Universidad Politécnica
de Madrid

**Escuela Técnica Superior de
Ingenieros Informáticos**

Grado en Matemáticas e Informática

Trabajo Fin de Grado

Algoritmos para el Cálculo de Invariantes de Nudos

Autor: Ignacio Espinosa Fernández
Tutor(a): Héctor Barge Yañez y Ramón Barral

Madrid, Enero 2025

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado
Grado en Matemáticas e Informática

Título: Algoritmos para el Cálculo de Invariantes de Nudos

Enero 2025

Autor: Ignacio Espinosa Fernández
Tutor: Héctor Barge Yañez y Ramón Barral
Departamento de Matemática Aplicada a las Tecnologías de la Información y las Comunicaciones
Escuela Técnica Superior de Ingenieros Informáticos
Universidad Politécnica de Madrid

Dedico este trabajo a todas las personas
que me han apoyado a lo largo de esta etapa.

Resumen

La teoría de nudos es una rama de la topología que estudia un tipo específico de objeto matemático denominado nudo. Un nudo se asemeja a los que encontramos en nuestra vida cotidiana, pero con los extremos unidos, de modo que puede representarse como una curva simple y cerrada en $\mathbb{R}^3$.

Un aspecto fundamental de esta rama es determinar si dos nudos son equivalentes, es decir, se estudia si se puede deformar uno en el otro de manera continua sin pasar sobre sí mismo. Resolver este problema no es sencillo, y se aborda mediante el estudio de las invariantes de cada nudo para poder diferenciarlos. Dos invariantes importantes en este sentido e íntimamente relacionados son el grupo del nudo y el polinomio de Alexander.

El objetivo de este Trabajo de Fin de Grado es estudiar e investigar la teoría de nudos, especialmente el grupo del nudo y el polinomio de Alexander, e implementar los algoritmos en Python para calcular las presentaciones de cada uno de ellos.

Abstract

Knot theory is a branch of topology that studies a specific type of mathematical object called a knot. A knot resembles those we encounter in daily life but with the ends joined together, so it can be represented as a simple closed curve in $\mathbb{R}^3$.

A fundamental aspect of this branch is determining whether two knots are equivalent, meaning the study focuses on whether one can be deformed into the other continuously without crossing itself. Solving this problem is not straightforward and is approached through the study of knot invariants to differentiate them. Two important and closely related invariants in this context are the knot group and the Alexander polynomial.

The aim of this Final Degree Project is to study and research knot theory, particularly the knot group and the Alexander polynomial, and to implement algorithms in Python to compute the presentations of each.

Tabla de contenidos

Capítulo 1

Introducción

La rama de las matemáticas que se estudia en este TFG es la teoría de nudos. Este área entra dentro de la topología y se encarga de estudiar los nudos y sus propiedades matemáticas, como por ejemplo cómo se pueden entrelazar y manipular los nudos en el espacio tridimensional, considerando que estos pueden deformarse sin cortarse ni unirse.

En esta teoría, un nudo se define como un bucle cerrado en el espacio tridimensional, es decir, un lazo de cuerda cuyos extremos están unidos, formando así una figura cerrada. Dos nudos se consideran equivalentes si se pueden deformar uno en el otro de manera continua, es decir, sin cortar ni unir partes de la cuerda.

Los nudos pueden tener una estructura complicada y no es fácil diferenciar dos a simple vista, es por ello, que para clasificar y diferenciar nudos, se usan unas características de cada nudo llamadas invariantes. Estos son números, polinomios u otras estructuras matemáticas que no cambian aunque el nudo se deforma. En este trabajo se estudiará y se trabajará sobre dos de estos invariantes, el grupo fundamental del nudo y el polinomio de Alexander.

La forma en la que se representan gráficamente los nudos es una proyección en $\mathbb{R}^2$ del nudo en $\mathbb{R}^3$. De esta forma, podemos ver los cruces del nudo, que serán fundamentales para el cálculo de estos dos invariantes.

Para obtener el grupo fundamental del nudo utilizaremos la presentación de Wirtinger, un método algebraico que nos permitirá obtener una presentación del invariante en términos de generadores y relaciones.

El algoritmo para conseguir el polinomio de Alexander sigue un procedimiento similar al de la presentación de Wirtinger, pero obtendremos una matriz como presentación del grupo fundamental, a partir de la cual, se puede conseguir el polinomio de Alexander.

Para el estudio de estos invariantes, el trabajo se apoya en las referencias [1, 2, 3].

En el trabajo, aparte de estudiar estos invariantes, se programarán una clase

en el lenguaje Python que nos permitirá introducir nudos, así como algoritmos para calcularlos.

Capítulo 2

Marco Teórico

2.1. Definición de nudo

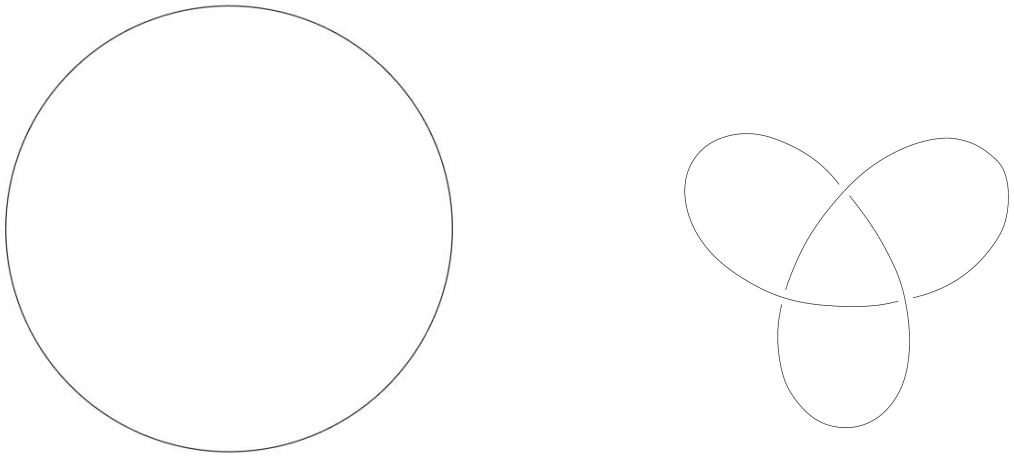
Los nudos son un elemento matemático que entra dentro de la teoría de nudos.

Definición 1 (Embebimiento)**.** Sea X e Y espacios topológicos, un **embebimiento** de X en Y es una función $f : X \to Y$ tal que $f(X)$ hereda la topología de Y, y la función f es un homeomorfismo entre X y $f(X)$.

Definición 2 (Nudo)**.** Un **nudo** es un embebimiento de la circunferencia S^1 en el espacio tridimensional euclídeo $\mathbb{R}^3$. Matemáticamente, un nudo se representa como una función $f : S^1 \to \mathbb{R}^3$ tal que f es un embebimiento.

Esto es equivalente a coger una cuerda con un nudo de nuestro día a día y uniésemos sus extremos de modo que ya no pudiésemos deshacerlo.

Los dos nudos más sencillos en esta teoría son el **nudo trivial** (Figura 2.1a) y el **nudo trébol** (Figura 2.1b).



(a) Nudo trivial.

(b) Nudo trébol.

Figura 2.1: Ejemplos de nudos.

A lo largo de este trabajo, utilizaremos el término nudo tanto para referirnos a la aplicación f como a su imagen. El contexto permitirá diferenciar entre ambos.

2.2. Diagrama de un nudo

Es importante que seamos capaces de representar un nudo de forma visual para trabajar con él. La herramienta que vamos a utilizar se conoce como el diagrama de un nudo.

Definición 3 (Diagrama de un nudo)**.** Dado un nudo K en el espacio tridimensional $\mathbb{R}^3$, un **diagrama de nudo** es una proyección $\pi : \mathbb{R}^3 \to \mathbb{R}^2$ de K orientada, tal que:

- $\pi(K)$ es inyectiva en todos los puntos excepto en un número finito de puntos llamados cruces.
- En cada cruce $p \in \mathbb{R}^2$ se puede distinguir que parte de K pasa por encima y cuál por debajo.

Sea K un nudo, podemos separarlo en dos clases de arcos segmentados cerrados y conectados que llamamos *overpasses* y *underpasses*, que se alternan alrededor del nudo. Es decir, cada punto del nudo pertenece a un *underpass* o a un *overpass*. La subdivisión se elige de tal manera que ningún *overpass* contenga un subcruce y ningún *underpass* contenga un sobrecruce. Esta separación se puede hacer de muchas formas diferentes pero la que vamos a escoger siempre es la que tenga el menor número de *underpasses* y de *overpasses*.

- A los *overpasses* los denotaremos como: A_1,...,A_n y su unión $A_1 \cup ... \cup A_n$ la llamaremos A.
- A los *underpasses* los denotaremos como B_1,...,B_n y su unión $B_1 \cup ... \cup B_n$ la llamaremos B.

Un ejemplo de los *overpasses* y *underpasses* del nudo sería:

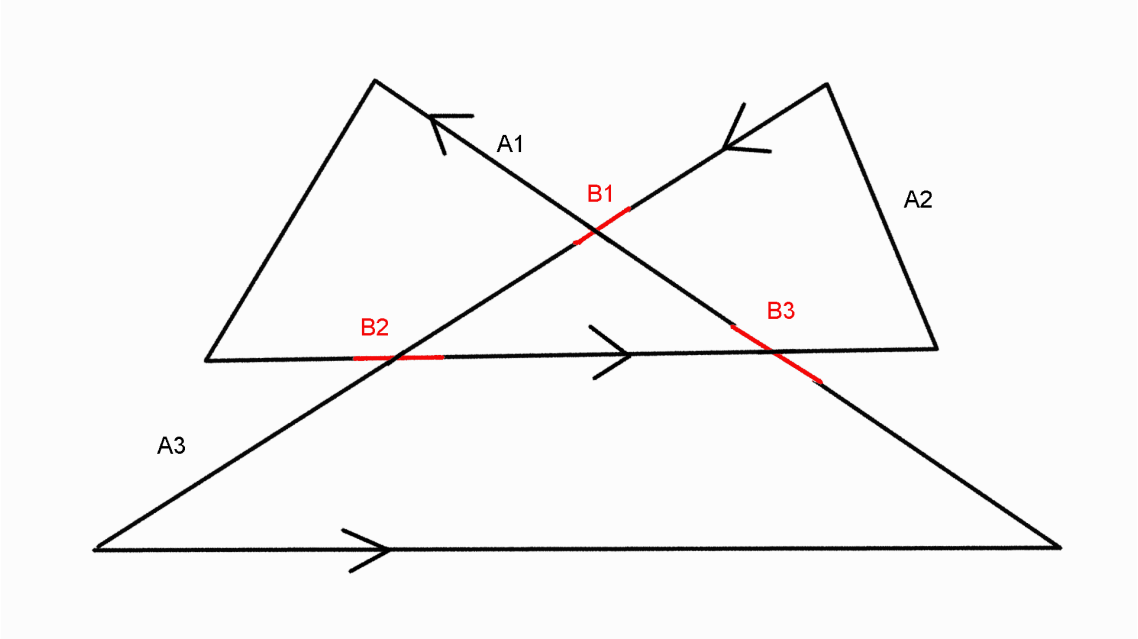


Figura 2.2: *Overpasses* y *underpasses* del nudo Trébol.

2.3. Grupo de un nudo

Definición 4 (Grupo de un nudo)**.** Dado un nudo K en el espacio tridimensional $\mathbb{R}^3$, su **grupo fundamental** es el grupo

$$\pi_1(\mathbb{R}^3 - K, p_0),$$

donde p_0 es un punto fijo fuera del nudo. Este grupo es conocido como el **grupo del nudo**.

Podemos ver que $\pi_1(\mathbb{R}^3 - K, p_0)$ está generado por lazos a_i que rodean cada rama del nudo que hay en un cruce (Figura 2.3).

Esto quiere decir que hay tantos generadores como cruces. Viendo la orientación de K, podemos orientar los generadores a_i alrededor de los arcos α_i utilizando la regla de la mano derecha.

Para hacer un lazo que contenga el cruce utilizamos las relaciones que se generan entre los lazos que rodean a las ramas que se encuentran en cada cruce, que podemos representar de manera algebraica. Depende de la orientación de los arcos existen 2 tipos de cruce, el positivo (Figura 2.4a) y el negativo (Figura 2.4b).

- Un cruce positivo se representa como:

$$a_i a_j^{-1} a_{i+1}^{-1} a_j = 1 \quad \text{o} \quad a_j a_i a_j^{-1} = a_{i+1}$$

- Un cruce negativo se representa como:

$$a_i a_j a_{i+1}^{-1} a_j^{-1} = 1 \quad \text{o} \quad a_j^{-1} a_i a_j = a_{i+1}$$

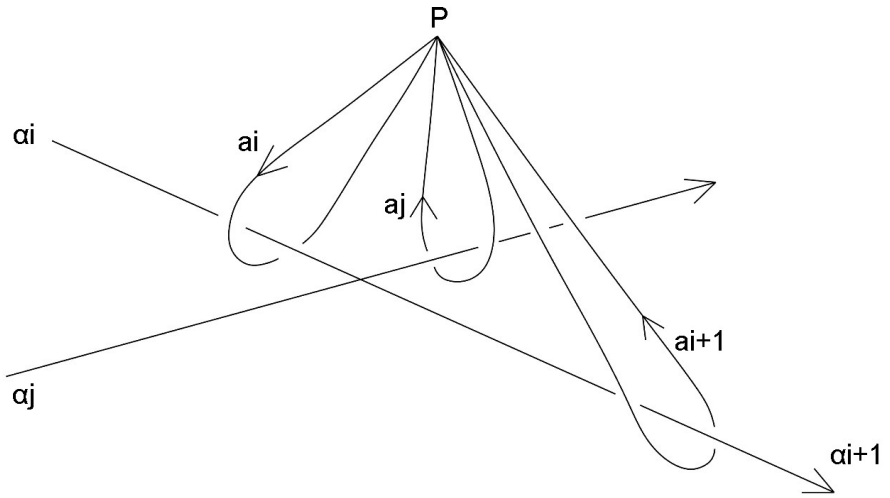


Figura 2.3: Lazos que generan el grupo fundamental de un cruce desde un punto P.

(a) Cruce positivo. (b) Cruce negativo.

Figura 2.4: Tipos de cruces.

Esto será necesario para explicar la presentación de Wirtinger, un algoritmo creado por Wilhelm Wirtinger, que nos permitirá calcular el grupo de un nudo de forma sencilla.

Si dos nudos son equivalentes, sus grupos del nudo son isomorfos, por lo tanto el grupo del nudo es un invariante del nudo.

Definición 5 (Invariante del nudo)**.** Sea K_1 y K_2 nudos, un invariante del nudo

es una propiedad que se mantiene por homeomorfismos ambiente, por lo tanto si K_1 y K_2 son equivalentes,comparten el mismo invariante.

Calcular invariantes de un nudo es útil, ya que hay ocasiones en las que no podemos distinguir si dos nudos son equivalentes a simple vista.

2.4. Presentación de Wirtinger

La presentación de Wirtinger es un método para calcular la presentación del grupo de un nudo utilizando las expresiones algebraicas mostradas en el apartado anterior.

- Primero se asocia un generador a cada arco del diagrama del nudo.
- En cada cruce se establece la relación que describe cómo se conectan los generadores de los arcos que participan en el cruce, basándose en las reglas del apartado anterior.

De esta manera, las relaciones estructuran el nudo y permiten construir su grupo fundamental.

Vamos a calcular la presentación de Wirtinger del nudo Trébol como ejemplo:

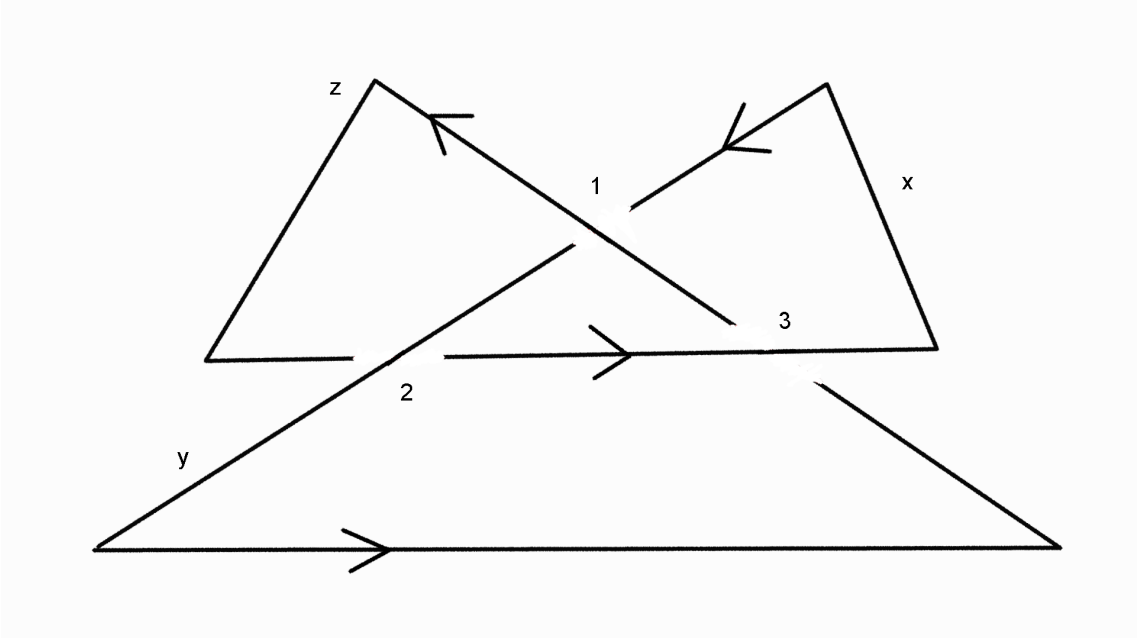


Figura 2.5: Nudo trébol poligonal.

- El cruce 1 queda como: $y = zxz^{-1}$
- El cruce 2 queda como: $x = yzy^{-1}$
- El cruce 3 queda como: $z = xyx^{-1}$

La presentación de Wirtinger se expresaría de la siguiente manera:

$$\pi_1(\mathbb{R}^3 - K) = |x, y, z : x = yzy^{-1}, y = zxz^{-1}, z = xyx^{-1}|$$

La expresión se puede simplificar, sustituimos z por xyx^{-1}:

$$|x, y : x = yxyx^{-1}y^{-1}, y = xyxy^{-1}x^{-1}|$$

$$|x, y : xyx = yxy|$$

2.5. Polinomio de Alexander

El polinomio de Alexander es otro invariante del nudo descubierto por James Waddell Alexander en 1928. Sea K un nudo orientado, con n cruces y n arcos. El algoritmo para calcular el polinomio de Alexander de K es el siguiente:

1. Numeramos cada cruce x y cada arco a.
2. En cada cruce establecemos un valor para cada arco del nudo involucrado:
   - $1 - t$ para el *overpass*.
   - t para el *underpass* entrante.
   - -1 para el *underpass* saliente.
3. Creamos una matriz $n \times n$ en la que introducimos los valores que hemos establecido para el arco a_j en el cruce x_i en la celda ij de la matriz. Si un arco se tiene dos valores en el mismo cruce, se introduce la suma de los valores. Si un arco no tiene un valor en el cruce, se introduce 0 en la celda. A esta matriz la llamaremos **Matriz de Alexander.**
4. Se elimina una columna y una fila de la matriz, obteniendo una matriz $(n-1) \times (n-1)$.
5. Se calcula el determinante de esta matriz. El resultado será un polinomio de variable t.
6. Se normaliza el polinomio haciendo que tenga un término independiente positivo. Este polinomio se conoce como el **Polinomio de Alexander.**

Vamos a calcular el polinomio de Alexander del nudo trébol como ejemplo.

1. Numeramos los arcos y los cruces y establecemos los polinomios:

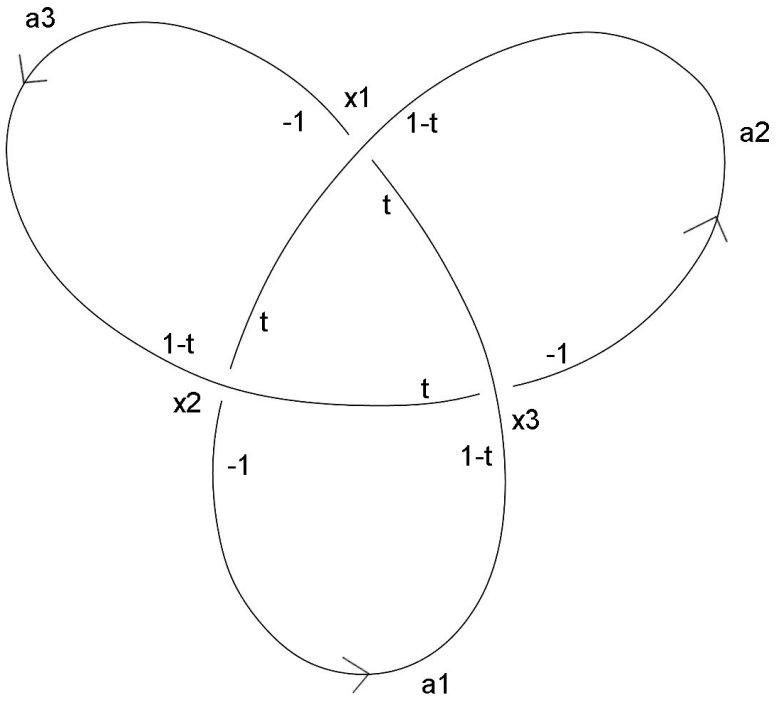


Figura 2.6: Nudo trébol etiquetado.

2. Creamos la matriz de Alexander:

$$\begin{bmatrix} & a_1 & a_2 & a_3 \\ x_1 & t & 1-t & -1 \\ x_2 & -1 & t & 1-t \\ x_3 & 1-t & -1 & t \end{bmatrix}$$

3. Eliminamos la tercera fila y la tercera columna y hacemos el determinante de la matriz:

$$\det \begin{bmatrix} t & 1-t \\ -1 & t \end{bmatrix} = t^2 - (-1)(1-t) = t^2 - t + 1.$$

4. Como el polinomio tiene término independiente positivo no es necesario operar con él para normalizarlo.

Por lo que podemos concluir que el polinomio de Alexander del nudo trébol es $P(K) = t^2 - t + 1$.

Capítulo 3

Código e implementación

3.1. Código

Todo el código del trabajo se ha hecho en el lenguaje de programación Python. Para implementarlo se ha usado Google Colab, un entorno basado en notebooks sencillo de utilizar y con facilidades para introducir diferentes librerías de Python, como la de sympy, que hemos empleado en el trabajo.

3.1.1. Introducir nudos

El primer paso antes de implementar los algoritmos para calcular los distintos invariantes será crear una manera para introducir los nudos en el programa informático. Para hacerlo, crearemos una clase Knot que crea nudos introduciendo como paramentros una lista con los arcos del nudo y una lista con los cruces. Para definir los cruces, haremos una clase Cruces que usará como parámetros el arco entrante del underpass, el arco saliente del underpass, el arco *overpass* y la orientación del nudo. De esta manera, la clase Knot, puede crear un objeto nudo con la suficiente información para calcular los invariantes mediante los algoritmos. A ambas clases les añadimos los métodos getters para obtener sus atributos.

- Clase Knot y sus getters:

```
1  import sympy
2
3  class Knot:
4
5   #La clase knot se inicializa con una lista de los overpasses y
       los underpasses
6    def __init__(self, arcos,cruces):
7      #Crea un nudo introduciendo los arcos y los cruces
8          self.arcos = arcos
9          self.cruces=cruces
10   def printarcos(self):
11     #printea los arcos del nudo
```

```
12        print(self.arcos)
13    def printcruces(self):
14      #printea los curces del nudo
15        n=1
16        for i in self.cruces:
17          print("Cruce␣", n, ":␣\n␣Arco␣entrante:␣", i.
                arcoentrante,"\n␣Arco␣saliente:␣", i.arcosaliente,"\n
                ␣Overpass:␣", i.overpass, "\n␣Orientación:␣", i.
                orientacion)
18          n=n+1
```

- Clase Cruce y sus getters:

```
1  class Cruce:
2
3    def __init__(self, arcoentrante, arcosaliente, overpass,
         orientacion):
4      #Inicializa un cruce con su arco entrante, su arco saliente,
           su overpass y su orientación
5      self.arcoentrante=arcoentrante
6      self.arcosaliente=arcosaliente
7      self.overpass=overpass
8      self.orientacion=orientacion
9    def get_arcoentrante(self):
10     #Devuelve el arco entrante del cruce
11     return self.arcoentrante
12   def get_arcosaliente(self):
13     #Devuelve el arco saliente del cruce
14     return self.arcosaliente
15   def get_overpass(self):
16     #Devuelve el overpass del cruce
17     return self.overpass
```

El código para introducir el nudo trébol (Figura 2.5) sería:

- Código:

```
1  nudotrebol= Knot(["x","y","z"], [Cruce("y","z","x", "+"), Cruce(
       "z","x","y", "+"), Cruce("x","y","z", "+")])
2  nudotrebol.printarcos()
3  nudotrebol.printcruces()
```

- Salida:

```
1  ['x', 'y', 'z']
2  Cruce  1 :
3   Arco entrante:  y
4   Arco saliente:  z
5   Overpass:  x
```

```
 Orientación:  +
Cruce  2 :
 Arco entrante:  z
 Arco saliente:  x
 Overpass:  y
 Orientación:  +
Cruce  3 :
 Arco entrante:  x
 Arco saliente:  y
 Overpass:  z
 Orientación:  +
```

3.1.2. Algoritmo de Wirtinger

Antes de programar el algoritmo vamos a establecer como va a ser la expresión que devuelva la función. La clase Expresion se va a iniciar con:

- **Una lista de generadores:** Se define con una clase Generador, que contiene el arco que identifican.
- **Una lista de relatores:** Se define con una clase Relator que se inicializa con una lista que contiene cada relator. Un relator es otra forma de presentar una relación de igualdad pero con todos los elementos en un lado de la igualdad, por ejemplo:
  - Relación: $y = zxz^{-1}$
  - Relator: $1 = y^{-1}zxz^{-1}$

  Cada elemento de la lista es una tupla que contiene el arco que identifica y su exponente 1 o -1.
- Clase Generador y sus función get:

```
class Generador:
  def __init__(self, generador):
    #Inicializa un generador
    self.generador=generador
  def get_generador(self):
    #Devuelve el generador
    return self.generador[0]
```

- Clase Relator y su función get:

```
class Relator:
  def __init__(self, relator):
    #Inicializa un relator
    self.relator=relator
  def get_relator(self):
    #Devuelve el relator.
    for i in range(len(self.relator)):
```

```
8         if i == 0:
9           output=self.relator[i][0] + "^-1"
10        else:
11          output=output + self.relator[i][0]
12          if self.relator[i][1] == -1:
13            output=output+"^-1"
14        output=output + "␣"
15      return output
```

La función de Wirtinger pertenece a la clase Knot y genera una expresión. Los pasos que realiza son los siguientes:

1. Crea un objeto generador a partir de los arcos del nudo.
2. Inicializa el objeto expresión con los generadores.
3. Por cada cruce va creando el relator correspondiente y lo añade a los relatores de la expresión.
4. Devuelve la expresión.

La clase Expresion tiene funciones getters y adders, para trabajar de forma más sencilla las expresiones, y una función que muestra por pantalla los generadores y relatores de forma clara para el usuario.

- Función wirtinger:

```
1  def wirtinger(self):
2      #Devuelve la expresion de wirtinger con una expresión que
           tiene primero los generadores y luego los relatores de la
            expresión
3      generadores = Generador([self.arcos])
4      expresion=Expresion(generadores,[])
5      if self.cruces:  # si hay cruces
6        for i in self.cruces:
7          if i.orientacion == "+":
8            relator = Relator([[i.get_arcosaliente(),-1], [i.
                 get_overpass(),+1], [i.get_arcoentrante(), +1] , [i
                 .get_overpass(),-1]])
9          else:
10           relator = Relator([[i.get_arcosaliente(),-1],[i.
                 get_overpass(),-1], [i.get_arcoentrante(), +1] , [i
                 .get_overpass(),+1]])
11         expresion.add_relator(relator)
12
13     return expresion
```

- Clase Expresión y sus funciones:

```
1  class Expresion:
2    def __init__(self, generadores,relatores):
```

```
    #Inicializa una expresión con sus generadores y sus
       relatores
    self.generadores=generadores
    self.relatores=relatores

  def get_generadores(self):
    #Devuelve los generadores de la expresión
    return self.generadores
  def get_relatores(self):
    #Devuelve los relatores de la expresión
    return self.relatores


  def add_generador(self, generador):
    #Anade un generador a la expresión
    self.generadores.append(generador)
  def add_relator(self, relator):
    #Anade un relator a la expresión
    self.relatores.append(relator)


  def print_expresion(self):
    #Función que printea la expresión de forma ordenada
       separando entre generadores y relatores.
    if len(self.relatores)==0:
      print("Generadores:␣")
      for generador in self.generadores.get_generador():
        print(generador, end="␣")
      print("\nRelatores:␣")
      print("[␣]")
    else:
      print("Generadores:␣")
      for generador in self.generadores.get_generador():
        print(generador, end="␣")
      print("\nRelatores:␣")
      for relator in self.relatores:
        print(relator.get_relator())
```

El código para calcular la presentación de Wirtinger del nudo trébol quedaría como:

- Código:

```
expresion=nudotrebol.wirtinger()
expresion.print_expresion()
```

- Salida:

```
Generadores:
```

```
x y z
Relatores:
z^-1 x y x^-1
x^-1 y z y^-1
y^-1 z x z^-1
```

3.1.3. Algoritmo del polinomio de Alexander

Para calcular el polinomio de Alexander tenemos 3 funciones:

- **matriz_alexander:** Función que está dentro de la clase Knots que inicializa una matriz, le asigna números a los cruces y va introduciendo el polinomio correspondiente (-1,t,1-t) en cada celda de la matriz.
- **simplificar_pol:** Función auxiliar que simplifica un polinomio para que esté en el estándar del polinomio de Alexander (con término independiente positivo).
- **polinomio_alexander:** Función que está dentro de la clase Knots y que calcula el determinante de la matriz de Alexander creada con matriz_alexander para devolver el polinomio de Alexander, reducido con la función simplificar_pol.
- Función matriz_alexander:

```
def matriz_alexander(self):
    #Devuelve el polinomio de Alexander del nudo.
    t = sympy.symbols('t')
    #Creamos la matriz de cruces.
    n_cruces=len(self.cruces)
    n_arcos=len(self.arcos)
    matriz = sympy.Matrix.zeros(n_cruces, n_arcos)
    #A cada cruce le asignamos un número y vamos buscando sus
       elementos para meterlos en su lugar correspondiente en la
        matriz
    for i, cruce in enumerate(self.cruces):
      arco_entrante = self.arcos.index(cruce.arcoentrante)
      arco_saliente = self.arcos.index(cruce.arcosaliente)
      overpass = self.arcos.index(cruce.overpass)
      #introducimos el polinomio correspondiente en su sitio de
         la matriz
      matriz[i, arco_saliente] = -1
      matriz[i, arco_entrante] = t
      matriz[i, overpass] = 1-t
    return matriz
```

- Función simplificar_pol:

```
  def simplificar_pol(polinomio):
    t = sympy.symbols('t')
```

```
    #Comprobamos que el término independiente no sea 0
    terminoIndependiente=polinomio.as_coeff_add()[0]
    while(terminoIndependiente==0):
      #Dividimos el polinomio entre t hasta que tenga termino
         independiente
      polinomio=sympy.expand(polinomio/t)
      terminoIndependiente=polinomio.as_coeff_add()[0]
      #hacemos positivo el término independiente
    if(terminoIndependiente>0):
      return polinomio
    else:
      return -1*polinomio
```

- Función polinomio_alexander:

```
def polinomio_alexander(self):
    #Función que calcula el determinante de la matriz de
       alexander reducida para devolver el polinomio de
       alexander
    matriz=self.matriz_alexander()
    matriz_reducida = matriz[1:, 1:]
    determinante = matriz_reducida.det()
    polinomio=simplificar_pol(determinante)
    return polinomio
```

El código para calcular la matriz y el polinomio de Alexander del nudo trébol quedaría como:

- Código:

```
matriz=nudotrebol.matriz_alexander()
print(matriz)
print(nudotrebol.polinomio_alexander())
```

- Salida:

```
Matrix([[1 - t, t, -1], [-1, 1 - t, t], [t, -1, 1 - t]])
t**2 - t + 1
```

3.2. Ejemplos de Uso

Para los ejemplos de uso vamos calcular los invariantes de 3 nudos de 5 cruces que a simple vista no podemos asegurar que sean distintos y de un nudo más grande de 9 cruces. Los diagramas que utilizamos están inspirados en los de la página Knot Atlas [4], que se dedica a estudiar los nudos y sus propiedades.

3.2.1. Ejemplo 1

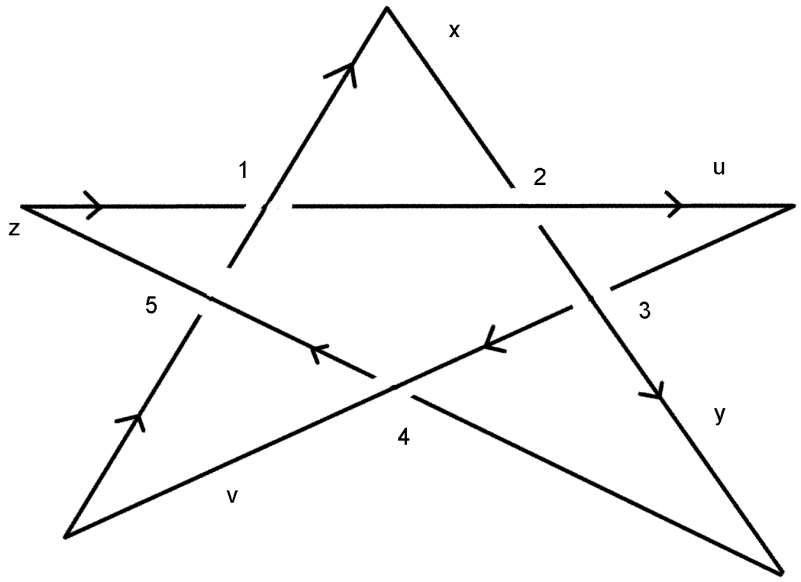


Figura 3.1: Nudo K_1.

Como se muestra en la Figura 3.1, el nudo K_1 tiene 5 cruces numerados y 5 arcos nombrados. Empezamos calculando la presentación de Wirtinger. Los generadores son:

$$(x, y, z, u, v)$$

Vamos calculando los cruces uno a uno:

- El cruce 1 queda como: $u = xzx^{-1}$
- El cruce 2 queda como: $y = uxu^{-1}$
- El cruce 3 queda como: $v = yuy^{-1}$
- El cruce 4 queda como: $z = vyv^{-1}$
- El cruce 5 queda como: $x = zvz^{-1}$

La presentación de Wirtinger se expresaría de la siguiente manera:

$$\pi_1(\mathbb{R}^3 - K_1) = |x, y, z, u, v : x = zvz^{-1}, y = uxu^{-1}, z = vyv^{-1}, u = xzx^{-1}, v = yuy^{-1}|$$

Para el polinomio de Alexander, construimos primero la matriz de Alexander:

$$\begin{bmatrix} & x & y & z & u & v \\ 1 & 1-t & 0 & t & -1 & 0 \\ 2 & t & -1 & 0 & 1-t & 0 \\ 3 & 0 & 1-t & 0 & t & -1 \\ 4 & 0 & t & -1 & 0 & 1-t \\ 5 & -1 & 0 & 1-t & 0 & t \end{bmatrix}$$

Eliminamos la primera fila y la primera columna para calcular el determinante y conseguir el polinomio de Alexander:

$$\det \begin{bmatrix} -1 & 0 & 1-t & 0 \\ 1-t & 0 & t & -1 \\ t & -1 & 0 & 1-t \\ 0 & 1-t & 0 & t \end{bmatrix} = -t^4 + t^3 - t^2 + t - 1.$$

Como el resultado tiene el término independiente negativo, multiplicamos el polinomio por -1 para normalizarlo quedando el polinomio de Alexander como:

$$P_A(K_1) = t^4 - t^3 + t^2 - t + 1.$$

Introducimos el nudo K_1 en nuestro programa y comprobamos que nuestros cálculos son correctos:

- Código:

```
nudoK1= Knot(["x","y","z","u","v"],[Cruce("z","u","x","+"),Cruce
    ("x","y", "u","+"), Cruce("u", "v","y","+") , Cruce("y", "z",
    "v","+"), Cruce("v", "x","z","+") ])
nudoK1.printarcos()
nudoK1.printcruces()
nudoK1.wirtinger()
expresion=nudoK1.wirtinger()
expresion.print_expresion()
print("La matriz de Alexander es: ",nudoK1.matriz_alexander())
print("El polinomio de Alexander es: ",nudoK1.
    polinomio_alexander())
```

- Salida:

```
['x', 'y', 'z', 'u', 'v']
Cruce  1 :
 Arco entrante:  z
 Arco saliente:  u
 Overpass:  x
 Orientación:  +
Cruce  2 :
 Arco entrante:  x
 Arco saliente:  y
 Overpass:  u
 Orientación:  +
Cruce  3 :
 Arco entrante:  u
 Arco saliente:  v
 Overpass:  y
 Orientación:  +
Cruce  4 :
 Arco entrante:  y
```

```
 Arco saliente:  z
 Overpass:  v
 Orientación:  +
Cruce  5 :
 Arco entrante:  v
 Arco saliente:  x
 Overpass:  z
 Orientación:  +
Generadores:
x y z u v
Relatores:
u^-1 x z x^-1
y^-1 u x u^-1
v^-1 y u y^-1
z^-1 v y v^-1
x^-1 z v z^-1
La matriz de Alexander es:  Matrix([[1 - t, 0, t, -1, 0], [t,
   -1, 0, 1 - t, 0], [0, 1 - t, 0, t, -1], [0, t, -1, 0, 1 - t],
    [-1, 0, 1 - t, 0, t]])
El polinomio de Alexander es:  t**4 - t**3 + t**2 - t + 1
```

3.2.2. Ejemplo 2

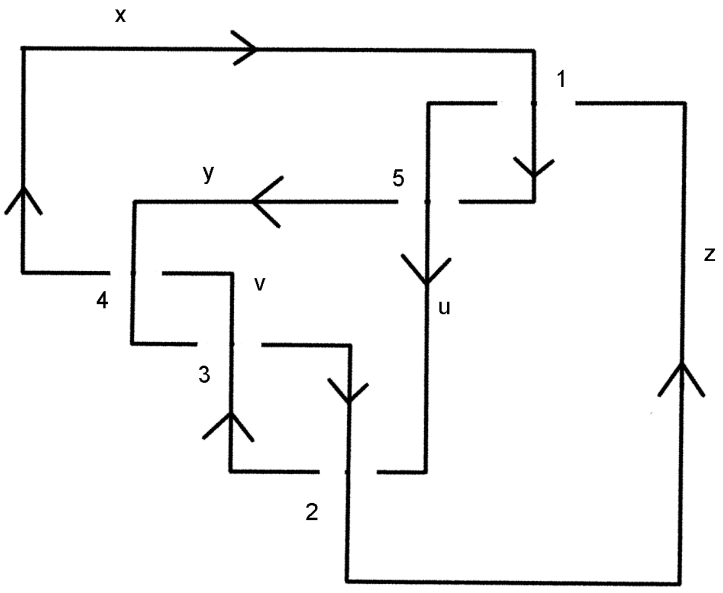


Figura 3.2: Nudo K_2.

Como se muestra en la Figura 3.2, el nudo K_2 tiene 5 cruces numerados y 5 arcos nombrados. Empezamos calculando la presentación de Wirtinger. Los

generadores son:

$$(x, y, z, u, v)$$

Vamos calculando los cruces uno a uno:

- El cruce 1 queda como: $u = xzx^{-1}$
- El cruce 2 queda como: $v = zuz^{-1}$
- El cruce 3 queda como: $z = vyv^{-1}$
- El cruce 4 queda como: $x = yvy^{-1}$
- El cruce 5 queda como: $y = uxu^{-1}$

La presentación de Wirtinger se expresaría de la siguiente manera:

$$\pi(\mathbb{R}^3 - K_2) = |x, y, z, u, v : x = yvy^{-1}, y = uxu^{-1}, z = vyv^{-1}, u = xzx^{-1}, v = zuz^{-1}|$$

Para el polinomio de Alexander, construimos primero la matriz de Alexander:

$$\begin{bmatrix} & x & y & z & u & v \\ 1 & 1-t & 0 & t & -1 & 0 \\ 2 & 0 & 0 & 1-t & t & -1 \\ 3 & 0 & t & -1 & 0 & 1-t \\ 4 & -1 & 1-t & 0 & 0 & t \\ 5 & t & -1 & 0 & 1-t & 0 \end{bmatrix}$$

Eliminamos la primera fila y la primera columna para calcular el determinante y conseguir el polinomio de Alexander:

$$\det \begin{bmatrix} 0 & 1-t & t & -1 \\ t & -1 & 0 & 1-t \\ 1-t & 0 & 0 & t \\ -1 & 0 & 1-t & 0 \end{bmatrix} = 2t^3 - 3t^2 + 2t.$$

Como el resultado no tiene término independiente, multiplicamos el polinomio por t^{-1} para normalizarlo quedando el polinomio de Alexander como:

$$P_A(K_2) = 2t^2 - 3t + 2.$$

Introducimos el nudo K_2 en nuestro programa y comprobamos que nuestros cálculos son correctos:

- Código:

```
nudoK2= Knot(["x","y","z","u","v"],[Cruce("z","u","x","+"),Cruce
    ("u","v", "z","+"), Cruce("y", "z","v","+") , Cruce("v", "x",
    "y","+"), Cruce("x", "y","u","+") ])
nudoK2.printarcos()
nudoK2.printcruces()
nudoK2.wirtinger()
expresion=nudoK2.wirtinger()
expresion.print_expresion()
```

```
print("La matriz de Alexander es: ",nudoK2.matriz_alexander())
print("El polinomio de Alexander es: ",nudoK2.
    polinomio_alexander())
```

- Salida:

```
['x', 'y', 'z', 'u', 'v']
Cruce  1 :
 Arco entrante:  z
 Arco saliente:  u
 Overpass:  x
 Orientación:  +
Cruce  2 :
 Arco entrante:  u
 Arco saliente:  v
 Overpass:  z
 Orientación:  +
Cruce  3 :
 Arco entrante:  y
 Arco saliente:  z
 Overpass:  v
 Orientación:  +
Cruce  4 :
 Arco entrante:  v
 Arco saliente:  x
 Overpass:  y
 Orientación:  +
Cruce  5 :
 Arco entrante:  x
 Arco saliente:  y
 Overpass:  u
 Orientación:  +
Generadores:
x y z u v
Relatores:
u^-1 x z x^-1
v^-1 z u z^-1
z^-1 v y v^-1
x^-1 y v y^-1
y^-1 u x u^-1
La matriz de Alexander es:  Matrix([[1 - t, 0, t, -1, 0], [0, 0,
     1 - t, t, -1], [0, t, -1, 0, 1 - t], [-1, 1 - t, 0, 0, t], [
    t, -1, 0, 1 - t, 0]])
El polinomio de Alexander es:  2*t**2 - 3*t + 2
```

3.2.3. Ejemplo 3

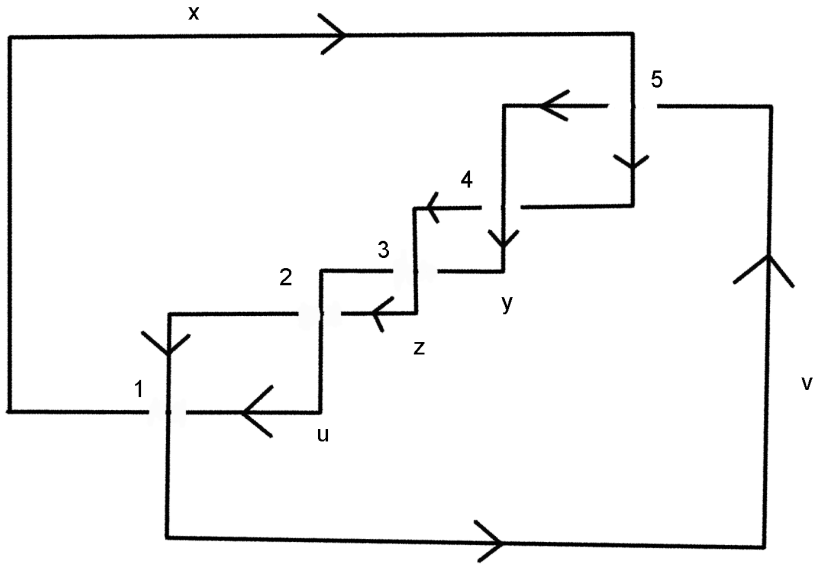


Figura 3.3: Nudo K_3.

Como se muestra en la Figura 3.3, el nudo K_3 tiene 5 cruces numerados y 5 arcos nombrados. Empezamos calculando la presentación de Wirtinger. Los generadores son:

$$(x, y, z, u, v)$$

Vamos calculando los cruces uno a uno:

- El cruce 1 queda como: $x = vuv^{-1}$
- El cruce 2 queda como: $v = uzu^{-1}$
- El cruce 3 queda como: $u = zyz^{-1}$
- El cruce 4 queda como: $z = yxy^{-1}$
- El cruce 5 queda como: $y = xvx^{-1}$

La presentación de Wirtinger se expresaría de la siguiente manera:

$$\pi(\mathbb{R}^3 - K_3) = |x, y, z, u, v : x = vuv^{-1}, y = xvx^{-1}, z = yxy^{-1}, u = zyz^{-1}, v = uzu^{-1}|$$

Para el polinomio de Alexander, construimos primero la matriz de Alexander:

$$\begin{bmatrix} & x & y & z & u & v \\ 1 & -1 & 0 & 0 & t & 1-t \\ 2 & 0 & 0 & t & 1-t & -1 \\ 3 & 0 & t & 1-t & -1 & 0 \\ 4 & t & 1-t & -1 & 0 & 0 \\ 5 & 1-t & -1 & 0 & 0 & t \end{bmatrix}$$

Eliminamos la primera fila y la primera columna para calcular el determinante y conseguir el polinomio de Alexander:

$$\det \begin{bmatrix} 0 & t & 1-t & -1 \\ t & 1-t & -1 & 0 \\ 1-t & -1 & 0 & 0 \\ -1 & 0 & 0 & t \end{bmatrix} = t^4 - t^3 + t^2 - t + 1.$$

Como el resultado tiene termino independiente positivo ya está normalizado, por lo tanto el polinomio de Alexander es:

$$P_A(K_3) = t^4 - t^3 + t^2 - t + 1.$$

Introducimos el nudo K_3 en nuestro programa y comprobamos que nuestros cálculos son correctos:

- Código:

```
nudoK3= Knot(["x","y","z","u","v"],[Cruce("u","x","v","+"),Cruce
    ("z","v", "u","+"), Cruce("y", "u","z","+") , Cruce("x", "z",
    "y","+"), Cruce("v", "y","x","+") ])
nudoK3.printarcos()
nudoK3.printcruces()
nudoK3.wirtinger()
expresion=nudoK3.wirtinger()
expresion.print_expresion()
print("La matriz de Alexander es: ",nudoK3.matriz_alexander())
print("El polinomio de Alexander es: ",nudoK3.
    polinomio_alexander())
```

- Salida:

```
['x', 'y', 'z', 'u', 'v']
Cruce  1 :
 Arco entrante:  u
 Arco saliente:  x
 Overpass:  v
 Orientación:  +
Cruce  2 :
 Arco entrante:  z
 Arco saliente:  v
 Overpass:  u
 Orientación:  +
Cruce  3 :
 Arco entrante:  y
 Arco saliente:  u
 Overpass:  z
 Orientación:  +
Cruce  4 :
 Arco entrante:  x
```

```
 Arco saliente:  z
 Overpass:  y
 Orientación:  +
Cruce  5 :
 Arco entrante:  v
 Arco saliente:  y
 Overpass:  x
 Orientación:  +
Generadores:
x y z u v
Relatores:
x^-1 v u v^-1
v^-1 u z u^-1
u^-1 z y z^-1
z^-1 y x y^-1
y^-1 x v x^-1
La matriz de Alexander es:  Matrix([[-1, 0, 0, t, 1 - t], [0, 0,
    t, 1 - t, -1], [0, t, 1 - t, -1, 0], [t, 1 - t, -1, 0, 0],
   [1 - t, -1, 0, 0, t]])
El polinomio de Alexander es:  t**4 - t**3 + t**2 - t + 1
```

3.2.4. Ejemplo 4

Para este ejemplo vamos a utilizar un nudo con más cruces para ver que el software puede manejarlo.

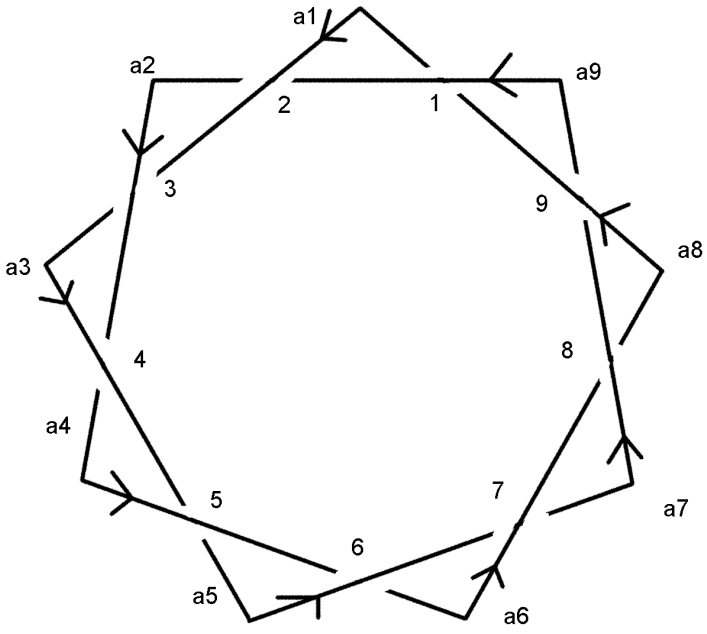


Figura 3.4: Nudo K_4.

Como se muestra en la Figura 3.4, el nudo K_4 tiene 9 cruces numerados y 9 arcos nombrados. Empezamos calculando la presentación de Wirtinger. Los generadores son:

$$(a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9)$$

Vamos calculando los cruces uno a uno:

- El cruce 1 queda como: $a_1 = a_9 a_8 a_9^{-1}$
- El cruce 2 queda como: $a_2 = a_1 a_9 a_1^{-1}$
- El cruce 3 queda como: $a_3 = a_2 a_1 a_3^{-1}$
- El cruce 4 queda como: $a_4 = a_3 a_2 a_3^{-1}$
- El cruce 5 queda como: $a_5 = a_4 a_3 a_5^{-1}$
- El cruce 6 queda como: $a_6 = a_5 a_4 a_5^{-1}$
- El cruce 7 queda como: $a_7 = a_6 a_5 a_6^{-1}$
- El cruce 8 queda como: $a_8 = a_7 a_6 a_7^{-1}$
- El cruce 9 queda como: $a_9 = a_8 a_7 a_8^{-1}$

La presentación de Wirtinger se expresaría de la siguiente manera:

$$\pi(\mathbb{R}^3 - K_4) = |a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9 : a_1 = a_9 a_8 a_9^{-1}, a_2 = a_1 a_9 a_1^{-1}, a_3 = a_2 a_1 a_3^{-1}, a_4 = a_3 a_2 a_3^{-1},$$
$$a_5 = a_4 a_3 a_5^{-1}, a_6 = a_5 a_4 a_5^{-1}, a_7 = a_6 a_5 a_6^{-1}, a_8 = a_7 a_6 a_7^{-1}, a_9 = a_8 a_7 a_8^{-1}|$$

Para el polinomio de Alexander, construimos primero la matriz de Alexander:

$$\begin{bmatrix} & a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 & a_8 & a_9 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & t & 1-t \\ 2 & 1-t & -1 & 0 & 0 & 0 & 0 & 0 & 0 & t \\ 3 & t & 1-t & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & t & 1-t & -1 & 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & t & 1-t & -1 & 0 & 0 & 0 & 0 \\ 6 & 0 & 0 & 0 & t & 1-t & -1 & 0 & 0 & 0 \\ 7 & 0 & 0 & 0 & 0 & t & 1-t & -1 & 0 & 0 \\ 8 & 0 & 0 & 0 & 0 & 0 & t & 1-t & -1 & 0 \\ 9 & 0 & 0 & 0 & 0 & 0 & 0 & t & 1-t & -1 \end{bmatrix}$$

Eliminamos la primera fila y la primera columna para calcular el determinante y conseguir el polinomio de Alexander:

$$\det \begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 0 & 0 & t \\ 1-t & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ t & 1-t & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & t & 1-t & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & t & 1-t & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & t & 1-t & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & t & 1-t & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & t & 1-t & -1 \end{bmatrix} = t^8 - t^7 + t^6 - t^5 + t^4 - t^3 + t^2 - t + 1.$$

Como el resultado tiene termino independiente positivo ya está normalizado, por lo tanto el polinomio de Alexander es:

$$P_A(K_4) = t^8 - t^7 + t^6 - t^5 + t^4 - t^3 + t^2 - t + 1.$$

Introducimos el nudo K_4 en nuestro programa y comprobamos que nuestros cálculos son correctos:

- Código:

```
nudoK4= Knot(["a1","a2","a3","a4","a5","a6","a7","a8","a9"],[
   Cruce("a8","a1","a9","+"),Cruce("a9","a2", "a1","+"), Cruce("
   a1", "a3","a2","+") , Cruce("a2", "a4","a3","+"), Cruce("a3",
    "a5","a4","+"), Cruce("a4", "a6","a5","+"), Cruce("a5", "a7"
   ,"a6","+"), Cruce("a6", "a8","a7","+"), Cruce("a7", "a9","a8"
   ,"+") ])
nudoK4.printarcos()
nudoK4.printcruces()
nudoK4.wirtinger()
expresion=nudoK4.wirtinger()
expresion.print_expresion()
print("La matriz de Alexander es: ",nudoK4.matriz_alexander())
print("El polinomio de Alexander es: ",nudoK4.
   polinomio_alexander())
```

- Salida:

```
['a1', 'a2', 'a3', 'a4', 'a5', 'a6', 'a7', 'a8', 'a9']
Cruce  1 :
 Arco entrante:  a8
 Arco saliente:  a1
 Overpass:  a9
 Orientación:  +
Cruce  2 :
 Arco entrante:  a9
 Arco saliente:  a2
 Overpass:  a1
 Orientación:  +
Cruce  3 :
 Arco entrante:  a1
 Arco saliente:  a3
 Overpass:  a2
 Orientación:  +
Cruce  4 :
 Arco entrante:  a2
 Arco saliente:  a4
 Overpass:  a3
 Orientación:  +
Cruce  5 :
```

```
 Arco entrante:  a3
 Arco saliente:  a5
 Overpass:  a4
 Orientación:  +
Cruce  6 :
 Arco entrante:  a4
 Arco saliente:  a6
 Overpass:  a5
 Orientación:  +
Cruce  7 :
 Arco entrante:  a5
 Arco saliente:  a7
 Overpass:  a6
 Orientación:  +
Cruce  8 :
 Arco entrante:  a6
 Arco saliente:  a8
 Overpass:  a7
 Orientación:  +
Cruce  9 :
 Arco entrante:  a7
 Arco saliente:  a9
 Overpass:  a8
 Orientación:  +
Generadores:
a1 a2 a3 a4 a5 a6 a7 a8 a9
Relatores:
a1^-1 a9 a8 a9^-1
a2^-1 a1 a9 a1^-1
a3^-1 a2 a1 a2^-1
a4^-1 a3 a2 a3^-1
a5^-1 a4 a3 a4^-1
a6^-1 a5 a4 a5^-1
a7^-1 a6 a5 a6^-1
a8^-1 a7 a6 a7^-1
a9^-1 a8 a7 a8^-1
La matriz de Alexander es:  Matrix([[-1, 0, 0, 0, 0, 0, 0, t, 1
   - t], [1 - t, -1, 0, 0, 0, 0, 0, 0, t], [t, 1 - t, -1, 0, 0,
   0, 0, 0, 0, 0], [0, t, 1 - t, -1, 0, 0, 0, 0, 0], [0, 0, t, 1 -
   t, -1, 0, 0, 0, 0], [0, 0, 0, t, 1 - t, -1, 0, 0, 0], [0, 0,
   0, 0, t, 1 - t, -1, 0, 0], [0, 0, 0, 0, 0, t, 1 - t, -1, 0],
   [0, 0, 0, 0, 0, 0, t, 1 - t, -1]])
El polinomio de Alexander es:  t**8 - t**7 + t**6 - t**5 + t**4
   - t**3 + t**2 - t + 1
```

3.2.5. Conclusiones

Comparamos los invariantes de K_1 y K_2. En cuanto a las presentaciones de Wirtinger:

$$\pi_1(\mathbb{R}^3 - K_1) = |x, y, z, u, v : x = zvz^{-1}, y = uxu^{-1}, z = vyv^{-1}, u = xzx^{-1}, v = yuy^{-1}|,$$

$$\pi_1(\mathbb{R}^3 - K_2) = |x', y', z', u', v' : x' = y'v'y'^{-1}, y' = u'x'u'^{-1}, z' = v'y'v'^{-1}, u' = x'z'x'^{-1}, v' = z'u'z'^{-1}|,$$

no son equivalentes ya que no podemos encontrar una transformación de una en otra. Para el polinomio de Alexander, vemos a simple vista que son diferentes:

$$P_A(K_1) = t^4 - t^3 + t^2 - t + 1.$$

$$P_A(K_2) = 2t^2 - 3t + 2.$$

Por lo tanto, al no compartir las invariantes podemos concluir que K_1 y K_2 no son equivalentes.

Para K_1 y K_3 podemos crear una transformación en sus presentaciones de Wirtinger para ver que son equivalentes:

$$\pi_1(\mathbb{R}^3 - K_1) = |x, y, z, u, v : x = zvz^{-1}, y = uxu^{-1}, z = vyv^{-1}, u = xzx^{-1}, v = yuy^{-1}|,$$

$$\pi_1(\mathbb{R}^3 - K_3) = |x', y', z', u', v' : x' = v'u'v'^{-1}, y' = x'v'x'^{-1}, z' = y'x'y'^{-1}, u' = z'y'z'^{-1}, v' = u'z'u'^{-1}|,$$

$$x \to x',$$
$$y \to z',$$
$$z \to v',$$
$$u \to y',$$
$$v \to u'.$$

1. $x = zvz^{-1}$

   Se sustituye y se transforma en $x' = v'u'v'^{-1}$

2. $y = uxu^{-1}$

   Se sustituye y se transforma en $z' = y'x'y'^{-1}$

3. $z = vyv^{-1}$

   Se sustituye y se transforma en $v' = u'z'u'^{-1}$

4. $u = xzx^{-1}$

   Se sustituye y se transforma en $y' = x'v'x'^{-1}$

5. $v = yuy^{-1}$

   Se sustituye y se transforma en $u' = z'y'z'^{-1}$

El polinomio de Alexander de ambos nudos es el mismo, por lo que podemos confirmar que K_1 y K_3 son equivalentes, ya que tienen los mismos invariantes. Por consecuencia, K_2 y K_3 no son equivalentes.

Con el nudo K_4 podemos ver que con un nudo más complejo por tener más cruces el software es capaz de manejarlo.

Capítulo 4

Análisis de impacto

El impacto potencial de los resultados obtenidos durante la realización del TFG se puede analizar desde distintos contextos:

- **Personal:** Este trabajo me ha permitido estudiar el área de la teoría de nudos, una rama de las matemáticas que no se incluye en ninguna asignatura de la carrera. El proyecto ha sido un reto en el que he mejorado mis habilidades de programación y me he demostrado que soy capaz de abordar problemas matemáticos complejos para conseguir un objetivo en concreto. Valoro especialmente este logro ya que, de cara al mundo laboral, creo que es una habilidad destacable.
- **Empresarial:** El trabajo desarrollado tiene un potencial interés para empresas que se dediquen a las matemáticas y se podrían beneficiar del software para calcular los invariantes del nudo. Además, la teoría de nudos tiene aplicaciones en los sectores de la biotecnología, sobre todo en el estudio del ADN, y en la criptografía, por lo que empresas de este sector podrían aprovechar el trabajo realizado para profundizar en sus respectivas áreas.
- **Social:** En la educación y divulgación matemática este trabajo puede ser una introducción accesible a la teoría de nudos para los estudiantes que se interesen por este campo. La implementación de los algoritmos facilita la enseñanza de conceptos topológicos abstractos que son complicados de entender al principio. Además, en el ámbito de la investigación, puede reducir significativamente el tiempo que se emplea para calcular los invariantes de los nudos.

En cuanto a los Objetivos de Desarrollo Sostenible (ODS) de la Agenda 2030 [5], el trabajo tendrá un potencial impacto sobre:

- Educación de calidad (ODS 4): Fomenta el aprendizaje matemático y tecnológico y es accesible para aquel que quiera estudiarlo.
- Industria, innovación e infraestructura (ODS 9): Este trabajo puede apoyar avances en tecnología de distintos sectores que no tienen que ver únicamente con las matemáticas, como la biología y la química.

Capítulo 5

Conclusiones y trabajo futuro

5.1. Conclusiones

5.1.1. Objetivos alcanzados

El objetivo principal de este TFG es la implementación de algoritmos para calcular los invariantes de los nudos, específicamente el grupo fundamental y el polinomio de Alexander. Ambos algoritmos se implementaron con éxito y se programaron en Python, con varios ejemplos que demuestran su funcionalidad y utilidad.

5.1.2. Impacto del trabajo

El software del trabajo permite a investigadores de este área calcular de forma eficiente invariantes del nudo, reduciendo el tiempo necesario para estos cálculos.

5.1.3. Reflexión personal

La rama de la teoría de nudos siempre me ha interesado, ya que lo que más me ha motivado durante la carrera es ver como las matemáticas se pueden aplicar a cosas del día a día en las que no encontramos nada matemático a simple vista. Además disfruté mucho la asignatura de Topología y Topología aplicada y computacional, lo que hizo que quisiese hacer el TFG enfocado a la topología.

Creo que este trabajo es un ejemplo perfecto de lo que es el Grado de Matemáticas e Informática, ya que, al no ser un doble grado, en la mayoría de las asignaturas se trata de juntar las matemáticas con la informática. Este proyecto, con la parte teórica y la parte de programación unifica ambas disciplinas demostrando que están íntimamente relacionadas y que se complementan para conseguir avances en ambas áreas.

5.2. Trabajo Futuro

La principal dificultad que se ha encontrado con el uso del software ha sido introducir los nudos. Como hay que introducir todos los arcos y cruces de forma manual es fácil equivocarse y con nudos grandes puede ser tedioso, lo que limita la accesibilidad y la eficiencia del programa.

Una posible mejora sería implementar alguna herramienta que analice el diagrama de un nudo para etiquetar los arcos y calcular los cruces, facilitando el uso para los usuarios y reduciendo los errores.

En el futuro, a la clase creada se le podrían añadir algoritmos para calcular otros invariantes, de esta manera el software sería más completo.

Por último, se podrían crear variaciones del software con distintos enfoques para que usuarios de otras áreas, como la biotecnología o la criptografía puedan utilizar el programa y las funcionalidades específicas que necesiten de manera más simple y especializadas en su sector concreto.

Bibliografía

[1] R. H. F. Richard H. Crowel, *Introduction to knot Theory*. Springer Verlag, 1963.

[2] J. Stillwell, *Classical Topology and Combinatorial Group Theory*. Springer Verlag, 1980.

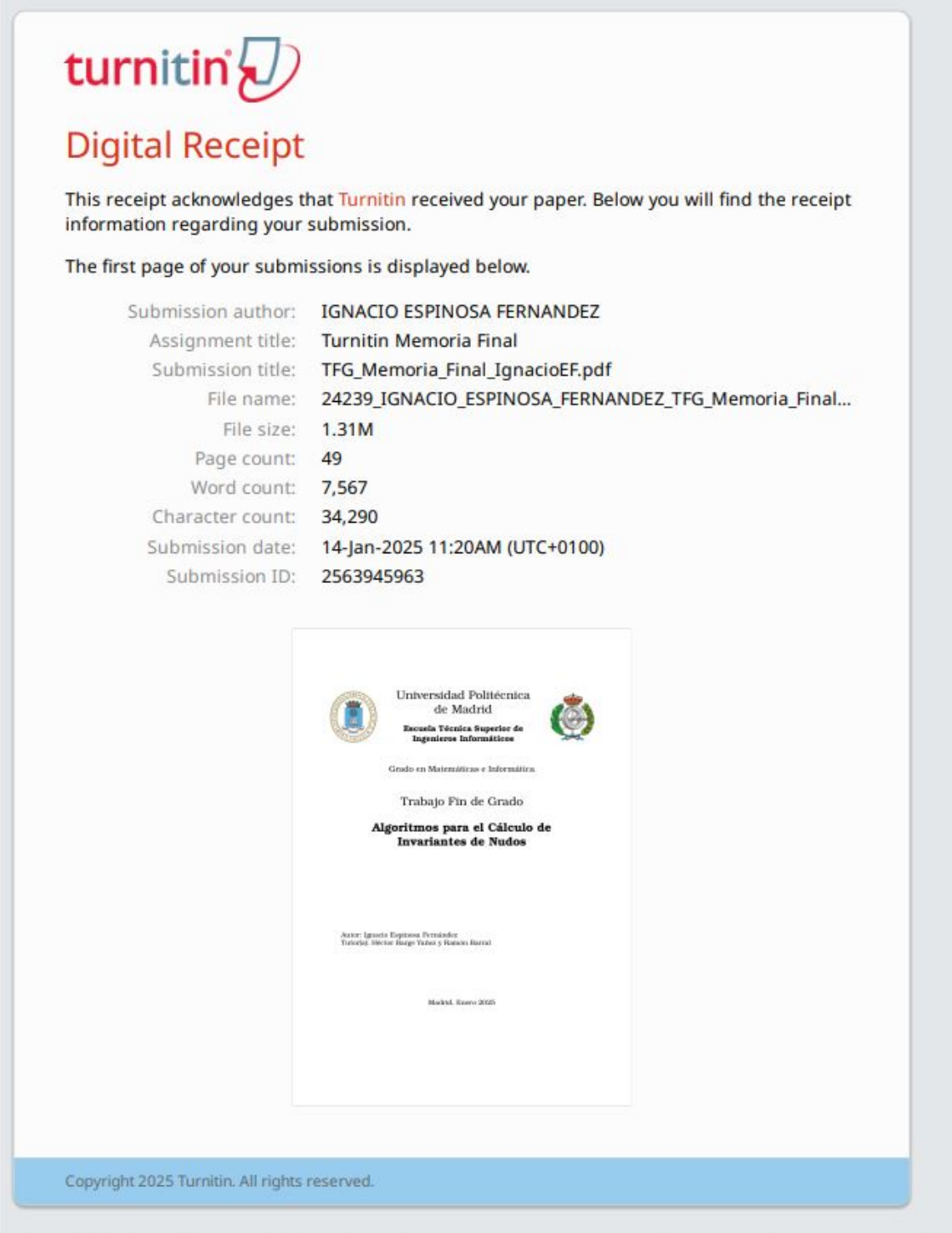
[3] P. S. Robert Messer, *Topology now*. American Mathematical Society, 2006.

[4] D. B.-N. y Scott Morrison. (2005) Knot atlas. Accessed: 08/01/2025. [Online]. Available: https://katlas.org/wiki/Main_Page

[5] un.org. (2015) Objetivos de desarrollo sostenible (ods) de la agenda 2030. Accessed: 02/01/2025. [Online]. Available: https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/

[6] I. E. Fernández, "Imágenes y diagramas del trabajo fin de grado," 2025, todas las imágenes han sido elaboradas por el autor.

Anexos



turnitin

Digital Receipt

This receipt acknowledges that Turnitin received your paper. Below you will find the receipt information regarding your submission.

The first page of your submissions is displayed below.

Submission author:	IGNACIO ESPINOSA FERNANDEZ
Assignment title:	Turnitin Memoria Final
Submission title:	TFG_Memoria_Final_IgnacioEF.pdf
File name:	24239_IGNACIO_ESPINOSA_FERNANDEZ_TFG_Memoria_Final...
File size:	1.31M
Page count:	49
Word count:	7,567
Character count:	34,290
Submission date:	14-Jan-2025 11:20AM (UTC+0100)
Submission ID:	2563945963

Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Grado en Matemáticas e Informática

Trabajo Fin de Grado

Algoritmos para el Cálculo de Invariantes de Nudos

Copyright 2025 Turnitin. All rights reserved.

Figura 1: Recibo digital de Turnitin.

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
	Fecha/Hora	Tue Jan 14 15:58:32 CET 2025
	Emisor del Certificado	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
	Numero de Serie	561
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)