USC
UNIVERSIDADE DE SANTIAGO DE COMPOSTELA
FACULTADE DE MATEMÁTICAS

Trabajo Fin de Grado

Resolución Numérica de problemas de contorno no lineales con métodos de gradiente conjugado

Julio Suárez Gómez

2020/2021

UNIVERSIDAD DE SANTIAGO DE COMPOSTELA

GRAO DE MATEMÁTICAS

Traballo Fin de Grao

Resolución Numérica de problemas de contorno no lineales con métodos de gradiente conjugado

Julio Suárez Gómez

data presentación

UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Trabajo propuesto

Área de Coñecemento: Matemática Aplicada
Título: Resolución numérica de problemas de contorno no lineales con métodos de gradiente conjugado.
Breve descrición do contido:
Una de las estrategias para abordar la resolución numérica de los problemas de contorno de ecuaciones diferenciales elípticas no lineales es la formulación como un problema de mínimos cuadrados de su discretización con elementos finitos. Esta técnica está descrita en el artículo de Glowinski y Reinhart "Continuation-conjugate gradient methods for the least squares solution of nonlinear boundary value problems". El objetivo de este trabajo es el estudio de dicho artículo y la elaboración de un programa Matlab que resuelva problemas de contorno para ecuaciones diferenciales no lineales con un operador diferencial de tipo elíptico mediante la discretización con elementos finitos, la formulación del problema en mínimos cuadrados y el método de gradiente conjugado precondicionado.
Recomendacións:
Haber superado la materia Métodos Numéricos en Optimización y Ecuaciones Diferenciales y estar matriculado o haber superado Métodos Numéricos en EDP.
Outras observacións: -
-

Índice general

Resumen

En este trabajo estudiaremos el artículo de Glowinski y Reinhart ("Continuation-conjugate gradient methods for the least squares solution of nonlinear boundary value problems") para construir un método de resolución de problemas de optimización no lineales sin restricciones. Para ello realizaremos un discretizado del problema mediante el método de elementos finitos (MEF), y resolveremos el sistema resultante empleando el método de gradiente conjugado. Para este último se seleccionará por su eficiencia y robustez la variante de Polak-Ribière. Tras un estudio teórico de estos métodos se construirá un código en Matlab, a través del cual resolveremos diversos ejemplos, algunos de ellos del propio artículo.

Abstract

In this memory the article of Glowinski and Reinhart ("Continuation-conjugate gradient methods for the least squares solution of nonlinear boundary value problems") will be studied to build a method to solve unconstrained non-linear optimization problems. First, finite elemets method (FEM) is employed to discretize the problem, then conjugate gradient method will be use to solve the system of equations. The solution of conjugate gradient is obtained by using Polak-Ribière variant, due to its eficciency and robustness. Finally, after a theoretical study, some examples will be solved with Matlab code.

Introducción

En este trabajo se realizará un estudio de algunos de los métodos descritos en el artículo de Glowinski y Reinhart ("Continuation-conjugate gradient methods for the least squares solution of nonlinear boundary value problems"). En este se realiza un estudio de diversos problemas conocidos, así como el desarrollo de varios de los métodos empleados para su resolución. En nuestro caso resolveremos únicamente problemas con condición de contorno Dirichlet, aunque el trabajo se podría adaptar a otro tipo de condiciones de contorno.

Para realizar el discretizado del problema se usará uno de los métodos más utilizados para obtener aproximaciones en problemas de ecuaciones diferenciales, el método de los elementos finitos (MEF). Este método fue desarrollado por Richard Courant en la década de los 40, más concretamente para dar solución a algunos problemas problemas concretos de la física. Posteriormente, en los años 50 se establecieron las bases matemáticas del MEF viendo su utilidad en problemas de cálculo de rigidez y deformación en mecánica de estructuras. Gracias al desarrollo de las computadoras y su adaptabilidad a problemas complejos de la vida real, el método de elementos finitos fue cobrando cada vez más importancia en muchos ámbitos de la física o la ingeniería.

Este método será desarrollado en el tercer capítulo, donde buscaremos una solución aproximada del problema en un espacio vectorial de dimensión finita que generaremos con bases de funciones de grado uno. El MEF transformará nuestro problema definido en términos de ecuaciones diferenciales en un problema en forma matricial que proporciona el resultado correcto para un número finito de puntos(nodos) e interpola posteriormente la solución al resto del dominio. Posteriormente se estudiará el error cometido en la aproximación a través de distintas normas: H1,L2 e infinito. Estas normas serán explicadas en el anexo, junto a los métodos de cuadratura em-

pleados para las mismas. En este caso el método elegido ha sido la Cuadratura de Gauss con 2 puntos. Además, se utilizará la Cuadratura de Poncelet para resolver las integrales para la generación del sistema a través del método de elementos finitos.

Para resolver el sistema matricial generado por el método de elementos finitos describiremos en el segundo capítulo un método de gradiente conjugado. A causa de los problemas a resolver y las recomendaciones del artículo junto a otras fuentes (ver bibliografía) se utilizará el método de gradiente conjugado no lineal con la variante de Polak-Ribiere. El método de gradiente conjugado fue propuesto inicialmente por M. Hestenes y E. Stiefel para resolver sistemas de ecuaciones lineales con matriz simétrica y definida positiva.Este es de gran utilidad para resolver sistemas con matriz de coeficientes de orden elevado o matrices con muchos ceros,que no pueden ser tratadas con métodos directos a causa del gran coste computacional de estos mismos.

Como se explica en la sección 2 del trabajo ,este método fue adaptado para funciones no lineales por Fletcher y Reeves, en nuestro caso emplearemos la variante de Polak-Ribière, además de la utilización de métodos de búsqueda inexacta como la Regla de Wolfe-Powell que nos permitirán seleccionar un paso más óptimo en cada iteración del gradiente conjugado, reduciendo así el coste computacional. Además, estudiaremos otra alternativa basándonos en los métodos de interpolación cuadrática, apoyados con la Regla de Goldstein.

A través de una combinación de estos métodos nos permitirá dar solución a varios problemas descritos en la cuarta sección, donde estudiaremos el comportamiento de las normas mencionadas para evaluar el funcionamiento del código implementado en Matlab del Método de elementos finitos junto al gradiente conjugado no lineal. El código del algoritmo principal se adjuntará en el anexo final. Para el código del método de gradiente conjugado se han seguido diversas fuentes [SY06] y [NW06], mientras que para el método de elementos finitos se ha adaptado el programa realizado en la asignatura de Análisis Numérico de Ecuaciones en Derivadas Parciales.

Comenzaremos el trabajo presentando un conjunto de definiciones previas que nos permitirán justificar las características que debe cumplir nuestra función objetivo para garantizar la existencia, convergencia y unicidad de los métodos empleados en la resolución del sistema matricial del capítulo 2. Además, se darán definiciones necesarias para el desarrollo del discretizado del sistema en el tercer capítulo.

Capítulo 1

Resultados y definiciones previas

En este capítulo inicial daremos las definiciones generales para el desarrollo posterior del trabajo. Para ellas, tomaremos como referencia [CCMT89],[SY06] y [BF84].

Definición 1.1. (**Espacio de Hilbert**) Dado un espacio dotado de un producto interior $(V, \langle \cdot, \cdot \rangle)$ decimos que este espacio es de Hilbert si el espacio métrico (V,d) es completo, donde V es un espacio normado ($\|v\| = \langle v, v \rangle^2 \ \forall v \in V$) y la distancia d se define de la siguiente manera:

$$d(x, y) = \|x - y\|, \ \ con \ \ x, y \in V \tag{1.1}$$

Definición 1.2. Definimos el espacio $L^2(a, b) := \{u : (a.b) \to \mathbb{R} \ tal \ que \ \int_a^b u^2(x)dx < \infty\}$. Con su norma correspondiente $\| \cdot \|_{L2} : L^2(a, b) \to \mathbb{R}$ tal que

$$\| \cdot \|_{L2} = \left(\int_a^b u^2(x)dx \right)^{\frac{1}{2}} \tag{1.2}$$

y un producto escalar definido por $\langle \cdot, \cdot \rangle L^2(a, b) \times L^2(a, b) \to \mathbb{R}$ donde

$$\langle u, v \rangle = \int_a^b u(x)v(x) \ dx \tag{1.3}$$

El espacio $L^2(a, b)$ es de Hilbert.

Observación 1.3. Sean u, v dos funciones de $L^2(a, b)$ iguales en casi todo punto, o iguales salvo en un conjunto de medida 0, entonces u, v son consideradas funciones equivalentes en $L^2(a, b)$.

Definición 1.4. (**Espacio de Sobolev**) $H^1(a,b) = \{u(x) \in L^2(a,b) \ : \ u'(x) \in L^2(a,b)\}$. Un espacio de Sobolev también será de Hilbert.

Observación 1.5. Definimos también $H_0^1(a,b) = \{u \in H^1(a,b) \ : \ u(a) = u(b) = 0\}$.

Definición 1.6. (**Desigualdad de Cauchy-Schwarz**). Sean $u, v \in L^2(a,b)$ se cumple la siguiente desigualdad:

$$\int_a^b |u(x)v(x)|dx \leq \left(\int_a^b (u^2(x)dx\right)^{\frac{1}{2}} \left(\int_a^b v^2(x)dx\right)^{\frac{1}{2}} \tag{1.4}$$

Definición 1.7. Dada $\phi : (a,b) \to \mathbb{R}$, *el soporte de* ϕ *se define como* $supp(\phi) = \{x \in (a,b) \ : \ \phi(x) \neq 0\}$

Definición 1.8. Se define $\mathcal{D}(a,b) := \{\phi : (a,b) \to \mathbb{R}$ *tal que* $\phi \in \mathcal{C}^\infty(a,b)$, *donde* $supp(\phi) \subset (a,b)\}$

Definición 1.9. Sea $V \subset \mathbb{R}^n$, se dice que V es convexo si $\forall a, b \in V \ y \ \forall t \in [0,1]$, $(1-t)a + tb \in V$.

Observación 1.10. Es sencillo demostrar que todo espacio de Hilbert es convexo [BF84].

Definición 1.11. (**Convexidad**)
Dado un conjunto convexo $V \subset \mathbb{R}^n$ *con* $V \neq \emptyset$,una función $J : V \to \mathbb{R}$ y un escalar $\alpha \in (0,1)$ se cumple que:

1. J convexa si $J(\alpha x + (1-\alpha)y) \leq \alpha J(x) + (1-\alpha)J(y) \ \ \forall x, y \in V$.
2. J estrictamente convexa si $J(\alpha x + (1-\alpha)y) < \alpha J(x) + (1-\alpha)J(y) \ \ \forall x, y \in V$.
3. J uniformemente convexa si dado $c > 0$ con $c \in \mathbb{R}$ se cumple que $J(\alpha x + (1-\alpha)y) \leq \alpha J(x) + (1-\alpha)J(y) - \frac{1}{2}c\alpha(1-\alpha)\|x-y\|^2 \ \ \forall x, y \in V$.

Teorema 1.12. ***(Relación convexidad - gradiente).***
Dado un conjunto convexo $V \subset \mathbb{R}^n$ *con* $V \neq \emptyset$ *y* $J : V \to \mathbb{R}$ *una función diferenciable se cumple que:*

1. *J convexa* $\iff$ *J(y)* $\geq$ *J(x)* + $\nabla J(x)^T$ *(y-x)* $\forall x, y \in V$.
2. *J estrictamente convexa* $\iff$ *J(y)* $>$ *J(x)* + $\nabla J(x)^T$ *(y-x)* $\forall x, y \in V$, *con* $x \neq y$.
3. *J uniformemente convexa* $\iff$ *dado* $c > 0$, *J(y)* $\geq$ *J(x)* + $\nabla J(x)^T$ *(y-x)*$+\frac{1}{2}c\|y-x\|^2$ $\forall x, y \in V$ y $c \in \mathbb{R}$.

Definición 1.13. Una matriz $A \in \mathbb{R}^{n\times n}$ se dice semi-definida positiva si $\forall v \in \mathbb{R}^n$ se cumple que $v^T Av \geq 0$. De la misma forma, diremos que es definida positiva si la desigualdad anterior es estricta $v^T Av > 0$.

Teorema 1.14. ***(Relación convexidad - hessiana).***
Dado un conjunto convexo $V \subset \mathbb{R}^n$ *con* $V \neq \emptyset$ *y* $J : V \to \mathbb{R}$ *una función dos veces diferenciable se cumple que:*

1. *La matriz hessiana de J es semi-definida positiva* $\forall x \in V \iff$ *J es convexa.*
2. *La matriz hessiana de J es definida positiva* $\forall x \in V \Rightarrow$ *J es estrictamente-convexa.*
3. *La matriz hessiana es uniformemente definida positiva* ($\exists m > 0 / m\|u\|^2 \leq u^T \nabla^2 J(x) u,\ \forall x \in, u \in \mathbb{R}^n$) $\forall x \in V \iff$ *J uniformemente convexa .*

Observación 1.15. En el caso 2. la implicación hacia la izquierda no sería cierta, se puede tomar como contraejemplo la función $J(x) = x^4$ con $x \in \mathbb{R}$, donde $J''(0) = 0$ y por tanto no sería estricta la desigualdad.

Definición 1.16. Un punto $y \in \mathbb{R}^n$ es llamado mínimo global si $f(y) \leq f(x),\ \forall x \in \mathbb{R}^n,$ *con* $x \neq y$. Este mínimo será estricto si la desigualdad también lo es.

Definición 1.17. Dada $J : \mathbb{R}^n \to \mathbb{R}$ una función continua y dos veces diferenciable en $x \in \mathbb{R}^n$ decimos que el vector $d \in \mathbb{R}^n$ es una dirección de descenso de J en x si verifica $\langle \nabla J(x), d \rangle < 0$.

Definición 1.18. Sea V un espacio normado se dice que $J : V \to \mathbb{R}^n$ es una función coercitiva si

$$\lim_{\|x\|_V \to \infty} J(x) = +\infty \tag{1.5}$$

Definición 1.19. Sea $J : V \to \mathbb{R}$ función sobre V un espacio de Hilbert, se dice que J es elíptica si es continuamente diferenciable en V y cumple que

$$\exists \alpha \in \mathbb{R}^+ \ tal \ que \ \langle \nabla J(v) - \nabla J(u), v - u \rangle \geq \alpha \|v - u\|^2 \ \ \forall u, v \in V \tag{1.6}$$

Teorema 1.20. ***(Condición para el mínimo relativo)***
Sea $J : V \subset \mathbb{R}^n$ función dos veces continuamente diferenciable en un abierto V. Un punto $x \in V$ será un mínimo local si y solo sí $\nabla J(x) = 0$ y $\nabla^2 J(x)$ es semidefinida positiva.

Teorema 1.21. *Sea V un conjunto convexo sobre un espacio normado se cumple que:*

1. *Si $J : V \to \mathbb{R}$ es una función convexa con un mínimo relativo en un punto $x \in V$, entonces tendrá un mínimo con respecto a todo el conjunto V.*
2. *Si $J : V \to \mathbb{R}$ es estrictamente convexa con un mínimo relativo en $x \in V$, este conjunto tendrá como máximo un mínimo, además será estricto.*
3. *Dada $J : U \subset V \to \mathbb{R}$ siendo V un conjunto abierto con U subconjunto abierto diferenciable en el punto $x \in U$, entonces J tendrá un mínimo en x con respecto a U si y solo si $J'(x) = 0$.*

Teorema 1.22. *Sea $J : \mathbb{R}^n \to \mathbb{R}$ una función diferenciable y convexa, entonces, x es un mínimo global si y solo si $\nabla J(x) = 0$.*

A continuación, en un primer capítulo, utilizaremos algunos de los resultados anteriores para establecer un algoritmo de resolución del problema de optimización no lineal sin restricciones. Para ello desarrollaremos técnicas de búsqueda inexacta como complementario del método de gradiente conjugado no lineal con la variante de Polak-Ribiére.

Capítulo 2

Métodos de gradiente conjugado con Polak-Ribière

En este capítulo trataremos de dar solución al siguiente problema de optimización sin restricciones :

$$Dada\ J : U \to \mathbb{R}\ encontrar\ u \in U\ tal\ que\ J(u) = \inf_{v \in U} J(v) \tag{2.1}$$

es decir, encontrar el mínimo de J(u) en U.

Para ello trabajaremos durante el capítulo con el conjunto U como un espacio de Hilbert y una función elíptica $J \in \mathcal{C}^4(U)$. Comenzaremos enunciando dos teoremas que nos permitirán demostrar la existencia y unicidad de solución para el problema anterior.

Teorema 2.1. ***(Existencia de solución)***.
Sea $U \subseteq \mathbb{R}^n$ un conjunto no vacío y cerrado con $J : \mathbb{R}^n \to \mathbb{R}$ una función continua, y coercitiva cuando U no está acotado. Entonces existe al menos un elemento $u \in U$ que verifica el problema

$$(P) \quad u \in U\ y\ J(u) = \inf_{v \in U} J(v) \tag{2.2}$$

Demostración. Al ser conjunto no vacío, podemos tomar un punto cualquiera $\tilde{u} \in U$, como J es una función coercitiva existirá $r \in \mathbb{R}$ *tal que* $\|v\| > r$, de este modo $J(\tilde{u}) < J(v)$. En consecuencia, el conjunto de soluciones del problema (P) coincide con el de las soluciones correspondientes al conjunto

$$\tilde{U} = U \cap \{v \in \mathbb{R}^n : \|v\| \leq r\} \tag{2.3}$$

Por tanto tendremos $\tilde{U}$ un conjunto no vacío, cerrado y acotado. Uniendo esto a que J es una función continua garantizaremos la existencia de mínimo.

□

Lema 2.2. *Una función elíptica $J : U \to \mathbb{R}$ es estrictamente convexa y coercitiva.*

Demostración. Tomamos $u, v \in U$, dado que la función es elíptica es continuamente diferenciable, por tanto podremos escribir bajo la fórmula de Taylor (ver [CCMT89] pag.228-229)

$$\begin{aligned} J(v) - J(u) = \int_0^1 \langle \nabla J(u + t(v-u)), v-u\rangle dt = \langle \nabla J(u), v-u\rangle + \\ \int_0^1 \langle \nabla J(u + t(v-u)) - \nabla J(u), v-u\rangle dt > \langle \nabla J(u), v-u\rangle. \end{aligned} \tag{2.4}$$

De esta desigualdad obtenemos que la función es estrictamente convexa, pues

$$J(v) > J(u) + \langle \nabla J(u), v-u\rangle, \ \forall u, v \in U, \ con\ u \neq v. \tag{2.5}$$

Además, también vemos que es coercitiva, ya que

$$\begin{aligned} \langle \nabla J(u), v-u\rangle + \int_0^1 \langle \nabla J(u + t(v-u)) - \nabla J(u), v-u\rangle dt \geq \\ \langle \nabla J(u), v-u\rangle + \int_0^1 \alpha t \|v-u\|^2 dt = \langle \nabla J(u), v-u\rangle + \frac{\alpha}{2}\|v-u\|^2. \end{aligned} \tag{2.6}$$

por tanto,

$$J(v) \geq J(0) + \langle \nabla J(0), v\rangle \frac{\alpha}{2}\|v\|^2 \geq J(0) - \|\nabla J(0)\|\|v\| + \frac{\alpha}{2}\|v\|^2. \tag{2.7}$$

□

Teorema 2.3. ***(Unicidad de solución).***
Sea V un espacio de Hilbert y $J : U \to \mathbb{R}$ una función elíptica, el problema

$$(P) \quad u \in U \ y \ J(u) = \inf_{v \in U} J(v) \tag{2.8}$$

tiene solución única.

Demostración. La existencia de la solución en el problema (P) viene dada por el teorema anterior, ya que una función elíptica es coercitiva. Además, como una función elíptica es estrictamente convexa se obtiene la unicidad de la solución. □

A continuación comenzaremos describiendo las estrategias de búsqueda lineal. Estas serán necesarias para resolver el problema optimización en una dimensión, que está presente en casi cualquier método de resolución de problemas de optimización sin restricciones. Por tanto, será fundamental obtener un proceso eficiente de resolución de dichos problemas 1D para la resolución posterior mediante el método de gradiente conjugado.

2.1. Introducción a la Búsqueda Lineal

En las estrategias de búsqueda lineal estableceremos una ecuación de búsqueda para aproximarnos a nuestro óptimo lo máximo posible y en el menor número de pasos posibles. Nuestra fórmula general para cada iteración vendrá dada por la siguiente expresión:

$$x_{k+1} = x_k + \alpha_k \cdot d_k \tag{2.9}$$

donde α_k será un escalar positivo que definirá el paso y d_k la dirección de descenso en la iteración k. Por tanto, el método consistirá en escoger una dirección de descenso apropiada y movernos en esa misma dirección obteniendo valores x_k cada vez más próximos a la solución. En cada iteración, será fundamental hallar el paso α minimizando la expresión anterior.

Como podemos observar, el éxito de este algoritmo depende directamente de las selecciones de la dirección de descenso y del paso apropiados. Estudiaremos primero como seleccionar dicho paso.

2.2. Selección de la longitud del paso

Para seleccionar el paso de forma exacta deberíamos escoger el mínimo de la función $\phi(\alpha)$ con $\alpha > 0$,

$$\phi(\alpha) = J(x_k + \alpha \cdot d_k) \tag{2.10}$$

pero este proceso podría ser excesivamente costoso, sobre todo si el punto inicial está lejos de la solución, dado que se requerirán de muchas evaluaciones de la función objetivo J y de su gradiente. Por estas cuestiones estudiaremos un proceso de búsqueda inexacta, donde intentaremos aproximar el mínimo sin calcular ∇J.

Para comenzar, discutiremos las condiciones que tendrá que cumplir la función objetivo para que cada etapa de iteración sea un avance hacia el mínimo. La primera y obvia de las condiciones es que la función objetivo J se debe ir reduciendo, es decir $J(x_{k+1}) < J(x_k)$ o lo que es lo mismo $J(x_{k+1}) < J(x_k + \alpha_k \cdot d_k)$. Aun así, está condición no garantiza la convergencia al mínimo, pues se podría tener una secuencia de iterantes de la forma $J(x_k) = 1/k$ y que el mínimo fuera negativo, por lo que la reducción de esta función sería insuficiente.

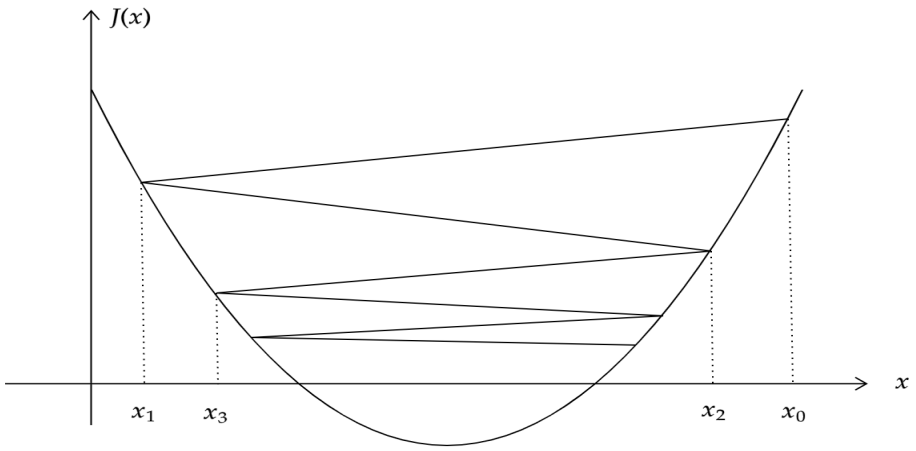

Figura 2.1: Descenso insuficiente de J

A continuación describiremos algunas condiciones para garantizar un decrecimiento suficiente.

2.2.1. Búsqueda inexacta: Condiciones de Goldstein y Wolfe-Powell

Como hemos mencionado anteriormente el proceso de búsqueda lineal puede ser a veces demasiado costoso, esto se debe mayoritariamente a tres factores: la propia función, la dirección de descenso o el iterante inicial seleccionado. Para ello, enunciaremos una condición que garantizará el decrecimiento suficiente de J en el algoritmo, y así evitar los problemas relacionados con la reducción de la función objetivo.

$$J(x_k + \alpha_k d_k) \leq J(x_k) + c_1 \alpha_k \nabla (J_k)^T d_k \quad , con\ c_1 \in (0,1)\ constante. \tag{2.11}$$

Nótese que para $c_1 = 1$ se tiene el lado derecho como una aproximación por Taylor de primer orden en el punto inicial. La condición anterior es conocida como **Condición de Armijo**. Esta condición que nos permite acotar el valor del nuevo punto $J(x_k + \alpha_k d_k)$ por debajo del valor previo, ya que c_1 es positivo y por lo tanto el lado derecho de la desigualdad nos muestra un valor inferior a $J(x_k)$.

Sin embargo, la condición de Armijo provoca que la reducción sea proporcional al tamaño del paso, lo cual funciona muy bien para pasos grandes, pero podría ser ineficiente para pasos pequeños no llegando a la convergencia. Por lo tanto, es necesario añadir una nueva restricción sobre el paso.

Para solucionar este problema distinguiremos entre dos tipos de condiciones, dividiendo así en dos métodos alternativos: **La Regla de Goldstein y el Método de Wolfe-Powell**. Estos últimos se diferenciarán entre sí por el uso de la derivada, lo que hará a cada método útil en su contexto.

2.2.2. Regla de Wolfe-Powell

Para obtener el método de Wolfe-Powell definiremos la **Condición de curvatura**, que exigirá a α_k cumplir la siguiente desigualdad:

$$\nabla J(x_k + \alpha_k d_k)^T d_k \geq c_2 \nabla J_k^T d_k \quad , con\ c_2 \in (c_1, 1)\ constante. \tag{2.12}$$

Vemos que el lado izquierdo es $\phi'(\alpha_k)$, por tanto podemos ver que lo que se busca es que la pendiente sea c_2 veces mayor en el nuevo punto de la iteración. Esto tiene

sentido porque $\phi'(\alpha_k)$ es claramente negativa, si no lo fuera, podría ser un buen indicativo de que debemos seleccionar otra dirección de descenso.

Por tanto las **Condiciones de Wolfe-Powell** será el conjunto de las condiciones (2.11) y (2.12). Podemos observar estas condiciones en la Figura 2.2.

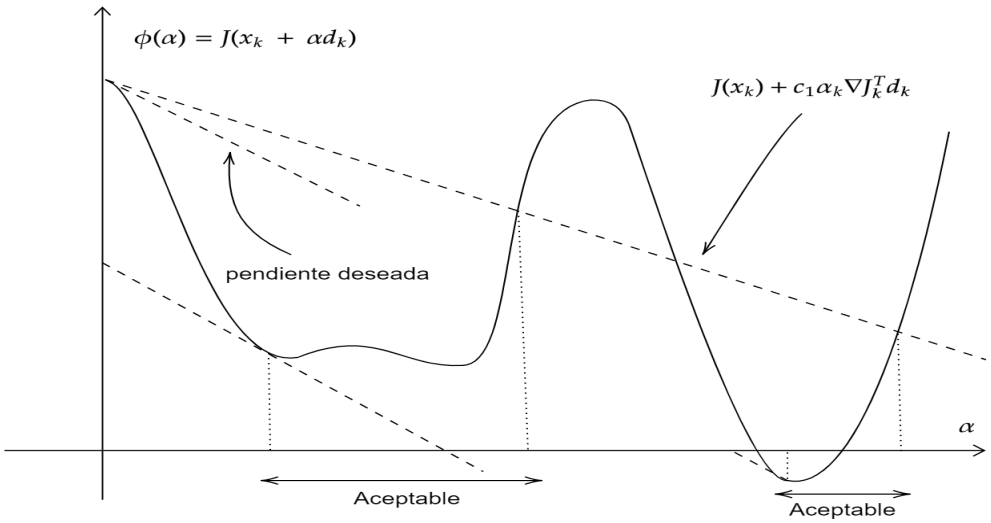

Figura 2.2: Condiciones Wolfe-Powell

Es interesante mencionar que una longitud de paso puede satisfacer las condiciones Wolfe-Powell sin estar próxima a la solución del problema $\min_\alpha \phi(\alpha) = J(x_k + \alpha d_k)$, sin embargo se puede modificar la condición de curvatura del siguiente modo

$$|\nabla J(x_k + \alpha_k d_k)^T d_k| \geq c_2 |\nabla J_k^T d_k| \tag{2.13}$$

En este caso, forzamos a que α_k pertenezca a una vecindad del mínimo o un punto estacionario de ϕ. De este modo, no permitimos que la derivada de $\phi(\alpha_k)$ sea muy grande, es decir, excluimos los puntos que están lejos de los puntos estacionarios de ϕ. Al conjunto de las condiciones (2.11) y (2.13) les llamaremos **Condiciones fuertes de Wolfe-Powell**.

Se puede consultar el Anexo (7.1) para observar que existen longitudes de paso que satisfacen las condiciones de Wolfe-Powell para una función J con las condiciones requeridas, y, además el algoritmo es convergente.

2.2.3. Regla de Armijo-Goldstein

Como vimos anteriormente, la regla de Wolfe necesita del cálculo de ∇J_k^T para cada paso α, esto en algunos casos puede resultar muy costoso, por ello introduciremos la conocida **Regla de Golstein**, que tendrá el mismo objetivo pero sin utilizar el cálculo del gradiente anterior en cada iteración.

$$J(x_k + \alpha_k d_k) \geq J(x_k) + c_2 \alpha_k \nabla (J_k)^T d_k \tag{2.14}$$

Esta condición añadida a la condición de Armijo (2.11) proporcionarán la regla para la selección del paso. En este caso se tendrá que cumplir que $0 < c_1 < c_2 < 1$. Podemos ver a continuación en (3.2) una representación gráfica.

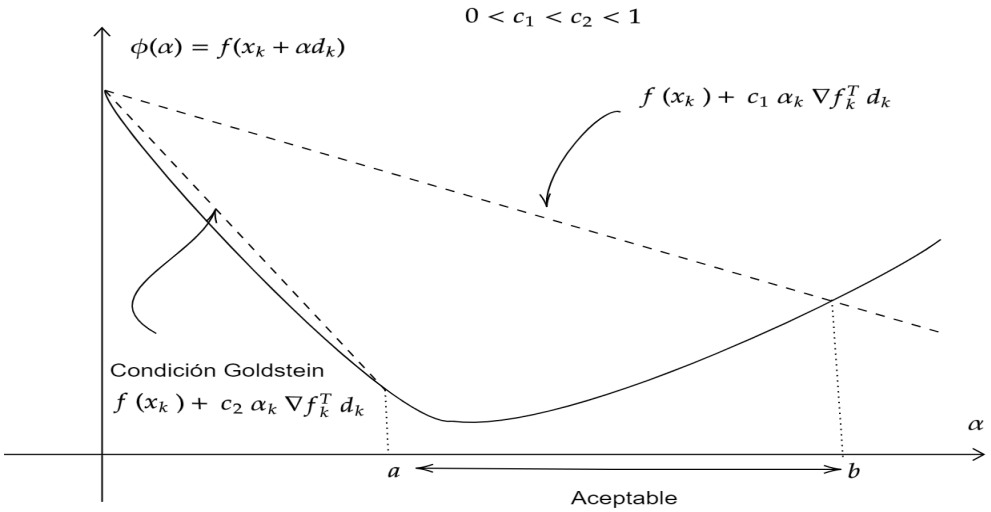

Figura 2.3: Condiciones de Goldstein

Como podemos observar, nos quedaremos con un intervalo $l = [a, b]$ de regiones aceptables para seleccionar el paso. Del mismo modo que para el método de Wolfe-Powell, se pueden consultar los teoremas de convergencia en el Anexo (7.1).

Implementación

Los dos métodos de búsqueda de α - óptimo mostrados seguirán el mismo esquema de computación:

Algoritmo 2.4. *Búsqueda con regla de Wolfe-Powell o Goldstein*

- *Paso 0: Se da una tolerancia* $\varepsilon > 0$ *y un punto inicial* $x_0 \in \mathbb{R}^n$.
- *Paso 1: Comprobación: si* $\|\nabla J(x_k)\| \leq \varepsilon$ *el algoritmo se detiene. En otro caso, buscamos una nueva dirección de descenso* d_k *que verifique* $d_k^T \nabla J(x_k) < 0$.
- *Paso 2: Búsqueda del* α *- óptimo con regla de Goldstein o regla de Wolfe-Powell.*
- *Paso 3: Ajustamos la ecuación monodimensional con los nuevos valores* $x_{k+1} = x_k + \alpha_k d_k$ *y* $k = k + 1$. *Volvemos al Paso 1.*

Por lo general, los procedimientos de búsqueda lineal requieren una estimación inicial α_0 y generan una secuencia α_i que finaliza con una longitud de paso que satisface las condiciones mostradas anteriormente, o por el contrario, determina que dicha longitud de paso no existe. Por tanto estos procedimientos se pueden dividir en dos partes: una fase de horquillado que encuentra un intervalo $[a, b]$ que contiene longitudes de paso aceptables, y una fase de selección dentro del intervalo previo que determina el "paso final".

En nuestro caso combinaremos el método de interpolación cuadrática con la regla de Goldstein anteriormente descrita para realizar un método eficiente.

2.2.4. Método de interpolación cuadrática con tres puntos

Los métodos de interpolación polinómica pretenden aproximar una función $\phi(\alpha)$ por un polinomio definido por diversas evaluaciones en la función que queremos aproximar. En nuestro caso, nos interesará realizarlo sin utilizar la derivada, por tanto, para la interpolación cuadrática necesitaremos 3 puntos distintos : $\alpha_1, \alpha_2, \alpha_3$ y sus valores correspondientes en la función $\phi(\alpha_1), \phi(\alpha_2), \phi(\alpha_3)$. Generalmente se escogen dos puntos que generen un intervalo donde pueda estar contenido el mínimo y, más adelante, un tercer punto dentro de ese intervalo. Las condiciones requeridas para construir nuestro polinomio de interpolación son:

$$q(\alpha_i) = a\alpha_i^2 + b\alpha_i + c = \phi(\alpha_i) \quad , \; i = 1, 2, 3. \tag{2.15}$$

Resolviendo estas ecuaciones podemos obtener a, b de forma que nos permita calcular trivialmente el mínimo de la función polinómica cuadrática, que se obtiene en el punto $-\frac{b}{2a}$. Entonces, primero, a través de (2.15) obtenemos los siguientes valores:

$$a = -\frac{(\alpha_2 - \alpha_3)\phi_1 + (\alpha_3 - \alpha_1)\phi_2 + (\alpha_1 - \alpha_2)\phi_3}{(\alpha_1 - \alpha_2)(\alpha_2 - \alpha_3)(\alpha_3 - \alpha_1)} \tag{2.16}$$

$$b = \frac{(\alpha_2^2 - \alpha_3^2)\phi_1 + (\alpha_3^2 - \alpha_1^2)\phi_2 + (\alpha_1^2 - \alpha_2^2)\phi_3}{(\alpha_1 - \alpha_2)(\alpha_2 - \alpha_3)(\alpha_3 - \alpha_1)} \tag{2.17}$$

Ahora calculamos el mínimo de la función polinómica cuadrática con la fórmula del vértice:

$$\bar{\alpha} = \frac{-b}{2a} = \frac{1}{2}\frac{(\alpha_2^2 - \alpha_3^2)\phi_1 + (\alpha_3^2 - \alpha_1^2)\phi_2 + (\alpha_1^2 - \alpha_2^2)J_3}{(\alpha_2 - \alpha_1)\phi_1 + (\alpha_3 - \alpha_1)J_2 + (\alpha_1 - \alpha_2)\phi_3} \tag{2.18}$$

$$= \frac{1}{2}(\alpha_1 + \alpha_2) + \frac{1}{2}\frac{(\phi_1 - \phi_2)(\alpha_2 - \alpha_3)(\alpha_3 - \alpha_1)}{(\alpha_2 - \alpha_1)\phi_1 + (\alpha_3 - \alpha_1)\phi_2 + (\alpha_1 - \alpha_2)J_3} \tag{2.19}$$

Finalmente obtenemos $\bar{\alpha}$ el mínimo de la función cuadrática con la que aproximamos ϕ. El algoritmo consistirá en repetir este proceso repetidas veces hasta cumplir la condición de convergencia.

Algoritmo 2.5. *Interpolación cuadrática utilizando tres puntos*

- *Paso 0: Dada una tolerancia $\varepsilon > 0$. Buscar un grupo $\{\alpha_1, \alpha_2, \alpha_3\}$ conteniendo a $\alpha*$ (óptimo). Computar $\phi_1 - \phi_2$ (2.15).*
- *Paso 1: Calcular $\bar{\alpha}$ usando la fórmula (2.18).*
- *Paso 2: Validar los datos, si $(\bar{\alpha} - \alpha_1)(\bar{\alpha} - \alpha_3) \geq 0$ entonces debemos volver al paso 1, construir un nuevo conjunto $\{\alpha_1, \alpha_2, \alpha_3\}$ y el mínimo $\bar{\alpha}$, ya que este último está fuera del intervalo principal $[\alpha_1, \alpha_3$.*
- *Paso 3: Si $|\bar{\alpha} - \alpha_2| < 0$ finalizamos. En otro caso, volvemos al Paso 1.*

A continuación nos centraremos en el método de gradiente conjugado no lineal, en concreto con la variante de Polak-Ribière. Este fue introducido para problemas no lineales en la década de los 60 por Fletcher y Reeves. En esta época tuvo un gran éxito gracias a su utilidad para resolver todo tipo de problemas de optimización no lineales, con el paso del tiempo se han ido añadiendo nuevas variantes del método, muchas con gran éxito en la práctica.

2.3. Métodos de direcciones conjugadas

La base principal de los métodos de gradiente conjugado son las llamadas direcciones conjugadas, que son indispensables para seleccionar la dirección de descenso apropiada en cada etapa k.

Definición 2.6 (Direcciones conjugadas)**.** Dada $G \in \mathcal{M}_{n\times m}$ simétrica y definida positiva, $d_1, d_2, ..., d_m \in \mathbb{R}^n$ vectores distintos de cero siendo $m \leq n$. Decimos que los vectores son G-conjugados si

$$d_i^T G d_j = 0, \forall i \neq j \tag{2.20}$$

Es claro que los vectores satisfacen la propiedad de independencia lineal, además se consideran como una generalización de las direcciones ortogonales ya que si $G = I$ la conjugación será equivalente a la ortogonalidad.

Un esquema del funcionamiento del método del método podría ser el siguiente, extraído de [SY06]. (Notamos que durante este trabajo se utilizará la notación $g(x) = \nabla J(x)$ utilizada en la referencia anterior).

Algoritmo 2.7. *Método General de direcciones conjugadas*

- *Paso 0: Dado un punto inicial x_0(iteración k=0), $\varepsilon > 0$, $g_0 = g(x_0)$. Definimos d_0 satisfaciendo $d_0^T < 0$.*

- *Paso 1: Si $||g_k|| \leq \varepsilon$, finalizamos.*

- *Paso 3: Definimos α_k tal que $J(x_k+\alpha_k \cdot d_k) = \min_{\alpha\geq 0} J(x_k+\alpha \cdot d_k)$. Ajustamos la iteración como $x_{k+1} = x_k + \alpha_k d_k$*

- *Paso 3: Seleccionamos las direcciones conjugadas en el método satisfaciendo la propiedad $d_{k+1}^T G d_j = 0, j = 0, 1, ..., k$.*

- *Paso 4: Ajustamos $k = k + 1$ y volvemos al Paso 1.*

2.4. Método de gradiente conjugado lineal

Para presentar un método de gradiente conjugado no lineal nos centraremos primero en la construcción del método para el caso lineal y posteriormente adaptarlo. El método de gradiente conjugado lineal se fundamentará en la resolución del sistema

$$Ax = b \tag{2.21}$$

donde A es una matriz nxn simétrica y definida positiva. Es sencillo deducir la equivalencia de (2.21) con el siguiente problema de minimización

$$\text{mín}\, \phi(x) = \frac{1}{2}x^T Ax - b^T x \tag{2.22}$$

Podemos interpretar el método de gradiente conjugado como una resolución de sistemas lineales o una técnica para minimizar funciones convexas cuadráticas. Por tanto, acorde a nuestra notación, definimos

$$J(x) = \frac{1}{2}x^T Gx + b^T x + c \tag{2.23}$$

Donde G es una matriz nxn, simétrica y definida positiva, $b \in \mathbb{R}^n$ y $c \in \mathbb{R}$. Además definiremos su gradiente como

$$g(x) = Gx + b \tag{2.24}$$

En consecuencia se define su dirección de descenso con el gradiente $d_0 = -g_0$,y se calcula α_0 mediante una búsqueda lineal

$$x_1 = x_0 + \alpha_0 d_0 \tag{2.25}$$

Teniendo en cuenta (2.20) podemos definir la siguiente dirección conjugada del siguiente modo

$$d_1 = -g_1 + \beta_0 d_0 \tag{2.26}$$

Multiplicando la ecuación anterior por $d_0^T G$ podemos deducir el valor de β_0

$$\beta_0 = \frac{g_1^T G d_0}{d_0^T G d_0} = \frac{g_1^T (g_1 - g_0)}{d_0^T (g_1 - g_0)} = \frac{g_1^T g_1}{g_0^T g_0} \tag{2.27}$$

Podemos entonces generalizar el método y definir para cada iteración k

$$d_k = -g_k + \sum_{i=1}^{k-1} \beta_i d_i \tag{2.28}$$

Escogiendo los β_i del mismo modo que (2.27) tendremos con $j = 0, 1, ..., k-1$

$$\beta_j = \frac{g_k^T G d_j}{d_j^T G d_j} = \frac{g_k^T (g_{j+1} - g_j)}{d_j^T (g_{j+1} - g_j)} \tag{2.29}$$

De esta forma

$$\beta_j = 0, j = 0, 1, ..., k-2 \tag{2.30}$$

$$\beta_{k-1} = \frac{g_k^T g_k}{g_{k-1}^T g_{k-1}} \tag{2.31}$$

Estableciendo por tanto el método como

$$x_{k+1} = x_k + \alpha_k d_k \tag{2.32}$$

$$d_k = -g_k + \beta_{k-1} d_{k-1} \tag{2.33}$$

Donde β_{k-1} se calcula mediante la fórmula de Fletcher-Reeves 2.31 y el paso, en particular del caso cuadrático, será

$$\alpha_k = \frac{-g_k^T d_k}{d_k^T G d_k} \tag{2.34}$$

Como podemos observar, en los métodos de gradiente conjugado es fundamental seleccionar un buen β_{k-1}, en nuestro caso introduciremos la fórmula de Polak-Ribière(PR) con la finalidad de optimizar dicho algoritmo para el caso lineal y no lineal. La fórmula de PR se considera mucho más eficiente ya que tiene la propiedad de **reinicio automático**, es decir, si el algoritmo va muy lento $\beta_k \approx 0$, propiedad que en el caso de FR no tenemos. Notamos que nuestro α_k vendrá delimitado por las técnicas de búsqueda lineal exacta e inexacta del los anteriores capítulos.

Cabe mencionar a continuación la propiedad más importante que cumple el gradiente conjugado para el caso cuadrático, donde se garantiza una convergencia muy rápida.

Teorema 2.8. *(Propiedad del método de gradiente conjugado lineal) Para una función cuadrática definida positiva (2.23), el método de gradiente conjugado descrito por la fórmula de Fletcher-Reeves (2.31) y (2.32) con búsqueda lineal exacta converge en $m \leq n$ pasos. Además, si el número de autovalores distintos de G es igual a m, entonces se cumplen las siguientes propiedades $\forall i \in \{0, ..., m\}$:*

$$d_i^T G d_j = 0, \quad j = 0, 1..., i-1 \tag{2.35}$$

$$g_i^T g_j = 0, \;\; j = 0, 1..., i-1 \tag{2.36}$$

$$d_i^T g_i = -g_i^T g_i, \;\; j = 0, 1..., i-1 \tag{2.37}$$

$$[g_0, g_1, ..., g_i] = [g_0, Gg_0, ..., G^i g_0] \tag{2.38}$$

$$[d_0, d_1, ..., d_i] = [g_0, Gg_0, ..., G^i g_0] \tag{2.39}$$

Demostración. La demostración se puede consultar en [SY06]. □

2.5. Método de gradiente conjugado no lineal

Ahora adaptaremos el método para el caso no lineal siendo J una función objetivo cualquiera. Para ello realizaremos algunos cambios sobre el algoritmo que describimos previamente para el caso lineal.

Método de búsqueda inexacta no lineal de Polak-Ribiére

Algoritmo 2.9. *GC-PR*

- *Paso 1:* *Dado x_0 un punto inicial, evaluamos $J(x_0) = J_0$ y $\nabla J_0 = \nabla f(x_0)$.*
- *Paso 2:* *Definimos de la misma manera que en el método lineal la dirección de descenso inicial $d_0 \leftarrow -\nabla J_0$ y la etapa $k \leftarrow 0$.*
- *Paso 3:* *Comprobamos que $\nabla J_k \neq 0$.*
- *Paso 4:* *Realizamos la siguiente etapa computando α_k y $x_{k+1} = x_k + \alpha_k \cdot d_k$.*
- *Paso 5:* *Evaluamos de nuevo ∇J_{k+1} y calculamos β_{k+1} mediante la fórmula de Polak-Ribiére para el caso no lineal. De esta forma escribimos*

$$\beta_{k+1}^{PR} \leftarrow \frac{\nabla J_{k+1}^T (\nabla J_{k+1} - \nabla J_k)}{||\nabla J_k||^2} \tag{2.40}$$

$$d_{k+1} \leftarrow -\nabla J_{k+1} + \beta_{k+1} d_k \tag{2.41}$$

$$k \leftarrow k + 1 \tag{2.42}$$

Si escogemos J como una función convexa cuadrática y α_k como el mínimo exacto, el algoritmo previo se trasforma en el algoritmo del caso lineal. Si se quiere ver más detalles sobre el código consultar [Vog02].

Para completar el algoritmo anterior, debemos especificar cómo seleccionamos el paso α_k. Para ello debemos recordar que para asegurar un descenso suficiente en el método estos términos α_k deben satisfacer ciertas condiciones.

Gracias a la fórmula del producto escalar obtenemos:

$$\nabla J_k^T d_k = -\|\nabla J_k\|^2 + \beta_k^{PR} \nabla J_k^T d_{k-1} \tag{2.43}$$

Si la búsqueda lineal es **exacta**, como α_{k-1} es un mínimo local de J en la dirección d_{k-1} tenemos que $\nabla J_k^T d_{k-1}$=0, por tanto, por (2.43) tenemos que $\nabla J_k^T d_k < 0$ y entonces d_k es una dirección de descenso.

En el caso de que la búsqueda sea **inexacta** el término positivo del lado derecho de la ecuación (2.43) tiene más peso, por tanto $\nabla J_k^T d_k < 0$. Para corregir esta situación debemos utilizar las anteriormente explicadas **Condiciones fuertes de Wolfe-Powell**.

$$J(x_k + \alpha_k p_k) \leq J(x_k) + c_1 \alpha_k \nabla J_k^T d_k \tag{2.44}$$

$$\|\nabla J(x_k + \alpha_k p_k)^T p_k\| \leq -c_2 \nabla J_k^T d_k \tag{2.45}$$

Donde $0 < c_1 < c_2 < \frac{1}{2}$.

Convergencia del método de Gradiente conjugado no lineal con Polak-Ribère

Como mencionamos anteriormente, este algoritmo tiene una peculiaridad que lo diferencia de FR, el reinicio automático. Esto es, si el algoritmo es demasiado lento y $\nabla J_{k+1} \approx \nabla J_k$, entonces la fórmula provoca que $\beta_k \approx 0$ y por tanto $d_{k+1} \approx -\nabla J_{k+1}$. Por tanto, esta propiedad es muy útil para evitar una tendencia lenta en el algoritmo. Se puede observar más concretamente en [Sha78].

A continuación, enunciaremos los teoremas de convergencia global para el método de Polak-Ribère, al igual que en la propia tesis [Rei80] demostraremos los resultados utilizando búsqueda exacta.

Lema 2.10. *Dada una función J tal que $\nabla J(x)$ es uniformemente continua en nuestro subconjunto $S \subseteq U$ y considerando θ_k como el ángulo entre la dirección de descenso d_k y $-\nabla J(x_k)$. Si se cumple*

$$\theta_k \leq \frac{\pi}{2} - \mu, \quad con\ \mu > 0, \tag{2.46}$$

entonces se tiene que $\nabla J(x_k) = 0$ para un cierto k, o $J(x_k) \to \infty$ o $\nabla J(x_k) \to 0$.

Demostración. La prueba de este resultado se puede ver más detalladamente en [A+20]. □

Teorema 2.11. *(Teorema del Valor Medio). Dada una función $J : \mathbb{R}^n \to \mathbb{R}$ se tiene que para cualquier vector $d \in \mathbb{R}^n$,*

$$J(x + \alpha d) = J(x) + \nabla J(x + \alpha d)^T d,\ \alpha \in (0, 1). \tag{2.47}$$

Ahora estaremos en condiciones óptimas para demostrar el teorema de convergencia global.

Teorema 2.12. *Sea $J : \mathbb{R}^n \to \mathbb{R}$ una función dos veces diferenciable en nuestro conjunto subconjunto $S \subseteq U$ acotado. Asumiendo que existe una constante positiva $m > 0$ que verifica para todo $x \in S,\ y \in \mathbb{R}^n$*

$$m\|y\|^2 \leq y^T \nabla^2 J(x) y \tag{2.48}$$

Entonces la secuencia de $\{x_k\}$ generada por el método de Polak-Ribiére con búsqueda lineal exacta converge a un único mínimo $x*$ de la función J.

Demostración. Por el lema anterior será suficiente probar (2.46), es decir, que existe una constante positiva $\omega > 0$ que verifique la siguiente expresión

$$-g_{k+1}^T d_{k+1} \geq \omega \|g_{k+1}\| \|d_{k+1}\| \tag{2.49}$$

esto es, $\cos\theta_k \geq \omega \geq 0$. Por tanto, con el lema mencionado se observa que $g_k \to 0$ y $g(x*) = 0$. De (2.48) se sigue que $\{x_k\} \to x*$, mínimo, el cual es único

en la función J. A causa de que es búsqueda lineal exacta, se puede deducir de $d_{k+1} = -g_{k+1} + \beta_k d_k$ y $g_{k+1}^T d_k = 0$ por tanto $g_{k+1}^T d_{k+1} = -\|g_{k+1}\|^2$. Entonces la expresión 2.49 será equivalente a

$$\frac{\|g_{k+1}\|}{\|d_{k+1}\|} \geq \omega \tag{2.50}$$

Recordamos que el valor de α_k se obtiene de

$$\alpha_k = -\frac{g_k^T d_k}{d_k^t A_k d_k} = \frac{\|g_k\|^2}{d_k^t A_k d_k} \tag{2.51}$$

donde

$$A_k = \int_0^1 \nabla^2 J(x_k + t\alpha_k d_k) dt \tag{2.52}$$

Utilizando el Teorema del Valor Medio, se puede deducir de la fórmula anterior lo siguiente

$$g_{k+1} - g_k = \nabla J(x_k + \alpha_k d_k) - \nabla J(x_k) = \alpha_k A_k d_k. \tag{2.53}$$

De esta forma la fórmula de β_k en el caso de PR se puede expresar

$$\beta_k^{PR} = \frac{g_{k+1}^T (g_{k+1} - g_k)}{g_k^T g_k} = \alpha_k \frac{g_{k+1}^T A d_k}{\|g_k\|^2} = \frac{g_{k+1}^T A_k d_k}{d_k^T A_k d_k} \tag{2.54}$$

Por tanto, como el conjunto S es acotado, existe $M > 0$ constante tal que para todo $x \in S$ y para cualquier $y \in \mathbb{R}^n$

$$y^T A(x) y \leq M\|y\|^2. \tag{2.55}$$

De este modo podemos establecer la siguiente cota

$$|\beta_k^{PR}| \leq \frac{\|g_{k+1}\|\|A_k d_k\|}{m\|d_k\|^2} \leq \frac{M}{m}\frac{\|g_{k+1}\|}{\|d_k\|} \tag{2.56}$$

Además,

$$\|d_{k+1}\| \leq \|g_{k+1}\| + |\beta_k^{PR}|\|d_k\| \leq \|d_{k+1}\| + \frac{M}{m}\|g_{k+1}\| = (1 + \frac{M}{m})\|g_{k+1}\|, \tag{2.57}$$

esto es,

$$\frac{\|g_{k+1}\|}{\|d_{k+1}\|} \geq \left(1 + \frac{M}{m}\right)^{-1} \tag{2.58}$$

De este modo, solo tendremos que tomar $\omega = \frac{m}{m+M}$ cumpliendo así 2.50.

□

La experiencia práctica nos indica que PR tiende a confirmar que es un algoritmo más robusto y eficiente que Fletcher-Reeves u otras fórmulas para resolver problemas de optimización sin restricciones, de ahí su gran interés en la práctica. En lo relativo a la búsqueda inexacta, es importante mencionar que desde el punto de vista teórico este no garantiza que d_k sea dirección de descenso con las condiciones fuertes de Wolfe-Powell (2.11) y (2.13).

Como en la programación del algoritmo utilizaremos PR con búsqueda lineal inexacta mencionaremos la siguiente formulación alternativa de β mencionada en varias de las fuentes consultadas para la realización de este trabajo.

$$\beta_{k+1} = \text{máx}\,\{\beta_{k+1}^{PR}, 0\} \tag{2.59}$$

Con esta sugerencia, propuesta ya por Powell en 1984, se desarrolla en [Noc92] para concluir la convergencia global de Polak-Ribière con las condiciones fuertes de Wolfe-Powell, es decir, con búsqueda inexacta. De esta forma se consigue alcanzar las propiedades de convergencia no lineal que cumplen otros métodos menos robustos como Fletcher-Reeves (estas se pueden ver en [SY06]).

En el siguiente capítulo desarrollaremos el Método de Elementos Finitos, el cual complementaremos con los métodos de Gradiente Conjugado descritos en esta sección para obtener un algoritmo que resuelva eficientemente los problemas de optimización no lineales sin restricciones.

Capítulo 3

Resolución de problemas elípticos 1D mediante el método de elementos finitos

3.1. Introducción

En esta sección estudiaremos uno de los métodos de discretizado más usados para resolver problemas de optimización en forma continua, usualmente asociados a campos de la física o la ingeniería, entre otros. El método que estudiaremos se conoce como el método de los elementos finitos (MEF), que desde la aparición de las computadoras ha gozado de un éxito más que notable para el estudio de problemas de optimización. Como mencionamos anteriormente, el MEF permite transformar un problema de optimización continuo en uno discreto, conociendo la forma aproximada de una función a través de un número finito de puntos.

Notación 3.1. Durante este tema, para simplificar utilizaremos la notación $u'(x) \equiv \frac{du(x)}{dx}$.

3.2. Descripción del MEF 1D

3.2.1. Introducción al problema de Sturm-Liouville

En este apartado resolveremos con el MEF el problema de Sturm-Liouville de segundo orden. Comenzaremos resolviendo el problema para unas condiciones de contorno Dirichlet, es decir, considerando el siguiente problema $(\mathcal{P})$

$$\begin{gathered}\text{Dados } \alpha, \beta \in \mathbb{R} \text{ y } f \in L^2(a,b) \\ p \in H^1(a,b),\ q \in L^\infty(a,b),\ p(x) \geq \alpha > 0,\ q(x) \geq 0,\ \forall x \in [a,b] \\ \text{determinar } u \in H^2(a,b) \textit{ que verifique}\end{gathered}$$

$$(\mathcal{P}) \begin{cases} -(p(x)u'(x))' + q(x)u(x) = f(x) \textit{ en } (a,b) \\ u(a) = \alpha, \quad u(b) = \beta \end{cases}$$

3.2.2. Formulaciones diferencial y variacional del problema

Ahora consideraremos una función $v \in D(a,b)$ que sea escalar, de clase infinito y con $supp(v) \subset (a,b)$ un compacto, entonces v(a)=v(b)=0. Si multiplicamos la igualdad diferencial del problema $(\mathcal{P})$ por v obtendremos la siguiente ecuación:

$$-(p(x)u'(x))'v(x) + q(x)u(x)v(x) \ = \ f(x)v(x) \tag{3.1}$$

Realizamos las operaciones e integramos en (a,b)

$$\int_a^b -(p(x)u'(x))'v(x) + q(x)u(x)v(x)dx \ = \ \int_a^b f(x)v(x)dx \textit{ en } (a,b) \tag{3.2}$$

Utilizando la linealidad de la integral

$$-\int_a^b (p(x)u'(x))'v(x)dx + \int_a^b q(x)u(x)v(x)dx \ = \ \int_a^b f(x)v(x)dx \textit{ en } (a,b) \tag{3.3}$$

Ahora, podemos integrar por partes el primer sumando tomando $u = v \rightarrow du = v'dx$ *y* $dv = (pu')' \rightarrow v = pu'$

$$-\int_a^b (p(x)u'(x))'v(x)dx = -v(x)(p(x)u'(x))]_a^b - \int_a^b -p(x)u'(x)v'(x)dx \tag{3.4}$$

Como v(a)=v(b)=0 se obtiene

$$= \int_a^b p(x)u'(x)v'(x)dx \tag{3.5}$$

Observación 3.2. Para un v con las condiciones seleccionadas anteriormente y una función $g \in \mathcal{L}^\infty$ se cumple la siguiente equivalencia $g(x) = 0 \ en \ (a,b) \iff \int_a^b g(x)v(x)dx = 0 \ \forall v$, entonces podremos formular el problema $(\mathcal{P})$ de dos formas equivalentes.

Para el resolver el problema $(\mathcal{P})$ para condiciones tipo Dirichlet enunciaremos las formulaciones diferencial (o clásica)$(\mathcal{D})$ y variacional $(\mathcal{V})$ que enunciaremos a continuación, para posteriormente realizar el proceso de discretización.

Definición 3.3. (**Formulación diferencial o clásica[Dirichlet]**)

$$(\mathcal{D}) \begin{cases} Determinar \ u \in H^1(a,b) \ tal \ que \ u(a) = \alpha, \ u(b) = \beta, \ verificando \\ \int_a^b p(x)u'(x)v'(x)dx + \int_a^b q(x)u(x)v(x)dx = \int_a^b f(x)v(x)dx \ \forall v \in H_0^1(a,b) \end{cases}$$

Definición 3.4. (**Formulación variacional[Dirichlet]**)

$$(\mathcal{V}) \begin{cases} -(p(x)u'(x))' + q(x)u(x) = f(x) \ en \ (a,b) \\ u(a) = \alpha, \quad u(b) = \beta \\ \int_a^b (-(p(x)u'(x))' + q(x)u(x) - f(x))v(x)dx = 0 \ \forall v \ tal \ que \ v(a) = v(b) = 0 \end{cases}$$

3.2.3. Discretización del problema variacional

Consideraremos una malla en $[a,b]$ formada por N elementos $T_i = [a_i, a_{i+1}]$ que forman una partición del intervalo

$$\bigcup_{i=1}^{N} T_i = [a,b]$$
$$a = a_1 < a_2 < ... < a_N < a_{N+1} = b$$
$$h_i = a_{i+1} - a_i = \text{longitud de } T_i, \ h = \max_{1 \le i \le N} h_i$$
$$i \in \{1, ..., N\}$$

Definición 3.5. Se define el espacio vectorial de polinomios de grado $\le k$ como $\mathbb{P}_k := \langle 1, x, x^2, ..., x^k \rangle, k \in \mathbb{N}. (\dim \mathbb{P}_k = k+1)$

Definición 3.6. Se define el espacio de funciones $\mathcal{X}_h^k := \{v_h \in \mathcal{C}'([a,b]) \ : \ v_h|_{T_i} \in \mathbb{P}_k \ para \ i = 1, ..., N\}$. Además, $\dim \mathcal{X}_h^k = kN + 1$.

Ejemplo 3.7. A continuación, podemos ver un ejemplo concreto para el espacio anterior con **k=2**.De este modo tendremos que $\dim \mathcal{X}_h^2 = N + 1 + N = 2N + 1$.

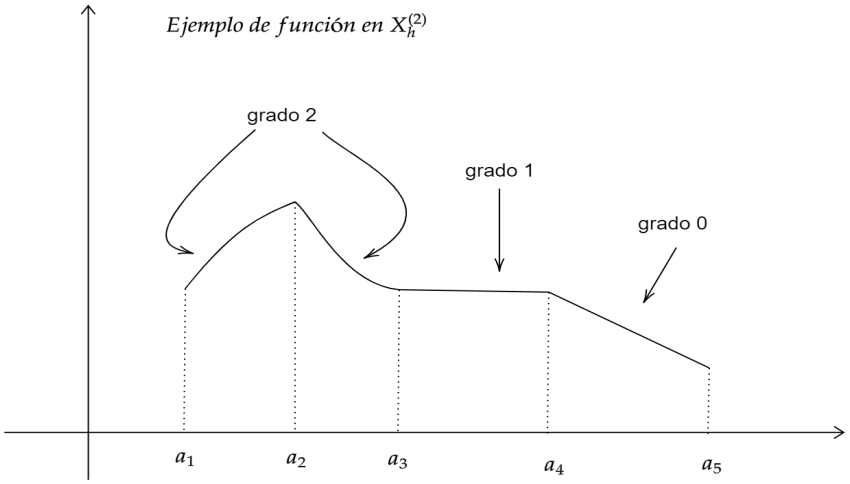

Figura 3.1: Ejemplo para K=2

Definición 3.8. Se define el espacio de funciones $\mathcal{V}_h^k := \{v_h \in \mathcal{X}_h^{(k)} \ : \ v(a) = v(b) = 0\}$. Además, $\dim \mathcal{V}_h^k = \dim \mathcal{X}_h^k - 2 = kN - 1$.

Observación 3.9. Resulta bastante claro que $\mathcal{X}_h^k \subset H^1(a,b)$, luego $\mathcal{X}_h^k$ es un subespacio vectorial de dimensión finita de $H^1(a,b)$.

Por tanto la discretización del problema $(\mathcal{V})$ nos transforma la expresión en la siguiente:

Definición 3.10.

$$(\mathcal{V}_\langle) \begin{cases} Hallar \ u_h(x) \in \mathcal{X}_h^k \ tal \ que \ u_h(a) = \alpha, \ u_h(b) = \beta \\ \underbrace{\int_a^b p(x)u_h'(x)v_h'(x)dx + \int_a^b q(x)u_h(x)v_h(x)dx}_{A(u_h,v_h) \ bilineal} = \underbrace{\int_a^b f(x)v_h(x)dx}_{L(v_h) \ lineal} \ \forall v_h(x) \in \mathcal{V}_h^k \end{cases}$$

3.2.4. Resolución del método de elementos finitos Langrange para K=1

Sea $\langle w_1, w_2, ..., w_{N+1} \rangle$ la base de $\mathcal{X}_h^{(1)}$ tal que, para cada j=1,...,N+1 se tiene:

$$\begin{cases} w_j \in \mathcal{X}_h^{(1)} \\ w_j(a_i) = \delta_{ij} \ para \ i = 1, ..., N+1 \end{cases}$$

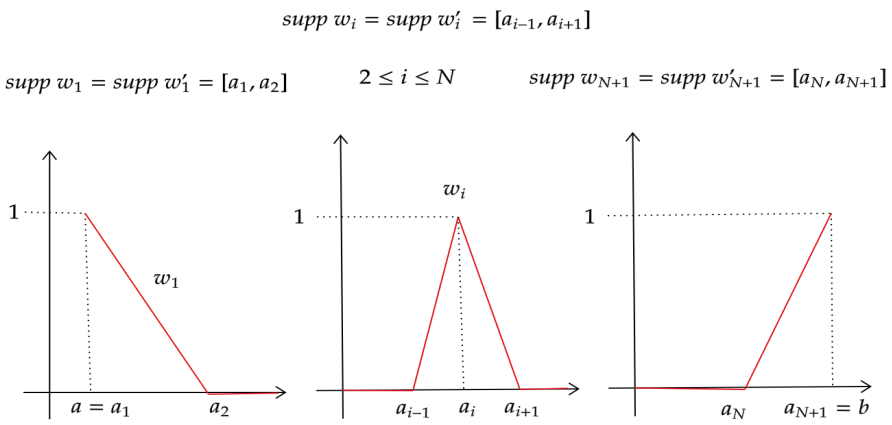

Figura 3.2: Base de w_i

Observación 3.11. Podemos observar fácilmente que si $v_h \in \mathcal{X}_h^{(1)}$, entonces $v_h(x) = \sum_{i=1}^{N+1} \varepsilon_i w_i(x) \ \forall x \in [a, b], \ con \ \varepsilon_i = v_h(a_i)$ para todo $i = 1, ..., N+1$. (*Nota* : $\langle w_2, ..., w_N \rangle$ es base de $\mathcal{V}_h^{(1)}$).

Definimos a continuación la formulación variacional del problema utilizando la discretización anterior, es decir, seleccionando

$$u_h = \sum_{j=1}^{N+1} u_h(a_j) w_j \ y \ v_h = \sum_{i=1}^{N+1} v_h(a_i) w_i \tag{3.6}$$

De este modo podemos renombrar la formulación del problema de la siguiente forma:

Definición 3.12. (**Formulación variacional del problema discreto Lagrange** $\mathbb{P}_1$[Dirichlet])

$$(\mathcal{V}_h^{(1)}) \begin{cases} Encontrar\ u_h(x) \in \mathcal{X}_h^k\ tal\ que\ u_h(a) = \alpha,\ u_h(b) = \beta \\ A(u_h, v_h) = L(v_h)\ \forall v_h \in \mathcal{V}_h^{(1)} \end{cases}$$

Utilizando (3.6) este problema será equivalente al siguiente:

$$(\mathcal{V}_h^{(1)}) \begin{cases} Encontrar\ u_h(x) \in \mathcal{X}_h^k\ tal\ que\ u_h(a) = \alpha,\ u_h(b) = \beta \\ \sum_{i=1}^{N+1}\sum_{j=1}^{N+1} v_h(a_i)A(w_j, w_i)u_h(a_j) \ = \ \sum_{i=1}^{N+1} v_h(a_i)L(w_i)\ \forall v_h \in \mathcal{V}_h^{(1)} \end{cases}$$

Observación 3.13. $A(u_h, v_h) = A(\sum_{j=1}^{N+1} u_h(a_j)w_j, \sum_{i=1}^{N+1} v_h(a_i)w_i), L(v_h) = L(\sum_{i=1}^{N+1} v_h(a_i w_i)))$

Definición 3.14. Si se definen $M \in \mathcal{M}_{NxN}(\mathbb{R}), \vec{u}, \vec{v} \in \mathbb{R}^N$ se cumple que $\vec{v}^T M \vec{u} \ = \ \sum_{i=1}^{N}\sum_{j=1}^{N} v_i m_{ij} u_j$

Basándonos de esta definición podremos definir el problema matricial a partir del problema variacional, denotando de la siguiente forma:

$$\vec{u} = (u_h(a_1), ..., u_h(a_{N+1}))^T, \vec{v} = (v_h(a_1), ..., v_h(a_{N+1}))^T$$

$$\vec{b} = (L(w_1), ..., L(w_{N+1}))^T,\ y\ A = (a_{ij})\ con\ 1 \leq i.j \leq N+1,\ con\ a_{ij} = A(w_j, w_i)$$

Observación 3.15. Cabe notar que A es una matriz tridiagonal y simétrica, por tanto no importa el cambio de orden $A(u, v) = A(v, u)$.

Definición 3.16. Formulación matricial

$$\begin{cases} Encontrar\ \vec{u} \in \mathbb{R}^{N+1}\ tal\ que\ u_1 = \alpha,\ u_{N+1} = \beta \\ \vec{v}^T A\vec{u} \ = \ \vec{v}^T\vec{b}\ \forall \vec{v} \in \mathbb{R}^{N+1}\ tal\ que\ v_1 = v_{n+1} = 0 \end{cases}$$

Observación 3.17. Recordamos que es equivalente resolver el problema matricial anterior $\vec{v}^T A\vec{u} \ = \ \vec{v}^T\vec{b}\ \forall \vec{v} \in \mathbb{R}^{N+1} \iff A\vec{u} = \vec{b}$.

En [Rav83] se encuentran los teoremas y demostraciones más relevantes de este método. A continuación formalizaremos todo el proceso por pasos para la resolución del problema Lagrange $\mathbb{P}_1$.

MEF Lagrange $\mathbb{P}_1$

Primero, aclararemos algunas definiciones para describir el proceso de forma adecuada

Definición 3.18. Se definen los grados de libertad (nodos) asociados al elemento T_i como $x_1^{(i)} = a_i$ y $x_2^{(i)} = a_{i+1}$

Definición 3.19. Se definen los polinomios de base asociados al elemento T_i como $w_1^{(i)} = w_{i|T_i}$ y $w_2^{(i)} = w_{i+1|T_i}$

Definición 3.20. Definimos la función $B^{(i)} : \mathbb{R}^{N+1} \longrightarrow \mathbb{R}^2$ donde llevamos $\vec{v} \longrightarrow B^{(i)}\vec{v} = \vec{v}^{(i)}$.

como *matriz de cambio* $B^{(i)} = \begin{pmatrix} 0 & 0 & ... & 0 & 1 & 0 & ... & 0 \\ 0 & 0 & ... & 0 & 0 & 1 & ... & 0 \end{pmatrix}$

En definitiva, $(B^{(i)})_{jk} = \delta_{i+j-1,k},\ 1 \leq j \leq 2,\ 1 \leq k \leq N+1$

Ahora definiremos paso a paso las operaciones correspondientes para la discretización y el ensamblado de la matriz y el vector de segundos miembros

- PASO 1: Tenemos el problema en formulación variacional descrito anteriormente

$$(\mathcal{V}_h^{(1)}) \begin{cases} \textit{Encontrar } u_h(x) \in \mathcal{X}_h^k \textit{ tal que } u_h(a) = \alpha,\ u_h(b) = \beta \\ A(u_h, v_h) = L(v_h)\ \forall v_h \in \mathcal{V}_h^{(1)} \end{cases}$$

- PASO 2: Ahora discretizamos el anterior problema teniendo en cuenta los intervalos T_i (i=1,...,N) y los N+1 nodos, de la misma forma que hicimos en la definición previa. De esta forma, definimos

$$A(u_h, v_h) = \sum_{i=1}^{N} \int_{T_i} p(x)(u_{h|T_i})'(x)(v_{h|T_i})'(x)dx + \int_{T_i} q(x)(u_{h|T_i})(x)(v_{h|T_i})(x)dx$$

$$L(v_h) = \sum_{i=1}^{N} \int_{T_i} f(x)(v_{h|T_i})(x)dx$$

- PASO 3: Expresión en base local. Partimos de los elementos la base w_i definida anteriormente, a continuación estableceremos un par de definiciones

  A continuación veremos como desarrollamos las expresiones del Paso 2:

$$(v_{h|T_i})(x) \;=\; v_h(a_i)w_i(x) + v_h(a_{i+1})w_{i+1}(x) \;=\; v_h(x_1^{(i)})w_1^{(i)}(x) + v_h(x_2^{(i)})w_2^{(i)}(x)$$

En consecuencia tenemos que

$$(v_{h|T_i})(x) \;=\; \sum_{j=1}^{2} v_h^{(i)}(x_j^{(i)})w_j^{(i)}(x) \;=$$

$$= \begin{pmatrix} w_1^{(i)}(x) & w_2^{(i)}(x) \end{pmatrix} \begin{pmatrix} v_h^{(1)}(x_j^{(1)}) \\ v_h^{(2)}(x_j^{(2)}) \end{pmatrix} = (\vec{w}_{(x)}^{(i)})^T \vec{v}^{(i)} = \vec{w}_{(x)}^{(i)} \cdot \vec{v}^{(i)}$$

Análogamente se puede realizar $(v_{h|T_i})'(x) = (D\vec{w}_x^{(i)})^T \vec{v}^{(i)}$, $(u_{h|T_i})(x) = (\vec{w}_{(x)}^{(i)})^T \vec{u}^{(i)}$ y $(u_{h|T_i})'(x) = (D\vec{w}_x^{(i)})^T \vec{u}^{(i)}$.

- PASO 4: Deducimos los términos relacionados con la matriz A.

$$(u_{h|T_i})' \cdot (v_{h|T_i})' \;=\; (\vec{v}^{(i)})^T D\vec{w}^{(i)} (D\vec{w}^{(i)})^T \vec{u}^{(i)}$$

$$(u_{h|T_i}) \cdot (v_{h|T_i}) \;=\; (\vec{v}^{(i)})^T \vec{w}^{(i)} (\vec{w}^{(i)})^T \vec{u}^{(i)}$$

- PASO 5: Deducimos los términos relacionados con el vector del segundo miembro $\vec{b}$:

$$(v_{h|T_i}) \;=\; (\vec{v}^{(i)})^T \vec{w}^{(i)}$$

- PASO 6: Ahora reescribimos la expresión del paso 2 con los elementos calculados.

$$A(u_h, v_h) =$$
$$\sum_{i=1}^{N} (\vec{v}^{(i)})^T \{ \textstyle\int_{T_i} [D\vec{w}_{(x)}^{(i)} p(x) (D\vec{w}_{(x)}^{(i)})^T] dx + \int_{T_i} [w_{(x)}^{(i)} q(x) (w_{(x)}^{(i)})^T] dx \} (\vec{u}^{(i)})$$

Dividiremos esta expresión en dos partes para cada (i), la primera denominada matriz de rigidez, y la segunda matriz de masa $A^{(i)} \;=\; A_R^{(i)} + A_M^{(i)}$, siendo las $A^{(i)}$ matrices elementales.

- PASO 7: Expresión de $A(u_h, v_h)$ y $L(v_h)$ en función de los grados de libertad globales.

- PASO 8: Reescribimos $A(u_h, v_h)$ y $L(v_h)$ utilizando la función $B^{(i)}$

$$A(u_h, v_h) = \sum_{i=1}^{N} (\vec{v}^{(i)})^T A^i \vec{u}^{(i)} = \vec{v}^T \{\sum_{i=1}^{N} (B^{(i)})^T A^{(i)} B^{(i)}\} \vec{u} = \vec{v}^T A \vec{u}$$

$$L(v_h) = \sum_{i=1}^{N} (\vec{v}^{(i)})^T \vec{b}^i = \vec{v}^T \{\sum_{i=1}^{N} (B^{(i)})^T \vec{b}^{(i)}\} = \vec{v}^T \vec{b}$$

- PASO 9: Ahora ya tenemos todos los elementos necesarios para proceder al ensamblado de la matriz A y del vector del segundo miembro $\vec{b}$. Comenzaremos por A

$$A = \sum_{i=1}^{N} (B^{(i)})^T A^{(i)} B^{(i)} \in \mathcal{M}_{N+1}(\mathbb{R}),$$

para $1 \leq j, k \leq N+1$ y $i + l - 1 = j$ y $i + m - 1 = k$ se desarrolla

$$[(B^{(i)})^T A^{(i)} B^{(i)}]_{jk} = \sum_{l=1}^{2} [(B^{(i)})^T]_{jl} [A^{(i)} B^{(i)}]_{lk} =$$
$$\sum_{l=1}^{2} (B^{(i)})_{lj} (\sum_{m=1}^{2} (A^{(i)})_{lm} (B^{(i)})_{mk}) = \sum_{l=1}^{2} \sum_{m=1}^{2} (B^{(i)})_{lm} (A^{(i)})_{lm} (B^{(i)})_{mk} =$$
$$\sum_{l=1}^{2} \sum_{m=1}^{2} \delta_{i+l-1,j} (A^{(i)})_{lm} \delta_{i+m-1,k} =$$

$$= \begin{cases} (A^{(i)})_{[j-i+1],[k-i+1]}, & si\ 1 \leq j-i+1 \leq 2 \ \ y \ \ 1 \leq k-i+1 \leq 2 \\ 0, & en\ otro\ caso \end{cases}$$

Ensamblamos el vector $\vec{b}$ de la misma forma

$$\vec{b} = \sum_{i=1}^{N} (B^{(i)})^T \vec{b}^{(i)}$$

$$[(B^{(i)})^T \vec{b}^{(i)}]_j = \sum_{k=1}^{2} [(B^{(i)})^T]_{jk} (\vec{b}^{(i)})_k = \sum_{k=1}^{2} (B^{(i)})_{kj} (\vec{b}^{(i)})_k = \sum_{k=1}^{2} \delta_{i+k-1,j} (\vec{b}^{(i)})_k =$$

$$= \begin{cases} (\vec{b}^{(i)})_{j-i+1}, & si\ 1 \leq j-i+1 \leq 2 \\ 0, & en\ otro\ caso \end{cases}$$

- PASO 10: Cálculo de las integrales para la resolución del problema:

  Teniendo en cuenta lo desarrollado anteriormente podemos escribir las matrices elementales de Masa y Rigidez en función de dos integrales, por tanto, será preciso calcularlas para realizar el MEF.

$$A_M^{(i)} = \int_{T_i} \vec{w}^{(i)}(x) q(x) (\vec{w}^{(i)}(x))^T dx, \quad \textit{matriz elemental de masa} \tag{3.7}$$

$$A_R^{(i)} = \int_{T_i} D\vec{w}^{(i)}(x) p(x) (D\vec{w}^{(i)}(x))^T dx, \quad \textit{matriz elemental de rigidez} \tag{3.8}$$

  A continuación, transformamos las expresiones anteriores a partir de las definiciones de las funciones vectoriales del Paso 3

$$A_M^{(i)} = \int_{T_i} w_j^{(i)}(x) q(x) (w_k^{(i)}(x))^T dx, \quad \textit{matriz elemental de masa} \tag{3.9}$$

$$A_R^{(i)} = \int_{T_i} (w_j^{(i)})'(x) p(x) (w_k^{(i)})'(x) dx, \quad \textit{matriz elemental de rigidez} \tag{3.10}$$

  Ahora, para calcular estas matrices estableceremos un elemento referencia $\hat{T} = [0,1]$ y una transformación afín $\varphi_i : \ [0,1] \to [a_i, a_{i+1}] = T_i$ *tal que* $\varphi_i(\hat{x}) = \hat{h_i} \cdot \hat{x} + a_i$. Además, las funciones base para este elemento referencia se pueden expresar a través de las funciones definidas previamente

  Proposición 3.21.

$$\hat{w}_j(\hat{x}) = w_j^{(i)}(\varphi_i(\hat{x})) \quad (1 \leq j \leq 2 \quad , \quad 1 \leq i \leq N). \tag{3.11}$$

  Demostración. Para probar la igualdad anterior es suficiente ver que $w_j^{(i)} \circ \varphi_i$ es un polinomio de grado 1 que vale δ_{jk} en $\hat{x}_k$ $(1 \leq k \leq 2)$.

  Como

$$\hat{w}_j = w_j^{(i)} \circ \varphi_i \tag{3.12}$$

En consecuencia

$$(\hat{w}_j)'(\hat{x}) = (w_j^{(i)})'(\varphi_i(\hat{x}))h_i \tag{3.13}$$

$$(\hat{w}_j)'(\varphi_i(\hat{x})) = \frac{1}{h_i}(\hat{w}_j^{(i)})'(\hat{x}) \tag{3.14}$$

entonces, se tiene que para la matriz de masa

$$\begin{aligned}\int_{T_i}(w_j^{(i)}(x)q(x)w_k^{(i)}(x))dx \underbrace{=}_{x=\varphi_i(\hat{x})} h_i\int_0^1(w_j^{(i)}(\varphi_i(\hat{x}))q(\varphi_i(\hat{x}))w_k^{(i)}(\varphi_i(\hat{x})))d\hat{x} = \\ = h_i\int_0^1(\underbrace{\hat{w}_j(\hat{x})}_{pol.\ de\ grado\ 1}\quad q(\varphi_i(\hat{x}))\quad \underbrace{\hat{w}_k(\hat{x})}_{pol.\ de\ grado\ 1})\ dx\end{aligned} \tag{3.15}$$

De equivalente forma, realizamos el mismo desarrollo para la matriz de rigidez

$$\begin{aligned}\int_{T_i}((w_j^{(i)})'(x)\ \ p(x)\ \ (w_k^{(i)})'(x))\ dx \underbrace{=}_{(3,13)(3,14)} \\ = \frac{1}{h_i}\int_0^1(\underbrace{(\hat{w}_j)'(\hat{x})}_{pol.\ de\ grado\ 0}\quad p(\varphi_i(\hat{x}))\quad \underbrace{(\hat{w}_k)'(\hat{x})}_{pol.\ de\ grado\ 0})\ dx\end{aligned} \tag{3.16}$$

□

Observación 3.22. Si se desease calcular estas integrales de forma exacta será necesario que p y q cumplan condiciones muy restrictivas para los problemas a desarrollar, por ello, aproximaremos estas mediante fórmulas de cuadratura. Como en nuestro caso tratamos de resolver el Método de Elementos Finitos para Lagrange $\mathbb{P}_1$, sería suficiente usar fórmulas exactas en $\mathbb{P}_0$. En nuestro caso utilizaremos la fórmula de cuadratura de Poncelet para calcular estas integrales.

Definición 3.23. (Fórmula del punto medio o de Poncelet.)

$$\int_a^b f(x)dx \approx (b-a)f(x_0),\ con\ x_0\ =\ \frac{a+b}{2} \tag{3.17}$$

Cálculo de $A^{(i)} = A_R^{(i)} + A_M^{(i)}$

Como hemos desarrollado anteriormente, se deduce que

$$(A_R^{(i)})_{ij} = \frac{1}{h_1}\int_0^1 (\hat{w}_i)'(\hat{x})p(\varphi_1(\hat{x}))(\hat{w}_j)'(\hat{x})d\hat{x},\ donde\ h_1 = a_2 - a_1,\ \varphi_1(\hat{x}) = h_1\hat{x} + a_1 \tag{3.18}$$

Por tanto,

$$\begin{aligned}(A_R^{(i)})_{11} &= \frac{1}{h_1}\int_0^1 \underbrace{(\hat{w}_1)'(\hat{x})}_{-1} p(\varphi_1(\hat{x}))\underbrace{(\hat{w}_1)'(\hat{x})}_{-1} d\hat{x} = \\ &= \frac{1}{h_1}\int_0^1 p(\varphi_1(\hat{x}))d\hat{x} \underbrace{\approx}_{(3,17)} \frac{1}{h_1}p(\varphi_1(\frac{1}{2})) = \frac{1}{h_1}p(x_m^{(1)})\end{aligned} \tag{3.19}$$

Del mismo modo se puede desarrollar para el resto de puntos de la matriz elemental $A_R^{(i)}$:

$$\begin{aligned}(A_R^{(i)})_{12} &= \frac{1}{h_1}\int_0^1 \underbrace{(\hat{w}_1)'(\hat{x})}_{-1} p(\varphi_1(\hat{x}))\underbrace{(\hat{w}_2)'(\hat{x})}_{+1} d\hat{x} = \\ &= -\frac{1}{h_1}\int_0^1 p(\varphi_1(\hat{x}))d\hat{x} \underbrace{\approx}_{(3,17)} -\frac{1}{h_1}p(\varphi_1(\frac{1}{2})) = -\frac{1}{h_1}p(x_m^{(1)})\end{aligned} \tag{3.20}$$

$$\begin{aligned}(A_R^{(i)})_{22} &= \frac{1}{h_1}\int_0^1 \underbrace{(\hat{w}_2)'(\hat{x})}_{+1} p(\varphi_1(\hat{x}))\underbrace{(\hat{w}_2)'(\hat{x})}_{+1} d\hat{x} = \\ &= \frac{1}{h_1}\int_0^1 p(\varphi_1(\hat{x}))d\hat{x} \underbrace{\approx}_{(3,17)} \frac{1}{h_1}p(\varphi_1(\frac{1}{2})) = \frac{1}{h_1}p(x_m^{(1)})\end{aligned} \tag{3.21}$$

En definitiva, tenemos la matriz elemental resultante

$$A_R^{(i)} \approx \frac{p(x_m^{(i)})}{h_i}\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} \tag{3.22}$$

Notación 3.24. Escribimos el punto medio del intervalo T_i como $x_m^{(i)} = \frac{x_1^{(i)} + x_2^{(i)}}{2}$.

Hallaremos la matriz de masa elemental $A_M^{(1)}$ bajo el mismo método realizado para la matriz elemental de rigidez.

$$(A_M^{(i)})_{ij} = h_1 \int_0^1 \hat{w}_i(\hat{x}) q(\varphi_1(\hat{x})) \hat{w}_j(\hat{x}) d\hat{x} \underbrace{\approx}_{(3,17)} h_1 \hat{w}_i(\frac{1}{2}) q(\varphi_i(\frac{1}{2})) \hat{w}_j(\frac{1}{2}) = \underbrace{=}_{\hat{w}_i(\frac{1}{2})=\frac{1}{2}, \hat{w}_j(\frac{1}{2})=\frac{1}{2}} \frac{h_1}{4} q(x_m^{(1)}) \tag{3.23}$$

En definitiva, realizando los cálculos obtenemos la matriz elemental resultante

$$A_M^{(1)} \approx \frac{h_1 q(x_m^{(1)})}{4} \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \tag{3.24}$$

Análogamente, para cualquier $A^{(i)}$ se tiene que

$$A^{(i)} = A_M^{(i)} + A_R^{(i)}, \ con \ 2 \leq i \leq N \tag{3.25}$$

$$A_R^{(i)} \approx \frac{p(x_m^{(i)})}{h_i} \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} \tag{3.26}$$

$$A_M^{(i)} \approx \frac{h_i q(x_m^{(i)})}{4} \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \tag{3.27}$$

Cálculo del vector $b^{\vec{(i)}}$ para $1 \leq i \leq N$

Sabemos que

$$\vec{b}^{(i)} = \int_{T_i} \vec{w}^{(i)}(x) f(x) dx \tag{3.28}$$

Por tanto, dado $j = 1, 2$ podemos desarrollar por componentes del vector elemental al igual que en la matriz elementar $A^{(i)}$

$$b_j^{(i)} = \int_{T_i} w_j^{(i)}(x) f(x) dx \underbrace{=}_{x=\varphi_i(\hat{x})} h_i \int_0^1 \hat{w}_j(\hat{x}) f(\varphi(\hat{x})) d\hat{x} \approx \underbrace{\approx}_{Poncelet} h_i \underbrace{\hat{w}_j}_{\frac{1}{2}}(\frac{1}{2}) f(x_m^{(i)}) = \frac{h_i f(x_m^{(i)})}{2} \tag{3.29}$$

En consecuencia,

$$\vec{b}^{(i)} \approx \frac{h_i f(x_m^{(i)})}{2} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \tag{3.30}$$

3.2.5. Implementación de las condiciones de contorno

Resolvemos el problema enunciado mediante la formulación variacional

$$\begin{cases} -(p(x)u'(x))' + q(x)u(x) = f(x) \ en \ (a,b) \\ L_1 u(a) + L_2 u'(a) = \alpha, \quad L_3 u(b) + L_4 u'(b) = \beta \end{cases}$$

Para abordar la implementación de las condiciones de contorno subdividiremos por casos los valores que podrán tomar los L_i $i = 1, 2, 3, 4$.

- Caso $L_1, L_3 \neq 0$ y $L_2, L_4 = 0$. (DIRICHLET-DIRICHLET) En este caso, el problema a resolver vendrá dado por

$$\begin{cases} Hallar \ u \in H^1(a,b) \ tal \ que \ u(a) = \frac{\alpha}{L_1}, \ u(b) = \frac{\beta}{L_3} \\ \int_a^b p(x)u'(x)v'(x)dx + \int_a^b q(x)u(x)v(x)dx = \int_a^b f(x)v(x), \ \forall v \in H_0^1(a,b) \end{cases}$$

Mediante el ensamblado se obtuvo

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & 0 & . & . & . & . & 0 \\ a_{2,1} & a_{2,2} & a_{2,3} & 0 & . & . & . & 0 \\ 0 & a_{3,2} & a_{3,3} & a_{3,4} & . & . & . & 0 \\ . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . \\ 0 & . & . & . & . & a_{N,N-1} & a_{N,N} & a_{N,N+1} \\ 0 & . & . & . & . & 0 & a_{N+1,N} & a_{N+1,N+1} \end{pmatrix} \quad \vec{b} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ . \\ . \\ . \\ b_N \\ b_{N+1} \end{pmatrix}$$

Por consiguiente los cambios que habrá que realizar sobre A y $\vec{b}$ son

$$a_{1,1} = 1, \ a_{1,2} = 0, \ a_{N+1,N} = 0, \ a_{N+1,N+1} = 1$$

$$b_1 = \frac{\alpha}{L_1}, \ b_{N+1} = \frac{\beta}{L_3}$$

Con esto, obtenemos el sistema matricial resultante a través del método de elementos finitos con las condiciones de contorno indicadas, en nuestro caso serán Dirichlet. Para esta resolución matricial será fundamental el uso del método de gradiente conjugado explicado en el capitulo anterior. A continuación, detallaremos una serie de ejemplos que resolveremos con la utilización de Matlab a través de los métodos descritos en este trabajo.

Capítulo 4

Formulación del MEF para el caso no lineal

4.1. Introducción

En este capítulo veremos como resolveremos algunos de los ejercicios propuestos en [Rei80], por tanto tendremos que adaptar el método de elementos finitos para el problema no lineal. En los ejercicios propuestos el coeficiente no lineal se encuentra en el vector $\vec{f}$, es decir, afectará a nuestra formulación del vector del segundo miembro en nuestro sistema. Finalmente, resolveremos el sistema utilizando el método de gradiente conjugado con la variante de Polak-Ribiére. Uno de los problemas no lineales estudiados en [Rei80] es problema de Bratu, que enunciaremos a continuación para el caso 1D.

4.2. El problema de Bratu 1D

El problema de Bratu en dimensión uno se puede formular de la siguiente manera:

$$\begin{cases} -u''(x) = \lambda e^{u(x)}, \ \forall x \in [0,1] \ y \ \lambda \in \mathbb{R}^+ \\ u(0) = 0, \quad u(1) = 0 \end{cases}$$

En consecuencia, el cambio se efectuará sobre $f = \lambda e^{u(x)}$. Seguidamente, se desarrollará como modificar el método de elementos finitos para adaptarlo a un problema con término no lineal en f.

4.2.1. Ajuste del modelo no lineal

A continuación describiremos como ajustar nuestro problema no lineal a la programación usual del MEF lineal. Por simplicidad tomaremos $\lambda = 1$, en el caso de tomar otro valor el procedimiento sería idéntico.

En virtud de las condiciones de contorno, la formulación variacional del problema sería de la siguiente forma:

$$\int_0^1 u'(x)v'(x)dx = \int_0^1 f(u(x))v(x)dx, \quad \forall v \in V \tag{4.1}$$

siendo $f(u(x)) = e^{u(x)}$.

Vemos que el término no lineal solo influye en el lado derecho de la ecuación, es decir, en el vector segundo miembro $\vec{b}$. Por ello idea será integrar ese mismo lado de la ecuación con una regla de integración que solo evalúe la función en los nodos, dejando exactamente igual los términos de la matriz A. Seguimos por tanto el mismo procedimiento que en el capítulo 3 del trabajo.

$$\begin{aligned}\int_0^1 f(u_h(x))v_h(x)dx &= \sum_{i=1}^{N+1}\int_{T_i} f(u_h(x))_{|T_i}v_h(x)_{|T_i}dx = \\ &= \sum_{i=1}^{N+1}\int_{T_i} f(u_h(x)_{|T_i})v_h(x)_{|T_i}dx =\end{aligned} \tag{4.2}$$

Sustituyendo ahora $v_h(x)_{|T_i} = (\vec{w}^{(i)}(x))^T\vec{v}^{(i)}$ y $u_h(x)_{|T_i} = (\vec{w}^{(i)}(x))^T\vec{u}^{(i)}$ se obtiene

$$= \sum_{i=1}^{N+1}\int_{T_i} f((\vec{w}^{(i)}(x))^T\vec{u}^{(i)})(\vec{w}^{(i)}(x))^T\vec{v}^{(i)}dx = \sum_{i=1}^{N+1}(\vec{v}^{(i)})^T\underbrace{\int_{T_i}(\vec{w}^{(i)}(x))f((\vec{w}^{(i)}(x))^T\vec{u}^{(i)})dx}_{\vec{b(u)}^{(i)}} \tag{4.3}$$

Esta última integral la resolveremos aproximando con la ya utilizada Regla de Poncelet (o punto medio),

$$\int_{T_i}(\vec{w}^{(i)}(x))f((\vec{w}^{(i)}(x))^T\vec{u}^{(i)})dx \approx h_i[\vec{w}^{(i)}(x_m^{(i)}))f((\vec{w}^{(i)}(x_m^{(i)}))^T\vec{u}^{(i)}] = \tag{4.4}$$

Como ya vimos $\vec{w}^{(i)}(x_m^{(i)}) = (0{,}5 \;\; 0{,}5)^T$ y $\vec{u}^{(i)} = (u_h(x_1^{(i)}) \;\; u_h(x_2^{(i)}))^T$ con lo cual se obtiene

$$= \frac{h_i f\left((0{,}5 \;\; 0{,}5)\begin{pmatrix} u_h(x_1^{(i)}) \\ u_h(x_2^{(i)})) \end{pmatrix}\right)}{2} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \tag{4.5}$$

De esta forma modificamos el vector aproximando la función u por los valores en los nodos correspondientes. Esta técnica se puede consultar en más profundidad en la referencia [TM21].

4.2.2. Solución analítica del problema de Bratu

Para implementar el algoritmo será importante hallar las soluciones analíticas al problema de Bratu 1D. Para ello haremos un breve resumen de las posibilidades, estas se pueden consultar fácilmente en [Moh14].

Se define por tanto λ_c, el punto crítico de la ecuación diferencial a partir del cual varía la cantidad de soluciones del problema, que serán las siguientes:

- Si $\lambda = \lambda_c$ entonces existirá una única solución exacta, que viene dada por la ecuación $u(x) = -2\ln\left(\frac{\cosh\left(\frac{\theta}{2}(x-\frac{1}{2})\right)}{\cosh\frac{\theta}{4}}\right)$ donde $\theta = \sqrt{2\lambda}\cosh\left(\frac{\theta}{4}\right)$.
- Si $\lambda > \lambda_c$ entonces el problema no tendrá soluciones exactas.
- Si finalmente $\lambda < \lambda_c$ el problema contará con dos soluciones. Estas se calculan a partir de la misma ecuación que en el primer caso.

En nuestro caso, analizaremos en la práctica las soluciones del problema de solución única. Realizando los cálculos obtendremos los valores $\lambda_c = 3{,}513830719$ y $\theta_c = 4{,}79871456$. Cabe recalcar que el enunciado del problema sería mucho más general que el presentado, pues se ha modificado las condiciones de λ y x para una mejor implementación del código.

Capítulo 5

Resultados obtenidos

5.1. Resultados de los programas base

5.1.1. Gradiente conjugado

A continuación, comenzaremos mostrando algunos de los resultados obtenidos para el programa de gradiente conjugado con la variante de Polak-Ribière utilizando la Regla de Wolfe-Powell y el método de interpolación cuadrática con la regla de Goldstein. Para ello tomaremos los coeficientes de la búsqueda inexacta en el intervalo $0 < c_1 < c_2 < 1$ y seleccionaremos una solución inicial acorde al problema correspondiente.

Ejemplo 1.

Analizaremos el caso no cuadrático tomando la función $J : \mathbb{R}^2 \to \mathbb{R}$ tal que $J(x, y) = (x-1)^2+(y-x^3)^2$. Para ello, seleccionaremos los parámetros de búsqueda inexacta inexacta $c_1 = 0{,}2$ y $c_2 = 0{,}4$. Además, el punto inicial seleccionado será $x = (1, 2)^T$.

	Iteraciones	**Norma del gradiente**	**Solución final**
Gradiente conjugado Bisección de Wolfe	30	2.74e-08	(1,1)
GC-Interpolación Cuadrática-Goldstein	20	2.29e-08	(1,1)

En este ejemplo se ha calculado el mínimo de forma exacta. Cabe mencionar que para el caso en el que se utiliza la interpolación parabólica se ha seleccionado

una tolerancia al cubo con respecto tolerancia utilizada para el resto del programa, obligando así a una convergencia mucho más estricta. Se recalca que la convergencia se realizó pasando los criterios de Goldstein, por tanto la búsqueda inexacta de Goldstein resultó eficaz para el caso no cuadrático con Interpolación.

Ejemplo 2.

Para este caso tomaremos la función de Rastrigin $J : \mathbb{R}^n \to \mathbb{R}$ tal que $J(x) = An + \sum_{i=1}^{n}(x_i^2 - Acos(2\pi x_i))$. Esta función suele utilizarse usualmente para evaluar el funcionamiento del algoritmo para el caso no lineal. En nuestro caso probaremos para $A = 10$, $x_i \in [-5{,}12, 5{,}12]$ y en dimensión $n = 2$. Para realizar los métodos de búsqueda inexacta se utilizarán los valores $c_1 = 0{,}2$ y $c_2 = 0{,}4$ y como punto inicial se usará $x = (0{,}2, 0{,}2)^T$.

	Iteraciones	**Norma del gradiente**	**Solución final**
Gradiente conjugado Bisección de Wolfe	13	2.480e-08	(-1.217e-09,-1.217e-09)
GC-Interpolación Cuadrática-Goldstein	301	-5.373e-08	(4.045e-10,4.045e-10)

En este caso vemos como la búsqueda inexacta de Wolfe ha sido mucho más rapida que la interpolación cuadrática. En ambas técnicas la solución no se ha podido calcular de forma exacta, de igual modo, ambas aproximaciones se pueden considerar de orden satisfactorio. Cabe destacar que el test pasa los criterios de Goldstein utilizando la búsqueda cuadrática, es decir, la búsqueda inexacta de Goldstein ha tenido éxito.

En la siguiente sección se mostrará el funcionamiento del programa principal de elementos finitos utilizando la resolución mediante gradiente conjugado que acabamos de comprobar. Para ello se mostrarán varios problemas, de los cuales estudiaremos su eficacia evaluando el orden de las normas usadas durante este trabajo (ver anexo).

5.1.2. Método de elementos finitos

Trataremos mediante la programación de resolver el problema de **Sturm-Liouville** descrito en el capítulo 2 del tabajo:

$$\begin{cases} -(p(x)u'(x))' + q(x)u(x) = f(x) \ en \ (a,b) \\ Con \ \ L_1u(a) + L_2u'(a) = \alpha, \quad L_3u(b) + L_4u'(b) = \beta \end{cases}$$

Donde $u(x)$ será la solución exacta y $L = [L_1, L_2, L_3, L_4]$ el vector que determine si trabajaremos con condiciones de contorno Dirichlet, Neumann o Robin para cada extremo del intervalo. En nuestro caso se utilizarán ejemplos de condiciones Dirichlet.

Primero, mostraremos los resultados obtenidos por el método para mallas que sea uniformes con soluciones exactas. Para ello utilizaremos la norma infinito, la norma L2 y la seminorma H1.

Enunciaremos un primer ejemplo

$$\begin{cases} -((2 + cos(x))u'(x))' + (1 + sin(x))u(x) = f(x) \ en \ (0, \pi) \\ u(0) = 1, \quad u(\pi) = 1 + \pi^3 \end{cases}$$

Con $\mathbf{f(x)} = -(2 + \cos(x))[6x - \cos(\sin(x))(\cos(x))^2 + \sin(\sin(x))\sin(x)] + (\sin(x))[3x^2 - \sin(\sin(x))\cos(x)] + (1 + \sin(x))[x^3 + \cos(\sin(x))]$ función del lado derecho, y, $\mathbf{u(x)} = x^3 + \cos(\sin(x))$ su solución exacta.

	$\|u - u_h\|_\infty$	$\|u - u_h\|_{L_2}$	$\|u - u_h\|_{H_1^0}$
10	1.2668e-01	6.0009e-02	1.7615e+00
100	1.2675e-03	5.7878e-04	1.7526e-01
1000	1.2674e-05	5.7857e-06	1.7525e-02
10000	1.2666e-07	5.7893e-08	1.7525e-03

Aquí mostramos los resultados obtenidos mediante unas tablas de error con las tres normas anteriormente mencionadas y con distinto número de elementos. Así podemos observar como el orden se mantiene en base a la teoría como habíamos predicho. Vemos como para las normas infinito y L_2 se valida el orden 2, mientras que para la semi-norma obtenemos orden 1.

Estas normas se han realizado utilizando las técnicas de cuadratura indicadas en el ANEXO.

5.1.3. MEF con resolución mediante GC con búsqueda lineal inexacta

Problemas lineales

Resolveremos primero un ejemplo sencillo donde $q(x) = 0$, para ello utilizaremos dos tipos de precondicionamiento,LU y Cholesky, descritos en el ANEXO IV

$$\begin{cases} -u''(x) = 1 \ en \ (0, 10) \\ u(0) = 0, \quad u(10) = -\frac{1}{2} \\ sol \quad exacta \quad u(x) = -\frac{x^2}{2} + 5x \end{cases}$$

Mostraremos la tabla del error para los resultados con un precondicionamiento de Cholesky utilizando la búsqueda inexacta de Wolfe

	$\|u - u_h\|_\infty$	$\|u - u_h\|_{L_2}$	$\|u - u_h\|_{H_1^0}$
10	7.3738 e-07	2.6352 e-01	9.1287 e-01
100	1.7714 e-07	2.8349 e-03	9.1287 e-02
1000	3.5110 e-08	2.6410 e-05	9.1287 e-03
10000	1.3227 e-09	2.6049 e-07	9.1287 e-04

Utilizando la interpolación parabólica con la búsqueda inexacta de Goldstein se han obtenido los siguientes resultados:

	$\|u-u_h\|_\infty$	$\|u-u_h\|_{L_2}$	$\|u-u_h\|_{H_1^0}$
10	7.1054 e-15	2.6352 e-01	9.1287 e-01
100	2.8066 e-13	2.6352 e-03	9.1287 e-02
1000	1.6521 e-11	2.6352 e-05	9.1287 e-03
10000	8.0952 e-10	2.6330 e-07	9.1287 e-04

Problemas no lineales

A continuación resolveremos el problema de Bratu, ya desarrollado en el anterior capítulo. Este problema se resolverá para el caso de solución única, es decir, con $\lambda = 3{,}513830719$ y $\theta_c = 4{,}79871456$.

$$\begin{cases} -u''(x) = \lambda e^{u(x)} \ \forall x \in [0,1] \\ Con\ u(0) = 0 = u(1) \end{cases}$$

Con solución exacta $u(x) = -2\ln\left(\frac{\cosh\left(\frac{\theta}{2}(x-\frac{1}{2})\right)}{\cosh\frac{\theta}{4}}\right)$ donde $\theta = \sqrt{2\lambda}\cosh\left(\frac{\theta}{4}\right)$.

	$\|u-u_h\|_\infty$	$\|u-u_h\|_{L_2}$	$\|u-u_h\|_{H_1^0}$
5	2.9924 e-01	2.4056 e-01	8.3628 e-01
10	1.7801 e-01	1.2936 e-01	4.5671 e-01
25	7.6672 e-02	5.4289 e-02	1.9370 e-01
50	9.3454 e-03	7.2600 e-03	9.8771 e-02
100	4.0952 e-03	4.6931 e-03	9.1287 e-03

Como vemos se mantiene el orden en los tres casos. Cabe resaltar el mallado escogido no es todo lo fino deseable debido a problemas de convergencia con el número de iteraciones utilizadas en la resolución del sistema con gradiente conjugado. Desde luego esto mejorable realizando una programación más eficiente que nos permita escoger un mallado mucho más fino.

Capítulo 6

Conclusiones y trabajo futuro

Durante este trabajo se han revisado las técnicas de gradiente conjugado para resolver sistemas de ecuaciones tanto para el caso lineal y no lineal. Para ello se ha optado por la variante de Polak-Ribiére, recomendada en la literatura del artículo principal del trabajo y en el resto de referencias consultadas y ya mencionadas. Además, se han utilizado técnicas de búsqueda inexacta como la Regla de Wolfe-Powell o la Regla de Goldstein junto a la interpolación cuadrática. Esto ha sido de gran utilidad para profundizar y complementar el contenido impartido en las asignaturas de métodos numéricos que se han impartido en el grado. Además, se han mostrado algunos de los resultados del programa realizado con resultados satisfactorios.

También se ha revisado la implementación del método de elementos finitos para la discretización de un problema 1D, así como modificado para la implementación del problema no lineal (Bratu) sugerido en el artículo [Rei80]. Para ello se han utilizado algunas de las técnicas sugeridas en la asignatura de Análisis Numérico de Ecuaciones en Derivadas Parciales del cuarto curso del grado. Aunque los resultados de la programación no han sido todo lo satisfactorio posible, esto nos puede servir para en el futuro realizar una programación mucho más eficiente y óptima del algoritmo basándonos en el estudio teórico mostrado durante el trabajo.

Como complementario al trabajo, cabe resaltar que existen otras técnicas de discretización para los problemas vistos, como el método de diferencias finitas, en las que se puede profundizar para una comparación con los resultados obtenidos mediante el método de elementos finitos. Además, también existen otros métodos iterativos para la resolución del sistema matricial como por ejemplo el Método de

Newton , revisado en varias de las asignaturas del grado. Estos métodos también son mencionados y estudiados en mucha de la bibliografía presentada en este trabajo.

Finalmente, cabe destacar que se han resuelto problemas para el caso unidimensional, pero para muchos de las ecuaciones diferenciales utilizadas, como por ejemplo en el caso del problema de Bratu, existen enunciados alternativos en dimensión superior expuestos en [Rei80].

$$\begin{cases} \Delta u(x) = \lambda e^{u(x)} \ \forall x \in \Omega \\ Con \ u(x) = 0 \ \forall x \in \partial\Omega \end{cases}$$

donde Δu denota el laplaciano de u.

Por tanto, una ampliación al estudio realizado durante el trabajo sería desarrollar los métodos vistos para un caso de dimensión superior. Cabe mencionar que en el artículo se utilizan métodos de continuación para resolver los problemas en dimensión superior.

Capítulo 7

ANEXOS

7.1. ANEXO I: Convergencia y existencia de los métodos de búsqueda inexacta

A continuación comentaremos los teoremas más relevantes para la convergencia de los métodos de búsqueda inexacta con Wolfe-Powell o Regla de Goldstein.

Lema 7.1. *Supongamos $J : \mathbb{R}^{\kappa} \to \mathbb{R}$ continuamente diferenciable. Sea d_k una dirección de descenso en x_k y supongamos que f es acotada a lo largo del conjunto $\{x_k + \alpha d_k : \alpha > 0\}$. Entonces si $0 < c_1 < c_2 < 1$, existen intervalos de longitudes de paso que cumplen las condiciones de Wolfe-Powell y las condiciones de Wolfe-Powell fuertes.*

Demostración. Como $\phi(\alpha) = J(x_k + \alpha d_k)$ es una función acotada para todo $\alpha > 0$ con $0 < c_1 < 1$, entonces si denotamos el lado derecho de la condición (2.11) por $l(\alpha) = J(x_k) + \alpha c_1 \nabla J_k^T d_k$ este debe intersecar al menos una vez con la gráfica de ϕ. Denotemos pues $\bar{\alpha} > 0$ el valor más pequeño de α para el cual sucede dicha intersección.

$$J(x_k + \bar{\alpha} d_k) = J(x_k) + \bar{\alpha} c_1 \nabla J_k^T d_k \tag{7.1}$$

La condición de decrecimiento suficiente claramente es válida para longitudes de paso inferiores a $\bar{\alpha}$. Utilizando el Teorema de Valor Medio, tenemos que existe $\bar{\bar{\alpha}} \in (0, \bar{\alpha})$ tal que

$$J(x_k + \bar{\bar{\alpha}} d_k) - J(x_k) = \bar{\alpha} \nabla J(x_k + \bar{\bar{\alpha}} d_k)^T d_k \tag{7.2}$$

Combinando las dos igualdades anteriores obtenemos la siguiente expresión:

$$\nabla J(x_k + \bar{\bar{\alpha}} d_k)^T d_k = c_1 \nabla J_k^T d_k > c_2 \nabla J_k^T d_k \tag{7.3}$$

ya que $c_1 < c_2$ y $\nabla J_k^T d_k < 0$. Por tanto, $\bar{\bar{\alpha}}$ satisface las condiciones de Wolfe-Powell. Tomando nuestras hipótesis sobre f, sabemos que existe un entorno de $\bar{\bar{\alpha}}$ para el cual se cumplen dichas condiciones de Wolfe-Powell. Además, dado que en la última igualdad (7.3) el lado izquierdo es negativo, las condiciones fuertes de Wolfe-Powell también se cumplen en el mismo intervalo.

□

Además, las condiciones de Wolfe-Powell son escalarmente invariantes, es decir, si multiplicamos la función objetiva por una constante o realizamos una transformación afín no alteramos las condiciones. (Para más detalles mirar [NW06]). A continuación veremos la convergencia para los métodos de Wolfe-Powell y Goldstein.

Convergencia del Algoritmo utilizado Regla de Goldstein o Wolfe-Powell

Para demostrar el descenso de los métodos será fundamental evitar los casos en los que las direcciones de búsqueda (que definiremos $s_k = \alpha_k d_k$) estén próximas a la ortogonalidad con $-g_k$, es decir, el ángulo θ_k entre s_k y $-g_k$ está uniformemente delimitado

$$\theta_k \leq \frac{\pi}{2} - \mu, \ \forall k \tag{7.4}$$

donde $\mu > 0$, $\theta \in [0, \frac{\pi}{2}]$ está definido por

$$\cos(\theta_k) = \frac{-g_k^T s_k}{(\|g_k\| \|s_k\|)} \tag{7.5}$$

garantizando el descenso.

Teorema 7.2. *Dado un α_k por la Regla de Goldstein o Wolfe-Powell y un s_k que satisface las condiciones definidas anteriormente. Si ∇J existe y es uniformemente continuo en el conjunto $\{x : J(x) \leq J(x_0)\}$ entonces $\nabla J(x_k) =$ para algún k, o $J(x_k) \to -\infty$, o $\nabla J(x_k) \to 0$.*

Demostración. Ver [SY06]. □

Convergencia del algortimo de Interpolación cuadrática

A continuación veremos el teorema que asegura la convergencia para el algritmo con Interpolaión cuadrática, para esto necesitaremos definir $\alpha*$ como el punto que verifica $\phi'(\alpha*) = 0$ y $\phi''(\alpha*) \neq 0$.

Teorema 7.3. *Dada $J : \mathbb{R} \longrightarrow \mathbb{R}$ una función continuamente diferenciable de orden 4 y el punto $\alpha*$ con las condiciones anteriores. Entonces la sucesión $\{\alpha_k\}$ generada por (2.18) y (2.19) es convergente con orden 1.32*

Demostración. Definimos la fórmula de interpolación de Lagrange $L(\alpha) = \phi_1 \frac{(\alpha-\alpha_2)(\alpha-\alpha_3)}{(\alpha_1-\alpha_2)(\alpha_1-\alpha_3)}$ $+ \phi_2 \frac{(\alpha-\alpha_1)(\alpha-\alpha_3)}{(\alpha_2-\alpha_1)(\alpha_2-\alpha_3)} + \phi_3 \frac{(\alpha-\alpha_1)(\alpha-\alpha_2)}{(\alpha_3-\alpha_1)(\alpha_3-\alpha_2)}$,de esta forma, será equivalente calcular $L'(\alpha) = 0$ con el desarrollo de las fórmulas anteriores (2.18) y (2.18). Establecemos a partir de esto la ecuación

$$\phi(\alpha) = L(\alpha) + R(\alpha) \tag{7.6}$$

Donde

$$R(\alpha) = \frac{1}{6}\phi''(\xi(\alpha))(\alpha - \alpha_1)(\alpha - \alpha_2)(\alpha - \alpha_3) \tag{7.7}$$

A partir de las hipótesis del teorema $0 = \phi'(\alpha^*) = L'(\alpha^*) + R'(\alpha^*)$, sustituyendo se obtiene:

$$\phi_1 \frac{2\alpha^* - (\alpha_2 + \alpha_3)}{(\alpha_1 - \alpha_2)(\alpha_1 - \alpha_3)} + \phi_2 \frac{2\alpha^* - (\alpha_3 + \alpha_1)}{(\alpha_2 - \alpha_1)(\alpha_2 - \alpha_3)} + \phi_3 \frac{2\alpha^* - (\alpha_1 + \alpha_2)}{(\alpha_3 - \alpha_1)(\alpha_3 - \alpha_2)} + R'(\alpha^*) = 0 \tag{7.8}$$

Reescribimos (2.18) dividiendo por $(\alpha_1-\alpha_2)(\alpha_2-\alpha_3)(\alpha_3-\alpha_1)$ en el denominador y numerador obteniendo el siguiente término de la sucesión:

$$\alpha_4 = \frac{1}{2} \frac{\frac{\phi_1(\alpha_2+\alpha_3)}{(\alpha_1-\alpha_2)(\alpha_1-\alpha_3)} + \frac{\phi_2(\alpha_3+\alpha_1)}{(\alpha_2-\alpha_1)(\alpha_2-\alpha_3)} + \frac{\phi_3(\alpha_1+\alpha_2)}{(\alpha_3-\alpha_1)(\alpha_3-\alpha_2)}}{\frac{J_1}{(\alpha_1-\alpha_2)(\alpha_1-\alpha_3)} + \frac{\phi_2}{(\alpha_2-\alpha_1)(\alpha_2-\alpha_3)} + \frac{\phi_3}{(\alpha_3-\alpha_1)(\alpha_3-\alpha_2)}} \tag{7.9}$$

Tomando (7.9) y (7.8) podemos ajustarlo tal que :

$$\alpha^* - \alpha_4 = \frac{1}{2} \frac{R_3'(\alpha^*)}{\frac{\phi_1}{(\alpha_1-\alpha_2)(\alpha_1-\alpha_3)} + \frac{\phi_2}{(\alpha_2-\alpha_1)(\alpha_2-\alpha_3)} + \frac{\phi_3}{(\alpha_3-\alpha_1)(\alpha_3-\alpha_2)}} \tag{7.10}$$

A continuación, establecemos $e_i = \alpha^* - \alpha_i,\ i = 1, 2, 3, 4$. De esta forma, reescribimos la ecuación anterior por:

$$e_4[-\phi_1(e_2-e_3)-\phi_2(e_3-e_1)-\phi_3(e_1-e_2)] = -\frac{1}{2}R'(\alpha^*)(e_1-e_2)(e_2-e_3)(e_3-e_1). \quad (7.11)$$

Sabiendo que $\phi'(\alpha^*) = 0$ se sigue desde la fórmula de expansión del polinomio de Taylor que

$$\phi_i = \phi(\alpha^*) + \frac{1}{2}e_i^2\phi''(\alpha^*) + O(e_i^3) \quad (7.12)$$

Tomando las dos fórmulas anteriores (7.11 sin el término de tercer orden) y (7.12) llegamos a la expresión siguiente:

$$e_4 = \frac{1}{\phi''(\alpha^*}R'(\alpha^*). \quad (7.13)$$

Además, por la fórmula de Interpolación de Lagrange (7.7) tenemos la siguiente igualdad:

$$R'(\alpha^*) = \frac{1}{6}J'''(\xi(\alpha^*))(e_1e_2 + e_2e_3 + e_3e_1) + \frac{1}{24}J^{(4)}(\eta)e_1e_2e_3. \quad (7.14)$$

Ahora, eliminando el término de cuarto orden y sustituyendo (7.13) en (7.14) obtenemos la siguiente expresión

$$e_4 = \frac{J'''(\xi(\alpha^*))}{6J''(\alpha^*)}(e_1e_2 + e_2e_3 + e_3e_1) = M(e_1e_2 + e_2e_3 + e_3e_1), \quad (7.15)$$

Donde M es una constante. Si generalizamos, tenemos la fórmula

$$e_{k+2} = M(e_{k-1}e_k + e_ke_{k+1} + e_{k+1}e_{k-1}) \quad (7.16)$$

Ya que $e_{k+1} = O(e_k) = O(e_{k-1})$ cuando $e_k \to 0$, entonces existe un $\bar{M} > 0$ verificando que

$$|e_{k+2}| \leq \bar{M}|e_{k-1}||e_k| \quad (7.17)$$

como $\bar{M} > 0$ esto es lo mismo que escribir

$$\bar{M}|e_{k+2}| \leq \bar{M}|e_{k-1}|\bar{M}|e_k| \quad (7.18)$$

Donde $|e_i|$ (i=1,2,3) es suficientemente pequeño para verificar la condición

$$\delta = \text{máx}\{\bar{M}|e_1|, \bar{M}|e_2|, \bar{M}|e_1|\} < 1, \tag{7.19}$$

Por un lado

$$\bar{M}|e_1| \leq \bar{M}|e_1|\bar{M}|e_2| \leq \delta^2. \tag{7.20}$$

Ajustamos por tanto

$$\bar{M}|e_k| \leq \delta^{q_k} \tag{7.21}$$

De esta forma

$$\bar{M}|e_{k+2}| \leq \bar{M}|e_k|\bar{M}|e_{k-1}| \leq \delta^{q_k}\delta^{q_{k-1}} \triangleq \delta^{q_{k+2}} \tag{7.22}$$

Por eso

$$q_{k+2} = q_k + q_{k-1}, \quad (k \geq 2) \tag{7.23}$$

donde $q_1 = q_2 = q_3$, su ecuación característica será por tanto $t^3 - t - 1 = 0$. Con la raíz real $t_1 \approx 1{,}32$ y otras dos raíces complejas $|t_2| = |t_3| < 1$. La solución general de la ecuación (7.23) tiene la forma

$$q_k = A \cdot t_1^1 + B \cdot t_2^k + C \cdot t_3^k, \tag{7.24}$$

donde A,B y C son coeficientes a determinar. Claramente cuando $k \to \infty$,

$q_{k+1} - t_1 q_k = Bt_2^K(t_2 - t_1) + Ct_3^K(t_3 - t_1) \to 0$

En este caso, cuando k es suficientemente grande $q_{k+1} - t_1 q_k \geq -0{,}1$. Utilizando (7.21) tenemos $|e_k| \leq (\frac{1}{\bar{M}}\delta^{q_k} \triangleq B_k, \ \ (k \geq 1)$. Por tanto utilizando la condición para k suficientemente grande se obtiene

$$\frac{B_{k+1}}{B_k} = \frac{\frac{\delta^{q_{k+1}}}{\bar{M}}}{\frac{\delta^{t_1 q_k}}{(\bar{M})_1^t}} = \bar{M}^{t_1 - 1}\delta^{q_{k+1} - t_1 q_k} \leq \delta^{-0{,}1}\bar{M}^{t_1 - 1} \tag{7.25}$$

Lo cual indica la convergencia con orden $t_1 \approx 1{,}32$.

□

7.2. Anexo II: Existencia y unicidad de solución MEF

Teorema 7.4. *(**Lax-Milgram**). Sea H un espacio de Hilbert y sea $\mathbb{A} : H \times H \to \mathbb{R}$ una forma bilineal, continua y coercitiva. Dada $\mathbb{L} : H \to \mathbb{R}$ lineal y continua sobre H. Entonces existe un único $u_0 \in H$ tal que $\mathbb{A}(u_0, v) = \mathbb{L}(v) \quad \forall v \in H$*

A través del Teorema de Lax-Milgram garantizamos la existencia de una solución del problema discreto y variacional.(Para ver una prueba y desarrollo del teorema de Lax-Milgram se puede consultar [CM16],([LB10]))

Además, podemos definir las hipótesis sobre las variables del problema de Sturm-Lioville para la existencia y unicidad de la solución:

- $p \in L^\infty(a, b)$ *donde* $\exists c \in \mathbb{R} \ : \ p(x) \geq c > 0$ casi por doquier en (a, b)
- $q \in L^\infty(a, b)$ *donde* $p(x) \geq 0$ casi por doquier en (a, b)
- $f \in L^2(a, b)$

7.3. Anexo III: Normas del error

Desarrollaremos a continuación las normas utilizadas para la aproximación de soluciones de los métodos utilizados en este trabajo. Para ello hemos utilizado las normas infinito, seminorma H_1 y norma L_2.

7.3.1. Norma infinito

Dado un vector $\vec{u} = (x_1, x_2, x_3, ..., x_n) \in \mathbb{R}^n$ definimos la norma infinito como

$$\|\vec{u}\|_\infty = max(|x_1|, |x_2|, ..., |x_n|) = \max_{i \in \{1,...,n\}} |x_i| \tag{7.26}$$

7.3.2. Seminorma H_1

Para cualquier solución exacta u y su aproximación u_h en el dominio $[a, b]$ se describe la norma de la energía (o semi-norma H1) como:

$$\|u - u_h\|_{H_1^0} = \left(\int_a^b \sum_{i=1}^{N+1} \left| \frac{du}{dx} - \frac{du_h}{dx} \right|^2 dx \right)^{\frac{1}{2}} \tag{7.27}$$

7.3.3. Norma L_2

Para cualquier solución exacta u y su aproximación u_h en el dominio $[a, b]$ se describe la norma L_2 como:

$$\|u - u_h\|_{L_2} = \left(\int_a^b |u - u_h|^2 dx \right)^{\frac{1}{2}} \tag{7.28}$$

A continuación de estas tres mediciones del error, producido a causa de la aproximación del método empleado, enunciaremos un teorema de vital importancia en referencia al error producido por el método de elementos finitos aplicado a las ecuaciones diferenciales elípticas. El siguiente Lema nos permitirá garantizar que u_h es del orden de la mejor aproximación en el espacio que hemos diseñado, es decir, con la norma elegida es el elemento de $\mathcal{V}_h$ que menos dista de la solución exacta u.

Lema 7.5. ***(Cea, cota del error)****. El lema de cea nos asegura que existe una constante $C > 0$, independiente de h tal que*

$$\|u(x) - u_h(x)\| \leq C \, \text{ínf}_{v_h \in \mathcal{V}_{h0}} \|u(x) - v_h(x)\| = C d(u(x), \mathcal{V}_h)$$

Se puede consultar su demostración y desarrollo en el trabajo original de J.Cea ([Céa64]).

7.4. ANEXO IV: Métodos de cuadratura

Durante este trabajo, sobre todo en la construcción del método de elementos finitos, se han resuelto diversas integrales mediante aproximación con métodos de cuadratura, fundamentales en la resolución de ecuaciones diferenciales.

7.4.1. Cuadratura de Gauss con 2 puntos

La técnica de cuadratura Gaussiana será necesaria para el cálculo de las normas enunciadas en el apartado anterior, manteniendo así el orden buscado. Está regla será una buena aproximación de una integral de la función $f(x)$ en el intervalo $[a, b]$ mediante:

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} w_i f(x_i) \tag{7.29}$$

Está será exacta para polinomios de grado $2n-1$ o menor. En nuestro caso particular, calcularemos los coeficientes de la aproximación para el caso $n=2$ mediante las raíces del polinomio de Legendre de primer grado.

$$\int_a^b f(x)dx \approx f(-\frac{1}{sqrt(3)}) + f(\frac{1}{sqrt(3)}) \tag{7.30}$$

Además, se puede demostrar que la fórmula general del término del error en las cuadraturas de Gauss es

$$Err(f) = \frac{2^{n+1}n!^4}{(2n+1)(2n)!^3} f^{2n)}(c) \quad (a \leq c \leq b)$$

7.4.2. Cuadratura de Poncelet

Para resolver las integrales y calcular la matriz y el vector del método aproximaremos estas mediante un método de Cuadratura del punto medio:

$$\int_a^b f(x)dx \approx (b-a) f(\frac{a+b}{2}) \tag{7.31}$$

Esto, al dividir el intervalo $[a,b]$ en N subintervalos, lo transformaremos de la siguiente forma, dado $\bar{x}_n = \frac{x_{n-1}-x_n}{2}$ se tiene

$$\int_a^b f(x)dx \approx h_i f(\bar{x}_n) \tag{7.32}$$

7.5. ANEXO V: Precondicionamiento en el método de gradiente conjugado

En esta sección explicaremos los métodos de precondicionamiento utilizados en los métodos de gradiente conjugado, en nuestro caso utilizaremos dos de los más conocidos, el método con precondicionamiento con matriz de Cholesky y el método de matriz de precondicionamiento LU.

En sistemas $Ax = b$ donde la matriz A tiene un condicionamiento muy alto los resultados de aproximación pueden verse afectados, perdiendo orden de aproximación o perdiendo velocidad en la convergencia. Para solucionarlo, la idea principal será multiplicar el sistema por una matriz M^{-1} de modo que el sistema a resolver sea el siguiente

$$M^{-1}Ax = M^{-1}b \tag{7.33}$$

M será un buen precondicionamiento cuando $M^{-1}A \approx I$.

7.5.1. Factorización Incompleta

En este caso descompondremos $A = LU$ de modo que nuestra matriz de precondicionamiento vendrá definida por $M = LU$. Este se muestra muy eficiente para métodos de gradiente conjugado.

7.5.2. Factorización incompleta de Cholesky

Otro de os precondicionamientos más eficaces en los métodos de gradiente conjugado es factorizar $A = L^T L$ y por tanto $M^{-1} = L^T L$.

La implementación del método de gradiente conjugado con precondicionamiento se puede consultar con más precisión en ([SY06]).

7.6. PROGRAMAS UTILIZADOS

Por simplicidad se incluirá solo el programa final que utiliza el método de elementos finitos y el método de gradiente conjugado para resolver el problema de Bratu:

```
%METODO DE ELEMENTOS FINITOS CON GRADIENTE CONJUGADO
clc
close all
format long
global N
%FICHERO DE SALIDA DE RESULTADOS
ofi = fopen('datos&resultadosMEF.txt','wt');
%Llama a ese fichero y lo lee (trae los datos)
data
%
fprintf(ofi,'\n\n****␣RESULTADOS****');
```

```
[x,h] = MEF_LagrangeP1(a,b,N,vmesh,ofi);

%SOLUCI N DEL SISTEMA MATRICIAL
%par : par metro de b squeda inexacta
%(par == 1 -> b squeda Wolfe     // par == 0 ->Interpolaci n parab lica
%pc: par metro de selecci n del precondicionamiento
%pc == 1 (Cholesky) , pc == 2 (LU) pc == 0(ninguno)

%Metodo GC con precondicionamiento
par = 1; %m todo de b squeda GC
pc = 1; %precondicionado
[uh,t,index_conver,A,bb] = PNGC_parabola(x,par,pc,L,alph,bet,funp,funq,fu

%Gradiente conjugado Matlab (caso lineal)
%x0=zeros(1,N+1);
%for i=1:2:N
%      x0(i)=2;
%      x0(i+1)=1;
%end
%x0=x0';
%M=ichol(A);
%uh = pcg(A,bb,1.e-6,1000,M,M',x0);

%M todos de resoluci n exacta
%res : par metro que selecciona el m todo de resoluci n directa en el
%(res == 1 -> Cholesky // res == 0 -> backslash reducida)
% res = 0;
% [uh]=SolMatlab(res);

%Escribimos en el fichero el vector soluci n
fprintf(ofi,'\n\n* El vector soluci n es \n');
escribe_v('uh=',uh,ofi);
```

```
% C lculo de las normas del error en el caso de tener la soluci n exact
% iquad : par metro de selecci n del m todo de quadratura para norma L
% seminorma H1.( iquad = 2 -> Cuadratura de Gauss, iquad =1 ->Punto medio)
iquad = 2;
if ex==1
    fprintf(ofi,'\n\n* Calculo del error:');

    %NORMA INFINITO
    ub=fexact(x);
    error=uh-ub';
    error0=max(abs(error));
    fprintf(ofi,'\n El error en norma infinito es %e ',error0);
    %Atenci n: Error en norma infito solo se calcula sobre los nodos
    %Si la malla es regular, es equivalente al error en diferencias finit
    %Por tanto la norma infinito debe ser aprox 0

    %NORMA L2
    normaL2=normaL2_quad(uh,x,fexact,iquad);
    fprintf(ofi,'\n El error en norma L2 es %e ',normaL2);

    %SEMINORMA H1
    semiH1=H1semi(uh,x,dfunf,iquad);
    fprintf(ofi,'\n El error en seminorma H1 es %e ',semiH1);

    %Dibujamos la soluci n exact frente a la soluci n del MEF
    figure (2)
    hold on
    plot(x,uh,'or','MarkerSize',6);
    plot(x,ub,'-b','MarkerSize',4);
    legend('Solucion MEF','Solucion exacta')
    title('Sol exacta vs Sol aproximada')
    hold off
end
```

```
fclose(ofi);


function [x,h] = MEF_LagrangeP1(a,b,N,vmesh,ofi)
%Generaci n MEF:
%x(1) = a<x(2)<...<x(N)<x(N+1)=b.
%h(i) = x(i+1) - x(i) for i = 1,...,N.

xuniform=linspace(a,b,N+1); %N+1 puntos, N elementos
%Creamos las gr ficas finales
hold on
figure (1)
plot(xuniform, zeros(1,N+1),'*r','MarkerSize',3)
%Creamos malla intermedia
[x,h] = meshgenerator(a,b,N,vmesh,ofi);
if(isempty(vmesh))
    fprintf(ofi,'\n\n*La malla final es uniforme y el paso es h=(b-a)/N
    plot([a b],[0 0],'k-')
    legend('Malla final uniforme','Intervalo [a,b]')
    title('Resultados Malla (uniforme)')
    %reg=1; %Malla regular
else
    %reg=0; %Malla no regular
    fprintf(ofi,'\n\n*La malla final no es uniforme\n');
%     %sg es para poner los cuadrados verdes, luego le cambia el tama o
%     plot(vmesh,zeros(1,length(vmesh)),'sg','MarkerSize',12,'LineWidth',
%     legend('UNIFORM MESH','INTERMEDIATE MESH','vmesh')
%     %Dibujamos el segmento [a,b] y k- los une con una linea negra
%     plot([a b],[0 0],'k-')
    hold on
    plot(x,zeros(1,N+1),'ob','MarkerSize',5)
    plot(vmesh,zeros(1,length(vmesh)),'sr','MarkerSize',12,'LineWidth',1)
    plot([a b],[0 0],'k-')
    legend('Malla uniforme','Malla final no uniforme','Puntos de acumulac
    title('Resultados Malla (no uniforme)')
```

```
end
hold off


%INTRODUCIMOS LOS RESULTADOS EN EL FICHERO
%MALLA FINAL
fprintf(ofi,'\n\n* El vector de la malla es \n ');
escribe_v('x=',x,ofi);

%MATRIZ A Y VECTOR b
%fprintf(ofi,'\n\n* La matriz A del problema es A = ');
%[rows,cols,vals] = find(A);
%fprintf(ofi,  '\n(%d,%d)\t %f\n',[rows,cols,vals]');
%
%fprintf(ofi,'\n\n* El vector del segundo miembro es \n ');
%escribe_v('b=',bb,ofi);


end


function [x,h] = meshgenerator(a,b,N,vmesh,ofi)
%
%[x,h] = meshgenerator(a,b,N,vmesh,ofi).
%
%Esta funcion construye un vector
%a = x(1) < ... < x(N+1) = b que divide el dominio 1D
%[a,b] en N elementos [x(1),x(2)], ..., [x(N),x(N+1)].
%
%Los puntos del vector malla x son acumulados alrededor de los
%dados por el vector vmesh, y el vector vmesh contenido en x.
%(La malla x es uniforme en el caso vmesh = [].)
%
%El grado de acumulacion es controlado por el exponente s > 1
%y el parametro k >= 0 en las funciones fmesh0 y fmesh1.
```

```
%
%h es el vector que contiene las longitudes de todos los elementos.
%
%vnp es un vector que contiene el numero de puntos en cada subintervalo
%abierto (a,vmesh(1)) (ONLY IF a < vmesh(1)), (vmesh(k),vmesh(k+1))
%para todo k = 1,...,length(vmesh)-1, and (vmesh(end),b) (ONLY IF
%vmesh(end) < b).
%
len = length(vmesh); %Numero de elementos que hay en el vector de acumul
vnp=[];

if(len==0)
    %Malla uniforme
    h(1:N)=(b-a)/N; %Son N+1 elementos
    %Vector x con los nodos
    x=linspace(a,b,N+1);
else
    %caso no trivial
    %Construiremos una malla intermedia x, lo mas uniforme posible
    %forzaremos a los puntos de vmesh ser nodos
    %la malla final debe tener N+1 nodos
    %Test de longitud de vmesh(para que este contenida en x)
    if(len>N+1)
        fprintf(ofi,'\n\n* La logitud de vmesh supera a los nodos: ERROR'
        error('La longitud de vmesh es mayor que el numero de puntos')
    end

    if(vmesh(1)==a && vmesh(len)==b)
        %caso 1
        NPL=N+1-len;   %Numero de puntos que quedan
        %Quiero meter en cada trozo, un numero de puntos
        %proporcional a la longitud de cada trozo
        vl=vmesh(2:len)-vmesh(1:len-1); %vector de longitudes divididas p
        vnp=round(NPL.*vl./(b-a)); % vector puntos que debo meter en cada
```

```
        %Realizamos una cofrecci n para ver si metimos todos los puntos
        %necesarios (por utilizar round).
        suma=sum(vnp);

        if(suma<NPL)
            nee=NPL-suma; % n  de puntos que me faltan
            ind=randperm(len-1);
            ind=ind(1:nee);
            vnp(ind)=vnp(ind)+1;
        elseif(suma>NPL)
            notnee=suma-NPL; % n  de puntos que me sobran
            ind=randperm(len-1);
            ind=ind(1:notnee);
            vnp(ind)=vnp(ind)-1;
        end

        %Construimos vector malla x
        x=[];
        for k=1:len-1
            xk=linspace(vmesh(k),vmesh(k+1),vnp(k)+2);
            x=[x xk];
        end
        x=unique(x); %mismo vector pero sin repeticiones (correcci n de

    elseif(vmesh(1)>a && vmesh(len)==b)
        NPL=N-len;
        vl=[vmesh(1)-a,vmesh(2:len)-vmesh(1:len-1)];
        vnp=round(NPL.*vl./(b-a));
        suma=sum(vnp);
        if(suma<NPL)
            nee=NPL-suma;
            ind=randperm(len);
            ind=ind(1:nee);
            vnp(ind)=vnp(ind)+1;
```

```
        elseif(suma>NPL)
            notnee=suma-NPL;
            ind=randperm(len);
            ind=ind(1:notnee);
            vnp(ind)=vnp(ind)-1;
        end

        %Construimos vector x
        x=linspace(a,vmesh(1),vnp(1)+2);
        for k=1:len-1
            xk=linspace(vmesh(k),vmesh(k+1),vnp(k+1)+2);
            x=[x xk];
        end
        x=unique(x);


    elseif(vmesh(1)==a && vmesh(len)<b)
        NPL=N-len;
        vl=[vmesh(2:len)-vmesh(1:len-1),b-vmesh(len)];
        vnp=round(NPL.*vl./(b-a));
        suma=sum(vnp);
        if(suma<NPL)
            nee=NPL-suma;
            ind=randperm(len);
            ind=ind(1:nee);
            vnp(ind)=vnp(ind)+1;
        elseif(suma>NPL)
            notnee=suma-NPL;
            ind=randperm(len);
            ind=ind(1:notnee);
            vnp(ind)=vnp(ind)-1;
        end
        %Construimos vector x
        x=[];
```

```
        for k=1:len-1
            xk=linspace(vmesh(k),vmesh(k+1),vnp(k)+2);
            x=[x xk];
        end
        xk=linspace(vmesh(len),b,vnp(len)+2);
        x=[x xk];
        x=unique(x);

    elseif(vmesh(1)>a && vmesh(len)<b)
        NPL=N-len-1;
        vl=[vmesh(1)-a,vmesh(2:len)-vmesh(1:len-1),b-vmesh(len)];
        vnp=round(NPL.*vl./(b-a));
        suma=sum(vnp);
        if(suma<NPL)
            nee=NPL-suma;
            ind=randperm(len+1);
            ind=ind(1:nee);
            vnp(ind)=vnp(ind)+1;
        elseif(suma>NPL)
            notnee=suma-NPL;
            ind=randperm(len+1);
            ind=ind(1:notnee);
            vnp(ind)=vnp(ind)-1;
        end
        %Construimos vector x
        x=linspace(a,vmesh(1),vnp(1)+2);
        for k=1:len-1
            xk=linspace(vmesh(k),vmesh(k+1),vnp(k+1)+2);
            x=[x xk];
        end
        xk=linspace(vmesh(len),b,vnp(len+1)+2);
        x=[x xk];
        x=unique(x);
    end
```

```
%     %Ahora esta creada la malla intermedia
%     fprintf(ofi,'\n\n* Malla intermedia\n');
%     fprintf(ofi,'\n   %4.4f  ', x);
%
%     %Coeficiente de uniformidad de la malla intermedia
%     h=x(2:length(x))-x(1:length(x)-1);
%     hmin=min(h); hmax=max(h);
%     fprintf(ofi,'\n\n* Coeficiente de uniformidad de la malla intermedi
%     fprintf(ofi,'\n\n* (Cuanto mas proxima a 1, mas uniforme) = %-.6f.

    %-------------------------------------------------------------
    %Procederemos a crear la malla final
    xfinal=[];

    if(vmesh(1)>a)
        p1=a; p2=vmesh(1);
        idx1=find(x>=p1 & x<=p2);
        z=affineA(x(idx1),p1,p2); %Nodos de [p1,p2] a [0,1]
        zz=fmesh1(z);
        xfinal=[xfinal affineAInv(zz,p1,p2)]; %Nodos de [0,1] a [p1,p2]
    end

    for k=1:len-1
        p1=vmesh(k);
        p2=vmesh(k+1);
        mk=(p1+p2)./2;
        idx1=find(x>=p1 & x<=mk);
        z=affineA(x(idx1),p1,mk);
        zz=fmesh0(z);
        xfinal=[xfinal affineAInv(zz,p1,mk)];
        idx1=find(x>=mk & x<=p2);
        z=affineA(x(idx1),mk,p2);
        zz=fmesh1(z);
```

```
        xfinal=[xfinal affineAInv(zz,mk,p2)];
    end

    if(vmesh(len)<b)
        p1=vmesh(len); p2=b;
        idx1=find(x>=p1 & x<=p2);
        z=affineA(x(idx1),p1,p2);
        zz=fmesh0(z);
        xfinal=[xfinal affineAInv(zz,p1,p2)];
    end

    x=unique(xfinal);
    h=x(2:length(x))-x(1:length(x)-1);
%     fprintf(ofi,'\n\n* Malla final x = ');
%     fprintf(ofi,'\n  %4.4f ', x);

end

fprintf(ofi,'\n\n*_CREACION_DE_LA_MALLA_COMPLETADA_\n');
end


function [uh]=SolMatlab(res)

global A bb
%Resolvemos el sistema con Matlab
%se podra hacer con matriz no simtrica, pero con bloqueo de 10^20 Ma
%detecta Matriz mal condicionada, pero si utilizamos un bloqueo de 10^10
%lo detecta

if res ==0
 uh = A\bb;
elseif res==1
    %Cholesky (la matriz debe ser definida positiva)
    L = chol(A);
```

```
    uh = L'\bb;
    uh = L\uh;
else
    fprintf(ofi,'\n\n* parametro de resolucion matriz sin cambios \n');
end


end

function [alpha, i0, index] = WolfeBiseccion(f, g, x, d, ...
                                             rho, sigma, ...
                                             alphainit, gama, ...
                                             fid)


global A bb N R Rb

% Wolfe   Seleccion de paso para un metodo de descenso mediante la regla
%         inexacta llamada regla de Wolfe. La generacion de un nuevo punt
%         alpha entre alpha_p y alpha_g se realiza por biseccion.
%
% Parametros de entrada:
%     f          : Funcion coste (psoblemente multi-dimensional).
%     g          : Gradiente de la funcion coste.
%     x          : Punto en el que se encuentra el algoritmo de descenso.
%     d          : Direccion de descenso.
%     rho, sigma: Parametros que definen los criterios de la regla de
%                   Wolfe. Se supone que 0 < rho < sigma < 1.
%                   Es aconsejable que   0 < rho < 1/2.
%     alphainit : Valor de alpha que se considera inicialmente.
%                   Deberia satisfacer el criterio de paso grande.
%     gama      : Parametro empleado para seleccionar un paso inicial que
%                   verifique el criterio de paso grande en caso de que
%                   alphag no lo cumpla.
```

```
%                 Se trata de un factor de dilatacion. En consecuencia de
%                 tomarse tal que gama > 1.
%     fid         : Identificador de un fichero para la salida.
%
%Parametros de salida:
%     alpha       : Paso seleccionado por la regla de Wolfe para el metodo
%                 descenso.
%     i0          : Numero de evaluaciones del funcinal coste.
%     index       : Indicador de convergencia: 1 => exito, <= => fracaso.
%                 Si index = 0  => No se ha podido inicializar.
%                 Si index = -1 => No se ha podido encontrar un alpha
%                 admisible.

%Inicializacion del algoritmo.
%Seleccionamos un alpha que cumpla el criterio de paso grande y otro que
%cumpla el criterio de paso pequeno.

index  = 0;
i0     = 0;    %Numero de evaluaciones del funcional coste.
itmax1 = 10;   %Numero maximo de iteraciones en la inicializacion.
itmax2 = 50;   %Numero maximo de iteraciones en la busqueda.

init   = false;
alphap = 0;
alpha  = alphainit;


j0  = f(x);
jp0 = g(x).'*d;

while ~init && ( i0 <= itmax1 )
    xalpha  = x + alpha*d;
    jalpha  = f(xalpha);
    jpalpha = g(xalpha).'*d;
```

```
    CPG = (jalpha > j0 + rho * jp0 * alpha); % Criterio de paso grande.
    CPP = (jpalpha < sigma * jp0) && (~CPG); % Criterio de paso pequeno.

    i0 = i0 + 1;

    if CPG                              % Alpha cumple CPG
        alphag   = alpha;
        init     = true;
    elseif CPP                          % Alpha cumple CPP
        alphap   = alpha;
        alpha    = alpha * gama;
    else                                % Alpha cumple CPA
        index = 1;

        return;                         % Salimos con exito de la funcion
    end
end

if (i0 > itmax1) && (init == false)     %No se ha podido inicializar Wol
%     fprintf(fid,'No se pudo inicializar. Alphap=%e alphag=%e \n',alphap
    return;
end

% Algoritmo inicializado. Disponemos de un alphap que cumple el criterio
% de paso pequeno y un alphag que cumple el criterio de paso grande.
% Buscamos un paso admisible entre ambos.
fprintf(fid,'inicializacion %e %e \n',alphap, alphag);

while ( i0 <= itmax2 )

% Seleccionamos el nuevo alpha con el metodo de biseccion.

    alpha = (alphag + alphap)/2;
```

```
    xalpha  = x + alpha*d;
    jalpha  = f(xalpha);
    jpalpha = g(xalpha).'*d;

    CPG = (jalpha > j0 + rho * jp0 * alpha); % Criterio de paso grande.
    CPP = (jpalpha < sigma * jp0) && (~CPG); % Criterio de paso pequeno.

    i0 = i0 + 1;

    if CPG                                   % Alpha cumple CPG
        alphag  = alpha;
    elseif CPP                               % Alpha cumple CPP
        alphap  = alpha;
    else                                     % Alpha cumple CPA
        index = 1;
        fprintf(fid,'salida de wolfe index=%i \n',index);
        fprintf(fid,'salida de wolfe alpha=%e \n',alpha);
        return;
    end
end

index = -1;
return;

end


function [x,iteracion,K]=Interpolacion_cuadratica(x0,d,c1,c2,f,g,itmax,to
global A bb N R Rb
%El m todo de interpol quad busca aproximar phi(alpha)=f(x0+alpha*d) por
%polinomio q(alpha)=a*alpha^2 + b*alpha+c, siendo d una direccion de
iteracion=-1;
%K=-10;
%descenso
```

```
g0=g(x0);
norm(g0)
%Tres puntos dados como datos:
x1=-2; x2=0 ; x3=2;
%Calculamos sus imgenes a travs de phi
phi1=f(x0+d*x1);
phi2=f(x0+d*x2);
phi3=f(x0+d*x3);
%COMENZAMOS EL MTODO CUADRTICO
%q=(xi)=a*xi^2+b*xi+c=phi(xi)=f(x0+xi*d)  i=1,2,3
%D=(phi1-phi2)*(phi2-phi3)*(phi3-phi1); %Denominador
%a = (x2-x3)*phi1+(x3-x1)*phi2+(x1-x2)*phi3/(-D);
%b = (x2^2-x3^2)*phi1+(x3^2-x1^2)*phi2+(x1^2-x2^2)*phi3/D;

%Clculo del mnimo de la parbola
%x=-b/2*a;
%0=1.e-14
phi0=f(x0);

%Realizamos el proceso de interpolacin cuadrtica
for k=1:itmax

    iteracion=k;
    %Construimos el mnimo
    D=(x2^2-x3^2)*phi1+(x3^2-x1^2)*phi2+(x1^2-x2^2)*phi3; %numerador
    D2=(x2-x3)*phi1+(x3-x1)*phi2+(x1-x2)*phi3; %denominador
    %nodos
            fprintf(fic2,'nodo 1= %e phi1= %e\n',x1,phi1);
            fprintf(fic2,'nodo 2= %e phi2= %e\n',x2,phi2);
            fprintf(fic2,'nodo 3= %e phi3= %e\n',x3,phi3);
        if abs(D2)<tol %vemos si la funcin es una recta (D2=0)
            x=x2;
            phi=phi1;
            fprintf(fic2,'primer caso degenerado \n');
```

```
            %Test de parada con regla de Goldstein
            [K]=Goldstein_Rule(x,phi,phi0,d,c1,c2,g0);
            if K==2
                fprintf(fic2,'El test ha pasado exitosamente todos lo
                return
            elseif K==0
                fprintf(fic2,'Este punto no ha pasado uno de los crit
            else
                fprintf(fic2,'El punto no ha pasado ninguno de los do
            end
        else
            %Construmos el vrtice de la parbola
            x=0.5*D/D2;
            fprintf(fic2,'el valor del vertice de la parabola es: %e\n',x
            fprintf(fic2,'el valor de la imagen del vertice es: %e\n',f(x
            phi=f(x0+d*x);
            fprintf(fic2,'nuevo punto \n');
            %Test de parada con regla de Goldstein
            [K]=Goldstein_Rule(x,phi,phi0,d,c1,c2,g0);
            if K==2
                fprintf(fic2,'El test ha pasado exitosamente todos los c
                return
            elseif K==0
                fprintf(fic2,'Este punto no ha pasado uno de los criteri

            %Test de lateralidad 1
            if (x-x1)*(x-x3)<0
                C=(x-x2)*(x-x3);
                %Test de lateralidad 2
                if C<0
                    if phi<phi2
                        %x est en el intervalo
                        x1=x2;
                        phi1=phi2;
```

```
                    %ademas
                    x2=x;
                    phi2=phi;
                else
                    %Se cambia el extremo
                    x3=x;
                    phi3=phi;
                end
            else
                if phi<phi2
                    %Se cambia el extremo
                    phi3=phi2;
                    x3=x2;
                    %ademas
                    x2=x;
                    phi2=phi;
                else
                    x1=x;
                    phi1=phi;
                end
            end %test de lateralidad x2-x3
        else
            x=x2;
            phi=phi1;
            fprintf(fic2,'segundo caso degenerado \n');
            %Test de parada con regla de Goldstein
            [K]=Goldstein_Rule(x,phi,phi0,d,c1,c2,g0);
            if K==2
                fprintf(fic2,'El test ha pasado exitosamente todos lo
                return
            elseif K==0
                fprintf(fic2,'Este punto no ha pasado uno de los crit
            else
                fprintf(fic2,'El punto no ha pasado ninguno de los do
```

```
                end
            end %lateralidad x1-x3
        end
    end %if D2
    %test absoluto
    if abs(x-x2)<tol
        fprintf(fic2,'Convergencia de los v rtices de las par bolas
        return
    end
end %for


function [K]=Goldstein_Rule(x,phi,phi0,d,c1,c2,g0)
%Busqueda inexacta de Goldstein (sin derivadas)
%f(x_k + alpha*d_k)<= f(x_k)+ c2*alpha*g(x0)*d
%f(x_k + alpha*d_k)>= f(x_k)+ c1*alpha*g(x0)*d
%0<c1<c2<1
%SALIDA: Goldstein se cumple si K=2



pendiente=g0'*d;
%test pendiente negativa
% if pendiente>0
%     fprintf('pendiente positiva =%e',pendiente);
%     return
% end

%Test para ver si las c_i han sido seleccionadas correctamente
if c1<0
    disp('selecci n de c1 negativa, debe ser positiva')
else
    if c2<c1
        disp('selecci n de c1 debe ser menor que c2 ')
    else
```

```
        if c2>1
            disp('Selecci n demasiado grande, c2 debe ser menor a 1')
        end
    end
end

K=0;
inferior=phi0 + c2*x*pendiente;
superior=phi0 + c1*x*pendiente;
%CONDICINES DE GOLSTEIN
if phi <= superior
    disp('la 1  condici n de Goldstein se cumple')
%     h=1; %constante l gica
    K=K+1;
else
%     h=-1;
    K=K-1;
    disp('la 1  condici n de Goldstein no se cumple')
end

if phi >= inferior
    disp('la 2  condici n de Goldstein se cumple')
%     k=1; %constante l gica
    K=K+1;
else
%     k=-1;
    K=K-1;
    disp('la 2  condici n de Goldstein no se cumple')
end
%K=k+h;
%K=2; el m todo pasa el test de Goldstein
end

function z=normaL2_quad(u,x,solexa,iquad)
```

```
num_pas=length(x)-1;
z=0;

%Regla del punto medio
if iquad==1
    for i=2:num_pas+1
        h=x(i)-x(i-1);
        aprox_h=(u(i)+u(i-1))/2;
        exa=solexa((x(i)+x(i-1))*0.5);
        z=z+h*(aprox_h-exa)^2;

    end

else
    if iquad==2
        %Cuadratura de Gauss con dos puntos
        for i=2:num_pas+1
        h=x(i)-x(i-1);
        val_1h=(u(i)+u(i-1))*0.5 + (u(i)-u(i-1))*0.5/sqrt(3);
        val_2h=(u(i)+u(i-1))*0.5 - (u(i)-u(i-1))*0.5/sqrt(3);
        xg1= (x(i)+x(i-1))*0.5 + (x(i)-x(i-1))*0.5/sqrt(3);
        xg2= (x(i)+x(i-1))*0.5 - (x(i)-x(i-1))*0.5/sqrt(3);
        aprox_1=solexa(xg1);
        aprox_2=solexa(xg2);
        z=z+h*((val_1h-aprox_1)^2+(val_2h-aprox_2)^2)*0.5;
        end
    end
end


z=sqrt(z); %calculamos por tanto la norma L2
```

```
end


function z=H1semi(u,x,dfunf,iquad)

%areg = 1 malla regular
%areg = 0 malla no regular


num_pas=length(x)-1;
z=0;

%Calculo del cuadrado de la funci n para obtener el integrando
if iquad==1
    %Regla del punto medio
for i=2:num_pas+1
        h=x(i)-x(i-1);
        aprox_derh=(u(i)-u(i-1))/h;
        aprox_der=dfunf((x(i)+x(i-1))*0.5);
        z=z+h*(aprox_derh-aprox_der)^2;

end
else
    if iquad==2
        %Cuadratura de Gauss con dos puntos
        for i=2:num_pas+1
        h=x(i)-x(i-1);
        aprox_derh=(u(i)-u(i-1))/h;
        xg1= (x(i)+x(i-1))*0.5 + (x(i)-x(i-1))/(2*sqrt(3));
        xg2= (x(i)+x(i-1))*0.5 - (x(i)-x(i-1))/(2*sqrt(3));
        aprox_der_1=dfunf(xg1);
        aprox_der_2=dfunf(xg2);
        z=z+h*((aprox_der_1-aprox_derh)^2+(aprox_der_2-aprox_derh)^2)*0.5
        end
    end
```

```
end

z=sqrt(z); %calculamos por tanto la norma L2

end


function [sol,t,index_conver,A,bb] = PNGC_parabola(y,par,pc,L,alph,bet,p,
%y es la malla del FEM
global N
%Iniciamos el gradiente

salida1=fopen('gradiente','w');
fic=fopen('gradCuadWolfe','w');
fic2=fopen('gradInterp','w');
[x0,itmax,itmax2,tol,tol2,tol3,rho,sigma,gama,alphainit]=datos_new;

%Inicializamos los par metros de salida
i0=1;
index=-1;
err=0;
index_conver=0;
sol=x0; %punto inicial

%-------------------------
%COMEZAMOS EL ENSAMBLADO MEF
%-------------------------
%Ensamblado(devuelve tres diagonales de A y el vector segundo miembro)
% [vd,vld,vud,bb]=assembly_Poncelet(N,x,h,p,q,f);

%Ensamblado de la matriz del problema A
[vd,vld,vud,vmd,vmld,vmud]=assembly_Poncelet_A(N,y,h,p,q);
%Creamos la Matriz A del m todo
vd= vd+vmd;
vld= vld+vmld;
```

```
vud=vud+vmud;
%Imposicion de las condiciones de contorno para la matriz A
%y construcci n de A con matriz sparse
[A]=BC_A(N,vd,vld,vud);

for i=1:itmax
    %Ensamblado del vector del segundo miembro b
    [bb]=NL_assembly_Poncelet_b(N,sol,h,f);
    %Imposicion de las condiciones de contorno para el vector b
    [bb]=BC_b(alph,bet,bb,L,vud,vld);

    [tf,tg,hg,hf]=datos_extra(A,bb,pc);

if i==1
    %Datos iniciales m todo de gradiente
    tgradiente = tg(x0);
    td= - tgradiente;
    %con precondicionamiento
    hgradiente=hg(x0); %No se cambia de signo la direccion
if pc==0
    hgradiente=tg(x0); %No se cambia de signo la direccion
end
    hd=-hgradiente; % direcci n de descenso inicial
    nor_grad=sqrt(td'*hd);

    %debemos minimizar la funci n g(alpha)=f(xk+alpha*dk);
    fprintf(salida1,'*****************\n');
    fprintf(salida1,'index=0  norgrad=0 alpha=0 \n');
    escribe_v('x=',sol,salida1);
    escribe_v('d=',hd,salida1);
    fprintf(salida1, 'error inicial =%e ',err);
    escribe_v('norm_grad =',nor_grad,salida1);
    %escribe_cabecera(fic,sol,d,err);
end
```

```
%ESCOGEMOS UN ALPHA  PTIMO
if par ==1
%Regla de Wolfe
   [alpha,i0,index]= WolfeBiseccion(tf,tg,sol,hd,rho, sigma, alphainit, g
elseif par==0
%Interpolaci n cuadr tica con regla de Goldstein
    [alpha,i0,index]=Interpolacion_cuadratica(sol,hd,rho,sigma,hf,hg,itma
end

%ACTUALIZAMOS EL PUNTO
    solold=sol;
    sol=sol+alpha*hd;
%MPLEMENTAMOS LA FRMULA DE POLAK-RIBI RE
    hgold=hg(solold); %guardamos el gradiente anterior
if pc==0
    hgold=tg(solold);
end
    tgold=tg(solold);
    nor_grad=(tgold'*hgold); %calculamos su norma
    hgradiente=hg(sol); %gradiente nuevo con condicionante
if pc==0
    hgradiente=tg(sol);
end
    tgradiente=tg(sol); %gradiente
%Polak-Ribi re
    beta=(tgradiente'*(hgradiente-hgold))/nor_grad;
    if beta<0
        beta=0;
    end
%NUEVA DIRECCI N DE DESCENSO
    hd=-hgradiente+beta*hd;

%TEST DE PARADA MEDIANTE LA NORMA DEL GRADIENTE y DIFERENCIA CON EL ITERA
```

```
    t=sqrt(nor_grad);
    err=solold-sol;
    r=sqrt(err'*err);

    fprintf(salida1,'*****************\n');
    fprintf(fic2,'********%i*********\n',i);
    fprintf(fic2,'alpha = %e \n',alpha);
    fprintf(salida1,'index=%i  norgrad=%e alpha=%e i0=%i index=%i norerr=
    escribe_v('x=',sol,salida1);
    escribe_v('d=',hd,salida1);
    escribe_v('x=',sol,fic2);
    escribe_v('d=',hd,fic2);
    %Primer test de parada, norma gradiente
    if t<tol
        fprintf('Se cumple el test de parada del gradiente\n');
        fprintf('Iteracion %i\n', i);
        %escribe_vo('La solucion es:',sol); %escribe por pantalla sol
        fprintf('La norma del gradiente %10.3e\n',t);
        fprintf('Diferencia iterante anterior %10.3e\n',r);
        index_conver=i;
        %Escribimos en el fichero MEF el vector soluci n
%         fprintf(ofi,'\n\n* El vector soluci n es \n ');
%         escribe_v('uh=',sol,ofi);
          residuo = A*sol-bb;
          residuo = residuo';
          normInf = norm(residuo,Inf);
        return;
    else
    %Segundo test de parada, distancia entre iterantes
        if r<tol3
            fprintf('Se cumple el test de parada del incremento\n');
            fprintf('Iteracion %i\n', i);
%             escribe_vo('La solucion es:',sol); %escribe por pantalla so
            fprintf('Diferencia iterante anterior %10.3e\n',r);
```

```
            fprintf('La norma del gradiente %10.3e\n',t);
            index_conver=i;
            %Escribimos en el fichero MEF el vector soluci n
%             fprintf(ofi,'\n\n* El vector soluci n es \n ');
%             escribe_v('uh=',sol,ofi);
            residuo = A*sol-bb;
            residuo = residuo';
            normInf = norm(residuo,Inf);
        return;
        end
    end

end

if index_conver==0
    fprintf('No hubo convergencia\n');
end
fclose(salida1);
fclose(fic);
fclose(fic2);
end

function vaffineA=affineA(x,a,b)
%Affine increasign map fomr [a,b] onto [0,1]
vaffineA=(x-a)./(b-a);
end

function vaffineAInv=affineAInv(x,a,b)
vaffineAInv=(b-a).*x+a;
end

function [b]=NL_assembly_Poncelet_b(N,uh,h,f)
%Realiza el ensamblado del vector b
%Inicializamos a cero todas las componentes de b
b=zeros(1,N+1);
```

```
  for i = 1:N
    %Calculamos el paso por Poncelet
    pto2=h(i)/2;
    ev=f((uh(i)+uh(i+1))/2);
    %Calculamos vector b
    b(i)=b(i)+pto2*ev;
    b(i+1)=b(i+1)+pto2*ev;
  end

%   for i = 1:N
%     %Calculamos el paso por Simpson
%     pto2=h(i)/6;
%     ev=f((uh(i)+uh(i+1))/2);
%     ev1=f(uh(i+1));
%     ev2=f(uh(i+1));
%     %Calculamos vector b
%     b(i)=b(i)+pto2*(ev1+2*ev);
%     b(i+1)=b(i+1)+pto2*(2*ev+ev2);
%   end


end


function [vd,vld,vud,vmd,vmld,vmud]=assembly_Poncelet_A(N,x,h,p,q)
%Realiza el clculo y ensamblado de la matriz de rigidez(vd,vld,vud)
%y de masa (vmd,vmld,vmud)
%Inicializamos a cero todas las diagonales principales
vd=zeros(1,N+1);
vld=zeros(1,N);
vud=zeros(1,N);
vmd=zeros(1,N+1);
vmld=zeros(1,N);
vmud=zeros(1,N);
```

```
  for i = 1:N
     %Calculamos punto medio
     xm=(x(i)+x(i+1))./2;
     pto=p(xm)./h(i);
     pto1=(h(i)*q(xm))./4;
     %Calculamos diagonal de la matriz de rigidez
     vd(i)=vd(i)+pto;
     vd(i+1)=vd(i+1)+pto;
     %Calculamos subdiagonal de la matriz de rigidez
     vld(i)=-pto;
     %Calculamos superdiagonal de la matriz de  rigidez
     vud(i)=-pto;
     %Calculamos diagonal de la matriz de masa
     vmd(i)=vmd(i)+pto1;
     vmd(i+1)=vmd(i+1)+pto1;
     %Calculamos subdiagonal de la matriz de masa
     vmld(i)=pto1;
     %Calculamos superdiagonal de la matriz de  masa
     vmud(i)=pto1;
  end

end


function [A]=BC_A(N,vd,vld,vud)
%Fijamos las condiciones de contorno para sacar A, la matriz del problema
     %Caso Dirichlet-Dirichlet
     vd(1)=1;
     vud(1)=0;
     vd(N+1)=1;
     vld(N)=0;

     vld(1)=0;
     vud(N)=0;
```

```
%Construimos matriz A
A = sparse(1:N+1,1:N+1,vd);
A = A+sparse(1:N,2:N+1,vud,N+1,N+1)+sparse(2:N+1,1:N,vld,N+1,N+1);


end

function [bb]=BC_b(alph,bet,bb,L,vud,vld)

global N a21 an_1n

a21=vld(1);
an_1n=vud(N);

%Fijamos las condiciones de contorno para el vector b
%Para el vector de segundo miembro b solo hace falta modificar
%Los valores 1 y N+1 para los nodos inicial y final.

%Caso Dirichlet-Dirichlet
bb(1)=alph/L(1);
bb(N+1)=bet/L(3);

bb(2) = bb(2)-alph*a21/L(1);
bb(N) = bb(N) - bet*an_1n/L(3);

bb=bb';

end

%ARCHIVO DE DATOS MEF
global N
%Extremo espacial izquierdo:
fprintf(ofi,'\n***** DATOS DEL PROBLEMA *****');
a = 0;
fprintf(ofi,'\n* Extremo espacial izquierdo: a = %-.15E.', a);
```

```
%Extremo espacial derecho:
b = 1;
fprintf(ofi,'\n\n* Extremo espacial derecho: b = %-.15E.', b);

%Numero de elementos
N=100;
fprintf(ofi,'\n\n* Numero de elementos N = %d.', N);

%Funciones del problema
funp= @p_NL_1;
funq=@q_NL_1;
funf=@f_NL_1;
fexact=@exact_NL_1;
dfunf=@df_NL_1;
%Condiciones de contorno
%L(1)*u(a)+L(2)*u'(a) = alph
%L(3)*u(b)+L(4)*u'(b) = bet
L=[1 0 1 0];
fprintf(ofi,'\n\n* vector de condiones de contorno (Dirichlet-Neumann-Di
fprintf(ofi,'\n\n* L= %-.15E.', L);
%Condiciones de contorno x=a & x=b
alph = 0;
fprintf(ofi,'\n\n* alpha = %-.15E.', alph);
bet=0;
fprintf(ofi,'\n\n* beta = %-.15E.', bet);

%Vector acumulador de puntos de la malla [] si es uniforme
vmesh=[];
len=length(vmesh);
if len == 0
    fprintf(ofi,'\n\n* el vector de puntos de acumulacion es vacio');
else
    fprintf(ofi,'\n\n* vector de puntos de acumulacion');
    fprintf(ofi, '\n \n* vmesh = ');
```

```
    fprintf(ofi,' %4.4f ', vmesh);
end
%Constantes para la funci n de acumulaci n
k=1;
s=10;



%-------------------------------------------------------------
%Opci n para soluci n exacta
 ex = 1;
 fprintf(ofi,'\n\n* Opcion de solucion exacta: ex = %i', ex);
 if (ex == 1)
     fprintf(ofi,' (la solucion exacta es conocida).');
 else
     fprintf(ofi,' (la solucion exacta no es conocida).');
 end


function [tf,tg,hg,hf]=datos_extra(A,bb,pc)
global N
%Sistema original
tg=@(x) A*x-bb;
tf=@(x) 0.5*x'*A*x-bb'*x; %no se usa en este caso


%ESTABLECEMOS EL PRECONDICIONAMIENTO
if pc == 1
    %Precondicionamiento con Fact de Cholesky incompleta
    M=ichol(A);
    M=M*M';
    inv = M\eye(N+1);
elseif pc == 2
    %Precondicionamiento con factorizaci n ILU
    [L,U]=ilu(A);
```

```
    M=L*U;
    inv = M\eye(N+1);
elseif pc == 3
    %Precondicionamiento con Inversa
    inv = A\eye(N+1);
else
    inv = eye(N+1);
end
%Nuestro sistema ser  inv'*A*inv*z = inv'b donde nuestra x = inv*z
W=inv*A;
Wb=inv*bb;
hg = @(x) inv* tg(x); %gradiente precondicionado
hf = @(x) 0.5*x'*W*x-Wb'*x; %f precondicionado

end


function [x0,itmax,itmax2,tol,tol2,tol3,rho,sigma,gama,alphainit]=datos_n
global N
%Par metros b squeda inexacta
rho=0.2;
sigma=0.8;

%Tolerancia convergencia norma del gradiente
tol=1.e-6;
%tolerancia distancia entre iterantes
tol3=1.e-20;
%Tolerancia par metro Interpolaci n cuadr tica
tol2=tol^3;
%Iterantes m ximos GC
itmax=10000;
%Iterantes m ximos Interpolaci n cuadr tica
itmax2=1000;
%Par metros Wolfe
alphainit=10;
```

```
gama=2;
%Punto inicial (Distinguimos entre la matriz reducida y la que no lo es)

x0=zeros(1,N+1);
x0=x0';

% x0=zeros(1,N+1);
% for i=2:N
%       x0(i)=1;
% end
% x0(1)=0; x0(N+1)=0;
% x0=x0';

end

function derf = df_NL_1(x)
%Problema de Bratu
% theta = 4.79871456;
% theta = 1.5171645;
% derf=-theta*tanh(0.5*(x-0.5)*theta);
%otro
derf=exp(x)/sqrt(1-exp(2*x));


end

function vf = f_NL_1(uh)
%Problema de Bratu
lambda = 3.513830719;
vf=lambda.*exp(uh);


end

function vp = p_NL_1(x)
```

```
%Problema de Bratu
vp=1;

end

function vq = q_NL_1(x)
%Problema de Bratu
vq=0;

end

function vexact = exact_NL_1(x)
% Problema de Bratu
theta= 4.79871456;
theta = 1.5171645;
vexact=-2*log((cosh(0.5*theta*(x-0.5)))/cosh(0.25*theta));



end
```

Bibliografía

[A+20] Neculai Andrei et al. *Nonlinear conjugate gradient methods for unconstrained optimization*. Springer, 2020.

[BF84] Haïm Brézis and Análisis Funcional. Teoría y aplicaciones. *Alianza Editorial*, 1984.

[CCMT89] Philippe G Ciarlet, Philippe Gaston Ciarlet, Bernadette Miara, and Jean-Marie Thomas. *Introduction to numerical linear algebra and optimisation*. Cambridge University Press, 1989.

[Céa64] Jean Céa. Approximation variationnelle des problèmes aux limites. In *Annales de l'institut Fourier*, volume 14, pages 345–444, 1964.

[CM16] François Clément and Vincent Martin. The lax-milgram theorem. a detailed proof to be formalized in coq. *arXiv preprint arXiv:1607.03618*, 2016.

[LB10] Mats G Larson and Fredrik Bengzon. The finite element method: theory, implementation, and practice. *Texts in Computational Science and Engineering*, 10:23–44, 2010.

[Moh14] Adel Mohsen. A simple solution of the bratu problem. *Computers & Mathematics with Applications*, 67(1):26–33, 2014.

[Noc92] Jorge Nocedal. Theory of algorithms for unconstrained optimization. *Acta numerica*, 1:199–242, 1992.

[NW06] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[Rav83] Pierre-Arnaud Raviart. Introduction à l'analyse numérique des équations aux dérivées partielles. 1983.

[Rei80] Laure Reinhart. *Sur la résolution numérique de problemes aux limites non linéaires par des méthodes de continuation.* PhD thesis, 1980.

[Sha78] David F Shanno. Conjugate gradient methods with inexact searches. *Mathematics of operations research*, 3(3):244–256, 1978.

[SY06] Wenyu Sun and Ya-Xiang Yuan. *Optimization theory and methods: nonlinear programming*, volume 1. Springer Science & Business Media, 2006.

[TM21] Kevin Tolle and Nicole Marheineke. Extended group finite element method. *Applied Numerical Mathematics*, 162:1–19, 2021.

[Vog02] Curtis R Vogel. *Computational methods for inverse problems.* SIAM, 2002.