Sobre la investigación

El objetivo de esta investigación es optimizar el cálculo de estructuras con el método de los elementos finitos utilizando el método de la base reducida. Se trabajan distintas alternativas de como armar la base reducida y como realizar la parte online. En particular se hace énfasis sobre métodos no intrusivos ya que en la mayoría de los softwares de este tipo no se conoce la ecuación si no que solo se puede trabajar con los snapshots obtenidos.

En una primera instancia se intento resolver el problema por métodos que requieren las ecuaciones diferenciales y las formulaciones débiles. Estos métodos funcionan bien, pero tienen la gran desventaja es que no son siempre aplicables ya que en la mayoría de casos se desconoce la ecuación. Puede ser fácil para probar estructuras simples como losas o barras apoyadas en los bordes, pero para combinar estas no es sencillo. Otra complejidad de este método es que muchas veces surgen problemas con la formulación débil para poder expresar las condiciones de borde. Si bien se pueden usar métodos como penalización o formulación mixta estos no siempre funcionan.

Es por esto que se considera ventajoso considerar métodos no intrusivos. Para esto obtenemos un conjunto de snapshots (para las pruebas usamos fenics pero la idea es que no dependa del software elegido) y se le aplica POD (con UQpy) para encontrar la base reducida. Con estos snapshots y su parámetro asociado entrenamos una red neuronal para buscar una función que dada un vector de parámetros podramos encontrar multiplicadores que minimicen el error relativo. Luego testeamos esta red con un conjunto de testeo para ver su efectividad.

Ventajas

* Prescindimos de la ecuación y la formulación débil
* La etapa online del problema es casi instantánea ya que solo requiere evaluar una red ya entrenada

Desventajas

* No es sencillo encontrar la arquitectura de la red adecuada. Los parámetros de la red como cantidad de neuronas, capas ocultas y funciones de activación los debe determinar uno para cada modelo
* Es menos exacto que el método de los elementos finitos
* Si bien se puede modificar algunos parámetros no se puede cambiar la geometría del problema ya que esto requeriría calcular la etapa offline denuevo

Documentos

El proyecto se divide en 3 partes. Todos los códigos fueron escritos en python

1. Investigación Teórica: Contiene la información necesaria para entender el problema. Trata sobre el problema físico, el método de los elementos finitos y el método de bases reducidas con diferentes alternativas
2. Documentos en Google colab: Contienen las primeras pruebas realizadas. Primero usando la biblioteca fenics que implementa el método de los elementos finitos y la biblioteca rbnics que implementa en conjunto con fencis. Como es complicado trabajar con las ecuaciones diferenciales que gobiernan el problema pasamos a una alternativa de hacer POD para hallar la base reducida y luego entrenar una red neuronal para poder realizar la parte online. En esta etapa se usa UQpy para realizar pruebas de POD y ejemplos con redes neuronales
3. Entrenamiento con redes: Esta carpeta esta pensada para ejecutar de forma local. Se realiza el proceso descrito arriba para 2 estructuras distintas un bloque elástico y una losa

Desglose por carpetas

1. Investigación Teórica
   * Marco teórico: Docuemento de jupyter (para abrir con Google colab)
2. Documentos en Google colab
   * Fenics
     1. Kirchhoff-Love plate fenics: Documento de jupyter que contiene la formulación débil del problema de la losa de Kirchhoff-Love y su resolución mediante FEM en FEniCS. Incluye comparación con resultados
     2. Solución a barra de Euler con fenics: Documento de jupyter que contiene las pruebas iniciales para validar la formulación débil y su implementación con FEniCS en el caso de la barra de Euler.
   * Rbnics
     1. Barra de Euler rbnics: Documento de jupyter que incluye la deducción y formulación débil de la ecuación de la barra de Euler. Se consideran primero fuerzas transversales y luego se incorporan fuerzas longitudinales. Se aplica el método de base reducida usando RBniCS y se compara con los resultados
     2. Copia de tutorial elastic block comentado: Ejemplo resuelto de un bloque elástico con rigidez variable. El extremo izquierdo está fijo y en el derecho se aplican tres fuerzas distintas. El documento explica detalladamente el uso de RBniCS comentando cada sección del código.
     3. Presentación Losa: Versión más organizada y visual del problema de la losa de Kirchhoff-Love, pensada para presentación.
     4. Kirchhoff-Love plate RBniCS: Similar al anterior, pero implementando la resolución con bases reducidas usando RBniCS. La losa se divide en 4 cuadrantes, variando las fuerzas aplicadas y el espesor.
   * Redes
     1. Elementos para red con rigidez: Contiene todos los elementos necesarios para entrenar una red neuronal que resuelva el problema del bloque elástico. Se incluyen los archivos:
        1. training\_set, testing\_set: parámetros muestreados aleatoriamente en el espacio paramétrico.
        2. snapshots\_matrix: soluciones FEM generadas con FEniCS.
        3. base\_reducida: obtenida mediante POD usando UQpy.
        4. salidas, salidas\_esperadas: coeficientes óptimos por mínimos cuadrados.
        5. result\_matrix: salidas de la red comparadas con las esperadas.
     2. Generalización redes: Explica cómo entrenar una red neuronal capaz de predecir los coeficientes óptimos de la base reducida para nuevos parámetros. Este proceso requiere previamente ejecutar el contenido de “Elementos\_para\_red\_con\_rigidez”.
3. Entrenamiento con redes neuronales: Estas carpetas están pensadas para ejecución local ya que el volumen de datos y operaciones que manejan son mayores. Además de redes neuronales para encontrar multiplicadores se prueba con diversas tencincas como regresión polinómica, random forest y otros modelos lineales. Para poder ejecutar el código se requieren tener instaladas las siguientes bibliotecas:
   * Fenics (versión del 2019, no fenicsx)
   * UQpy
   * Pytorch
   * Sklearn
   * Entre otras bibliotecas clásicas

Se divide en 2 experimentos:

1. Bloque elástico: Es un cuadrado dividido en 9 subcuadrados que esta empotrado en el lado izquierdo y se le aplican fuerzas de empujar o tirar en 3 secciones del lado derecho (ver imagen en anexos)
   * 1. Main: Contiene la estructura del código principal. Carga o crea los datos de entrenamiento y teste. Luego entrena la red y la evalúa
     2. Mallado: Importa los archivos del mallado
     3. POD: contiene las funciones que implementan POD con la biblioteca de UQpy
     4. Polinomio: hace la regresión polinómica para aproximar la función de los parámetros en los multiplicadores
     5. Prueba\_test: Contiene datos de testeo
     6. Prueba\_train: Contiene datos de entrenamiento
     7. Redes: Clase usada para el manejo de redes con torch
     8. Testing: Crea los datos de testeo
     9. Training: Crea los datos de entrenamiento
     10. Training2: Crea datos de entrenamiento en paralelo (es recomendable usar este)
2. Losa: Es una losa cuadrada dividida en 4 cuadrantes, que esta fijada en los bordes y en cada cuadrante se le aplica una fuerza (hacia arriba o hacia abajo). También se hace variar el espesor de la losa (ver imagen en anexos). La estructura es similar al anterior:
   * 1. Elementos de prueba: directorio que contiene los conjuntos de entrenamiento y testeo
     2. Lineales: batería de métodos lineales para testear
     3. Lineales2: muy parecido
     4. Main: Contiene la estructura del código principal. Carga o crea los datos de entrenamiento y teste. Luego entrena la red y la evalúa
     5. POD: contiene las funciones que implementan POD con la biblioteca de UQpy
     6. Polinomio: hace la regresión polinómica para aproximar la función de los parámetros en los multiplicadores
     7. Random Forest: Contiene el código para intentar aproximar los multiplicadores con un random forest
     8. Redes: Clase usada para el manejo de redes con torch
     9. Redes\_paralelo: Código que permite hacer pruebas con redes de distintas arquitecturas sobre el mismo conjunto de datos simultáneamente
     10. Relaciones: Permite visualizar relaciones entre coordenadas de entrada y salida
     11. SVR: Codigo que implementa SVR
     12. Training\_losa: Crea los datos de entrenamiento
     13. Testing\_losa: Crea los datos de testeo

Conclusión

En el bloque elástico con 1000 datos de entrenamiento, 30 datos de testeo, una base reducida de 10 elementos y una red de 3 capas y 48 neuronas por capa se genera un modelo con estos resultados

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Error Relativo | | Error relativo al desplazamiento máximo | |
|  | Entrenamiento | Testeo | Entrenamiento | Testeo |
| Promedio | 0.0049 | 0.0137 | 0.0055 | 0.0157 |
| Varianza | 3.34e-5 | 5.23e-5 | 2.98e-5 | 5.86e-5 |
| Máximo | 0.0414 | 0.0334 | 0.089 | 0.0316 |

Esta aproximación es muy buena con los dos métodos de evaluación.

También se probó con los otros métodos lineales y el mejor fue la regresión polinómica que para un polinomio de grado 3 genera con error relativo

|  |  |  |
| --- | --- | --- |
|  | Entrenamiento | Testeo |
| Error relativo promedio | 0.005451 | 0.0048683 |
| Error relativo máximo | 0.018261 | 0.0108789 |

Se puede ver que la regresión polinómica genera una muy buena aproximación.

En la losa con 4000 datos de entrenamiento, 40 de testeo, una base reducida de 4 elementos y una red de 3 capas y 150 neuronas se genera un modelo con estos resultados

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Error Relativo | | Error relativo al desplazamiento máximo | |
|  | Entrenamiento | Testeo | Entrenamiento | Testeo |
| Promedio |  |  |  |  |
| Varianza |  |  |  |  |
| Máximo |  |  |  |  |

La aproximación no es tan buena como precisamos. Los modelos lineales tampoco funcionaron para este modelo.

En vista a estos resultados si bien este método no intrusivo de POD puede generar buenos resultados para algunas estructuras carece de exactitud (un problema habitual cuando se trabaja con redes neuronales). Si se tuviese mucho tiempo y recursos se podría intentar ver que pasa con un conjunto de entrenamiento más grande para ver si esto mejora los resultados.

Análisis temporal

El proceso se divide en 2 etapas que llamamos offline y online:

* Etapa offline: Esta es la primera etapa, aquí se calculan los snapshots[[1]](#footnote-1) necesarios de testeo y entrenamiento con el método de los elementos finitos. Se calcula la base reducida con POD y para esto puede ser necesario hacer la descomposición de valores singulares para hallar la cantidad de modos que deben estar en la base reducida (tener en cuenta que a mayor cantidad de modos más difícil es de aproximar la función para la red). Con la base reducida calculamos las salidas esperadas para la red con mínimos cuadrados ya que esta es la mejor combinación lineal de vectores de la base reducida para aproximar las soluciones. Por último, se entrena y testea la red neuronal.
* Etapa online: Dado un parámetro dentro del espacio de parámetros que queremos hallar el valor de la solución para ese parámetro. Esto solo requiere hacer una única evaluación en la red

En la etapa offline la idea es que no haya nadie delante de la computadora ya que es un proceso bastante largo, depende de la cantidad de snapshots elegidos, pero pueden ser varias horas. La etapa online buscamos que sea casi instantánea lo cual ocurre ya que es solo hacer una única evaluación en la red.

Anexos

Imagen bloque elástico

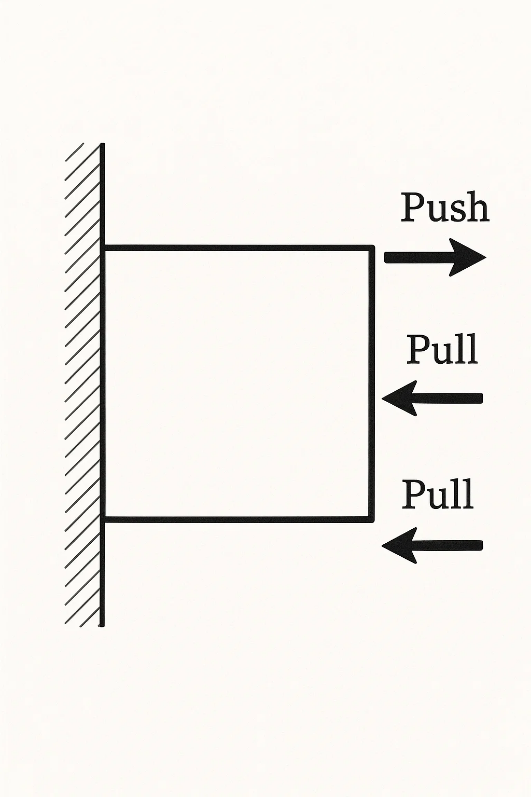
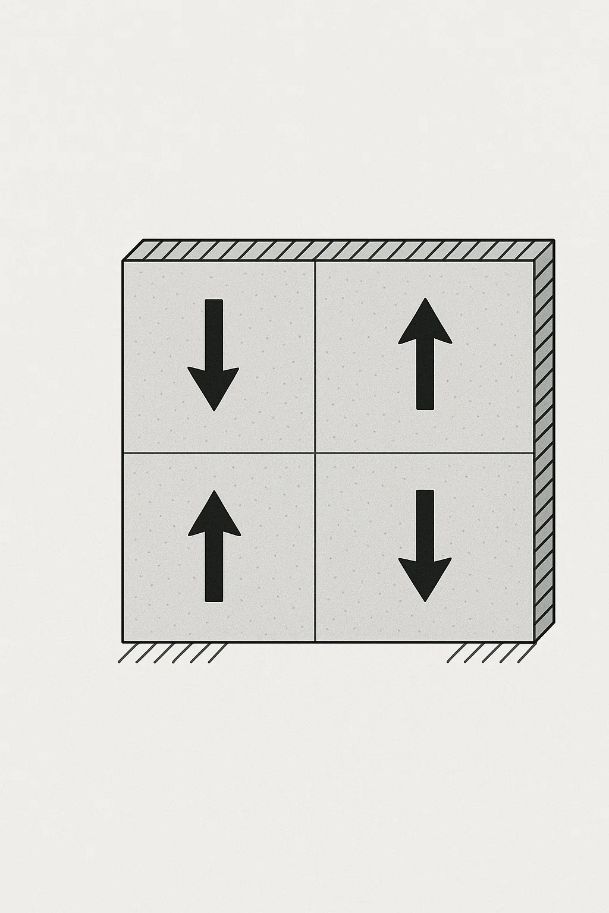


Imagen losa



1. Los snapshots son resultados para cierto parámetro elegido aleatoriamente, dentro del espacio de parametros [↑](#footnote-ref-1)