`Scikit-ReducedModel`: reduced order models with automated parameter domain decomposition.

Trabajo final para el curso doctoral Diseño de software para cómputo científico
23-Febrero–2023

Franco Cerino[a], Agustín Rodríguez-Medrano[b,c]

[a] *Facultad de Matemática, Astronomía y Física, Universidad Nacional de Córdoba (FaMAF–UNC) Bvd. Medina Allende s/n, Ciudad Universitaria, X5000HUA, Córdoba, Argentina*
[b] *Instituto de Astronomía Teórica y Experimental - Observatorio Astronómico de Córdoba (IATE, UNC–CONICET), Córdoba, Argentina.*
[c] *Observatorio Astronómico de Córdoba, Universidad Nacional de Córdoba, Laprida 854, X5000BGR, Córdoba, Argentina*

Abstract

In science and engineering is known that obtaining numerical simulations by solving differential equations can be more computational demanding than desired. For example, in the field of general relativity, to obtain expressions of gravitational waves could cost months using supercomputers. Furthermore, there are studies as parameter estimation that can require up to millions of sequential estimations, dominating the computational expense of the problem. In the last years, these problems were addresed building surrogate models from high cost simulations, taking advantage of the redundancy of the solutions with respect to the parameter space, which can build solutions in real time with no loss of accuray. They are comprised of three sequential steps of data processing that extract the most relevant information and use it to learn the patterns of the solutions and build easy to evaluate expressions: building a reduced basis, applying the empirical interpolation method and using a machine learning algorithm to extrapolate the behavior of the solutions to obtain the surrogate model. In this work we made a package that implements all the stages of surrogate models in a *scikit-learn* approach, helping to obtain models easily and build customized models, including a partitioning option of the parameter space that can help to build solutions faster, named hp-greedy approach. The package brings the possibility to do separate studies with each one of the preprocessing steps, building customized models and with the possibility of use them to create new surrogate models.

Keywords: reduced order model; gravitational waves; Python Package

1. Introducción

Los modelos de orden reducido (ROM) se refieren a una variedad de técnicas de modelado donde se incluye la reducción de dimensionalidad del problema tratado, diseñadas con la finalidad de evitar realizar simulaciones numéricas de excesivo costo computacional. En particular, dentro del área de ROM, existen los llamados modelos sustitutos (o surrogates, en inglés), los cuales a partir de un conjunto de soluciones fidedignas logran generalizar el conocimiento para predecir nuevas expresiones de forma rápida y sin pérdida de resolución. Por ejemplo, estos modelos han sido ampliamente adoptados para el estudio de ondas gravitacionales (GWs). Esto se debe a que para modelar las ondas emitidas durante la coalescencia de un sistema binario de agujeros negros se utiliza relatividad numérica (NR), una tarea que requiere un poder computacional costoso, de 10^4 – 10^5 horas CPU por solución Lehner & Pretorius (2014). Por lo tanto, es necesaria una metodología diferente que permita una evaluación y análisis rápido y preciso de las soluciones. Más desarrollo de los estudios en el área de modelos sustitutos y las metodologías utilizadas se puede ver en Tiglio & Villanueva (2022); Field et al. (2014).

Los modelos sustitutos constan de dos etapas: una llamada *offline*, que consta de la construcción y entrenamiento del modelo, donde ocurre la parte computacional más intensiva, y otra llamada *online*, que es la etapa donde se evalúa de forma rápida una expresión para las soluciones. En la parte offline, primero se utiliza el método de bases reducidas, donde se construye una base de baja dimensionalidad a partir de un conjunto de soluciones de entrenamiento, que permite una compresión considerable y precisa de las características del espacio de soluciones. Luego, se aplica el método de interpolación empírico, donde se realiza una compresión en la dimensión temporal de las soluciones y se construye una expresión para un interpolante construido a partir de la base reducida, denominado *interpolante empírico*, que tiene un costo de evaluación muy bajo y será utilizado para proveer las expresiones finales que otorgará el modelo sustituto. Finalmente, para evaluar el interpolante empírico y obtener una solución para cualquier parámetro dentro del dominio de estudio, se realizan regresiones en el espacio de parámetros. Luego, en la parte *online* se obtienen soluciones asociadas a parámetros introducidos por el usuario. Aquí, se evalúa las regresiones realizadas para poder obtener una expresión del interpolante, que es la aproximación entregada por el modelo sustituto.

En cuanto al desarrollo de modelos sustitutos, en la actu-

alidad existen intentos de mejorar los actuales. Un enfoque derivado del método de bases reducidas estándar es *hp-greedy* (Eftang et al. 2010; Cerino et al. 2022), el cual se basa en construir una partición (refinamiento h) del espacio de parámetros y una base reducida (refinamiento p, aquí denominada como RB) para cada partición. Inicialmente se construye una base reducida global y si esta no es tan precisa y compacta como se desea, se particiona el dominio y se construyen nuevas bases locales en cada subdominio. La idea es encontrar bases reducidas para espacios con menor complejidad mediante una estrategia de divide y vencerás, para permitir evaluaciones aun más rápidas de los modelos obtenidos.

Bajo este contexto es necesario el desarrollo de códigos que generen modelos sustitutos, debido a la considerable aplicabilidad que pueden tener, y que permitan a no expertos en el tema realizar estudios con ellos. Un caso particular, sobre el que se monta este proyecto, es `ARBY` (Villanueva et al. 2021). En este trabajo, nos proponemos re-diseñar y ampliar las funcionalidad del mencionado proyecto, de modo que elaboramos un paquete de Python que consta de conjunto de clases y métodos para la construcción de bases reducidas con y sin partición de dominio, interpolación empírica y modelos sustitutos.

2. Introducción Teórica

Tal como se mencionó en la sección 1, la combinación de *bases reducidas* y *interpolación empírica* sirve como técnica para la construcción de modelos sustitutos. En esta sección daremos una introducción teórica a estas dos metodologías y como su combinación se utiliza para la construcción de modelos sustitutos. Para más detalles de los enfoques mostrados se recomienda ver Tiglio & Villanueva (2022). Además, ampliaremos sobre la idea de bases reducidas con partición de dominio, *hp-greedy*.

2.1. Bases reducidas

En el método de bases reducidas se trabaja sobre un espacio de funciones $\mathcal{F} := h_\lambda(t) := h(\lambda, t)$ donde λ es un parámetro multidimensional en un dominio compacto $\mathcal{D}$ y $h_\lambda(t)$ una función con un dominio físico t (por ejemplo, una serie temporal). La construcción de una base reducida se realiza a través de un algoritmo greedy, el cual elige cuidadósamente soluciones de un conjunto de entrenamiento (*training set*) para conformar la base, denotado como $\mathcal{T} := \{h_{\lambda_i}\}_{i=1}^{m}$. A partir de la base reducida $\{e_i\}_{i=1}^{n}$, donde en general $n << m$, se puede realizar aproximaciones de la forma

$$h_\lambda(t) \approx \sum_{i=1}^{n} c_i(\lambda)\, e_i(t), \tag{1}$$

donde los coeficientes $c_i(\lambda)$ se obtienen realizando una proyección ortogonal de $h_\lambda(t)$ sobre la base reducida.

El algoritmo greedy para construir bases reducidas se resume como un método iterativo, donde en cada paso el elemento del conjunto de entrenamiento peor representado por la base se agrega a la misma, hasta llegar a un error de representación menor a una tolerancia ϵ o a un número máximo de elementos n_{max}, determinados por el usuario. Cabe notar que este método es muy preciso ya que busca representar al espacio de soluciones a partir de una base dada por un subconjunto de ellas mismas. Por ejemplo, si se eligiera representar a las soluciones por una base dada por polinomios o funciones periódicas, como se realiza en series de Fourier, es natural esperar que espacios de la misma dimensión representen al espacio de soluciones de una forma menos eficiente.

Como se puede ver, en este enfoque se intenta encontrar una base reducida global que represente a todas las soluciones parametrizadas bajo el dominio $\mathcal{D}$. Este método ha dado resultados en numerosas aplicaciones, pero puede ocurrir que particionando el espacio de parámetros y construyendo bases reducidas localizadas se obtengan aun mejores resultados, logrando bases de menor dimensionalidad al tratar problemas de menor complejidad. Un algoritmo que realiza esto se denomina *hp-greedy*, que comienza construyendo una base reducida global como en el caso estandard, pero si esta no representa suficientemente bien el espacio de funciones para una tolerancia ϵ, se procede a particionar el dominio. Se toma los dos primeros parámetros elegidos para construir la base global (notar que el segundo parámetro está asociado a la función que más difiere de la asociada al primer parámetro) y se divide el espacio en dos partes, trazando un plano ortogonal a la recta que une los dos parámetros, que se ubica al medio de la misma. La modalidad de esta partición es adaptativa, ya que se adapta al comportamiento de los datos de entrenamiento utilizados. Luego, se trata cada uno de los subespacios de la misma forma que al espacio $\mathcal{D}$, construyendo bases reducidas locales, y en caso de que una no sea lo suficientemente buena, se procederá a particionar el subespacio hasta que se logre la tolerancia deseada o se llegue a una cantidad máxima de particiones locales l_{max}. Al particionar en dos cada subespacio, se puede esquematizar a las particiones en un árbol binario, donde l_{max} representa la profundidad máxima que este puede tener. Se debe tener en cuenta que la especificación de este parámetro puede influir considerablemente en el aprendizaje de las funciones, pudiendo llevar a casos de underfitting u overfitting en caso de no estar bien especificado. Para encontrar un valor acertado de l_{max} y n_{max}, se recomienda realizar una búsqueda de hiperparámetros con ellos.

2.2. Interpolación Empírica

El Método de Interpolación Empírica busca realizar una compresión en la dimensión temporal y obtener una aproximación precisa del espacio de funciones utilizando la misma.

A partir de la base reducida encontrada, se emplea un algoritmo greedy para realizar la compresión temporal, donde se seleccionan los tiempos más relevantes, denominados *tiempos empíricos* denotados como $\{T_i\}_{i=1}^{n}$,y se obtiene una expresión para un *interpolante empírico* que interpola en los tiempos encontrados. Se debe notar que utiliza la base reducida como base para construirse (a diferencia de la interpolación estandard, donde por ejemplo se utilizan polinomios para construir un interpolante que pase por nodos previamente definidos), lo cual le permite otorgar representaciones muy precisas. Este es de la forma

$$\mathcal{I}[h](\lambda; t) := \sum_{i=1}^{n} B_i(t) h(\lambda, T_i)\,, \tag{2}$$

donde los valores $B_i(t)$ cumplen $B_i(T_j) = \delta_{ij}$ y se obtienen únicamente a partir de la base reducida y de los tiempos empíricos.

2.3. Modelos Sustitutos

A partir de la ecuación (2), se puede observar que para obtener una expresión de una solución para un λ arbitrario, es necesario tener un valor de $h(\lambda, T_i)$ para todo T_i. Si se procede a obtener estos valores, el modelo sustituto queda completamente determinado. Por lo tanto, para cada T_i se toma los valores $\{h(\lambda_j, T_i)\}_j$ del conjunto de entrenamiento para entrenar un algoritmo de machine learning y poder tener una expresión $h_i^{\mathrm{ML}}(\lambda, T_i)$ para todo λ en $\mathcal{D}$.

La expresión final de una solución queda dada por

$$h_{\mathrm{ML}}(\lambda; t) := \sum_{i=1}^{n} B_i(t) h_i^{\mathrm{ML}}(\lambda, T_i)\,, \tag{3}$$

la cual es fácil de evaluar para obtener una solución, a diferencia de tener que resolver un sistema de ecuaciones diferenciales para un parámetro λ dado.

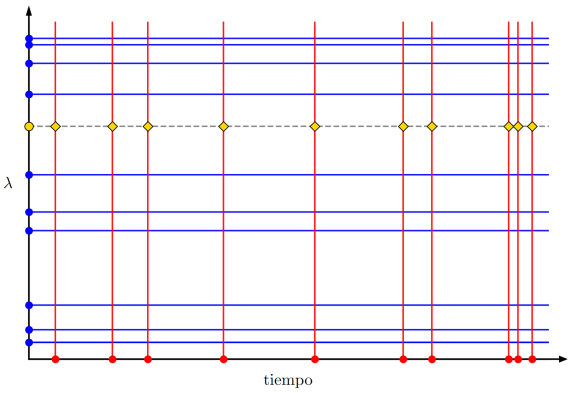

Figure 1: Representación esquemática de los tres pasos utilizados para construir un modelo sustituto. Las líneas horizontales azules representan las soluciones elegidas para la base reducida, donde los puntos azules representan a los parámetros asociados (paso 1). Los puntos rojos representan los tiempos empíricos obtenidos (paso 2). Las líneas verticales rojas representan las regresiones realizadas en los tiempos empíricos (paso 3). La línea horizontal de color amarillo es una solución predicha para un parámetro dado por el usuario (círculo amarillo), que se obtiene evaluando las regresiones (rombos amarillos) en dicho parámetro y utilizando estas para obtener un interpolante empírico.

3. Paquete de python

Se ha desarrollado un paquete *open-source* de modelos de orden reducido utilizable en diversas áreas científicas, basándonos en la idea de diseño de la librería `scikit-learn` (Pedregosa et al. 2011). La filosofía de diseño de `scikit-learn` es proporcionar una interfaz fácil de usar, eficiente y bien documentada para algoritmos de aprendizaje automático en el lenguaje de programación Python, que permita incluso a usuarios no expertos en el área de conocimiento del paquete resolver problemas de aprendizaje automático de manera eficiente y sin interrupciones (Buitinck et al. 2013).

Se ha seguido los siguientes principios de diseño:

- **No proliferación de clases.** Los algoritmos de aprendizaje son los únicos objetos que se deben representar mediante una clase de Python.
- **Composición.** Si la constitución de un algoritmo de *machine learning* es posible interpretar a partir de un conjunto de componentes fundamentales, representarlo de esta forma.
- **Valores por default.** Existencia parámetros definidos por default para cada algoritmo, permitiendo al usuario poder obtener primeras estimaciones de forma sencilla.

Al igual que todos los objetos de `scikit-learn`, el uso de `scikit-reducedmodel` se basa en aplicar tres interfaces complementarias:

- *estimadores*: definen la instanciación de objetos con sus respectivos hiperparámetros y exponen un método `fit`, al cual se le dan datos de entrenamiento y para entrenar un modelo a partir de ellos.
- *predictores*: extienden la noción de estimadores, agregando un método `predict`, el cual recibe un input y genera una predicción de datos no utilizados para entrenamiento, utilizando el modelo entrenado.
- *transformadores:* toman como datos como input y obtienen una versión transformada de ellos, a través del método `transform`. Por ejemplo, una operación que debe tomar este rol es el de una proyección de un vector sobre un espacio vectorial dado.

3.1. Arquitectura e implementación

La API pública de `scikit-reducedmodel` consta de tres módulos principales y un método factory. Cada módulo implementa una etapa de la secuencia para construir un modelo sustituto, el cual se obtiene a través de la composición de los mismos (ver figura 2). Se debe notar que la modularización realizada en el proyecto permite realizar estudios específicos de bases reducidas o el método de interpolación empírico:

- reducedbasis: contiene la clase `ReducedBasis`, que dispone de la implementación para obtener una base reducida y, en caso que se indique, realizar particiones de dominio con sus respectivas bases. La clase es un estimador que puede instanciarse con parámetros opcionales (hiperparámetros de la base reducida o hp-greedy), como la dimensión máxima de una base reducida, `nmax`, la profundidad máxima del árbol de particiones, `lmax`, y la tolerancia de representación, `greedy_tol`. El método principal de esta clase es el método `fit` con el cual se entrena la

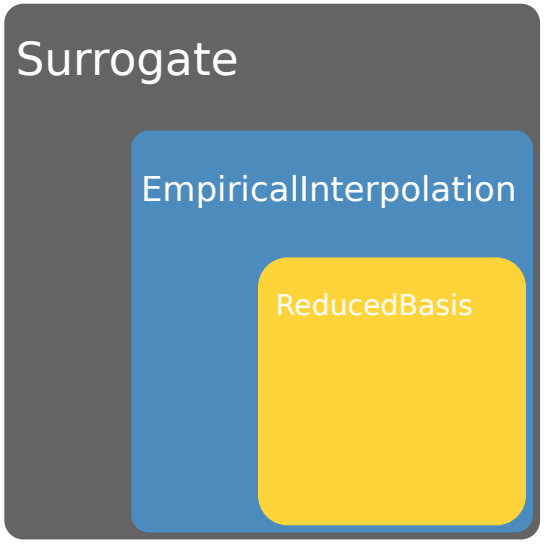

Figure 2: Esquema representativo de las clases de `Scikit-ReducedModel`. La clase Surrogates se instancia a partir de la clase EmpiricalInterpolation, que a su vez se instancia a partir de ReducedBasis.

base reducida. Otro método implementado es `transform`, el cual se utiliza para proyectar una función dada en una base reducida creada. Si el espacio de parámetros está particionado, a partir de una estructura de árbol binario creada al particionar, se realiza la búsqueda del subespacio a partir de un parámetro λ dado por el usuario.

- empiricalinterpolation: contiene la clase `EmpiricalInterpolation`, la cual es un estimador que necesita una instancia de `ReducedBasis` para ser inicializado. Contiene el método `fit` con el cual se comprime la dimensión temporal y construye la expresión para obtener un interpolante empírico para cada partición del dominio. Contiene el método `transform`, al cual se le debe otorgar una función con su parámetro λ asociado para obtener el interpolante empírico asociado a la misma. Al igual que en `ReducedBasis`, λ se utiliza para encontrar el subespacio correspondiente.

- surrogate: contiene la clase `Surrogate`, la cual también es un estimador y necesita una instancia de `EmpiricalInterpolation` para ser inicializada. Dispone del método `fit`, con el cual en cada partición se entrenan las regresiones en los tiempos empíricos obtenidos al entrenar la instancia de `EmpiricalInterpolation` y se finaliza la etapa de entrenamiento de un modelo sustituto. El modelo utilizado para realizar las regresiones es Splines d. Boor (1978). Además, cuenta con un método `predict`, al cual se le debe dar un parámetro λ y otorgará una predicción del modelo reducido.

- mksurrogate: contiene la función `mksurrogate`, que se puede clasificar como un patrón de diseño factory, ya que utiliza una serie de subclases y toma toda la responsabilidad para construir una instancia de una nueva clase. Al utilizarla se deben otorgar datos de entrenamiento y se pueden especificar hiperparámetros de cualquiera de las clases involucradas para conformar un modelo sustituto. Crea y entrena una instancia de `ReducedBasis` con los hiperparámetros y datos de entrenamiento dados, para luego construir una instancia entrenada de `EmpiricalInterpolation` y por último entrena una nueva instancia de `Surrogate`. Se devuelve un modelo sustituto listo para realizar predicciones, permitiendo una interfaz simple al usuario y evitando tener que inicializar todas las clases por separado.

3.2. *Estándares de calidad y versionado*

Para controlar la calidad del software se ha implementado test unitarios a todas las funcionalidades del paquete. Mediante la implementación de test, se ha logrado alcanzar un *coverage* del código escrito de 96%. Los test se centran en controlar que los *input* y *output* de las clases y funciones sean correctos, en comparar los resultados generados con los obtenidos con otros softwares y en comparaciones con resultados teóricos.

Todo el código esta escrito siguiendo la guía de estilos de PEP-8 [1]. A su vez, la documentación de todas las clases y funciones sigue la convención de `numpy`. Estos estándares contribuyen a una buena legibilidad del código y facilitan su mantenimiento.

Para simplificar el entendimiento del código, toda la documentación puede accederse mediante la web de `readthedocs` [2], donde también se encuentran tutoriales del paquete. Estos tutoriales contienen ejemplos sencillos para los usuarios no expertos en el tema y ejemplos más complejos donde nos centramos en la ciencia que puede realizarse con `Scikit-ReducedModel`. Toda la documentación del proyecto fue generada mediante la herramienta `Sphinx`.

Para facilitar el desarrollo en equipo del proyecto, se ha controlado los test y estilos utilizando la herramienta `tox` y realizando integración continua en el repositorio público del proyecto en Github [3].

Finalmente se puede destacar que `Scikit-ReducedModel` es un proyecto *open-source* que funciona bajo una licencia MIT. El mismo se instala a través del repositorio `PyPi` [4]

3.3. *Ejemplo de uso: péndulo simple amortiguado*

En este breve ejemplo, se construye una base reducida e interpolante empírico de un péndulo simple. Se utiliza los métodos *transform* de la base reducida e interpolante empírico para obtener las respectivas aproximaciones. Por último se construye un modelo sustituto para el péndulo y se obtiene una solución a través del método *predict*.

```
>>> import numpy as np
>>> import matplotlib.pylab as plt
```

[1] https://peps.python.org/pep-0008/
[2] https://scikit-reducedmodel.readthedocs.io/en/latest/
[3] https://github.com/francocerino/scikit-reducedmodel
[4] https://pypi.org/project/Scikit-ReducedModel/

```
# Primero se construye una expresion teórica
# para un péndulo simple amortiguado.

>>> from scipy.integrate import odeint
>>> def pend(y, t, b, λ):
>>>     θ, σ = y
>>>     dydt = [σ, -b*σ -
>>>             λ*np.sin(θ)]
>>>     return dydt

>>> b = 0.2 # fricción
>>> y0 = [np.pi/2, 0.] # condición inicial
>>> times = np.linspace(0,50,1001)

# A partir de expresiones teóricas
# se construye un conjunto de entrenamiento
# (training_set) para un conjunto de
# parámetros (λ_train).

>>> λ_train = np.linspace(1,5,101)
>>> training_set = []
>>> for λ in λ_train:
>>>     sol = odeint(pend,y0, times, (b,λ))
>>>     training_set.append(sol[:,0])
>>> training_set = np.array(training_set)

# Solución de test.
>>> λ_test = 2.94
>>> sol_test = odeint(pend, y0, times, (b,λ_test))

# Se importa la clase ReducedBasis
# y se la instancia con hiperparámetros
# para contruir una base reducida.

>>> from skreducedmodel.reducedbasis
>>> import ReducedBasis
>>> rb = ReducedBasis(greedy_tol=1e-16,
>>>                   lmax=2,
>>>                   nmax=10
>>>                   )
>>> rb.fit(training_set=training_set,
>>>        parameters=λ_train,
>>>        physical_points=times
>>>        )
>>> sol_rb = rb.transform(sol_test, λ_test)

# Se accede a la base reducida en
# diferentes hojas del tree mediante
# el método rb.tree.leaves
# ej: coloreamos los parametros de
# entrenamiento de cada partición.

>>> np.random.seed(seed=4)
>>> for leaf in rb.tree.leaves:
>>>     color = np.random.rand(3,)
>>>     for p in leaf.train_parameters[:,0]:
>>>         plt.axvline(p, c=color, lw=.5)

# Se construye el interpolante empírico.

>>> from skreducedmodel.empiricalinterpolation
>>> import EmpiricalInterpolation
>>> eim = EmpiricalInterpolation(rb)
>>> eim.fit()
>>> sol_eim = eim.transform(sol_test, λ_test)

# Se construye el modelo sustituto.

>>> from skreducedmodel.surrogate
>>> import Surrogate
>>> surrogate = Surrogate(eim,
>>>                       poly_deg=3
>>>                       )
>>> surrogate.fit()
>>> sol_surr = surrogate.predict(λ_test)

# Si el usuario solamente va a realizar
# predicciones con modelos sustitutos, puede
# proceder de forma más sencilla utilizando
# la función factory mksurrogate:

>>> from skreducedmodel.mksurrogate
>>> import mksurrogate
>>> surrogate = mksurrogate(
>>>                 training_set=training_set,
>>>                 parameters=λ_train,
>>>                 physical_points=times
>>>                 greedy_tol=1e-16,
>>>                 lmax=2,
>>>                 nmax=10
>>>                 )

>>> sol surr = surrogate.predict(λ_test)
```

En el panel superior de la figura 3 se muestras las ondas construidas en el ejemplo anterior: `sol_rb`, `sol_eim` y `sol_surr`. Puede observarse que a simple vista son indistinguibles de la solución teórica. En el panel inferior de la misma se muestras el error de los tres métodos a lo largo de todo el espacio de parámetros utilizado. Las líneas verticales indican diferentes parámetros de entrenamiento y están coloreadas de acuerdo a la partición a la que pertenecen.

4. Conclusiones

Con este trabajo se lanza la versión 1.0 del código `Scikit-ReducedModel`, el cual es un software para construir modelos de orden reducido, que sirven para construir aproximaciones de espacios de funciones muy precisas y de rápida evaluación, utilizando los enfoques de bases reducidas, interpolantes empíricos y modelos sustitutos. Mas allá del conjunto básico de herramientas para la construcción de modelos reducidos, el

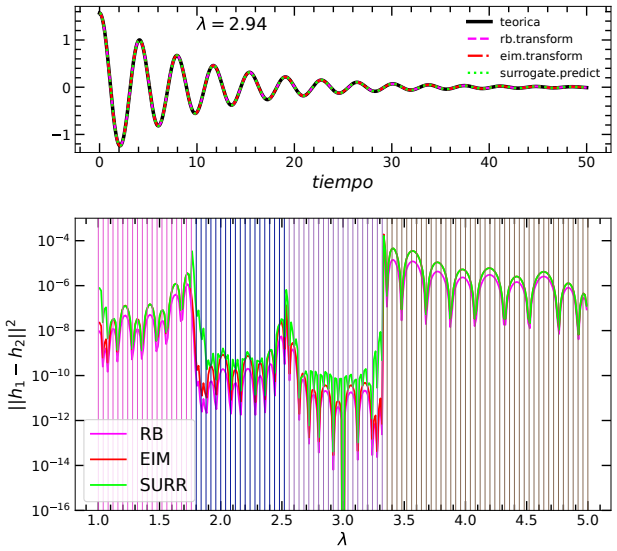

Figure 3: El panel superior contiene las soluciones construidas en el ejemplo de la sección 3.3. Es decir, en una curva negra continua la solución teórica, en una curva discontinua rosa la proyección de esta onda en la base reducida, en línea discontinua punteada roja la interpolación empírica y en una curva de puntos verde la construida mediante el modelo sustituto. El panel inferior muestra los errores a lo largo del espacio de parámetros de los 3 métodos con respecto a soluciones teóricas que no han sido utilizadas para entrenar los modelos. Para mostrar las particiones, se muestran líneas verticales que indican diferentes parámetros de soluciones de entrenamiento del modelo, coloreadas de acuerdo a la partición que pertenecen.

código implementa el refinamiento *hp-greedy*, el cual es una técnica para poder construir bases reducidas de menor dimensionalidad y dar lugar a modelos con un tiempo de evaluación aun menor. `Scikit-ReducedModel` fue diseñado en base a los estandares de `Scikit-Learn`, haciendo incapié en usabilidad sencilla y curva de aprendizaje baja para el usuario, pensando en que sea fácilmente utilizable por el público no experto.

Como trabajo futuro queda pendiente agregar mayor flexibilidad al paquete al momento de realizar las regresiones para un modelo sustituto. Este paso es el que mayor error introduce el modelo, y también se debe tener en cuenta que el tiempo de evaluación de las regresiones influye en el tiempo de predicción de una solución. Por lo tanto se planea desarrollar una interfaz a través de la cual el usuario pueda introducir un algoritmo de otra librería con el cual se realizarán las regresiones. Como justificación a esta nueva implementación se puede mencionar al teorema de No Free Lunch Wolpert (2002), el cual menciona que no existe un modelo de machine learning que pueda tener mejor performance que otros a lo largo de todos los conjuntos de entrenamiento posibles, y el trabajo Barsotti et al. (2022), que hace un análisis de la preponderancia que tiene la elección de un algoritmo de regresión correcto al momento de diseñar modelos sustitutos. Otro desarrollo posible es permitir realizar modelos sustitutos con un espacio de parámetros multidimensional, que tendría una aplicación directa por ejemplo en modelos de coalescencia de sistemas binarios de agujeros negros con spin arbitrario. Por último, el paquete se puede enviar para ser introducido como parte de la librería `scikit-learn`.

References

Barsotti, D., Cerino, F., Tiglio, M., & Villanueva, A. 2022, Classical and Quantum Gravity, 39, 085011

Buitinck, L., Louppe, G., Blondel, M., et al. 2013

Cerino, F., Diaz-Pace, J. A., & Tiglio, M. 2022, An automated parameter domain decomposition approach for gravitational wave surrogates using hp-greedy refinement

d. Boor, C. 1978, A Practical Guide to Splines (New York: Springer Verlag)

Eftang, J. L., Patera, A. T., & Ronquist, E. M. 2010, SIAM J. Sci. Comput., 32, 3170

Field, S. E., Galley, C. R., Hesthaven, J. S., Kaye, J., & Tiglio, M. 2014, Phys. Rev. X, 4, 031006

Lehner, L. & Pretorius, F. 2014, ARAA, 52, 661

Pedregosa, F., Varoquaux, G., Gramfort, A., et al. 2011, Journal of Machine Learning Research, 12, 2825

Tiglio, M. & Villanueva, A. 2022, Living Reviews in Relativity, 25

Villanueva, A., Beroiz, M., Cabral, J., Chalela, M., & Dominguez, M. 2021, arXiv e-prints, arXiv:2108.01305

Wolpert, D. H. 2002, The Supervised Learning No-Free-Lunch Theorems, ed. R. Roy, M. Köppen, S. Ovaska, T. Furuhashi, & F. Hoffmann (London: Springer London), 25–42