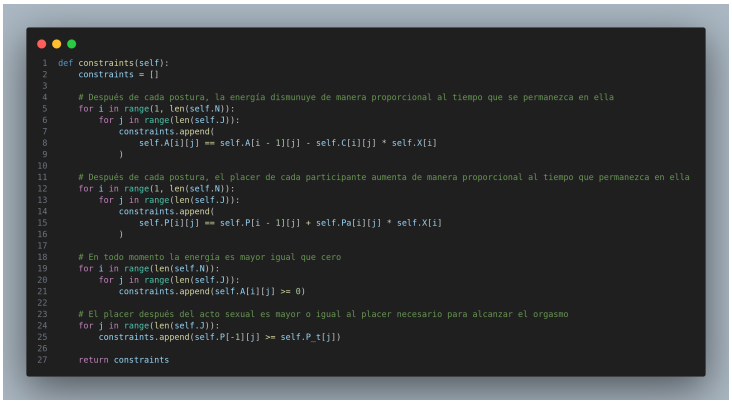
Figure 2: Clase AbstractProblem

hereda de esta clase abstracta y sobrescribe los métodos mencionados. Para poder resolver un problema, es necesaria la creación de una clase que herede de AbstractProblem y sobrescriba los métodos mencionados.



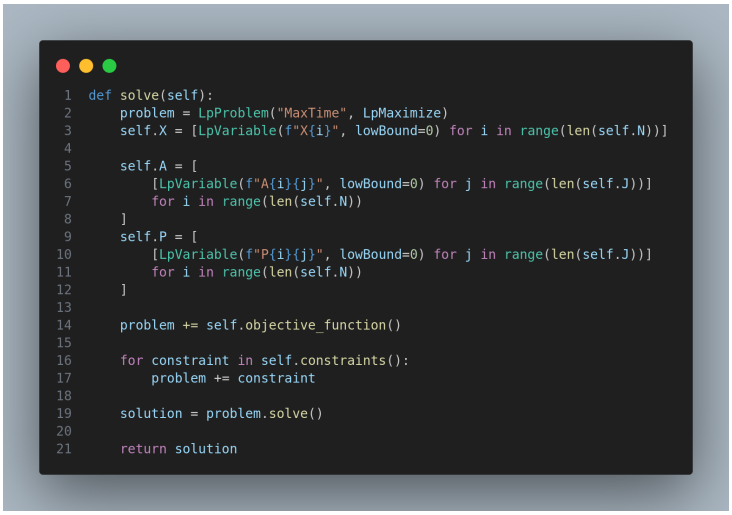
```python
def objective_function(self):
    return lpSum([self.X[i] for i in range(len(self.N))])
```

Figure 3: Función objetivo para el problema 1



```python
def constraints(self):
    constraints = []

    # Después de cada postura, la energía dismunuye de manera proporcional al tiempo que se permanezca en ella
    for i in range(1, len(self.N)):
        for j in range(len(self.J)):
            constraints.append(
                self.A[i][j] == self.A[i - 1][j] - self.C[i][j] * self.X[i]
            )

    # Después de cada postura, el placer de cada participante aumenta de manera proporcional al tiempo que permanezca en ella
    for i in range(1, len(self.N)):
        for j in range(len(self.J)):
            constraints.append(
                self.P[i][j] == self.P[i - 1][j] + self.Pa[i][j] * self.X[i]
            )

    # En todo momento la energía es mayor igual que cero
    for i in range(len(self.N)):
        for j in range(len(self.J)):
            constraints.append(self.A[i][j] >= 0)

    # El placer después del acto sexual es mayor o igual al placer necesario para alcanzar el orgasmo
    for j in range(len(self.J)):
        constraints.append(self.P[-1][j] >= self.P_t[j])

    return constraints
```

Figure 4: Restricciones para el problema 1

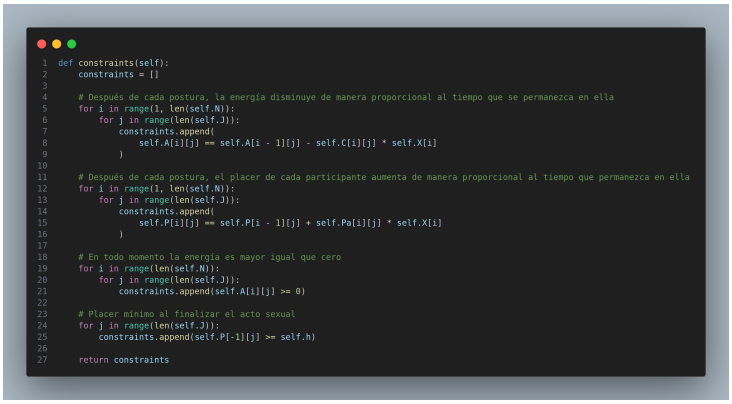


```python
def solve(self):
    problem = LpProblem("MaxTime", LpMaximize)
    self.X = [LpVariable(f"X{i}", lowBound=0) for i in range(len(self.N))]

    self.A = [
        [LpVariable(f"A{i}{j}", lowBound=0) for j in range(len(self.J))]
        for i in range(len(self.N))
    ]
    self.P = [
        [LpVariable(f"P{i}{j}", lowBound=0) for j in range(len(self.J))]
        for i in range(len(self.N))
    ]

    problem += self.objective_function()

    for constraint in self.constraints():
        problem += constraint

    solution = problem.solve()

    return solution
```

Figure 5: Resolución del problema 1

```python
def __init__(self, peoples: list[People], positions: list[SexualPosition]):
    super().__init__(peoples, positions)
    self.h = LpVariable("h", cat="Continuous")

def objective_function(self):
    return self.h
```

Figure 6: Función objetivo para el problema 2



```python
def constraints(self):
    constraints = []

    # Después de cada postura, la energía disminuye de manera proporcional al tiempo que se permanezca en ella
    for i in range(1, len(self.N)):
        for j in range(len(self.J)):
            constraints.append(
                self.A[i][j] == self.A[i - 1][j] - self.C[i][j] * self.X[i]
            )

    # Después de cada postura, el placer de cada participante aumenta de manera proporcional al tiempo que permanezca en ella
    for i in range(1, len(self.N)):
        for j in range(len(self.J)):
            constraints.append(
                self.P[i][j] == self.P[i - 1][j] + self.Pa[i][j] * self.X[i]
            )

    # En todo momento la energia es mayor igual que cero
    for i in range(len(self.N)):
        for j in range(len(self.J)):
            constraints.append(self.A[i][j] >= 0)

    # Placer mínimo al finalizar el acto sexual
    for j in range(len(self.J)):
        constraints.append(self.P[-1][j] >= self.h)

    return constraints
```

Figure 7: Restricciones para el problema 2

```python
def solve(self):

    problem = LpProblem("MaxMinPleasure", LpMaximize)

    self.X = [LpVariable(f"X{i}", lowBound=0) for i in range(len(self.N))]
    self.A = [
        [LpVariable(f"A{i}{j}", lowBound=0) for j in range(len(self.J))]
        for i in range(len(self.N))
    ]
    self.P = [
        [LpVariable(f"P{i}{j}", lowBound=0) for j in range(len(self.J))]
        for i in range(len(self.N))
    ]

    problem += self.objective_function()

    for constraint in self.constraints():
        problem += constraint

    solution = problem.solve()

    return solution
```

Figure 8: Resolución del problema 2

4 Consideraciones

Primeramente, abarcar todos los problemas planteados en la investigación no fue posible, por lo que se decidió implementar los problemas 1 y 2. La implementación de los problemas 3, 4 y 5 se deja como trabajo futuro.
Los problemas pueden ser modelados teniendo en cuenta la preferencia sexual de cada participante, es decir, si el participante J_i está dispuesto a ejecutar la postura dada con el participante J_j, con $0 \leq i < j \leq |J|$.