BAL: Biblioteca de Álgebra Lineal Numérica

Gastón Simone, Pablo Ezzatti

Instituto de Computación - Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay
gaston.simone@gmail.com
pezzatti@find.edu.uy

Junio, 2008

Palabras claves: Álgebra Lineal Numérica, C, Algoritmos.

Resumen

La resolución de una gran cantidad de modelos presentes en la computación científica se basan en la solución de problemas del álgebra lineal numérica (ALN). Esta situación ha motivado fuertemente el desarrollo del área. En contraposición al importante desarrollo se ha incrementado en forma abrupta la complejidad de las estrategias de resolución, dificultando la comprensión por parte de los alumnos de los algoritmos utilizados.

En este contexto la propuesta se centra en el desarrollo de una biblioteca de ALN de carácter didáctico. Por esta razón, la documentación es vasta y el código fue escrito pensando en su fácil lectura. Las rutinas implementadas son eficientes desde el punto de vista algorítmico. Pero determinadas mejoras de desempeño, propias de la implementación, fueron descartadas para mantener la legibilidad del código.

Este documento presenta el diseño, las funcionalidades y la implementación realizada.

Índice

1. Biblioteca the Álgebra Lineal (BAL)

1.1. Introducción

BAL es una biblioteca que contiene un conjunto de utilidades e implementaciones de algoritmos esenciales para el estudio del álgebra lineal numérica. Está centrada en técnicas utilizadas para matrices dispersas, pero no necesariamente se limita a este campo.

El objetivo principal de esta biblioteca es didáctico. Pretende servir como ejemplo de implementación acompañada de documentación de estructuras de datos y algoritmos para matrices dispersas.

Es de destacar que esta documentación es totalmente generada a partir de los archivos fuentes de BAL (incluso esto que estás leyendo).

1.2. Historia de BAL

- **Versión:** 1.0.0
- **Cuándo:** Curso ALN 2008
- **Quién:** Gastón Simone (gaston.simone@gmail.com) con la colaboración de Pablo Ezzatti (pezzatti@fing.edu.uy)
- **Descripción:** Implementación inicial
- **Notas:** Ver notas
- **Detalle:**
  - Cargas de datos desde archivo
    - Parser de definición de matrices en formato matlab leídas desde archivo (bal_cargar_matriz())
    - Carga de matrices en formato simple por coordenadas desde archivo (bal_load_coord())
  - Distintos formatos de matrices dispersas:
    - Simple por coordenadas (sp_coord)
    - Empaquetado por columnas (sp_packcol)
    - CDS o comprimido por diagonales (sp_cds)

- Funciones para imprimir estructuras de datos:
  - Matriz completa (bal_imprimir_matriz())
  - Detalle para formato simple por coordenadas (bal_imprimir_coord())
  - Detalle para formato empaquetado por columna (bal_imprimir_packcol())
  - Detalle para formato CDS (bal_imprimir_cds())
  - De simple por coordenadas a formato texto específico (bal_save_coord())
  - De empaquetado por columna a formato matlab (bal_save_packcol(), bal_save_packcol_-symmetric())
  - De CDS a formato matlab (bal_save_cds())
- Conversiones entre formatos:
  - De matriz completo a simple por coordenadas (bal_mat2coord())
  - De simple por coordenadas a empaquetado por columnas (bal_coord2packcol())
  - De simple por coordenadas a empaquetado por columnas específico para matrices simétricas (bal_coord2packcol_symmetric())
  - De simple por coordenadas a CDS (bal_coord2cds())
- Funciones de soporte:
  - Funciones de liberación de memoria reservada por estructuras de datos (bal_free_coord(), bal_free_packcol(), bal_free_cds())
- Operaciones básicas:
  - Multiplicar una matriz empaquetada por columna por un vector (bal_mult_vec_packcol())
  - Multiplicar una matriz simétrica empaquetada por columna por un vector (bal_mult_vec_-packcol_symmetric())
  - Multiplicar una matriz CDS por un vector (bal_mult_vec_cds())
  - Multiplicar dos matrices CDS (bal_mult_mat_cds())
  - Permutar las columnas de una matriz empaquetada por columna (bal_permutar_packcol())
- Factorización de Cholesky (bal_cholesky_solver())
  - Funcionalidad para recorrer eficientemente por filas una matriz dispersa empaquetada por columnas (bal_row_traversal_packcol())
  - Funcionalidad para calcular el árbol de eliminación de una matriz empaquetada por columna (bal_elimination_tree())
  - Factorización simbólica para matrices simétricas empaquetadas por columna (bal_-symbolic_factorization())
  - Factorización numérica (algoritmo de Cholesky) para matrices simétricas empaquetadas por columna (bal_numerical_factorization())
  - Resolución de sistemas con matrices triangulares empaquetadas por columna (bal_-cholesky_Lsolver(), bal_cholesky_LTsolver())
- Programas de prueba de todas las funcionalidades implementadas (ver la sección de Pruebas)

Tareas Pendientes

A continuación se enumeran algunas de las características que por distintas razones no fueron implementadas en BAL 1.0.0, pero que destacamos como importantes. Algunas de las tareas ya se están abordando mientras que otras se preve atacarlas en un futuro cercano.

Soporte para más estructuras de matrices dispersas.

Más conversiones entre estructuras de matrices dispersas.

Extender el parser para que cargue una matriz desde archivo directamente en un formato disperso.

Algoritmos de reordenamiento (ver reordenamiento.c).

Más algoritmos de factorización simbólica.

Más algoritmos de factorización numérica (LU por ejemplo).

Métodos iterativos (Jacobi, Gauss-Seidel, Gradiente conjugado, etc).

Algoritmos de creación de precondicionadores (ILU por ejemplo).

Agregar salidas detalladas para cada uno de los algoritmos, en base a un parámetro, mediante el que se pueda controlar la información a mostrar. Por ejemplo: memoria utilizada, tiempos de ejecución, cantidad de operaciones.

Extender las funciones que corresponda (por ejemplo, cholesky_solver()) para que reciban un vector de parámetros, en el cual se indique la estrategia de ordenamiento, el método de factorización simbólica, etc.

Agregar soporte para paralelismo.

Agregar soporte para el uso de bibliotecas BLAS.

Agregar algoritmos que utilicen técnicas por bloques.

Rutinas de refinamiento iterativo.

Vectores y valores propios.

Escalado.

Agregar la posibilidad de compilar/instalar sin el parser (para sistemas que no cumplan con todas las dependencias necesarias).

1.3. Cómo usar BAL

En esta sección se describe información de utilidad para poder trabajar con BAL, ya sea extendiendo BAL o simplemente utilizándola.

1.3.1. Requisitos para compilar BAL

La biblioteca fue desarrollada y probada bajo la plataforma GNU/Linux (concretamente en Ubuntu 7.10). Sin embargo no hay razón para que BAL no funcione en otros sistemas, incluso con Windows y Cygwin.

Todos los requisitos para ejecutar BAL deberían estar disponibles en las distribuciones más utilizadas de Linux (no quizás en la instalación por defecto, pero sí como paquetes fácilmente instalables), sin embargo se agregaron enlaces a la lista para que sea más fácil encontrar las dependencias en caso de ser necesario.

BAL depende de los siguientes componentes para compilar correctamente:

- GNU Compiler Collection (GCC). Concretamente el compilador de C (BAL fue probado con la versión 4.1.3).
- GNU Make (BAL fue probado con la versión 3.81)
- GLib (BAL fue probado con la versión 2.0)
- GNU Bison (BAL fue probado con la versión 2.3)
- Flex (BAL fue probado con la versión 2.5.33)

Aunque no es estrictamente necesario, puede ser útil contar con los siguientes componentes adicionales:

- Doxygen (necesario para generar esta documentación, BAL fue probado con la versión 1.5.3)
- Una distribución del sistema LaTeX (necesario para generar la documentación, tanto en PDF como en HTML, BAL fue probado con TeX Live 2007)
- GNU Debugger (gdb)
- Alguna interfaz gráfica para gdb, como Anjuta, insight, Data Display Debugger (ddd) o Nemiver
- Un generador de tablas de referencias al estilo CTags (para el desarrollo de BAL se utilizó Exuberant CTags)

1.3.2. Cómo compilar e instalar BAL

BAL incluye un conjunto de `makefiles` que automatizan el trabajo de compilación e instalación. A continuación se listan las distintas llamadas que se le pueden realizar al `makefile` de BAL:

- `make`: Este modo compila todo lo necesario para generar la biblioteca BAL. El resultado principal es el archivo `libbal.a`
- `make clean`: Borra todos los archivos intermedios generados, necesarios para construir BAL.
- `make purge`: Igual que `make clean`, pero también borra la biblioteca BAL generada.
- `make tags`: Genera el archivo de etiquetas mediante una herramienta ctags.
- `make doc`: Genera esta documentación a partir de los archivos fuente (se recomienda ejecutarlo dos veces). **NOTA**: Se recomienda hacer un `make clean` antes de generar la documentación. Si se genera la documentación con los fuentes generados para el parser, la misma contendrá una gran cantidad de referencia a código autogenerado (muy feo) y no propiamente documentado que hacen que la documentación no se vea bien.
- `make cleandoc`: Borra la documentación generada.
- `make install`: Instala BAL como una biblioteca más del sistema (para sistemas UNIX). Este comando debe ser ejecutado con permisos de administrador (root).
- `make uninstall`: Deshace lo hecho con el comando `make install`. También debe ser ejecutado como el usuario root.

Es importante aclarar que la compilación de BAL no produce ningún código directamente ejecutable. Solo produce un archivo (`libbal.a`) ya compilado, pronto para ser enlazado (*linkeado*) con la aplicación que necesite utilizar las prestaciones de BAL. En la siguiente sección se da detalle de cómo utilizar BAL desde otros programas (ver Cómo usar BAL desde otro programa).

1.3.2.1. Secuencia de instalación Si se tienen todas las dependencias necesarias y todo va bien, la siguiente secuencia de comandos compila, genera la documentación e instala BAL en el sistema:

```
make doc

make

sudo make install
```

El comando `sudo` provoca que el comando a continuación se ejecute con permisos de root. Puede llegar a pedir una contraseña.

1.3.2.2. Modos de compilación Los archivos `makefile` de BAL (el propio Makefile y los archivos *.mak) definen una variable llamada `CFLAGS`, la cual indica algunos argumentos extra a utilizar a la hora de invocar al compilador de C. Los archivos contienen dos definiciones para esta variable, una específica para depuración de BAL y otra para generar código optimizado. Revise los archivos `makefile` y utilice la definición que más se ajuste a sus necesidades.

1.3.3. Cómo usar BAL desde otro programa

En esta sección se muestra cómo escribir código que utilice las implementaciones de BAL y luego cómo compilarlo.

1.3.3.1. Cómo referenciar a BAL desde código externo Escribir código que utilice BAL es muy sencillo (la parte, un poco, más compleja es a la hora de compilar y es detalla en la siguiente sección). Simplemente se debe agregar la referencia a BAL

```
#include <bal.h>
```

y luego utilizar las funciones descritas en este documento. Es importante recordar que el punto de entrada a BAL desde un programa externo es siempre y únicamente el archivo de cabecera `bal.h`. Toda la funcionalidad de BAL es expuesta mediante este archivo. Por lo tanto, es suficiente con incluir este archivo en el programa "cliente".

1.3.3.2. Cómo compilar un programa que usa BAL La mejor forma de ver cómo utilizar BAL es viendo cómo se hizo un programa que la usa. Junto con BAL se distribuye un juego de programas de prueba que hacen justamente esto. Utilizan BAL para probarla. De todos modos, a continuación se describe cómo sería una llamada al compilador de C que enlace BAL con el programa cliente.

Supongamos, para facilitar el ejemplo que nuestro programa cliente consta de un solo archivo `miprog.c` y que este archivo lo precompilamos de la manera clásica produciendo el archivo `miprog.o`. Esto lo obtenemos con un comando similar al siguiente:

```
gcc -c miprog.c -o miprog.o
```

Ahora lo que resta es la etapa de enlazado. Supongamos que tenemos BAL (particularmente, el archivo `libbal.a` y los archivos de cabecera de BAL) compilada y pronta en el directorio `/bal` (aunque podría ser cualquier directorio), pero no instalamos BAL con el comando `make install`. El comando para enlazar es el siguiente:

```
gcc -I/bal miprog.o -o miprog -L/bal -lbal `pkg-config -cflags -libs
glib-2.0` -lfl -lm
```

Esto generará un programa ejecutable llamado `miprog`. Expliquemos ahora la composición de esa llamada a `gcc`:

- El argumento `-I/bal` le indica al compilador que debe agregar el directorio `/bal` a su lista de directorios de búsqueda para la resolución de las instrucciones de precompilador tipo `#include`.
- El argumento `-L/bal` le indica al compilador que debe agregar el directorio `/bal` a su lista de directorios de búsqueda para la resolución de los comandos `-l`.
- El argumento `-lbal` le indica al compilador que debe enlazar la biblioteca `libbal.a` al programa resultante.
- La ejecución embebida `` `pkg-config -cflags -libs glib-2.0` `` genera las banderas necesarias para enlazar `glib 2.0`, necesario por BAL. Se puede probar ejecutar solo esto para

ver el resultado que produce. De no contar con la herramienta `pkg-config`, se pueden agregar las referencias manualmente. Las mismas serían similares a: `-I/usr/include/glib-2.0 -I/usr/lib/glib-2.0/include -lglib-2.0`

- Los argumentos `-lfl` y `-lm` enlazan las bibliotecas `libfl.a` y `libm.a` respectivamente, la primera necesaria por el uso de `glib` y la segunda necesaria por BAL.

Si instalamos BAL con el comando `make install`, podemos referenciar a BAL en el código cliente con la instrucción

```
#include <BAL/bal.h>
```

y el comando de compilación sería el siguiente:

```
gcc miprog.o -o miprog -lbal `pkg-config -libs glib-2.0` -lfl -lm
```

1.4. El juego de pruebas de BAL

BAL contiene un conjunto de programas que sirven, no solo para verificar la correctitud de BAL, sino también como ejemplo de su uso.

Estos tests se encuentran en el directorio `bal_test`. Cada prueba es un programa C diferente y básicamente cada prueba se limita a probar una funcionalidad particular de BAL. El nombre del archivo `.c` da idea de lo que se pretende probar.

El juego de pruebas tiene su propio `makefile` para la fácil compilación. Pero más aun, el directorio de pruebas contiene un script llamado `run_tests.sh` para la ejecución fácil de las pruebas. Este script, primero ejecuta `make` para cerciorar que los tests han sido compilados. Luego busca aquellos tests que fueron compilados correctamente y los ejecuta. Para cada test ejecutado, redirecciona su salida estándar a un archivo de extensión `.out` y su salida de errores a una archivo de extensión `.error`. El prefijo de ambos archivos es el nombre del ejecutable del test.

NOTA: Para que este mecanismo funcione es importante que todos los ejecutables de las pruebas tengan sufijo `_test`.

Al finalizar, el script muestra en pantalla una lista de aquellos tests que generaron algún tipo de salida en la salida de errores (*standard error*). De este modo es fácil localizar cuáles tests fallaron.

1.5. Referencias

La siguiente es una lista de documentos que fueron utilizados para la implementación de BAL:

- G. W. Stewart. *Building an Old-Fashioned Sparse Solver*. Agosto 2003.

  Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland

  http://hdl.handle.net/1903/1312

1.6. Licencia

- Está permitido cualquier tipo de uso de este material, ya sea personal, comercial o cualquier otro.
- Está permitido mentir acerca del origen de este trabajo, de todos modos nadie se va a enterar. Pero esta actitud pesará eternamente en la conciencia de la persona que lo haga y en la conciencia de sus hijos, y los hijos de sus hijos, etc.
- **Este trabajo (software y documentación) se entrega "TAL CUAL" y se renuncia a toda responsabilidad por las garantías, implícitas o explícitas, incluidas, sin limitación. UTILÍCELO BAJO SU PROPIA RESPONSABILIDAD.**

2. Listado de Tareas Pendientes

page Biblioteca the Álgebra Lineal (BAL)

Global reordenar_packcol Implementar un algoritmo de reordenamiento

3. Notas de las versiones de BAL

3.1. Notas a la versión 1.0.0 de BAL

Al ser esta la primer versión de BAL, hubo un trabajo fuerte en armar la estructura básica de la biblioteca. Para ello se tuvo que estudiar la forma de armar bibliotecas en ambientes UNIX.

Por otra parte, se hizo hincapié en la documentación. Por ello se utilizó la herramienta Doxygen para generar documentación a partir del código. Esto facilitó mucho la tarea. Espero que se encuentre útil la documentación generada y que las próximas versiones también utilicen este mecanismo para documentar sus extensiones.

Al partir de cero, fue necesario implementar algunas funcionalidades básicas para comenzar con la biblioteca, por ejemplo el parser de matrices en formato matlab, las funciones de impresión, etc.

Por otra parte, junto con BAL se entregan en esta versión un conjunto de programas que sirven no solo como pruebas de correctitud de las funcionalidades de BAL, sino también como ejemplos de cómo utilizar BAL y las salidas que produce. Éstos se pueden encontrar en el directorio `bal_test`, el cual también viene con un `Makefile` para la fácil compilación.

Sin embargo, la implementación más importante de esta versión es sin duda el algoritmo de Cholesky. Por tanto podemos decir que la función más importante de esta versión es cholesky_solver(), la cual engloba los algoritmos más importantes y complejos implementados.

La gran mayoría de los algoritmos implementados en esta versión (sobre todo los más complejos) están basados en las descripciones del paper *Building an Old-Fashioned Sparse Solver*, por G. W. Stewart (ver las referencias). Aunque el solver es *old-fashioned* esto no quiere decir que sea un mal solver. Citando al propio Stewart, *No es un juguete. Cerca de 1975, grandes investigadores estaban trabajando duro para perfeccionar un solver como el nuestro.* Esto es importante señalarlo. Porque si bien no es una implementación con lo último del estado del arte, es una excelente muestra para el aprendizaje de solvers profesionales (de hecho este solver lo fue en algún momento!).

Personalmente recomiendo mucho la lectura del paper de Stewart. Está muy bien detallado. Sin embargo, algunos de los pseudo-códigos del paper tienen errores (muy sutiles). Pero dichos errores fueron corregidos durante la implementación de BAL. Por tanto se recomienda comparar el pseudo-código de Stewart con la implementación en BAL para cada uno de los algoritmos. Creo que esto ayudará a entender mejor los algoritmos, que no son nada triviales.

Gastón Simone - Mayo 2008

4. Documentación de Directorio

4.1. Referencia del Directorio cholesky/

Directorio con implementación de la factorización de Cholesky.

Archivos

- archivo cholesky.c

  Implementación de la factorización de Cholesky para matrices dispersas.

- archivo cholesky.h

  Archivo de cabecera para la factorización de Cholesky.

- archivo reordenamiento.c

  Implementación de algoritmos de reordenamiento.

- archivo reordenamiento.h

  Archivo de cabecera para algoritmos de reordenamiento.

4.1.1. Descripción detallada

Directorio con implementación de la factorización de Cholesky.

4.2. Referencia del Directorio parser/

Directorio con implementación del parser de matrices en formato matlab.

Archivos

- archivo matriz_parser.y

  Archivo de definición del parser generado por bison.

- archivo matriz_scanner.lex

  Archivo de definición del analizador lexicográfico generado por flex.

4.2.1. Descripción detallada

Directorio con implementación del parser de matrices en formato matlab.

Este directorio contiene todos los archivos que implementan el parser de matrices en formato matlab, así como también un archivo `makefile` para la fácil generación del parser.

4.3. Referencia del Directorio sparse/

Directorio con archivos de estructuras de matrices dispersas.

Archivos

- archivo sp_cds.c

  Archivo de implementación para matriz dispersa, formato simple.

- archivo sp_cds.h

  Archivo de cabecera para matriz dispersa, formato CDS (comprimido por diagonal).

- archivo sp_coord.c

  Archivo de implementación para matriz dispersa, formato simple.

- archivo sp_coord.h

  Archivo de cabecera para matriz dispersa, formato simple.

- archivo sp_packcol.c

  Archivo de implementación para formato de matriz dispersa empaquetado por columnas.

- archivo sp_packcol.h

  Archivo de cabecera para matriz dispersa, formato empaquetado por columna.

4.3.1. Descripción detallada

Directorio con archivos de estructuras de matrices dispersas.

Este directorio contiene las definiciones de las estructuras de datos para matrices dispersas, así como también la implementación de las funciones de conversión entre estas representaciones.

5. Documentación de las estructuras de datos

5.1. Referencia de la Estructura sp_cds

Estructura de matriz dispersa CDS.

`#include <sp_cds.h>`

Campos de datos

- unsigned int nrow

  Cantidad de filas.

- unsigned int ncol

  Cantidad de columnas.

- unsigned int ndiag

  Cantidad de diagonales con al menos una entrada no cero.

- unsigned int maxdiaglength

  El largo de la diagonal más larga.

- int * dx

  Arreglo de índices de diagonal.

- double ** val

  Matriz de valores. Cada fila es una diagonal.

5.1.1. Descripción detallada

Estructura de matriz dispersa CDS.

Almacena una matriz dispersa en memoria utilizando el formato CDS.

Los valores se almacenan de la siguiente forma. A cada diagonal se le asigna un índice que la identifica. La diagonal mayor es la diagonal 0. Las diagonales superiores a la mayor tienen índices positivos que se van incrementando a medida que nos alejamos de la diagonal mayor. Las diagonales inferiores a la mayor tienen índices negativos que se van decrementando a medida que nos alejamos de la diagonal mayor. Por ejemplo, las primeras subdiagonales superior e inferior les corresponden los índices 1 y -1 respectivamente.

De este modo, el elemento A_{ij} de la matriz está ubicado en la diagonal $j - i$.

Solo se almacenan las diagonales con al menos un valor no cero. Cada fila en `val` es una diagonal de `A`. El arreglo `dx` indica a qué diagonal corresponde cada una de las filas de `val`. Es decir, si queremos obtener valores de la diagonal `i`, éstos están ubicados en la fila `j` de `val` tal que `i = dx[j]`.

Por otra parte, la ubicación de los valores de la diagonal en una fila de `val` está dada por el índice de fila en la matriz. Es decir A_{ij} corresponde a la diagonal $j - i$ y su valor está en la columna `i` de `val`. Esto provoca que las diagonales inferiores estén alineadas a la derecha y las superiores alineadas a la izquierda.

Por ejemplo, sea la matriz

$$A = \begin{pmatrix} 1 & 2 & 0 & 0 & 0 \\ 3 & 4 & 5 & 0 & 0 \\ 0 & 6 & 7 & 8 & 0 \\ 0 & 0 & 9 & 10 & 11 \\ 0 & 0 & 0 & 12 & 13 \end{pmatrix}$$

La estructura CDS para `A` es la siguiente. La diagonal de menor índice es la -1 y la de mayor índice es la +1.

```
dx = [-1 0 1]
val[0,0:4] = [0 3 6 9 12]
val[1,0:4] = [1 4 7 10 13]
val[2,0:4] = [2 5 8 11 0]
```

Resumiendo, podemos decir que el valor A_{ij} se encuentra en `val[k,i]`, con `(dx[k] = j - i)`. Recíprocamente podemos decir que el elemento en `val[k,i]` corresponde al elemento $A_{i,(dx[k]+i)}$.

Nota:

Observar que esta estructura guarda toda diagonal con al menos un elemento no cero de forma completa. Es decir, los ceros en una diagonal "no vacía" son guardados.

Definición en la línea 68 del archivo sp_cds.h.

La documentación para esta estructura fue generada a partir del siguiente fichero:

- sparse/sp_cds.h

5.2. Referencia de la Estructura sp_coord

Estructura de matriz dispersa simple por coordenadas.

`#include <sp_coord.h>`

Campos de datos

- unsigned int nrow
  Cantidad de filas.
- unsigned int ncol
  Cantidad de columnas.
- unsigned int nnz
  Cantidad de elementos no cero.
- unsigned int * rx
  Arreglo de indices de fila.
- unsigned int * cx
  Arreglo de indices de columna.
- double * val
  Arreglos de valores.

5.2.1. Descripción detallada

Estructura de matriz dispersa simple por coordenadas.

Almacena una matriz dispersa en memoria utilizando el formato simple por coordenadas. Por más información acerca de este formato, vea la sección 5.1 del paper de Stewart (ver las referencias).

Definición en la línea 21 del archivo sp_coord.h.

La documentación para esta estructura fue generada a partir del siguiente fichero:

- sparse/sp_coord.h

5.3. Referencia de la Estructura sp_packcol

Estructura de matriz dispersa empaquetada por columna.

`#include <sp_packcol.h>`

Campos de datos

- unsigned int nrow
  Cantidad de filas de la matriz.
- unsigned int ncol

Cantidad de columnas de la matriz.

- unsigned int nnz

  Cantidad de elementos no cero.

- unsigned int * colp

  Arreglo de punteros a los inicios de las columnas.

- unsigned int * rx

  Arreglo de indices de fila.

- double * val

  Arreglo de valores por debajo de la diagonal.

5.3.1. Descripción detallada

Estructura de matriz dispersa empaquetada por columna.

Almacena una matriz dispersa en memoria utilizando el formato simple por coordenadas. Por más información acerca de este formato, vea la sección 5.1 del paper de Stewart (ver las referencias).

Definición en la línea 22 del archivo sp_packcol.h.

La documentación para esta estructura fue generada a partir del siguiente fichero:

- sparse/sp_packcol.h

6. Documentación de archivos

6.1. Referencia del Archivo bal.c

Biblioteca de Algebra Lineal, archivo principal.

```
#include <glib.h>
#include <stdio.h>
#include "bal.h"
#include "oper.h"
#include "sparse/sp_coord.h"
#include "sparse/sp_packcol.h"
#include "sparse/sp_cds.h"
#include "cholesky/cholesky.h"
#include "cholesky/reordenamiento.h"
```

Funciones

- int yyparse (const char *archivo, double ***matriz, int *n, int *m)

  Parser de matrices en formato matlab generado con bison y flex.

- int bal_cargar_matriz (const char *archivo, double ***matriz, int *n, int *m)
  Carga una matriz en memoria desde un archivo.
- void bal_imprimir_matriz (FILE *fp, double **matriz, int n, int m)
  Imprime la matriz en el archivo `fp`.
- sp_coord * bal_mat2coord (int n, int m, double **mat)
  Genera la matriz dispersa equivalente a la matriz completa mat (n x m).
- sp_packcol * bal_coord2packcol (sp_coord *mat)
  Genera una instancia de la matriz mat en formato empaquetado por columna.
- sp_packcol * bal_coord2packcol_symmetric (sp_coord *mat)
  Genera una instancia de la matriz mat en formato empaquetado por columna para matrices simétricas.
- void bal_imprimir_coord (FILE *fp, sp_coord *mat)
  Imprime la matriz guardada en formato simple por coordenadas en fp.
- void bal_imprimir_packcol (FILE *fp, sp_packcol *mat)
  Imprime la matriz guardada en formato empaquetado por columna en fp.
- void bal_mult_vec_packcol_symmetric (sp_packcol *A, double *x, double *y)
  Multiplica una matriz simétrica empaquetada por columna por un vector.
- void bal_mult_vec_packcol (sp_packcol *A, double *x, double *y)
  Multiplica una matriz empaquetada por columna por un vector.
- int bal_row_traversal_packcol (sp_packcol *A, int *i, int *j, int *posij)
  Implementa un mecanismo eficiente para recorrer por filas una matriz dispersa empaquetada por columnas.
- sp_coord * bal_load_coord (FILE *fp)
  Escribe en `fp` *la matriz* `A` *en un formato entendible por load_coord().*
- void bal_save_coord (FILE *fp, sp_coord *A)
  Escribe en `fp` *la matriz* `A` *en un formato entendible por load_coord().*
- void bal_save_packcol_symmetric (FILE *fp, sp_packcol *A)
  Imprime la matriz simétrica empaquetada por columna en formato matlab en el archivo fp.
- void bal_save_packcol (FILE *fp, sp_packcol *A)
  Imprime la matriz `A` *en formato matlab en el archivo* `fp`.
- int * bal_elimination_tree (sp_packcol *A, int *nnz)
  Calcula el árbol de eliminación de la matriz simétrica A.
- sp_packcol * bal_symbolic_factorization (sp_packcol *A)
  Factorización simbólica de la matriz A.
- void bal_numerical_factorization (sp_packcol *A, sp_packcol *L)

Sobreescribe `L` *con la factorización de Cholesky de A.*

- void bal_cholesky_Lsolver (sp_packcol *L, double *b)

  Sobrescribe `b` *con la solución del sistema* $Ly = b$.

- void bal_cholesky_LTsolver (sp_packcol *L, double *b)

  Sobreescribe `b` *con la solución de* $L^T x = b$.

- void bal_cholesky_solver (sp_packcol *A, double *b)

  Resuelve un sistema lineal mediante el algoritmo de Cholesky optimizado para matrices dispersas.

- sp_cds * bal_coord2cds (sp_coord *mat)

  Genera la matriz en formato CDS equivalente a la matriz mat en formato simple.

- void bal_imprimir_cds (FILE *fp, sp_cds *mat)

  Imprime la matriz guardada en formato simple por coordenadas en fp.

- void bal_mult_vec_cds (sp_cds *A, double *x, double *y)

  Multiplica una matriz en formato CDS por un vector.

- sp_cds * bal_mult_mat_cds (sp_cds *A, sp_cds *B)

  Multiplica dos matrices en formato CDS.

- void bal_save_cds (FILE *fp, sp_cds *A)

  Imprime la matriz `A` *en formato matlab en el archivo* `fp`.

- sp_packcol * bal_permutar_packcol (unsigned int *p, sp_packcol *A)

  Aplica una permutación de columnas sobre la matriz `A`.

- void bal_free_coord (sp_coord *A)

  Libera la memoria reservada por la matriz `A`.

- void bal_free_packcol (sp_packcol *A)

  Libera la memoria reservada por la matriz `A`.

- void bal_free_cds (sp_cds *A)

  Libera la memoria reservada por la matriz `A`.

6.1.1. Descripción detallada

Biblioteca de Algebra Lineal, archivo principal.

Este archivo implementa las funciones disponibles mediante BAL.

Definición en el archivo bal.c.

6.1.2. Documentación de las funciones

6.1.2.1. int bal_cargar_matriz (const char * *archivo*, double *** *matriz*, int * *n*, int * *m*)

Carga una matriz en memoria desde un archivo.

Parámetros:

archivo ENTRADA: El camino al archivo en el que esta definida la matriz

matriz SALIDA: Puntero a la estructura de memoria donde fue alocada la matriz

n SALIDA: Cantidad de filas

m SALIDA: Cantidad de columnas

La matriz es cargada en memoria de forma "convencional". El formato esperado es el mismo formato usado por matlab para definir matrices. El parser fue implementado utilizando las herramientas bison y flex de forma combinada. La función retorna 0 si todo funciono correctamente y otro valor en caso contrario.

Definición en la línea 43 del archivo bal.c.

Hace referencia a yyparse().

```
44 {
45     if (yyparse(archivo, matriz, n, m) != 0)
46         return -1;
47
48     return 0;
49 }
```

6.1.2.2. void bal_cholesky_Lsolver (sp_packcol * *L*, double * *b*)

Sobrescribe b con la solución del sistema $Ly = b$.

Por más información vea cholesky_Lsolver().

Definición en la línea 222 del archivo bal.c.

Hace referencia a cholesky_Lsolver().

```
223 {
224     cholesky_Lsolver(L, b);
225 }
```

6.1.2.3. void bal_cholesky_LTsolver (sp_packcol * *L*, double * *b*)

Sobreescribe b con la solución de $L^T x = b$.

Por más información vea cholesky_LTsolver().

Definición en la línea 232 del archivo bal.c.

Hace referencia a cholesky_LTsolver().

```
233 {
234     cholesky_LTsolver(L, b);
235 }
```

6.1.2.4. void bal_cholesky_solver (sp_packcol * *A*, double * *b*)

Resuelve un sistema lineal mediante el algoritmo de Cholesky optimizado para matrices dispersas.

Por más información vea cholesky_solver().

Definición en la línea 242 del archivo bal.c.

Hace referencia a cholesky_solver().

```
243 {
244     cholesky_solver(A, b);
245 }
```

6.1.2.5. sp_cds * bal_coord2cds (sp_coord * *mat*)

Genera la matriz en formato CDS equivalente a la matriz mat en formato simple.

Por más información vea coord2cds().

Definición en la línea 252 del archivo bal.c.

Hace referencia a coord2cds().

```
253 {
254     return coord2cds(mat);
255 }
```

6.1.2.6. sp_packcol * bal_coord2packcol (sp_coord * *mat*)

Genera una instancia de la matriz mat en formato empaquetado por columna.

Por más información vea coord2packcol().

Definición en la línea 82 del archivo bal.c.

Hace referencia a coord2packcol().

```
83 {
84     return coord2packcol(mat);
85 }
```

6.1.2.7. sp_packcol * bal_coord2packcol_symmetric (sp_coord * *mat*)

Genera una instancia de la matriz mat en formato empaquetado por columna para matrices simétricas.

Por más información vea coord2packcol_symmetric().

Definición en la línea 92 del archivo bal.c.

Hace referencia a coord2packcol_symmetric().

```
93 {
94     return coord2packcol_symmetric(mat);
95 }
```

6.1.2.8. int * bal_elimination_tree (sp_packcol * *A*, int * *nnz*)

Calcula el árbol de eliminación de la matriz simétrica A.

Por más información vea elimination_tree().

Definición en la línea 192 del archivo bal.c.

Hace referencia a elimination_tree().

```
193 {
194     return elimination_tree(A, nnz);
195 }
```

6.1.2.9. void bal_free_cds (sp_cds * *A*)

Libera la memoria reservada por la matriz `A`.

Por más información ver free_cds()

Definición en la línea 332 del archivo bal.c.

Hace referencia a free_cds().

```
333 {
334     free_cds(A);
335 }
```

6.1.2.10. void bal_free_coord (sp_coord * *A*)

Libera la memoria reservada por la matriz `A`.

Por más información ver free_coord()

Definición en la línea 312 del archivo bal.c.

Hace referencia a free_coord().

```
313 {
314     free_coord(A);
315 }
```

6.1.2.11. void bal_free_packcol (sp_packcol * *A*)

Libera la memoria reservada por la matriz `A`.

Por más información ver free_packcol()

Definición en la línea 322 del archivo bal.c.

Hace referencia a free_packcol().

```
323 {
324     free_packcol(A);
325 }
```

6.1.2.12. void bal_imprimir_cds (FILE * *fp*, sp_cds * *mat*)

Imprime la matriz guardada en formato simple por coordenadas en fp.

Por más información vea sp_imprimir_cds().

Definición en la línea 262 del archivo bal.c.

Hace referencia a sp_imprimir_cds().

```
263 {
264     sp_imprimir_cds(fp, mat);
265 }
```

6.1.2.13. void bal_imprimir_coord (FILE * *fp*, sp_coord * *mat*)

Imprime la matriz guardada en formato simple por coordenadas en fp.

Por más información vea sp_imprimir_coord().

Definición en la línea 102 del archivo bal.c.

Hace referencia a sp_imprimir_coord().

```
103 {
104     sp_imprimir_coord(fp, mat);
105 }
```

6.1.2.14. void bal_imprimir_packcol (FILE * *fp*, sp_packcol * *mat*)

Imprime la matriz guardada en formato empaquetado por columna en fp.

Por más información vea sp_imprimir_packcol().

Definición en la línea 112 del archivo bal.c.

Hace referencia a sp_imprimir_packcol().

```
113 {
114     sp_imprimir_packcol(fp, mat);
115 }
```

6.1.2.15. sp_coord * bal_load_coord (FILE * *fp*)

Escribe en `fp` la matriz `A` en un formato entendible por load_coord().

Por más información vea load_coord().

Definición en la línea 152 del archivo bal.c.

Hace referencia a load_coord().

```
153 {
154     return load_coord(fp);
155 }
```

6.1.2.16. sp_coord * bal_mat2coord (int *n*, int *m*, double ** *mat*)

Genera la matriz dispersa equivalente a la matriz completa mat (n x m).

Por más información, vea mat2coord().

Definición en la línea 72 del archivo bal.c.

Hace referencia a mat2coord().

```
73 {
74     return mat2coord(n, m, mat);
75 }
```

6.1.2.17. sp_cds * bal_mult_mat_cds (sp_cds * *A*, sp_cds * *B*)

Multiplica dos matrices en formato CDS.

Por más información ver mult_mat_cds().

Definición en la línea 282 del archivo bal.c.

Hace referencia a mult_mat_cds().

```
283 {
284     return mult_mat_cds(A, B);
285 }
```

6.1.2.18. void bal_mult_vec_cds (sp_cds * *A*, double * *x*, double * *y*)

Multiplica una matriz en formato CDS por un vector.

Por más información ver mult_vec_cds().

Definición en la línea 272 del archivo bal.c.

Hace referencia a mult_vec_cds().

```
273 {
274     mult_vec_cds(A, x, y);
275 }
```

6.1.2.19. void bal_mult_vec_packcol (sp_packcol * *A*, double * *x*, double * *y*)

Multiplica una matriz empaquetada por columna por un vector.

Por más información vea mult_vec_packcol().

Definición en la línea 132 del archivo bal.c.

Hace referencia a mult_vec_packcol().

```
133 {
134     mult_vec_packcol(A, x, y);
135 }
```

6.1.2.20. void bal_mult_vec_packcol_symmetric (sp_packcol * *A*, double * *x*, double * *y*)

Multiplica una matriz simétrica empaquetada por columna por un vector.

Por más información vea mult_vec_packcol_symmetric().

Definición en la línea 122 del archivo bal.c.

Hace referencia a mult_vec_packcol_symmetric().

```
123 {
124     mult_vec_packcol_symmetric(A, x, y);
125 }
```

6.1.2.21. void bal_numerical_factorization (sp_packcol * *A*, sp_packcol * *L*)

Sobreescribe `L` con la factorización de Cholesky de A.

Por más información vea numerical_factorization().

Definición en la línea 212 del archivo bal.c.

Hace referencia a numerical_factorization().

```
213 {
214     numerical_factorization(A, L);
215 }
```

6.1.2.22. sp_packcol * bal_permutar_packcol (unsigned int * *p*, sp_packcol * *A*)

Aplica una permutación de columnas sobre la matriz `A`.

Por más información vea permutar_packcol().

Definición en la línea 302 del archivo bal.c.

Hace referencia a permutar_packcol().

```
303 {
304     return permutar_packcol(p, A);
305 }
```

6.1.2.23. int bal_row_traversal_packcol (sp_packcol * *A*, int * *i*, int * *j*, int * *posij*)

Implementa un mecanismo eficiente para recorrer por filas una matriz dispersa empaquetada por columnas.

Por más información vea row_traversal_packcol().

Definición en la línea 142 del archivo bal.c.

Hace referencia a row_traversal_packcol().

```
143 {
144     return row_traversal_packcol(A, i, j, posij);
145 }
```

6.1.2.24. void bal_save_cds (FILE * *fp*, sp_cds * *A*)

Imprime la matriz `A` en formato matlab en el archivo `fp`.

Por más información vea save_cds().

Definición en la línea 292 del archivo bal.c.

Hace referencia a save_cds().

```
293 {
294     save_cds(fp, A);
295 }
```

6.1.2.25. void bal_save_coord (FILE * *fp*, sp_coord * *A*)

Escribe en `fp` la matriz `A` en un formato entendible por load_coord().

Por más información vea save_coord().

Definición en la línea 162 del archivo bal.c.

Hace referencia a save_coord().

```
163 {
164     save_coord(fp, A);
165 }
```

6.1.2.26. void bal_save_packcol (FILE * *fp*, sp_packcol * *A*)

Imprime la matriz `A` en formato matlab en el archivo `fp`.

Por más información vea save_packcol().

Definición en la línea 182 del archivo bal.c.

Hace referencia a save_packcol().

```
183 {
184     save_packcol(fp, A);
185 }
```

6.1.2.27. void bal_save_packcol_symmetric (FILE * *fp*, sp_packcol * *A*)

Imprime la matriz simétrica empaquetada por columna en formato matlab en el archivo fp.

Por más información vea save_packcol_symmetric().

Definición en la línea 172 del archivo bal.c.

Hace referencia a save_packcol_symmetric().

```
173 {
174     save_packcol_symmetric(fp, A);
175 }
```

6.1.2.28. sp_packcol * bal_symbolic_factorization (sp_packcol * *A*)

Factorización simbólica de la matriz A.

Por más información vea symbolic_factorization().

Definición en la línea 202 del archivo bal.c.

Hace referencia a symbolic_factorization().

```
203 {
204     return symbolic_factorization(A);
205 }
```

6.1.2.29. int yyparse (const char * *archivo*, double *** *matriz*, int * *n*, int * *m*)

Parser de matrices en formato matlab generado con bison y flex.

Parámetros:

archivo ENTRADA: Camino al archivo donde esta definida la matriz

matriz SALIDA: Matriz leída como lista de filas. Cada fila es una lista de valores.

n SALIDA: Cantidad de filas de la matriz leída.

m SALIDA: Cantidad de columnas de la matriz leída.

NOTA: Esta función es de uso interno de BAL. No debería ser necesario su uso externo. Vea la función bal_cargar_matriz.

Referenciado por bal_cargar_matriz().

6.2. Referencia del Archivo bal.h

Biblioteca de Algebra Lineal, archivo de cabecera.

```
#include <stdio.h>
#include "sparse/sp_coord.h"
#include "sparse/sp_packcol.h"
#include "sparse/sp_cds.h"
```

Funciones

- int bal_cargar_matriz (const char *archivo, double ***matriz, int *n, int *m)

  Carga una matriz en memoria desde un archivo.

- void bal_imprimir_matriz (FILE *fp, double **matriz, int n, int m)

  Imprime la matriz en el archivo `fp`.

- sp_coord * bal_mat2coord (int n, int m, double **mat)

  Genera la matriz dispersa equivalente a la matriz completa mat (n x m).

- sp_packcol * bal_coord2packcol (sp_coord *mat)

  Genera una instancia de la matriz mat en formato empaquetado por columna.

- sp_packcol * bal_coord2packcol_symmetric (sp_coord *mat)

  Genera una instancia de la matriz mat en formato empaquetado por columna para matrices simétricas.

- sp_cds * bal_coord2cds (sp_coord *mat)

  Genera la matriz en formato CDS equivalente a la matriz mat en formato simple.

- void bal_imprimir_coord (FILE *fp, sp_coord *mat)

  Imprime la matriz guardada en formato simple por coordenadas en fp.

- void bal_imprimir_packcol (FILE *fp, sp_packcol *mat)

  Imprime la matriz guardada en formato empaquetado por columna en fp.

- void bal_imprimir_cds (FILE *fp, sp_cds *mat)

  Imprime la matriz guardada en formato simple por coordenadas en fp.

- void bal_mult_vec_packcol (sp_packcol *A, double *x, double *y)

  Multiplica una matriz empaquetada por columna por un vector.

- void bal_mult_vec_packcol_symmetric (sp_packcol *A, double *x, double *y)

  Multiplica una matriz simétrica empaquetada por columna por un vector.

- int bal_row_traversal_packcol (sp_packcol *A, int *i, int *j, int *posij)

  Implementa un mecanismo eficiente para recorrer por filas una matriz dispersa empaquetada por columnas.

- sp_coord * bal_load_coord (FILE *fp)

  Escribe en `fp` *la matriz* `A` *en un formato entendible por load_coord().*

- void bal_save_coord (FILE *fp, sp_coord *A)

  Escribe en `fp` *la matriz* `A` *en un formato entendible por load_coord().*

- void bal_save_packcol_symmetric (FILE *fp, sp_packcol *A)

  Imprime la matriz simétrica empaquetada por columna en formato matlab en el archivo fp.

- void bal_save_packcol (FILE *fp, sp_packcol *A)

  Imprime la matriz `A` *en formato matlab en el archivo* `fp`*.*

- void bal_save_cds (FILE *fp, sp_cds *A)

  Imprime la matriz `A` *en formato matlab en el archivo* `fp`*.*

- int * bal_elimination_tree (sp_packcol *A, int *nnz)

  Calcula el árbol de eliminación de la matriz simétrica A.

- sp_packcol * bal_symbolic_factorization (sp_packcol *A)

  Factorización simbólica de la matriz A.

- void bal_numerical_factorization (sp_packcol *A, sp_packcol *L)

  Sobreescribe `L` *con la factorización de Cholesky de A.*

- void bal_cholesky_Lsolver (sp_packcol *L, double *b)

Sobrescribe b *con la solución del sistema* $Ly = b$.

- void bal_cholesky_LTsolver (sp_packcol *L, double *b)
  Sobreescribe b *con la solución de* $L^T x = b$.
- void bal_cholesky_solver (sp_packcol *A, double *b)
  Resuelve un sistema lineal mediante el algoritmo de Cholesky optimizado para matrices dispersas.
- void bal_mult_vec_cds (sp_cds *A, double *x, double *y)
  Multiplica una matriz en formato CDS por un vector.
- sp_cds * bal_mult_mat_cds (sp_cds *A, sp_cds *B)
  Multiplica dos matrices en formato CDS.
- sp_packcol * bal_permutar_packcol (unsigned int *p, sp_packcol *A)
  Aplica una permutación de columnas sobre la matriz A.
- void bal_free_coord (sp_coord *A)
  Libera la memoria reservada por la matriz A.
- void bal_free_packcol (sp_packcol *A)
  Libera la memoria reservada por la matriz A.
- void bal_free_cds (sp_cds *A)
  Libera la memoria reservada por la matriz A.

6.2.1. Descripción detallada

Biblioteca de Algebra Lineal, archivo de cabecera.

Este archivo define las funciones disponibles mediante BAL.

Definición en el archivo bal.h.

6.2.2. Documentación de las funciones

6.2.2.1. int bal_cargar_matriz (const char * *archivo*, double *** *matriz*, int * *n*, int * *m*)

Carga una matriz en memoria desde un archivo.

Parámetros:

archivo ENTRADA: El camino al archivo en el que esta definida la matriz

matriz SALIDA: Puntero a la estructura de memoria donde fue alocada la matriz

n SALIDA: Cantidad de filas

m SALIDA: Cantidad de columnas

La matriz es cargada en memoria de forma "convencional". El formato esperado es el mismo formato usado por matlab para definir matrices. El parser fue implementado utilizando las herramientas bison y flex de forma combinada. La función retorna 0 si todo funciono correctamente y otro valor en caso contrario.

Definición en la línea 43 del archivo bal.c.

Hace referencia a yyparse().

```
44 {
45     if (yyparse(archivo, matriz, n, m) != 0)
46         return -1;
47 
48     return 0;
49 }
```

6.2.2.2. void bal_cholesky_Lsolver (sp_packcol * *L*, double * *b*)

Sobrescribe b con la solución del sistema $Ly = b$.

Por más información vea cholesky_Lsolver().

Definición en la línea 222 del archivo bal.c.

Hace referencia a cholesky_Lsolver().

```
223 {
224     cholesky_Lsolver(L, b);
225 }
```

6.2.2.3. void bal_cholesky_LTsolver (sp_packcol * *L*, double * *b*)

Sobreescribe b con la solución de $L^T x = b$.

Por más información vea cholesky_LTsolver().

Definición en la línea 232 del archivo bal.c.

Hace referencia a cholesky_LTsolver().

```
233 {
234     cholesky_LTsolver(L, b);
235 }
```

6.2.2.4. void bal_cholesky_solver (sp_packcol * *A*, double * *b*)

Resuelve un sistema lineal mediante el algoritmo de Cholesky optimizado para matrices dispersas.

Por más información vea cholesky_solver().

Definición en la línea 242 del archivo bal.c.

Hace referencia a cholesky_solver().

```
243 {
244     cholesky_solver(A, b);
245 }
```

6.2.2.5. sp_cds* bal_coord2cds (sp_coord * *mat*)

Genera la matriz en formato CDS equivalente a la matriz mat en formato simple.

Por más información vea coord2cds().

Definición en la línea 252 del archivo bal.c.

Hace referencia a coord2cds().

```
253 {
254     return coord2cds(mat);
255 }
```

6.2.2.6. sp_packcol* bal_coord2packcol (sp_coord * *mat*)

Genera una instancia de la matriz mat en formato empaquetado por columna.

Por más información vea coord2packcol().

Definición en la línea 82 del archivo bal.c.

Hace referencia a coord2packcol().

```
83 {
84     return coord2packcol(mat);
85 }
```

6.2.2.7. sp_packcol* bal_coord2packcol_symmetric (sp_coord * *mat*)

Genera una instancia de la matriz mat en formato empaquetado por columna para matrices simétricas.

Por más información vea coord2packcol_symmetric().

Definición en la línea 92 del archivo bal.c.

Hace referencia a coord2packcol_symmetric().

```
93 {
94     return coord2packcol_symmetric(mat);
95 }
```

6.2.2.8. int* bal_elimination_tree (sp_packcol * *A*, int * *nnz*)

Calcula el árbol de eliminación de la matriz simétrica A.

Por más información vea elimination_tree().

Definición en la línea 192 del archivo bal.c.

Hace referencia a elimination_tree().

```
193 {
194     return elimination_tree(A, nnz);
195 }
```

6.2.2.9. void bal_free_cds (sp_cds * *A*)

Libera la memoria reservada por la matriz `A`.

Por más información ver free_cds()

Definición en la línea 332 del archivo bal.c.

Hace referencia a free_cds().

```
333 {
334     free_cds(A);
335 }
```

6.2.2.10. void bal_free_coord (sp_coord * A)

Libera la memoria reservada por la matriz A.

Por más información ver free_coord()

Definición en la línea 312 del archivo bal.c.

Hace referencia a free_coord().

```
313 {
314     free_coord(A);
315 }
```

6.2.2.11. void bal_free_packcol (sp_packcol * A)

Libera la memoria reservada por la matriz A.

Por más información ver free_packcol()

Definición en la línea 322 del archivo bal.c.

Hace referencia a free_packcol().

```
323 {
324     free_packcol(A);
325 }
```

6.2.2.12. void bal_imprimir_cds (FILE * fp, sp_cds * mat)

Imprime la matriz guardada en formato simple por coordenadas en fp.

Por más información vea sp_imprimir_cds().

Definición en la línea 262 del archivo bal.c.

Hace referencia a sp_imprimir_cds().

```
263 {
264     sp_imprimir_cds(fp, mat);
265 }
```

6.2.2.13. void bal_imprimir_coord (FILE * fp, sp_coord * mat)

Imprime la matriz guardada en formato simple por coordenadas en fp.

Por más información vea sp_imprimir_coord().

Definición en la línea 102 del archivo bal.c.

Hace referencia a sp_imprimir_coord().

```
103 {
104     sp_imprimir_coord(fp, mat);
105 }
```

6.2.2.14. void bal_imprimir_packcol (FILE * *fp*, sp_packcol * *mat*)

Imprime la matriz guardada en formato empaquetado por columna en fp.

Por más información vea sp_imprimir_packcol().

Definición en la línea 112 del archivo bal.c.

Hace referencia a sp_imprimir_packcol().

```
113 {
114     sp_imprimir_packcol(fp, mat);
115 }
```

6.2.2.15. sp_coord* bal_load_coord (FILE * *fp*)

Escribe en `fp` la matriz `A` en un formato entendible por load_coord().

Por más información vea load_coord().

Definición en la línea 152 del archivo bal.c.

Hace referencia a load_coord().

```
153 {
154     return load_coord(fp);
155 }
```

6.2.2.16. sp_coord* bal_mat2coord (int *n*, int *m*, double ** *mat*)

Genera la matriz dispersa equivalente a la matriz completa mat (n x m).

Por más información, vea mat2coord().

Definición en la línea 72 del archivo bal.c.

Hace referencia a mat2coord().

```
73 {
74     return mat2coord(n, m, mat);
75 }
```

6.2.2.17. sp_cds* bal_mult_mat_cds (sp_cds * *A*, sp_cds * *B*)

Multiplica dos matrices en formato CDS.

Por más información ver mult_mat_cds().

Definición en la línea 282 del archivo bal.c.

Hace referencia a mult_mat_cds().

```
283 {
284     return mult_mat_cds(A, B);
285 }
```

6.2.2.18. void bal_mult_vec_cds (sp_cds * A, double * x, double * y)

Multiplica una matriz en formato CDS por un vector.

Por más información ver mult_vec_cds().

Definición en la línea 272 del archivo bal.c.

Hace referencia a mult_vec_cds().

```
273 {
274     mult_vec_cds(A, x, y);
275 }
```

6.2.2.19. void bal_mult_vec_packcol (sp_packcol * A, double * x, double * y)

Multiplica una matriz empaquetada por columna por un vector.

Por más información vea mult_vec_packcol().

Definición en la línea 132 del archivo bal.c.

Hace referencia a mult_vec_packcol().

```
133 {
134     mult_vec_packcol(A, x, y);
135 }
```

6.2.2.20. void bal_mult_vec_packcol_symmetric (sp_packcol * A, double * x, double * y)

Multiplica una matriz simétrica empaquetada por columna por un vector.

Por más información vea mult_vec_packcol_symmetric().

Definición en la línea 122 del archivo bal.c.

Hace referencia a mult_vec_packcol_symmetric().

```
123 {
124     mult_vec_packcol_symmetric(A, x, y);
125 }
```

6.2.2.21. void bal_numerical_factorization (sp_packcol * A, sp_packcol * L)

Sobreescribe `L` con la factorización de Cholesky de A.

Por más información vea numerical_factorization().

Definición en la línea 212 del archivo bal.c.

Hace referencia a numerical_factorization().

```
213 {
214     numerical_factorization(A, L);
215 }
```

6.2.2.22. sp_packcol* bal_permutar_packcol (unsigned int * *p*, sp_packcol * *A*)

Aplica una permutación de columnas sobre la matriz `A`.

Por más información vea permutar_packcol().

Definición en la línea 302 del archivo bal.c.

Hace referencia a permutar_packcol().

```
303 {
304     return permutar_packcol(p, A);
305 }
```

6.2.2.23. int bal_row_traversal_packcol (sp_packcol * *A*, int * *i*, int * *j*, int * *posij*)

Implementa un mecanismo eficiente para recorrer por filas una matriz dispersa empaquetada por columnas.

Por más información vea row_traversal_packcol().

Definición en la línea 142 del archivo bal.c.

Hace referencia a row_traversal_packcol().

```
143 {
144     return row_traversal_packcol(A, i, j, posij);
145 }
```

6.2.2.24. void bal_save_cds (FILE * *fp*, sp_cds * *A*)

Imprime la matriz `A` en formato matlab en el archivo `fp`.

Por más información vea save_cds().

Definición en la línea 292 del archivo bal.c.

Hace referencia a save_cds().

```
293 {
294     save_cds(fp, A);
295 }
```

6.2.2.25. void bal_save_coord (FILE * *fp*, sp_coord * *A*)

Escribe en `fp` la matriz `A` en un formato entendible por load_coord().

Por más información vea save_coord().

Definición en la línea 162 del archivo bal.c.

Hace referencia a save_coord().

```
163 {
164     save_coord(fp, A);
165 }
```

6.2.2.26. void bal_save_packcol (FILE * *fp*, sp_packcol * *A*)

Imprime la matriz `A` en formato matlab en el archivo `fp`.

Por más información vea save_packcol().

Definición en la línea 182 del archivo bal.c.

Hace referencia a save_packcol().

```
183 {
184     save_packcol(fp, A);
185 }
```

6.2.2.27. void bal_save_packcol_symmetric (FILE * *fp*, sp_packcol * *A*)

Imprime la matriz simétrica empaquetada por columna en formato matlab en el archivo fp.

Por más información vea save_packcol_symmetric().

Definición en la línea 172 del archivo bal.c.

Hace referencia a save_packcol_symmetric().

```
173 {
174     save_packcol_symmetric(fp, A);
175 }
```

6.2.2.28. sp_packcol* bal_symbolic_factorization (sp_packcol * *A*)

Factorización simbólica de la matriz A.

Por más información vea symbolic_factorization().

Definición en la línea 202 del archivo bal.c.

Hace referencia a symbolic_factorization().

```
203 {
204     return symbolic_factorization(A);
205 }
```

6.3. Referencia del Archivo cholesky/cholesky.c

Implementación de la factorización de Cholesky para matrices dispersas.

```
#include <stdlib.h>
#include <math.h>
#include "cholesky.h"
#include "../sparse/sp_packcol.h"
#include "../utils.h"
```

Funciones

- int * elimination_tree (sp_packcol *A, int *nnz)

Calcula el árbol de eliminación de la matriz simétrica A.

- void merge (sp_packcol *B, int j, int k, int *ma)

  Fusiona la estructura de la columna j de B en la estructura actual de la columna k, representada por ma.

- void make_column (int k, int *ma, sp_packcol *L)

  Arma la k-esima columna de L.

- sp_packcol * symbolic_factorization (sp_packcol *A)

  Factorización simbólica de la matriz A.

- void numerical_factorization (sp_packcol *A, sp_packcol *L)

  Sobreescribe `L` *con la factorización de Cholesky de A.*

- void cholesky_Lsolver (sp_packcol *L, double *b)

  Sobrescribe `b` *con la solución del sistema* $Ly = b$.

- void cholesky_LTsolver (sp_packcol *L, double *b)

  Sobreescribe `b` *con la solución de* $L^T x = b$.

- void cholesky_solver (sp_packcol *A, double *b)

  Resuelve un sistema lineal mediante el algoritmo de Cholesky optimizado para matrices dispersas.

6.3.1. Descripción detallada

Implementación de la factorización de Cholesky para matrices dispersas.

Este archivo contiene la implementación de la factorización de Cholesky, optimizado para la estructura de matrices dispersas empaquetadas por columna.

Esta implementación realiza los siguientes pasos:

- Construcción de árbol de eliminación (cálculo del número máximo de elementos no cero de la factorización)
- Factorización simbólica
- Factorización numérica

Para obtener el mejor rendimiento de este algoritmo, se recomienda preprocesar la matriz a factorizar con algún algoritmo de reordenamiento, para así reducir el *fill-in* producido.

Definición en el archivo cholesky.c.

6.3.2. Documentación de las funciones

6.3.2.1. void cholesky_Lsolver (sp_packcol * *L*, double * *b*)

Sobrescribe `b` con la solución del sistema $Ly = b$.

Parámetros:

L Matriz `nxn` triangular inferior, factorización de Cholesky

b Vector de largo `n`

Al final de la ejecución el vector `b` es igual al vector `y` tal que $Ly = b$

Nota:

Ver el algoritmo 9.2 descrito en la sección 9.2 del paper de Stewart (ver las referencias)

Definición en la línea 329 del archivo cholesky.c.

Hace referencia a sp_packcol::colp, sp_packcol::ncol, sp_packcol::rx, y sp_packcol::val.

Referenciado por bal_cholesky_Lsolver(), y cholesky_solver().

```
330 {
331     int ii, i, j;
332
333     for (j = 0; j < L->ncol; ++j) {
334         b[j] /= L->val[L->colp[j]];
335
336         for (ii = L->colp[j]+1; ii < L->colp[j+1]; ++ii) {
337             i = L->rx[ii];
338             b[i] -= ( b[j] * L->val[ii] );
339         }
340     }
341 }
```

6.3.2.2. void cholesky_LTsolver (sp_packcol * *L*, double * *b*)

Sobreescribe `b` con la solución de $L^T x = b$.

Parámetros:

L Matriz `n`x`n`triangular inferior, factorización de Cholesky

b Vector de largo `n`

Al final de la ejecución el vector `b` es igual al vector `y` tal que $L^T y = b$

Nota:

Ver el algoritmo 9.3 descrito en la sección 9.2 del paper de Stewart (ver las referencias)

Definición en la línea 354 del archivo cholesky.c.

Hace referencia a sp_packcol::colp, sp_packcol::ncol, sp_packcol::rx, y sp_packcol::val.

Referenciado por bal_cholesky_LTsolver(), y cholesky_solver().

```
355 {
356     int ii, i, j;
357
358     for (j = L->ncol-1; j >= 0; --j) {
359         for (ii = L->colp[j]+1; ii < L->colp[j+1]; ++ii) {
360             i = L->rx[ii];
361             b[j] -= ( b[i] * L->val[ii] );
362         }
363         b[j] /= L->val[L->colp[j]];
364     }
365 }
```

6.3.2.3. void cholesky_solver (sp_packcol * *A*, double * *b*)

Resuelve un sistema lineal mediante el algoritmo de Cholesky optimizado para matrices dispersas.

Parámetros:

A Matriz del sistema lineal, simétrica definida positiva, dispersa empaquetada por columna.

b ENTRADA/SALIDA: Vector a igualar mediante `Ax`

Esta función calcula un vetor `x` tal que $Ax = b$ y lo guarda en el mismo vector `b`. Esta implementación está completamente basada en el formato de matriz dispersa empaquetado por columna, para mejorar el desempeño de la aplicación.

La forma de proceder del algoritmo es la siguiente:

1. Se realiza la factorización simbólica de `A`
2. Se realiza la factorización numérica de `A`
3. Se resuelven los sistemas triangulares $Ly = b$ y luego $L^T x = y$

Definición en la línea 384 del archivo cholesky.c.

Hace referencia a BAL_ERROR, cholesky_Lsolver(), cholesky_LTsolver(), sp_packcol::ncol, sp_packcol::nrow, numerical_factorization(), y symbolic_factorization().

Referenciado por bal_cholesky_solver().

```
385 {
386     sp_packcol *L;
387 
388     if (A->nrow != A->ncol) {
389         /* Habría que verificar además que es simétrica y definida positiva */
390         BAL_ERROR("La matriz no es cuadrada");
391         return;
392     }
393 
394     L = symbolic_factorization(A);
395     numerical_factorization(A, L);
396     cholesky_Lsolver(L, b);
397     cholesky_LTsolver(L, b);
398 }
```

6.3.2.4. int * elimination_tree (sp_packcol * *A*, int * *nnz*)

Calcula el árbol de eliminación de la matriz simétrica A.

Parámetros:

A ENTRADA: Matriz simétrica dispersa empaquetada por columna a procesar

nnz SALIDA: Cantidad de elementos no cero en la factorización de Cholesky

Devuelve:

Estructura de padres del árbol de eliminación

Esta función calcula el árbol de eliminación de la matriz A, devuelve la estructura de padres del árbol y la cantidad de elementos no cero de la factorización de Cholesky.

Nota:

Este paso es un paso previo a la factorización simbólica.
Esta implementación está basada en el algoritmo descrito en la sección 7.2 del paper de Stewart (ver las referencias).

Definición en la línea 42 del archivo cholesky.c.

Hace referencia a BAL_ERROR, sp_packcol::ncol, sp_packcol::nrow, y row_traversal_packcol().

Referenciado por bal_elimination_tree(), y symbolic_factorization().

```
43 {
44     int *touched, *parent;
45     int ix, i, j, posij, js;
46 
47     /* Inicialización */
48     *nnz = 0;
49 
50     if (A->nrow != A->ncol) {
51         BAL_ERROR("La matriz debe ser cuadrada");
52         return NULL;
53     }
54 
55     /* Inicializa estructuras */
56     touched = (int*)malloc(sizeof(int) * A->ncol);
57     parent = (int*)malloc(sizeof(int) * A->ncol);
58     for (ix = 0; ix < A->ncol; ++ix)
59         touched[ix] = parent[ix] = -1;
60 
61     /* Recorre las filas de A */
62     i = -2;
63     row_traversal_packcol(A, &i, &j, &posij);
64     for (ix = 0; ix < A->nrow; ++ix) {
65         while (row_traversal_packcol(A, &i, &j, &posij) != -1) {
66             if (i == j) {
67                 /* Procesar elemento de la diagonal */
68                 *nnz += 1;
69                 touched[j] = i;
70             }
71             else {
72                 /* Elemento no en la diagonal. Buscar el arbol */
73                 js = j;
74                 while (touched[js] != i) {
75                     touched[js] = i;
76                     *nnz += 1;
77 
78                     if (parent[js] == -1) {
79                         parent[js] = i;
80                         break;
81                     }
82 
83                     js = parent[js];
84                 }
85             }
86         }
87     }
88 
89     /* Liberamos la memoria utilizada */
90     i = -2;
91     row_traversal_packcol(NULL, &i, &j, &posij);
92     free(touched);
93 
94     return parent;
95 }
```

6.3.2.5. void make_column (int *k*, int * *ma*, sp_packcol * *L*)

Arma la k-esima columna de L.

Parámetros:

k Columna a contruir

ma Estructura de la columna a contruir

L Matriz dispersa donde construir la columna

Esta función toma la estructura de la columna k, contenida en ma y la transfiere a la k-esima columna de L. Además reinicializa ma.

Nota:

Ver el algoritmo 8.3 descrito en la sección 8.2 del paper de Stewart (ver las referencias).

Definición en la línea 153 del archivo cholesky.c.

Hace referencia a sp_packcol::colp, sp_packcol::ncol, y sp_packcol::rx.

Referenciado por symbolic_factorization().

```
154 {
155     int ii, m, mt;
156 
157     if (k == 0)
158         L->colp[0] = 0;
159 
160     ii = L->colp[k];
161     m = k;
162 
163     while (m < L->ncol) {
164         L->rx[ii] = m;
165         ++ii;
166         mt = ma[m];
167         ma[m] = L->ncol;
168         m = mt;
169     }
170 
171     L->colp[k+1] = ii;
172 }
```

6.3.2.6. void merge (sp_packcol * *B*, int *j*, int *k*, int * *ma*)

Fusiona la estructura de la columna j de B en la estructura actual de la columna k, representada por ma.

Nota:

Ver el algoritmo 8.2 descrito en la sección 8.2 del paper de Stewart (ver las referencias).

Atención:

El pseudo-código mostrado en el paper de Stewart contiene errores. Se recomienda fuertemente comparar esta implementación (buscando los lugares marcados como corrección) con el pseudo-código de Stewart y ver las diferencias.

Definición en la línea 108 del archivo cholesky.c.

Hace referencia a sp_packcol::colp, y sp_packcol::rx.

Referenciado por symbolic_factorization().

```
109 {
110     int m, m1, i, ii;
111 
112     m = k;
113 
114     /* Itera en los elementos de la columna j de B */
115     for (ii = B->colp[j]; ii < B->colp[j+1]; ++ii) { /* Corrección al algoritmo de Stewart */
116         i = B->rx[ii];
117 
118         /* Corrección al algoritmo de Stewart */
119         if (i-j < 1)
120             continue;
121 
122         /* Busca m y m1 con m < i <= m1 */
123         m1 = m;
124         while (i > m1) {
125             m = m1;
126             m1 = ma[m];
127         }
128 
129         if (i != m1) {
130             /* Insertar i en ma */
131             ma[m] = i;
132             ma[i] = m1;
133         }
134 
135         m = i;
136     }
137 }
```

6.3.2.7. void numerical_factorization (sp_packcol * *A*, sp_packcol * *L*)

Sobreescribe `L` con la factorización de Cholesky de A.

Parámetros:

A Matriz a factorizar con el algoritmo de Cholesky

L Matriz pre inicializada donde guardar la factorización

Nota:

La pre inicialización de la matriz `L` se realiza con la función symbolic_factorization().
Ver el algoritmo 9.1 descrito en la sección 9.1 del paper de Stewart (ver las referencias).

Definición en la línea 274 del archivo cholesky.c.

Hace referencia a sp_packcol::colp, sp_packcol::ncol, row_traversal_packcol(), sp_packcol::rx, y sp_packcol::val.

Referenciado por bal_numerical_factorization(), y cholesky_solver().

```
275 {
276     int ii, i, j, poskj, kx, k;
277     double *accum, Lkj, Lkkinv;
278 
279     accum = (double*)malloc(sizeof(double) * L->ncol);
280 
281     k = -2;
282     row_traversal_packcol(L, &k, &j, &poskj);
283     for (kx = 0; kx < L->ncol; ++kx) {  /* Procesa la columna k */
284         while (row_traversal_packcol(L, &k, &j, &poskj) != -1) {
285             if (j == k) { /* Inicializar accum */
```

```
286                 for (ii = L->colp[k]; ii < L->colp[k+1]; ++ii)
287                     accum[L->rx[ii]] = 0;
288                 for (ii = A->colp[k]; ii < A->colp[k+1]; ++ii)
289                     accum[A->rx[ii]] = A->val[ii];
290             }
291             else { /* Restar L[k:n,j] de L[k:n,k] */
292                 Lkj = L->val[poskj];
293                 for (ii = poskj; ii < L->colp[j+1]; ++ii) {
294                     i = L->rx[ii];
295                     accum[i] -= ( Lkj * L->val[ii] );
296                 }
297             }
298         }
299
300         /* Mueve L[k:n,k] de accum a L, ajustando sus componentes */
301         for (ii = L->colp[k]; ii < L->colp[k+1]; ++ii) { /* Corrección al algoritmo de Stewart */
302             i = L->rx[ii];
303             if (i == k) {
304                 L->val[ii] = sqrt(accum[i]);
305                 Lkkinv = 1 / L->val[ii];
306             }
307             else
308                 L->val[ii] = Lkkinv * accum[i];
309         }
310     }
311
312     /* Libera memoria auxiliar */
313     k = -2;
314     row_traversal_packcol(NULL, &k, &j, &poskj);
315     free(accum);
316 }
```

6.3.2.8. sp_packcol * symbolic_factorization (sp_packcol * *A*)

Factorización simbólica de la matriz A.

Parámetros:

A Matriz a factorizar

Devuelve:

La factorización simbólica de la matriz A

Nota:

Ver el algoritmo 8.1 descrito en la sección 8.2 del paper de Stewart (ver las referencias).

Atención:

El pseudo-código mostrado en el paper de Stewart contiene errores. Se recomienda fuertemente comparar esta implementación (buscando los lugares marcados como corrección) con el pseudo-código de Stewart y ver las diferencias.

Definición en la línea 188 del archivo cholesky.c.

Hace referencia a BAL_ERROR, sp_packcol::colp, elimination_tree(), make_column(), merge(), sp_packcol::ncol, sp_packcol::nnz, sp_packcol::nrow, sp_packcol::rx, y sp_packcol::val.

Referenciado por bal_symbolic_factorization(), y cholesky_solver().

```
189 {
```

```
190     int nnz;                /* Cantidad de elementos no cero */
191     int *parents;           /* Arbol de eliminacion */
192     int *bs;                /* Hijos por nodo ("Baby Sitter") */
193     int *ma;                /* "Merge array" */
194     sp_packcol *L;          /* Resultado de la factorizacion simbolica */
195     int i, j, k, jt;
196
197     if (A->nrow != A->ncol) {
198         BAL_ERROR("La matriz debe ser cuadrada");
199         return NULL;
200     }
201
202     /* Calcula la cantidad de elementos no cero */
203     parents = elimination_tree(A, &nnz);
204     free(parents);
205
206     /* Inicializacion */
207     L = (sp_packcol*)malloc(sizeof(sp_packcol));
208     L->nrow = A->nrow;
209     L->ncol = A->ncol;
210     L->nnz = (unsigned)nnz;
211     L->colp = (unsigned int*)malloc(sizeof(unsigned int) * (A->ncol+1));
212     L->rx = (unsigned int*)malloc(sizeof(unsigned int) * nnz);
213     L->val = (double*)malloc(sizeof(double) * nnz);
214     for (i = 0; i <= A->ncol; ++i)
215         L->colp[i] = 0;
216     for (i = 0; i < nnz; ++i) {
217         L->rx[i] = 0;
218         L->val[i] = 0;
219     }
220
221     bs = (int*)malloc(sizeof(int) * A->ncol);
222     ma = (int*)malloc(sizeof(int) * A->ncol);
223     for (i = 0; i < A->ncol; ++i) {
224         bs[i] = -1;
225         ma[i] = A->ncol;
226     }
227
228     /* Itera en las columnas de A */
229     for (k = 0; k < A->ncol; ++k) {
230         /* Computar la estructura de la k-esima columna */
231         merge(A, k, k, ma);
232         j = bs[k];
233         bs[k] = -1; /* Corrección al algoritmo de Stewart */
234         while (j != -1) {
235             merge(L, j, k, ma);
236             jt = bs[j];
237             bs[j] = -1;
238             j = jt;
239         }
240
241         /* Establecer la k-esima columna en L */
242         make_column(k, ma, L);
243
244         /* Actualizar bs */
245         if (k != j) {
246             j = L->rx[L->colp[k] + 1];      /* j es el padre de k */
247             while (j != -1) {
248                 jt = j;
249                 j = bs[j];
250             }
251             bs[jt] = k;
252         }
253     }
254
255     /* Libera memoria auxiliar */
256     free(bs);
```

```
257     free(ma);
258
259     return L;
260 }
```

6.4. Referencia del Archivo cholesky/cholesky.h

Archivo de cabecera para la factorización de Cholesky.

```
#include "../sparse/sp_packcol.h"
```

Funciones

- int ∗ elimination_tree (sp_packcol ∗A, int ∗nnz)
  Calcula el árbol de eliminación de la matriz simétrica A.
- sp_packcol ∗ symbolic_factorization (sp_packcol ∗A)
  Factorización simbólica de la matriz A.
- void numerical_factorization (sp_packcol ∗A, sp_packcol ∗L)
  Sobreescribe `L` *con la factorización de Cholesky de A.*
- void cholesky_Lsolver (sp_packcol ∗L, double ∗b)
  Sobrescribe `b` *con la solución del sistema* $Ly = b$.
- void cholesky_LTsolver (sp_packcol ∗L, double ∗b)
  Sobreescribe `b` *con la solución de* $L^T x = b$.
- void cholesky_solver (sp_packcol ∗A, double ∗b)
  Resuelve un sistema lineal mediante el algoritmo de Cholesky optimizado para matrices dispersas.

6.4.1. Descripción detallada

Archivo de cabecera para la factorización de Cholesky.

Este archivo define las funciones que implementan la factorización de Cholesky para matrices dispersas. Por más detalles acerca de la implementación, vea la documentación del archivo cholesky.c

Definición en el archivo cholesky.h.

6.4.2. Documentación de las funciones

6.4.2.1. void cholesky_Lsolver (sp_packcol ∗ *L*, double ∗ *b*)

Sobrescribe `b` con la solución del sistema $Ly = b$.

Parámetros:

L Matriz `nxn` triangular inferior, factorización de Cholesky

b Vector de largo `n`

Al final de la ejecución el vector b es igual al vector y tal que $Ly = b$

Nota:

Ver el algoritmo 9.2 descrito en la sección 9.2 del paper de Stewart (ver las referencias)

Definición en la línea 329 del archivo cholesky.c.

Hace referencia a sp_packcol::colp, sp_packcol::ncol, sp_packcol::rx, y sp_packcol::val.

Referenciado por bal_cholesky_Lsolver(), y cholesky_solver().

```
330 {
331     int ii, i, j;
332 
333     for (j = 0; j < L->ncol; ++j) {
334         b[j] /= L->val[L->colp[j]];
335 
336         for (ii = L->colp[j]+1; ii < L->colp[j+1]; ++ii) {
337             i = L->rx[ii];
338             b[i] -= ( b[j] * L->val[ii] );
339         }
340     }
341 }
```

6.4.2.2. void cholesky_LTsolver (sp_packcol * L, double * b)

Sobreescribe b con la solución de $L^T x = b$.

Parámetros:

L Matriz nxntriangular inferior, factorización de Cholesky

b Vector de largo n

Al final de la ejecución el vector b es igual al vector y tal que $L^T y = b$

Nota:

Ver el algoritmo 9.3 descrito en la sección 9.2 del paper de Stewart (ver las referencias)

Definición en la línea 354 del archivo cholesky.c.

Hace referencia a sp_packcol::colp, sp_packcol::ncol, sp_packcol::rx, y sp_packcol::val.

Referenciado por bal_cholesky_LTsolver(), y cholesky_solver().

```
355 {
356     int ii, i, j;
357 
358     for (j = L->ncol-1; j >= 0; --j) {
359         for (ii = L->colp[j]+1; ii < L->colp[j+1]; ++ii) {
360             i = L->rx[ii];
361             b[j] -= ( b[i] * L->val[ii] );
362         }
363         b[j] /= L->val[L->colp[j]];
364     }
365 }
```

6.4.2.3. void cholesky_solver (sp_packcol * *A*, double * *b*)

Resuelve un sistema lineal mediante el algoritmo de Cholesky optimizado para matrices dispersas.

Parámetros:

A Matriz del sistema lineal, simétrica definida positiva, dispersa empaquetada por columna.

b ENTRADA/SALIDA: Vector a igualar mediante `Ax`

Esta función calcula un vetor `x` tal que $Ax = b$ y lo guarda en el mismo vector `b`. Esta implementación está completamente basada en el formato de matriz dispersa empaquetado por columna, para mejorar el desempeño de la aplicación.

La forma de proceder del algoritmo es la siguiente:

1. Se realiza la factorización simbólica de `A`
2. Se realiza la factorización numérica de `A`
3. Se resuelven los sistemas triangulares $Ly = b$ y luego $L^T x = y$

Definición en la línea 384 del archivo cholesky.c.

Hace referencia a BAL_ERROR, cholesky_Lsolver(), cholesky_LTsolver(), sp_packcol::ncol, sp_packcol::nrow, numerical_factorization(), y symbolic_factorization().

Referenciado por bal_cholesky_solver().

```
385 {
386     sp_packcol *L;
387 
388     if (A->nrow != A->ncol) {
389         /* Habría que verificar además que es simétrica y definida positiva */
390         BAL_ERROR("La matriz no es cuadrada");
391         return;
392     }
393 
394     L = symbolic_factorization(A);
395     numerical_factorization(A, L);
396     cholesky_Lsolver(L, b);
397     cholesky_LTsolver(L, b);
398 }
```

6.4.2.4. int* elimination_tree (sp_packcol * *A*, int * *nnz*)

Calcula el árbol de eliminación de la matriz simétrica A.

Parámetros:

A ENTRADA: Matriz simétrica dispersa empaquetada por columna a procesar

nnz SALIDA: Cantidad de elementos no cero en la factorización de Cholesky

Devuelve:

Estructura de padres del árbol de eliminación

Esta función calcula el árbol de eliminación de la matriz A, devuelve la estructura de padres del árbol y la cantidad de elementos no cero de la factorización de Cholesky.

Nota:

Este paso es un paso previo a la factorización simbólica.
Esta implementación está basada en el algoritmo descrito en la sección 7.2 del paper de Stewart (ver las referencias).

Definición en la línea 42 del archivo cholesky.c.

Hace referencia a BAL_ERROR, sp_packcol::ncol, sp_packcol::nrow, y row_traversal_packcol().

Referenciado por bal_elimination_tree(), y symbolic_factorization().

```
43 {
44     int *touched, *parent;
45     int ix, i, j, posij, js;
46 
47     /* Inicialización */
48     *nnz = 0;
49 
50     if (A->nrow != A->ncol) {
51         BAL_ERROR("La matriz debe ser cuadrada");
52         return NULL;
53     }
54 
55     /* Inicializa estructuras */
56     touched = (int*)malloc(sizeof(int) * A->ncol);
57     parent = (int*)malloc(sizeof(int) * A->ncol);
58     for (ix = 0; ix < A->ncol; ++ix)
59         touched[ix] = parent[ix] = -1;
60 
61     /* Recorre las filas de A */
62     i = -2;
63     row_traversal_packcol(A, &i, &j, &posij);
64     for (ix = 0; ix < A->nrow; ++ix) {
65         while (row_traversal_packcol(A, &i, &j, &posij) != -1) {
66             if (i == j) {
67                 /* Procesar elemento de la diagonal */
68                 *nnz += 1;
69                 touched[j] = i;
70             }
71             else {
72                 /* Elemento no en la diagonal. Buscar el arbol */
73                 js = j;
74                 while (touched[js] != i) {
75                     touched[js] = i;
76                     *nnz += 1;
77 
78                     if (parent[js] == -1) {
79                         parent[js] = i;
80                         break;
81                     }
82 
83                     js = parent[js];
84                 }
85             }
86         }
87     }
88 
89     /* Liberamos la memoria utilizada */
90     i = -2;
91     row_traversal_packcol(NULL, &i, &j, &posij);
92     free(touched);
93 
94     return parent;
95 }
```

6.4.2.5. void numerical_factorization (sp_packcol * *A*, sp_packcol * *L*)

Sobreescribe L con la factorización de Cholesky de A.

Parámetros:

A Matriz a factorizar con el algoritmo de Cholesky

L Matriz pre inicializada donde guardar la factorización

Nota:

La pre inicialización de la matriz L se realiza con la función symbolic_factorization().
Ver el algoritmo 9.1 descrito en la sección 9.1 del paper de Stewart (ver las referencias).

Definición en la línea 274 del archivo cholesky.c.

Hace referencia a sp_packcol::colp, sp_packcol::ncol, row_traversal_packcol(), sp_packcol::rx, y sp_packcol::val.

Referenciado por bal_numerical_factorization(), y cholesky_solver().

```
275 {
276     int ii, i, j, poskj, kx, k;
277     double *accum, Lkj, Lkkinv;
278 
279     accum = (double*)malloc(sizeof(double) * L->ncol);
280 
281     k = -2;
282     row_traversal_packcol(L, &k, &j, &poskj);
283     for (kx = 0; kx < L->ncol; ++kx) {  /* Procesa la columna k */
284         while (row_traversal_packcol(L, &k, &j, &poskj) != -1) {
285             if (j == k) { /* Inicializar accum */
286                 for (ii = L->colp[k]; ii < L->colp[k+1]; ++ii)
287                     accum[L->rx[ii]] = 0;
288                 for (ii = A->colp[k]; ii < A->colp[k+1]; ++ii)
289                     accum[A->rx[ii]] = A->val[ii];
290             }
291             else { /* Restar L[k:n,j] de L[k:n,k] */
292                 Lkj = L->val[poskj];
293                 for (ii = poskj; ii < L->colp[j+1]; ++ii) {
294                     i = L->rx[ii];
295                     accum[i] -= ( Lkj * L->val[ii] );
296                 }
297             }
298         }
299 
300         /* Mueve L[k:n,k] de accum a L, ajustando sus componentes */
301         for (ii = L->colp[k]; ii < L->colp[k+1]; ++ii) { /* Corrección al algoritmo de Stewart */
302             i = L->rx[ii];
303             if (i == k) {
304                 L->val[ii] = sqrt(accum[i]);
305                 Lkkinv = 1 / L->val[ii];
306             }
307             else
308                 L->val[ii] = Lkkinv * accum[i];
309         }
310     }
311 
312     /* Libera memoria auxiliar */
313     k = -2;
314     row_traversal_packcol(NULL, &k, &j, &poskj);
315     free(accum);
316 }
```

6.4.2.6. sp_packcol* symbolic_factorization (sp_packcol * A)

Factorización simbólica de la matriz A.

Parámetros:

A Matriz a factorizar

Devuelve:

La factorización simbólica de la matriz A

Nota:

Ver el algoritmo 8.1 descrito en la sección 8.2 del paper de Stewart (ver las referencias).

Atención:

El pseudo-código mostrado en el paper de Stewart contiene errores. Se recomienda fuertemente comparar esta implementación (buscando los lugares marcados como corrección) con el pseudo-código de Stewart y ver las diferencias.

Definición en la línea 188 del archivo cholesky.c.

Hace referencia a BAL_ERROR, sp_packcol::colp, elimination_tree(), make_column(), merge(), sp_packcol::ncol, sp_packcol::nnz, sp_packcol::nrow, sp_packcol::rx, y sp_packcol::val.

Referenciado por bal_symbolic_factorization(), y cholesky_solver().

```
189 {
190     int nnz;                /* Cantidad de elementos no cero */
191     int *parents;           /* Arbol de eliminacion */
192     int *bs;                /* Hijos por nodo ("Baby Sitter") */
193     int *ma;                /* "Merge array" */
194     sp_packcol *L;          /* Resultado de la factorizacion simbolica */
195     int i, j, k, jt;
196 
197     if (A->nrow != A->ncol) {
198         BAL_ERROR("La matriz debe ser cuadrada");
199         return NULL;
200     }
201 
202     /* Calcula la cantidad de elementos no cero */
203     parents = elimination_tree(A, &nnz);
204     free(parents);
205 
206     /* Inicializacion */
207     L = (sp_packcol*)malloc(sizeof(sp_packcol));
208     L->nrow = A->nrow;
209     L->ncol = A->ncol;
210     L->nnz = (unsigned)nnz;
211     L->colp = (unsigned int*)malloc(sizeof(unsigned int) * (A->ncol+1));
212     L->rx = (unsigned int*)malloc(sizeof(unsigned int) * nnz);
213     L->val = (double*)malloc(sizeof(double) * nnz);
214     for (i = 0; i <= A->ncol; ++i)
215         L->colp[i] = 0;
216     for (i = 0; i < nnz; ++i) {
217         L->rx[i] = 0;
218         L->val[i] = 0;
219     }
220 
221     bs = (int*)malloc(sizeof(int) * A->ncol);
222     ma = (int*)malloc(sizeof(int) * A->ncol);
223     for (i = 0; i < A->ncol; ++i) {
```

```
224         bs[i] = -1;
225         ma[i] = A->ncol;
226     }
227 
228     /* Itera en las columnas de A */
229     for (k = 0; k < A->ncol; ++k) {
230         /* Computar la estructura de la k-esima columna */
231         merge(A, k, k, ma);
232         j = bs[k];
233         bs[k] = -1; /* Corrección al algoritmo de Stewart */
234         while (j != -1) {
235             merge(L, j, k, ma);
236             jt = bs[j];
237             bs[j] = -1;
238             j = jt;
239         }
240 
241         /* Establecer la k-esima columna en L */
242         make_column(k, ma, L);
243 
244         /* Actualizar bs */
245         if (k != j) {
246             j = L->rx[L->colp[k] + 1];      /* j es el padre de k */
247             while (j != -1) {
248                 jt = j;
249                 j = bs[j];
250             }
251             bs[jt] = k;
252         }
253     }
254 
255     /* Libera memoria auxiliar */
256     free(bs);
257     free(ma);
258 
259     return L;
260 }
```

6.5. Referencia del Archivo cholesky/reordenamiento.c

Implementación de algoritmos de reordenamiento.

```
#include <stdio.h>
#include <stdlib.h>
#include "reordenamiento.h"
#include "../sparse/sp_packcol.h"
```

Funciones

- sp_packcol * permutar_packcol (unsigned int *p, sp_packcol *A)

  Aplica una permutación de columnas sobre la matriz `A`.

- unsigned int * reordenar_packcol (sp_packcol *A)

  Reordena las columnas de A para minimizar el fill-in *producido por una etapa posterior de factorización.*

6.5.1. Descripción detallada

Implementación de algoritmos de reordenamiento.

Definición en el archivo reordenamiento.c.

6.5.2. Documentación de las funciones

6.5.2.1. sp_packcol * permutar_packcol (unsigned int * *p*, sp_packcol * *A*)

Aplica una permutación de columnas sobre la matriz `A`.

Parámetros:

p Permutación. `p[i]` = La p[i]-ésima columna pasa a ser la columna i

A Matriz a permutarle las columnas

Devuelve:

La misma matriz `A` pero con las matrices permutadas según `p`

Nota:

La matriz `A` y el resultado no comparten memoria.

Definición en la línea 21 del archivo reordenamiento.c.

Hace referencia a sp_packcol::colp, sp_packcol::ncol, sp_packcol::nnz, sp_packcol::nrow, sp_packcol::rx, y sp_packcol::val.

Referenciado por bal_permutar_packcol().

```
22 {
23     unsigned int i, j, col;
24     sp_packcol* B;
25 
26     B = (sp_packcol*)malloc(sizeof(sp_packcol));
27     B->nrow = A->nrow;
28     B->ncol = A->ncol;
29     B->nnz = A->nnz;
30 
31     if (B->nnz == 0) {
32         B->colp = NULL;
33         B->rx = NULL;
34         B->val = NULL;
35     }
36     else {
37         B->colp = (unsigned int*)malloc(sizeof(unsigned int) * (B->ncol+1));
38         B->rx = (unsigned int*)malloc(sizeof(unsigned int) * B->nnz);
39         B->val = (double*)malloc(sizeof(double) * B->nnz);
40 
41         j = 0;
42         for(col=0; col < B->ncol; ++col) {  /*< Para cada columna col en B */
43             B->colp[col] = j;                   /*< Los elementos de la columna col comenzarán en la en
44             for (i=A->colp[p[col]]; i < A->colp[p[col]+1]; ++i) {   /*< Recorre la columna p[col]-ésima
45                 B->rx[j] = A->rx[i];                                /*< y copia el contenido */
46                 B->val[j] = A->val[i];
47                 ++j;
48             }
49         }
50         B->colp[col] = j;                   /*< Elemento extra para indicar el fin de la ultima columna
51     }
```

```
52
53     return B;
54 }
```

6.5.2.2. unsigned int ∗ reordenar_packcol (sp_packcol ∗ *A*)

Reordena las columnas de A para minimizar el *fill-in* producido por una etapa posterior de factorización.

Parámetros:

A Matriz a reordenar

Tareas Pendientes

Implementar un algoritmo de reordenamiento

Actualmente esta función no implementa ningún algoritmo de reordenamiento. Simplemente retorna la permutación trivial para la cantidad de columnas de `A`.

Un algoritmo de reordenamiento básicamente calcula una permutación a ser aplicada en las filas/columnas de la matriz.

Mientras que no se cuente con una implementación de un algoritmo de reordenamiento, se puede calcular una permutación mediante código externo y utilizar la función permutar_packcol(), que recibe una permutación como parámetro y realiza la permutación sobre una matriz empaquetada por columna.

Nota:

Para aquel que pretenda implementar un algoritmo de reordenamiento para BAL, una posibilidad es el algoritmo de Cuthill-McKee. Puede realizarse una implementación de cero, o tratar de reutilizar una existente, como la disponible en el siguiente sitio: `http://www.math.temple.edu/~daffi/software/rcm/`

Definición en la línea 79 del archivo reordenamiento.c.

Hace referencia a sp_packcol::ncol.

```
80 {
81     unsigned int *p;
82     unsigned int i;
83
84     p = (unsigned int*)malloc(sizeof(unsigned int) * A->ncol);
85
86     for (i=0; i < A-> ncol; ++i)
87         p[i] = i;
88
89     return p;
90 }
```

6.6. Referencia del Archivo cholesky/reordenamiento.h

Archivo de cabecera para algoritmos de reordenamiento.

```
#include "../sparse/sp_packcol.h"
```

Funciones

- unsigned int * reordenar_packcol (sp_packcol *A)

  Reordena las columnas de A para minimizar el fill-in *producido por una etapa posterior de factorización.*

- sp_packcol * permutar_packcol (unsigned int *p, sp_packcol *A)

  Aplica una permutación de columnas sobre la matriz `A`.

6.6.1. Descripción detallada

Archivo de cabecera para algoritmos de reordenamiento.

Este archivo contiene (o contendrá) implementaciones de algoritmos de reordenamiento.

El reordenamiento es el primer paso en la resolución eficiente de sistemas de ecuaciones representados mediante matrices dispersas. El reordenamiento previo de una matriz tiene como principal objetivo la minimización del *fill-in* producido por las etapas de factorización que lo suceden.

Definición en el archivo reordenamiento.h.

6.6.2. Documentación de las funciones

6.6.2.1. sp_packcol* permutar_packcol (unsigned int * *p*, sp_packcol * *A*)

Aplica una permutación de columnas sobre la matriz `A`.

Parámetros:

p Permutación. `p[i]` = La p[i]-ésima columna pasa a ser la columna i

A Matriz a permutarle las columnas

Devuelve:

La misma matriz `A` pero con las matrices permutadas según `p`

Nota:

La matriz `A` y el resultado no comparten memoria.

Definición en la línea 21 del archivo reordenamiento.c.

Hace referencia a sp_packcol::colp, sp_packcol::ncol, sp_packcol::nnz, sp_packcol::nrow, sp_packcol::rx, y sp_packcol::val.

Referenciado por bal_permutar_packcol().

```
22 {
23     unsigned int i, j, col;
24     sp_packcol* B;
25 
26     B = (sp_packcol*)malloc(sizeof(sp_packcol));
27     B->nrow = A->nrow;
28     B->ncol = A->ncol;
29     B->nnz = A->nnz;
30 
31     if (B->nnz == 0) {
32         B->colp = NULL;
```

```
33          B->rx = NULL;
34          B->val = NULL;
35      }
36      else {
37          B->colp = (unsigned int*)malloc(sizeof(unsigned int) * (B->ncol+1));
38          B->rx = (unsigned int*)malloc(sizeof(unsigned int) * B->nnz);
39          B->val = (double*)malloc(sizeof(double) * B->nnz);
40
41          j = 0;
42          for(col=0; col < B->ncol; ++col) {  /*< Para cada columna col en B */
43              B->colp[col] = j;                   /*< Los elementos de la columna col comenzarán en la en
44              for (i=A->colp[p[col]]; i < A->colp[p[col]+1]; ++i) {   /*< Recorre la columna p[col]-ésima
45                  B->rx[j] = A->rx[i];                                /*< y copia el contenido */
46                  B->val[j] = A->val[i];
47                  ++j;
48              }
49          }
50          B->colp[col] = j;                   /*< Elemento extra para indicar el fin de la ultima columna
51      }
52
53      return B;
54  }
```

6.6.2.2. unsigned int* reordenar_packcol (sp_packcol * *A*)

Reordena las columnas de A para minimizar el *fill-in* producido por una etapa posterior de factorización.

Parámetros:

A Matriz a reordenar

Tareas Pendientes

Implementar un algoritmo de reordenamiento

Actualmente esta función no implementa ningún algoritmo de reordenamiento. Simplemente retorna la permutación trivial para la cantidad de columnas de `A`.

Un algoritmo de reordenamiento básicamente calcula una permutación a ser aplicada en las filas/columnas de la matriz.

Mientras que no se cuente con una implementación de un algoritmo de reordenamiento, se puede calcular una permutación mediante código externo y utilizar la función permutar_packcol(), que recibe una permutación como parámetro y realiza la permutación sobre una matriz empaquetada por columna.

Nota:

Para aquel que pretenda implementar un algoritmo de reordenamiento para BAL, una posibilidad es el algoritmo de Cuthill-McKee. Puede realizarse una implementación de cero, o tratar de reutilizar una existente, como la disponible en el siguiente sitio: `http://www.math.temple.edu/~daffi/software/rcm/`

Definición en la línea 79 del archivo reordenamiento.c.

Hace referencia a sp_packcol::ncol.

```
80  {
81      unsigned int *p;
82      unsigned int i;
83
```

```
84     p = (unsigned int*)malloc(sizeof(unsigned int) * A->ncol);
85
86     for (i=0; i < A-> ncol; ++i)
87         p[i] = i;
88
89     return p;
90 }
```

6.7. Referencia del Archivo oper.c

Operaciones básicas, implementación.

```
#include <stdlib.h>
#include <glib.h>
#include "sparse/sp_packcol.h"
#include "sparse/sp_cds.h"
#include "utils.h"
#include "oper.h"
```

Funciones

- void mult_vec_packcol (sp_packcol *A, double *x, double *y)
  Multiplica una matriz empaquetada por columna por un vector.
- void mult_vec_packcol_symmetric (sp_packcol *A, double *x, double *y)
  Multiplica una matriz simétrica empaquetada por columna por un vector.
- void mult_vec_cds (sp_cds *A, double *x, double *y)
  Multiplica una matriz en formato CDS por un vector.
- sp_cds * mult_mat_cds (sp_cds *A, sp_cds *B)
  Multiplica dos matrices en formato CDS.

6.7.1. Descripción detallada

Operaciones básicas, implementación.

Este archivo contiene las implementaciones de las funciones que implementan operaciones básicas entre matrices y vectores utilizando estructuras de datos para matrices dispersas.

Definición en el archivo oper.c.

6.7.2. Documentación de las funciones

6.7.2.1. sp_cds * mult_mat_cds (sp_cds * *A*, sp_cds * *B*)

Multiplica dos matrices en formato CDS.

Parámetros:

A Matriz operando izquierdo de la multiplicación

B Matriz operando derecho de la multiplicación

Esta operación computa la operación $C = AB$ y devuelve un puntero a la matriz `C`.

La implementación de esta operación está dividida en dos etapas claramente diferenciadas:

1. Cálculo anticipado de la estructura de `C` (cuyo objetivo es equivalente al de una factorización simbólica)
2. Cálculo numérico de `C`

La lógica de ambas etapas está fundamentada en la misma observación que puede hacerse del algoritmo *"clásico"* de multiplicación de matrices. Recordando nuestra definición de diagonales (el elemento a_{ij} pertenece a la diagonal $j - i$) y observando el algoritmo clásico

$$c_{ij} = \sum_{k=0}^{n} a_{ik} b_{kj}$$

obtenemos las siguientes conclusiones:

Primero, toda diagonal `t` de `C` se verá afectada únicamente por productos de entradas de todo par de diagonales `r` y `s` de `A` y `B` respectivamente, tales que `t = r+s`. Para convencernos de esto, basta ver el algoritmo clásico. Vemos que el elemento c_{ij} (perteneciente a la diagonal `(j-i)`) se ve afectado por la multiplicación de $a_{ik}b_{kj}$ para todo `k`. Es decir, un elemento de la diagonal `(k-i)` de `A` y un elemento de la diagonal `(j-k)` de `B`. Por lo tanto, como tenemos que `(k-i)+(j-k) = j-i`, nuestro enunciado es correcto.

Esta observación nos da una pauta de cómo realizar ambas etapas del algoritmo. En particular, para la etapa de cálculo de la estructura del resultado `C`, podemos afirmar lo siguiente: Si la diagonales `r` de `A` y `s` de `B` son no vacías (entendiendo por diagonal vacía aquella con cero en todas sus entradas), entonces la diagonal `(r+s)` de `C` (si existe) es no vacía.

La acotación "si existe" del enunciado anterior es porque van a haber diagonales que no se van a cruzar en el producto. Basta ver cómo el algoritmo calcula las diagonales máxima y mínima (variables `mindiag` y `maxdiag`) antes de comenzar con el cálculo de la estructura.

Con esto tenemos resuelto el cálculo de la estructura de `C`. Observamos que también será una matriz de banda, pero más diagonales tendrán valores no cero.

Con respecto al cálculo de las entradas de `C`, hace falta aclarar cómo se multiplican las entradas de las diagonales de `A` y `B`. Ya vimos que un elemento de la diagonal de `A` se multiplicará con uno de `B`. Pero, ¿cuál con cuál?

Observamos, gracias al algoritmo "clásico", que todo producto de entradas de `A` y `B` siempre cumple que la columna de `A` se corresponde con la misma fila de `B` (ambas tienen el mismo índice `k` en el algoritmo). Por lo tanto, dadas dos diagonales `r` y `s` de `A` y `B` respectivamente, basta encontrar el primer par de elementos que cumplan con esta condición, para que los siguientes pares (avanzando un lugar en cada una de las diagonales) también lo cumpla. Esto es lo que hace el algoritmo en la etapa comentada como la "alineación" de las diagonales a multiplicar. Luego de esto, se hace el producto entrada por entrada.

Nota:

La etapa de cálculo de la estructura ejecuta en un tiempo que está en el orden *O(A.ndiag+B.ndiag)*, mientras que el cálculo numérico ejecuta en un tiempo que está en el orden *O(D(A)+D(B))*, siendo `D(X)` la cantidad de entradas (con valor cero o no cero) que componen todas las diagonales no vacías de `X`.

Definición en la línea 205 del archivo oper.c.

Hace referencia a BAL_ERROR, binary_search(), sp_cds::dx, insert_sorted(), sp_cds::maxdiaglength, sp_cds::ncol, sp_cds::ndiag, sp_cds::nrow, y sp_cds::val.

Referenciado por bal_mult_mat_cds().

```
206 {
207     unsigned int n, m, tope, dxl, ka, kb, i, j, k;
208     int mindiag, maxdiag, diaga, diagb, diag, inia, fina, inib, finb, ja, diff;
209     sp_cds* c;
210 
211     if (A->ncol != B->nrow) {
212         BAL_ERROR("Las dimensiones de las matrices no permiten su multiplicación");
213         return NULL;
214     }
215 
216     c = (sp_cds*)malloc(sizeof(sp_cds));
217     c->nrow = n = A->nrow;
218     c->ncol = m = B->ncol;
219     c->maxdiaglength = MIN(n, m);
220     tope = m + n - 1; /* Cantidad máxima de diagonales */
221     c->dx = (int*)malloc(sizeof(int) * tope);
222 
223     /* Calcula las diagonales que tendrán datos en el resultado */
224     mindiag = 1 - n;
225     maxdiag = m - 1;
226     dxl = 0;
227     for (ka=0; ka < A->ndiag; ++ka) {       /* Por cada diagonal con datos en A */
228         diaga = A->dx[ka];
229         for (kb=0; kb < B->ndiag; ++kb) {   /* Por cada diagonal con datos en B */
230             diagb = B->dx[kb];
231 
232             diag = diaga + diagb;           /* Diagonal en C que tendrá datos */
233             if (diag >= mindiag && diag <= maxdiag) {
234                 if (binary_search(c->dx, dxl, diag) == -1) {
235                     insert_sorted(c->dx, dxl - 1, diag);
236                     ++dxl;
237                 }
238             }
239         }
240     }
241 
242     /* Inicializa c->val */
243     c->ndiag = dxl;
244     c->val = (double**)malloc(sizeof(double*) * dxl);
245     for (i=0; i < dxl; ++i) {
246         c->val[i] = (double*)malloc(sizeof(double) * c->maxdiaglength);
247         for (j=0; j < c->maxdiaglength; ++j)
248             c->val[i][j] = 0;
249     }
250 
251     /* Calcula los valores de c */
252     for (ka=0; ka < A->ndiag; ++ka) {       /* Por cada diagonal con datos en A */
253         diaga = A->dx[ka];
254         for (kb=0; kb < B->ndiag; ++kb) {   /* Por cada diagonal con datos en B */
255             diagb = B->dx[kb];
256 
257             diag = diaga + diagb;
258             k = binary_search(c->dx, dxl, diag);
259 
260             inia = MAX(0, -diaga);
261             fina = A->maxdiaglength - MAX(0, diaga);
262             inib = MAX(0, -diagb);
263             finb = B->maxdiaglength - MAX(0, diagb);
264 
265             /* Alinea las diagonales para multiplicar */
266             ja = A->dx[ka] + inia;
267             diff = ja - inib;
```

```
268             if (diff > 0)
269                 inib += diff;
270             else
271                 inia -= diff;
272
273             for (i=inia,j=inib; i < fina && j < finb; ++i, ++j)
274                 c->val[k][i] += ( A->val[ka][i] * B->val[kb][j] );
275         }
276     }
277
278     return c;
279 }
```

6.7.2.2. void mult_vec_cds (sp_cds * *A*, double * *x*, double * *y*)

Multiplica una matriz en formato CDS por un vector.

Parámetros:

- ***A*** ENTRADA: Matriz dispersa en formato CDS
- ***x*** ENTRADA: Vector a multiplicar
- ***y*** SALIDA: Resultado de la multiplicación

Esta función realiza la operación $y = Ax$ siendo `A` una matriz en formato CDS.

Teniendo en cuenta que cada elemento de una diagonal de `A` solo afecta una entrada del resultado `y`, la estrategia es recorrer la matriz por diagonales (aprovechando la estructura CDS) y, para cada elemento de la diagonal, actualizar cada entrada de `y` según cómo la afecta el elemento procesado. Luego de recorrer todas las diagonales, el vector `y` contiene el resultado esperado.

Dado que una fila de `val` puede tener entradas que ni siquiera representan un elemento válido en `A`, es importante ver desde dónde y hasta dónde se recorre cada fila en `val`. Aprovechando la alineación que se le da a cada diagonal dentro de la estructura (a la izquierda si es superior y a la derecha si es inferior, ver la descripción de la estructura sp_cds) el algoritmo precalcula el rango de entradas donde hay datos válidos antes de iterar en los elementos de la diagonal, es decir, antes de comenzar con el loop interior.

Nota:

Es importante notar que las entradas del vector `y` se van construyendo a medida que se van recorriendo las diagonales de `A`, y no son construidas en un único paso, como es el caso del algoritmo del producto de matriz-vector clásico.
Este algoritmo ejecuta en un tiempo que está en el orden *O(D(A))*, siendo `D(A)` la cantidad de entradas (con valor cero o no cero) que componen todas las diagonales no vacías de `A`.

Atención:

Esta función no reserva memoria. El vector `y` ya debe estar inicializado en el tamaño correcto antes de llamar a esta función.

Definición en la línea 130 del archivo oper.c.

Hace referencia a sp_cds::dx, sp_cds::maxdiaglength, sp_cds::ndiag, sp_cds::nrow, y sp_cds::val.

Referenciado por bal_mult_vec_cds().

```
131 {
132     int i, j, k, diag, ini, fin;
```

```
133
134     /* Inicializa vector y */
135     for (j = 0; j < A->nrow; ++j)
136         y[j] = 0;
137
138     for (k=0; k < A->ndiag; ++k) {      /* Por cada diagonal no vacía en A */
139         diag = A->dx[k];
140
141         /* Calcula inicio y fin de la diagonal en la fila de val */
142         ini = MAX(0, -diag);
143         fin = A->maxdiaglength - MAX(0, diag);
144
145         for (i=ini; i < (A->maxdiaglength); ++i) {  /* Por cada elemento de la diagonal */
146             j = diag + i;
147             y[i] += ( A->val[k][i] * x[j] );
148         }
149     }
150 }
```

6.7.2.3. void mult_vec_packcol (sp_packcol * *A*, double * *x*, double * *y*)

Multiplica una matriz empaquetada por columna por un vector.

Parámetros:

- ***A*** ENTRADA: Matriz dispersa empaquetada por columna simétrica
- ***x*** ENTRADA: Vector a multiplicar
- ***y*** SALIDA: Resultado de la multiplicación

Esta función realiza la operación $y = Ax$ siendo `A` una matriz empaquetada por columna.

Nota:

La estructura en memoria es tal como la generada por la función coord2packcol()

Atención:

Esta función no reserva memoria. El vector `y` ya debe estar inicializado en el tamaño correcto antes de llamar a esta función.

Definición en la línea 32 del archivo oper.c.

Hace referencia a sp_packcol::colp, sp_packcol::ncol, sp_packcol::nrow, sp_packcol::rx, y sp_packcol::val.

Referenciado por bal_mult_vec_packcol().

```
33 {
34     int i, ii, j;
35
36     /* Inicializa vector y */
37     for (j = 0; j < A->nrow; ++j)
38         y[j] = 0;
39
40     for(j = 0; j < A->ncol; ++j) {
41         for(ii = A->colp[j]; ii < A->colp[j+1]; ++ii) {
42             i = A->rx[ii];
43             y[i] += ( x[j] * A->val[ii] );
44         }
45     }
46 }
```

6.7.2.4. void mult_vec_packcol_symmetric (sp_packcol * *A*, double * *x*, double * *y*)

Multiplica una matriz simétrica empaquetada por columna por un vector.

Parámetros:

A Matriz dispersa empaquetada por columna simétrica

x Vector a multiplicar

y Resultado de la multiplicación

Esta función calcula la operación $y = Ax$ bajo las siguientes consideraciones:

- La matriz A está guardada con la mejora para matrices simétricas, tal cual lo hace la función coord2packcol_symmetric()
- Los vectores `x` e `y` tienen `A.nrow` elementos (Ver definición de la estructura sp_packcol) y ya están inicializados

Nota:

Ver el algoritmo 5.1 (sección 5.2 Matrix-vector multiplication) en el paper de Stewart (vea las referencias).

Atención:

La implementación sugerida por Stewart tiene un bug: el algoritmo no devuelve el resultado correcto si la diagonal mayor de `A` contiene ceros. En esta implementación se corrigió esta falla. Se sugiere comparar las dos implementaciones para ver las diferencias.

Definición en la línea 70 del archivo oper.c.

Hace referencia a sp_packcol::colp, sp_packcol::ncol, sp_packcol::nrow, sp_packcol::rx, y sp_packcol::val.

Referenciado por bal_mult_vec_packcol_symmetric().

```
71 {
72     int i, ii, j, r;
73 
74     /* Inicializa vector y */
75     for (j = 0; j < A->nrow; ++j)
76         y[j] = 0;
77 
78     for (j = 0; j < A->ncol; ++j) {
79         i = A->rx[A->colp[j]];                      /* Indice de fila del 1er elem no cero de la column
80 
81         if (i == j) {                               /* Si el 1er no cero es la diag */
82             y[j] += ( x[j] * A->val[A->colp[j]] );  /* Procesar diagonal: y_j += x_j * A_jj */
83             r = 1;                                  /* Ignorar la diagonal en el loop interno */
84         }
85         else
86             r = 0;                                  /* El 1er elem no cero no es la diagonal. No ignora
87 
88         /* Procesa los elementos no en la diagonal utilizando la propiedad de simetria */
89         for (ii = A->colp[j] + r; ii <= A->colp[j+1] - 1; ++ii) {
90             i = A->rx[ii];                          /* Obtiene el indice de fila */
91             y[i] += ( x[j] * A->val[ii] );          /* y_i += x_j * A_ij */
92             y[j] += ( x[i] * A->val[ii] );          /* y_i += x_i * A_ji */
93         }
94     }
95 }
```

6.8. Referencia del Archivo oper.h

Operaciones básicas, archivo de cabecera.

```
#include "sparse/sp_packcol.h"
```

Funciones

- void mult_vec_packcol (sp_packcol *A, double *x, double *y)
  Multiplica una matriz empaquetada por columna por un vector.
- void mult_vec_packcol_symmetric (sp_packcol *A, double *x, double *y)
  Multiplica una matriz simétrica empaquetada por columna por un vector.
- void mult_vec_cds (sp_cds *A, double *x, double *y)
  Multiplica una matriz en formato CDS por un vector.
- sp_cds * mult_mat_cds (sp_cds *A, sp_cds *B)
  Multiplica dos matrices en formato CDS.

6.8.1. Descripción detallada

Operaciones básicas, archivo de cabecera.

Este archivo contiene las definiciones de las funciones que implementan operaciones básicas entre matrices y vectores utilizando estructuras de datos para matrices dispersas.

Definición en el archivo oper.h.

6.8.2. Documentación de las funciones

6.8.2.1. sp_cds* mult_mat_cds (sp_cds * *A*, sp_cds * *B*)

Multiplica dos matrices en formato CDS.

Parámetros:

- ***A*** Matriz operando izquierdo de la multiplicación
- ***B*** Matriz operando derecho de la multiplicación

Esta operación computa la operación $C = AB$ y devuelve un puntero a la matriz `C`.

La implementación de esta operación está dividida en dos etapas claramente diferenciadas:

1. Cálculo anticipado de la estructura de `C` (cuyo objetivo es equivalente al de una factorización simbólica)
2. Cálculo numérico de `C`

La lógica de ambas etapas está fundamentada en la misma observación que puede hacerse del algoritmo *"clásico"* de multiplicación de matrices. Recordando nuestra definición de diagonales (el elemento a_{ij}

pertenece a la diagonal $j - i$) y observando el algoritmo clásico

$$c_{ij} = \sum_{k=0}^{n} a_{ik} b_{kj}$$

obtenemos las siguientes conclusiones:

Primero, toda diagonal `t` de `C` se verá afectada únicamente por productos de entradas de todo par de diagonales `r` y `s` de `A` y `B` respectivamente, tales que `t = r+s`. Para convencernos de esto, basta ver el algoritmo clásico. Vemos que el elemento c_{ij} (perteneciente a la diagonal `(j-i)`) se ve afectado por la multiplicación de $a_{ik}b_{kj}$ para todo `k`. Es decir, un elemento de la diagonal `(k-i)` de `A` y un elemento de la diagonal `(j-k)` de `B`. Por lo tanto, como tenemos que `(k-i)+(j-k) = j-i`, nuestro enunciado es correcto.

Esta observación nos da una pauta de cómo realizar ambas etapas del algoritmo. En particular, para la etapa de cálculo de la estructura del resultado `C`, podemos afirmar lo siguiente: Si la diagonales `r` de `A` y `s` de `B` son no vacías (entendiendo por diagonal vacía aquella con cero en todas sus entradas), entonces la diagonal `(r+s)` de `C` (si existe) es no vacía.

La acotación "si existe" del enunciado anterior es porque van a haber diagonales que no se van a cruzar en el producto. Basta ver cómo el algoritmo calcula las diagonales máxima y mínima (variables `mindiag` y `maxdiag`) antes de comenzar con el cálculo de la estructura.

Con esto tenemos resuelto el cálculo de la estructura de `C`. Observamos que también será una matriz de banda, pero más diagonales tendrán valores no cero.

Con respecto al cálculo de las entradas de `C`, hace falta aclarar cómo se multiplican las entradas de las diagonales de `A` y `B`. Ya vimos que un elemento de la diagonal de `A` se multiplicará con uno de `B`. Pero, ¿cuál con cuál?

Observamos, gracias al algoritmo "clásico", que todo producto de entradas de `A` y `B` siempre cumple que la columna de `A` se corresponde con la misma fila de `B` (ambas tienen el mismo índice `k` en el algoritmo). Por lo tanto, dadas dos diagonales `r` y `s` de `A` y `B` respectivamente, basta encontrar el primer par de elementos que cumplan con esta condición, para que los siguientes pares (avanzando un lugar en cada una de las diagonales) también lo cumpla. Esto es lo que hace el algoritmo en la etapa comentada como la "alineación" de las diagonales a multiplicar. Luego de esto, se hace el producto entrada por entrada.

Nota:

La etapa de cálculo de la estructura ejecuta en un tiempo que está en el orden *O(A.ndiag+B.ndiag)*, mientras que el cálculo numérico ejecuta en un tiempo que está en el orden *O(D(A)+D(B))*, siendo `D(X)` la cantidad de entradas (con valor cero o no cero) que componen todas las diagonales no vacías de `X`.

Definición en la línea 205 del archivo oper.c.

Hace referencia a BAL_ERROR, binary_search(), sp_cds::dx, insert_sorted(), sp_cds::maxdiaglength, sp_cds::ncol, sp_cds::ndiag, sp_cds::nrow, y sp_cds::val.

Referenciado por bal_mult_mat_cds().

```
206 {
207     unsigned int n, m, tope, dxl, ka, kb, i, j, k;
208     int mindiag, maxdiag, diaga, diagb, diag, inia, fina, inib, finb, ja, diff;
209     sp_cds* c;
210 
211     if (A->ncol != B->nrow) {
212         BAL_ERROR("Las dimensiones de las matrices no permiten su multiplicación");
213         return NULL;
214     }
```

```
215
216     c = (sp_cds*)malloc(sizeof(sp_cds));
217     c->nrow = n = A->nrow;
218     c->ncol = m = B->ncol;
219     c->maxdiaglength = MIN(n, m);
220     tope = m + n - 1; /* Cantidad máxima de diagonales */
221     c->dx = (int*)malloc(sizeof(int) * tope);
222
223     /* Calcula las diagonales que tendrán datos en el resultado */
224     mindiag = 1 - n;
225     maxdiag = m - 1;
226     dxl = 0;
227     for (ka=0; ka < A->ndiag; ++ka) {       /* Por cada diagonal con datos en A */
228         diaga = A->dx[ka];
229         for (kb=0; kb < B->ndiag; ++kb) {   /* Por cada diagonal con datos en B */
230             diagb = B->dx[kb];
231
232             diag = diaga + diagb;           /* Diagonal en C que tendrá datos */
233             if (diag >= mindiag && diag <= maxdiag) {
234                 if (binary_search(c->dx, dxl, diag) == -1) {
235                     insert_sorted(c->dx, dxl - 1, diag);
236                     ++dxl;
237                 }
238             }
239         }
240     }
241
242     /* Inicializa c->val */
243     c->ndiag = dxl;
244     c->val = (double**)malloc(sizeof(double*) * dxl);
245     for (i=0; i < dxl; ++i) {
246         c->val[i] = (double*)malloc(sizeof(double) * c->maxdiaglength);
247         for (j=0; j < c->maxdiaglength; ++j)
248             c->val[i][j] = 0;
249     }
250
251     /* Calcula los valores de c */
252     for (ka=0; ka < A->ndiag; ++ka) {       /* Por cada diagonal con datos en A */
253         diaga = A->dx[ka];
254         for (kb=0; kb < B->ndiag; ++kb) {   /* Por cada diagonal con datos en B */
255             diagb = B->dx[kb];
256
257             diag = diaga + diagb;
258             k = binary_search(c->dx, dxl, diag);
259
260             inia = MAX(0, -diaga);
261             fina = A->maxdiaglength - MAX(0, diaga);
262             inib = MAX(0, -diagb);
263             finb = B->maxdiaglength - MAX(0, diagb);
264
265             /* Alinea las diagonales para multiplicar */
266             ja = A->dx[ka] + inia;
267             diff = ja - inib;
268             if (diff > 0)
269                 inib += diff;
270             else
271                 inia -= diff;
272
273             for (i=inia,j=inib; i < fina && j < finb; ++i, ++j)
274                 c->val[k][i] += ( A->val[ka][i] * B->val[kb][j] );
275         }
276     }
277
278     return c;
279 }
```

6.8.2.2. void mult_vec_cds (sp_cds * *A*, double * *x*, double * *y*)

Multiplica una matriz en formato CDS por un vector.

Parámetros:

A ENTRADA: Matriz dispersa en formato CDS

x ENTRADA: Vector a multiplicar

y SALIDA: Resultado de la multiplicación

Esta función realiza la operación $y = Ax$ siendo `A` una matriz en formato CDS.

Teniendo en cuenta que cada elemento de una diagonal de `A` solo afecta una entrada del resultado `y`, la estrategia es recorrer la matriz por diagonales (aprovechando la estructura CDS) y, para cada elemento de la diagonal, actualizar cada entrada de `y` según cómo la afecta el elemento procesado. Luego de recorrer todas las diagonales, el vector `y` contiene el resultado esperado.

Dado que una fila de `val` puede tener entradas que ni siquiera representan un elemento válido en `A`, es importante ver desde dónde y hasta dónde se recorre cada fila en `val`. Aprovechando la alineación que se le da a cada diagonal dentro de la estructura (a la izquierda si es superior y a la derecha si es inferior, ver la descripción de la estructura sp_cds) el algoritmo precalcula el rango de entradas donde hay datos válidos antes de iterar en los elementos de la diagonal, es decir, antes de comenzar con el loop interior.

Nota:

Es importante notar que las entradas del vector `y` se van construyendo a medida que se van recorriendo las diagonales de `A`, y no son construidas en un único paso, como es el caso del algoritmo del producto de matriz-vector clásico.
Este algoritmo ejecuta en un tiempo que está en el orden *O(D(A))*, siendo `D(A)` la cantidad de entradas (con valor cero o no cero) que componen todas las diagonales no vacías de `A`.

Atención:

Esta función no reserva memoria. El vector `y` ya debe estar inicializado en el tamaño correcto antes de llamar a esta función.

Definición en la línea 130 del archivo oper.c.

Hace referencia a sp_cds::dx, sp_cds::maxdiaglength, sp_cds::ndiag, sp_cds::nrow, y sp_cds::val.

Referenciado por bal_mult_vec_cds().

```
131 {
132     int i, j, k, diag, ini, fin;
133 
134     /* Inicializa vector y */
135     for (j = 0; j < A->nrow; ++j)
136         y[j] = 0;
137 
138     for (k=0; k < A->ndiag; ++k) {      /* Por cada diagonal no vacía en A */
139         diag = A->dx[k];
140 
141         /* Calcula inicio y fin de la diagonal en la fila de val */
142         ini = MAX(0, -diag);
143         fin = A->maxdiaglength - MAX(0, diag);
144 
145         for (i=ini; i < (A->maxdiaglength); ++i) {  /* Por cada elemento de la diagonal */
146             j = diag + i;
147             y[i] += ( A->val[k][i] * x[j] );
148         }
149     }
150 }
```

6.8.2.3. void mult_vec_packcol (sp_packcol * A, double * x, double * y)

Multiplica una matriz empaquetada por columna por un vector.

Parámetros:

A ENTRADA: Matriz dispersa empaquetada por columna simétrica

x ENTRADA: Vector a multiplicar

y SALIDA: Resultado de la multiplicación

Esta función realiza la operación $y = Ax$ siendo `A` una matriz empaquetada por columna.

Nota:

La estructura en memoria es tal como la generada por la función coord2packcol()

Atención:

Esta función no reserva memoria. El vector `y` ya debe estar inicializado en el tamaño correcto antes de llamar a esta función.

Definición en la línea 32 del archivo oper.c.

Hace referencia a sp_packcol::colp, sp_packcol::ncol, sp_packcol::nrow, sp_packcol::rx, y sp_packcol::val.

Referenciado por bal_mult_vec_packcol().

```
33 {
34     int i, ii, j;
35 
36     /* Inicializa vector y */
37     for (j = 0; j < A->nrow; ++j)
38         y[j] = 0;
39 
40     for(j = 0; j < A->ncol; ++j) {
41         for(ii = A->colp[j]; ii < A->colp[j+1]; ++ii) {
42             i = A->rx[ii];
43             y[i] += ( x[j] * A->val[ii] );
44         }
45     }
46 }
```

6.8.2.4. void mult_vec_packcol_symmetric (sp_packcol * A, double * x, double * y)

Multiplica una matriz simétrica empaquetada por columna por un vector.

Parámetros:

A Matriz dispersa empaquetada por columna simétrica

x Vector a multiplicar

y Resultado de la multiplicación

Esta función calcula la operación $y = Ax$ bajo las siguientes consideraciones:

- La matriz A está guardada con la mejora para matrices simétricas, tal cual lo hace la función coord2packcol_symmetric()

- Los vectores `x` e `y` tienen `A.nrow` elementos (Ver definición de la estructura sp_packcol) y ya están inicializados

Nota:

Ver el algoritmo 5.1 (sección 5.2 Matrix-vector multiplication) en el paper de Stewart (vea las referencias).

Atención:

La implementación sugerida por Stewart tiene un bug: el algoritmo no devuelve el resultado correcto si la diagonal mayor de `A` contiene ceros. En esta implementación se corrigió esta falla. Se sugiere comparar las dos implementaciones para ver las diferencias.

Definición en la línea 70 del archivo oper.c.

Hace referencia a sp_packcol::colp, sp_packcol::ncol, sp_packcol::nrow, sp_packcol::rx, y sp_packcol::val.

Referenciado por bal_mult_vec_packcol_symmetric().

```
71 {
72     int i, ii, j, r;
73 
74     /* Inicializa vector y */
75     for (j = 0; j < A->nrow; ++j)
76         y[j] = 0;
77 
78     for (j = 0; j < A->ncol; ++j) {
79         i = A->rx[A->colp[j]];                      /* Indice de fila del 1er elem no cero de la column
80 
81         if (i == j) {                               /* Si el 1er no cero es la diag */
82             y[j] += ( x[j] * A->val[A->colp[j]] );  /* Procesar diagonal: y_j += x_j * A_jj */
83             r = 1;                                  /* Ignorar la diagonal en el loop interno */
84         }
85         else
86             r = 0;                                  /* El 1er elem no cero no es la diagonal. No ignora
87 
88         /* Procesa los elementos no en la diagonal utilizando la propiedad de simetria */
89         for (ii = A->colp[j] + r; ii <= A->colp[j+1] - 1; ++ii) {
90             i = A->rx[ii];                          /* Obtiene el indice de fila */
91             y[i] += ( x[j] * A->val[ii] );          /* y_i += x_j * A_ij */
92             y[j] += ( x[i] * A->val[ii] );          /* y_i += x_i * A_ji */
93         }
94     }
95 }
```

6.9. Referencia del Archivo parser/matriz_parser.y

Archivo de definición del parser generado por bison.

```
#include <stdio.h>

#include <stdlib.h>

#include <glib.h>

#include <errno.h>

#include <string.h>
```

Funciones

- void yyerror (const char *archivo, double ***matriz, int *n, int *m, const char *msg)

  Función de manejo de errores para el parser generado por bison.

Variables

- filas **__pad0__**
- filas **gpointer**
- fila **__pad1__**
- * **val** = $1

6.9.1. Descripción detallada

Archivo de definición del parser generado por bison.

Este archivo es la entrada de bison, mediante el cual se genera el código C que implementa el parser que lee definiciones de matrices en formato matlab, produciendo el archivo `matriz_parser.tab.c`.

El parser está definido de forma que cargue en memoria la matriz parseada en formato de matrices convencional del lenguaje C mediante el puntero `double ***matriz`.

Definición en el archivo matriz_parser.y.

6.9.2. Documentación de las funciones

6.9.2.1. void yyerror (const char * *archivo*, double *** *matriz*, int * *n*, int * *m*, const char * *msg*)

Función de manejo de errores para el parser generado por bison.

Parámetros:

archivo Camino al archivo donde esta definida la matriz

matriz Puntero en donde se va a devolver la matriz parseada (en caso de recuperarse del error)

n Cantidad de filas de la matriz leída

m Cantidad de columnas de la matriz leída

msg Mensaje de error devuelto por el parser generado por bison

Definición en la línea 127 del archivo matriz_parser.y.

```
128 {
129     fprintf(stderr, "%s\n", msg);
130 }
```

6.10. Referencia del Archivo parser/matriz_scanner.lex

Archivo de definición del analizador lexicográfico generado por flex.

```
#include <glib.h>
#include "matriz_parser.tab.h"
```

Variables

- return **CERRADO**
- return **SEMICOLON**
- return **NUMERO**

6.10.1. Descripción detallada

Archivo de definición del analizador lexicográfico generado por flex.

Este archivo es la entrada de la herramienta `flex`, mediante la cual se genera el código C que implementa el analizador lexicográfico que reconoce los tokens necesarios para parsear una matriz en formato matlab.

Definición en el archivo matriz_scanner.lex.

6.11. Referencia del Archivo sparse/sp_cds.c

Archivo de implementación para matriz dispersa, formato simple.

```
#include <stdlib.h>
#include <stdio.h>
#include <glib.h>
#include "sp_cds.h"
#include "../utils.h"
```

Funciones

- sp_cds ∗ coord2cds (sp_coord ∗mat)

  Genera la matriz en formato CDS equivalente a la matriz mat en formato simple.

- void sp_imprimir_cds (FILE ∗fp, sp_cds ∗mat)

  Imprime la matriz guardada en formato simple por coordenadas en fp.

- void save_cds (FILE ∗fp, sp_cds ∗A)

  Imprime la matriz `A` *en formato matlab en el archivo* `fp`*.*

- void free_cds (sp_cds ∗A)

  Borra toda la memoria reservada por la matriz A.

6.11.1. Descripción detallada

Archivo de implementación para matriz dispersa, formato simple.

Este archivo contiene la implementación de las funciones de utilidad para el formato de matriz dispersa simple.

Definición en el archivo sp_cds.c.

6.11.2. Documentación de las funciones

6.11.2.1. sp_cds * coord2cds (sp_coord * *mat*)

Genera la matriz en formato CDS equivalente a la matriz mat en formato simple.

Parámetros:

mat Matriz dispersa en formato simple.

Devuelve:

Matriz equivalente en formato CDS.

Definición en la línea 19 del archivo sp_cds.c.

Hace referencia a binary_search(), sp_coord::cx, sp_cds::dx, insert_sorted(), sp_cds::maxdiaglength, sp_coord::ncol, sp_cds::ncol, sp_cds::ndiag, sp_coord::nnz, sp_coord::nrow, sp_cds::nrow, sp_coord::rx, sp_coord::val, y sp_cds::val.

Referenciado por bal_coord2cds().

```
20 {
21     unsigned int n, m, tope, ii, i, j, dxl;
22     int diag, k;
23     sp_cds *cds;
24 
25     cds = (sp_cds*)malloc(sizeof(sp_cds));
26 
27     cds->nrow = n = mat->nrow;
28     cds->ncol = m = mat->ncol;
29     cds->maxdiaglength = MIN(n, m);
30     tope = m + n - 1; /* Cantidad máxima de diagonales */
31     cds->dx = (int*)malloc(sizeof(int) * tope);
32 
33     /* Busca diagonales con datos */
34     dxl = 0;
35     for (ii=0; ii < mat->nnz; ++ii) {
36         i = mat->rx[ii];
37         j = mat->cx[ii];
38         diag = j - i;
39 
40         if (binary_search(cds->dx, dxl, diag) == -1) {
41             insert_sorted(cds->dx, dxl - 1, diag);
42             ++dxl;
43         }
44     }
45 
46     /* Inicializa cds->val */
47     cds->ndiag = dxl;
48     cds->val = (double**)malloc(sizeof(double*) * dxl);
49     for (i=0; i < dxl; ++i) {
50         cds->val[i] = (double*)malloc(sizeof(double) * cds->maxdiaglength);
51         for (j=0; j < cds->maxdiaglength; ++j)
52             cds->val[i][j] = 0;
53     }
54 
55     /* Carga los valores */
56     for (ii=0; ii < mat->nnz; ++ii) {
57         i = mat->rx[ii];
58         j = mat->cx[ii];
59         diag = j - i;
60 
61         k = binary_search(cds->dx, dxl, diag);
62         cds->val[k][i] = mat->val[ii];
```

```
63      }
64
65      return cds;
66  }
```

6.11.2.2. void free_cds (sp_cds * A)

Borra toda la memoria reservada por la matriz A.

Libera la memoria reservada por la estructura de datos sp_cds.

Definición en la línea 134 del archivo sp_cds.c.

Hace referencia a sp_cds::dx, sp_cds::ndiag, y sp_cds::val.

Referenciado por bal_free_cds().

```
135 {
136     unsigned int d;
137
138     for (d=0; d < A->ndiag; ++d)
139         free(A->val[d]);
140
141     free(A->val);
142     free(A->dx);
143     free(A);
144 }
```

6.11.2.3. void save_cds (FILE * *fp*, sp_cds * *A*)

Imprime la matriz `A` en formato matlab en el archivo `fp`.

Parámetros:

fp Archivo en donde imprimir

A Matriz a imprimir, en formato CDS

Esta función es útil para respaldar matrices.

Definición en la línea 104 del archivo sp_cds.c.

Hace referencia a binary_search(), sp_cds::dx, sp_cds::ncol, sp_cds::ndiag, sp_cds::nrow, y sp_cds::val.

Referenciado por bal_save_cds().

```
105 {
106     unsigned int i, j;
107     int diag, k;
108
109     fprintf(fp, "[\n");
110
111     for (i=0; i < A->nrow; ++i) {
112         for (j=0; j < A->ncol; ++j) {
113             diag = j - i;
114             k = binary_search(A->dx, A->ndiag, diag);
115             if (k != -1)
116                 fprintf(fp, " %g", A->val[k][i]);
117             else
118                 fprintf(fp, " 0");
119         }
120         if (i+1 < A->nrow)
```

```
121             fprintf(fp, ";\n");
122         else
123             fprintf(fp, "\n");
124     }
125
126     fprintf(fp, "]\n");
127 }
```

6.11.2.4. void sp_imprimir_cds (FILE * *fp*, sp_cds * *mat*)

Imprime la matriz guardada en formato simple por coordenadas en fp.

Parámetros:

fp Archivo en el cual se imprimirá la matriz

mat Matriz a imprimir en formato CDS

Definición en la línea 74 del archivo sp_cds.c.

Hace referencia a sp_cds::dx, sp_cds::maxdiaglength, sp_cds::ncol, sp_cds::ndiag, sp_cds::nrow, y sp_cds::val.

Referenciado por bal_imprimir_cds().

```
75 {
76     unsigned int i, j;
77
78     fprintf(fp, "Cantidad de filas: %d\n", mat->nrow);
79     fprintf(fp, "Cantidad de columnas: %d\n", mat->ncol);
80     fprintf(fp, "Cantidad de diagonales no-cero: %d\n", mat->ndiag);
81     fprintf(fp, "Largo maximo de diagonal: %d\n", mat->maxdiaglength);
82
83     fprintf(fp, "Mapeo de diagonales:");
84     for (i=0; i < mat->ndiag; ++i)
85         fprintf(fp, " %d", mat->dx[i]);
86
87     fprintf(fp, "\nValores:\n");
88     for (i=0; i < mat->ndiag; ++i) {
89         fprintf(fp, "val[%d,0:%d] =", i, mat->maxdiaglength-1);
90         for (j=0; j < mat->maxdiaglength; ++j)
91             fprintf(fp, " %g", mat->val[i][j]);
92         fprintf(fp, "\n");
93     }
94 }
```

6.12. Referencia del Archivo sparse/sp_cds.h

Archivo de cabecera para matriz dispersa, formato CDS (comprimido por diagonal).

```
#include <stdio.h>
```

```
#include "sp_coord.h"
```

Estructuras de datos

- struct sp_cds

  Estructura de matriz dispersa CDS.

Funciones

- sp_cds ∗ coord2cds (sp_coord ∗mat)

  Genera la matriz en formato CDS equivalente a la matriz mat en formato simple.

- void sp_imprimir_cds (FILE ∗fp, sp_cds ∗mat)

  Imprime la matriz guardada en formato simple por coordenadas en fp.

- void save_cds (FILE ∗fp, sp_cds ∗A)

  Imprime la matriz `A` *en formato matlab en el archivo* `fp`.

- void free_cds (sp_cds ∗A)

  Borra toda la memoria reservada por la matriz A.

6.12.1. Descripción detallada

Archivo de cabecera para matriz dispersa, formato CDS (comprimido por diagonal).

Este archivo contiene la definición de la estructura de datos mediante la cual se almacena una matriz dispersa según el formato comprimido por diagonal, conocido como CDS (por sus siglas en inglés *Compressed Diagonal Storage*).

Definición en el archivo sp_cds.h.

6.12.2. Documentación de las funciones

6.12.2.1. sp_cds∗ coord2cds (sp_coord ∗ *mat*)

Genera la matriz en formato CDS equivalente a la matriz mat en formato simple.

Parámetros:

mat Matriz dispersa en formato simple.

Devuelve:

Matriz equivalente en formato CDS.

Definición en la línea 19 del archivo sp_cds.c.

Hace referencia a binary_search(), sp_coord::cx, sp_cds::dx, insert_sorted(), sp_cds::maxdiaglength, sp_coord::ncol, sp_cds::ncol, sp_cds::ndiag, sp_coord::nnz, sp_coord::nrow, sp_cds::nrow, sp_coord::rx, sp_coord::val, y sp_cds::val.

Referenciado por bal_coord2cds().

```
20 {
21     unsigned int n, m, tope, ii, i, j, dxl;
22     int diag, k;
23     sp_cds *cds;
24 
25     cds = (sp_cds*)malloc(sizeof(sp_cds));
26 
27     cds->nrow = n = mat->nrow;
28     cds->ncol = m = mat->ncol;
```

```
29      cds->maxdiaglength = MIN(n, m);
30      tope = m + n - 1; /* Cantidad máxima de diagonales */
31      cds->dx = (int*)malloc(sizeof(int) * tope);
32
33      /* Busca diagonales con datos */
34      dxl = 0;
35      for (ii=0; ii < mat->nnz; ++ii) {
36          i = mat->rx[ii];
37          j = mat->cx[ii];
38          diag = j - i;
39
40          if (binary_search(cds->dx, dxl, diag) == -1) {
41              insert_sorted(cds->dx, dxl - 1, diag);
42              ++dxl;
43          }
44      }
45
46      /* Inicializa cds->val */
47      cds->ndiag = dxl;
48      cds->val = (double**)malloc(sizeof(double*) * dxl);
49      for (i=0; i < dxl; ++i) {
50          cds->val[i] = (double*)malloc(sizeof(double) * cds->maxdiaglength);
51          for (j=0; j < cds->maxdiaglength; ++j)
52              cds->val[i][j] = 0;
53      }
54
55      /* Carga los valores */
56      for (ii=0; ii < mat->nnz; ++ii) {
57          i = mat->rx[ii];
58          j = mat->cx[ii];
59          diag = j - i;
60
61          k = binary_search(cds->dx, dxl, diag);
62          cds->val[k][i] = mat->val[ii];
63      }
64
65      return cds;
66  }
```

6.12.2.2. void free_cds (sp_cds * *A*)

Borra toda la memoria reservada por la matriz A.

Libera la memoria reservada por la estructura de datos sp_cds.

Definición en la línea 134 del archivo sp_cds.c.

Hace referencia a sp_cds::dx, sp_cds::ndiag, y sp_cds::val.

Referenciado por bal_free_cds().

```
135 {
136     unsigned int d;
137
138     for (d=0; d < A->ndiag; ++d)
139         free(A->val[d]);
140
141     free(A->val);
142     free(A->dx);
143     free(A);
144 }
```

6.12.2.3. void save_cds (FILE * *fp*, sp_cds * *A*)

Imprime la matriz `A` en formato matlab en el archivo `fp`.

Parámetros:

fp Archivo en donde imprimir

A Matriz a imprimir, en formato CDS

Esta función es útil para respaldar matrices.

Definición en la línea 104 del archivo sp_cds.c.

Hace referencia a binary_search(), sp_cds::dx, sp_cds::ncol, sp_cds::ndiag, sp_cds::nrow, y sp_cds::val.

Referenciado por bal_save_cds().

```
105 {
106     unsigned int i, j;
107     int diag, k;
108 
109     fprintf(fp, "[\n");
110 
111     for (i=0; i < A->nrow; ++i) {
112         for (j=0; j < A->ncol; ++j) {
113             diag = j - i;
114             k = binary_search(A->dx, A->ndiag, diag);
115             if (k != -1)
116                 fprintf(fp, " %g", A->val[k][i]);
117             else
118                 fprintf(fp, " 0");
119         }
120         if (i+1 < A->nrow)
121             fprintf(fp, ";\n");
122         else
123             fprintf(fp, "\n");
124     }
125 
126     fprintf(fp, "]\n");
127 }
```

6.12.2.4. void sp_imprimir_cds (FILE * *fp*, sp_cds * *mat*)

Imprime la matriz guardada en formato simple por coordenadas en fp.

Parámetros:

fp Archivo en el cual se imprimirá la matriz

mat Matriz a imprimir en formato CDS

Definición en la línea 74 del archivo sp_cds.c.

Hace referencia a sp_cds::dx, sp_cds::maxdiaglength, sp_cds::ncol, sp_cds::ndiag, sp_cds::nrow, y sp_cds::val.

Referenciado por bal_imprimir_cds().

```
75 {
76     unsigned int i, j;
77 
78     fprintf(fp, "Cantidad de filas: %d\n", mat->nrow);
79     fprintf(fp, "Cantidad de columnas: %d\n", mat->ncol);
```

```
80      fprintf(fp, "Cantidad de diagonales no-cero: %d\n", mat->ndiag);
81      fprintf(fp, "Largo maximo de diagonal: %d\n", mat->maxdiaglength);
82
83      fprintf(fp, "Mapeo de diagonales:");
84      for (i=0; i < mat->ndiag; ++i)
85          fprintf(fp, " %d", mat->dx[i]);
86
87      fprintf(fp, "\nValores:\n");
88      for (i=0; i < mat->ndiag; ++i) {
89          fprintf(fp, "val[%d,0:%d] =", i, mat->maxdiaglength-1);
90          for (j=0; j < mat->maxdiaglength; ++j)
91              fprintf(fp, " %g", mat->val[i][j]);
92          fprintf(fp, "\n");
93      }
94  }
```

6.13. Referencia del Archivo sparse/sp_coord.c

Archivo de implementación para matriz dispersa, formato simple.

```
#include <stdlib.h>
#include <stdio.h>
#include "sp_coord.h"
#include "../utils.h"
```

Funciones

- sp_coord * mat2coord (int n, int m, double **mat)

  Genera la matriz dispersa equivalente a la matriz completa mat nxm.

- sp_coord * load_coord (FILE *fp)

  Lee una matriz en formato simple por coordenadas desde archivo.

- void save_coord (FILE *fp, sp_coord *A)

  Escribe en `fp` *la matriz* `A` *en un formato entendible por load_coord().*

- void sp_imprimir_coord (FILE *fp, sp_coord *mat)

  Imprime la matriz guardada en formato simple por coordenadas en fp.

- void free_coord (sp_coord *A)

  Borra toda la memoria reservada por la matriz A.

6.13.1. Descripción detallada

Archivo de implementación para matriz dispersa, formato simple.

Este archivo contiene la implementación de las funciones de utilidad para el formato de matriz dispersa simple.

Definición en el archivo sp_coord.c.

6.13.2. Documentación de las funciones

6.13.2.1. void free_coord (sp_coord * *A*)

Borra toda la memoria reservada por la matriz A.

Esta función libera toda la memoria reservada por la estructura de datos sp_coord.

Definición en la línea 206 del archivo sp_coord.c.

Hace referencia a sp_coord::cx, sp_coord::rx, y sp_coord::val.

Referenciado por bal_free_coord(), y load_coord().

```
207 {
208     free(A->rx);
209     free(A->cx);
210     free(A->val);
211     free(A);
212 }
```

6.13.2.2. sp_coord * load_coord (FILE * *fp*)

Lee una matriz en formato simple por coordenadas desde archivo.

Parámetros:

fp Archivo del cual leer la matriz

Devuelve:

La matriz leída en formato simple por coordenadas

Lee el archivo `fp` en busca de una definición de matriz en formato simple por coordenadas. En caso de encontrarlo, construye la estructura en memoria y la devuelve al finalizar.

El formato buscado es el siguiente:

1. El primer número leído indica la cantidad de filas de la matriz (`nrow`)
2. El segundo número leído indica la cantidad de columnas de la matriz (`ncol`)
3. El tercer número leído indica la cantidad de entradas no cero de la matriz (`nnz`)
4. Luego trata de leer `nnz` números enteros, que representan los índices de fila donde hay elementos no cero (`rx`)
5. Luego trata de leer `nnz` números enteros, que representan los índices de columna donde hay elementos no cero (`cx`)
6. Luego trata de leer `nnz` números reales, que representan los índices las entradas no cero de la matriz (`val`)

Todos los números pueden estar separados por cualquier cantidad de espacios, tabulaciones o fines de línea.

Nota:

La función save_coord() genera el formato esperado por esta función.

Definición en la línea 84 del archivo sp_coord.c.

Hace referencia a BAL_ERROR, sp_coord::cx, free_coord(), sp_coord::ncol, sp_coord::nnz, sp_coord::nrow, sp_coord::rx, y sp_coord::val.

Referenciado por bal_load_coord().

```
85 {
86     int x, i;
87     float y;
88     sp_coord *m;
89 
90     m = (sp_coord*)malloc(sizeof(sp_coord));
91 
92     if (fscanf(fp, " %d", &x) != 1) {
93         BAL_ERROR("No se pudo leer la cantidad de filas");
94         free(m);
95         return NULL;
96     }
97     m->nrow = x;
98 
99     if (fscanf(fp, " %d", &x) != 1) {
100         BAL_ERROR("No se pudo leer la cantidad de columnas");
101         free(m);
102         return NULL;
103     }
104     m->ncol = x;
105 
106     if (fscanf(fp, " %d", &x) != 1) {
107         BAL_ERROR("No se pudo leer la cantidad de elementos no cero");
108         free(m);
109         return NULL;
110     }
111     m->nnz = x;
112 
113     m->rx = (unsigned int*)malloc(sizeof(unsigned int) * x);
114     m->cx = (unsigned int*)malloc(sizeof(unsigned int) * x);
115     m->val = (double*)malloc(sizeof(double) * x);
116 
117     /* Lee los índices de fila */
118     for (i=0; i < m->nnz; ++i) {
119         if (fscanf(fp, " %d", &x) != 1) {
120             BAL_ERROR("No se pudo leer uno de los índices de fila");
121             free_coord(m);
122             return NULL;
123         }
124         m->rx[i] = (unsigned int)x;
125     }
126 
127     /* Lee los índices de columna */
128     for (i=0; i < m->nnz; ++i) {
129         if (fscanf(fp, " %d", &x) != 1) {
130             BAL_ERROR("No se pudo leer uno de los índices de columna");
131             free_coord(m);
132             return NULL;
133         }
134         m->cx[i] = (unsigned int)x;
135     }
136 
137     /* Lee las entradas de la matriz */
138     for (i=0; i < m->nnz; ++i) {
139         if (fscanf(fp, " %f", &y) != 1) {
140             BAL_ERROR("No se pudo leer una de las entradas de la matriz");
141             free_coord(m);
142             return NULL;
143         }
144         m->val[i] = (double)y;
```

```
145     }
146
147     return m;
148 }
```

6.13.2.3. sp_coord * mat2coord (int *n*, int *m*, double ** *mat*)

Genera la matriz dispersa equivalente a la matriz completa mat nxm.

Parámetros:

n Cantidad total de filas en mat

m Cantidad total de columnas en mat

mat Matriz completa en el formato clásico de C.

Definición en la línea 19 del archivo sp_coord.c.

Hace referencia a sp_coord::cx, sp_coord::ncol, sp_coord::nnz, sp_coord::nrow, sp_coord::rx, y sp_coord::val.

Referenciado por bal_mat2coord().

```
20 {
21     unsigned int i, j, count;
22     sp_coord *sp;
23
24     count = 0;
25     for(i=0; i < n; ++i) {
26         for(j=0; j < m; ++j) {
27             if (mat[i][j] != 0)
28                 ++count;
29         }
30     }
31
32     sp = (sp_coord*)malloc(sizeof(sp_coord));
33     sp->nrow = n;
34     sp->ncol = m;
35     sp->nnz = count;
36     sp->rx = NULL;
37     sp->cx = NULL;
38     sp->val = NULL;
39
40     if (count > 0) {
41         sp->rx = (unsigned int*)malloc(sizeof(unsigned int) * count);
42         sp->cx = (unsigned int*)malloc(sizeof(unsigned int) * count);
43         sp->val = (double*)malloc(sizeof(double) * count);
44
45         count = 0;
46         for(i=0; i < n; ++i) {
47             for(j=0; j < m; ++j) {
48                 if (mat[i][j] != 0) {
49                     sp->rx[count] = i;
50                     sp->cx[count] = j;
51                     sp->val[count] = mat[i][j];
52                     ++count;
53                 }
54             }
55         }
56     }
57
58     return sp;
59 }
```

6.13.2.4. void save_coord (FILE * *fp*, sp_coord * *A*)

Escribe en `fp` la matriz `A` en un formato entendible por load_coord().

Esta función es útil para respaldar matrices en formato simple por coordenadas. Una matriz guardada mediante esta función puede ser cargada nuevamente mediante load_coord().

Definición en la línea 156 del archivo sp_coord.c.

Hace referencia a sp_coord::cx, sp_coord::ncol, sp_coord::nnz, sp_coord::nrow, sp_coord::rx, y sp_coord::val.

Referenciado por bal_save_coord().

```
157 {
158     unsigned int i;
159 
160     fprintf(fp, "%d\n", A->nrow);
161     fprintf(fp, "%d\n", A->ncol);
162     fprintf(fp, "%d\n", A->nnz);
163 
164     for (i=0; i < A->nnz; ++i)
165         fprintf(fp, " %d", A->rx[i]);
166     fprintf(fp, "\n");
167 
168     for (i=0; i < A->nnz; ++i)
169         fprintf(fp, " %d", A->cx[i]);
170     fprintf(fp, "\n");
171 
172     for (i=0; i < A->nnz; ++i)
173         fprintf(fp, " %g", A->val[i]);
174     fprintf(fp, "\n");
175 }
```

6.13.2.5. void sp_imprimir_coord (FILE * *fp*, sp_coord * *mat*)

Imprime la matriz guardada en formato simple por coordenadas en fp.

Parámetros:

fp Archivo en el cual se imprimirá la matriz

mat Matriz a imprimir en formato simple por coordenadas

Definición en la línea 183 del archivo sp_coord.c.

Hace referencia a sp_coord::cx, sp_coord::ncol, sp_coord::nnz, sp_coord::nrow, sp_coord::rx, y sp_coord::val.

Referenciado por bal_imprimir_coord().

```
184 {
185     int i;
186 
187     if (mat == NULL) {
188         fprintf(fp, "Matriz nula\n");
189         return;
190     }
191 
192     fprintf(fp, "Cantidad de filas: %d\n", mat->nrow);
193     fprintf(fp, "Cantidad de columnas: %d\n", mat->ncol);
194     fprintf(fp, "Cantidad de elementos no cero: %d\n", mat->nnz);
195 
```

```
196     for(i=0; i < mat->nnz; ++i) {
197         fprintf(fp, "(%d) [%d,%d] = %g\n", i, mat->rx[i], mat->cx[i], mat->val[i]);
198     }
199 }
```

6.14. Referencia del Archivo sparse/sp_coord.h

Archivo de cabecera para matriz dispersa, formato simple.

`#include <stdio.h>`

Estructuras de datos

- struct sp_coord

  Estructura de matriz dispersa simple por coordenadas.

Funciones

- sp_coord * mat2coord (int n, int m, double **mat)

  Genera la matriz dispersa equivalente a la matriz completa mat nxm.

- sp_coord * load_coord (FILE *fp)

  Lee una matriz en formato simple por coordenadas desde archivo.

- void save_coord (FILE *fp, sp_coord *A)

  Escribe en `fp` *la matriz* `A` *en un formato entendible por load_coord().*

- void sp_imprimir_coord (FILE *fp, sp_coord *mat)

  Imprime la matriz guardada en formato simple por coordenadas en fp.

- void free_coord (sp_coord *A)

  Borra toda la memoria reservada por la matriz A.

6.14.1. Descripción detallada

Archivo de cabecera para matriz dispersa, formato simple.

Este archivo contiene la definición de la estructura de datos mediante la cual se almacena una matriz dispersa según el formato simple, representación por coordenadas.

Definición en el archivo sp_coord.h.

6.14.2. Documentación de las funciones

6.14.2.1. void free_coord (sp_coord * *A*)

Borra toda la memoria reservada por la matriz A.

Esta función libera toda la memoria reservada por la estructura de datos sp_coord.

Definición en la línea 206 del archivo sp_coord.c.

Hace referencia a sp_coord::cx, sp_coord::rx, y sp_coord::val.

Referenciado por bal_free_coord(), y load_coord().

```
207 {
208     free(A->rx);
209     free(A->cx);
210     free(A->val);
211     free(A);
212 }
```

6.14.2.2. sp_coord* load_coord (FILE * *fp*)

Lee una matriz en formato simple por coordenadas desde archivo.

Parámetros:

fp Archivo del cual leer la matriz

Devuelve:

La matriz leída en formato simple por coordenadas

Lee el archivo `fp` en busca de una definición de matriz en formato simple por coordenadas. En caso de encontrarlo, construye la estructura en memoria y la devuelve al finalizar.

El formato buscado es el siguiente:

1. El primer número leído indica la cantidad de filas de la matriz (`nrow`)
2. El segundo número leído indica la cantidad de columnas de la matriz (`ncol`)
3. El tercer número leído indica la cantidad de entradas no cero de la matriz (`nnz`)
4. Luego trata de leer `nnz` números enteros, que representan los índices de fila donde hay elementos no cero (`rx`)
5. Luego trata de leer `nnz` números enteros, que representan los índices de columna donde hay elementos no cero (`cx`)
6. Luego trata de leer `nnz` números reales, que representan los índices las entradas no cero de la matriz (`val`)

Todos los números pueden estar separados por cualquier cantidad de espacios, tabulaciones o fines de línea.

Nota:

La función save_coord() genera el formato esperado por esta función.

Definición en la línea 84 del archivo sp_coord.c.

Hace referencia a BAL_ERROR, sp_coord::cx, free_coord(), sp_coord::ncol, sp_coord::nnz, sp_coord::nrow, sp_coord::rx, y sp_coord::val.

Referenciado por bal_load_coord().

```
85 {
86     int x, i;
87     float y;
88     sp_coord *m;
89 
90     m = (sp_coord*)malloc(sizeof(sp_coord));
91 
92     if (fscanf(fp, " %d", &x) != 1) {
93         BAL_ERROR("No se pudo leer la cantidad de filas");
94         free(m);
95         return NULL;
96     }
97     m->nrow = x;
98 
99     if (fscanf(fp, " %d", &x) != 1) {
100         BAL_ERROR("No se pudo leer la cantidad de columnas");
101         free(m);
102         return NULL;
103     }
104     m->ncol = x;
105 
106     if (fscanf(fp, " %d", &x) != 1) {
107         BAL_ERROR("No se pudo leer la cantidad de elementos no cero");
108         free(m);
109         return NULL;
110     }
111     m->nnz = x;
112 
113     m->rx = (unsigned int*)malloc(sizeof(unsigned int) * x);
114     m->cx = (unsigned int*)malloc(sizeof(unsigned int) * x);
115     m->val = (double*)malloc(sizeof(double) * x);
116 
117     /* Lee los índices de fila */
118     for (i=0; i < m->nnz; ++i) {
119         if (fscanf(fp, " %d", &x) != 1) {
120             BAL_ERROR("No se pudo leer uno de los índices de fila");
121             free_coord(m);
122             return NULL;
123         }
124         m->rx[i] = (unsigned int)x;
125     }
126 
127     /* Lee los índices de columna */
128     for (i=0; i < m->nnz; ++i) {
129         if (fscanf(fp, " %d", &x) != 1) {
130             BAL_ERROR("No se pudo leer uno de los índices de columna");
131             free_coord(m);
132             return NULL;
133         }
134         m->cx[i] = (unsigned int)x;
135     }
136 
137     /* Lee las entradas de la matriz */
138     for (i=0; i < m->nnz; ++i) {
139         if (fscanf(fp, " %f", &y) != 1) {
140             BAL_ERROR("No se pudo leer una de las entradas de la matriz");
141             free_coord(m);
142             return NULL;
143         }
144         m->val[i] = (double)y;
145     }
146 
147     return m;
148 }
```

6.14.2.3. sp_coord* mat2coord (int *n*, int *m*, double ** *mat*)

Genera la matriz dispersa equivalente a la matriz completa mat nxm.

Parámetros:

n Cantidad total de filas en mat

m Cantidad total de columnas en mat

mat Matriz completa en el formato clásico de C.

Definición en la línea 19 del archivo sp_coord.c.

Hace referencia a sp_coord::cx, sp_coord::ncol, sp_coord::nnz, sp_coord::nrow, sp_coord::rx, y sp_coord::val.

Referenciado por bal_mat2coord().

```
20 {
21     unsigned int i, j, count;
22     sp_coord *sp;
23
24     count = 0;
25     for(i=0; i < n; ++i) {
26         for(j=0; j < m; ++j) {
27             if (mat[i][j] != 0)
28                 ++count;
29         }
30     }
31
32     sp = (sp_coord*)malloc(sizeof(sp_coord));
33     sp->nrow = n;
34     sp->ncol = m;
35     sp->nnz = count;
36     sp->rx = NULL;
37     sp->cx = NULL;
38     sp->val = NULL;
39
40     if (count > 0) {
41         sp->rx = (unsigned int*)malloc(sizeof(unsigned int) * count);
42         sp->cx = (unsigned int*)malloc(sizeof(unsigned int) * count);
43         sp->val = (double*)malloc(sizeof(double) * count);
44
45         count = 0;
46         for(i=0; i < n; ++i) {
47             for(j=0; j < m; ++j) {
48                 if (mat[i][j] != 0) {
49                     sp->rx[count] = i;
50                     sp->cx[count] = j;
51                     sp->val[count] = mat[i][j];
52                     ++count;
53                 }
54             }
55         }
56     }
57
58     return sp;
59 }
```

6.14.2.4. void save_coord (FILE * *fp*, sp_coord * *A*)

Escribe en `fp` la matriz `A` en un formato entendible por load_coord().

Esta función es útil para respaldar matrices en formato simple por coordenadas. Una matriz guardada mediante esta función puede ser cargada nuevamente mediante load_coord().

Definición en la línea 156 del archivo sp_coord.c.

Hace referencia a sp_coord::cx, sp_coord::ncol, sp_coord::nnz, sp_coord::nrow, sp_coord::rx, y sp_coord::val.

Referenciado por bal_save_coord().

```
157 {
158     unsigned int i;
159 
160     fprintf(fp, "%d\n", A->nrow);
161     fprintf(fp, "%d\n", A->ncol);
162     fprintf(fp, "%d\n", A->nnz);
163 
164     for (i=0; i < A->nnz; ++i)
165         fprintf(fp, " %d", A->rx[i]);
166     fprintf(fp, "\n");
167 
168     for (i=0; i < A->nnz; ++i)
169         fprintf(fp, " %d", A->cx[i]);
170     fprintf(fp, "\n");
171 
172     for (i=0; i < A->nnz; ++i)
173         fprintf(fp, " %g", A->val[i]);
174     fprintf(fp, "\n");
175 }
```

6.14.2.5. void sp_imprimir_coord (FILE * *fp*, sp_coord * *mat*)

Imprime la matriz guardada en formato simple por coordenadas en fp.

Parámetros:

fp Archivo en el cual se imprimirá la matriz

mat Matriz a imprimir en formato simple por coordenadas

Definición en la línea 183 del archivo sp_coord.c.

Hace referencia a sp_coord::cx, sp_coord::ncol, sp_coord::nnz, sp_coord::nrow, sp_coord::rx, y sp_coord::val.

Referenciado por bal_imprimir_coord().

```
184 {
185     int i;
186 
187     if (mat == NULL) {
188         fprintf(fp, "Matriz nula\n");
189         return;
190     }
191 
192     fprintf(fp, "Cantidad de filas: %d\n", mat->nrow);
193     fprintf(fp, "Cantidad de columnas: %d\n", mat->ncol);
194     fprintf(fp, "Cantidad de elementos no cero: %d\n", mat->nnz);
195 
196     for(i=0; i < mat->nnz; ++i) {
197         fprintf(fp, "(%d) [%d,%d] = %g\n", i, mat->rx[i], mat->cx[i], mat->val[i]);
198     }
199 }
```

6.15. Referencia del Archivo sparse/sp_packcol.c

Archivo de implementación para formato de matriz dispersa empaquetado por columnas.

```
#include <stdlib.h>
#include <stdio.h>
#include "sp_packcol.h"
#include "sp_coord.h"
#include "../utils.h"
```

Funciones

- sp_packcol * coord2packcol (sp_coord *mat)
  Genera una instancia de la matriz mat en formato empaquetado por columna.
- sp_packcol * coord2packcol_symmetric (sp_coord *mat)
  Genera una instancia de la matriz mat en formato empaquetado por columna para matrices simétricas.
- void sp_imprimir_packcol (FILE *fp, sp_packcol *mat)
  Imprime la matriz guardada en formato empaquetado por columna en `fp`.
- int row_traversal_packcol (sp_packcol *A, int *i, int *j, int *posij)
  Implementa un mecanismo eficiente para recorrer por filas una matriz dispersa empaquetada por columnas.
- void save_packcol_symmetric (FILE *fp, sp_packcol *A)
  Imprime la matriz simétrica empaquetada por columna en formato matlab en el archivo fp.
- void save_packcol (FILE *fp, sp_packcol *A)
  Imprime la matriz `A` *en formato matlab en el archivo* `fp`.
- void free_packcol (sp_packcol *A)
  Borra toda la memoria reservada por la matriz A.

6.15.1. Descripción detallada

Archivo de implementación para formato de matriz dispersa empaquetado por columnas.

Este archivo contiene la implementación de las funciones de utilidad para el formato de matriz dispersa empaquetado por columnas.

Definición en el archivo sp_packcol.c.

6.15.2. Documentación de las funciones

6.15.2.1. sp_packcol * coord2packcol (sp_coord * *mat*)

Genera una instancia de la matriz mat en formato empaquetado por columna.

Parámetros:

mat Matriz dispersa en formato simple por coordenadas

Definición en la línea 18 del archivo sp_packcol.c.

Hace referencia a sp_packcol::colp, sp_coord::cx, sp_coord::ncol, sp_packcol::ncol, sp_coord::nnz, sp_packcol::nnz, sp_coord::nrow, sp_packcol::nrow, sp_coord::rx, sp_packcol::rx, sp_coord::val, y sp_packcol::val.

Referenciado por bal_coord2packcol().

```
19 {
20     int i, j, col;
21     sp_packcol* packcol = (sp_packcol*)malloc(sizeof(sp_packcol));
22
23     packcol->nrow = mat->nrow;
24     packcol->ncol = mat->ncol;
25     packcol->nnz = mat->nnz;
26
27     if (packcol->nnz == 0) {
28         packcol->colp = NULL;
29         packcol->rx = NULL;
30         packcol->val = NULL;
31     }
32     else {
33         packcol->colp = (unsigned int*)malloc(sizeof(unsigned int) * (packcol->ncol+1));
34         packcol->rx = (unsigned int*)malloc(sizeof(unsigned int) * packcol->nnz);
35         packcol->val = (double*)malloc(sizeof(double) * packcol->nnz);
36
37         j = 0;
38         for(col=0; col < packcol->ncol; ++col) {    /*< Para cada columna col */
39             packcol->colp[col] = j;                 /*< El 1er elem de la columna col esta en la posici
40             for(i=0; i < mat->nnz; ++i) {           /*< Recorro los elementos no-cero de mat */
41                 if (mat->cx[i] == col) {            /*< Si es un elemento de la columna col */
42                     packcol->val[j] = mat->val[i];  /*< Guardo el valor */
43                     packcol->rx[j] = mat->rx[i];    /*< Guardo el numero de fila */
44                     ++j;
45                 }
46             }
47         }
48         packcol->colp[col] = j;                 /*< Elemento extra para indicar el fin de la ultima col
49     }
50
51     return packcol;
52 }
```

6.15.2.2. sp_packcol ∗ coord2packcol_symmetric (sp_coord ∗ *mat*)

Genera una instancia de la matriz mat en formato empaquetado por columna para matrices simétricas.

Parámetros:

mat Matriz dispersa en formato simple por coordenadas

En el caso de que mat sea una matriz simétrica, no es necesario guardar todos los elementos no cero. Basta con guardar los elementos no cero de una de las triangulares.

Esta función genera una representación de la matriz mat en formato empaquetado por columna, guardando en ella solo los elementos que están en la diagonal mayor y por debajo de ella, economizando memoria y sin perder información.

Por supuesto, esta función debe ser utilizada solo cuando mat es simétrica y se desea tener la ganancia de memoria. Las rutinas que utilicen la estructura generada por esta función deberán tener en cuenta sus características para realizar las tareas de forma correcta.

Definición en la línea 69 del archivo sp_packcol.c.

Hace referencia a BAL_ERROR, sp_packcol::colp, sp_coord::cx, sp_packcol::ncol, sp_coord::ncol, sp_packcol::nnz, sp_coord::nnz, sp_packcol::nrow, sp_coord::nrow, sp_coord::rx, sp_packcol::rx, sp_coord::val, y sp_packcol::val.

Referenciado por bal_coord2packcol_symmetric().

```
70 {
71     int i, j, col, nnz;
72     sp_packcol* packcol = (sp_packcol*)malloc(sizeof(sp_packcol));
73 
74     if (mat->nrow != mat->ncol) {
75         BAL_ERROR("La matriz debe ser cuadrada para poder ser simetrica.");
76         return NULL;
77     }
78 
79     packcol->nrow = mat->nrow;
80     packcol->ncol = mat->ncol;
81 
82     if (mat->nnz == 0) {
83         packcol->colp = NULL;
84         packcol->rx = NULL;
85         packcol->val = NULL;
86     }
87     else {
88         /* Calcula la cantidad de elementos no cero debajo y en la diagonal */
89         nnz = 0;
90         for(i=0; i < mat->nnz; ++i) {
91             if (mat->rx[i] >= mat->cx[i])
92                 ++nnz;
93         }
94 
95         packcol->nnz = nnz;
96         packcol->colp = (unsigned int*)malloc(sizeof(unsigned int) * (packcol->ncol+1));
97         packcol->rx = (unsigned int*)malloc(sizeof(unsigned int) * nnz);
98         packcol->val = (double*)malloc(sizeof(double) * nnz);
99 
100         j = 0;
101         for(col=0; col < packcol->ncol; ++col) {                    /*< Para cada columna col */
102             packcol->colp[col] = j;                                 /*< El 1er elem de la columna
103             for(i=0; i < mat->nnz; ++i) {                           /*< Recorro los elementos no-c
104                 if (mat->rx[i] >= mat->cx[i] && mat->cx[i] == col) {    /*< Si es un elemento de la co
105                     packcol->val[j] = mat->val[i];                  /*< Guardo el valor */
106                     packcol->rx[j] = mat->rx[i];                    /*< Guardo el numero de fila *
107                     ++j;
108                 }
109             }
110         }
111         packcol->colp[col] = j;                 /*< Elemento extra para indicar el fin de la ultima co
112     }
113 
114     return packcol;
115 }
```

6.15.2.3. void free_packcol (sp_packcol * A)

Borra toda la memoria reservada por la matriz A.

Esta función libera toda la memoria reservada por las estructura de datos sp_packcol.

Definición en la línea 348 del archivo sp_packcol.c.

Hace referencia a sp_packcol::colp, sp_packcol::rx, y sp_packcol::val.

Referenciado por bal_free_packcol().

```
349 {
350     free(A->colp);
351     free(A->rx);
352     free(A->val);
353     free(A);
354 }
```

6.15.2.4. int row_traversal_packcol (sp_packcol * *A*, int * *i*, int * *j*, int * *posij*)

Implementa un mecanismo eficiente para recorrer por filas una matriz dispersa empaquetada por columnas.

Parámetros:

A ENTRADA: Matriz a recorrer por filas en formato empaquetado por columna con mejora por simetría.

i ENTRADA/SALIDA: Si `i = -2`, la rutina es inicializada. En otro caso, `i` guarda el índice de la fila leída.

j SALIDA: Guarda el índice de la columna leída.

posij SALIDA: Indica la posición de `A[i,j]` en `A->val`.

Devuelve:

-1 si se llegó al final de una línea, `j` en caso contrario.

Dada la forma de guardar los datos en memoria de la estructura sp_packcol, no es trivial recorrer la matriz por filas de forma eficiente. Esta función implementa un mecanismo para ir obteniendo los valores de una sp_packcol por fila.

Atención:

Esté método solo es útil para matrices simétricas guardadas tal como lo hace coord2packcol_symmetric().
No se debe cambiar el valor de las variables `i` y `j` mientras se está recorriendo una matriz utilizando esta función.
Esta función devuelve todos los elementos de una fila `i` antes de devolver los elementos de la fila `i+1`, pero los elementos de una misma fila no son devueltos en ningún orden en particular (por ejemplo, ordenados por columna, que sería lo más natural). Más precisamente, la rutina siempre devuelve primero el elemento de la diagonal mayor `(i,i)`, (en caso que no sea cero), pero el resto de los elementos no tienen un orden particular definido.

Nota:

Este algoritmo brinda un método de recorrido por fila que presenta un tiempo de ejecución de *O(A.nnz)* y conlleva un costo extra en memoria de *2n* variables de tipo entero, siendo `n` la cantidad de filas y columnas de `A`.
Esta función está basada en la descripción de la sección 5.3 del paper de Stewart (Ver las referencias). Puede ver ese documento o el juego de rutinas de prueba de BAL por un ejemplo de cómo utilizar esta rutina.
Esta rutina utiliza memoria dinámica para trabajar. Puede llamar a esta función con `A = NULL` y `i = -2` para liberar la memoria utilizada. Note que esto es diferente a llamar la rutina para que se inicialice, en la que la misma reserva memoria para trabajar.

Definición en la línea 203 del archivo sp_packcol.c.

Hace referencia a sp_packcol::colp, sp_packcol::ncol, y sp_packcol::rx.

Referenciado por bal_row_traversal_packcol(), elimination_tree(), numerical_factorization(), y save_-packcol_symmetric().

```
204 {
205     static int *link = NULL;
206     static int *pos = NULL;
207     static int nextj = -1;
208     int x, nextdown, id;
209 
210     if (*i == -2) { /* Inicializacion */
211         if (link != NULL)
212             free(link);
213 
214         if (pos != NULL)
215             free(pos);
216 
217         if (A == NULL) {
218             link = pos = NULL;
219             return -1;
220         }
221 
222         link = (int*)malloc(sizeof(int) * A->ncol);
223         pos = (int*)malloc(sizeof(int) * A->ncol);
224         for (x = 0; x < A->ncol; ++x)
225             link[x] = pos[x] = -1;
226 
227         *i = *j = -1;
228         return -1;
229     }
230 
231     if (*j == -1) { /* Preparamos la fila i */
232         *i += 1;
233         *j = *i;
234         *posij = A->colp[*i];
235     }
236     else { /* Obtener el siguiente elemento de la fila i */
237         *j = nextj;
238 
239         if (*j == -1)
240             return *j;  /* Fin de fila */
241 
242         *posij = pos[*j];
243     }
244 
245     nextj = link[*j];
246     link[*j] = -1;
247     nextdown = *posij + 1;
248 
249     if (nextdown < A->colp[*j + 1]) {   /* Hay un elemento en la columna j, recordarlo */
250         pos[*j] = nextdown;
251         id = A->rx[nextdown];
252         link[*j] = link[id];
253         link[id] = *j;
254     }
255 
256     return *j;
257 }
```

6.15.2.5. void save_packcol (FILE * *fp*, sp_packcol * *A*)

Imprime la matriz `A` en formato matlab en el archivo `fp`.

Parámetros:

fp Puntero a archivo donde imprimir la matriz A.

A Matriz simétrica empaquetada por columna a imprimir en formato matlab.

Esta función es útil para respaldar matrices.

Definición en la línea 316 del archivo sp_packcol.c.

Hace referencia a sp_packcol::colp, sp_packcol::ncol, sp_packcol::nrow, sp_packcol::rx, y sp_packcol::val.

Referenciado por bal_save_packcol().

```
317 {
318     unsigned int i, j, k, encontrado;
319
320     fprintf(fp, "[\n");
321     for (i=0; i < A->nrow; ++i) {           /* Por cada fila */
322         for (j=0; j < A->ncol; ++j) {       /* Por cada columna */
323             /* Busca la entrada (i,j) en la columna j de A */
324             encontrado = 0;
325             for (k=A->colp[j]; k < A->colp[j+1]; ++k) {
326                 if (A->rx[k] == i) {
327                     encontrado = 1;
328                     fprintf(fp, " %g", A->val[k]);
329                     break;
330                 }
331             }
332             if (!encontrado)
333                 fprintf(fp, " 0");
334         }
335         if (i+1 < A->nrow)
336             fprintf(fp, ";\n");
337         else
338             fprintf(fp, "\n");
339     }
340     fprintf(fp, "]\n");
341 }
```

6.15.2.6. void save_packcol_symmetric (FILE * *fp*, sp_packcol * *A*)

Imprime la matriz simétrica empaquetada por columna en formato matlab en el archivo fp.

Parámetros:

fp Puntero a archivo donde imprimir la matriz A.

A Matriz simétrica empaquetada por columna a imprimir en formato matlab.

Esta función es útil para respaldar matrices.

Nota:

Esta función solo funciona para matrices empaquetadas por columna que fueron guardadas con la mejora para matrices simétricas, tal como lo hace la función coord2packcol_symmetric.

Atención:

La salida produce solo la triangular inferior de la matriz, para una salida completa, utilice save_packcol().

Definición en la línea 275 del archivo sp_packcol.c.

Hace referencia a sp_packcol::ncol, sp_packcol::nrow, row_traversal_packcol(), y sp_packcol::val.

Referenciado por bal_save_packcol_symmetric().

```
276 {
277     int x, y, i, j, posij;
278     double *fila;
279 
280     fila = (double*)malloc(sizeof(double) * A->ncol);
281 
282     fprintf(fp, "[\n");
283 
284     i = -2;
285     row_traversal_packcol(A, &i, &j, &posij);
286     for (x = 0; x < A->nrow; ++x) { /* Por cada fila */
287 
288         for (y=0; y < A->ncol; ++y)
289             fila[y] = 0;
290 
291         while(row_traversal_packcol(A, &i, &j, &posij) != -1)
292             fila[j] = A->val[posij];
293 
294         for (y=0; y < A->ncol; ++y)
295             fprintf(fp, " %g", fila[y]);
296 
297         if (x+1 < A->nrow)
298             fprintf(fp, ";\n");
299         else
300             fprintf(fp, "\n");
301     }
302 
303     fprintf(fp, "]\n");
304 
305     free(fila);
306 }
```

6.15.2.7. void sp_imprimir_packcol (FILE * *fp*, sp_packcol * *mat*)

Imprime la matriz guardada en formato empaquetado por columna en `fp`.

Parámetros:

fp Archivo en el cual se imprimirá la matriz

mat Matriz a imprimir en formato empaquetado por columna

NOTA: Ver el código (5.4) en el paper de Stewart (vea las referencias).

Definición en la línea 125 del archivo sp_packcol.c.

Hace referencia a sp_packcol::colp, sp_packcol::ncol, sp_packcol::nnz, sp_packcol::nrow, sp_packcol::rx, y sp_packcol::val.

Referenciado por bal_imprimir_packcol().

```
126 {
127     int i, j;
128 
129     if (mat == NULL) {
130         fprintf(fp, "Matriz nula.\n");
131         return;
132     }
```

```
133
134     fprintf(fp, "Cantidad de filas: %d\n", mat->nrow);
135     fprintf(fp, "Cantidad de columnas: %d\n", mat->ncol);
136     fprintf(fp, "Cantidad de elementos no cero: %d\n", mat->nnz);
137
138     fprintf(fp, "Inicios de columnas:\n");
139     for(i=0; i <= mat->ncol; ++i) {
140         fprintf(fp, "%d ", mat->colp[i]);
141     }
142     fprintf(fp, "\n");
143
144     fprintf(fp, "Indices de filas:\n");
145     for(i=0; i < mat->nnz; ++i) {
146         fprintf(fp, "%d ", mat->rx[i]);
147     }
148     fprintf(fp, "\n");
149
150     fprintf(fp, "Valores:\n");
151     for(i=0; i < mat->nnz; ++i) {
152         fprintf(fp, "%g ", mat->val[i]);
153     }
154     fprintf(fp, "\n");
155
156     fprintf(fp, "Salida indexada (por columna):\n");
157     i=0;
158     for(j=0; j < mat->ncol; ++j) {
159         for (i = mat->colp[j]; i < mat->colp[j+1]; ++i) {
160             fprintf(fp, "(%d) [%d,%d] = %g\n", i, mat->rx[i], j, mat->val[i]);
161         }
162     }
163 }
```

6.16. Referencia del Archivo sparse/sp_packcol.h

Archivo de cabecera para matriz dispersa, formato empaquetado por columna.

`#include <stdio.h>`

`#include "sp_coord.h"`

Estructuras de datos

- struct sp_packcol

  Estructura de matriz dispersa empaquetada por columna.

Funciones

- sp_packcol * coord2packcol (sp_coord *mat)

  Genera una instancia de la matriz mat en formato empaquetado por columna.

- sp_packcol * coord2packcol_symmetric (sp_coord *mat)

  Genera una instancia de la matriz mat en formato empaquetado por columna para matrices simétricas.

- void sp_imprimir_packcol (FILE *fp, sp_packcol *mat)

  Imprime la matriz guardada en formato empaquetado por columna en `fp`.

- int row_traversal_packcol (sp_packcol *A, int *i, int *j, int *posij)

Implementa un mecanismo eficiente para recorrer por filas una matriz dispersa empaquetada por columnas.

- void save_packcol_symmetric (FILE *fp, sp_packcol *A)

  Imprime la matriz simétrica empaquetada por columna en formato matlab en el archivo fp.

- void save_packcol (FILE *fp, sp_packcol *A)

  Imprime la matriz `A` *en formato matlab en el archivo* `fp`.

- void free_packcol (sp_packcol *A)

  Borra toda la memoria reservada por la matriz A.

6.16.1. Descripción detallada

Archivo de cabecera para matriz dispersa, formato empaquetado por columna.

Este archivo contiene la definición de la estructura de datos mediante la cual se almacena una matriz dispersa según el formato de empaquetado por columnas.

Definición en el archivo sp_packcol.h.

6.16.2. Documentación de las funciones

6.16.2.1. sp_packcol* coord2packcol (sp_coord * *mat*)

Genera una instancia de la matriz mat en formato empaquetado por columna.

Parámetros:

mat Matriz dispersa en formato simple por coordenadas

Definición en la línea 18 del archivo sp_packcol.c.

Hace referencia a sp_packcol::colp, sp_coord::cx, sp_coord::ncol, sp_packcol::ncol, sp_coord::nnz, sp_packcol::nnz, sp_coord::nrow, sp_packcol::nrow, sp_coord::rx, sp_packcol::rx, sp_coord::val, y sp_packcol::val.

Referenciado por bal_coord2packcol().

```
19 {
20     int i, j, col;
21     sp_packcol* packcol = (sp_packcol*)malloc(sizeof(sp_packcol));
22 
23     packcol->nrow = mat->nrow;
24     packcol->ncol = mat->ncol;
25     packcol->nnz = mat->nnz;
26 
27     if (packcol->nnz == 0) {
28         packcol->colp = NULL;
29         packcol->rx = NULL;
30         packcol->val = NULL;
31     }
32     else {
33         packcol->colp = (unsigned int*)malloc(sizeof(unsigned int) * (packcol->ncol+1));
34         packcol->rx = (unsigned int*)malloc(sizeof(unsigned int) * packcol->nnz);
35         packcol->val = (double*)malloc(sizeof(double) * packcol->nnz);
36 
37         j = 0;
```

```
38          for(col=0; col < packcol->ncol; ++col) {    /*< Para cada columna col */
39              packcol->colp[col] = j;                 /*< El 1er elem de la columna col esta en la posici
40              for(i=0; i < mat->nnz; ++i) {           /*< Recorro los elementos no-cero de mat */
41                  if (mat->cx[i] == col) {            /*< Si es un elemento de la columna col */
42                      packcol->val[j] = mat->val[i];  /*< Guardo el valor */
43                      packcol->rx[j] = mat->rx[i];    /*< Guardo el numero de fila */
44                      ++j;
45                  }
46              }
47          }
48          packcol->colp[col] = j;                 /*< Elemento extra para indicar el fin de la ultima col
49      }
50
51      return packcol;
52  }
```

6.16.2.2. sp_packcol* coord2packcol_symmetric (sp_coord * *mat*)

Genera una instancia de la matriz mat en formato empaquetado por columna para matrices simétricas.

Parámetros:

mat Matriz dispersa en formato simple por coordenadas

En el caso de que mat sea una matriz simétrica, no es necesario guardar todos los elementos no cero. Basta con guardar los elementos no cero de una de las triangulares.

Esta función genera una representación de la matriz mat en formato empaquetado por columna, guardando en ella solo los elementos que están en la diagonal mayor y por debajo de ella, economizando memoria y sin perder información.

Por supuesto, esta función debe ser utilizada solo cuando mat es simétrica y se desea tener la ganancia de memoria. Las rutinas que utilicen la estructura generada por esta función deberán tener en cuenta sus características para realizar las tareas de forma correcta.

Definición en la línea 69 del archivo sp_packcol.c.

Hace referencia a BAL_ERROR, sp_packcol::colp, sp_coord::cx, sp_packcol::ncol, sp_coord::ncol, sp_packcol::nnz, sp_coord::nnz, sp_packcol::nrow, sp_coord::nrow, sp_coord::rx, sp_packcol::rx, sp_coord::val, y sp_packcol::val.

Referenciado por bal_coord2packcol_symmetric().

```
70  {
71      int i, j, col, nnz;
72      sp_packcol* packcol = (sp_packcol*)malloc(sizeof(sp_packcol));
73
74      if (mat->nrow != mat->ncol) {
75          BAL_ERROR("La matriz debe ser cuadrada para poder ser simetrica.");
76          return NULL;
77      }
78
79      packcol->nrow = mat->nrow;
80      packcol->ncol = mat->ncol;
81
82      if (mat->nnz == 0) {
83          packcol->colp = NULL;
84          packcol->rx = NULL;
85          packcol->val = NULL;
86      }
87      else {
88          /* Calcula la cantidad de elementos no cero debajo y en la diagonal */
```

```
89          nnz = 0;
90          for(i=0; i < mat->nnz; ++i) {
91              if (mat->rx[i] >= mat->cx[i])
92                  ++nnz;
93          }
94
95          packcol->nnz = nnz;
96          packcol->colp = (unsigned int*)malloc(sizeof(unsigned int) * (packcol->ncol+1));
97          packcol->rx = (unsigned int*)malloc(sizeof(unsigned int) * nnz);
98          packcol->val = (double*)malloc(sizeof(double) * nnz);
99
100          j = 0;
101          for(col=0; col < packcol->ncol; ++col) {                        /*< Para cada columna col */
102              packcol->colp[col] = j;                                     /*< El 1er elem de la columna
103              for(i=0; i < mat->nnz; ++i) {                               /*< Recorro los elementos no-c
104                  if (mat->rx[i] >= mat->cx[i] && mat->cx[i] == col) {    /*< Si es un elemento de la co
105                      packcol->val[j] = mat->val[i];                      /*< Guardo el valor */
106                      packcol->rx[j] = mat->rx[i];                        /*< Guardo el numero de fila *
107                      ++j;
108                  }
109              }
110          }
111          packcol->colp[col] = j;                 /*< Elemento extra para indicar el fin de la ultima co
112      }
113
114      return packcol;
115  }
```

6.16.2.3. void free_packcol (sp_packcol * A)

Borra toda la memoria reservada por la matriz A.

Esta función libera toda la memoria reservada por las estructura de datos sp_packcol.

Definición en la línea 348 del archivo sp_packcol.c.

Hace referencia a sp_packcol::colp, sp_packcol::rx, y sp_packcol::val.

Referenciado por bal_free_packcol().

```
349 {
350     free(A->colp);
351     free(A->rx);
352     free(A->val);
353     free(A);
354 }
```

6.16.2.4. int row_traversal_packcol (sp_packcol * A, int * i, int * j, int * posij)

Implementa un mecanismo eficiente para recorrer por filas una matriz dispersa empaquetada por columnas.

Parámetros:

A ENTRADA: Matriz a recorrer por filas en formato empaquetado por columna con mejora por simetría.

i ENTRADA/SALIDA: Si `i = -2`, la rutina es inicializada. En otro caso, `i` guarda el índice de la fila leída.

j SALIDA: Guarda el índice de la columna leída.

posij SALIDA: Indica la posición de `A[i.j]` en `A->val`.

Devuelve:

-1 si se llegó al final de una línea, `j` en caso contrario.

Dada la forma de guardar los datos en memoria de la estructura sp_packcol, no es trivial recorrer la matriz por filas de forma eficiente. Esta función implementa un mecanismo para ir obteniendo los valores de una sp_packcol por fila.

Atención:

Esté método solo es útil para matrices simétricas guardadas tal como lo hace coord2packcol_-symmetric().
No se debe cambiar el valor de las variables `i` y `j` mientras se está recorriendo una matriz utilizando esta función.
Esta función devuelve todos los elementos de una fila `i` antes de devolver los elementos de la fila `i+1`, pero los elementos de una misma fila no son devueltos en ningún orden en particular (por ejemplo, ordenados por columna, que sería lo más natural). Más precisamente, la rutina siempre devuelve primero el elemento de la diagonal mayor `(i,i)`, (en caso que no sea cero), pero el resto de los elementos no tienen un orden particular definido.

Nota:

Este algoritmo brinda un método de recorrido por fila que presenta un tiempo de ejecución de *O(A.nnz)* y conlleva un costo extra en memoria de *2n* variables de tipo entero, siendo `n` la cantidad de filas y columnas de `A`.
Esta función está basada en la descripción de la sección 5.3 del paper de Stewart (Ver las referencias). Puede ver ese documento o el juego de rutinas de prueba de BAL por un ejemplo de cómo utilizar esta rutina.
Esta rutina utiliza memoria dinámica para trabajar. Puede llamar a esta función con `A = NULL` y `i = -2` para liberar la memoria utilizada. Note que esto es diferente a llamar la rutina para que se inicialice, en la que la misma reserva memoria para trabajar.

Definición en la línea 203 del archivo sp_packcol.c.

Hace referencia a sp_packcol::colp, sp_packcol::ncol, y sp_packcol::rx.

Referenciado por bal_row_traversal_packcol(), elimination_tree(), numerical_factorization(), y save_-packcol_symmetric().

```
204 {
205     static int *link = NULL;
206     static int *pos = NULL;
207     static int nextj = -1;
208     int x, nextdown, id;
209
210     if (*i == -2) { /* Inicializacion */
211         if (link != NULL)
212             free(link);
213
214         if (pos != NULL)
215             free(pos);
216
217         if (A == NULL) {
218             link = pos = NULL;
219             return -1;
220         }
221
222         link = (int*)malloc(sizeof(int) * A->ncol);
223         pos = (int*)malloc(sizeof(int) * A->ncol);
224         for (x = 0; x < A->ncol; ++x)
```

```
225             link[x] = pos[x] = -1;
226
227         *i = *j = -1;
228         return -1;
229     }
230
231     if (*j == -1) { /* Preparamos la fila i */
232         *i += 1;
233         *j = *i;
234         *posij = A->colp[*i];
235     }
236     else { /* Obtener el siguiente elemento de la fila i */
237         *j = nextj;
238
239         if (*j == -1)
240             return *j;  /* Fin de fila */
241
242         *posij = pos[*j];
243     }
244
245     nextj = link[*j];
246     link[*j] = -1;
247     nextdown = *posij + 1;
248
249     if (nextdown < A->colp[*j + 1]) {   /* Hay un elemento en la columna j, recordarlo */
250         pos[*j] = nextdown;
251         id = A->rx[nextdown];
252         link[*j] = link[id];
253         link[id] = *j;
254     }
255
256     return *j;
257 }
```

6.16.2.5. void save_packcol (FILE * *fp*, sp_packcol * *A*)

Imprime la matriz `A` en formato matlab en el archivo `fp`.

Parámetros:

fp Puntero a archivo donde imprimir la matriz A.

A Matriz simétrica empaquetada por columna a imprimir en formato matlab.

Esta función es útil para respaldar matrices.

Definición en la línea 316 del archivo sp_packcol.c.

Hace referencia a sp_packcol::colp, sp_packcol::ncol, sp_packcol::nrow, sp_packcol::rx, y sp_packcol::val.

Referenciado por bal_save_packcol().

```
317 {
318     unsigned int i, j, k, encontrado;
319
320     fprintf(fp, "[\n");
321     for (i=0; i < A->nrow; ++i) {           /* Por cada fila */
322         for (j=0; j < A->ncol; ++j) {       /* Por cada columna */
323             /* Busca la entrada (i,j) en la columna j de A */
324             encontrado = 0;
325             for (k=A->colp[j]; k < A->colp[j+1]; ++k) {
326                 if (A->rx[k] == i) {
327                     encontrado = 1;
```

```
328                     fprintf(fp, " %g", A->val[k]);
329                     break;
330                 }
331             }
332             if (!encontrado)
333                 fprintf(fp, " 0");
334         }
335         if (i+1 < A->nrow)
336             fprintf(fp, ";\n");
337         else
338             fprintf(fp, "\n");
339     }
340     fprintf(fp, "]\n");
341 }
```

6.16.2.6. void save_packcol_symmetric (FILE * *fp*, sp_packcol * *A*)

Imprime la matriz simétrica empaquetada por columna en formato matlab en el archivo fp.

Parámetros:

fp Puntero a archivo donde imprimir la matriz A.

A Matriz simétrica empaquetada por columna a imprimir en formato matlab.

Esta función es útil para respaldar matrices.

Nota:

Esta función solo funciona para matrices empaquetadas por columna que fueron guardadas con la mejora para matrices simétricas, tal como lo hace la función coord2packcol_symmetric.

Atención:

La salida produce solo la triangular inferior de la matriz, para una salida completa, utilice save_packcol().

Definición en la línea 275 del archivo sp_packcol.c.

Hace referencia a sp_packcol::ncol, sp_packcol::nrow, row_traversal_packcol(), y sp_packcol::val.

Referenciado por bal_save_packcol_symmetric().

```
276 {
277     int x, y, i, j, posij;
278     double *fila;
279
280     fila = (double*)malloc(sizeof(double) * A->ncol);
281
282     fprintf(fp, "[\n");
283
284     i = -2;
285     row_traversal_packcol(A, &i, &j, &posij);
286     for (x = 0; x < A->nrow; ++x) { /* Por cada fila */
287
288         for (y=0; y < A->ncol; ++y)
289             fila[y] = 0;
290
291         while(row_traversal_packcol(A, &i, &j, &posij) != -1)
292             fila[j] = A->val[posij];
293
```

```
294         for (y=0; y < A->ncol; ++y)
295             fprintf(fp, " %g", fila[y]);
296 
297         if (x+1 < A->nrow)
298             fprintf(fp, ";\n");
299         else
300             fprintf(fp, "\n");
301     }
302 
303     fprintf(fp, "]\n");
304 
305     free(fila);
306 }
```

6.16.2.7. void sp_imprimir_packcol (FILE * *fp*, sp_packcol * *mat*)

Imprime la matriz guardada en formato empaquetado por columna en `fp`.

Parámetros:

fp Archivo en el cual se imprimirá la matriz

mat Matriz a imprimir en formato empaquetado por columna

NOTA: Ver el código (5.4) en el paper de Stewart (vea las referencias).

Definición en la línea 125 del archivo sp_packcol.c.

Hace referencia a sp_packcol::colp, sp_packcol::ncol, sp_packcol::nnz, sp_packcol::nrow, sp_packcol::rx, y sp_packcol::val.

Referenciado por bal_imprimir_packcol().

```
126 {
127     int i, j;
128 
129     if (mat == NULL) {
130         fprintf(fp, "Matriz nula.\n");
131         return;
132     }
133 
134     fprintf(fp, "Cantidad de filas: %d\n", mat->nrow);
135     fprintf(fp, "Cantidad de columnas: %d\n", mat->ncol);
136     fprintf(fp, "Cantidad de elementos no cero: %d\n", mat->nnz);
137 
138     fprintf(fp, "Inicios de columnas:\n");
139     for(i=0; i <= mat->ncol; ++i) {
140         fprintf(fp, "%d ", mat->colp[i]);
141     }
142     fprintf(fp, "\n");
143 
144     fprintf(fp, "Indices de filas:\n");
145     for(i=0; i < mat->nnz; ++i) {
146         fprintf(fp, "%d ", mat->rx[i]);
147     }
148     fprintf(fp, "\n");
149 
150     fprintf(fp, "Valores:\n");
151     for(i=0; i < mat->nnz; ++i) {
152         fprintf(fp, "%g ", mat->val[i]);
153     }
154     fprintf(fp, "\n");
155 
156     fprintf(fp, "Salida indexada (por columna):\n");
```

```
157     i=0;
158     for(j=0; j < mat->ncol; ++j) {
159         for (i = mat->colp[j]; i < mat->colp[j+1]; ++i) {
160             fprintf(fp, "(%d) [%d,%d] = %g\n", i, mat->rx[i], j, mat->val[i]);
161         }
162     }
163 }
```

6.17. Referencia del Archivo utils.c

Implementación de utilidades generales.

```
#include "utils.h"
```

Funciones

- int binary_search (int *list, unsigned int size, int key)

  Busca un elemento en el array mediante bipartición.

- void insert_sorted (int *list, int n, int key)

  Inserta un elemento en un arreglo ordenado.

6.17.1. Descripción detallada

Implementación de utilidades generales.

Definición en el archivo utils.c.

6.17.2. Documentación de las funciones

6.17.2.1. int binary_search (int * *list*, unsigned int *size*, int *key*)

Busca un elemento en el array mediante bipartición.

Parámetros:

list Arreglo de elementos

size Tamaño de la lista

key Elemento a buscar

Devuelve:

El índice en donde se ubica el elemento en el arreglo, -1 si no fue encontrado

Pos más información ver http://es.wikipedia.org/wiki/B%C3%BAsqueda_dicot%C3%B3mica

Definición en la línea 18 del archivo utils.c.

Referenciado por coord2cds(), mult_mat_cds(), y save_cds().

```
19 {
20     int izq, der, medio;
```

```
21
22      izq = 0;
23      der = size - 1;
24
25      while (izq <= der)
26      {
27          medio = (int)((izq + der) / 2);
28          if (key == list[medio])
29              return medio;
30          else if (key > list[medio])
31              izq = medio + 1;
32          else
33              der = medio - 1;
34      }
35
36      return -1;
37  }
```

6.17.2.2. void insert_sorted (int * *list*, int *n*, int *key*)

Inserta un elemento en un arreglo ordenado.

Parámetros:

list Arreglo ordenado

n Posición del último elemento del arreglo

key elemento a insertar

Inserta un elemento en un array ordenado preservando el orden. Esta función corresponde al loop interno de un *insertion sort*. Por más información ver http://es.wikipedia.org/wiki/Ordenamiento_por_inserci%C3%B3n

Atención:

Esta función no solicita memoria. Se asume que la cantidad de memoria reservada es suficiente como para albergar al nuevo elemento.

Definición en la línea 55 del archivo utils.c.

Referenciado por coord2cds(), y mult_mat_cds().

```
56  {
57      int pos = n;
58
59      while (pos >= 0 && list[pos] > key) {
60          list[pos+1] = list[pos];
61          --pos;
62      }
63      list[pos+1] = key;
64  }
```

6.18. Referencia del Archivo utils.h

Archivo con definiciones y funciones de utilidad general.

```
#include <stdio.h>

#include "utils.h"
```

Definiciones

- #define BAL_ERROR(x) fprintf(stderr, "bal: %s: %d: %s\n", __FILE__, __LINE__, (x))
  Función para reportar errores detectados por código de BAL.

Funciones

- int binary_search (int *list, unsigned int size, int key)
  Busca un elemento en el array mediante bipartición.
- void insert_sorted (int *list, int n, int key)
  Inserta un elemento en un arreglo ordenado.

6.18.1. Descripción detallada

Archivo con definiciones y funciones de utilidad general.

Definición en el archivo utils.h.

6.18.2. Documentación de las definiciones

6.18.2.1. #define BAL_ERROR(x) fprintf(stderr, "bal: %s: %d: %s\n", __FILE__, __LINE__, (x))

Función para reportar errores detectados por código de BAL.

Esta macro es de uso interno de BAL y no está disponible desde código externo.

Definición en la línea 16 del archivo utils.h.

Referenciado por cholesky_solver(), coord2packcol_symmetric(), elimination_tree(), load_coord(), mult_-mat_cds(), y symbolic_factorization().

6.18.3. Documentación de las funciones

6.18.3.1. int binary_search (int * *list*, unsigned int *size*, int *key*)

Busca un elemento en el array mediante bipartición.

Parámetros:

list Arreglo de elementos

size Tamaño de la lista

key Elemento a buscar

Devuelve:

El índice en donde se ubica el elemento en el arreglo, -1 si no fue encontrado

Pos más información ver http://es.wikipedia.org/wiki/B%C3%BAsqueda_-dicot%C3%B3mica

Definición en la línea 18 del archivo utils.c.

Referenciado por coord2cds(), mult_mat_cds(), y save_cds().

```
19 {
20     int izq, der, medio;
21 
22     izq = 0;
23     der = size - 1;
24 
25     while (izq <= der)
26     {
27         medio = (int)((izq + der) / 2);
28         if (key == list[medio])
29             return medio;
30         else if (key > list[medio])
31             izq = medio + 1;
32         else
33             der = medio - 1;
34     }
35 
36     return -1;
37 }
```

6.18.3.2. void insert_sorted (int * *list*, int *n*, int *key*)

Inserta un elemento en un arreglo ordenado.

Parámetros:

list Arreglo ordenado

n Posición del último elemento del arreglo

key elemento a insertar

Inserta un elemento en un array ordenado preservando el orden. Esta función corresponde al loop interno de un *insertion sort*. Por más información ver http://es.wikipedia.org/wiki/Ordenamiento_por_inserci%C3%B3n

Atención:

Esta función no solicita memoria. Se asume que la cantidad de memoria reservada es suficiente como para albergar al nuevo elemento.

Definición en la línea 55 del archivo utils.c.

Referenciado por coord2cds(), y mult_mat_cds().

```
56 {
57     int pos = n;
58 
59     while (pos >= 0 && list[pos] > key) {
60         list[pos+1] = list[pos];
61         --pos;
62     }
63     list[pos+1] = key;
64 }
```

Índice alfabético