

Trabajo Práctico 1

“(No) Todo Pasa”

Metodos numericos
Primer Cuatrimestre de 2016

Integrante	LU	Correo electrónico
Leonardo Raed	579/04	leo_raed@yahoo.com
Ricardo Colombo	156/08	ricardogcolombo@gmail.com
Diego Santos	874/03	diego.h.santos@gmail.com
Luis Badell	246/13	luisbadell@gmail.com

¿Abstract?
¿Palabras clave?



Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
<http://www.fcen.uba.ar>

Índice

1. Introducción teórica	4
2. Desarrollo	5
2.1. Entrada y salida de los algoritmos	5
2.2. Sistema a resolver	6
2.2.1. Eliminación Gaussiana	7
2.2.2. Cholesky	8
2.3. Porcentaje de Victorias	9
3. Experimentación	10
3.1. Ranking	10
3.2. ¿Importa a quien se le gana?	11
3.3. Racha ganadora	12
3.4. Escalando Posiciones	17
3.4.1. Ganarle siempre al primero	17
3.4.2. Ganarle al que esta inmediatamente arriba	18
3.5. Análisis Cuantitativo	18
4. Discusión	23
4.1. Ranking	23
4.2. Importa a quien se le gana	23
4.3. Racha ganadora	23
4.4. Escalando Posiciones	23
4.5. Análisis Cuantitativo	24
4.6. La aritmética importa	24
4.7. Empates	24
5. Conclusiones	25
6. Apéndice	26
6.1. Archivos de test usados	26
7. Bibliografía	27
7.1. Bibliografía	27
8. Código	28
8.1. Sobre los archivos e implementacion	28

8.2. Código implementado	28
------------------------------------	----

1. Introducción teórica

Las Competencias deportivas de cualquier índole requieren la comparación de equipos mediante la confección de las tablas de Posiciones y rankings en base a los resultados obtenidos durante un cierto período de tiempo.

En este trabajo práctico intentaremos modelar y resolver el problema de generar un ranking de equipos a partir de los resultados con la condición de que no haya empates. Para confeccionar dicho ranking haremos uso de 2 métodos diferentes. El primero es el Winning Percentage y el ② es el Colley Matrix Method (CMM). Con estos métodos es posible obtener 2 rankings y nos proponemos hallar la similitud y diferencias entre ellos.

¿Por qué es importante el ranking ?

Es importante porque determina quienes fueron los mejores y peores equipos al final de la temporada, quienes avanzan a la siguiente ronda y en ciertas ligas como la liga de basquet y de baseball de los estados unidos determina la prioridad para los drafts. Además de eso tenemos la justicia deportiva y una gran cantidad de dinero en juego

¿Por qué es importante el análisis de datos ?

Es importante porque permite en las manos adecuadas, optimizar el rendimiento de los jugadores y además decidir si los jugadores son buenos en base a ciertos indicadores.

Muy pobre la redacción. No hay un acento. Está escrita a las apuradas. No se mencionan los métodos. No se describe correctamente el problema a resolver, ni que se busca con el trabajo.

2. Desarrollo

Esta sección **Entrada y salida de los algoritmos** da de datos no es mas importante que los métodos y los algoritmos.

Dados los requerimientos de la cátedra el programa toma como parametros 3 argumentos, el primero es el archivo de entrada, luego el archivo de salida y por último el modo. La cátedra solicitaba 3 modos el modo 0,1 y 2 para los 3 métodos solicitados para ejecutar sobre la matriz Colley, luego agregamos 3 modos más utilizados durante la etapa de experimentación.

1. Eliminación Gaussiana(EG)
2. Factorización de Cholesky(CL)
3. WP
4. Cholesky con modificación de partidos jugados
5. Cholesky haciendo ganar al último con el siguiente reiteradas veces hasta quedar primero
6. Cholesky haciendo ganar al último con el primero del ranking hasta quedar primero

No se explica previamente como se compone la liga, cuáles son las condiciones que cumple, cuáles son los parámetros y definiciones a considerar. El informe tiene que ser autodenido.

En todos los modos como paso previo a la realización de algunos de los métodos arma la matriz de Colley, explicada en la sección siguiente, para luego utilizando los métodos 0-1 en el programa (1 o 2 en la lista) resolver el sistema pedido.

El modo 3 corre cholesky, como paso siguiente busca 2 equipos que hayan jugado previamente para cambiar su resultado y luego volver a ejecutar cholesky, Este modo contiene un ciclo para repetir esta operación 100 veces. ¿Con qué objetivo? No se entiende.

Para el modo 4 corre cholesky y luego ejecuta un ciclo donde el objetivo es lograr que el que haya salido último luego de obtener el ranking de cholesky llegue al primer puesto ganandole al que tiene por arriba inmediato en el ranking. En cada paso agrega un partido más y vuelve a calcular cholesky para la nueva matriz, así obtenemos un nuevo ranking y continuamos iterando hasta quedar en la primera posición, siempre utilizando al que salio último en la primera utilización del metodo de cholesky.

Una vez finalizado por stdout devuelve la cantidad de partidos jugados, además de que en el archivo rankingSTEPS.4.out dentro de la carpeta test se encuentran los rankings en cada iteración y como es la evolución. El modo 5 es similar al anterior con la util diferencia que el participante que salio último la primera vez que corrio cholesky llegue al primer puesto jugandole al que se encuentra en el primer puesto en cada iteración.

Tanto el formato de entrada y de salida del programa son los solicitados por la cátedra. Para todos los métodos el archivo de entrada es el mismo, que contiene el siguiente formato:

$$\begin{pmatrix} (n) & (k) \\ (x1) & (e1) & (r1) & (t1) & (s1) \\ (x2) & (e2) & (r2) & (t2) & (s1) \\ \dots \\ (xk) & (ek) & (rk) & (tk) & (s1) \end{pmatrix}$$

Redacción. La primer linea tiene 2 valores n representa la cantidad de equipos y k representa la cantidad de partidos, seguido $\$$ líneas que representa cada partido, donde $x1$ representa una fecha que en nuestro caso no utilizamos, luego ei y ti representan los numeros de los equipos, y por último ri y si representan

Esto es justamente lo que pueden asumir que está explicado en el enunciado. No es relevante en el desarrollo explicar cual es el formato de los archivos.

"con el fin"

las anotaciones de cada equipo respectivamente.

en nuestra experimentación, con fines de no complejizar aún más el problema no utilizamos en campo que representa la fecha si no que asumimos que dado el orden que venian los resultados era el orden de los partidos.

Luego una vez se ejecutan los métodos 0-4 (1 a 4 en la lista) devuelven un archivo de \$ líneas donde en la línea se obtiene el ranking del equipo i.

¿Qué línea? ¿Hay una única línea?

Para los métodos 4 y 5 se devuelve el ranking para cada iteración antes de jugar el partido, estas se repiten hasta que el que comenzó último termine primero utilizando dos métodos descriptos anteriormente, en la última línea se obtiene la cantidad total de partidos, estos dos métodos además devuelven por stdout la cantidad de partidos jugados hasta que terminó en la primera posición el que comenzó último. Esta palabra no debe figurar en el cuerpo del informe.

No queda claro que hacen en este caso.

No parece ser lo que pide el enunciado.

2.2. Sistema a resolver

Para la implementación del problema presentado en la sección introducción nos basamos en el paper **The Colley Matrix Explained**. La cual consiste en plantear el siguiente sistema de ecuaciones en forma matricial

Redacción: así no se puede empezar una oración.

Aca debería ser "-"

$$C_{i,j} = \begin{cases} n_{i,j} & \text{si } i \neq j \\ 2 + n_i & \text{si } i = j \end{cases}$$

Redacción. y $b_i = 1 + (w_i - l_i) / 2$.

Así no se. Donde n_i es la cantidad de partidos totales jugados por el equipo i. $n_{i,j}$ representa la cantidad de empieza partidos jugados entre el equipo i y j.

Solo se necesita SDP.

Esta matriz tiene la particularidad de ser simétrica y definida positiva, además de ser estrictamente diagonal dominante. Estas dos propiedades nos permiten encontrar la factorización de Cholesky sobre esta matriz, y que con todas estas propiedades nos indican que la matriz también cumple que es no singular con lo cual al aplicar la eliminación gaussiana no vamos a necesitar realizar pivoteo ya que no nos encontraremos con 0 en la diagonal, dandonos así la factorización LU. Ambos métodos, Eliminación gaussiana y Factorización de Cholesky, combinadas con otras técnicas de remplazo nos permitirán resolver el sistema presentado previamente.

Esto está conceptualmente mal. Ver abajo.

La principal fortaleza de este método es que es útil para obtener rankings en torneos donde los participantes no juegan la misma cantidad de partidos, lo cual podrías hacer una análisis más exhaustivo sobre cada uno de los partidos jugados y como afecta esto en todo un torneo, además de que al armar el sistema en base a los resultados pasados, da relevancia al calendario de juegos de cada participante. Luego se intentará demostrar en la sección de experimentos, utilizando esta técnica importa contra quien se gana y contra quien se pierde.

Falta un "que"?

Por otro lado las desventajas que notamos es que no es aplicable a muchos deportes es que los empates no pueden ser modelados, además de que el ranking no toma en cuenta el margen de victoria de los equipos lo cual en los casos donde hay empate en el ranking podríamos utilizarlos para desempatar.

Redacción

No entiendo que significa "escalar" en este contexto.

En cuanto a escalar en este método no parece ser muy intuitivo mirando la matriz a simple vista pero nuestra intuición nos dice que importa a quien se le gana, que no es lo mismo ganarle al que está último que ganarle al que está primero y en esto nos vamos a basar para realizar nuestras experimentaciones en las secciones de más adelante. Para esto utilizaremos un algoritmo greedy tomando al equipo que

A = (0 1) es no singular, y EG encuentra un 0 en la diagonal. Además, que no hayan ceros en la diagonal (1 0)

Nuevamente, no queda claro como llevan adelante esta parte. Es importante que el análisis sea inverso, es decir, que el fixture esté fijo y decidir qué partidos conviene ganar para quedar en la mayor posición posible.

salga último en el ranking y haciéndolo jugar con otro equipo que este mejor posicionado que el, para esto utilizaremos 2 heurísticas distintas con el fin de obtener una mejor posición, la primera sera contra el inmediato siguiente en el ranking y la otra contra el que este primero, siempre tomando el ranking que resulta luego de cada partido.

Una vez obtenida la **Matriz de Colley** vamos a presentar dos técnicas de resolución del sistema de ecuaciones solicitado.

2.2.1. Eliminación Gaussiana

La implementación del algoritmo de **Eliminación Gaussiana** que elegimos es la que se encuentra en el libro **Burden**^[1]. Con el agregado de backwards substitution para obtener el vector de la ecuación $Cx=b$.

Presentamos un pseudo código del algoritmo de eliminación gaussiana que utilizamos en nuestra implementación que luego en secciones futuras vamos a detallar.

TP1 1 vector Gauss(matriz A, vector b)

```
1: Para $k$1...n - 1
2:   Para $j$1...n - 1
3:     Tomo el elemento  $a_{k,k}$  como pivot      "Tomamos". "Se toma". "Tomar". El TP no es unipersonal.
4:     Para  $j = i + 1, \dots, n$ 
5:        $a_{i,j} = a_{i,j} - a_{i,k} * (a_{i,k}/a_{k,k})$ 
6:        $b_i = b_i - a_{i,j} * (a_{i,k}/a_{k,k})$ 
7:      $x_n = a_{n,n+1}/a_{n,n}$ 
8:   para  $i = n - 1, 1$ 
9:     para  $j = i + 1..n$ 
10:       $sum+ = a_{i,j} * x_j$ 
11:
12: return x
```

A qué corresponde este código? Además, en caso de ser backward substitution parece ser incorrecto. En ningún momento calculan la solución del sistema (que, asumo, es "x" el parametro de salida).

Para este algoritmo como se puede observar es de complejidad $O(n^3)$ en el peor caso, ya que en ciclo interno de las posiciones 4 a 6 se ejecuta n veces y el ciclo de las líneas 2 a 6 se ejecuta n veces por lo tanto ya tendríamos n^2 iteraciones en el peor caso y finalizando con el ciclo de las líneas 1 a 6 que se ejecuta otras n veces. Luego en las líneas 7 a 10 se realiza el backward substitution que tiene en el peor caso se ejecuta n^2 veces. Redacción. No queda claro, y el análisis es muy escueto.
Con lo cual lo que esperamos en nuestros análisis de cantidad de equipos sobre tiempos de en la sección experimentación es encontrarnos con un gráfico de una función cúbica.

2.2.2. Cholesky

La implementación del algoritmo de factorización de cholesky que elegimos al igual que la eliminación gaussiana es la que se encuentra en el libro **Burden**. Agregandole los pasos que menciona en el libro (pagina 420) para resolver el sistema $\mathbf{Cr} = \mathbf{b}$

Este pseudocódigo representa nuestra implementación **sobre** la factorización de cholesky.

TP1 2 vector Cholesky(matriz A, vector b)

```
1:  $l_{1,1} = \sqrt{a_{1,1}}$ 
2: Para  $j = 2, \dots, n$ 
3:    $l_{j,1} = a_{j,1}/l_{1,1}$ 
4: Para  $i = 2, \dots, n-1$ 
5:    $l_{i,i} = (a_{i,i} - \sum_{k=1}^{i-1} l_{i,k}^2)^{1/2}$ 
6:   Para  $j = i+1, \dots, n$ 
7:      $l_{j,i} = (a_{j,i} - \sum_{k=1}^{i-1} l_{j,k} l_{i,k} / l_{i,i})$ 
8:    $l_{n,n} = (a_{n,n} - \sum_{k=1}^{n-1} l_{n,k}^2)$ 
9:    $y_1 = b_1 l_{1,1}$ 
10:  Para  $i=2..n$ 
11:    Para  $j=1..i-1$ 
12:      sum= $l_{i,j} * y_j$ 
13:       $y_i = (b_i - sum) l_{i,i}$ 
14:    $x_n = y_n l_{n,n}$ 
15:  Para  $i=n-1..1$ 
16:    Para  $j=i+1..n$ 
17:      sum= $l_{i,j} * x_j$ 
18:       $x_i = (y_i - sum) l_{i,i}$ 
19:
20: return x
```

Presentarlo como un pseudocódigo distinto.

Presentarlo como un pseudocódigo distinto.

Para las líneas 2-3 se puede ver que se ejecuta n^2 veces. Dentro de este algoritmo en las líneas 4-7 se realizan 3 ciclos donde en el peor caso se obtienen n^3 ciclos. Por último para las líneas 10 - 13 y 15 - 18 se realizan 2 ciclos de n iteraciones cada uno teniendo como total n^2 iteraciones. Con lo cual lo que esperamos en nuestros análisis de tiempos por cantidad de equipos es que obtengamos una cuadrica.

Muy confuso el análisis de complejidad. No es obligatorio hacerlo, pero si lo hacen, tiene que ser mas cuidado.

2.3. Porcentaje de Victorias

Por otro lado analizaremos otra técnica en base a el **Porcentaje de Victorias** que a lo largo del análisis denominaremos **WP** que consiste en tomar el promedio de partidos ganados sobre partidos jugados.

Esta técnica básicamente analiza la performance de un equipo participante en los partidos jugados en base a partidos ganados.

En este caso el score de un equipo no es afectado por la cantidad de partidos y resultados obtenidos de los demás participantes, pero esto sí afecta su posición final en el ranking.

Es la primera vez que mencionan esto, y no

Esta técnica a priori no aporta mucha información respecto a la posibilidad de victoria en el siguiente encuentro y tampoco considera el ranking del rival enfrentado. Ya que todos los partidos valen lo mismo.

relación con el otro método. Además, esa idea es una motivación para derivar el método de Colley, no la interpretación final del resultado.

La implementación consiste en calcular: $\sum_{i=1}^n \frac{G_i}{T}$

Donde n es la cantidad de partidos jugados, G corresponde a partidos ganados y T al total de partidos jugados.

En la cuenta usan G_i , que no está definido. No está bien la explicación.

3. Experimentación

Para analizar la efectividad y ecuanimidad de esta nueva forma de calcular el ranking vamos a realizar una serie de **test** a fin de obtener un análisis cuantitativo y cualitativo que nos permita compararlo con el clásico método de **WP**.

Con los **test** esperamos encontrar ventajas y desventajas de esta forma de medición, particularmente en escenarios donde no todos los participantes jueguen la misma cantidad de partidos.

Además realizaremos una comparación de los métodos de **Eliminación Gaussiana** y **Cholesky** para ver cual de los dos computa los rankings de manera mas eficiente.

En esta sección solo presentaremos los experimentos realizados y los resultados obtenidos. Las conclusiones de cada experimento las presentaremos en la **siguiente** sección.

3.1. Ranking

Vamos a comparar la tabla de ranking obtenida a partir de un set de datos de la **ATP 2007**. Es decir calculamos el ranking a partir de la técnica **WP**, considerando **partidos ganado** **partidos jugados**, a pesar de que no todos los jugadores hayan participado de la misma cantidad de partidos. Comparandolo con el **CMM** implementado con **Cholesky**. **Redacción**.

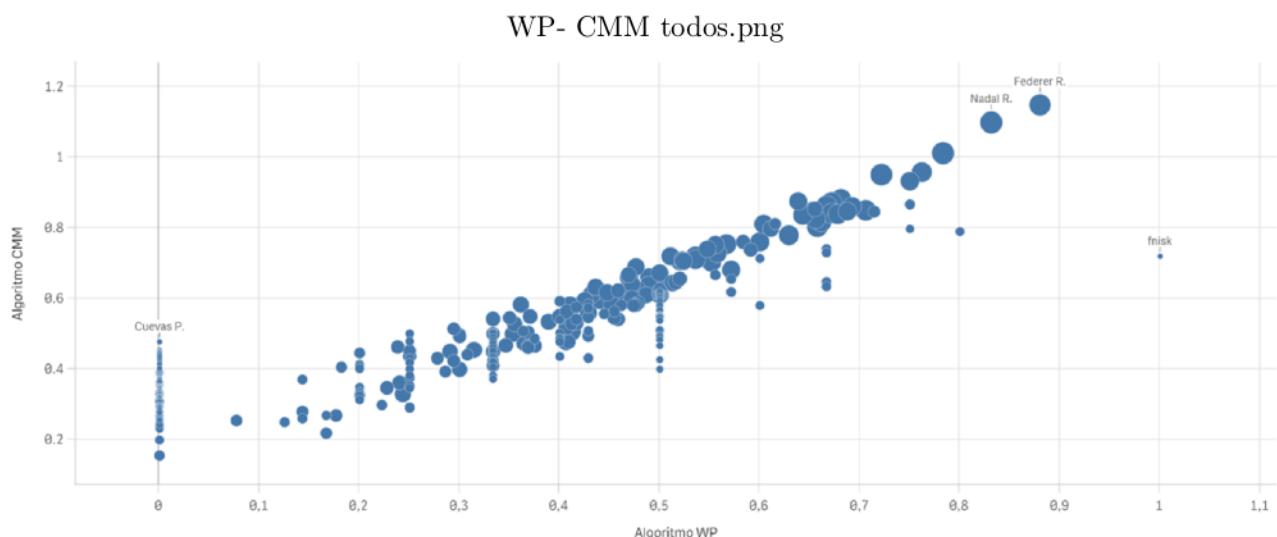


Figura 1: Rankings para comparar WP vs CMM

El **grafico** elegido para graficar es un **grafico de dispersión**, que muestra:

- Eje X el valor obtenido al ejecutar el algoritmo WP.
- Eje Y el valor obtenido al ejecutar CMM implementado con Cholesky.
- El tamaño de la burbuja es la cantidad de partidos jugados.

Ojo con Lo que se observa en el **grafico** es que el ranking CMM parece darle relevancia a la cantidad de **esta** partidos jugados ya que las burbujas de los top 10 son mas grandes. **Esto esta** dado en principio por afirmación, es fuerte. También pasa que los que más juegan son los que mas ganan. Podrían plantear un experimento adicional para corroborar/refutar esta hipótesis.

la característica del deporte (tenis) que permite a los que ganan jugar más partidos en los torneos. Lo que nos llama la atención es que si observamos el rank arrojado por WP, se observa que el primero es fnisk. El mismo es un jugador que solo jugó 1 partido y lo ganó, por lo cual tiene un 100% de efectividad y figura como primero.

Además, si observamos los top 10 del rank WP, observamos algunas burbujas de tamaño pequeño. Esto es porque en algunos casos, este ranking beneficia jugar pocos partidos pero ganarlos.

Bien la observación.

Hemos realizado un zoom dentro del gráfico para corroborar el nombre y posición de ambos rankings.

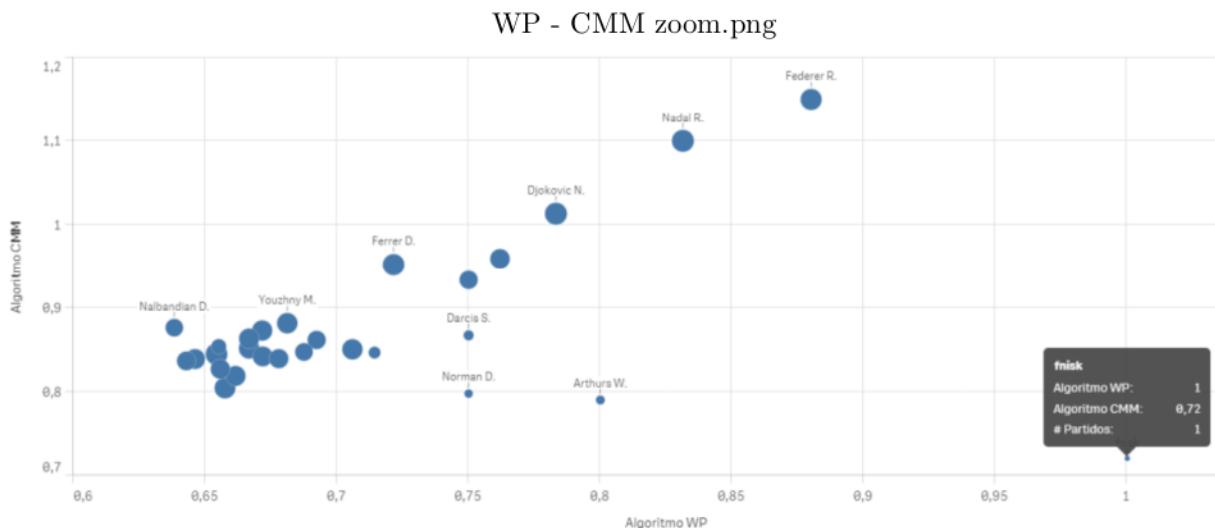


Figura 2: Zoom de las primeras posiciones

En el gráfico se resalta el jugador fnisk y se muestra información adicional.

3.2. ¿Importa a quién se le gana?

En el escenario que se utilice WP realmente no importa a qué equipo se le gane, ya que todos los partidos tienen la misma importancia y se les asigna el mismo puntaje. Pero en el caso de CMM resulta más interesante plantearse esta pregunta.

La hipótesis que tenemos es que tomando un equipo de mitad de tabla, que denominamos **medio** el hecho de que le gane al **líder** de la tabla va a mejorar mucho más el ranking que derrotando al que ocupe la última posición.

Realizamos un test tomando al equipo **medio**, y agregando un partido victorioso contra el puntero y analizamos como se modifica su ranking. Luego tomamos la tabla inicial, es decir sin ganarle al puntero, y repetimos el experimento esta vez derrotando al último. **Ok, bien planteado.**

Presentamos los resultados obtenidos.

a quien le gano.png

Posicion CMM	Jugador	Ranking
1	Federer	1,131919
164	Vasallo Arguello	0,474265
334	Vicente F.	0,198425

Ganandole al primero

Posicion CMM	Jugador	Ranking
1	Federer	1,117437
141	Vasallo Arguello	0,508447
334	Vicente F.	0,198271

Ganandole al ultimo

Posicion CMM	Jugador	Ranking
1	Federer	1,132469
162	Vasallo Arguello	0,480302
334	Vicente F.	0,170821

Figura 3: Ranking de jugadores

Se observa una mejoría en el ranking luego de haber ganado contra el primero.

3.3. Racha ganadora

Para estudiar la ecuanimidad del **CMM** realizamos un experimento tomando al participante del **ATP 2007** que se encontraba en el último puesto y le asignamos una racha ganadora contra los

No es una buena forma de mostrarlo, es muy desprolija. Armar una tabla directamente en LaTex con la información necesaria.

primeros diez jugadores del ranking.

Además este test nos permite observar como la racha de un jugador afecta al ranking global y si ganandole a los mejores realmente escala una considerable cantidad de posiciones en el ranking.

A continuación presentamos el ranking calculado construido de la siguiente manera:

- Eje X el valor obtenido al ejecutar CMM **implementado con Cholesky.**
- Eje Y el valor obtenido al ejecutar CMM **implementado con Eliminación Gaussiana.**
- El tamaño de la burbuja es la cantidad de partidos jugados.

En qué impacta cambiar el método de resolución?

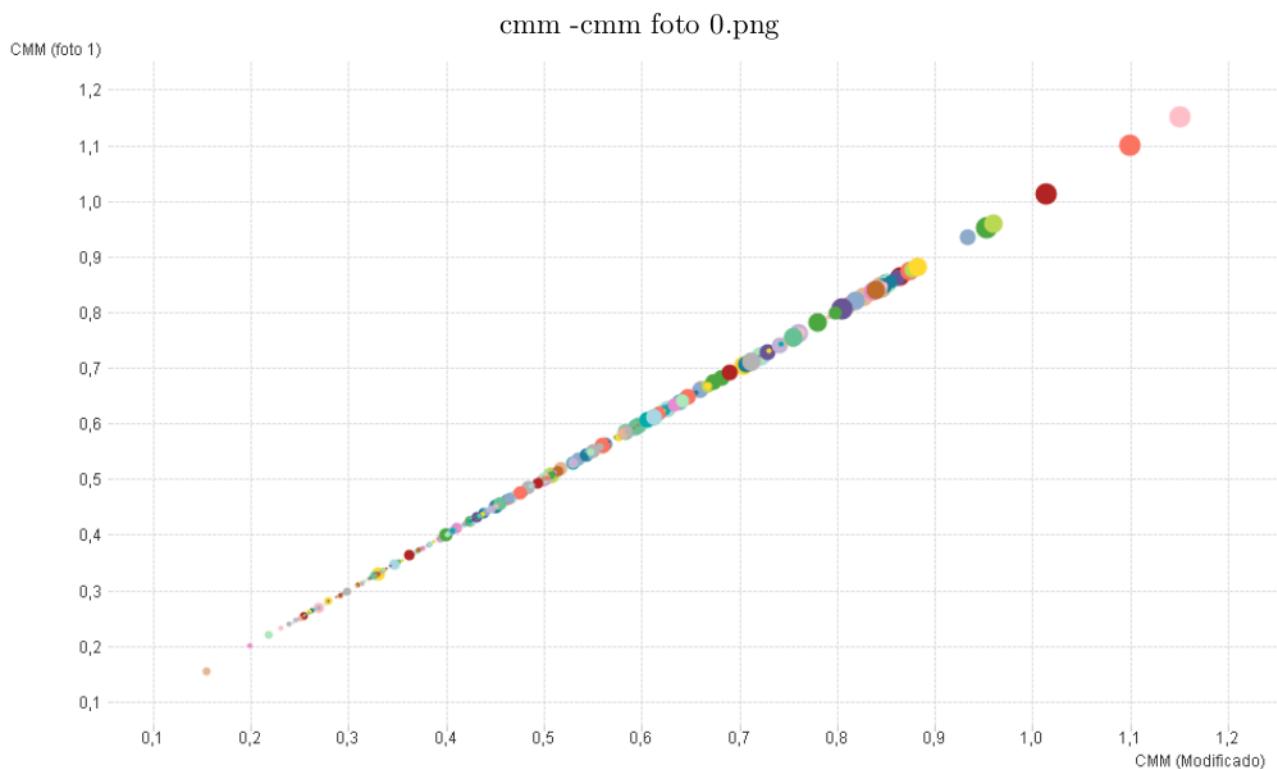


Figura 4: Ranking de jugadores

Es lógico que se observe una diagonal ya que ambos ejes son la misma métrica de CMM. Luego el experimento realizado fue agregar partidos al schedule, de manera que el último le gane a los 10 primeros y entender que tipo de reacción tiene el algoritmo CMM. Para ello se realizaron 10 schedules distintos de tal forma que en cada uno de esos archivos de entrada exista un partido mas que en el caso anterior y el mismo sea una victoria del último jugador del ranking vs alguno de los top 10.

Para poder mostrar una evolución a medida que se van jugando los 10 partidos, hemos dejado el eje Y del gráfico con el ranking inicial, mientras que el eje X pasa a ser el ranking recalculado con la agregación de partidos.

En el gráfico se observa mediante una flecha el salto del ranking de un partido a otro y evidencia cual

es la ganancia que se tiene al ganarle a los top 10.

Está muy bien la idea del gráfico.

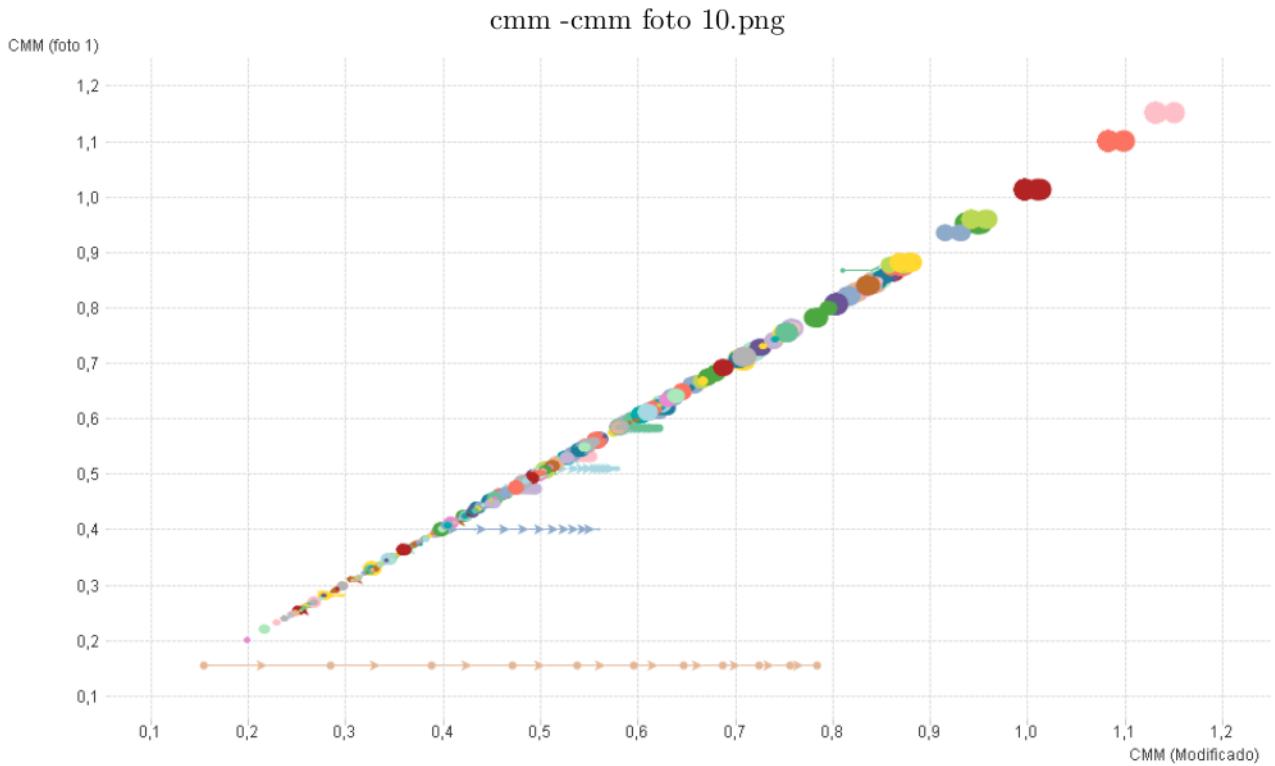


Figura 5: Evolucion de jugadores con el pasar de los partidos

Por último y para entender si el jugador en la última posición podía llegar a ser primero en algún momento, hemos agregado al archivo un total de 100 partidos ganados por el último contra los top 10.

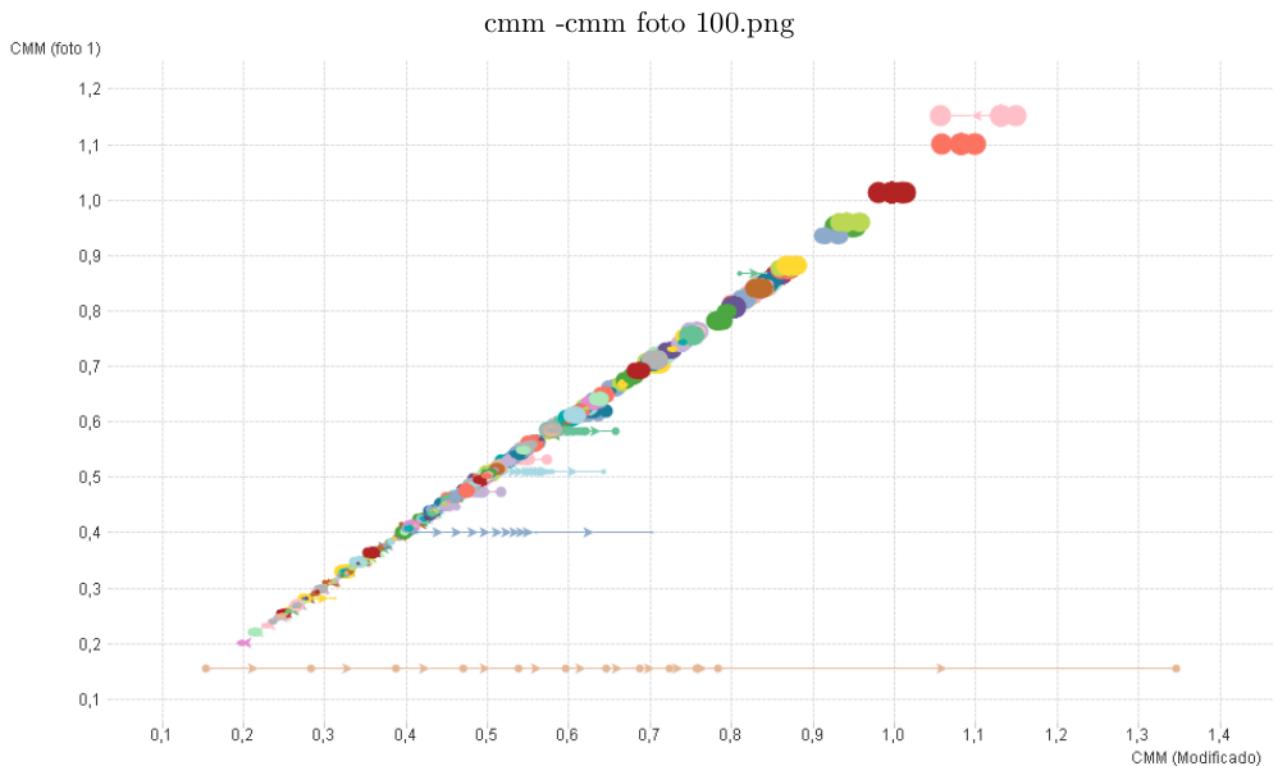


Figura 6: Evolucion de jugadores con el pasar de los partidos

Remarco: Y tal como vemos, si es posible pero a un costo altísimo (jugar alrededor de 100 partidos).
esto no es lo que pedía el enunciado. Lo interesante es que aquellos equipos contra los que el último jugador jugó también sufrieron modificaciones en su ranking.
A continuación se observa un zoom dentro del gráfico para observar este comportamiento.

El último jugador del ranking se llama Verkerk M., sus partidos fueron los siguientes:

jugados vs verkerk.png

NroPartido	Winner	Wsets	Loser	Lsets
535	de Voest R.	2	Verkerk M.	0
757	Grosjean S.	2	Verkerk M.	0
832	Hanescu V.	2	Verkerk M.	0
902	Chela J.I.	2	Verkerk M.	0
1063	Brands D.	2	Verkerk M.	0
1092	Acasuso J.	2	Verkerk M.	0
1145	Almagro N.	2	Verkerk M.	0
1212	Lapentti N.	2	Verkerk M.	0
1234	Bolelli S.	3	Verkerk M.	0

Figura 7: Partidos jugados por Verkerk

En el siguiente gráfico se observa el movimiento de ranking de los que le ganaron partidos a Verkerk, teniendo en cuenta la escalada al primer puesto de este jugador.

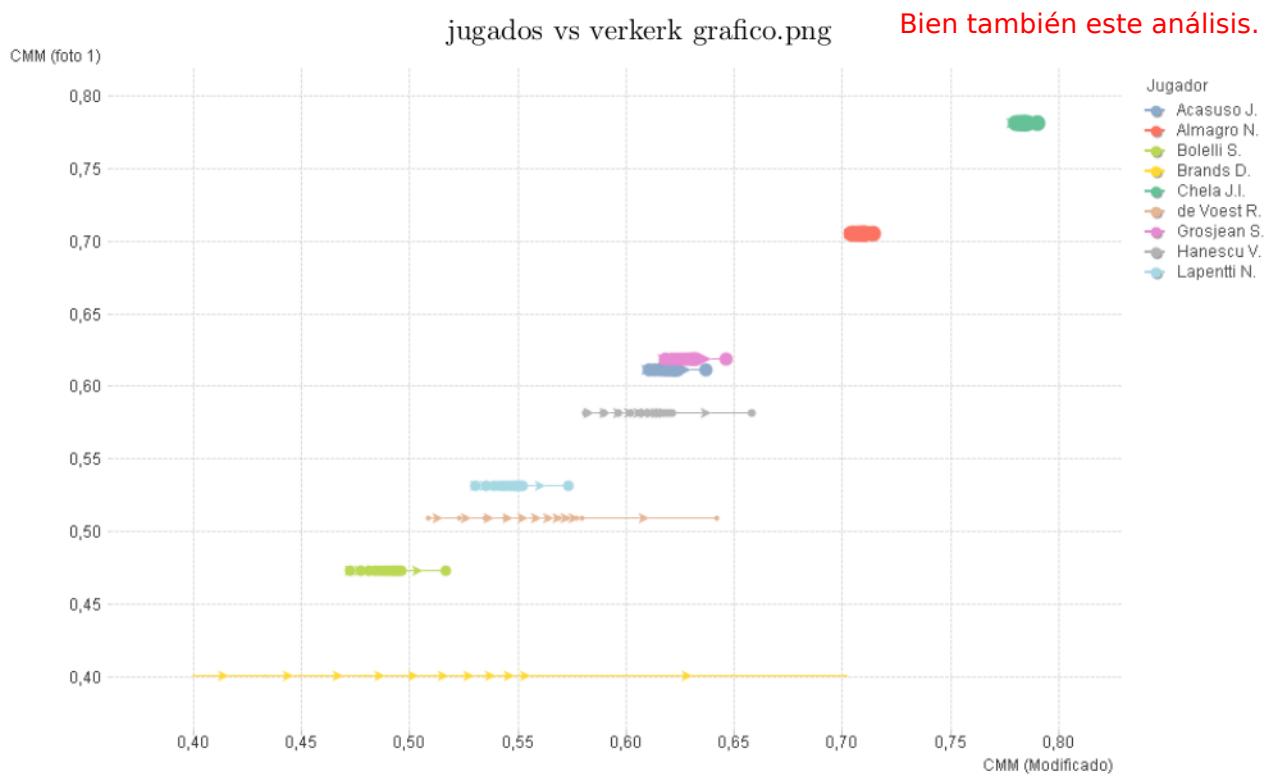


Figura 8: Evolución de aquellos que le ganaron a Verkerk

3.4. Escalando Posiciones

Una de las consignas del trabajo era encontrar una técnica para hacer escalar en el ranking a un equipo, pensamos en 2 tácticas.

De nuevo: Ambas se basan en, dado un schedule ya definido donde existe un jugador que está último en el ranking, hacerlo jugar y ganar vs 'el que siempre está primero en la fecha correspondiente' y 'contra el que tiene inmediatamente arriba de él en el rank'

Pensamos en una primera instancia era que tenía que ascender más pronto aquel jugador que juega y gana contra el que ocupa el primer puesto que aquel que juega y gana contra cualquier otro jugador. Los algoritmos realizados cortan la ejecución al llegar al primer puesto, arrojando la cantidad de partidos en total que debió jugar, y ganar, ese jugador.

A continuación se muestran los graficos.

3.4.1. Ganarle siempre al primero

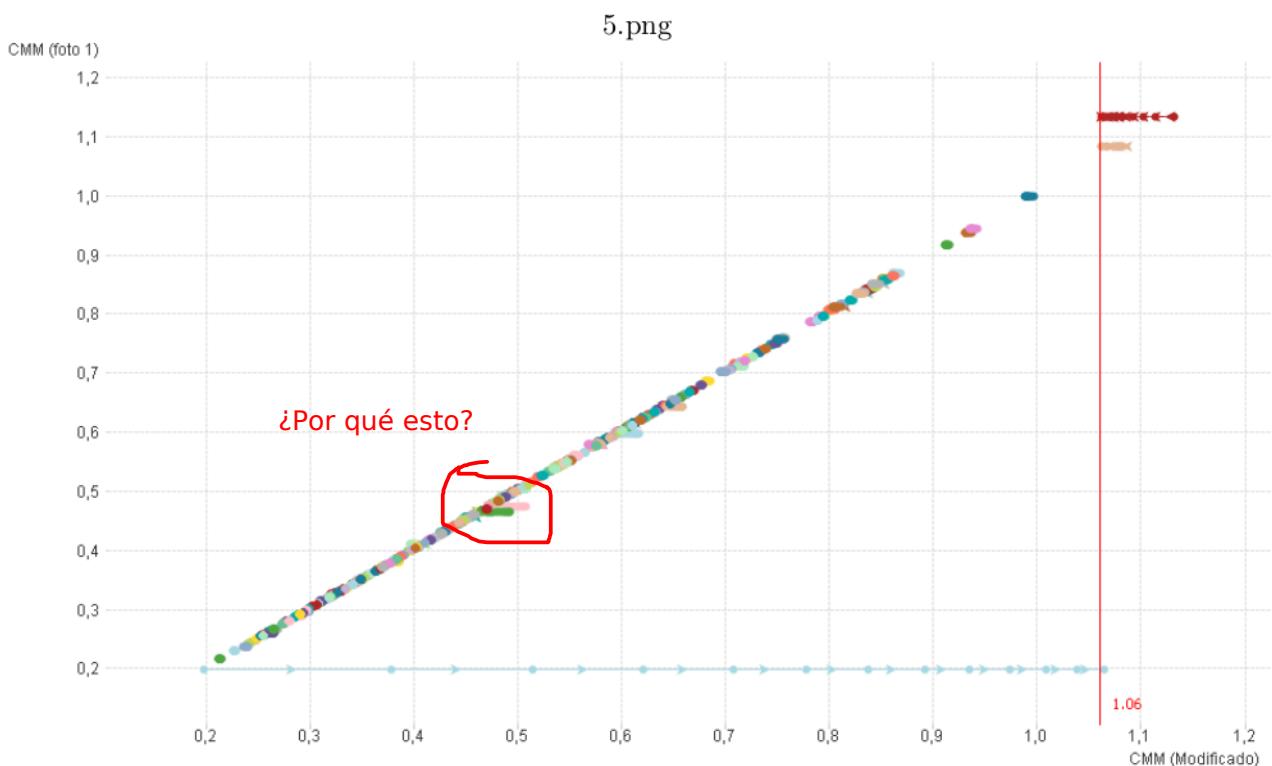


Figura 9: Ganarle siempre al primero

En este gráfico se observa lo siguiente.

Cada burbuja es un jugador, el eje de las X es el rank que arroja CMM para las distintas fechas jugadas, mientras que el eje Y tiene el rank de CMM base para realizar el análisis y comparar la evolución de los jugadores.

En este gráfico se observa que el crecimiento es rápido en un inicio y luego va reduciendo su velocidad de crecimiento. Se observa que el algoritmo necesita de 12 fechas para llegar a la cima del campeonato, validando nuestra teoría.

Se observa también una línea de referencia que indica cuando llega el jugador a superar al primero del ranking.

3.4.2. Ganarle al que esta inmediatamente arriba

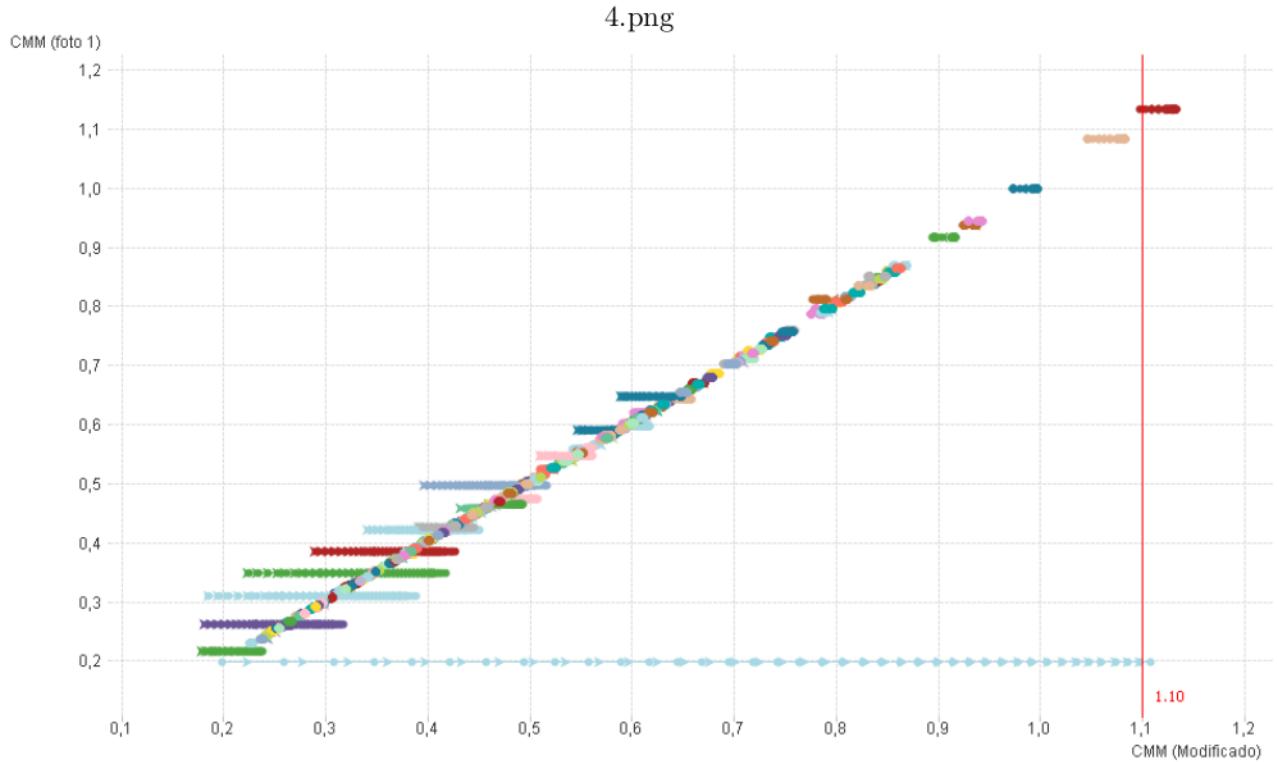


Figura 10: Ganarle siempre al que esta inmediatamente arriba

En este grafico se observa que el crecimiento es lento y que se necesitan aproximadamente 40 fechas para llegar a la cima del campeonato.

Se observa tambien una linea de referencia que indica cuando llega el jugador a superar al primero del ranking.

3.5. Análisis Cuantitativo

Vamos a estudiar la eficiencia de ambas técnicas incrementando y variando los volúmenes de datos. La idea es repetir el cómputo de los rankings para la misma instancia de datos al azar, y posteriormente ir incrementando la cantidad de datos.

Nuestra hipótesis es el que método de basado en **WP** va a tardar lo mismo para instancias de datos iguales, y se irá incrementando de forma casi lineal a medida que incrementemos los datos. En cambio con **CMM** basando en **Eliminación Gaussiana** y **Cholesky** esperamos que difieran en para las mismas intancias. Nuestra hipótesis sobre esto es que la implementación de **Cholesky** va a demorar menos tiempo. No sería solo la implementación, sino también el método.

Para esta prueba se generaron schedules variando la cantidad de equipos en 6, 50, 100, 200, 300, 500, 700, 1000 y 2000.

Adicionalmente se varió la cantidad de partidos jugados por cada equipo. Dado el análisis de complejidad de los algoritmos implementados, solo varían el tiempo de **calculo** en función del tamaño de la matriz definida por la cantidad de equipos, por lo cual al variar la cantidad de partidos no esperamos encontrarnos con variaciones en el tiempo.

Ok, bien la hipótesis.

Este no es lo que más nos importa.
Por qué encerrarlo entre dos cuadráticas?

Ejecutamos los test para nuestra implementación de Eliminación Gaussiana. A continuación se muestran los resultados de tiempos de ejecución dependiendo la cantidad de equipos. Para evidenciar la complejidad **cubica** del algoritmo lo hemos encerrado entre 2 funciones **cuadráticas** que evidencian que no pueden contener la curva de tiempos de nuestro algoritmo.

cuadrativo.png

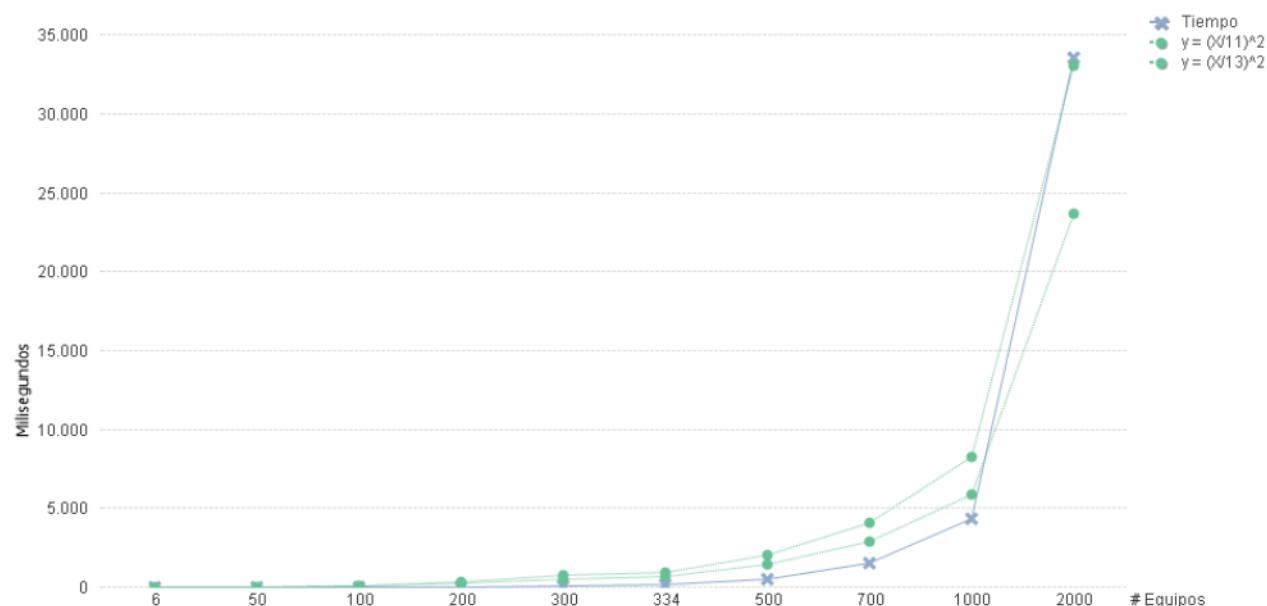


Figura 11: Gauss cuadrático

El título del gráfico confunde.

Luego observamos la misma gráfica pero con **líneas** de referencia de 2 funciones **cubicas**.

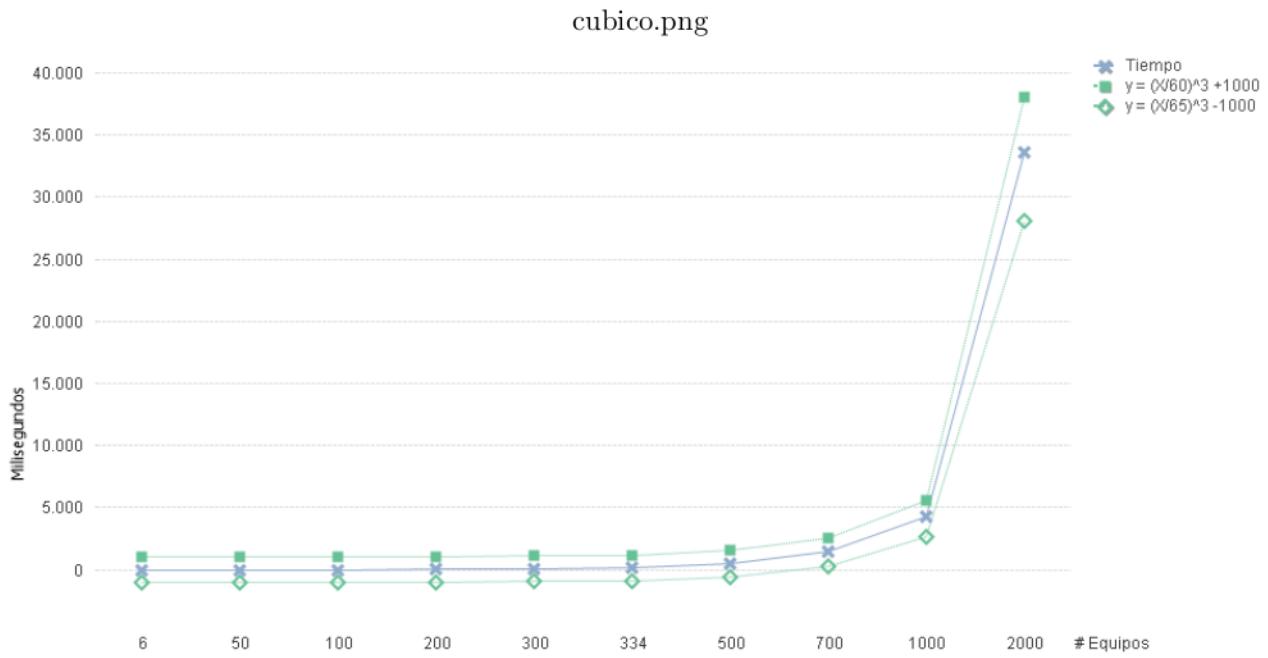


Figura 12: Gauss cubico

Ejecutamos los test para nuestra implementación de Cholesky. A continuación se muestran los resultados de tiempos de ejecución dependiendo la cantidad de equipos. Para mostrar la complejidad cúbica del algoritmo lo hemos encerrado entre 2 funciones cuadráticas que evidencian que no pueden contener la curva de tiempos de nuestro algoritmo.

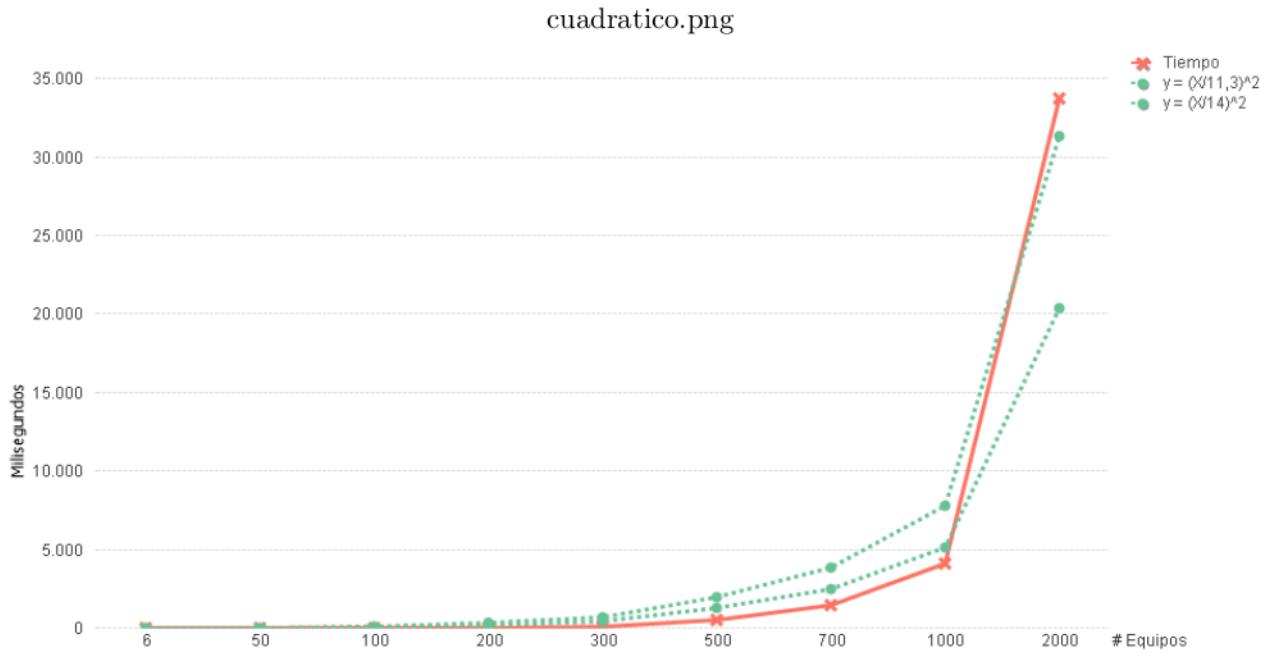


Figura 13: Cholesky cuadratico

Luego observamos la misma gráfica pero con líneas de referencia de 2 funciones cúbicas.

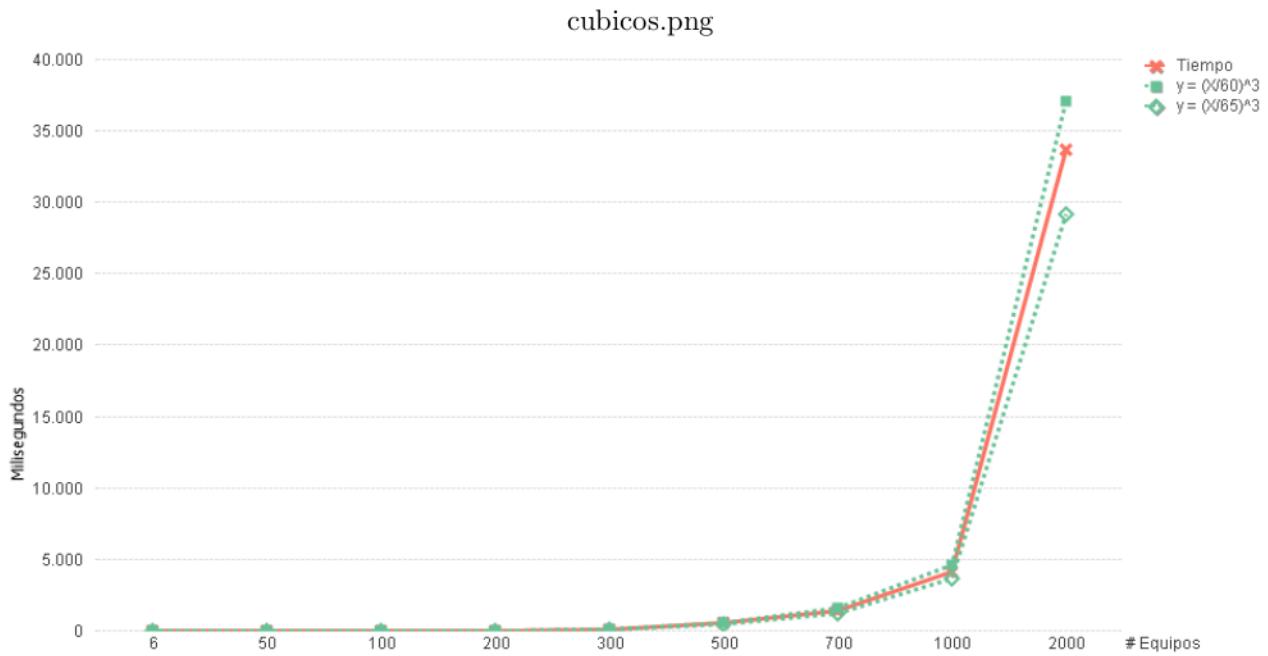


Figura 14: Cholesky cúbico

Ejecutamos los test para nuestra implementación de WP. A continuación se muestran los resultados de tiempos de ejecución dependiendo la cantidad de equipos:

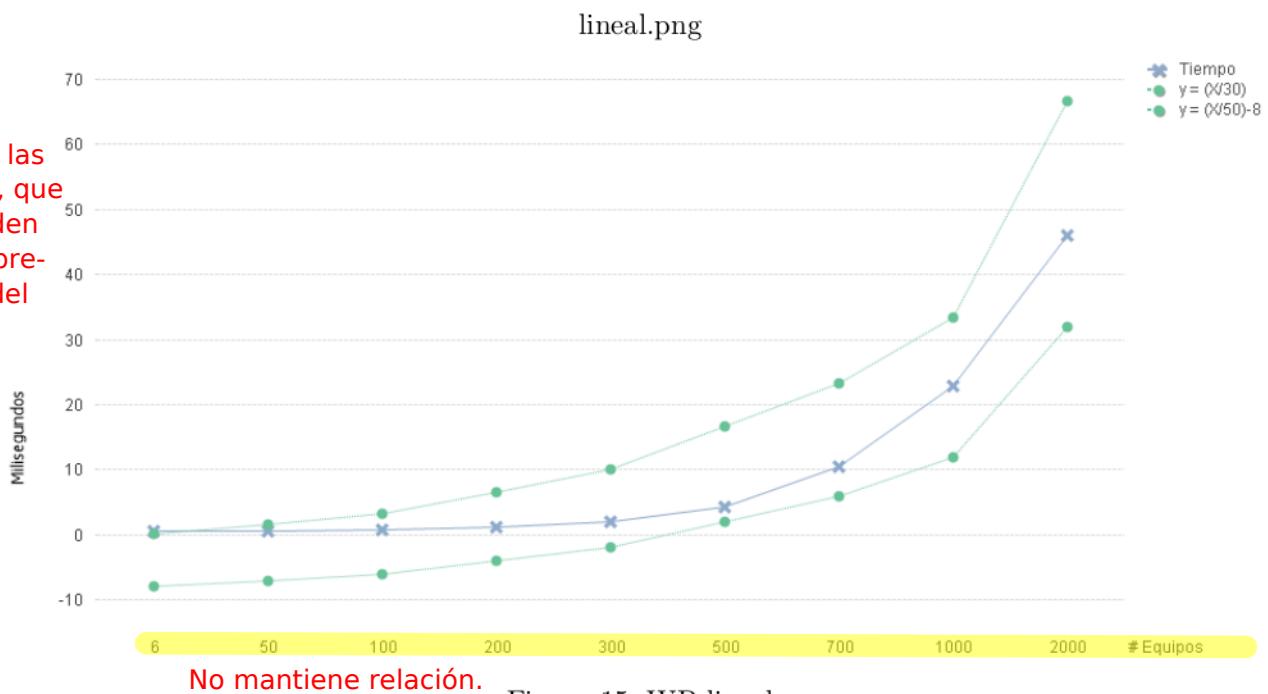


Figura 15: WP lineal

Cuando leímos el enunciado encontramos una frase que nos llamó la atención y era la siguiente:

Se pide comparar, para distintos tamaños de matrices, el tiempo de cálculo requerido para cada método en el contexto donde la información de la matriz del sistema (C) se mantiene invariante, pero varía el término independiente (b)"

No comparan EG contra Cholesky. Si bien no se pide explícitamente, ustedes lo mencionan en la sección "Discusión". Gauss y Cholesky parecen dar exactamente los mismos tiempos, lo cual en principio parece raro.

Entendemos que nuestros tiempos de cálculo no debían variar con la modificación del término independiente, pero para resolver esta incógnita decidimos realizar la prueba.

Lo que hicimos fue tomar una matriz C, calcularle CMM, luego modificar algunos partidos de la matriz y cambiarlos (esto significa cambiar el resultado de ganadosperdidos) y calculamos nuevamente CMM.

No es esto lo que pide el TP. El objetivo es que identifiquen en base al contexto la ventaja de tener una factorización, en este caso la de Cholesky (mencionada varias veces en clase). Hay que comparar el tiempo de cómputo entre EG y Cholesky si la matriz no se modifica, pero tenemos distintos términos independientes. Como usamos la factorización?

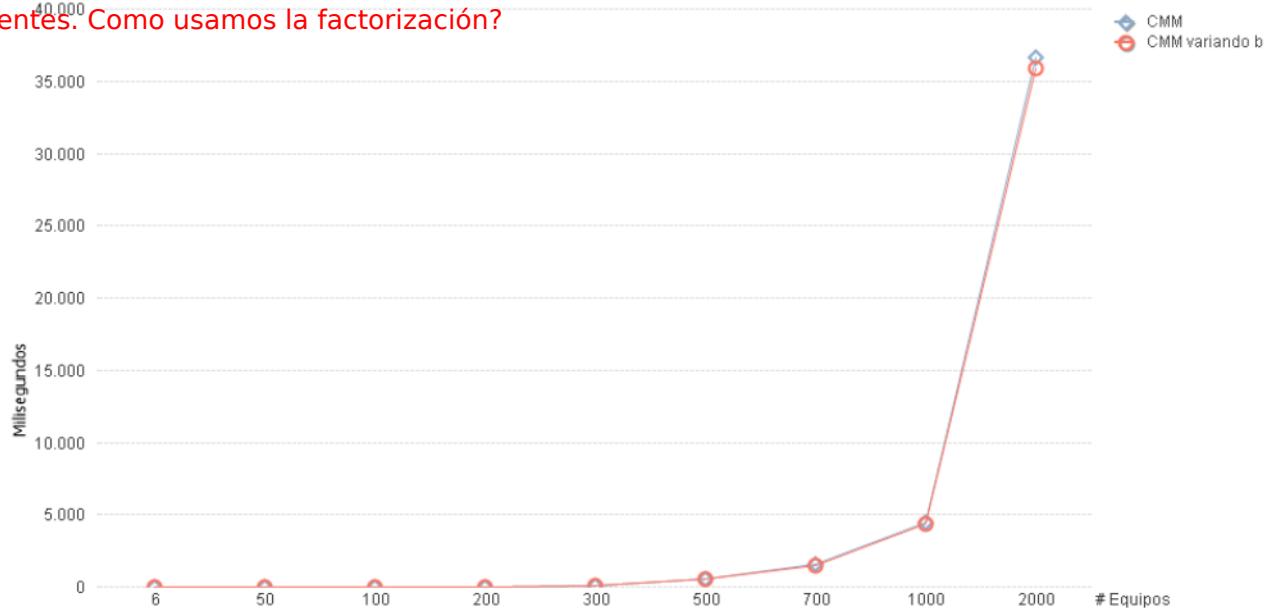


Figura 16: Cholesky con otro término independiente

4. Discusión

No, en esta sección presentan la discusión.

En esta sección presentamos nuestras conclusiones sobre los resultados obtenidos de los experimentos del punto anterior.

4.1. Ranking

De los resultados obtenidos podemos ver que el ranking obtenido con WP no es muy realista, ya que la primera posición es ocupada por un participante que jugó y ganó un solo partido.

El ranking obtenido por CMM refleja de forma mucho más realista el desempeño de cada jugador en el torneo.

En un escenario donde tenemos participantes que jugaron una cantidad distinta de partidos pensamos que refleja mejor la realidad del torneo el método de CMM. Ok, esto no sería una discusión.

4.2. Importa a quien se le gana

Como podemos observar realmente importa contra quien se gane, del experimento realizado observamos que ganarle a un participante que está primero afecta más el puntaje del ranking que ganando contra el último. Y por qué esto? Alguna intuición? Relación con el sistema de ecuaciones?

La hipótesis con la que calculamos el experimento resultó ser verdadera.

4.3. Racha ganadora

Por lo visto el jugador escalo rápidamente desde el último puesto en la tabla de posiciones a casi la mitad de tabla. Si bien mejoró su posición, no alcanzó el top ten, y en las últimas victorias su ascenso fue más lento. Esto nos hace concluir que solo haciendo jugar y ganar a un participante, la capacidad que tiene para crecer en el ranking está limitada por la falta de juego de sus rivales. Ok, que significa esto?

Además sus victorias representaron una mejoría en el ranking de los participantes que lo vencieron a él. De esta forma podemos comprobar que la racha de un jugador afecta al ranking global. Ok, y esto por qué?

Por qué limitada, si están agregando partidos arbitrariamente?

4.4. Escalando Posiciones

Redacción.

Redacción. Como mencionamos al principio y confirmamos con estos experimentos es que si importa a quien hay que ganarle ya que como se demostró no es lo mismo ganarle al inmediato en el ranking que al que se encuentra primero, de esta manera la forma más rápida de llegar al primer puesto es ganandole siempre al que se encuentre primero. Solo probaron un caso. Es poco como para afirmarlo. Además, esto ya lo dicen arriba.

4.5. Análisis Cuantitativo

Como era de esperar en el caso de **WP** para instancias el tiempo de ejecución fue el mismo, y el tiempo demorado a medida que crecían los datos de la instancia fue líneal.

crecen.

es

En el caso de **CMM**, no podemos decir que fue más eficiente la implementación de **Cholesky** que **Eliminación Gaussiana**, porque los tiempos observados son parecidos entre sí. Pero en este caso esperábamos una mejoría dado que los algoritmos se implementaron basadas en el libro **Burden**, que afirma que **Cholesky** consume $1/3 n^3$ flops y **Eliminación Gaussiana** $23 n^3$ flops.

[Revisar experimentos.](#)

4.6. La aritmética importa

De los experimentos realizados notamos que es importante el tipo de datos utilizados. Principalmente cuando se utiliza **CMM**.

Los errores de redondeo pueden derivar en un mal cálculo del ranking. Es decir, no considerar los suficientes decimales puede derivar en que un participante con un ranking decimalmente menor quede mejor rankeado que otro con mayor puntaje.

Por ejemplo: El participante A con ranking 0,5819 y el participante B con ranking 0,5816 si se consideran solo dos decimales ambos tienen 0,58 y esto podría afectar su orden en el ranking global.

Para evitar esta situación nuestra implementación usa el tipo de datos float con 5 decimales después de la coma.

No, la implementación trabaja con double.

4.7. Empates

Encontramos que los empates pueden modelarse en el caso de **WP**, asignando un puntaje al partido empatado y continuando con el procedimiento normal.

En el caso de **CMM** nos resultó muy difícil tratar de modelarlo, como alternativa pensamos que una buena idea sería, en el caso de existir un empate entre el jugador A y el jugador B, podríamos dar un partido ganado y otro perdido a cada uno de los jugadores. Esto nos permite reutilizar el método y de alguna forma penalizar y premiar a ambos equipos por haber empatado su partido.

Como altera ésto la relación entre n_i y w_i y l_i ?

5. Conclusiones

No responden la pregunta respecto de la "justicia", es decir, si se puede influencia negativamente a otros dos equipos. Lo que pide el TP no se refiere a esto.

Luego de la experimentación y análisis de los resultados, concluimos el método de **calculo** basado en **CMM** es más justo en el caso de torneos donde los equipos no juegan la misma cantidad de partidos y donde el empate no es una opción. Ya que asigna un puntaje en base no solo a los resultados obtenidos, sino contra quien fueron obtenidos. Obteniendo un ranking basado en la meritocracia del resultado. **Redacción.**

Redacción. Para el caso de torneos donde cada equipo juegue la misma cantidad de partidos el método de **WP** a nuestro criterio resulta más justo. Debido a que todos se pusieron a prueba la misma cantidad de veces. De lo contrario se vuelve mas justo el CMM porque no seria justo que alguien que juega menos partidos este mas alto en el ranking que alguien que no y eso en WP se nota cuando comienza a haber diferencias de partidos.t

Redacción. Respecto a que implementación resultan igual de eficiente ambas, se obtienen el mismo resultado **Este exp.** con lo cual no notamos diferencias para este caso en cuanto preferencias por tiempo o implementacion. **no está planteado.**

Por último sobre la frase **La utilización de técnicas avanzadas de análisis de datos son imprescindibles para mejorar cualquier deporte**, y en base a las peliculas vistas , consideramos que el Analisis de datos es una herramienta importante para mejorar el rendimiento deportivo de un equipo, siempre y cuando se utilízen correctamente. Las estadísticas son indicadores de que tan bien rinden los jugadores pero hay factores externos como el **animo**, el nivel de los competidores que no son considerados en las estadísticas ya que no son medibles y pueden modificar bastante el rendimiento de los jugadores. Afortunadamente la frialdad de los números no es aplicable a la pasión de todos los deportes.

Mientras en contados deportes el resultado puede predecirse de antemano, debido a las características de los rivales, como en el caso del Polo, esta analogía no puede aplicarse a deportes como el Fútbol donde en innumerables ocasiones el equipo menos favorito termina llevándose el partido, aunque hay casos en donde Sports Analytics parece haber dado buenos resultados, sin ir más lejos en el último mundial de futbol donde **gano** Alemania, se utilizó Sports Analytics con una herramienta llamada SAP Sports One con la cual se pueden saber todas las estadísticas de los jugadores y se podrían tener datos de los rivales para realizar cierto análisis previo a cada partido, para este dejamos una nota de un diario canadiense en la bibliografía (item 3 bibliografía).

REDACCIÓN! 8 Renglones para una oración. Se entiende la idea, pero la redacción tiene que ser mejorada considerablemente.

6. Apéndice

6.1. Archivos de test usados

Dentro de la carpeta /src/tests se encuentran los siguientes archivos usados en la experimentación

- ATP2007.in este archivo es usado en la experimentación de escalar posiciones gandandole al siguiente o al primero del ranking
- ATP2007_100.in este archivo se utilizo en el analisis de salto de posiciones en cuanto a 100 partidos jugados para obtener una cota
- test1.in archivo provisto por la materia
- test2.ina archivo de prueba provisto por la materia
- carpeta random test tiene todos los archivos de test que se utilizaron para la medición de tiempos

7. Bibliografía

7.1. Bibliografía

- Numerical Analysis, Richard L. Burden & J. Douglas Faires, Chapter 6: Direct Methods for Solving Linear Systems.
- Paper The Colley Matrix Explained <http://www.colleyrankings.com/matracte.pdf>.
- SAP Sports Analytics <http://goo.gl/Q0dnH0>

8. Codigo

8.1. Sobre los archivos e implementacion

Para la implementacion de los archivo se utilizo C++, la siguiente es la lista de archivos y consecutivo a la misma hay una descripcion sobre los distintos archivos.

1. .. /src/main.cpp - este contiene la lectura de los archivos de entrada y escritura de la salida, asi como le ejecucion de cada metodo
2. .. /src/instancia/instancia.h - clase instancia, una instancia esta compuesta por la matriz CMM , el vector B, una matriz con los partidos ganados del equipo i al equipo j en la posicion (i,j), un arreglo con el total de los partidos. y las definiciones de los metodos para la clase
3. .. /src/instancia/instancia.cpp - este archivo contiene todas las implementaciones de los metodos, tanto para generar las matrizes como los setters y getters de los partes privadas de la clase instancia.
4. .. /src/matriz/matriz.h - Definicion clase matriz, con metodos de get y set y definicion de sus partes privadas y publicas.
5. .. /src/matriz/matriz.cpp - Aquise encuentra la implementacion de los metodos de la matriz.
6. .. /src/eliminaciongauss/elimgauss.h
7. .. /src/eliminaciongauss/elimgauss.cpp - aquise encuentra la implementaciond de EG
8. .. /src/cholesky/cholesky.h
9. .. /src/cholesky/cholesky.cp - Aqui se encuentra la implementacion de la factorizacion de Cholesky.
10. .. /src/wp/wp.h
11. .. /src/wp/wp.cpp - Aqui se encuentra la implementacion del metodo WP

La clase instancia la definimos para que sea mas facil el manejo de una instancia en general de juego, en base a su matriz CMM , matriz de partidos ganados y vector b, con fin de facilitarnos el uso de la entrada.

En cuanto a la implementacion del clase matri se utilizo un puntero a double donde en cada posicion hay otro puntero a double, ademas definimos setters y getters para las posiciones para que sea mas facil si uso y modularizar cada parte del programa, como los algoritmos relevantes a Eliminacion gaussiana, Cholesky y WP para una mas facil lectura.

8.2. Codigo implementado

```
#include "eliminaciongauss/elimgauss.h"
#include "matriz/matriz.h"
#include "instancia/instancia.h"
#include "wp/wp.h"
#include "cholesky/cholesky.h"
```

```

#include <algorithm>      // std::sort
#include <cmath>
#include <climits>
#include <vector>          // std::vector
#include <iostream>
#include <fstream>
#include <stdio.h>
#include <stdlib.h>
#include <sstream>
#include <fstream>
#include <string.h>
#include <sys/time.h>

using namespace std;

string intToString(int pNumber);
instancia * generarInstanciaDesdeArchivo(ifstream &archivoDeEntrada);
instancia * generarInstanciaVacia(ifstream &archivoDeEntrada);
void printVector(double * ,int );
bool pairCompare(const std::pair<int, double>& firstElem, const std::pair<int, double>& secondElem);

//El programa requiere 3 parametros, un archivo de entrada, uno de salida y el modo a ejecutar.
int main(int argc, char *argv[]) {
    timeval startGauss, endGauss;
    timeval startCholesky, endCholesky;
    timeval startModificado, endModificado;
    timeval startWP, endWP;
    int totalEquipos;
    long elapsed_mtime; /* elapsed time in milliseconds */
    long elapsed_seconds; /* diff between seconds counter */
    long elapsed_useconds; /* diff between microseconds counter */

    int i;
    // argumentos
    // 0 - main
    // 1 archivo entrada
    // 2 archivo salida
    // 3 modo
    if (argc != 4) {
        cout << "Error, Faltan Argumentos" << endl;
        return 1;
    }

    //leo archivo entrada
    ifstream archivoDeEntrada (argv[1]);

    //preparo archivo salida para escritura

```

```

ofstream archivoDeSalida;
ofstream archivoTiempos;
archivoDeSalida.setf(ios::fixed, ios::floatfield); // tipo salida
archivoTiempos.setf(ios::fixed, ios::floatfield); // tipo salida
archivoTiempos.precision(6); // cant decimales
archivoDeSalida.precision(6); // cant decimales
archivoDeSalida.open(argv[2]);


// genero una instancia Matriz de resultados Ganadores y vector
// de totales
instancia *ins= generarInstanciaDesdeArchivo(archivoDeEntrada);

totalEquipos = ins->getTotalEquipos();
// base para el resultado
double* respuesta = new double[ins->getTotalEquipos()];
for (i = 0; i < totalEquipos; ++i) {
    respuesta[i] = 0.0;
}

Matriz * CMM = ins->getCMM();
string totales = intToString(totalEquipos) + " " +
    intToString(ins->getTotalPartidos()) + " ";

// metodo Metodo CMM Con Gauss
if (strcmp(argv[3], "0") == 0) {
    cout << "Corriendo " "Metodo " "Gauss..." << endl;

    double timeGauss= 0.0;
    for (int iteraciones = 0; iteraciones<5; iteraciones++){
        gettimeofday(&startGauss, NULL);

        respuesta =gauss(CMM,ins->getVectorB());

        gettimeofday(&endGauss, NULL);

        elapsed_seconds = endGauss.tv_sec - startGauss.tv_sec;
        elapsed_useconds = endGauss.tv_usec - startGauss.tv_usec;

        //if (((elapsed_seconds) * 1000 + elapsed_useconds /
        //      1000.0) + 0.5 < timeGauss){
        timeGauss+= ((elapsed_seconds) * 1000 + elapsed_useconds
            / 1000.0) + 0.5;
        //}
    }
    timeGauss= timeGauss/5;

    archivoTiempos.open("tiempos/tiempos0.txt", std::ofstream::
        out | std::ofstream::app);
    archivoTiempos << totalEquipos << " " << ins->
        getTotalPartidos() << " " <<timeGauss<< endl;
}

```

```

        archivoTiempos.close();
    }
    // metodo Metodo CMM Con CHOLESKY

    if (strcmp(argv[3], "1") == 0 || strcmp(argv[3], "3") == 0 ||
        strcmp(argv[3], "4") == 0 || strcmp(argv[3], "5") == 0) {
        cout << "Corriendo\u2014Metodo\u2014Cholesky..." << endl;
        gettimeofday(&startCholesky, NULL);

        //LLAMO CHOLESKY
        respuesta = cholesky(CMM, ins->getVectorB());

        gettimeofday(&endCholesky, NULL);
        elapsed_seconds = endCholesky.tv_sec - startCholesky.tv_sec;
        elapsed_useconds = endCholesky.tv_usec -
            startCholesky.tv_usec;
        // aca se guarda el tiempo
        double timeCholesky = ((elapsed_seconds) * 1000 +
            elapsed_useconds / 1000.0) + 0.5;

        archivoTiempos.open("tiempos/tiempos1.txt", std::ofstream::
            out | std::ofstream::app);
        archivoTiempos << totalEquipos << "\u2014" << ins->
            getTotalPartidos() << "\u2014" << timeCholesky << endl;
        archivoTiempos.close();
    }

    // metodo WP
    if (strcmp(argv[3], "2") == 0) {
        cout << "Corriendo\u2014Metodo\u2014WP..." << endl;

        gettimeofday(&startWP, NULL);
        respuesta = wp(ins);
        gettimeofday(&endWP, NULL);
        elapsed_seconds = endWP.tv_sec - startWP.tv_sec;
        elapsed_useconds = endWP.tv_usec - startWP.tv_usec;
        double timeWP = ((elapsed_seconds) * 1000 + elapsed_useconds
            / 1000.0) + 0.5;
        archivoTiempos.open("tiempos/tiempos2.txt", std::ofstream::
            out | std::ofstream::app);
        archivoTiempos << totalEquipos << "\u2014" << ins->
            getTotalPartidos() << "\u2014" << timeWP << endl;
        archivoTiempos.close();
    }

    if (strcmp(argv[3], "3") == 0) {
        ofstream archivoModificadoCHOLESKY;
        // aca tengo el ranking // base para el resultado
    }
}

```

```

double* respuestaModificada = new double[totalEquipos];
for (i = 0; i < totalEquipos; ++i) {
    respuestaModificada[i] = 0.0;
}

cout << "Corriendo \u2225 Metodo \u2225 CHOLESKY \u2225 RANDOM \u2225 100 \u2225 partidos..." <<
     endl;
// aca se modifican los partidos ganados
// esto pudo haber sido por entrada
for (i = 0; i < 100; ++i) {
    // agarro uno random
    int e1 = rand() % totalEquipos;
    // le hago ganar un perder contra uno que haya ganado
    ins->ganaPartido(e1);
}
// genero el nuevo B
ins->generarVectorB();

gettimeofday(&startModificado, NULL);
//LLAMO CHOLESKY
respuestaModificada = cholesky(ins->getCMM(), ins->
                                getVectorB());

gettimeofday(&endModificado, NULL);
elapsed_seconds = endModificado.tv_sec -
                  startModificado.tv_sec;
elapsed_useconds = endModificado.tv_usec -
                     startModificado.tv_usec;
// aca se guarda el tiempo
double timeModificado = ((elapsed_seconds) * 1000 +
                           elapsed_useconds / 1000.0) + 0.5;

archivoTiempos.open("tiempos/tiempos4.txt", std::ofstream::
                    out | std::ofstream::app);
archivoTiempos << totalEquipos << "\u2225" << ins->
                    getTotalPartidos() << "\u2225" << timeModificado << endl;
archivoTiempos.close();

archivoModificadoCHOLESKY.open("tests/resultadosCholeskyModificado.out"
                                 std::ofstream::out | std::ofstream::app);
for (int w = 0; w < totalEquipos; w++) {
    archivoModificadoCHOLESKY << respuestaModificada[w] <<
        endl;
}
archivoModificadoCHOLESKY.close();
}

if (strcmp(argv[3], "4") == 0 || strcmp(argv[3], "5") == 0) {
    int t;

```

```

const char* salida;
if(strcmp(argv[3], "4") == 0 ){
    salida= "tests/rankingSTEPS_4.out";
} else{
    salida= "tests/rankingSTEPS_5.out";
}

ofstream archivoModificadoCHOLESKY;

double min = INT_MAX +0.0;
int minPOS =0;
int actPOS =0;
bool esPrimero = false;
// busco el minimo;
for (i = 0; i < totalEquipos; i++) {
    if(min>respuesta[i]){
        minPOS = i;
        min = respuesta[i];
    }
}
// esto no es lo mas lindo pero es solo para limpiar el
// archivo anterior
archivoModificadoCHOLESKY.open(salida, std::ofstream::out |
    std::ofstream::trunc);
archivoModificadoCHOLESKY<<"Imprimo\u00d7Cholesky\u00d7en\u00d7el\u00d7primer\u00d7
    paso\u00d7"<< endl;
archivoModificadoCHOLESKY.close();

// esto se va a ejecutar mientras el jugador no este en el
// primer puesto
for (t = 0; !esPrimero; t++) {
    double nextminPOS =0;
    vector<pair<int,double> > rankSorted;
    for (i = 0; i < totalEquipos;i++) {
        pair<int,double> p(i,respuesta[i]);
        rankSorted.push_back(p);
    }
    // ordeno el ranking actual
    std::
        sort(rankSorted.begin(),rankSorted.end(),pairCompare);
    for (i = 0; i < totalEquipos; i++) {
        if(rankSorted[i].first==minPOS){
            if(strcmp(argv[3], "4") == 0 ){
                // si es el metodo 4 agarro el siguiente
                nextminPOS = i+1;
            } else{
                // si es el metodo 5 agarro el q esta primero
                nextminPOS = totalEquipos-1;
            }
            actPOS = i;
        }
    }
}

```

```

        }
    }

    // imprimo el ranking y numero de partido
    archivoModificadoCHOLESKY.open(salida, std::ofstream::out
        | std::ofstream::app);
    for (int w = 0; w < totalEquipos; w++) {
        archivoModificadoCHOLESKY << intToString(t) << " "
            << rankSorted[w].first << " " <<
            rankSorted[w].second << endl;
    }
    archivoModificadoCHOLESKY << endl;

    //esta funcion hace ganar un partido al primero contra el
    //segundo
    ins->
        ganaPartidoContra(minPOS, rankSorted[nextminPOS].first);
    if(actPOS==totalEquipos-1){
        esPrimero = true;
        archivoModificadoCHOLESKY << "Cantidad_total_de_"
            "partidos" << t << endl;
        cout << "Cantidad_total_de_partidos" << t-1 << endl;
        break;
    }
    archivoModificadoCHOLESKY.close();
    // ejecuto cholesky para el nuevo partido
    respuesta= cholesky(ins->getCMM(),ins->getVectorB());
}

return 0;
}

//para imprimir una instancia (Matriz resultados, Vector totales
//y matriz CMM)
// ins->print();
for (int w = 0; w < ins->getTotalEquipos(); w++) {
    archivoDeSalida << respuesta[w] << endl;
}

archivoDeSalida.close();
archivoDeEntrada.close();
return 0;
}

string intToString(int pNumber)
{
    ostringstream oOStrStream;
    oOStrStream << pNumber;
    return oOStrStream.str();
}

```

```

instancia *generarInstanciaDesdeArchivo(ifstream &archivoDeEntrada){
    int n,k,i,fecha;
    int equipo1, equipo2, goles1, goles2;

    //leo cantidad de equipos
    archivoDeEntrada >> n;
    //leo cantidad de partidos
    archivoDeEntrada >> k;
    //creo la tabla de resultados ganadores
    Matriz * tablaResultados = new Matriz(n,n);
    //creo la tabla de partidos totales
    int* totales = new int[n];
    for (i = 0; i < n; ++i) {
        totales[i]=0;
    }

    if (archivoDeEntrada.is_open())
    {
        for (i = 0; i < k; ++i) {
            //primer linea es fecha
            archivoDeEntrada >> fecha;
            //segunda linea es el numero del primer equipo
            archivoDeEntrada >> equipo1;
            //tercer linea es la cantidad de goles del primer equipo
            archivoDeEntrada >> goles1;
            //cuarta linea es el numero del segundo equipo
            archivoDeEntrada >> equipo2;
            //quinta linea es la cantidad de goles del segundo equipo
            archivoDeEntrada >> goles2;

            totales[equipo1-1]++;
            totales[equipo2-1]++;

            if(goles1>goles2){
                int actual = tablaResultados->
                    getVal(equipo1-1,equipo2-1);

                tablaResultados->setVal(equipo1-1,equipo2-1,actual+1);

            }else{
                int actual = tablaResultados->
                    getVal(equipo2-1,equipo1-1);

                tablaResultados->setVal(equipo2-1,equipo1-1,actual+1);
            }
        }
        archivoDeEntrada.close();
    }
    instancia *res =new instancia();
    res->setTotalPartidos(k);
}

```

```

res->setGanados(tablaResultados);
res->setTotales(totales);
res->generarCMM();
res->generarVectorB();
return res;
}

void printVector(double * vec,int longitud){
    int i;
    for (i = 0; i < longitud; ++i) {
        cout << vec[i]<< endl;
    }
}

bool pairCompare(const std::pair<int, double>& firstElem, const std::pair<int, double>& secondElem) {
    return firstElem.second < secondElem.second;
};

#ifndef __Matriz_H_INCLUDED__ // #define this so the compiler
// knows it has been included
#define __Matriz_H_INCLUDED__ // #define this so the compiler
// knows it has been included

class Matriz {
private:
    int m;
    int n;
    double** matrix;
    bool posicionValida(int x, int y);
    Matriz * L;
    Matriz * U;

public:
    Matriz(int a, int b);
    ~Matriz();
    int getM();
    int getN();
    double** getMatrix();
    void setVal(int x, int y, double val);
    double getVal(int x, int y);
    void printM();
    Matriz *copy();
    void gauss(Matriz *m,double *b);
    void restar_fila(Matriz *m, int fila_minuendo, int
                      fila_sustraendo, double *b);
};

#endif

#include "matriz.h"

```

```

#include <algorithm>
#include <iostream>
#include <cmath>

using namespace std;

/**
 * Constructor de Matriz banda
 *
 * @param {int} dimension
 * @param {int} longitud de banda
 *
 * @return {Matriz} nueva instancia de Matriz
 */

Matriz::Matriz(int _n,int _m) {
    m = _m;
    n = _n;
    // Seteo el tamano del vector que representa la matriz
    // con tamano banda*2 + 1
    matrix = new double*[n*m];
    //inicializo la matriz de manera segura (evitando problemas de
    manejo de memoria)
    for (int i = 0; i < n; i++) {
        matrix[i] = new double[n];
        for (int j = 0; j < m; j++) {
            matrix[i][j] = 0.0;
        }
    }
}

Matriz::~Matriz() {
    for (int i = 0; i <n ; i++) {
        delete[] matrix[i];
    }
    delete [] matrix;
}

/**
 * matrix getter
 */

double** Matriz::getMatrix() {
    return matrix;
}

/**
 * n getter
 */

int Matriz::getN() {

```

```

        return n;
    }

/***
 * m getter
 **/

int Matriz::getM() {
    return m;
}

/***
 * Obtener el valor de la matriz en un punto
 *
 * @param {int} x
 * @param {int} y
 *
 * @return {double} m(x, y)
 *
 * x1   x2   0   0   0
 * x3   x4   x5   0   0
 * 0    x6   x7   x8   0
 * 0    0    x9   x10  x11
 *
 * Para ahorrar los 0s lo transformo en =>
 * x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11
 **/

double Matriz::getVal(int x, int y) {
    if (posicionValida(x, y)) {
        return matrix[x][y];
    } else {
        return 0.0;
    }
}

/***
 * Setear valor en un punto
 *
 * @param {int} x
 * @param {int} y
 * @param {double} valor a setear
 **/

void Matriz::setVal(int x, int y, double val) {
    if (posicionValida(x, y)) {
        matrix[x][y] = val;
    } else {
        // Si x, y est n fuera de rango tira una excepcion
        cout << "Intento de seteo en coordenadas: (" << x << ", "
             << y << ")" << endl;
    }
}
```

```

        throw 0;
    }
}

<**
 * Devuelve true si x e y forman una posicion valida en la matriz
 *
 * @param {int} x
 * @param {int} y
 *
 * @return {bool} true si es valida, false si no
 *
 * x1  x2  0  0  0
 * 0  x5  x6  0  0
 * 0  0  x9  x10 0  0
 */

```

```

bool Matriz::posicionValida(int x, int y) {
    // Si no cumple ninguna de las condiciones previas, es verdadero
    return x<this->getN() && y<this->getM();
}

<**
 * Imprime la matriz de un modo conveniente
 */

```

```

void Matriz::printM() {
    int l = 0;
    int i, j;
    int n = this->getN();
    int m = this->getM();
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            if (this->getVal(i, j) == 0) {
                cout << this->getVal(i, j);
            } else {
                cout << this->getVal(i, j);
                l++;
            }
            cout << "\t";
        }
        cout << endl;
    }
}

```

```

Matriz *Matriz::copy() {
    Matriz* aux = new Matriz(n,m);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            aux->matrix[i][j] = matrix[i][j];
        }
    }
}

```

```

        }
    }
    return aux;
}
// NO SE USO
void Matriz::gauss(Matriz *m, double * b){
    int filas = m->getN();
    int columnas = m->getM();

    for (int i = 0; i < columnas; ++i) {
        if(!(abs(m->getVal(i,i))<0)){
            for (int fila = i + 1 ; fila < filas; fila++) {
                restar_fila(m, fila, i, b);
            }
        }
    }
};

// NO SE USO SE PUEDE BORRAR
void Matriz::restar_fila(Matriz *m, int fila_minuendo, int
fila_sustraendo, double *b) {
    int columna = fila_sustraendo;
    if (fila_minuendo >= m->getN())
        return;
    double coeficiente = m->getVal(fila_minuendo, fila_sustraendo) /
        m->getVal(fila_sustraendo, fila_sustraendo);
    m->setVal(fila_minuendo, columna, 0.0);
    columna++;
    if ( abs(coeficiente)!= 0){
        b[fila_minuendo] = b[fila_minuendo] - b[fila_sustraendo] *
        coeficiente;
        while (columna <= m->getN() + fila_sustraendo) {
            if (columna > m->getN() - 1) {
                break;
            }
            m->setVal(fila_minuendo, columna, m->
                getVal(fila_minuendo, columna) - m->
                getVal(fila_sustraendo, columna)*coeficiente);
            columna++;
        }
    }
}

#ifndef __instancia_H_INCLUDED__ //      #define this so the compiler
knows it has been included
#define __instancia_H_INCLUDED__ //      #define this so the compiler
knows it has been included

#include "../matriz/matriz.h"
#include <iostream>
using namespace std;

class instancia{

```

```

private:
    // esta matriz tiene en cada posicion cuantos partidos gano
    // el equipo i contar el j
    // para obtener los que perdio ir a la posicion j , i
    Matriz * ganados;

    // esta es la matriz CMM
    Matriz * CMM;

    // este tiene para cada equipo los partidos totales
    int * totales ;
    int totalPartidos;

    double * b;

public:
    ~instancia();
    void print();
    // estas funciones son para la mariz ganados
    // esta funcion te da el total de equipos en este caso es
    // igual la cant de columnas de ganados
    int getTotalEquipos();

    int getTotalPartidos();
    // el primero siempre es el que gana el partido
    bool ganaPartido(int);
    void ganaPartidoContra(int,int);
    void setTotalPartidos(int);
    // esta funcion te dice cuantos partidos GANADOS tiene el
    // equipo q recibe por parametro
    double getTotalGanados(int);

    // esta funcion te dice cuantos partidos PERDIDOS tiene el
    // equipo q recibe por parametro
    double getTotalPerdidos(int);

    // esta funcion te da la cantidad total de partidos jugados
    // para el equipo que recibe por parametro
    double getTotalJugados(int);

    // esta funcion te da el total de encuentros GANADOS o
    // PERDIDOS entre dos equipos
    double getTotalJugadosEntreEquipos(int,int);

    // setters para privados
    void setGanados(Matriz* );
    void setTotales(int* );

    // getters para privados
    Matriz* getGanados();
    int* getTotales();

```

```

    Matriz* getCM();
    // esta funcion genera la matriz CMM
    void generarCM();
    void generarVectorB();
    double* getVectorB();
};

#endif

#include "instancia.h"

instancia::~instancia(){
    delete ganados;
    delete totales;
}

void instancia::print(){
    int i, totalEquipos;

    totalEquipos=this->getTotalEquipos();

    cout <<"Matriz de Partidos en enfrentamientos ganados" << endl;
    ganados->printM();
    cout <<"vector de totales" << endl;
    for (i = 0; i < totalEquipos; ++i) {
        cout <<totales[i] << endl;
    }

    cout <<"Matriz CMM" << endl;
    CMM->printM();

    cout <<"vector b" << endl;
    for (i = 0; i < totalEquipos; ++i) {
        cout <<b[i] << endl;
    }
}

void instancia::setTotalPartidos(int _totales){
    this->totalPartidos = _totales;
};

int instancia::getTotalPartidos(){
    return this->totalPartidos;
};

int instancia::getTotalEquipos(){
    return ganados->getN();
}

```

```

double instancia::getTotalGanados(int numeroDeEquipo){
    int totalEquipos = getTotalEquipos();
    int i;
    double resultado = 0.0 ;

    for (i = 0; i < totalEquipos; ++i) {
        resultado += ganados->getVal(numeroDeEquipo ,i);
    }
    return resultado;
}

//Total partidos perdidos = total partidos jugados - partidos ganados
double instancia::getTotalPerdidos(int numeroDeEquipo){
    return this->getTotalJugados(numeroDeEquipo)-this->
        getTotalGanados(numeroDeEquipo);
}

// ESTA FUNCION SIRVE PARA ARMAR CMM
// n_i,j numero de enfrentamientos entre i y j= los ganados por i +
// los ganados por j
double instancia::getTotalJugadosEntreEquipos(int numeroDeEquipo1,int
    numeroDeEquipo2){
    return this->ganados->
        getVal(numeroDeEquipo1,numeroDeEquipo2)+this->ganados->
        getVal(numeroDeEquipo2,numeroDeEquipo1);
}

double instancia::getTotalJugados(int numeroDeEquipo){
    return this->totales[numeroDeEquipo];
}

// Cij = nij si i!=j,
// Cij = 2 + ni si i = j.
void instancia::generarCMM(){
    int i,j;
    double nij;
    int totalEquipos = this->getTotalEquipos();
    CMM = new Matriz(totalEquipos,totalEquipos);

    for (i = 0; i < totalEquipos; ++i) {
        for (j = 0; j < totalEquipos; ++j) {
            if(i==j){
                nij =this->getTotalJugados(i)+2.0;
            }else{
                nij =(-1.0)* this->getTotalJugadosEntreEquipos(i,j);
            }
            CMM->setVal(i,j,nij);
        }
    }
}

```

```

// setters
void instancia::setTotales(int* _totales){
    totales=_totales;
};
void instancia::setGanados(Matriz* _ganados){
    ganados = _ganados;
};

// getters
int* instancia::getTotales(){
    return totales;
};

Matriz *instancia::getGanados(){
    return ganados;
};

Matriz *instancia::getCMM(){
    int i, j;

    Matriz * newCMM = new Matriz(this->getTotalEquipos(),this->
        getTotalEquipos());

    for (i = 0; i < this->getTotalEquipos(); i++) {
        for (j = 0; j < this->getTotalEquipos(); j++) {
            newCMM->setVal(i,j,this->CMM->getVal(i,j));
        }
    }

    return CMM;
};

// esta funcion genera el vector B pedido por CMM
void instancia::generarVectorB(){
    b = new double[ganados->getN()];
    int i;

    for (i = 0; i < ganados->getN(); ++i) {
        b[i]=
            1.0+((double) getTotalGanados(i)-(double) getTotalPerdidos(i))/2.0;
    }
}

// esta funcion lo que hace es darle un partido mas ganado al equipo1
void instancia::ganaPartidoContra(int equipo1,int equipo2){
    int totalJugados1 = this->getTotalJugados(equipo1);
    int totalJugados2 = this->getTotalJugados(equipo2);
    // le sumo un partido mas en partidos totales
}

```

```

this->totales[equipo1]=totalJugados1+1;
this->totales[equipo2]=totalJugados2+1;
// en la matriz de partidos ganados le sumo uno al primero
int totalganadose1e2 = this->ganados->getVal(equipo1,equipo2);
this->ganados->setVal(equipo1,equipo2,totalganadose1e2+1);
// genero la nueva CMM
this->generarCMM();
// genero el nuevo vector B
this->generarVectorB();
};

// Esta funcion lo que hace es cambiar un partido ganado por perdido
// y se lo suma al contrincante,
// el partido seleccionado es uno que se jugo y fue ganado
bool instancia::ganaPartido(int equipo1){
    bool encontroUno = false;
    int i;
    for (i = 0; i < this->getTotalEquipos(); i++) {
        if(this->ganados->getVal(equipo1,i)!=0 && i!=equipo1){
            // agarro el valor viejo
            int valor = this->ganados->getVal(equipo1,i);
            // le cambio el resultado al partido
            this->ganados->setVal(equipo1,i,valor-1);
            // le hago ganar al otro equipo
            int valor2 = this->ganados->getVal(i,equipo1);
            this->ganados->setVal(i,equipo1,valor2+1);
            encontroUno = true;
        }
    }
    return encontroUno;
}

double* instancia::getVectorB(){
    return b;
}

#ifndef __Gauss_H_INCLUDED__
#define __Gauss_H_INCLUDED__

#include "../matriz/matriz.h"
#include <iostream>
using namespace std;

double* gauss(Matriz*,double *b);

#endif

#include <cmath>
#include "elimgauss.h"

using namespace std;
/**/

```

```

* Chequea que 2 doubles difieran en menos que un delta
*
* @param {double} a
* @param {double} b
* @param {double} delta
*
* @return {bool} son iguales
*/
bool son_iguales(double a, double b, double delta = 1.0e-10) {
    return abs(a - b) < delta;
}
/***
 * Algoritmo de eliminacion gaussiana
 *
* @param {MatrizB*} m
* @param {double*} b
*/
double* gauss(Matriz *m, double * b) {

    //elijo el valor para el pivot
    for (int pivot = 0; pivot < m->getM()-1; pivot++) {

        // por cada fila desde el valor pivot + 1 para abajo
        for (int fila = pivot+1 ; fila < m->getN() ; fila++) {

            //si el valor de la diagonal es cero paso a la siguiente
            //columna
            // if (! son_iguales(m->getVal(pivot, fila), 0.0)) {

                // genero el valor que va a multiplicar la fila pivot
                // (a_i_pivot / a_pivot_pivot)
                double valor_pivot = m->getVal(fila, pivot) / m->
                    getVal(pivot, pivot) ;

                //cout<<"valor pivot = "<<valor_pivot<<" : getval("<<
                // fila<<","<<pivot <<") = " << m->getVal(fila, pivot) <<
                // " / getval("<< pivot<<","<< pivot<<")" <<m->
                // getVal(pivot, pivot) <<endl;

                b[fila] = b[fila] - (b[pivot] * valor_pivot);
                // Teniendo el valor_pivot , resto toda las columnas de
                // esa fila
                for (int columna = pivot ; columna < m->getN() ;
                    columna++) {

                    //((BORRAR LUEGO !!!)) cout << m->getVal(fila, columna)
                    // << " - " << m->getVal(pivot, columna) << " * "<<
                    // valor_pivot << " = "<< (m->getVal(pivot, columna) *
                    // valor_pivot) << " = "<< m->getVal(fila, columna) -
                    // (m->getVal(pivot, columna) * valor_pivot) <<endl;

```

```

// genero los valores de la matriz de gauss F_fila =
F_fila - F_pivot * valor_pivot
double valor_fila_columna = m->getVal(fila, columnna)
- (m->getVal(pivot, columnna) * valor_pivot);

// chequeo para descartar errores de representacion
if(son_iguales(valor_fila_columna, 0.0)){
    valor_fila_columna=0.0;
}
m->setVal(fila, columnna, valor_fila_columna);
//(BORRAR LUEGO !!!) cout <<
"-----" << endl;

}
// }
}
// cout << "a resolver
gauss=====
// resolver sistema Gr=b
double* result = new double[m->getN()];

for (int i = m->getN()-1 ; i >= 0 ; i--){
// cout << "i= "<<i << endl;
double acum_suma=0.0;
for(int j = m->getN()-1 ; j > i ; j-- ){
// cout << "j= "<<j << endl;

acum_suma += m->getVal(i,j)*result[j] ;

}

// cout << "b ["<<i<<"]= "<<b[i]<< endl;
// cout << "acum_suma= "<<acum_suma<< endl;
// cout << "m ["<<i<<","<<i<<"]= "<<m->getVal(i,i)<< endl;

result [i] = (b[i] - acum_suma) /(double) m->getVal(i,i);

// cout << "result ["<<i<<"]= "<<result[i]<< endl;

}
// cout << "vector solucion" << endl;
// for (int i=0 ; i<m->getM();i++){
// cout << "x" <<i<<"=" << result [i] << endl;
// }
// cout << "a resolver
gauss=====
return result;
}

#ifndef __CHOLESKY_H_INCLUDED__ // #define this so the compiler
knows it has been included

```

```

#define __CHOLESKY_H_INCLUDED__      //      #define this so the compiler
                                    knows it has been included
#include "../matriz/matriz.h"

double* cholesky(Matriz* m ,double *b);

#endif

#include "../cholesky/cholesky.h"
#include <iostream>

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
using namespace std;

double* cholesky(Matriz* m,double *b){
    int cantFilas = m->getN();
    int i,k,j;
    double *rank = new double[cantFilas];
    double *y= new double[cantFilas];
    Matriz * L = new Matriz(cantFilas,cantFilas);

    // ACA EMPIEZO A GENERAR LA MATRIZ L
    double l11 = sqrt(m->getVal(0,0));
    L->setVal(0,0,l11);

    for (i = 1; i < cantFilas; ++i) {
        double aj1 = m->getVal(i,0);
        double lj1= aj1/l11;
        L->setVal(i,0,lj1);
    }

    for (i = 1; i < cantFilas-1; i++) {

        //  $l_{ii} = a_{ii} - \sum_{k=1, k \neq i} a_{ik} l_{kk}$ 
        // step 4
        double aii= m->getVal(i,i);
        double acum = 0.0;
        for (k = 0; k < i; k++) {
            double lik = L->getVal(i,k);
            acum += lik*lik;
        }
        L->setVal(i,i,sqrt(aii-acum));
        // step 5
        for (j = i+1; j< cantFilas; j++) {
            double aux = 0.0;
            double aji = m->getVal(j,i);
            for( k = 0; k < i ; k++){
                double ljk = L->getVal(j,k);
                double lik = L->getVal(i,k);
                aux +=ljk*lik;
            }
            L->setVal(j,i,aux);
        }
    }
}

```

```

        }

        double lji = (aji-aux)/L->getVal(i,i);
        L->setVal(j,i,lji);
    }
}

double acum = 0.0;
for( k = 0; k < cantFilas ; k++){
    acum += L->getVal(cantFilas-1,k)*L->getVal(cantFilas-1,k);
}
L->setVal(cantFilas-1,cantFilas-1,sqrt(m->
    getVal(cantFilas-1,cantFilas-1)-acum));
// ACA TERMINE DE GENERAR L
// en L tengo la matriz diagonal inferior de cholesky

// pasos 8 y 9 del burden L*y= b
y[0]= b[0]/L->getVal(0,0);

for (i = 1; i < cantFilas; i++) {
    double acum=0.0;
    for (j = 0; j < i; j++) {
        acum += y[j]*L->getVal(i,j);
    }

    y[i] =( b[i]- acum)/L->getVal(i,i);
}

// aca genero x para la salida
rank[cantFilas-1]= y[cantFilas-1]/L->
    getVal(cantFilas-1,cantFilas-1);

for (i = cantFilas-2; i >-1; i--) {
    double acum=0.0;
    for (j = i+1; j < cantFilas; j++) {
        acum += rank[j]*L->getVal(j,i);
    }
    rank[i] =( y[i]- acum)/L->getVal(i,i);
}
return rank;
};

#ifndef __wq_H_INCLUDED__ // #define this so the compiler knows
it has been included
#define __wq_H_INCLUDED__ // #define this so the compiler knows
it has been included

#include "../instancia/instancia.h"
#include <iostream>
using namespace std;

double* wp(instancia*);
```

```

#endif

#include "wp.h"
using namespace std;

double* wp(instancia* ins){
    int i = 0;
    int cantEquipos = ins->getTotalEquipos();
    double* res = new double[cantEquipos];
    //limpio array
    for (i = 0; i < cantEquipos; ++i) {
        res [i]=0.0;
    }

    for (i = 0; i < cantEquipos; ++i) {
        res[i] = double(ins->getTotalGanados(i))/double(ins->
            getTotalJugados(i));
    }

    return res;
};

```