Grado en Matemática Computacional

Estancia en Prácticas y Proyecto Final de Grado

Ecuación de Schrödinger de una partícula restringida a una superficie de revolución

Autor:
Eva Palomar Sarrió

Supervisor:
Paco Falomir
Tutor académico:
Vicent Gimeno García

Fecha de lectura: Noviembre de 2020
Curso académico 2019/2020

Resumen

En este documento se recoge la memoria de la estancia en prácticas y el Trabajo Final de Grado que se ha realizado.

Por un lado, el capítulo Memoria de prácticas se contextualiza y detalla el trabajo realizado como programadora en la empresa UBE CORPORATION EUROPE de Castellón. En esta estancia en prácticas se realizó un Proyecto de Dada Science de recolección, análisis y aprovechamiento de datos para predecir el comportamiento de un equipo en particular. El proyecto fue desarrollado con Python.

Por otro lado, el Trabajo de fin de grado. Este no se ha desarrollado en base la experiencia en prácticas, sino que se trata de un trabajo bibliográfico sobre la Función de Schrödinger y la geometría diferencial.

Palabras clave

Redes neuronales, Python, Función Schrödinger, superficies de revolución.

Keywords

Neural Networks, Python, Schrödinger function , revolution surfaces.

Índice general

Índice de figuras

.

Capítulo 1

Estancia en prácticas

1.1. Introducción a la empresa

La empresa en la que he desarrollado mis practicas del Proyecto de final de Grado ha sido UBE CORPORATION EUROPE. Esta corporación consta de un grupo de empresas dedicadas a la producción y distribución de productos químicos de valor añadido y a la creación de soluciones tecnológicas y sostenibles para satisfacer las demandas de la industria.

La sede en España está ubicada en un complejo industrial cerca del Puerto de Castellón. Este complejo industrial cuenta con líneas de producción de caprolactama (materia prima para el nylon o poliamida 6), fertilizantes, productos de química fina (1,6-hexanodiol, 1,5-pentanodiol, policarbonatodioles), nylon y copoliamidas.

Las prácticas fueron realizadas en el departamento Programming Control System (PCS). Este departamento se encarga, entre otras cosas, del buen funcionamiento de los sistemas de control y monitorización de los equipos de planta. Asimismo, el responsable de este proyecto fue Francisco Falomir Areas.

1.2. Objetivos del proyecto formativo

El objetivo principal de mi estancia en prácticas era la elaboración de una red neuronal que evaluará el estado de unos determinados compresores de hidrógeno. Estos equipos suelen ser bastante críticos, ya que el deterioro o paro de estos puede afectar al sistema de producción global de la empresa. Por tanto, se buscaba un software con el que automatizar las alertas por

mal funcionamiento y que dejara históricos de funcionamiento en una base de datos para poder consultarlos si fuera necesario.

Puesto que era un proyecto que iba a hacer un alumno de prácticas, también era un objetivo que el alumno en prácticas aprendiera a trabajar con el Software de la empresa. Viéndose inmerso por primera vez en el ámbito laboral, con un proyecto extenso y con una necesidad de aplicar los conocimientos adquiridos en el grado que estaba cursando.

1.3. Explicación detallada del proyecto

1.3.1. Conceptos previos

Introducción a los equipos

Mi proyecto está desarrollado sobre dos compresores[1] que se encuentran en la Unidad-444, el compresor A (C1-A) y el compresor B (C1-B). Esta unidad pertenece a la parte del proceso que trata con Hexanodiol. La función de estos compresores es comprimir hidrógeno y para ello constan de 5 etapas donde aumentan la presión del fluido a medida que este avanza por el circuito. Estos dos compresores funcionan en paralelo y aunque aparentemente parezcan iguales, no lo son. La complejidad del C1-A frente al C1-B es evidente, debido a que el C1-A es mucho más antiguo que el C1-B, con todo lo que ello supone. Además el compresor A posee un equipo auxiliar llamado soplante[2]. Este equipo auxiliar hace que los registros del C1-A sean muy diferentes a los registros del C1-B.

Asimismo, desde el primer momento di prioridad al C1-B porque presentaba más irregularidades en su funcionamiento. La necesidad de realizar este proyecto se dio por la detección de valores irregulares en las mediciones de presión entre etapas de compresión, pero como la presión final se mantenía en los rangos normales estipulados por el fabricante, la máquina estuvo trabajando durante un tiempo en este estado de irregularidad.

Por tanto, se necesitaba una herramienta que avisara automáticamente de las posibles mediciones irregulares de algún sensor en estos equipos. Cabe destacar que este proyecto es puramente preventivo y su finalidad es detectar y notificar de un mal funcionamiento antes de que sea necesario parar la máquina por completo.

[1] Un compresor es una máquina, cuyo trabajo consiste en incrementar la presión de un fluido.

[2] La soplante es un equipo centrífugo autónomo cuya principal función es impulsar un fluido. Para este equipo en concreto, impulsa aire para ayudar a la puesta en marcha y al movimiento del vástago, ayudando a la relación de compresión de las distintas etapas.

Extracción y manipulación de datos

Como se explica posteriormente, para que una red neuronal con entrenamiento supervisado funcione correctamente es necesario tener a nuestra disposición datos de funcionamiento del equipo en concreto, tanto historial de fallos como periodos de buen funcionamiento. En este apartado se mostrará la información más relevante en cuanto a la extracción y almacenado de datos.

- **Almacenamiento de datos:** UBE utiliza una herramienta llamada Exacuantum. Esta es un sistema de gestión de bases de datos diseñado para plantas industriales (PIMS[3]) disponible en las industrias de procesos. Es capaz de obtener datos de todas las partes de un proceso y convertirlos en información fácilmente utilizable. Además, estos registros son distribuidos sencillamente en toda la empresa y se puede acceder a ellos desde múltiples sistemas de control de procesos (Ver [5]). Además la empresa también utiliza numerosas bases de datos para almacenar los registros de todos los equipos.
- **Manipulación de los datos:** Para manipular grandes cantidades de datos he utilizado Microsoft Excel: Excel es un programa informático desarrollado y distribuido por Microsoft Corp. Se trata de un software que permite realizar tareas contables y financieras gracias a sus funciones, desarrolladas específicamente para ayudar a crear y trabajar con hojas de cálculo. Este programa nos facilita en gran medida el trabajo con números y nos permite analizarlos fácilmente y generar reportes con herramientas como los gráficos y las tablas dinámicas, además de archivos csv[4]. y múltiples extensiones más.

Nota: Cabe destacar que Exel consta de una herramienta de sincronización automática con Exacuantum y es muy sencillo exportar datos históricos para trabajar con ellos en Exel.

Lenguajes de programación utilizados

- **SQL**: Es un sistema para la gestión de bases de datos producido por Microsoft. También es un lenguaje de programación estándar para manejar información desde una base de datos relacional (distintas entidades o tablas). Este entorno nos permite realizar un mantenimiento de la base de datos, es decir, crear, insertar, modificar, eliminar y consultar registros. Para poder manipular el lenguaje SQL se emplea un sistema gestor de base de datos (MySQL, SQL Server, ORACLE, etc).

[3]PIM es el acrónimo de Product Information Management, gestión de información de productos. Un sistema PIM permite recopilar, enriquecer, gestionar y distribuir de forma eficiente la información de los productos en un entorno

[4]Los archivos CSV son un tipo de documento en formato abierto sencillo para representar datos en forma de tabla, en las que las columnas se separan por comas y las filas por saltos de línea.

- **Python**: Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en la legibilidad de su código. Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, dinámico y multiplataforma.

¿Porqué utilizar Python para redes neuronales?

En primer lugar, Python proporciona acceso a la **comunidad abierta** de desarrolladores más grande del mundo en cuanto a lenguajes de programación. Por tanto, puedes obtener ayuda de infinidad de páginas, lo que resulta de gran utilidad. En segundo lugar y centrándome en las redes neuronales, ha sido de gran ayuda tener un **gran abanico de cursos** sobre este campo en python, además de ejemplos y foros dónde preguntar dudas.

Cabe destacar también la posibilidad de utilizar múltiples librerías gratuitas de Python, puesto que es un lenguaje de programación de **código abierto**. Estas permiten acelerar los procesos de desarrollo. Las librerias más útiles ha sido:

- Librerías para machine learning en TensorFlow de Google.
- Librerías para deep learning como Keras.

Además aporta una gran versatilidad por ser **multiplataforma**: Válido para Windows, MacOS, Linux, Unix, entre otras. Esto me ha permitido programar tanto en los ordenadores de la empresa como con mi propio ordenador..

Por último, este lenguaje de programación proporciona una gran cantidad de **herramientas para la visualización de datos**. Estas permiten a los científicos de datos agilizar los análisis, la detección de errores, aplicación de mejoras, etc. En mi caso particular, me ha servido para visualizar gráficas y resultados de una manera clara.

Librerias más importantes:

En informática, una librería es un conjunto de implementaciones funcionales, codificadas en un lenguaje de programación, la cual ofrece una interfaz bien definida para la funcionalidad que se invoca. Es decir, facilita la programación, pero siempre hay que tener en cuenta hay que hacer buen uso de ellas y leerse la documentación necesaria para saber usarlas.

- **Keras:** Es una biblioteca de Redes Neuronales de Código Abierto escrita en Python. A grandes rasgos, está especialmente diseñada para facilitar la experimentación con redes de Aprendizaje Profundo. (Ver [1]).En el proyecto se ha utilizado esta biblioteca porque nos proporciona la posibilidad de guardar tanto la arquitectura de modelo como guardar los pesos de este.

  - Los **pesos** de los modelos se guardan en formato HDF5. Este es un formato de cuadrícula que es ideal para almacenar matrices multidimensionales de números.
  - El **modelo** en sí lo guardamos con el formato JSON. Los datos en los archivos JSON son pares de propiedad valor separados por dos puntos. Estos pares se separan mediante comas y se encierran entre llaves. El valor de una propiedad puede ser otro objeto JSON, lo que ofrece una gran flexibilidad a la hora de estructurar información.

  Importante: En el siguiente apartado veremos con más detalle a qué hacen referencia estos términos que son partes básicas de la composición de una red neuronal.

- **TensorFlow** es una biblioteca de código abierto para aprendizaje automático a través de un rango de tareas. Este ha sido desarrollado por Google para satisfacer la necesidad de implementar sistemas capaces de construir y entrenar redes neuronales. Con el fin de detectar y descifrar patrones de razonamiento usados por los humanos. (Ver [4])

Redes neuronales

Las **redes neuronales artificiales** (RNA) son un modelo computacional dentro del campo del machine learning (ML). Este modelo pretende imitare el funcionamiento de las neuronas biológicas. Dicha imitación se basa en que las RNA están compuestas por 'neuronas' (nodos) que intercambian datos entre sí gracias a sus interconexiones, tal como las neuronas biológicas lo hacen a través de las sinapsis. Por tanto, el objetivo de las RNA es proporcionarle a los sistemas informáticos la capacidad de pensamiento y aprendizaje. Desarrollar redes neuronales con Python es una de las prácticas más valoradas para la consecución de dicho objetivo, por simplicidad y eficacia.

A continuación, se presenta las redes neuronales desde un **punto de vista informático**, puesto que el proyecto se ha desarrollado desde un departamento centrado en resolver, utilizando recursos informáticos, las necesidades de los equipos industriales de planta. Asimismo, profundizaremos en cómo se estructura y cómo se entrena y se evalúa una RNA, presentado lo métodos informáticos más importantes y el porqué se han utilizado.

Las redes neuronales son composiciones de neuronas y la función de las neuronales es procesar una serie de datos y pesos con el fin de devolver un valor que minimice el error al resultado final. (Ver [6])

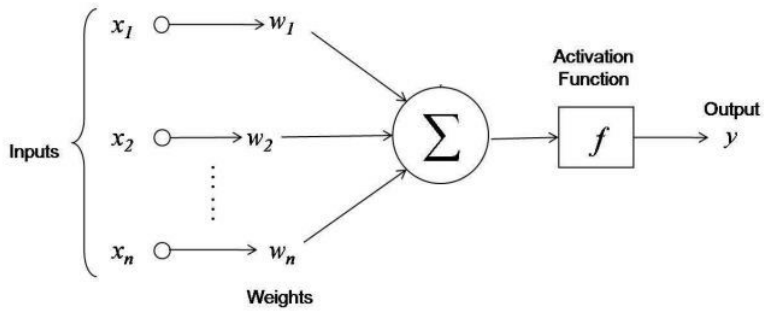


Figura 1.1: Neurona artificial

Como podemos observar el resultado parcial de cada **neurona** es evaluado por la función de activación. En la red neuronal se ha usado las siguientes funciones de activación:

- La función ReLU en todas las capas menos en la de salida. Esta función elimina los negativos dejándolos a cero.

$$\boxed{f(x) = \text{máx}(0, \text{x}) = \begin{cases} 0 \text{ for } x < 0 \\ x \text{ for } x \geq 0 \end{cases}}$$

- En la de salida se utiliza la función de activación Sigmoidea. Esta es buena para la salida porque devuelve un número entre 0 y 1. Siendo el dígito 0 el estado de buen funcionamiento y en contraposición el dígito 1 el de mal funcionamiento.

$$\boxed{f(x) = \frac{1}{1 - e^{-x}}}$$

En consecuencia, del mismo modo que nuestro cerebro está compuesto por neuronas interconectadas entre sí, una red neuronal artificial está formada por neuronas artificiales conectadas entre sí y agrupadas en diferentes niveles que denominamos capas.

Una **capa** es un conjunto de neuronas cuyas entradas provienen de una capa anterior (o de los datos de entrada en el caso de la primera capa) y sus salidas son la entrada de una capa posterior. La red neuronal de este proyecto es secuencial y multicapa. Además la red cuenta con 2 capas internas, exactamente como la que se presenta a continuación:

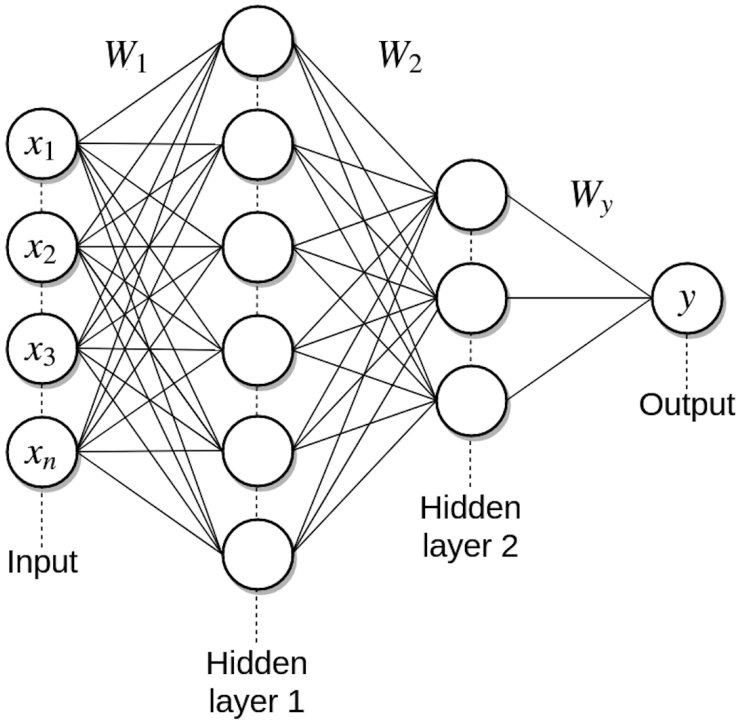


Figura 1.2: Estructura de una red neuronal secuencial multicapa

Otra característica básica de una RNA es el tipo de entrenamiento, en este caso el entrenamiento ha sido supervisado. El **entrenamiento supervisado** se trata de una técnica para deducir una función a partir de datos de entrenamiento. Es decir, en este en modo aprendizaje se muestran los patrones a la red y la salida deseada para esos patrones y se usa una fórmula matemática de minimización del error que ajuste los pesos para dar la salida más cercana posible a la salida deseada.

Profundizando más en lo comentado anteriormente, lo que se quiere mostrar es que se tiene una 'variable X' que representa las distintas mediciones (en uno o varios instantes de tiempo) y esta está asociada a su correspondiente 'variable Y' (estado). Entonces, utilizando está información, una red neuronal es capaz de aprender qué futuras mediciones corresponden a un estado u a otro. La 'variable Y' la ofrecemos de antemano puesto que la hemos obtenido del histórico proporcionado por la empresa y los informes de fallos en el equipo.

En conclusión, entrenar una red neuronal consiste en ajustar cada uno de los pesos w_{ij} de las entradas de todas las neuronas que forman parte de la red neuronal, para que las respuestas de la capa de salida se ajusten lo mejor posible a los datos que conocemos.

Overfiting

El overfiting o sobreentrenamiento es el efecto de sobreentrenar un algoritmo de aprendizaje con unos ciertos datos para los que se conoce el resultado deseado.

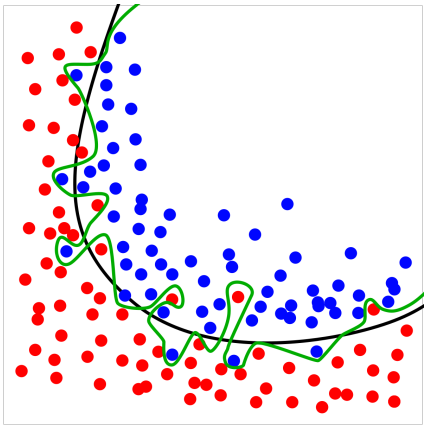


Figura 1.3: Ejemplo de overfiting o sobreentrenamiento

En consecuencia, el periodo de evaluación es poco preciso porque se ajusta demasiado a los valores con los que ha sido entrenado.

La solución al sobreentrenamiento se encuentra en los Dropout. Un **Dropout** no es más que una capa intermedia, entre capas de neuronas. Su función es apagar neuronas al azar con el fin de que las neuronas no se vuelvan tan dependientes de los datos. Con ello se evita el overfiting.

1.3.2. Etapas de desarrollo del proyecto

En primer lugar, estuve las dos primeras semanas haciendo cursos de técnicas de Machine Learning y técnicas de minería de datos. Utilicé tanto información extraída de internet como la de cursos de redes neuronales (Ver [8]), esto me facilitó la esquematización del problema de una manera clara y detallada.

Una vez tenía un poco de idea de que quería hacer y cuál tenía que ser el resultado. Decidí que las 'variables de entrada X' serían toda la información recopilada por los sensores y mi 'variable de salida Y' un dígito binario el cual representa si va bien o va mal el equipo. Empecé a buscar ejemplos de redes neuronales en Python y ver cómo se utilizaban las librerías necesarias. En este paso, aprendí a utilizar Keras y TensorFlow.

Dediqué tiempo a instalar paquetes y librerías en mi ordenador de trabajo. Al terminar estas dos semanas sabía cómo quería programar mi proyecto pero necesitaba información de las máquinas a las que les iba a implantar la red neuronal, pero empezaba a tener claras cómo se utilizaban las herramientas de programación de Python y eso ya suponía un avance.

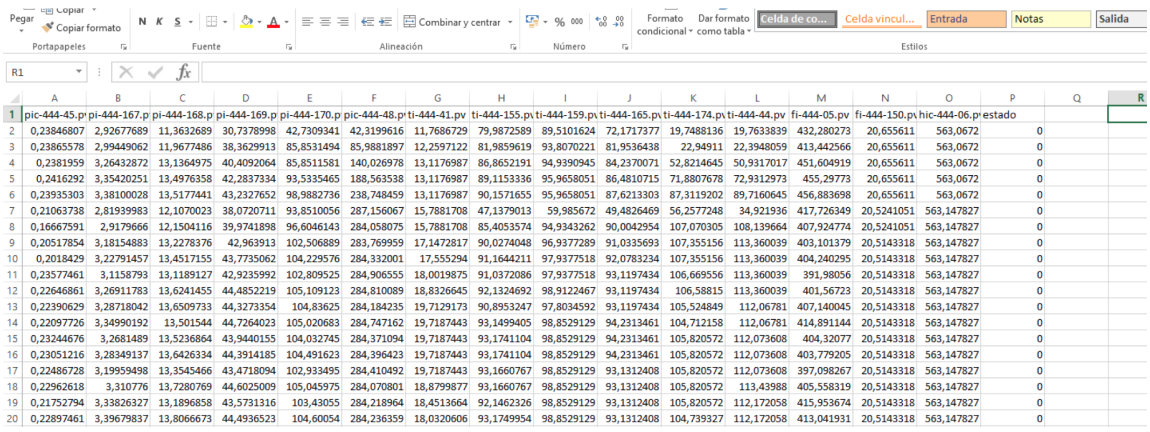
La segunda quincena de mi periodo de prácticas fue donde más avancé. Previamente ya había solicitado una reunión con el jefe del departamento de mantenimiento rotativo. En esta reunión el encargado me preparó una documentación detallada de los fallos que se habían registrado en ambos compresores y además me proporcionó una tabla con los sensores más relevantes para la medición del estado de estos, dando especial importancia a los rátios de compresión entre las etapas.

Por tanto, vi conveniente, a parte de tener en cuenta los valores de los sensores, crear las variables necesarias con los ratios de compresión, para que la red neuronal tuviera en cuenta estos valores a la hora de detectar si la máquina funciona correctamente o no.

Posteriormente, tuve una visita a planta para explicarme el funcionamiento de la máquina con mucho detalle y la ubicación de los sensores. Me pareció muy importante entender el funcionamiento del equipo a la hora de programar una herramienta auxiliar.

Después de tener la información necesaria, creé una hoja de datos para poder empezar a entrenar la red neuronal y ver cómo respondía a la evaluación. Esta hoja de datos en Exel tiene la siguiente estructura: Cada fila corresponde a un instante de tiempo y cada columna a la información extraída de cada sensor. Esta información forma mi 'variable X', es decir los datos de entrada. Cabe destacar que la salida de mi red va a ser 0 o 1. Dependiendo de si tiene un buen funcionamiento o un mal funcionamiento respectivamente.

Para el aprendizaje supervisado, el .csv de donde la red neuronal extrae información debe tener épocas donde la máquina funcionó bien y épocas donde funcionó mal. En la última columna de la hoja de datos de Exel irá este valor binario, representando el estado.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	pic-444-45.p	pi-444-167.p	pi-444-168.p	pi-444-169.p	pi-444-170.p	pic-444-48.p	ti-444-41.pv	ti-444-155.p	ti-444-159.p	ti-444-165.p	ti-444-174.p	ti-444-44.pv	fi-444-05.pv	fi-444-150.p	hic-444-06.p	estado
2	0,23846807	2,92677689	11,3632689	30,7378998	42,7309341	42,3199616	11,7686729	79,9872589	89,5101624	72,1717377	19,7488136	19,7633839	432,280273	20,655611	563,0672	0
3	0,23865578	2,99449062	11,9677486	38,3629913	85,8531494	85,9881897	12,2597122	81,9859619	93,8070221	81,9536438	22,94911	22,3948059	413,442566	20,655611	563,0672	0
4	0,2381959	3,26432872	13,1364975	40,4092064	85,8511581	140,026978	13,1176987	86,8652191	94,9390945	84,2370071	52,8214645	50,9317017	451,604919	20,655611	563,0672	0
5	0,2416292	3,35420251	13,4976358	42,2837334	93,5335465	188,563538	13,1176987	89,1153336	95,9658051	86,4810715	71,8807678	72,9312973	455,29773	20,655611	563,0672	0
6	0,23935303	3,38100028	13,5177441	43,2327652	98,9882736	238,748459	13,1176987	90,1571655	95,9658051	87,6213303	87,3119202	89,7160645	456,883698	20,655611	563,0672	0
7	0,21063738	2,81939983	12,1070023	38,0720711	93,8510056	287,156067	15,7881708	47,1379013	59,985672	49,4826469	56,2577248	34,921936	417,726349	20,5241051	563,147827	0
8	0,16667591	2,9179666	12,1504116	39,9741898	96,6046143	284,058075	15,7881708	85,4053574	94,9343262	90,0042954	107,070305	108,139664	407,924774	20,5241051	563,147827	0
9	0,20517854	3,18154883	13,2278376	42,963913	102,506889	283,769959	17,1472817	90,0274048	96,9377289	91,0335693	107,355156	113,360039	403,101379	20,5143318	563,147827	0
10	0,2018429	3,22791457	13,4517155	43,7735062	104,229576	284,332001	17,555294	91,1644211	97,9377518	92,0783234	107,355156	113,360039	404,240295	20,5143318	563,147827	0
11	0,23577461	3,1158793	13,1189127	42,9235992	102,809525	284,906555	18,0019875	91,0372086	97,9377518	93,1197434	106,669556	113,360039	391,98056	20,5143318	563,147827	0
12	0,22646861	3,26911783	13,6241455	44,4852219	105,109123	284,810089	18,8326645	92,1324692	98,9122467	93,1197434	106,58815	113,360039	401,56723	20,5143318	563,147827	0
13	0,22390629	3,28718042	13,6509733	44,3273354	104,83625	284,184235	19,7129173	90,8953247	97,8034592	93,1197434	105,524849	112,06781	407,140045	20,5143318	563,147827	0
14	0,22097726	3,34990192	13,501544	44,7264023	105,020683	284,747162	19,7187443	93,1499405	98,8529129	94,2313461	104,712158	112,06781	414,891144	20,5143318	563,147827	0
15	0,23244676	3,2681489	13,5236864	43,9440155	104,032745	284,371094	19,7187443	93,1741104	98,8529129	94,2313461	105,820572	112,073608	404,32077	20,5143318	563,147827	0
16	0,23051216	3,28349137	13,6426334	44,3914185	104,491623	284,396423	19,7187443	93,1741104	98,8529129	94,2313461	105,820572	112,073608	403,779205	20,5143318	563,147827	0
17	0,22486728	3,19959498	13,3545466	43,4718094	102,933495	284,410492	19,7187443	93,1660767	98,8529129	93,1312408	105,820572	112,073608	397,098267	20,5143318	563,147827	0
18	0,22962618	3,310776	13,7280769	44,6025009	105,045975	284,070801	18,8799877	93,1660767	98,8529129	93,1312408	105,820572	113,43988	405,558319	20,5143318	563,147827	0
19	0,21752794	3,33826327	13,1896858	43,5731316	103,43055	284,218964	18,4513664	92,1462326	98,8529129	93,1312408	105,820572	112,172058	415,953674	20,5143318	563,147827	0
20	0,22897461	3,39679837	13,8066673	44,4936523	104,60054	284,236359	18,0320606	93,1749954	98,8529129	93,1312408	104,739327	112,172058	413,041931	20,5143318	563,147827	0

Figura 1.4: Estructura del set de entrenamiento inicial

- Periodos de entrenamiento:
  - Periodo de buen funcionamiento: 08/02/2017 – 01/04/2017
  - Tipo de dato: media horaria
  - Frecuencia: 6h
  - Periodo de mal funcionamiento: 01/07/2017 – 01/08/2017
  - Tipo de dato: media horaria
  - Frecuencia: 6 h
- Periodos de evaluación:
  - Periodo de buen funcionamiento: 01/01/2017 – 28/01/2017
  - Tipo de dato: media horaria
  - Frecuencia: 6h
  - Periodo de mal funcionamiento: 01/08/2017 – 28/08/2017
  - Tipo de dato: media horaria
  - Frecuencia: 6 h

Como ya he comentado anteriormente, los periodos han sido escogidos porque se me informó que tras de una revisión hecha por el fabricante de los propios compresores, el compresor B estuvo trabajando durante un tiempo con niveles que variaban de lo habitual, sin afectar al nivel de compresión final pero sí a los rátios de compresión entre etapas.

Retomando el almacenamiento de datos, esta información se recopiló en dos '.csv'. Un csv contiene los datos para el entrenamiento y el otro para la evaluación.

Una vez obtenida la información, me centré en la lectura y la manipulación de los datos. Esto fue bastante sencillo puesto que es básico de programación. Posteriormente empecé con la declaración del modelo de la red neuronal. Este era un modelo básico que en pocos días hice que compilara, pero daba una precisión del 68 %. Como este resultado era bastante pésimo dediqué el resto del tiempo a buscar herramientas, métodos y algoritmos que me pudieran ser útiles para mejorar la predicción de la red neuronal.

Proceso de refinamiento

Después de entender cada elemento de la red neuronal, me di cuenta que te para mejorarla es necesario esquematizarse dónde estamos, qué queremos conseguir y cómo podemos hacerlo. Es muy habitual que los resultados no sean los esperados y por ello hay que esquematizarse el proceso y volver atrás siempre que sea necesario. Para ello utilizamos el siguiente esquema:

Figura 1.5: Esquema organizativo del curso de redes neuronales (Ver [8])

Una vez visto los resultados, con eficacia del 68 %, era evidente que había que retroceder de

la etapa 5 a la 4 y buscar posibles mejoras para aumentar la eficiencia de nuestro algoritmo. Asimismo, se decide que hay que cuestionarse cada parámetro de la red neuronal con el fin de encontrar una mejor combinación de estos. Estas son las preguntas base para crear un buen modelo de red neuronal:

1. ¿Cuántas neuronas tendrán cada una de las capas?

2. ¿Cuál será el valor de peso de los Dropouts?

3. La red neuronal trabaja con bloques de datos, ¿Qué peso deben tener esos bloques de datos?

4. ¿Cuántas veces repetimos el proceso de entrenamiento para no sobreentrenar la red neuronal inútilmente?

La mejor opción que se encontró para elegir los parámetros cruciales de la red neuronal se llama **Finne Tunning**. Esto no es más que un algoritmo que, mediante las funciones de Python que se presentaran a continuación, prueba todas las posibles combinaciones con el fin de mostrarte la óptima. A grandes rasgos, es lo mismo que automatizar el proceso de prueba y error.

¿Qué funciones son necesarias para entender este proceso?

Las clases KerasClassifier y KerasRegressor de Keras toman un argumento build_fn que es el nombre de la función a llamar para crear su modelo. Se debe definir una función llamada para definir tu modelo, y esta lo compila y lo devuelve. En el código diseñamos un método para build_model() que crea una simple red neuronal multicapa para el problema.

Antes de explicar la diferentes funciones que utiliza este proceso es necesario hacer una breve mención al método de **validación cruzada k-fold** (Ver [9]). En este, los datos de muestra se dividen en K subconjuntos. En cada iteración, uno de los subconjuntos se utiliza como datos de prueba y el resto $(K-1)$ como datos de entrenamiento y esto se repite en cada iteración. El proceso de validación cruzada es repetido durante k iteraciones, con cada uno de los posibles subconjuntos de datos de prueba.

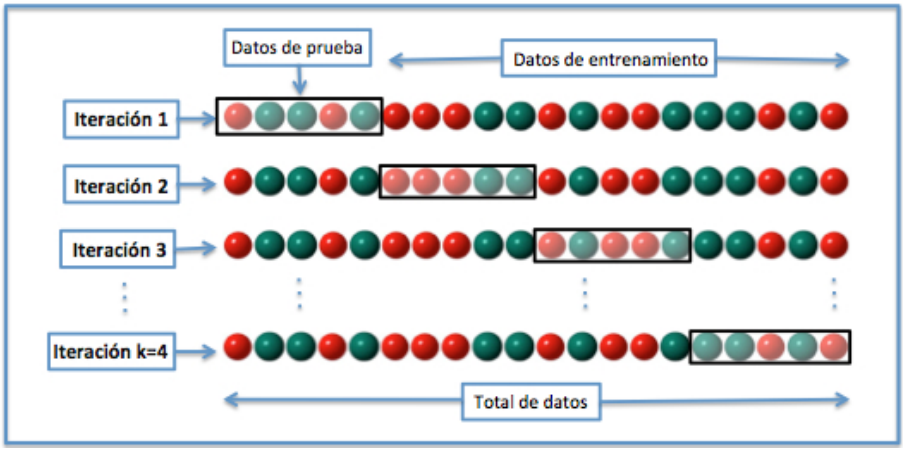


Figura 1.6: Validación cruzada k-fold

Función **KerasClassifier** (Ver documentación [2]): Esta tiene los siguientes argumentos:

1. Build_fn: Llama a la función que crea y compila la red neuronal.
2. Verbose: Nivel de información que se muestra en la salida por terminal. Le ponemos a 0 para que no muestre salida.
3. Batch_size: Es el número de veces que se realiza la validación cruzada, es decir, es n.
4. Epoch: Número de veces que repite el entrenamiento.

Los argumentos epoch y bach_size se agrupan automáticamente y se pasan a la función fit(), esta es ejecutada internamente por la función KerasClassifier. La función devuelve un modelo Keras, que luego se usará para ajustar / predecir.

Función **GridSearchCV()** (Ver documentación [3]): Esta tiene los siguiente parámetros.

1. 1º argumeto: Objeto de la clase KerasClassifier a utilizar para el ajuste de datos.
2. Grid_param: Diccionario con nombres de parámetros.
3. Scoring: Función de puntuación utilizada en los datos extendidos para elegir los mejores parámetros para el modelo.
4. Cv: : Determina la estrategia de división de validación cruzada. Hay muchos tipos de validación cruzada pero es un entero, determina el número de pliegues en StratifiedKFold si y es binario o multiclase y el estimador es un clasificador, o el número de pliegues en KFold de lo contrario. Si ninguno, es equivalente a cv = 3.

Ahora vamos a poner un ejemplo de Finne Tunning, con el fin de explicar más claramente cómo se encuentra la mejor combinación de parámetros de la red neuronal.

Este es un ejemplo de uso de la validación cruzada:

```
parametros = {'n1': [8,16], 'n2': [8,16]}
estimator = KerasClassifier(build_fn=build_model, verbose=0, batch_size
   =8, epochs=100)
grid_search = GridSearchCV(estimator=estimator, param_grid=parametros,
scoring='accuracy',cv=10)
grid_search.fit(X_train,Y_train)
print(grid_search.best_params_)
```

Para cada parámetro que queramos refinar, definimos un diccionario con las opciones que más adecuadas sean. En este caso n_1 y n_2 hacen referencia al número de neuronas que tendrán las dos capas internas, se probará con 8 y 16 neuronas. Este proceso ocupa muchos recursos, ya que prueba todas las posibles combinaciones con los parámetros elegidos y muestra por pantalla la mejor combinación.

En conclusión, si hay un parámetro que no tenemos claro con qué valor definirlo, este es un buen aliado puesto que Finne Tunning automatiza el proceso de prueba y error. En este caso terminó la red a un 98 % de precisión.

Obtención de los primeros resultados

Después de obtener buenos resultados en la red neuronal del compresor C1-B, se elevó el volumen de datos de evaluación. El objetivo de esto era ver en global como funcionaba la red neuronal y si el entrenamiento era bueno. Por tanto, se crearon 2 .csv de evaluación para cada compresor donde la diferencia es el tipo de dato. Un archivo contiene el registro de las medias horarias de cada hora y otro el registro del valor puntual de cada hora.

A continuación, se mostrarán algunos resultados:

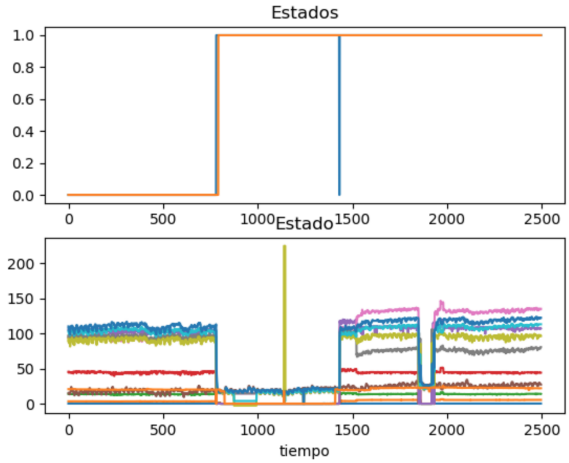


Figura 1.7: Resultados obtenidos de C1-B con **valores puntuales registrados cada 1 hora**

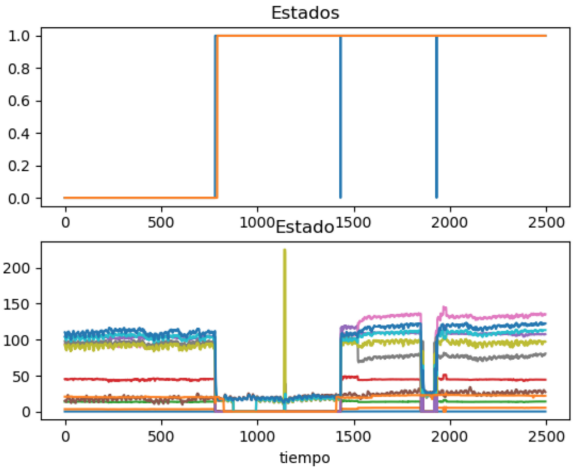


Figura 1.8: Resultados obtenidos de C1-B con **medias horarias registradas cada 1 hora**

Explicación de las gráficas:

- La subgráfica superior hace referencia al valor binario. En azul la predicción y en naranja el bit que indica el funcionamiento real del compresor. Recordemos que cuando es 0 representa que va bien y cuando es 1 representa que no funciona correctamente.
- La subgráfica inferior hace referencia a los valores de los sensores en las distintas tomas. Se utiliza para ubicarse, tomando de referencia la parada (donde los sensores valen 0).

El eje horizontal hace referencia al índice de tomas, se tomaron 2500, esto hace imposible poner una fecha porque sería caótico. Pero con el índice de la toma es muy fácil acceder al día y a la hora en la que se tomó.

Si los resultados hubieran sido malos, hubiese vuelto a la fase anterior y hubiera ajustado la red neuronal para una mejor predicción. En este punto el proyecto se encontraba muy avanzado (al menos la red neuronal del compresor C1-B). Fue así que tuve que hacer una presentación para los empleados que más en contacto estaban con los compresores y explicarles cómo estaba yendo el proyecto y que función iba a tener. Esto también se realizó para que ellos me aportaran un punto de vista más encarado al Hardware de la máquina o que me explicaran qué es lo que esperaban de ella y cómo podía hacer que se adaptase a lo que necesitaban.

Finalmente, antes de ponerme con el desarrollo del proyecto a tiempo real, lo que hice fue ajustar la red neuronal a otro periodo de tiempo. Recientemente se había dado un fallo (fallo en 03/02/2020 de C1-B) y analice y ajuste la red para ese fallo. El entrenamiento era el mismo pero el periodo de evaluación era el siguiente.

Periodo de evaluación: 01/02/2020 0:00:00 a 06/02/2020 0:00:00

El fallo fue avisado a las 9:01 A.M., cuando se paró el equipo. Pero como podemos ver en la gráfica obtenida de la plataforma Exacuantum, entorno las 4:15 a.m los sensores ya presentaban irregularidades. A continuación, vemos cómo se comportan los sensores en el periodo en el que se detecta el fallo.

Figura 1.9: Interfaz de Exacuantum

Valores Puntuales registrados cada 10 minutos

La siguiente gráfica muestra el estado del compresor evaluado por la red junto a los valores de los sensores. En la gráfica que se muestra a continuación debido al volumen de datos, no se aprecia el momento exacto donde se detecta el primer fallo. Por eso más abajo se muestra el momento exacto de tiempo en el que la red neuronal detecta que el funcionamiento de la máquina no es correcto.

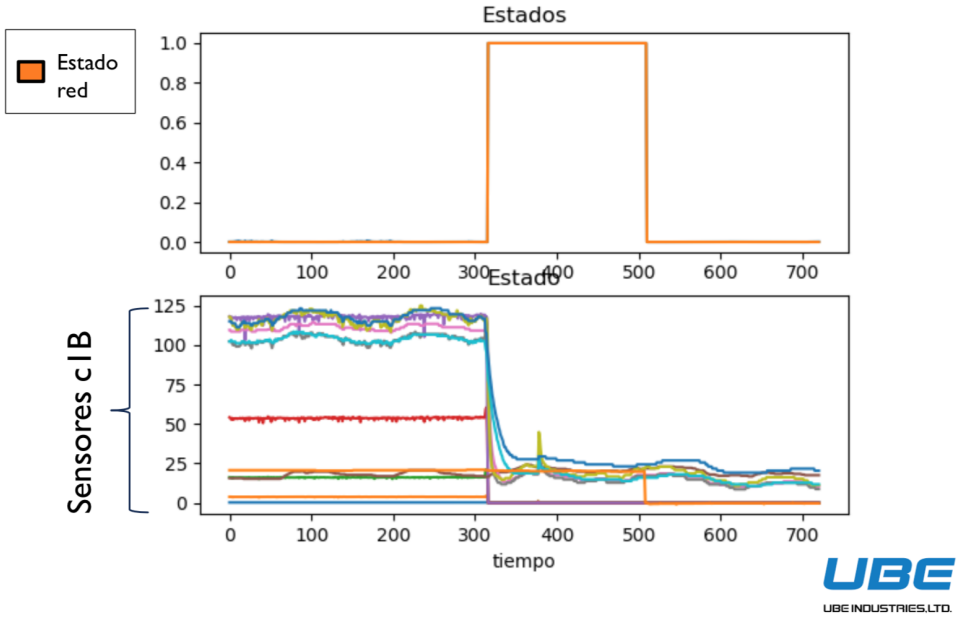


Figura 1.10: Resultados de la evaluación del modelo del compresor C1-B

```
(base) C:\Users\epalomar\Desktop\definitivo>python red_def_ev
Using TensorFlow backend.
la primera toma de error se dio en:  03/02/2020 4:00
El primer error ha sido detectado en:  03/02/2020 4:40
```

Figura 1.11: Salida terminal

En conclusión, la red neuronal tardo 40 minutos, es decir, 4 tomas de 10 minutos en ver que el compresor no estaba funcionando correctamente. Si la red neuronal fuera perfecta a las 4:00 hubiera detectado la primera toma de error. Los resultados de la evaluación no son malo,s puesto que hasta las 9:01 no saltaron los avisos preinstalados en el compresor de detección de fallo. En mi opinión, 5 horas de margen para poder solucionar el error antes de que la máquina se pare completamente es un buen resultado.

1.3.3. Proyecto final

Finalmente, me dispuse a crear mi proyecto final. Este consiste en exactamente lo mismo, pero ahora solo evaluará un instante de tiempo. Como ya he comentado anteriormente este era el objetivo del proyecto de prácticas. El correcto funcionamiento de la red no se consigue con un único programa, a continuación mostraré un esquema de los programas que componen el correcto funcionamiento y qué utilidad tienen.

Esquema de guardado:

Lo primero que debemos entender es que la red neuronal tiene que guardarse de una determinada manera. Esta debe guardarse entrenada para que cuando queramos utilizarla no sea necesario entrenarla y podamos comprobar el estado de un compresor sin un exceso de cálculo computacional. Este proceso de guardado se ejecuta en un ordenador cualquiera y termina generando 2 archivos donde se almacena la información necesaria de la red neuronal para que pueda ser utilizada sin previo entrenamiento.

A continuación se muestra un esquema de cómo se guarda el modelo de la red neuronal y sus respectivos pesos.

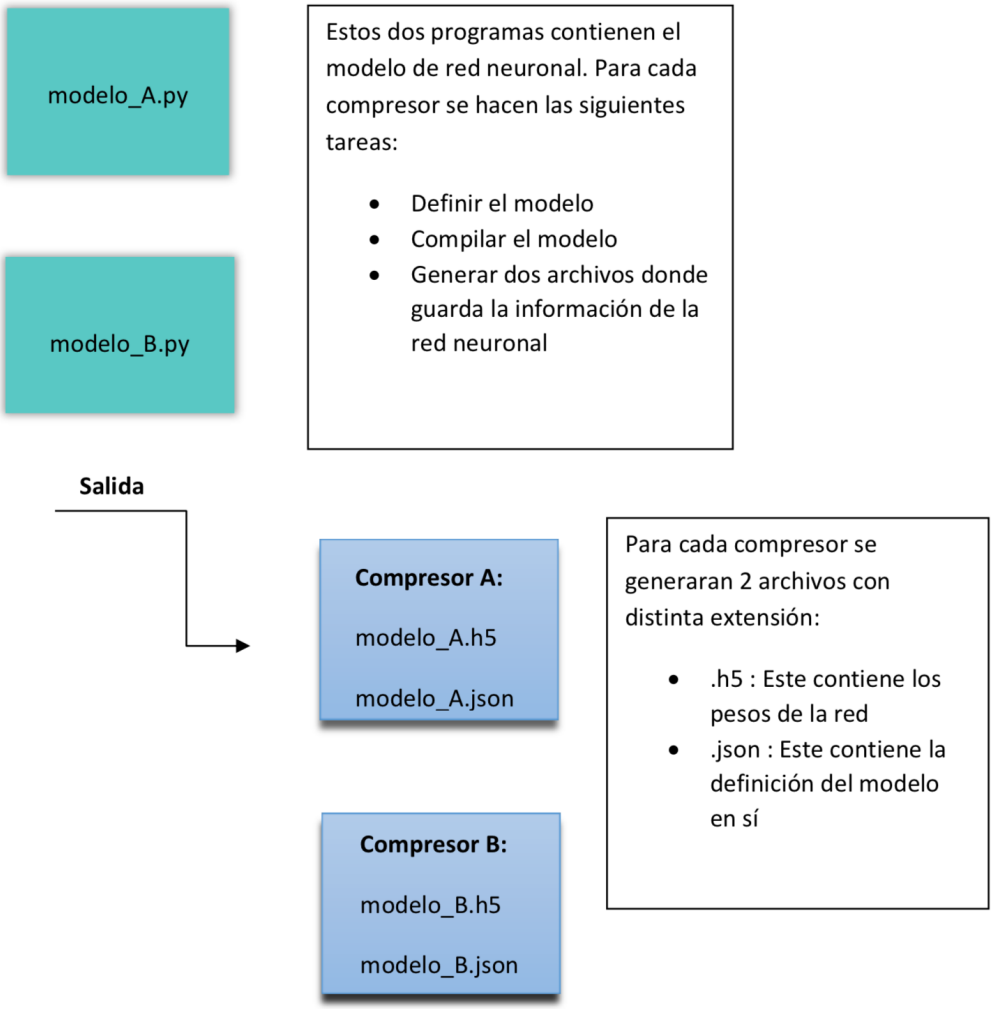


Figura 1.12: Esquema de guardado de la red neuronal

Una vez esquematizado el guardado de la red neuronal, explicaré más detalladamente qué hacen los programas desarrollados en Python.

Modelo_A.py / Modelo_B.py

Estos dos programas contienen información parecida. Los compresores son parecidos pero no iguales. Por tanto, los dos programas se encargan de leer su respectivo set de entrenamiento, procesando y almacenando en diferentes estructuras de datos la información proporcionada.

Posteriormente cada programa define su modelo con los parámetros más adecuados. Este proceso ha sido determinado en las semanas anteriores y es el más tedioso de toda la programación de la red neuronal. No voy a extenderme más en explicar la lectura y el entrenamiento puesto que se hace identicamente igual que en la fase anterior dónde tratábamos de evaluar el funcionamiento de la red.

Finalmente los dos programas guardan los modelos por separado y generan una salida. Esta salida contiene 2 archivos, uno con extensión .h5 y otro con extensión .json. Recordemos que el archivo .json guarda el modelo en sí y el archivo .h5 los pesos adecuados con el que la red ha calculado que se obtienen mejores resultados.

Podemos ver el código en el Anexo A.

Esquema de cargado:

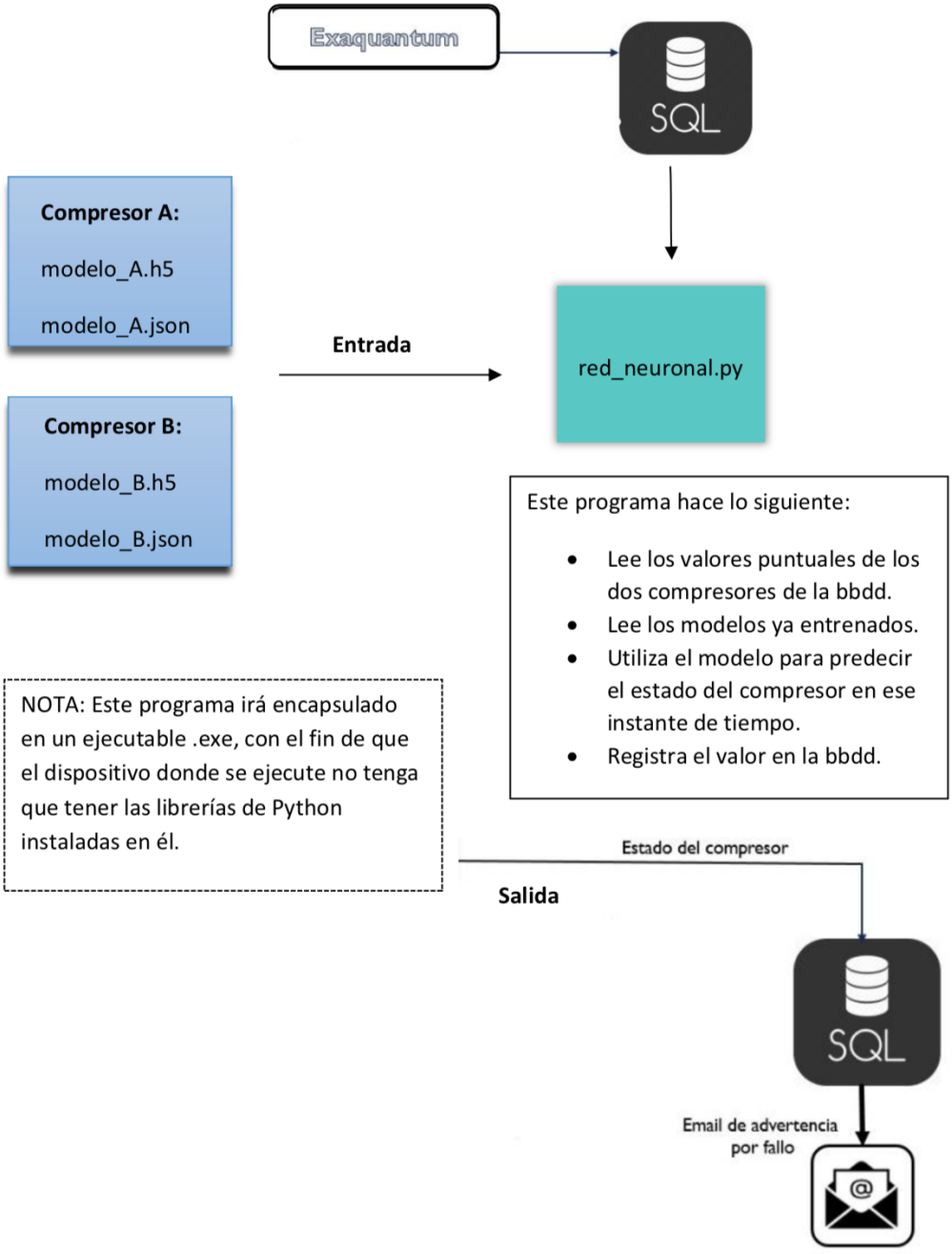


Figura 1.13: Esquema de cargado de la red neuronal

A continuación, se explicará con más detalle el funcionamiento de la red neuronal y de cada programa que la compone:

Red_neuronal.py

Este programa será el que se automatice en un servidor y obtenga el estado del compresor. Este tiene diversas tareas diferenciadas. A continuación, explicaré con más detalle cada uno de estas tareas:

- **Conexión mediante SQL para la lectura** de las mediciones de los sensores en un instante de tiempo.

  Para esta conexión utilizaremos la librería pyodbc. Aclaro algunos campos importantes:

  - Nombre del servidor: CTOPRO\SQLEXPRESS
  - Nombre de la base de datos: UBECPM
  - Nombre de la tabla: V_MachineLearning_Raw

  A continuación, guardo toda la información obtenida en vectores de datos, uno para cada compresor. Por tanto, para cada compresor tendré la información de un instante de tiempo con la fecha y los valores de los sensores.

  El formato de salida de cada una de las filas de la tabla es el siguiente:

```
('nombre_tag_sensor', 'nombre_compresor',fecha, valor_medicion)
```

  Entonces tengo para cada línea una tupla con 4 elementos, lo que me facilita mucho el acceso a cada elemento. Como mi objetivo es simplemente, para cada compresor, crear una lista que contenga la fecha y los valores de los sensores en un orden ya establecido por la lectura de mi red neuronal. Obtendremos dos listas c_1 y c_2 que posteriormente mandaremos a la red como atributos de entrada (w_i),para que obtenga resultados.

- **Lectura** de los modelos ya entrenados.

  Esta tarea consiste en cargar el modelo y los pesos. Para que la lectura sea sencilla, Python incorpora dos funciones que leen automáticamente los ficheros: model_from_json() y load_weights. En el anexo A se encuentra el código donde se puede observar que la lectura es sencilla y muy visual.

- **Evaluación** del estado del compresor.

  Este paso es exactamente igual que cuando evaluábamos un histórico de datos, pero en este caso solo se evalúa un instante de tiempo. Esto resulta computacionalmente poco pesado.

- **Conexión mediante SQL para almacenar** el estado del compresor en la BBDD.

  Finalmente una vez evaluado el estado del compresor, el resultado se tiene que guardar en una tabla de la base de datos. La conexión mediante SQL para almacenar es tan simple como:

```
cursor.execute('''INSERT INTO MachineLearning_Results (Equipment,
   TS, FaultDetection) VALUES ('C-444-1A','2020-01-22 15:00:00',0),
    ('C-444-1B','2020-01-22 15:00:00',1)''')
```

  El nombre de la tabla donde almacenamos los resultados es: MachineLearning_Results. Y como podemos ver se almacena para cada compresor la fecha completa y el bite de estado.

- **Automatización del compresor**

  El programa red Red_neuronal.py estará encapsulado en un ejecutable .exe y se automatizará su ejecución en un servidor. El programa desarrollado para encapsular programadas en Python se encuentra en el Anexo A.

1.4. Conclusiones y valoración personal

Mi grado de satisfacción con lo aprendido en las prácticas es muy alto. El campo del Machine Learning es muy amplio y útil. Ahora mismo es una parte de la computación que más en desarrollo está y esto me ha servido para tener una iniciación muy completa a estos conocimientos.

Por un lado, creo que con un poco más de tiempo me hubiera dado tiempo a hacer la red neuronal de compresor A. Pero no ha sido posible, aunque no me quedo descontenta puesto que se me avisó que el compresor B era el más crítico. También he de destacar que el trabajo del compresor B ha resultado ser muy productivo y estoy contenta con los resultados.

Por otro lado, mi aprendizaje no habría sido posible sin el equipo profesional de la empresa. Han estado todo el tiempo ayudándome en cualquier problema que me surgiera. Sobretodo al principio del periodo de prácticas donde tenía que aprender a usar herramientas propias de UBE. Asimismo han realizado tareas auxiliares como la creación de tablas en bases de datos de la empresa para mi posterior uso en el proyecto. En definitiva, si he aprendido tanto, en gran parte, ha sido gracias a ellos.

Capítulo 2

Memoria TFG

2.1. Motivación y Objetivos

A grandes rasgos, el estudio de la Mecánica Cuántica es importante por varias razones. En primer lugar porque pone de manifiesto la metodología esencial de la Física, dando comprensión de aquello que no podemos ver, y con ello, mejora la descripción y predicción del comportamiento del universo. En segundo lugar porque tuvo un gran éxito dando respuestas válidas a casi todos los problemas en los que se la ha aplicado y en los que la Mecánica clásica no ha hallado solución. En tercer lugar, porque hoy en día es la herramienta teórica básica para numerosas disciplinas de gran importancia, como puede ser la Física Molecular, Atómica y Nuclear, la Física de la Materia Condensada y la Física de Partículas.

Asimismo, se recordará la ecuación de Schrödinger. Esta formulación fue muy importante en la teoría de la mecánica cuántica, donde representa un papel análogo a la segunda ley de Newton en la mecánica clásica. Más adelante se verá que con la mecánica cuántica no se puede saber dónde se encuentra un electrón (Heisenberg), pero sí define la región en la que puede encontrarse en un momento dado. Cada solución de la ecuación de ondas de Schrödinger, ψ, denota un posible estado del electrón. Por tanto, la función de onda facilita, gracias a su obtención mediante la ecuación de ondas de Schrödinger, el conocimiento del estado energético del átomo.

En este caso concreto, se ha encontrado atractivo al suceso que se da cuando la partícula en cuestión está restringida a una superficie de revolución. Restringiendo la partícula a dicha superficie, con ciertas condiciones, se obtiene un potencial geométrico en la fórmula de Schrödinger. Posteriormente, se profundizará en la explicación y finalmente concluiremos con un programa, con él se pretende visualizar más claramente los resultados obtenidos.

2.2. Introducción histórica a la mecánica cuántica

La Física se basa en medidas y observaciones experimentales de la realidad que nos rodea. Es decir, en cuantificar o caracterizar los distintos fenómenos naturales.

Hasta finales del siglo XIX se pensaba que la física clásica, derivada de los postulados de Newton, podía resolver cualquier problema. Pero varios resultados experimentales a los que no se les hallaba solución revolucionaron el mundo de la física. A lo largo de este periodo hubo numerosos descubrimientos y teorías. Como ideas clave a destacar tenemos: la dualidad onda-partícula y principio de incertidumbre.

La mecánica cuántica nace a principios del siglo XX con el fin de encontrar una solución a los problemas que las teorías conocidas hasta el momento de la física clásica eran incapaces de explicar. Estudiando estos fenómenos anómalos, los investigadores comenzaron a ver similitudes entre el comportamiento de las ondas y las partículas.

Al estudiar el espectro de luz emitido por el cuerpo negro se tuvieron las primeras sospechas de la **relación onda-partícula**. Fue el físico Alemán Max Planck el que trató de encontrar una solución a la catástrofe ultravioleta y después de numerosos pasos intermedios, llegó a la conclusión de que la energía no se transmite de forma continua, sino mediante pequeños paquetes o cuantos de energía llamados fotones. La energía que porta un fotón es directamente proporcional a su frecuencia, $E = hv$ donde h es la constante que lleva el nombre de su descubridor, Max Planck. Con este descubrimiento inició el desarrollo de las primeras ideas de la física cuántica. Después de que Planck plantease esta hipótesis, esta fue usada por Albert Einstein para explicar el efecto fotoeléctrico, y posteriormente también fue usada por Niels Bohr al formular un modelo atómico con niveles de energía cuantizados.

Profundizando más en el **efecto fotoeléctrico**. El experimento consistió en emitir rayos de luz contra metal, esto puso de manifiesto la dualidad onda-partícula. Ya que esto les condujo a afirmar que la luz se comporta como ondas pudiendo producir interferencias y difracción, pero intercambia energía de forma discreta en paquetes de energía, llamados fotones. Este fenómeno solo se podía explicar si luz toma algunas propiedades de la materia. Por esta explicación del efecto fotoeléctrico Einstein recibió el Premio Nobel de Física en 1921.

Otra evidencia de la dualidad onda-partícula fue el **efecto Compton** que surgió al proyectar un haz de luz con longitud de onda λ en un electrón y ver que este se dispersaba cambiando su longitud a $\lambda + \Delta\lambda$, es decir aumenta la longitud de onda, por lo que se dispersa con menor energía. Este efecto es de especial relevancia científica, ya que no puede ser explicado a través de la naturaleza ondulatoria de la luz. Y por tanto, esta debe comportarse como partícula para poder explicar dichas observaciones.

En conclusión, como hemos podido ver hay fenómenos en los que diríamos que las partículas son ondas pero otros en los que diríamos que las ondas son partículas, en esto consiste la dualidad onda-materia. Pero además de esto debemos tener en cuenta que en la física cuántica no todo se puede medir con unas medidas exactas, y en esto se inspiró Werner Heisenberg en 1927 para llegar al **principio de incertidumbre**. Este principio afirma que no se puede determinar simultáneamente, en términos de la física cuántica, la posición y el momento lineal (cantidad de movimiento) de un objeto dado. Esta fue la base para que Schrödinger llegara a la función de onda de una partícula $\Psi(x,t)$, dando su amplitud en función de la posición y el tiempo y no su intensidad que coincide con la densidad de probabilidad de esta.

Todas estas investigaciones culminaron en la interpretación de Copenhague de 1927, que no fue otra cosa que la interpretación de la mecánica cuántica considerada tradicional y con el objetivo de reconciliar el contra intuitivo dualismo onda-partícula de un modo adecuado a la comprensión humana.

2.3. Conceptos previos: Mecánica cuántica

La ecuación de Schrödinger proporciona una ecuación determinista para explicar la evolución temporal de la **función de onda** $\psi(\mathbf{x};t)$, que es una forma de representar el estado físico de un sistema de partículas y, por tanto, del estado físico del sistema en el intervalo comprendido entre dos medidas. [Ver [12]]

En la formulación moderna, la función de onda se interpreta como un objeto mucho más abstracto, que representa un elemento de un cierto espacio de Hilbert de dimensión infinita que agrupa a los posibles estados del sistema.

En la geometría diferencial, un **tensor métrico** es un tipo de función que toma como entrada un par de vectores tangente v y w en un punto de una superficie (variedad diferenciable) y produce un número real $g(v,w)$ de una manera que generaliza muchas de las propiedades conocidas del producto escalar de los vectores en el espacio euclidiano. De la misma manera que un producto escalar, los tensores métricos se utilizan para definir la longitud de y el ángulo entre los vectores tangentes.

En mecánica cuántica, bajo la **interpretación probabilística**, las partículas no pueden ser consideradas puntuales, sino que se encuentran deslocalizadas espacialmente antes de realizar una medida sobre su posición. La densidad es una distribución que determina la probabilidad espacial de una o más partículas idénticas.

En el sentido físico, la función densidad $\rho\left(\vec{r},t\right)$ de un sistema determina la probabilidad de encontrar un electrón en la posición $\vec{r}$ en un tiempo t. Como tal es una función positiva y real.

La integral de la densidad sobre todo el espacio se normaliza al número total de partículas del sistema:

$$\int dx\, \rho(x) = N$$

En mecánica cuántica, la densidad puede ser obtenida a partir de una función de onda de N partículas $\Psi^{(N)}$ como

$$\rho(x) = \int\, dx_2\, ...\, dx_N\, |\Psi^{(N)}(x,x_2,...,x_N)|^2 \rho(x) = \int\, dx_2\, ...\, dx_N\, |\Psi^{(N)}(x,x_2,...,x_N)|^2$$

Para terminar, un **pozo cuántico** es la denominación que recibe un pozo de potencial que

confina, en dos dimensiones, partículas que originalmente tenían libertad para moverse en tres, forzándolas a ocupar una zona acotada.

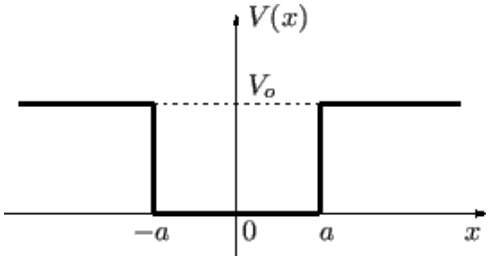


Figura 2.1: Pozo cuántico

2.4. Origen de la ecuación de Schrödinger

Schröndinger propuso (1927) una ecuación de onda para explicar el movimiento de partículas subatómicas [ver [14]]. Es decir, que esta ecuación es la equivalencia a mecánica cuántica de la fórmula de Newton $F = m \cdot a$. Su idea era describir cualquier partícula con propiedades de onda mediante una ecuación matemática denominada función de onda (Ψ).

Esta recoge todos los conceptos cuánticos para la descripción del átomo de hidrógeno:

$$\boxed{\hat{H}\Psi = i\hbar\frac{\partial y}{\partial t}} \tag{2.1}$$

con $\hat{H} = -\frac{\hbar^2}{2m}\Delta + V(x)$, siendo Δ el Laplaciano de $\mathbb{R}^3$ y V la función potencial.

2.4.1. Ecuación de Schrödinger independiente del tiempo

La ecuación de Schrödinger independiente del tiempo predice que las funciones de onda pueden tener la forma de ondas estacionarias, denominados estados estacionarios.

$$\hat{H}\Psi = E\Psi$$

Cuando el operador Hamiltoniano ($\hat{H}$) actúa sobre cierta función de onda Ψ, y el resultado es proporcional a la misma función de onda Ψ (siendo una autofunción del operador Hamiltoniano), entonces Ψ es un estado estacionario, y la constante de proporcionalidad, E, es la energía del sistema Ψ.

La ecuación de Schrödinger combina el concepto de la conservación de la energía y la formulación matemática del concepto de Onda.

2.5. Mecánica cuántica de una partícula restringida

2.5.1. Introducción

El movimiento de una partícula en un dominio unidimensional o bidimensional en un espacio tridimensional cartesiano es un problema bien conocido en la mecánica clásica. Por lo general, es tratado de diferente manera. A continuación, se verán los dos enfoques:

- **Enfoque newtoniano:** Se cree que la partícula se mueve libremente (es decir, sin restricciones) en el espacio tridimensional, pero está sujeta a fuerzas espaciales que mantienen su velocidad orientada a lo largo de un espacio preseleccionado en todos los instantes de tiempo.
- **Enfoque lagrangiano:** La restricción se introduce desde el principio, a través de coordenadas generalizadas. Además, los cálculos proceden sin ninguna mención necesaria al espacio donde la superficie (o curva) se ha supuesto que está.

Para las restricciones puramente espaciales consideradas hasta el momento, estos dos enfoques producen las mismas ecuaciones de movimiento. Siendo la elección entre uno de ellos una cuestión de conveniencia. En mecánica cuántica, sin embargo, la situación es mucho más confusa. Entonces, ¿Qué enfoque escogemos?

1. En el primer enfoque o newtoniano nos encontramos con que la ecuación de Schrödinger está lista para usar, pero tendremos que lidiar con una situación desconocida en la que la restricción solo se puede considerar como una especie de proceso limitante. De hecho, debido a las relaciones de incertidumbre, hay que tener en cuenta las siguientes consideraciones:
    - Las infinitas 'fuerzas de compresión', para contener la propagación transversal del paquete de ondas, que estará presente incluso en el caso de que una partícula se mueva a lo largo de una superficie perfectamente plana.
    - El análogo cuántico de las fuerzas clásicas para doblar el momento de la partícula.
2. Si elegimos el segundo enfoque o lagrangiano, podemos olvidar por el momento todas las propiedades relacionadas con el espacio externo, pero aún tendremos que elaborar un procedimiento de cuantización adecuado para el movimiento curvo a priori.

Siguiendo el razonamiento de Da Costa [13], elegimos el primer enfoque y vemos hasta dónde podemos llegar. El punto principal es la elección de las fuerzas espaciales que simulan la restricción mecánica en un cierto límite. Para desarrollar mejor nuestro razonamiento, consideremos una restricción de superficie.

Puesto que en mecánica cuántica ya no podemos predecir la posición de la partícula con precisión puntual, es perfectamente natural considerar únicamente las fuerzas restrictivas que son ortogonales a la superficie en todos los puntos del espacio. Es precisamente en estos puntos donde posiblemente se pueda encontrar la partícula. Cabe destacar que posteriormente también se desarrollará un procedimiento similar para las curvas. Esta idea se puede poner en práctica fácilmente considerando un potencial constante sobre la superficie, pero que aumenta bruscamente con cada pequeño desplazamiento en la dirección normal. De tal manera que proporciona una 'reacción' normal en un entorno fino de la superficie en cuestión. Es posible que se puedan encontrar requisitos más débiles, pero el que presentamos aquí es perfectamente válido para lo que queremos mostrar.

En conclusión, se considerará la restricción como el límite de un potencial atractivo infinitamente fuerte que mantiene la partícula permanentemente unida a la superficie preestablecida. Para que el límite sea independiente del tipo de potencial atractivo, debemos obtener algún tipo de separación de la ecuación de Schrödinger. En esta separación, la parte de la superficie de la función de onda debe de obedecer a una ecuación especial que no contenga la variable transversal que aparece en el potencial restrictivo. A continuación, veremos esto con más detalle.

2.5.2. Partícula unida a una superficie

Sea una partícula de masa m unida permanentemente a la superficie S, con las siguientes ecuaciones paramétricas: $\vec{r} = \vec{r}(q_1, q_2)$ donde $\vec{r}$ es la posición vectorial de un punto arbitrario P sobre la superficie S. La porción del espacio de un entorno de S se puede parametrizar como en la figura [2.2].

$$\vec{R}\,(q_1, q_2, q_3) = \vec{\mathrm{r}}\,(q_1, q_2) + q_3 \hat{N}\,(q_1, q_2) \tag{2.2}$$

donde $\hat{N}\,(q_1, q_2)$ es vector normal en el punto P. El valor absoluto de la coordenada q_3 da, en los puntos donde (2.2) no es singular, la distancia entre la superficie S y el punto Q de coordenadas (q_1, q_2, q_3).

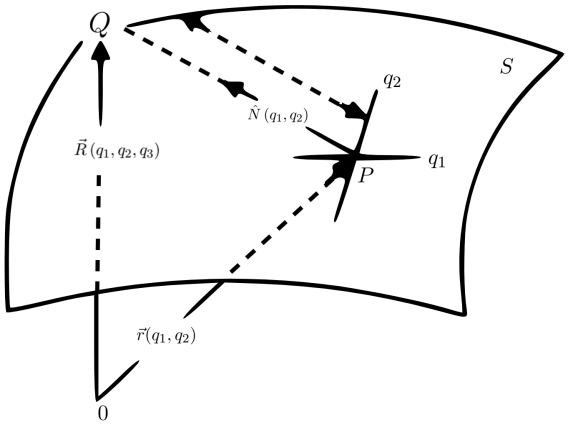


Figura 2.2: Sistema de coordenadas curvilíneo basado en una superficie S con las ecuaciones paramétricas $\vec{r} = \vec{r}(q_1, q_2)$.

De acuerdo con las ideas presentadas en el apartado anterior, ahora se considera el potencial espacial $V = V_\lambda(q_3)$ tal que:

$$\lim_{\lambda \to \infty} V_\lambda(q_3) = \begin{cases} 0, & q_3 = 0 \\ \infty, & q_3 \neq 0 \end{cases}$$

Siendo λ un 'parámetro de compresión' que mide la fuerza del potencial.

Antes de pasar a la ecuación de Schrödinger, repasaremos brevemente las propiedades matemáticas del sistema de coordenadas (2.2). Sean $g_{ij} = \frac{\partial \vec{r}}{\partial q_i} \cdot \frac{\partial \vec{r}}{\partial q_j}$ con $i, j = 1, 2$ los coeficientes covariantes de los tensores métricos de una superficie S. Denotamos $g = det(g_{ij})$ y $h_{ij} = h_{ji}$ son los coeficientes de la segunda forma fundamental. Como las derivadas del normal $\hat{N}(q_1, q_2)$ se encuentran en el plano que tiene:

$$\frac{\partial \hat{N}}{\partial q_i} = \sum_{j=1}^{2} \alpha_{ij} \frac{\partial \vec{r}}{\partial q_j} \tag{2.3}$$

con:

$$\begin{aligned} \alpha_{11} &= \tfrac{1}{g}\left(g_{12}h_{21} - g_{22}h_{11}\right), \quad \alpha_{12} = \tfrac{1}{g}\left(h_{11}g_{21} - h_{21}g_{11}\right) \\ \alpha_{21} &= \tfrac{1}{g}\left(h_{22}g_{12} - h_{12}g_{22}\right), \quad \alpha_{22} = \tfrac{1}{g}\left(h_{21}g_{12} - h_{22}g_{11}\right) \end{aligned} \tag{2.4}$$

Estas son las ecuaciones de Weingarten. De (2.2) y (2.3) obtenemos:

$$\begin{aligned} &\frac{\partial \vec{R}}{\partial q_i} = \sum_{j=1}^{2} \left(\delta_{ij} + \alpha_{ij} q_3\right) \frac{\partial \vec{r}}{\partial q_j}, \quad i,j = 1,2 \\ &\tfrac{\partial \vec{R}}{\partial q_3} = \hat{N}\left(q_1, q_2\right) \end{aligned} \tag{2.5}$$

En nuestro entorno tridimensional de S, las componentes covariantes del tensor métrico están dadas por

$$G_{ij} = G_{ji} = \frac{\partial \vec{R}}{\partial q_i}\frac{\partial \vec{R}}{\partial q_j}, \quad i,j = 1,2,3$$

Usando (2.5) y denotando la matriz traspuesta con el superíndice T, entonces se obtiene:

$$\begin{aligned} G_{ij} &= g_{ij} + \left[\alpha g + (\alpha g)^T\right]_{ij} q_3 + \left(\alpha g \alpha^T\right)_{ij} q_3^2 \\ G_{i3} &= G_{3i} = 0, \quad i = 1,2; G_{33} = 1 \end{aligned} \tag{2.6}$$

Llegados a este punto, ya podemos centrar nuestra atención en la ecuación de Schrödinger. Vamos a escribir el Laplaciano en las coordenadas curvilíneas (q_1, q_2, q_3), obteniendo:

$$-\frac{\hbar^2}{2m}\sum_{i,j=1}^{3} \frac{1}{\sqrt{G}}\frac{\partial}{\partial q_i}\left(\sqrt{G}\left(G^{-1}\right)_{ij}\frac{\partial \psi}{\partial q_j}\right) + V_\lambda\left(q_3\right)\psi = i\hbar\frac{\partial \psi}{\partial t} \tag{2.7}$$

Donde $G = det(G_{ij})$. Debido a la estructura de G dada en (2.6) podemos dividir el Laplaciano en dos partes:

- La parte de la superficie dada por $i, j = 1, 2$
- La parte normal definida por $i = j = 3$

Por tanto, se puede escribir como:

$$-\frac{\hbar^2}{2m}\mathfrak{D}\left(q_1, q_2, q_3\right)\psi - \frac{\hbar^2}{2m}\left(\frac{\partial^2\psi}{\partial q_3^2} + \frac{\partial}{\partial q_3}(\ln\sqrt{G})\frac{\partial\psi}{\partial q_3}\right) + V_2\left(q_3\right)\psi = i\hbar\frac{\partial\psi}{\partial t} \tag{2.8}$$

Dado que esperamos la existencia de una función de onda superficial y que dependa solo de las variables q_1 y q_2, introducimos una nueva función de onda χ tal que, en el caso de una separación $\chi\left(q_1, q_2, q_3\right) = \chi_t\left(q_1, q_2\right)\chi_n\left(q_3\right)$ se puede definir la probabilidad de densidad superficial $\left|\chi_t\left(q_1 q_2\right)\right|^2 \int \left|\chi_n\left(q_3\right)\right|^2 dq_3$. La transformación adecuada $\Psi \to \chi$ se puede inferir fácilmente del volumen dV expresado en términos de las coordenadas curvilíneas (q_1, q_2, q_2). Realmente, usando 2.5 obtenemos

$$dV = f(q_1, q_2, q_3)dSdq_3 \tag{2.9}$$

Donde $dS = \sqrt{g}\ dq_1 dq_2$ (área de la superficie) y

$$f\left(q_1, q_2, q_3\right) = 1 + \mathrm{Tr}\left(\alpha_{ij}\right)q_3 + \det\left(\alpha_{ij}\right)q_3^2 \tag{2.10}$$

La expresión (2.9) ahora da el resultado deseado:

$$\chi\left(q_1, q_2, q_3\right) = \left[f\left(q_1, q_2, q_3\right)\right]^{\frac{1}{2}}\psi\left(q_1, q_2, q_3\right) \tag{2.11}$$

Al introducir el anterior resultado en (2.8), obtenemos la siguiente ecuación:

$$\sqrt{f}\left[-\frac{\hbar^2}{2m}\mathfrak{D}\left(\frac{\chi}{\sqrt{f}}\right)\right] - \frac{\hbar^2}{2m}\left\{\frac{\partial^2\chi}{\partial q_3^2} + \frac{1}{4f^2}\left[\left(\frac{\partial f}{\partial q_3}\right)^2 - 2f\frac{\partial^2 f}{\partial q_3^2}\right]\chi\right\} + V_\lambda\left(q_3\right)\chi = i\hbar\frac{\partial\chi}{\partial t} \tag{2.12}$$

Ahora estamos listos para tener en cuenta el efecto del potencial $V_\lambda(q_3)$. Dado que cuando $\lambda \to \infty$ la función de onda 've' dos barreras de potencial en ambos lados de la superficie, su valor será significativamente diferente de cero solo para un rango muy pequeño de valores de q_3 alrededor de $q_3 = 0$. En este caso, podemos tomar $q_3 \to 0$ con seguridad en todos los coeficientes de la ecuación (2.2) (excepto en el término que contiene V_{λ_3}).

Entonces de (2.6) y (2.10) obtenemos:

$$-\frac{\hbar^2}{2m}\sum_{i,j=1}^{2}\frac{1}{\sqrt{g}}\frac{\partial}{\partial q_i}\left(\sqrt{g}\left(\bar{g}^1\right)_{ij}\frac{\partial\chi}{\partial q_j}\right)-\frac{\hbar^2}{2m}\left(\left[\frac{1}{2}\operatorname{Tr}\left(\alpha_{ij}\right)\right]^2-\det\left(\alpha_{ij}\right)\right)\chi- \\ -\frac{\hbar^2}{2m}\frac{\partial^2\chi}{\partial q_3^2}+V_\lambda\left(q_3\right)\chi=i\hbar\frac{\partial\chi}{\partial t} \tag{2.13}$$

La ecuación (2.13) ahora puede separarse fácilmente estableciendo $\chi = \chi_t\left(q_1, q_2, t\right)\times\chi_n\left(q_3, t\right)$, donde los subíndices t y n significan 'tangente' y 'normal', respectivamente. El procedimiento habitual da como resultado:

$$-\frac{\hbar^2}{2m}\frac{\partial^2\chi_n}{\partial q_3^2}+V_\lambda\left(q_3\right)\chi_n=i\hbar\frac{\partial\chi_n}{\partial t} \tag{2.14}$$

$$-\frac{\hbar^2}{2m}\sum_{i,j=1}^{2}\frac{1}{\sqrt{g}}\frac{\partial}{\partial q_i}\left(\sqrt{g}\left(\bar{g}^1\right)_{ij}\frac{\partial\chi_t}{\partial q_j}\right)-\underbrace{\frac{\hbar^2}{2m}\left(\left[\frac{1}{2}\operatorname{Tr}\left(\alpha_{ij}\right)\right]^2-\det\left(\alpha_{ij}\right)\right)}_{V_s(q_1,q_2)}\chi_t=i\hbar\frac{\partial\chi_t}{\partial t} \tag{2.15}$$

Expresión (2.14) es solo la ecuación de Schrödinger unidimensional para una partícula limitada por el potencial transversal $V_\lambda(q3)$, y puede ignorarse en todos los cálculos futuros.

Sin embargo, la expresión (2.15) es mucho más interesante, debido a la presencia del potencial de superficie $V_s(q_1, q_2)$.

Además usando (2.4) se puede reescribir este término de modo que nos sea más útil.

$$V_s\left(q_1,q_2\right)=-\frac{\hbar^2}{2m}\left(H^2-K\right)=-\frac{\hbar^2}{8m}\left(k_1-k_2\right)^2 \tag{2.16}$$

Donde k_1 y k_2 son las curvaturas principales de la superficie S y siendo

- Curvatura media:

$$H=\frac{1}{2}\left(k_1+k_2\right)=\frac{1}{2g}\left(g_{11}h_{22}+g_{22}h_{11}-2g_{12}h_{12}\right) \tag{2.17}$$

- Curvatura de Gauss:

$$K=k_1k_2=\frac{1}{g}\det\left(h_{ij}\right) \tag{2.18}$$

La dependencia de V_s respecto de q es especialmente notable, debido a la presencia de la curvatura media (H), ya que no se puede obtener los g_{ij}'s y sus derivadas (al contrario de lo que ocurre con la curvatura de Gauss (K)). Este resultado tiene una consecuencia importante, $V_S\left(q_1,q_2\right)$ no podrá ser el mismo para dos superficies isométricas.

Esto contrasta notablemente con los resultados de la mecánica clásica, donde el Lagrangiano del movimiento de la superficie libre

$$\mathcal{L}\left(q_1,q_2,\dot{q}_1,\dot{q}_2\right)=\frac{1}{2}m(\frac{ds}{dt})^2=\frac{1}{2}m\sum_{i,j=1}^{2}g_{ij}\left(q_1q_2\right)\dot{q}_i\dot{q}_j$$

depende solo de las propiedades métricas de la superficie. Por extraño que puede aparecer a primera vista, esto no es un resultado inesperado. Puesto que, independientemente de cuán pequeño sea el rango del valor que se supone para q_3, la función onda siempre se 'mueve' en una porción tridimensional del espacio. Por tanto, la partícula es 'consciente' de las propiedades externas del límite de la superficie S.

Ejemplo:

Para entender mejor las propiedades de $V_s\left(q_1,q_2\right)$ vamos a considerar una superficie en forma de tapa de libro obtenida doblando un plano alrededor de la superficie de un cilindro de radio a. Lo podemos ver en la figura [2.3]. Seleccionamos como parámetros los arcos de la sección transversal C y la coordenada cartesiana z perpendicular al plano de la figura.

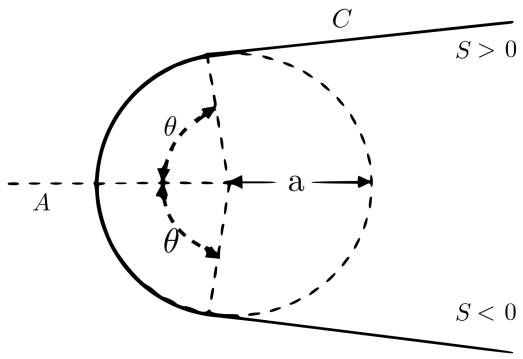


Figura 2.3: Sección transversal C de la superficie de la 'tapa del libro': Un plano doblado alrededor de la superficie de un cilindro de radio a. El punto medio A fue elegido para el origen del arco S (2 Dimensiones)

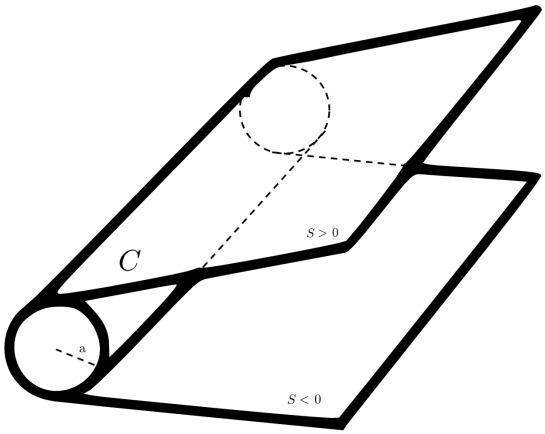


Figura 2.4: Un plano doblado alrededor de la superficie de un cilindro de radio a (3 dimensiones)

De las ecuaciones (2.15) a (2.18) se obtiene:

$$-\frac{\hbar^2}{2m}\left(\frac{\partial^2\chi_t}{\partial s^2}+\frac{\partial^2\chi_t}{\partial z^2}\right)-\frac{\hbar^2}{8m}k(s)^2\chi_t=i\hbar\frac{\partial\chi_t}{\partial t} \tag{2.19}$$

Donde $k(s)$ es la curvatura de C en el punto de arco S.

Si consideramos la solución $\chi_t(s,t)$, independientemente de z, se obtiene la ecuación de

Schrödinger unidimensional en presencia del potencial del pozo cuántico cuadrado.

$$V(s) = \begin{cases} V_0 = -\frac{\hbar^2}{8ma^2}, & |s| < a\theta \\ 0, & |s| > a\theta \end{cases} \tag{2.20}$$

Dado $\left[\left(2m/\hbar^2\right)|V_0|\,(a\theta)^2\right]^{1/2} = \theta/2 < \pi/4$, sabemos que solo tiene un estado ligado de energía E_0, $V_0 < E_0 < 0$. En el límite $a \to 0$, θ es constante y por tanto, el coeficiente de transmisión de (2.20) tiende a 0. Las dos partes, $S < 0$ y $S > 0$, de la figura [2.3] quedan entonces completamente inconexos, en contraposición con las soluciones habituales donde el término $V(s)$ está ausente.

Cabe destacar, en cuanto a los resultados anteriores, que el límite obtenido aquí no existe realmente. Algunos físicos creen que esos cálculos, aunque matemáticamente impecables, están mal concebidos desde el punto de vista físico, ya que involucran potenciales con fuerzas tangentes distintas de cero en cada entorno de la superficie S.

2.5.3. Partícula unida a una curva

Consideremos una partícula puntual de masa m, delimitada rígidamente a una curva C de arco q_1, siendo su ecuación paramétrica $\vec{r} = \vec{r}(q_1)$. Siendo el tangente, normal y binormal denotados respectivamente por $\hat{t}\,(q_1)\,, \hat{n}\,(q_1)$ y $\hat{b}\,(q_1)$.

Siguiendo el mismo razonamiento que en el apartado anterior, se introducirá un sistema de coordenadas curvilíneas basado en la curva C (Fig. 3):

$$\vec{R}\,(q_1, q_2, q_3) = \vec{r}\,(q_1) + q_2\hat{n}_2\,(q_1) + q_3\hat{n}_3\,(q_1) \tag{2.21}$$

Siendo

$$\begin{aligned} \hat{n}_2 &= \cos\theta\,(q_1)\,\hat{n}\,(q_1) - \sin\theta\,(q_1)\,\hat{b}\,(q_1) \\ \hat{n}_3 &= \sin\theta\,(q_1)\,\hat{n}\,(q_1) + \cos\theta\,(q_1)\,\hat{b}\,(q_1) \end{aligned} \tag{2.22}$$

Con

$$\frac{d\theta}{dq_1} = \tau\,(q_1) \tag{2.23}$$

Donde $\tau(q_1)$ es la torsión de C.

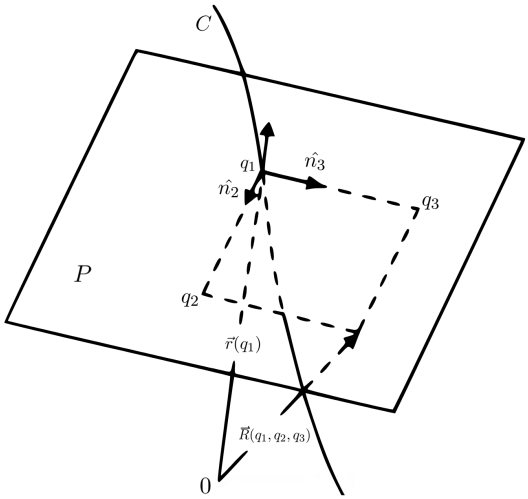


Figura 2.5: Sistema de coordenadas curvilíneo basado en la curva C con las ecuaciones paramétricas $\vec{r} = \vec{r}(q_1)$.

Para simplificar hemos utilizado un sistema de coordenadas cartesianas para cada plano

normal de C. De la ecuaciones (2.21), (2.22)y (2.23) obtenemos:

$$\frac{d\vec{R}}{dq_1} = \left[1 - k\left(q_1\right) f\left(q_1, q_2, q_3\right)\right] \hat{t}\left(q_1\right) \tag{2.24}$$

$$\frac{d\vec{R}}{dq_j} = \hat{n}_j\left(q_1\right) \tag{2.25}$$

Donde

$$f\left(q_1, q_2, q_3\right) = \cos\theta\left(q_1\right) q_2 + \sin\theta\left(q_1\right) q_3 \tag{2.26}$$

Y $k\left(q_1\right) = \left|\frac{d\hat{t}}{dq_1}\right|$ es la curvatura de C en el punto de arco q_1. Debido a que el sistema de coordenadas es ortogonal, es decir, $\left(\frac{\partial\vec{R}}{\partial q_i}\right)\left(\frac{\partial\vec{R}}{\partial q_j}\right) = h_i^2\delta_{ij}$ podemos escribir la fuerza clásica F como el potencial $V(q_1, q_2, q_3)$ tal que

$$\vec{F} = -\operatorname{grad} V = -\sum_{j=1}^{3}\left(\frac{1}{h_j^2}\frac{\partial V}{\partial q_j}\right)\frac{\partial\vec{R}}{\partial q_j} \tag{2.27}$$

Procediendo como en el caso de la restricción superficial, seleccionamos de (2.27), un potencial de enlace $V_\lambda(q_2, q_3)$ independientemente de q_1, con el objetivo de mantener siempre$F = -gradV_\lambda$, en los planos normales a C. Por tanto, la ecuación de Schrödinger se escribe como

$$-\frac{\hbar^2}{2m}\left[\frac{1}{1-kf}\frac{\partial}{\partial q_1}\left(\frac{1}{1-kf}\frac{\partial\psi}{\partial q_1}\right) + \sum_{j=2}^{3}\left(\frac{\partial^2\psi}{\partial q_j^2} + \frac{\partial}{\partial q_j}\ln(1-kf)\frac{\partial\psi}{\partial q_j}\right)\right] + V_\lambda\left(q_2, q_3\right)\psi = i\hbar\frac{\partial\psi}{\partial t} \tag{2.28}$$

El volumen del elemento viene dado por $dV = (1-kf)\times dq_1dq_2dq_3$, lo que nos lleva a la introducción de la nueva función de onda: $\chi\left(q_1, q_2, q_3\right) = (1-kf)^{\frac{1}{2}}\psi\times\left(q_1, q_2, q_3\right)$. Entonces la ecuación (2.28) queda expresada como:

$$\begin{gathered}-\frac{\hbar^2}{2m}\frac{1}{(1-kf)^{\frac{1}{2}}}\frac{\partial}{\partial q_1}\left(\frac{1}{1-kf}\frac{\partial}{\partial q_1}\frac{\chi}{(1-kf)^{\frac{1}{2}}}\right) - \frac{\hbar^2}{8m}\frac{k^2}{(1-kf)^2}\chi - \\ -\frac{\hbar^2}{2m}\left(\frac{\partial^2\chi}{\partial q_2^2} + \frac{\partial^2\chi}{\partial q_3^2}\right) + V_\lambda\left(q_2, q_3\right)\chi = i\hbar\frac{\partial\chi}{\partial t}\end{gathered} \tag{2.29}$$

Suponiendo para V_λ las propiedades esperadas del potencial de una partícula limitada.

$$\lim_{\lambda\to\infty} V_\lambda\left(q_2,q_3\right)=\begin{cases} 0, & q_2^2+q_3^2=0 \\ \infty, & q_2^2+q_3^2\neq 0 \end{cases} \tag{2.30}$$

Se coge directamente $f\to 0$ en (2.29) y se obtiene:

$$-\frac{\hbar^2}{2m}\frac{\partial^2\chi}{\partial q_1^2}-\frac{\hbar^2}{8m}k\left(q_1\right)^2\chi-\frac{\hbar^2}{2m}\left(\frac{\partial^2\chi}{\partial q_2^2}+\frac{\partial^2\chi}{\partial q_3^2}\right)+V_\lambda\left(q_2,q_3\right)\chi=i\hbar\frac{\partial\chi}{\partial t} \tag{2.31}$$

Separamos la ecuación (2.31) debido a $\chi=\chi_t\left(q_1,t\right)\times x_n\left(q_2,q_3,t\right)$. El resultado es:

$$-\frac{\hbar^2}{2m}\left(\frac{\partial^2\chi_n}{\partial q_2^2}+\frac{\partial^2\chi_n}{\partial q_3^2}\right)+V_\lambda\left(q_2,q_3\right)\chi_n=i\hbar\frac{\partial\chi_n}{\partial t} \tag{2.32}$$

$$-\frac{\hbar^2}{2m}\frac{\partial^2\chi_t}{\partial q_1^2}-\frac{\hbar^2}{8m}k\left(q_1\right)^2\chi_t=i\hbar\frac{\partial\chi_t}{\partial t} \tag{2.33}$$

La ecuación (2.33) tiene la misma propiedad que la ecuación (2.15). Aunque todas las curvas son isométricas, cada una tiene su propia mecánica cuántica (dependiendo de la curvatura).

También debe tenerse en cuenta que la ecuación (2.33) no depende del comportamiento detallado del potencial $V_\lambda(q_2,q_3)$ (es decir, sus equipotenciales alrededor de la curva C pueden ser círculos, elipses, rectángulos, etc), siempre que una vez definido en un plano normal, sea conocido en todos los puntos del espacio al dar el mismo potencial a todas las curvas 'paralelas' con los mismos valores de q_2 y q_3.

En cierto sentido se puede decir que en $V_\lambda(q_2,q_3)$ hemos introducido una generalización del potencial bidimensional ordinario (obtenido cuando C es una línea recta).

Como última observación respecto a la posibilidad de unir una partícula a una curva, hay que tener en cuenta las siguientes consideraciones:

1. Utilizar una restricción de superficie del tipo empleado en el apartado anterior.

2. Asumir un potencial de superficie extra para reducir el movimiento a una curva.

No es complicado darse cuenta de que el resultado obtenido de esta manera dependerá, en general, de la superficie intermedia seleccionada en el proceso. La razón es que las normales a esta superficie intermedia no están necesariamente contenidas en un plano normal de la curva. Esto significa que el potencial responsable de la restricción de superficie puede dar lugar a fuerzas con componentes tangenciales que no desaparecen en un entorno de la curva, contradiciendo la definición $V_\lambda(q_2, q_3)$.

También se puede demostrar que se puede obtener el mismo resultado (2.30) si la superficie elegida pertenece a la misma familia:

$$\vec{R}\,(q_1, S) = \vec{r}\,(q_1) + q_2(s)\hat{n}_2\,(q_1) + q_3(s)\hat{n}_3\,(q_1) \tag{2.34}$$

Donde q_1 y q_2 dan la intersección de la superficie con los planos normales de C. Además se observa que, dado q_2 y q_3, (no dependendiente de q_1), la superficie está completamente determinada a partir del conocimiento de su intersección con uno de los planos normales.

2.6. Descripción geométrica de curvas y superficies

En este apartado se introducirá los conceptos matemáticos necesarios.

2.6.1. Ecuaciones de Frenet-Serret en $\mathbb{R}^2$

Sea $\alpha : I \longrightarrow \mathbb{R}^2$ una curva diferenciable parametrizada por la longitud del arco, el vector tangente $\alpha'(s)$ es unitario. Al vector tangente a partir de ahora lo denotaremos como $t(s) = \alpha'(s)$.

Definición 1. *Denotamos el vector normal unitario a la curva α en S al vector $n(s) = Jt(s)$, donde $J : \mathbb{R}^2 \longrightarrow \mathbb{R}^2$ es el giro de 90^o en sentido antihorario y $t(s) = \alpha'(s)$ es el vector tangente.*

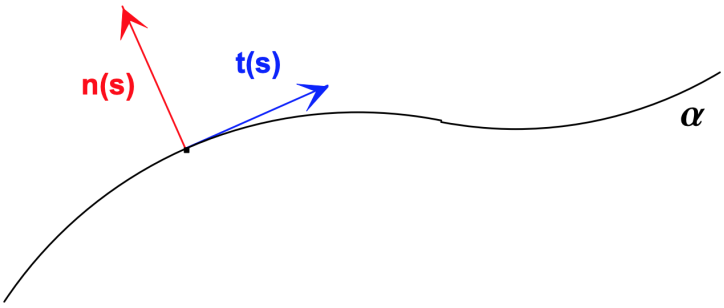


Figura 2.6: Aplicación y superficie.

Los vectores $t(s)$ y $n(s)$ constituyen una base ortonormal de $\mathbb{R}^2$, para cada $s \in I$.

Definición 2. *Sea $\alpha : I \longrightarrow \mathbb{R}^2$. Sean $t(s)$ y (s) su vector tangente unitario y vector normal unitario respectivamente, $\forall s \in I$. La base $\{t(s), n(s)\}$ se llama diedro (orientado) de Frenet.*

Definición 3. *Sea $\alpha : I \longrightarrow \mathbb{R}^2$. Se define a $k : I \longrightarrow \mathbb{R}$ como curvatura de la curva plana α. Observemos que la curvatura también se puede expresar como*

$$k(s) = \langle \alpha''(s), J\alpha'(s) \rangle$$

Definición 4. *Sea $\alpha : I \longrightarrow \mathbb{R}^2$ una curva. Sean $t(s)$ y (s) su vector tangente unitario y vector normal unitario respectivamente, $\forall s \in I$. Las fórmulas de Frenet en $\mathbb{R}^2$ se definen como:*

$$\begin{cases} \mathbf{t}'(s) = k(s)\mathbf{n}(s) \\ \mathbf{n}'(s) = -k(s)\mathbf{t}(s) \end{cases} \tag{2.35}$$

2.6.2. Superficies

Definición 5. *Una superficie parametrizada S en el espacio es una aplicación continua*

$$X : D \subset \mathbb{R}^2 \longrightarrow \mathbb{R}^3$$
$$X(u, v) = (x(u, v), y(u, v), z(u, v))$$

donde x, y, z son funciones de clase C^2.

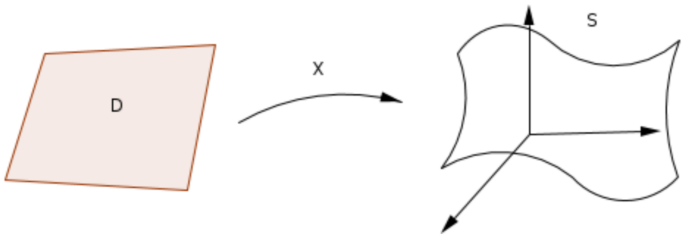


Figura 2.7: Aplicación y superficie

Denotamos por S a la gráfica de la superficie:

$$S = \{(x, y, z) \in \mathbb{R}^3 | (x, y, z) = (x(u, v), y(u, v), z(u, v)), (u, v) \in D\}$$

Definición 6. *Sea* $\alpha(s) = (x(s), y(s))$ *una curva en el plano* $z = 0$ *en* $\mathbb{R}^3$. *Llamamos I al dominio donde s está definido y suponemos que* $y(s) > 0$ *en I. Definimos una superficie de revolución S en* $\mathbb{R}^3$, *cuya curva generatriz es* α *y cuyo eje de rotación es el x como*

$$S = (x(s), y(s)cos\theta, y(s)sen\theta) \in \mathbb{R}^3 | s \in I, 0 \leq \theta \leq 2\pi.$$

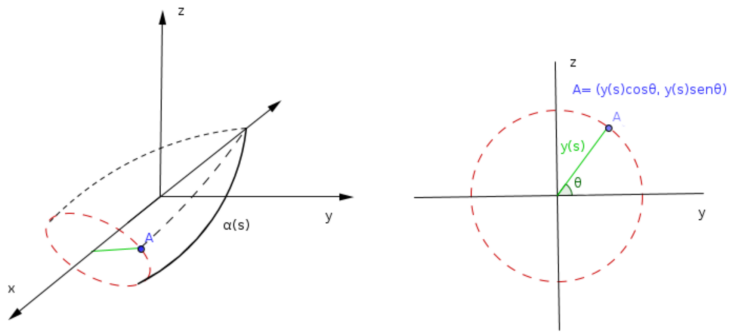


Figura 2.8: Superficie de revolución generada por α (derecha) y vista y-z de la misma (izquierda).

2.6.3. Curvaturas de una superficie

Sea $\alpha(s) = (x(s), y(s))$ una curva parametrizada por longitud de arco su curvatura es $||\alpha''(s)||$. Una vez que hemos introducido el concepto de curvatura para una curva plana vamos a ver qué se entiende por la curvatura de una superficie.

Definición 7. *La curvatura normal de una superficie en un punto P y en la dirección de $\vec{v}$ es la curvatura de la correspondiente sección normal.*

Definición 8. *Las direcciones principales de la superficie en el punto P son las direcciones para las cuales se obtienen las curvaturas principales:*

$k_1(P)$ = valor mínimo de la curvatura normal

$k_2(P)$ = valor máximo de la curvatura normal

La curvatura media de una superficie es entonces la media de estas curvaturas principales. Existen casos extremos como el plano y la esfera en los que las secciones normales, para todos los planos del haz, son las mismas. En el plano todas son rectas luego tienen curvatura 0 y en la esfera todas son circunferencias del mismo radio r luego tienen curvatura 1, pero esto no es r lo frecuente.

Definición 9. *Dadas las curvaturas principales, se define curvatura de Gauss K de una superficie S en un punto $P \in S$ al producto de las curvaturas principales.*

$$K = k_1 \cdot k_2 = \frac{eg - f^2}{EG - F^2} \tag{2.36}$$

Definición 10. *Se denomina curvatura media H de una superficie S en un punto $P \in S$ a la media aritmética de las curvaturas:*

$$H = \frac{k_1 + k_2}{2} = \frac{1}{2}\frac{eG - 2fF + gE}{EG - F^2} \tag{2.37}$$

2.6.4. Formas fundamentales

Definición 11. *Llamamos primera forma fundamental de S en P a la forma bilineal simétrica definida positiva IP asociada al producto escalar inducido en el plano tangente a S en P por el producto escalar en $\mathbb{R}^3$. Dada una superficie regular S de $\mathbb{R}^3$ definida por $X(u,v) : U \subset \mathbb{R}^2 \to S \subset \mathbb{R}^3$ y un vector $\vec{w} \in T_p(S)$, se define la primera forma fundamental como la aplicación*

$$I_p(w) : T_p(S) \to \mathbb{R} \text{ con } I_p(w) = \langle w, w\rangle_p = |w|^2 \geq 0$$

Cmo el vector $\vec{w}$ es tangente a S en el punto p, se puede escribir como $\alpha(t) = X(u(t), v(t))$, siendo $\alpha(0) = p$ y $\alpha'(0) = \vec{w}$. Aplicando la primera forma fundamental obtenemos:

$$\begin{aligned} I_p\left(\alpha'(0)\right) &= \left\langle \alpha'(0), \alpha'(0)\right\rangle_p = \left\langle X_u u' + X_v v', X_u u' + X_v v'\right\rangle_p = \\ &= \langle X_u, X_u\rangle_p \left(u'\right)^2 + 2\langle X_u, X_v\rangle_p u'v' + \langle X_v, X_v\rangle_p \left(v'\right)^2 \end{aligned}$$

Por tanto, los **coeficientes de la primera forma fundamental** son:

$$\boxed{E = \langle X_u, X_u\rangle_p \ ; \ F = \langle X_u, X_v\rangle_p \ ; \ G = \langle X_v, X_v\rangle_p \ ;}$$

Definición 12. *Sean $\vec{v}, \vec{w} \in T_p(S)$ expresados en función de la base del plano tangente en* p, $\{X_u, X_v\}$*, entonces la forma matricial el producto escalar entre dos vectores se expresa como*

$$\langle \vec{v}, \vec{w}\rangle = \vec{v}^t \begin{pmatrix} E & F \\ F & G \end{pmatrix} \vec{w}$$

Definición 13. *Dada una superficie regular orientable S definida por $X(u,v) : U \subset \mathbb{R}^2 \to S \subset \mathbb{R}^3$ y un vector $\vec{v} \in T_p(S)$, se define la segunda forma fundamental como la aplicación*

$$\Pi_p : T_p(S) \to \mathbb{R} \text{ con } \Pi_p(v) = -\langle dN_p(v), v\rangle$$

como el vector $\vec{v}$ es tangente a S en el punto p, tenemos que puede escribirse como $\alpha(t) = X(u(t), v(t)), \text{con}\, \alpha(0) = p$ y $\alpha'(0) = \vec{v}$. Aplicando la segunda forma fundamental obtenemos:

$$\begin{aligned}\Pi_p\left(\alpha'(0)\right) &= -\left\langle dN\left(\alpha'(0)\right), \alpha'(0)\right\rangle_p \\ &= -\left\langle N_u u' + N_v v', X_u u' + X_v v'\right\rangle_p \\ &= -\left\langle N_u, X_u\right\rangle_p \left(u'\right)^2 - \left\langle N_u, X_v\right\rangle_p u'v' - \left\langle N_v, X_u\right\rangle_p u'v' - \left\langle N_v, X_v\right\rangle_p \left(v'\right)^2\end{aligned}$$

Puesto que X_u, X_v son perpendiculares al vector normal N, sabemos que $\langle N, X_u\rangle = 0 = \langle N, X_v\rangle$. A continuación, derivamos respecto de u y obtenemos

$$\langle N_u, X_u\rangle + \langle N, X_{uu}\rangle = 0 = \langle N_u, X_v\rangle + \langle N, X_{vu}\rangle$$

Derivando respecto de v obtenemos

$$\langle N_v, X_u\rangle + \langle N, X_{uv}\rangle = 0 = \langle N_v, X_v\rangle + \langle N, X_{vv}\rangle$$

Por tanto llegamos a:

$$\begin{cases} -\langle N_u, X_u\rangle = \langle N, X_{uu}\rangle \\ -\langle N_u, X_v\rangle = \langle N, X_{vu}\rangle \\ -\langle N_v, X_u\rangle = \langle N, X_{uv}\rangle \\ -\langle N_v, X_v\rangle = \langle N, X_{vv}\rangle \end{cases} \longrightarrow \text{ Puesto que } X_{vu} = X_{uv} \longrightarrow \langle N_u, X_v\rangle = \langle N_v, X_u\rangle$$

Retomando el anterior desarrollo y aplicando lo visto anteriormente obtenemos:

$$\Pi_p\left(\alpha'(0)\right) = \langle N, X_{uu}\rangle \left(u'\right)^2 - 2\langle N, X_{vu}\rangle u'v' - \langle N, X_{vv}\rangle \left(v'\right)^2$$

Por tanto, los **coeficientes de la segunda forma fundamental** son:

$$\boxed{e = \langle N, X_{uu}\rangle \ ; \ f = \langle N, X_{uv}\rangle = \langle N, X_{vu}\rangle \ ; \ g = \langle N, X_{vv}\rangle}$$

2.6.5. Laplaciano de una superficie

Definición 14. *Sea S' una superficie de $\mathbb{R}^3$ y sea $\phi : S \to \mathbb{R}^3$ una función diferenciable. Se define el Laplaciano de ϕ (ver [15]) como*

$$\begin{aligned}\Delta^{S'}\phi &:= \left(\frac{1}{\sqrt{g}}\right)\cdot\sum_{k=1}^{2}\left(\partial x_1\left(g^{1k}\cdot\sqrt{g}\cdot\partial x_k\phi\right)+\partial x_2\left(g^{2k}\cdot\sqrt{g}\cdot\partial x_k\phi\right)\right)= \\ &= \frac{1}{\sqrt{g}}\cdot\left(\partial x_1\left(g^{11}\cdot\sqrt{g}\cdot\partial x_1\phi\right)+\partial x_1\left(g^{12}\cdot\sqrt{g}\cdot\partial_{x_2}\phi\right)+\right. \\ &+ \left.\left(\partial x_2\left(g^{21}\cdot\sqrt{g}\cdot\partial x_2\phi\right)+\partial x_1\left(g^{22}\cdot\sqrt{g}\cdot\partial_{x_2}\phi\right)\right.\right.\end{aligned}$$

Observación 1. *En cuento a la notación seguiremos las siguientes consideraciones:*

1. $\sqrt{g} := \sqrt{\det g}$
2. $g^{jk} := \left[g^{-1}\right]_{jk}$
3. $g = \begin{pmatrix} E & F \\ F & G \end{pmatrix}$ *y* $det(g) = EG - F^2$
4. $g^{-1} = \frac{1}{EG-F^2}\begin{pmatrix} G & -F \\ -F & E \end{pmatrix}$

Proposición 1. *Sea $X : U \subset \mathbb{R}^2 \to \mathbb{R}^3$ una superficie parametrizada regular, entonces se puede expresar el laplaciano de la superficie en función de los coeficientes de primera y segunda generación como:*

$$\begin{aligned}-\Delta\phi = \frac{-1}{\sqrt{EG-F^2}}\cdot\Bigg[&\frac{G}{\sqrt{EG-F^2}}\cdot\frac{\partial^2\phi}{\partial^2 u}+\frac{E}{\sqrt{EG-F^2}}\cdot\frac{\partial^2\phi}{\partial^2 v}-\frac{2}{\sqrt{EG-F^2}}\cdot\frac{\partial^2\phi}{\partial u\partial v} \\ &+\frac{\left(2(EG-F^2)(\frac{\partial G}{\partial u}-\frac{\partial F}{\partial v})-G(\frac{\partial E}{\partial u}\cdot G+E\frac{\partial G}{\partial u}-2F)+F(\frac{\partial E}{\partial v}\cdot G+E\frac{\partial G}{\partial v}-2F)\right)}{2(EG-F^2)^{\frac{3}{2}}}\cdot\frac{\partial\phi}{\partial u} \\ &+\frac{\left(2(EG-F^2)(\frac{\partial F}{\partial u}-\frac{\partial E}{\partial v})-F(\frac{\partial E}{\partial u}\cdot G+E\frac{\partial G}{\partial v}-2F)+F(\frac{\partial E}{\partial v}\cdot G+E\frac{\partial G}{\partial v}-2F)\right)}{2(EG-F^2)^{\frac{3}{2}}}\cdot\frac{\partial\phi}{\partial v}\Bigg]\end{aligned}$$

Demostración. La demostración consiste en desarrollar el Laplaciano.

NOTA: x_1 y x_2 son parámetros. En la demostración se utiliza u y v respectivamente.

$$\Delta\phi = \frac{1}{\sqrt{g}} \cdot \left[\frac{\partial}{\partial u}\left(g^{11} \cdot \sqrt{g} \cdot \frac{\partial \phi}{\partial u} \right) + \frac{\partial}{\partial u}\left(g^{12} \cdot \sqrt{g} \cdot \frac{\partial \phi}{\partial v} \right) + \right.$$
$$\left. + \frac{\partial}{\partial v}\left(g^{21} \cdot \sqrt{g} \cdot \frac{\partial \phi}{\partial u} \right) + \frac{\partial}{\partial v}\left(g^{22} \cdot \sqrt{g} \cdot \frac{\partial \phi}{\partial v} \right) \right] =$$

$$= \frac{1}{\sqrt{EG - F^2}} \cdot \left[\frac{\partial}{\partial u}\left(\frac{G \cdot g^{\frac{1}{2}}}{g} \cdot \frac{\partial \phi}{\partial u} \right) + \frac{\partial}{\partial u}\left(\frac{-F \cdot g^{\frac{1}{2}}}{g} \cdot \frac{\partial \phi}{\partial v} \right) + \right.$$
$$\left. + \frac{\partial}{\partial v}\left(\frac{-F \cdot g^{\frac{1}{2}}}{g} \cdot \frac{\partial \phi}{\partial u} \right) + \frac{\partial}{\partial v}\left(\frac{E \cdot g^{\frac{1}{2}}}{g} \cdot \frac{\partial \phi}{\partial v} \right) \right] =$$

$$= \frac{1}{\sqrt{EG - F^2}} \cdot \left[\frac{\partial}{\partial u}\left(\frac{G}{\sqrt{g}} \cdot \frac{\partial \phi}{\partial u} \right) + \frac{\partial}{\partial u}\left(\frac{-F}{\sqrt{g}} \cdot \frac{\partial \phi}{\partial v} \right) + \right.$$
$$\left. \frac{\partial}{\partial v}\left(\frac{-F}{\sqrt{g}} \cdot \frac{\partial \phi}{\partial u} \right) + \frac{\partial}{\partial v}\left(\frac{E}{\sqrt{g}} \cdot \frac{\partial \phi}{\partial v} \right) \right] =$$

$$= \frac{1}{\sqrt{EG - F^2}} \cdot \left[\underbrace{\frac{\partial}{\partial u}\left(\frac{G}{\sqrt{EG - F^2}} \cdot \frac{\partial \phi}{\partial u} \right)}_{(1)} + \underbrace{\frac{\partial}{\partial u}\left(\frac{-F}{\sqrt{EG - F^2}} \cdot \frac{\partial \phi}{\partial v} \right)}_{(2)} + \right.$$
$$\left. + \underbrace{\frac{\partial}{\partial v}\left(\frac{-F}{\sqrt{EG - F^2}} \cdot \frac{\partial \phi}{\partial u} \right)}_{(3)} + \underbrace{\frac{\partial}{\partial v}\left(\frac{E}{\sqrt{EG - F^2}} \cdot \frac{\partial \phi}{\partial v} \right)}_{(4)} \right] =$$

Hacemos las derivadas por separado:

- Derivada (1):

$$\frac{\partial}{\partial u}\left(\frac{G}{\sqrt{EG-F^2}}\cdot\frac{\partial\phi}{\partial u}\right)=$$

$$=\frac{\frac{\partial G}{\partial u}\sqrt{EG-F^2}-G\cdot\frac{\frac{\partial E}{\partial u}G-E\frac{\partial G}{\partial u}-2F}{2\sqrt{EG-F^2}}}{EG-F^2}\cdot\frac{\partial\phi}{\partial u}+\frac{G}{\sqrt{EG-F^2}}\cdot\frac{\partial^2\phi}{\partial^2 u}$$

$$=\frac{-2\frac{\partial G}{\partial u}(EG-F^2)+G(\frac{\partial E}{\partial u}G-E\frac{\partial G}{\partial u}-2F)}{2(EG-F^2)^{\frac{3}{2}}}\cdot\frac{\partial\phi}{\partial u}+\frac{G}{\sqrt{EG-F^2}}\cdot\frac{\partial^2\phi}{\partial^2 u}$$

- Derivada (2):

$$\frac{\partial}{\partial u}\left(\frac{-F}{\sqrt{EG-F^2}}\cdot\frac{\partial\phi}{\partial v}\right)=$$

$$=\frac{-\frac{\partial F}{\partial u}\sqrt{EG-F^2}+F\cdot\frac{\frac{\partial E}{\partial u}G-E\frac{\partial G}{\partial u}-2F}{2\sqrt{EG-F^2}}}{EG-F^2}\frac{\partial\phi}{\partial v}+\frac{-F}{\sqrt{EG-F^2}}\cdot\frac{\partial^2\phi}{\partial v\partial u}$$

$$=\frac{-2\frac{\partial F}{\partial u}(EG-F^2)+F(\frac{\partial E}{\partial u}G-E\frac{\partial G}{\partial u}-2F)}{2(EG-F^2)^{\frac{3}{2}}}\cdot\frac{\partial\phi}{\partial v}+\frac{-F}{\sqrt{EG-F^2}}\cdot\frac{\partial^2\phi}{\partial v\partial u}$$

- Derivada (3):

$$\frac{\partial}{\partial v}\left(\frac{-F}{\sqrt{EG-F^2}}\cdot\frac{\partial\phi}{\partial u}\right)=$$

$$=\frac{-\frac{\partial F}{\partial v}\sqrt{EG-F^2}+F\cdot\frac{\frac{\partial E}{\partial v}G-E\frac{\partial G}{\partial v}-2F}{2\sqrt{EG-F^2}}}{EG-F^2}\frac{\partial\phi}{\partial u}+\frac{-F}{\sqrt{EG-F^2}}\cdot\frac{\partial^2\phi}{\partial v\partial u}$$

$$=\frac{-2\frac{\partial F}{\partial v}(EG-F^2)+F(\frac{\partial E}{\partial v}G-E\frac{\partial G}{\partial v}-2F)}{2(EG-F^2)^{\frac{3}{2}}}\cdot\frac{\partial\phi}{\partial u}+\frac{-F}{\sqrt{EG-F^2}}\cdot\frac{\partial^2\phi}{\partial v\partial u}$$

- Derivada (4):

$$\frac{\partial}{\partial v}\left(\frac{E}{\sqrt{EG-F^2}}\cdot\frac{\partial\phi}{\partial v}\right)=$$

$$=\frac{\frac{\partial G}{\partial v}\sqrt{EG-F^2}-G\cdot\frac{\frac{\partial E}{\partial u}G-E\frac{\partial G}{\partial v}-2F}{2\sqrt{EG-F^2}}}{EG-F^2}\cdot\frac{\partial\phi}{\partial v}+\frac{G}{\sqrt{EG-F^2}}\cdot\frac{\partial^2\phi}{\partial^2 v}$$

$$=\frac{-2\frac{\partial G}{\partial v}(EG-F^2)+G(\frac{\partial E}{\partial v}G-E\frac{\partial G}{\partial v}-2F)}{2(EG-F^2)^{\frac{3}{2}}}\cdot\frac{\partial\phi}{\partial v}+\frac{G}{\sqrt{EG-F^2}}\cdot\frac{\partial^2\phi}{\partial^2 v}$$

Agrupando debidamente, obtenemos el resultado esperado.

□

2.7. Desarrollo del operador Hamiltoniano ($\hat{H}$) de la formula de Schrödinger

Proposición 2. *Sea $\phi X(u) \to \mathbb{R}^3$ una función diferenciable. Sea $\hat{H}\phi = -\Delta^S\phi - (K_G - H^2)\phi$ el operador Hamiltoniano de la fórmula de Schrödinger. Se puede expresar $\hat{H}$ como:*

$$\hat{H}\phi = \frac{-1}{\sqrt{EG-F^2}} \cdot \left[\frac{G}{\sqrt{EG-F^2}} \cdot \frac{\partial^2\phi}{\partial^2 u} + \frac{E}{\sqrt{EG-F^2}} \cdot \frac{\partial^2\phi}{\partial^2 v} - \frac{2F}{\sqrt{EG-F^2}} \cdot \frac{\partial^2\phi}{\partial u \partial v} \right.$$

$$+ \frac{\left(2(EG-F^2)(\frac{\partial G}{\partial u} - \frac{\partial F}{\partial v}) - G(\frac{\partial E}{\partial u} \cdot G + E\frac{\partial G}{\partial u} - 2F) + F(\frac{\partial E}{\partial v} \cdot G + E\frac{\partial G}{\partial v} - 2F)\right)}{2(EG-F^2)^{\frac{3}{2}}} \cdot \frac{\partial\phi}{\partial u}$$

$$\left. + \frac{\left(2(EG-F^2)(\frac{\partial F}{\partial u} - \frac{\partial E}{\partial v}) - F(\frac{\partial E}{\partial u} \cdot G + E\frac{\partial G}{\partial v} - 2F) + F(\frac{\partial E}{\partial v} \cdot G + E\frac{\partial G}{\partial v} - 2F)\right)}{2(EG-F^2)^{\frac{3}{2}}} \cdot \frac{\partial\phi}{\partial v} \right] +$$

$$+ \frac{-6egEG + 8F^2f^2 - 4f^2EG - 4egF^2 - e^2G^2 - g^2E^2 + 4efGF + 4gfEF}{4(EG-F^2)^2}$$

Demostración. En primer lugar, expresamos $-(k_G - H^2)$ en función de los coeficientes de la primera y segunda forma fundamental.

$$-\left(\frac{eg-f^2}{EG-F^2} + \frac{(eG-2fF+gE)^2}{4(EG-F^2)^2}\right) =$$

$$= \frac{-6egEG + 8F^2f^2 - 4f^2EG - 4egF^2 - e^2G^2 - g^2E^2 + 4efGF + 4gfEF}{4(EG-F^2)^2}$$

Por otro lado, utilizamos la proposición anterior y con ello obtenememos el resultado al que queríamos llegar.

□

En conclusión, se ha obtenido que $\hat{H}\phi = -\Delta^S\phi - (K_G - H^2)\phi$ también se puede expresar en función de los coeficientes de la primera y la segunda forma fundamental.

2.7.1. Ejemplo

Sea la curva $\alpha : I \subset \mathbb{R} \to \mathbb{R}^2, \quad u \mapsto (r(u), u)$, siendo $h(u) = u$

Las derivadas de la curva son:

- $\alpha(u)' = (r', 1)$ siendo $h' = 1$
- $\alpha(u)'' = (r'', 0)$ siendo $h'' = 0$

La primera forma fundamental es:

- $E = r'^2 + 1$
- $F = 0$
- $G = r^2$

La segunda forma fundamental es:

- $e = \dfrac{-r''}{\sqrt{r'^2+1}}$
- $f = 0$
- $g = \dfrac{r}{\sqrt{r'^2+1}}$

El objetivo es mostrar cómo queda $\hat{H}\phi = -\Delta^S\phi - (K_G - H^2)\phi$. Para obtenerlo solo tenemos que substituir los coeficientes de la primera y la segunda forma fundamental en la fórmula simplificada que ya hemos obtenido de $E\phi$ de este tipo de superficies.

$$\begin{aligned}
\hat{H}\phi &= -\frac{\phi_{uu}}{r'^2+1} - \frac{\phi_{vv}}{r^2} - \frac{1}{r'^2+1}\left(\frac{\partial}{\partial u}\log\frac{(r^2)^{\frac{1}{2}}}{r'^2+1}\right)\phi_u + \frac{\frac{r''^2r^4}{r'^2+1} + \frac{r^2}{r'^2+1}(r'^2+1)^2 - 6r^3r''(r'^2+1)}{4r^4(r'^2+1)^2} \\
&= -\frac{\phi_{uu}}{r'^2+1} - \frac{\phi_{vv}}{r^2} - \frac{1}{r'^2+1}\left(\frac{\partial}{\partial u}\log\frac{r}{r'^2+1}\right)\phi_u + \frac{\frac{r''^2r^4}{r'^2+1} + \frac{r^2}{r'^2+1}(r'^2+1)^2 - \frac{6r^3r''(r'^2+1)}{(r'^2+1)}}{4r^4(r'^2+1)^2} \\
&= -\frac{\phi_{uu}}{r'^2+1} - \frac{\phi_{vv}}{r^2} - \frac{1}{r'^2+1}\left(\frac{\partial}{\partial u}\log\frac{r}{r'^2+1}\right)\phi_u + \frac{r''^2r^4 + r^2(r'^2+1)^2 - 6r^3r''(r'^2+1)}{4r^4(r'^2+1)^3} \\
&= -\frac{\phi_{uu}}{r'^2+1} - \frac{\phi_{vv}}{r^2} - \frac{1}{r'^2+1}\left(\frac{\partial}{\partial u}\log\frac{r}{r'^2+1}\right)\phi_u + \frac{r''^2}{4(r'^2+1)^2} + \frac{r}{4r^2(r'^2+1)} - \frac{3r''}{2r(r'^2+1)^2}
\end{aligned}$$

2.8. Superficies de revolución

El objetivo de este apartado es expresar el operador Hamiltoniano ($\hat{H}$) de la fórmula de Schrödinger para este tipo de superficies en función de los coeficientes de primera y segunda generación.

2.8.1. Teorema principal

Teorema 1. *Sea $\alpha : I \subset \mathbb{R} \to \mathbb{R}^2$ una curva suave parametrizada por longitud de arco. Sea*

$$X : I \times (0, 2\pi) \to \mathbb{R}^3, \quad (u, v) \mapsto X(u, v) := (\alpha_1(u)\cos(v), \alpha_1(u)\sin(v), \alpha_2(u))$$

la superficie de revolución que tiene por perfil la curva α. Sea $\phi : X(I \times (0, 2\pi)) \to \mathbb{R}$ una función suave definida sobre la superficie. Entonces,

$$\hat{H}\phi = -\phi_{uu} - \frac{\phi_{vv}}{\alpha_1^2(u)} - \frac{\alpha_1'(u)}{\alpha_1(u)}\phi_u + \frac{1}{4}\left(k(u) - \frac{\alpha_2'(u)}{\alpha_1(u)}\right)^2 \phi$$

siendo $k : I \to \mathbb{R}$ la función curvatura de α.

Demostración. En primer lugar, calculamos las derivadas parciales que nos proporcionan información sobre la superficie X:

- $\frac{\partial X}{\partial u} = (\alpha_1' cos(v), \alpha_1' sin(v), \alpha_2')$
- $\frac{\partial X}{\partial v} = (-\alpha_1 sin(v), \alpha_1 cos(v), 0)$
- $\frac{\partial^2 X}{\partial^2 u} = (\alpha_1'' cos(v), \alpha_1'' sin(v), \alpha_2'')$
- $\frac{\partial^2 X}{\partial^2 u} = (-\alpha_1' cos(v), -\alpha_1' sin(v), 0)$
- $\frac{\partial^2 X}{\partial^2 v} = (-\alpha_1' sin(v), \alpha_1' cos(v), 0)$

Nota: En cuestiones de notación debemos tener en cuenta que como las funciones r y h son funciones que solo dependen del u, $\frac{\partial \alpha_1}{\partial u} = \alpha_1'$ y $\frac{\partial \alpha_2}{\partial u} = \alpha_2'$

Mediante la propia definición de los coeficientes de la primera forma fundamental (**??**) obtenemos:

- $E = (\alpha_1')^2 + (\alpha_2')^2$
- $F = 0$
- $G = \alpha_1^2$

A continuación, calculamos el vector normal:

$$N = \frac{1}{\left\| \frac{\partial \phi}{\partial u} \times \frac{\partial \phi}{\partial v} \right\|} \cdot \left(\frac{\partial \phi}{\partial u} \times \frac{\partial \phi}{\partial v} \right) = \frac{1}{\sqrt{(\alpha_1')^2 + (\alpha_2')^2}} \cdot (-\alpha_2' cos(v, -\alpha_2' sin(v), \alpha_1')$$

Por tanto, utilizando definición de los coeficientes de la segunda forma fundamental(**??**) obtenemos:

- $e = \dfrac{-\alpha_1''\alpha_2' + \alpha_1'\alpha_2''}{\sqrt{(\alpha_1')^2 + (\alpha_2')^2}}$
- $f = 0$
- $g = \dfrac{rh'}{\sqrt{(\alpha_1')^2 + (\alpha_2')^2}}$

Las siguientes expresiones se pueden simplificar más aun. Pero es necesario tener en cuenta que:

$$T(s) = (\alpha_1'(s), \alpha_2'(s)), \quad N(s) = JT(s) = (-\alpha_2'(s), \alpha_1'(s))$$

Por las Ecuaciones de Frenet en $\mathbb{R}^2$ (2.35)

$$T'(s) = k(s)N(s), \quad (\alpha_1''(s), \alpha_2''(s)) = k(s)(-\alpha_2'(s), \alpha_1'(s))$$

Como $\|T'(s)\|^2 = 1 \longrightarrow (\alpha_1'(s))^2 + (\alpha_1'(s))^2 = 1$ i

$$-\alpha_1''(s)\alpha_2'(s) + \alpha_2''(s)\alpha_1'(s) = k(s)\left((\alpha_1'(s))^2 + (\alpha_1'(s))^2\right) = k(s)$$

Los coeficientes de la**primera forma fundamental** son:

- $E = (\alpha_1')^2 + (\alpha_2')^2 = 1$
- $F = 0$

- $G = \alpha_1^2$

Los coeficientes de la**segunda forma fundamental** son:

- $e = \dfrac{-\alpha_1''\alpha_2' + \alpha_1'\alpha_2''}{\sqrt{(\alpha_1')^2 + (\alpha_2')^2}} = -\alpha_1''(s)\alpha_2'(s) + \alpha_2''(s)\alpha_1'(s) = k(s)$
- $f = 0$
- $g = \dfrac{\alpha_1\alpha_2'}{\sqrt{(\alpha_1')^2 + (\alpha_2')^2}} = \alpha_1\alpha_2'$

El siguiente paso de la demostración consistirá en ver cómo expresamos el Operador Hamiltoniano ($\hat{H}$) de la fórmula de Schrödinger en superficies de revolución. Aplicando la (proposición 2) obtenemos el Hamiltoniano.

$$\hat{H}\phi = -\Delta^S\phi - (K_G - H^2)\phi$$

expresado en función de los coeficientes de primera y segunda generación.

Por una parte, obtenemos el Laplaciano $-\Delta^S\phi$:

$$\begin{aligned}
\Delta\phi &= \frac{1}{\sqrt{EG}} \cdot \left[\frac{G}{\sqrt{EG}} \cdot \frac{\partial^2\phi}{\partial^2 u} + \frac{E}{\sqrt{EG}} \cdot \phi_{vv} + \left(\frac{2EG \cdot G_u - G^2E_u - GEG_u}{2(EG)^{3/2}} \right) \phi_u \right] = \\
&= \frac{\phi_{uu}}{E} + \frac{\phi_{vv}}{G} + \left(\frac{G_u}{2EG} - \frac{E_u}{E^2} \right) \phi_u \\
&= \frac{\phi_{uu}}{E} + \frac{\phi_{vv}}{G} + \frac{1}{E} \left(\frac{1}{2} \cdot \frac{G_u}{G} - \frac{E_u}{E} \right) \phi_u \\
&= \frac{\phi_w}{E} + \frac{\phi_{vv}}{G} + \frac{1}{E} \left(\frac{1}{2} \frac{\partial}{\partial u} \log G - \frac{\partial}{\partial u} \log E \right) \phi_u \\
&= \frac{\phi_{uu}}{E} + \frac{\phi_{vv}}{G} + \frac{1}{E} \left(\frac{\partial}{\partial u} \log \frac{G^{\frac{1}{2}}}{E} \right) \phi_u \\
&= \frac{\phi_w}{E} + \frac{\phi_{vv}}{G} + \frac{1}{E} \left(\frac{1}{2} \frac{\partial}{\partial u} \log G - \frac{\partial}{\partial u} \log E \right) \phi_u \\
&= \frac{\phi_{uu}}{E} + \frac{\phi_{vv}}{G} + \frac{1}{E} \left(\frac{\partial}{\partial u} \log \frac{G^{\frac{1}{2}}}{E} \right) \phi_u
\end{aligned}$$

Por otra parte, obtenemos la segunda parte de la expresión: $-(K_G - H^2)\phi$, esta parte ya la teníamos calculada, solo tenemos que sustituir los coeficientes que se anulan y ver como queda la expresión :

$$\begin{aligned} &= \frac{-6egEG + 8F^2f^2 - 4f^2EG - 4egF^2 - e^2G^2 - g^2E^2 + 4efGF + 4gfEF}{4(EG - F^2)^2} \\ &= \frac{-e^2G^2 + g^2E^2 + 6geGE}{4(EG)^2} \end{aligned}$$

Entonces,

$$\hat{H}\phi = -\frac{\phi_{uu}}{E} - \frac{\phi_{vv}}{G} - \frac{1}{E}\left(\frac{\partial}{\partial u}\log\frac{G^{\frac{1}{2}}}{E}\right)\phi_u + \frac{-e^2G^2 + g^2E^2 + 6geGE}{4(EG)^2} \tag{2.38}$$

Substituyendo los coeficientes de la primera y la segunda forma fundamental en la anterior ecuación [2.38] obtenemos

$$\hat{H}\phi = -\phi_{uu} - \frac{\phi_{vv}}{\alpha_1^2(u)} - \frac{\alpha_1'(u)}{\alpha_1(u)}\phi_u + \underbrace{\frac{1}{4}\left(k(u) - \frac{\alpha_2'(u)}{\alpha_1(u)}\right)^2\phi}_{V(r)}$$

Los cálculos de V(r) para obtener la expresión final han sido:

$$\begin{aligned} V(r) &= \frac{2EgeG - E^2g^2 - G^2e^2}{4E^2G^2} \\ &= \frac{2\alpha_1^3\alpha_2'k - {\alpha_1}^2{\alpha_2'}^2 - \alpha_1^4k^2}{4\alpha_1^4} \\ &= \frac{1}{4}(\frac{2\alpha_2'k}{\alpha_1} - \frac{{\alpha_2'}^2}{{\alpha_1}^2} - k^2) \\ &= +\frac{1}{4}\left(k(s) - \frac{\alpha_2'}{\alpha_1}\right)^2\phi \end{aligned}$$

□

2.8.2. Programa auxiliar

A continuación, se presentará el programa desarrollado para representar y dar mayor claridad al teorema principal. En primer lugar, la interfaz de entrada es la siguiente:

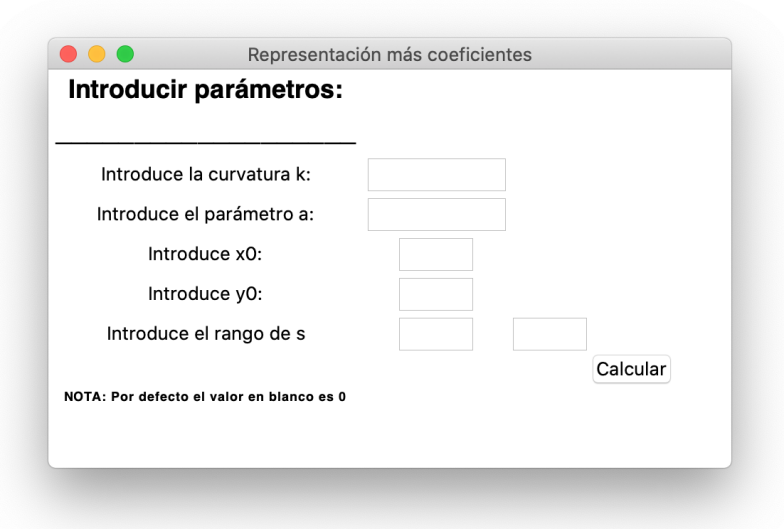


Figura 2.9: Interfaz de entrada

La interfaz de entrada nos proporciona la facilidad de poder rellenarlo con los parámetros que se desee. Los parámetros son:

- Curvatura de la curva que crea la superficie de revolución. Campo obligatorio.
- Valor a.
- Valor del x_o. Parámetro opcional, sino se pone se rellena por defecto con un 0.
- Valor del y_o. Parámetro opcional, sino se pone se rellena por defecto con un 0.
- Rango que va a tomar el parámetro s. Campo obligatorio.

Ejemplo1

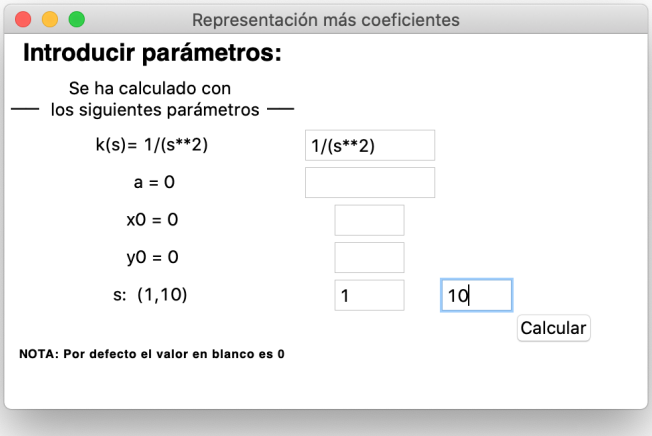


Figura 2.10: Parámetros de entrada ejemplo 1

El primer resultado que obtenemos al ejecutar es la superficie de revolución.

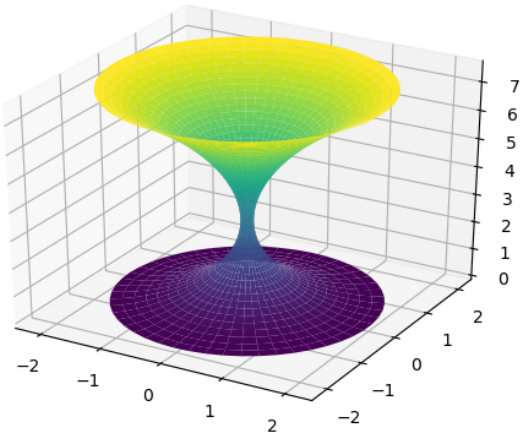
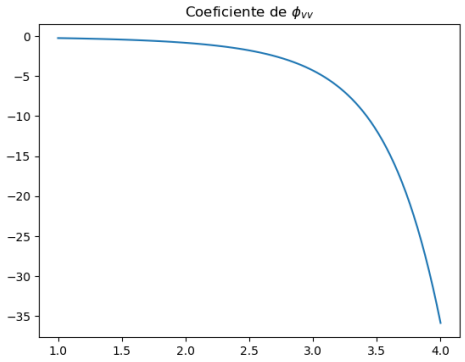
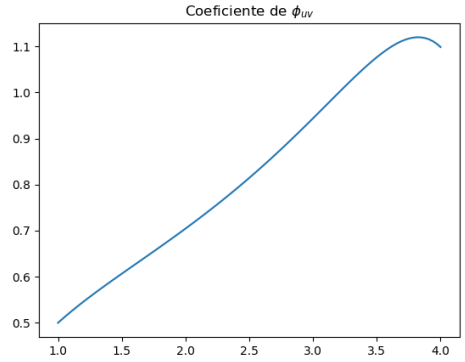
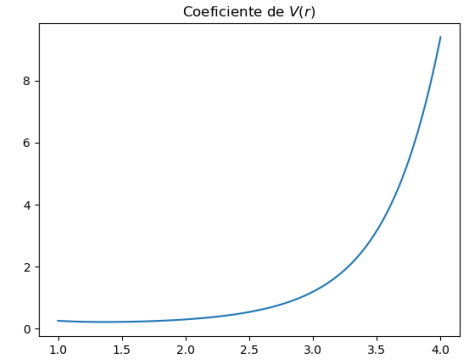


Figura 2.11: Superficie de revolución con curvatura $1/s$

Después obtenemos 3 gráficas en $\mathbb{R}^2$ la cuales hacen referencia a como se comporta cada uno de los coeficientes del Hamiltoniano.







Por último, se obtienen los coeficientes del Hamiltoniano se manera gráfica y simplificados

al máximo según los módulos de simplificación de Python.

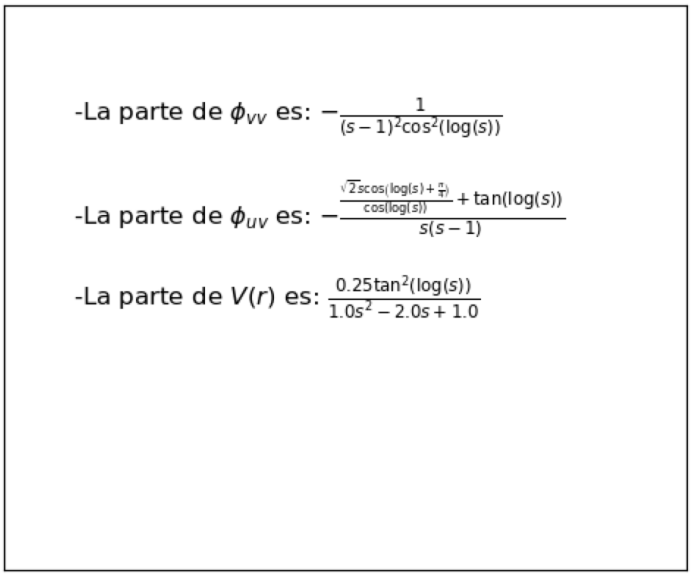


Figura 2.12: Interfaz de salida ejemplo 1

Cabe destacar que tanto las fórmulas simplificadas como las no simplificadas, las podemos obtener por el terminal en lenguaje Latex y tenerlas para futuros documentos.

Ejemplo2

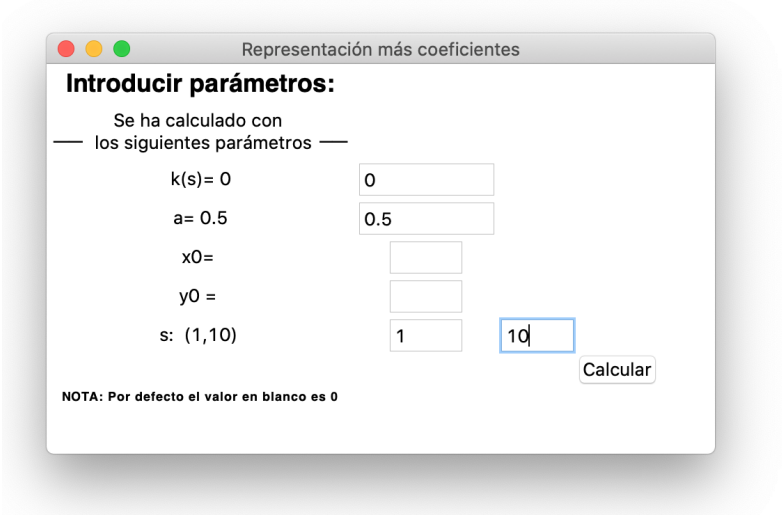


Figura 2.13: Parámetros de entrada ejemplo 2

El primer resultado que obtenemos al ejecutar es la superficie de revolución.

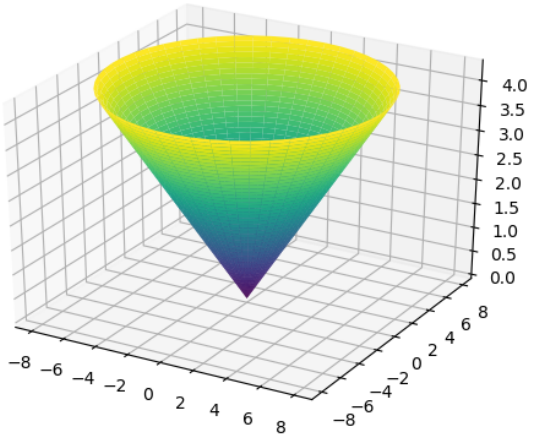
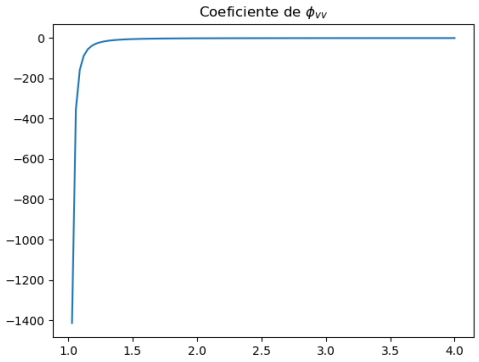
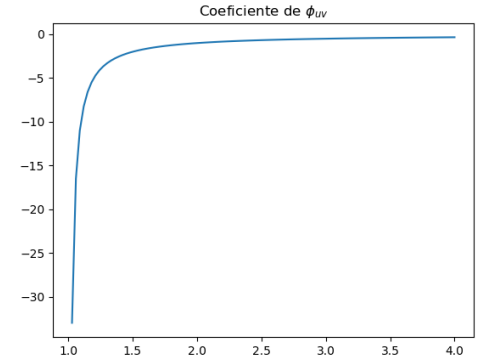
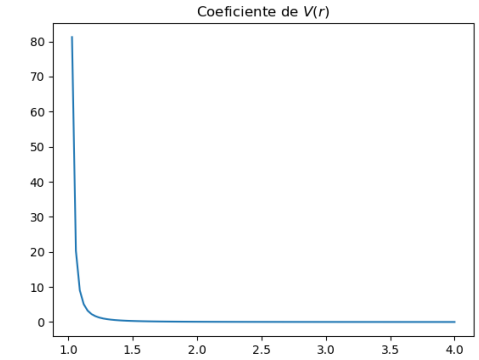


Figura 2.14: Superficie de revolución con curvatura 0

Después obtenemos 3 gráficas en $\mathbb{R}^2$ la cuales hacen referencia a como se comporta cada uno de los coeficientes del Hamiltoniano.







Por último, se obtienen los coeficientes del Hamiltoniano se manera gráfica y simplificados

al máximo según los módulos de simplificación de Python.

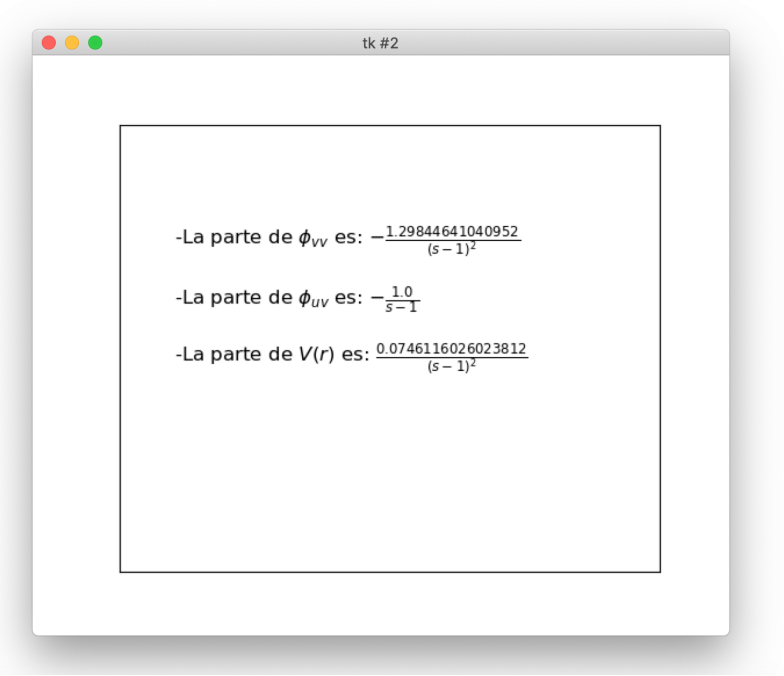


Figura 2.15: Interfaz de salida ejemplo 2

Ejemplo3

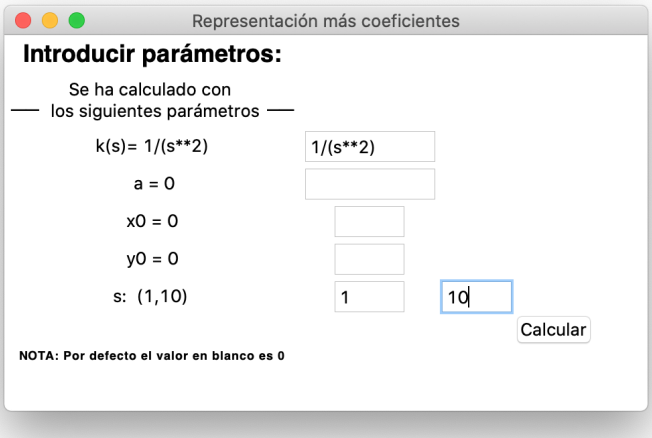


Figura 2.16: Parámetros de entrada ejemplo 3

El primer resultado que obtenemos al ejecutar es la superficie de revolución.

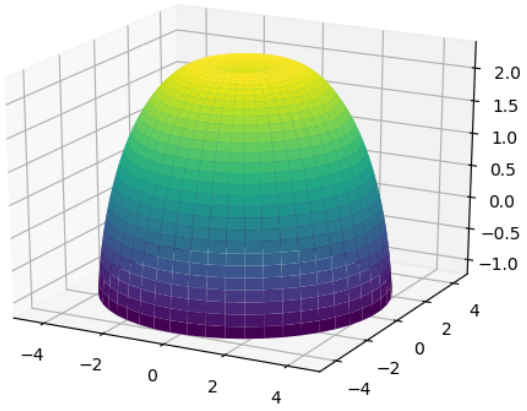
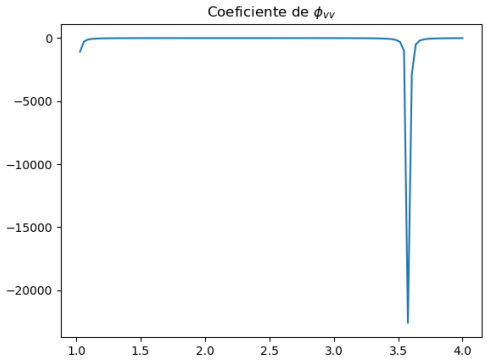
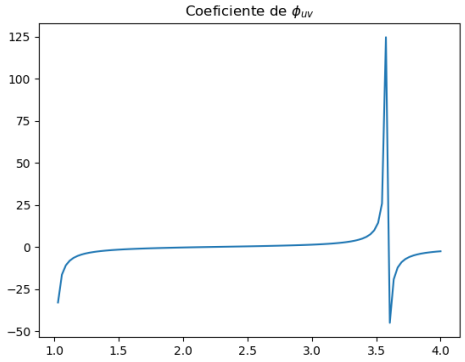
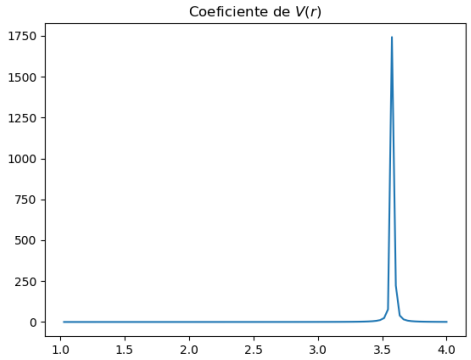


Figura 2.17: Superficie de revolución con curvatura $\frac{1}{s^2}$

Después obtenemos 3 gráficas en $\mathbb{R}^2$ la cuales hacen referencia a como se comporta cada uno de los coeficientes del Hamiltoniano.







Por último, se obtienen los coeficientes del Hamiltoniano se manera gráfica y simplificados

al máximo según los módulos de simplificación de Python.

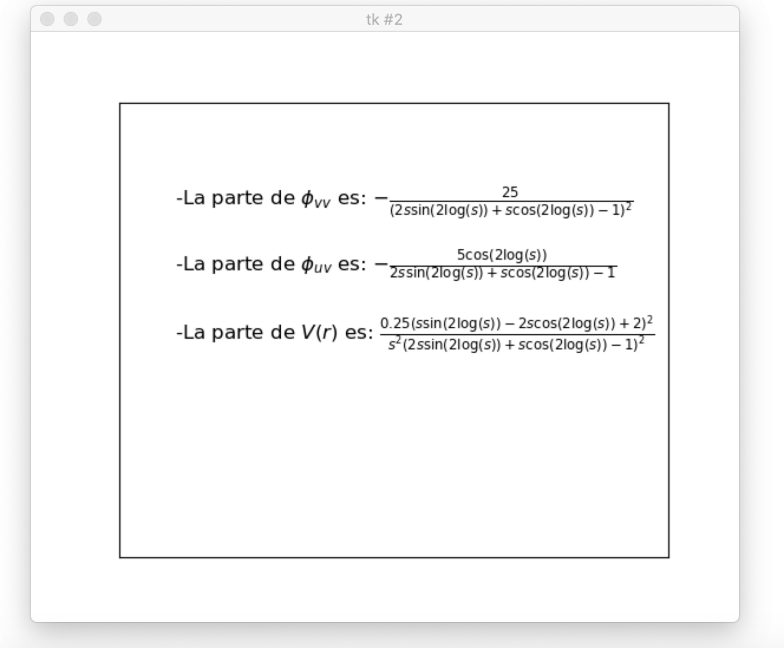


Figura 2.18: Interfaz de salida ejemplo 3

Bibliografía

Memoria prácticas

[1] *Keras.*

https://piperlab.es/glosario-de-big-data/keras/

[2] *KerasClassifier documentation.*

https://www.tensorflow.org/api_docs/python/tf/keras/wrappers/scikit_learn/KerasClassifier

[3] *GridSearchCV documentation.*

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

[4] *TensorFlow.*

https://opensource.com/article/17/11/intro-tensorflow

[5] *Exacuantum.*

https://www.yokogawa.com/solutions/products-platforms/solution-based-software/data-historian/data-historian-exaquantum/

[6] *Redes neuronales.*

http://ocw.uv.es/ingenieria-y-arquitectura/1-2/libro_ocw_libro_de_redes.pdf

[7] Peter Farrell. *Math adventures with python* ISBN-10: 1-59327-867-5

[8] Curso completo de Machine Learning: Data Science en Python

https://www.udemy.com/course/machinelearningpython/

[9] Validación cruzada k-fold

http://www.oldemarrodriguez.com/yahoo_site_admin/assets/docs/Presentaci%C3%83%C2%B3n_-_CV.293124233.pdf

[10] Librerias para programar ecuaciones

https://docs.sympy.org/latest/tutorial/index.html

Memoria tfg

[11] Capítulo1. Curvas en el plano y en el espacio.

https://webs.um.es/pherrero/Geo_dif/curvas2.pdf

[12] Albert Messiah. *Quantum machanics (Volumen 1).* Saclay france

[13] R. C. T. da Costa, *Quantum machanics of a constrained particle, Phys. Rev. A (3) 23 (1981), no. 4, 1982–1987. MR 607422.*

[14] José L. López http://www.ugr.es/~jllopez/Cap3-Sch.pdf

[15] Isaac Chavel *Eigenvalues in Riemannian geometry, Pure and Applied Mathematics, vol. 115, Academic Press Inc., Orlando, FL, 1984, Including a chapter by Burton Randol,With an appendix by Jozef Dodziuk. MR 768584 (86g:58140)*

[16] *Información oficial de Python - interfaces.*

https://docs.python.org/3/library/tk.html

[17] Contenidos gráficos Pyhon

https://matplotlib.org/

Anexo A

Programas

A.1. Programas memoria tfg

```
import tkinter as tk
import math
import numpy as np
import sympy as sp
from sympy.plotting import plot3d_parametric_surface
from sympy.plotting import plot
from mpl_toolkits.mplot3d.axes3d import Axes3D, get_test_data
import cmath
from mpmath import *
from Equation import Expression
from reportlab.pdfgen import canvas
from sympy.parsing.sympy_parser import parse_expr
from sympy import latex
import matplotlib
import matplotlib.pyplot as mpl

from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
matplotlib.use('TkAgg')

from tkinter import *

#----------------------INTERFAZ DE ENTRADA-----------------------
ventana = tk.Tk()
ventana.config(width=400, height=400)
ventana.geometry('480x280')

```

```
entry = tk.Entry(ventana)
entry.place(x=500, y=500)
ventana.title("Representacion mas coeficientes")

#inicializamos las variables donde se van a guardar los datos
var_k = tk.StringVar()
var_a = tk.StringVar()
var_x0 = tk.StringVar()
var_y0 = tk.StringVar()
var_smin = tk.StringVar()
var_smax = tk.StringVar()


#creamos las casillitas de entrada
etiqueta = tk.Label(ventana, text="Introducir parametros:")
etiqueta.configure(font='Helvetica 18 bold')
etiqueta.grid(column=1, row=0)

etiqueta2 = tk.Label(ventana, text="____________________")
etiqueta2.configure(font='Helvetica 20 bold')
etiqueta2.grid(column=1, row=1)

k = tk.Label(ventana, text="Introduce la curvatura k:")
k.grid(column=1, row=2)

a = tk.Label(ventana, text="Introduce el parametro a:")
a.grid(column=1, row=3)

x0 = tk.Label(ventana, text="Introduce x0:")
x0.grid(column=1, row=4)

y0 = tk.Label(ventana, text="Introduce y0:")
y0.grid(column=1, row=5)

etiq_s = tk.Label(ventana, text="Introduce el rango de s")
etiq_s.grid(column=1, row=6)



txt_k = tk.Entry(ventana,textvariable=var_k,width=10)
txt_k.grid(column=2, row=2)

txt_a = tk.Entry(ventana,textvariable=var_a,width=10)
txt_a.grid(column=2, row=3)

txt_x0 = tk.Entry(ventana,textvariable=var_x0,width=5)
txt_x0.grid(column=2, row=4)

```

```
75  txt_y0 = tk.Entry(ventana,textvariable=var_y0,width=5)
76  txt_y0.grid(column=2, row=5)
77
78  txt_smin = tk.Entry(ventana,textvariable=var_smin,width=5)
79  txt_smin.grid(column=2, row=6)
80
81  txt_smax = tk.Entry(ventana,textvariable=var_smax,width=5)
82  txt_smax.grid(column=3, row=6)
83
84
85  def clicked_calcular():
86
87          res = "k(s)= " + txt_k.get()
88          k.configure(text=res)
89
90          if var_a.get() != "":
91                  res2 = "a= " + txt_a.get()
92          else:
93                  res2 = " a = 0"
94          a.configure(text=res2)
95
96          if var_a.get() != "":
97                  res3 = "x0= " + txt_x0.get()
98          else:
99                  res3 = "x0 = 0"
100         x0.configure(text=res3)
101
102         if var_a.get() != "":
103                 res4 = "y0 = " + txt_y0.get()
104         else:
105                 res4 = "y0 = 0"
106         y0.configure(text=res4)
107
108         res5 = "s: " + " (" +txt_smin.get() + "," + txt_smax.get()+ ")
                "
109         etiq_s.configure(text=res5)
110
111         texto = tk.Label(ventana, text="Se ha calculado con \n los
               siguientes parametros")
112         texto.grid(column=1, row=1)
113
114
115 boton_calcular = tk.Button(ventana, text="Calcular", command=
       clicked_calcular)
116 boton_calcular.grid(column=6, row=8 )
117
118 etiqueta2 = tk.Label(ventana, text="NOTA: Por defecto el valor en
       blanco es 0")
```

```
119 etiqueta2.configure(font='Helvetica 9 bold')
120 etiqueta2.grid(column=1, row=9)
121
122 ventana.mainloop()
123
124 s = sp.Symbol('s')
125 t = sp.Symbol('t')
126 v = sp.Symbol('v')
127
128
129 k_entrada = parse_expr(var_k.get())
130
131 if var_a.get() != "":
132         a_entrada = float(var_a.get())
133 else:
134         a_entrada = float(0)
135
136 if var_x0.get() != "":
137         x0_entrada = float(var_x0.get())
138 else:
139         x0_entrada = float(0)
140
141
142 if var_y0.get() != "":
143         y0_entrada = float(var_y0.get())
144 else:
145         y0_entrada = float(0)
146
147 smin = float(var_smin.get())
148 smax = float(var_smax.get())
149
150 #-----------------METODOS AUXILIARES--------------------
151
152 def definir_k():
153         res = k_entrada
154         res = res.subs(s,t)
155         return res
156
157 def definir_theta():
158         res = sp.integrate(k, (t, 1, s)) + a_entrada
159         res = res.subs(s,t)
160         return res
161
162 def definir_alpha():
163         res = []
164         res.append(sp.integrate(sp.cos(theta), (t, 1, s)) + x0_entrada)
165         res.append(sp.integrate(sp.sin(theta), (t, 1, s)) + y0_entrada)
166         return res
```

```
167
168 def X():
169         sup = []
170         x1 = sp.cos(v)*(curva[0])
171         sup.append(x1)
172         x2 = sp.sin(v)*(curva[0])
173         sup.append(x2)
174         x3 = y0_entrada + curva[1]
175         sup.append(x3)
176
177         return sup,x1,x2,x3
178
179
180 #---------------PROGRAMA PRINCIPAL----------------------
181
182
183 k = definir_k()
184 print(k)
185 theta = definir_theta()
186 print(theta)
187 alpha = definir_alpha()
188 print(alpha)
189
190 #renombramos
191 curva = alpha
192
193 sup,x1,x2,x3 = X()
194 print(sup)
195
196
197 #representamos
198 plot3d_parametric_surface(x1, x2, x3, (s, smin, smax), (v, 0, 2*math.pi
       ) )
199
200
201 #---------------COEFICIENTES HAMILTONIANO--------------------
202
203
204
205 #parte de phi_{vv}-------------------------------------
206 res2 = -1 /(curva[0]**2)
207
208 lam_x = sp.lambdify(s, res2, modules=['numpy'])
209
210 x_vals = np.linspace(1, 4, 100)
211 y_vals = lam_x(x_vals)
212
213 mpl.plot(x_vals, y_vals)
```

```
214 mpl.title("Coeficiente de $\phi_{vv}$")
215 mpl.show()
216 res2 = sp.latex(sp.simplify(res2))
217
218 #parte del phi_{u}-----------------------------------
219
220 res3 = - sp.diff(curva[0], s) / curva[0]
221
222 lam_x = sp.lambdify(s, res3, modules=['numpy'])
223
224 x_vals = np.linspace(1, 4, 100)
225 y_vals = lam_x(x_vals)
226
227 mpl.plot(x_vals, y_vals)
228 mpl.title("Coeficiente de $\phi_{uv}$")
229 mpl.show()
230 res3 = sp.latex(sp.simplify(res3))
231
232 #parte del v(r)--------------------------------------
233
234 res4 = 1/4*(k_entrada - (sp.diff(curva[1],s) / curva[0]))**2
235
236 lam_x = sp.lambdify(s, res4, modules=['numpy'])
237
238 x_vals = np.linspace(1, 4, 100)
239 y_vals = lam_x(x_vals)
240
241 mpl.plot(x_vals, y_vals)
242 mpl.title("Coeficiente de $V(r)$")
243 mpl.show()
244
245 res4 = sp.latex(sp.simplify(res4))
246
247
248 #--------------INTERFAZ DE SALIDA----------------------
249
250 root = Tk()
251
252 mainframe = Frame(root)
253 mainframe.pack()
254
255
256 label = Label(mainframe)
257 label.pack()
258
259 fig = matplotlib.figure.Figure(figsize=(6, 5), dpi=100)
260 ax = fig.add_subplot(111)
261
```

```
262 canvas = FigureCanvasTkAgg(fig, master=label)
263 canvas._tkcanvas.pack(side=TOP, fill=BOTH, expand=1)
264
265 ax.get_xaxis().set_visible(False)
266 ax.get_yaxis().set_visible(False)
267
268
269 tmptext = "-La parte de $\phi_{vv}$ es: " +"$"+res2+"$\n \n"+ "-La
       parte de $\phi_{uv}$ es: " + "$"+res3 +"$\n \n"+ "-La parte de $V(r)
       $ es: " +"$"+ res4+"$\n"
270
271
272 ax.clear()
273 ax.text(0.1, 0.4, tmptext, fontsize = 12)
274 canvas.draw()
275
276 root.mainloop()
```

A.2. Programas prácticas

Guardado de la red neuronal del compresor 444-C1B:

```
1
2  from pandas import read_csv
3  import matplotlib.pyplot as plt
4  import numpy as np
5  from keras.models import Sequential
6  from keras.layers import Dense
7  from keras.layers import Dropout
8  from sklearn.metrics import accuracy_score
9
10
11 #----------CARGAR LOS DATOS-------------
12 '''
13 -El num de atributos es el mismo que el nombre de columnas
14 -X:(training_data) Es una matriz que representa el conjunto de datos
15  donde cada fila es un instante de tiempo
16 -Y:(training_targets) son para cada fila el valor binario
      correspondiente
17 '''
18
19 #DATOS DE ENTRENAMIENTO
20
21 dataframe1 = read_csv("set_entrenamiento1B.csv", header = None,
      skiprows=1, sep=';')
22 dataset = dataframe1.values
23
24 num_col = len(dataset[0])
25 num_fil = len(dataset)
26
27 for i in range(num_fil):
28         for j in range(num_col):
29                 if dataset[i][j] != 1 and dataset[i][j] != 0:
30                         dataset[i][j] = dataset[i][j].replace(",",".")
31                         dataset[i][j] = float(dataset[i][j])
32
33
34 #Debemos anyadir al dataset los ratios de compresion.
35
36 PIC45 = dataset[:,0]
37 PI85 = dataset[:,1]
38 PI86 = dataset[:,2]
39 PI87 = dataset[:,3]
40 PI88 = dataset[:,4]
```

```
41  PIC1 = dataset[:,5]
42
43  #creamos las variables con los ratios de compresion
44  ratioC_ET1_2 = []
45  ratioC_ET2_3 = []
46  ratioC_ET3_4 = []
47  ratioC_ET4_5 = []
48  ratioC_ET5_fin = []
49
50  for i in range(num_fil):
51
52                  ratioC_ET1_2.append(PI85[i] / PIC45[i])
53                  ratioC_ET2_3.append(PI86[i] / PI85[i])
54                  ratioC_ET3_4.append(PI87[i] / PI86[i])
55                  ratioC_ET4_5.append(PI88[i] / PI87[i])
56                  ratioC_ET5_fin.append(PIC1[i] / PI88[i])
57
58
59  dataset = np.c_[dataset, ratioC_ET1_2, ratioC_ET2_3, ratioC_ET3_4,
        ratioC_ET4_5, ratioC_ET5_fin]
60
61
62  #actualizamos los rangos del dataset
63  num_col = len(dataset[0])-1
64  num_fil = len(dataset)
65
66  X_train = dataset[:,:num_col]
67  Y_train = dataframe1.iloc[:,-1]
68
69  n_atributos = num_col
70
71  #DATOS DE EVALUACION
72
73  dataframe2 = read_csv("eval1b_1min.csv", header = None, skiprows=1, sep
        =';')
74  #dataframe2 = read_csv("eval1b_10min.csv", header = None, skiprows=1,
        sep=';')
75  #dataframe2 = read_csv("eval1b_1h_medias.csv", header = None, skiprows
        =1, sep=';')
76  dataset2 = dataframe2.values
77
78  fechas = dataset2[:,0]
79  dataset2 = np.delete(dataset2,0, axis=1)
80
81  num_col = len(dataset2[0])
82  num_fil = len(dataset2)
83
84  for i in range(num_fil):
```

```
85          for j in range(num_col):
86                  if dataset2[i][j] != 1 and dataset2[i][j] != 0:
87                          dataset2[i][j] = dataset2[i][j].replace
                               (",",".")
88                          dataset2[i][j] = float(dataset2[i][j])
89
90
91  PIC45 = dataset2[:,0]
92  PI85 = dataset2[:,1]
93  PI86 = dataset2[:,2]
94  PI87 = dataset2[:,3]
95  PI88 = dataset2[:,4]
96  PIC1 = dataset2[:,5]
97
98  #creamos las variables con los ratios de compresion
99
100 ratioC_ET1_2_eval = []
101 ratioC_ET2_3_eval = []
102 ratioC_ET3_4_eval = []
103 ratioC_ET4_5_eval = []
104 ratioC_ET5_fin_eval = []
105
106 def div_cero(num,den):
107         if den != 0:
108                 return num / den
109         else:
110                 return 0
111 for i in range(num_fil):
112
113                 ratioC_ET1_2_eval.append(div_cero(PI85[i], PIC45[i]))
114                 ratioC_ET2_3_eval.append(div_cero(PI86[i], PI85[i]))
115                 ratioC_ET3_4_eval.append(div_cero(PI87[i], PI86[i]))
116                 ratioC_ET4_5_eval.append(div_cero(PI88[i], PI87[i]))
117                 ratioC_ET5_fin_eval.append(div_cero(PIC1[i], PI88[i]))
118
119
120 dataset2 = np.c_[dataset2, ratioC_ET1_2_eval, ratioC_ET2_3_eval,
       ratioC_ET3_4_eval, ratioC_ET4_5_eval, ratioC_ET5_fin_eval]
121
122
123 #actualizamos los rangos del dataset
124
125 num_col = len(dataset2[0])
126 num_fil = len(dataset2)
127
128 X_eval = dataset2[:,:num_col]
129 Y_eval = dataframe2.iloc[:,-1]
130
```

```
131 n_atributos = num_col
132
133
134 #----------DEFINICION DEL MODELO-------------
135
136 def build_model():
137
138         model = Sequential()
139         model.add(Dense(24, input_dim=n_atributos, kernel_initializer='
            uniform',activation ='relu'))
140         model.add(Dropout(0.2))
141         model.add(Dense(16, kernel_initializer='uniform', activation ='
            relu'))
142         model.add(Dropout(0.1))
143         model.add(Dense(1,kernel_initializer='uniform',activation ='
            sigmoid'))
144
145         model.compile(loss='binary_crossentropy',optimizer='adam',
            metrics=['accuracy'])
146
147         return model
148
149
150 #----------FASE DE ENTRENAMIENTO-------------
151
152 model_B = build_model()
153 model_B.fit(X_train, Y_train, epochs=200, batch_size=8, verbose=0)
154
155 #----------------FASE EVALUACION--------------------
156
157 pred_B = model_B.predict(X_eval)
158 pred_B_binaria = pred_B.round()
159
160 #------------------------------------------------------
161 general = False
162 '''
163 -En el caso de que sea general, hay que cambiar esta variable a true
164
165 -Como podemos obserbar estos metodos sirven para calcular la precision
166 para una fallo dado
167 '''
168 #----------------METODOS AUXILIARES--------------------
169
170 def buscar_fecha_primererror():
171         estado = 0
172         Y_eval = []
173         for i in range(len(fechas)):
174                 '''
```

```
175                     Si queremos que se ajuste a un error en concreto
                            ponemos la fecha
176                     '''
177                     if fechas[i] == "22/01/2020 5:40":
178                             estado = 1
179                     Y_eval.append(estado)
180
181             return Y_eval
182
183     if general == false:
184             Y_eval = buscar_fecha_primererror()
185
186     def primer_error(pred):
187             for i in range(len(pred)):
188                     if pred[i] == 1.:
189
190                             print("la primera toma de error se dio en: ",
                                    fechas[buscar_fecha_primererror()]) #
                                    resultado comprovado a mano para las pruebas
191                             print("El primer error ha sido detectado en: ",
                                    fechas[i])
192                             Print("En el indice: ",i)
193
194                             break
195
196
197     #----------------GUARDAR MODELO--------------------
198
199     if general == False:
200             precision = accuracy_score(Y_eval,Y_binaria)
201             print("La precision de la red es: ",precision)
202             if precision > 0.7 :
203                     model_json_B = model_B.to_json()
204                     with open("model_B.json", "w") as json_file:
205                             json_file.write(model_json_B)
206                     model_A.save_weights("model_B.h5")
207                     print("Modelo B guardado en el PC")
208             else :
209                     print("Vuelve a entrenar la red")
210
211     else:
212             model_json_B = model_B.to_json()
213             with open("model_B.json", "w") as json_file:
214                     json_file.write(model_json_B)
215             model_B.save_weights("model_B.h5")
216             print("Modelo B guardado en el PC")
217
218     #------------------VISUALIZACION--------------------------
```

```
219
220
221 N = len(pred_B)
222 N = float(N)
223
224 x = np.arange(0,N,1)
225
226 pred = []
227 Y = []
228
229 for i in range(len(pred_B)):
230         if Y_eval[i] == 0:
231                 Y.append(200)
232         else:
233                 Y.append(300)
234
235         if pred_B[i] == 0 :
236                 pred.append(200)
237         else:
238                 pred.append(300)
239
240
241 fig, (ax1, ax2) = plt.subplots(nrows=2, ncols=1)
242
243 ax1.plot(x,pred_B, label="Predicciones")
244 ax1.plot(x,pred_B_binaria, label="Predicciones")
245 ax1.set_title('Estados');
246
247 ax2.plot(x, dataset2[:,0:5])
248 ax2.plot(x, dataset2[:,6:12])
249 ax2.plot(x, dataset2[:,13])
250
251 ax2.set_xlabel('tiempo')
252 ax2.set_title('Estado');
253
254 fig2, (ax1, ax2) = plt.subplots(nrows=2, ncols=1)
255
256 ax1.plot(x,pred_B, label="Predicciones")
257 ax1.plot(x,pred_B_binaria, label="Predicciones")
258 ax1.set_title('Estados');
259
260 ax2.plot(x, dataset2[:,5])
261 ax2.plot(x, dataset2[:,12])
262 ax2.plot(x, dataset2[:,14])
263 ax2.set_xlabel('tiempo');
264 plt.show()
```

Cargado y ejecución de la red neuronal del compresor 444-C1B:

```
import pyodbc
from keras.models import model_from_json
from tensorflow.python import pywrap_tensorflow
from pandas import read_csv
import numpy as np


'''--------------------CONEXION SQL-----------------

Declaramos los parametros necesarios para acceder a nuestra bbdd,
   siendo:

        -Nombre del servidor: CTOPRO\SQLEXPRESS
        -Nombre de la base de datos: UBECPM
        -Nombre de la tabla: V_MachineLearning_Raw

'''
conn = pyodbc.connect('Driver={SQL Server};'
                      'Server=CTOPRO\SQLEXPRESS;'
                      'Database=UBECPM;'
                      'Trusted_Connection=yes;')

cursor = conn.cursor()
cursor.execute('SELECT * FROM V_MachineLearning_Raw')

datos = []

for row in cursor:
    datos.append(row)

X_cB = []
tags_cB = []
tags_cB.append('Hora')

dt = datos[0][2]
dt = dt.isoformat()
datos[0][2] = dt

fechas_cA = datos[0][2]

for i in range(len(datos)):

        if datos[i][1] != 'C-444-1A':
                tags_cB.append(datos[i][0])
                X_cB.append(datos[i][3])

```

```
46  ratioC_ET1_2_cB = X_cB[0]
47  ratioC_ET2_3_cB = X_cB[1]
48  ratioC_ET3_4_cB = X_cB[2]
49  ratioC_ET4_5_cB = X_cB[3]
50  ratioC_ET5_fin_cB = X_cB[4]
51
52
53  def div_cero(num,den):
54          if den != 0:
55                  return num / den
56          else:
57                  return 0
58
59  ratioC_ET1_2_cB = div_cero(X_cB[1], X_cB[0])
60  ratioC_ET2_3_cB = div_cero(X_cB[2], X_cB[1])
61  ratioC_ET3_4_cB = div_cero(X_cB[3], X_cB[2])
62  ratioC_ET4_5_cB = div_cero(X_cB[4], X_cB[3])
63  ratioC_ET5_fin_cB = div_cero(X_cB[5], X_cB[4])
64
65
66  X_cB.append(ratioC_ET1_2_cB)
67  X_cB.append(ratioC_ET2_3_cB)
68  X_cB.append(ratioC_ET3_4_cB)
69  X_cB.append(ratioC_ET4_5_cB)
70  X_cB.append(ratioC_ET5_fin_cB)
71
72  X_cB = np.array(X_cB)
73  X_cB = X_cB.reshape( (1,20) )
74
75  #---------------CARGAMOS EL MODELO----------------
76
77  def cargar_modelo_c1B():
78          json_file = open('model_B.json', 'r')
79          loaded_model_json = json_file.read()
80          json_file.close()
81          loaded_model = model_from_json(loaded_model_json)
82          # cargar pesos al nuevo modelo
83          loaded_model.load_weights("model_B.h5")
84          print("Cargado modelo B.")
85          return loaded_model
86
87
88  model_B = cargar_modelo_c1B()
89  model_B.compile(loss='binary_crossentropy',optimizer='adam',metrics=['
        binary_accuracy'])
90
91  pred_B = model_B.predict(X_cB)
92  pred_B = pred_B[0][0].round()
```

```
93
94  print("El estado del compresor c1B es:" , int(pred_B))
95
96
97  #-------------------ESCRITURA EN BBDD-------------------------
98
99  if int(pred_B) == 1:
100         cursor.execute('''INSERT INTO MachineLearning_Results (
               Equipment, TS, FaultDetection) VALUES ('C-444-1A
               ','2020-01-22 15:00:00',0), ('C-444-1B','2020-01-22
               15:00:00',1)''')
101 else:
102         cursor.execute('''INSERT INTO MachineLearning_Results (
               Equipment, TS, FaultDetection) VALUES ('C-444-1A
               ','2020-01-22 15:00:00',0), ('C-444-1B','2020-01-22
               15:00:00',0)''')
103 conn.commit()
```