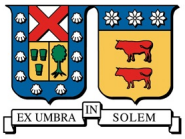


UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

Tarea N°2
Sistemas Lineales, Interpolación e Integración Numérica

Computación Científica

IMP-458 - 2022

Departamento Ing. Mecánica UTFSM

Martín Achondo Mercado

Rol: 201860005-9

Profesor: Franco Perazzo M.

17 de Junio de 2022

Resumen

En el presente trabajo se buscará implementar métodos numéricos para resolver 2 problemas planteados.

El primero consiste en encontrar las fuerzas sobre una armadura plana con 3 cargas externas de 10, 15 y 20 [ton]. Para esto, se plantearon las ecuaciones de conservación y se resolvió el sistema lineal resultante mediante factorización LU. Este mismo porcedimiento se realizó considerando viento de 1.6 [ton] en la dirección +x y -x, obteniendo variaciones comprendidas entre 1 % y 10 % para las fuerzas de las barras. De esto, se pudo obtener que el caso más desfavorable fue en el cual el viento sopla en dirección +x dado el aumento en promedio de las fuerzas sobre las barras y que se encuentra la barra con mayor carga (35.66 [ton]) en compresión.

Para el segundo problema se requería interpolar la geometría de un gancho de grúa para así obtener una descripción del esfuerzo normal a lo largo de una sección. Con la interpolación se pudo integrar numéricamente mediante dos métodos para obtener el área (1189 [mm^2]), radio al centroide de 46.3 [mm] y radio al eje neutro de 42.3 [mm]. De esta manera, se pudo evaluar el esfuerzo en la sección para una carga excéntrica de 2000 [N], resultando un esfuerzo máximo de 182 [MPa]. Por último se pudo obtener que la posición en donde el esfuerzo es nulo esta comprendida entre el radio al centroide y el radio al eje neutro, dando un valor de 45.8 [mm].

Índice

1. Introducción

En el presente trabajo se implementará un código para resolver los enunciados del problema presentado. Estos incluyen la resolución de un sistema lineal de ecuaciones para encontrar las fuerzas de las barras de una armadura y la interpolación junto con integración numérica para describir el estado de esfuerzo de un gancho de grúa bajo flexión y carga axial. Los códigos se elaborarán en Fortran dada su gran funcionalidad para métodos numéri cos .*Eneldocumentopresentarnlosresultad*

1.1. Presentación de los Problemas

En el presente se pide resolver lo siguiente:

1. Para la siguiente armadura, se pide obtener el valor de cada fuerza sobre todas las barras. Este sistema lineal que aparece deber ser resuelto mediante la factorización LU. Se considera la articulación 1 como fija en dirección vertical y horizontal y la articulación 8 fija en la dirección vertical. El ángulo es $\alpha = 45°$ mostrado en la figura. Además, se cuentan con 3 cargas en los nodos 2, 5 y 6 con los valores en [ton] en la figura:

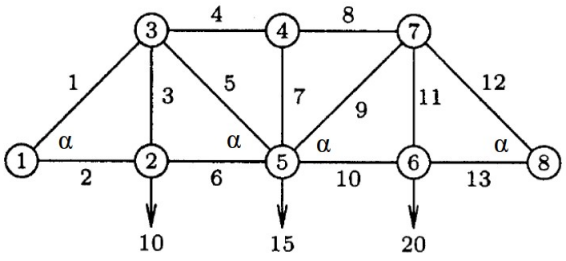
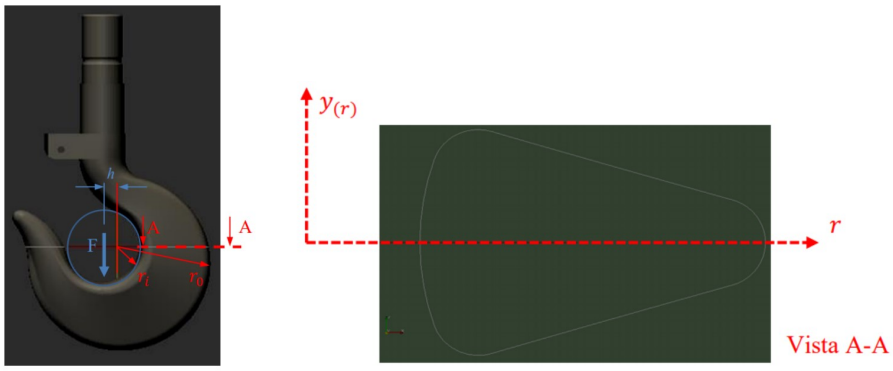


Figura 1: Armadura a resolver

   Adicionalmente se debe volver a calcular todas las fuerzas pero considerando la acción del viento (1.6 ton) en las direcciones +x y -x. Sumado a esto, se se deberá calcular la inversa de la matriz obtenida con el sistema lineal y por último, comparar los valores de las fuerzas con la librería de Blas-Lapack.

2. Se pide encontrar el esfuerzo normal $\sigma(r)$ en la sección de un gancho de grúa presentado en la siguiente figura:



(a) Gancho (b) Vista A-A

Figura 2: Modelo del gancho de grúa

El gancho se encuentra sometido a una carga excentrica de 2000 [N]. Los radios internos y externos desde el centro de la curvatura son 24[mm] y 74.3[mm] respectivamente. La distancia entre la línea de acción de la fuerza y el centro de la curvatura es 6 [mm]. Para esto se dispone de la tabla 7 presentada en el anexo con los pares (r_j, y_j) que modelan la geometría. Con esta se podrá interpolar mediante splines cúbicos los puntos medios y así integrar numéricamente para obtener el área, la distancia al eje del centroide y la distancia al eje neutro. Con todo lo anterior se podrá encontrar los valores del esfuerzo para cada punto r_j, $\sigma(r_j)$ y así, mediante interpolación inversa, encontrar el punto donde el esfuerzo es nulo.

2. Metodología

2.1. Marco Teórico

A continuación se presenta una breve teoría de los métodos a elaborar para ambos problemas.

2.1.1. Factorización LU

Es un método numérico que permite factorizar una matriz A como el producto de una matriz triangular inferior L y una matriz triangular superior U, de tal forma que: $A = LU$. Por componentes se puede visualizar como:

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} = \begin{pmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & u_{nn} \end{pmatrix} \tag{1}$$

Para generar esta factorización eficientemente, una eliminación Gaussiana a la matriz A para así obtener U, de tal manera que los factores de las operaciones filas se guardan y formas a L. Notas que la aparición de L mediante este método forma solo 1 en la diagonal.

2.1.2. Sistema Lineal con LU

Teniendo la matriz factorizada mediante LU, un sistema lineal $Ax = b$ puede ser descrito de la siguiente manera:

$$\begin{aligned} Ly &= b \\ Ux &= y \end{aligned} \tag{2}$$

De esta forma, resolver para el vector x implica un dos pasos:

- Resolver un sistema triangular inferior para y usando "sustitución hacia adelante".
- Resolver un sistema triangular superior para x usando "sustitución hacia atrás".

Este método es útil para resolver m sistemas lineales con la misma matriz A.

$$Ax_j = b_j \tag{3}$$

Si ya se tiene la factorización LU para la matriz, resolver cada sistema implica resolver solo un sistema superior y uno inferior, lo que disminuye bastante el costo computacional.

2.1.3. LU para Matriz Inversa

Con la factorización LU de la matriz A, resulta simple obtener la matriz inversa A^{-1}. Para ello, se debería resolver n sistemas lineales de la forma:

$$Ac_j = \mathrm{e}_j \tag{4}$$

En donde e_j corresponden a vectores de la base canónica de $\mathbb{R}^n$ y el vector solución c_j será la columna j de la matriz A^{-1}.

2.1.4. Interpolación Mediante Splines Cúbicos

El propósito es encontrar una función que interpole una tabla n pares de datos (x_i, y_i). Para ello, se utiliza Splines Cúbicos, lo que significa que cada sub intervalo $[x_i, x_{i+1}]$ es interpolado por un polinomio cúbico de la forma:

$$S_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3 \tag{5}$$

Los coeficientes de los $n - 1$ polinomios se encuentran considerando continuidad en la función y sus primeras y segundas derivadas:

$$\begin{cases} S_j(x_j) = y_j \\ S_j(x_{j+1}) = S_{j+1}(x_{j+1}) \\ S'_j(x_{j+1}) = S'_{j+1}(x_{j+1}) \\ S''_j(x_{j+1}) = S''_{j+1}(x_{j+1}) \end{cases} \tag{6}$$

Y se agrega la condición de "Natural Spline":

$$S''(x_1) = S''(x_n) = 0 \tag{7}$$

Encontrar los coeficientes de los $n - 1$ polinomios consiste en resolver un sistema trigonal de ecuaciones, formado por:

$$h_{j-1}c_{j-1} + 2(h_{j-1} + h_j)c_j + h_j c_{j+1} = \frac{3}{h_j}(a_{j+1} - a_j) - \frac{3}{h_{j-1}}(a_j - a_{j-1}) \tag{8}$$

En donde $a_j = y_j$ y $h_j = x_{j+1} - x_j$. Notar que la ecuación anterior resuelve el coeficiente c_j. El resto se deduce de las siguientes relaciones:

$$\begin{cases} b_j = \frac{1}{h_j}(a_{j+1} - a_j) - \frac{h_j}{3}(2c_j + c_{j+1}) \\ d_j = \frac{1}{3h_j}(c_{j+1} - c_j) \end{cases} \tag{9}$$

2.1.5. Interpolación Inversa

Tener una tabla de datos interpolada implica tener una función que modele los pares (x_i, y_i). Con el uso de cada interpolación por intervalo es posible obtener el cero de esta función interpoladora:

$$S_j(x^*) = 0 \tag{10}$$

Para esto, se realizan dos pasos:

- Encontrar el tramo i en el cual y_i e y_{i+1} tienen signos distintos.
- Resolver la ecuación no lineal para el i spline: $S_i(x) = 0$. Para este último paso se utilizará el método de Newton-Rapshon con punto de pártida x_i, dando la siguiente iteración:

$$x_{k+1} = x_k - \frac{S_i(x_k)}{S_i'(x_k)} \tag{11}$$

  La cual es sencilla de programar dado que S_i es un polinomio cúbico.

2.1.6. Integración Numérica

Consiste en una manera de evaluar integrales de manera numérica o simplemente integrar funciones donde no se tiene explícitamente la función a integrar, pero si, pares de datos $(x_i, f(x_i))$. Se desarrollarán 3 métodos cuando se tienen datos igualmente espaciados $h = \frac{(x_{i+1}-x_i)}{2} = \text{cste}$

- **Método del Trapecio**: Considera una interpolación lineal en cada sub intervalo $[x_i, x_{i+1}]$.

$$\int_{x_i}^{x_{i+1}} f(x)dx \approx h\big(f(x_i) + f(x_{i+1})\big) \tag{12}$$

- **Método de Simpson 1/3**: Considera una interpolación cuadrática cada 2 sub intervalos $[x_i, x_{i+2}]$.

$$\int_{x_i}^{x_{i+2}} f(x)dx \approx \frac{h}{3}\big(f(x_i) + 4f(x_{i+1}) + f(x_{i+2})\big) \tag{13}$$

- **Método de Simpson 3/8**: Considera una interpolación cúbica cada 3 sub intervalos $[x_i, x_{i+3}]$.

$$\int_{x_i}^{x_{i+3}} f(x)dx \approx \frac{3h}{8}\big(f(x_i) + 3f(x_{i+1}) + 3f(x_{i+2}) + f(x_{i+3})\big) \tag{14}$$

Regla Compuesta: De esta manera se puede obtener una regla compuesta para toda una tabla de datos desigualmente espaciados en el intervalo $[x_0, x_n]$

$$\int_{x_0}^{x_n} f(x)dx \approx \sum_{j=0}^{n-1} \int_{x_j}^{x_{j+1}} f(x)dx \tag{15}$$

En donde cada integral se evalúa con las reglas descritas anteriormente. El criterio para cada método es el siguinte:

- Utilizar Simpson 1/8 si se tienen dos segmentos consecutivos de igual longitud.

- Utilizar Simpson 3/8 si se tienen tres segmentos consecutivos de igual longitud.
- Utilizar Trapecio si los segmentos consecutivos son de diferente longitud.

2.2. Planteaiento Problema 1

Con lo dicho anteriormente, para la armadura de la figura 1, se formará un sistema lineal de ecuaciones para encontrar las fuerzas sobre cada barra. Para obtener la matriz A se considera la armadura sin cargas externas en el siguiente DCL. Se usa F_i para cada fuerza, R para las reacciones, las cuales son las incógnitas.

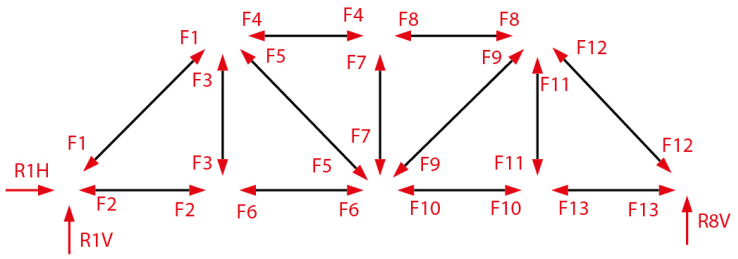


Figura 3: Diagrama de cuerpo libre para las barras de la armadura

En la imagen no se presentan las cargas externas, pero se tienen en consideración en cada nodo. Los valores reales de estas entregarán los vectores del lado derecho b. En el anexo (sección 7.1) se presenta un detalle de la obtención de las ecuaciones a partir de las condiciones de equilibrio:

$$\begin{aligned} \sum_j F_j &= 0 \\ \sum_j M_j &= 0 \end{aligned} \tag{16}$$

Dicho esto, mediante ese procedimiento aparece la siguiente matriz para el sistema a resolver

$Ax = b$:

$$A = \left[\begin{array}{cccccccccccccccc}
-0.707 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\
-0.707 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\
0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0.707 & 0 & 0 & -1 & -0.707 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0.707 & 0 & 1 & 0 & 0.707 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0.707 & 1 & 0 & 0 & -0.707 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -0.707 & 0 & -1 & 0 & -0.707 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0.707 & 0 & 0 & -0.707 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.707 & 0 & 1 & 0.707 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.707 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.707 & 0 & 0 & 0 & -1
\end{array}\right]$$

Con el vector x dado por:

$$x = \left[\begin{array}{cccccccccccccccc} F_1 & F_2 & F_3 & F_4 & F_5 & F_6 & F_7 & F_8 & F_9 & F_{10} & F_{11} & F_{12} & F_{13} & R_{1H} & R_{1V} & R_{8V} \end{array}\right]^T \quad (17)$$

Y el vector b:

$$x = \left[\begin{array}{cccccccccccccccc} E_{1H} & E_{1V} & E_{2H} & E_{2V} & E_{3H} & E_{3V} & E_{4H} & E_{4V} & E_{5H} & E_{5V} & E_{6H} & E_{6V} & E_{7H} & E_{7V} & E_{8H} & E_{8V} \end{array}\right]^T \quad (18)$$

Se formula los siguientes vectores b para cada situación:

1. Sin viento:

$$x = \left[\begin{array}{cccccccccccccccc} 0 & 0 & 0 & -10 & 0 & 0 & 0 & 0 & 0 & -15 & 0 & -20 & 0 & 0 & 0 & 0 \end{array}\right]^T \quad (19)$$

2. Con viento +x:

$$x = \left[\begin{array}{cccccccccccccccc} 0.8 & 0 & 0 & -10 & 0.8 & 0 & 0 & 0 & 0 & -15 & 0 & -20 & 0 & 0 & 0 & 0 \end{array}\right]^T \quad (20)$$

3. Con viento -x:

$$x = \left[\begin{array}{cccccccccccccccc} 0 & 0 & 0 & -10 & 0 & 0 & 0 & 0 & 0 & -15 & 0 & -20 & -0.8 & 0 & -0.8 & 0 \end{array}\right]^T \quad (21)$$

Formando 3 sistemas lineales $Ax_j = b_j$. La matriz A obtenida de este sistema lineal será

factorizada mediante LU, sec. 2.1.1, y así se podrá resolver para las fuerzas en los casos que exista o no viento (cambia solo el vector b), detallado en sec. 2.1.2 y además, se podrá obtener la matriz inversa A^{-1}, detallado en sec. 2.1.3. Por último se compararán los resultados con la librería Blas-Lapack. Para la comparación se caclula el error, en donde x_a corresponde al valor aproximado (obtenido por el código elaborado) y x_r el entregado por la librería.

$$\epsilon_r = \frac{\|x_a - x_r\|}{\|x_r\|} \tag{22}$$

2.3. Planteamiento Problema 2

Como se dijo anteriormene, se tiene la tabla 7 en el anexo con los pares de datos (r_j, y_j) para la sección del gancho, figura 2. Con estos datos se realizará el siguiente procedimiento:

1. Se interpolarán los datos de la tabla 7 mediante lo dicho en la sección 2.1.4 (splines cúbicos natural) para obtener pares de datos $(r_{j+1/2}, y_{j+1/2})$

2. Con la nueva tabla con los datos interpolados, mediante integración numérica (sección 2.1.6), se calcularán las integrales para la sección del gancho:

$$\begin{aligned} A &= \int_{r_i}^{r_o} 2y(r)dr \\ R &= \frac{1}{A}\int_{r_i}^{r_o} 2ry(r)dr \\ r_n &= \frac{A}{\int_{r_i}^{r_o} \frac{2y(r)}{r}dr} \end{aligned} \tag{23}$$

   Mediante la regla del trapecio y combinación de trapecio y Simpson para datos desigualmente espaciados.

3. Se calculará el esfuerzo normal para cada posición con:

$$\sigma(r) = \frac{F}{A} + \frac{F(R+h)(r_n - r)}{A(R - r_n)r} \tag{24}$$

   Y con la tabla formada, mediante interpolación inversa (sección 2.1.5), se encontrará la posición r^* tal que $\sigma(r^*) = 0$

3. Resultados

En esta sección se presentarán los resultados de ambos problemas.

3.1. Problema 1

Se presentan los valores de las fuerzas de las barras de las armaduras y las reacciones en el nodo 1 y 8.

Tabla 1: Fuerzas para las 3 configuraciones

Fuerza [ton]	Sin viento	Viento +x	Viento -x	Estado
F_1	-28.289	28.006	-28.571	Compresión
F_2	20.000	20.600	18.600	Tracción
F_3	10.000	10.000	10.000	Tracción
F_4	-30.000	-30.340	-30.340	Compresión
F_5	14.144	13.861	14.427	Tracción
F_6	20.000	20.600	18.600	Tracción
F_7	0.000	0.000	0.000	-
F_8	-30.000	-30.400	-30.400	Compresión
F_9	7.072	7.355	6.789	Tracción
F_{10}	25.000	25.200	24.000	Tracción
F_{11}	20.000	20.000	20.000	Tracción
F_{12}	-35.361	-35.644	-35.078	Compresión
F_{13}	25.000	25.200	24.000	Tracción
R_{1H}	0.000	-1.600	1.600	-
R_{1V}	20.000	19.800	20.200	-
R_{8V}	25.000	25.200	24.800	-

Se presenta la el cálculo de la matriz inversa A^{-1}.

$$A^{-1} = \begin{bmatrix}
-0 & -0 & -0 & 1.061 & 0.354 & 1.061 & 0.354 & 0.707 & -0 & 0.707 & -0 & 0.354 & 0.354 & 0.354 & -0 & -0 \\
-0 & -0 & 1 & -0.75 & 0.75 & -0.75 & 0.75 & -0.5 & 1 & -0.5 & 1 & -0.25 & 0.75 & -0.25 & 1 & -0 \\
-0 & -0 & -0 & -1 & -0 & -0 & -0 & -0 & -0 & -0 & -0 & -0 & -0 & -0 & -0 & -0 \\
-0 & -0 & -0 & 0.5 & -0.5 & 0.5 & 0.5 & 1 & -0 & 1 & -0 & 0.5 & 0.5 & 0.5 & -0 & -0 \\
0 & 0 & 0 & 0.354 & -0.354 & 0.354 & -0.354 & -0.707 & 0 & -0.707 & 0 & -0.354 & -0.354 & -0.354 & 0 & 0 \\
-0 & -0 & -0 & -0.75 & 0.75 & -0.75 & 0.75 & -0.5 & 1 & -0.5 & 1 & -0.25 & 0.75 & -0.25 & 1 & -0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-0 & -0 & -0 & 0.5 & -0.5 & 0.5 & -0.5 & 1 & -0 & 1 & -0 & 0.5 & 0.5 & 0.5 & -0 & -0 \\
-0 & -0 & -0 & -0.354 & 0.354 & -0.354 & 0.354 & -0.707 & -0 & -0.707 & -0 & 0.354 & 0.354 & 0.354 & -0 & -0 \\
-0 & -0 & -0 & -0.25 & 0.25 & -0.25 & 0.25 & -0.5 & -0 & -0.5 & 1 & -0.75 & 0.25 & -0.75 & 1 & -0 \\
-0 & -0 & -0 & -0 & -0 & -0 & -0 & -0 & -0 & -0 & -0 & -1 & -0 & -0 & -0 & -0 \\
0 & 0 & 0 & 0.354 & -0.354 & 0.354 & -0.354 & 0.707 & 0 & 0.707 & 0 & 1.061 & -0.354 & 1.061 & 0 & 0 \\
-0 & -0 & -0 & -0.25 & 0.25 & -0.25 & 0.25 & -0.5 & -0 & -0.5 & -0 & -0.75 & 0.25 & -0.75 & 1 & -0 \\
-1 & -0 & -1 & -0 & -1 & -0 & -1 & -0 & -1 & -0 & -1 & -0 & -1 & -0 & -1 & -0 \\
-0 & -1 & -0 & -0.75 & -0.25 & -0.75 & -0.25 & -0.5 & -0 & -0.5 & -0 & -0.25 & -0.25 & -0.25 & -0 & -0 \\
-0 & -0 & -0 & -0.25 & 0.25 & -0.25 & 0.25 & -0.5 & -0 & -0.5 & -0 & -0.75 & 0.25 & -0.75 & -0 & -1
\end{bmatrix}$$

Adicionalmente se calculan los errores obtenidos respecto a la librería Blas-Lapack.

Tabla 2: Errores respecto a Blas-Lapack

Situación	Sin viento	Viento +x	Viento -x
ϵ_a	2.73E-16	3.84E-16	3.12E-16

3.2. Problema 2

Se presenta la tabla para los datos interpolados:

Tabla 3: Datos Interpolados para pares (r_j, y_j)

r [mm]	y [mm]	r [mm]	y [mm]	r [mm]	y [mm]	r [mm]	y [mm]
24.0000	0.0000	28.7067	15.1881	60.6412	10.0251	73.7227	6.0646
24.0214	1.1806	29.4725	15.4957	63.0952	9.5269	73.9054	5.3162
24.0428	2.2726	30.7897	15.7739	64.8578	9.1702	74.0272	4.4052
24.1940	5.9255	32.1068	15.7861	66.6205	8.8107	74.1489	3.2997
24.3451	6.0237	33.7595	15.5324	68.0733	8.5115	74.1753	3.0358
24.4945	6.7701	35.4121	15.1510	69.5261	8.2204	74.2017	2.6654
24.6438	8.1315	37.3608	14.7377	70.5309	8.0253	74.2332	2.0456
24.8910	9.7681	39.3096	14.3592	71.5357	7.8121	74.2647	1.6038
25.1383	10.7108	41.8123	13.8579	72.0714	7.6603	74.2791	1.4585
25.8331	12.4514	44.3150	13.3423	72.6071	7.3653	74.2935	0.6931
26.5280	13.5181	51.2511	11.9275	73.0736	6.9208	74.2968	0.3641
27.2344	14.2456	58.1873	10.5240	73.5400	6.3660	74.3000	0.0000
27.9409	14.7633						

Con los datos interpolados (tabla 3) se presenta la comparación con los pares entregados en el problema para el contorno del gancho.

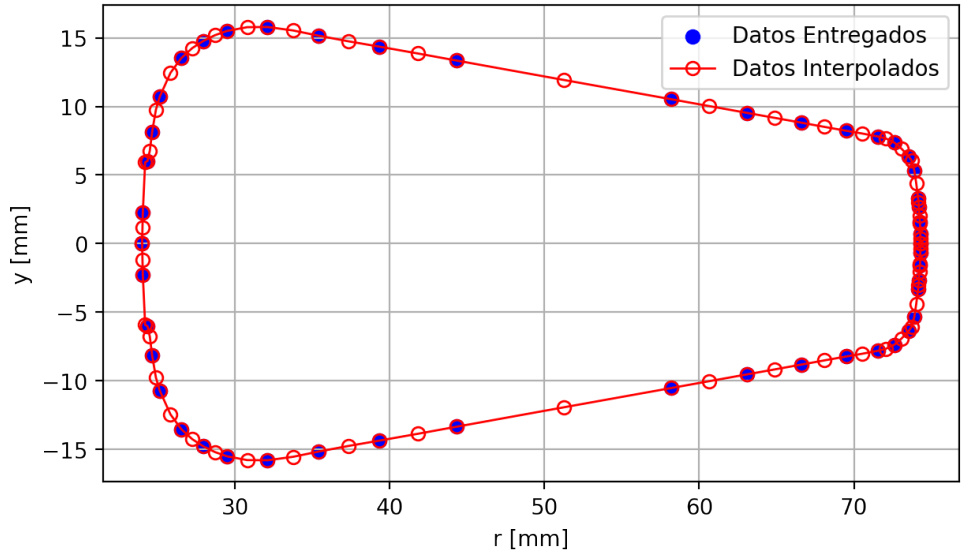


Figura 4: Comparación valores interpolados en geometría del gancho

Adicionalmente, con los datos interpolados (tabla 3) se presentan los resultados para los cálculos de la geometría de la sección del gancho:

Tabla 4: Resultados para el área, radio al centroide y radio al eje neutro

Variables	Trapecio	Compuesta
A [mm^2]	1188.417	1189.092
R [mm]	46.346	46.381
r_n [mm]	42.281	42.339

Con los datos de las dos tablas antes mencionadas, se puede construir una tabla para el esfuerzo normal en función del radio.

Tabla 5: Datos para pares (r_j, σ_j)

r [mm]	σ [MPa]	r [mm]	σ [MPa]	r [mm]	σ [MPa]	r [mm]	σ [MPa]
24.0000	182.0320	28.7067	119.3951	60.6412	-48.8014	73.7227	-75.6299
24.0214	181.6916	29.4725	111.0962	63.0952	-54.6820	73.9054	-75.9374
24.0428	181.3519	30.7897	97.7874	64.8578	-58.6311	74.0272	-76.1415
24.1940	178.9687	32.1068	85.5715	66.6205	-62.3715	74.1489	-76.3448
24.3451	176.6166	33.7595	71.5915	68.0733	-65.3087	74.1753	-76.3888
24.4945	174.3195	35.4121	58.9171	69.5261	-68.1231	74.2017	-76.4328
24.6438	172.0518	37.3608	45.4124	70.5309	-70.0018	74.2332	-76.4852
24.8910	168.3568	39.3096	33.2460	71.5357	-71.8277	74.2647	-76.5376
25.1383	164.7331	41.8123	19.2851	72.0714	-72.7804	74.2791	-76.5615
25.8331	154.9234	44.3150	6.9011	72.6071	-73.7190	74.2935	-76.5854
26.5280	145.6263	51.2511	-21.0997	73.0736	-74.5252	74.2968	-76.5909
27.2344	136.6616	58.1873	-42.4251	73.5400	-75.3210	74.3000	-76.5962
27.9409	128.1490						

Con lo presentado, se presenta un gráfico para el esfuerzo normal en la sección del gancho.

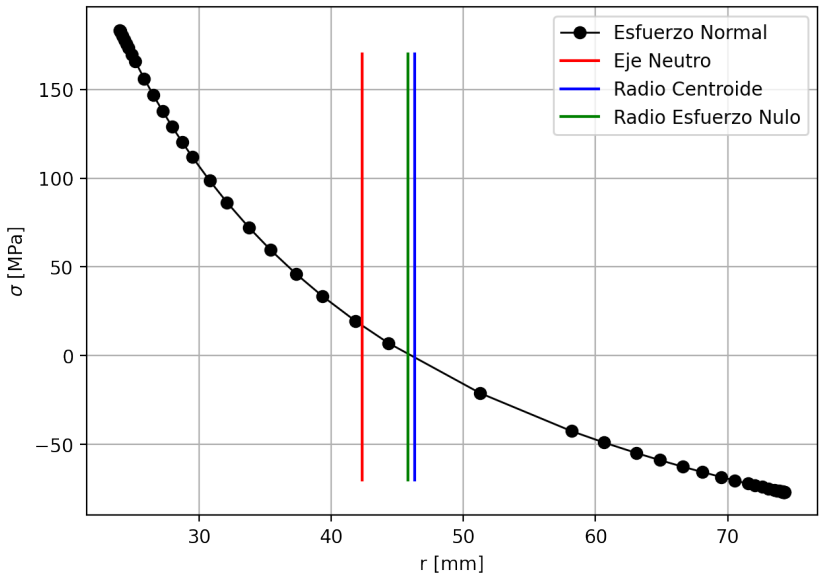


Figura 5: Esfuerzo normal en la sección del gancho

Se presenta la siguiente posición para que el esfuerzo normal sea nulo:

Tabla 6: Resultados interpolación inversa

r [mm]	$\sigma(r)$ [MPa]
45.878	7.7E-15

4. Análisis de Resultados

4.1. Problema 1

Luego de resolver para las fuerzas en la armadura se nota que la mayoría tiene una magnitud menor a 30 [ton]. Este resultado tiene sentido dada las cargas externas situadas en ella. Para llegar a este resultado se utilizó el método exacto por la factorización LU. De la misma forma, se pudo calcular la fuerza sobre todas las barras y reacciones cuando existe viento de 1.8 [ton] en la dirección +x y -x. El hecho de utilizar la factorización LU resultó de gran utilidad ya que para resolver estos 3 sistemas, la factorización se realizó solo 1 vez, tal como se estipuló en el marco teórico. Por la existencia del viento se pueden realizar dos conclusiones generales:

- El viento no cambia el estado de tracción o compresión de las barras, solo su magnitud.
- El efecto del viento genera variaciones en las fuerzas de las barras entre 1 % y 10 %.

Estos dos fenómenos visualizados se deben principalmente a que la carga que genera el viento es bastante menor a la suma de las cargas verticales que se encuentran en la armadura, por lo que el viento genera una perturbación del estado sin considerar el viento. Si existiera vientos de mayor carga, estas conclusiones podrían cambiar.

Analizando el peor escenario, se nota que para el caso en que el viento sopla en dirección -x, en promedio, las fuerzas en las barras tienden a aumentar en magnitud. Además, la carga máxima ocurre en este estado en la barra 12, alcanzando una fuerza de compresión de 35.6 [ton]. Con estos resultados sería interesante considerar la geometría y dimensiones de las barras para analizar como varían los factores de seguridad y verificar si alguna se encuentra con posibilidad de falla.

Para entender el significado de la matriz inversa obtenida, se realiza el siguiente análisis. En este trabajo se resolvió un sistema $Ax = b$, el cual proviene de una ecuación de balance, el equilibrio estático. Se nota que la matriz A puede representar las interacciones del sistema y el vector b los estímulos externos (viento y cargas). De esta forma el vector x da las fuerzas, es decir, la

cantidad que se balancea. Si se resuelve para las fuerzas como $x = A^{-1}b$, se puede concluir lo siguiente. La matriz inversa A^{-1}, la cual tiene información sobre las interacciones del sistema, representa la proporcionalidad entre los estímulos externos y la fuerza resultante en las barras. De esta forma, las columnas de la matriz inversa corresponden a las fuerzas resultantes cuando existen estímulos externos (b) unitarios. Con este razonamiento se calculó la matriz inversa para este problema.

Por último, los resultados para las fuerzas bajo las 3 condiciones estudiadas tienen variaciones de 1E-16 respecto a la librería Blas-Lapack, es decir, una variación bastante pequeña.

4.2. Problema 2

Con los datos de la tabla 7 presentada en el problema, se nota como a partir de interpolación con splines cúbicos se puede llegar a obtener más datos de manera confiable. Esto claramente sirve para modelar de mejor manera la geometría del gancho, el cual se visualiza en la figura 4. El hecho de utilizar splines cúbicos reduce bastante el error respecto a interpolaciones de splines lineales o de todo tipo sin utilizar splines. Esto se debe a que se utiliza un polinomio cúbico para cada tramo, conservando continuidad en la curva, primera y segundas derivadas. De todas formas, para cantidades grandes de datos puede aumentar el costo computacional.

Teniendo una mayor cantidad de datos producto de la interpolación, 3, se puede integrar de mejor manera las integrales presentadas para la geometría del gancho, dado que se tienen más nodos. Para esto se usó el método del trapecio y una combinación entre el método del trapecio y Simpson. De todas formas, se obtuvo una variación menor a 0.1 % entre ambos métodos. Esta pequeña variación se debe a la gran cantidad de nodos que se tienen con pequeñas longitudes de intervalos. De todas formas, se considera que la combinación del método del trapecio y Simpson entrega resultados más confiables dada su mayor convergencia.

Por último, con todo lo calculado se pudo encontrar el esfuerzo normal a lo largo de la sección del gancho. En la figura 5 se nota la curva del esfuerzo y las posiciones al radio del centroide y al eje neutro. De aquí, claramente el esfuerzo máximo es de 182 [MPa] en el radio interior. Esto se debe a la carga de flexión y axial existente, lo que provoca este esfuerzo de tracción en toda esa zona. Mediante interpolación inversa se pudo obtener la posición en donde el esfuerzo es nulo, dando 45.88 [mm]. Dado que se presenta una carga excéntrica, este valor queda contenido entre el radio al centroide y el radio al eje neutro, como lo muestra la figura 5.

5. Conclusión

Teniendo ya las simulaciones realizadas, se nota la gran importancia que tienen los métodos numéricos para resolver y modelar fenómenos físicos.

En el primer problema se pudo modelar una armadura para obtener las fuerzas sobre todas las barras y las reacciones. Esto se realizó aplicando las ecuaciones de equilibrio para formular un sistema lineal de ecuaciones. Este sistema obtenido fue resulto mediante la factorización LU. Es importante destacar que esta factorización fue de gran utilidad ya que se pudo resolver 3 situaciones distintas para la armadura, en donde se incluía el viento en ambas direcciones en el eje x o no se incluía. A modo de resultado, se obtuvo que el caso más desfavorable para la armadura es cuando existe un viento que sopla en la dirección +x ya que las fuerzas en promedio aumentan en las barras. Los resultados fueron comparados con la librería presentada obteniendo variaciones relativas de 1E-16, lo que implica convergencia.

Para el segundo problema, se pudo modelar el estado de esfuerzo para un gancho de grúa. Lo interesante fue que solo mediante ciertos puntos que modelan la geometría, con interpolación, se pudo obtener el doble de nodos así aumentar la precisión en los cálculos posteriores. La interpolación se llevó cabo utilizando la técnica de splines cúbicos. Con los datos interpolados se integró numéricamente la sección para obtener el área, radio al centroide y el radio al eje neutro. De esta manera, se pudo tener el valor del esfuerzo normal a lo largo de toda la sección. Así, mediante interpolación inversa se pudo obtener la posición en que el esfuerzo es nulo.

6. Referencias

[1] Steven C. Chapra Raymond P. Canale. (2007). Métodos numéricos para ingenieros. Mexico: The McGraw-Hill.

[2] Beer, F; Johnston, E.D.; DeWolf, J; Mazurek, D. (2010) Mecánica de Materiales, 5a edición: Mc Graw-Hill.

[3] Anderson, Bai, Bischof, Blackford, Demmel, Dongarra, Du Croz, Greenbaum, Hammarling, McKenney and Sorensen. LAPACK Users'Guide. Siam.

[4] Perazzo, F. Solución numérica de sistemas de ecuaciones no-lineales, IPM-458. (2022)

7. Anexos

7.1. Ecuaciones Armadura

Se muestran las ecuaciones de equilibrio para todos los nodos de la armadura.

Nodo 1:

$$\begin{aligned} -R_{1H} - F_1\cos(\pi/4) - F_2 &= E_{1H} \\ -R_{1V} - F_1\sin(\pi/4) &= E_{1V} \end{aligned} \tag{25}$$

Nodo 2:

$$\begin{aligned} F_2 - F_6 &= E_{2H} \\ -F_3 &= E_{2V} \end{aligned} \tag{26}$$

Nodo 3:

$$\begin{aligned} F_1\cos(\pi/4) - F_5\cos(\pi/4) - F_4 &= E_{3H} \\ F_1\sin(\pi/4) + F_5\sin(\pi/4) + F_3 &= E_{3V} \end{aligned} \tag{27}$$

Nodo 4:

$$\begin{aligned} F_4 - F_8 &= E_{4H} \\ F_7 &= E_{4V} \end{aligned} \tag{28}$$

Nodo 5:

$$\begin{aligned} -F_{10} + F_5\cos(\pi/4) + F_6 - F_9\cos(\pi/4) &= E_{5H} \\ -F_9\sin(\pi/4) - F_5\sin(\pi/4) - F_7 &= E_{5V} \end{aligned} \tag{29}$$

Nodo 6:

$$\begin{aligned} -F_{13} + F_{10} &= E_{6H} \\ -F_{11} &= E_{6V} \end{aligned} \tag{30}$$

Nodo 7:

$$\begin{aligned} F_8 - F_{12}\cos(\pi/4) + F_9\cos(\pi/4) &= E_{7H} \\ F_{11} + F_9\sin(\pi/4) + F_{12}\sin(\pi/4) &= E_{7V} \end{aligned} \tag{31}$$

Nodo 8:

$$\begin{aligned} F_{13} + F_{12}\cos(\pi/4) &= E_{8H} \\ -R_{8V} - F_{12}\sin(\pi/4) &= E_{8V} \end{aligned} \tag{32}$$

7.2. Tabla Problema 2

Tabla 7: Datos presentados para pares (r_j, y_j)

r [mm]	y [mm]	r [mm]	y [mm]
24.0000	0.0000	63.0952	9.5269
24.0428	2.2726	66.6205	8.8107
24.3451	6.0237	69.5261	8.2204
24.6438	8.1315	71.5357	7.8121
25.1383	10.7108	72.6071	7.3653
26.5280	13.5181	73.5400	6.3660
27.9409	14.7633	73.9054	5.3162
29.4725	15.4957	74.1489	3.2997
32.1068	15.7861	74.2017	2.6654
35.4121	15.1510	74.2647	1.6038
39.3096	14.3592	74.2935	0.6931
44.3150	13.3423	74.3000	0.000
58.1873	10.5240		

7.3. Códigos Elaborados

7.3.1. Pregunta 1

```
1
2  program p1
3      implicit none
4      character(10) :: file1
5      integer,parameter :: n = 16
6      real(8),parameter :: tol = 1.0d-8
7      integer :: er,o(n),q,i,oi(n)
8      real(8) :: a(n,n),b(n),x(n),s(n),ai(n,n),a2(n,n),a3(n,n),e1,e2,e3,
           error_app,bi(n),az(n,n),si(n),ai1(n,n),xi(n),aco(n,n)
9      real(8) :: a1(n,n),b1(n),x1(n),b2(n),x2(n),b3(n),x3(n),bc(n),bc2(n)
           ,bc3(n)
10
11     file1 = "data_a.txt"
12     write(*,*) file1
13     call ReadFileA(a,n,file1)
14     call ReadFileB(b,n,"dat_b1.txt")
15     a1 = a
16     a2 = a
17     a3 = a
18     aco = a
19     ai1 = a
20     b1 = b
21     bc = b
```

```
22
23      call ReadFileB(b,n,"dat_b2.txt")
24      b2 = b
25      bc2 = b2
26      call ReadFileB(b,n,"dat_b3.txt")
27      b3 = b
28      bc3 = b3
29
30      call Ludecomp(a1,b1,n,tol,x1,er,o,s)
31
32      write(*,*)
33      write(*,*) 'Resultado Caso sin Viento'
34      do q=1,n
35          write(*,*) x1(q)
36      end do
37      write(*,*)
38
39
40
41
42      call Substitute(a1,o,n,b2,x2)
43
44      write(*,*)
45      write(*,*) 'Resultado Viento +x'
46      do q=1,n
47          write(*,*) x2(q)
48      end do
49      write(*,*)
50
51
52
53      call Substitute(a1,o,n,b3,x3)
54
55      write(*,*)
56      write(*,*) 'Resultado Viento -x'
57      do q=1,n
58          write(*,*) x3(q)
59      end do
60      write(*,*)
61
62      write(*,*)
63      write(*,*) 'Resultado Lapcack sin viento'
64      call sist_lineal(a,bc,n)
65
66    do q=1,n
67      write(*,*) bc(q)
68    end do
69
```

```
70
71      write(*,*)
72      write(*,*) 'Resultado Lapcack viento +x'
73      call sist_lineal(a2,bc2,n)
74
75     do q=1,n
76      write(*,*) bc2(q)
77     end do
78
79
80     write(*,*)
81     write(*,*) 'Resultado Lapcack viento -x'
82     call sist_lineal(a3,bc3,n)
83
84    do q=1,n
85     write(*,*) bc3(q)
86    end do
87
88    write(*,*)
89    write(*,*)'Error Sin Viento:'
90    write(*,*) error_app(x1,bc,n,e1)
91    write(*,*) 'Error Viento +x'
92    write(*,*) error_app(x2,bc2,n,e2)
93    write(*,*) 'Error Viento -x'
94    write(*,*) error_app(x3,bc3,n,e3)
95
96
97    call InverseMa(ai1,n,tol,oi,si,er,bi,xi,ai)
98    write(*,*)
99    write(*,*) 'Matriz Inversa'
100   do i=1,n
101      write(*,'(16F8.4)')(ai(i,q),q=1,n)
102   end do
103
104   write(*,*)
105   write(*,*) 'Comprobacion'
106   az = matmul(aco,ai)
107   do i=1,n
108      write(*,'(16F8.4)')(az(i,q),q=1,n)
109   end do
110
111 end program p1
112
113 real(8) function error_app(x1,x2,n,e)
114     implicit none
115     integer :: n,i,j
116     real(8),intent(in) :: x1(n),x2(n)
117     real(8) :: e,s1,s2
```

```
118     s1 = 0
119     s2 = 0
120     do i=1,n
121         s1 = s1 + (x1(i)-x2(i))**2
122         s2 = s2 + x2(i)**2
123     end do
124 
125     e = sqrt(s1/s2)
126 
127     error_app = e
128 
129 end function error_app
130 
131 
132 subroutine Ludecomp(a,b,n,tol,x,er,o,s)
133     implicit none
134     integer :: n
135     integer :: o(n),er
136     real(8) :: a(n,n),b(n),x(n)
137     real(8) :: s(n)
138     real(8) :: tol
139     er = 0
140     call Decompose(a,n,tol,o,s,er)
141     if (er.ne.-1) then
142         call Substitute(a,o,n,b,x)
143     end if
144 
145 end subroutine Ludecomp
146 
147 ! Calcula inversa (LU debe estar hecha)
148 subroutine InverseM(a,n,o,er,b,x,ai)
149     implicit none
150     integer :: o(n),er,i,j,n
151     real(8) :: a(n,n),b(n),x(n),ai(n,n)
152 
153     er = 0
154     if (er.eq.0) then
155         do i=1,n
156             do j=1,n
157                 if (i.eq.j) then
158                     b(j) = 1
159                 else
160                     b(j) = 0
161                 end if
162             end do
163             call Substitute(a,o,n,b,x)
164             do j=1,n
165                 ai(j,i) = x(j)
```

```
166             end do
167         end do
168     end if
169
170 end subroutine InverseM
171
172 subroutine InverseMa(a,n,tol,o,s,er,b,x,ai)
173     implicit none
174     integer :: n
175     integer :: o(n),er,i,j
176     real(8) :: a(n,n),b(n),x(n),ai(n,n)
177     real(8) :: s(n),tol
178
179     er = 0
180     call Decompose(a,n,tol,o,s,er)
181     if (er.eq.0) then
182         do i=1,n
183             do j=1,n
184                 if (i.eq.j) then
185                     b(j) = 1
186                 else
187                     b(j) = 0
188                 end if
189             end do
190             call Substitute(a,o,n,b,x)
191             do j=1,n
192                 ai(j,i) = x(j)
193             end do
194         end do
195     end if
196
197 end subroutine InverseMa
198
199
200
201 subroutine Decompose(a,n,tol,o,s,er)
202     implicit none
203     integer :: i,j,k,o(n),er,n
204     real(8) :: a(n,n),tol
205     real(8) :: s(n),factor
206
207     do i=1,n
208         o(i) = i
209         s(i) = abs(a(i,1))
210
211         do j=2,n
212             if (abs(a(i,j)).gt.s(i)) then
213                 s(i) = abs(a(i,j))
```

```
214             end if
215         end do
216     end do
217 
218     do k=1,n-1
219         call Pivot(a,o,s,n,k)
220         if (abs(a(o(k),k)/s(o(k))).lt.tol) then
221             er = -1
222             exit
223         end if
224         do i=k+1,n
225             factor = a(o(i),k)/a(o(k),k)
226             a(o(i),k) = factor
227             do j=k+1,n
228                 a(o(i),j) = a(o(i),j) - factor*a(o(k),j)
229             end do
230         end do
231     end do
232 
233     if (abs(a(o(k),k)/s(o(k))).lt.tol) then
234         er = -1
235     end if
236 
237 end subroutine Decompose
238 
239 
240 subroutine Pivot(a,o,s,n,k)
241     implicit none
242     integer :: ii,k,o(n),p,big,n
243     real(8) :: a(n,n),dummy
244     real(8) :: s(n)
245 
246     p = k
247     big = abs(a(o(k),k)/s(o(k)))
248 
249     do ii=k+1,n
250         dummy = abs(a(o(ii),k)/s(o(ii)))
251         if (dummy.gt.big) then
252             big = dummy
253             p = ii
254         end if
255     end do
256 
257     dummy = o(p)
258     o(p) = o(k)
259     o(k) = dummy
260 
261 end subroutine Pivot
```

```
262
263
264 subroutine Substitute(a,o,n,b,x)
265     implicit none
266     integer :: o(n),i,j,n
267     real(8) :: a(n,n),b(n),x(n),sum
268
269     do i=2,n
270         sum = b(o(i))
271         do j=1,i-1
272             sum = sum - a(o(i),j)*b(o(j))
273         end do
274         b(o(i)) = sum
275     end do
276
277     x(n) = b(o(n))/a(o(n),n)
278
279     do i=n-1,1,-1
280         sum = 0
281         do j=i+1,n
282             sum = sum + a(o(i),j)*x(j)
283         end do
284         x(i) = (b(o(i))-sum)/a(o(i),i)
285     end do
286
287 end subroutine Substitute
288
289
290 subroutine sist_lineal(A,b,n)
291     implicit none
292     integer, intent(in) :: n
293     real(kind=8) :: A(n,n),b(n)
294     integer :: IPIV(n),NRHS,LDA,LDB,INFO
295     NRHS = 1
296     LDA = n
297     LDB = n
298     call DGESV(n,NRHS,A,LDA,IPIV,b,LDB,INFO)
299 end subroutine
300
301
302
303 subroutine ReadFileA(a,n,file)
304     implicit none
305     character(10) :: file
306     integer :: i,j,stat,n
307     real(8) :: a(n,n)
308
309     open(unit=10,file=file,iostat=stat,action='read')
```

```
310
311         if(stat/=0) then
312             write(*,*)'Error al abrir el archivo con iostat',stat
313         end  if
314
315         do i=1,n
316             read(10,*)(a(i,j),j=1,n)
317         end do
318
319     close(unit=10,iostat=stat)
320
321     if(stat/=0) then
322         write(*,*)'Error al cerrar el archivo con iostat',stat
323     end if
324
325     write(*,*) 'Archivo Leido'
326     do i=1,n
327         write(*,'(16F8.4)')(a(i,j),j=1,n)
328     end do
329
330 end subroutine ReadFileA
331
332
333 subroutine ReadFileB(b,n,file)
334     implicit none
335     character(10) :: file
336     integer :: i,j,stat,n
337     real(8) :: b(n)
338
339     open(unit=10,file=file,iostat=stat,action='read')
340
341         if(stat/=0) then
342             write(*,*)'Error al abrir el archivo con iostat',stat
343         end  if
344
345         do i=1,n
346             read(10,*)b(i)
347         end do
348
349     close(unit=10,iostat=stat)
350
351     if(stat/=0) then
352         write(*,*)'Error al cerrar el archivo con iostat',stat
353     end if
354
355     write(*,*) 'Archivo Leido'
356     do i=1,n
357         write(*,*)b(i)
```

```
358     end do
359
360 end subroutine ReadFileB
```

7.3.2. Pregunta 2

Interpolar Datos

```
1
2  program interpolate
3      implicit none
4      integer, parameter :: n=25
5      real(8) :: r(n),y(n),a(n),b(n),c(n),d(n),r2(n-1),v(n-1),x(2*n-1),yf
           (2*n-1)
6      real(8) :: eval_spline
7      integer :: i,z
8
9      call ReadFile(r,y,n,"data_ry.txt")
10
11     a = y
12
13     call spline(n,r,a,b,c,d)
14
15     z = 1
16     do i=1,n-1
17         r2(i) = (r(i+1) + r(i))/2
18         v(i) = eval_spline(n,a,b,c,d,r,r2(i))
19         x(z) = r(i)
20         x(z+1) = r2(i)
21         yf(z) = y(i)
22         yf(z+1) = v(i)
23         z = z + 2
24     end do
25     x(2*n-1) = r(n)
26     yf(2*n-1) = y(n)
27
28     write(*,*)
29     write(*,*) 'Valores Interpolados r,y  (tabla 2)'
30     do i=1,2*n-1
31         write(*,*) x(i),yf(i)
32     end do
33
34     call WriteFile(x,yf,2*n-1,"data_ry_inter.txt")
35
36
37
38
39 end program interpolate
40
```

```
41
42  real(8) function eval_spline(n,a,b,c,d,x,v)
43      implicit none
44      integer, intent(in) :: n
45      real(8), intent(in) :: a(n),b(n),c(n),d(n),x(n),v
46      integer :: i
47      real(8) :: s,z
48
49      do i=1,n
50          if (x(i).le.v) then
51              if (x(i+1).ge.v) then
52                  z = v - x(i)
53                  s = a(i) + b(i)*z + c(i)*z**2 + d(i)*z**3
54              end if
55          end if
56      end do
57      eval_spline = s
58
59  end function eval_spline
60
61  subroutine spline(n,x,a,b,c,d)
62      implicit none
63      integer, intent(in) :: n
64      real(8), intent(in) :: x(n),a(n)
65      real(8), intent(out) :: b(n),c(n),d(n)
66      integer :: i,j
67      real(8) :: h(n),alfa(n),l(n),mu(n),z(n)
68
69      do i=1,n-1
70          h(i) = x(i+1) - x(i)
71      end do
72
73      do i=2,n-1
74          alfa(i) = (3d0/h(i))*(a(i+1)-a(i))-((3d0/h(i-1))*(a(i)-a(i-1)))
75      end do
76
77      l(1) = 1d0
78      mu(1) = 0d0
79      z(1) = 0d0
80
81      do i=2,n-1
82          l(i) = 2d0*(x(i+1)-x(i-1))-h(i-1)*mu(i-1)
83          mu(i) = h(i)/l(i)
84          z(i) = (alfa(i)-h(i-1)*z(i-1))/l(i)
85      end do
86
87      l(n) = 1d0
88      z(n) = 0d0
```

```
89      c(n) = 0d0
90
91      do j=n-1,1,-1
92          c(j) = z(j)-mu(j)*c(j+1)
93          b(j) = (a(j+1)-a(j))/h(j) - h(j)*(c(j+1)+2d0*c(j))/3d0
94          d(j) = (c(j+1)-c(j))/(3d0*h(j))
95      end do
96
97  end subroutine spline
98
99
100 subroutine ReadFile(a,b,n,file)
101     implicit none
102     character(11),intent(in) :: file
103     integer, intent(in) :: n
104     integer :: i,stat
105     real(kind=8) :: a(n),b(n)
106
107     open(unit=10,file=file,iostat=stat,action='read')
108
109         if(stat/=0) then
110             write(*,*)'Error al abrir el archivo con iostat',stat
111         end  if
112
113         do i=1,n
114             read(10,*)a(i),b(i)
115         end do
116     close(unit=10,iostat=stat)
117
118     if(stat/=0) then
119         write(*,*)'Error al cerrar el archivo con iostat',stat
120     end if
121
122     write(*,*)
123     write(*,*) "Archivo Leido:"
124     do i=1,n
125         write(*,*)a(i),b(i)
126     end do
127     write(*,*)
128
129 end subroutine ReadFile
130
131
132 subroutine WriteFile(a,b,n,file)
133     implicit none
134     character(17),intent(in) :: file
135     integer, intent(in) :: n
136     integer :: i,stat
```

```
137     real(kind=8) :: a(n),b(n)
138 
139     open(unit=20,file=file,iostat=stat,action='write')
140 
141         if(stat/=0) then
142             write(*,*)'Error al abrir el archivo con iostat',stat
143         end  if
144 
145         do i=1,n
146             write(20,'(2F15.10)')a(i),b(i)
147         end do
148     close(unit=20,iostat=stat)
149 
150     if(stat/=0) then
151         write(*,*)'Error al cerrar el archivo con iostat',stat
152     end if
153 
154     write(*,*)
155     write(*,*) "Archivo Guardado"
156     write(*,*)
157 
158 end subroutine WriteFile
```

Integrar Datos

```
1  
2  program integrate
3      implicit none
4      integer,parameter :: n=49
5      real(8) :: x(n),y(n),fi(n),A,R,rn
6  
7      call ReadFile(x,y,n,"data_ry_inter.txt")
8  
9      write(*,*)
10     write(*,*) 'Valores utilizando Trapsun (A,R,rn)'
11     fi(:) = 2*y(:)
12     call Trapsun(x,fi,n,A)
13     write(*,*) 'A:',A
14     fi(:) = 2*x(:)*y(:)
15     call Trapsun(x,fi,n,R)
16     R = R/A
17     write(*,*) 'R:',R
18     fi(:) = 2*y(:)/x(:)
19     call Trapsun(x,fi,n,rn)
20     rn = A/rn
21     write(*,*) 'rn',rn
22 
23     write(*,*)
24     write(*,*) 'Valores utilizando Uneven (A,R,rn)'
```

```
25      fi(:) = 2*y(:)
26      call Uneven(x,fi,n,A)
27      write(*,*) 'A:',A
28      fi(:) = 2*x(:)*y(:)
29      call Uneven(x,fi,n,R)
30      R = R/A
31      write(*,*) 'R:',R
32      fi(:) = 2*y(:)/x(:)
33      call Uneven(x,fi,n,rn)
34      rn = A/rn
35      write(*,*) 'rn',rn
36      write(*,*)
37
38  end program integrate
39
40
41  subroutine Trapsun(x,y,n,Int)
42          implicit none
43          integer, intent(in) :: n
44          real(8), intent(in) :: x(n),y(n)
45          real(8), intent(out) :: Int
46          integer :: i
47          real(8) :: sum
48
49          sum = 0
50          do i=2,n
51              sum = sum + (x(i)-x(i-1))*(y(i)+y(i-1))/2
52          end do
53          Int = sum
54
55  end subroutine Trapsun
56
57  subroutine Uneven(x,f,n,Int)
58      implicit none
59      integer, intent(in) :: n
60      real(8), intent(in) :: x(n),f(n)
61      real(8), intent(out) :: Int
62      integer :: k,j
63      real(8) :: h,hf,sum,Trap,Simp13,Simp38
64
65      h = x(2) - x(1)
66      k = 1
67      sum = 0
68      do j=2,n
69          hf = x(j+1) - x(j)
70          if (abs(h-hf).lt.0.000001) then
71              if (k.eq.3) then
72                  sum = sum + Simp13(h,f(j-3),f(j-2),f(j-1))
```

```
73                  k = k - 1
74              else
75                  k = k + 1
76              end if
77          else
78              if (k.eq.1) then
79                  sum = sum + Trap(h,f(j-1),f(j))
80              else
81                  if (k.eq.2) then
82                      sum = sum + Simp13(h,f(j-2),f(j-1),f(j))
83                  else
84                      sum = sum + Simp38(h,f(j-3),f(j-2),f(j-1),f(j))
85                  end if
86                  k = 1
87              end if
88          end if
89          h = hf
90      end do
91      Int = sum
92
93  end subroutine Uneven
94
95
96  real(8) function Trap(h,f0,f1)
97      real(8), intent(in) :: h,f0,f1
98      Trap = h*(f0 + f1)/2
99  end function Trap
100
101 real(8) function Simp13(h,f0,f1,f2)
102     real(8), intent(in) :: h,f0,f1,f2
103     Simp13 = 2*h*(f0 + 4*f1 + f2)/6
104 end function Simp13
105
106 real(8) function Simp38(h,f0,f1,f2,f3)
107     real(8), intent(in) :: h,f0,f1,f2,f3
108     Simp38 = 3*h*(f0+3*(f1+f2)+f3)/8
109 end function Simp38
110
111
112
113 subroutine ReadFile(a,b,n,file)
114     implicit none
115     character(17),intent(in) :: file
116     integer, intent(in) :: n
117     integer :: i,stat
118     real(kind=8) :: a(n),b(n)
119
120     open(unit=10,file=file,iostat=stat,action='read')
```

```
121
122         if(stat/=0) then
123             write(*,*)'Error al abrir el archivo con iostat',stat
124         end  if
125
126         do i=1,n
127             read(10,*)a(i),b(i)
128         end do
129     close(unit=10,iostat=stat)
130
131     if(stat/=0) then
132         write(*,*)'Error al cerrar el archivo con iostat',stat
133     end if
134
135     write(*,*)
136     write(*,*) "Archivo Leido:"
137     do i=1,n
138         write(*,*)a(i),b(i)
139     end do
140     write(*,*)
141
142 end subroutine ReadFile
```

Interpolación Inversa

```
1
2  program inverse_inter
3      implicit none
4      integer,parameter :: n=49
5      real(8) :: r(n),y(n),AA,RR,rn,h,e,F,M
6      real(8) :: a(n),b(n),c(n),d(n)
7      integer :: i
8      real(8) :: sigma(n),rx,eval_spline
9
10     call ReadFile(r,y,n,"data_ry_inter.txt")
11     AA = 1189.0921636889593
12     RR = 46.380989762466697
13     rn = 42.338586884955298
14     F = 20000
15     h = 6
16     e = RR - rn
17     M = F*(RR + h)
18
19     do i=1,n
20         sigma(i) = F/AA + M*(rn-r(i))/(AA*e*r(i))
21         write(*,*) r(i),sigma(i)
22     end do
23
24     call WriteFile(r,sigma,n,"data_rs_inter.txt")
```

```
25
26      a = sigma
27      call spline(n,r,a,b,c,d)
28
29      call findroot(n,r,a,b,c,d,rx)
30
31      write(*,*) 'Valor de: r , sigma(r)'
32      write(*,*)rx,eval_spline(n,a,b,c,d,r,rx)
33      write(*,*)
34
35  end program inverse_inter
36
37  subroutine findroot(n,r,a,b,c,d,xf)
38      implicit none
39      integer, intent(in) :: n
40      real(8), intent(in) :: r(n),a(n),b(n),c(n),d(n)
41      real(8), intent(out) :: xf
42      integer :: i,index
43      real(8) :: v1,K(4),w,tol
44
45      v1 = a(1)
46      do i=2,n
47          if (v1*a(i).lt.0) then
48              index = i - 1
49              exit
50          end if
51      end do
52
53      K = (/a(index),b(index),c(index),d(index)/)
54      w = r(index)
55
56      tol = 1.0d-9
57      call newton_rapshon(K,w,4,w,tol,xf)
58
59  end subroutine findroot
60
61
62  subroutine newton_rapshon(K,w,n,x0,tol,xf)
63      implicit none
64      integer, intent(in) :: n
65      real(8), intent(in) :: K(n),w,x0,tol
66      real(8), intent(out) :: xf
67      integer :: iter
68      integer, parameter :: imax = 500
69      real(8) :: xrold,xr,ea,fr,eval_f_df
70      xrold = x0
71      xr = xrold
72      iter = 0
```

```
73      do while (iter.lt.imax)
74          xrold = xr
75          iter = iter + 1
76          fr = eval_f_df(xrold,K,w,n)
77          xr = xrold - fr
78          if (xr.ne.0) then
79              ea = abs((xr-xrold)/xr)*100
80          end if
81          if(ea.lt.tol.or.iter.gt.imax) then
82              exit
83          end if
84      end do
85      write(*,*) 'Newton Rapshon resuelto: Iteracion, ea'
86      write(*,*) iter,ea
87      write(*,*)
88      xf = xr
89  end subroutine
90
91  real(8) function eval_f_df(x,K,w,n)
92      implicit none
93      integer, intent(in) :: n
94      real(8), intent(in) :: K(n),x,w
95      integer :: j
96      real(8)::sum1,sum2
97      sum1 = 0
98      do j = 1,n
99          sum1 = sum1 + K(j)*(x-w)**(j-1)
100     end do
101
102     sum2 = 0
103     do j = 1,n-1
104         sum2 = sum2 + (j)*K(j+1)*(x-w)**(j-1)
105     end do
106
107     eval_f_df = sum1/sum2
108
109 end function
110
111
112 real(8) function eval_spline(n,a,b,c,d,x,v)
113     implicit none
114     integer, intent(in) :: n
115     real(8), intent(in) :: a(n),b(n),c(n),d(n),x(n),v
116     integer :: i
117     real(8) :: s,z
118
119     do i=1,n
120         if (x(i).le.v) then
```

```
121             if (x(i+1).ge.v) then
122                 z = v - x(i)
123                 s = a(i) + b(i)*z + c(i)*z**2 + d(i)*z**3
124             end if
125         end if
126     end do
127     eval_spline = s
128
129 end function eval_spline
130
131 subroutine spline(n,x,a,b,c,d)
132     implicit none
133     integer, intent(in) :: n
134     real(8), intent(in) :: x(n),a(n)
135     real(8), intent(out) :: b(n),c(n),d(n)
136     integer :: i,j
137     real(8) :: h(n),alfa(n),l(n),mu(n),z(n)
138
139     do i=1,n-1
140         h(i) = x(i+1) - x(i)
141     end do
142
143     do i=2,n-1
144         alfa(i) = (3d0/h(i))*(a(i+1)-a(i))-((3d0/h(i-1))*(a(i)-a(i-1)))
145     end do
146
147     l(1) = 1d0
148     mu(1) = 0d0
149     z(1) = 0d0
150
151     do i=2,n-1
152         l(i) = 2d0*(x(i+1)-x(i-1))-h(i-1)*mu(i-1)
153         mu(i) = h(i)/l(i)
154         z(i) = (alfa(i)-h(i-1)*z(i-1))/l(i)
155     end do
156
157     l(n) = 1d0
158     z(n) = 0d0
159     c(n) = 0d0
160
161     do j=n-1,1,-1
162         c(j) = z(j)-mu(j)*c(j+1)
163         b(j) = (a(j+1)-a(j))/h(j) - h(j)*(c(j+1)+2d0*c(j))/3d0
164         d(j) = (c(j+1)-c(j))/(3d0*h(j))
165     end do
166
167 end subroutine spline
168
```

```
169 subroutine ReadFile(a,b,n,file)
170     implicit none
171     character(17),intent(in) :: file
172     integer, intent(in) :: n
173     integer :: i,stat
174     real(kind=8) :: a(n),b(n)
175 
176     open(unit=10,file=file,iostat=stat,action='read')
177 
178         if(stat/=0) then
179             write(*,*)'Error al abrir el archivo con iostat',stat
180         end  if
181 
182         do i=1,n
183             read(10,*)a(i),b(i)
184         end do
185     close(unit=10,iostat=stat)
186 
187     if(stat/=0) then
188         write(*,*)'Error al cerrar el archivo con iostat',stat
189     end if
190 
191     write(*,*)
192     write(*,*) "Archivo Leido:"
193     do i=1,n
194         write(*,*)a(i),b(i)
195     end do
196     write(*,*)
197 
198 end subroutine ReadFile
199 
200 subroutine WriteFile(a,b,n,file)
201     implicit none
202     character(17),intent(in) :: file
203     integer, intent(in) :: n
204     integer :: i,stat
205     real(kind=8) :: a(n),b(n)
206 
207     open(unit=20,file=file,iostat=stat,action='write')
208 
209         if(stat/=0) then
210             write(*,*)'Error al abrir el archivo con iostat',stat
211         end  if
212 
213         do i=1,n
214             write(20,'(2F10.4)')a(i),b(i)
215         end do
216     close(unit=20,iostat=stat)
```

```
217
218     if(stat/=0) then
219         write(*,*)'Error al cerrar el archivo con iostat',stat
220     end if
221
222     write(*,*)
223     write(*,*) "Archivo Guardado"
224     write(*,*)
225
226 end subroutine WriteFile
```