Examen

Isaac Rodríguez Bribiesca

Resumen Reporte con los resultados de la resolución de los 4 problemas del examen.

1. Problema 1

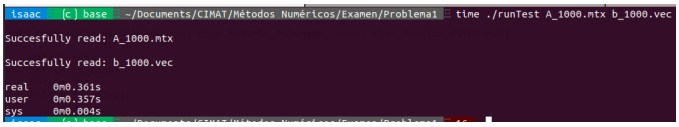
1.1. Metodología

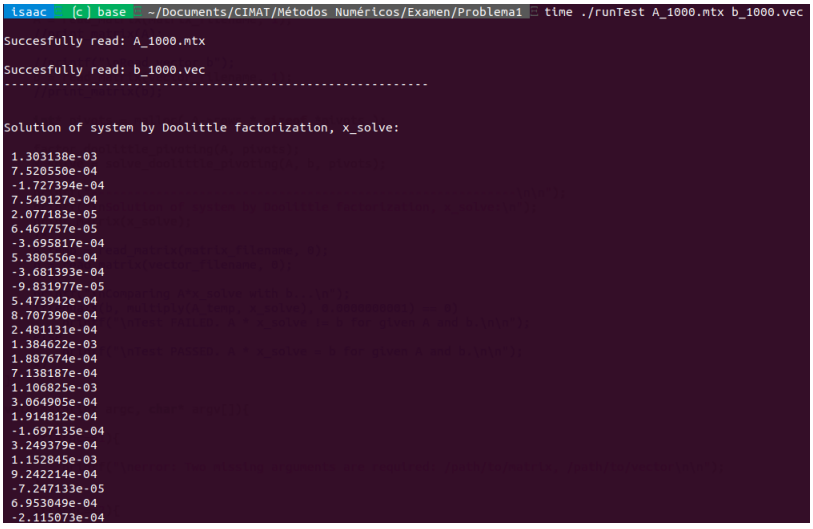
Para el caso de la matriz inversa A^{-1} se hizo uso de la factorización Doolittle y resolviendo n sistemas de sistemas de ecuaciones donde n es el tamaño de la matriz original A. Se realizó de esta forma ya este método nos da el beneficio de que una vez que se obtiene la factorización de $A = LU$ se pueden resolver sistemas de ecuaciones $Ax = b_i$ para diferentes vectores b_i haciendo uso sólamente de sustitución hacia adelante y hacia atrás.

Por la misma razón comentada previamente, para la resolución del sistema de ecuaciones lineales $Ax = b$ se uso la factorización Doolittle.

1.2. Ejemplo de prueba

Solución de sistema de ecuaciones lineales por Doolittle





Calculo de matriz inversa mediante Doolittle

```
isaac (c) base ~/Documents/CIMAT/Métodos Numéricos/Examen/Problema1 ./runTest A_1000.mtx

Succesfully read: A_1000.mtx
------------------------------------------------------------

Comparing A * A^-1 with identity matrix I...

Test PASSED. A * A^-1 = I for given A and b.
```

```
isaac (c) base ~/Documents/CIMAT/Métodos Numéricos/Examen/Problema1 time ./runTest A_1000.mtx > inverse.txt

real    0m2.155s
user    0m2.027s
sys     0m0.020s
```

1.3. Resultados

Se pudo observar en los resultados que a pesar de que se trataba de una matriz de tamaño 1000, se pudo resolver el sistema y calcular la matriz inversa en tiempo del orden de 2 segundos. Comprobando que se obtiene la matriz inversa A^{-1} multiplicándola con A obteniendo la matriz identidad.

2. Problema 2

2.1. Metodología

En la parte de métodos iterativos se implementó Gradiente Conjugado, por varios motivos. Por una parte se implementó el uso de matriz rala en la forma CSR ya que es muy eficiente a la hora de realizar multiplicación de matriz-vector y esto es justo lo que utiliza gradiente conjugado, además de producto punto entre vectores. Por otro lado, estas operaciones son altamente paralelizables por lo que se pudo aprovechar el uso de openmp para paralelizar los productos punto y producto matriz-vector.

Para el caso de los métodos directos se resolvió el sistema de ecuaciones mediante Gauss con pivoteo, Cholesky modificado y Doolittle.

2.2. Ejemplo de prueba

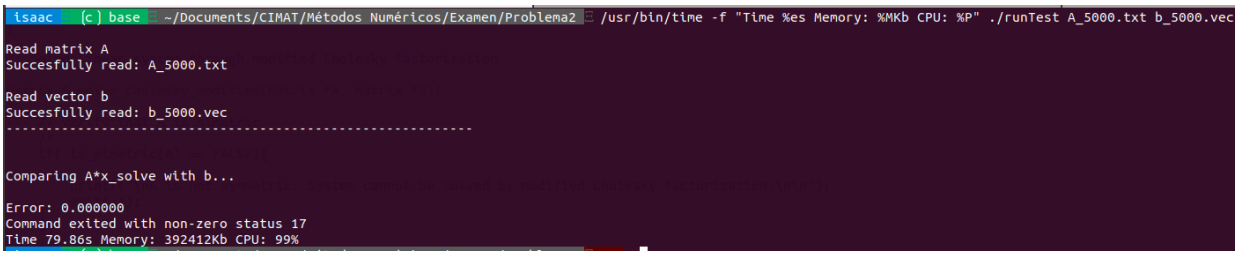
Gradiente conjugado

```
isaac (c) base ~/Documents/CIMAT/Métodos Numéricos/Examen/Problema2 1 /usr/bin/time -f "Time %es Memory: %MKb CPU: %P" g++ -std=c++11 -O2 -fopenmp conjGradient.cp
st -O2 && ./runTest A_5000.mtx b_5000.vec
Time 2.85s Memory: 195988Kb CPU: 99%

Read matrix of size: (5000,5000) with 68508 non zero entries.

Read vector of size: 5000

Read vector of size: 5000


Algorithm converged after 1339 iterations

5000 5000

Read vector of size: 5000


Error: 19.286317
```

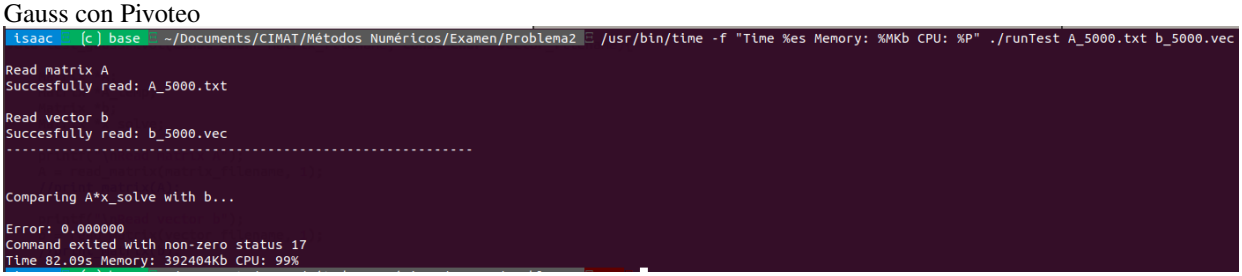
Doolittle

```
isaac (c) base ~/Documents/CIMAT/Métodos Numéricos/Examen/Problema2 17 /usr/bin/time -f "Time %es Memory: %MKb CPU: %P" ./runTest A_5000.txt b_5000.vec

Succesfully read: A_5000.txt

Succesfully read: b_5000.vec

Comparing A*x_solve with b...

Error: 0.000000
Command exited with non-zero status 17
Time 50.52s Memory: 392564Kb CPU: 99%
```

Cholesky Modificado



Gauss con Pivoteo



2.3. Resultados

Método	Tiempo ejecución [s]	Memoria [MB]	Error ($\|Ax - b\|$)
Gradiente conjugado	2.85	191.39	19.28
Doolittle	50.52	383.36	0
Gauss pivoteo completo	82.09	382.2	0
Cholesky modificado	79.86	383.21	0

Se pudo notar un gran resultado mediante gradiente conjugado combinando el uso de matriz rala CSR y openmp resolviendo el sistema en 2.85 segundos, debido a la eficacia de la codificación de CSR ya que el acceso se realiza en tiempo constante directo a los valores que son diferentes de cero, por lo que no tiene que iterar sobre toda la matriz ni revisar si algún elemento existe en la matriz o no. También se puede ver que en cuanto a tiempo de procesador usado, fue el más rápido, lo cual también es debido a la iteración más eficiente sobre las columnas de la matriz rala. Hay que tomar en cuenta que este algoritmo es iterativo por lo que en términos de error resultó mayor que los demás algoritmos, ya que se pone un límite de tolerancia de error, permitiendo tener una aproximación buena sin demasiadas iteraciones. Finalmente, en cuanto a memoria ocupada, también fue el que tuvo mejor desempeño, lo cual también fue debido a la codificación de la matriz. Una desventaja de este método es que se requiere que la matriz sea simétrica y positiva definida lo cual no siempre se cumple, aunque por otro lado tiene la característica de que converge en un número finito de iteraciones que no es mayor al tamaño de la matriz.

En cuanto a los restantes algoritmos, que son directos, se observó que el error fue cero, sin embargo, los tiempos de ejecución fueron superiores y la memoria usada también fue mayor. Dentro de estos algoritmos, se observó que Doolittle fue el más rápido, ya que tanto la factorización como las sustituciones hacia adelante y atrás son rápidas. Además de que esta factorización nos permite resolver múltiples sistemas de ecuaciones para diferentes valores del lado derecho de $Ax = b$.

Finalmente para el caso de la factorización por Cholesky modificado se caracteriza por trabajar con matrices simétricas definidas. Realizando la factorización de $A = LDL^T$. Así como con la factorización de Doolittle, Cholesky o cholesky modificado puede ser útil para resolver sistemas de ecuaciones linales de manera eficiente y como una prueba de que que una matriz es positiva definida, ya que de no serlo se encontrará con algún elemento de la diagonal que sea cero.

3. Problema 3

3.1. Metodología

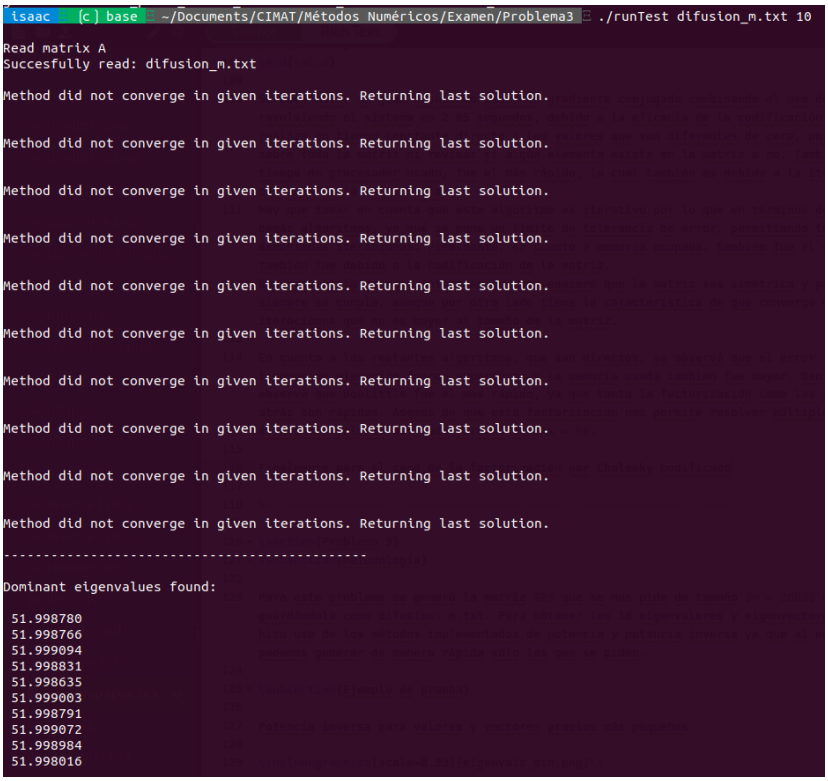
Para este problema se generó la matriz D que se nos pide de tamaño $n = 2000$ mediante un script de python guardándola como difusion_m.txt. Para obtener los 10 eigenvalores y eigenvectores más grandes y más pequeños se hizo uso de los métodos implementados de potencia y potencia inversa ya que al no requerir calcularlos todos, podemos generar de manera rápida sólo los que se piden.

3.2. Ejemplo de prueba

Potencia inversa para valores y vectores propios más pequeños

```
isaac (c) base ~/Documents/CIMAT/Métodos Numéricos/Examen/Problema3 130 time ./runTest difusion_m.txt 10

Read matrix A
Succesfully read: difusion_m.txt

Method did not converge in given iterations. Returning last solution.


Method did not converge in given iterations. Returning last solution.


Method did not converge in given iterations. Returning last solution.


Method did not converge in given iterations. Returning last solution.


Method did not converge in given iterations. Returning last solution.


Method did not converge in given iterations. Returning last solution.


Method did not converge in given iterations. Returning last solution.


Method did not converge in given iterations. Returning last solution.


Method did not converge in given iterations. Returning last solution.


Method did not converge in given iterations. Returning last solution.

-----------------------------------------


 16.000555
 16.001121
 16.000773
 16.000496
 16.000319
 16.000827
 16.000716
 16.000300
 16.000378
 16.000187


real    9m0.997s
user    8m58.505s
sys     0m2.432s
```

Potencia para valores y vectores propios más grandes



3.3. Resultados

Se pudo observar que a pesar de haber calculado sólo 10 valores y vectores propios, al tratarse de una matriz de tamaño 2000, el tiempo de ejecución fue de cerca de 10 minutos. Algo a considerar es que el tiempo de ejecución también dependerá de la precisión y el número de iteraciones que se fijen, lo cual a su vez también influirá en que tan buenos resultados obtengamos. Como se pudo comprobar con python los resultados se aproximan bastante, pero sin ser exactos. Algo que se podría mejorar es analizar si se puede realizar alguna simplificación debido a la forma de la matriz ya que tiene muchos ceros.

4. Problema 4

4.1. Metodología

Desarrollo de polinomio de Hermite.

GUAR. MÁS

$$L_i(x) = \prod_{\substack{j=0 \\ j\neq i}}^{2} \frac{x - x_j}{x_i - x_j} \qquad x = x_0 + sh \;,\; x_j = x_0 + jh$$

$$x - x_j = h(s-j)$$

$$x_i - x_j = h(i-j)$$

$$L_i^2(s) = \prod_{\substack{j=0 \\ j\neq i}}^{2} \frac{(s-j)^2}{(i-j)^2}$$

$$\int_{x_0}^{x_2} P_H(x) = y_0 \int_0^2 L_0^2(s)dx + y_1 \int_0^2 L_1^2(s)dx + y_2 \int_0^2 L_2^2(s)dx$$

$$+ (y_0' + \tfrac{3y_0}{h})h\int_0^2 s L_0^2(s)\,dx + y_2' h \int_0^2 (s-1) L_1^2(s)dx$$

$$+ (y_2' - \tfrac{3}{h}) \int_0^2 h(s-2) L_2^2(s)dx$$

Como $dx = h\,ds$,

$$\int_{x_0}^{x_2} P_H(x) = y_0 h \int_0^2 L_0^2(s)\,ds + y_1 h \int_0^2 L_1^2(s)\,ds + y_2 h \int_0^2 L_2^2(s)\,ds$$

$$+ (y_0' + \tfrac{3}{h})h^2 \int_0^2 s L_0^2(s)\,ds + y_2' h^2 \int_0^2 (s-1) L_1^2(s)\,d$$

$$+ (y_1' - \tfrac{3y_2}{h}) \int_0^2 \ldots L_1^2(s)\,ds$$

Dibujar Recortar Compartir

GUAR. MÁS

$$L_i(x) = \prod_{\substack{j=0 \\ j\neq i}}^{2} \frac{x - x_j}{x_i - x_j} \qquad x = x_0 + sh \;,\; x_j = x_0 + jh$$

$$x - x_j = h(s-j)$$

$$x_i - x_j = h(i-j)$$

$$L_i^2(s) = \prod_{\substack{j=0 \\ j\neq i}}^{2} \frac{(s-j)^2}{(i-j)^2}$$

$$\int_{x_0}^{x_2} P_H(x) = y_0 \int_0^2 L_0^2(s)dx + y_1 \int_0^2 L_1^2(s)dx + y_2 \int_0^2 L_2^2(s)dx$$

$$+ (y_0' + \tfrac{3y_0}{h})h\int_0^2 s L_0^2(s)\,dx + y_2' h \int_0^2 (s-1) L_1^2(s)dx$$

$$+ (y_2' - \tfrac{3}{h}) \int_0^2 h(s-2) L_2^2(s)dx$$

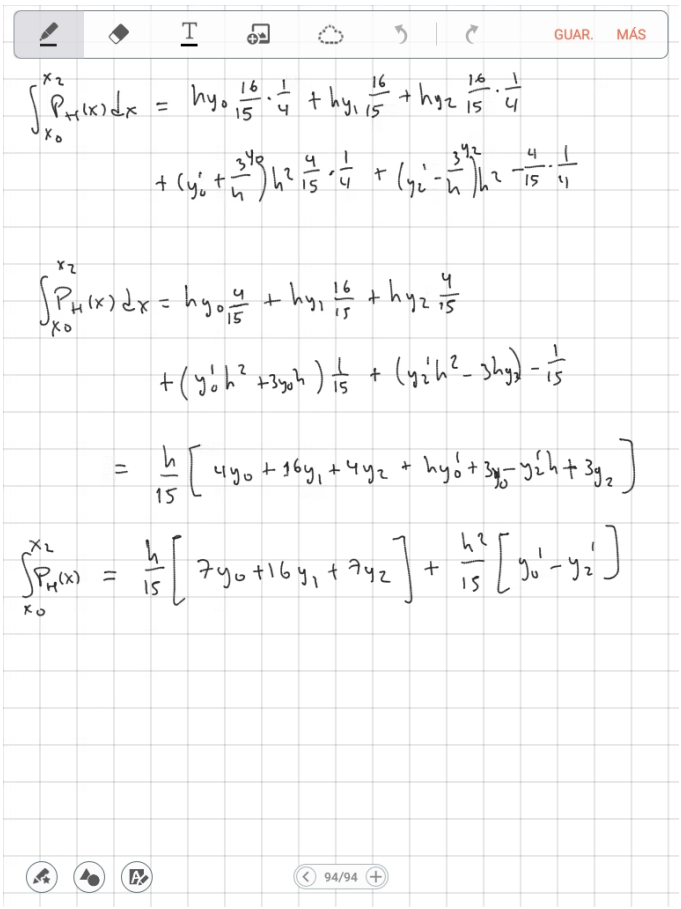
Como $dx = h\,ds$,

$$\int_{x_0}^{x_2} P_H(x) = y_0 h \int_0^2 L_0^2(s)\,ds + y_1 h \int_0^2 L_1^2(s)\,ds + y_2 h \int_0^2 L_2^2(s)\,ds$$

$$+ (y_0' + \tfrac{3}{h})h^2 \int_0^2 s L_0^2(s)\,ds + y_2' h^2 \int_0^2 (s-1) L_1^2(s)\,d$$

$$+ (y_1' - \tfrac{3y_2}{h}) \int_0^2 \ldots L_1^2(s)\,ds$$

Dibujar Recortar Compartir



Por lo que la integral del polinomio de lagrange queda en una expresión cerrada:

$$\int_{x_0}^{x_2} P_H(x) = \frac{h}{15}(7y_0 + 16y_1 + 7y_2) + \frac{h^2}{15}(y_0' - y_2') \tag{1}$$

Por lo que el algoritmo final queda de la siguiente forma:
Se usa la siguiente ecuación para generar los primeros i = 1, 2 puntos de y_i:

$$y_i = y_{i-1} + \int_{x_{i-1}}^{x_i} f(x_{i-1}, y_{i-1})dx \tag{2}$$

Que al tener un paso de ancho h entre x_{i-1} y x_i se reduce a:

$$y_i = y_{i-1} + hf(x_{i-1}, y_{i-1}) \tag{3}$$

Donde $f(x, y) = y + sin(x/2)$
Ahora ya con los 3 puntos, se podrán seguir calculando y_k para $k = 3, 4, ...$ mediante:

$$y_{i+1} = y_{i-2} + \int_{x_{i-2}}^{x_i} P_H(x)dx \tag{4}$$

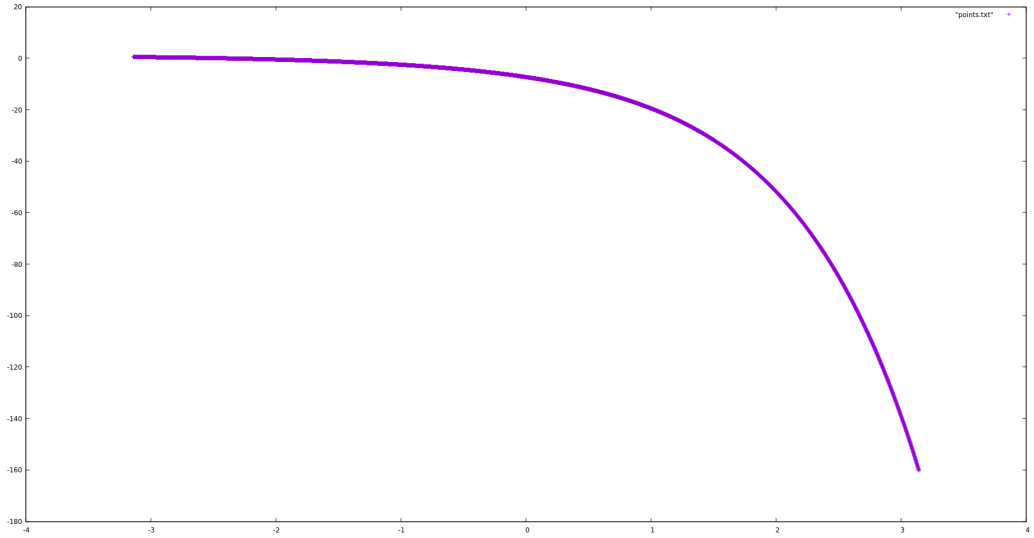
Aunque durante el desarrollo del algoritmo se observó que se obtuvo una mejor aproximación a la solución de la ecuación diferencial tomando un valor menos, esto es:

$$y_{i+1} = y_{i-1} + \int_{x_{i-2}}^{x_i} P_H(x)dx \tag{5}$$

Donde ya se conoce la expresión cerrada para $\int_{x_{i-2}}^{x_i} P_H(x)dx$

4.2. Ejemplo de prueba

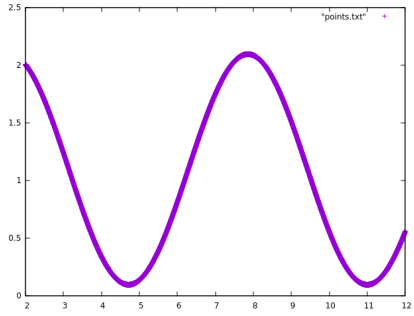
Estimación de la función solución a la ecuación diferencial $y' = y + sin(x/2)$



4.3. Resultados

En este método se puede observar que se resuelve la ecuación diferencial para las condiciones iniciales dadas.

Para comprobar el resultado se hizo una prueba con otra ecuación diferencial donde se trata de obtener $y = cos(x)$, mediante $f(x, y) = cos(x)$ obteniendo la siguiente gráfica:



Por lo que resulta un método eficiente de resolver ecuaciones diferenciales de primer orden, cuando se conocen las condiciones iniciales del sistema. Además de mostrar la importancia y utilidad de los métodos de interpolación e integración y como se pueden combinar para resolver otro tipo de problemas.

En cuanto al detalle de la índice i, se calculó el error promedio $e = \frac{\sum_{i=0}^{n} |y_i - f(x_i)|}{n}$ entre la función que se quiere estimar $f(x)$ y los valores estimados y_i, dando los siguientes resultados:

Para $y_{i+1} = y_{i-1} + \int_{x_{i-2}}^{x_i} P_H(x)dx$ se obtuvo un error promedio de $e = 0{,}144857$ mientras que con la ecuación de $y_{i+1} = y_{i-2} + \int_{x_{i-2}}^{x_i} P_H(x)dx$ se obtuvo un error de $e = 23{,}283323$.

Finalmente, algo que se podría explorar es el uso de diferentes polinomios de interpolación para como afecta en la estimación de la función o solución a la ecuación diferencial.