

# Implémentation d'une matrice CSR et utilisation de cette matrice pour la résolution de système linéaire

Généré par Doxygen 1.8.13



# Table des matières

<b>1</b>	<b>Utilisation de matrice CSR pour la résolution de <math>AX = B</math></b>	<b>1</b>
1.1	Présentation . . . . .	1
1.2	CSR . . . . .	1
1.3	LU . . . . .	1
1.4	Gradient Conjugué . . . . .	1
1.5	Solver . . . . .	2
1.6	TEST . . . . .	2
1.7	Compilation . . . . .	2
1.8	Exemple . . . . .	2
<b>2</b>	<b>Index des modules</b>	<b>5</b>
2.1	Liste des modules . . . . .	5
<b>3</b>	<b>Index du type de données</b>	<b>7</b>
3.1	Liste des types de données . . . . .	7
<b>4</b>	<b>Index des fichiers</b>	<b>9</b>
4.1	Liste des fichiers . . . . .	9

<b>5</b>	<b>Documentation des modules</b>	<b>11</b>
5.1	Référence du module <code>conjugate_gradient</code>	11
5.1.1	Description détaillée	11
5.1.2	Documentation de la fonction/subroutine	11
5.1.2.1	<code>conjugate_gradient_direct()</code>	11
5.2	Référence du module <code>csr</code>	12
5.2.1	Description détaillée	13
5.2.2	Documentation de la fonction/subroutine	13
5.2.2.1	<code>create_csr_matrix()</code>	14
5.2.2.2	<code>create_csr_matrix_from_file()</code>	15
5.2.2.3	<code>create_csr_matrix_recopy()</code>	15
5.2.2.4	<code>create_csr_matrix_square()</code>	16
5.2.2.5	<code>csr_matrix_additive_inverse()</code>	17
5.2.2.6	<code>csr_matrix_equal()</code>	17
5.2.2.7	<code>csr_matrix_inner_product()</code>	20
5.2.2.8	<code>csr_matrix_prod()</code>	21
5.2.2.9	<code>csr_matrix_save_to_file()</code>	22
5.2.2.10	<code>csr_matrix_scalar_prod()</code>	23
5.2.2.11	<code>csr_matrix_vector_add()</code>	23
5.2.2.12	<code>csr_matrix_vector_sub()</code>	24
5.2.2.13	<code>display()</code>	25
5.2.2.14	<code>display_debug()</code>	26
5.2.2.15	<code>free_csr_matrix()</code>	26
5.2.2.16	<code>get()</code>	27
5.2.2.17	<code>get_column_index()</code>	27
5.2.2.18	<code>get_column_value()</code>	28
5.2.2.19	<code>is_column_vector()</code>	29
5.2.2.20	<code>is_in_matrix()</code>	29
5.2.2.21	<code>set()</code>	30
5.2.2.22	<code>tr()</code>	31

5.3	Référence du module lu . . . . .	32
5.3.1	Description détaillée . . . . .	32
5.3.2	Documentation de la fonction/subroutine . . . . .	33
5.3.2.1	ln_csr_matrix() . . . . .	33
5.3.2.2	lu_csr_matrix_simple() . . . . .	33
5.4	Référence du module solver . . . . .	34
5.4.1	Description détaillée . . . . .	34
5.4.2	Documentation de la fonction/subroutine . . . . .	35
5.4.2.1	solver_backward_substitution() . . . . .	35
5.4.2.2	solver_forward_substitution() . . . . .	36
5.4.2.3	solver_gc() . . . . .	36
5.4.2.4	solver_lu() . . . . .	37
5.5	Référence du module unit_test . . . . .	38
5.5.1	Description détaillée . . . . .	39
5.5.2	Documentation de la fonction/subroutine . . . . .	39
5.5.2.1	test() . . . . .	39
5.5.2.2	test_inner_product_1() . . . . .	41
5.5.2.3	test_lu_1() . . . . .	41
5.5.2.4	test_lu_2() . . . . .	42
5.5.2.5	test_recopy() . . . . .	43
5.5.2.6	test_set_1() . . . . .	43
5.5.2.7	test_solver_backward_1() . . . . .	44
5.5.2.8	test_solver_forward_1() . . . . .	45
5.5.2.9	test_solver_lu_1() . . . . .	45
5.5.2.10	test_solver_lu_2() . . . . .	46
5.5.2.11	test_transpose_1() . . . . .	47
5.5.2.12	test_transpose_2() . . . . .	48

<b>6</b>	<b>Documentation du type de données</b>	<b>49</b>
6.1	Référence du type <code>csr</code> : <code>csr_matrix</code>	49
6.1.1	Description détaillée	49
6.1.2	Documentation des fonctions/subroutines membres	49
6.1.2.1	<code>csr_matrix_additive_inverse()</code>	50
6.1.2.2	<code>csr_matrix_save_to_file()</code>	50
6.1.2.3	<code>csr_matrix_scalar_prod()</code>	50
6.1.2.4	<code>display()</code>	50
6.1.2.5	<code>display_debug()</code>	50
6.1.2.6	<code>get()</code>	50
6.1.2.7	<code>get_column_index()</code>	50
6.1.2.8	<code>get_column_value()</code>	50
6.1.2.9	<code>is_column_vector()</code>	51
6.1.2.10	<code>is_in_matrix()</code>	51
6.1.2.11	<code>set()</code>	51
6.1.2.12	<code>tr()</code>	51
6.1.3	Documentation des données membres	51
6.1.3.1	<code>m_col</code>	51
6.1.3.2	<code>m_m</code>	51
6.1.3.3	<code>m_n</code>	52
6.1.3.4	<code>m_rowidx</code>	52
6.1.3.5	<code>m_val</code>	52
<b>7</b>	<b>Documentation des fichiers</b>	<b>53</b>
7.1	Référence du fichier <code>Source/CONJUGATE_GRADIENT.f90</code>	53
7.2	Référence du fichier <code>Source/CSR.f90</code>	53
7.3	Référence du fichier <code>Source/LU.f90</code>	54
7.4	Référence du fichier <code>Source/Projet.f90</code>	55
7.4.1	Documentation de la fonction/subroutine	55
7.4.1.1	<code>main()</code>	55
7.5	Référence du fichier <code>Source/SOLVER.f90</code>	56
7.6	Référence du fichier <code>Source/UNIT_TEST.f90</code>	57
	<b>Index</b>	<b>59</b>

# Chapitre 1

## Utilisation de matrice CSR pour la résolution de $AX = B$

### 1.1 Présentation

Le module CSR implémente un stockage efficient, en terme de mémoire, pour des matrices creuses. Dans un premier temps, on utilise la décomposition LU pour résoudre le problème. Dans un second temps, on utilise la méthode du gradient conjugué pour résoudre le problème.

### 1.2 CSR

```
TYPE csr_matrix
  INTEGER , DIMENSION(:), ALLOCATABLE :: m_rowidx
  INTEGER , DIMENSION(:), ALLOCATABLE :: m_col
  REAL(KIND=selected_real_kind(15, 307)) , DIMENSION(:), ALLOCATABLE :: m_val

  INTEGER :: m_m ! Number of row
  INTEGER :: m_n ! Number of column

END TYPE csr_matrix
```

L'implémentation du mode de stockage CSR est fait de tel sorte que rowidx commence par un 0 et que les colonnes sont indexés à partir de 1. Pour utiliser des matrices avec des matrices ayant des colonnes indexés à partir de 0, il suffit juste de rajouter 1 à m\_col. Ceci peut être mis en place très facilement grâce à une subfonction.

### 1.3 LU

On utilise la décomposition LU de la matrice A pour obtenir la solution. Cela revient, alors, à résoudre deux système triangulaire.

### 1.4 Gradient Conjugué

La méthode du gradient conjugué peut être utilisé de deux façon. Soit de manière complète, dans ce cas, l'intégralité des calculs sont fait pour obtenir la solution, soit de manière à rechercher une solution avec une précision donnée, dans ce cas, quand la précision de la solution est suffisante la fonction la retourne directement.

## 1.5 Solver

Ce module regroupe les appels pour résoudre un système  $AX = B$ . Les deux méthodes proposés sont LU et gradient conjugué.

## 1.6 TEST

Ce module implémente quelque test pour vérifier le fonctionnement basique de l'ensemble des modules. Ceci permet d'évaluer rapidement si l'ajout ou modification de fonction/subroutine aux différents modules modifie le comportement attendu.

## 1.7 Compilation

Pour compiler le code exécuter le script *compile.sh*.

## 1.8 Exemple

On importe, dans un premier temps les modules nécessaires.

```
USE csr
USE lu
USE conjugate_gradient
USE solver
USE unit_test
```

Dans cet exemple, on va créer une matrice **A** et un vecteur-colonne **B** charger depuis un fichier. On crée aussi un vecteur-colonne pour chaque solution X et Y.

```
TYPE(csr_matrix) a
TYPE(csr_matrix) b
TYPE(csr_matrix) x
TYPE(csr_matrix) y
TYPE(csr_matrix) r
```

On peut alors charger les matrices depuis les fichiers.

```
a = create_csr_matrix_from_file("A.csr")
b = create_csr_matrix_from_file("B.csr")
```

On va ensuite calculer la solution du système  $AX = B$  en utilisant les deux méthodes implémentées. X est la solution utilisant la décomposition LU tandis que Y est la solution utilisant la méthode du gradient conjugué.

```
x = solver_lu(a, b)
y = solver_gc(a, b)
```

On peut alors, afficher le résultat des deux méthodes.



```

WRITE (*,*) "Solution"
WRITE (*,*) "LU :"
CALL x%DISPLAY
WRITE (*,*) "GC :"
CALL y%DISPLAY

```

On peut alors, afficher l'erreur des deux méthodes.

```

WRITE (*,*) "Erreur"
r = csr_matrix_vector_sub(csr_matrix_prod(a,x), b)
WRITE (*,*) "LU :", sqrt(csr_matrix_inner_product(r,r))
r = csr_matrix_vector_sub(csr_matrix_prod(a,y), b)
WRITE (*,*) "GC :", sqrt(csr_matrix_inner_product(r,r))

```

On sauvegarde alors la matrice A dans le fichier **svg1**.

```

CALL a%CSR_MATRIX_SAVE_TO_FILE("A.csr")
CALL b%CSR_MATRIX_SAVE_TO_FILE("B.csr")

```

Il est possible d'initialiser à la main les matrices **A** et **B**. On doit alors créer une matrice vide de dimension voulue puis initialiser les valeurs non nulles

```

a = create_csr_matrix_square(3)
b = create_csr_matrix(3, 1)

CALL set(a, 1, 1, 5.d0)
CALL set(a, 2, 1, 1.d0)
CALL set(a, 2, 2, 1.d0)
CALL set(a, 2, 3, 8.d0)
CALL set(a, 3, 1, 7.d0)
CALL set(a, 3, 2, 1.d0)
CALL set(a, 3, 3, 2.d0)
CALL set(a, 3, 2, 10.d0)

CALL set(b, 1, 1, 1.d0)
CALL set(b, 2, 1, 1.d0)
CALL set(b, 3, 1, 1.d0)

```

Il est aussi possible de tester le fonctionnement basique des modules.

```

CALL test

```

A la fin, on doit désallouer les matrices utilisés.

```

CALL free_csr_matrix(a)
CALL free_csr_matrix(b)
CALL free_csr_matrix(x)
CALL free_csr_matrix(y)
CALL free_csr_matrix(r)

```



## Chapitre 2

# Index des modules

### 2.1 Liste des modules

Liste de tous les modules avec une brève description :

<a href="#">conjugate_gradient</a>	Ce module permet d'utiliser l'algorithme du gradient conjugué. Deux implémentations sont fournies, la première effectue le nombre maximal d'étape, la seconde s'arrête à une précision donnée	11
<a href="#">csr</a>	Ce module permet de créer, stocker et manipuler une matrice sous la forme CSR. Plusieurs opérations courantes sont implémentées, avec certains raffinements pour le cas de vecteur	12
<a href="#">lu</a>	Ce module permet d'obtenir la décomposition LU d'une matrice	32
<a href="#">solver</a>	Ce module permet de résoudre des systèmes linéaires en utilisant soit la décomposition LU, soit la méthode du gradient conjugué	34
<a href="#">unit_test</a>	Ce module contient plusieurs tests permettant d'assurer le bon fonctionnement des modules CSR, LU, CONJUGATE_GRADIENT et SOLVER	38



## Chapitre 3

# Index du type de données

### 3.1 Liste des types de données

Liste des types de données avec une brève description :

`csr : :csr_matrix`

Type décrivant une matrice CSR Attention `m_col` stocke l'index des colonnes en commençant par 1. [CSR] . . . . .

49



## Chapitre 4

# Index des fichiers

### 4.1 Liste des fichiers

Liste de tous les fichiers avec une brève description :

Source/ <a href="#">CONJUGATE_GRADIENT.f90</a> . . . . .	53
Source/ <a href="#">CSR.f90</a> . . . . .	53
Source/ <a href="#">LU.f90</a> . . . . .	54
Source/ <a href="#">Projet.f90</a> . . . . .	55
Source/ <a href="#">SOLVER.f90</a> . . . . .	56
Source/ <a href="#">UNIT_TEST.f90</a> . . . . .	57





## Chapitre 5

# Documentation des modules

### 5.1 Référence du module `conjugate_gradient`

Ce module permet d'utiliser l'algorithme du gradient conjugué. Deux implémentations sont fournies, la première effectue le nombre maximal d'étape, la seconde s'arrête à une précision donnée.

#### Fonctions/Subroutines

- `type(csr_matrix)` fonction `conjugate_gradient_direct` (`A`, `B`, `eps`)  
*Résouds le système  $AX = B$  en utilisant la méthode du gradient conjugué* Source [https://www.ljll.math.upmc.fr/hecht/ftp/InfoSci/doc-pdf/Master\\_book\\_GC.pdf](https://www.ljll.math.upmc.fr/hecht/ftp/InfoSci/doc-pdf/Master_book_GC.pdf).

#### 5.1.1 Description détaillée

Ce module permet d'utiliser l'algorithme du gradient conjugué. Deux implémentations sont fournies, la première effectue le nombre maximal d'étape, la seconde s'arrête à une précision donnée.

#### Auteur

Mechineau Alexandre

#### 5.1.2 Documentation de la fonction/subroutine

##### 5.1.2.1 `conjugate_gradient_direct()`

```
type(csr_matrix) function conjugate_gradient::conjugate_gradient_direct (  
    type(csr_matrix), intent(in) A,  
    type(csr_matrix), intent(in) B,  
    real, intent(in), optional eps )
```

Résouds le système  $AX = B$  en utilisant la méthode du gradient conjugué Source [https://www.ljll.math.upmc.fr/hecht/ftp/InfoSci/doc-pdf/Master\\_book\\_GC.pdf](https://www.ljll.math.upmc.fr/hecht/ftp/InfoSci/doc-pdf/Master_book_GC.pdf).

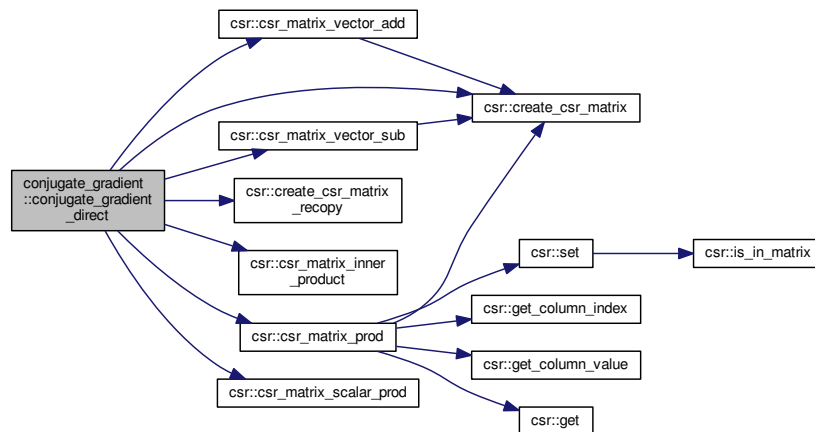
## Paramètres

<i>A</i>	Matrice CSR
<i>B</i>	Matrice CSR représentant un vecteur colonne
<i>eps</i>	Précision requise (Optionel)

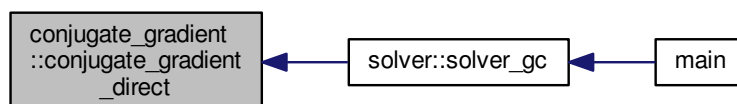
## Renvoie

Approximation, dans le pire des cas, de la solution du système

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



## 5.2 Référence du module csr

Ce module permet de créer, stocker et manipuler une matrice sous la forme CSR. Plusieurs opérations courantes sont implémentées, avec certains raffinement pour le cas de vecteur.

### Les types de données

— type `csr_matrix`

Type décrivant une matrice CSR Attention `m_col` stocke l'index des colonnes en commençant par 1. [CSR].

## Fonctions/Subroutines

- type([csr\\_matrix](#)) fonction [create\\_csr\\_matrix\\_from\\_file](#) (f)  
*Cette fonction permet de créer une matrice depuis un fichier. La matrice dans le fichier doit utiliser le style C-CSR modifié pour que les colonnes commencent par être indexées par 1. **On admet que le fichier passé est valide***
- subroutine [csr\\_matrix\\_save\\_to\\_file](#) (M, f)  
*Cette subroutine permet de sauvegarder une matrice CSR dans un fichier.*
- type([csr\\_matrix](#)) fonction [create\\_csr\\_matrix\\_square](#) (n)  
*Cette fonction permet de créer une matrice carrée vide.*
- type([csr\\_matrix](#)) fonction [create\\_csr\\_matrix\\_recopy](#) (M)  
*Cette fonction permet de dupliquer une matrice.*
- type([csr\\_matrix](#)) fonction [create\\_csr\\_matrix](#) (m, n)  
*Cette fonction crée une matrice de taille quelconque.*
- subroutine [free\\_csr\\_matrix](#) (M)  
*Cette subroutine permet de détruire une matrice CSR.*
- logical fonction [is\\_in\\_matrix](#) (M, row, col)  
*Cette fonction permet de vérifier que des indices sont bien valides pour une matrice donnée.*
- logical fonction [is\\_column\\_vector](#) (V)  
*Cette fonction permet de vérifier si une matrice est un vecteur colonne.*
- real(kind=selected\_real\_kind(15, 307)) fonction [get](#) (M, row, col)  
*Cette fonction permet d'obtenir la valeur d'un élément d'une matrice.*
- subroutine [set](#) (M, row, col, val)  
*Cette subroutine permet de modifier une valeur dans une matrice CSR.*
- subroutine [display\\_debug](#) (M)  
*Cette subroutine affiche une matrice CSR avec des informations avancées.*
- subroutine [display](#) (M)  
*Cette subroutine affiche une matrice CSR.*
- integer fonction, dimension( : ), allocatable [get\\_column\\_index](#) (M, row)  
*Cette fonction permet d'obtenir les colonnes non-nulles d'une ligne donnée.*
- real(kind=selected\_real\_kind(15, 307)) fonction, dimension( : ), allocatable [get\\_column\\_value](#) (M, row)  
*Cette fonction permet d'obtenir les valeurs non-nulles d'une ligne donnée.*
- type([csr\\_matrix](#)) fonction [csr\\_matrix\\_prod](#) (A, B)  
*Cette fonction calcule le produit de deux matrices.*
- logical fonction [csr\\_matrix\\_equal](#) (A, B)  
*Cette fonction permet de vérifier si deux matrices sont égales.*
- type([csr\\_matrix](#)) fonction [tr](#) (M)  
*Cette fonction calcule la transposée d'une matrice.*
- real(kind=selected\_real\_kind(15, 307)) fonction [csr\\_matrix\\_inner\\_product](#) (A, B)  
*Cette fonction calcule le produit scalaire de deux vecteurs.*
- type([csr\\_matrix](#)) fonction [csr\\_matrix\\_vector\\_add](#) (A, B)  
*Cette fonction calcule la somme de deux vecteurs colonnes.*
- type([csr\\_matrix](#)) fonction [csr\\_matrix\\_vector\\_sub](#) (A, B)  
*Cette fonction calcule la différence de deux vecteurs colonnes.*
- type([csr\\_matrix](#)) fonction [csr\\_matrix\\_additive\\_inverse](#) (A)  
*Cette fonction calcule l'inverse associé à la loi additive d'une matrice.*
- type([csr\\_matrix](#)) fonction [csr\\_matrix\\_scalar\\_prod](#) (M, K)  
*Cette fonction calcule le produit entre un réel et une matrice.*

## 5.2.1 Description détaillée

Ce module permet de créer, stocker et manipuler une matrice sous la forme CSR. Plusieurs opérations courantes sont implémentées, avec certains raffinements pour le cas de vecteur.

## Auteur

Mechineau Alexandre

## 5.2.2 Documentation de la fonction/subroutine



## 5.2.2.2 create\_csr\_matrix\_from\_file()

```
type (csr_matrix) function csr::create_csr_matrix_from_file (  
    character(len=*), intent(in) f )
```

Cette fonction permet de créer une matrice depuis un fichier. La matrice dans le fichier doit utiliser le style C-CSR modifié pour que les colonnes commencent à être indexées par 1. **On admet que le fichier passé est valide**

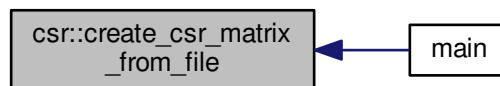
## Paramètres

<i>f</i>	Fichier stockant une matrice CSR
----------	----------------------------------

## Renvoie

Matrice CSR créée depuis le fichier

Voici le graphe des appelants de cette fonction :



## 5.2.2.3 create\_csr\_matrix\_recopy()

```
type (csr_matrix) function csr::create_csr_matrix_recopy (  
    type (csr_matrix), intent(in) M )
```

Cette fonction permet de dupliquer une matrice.

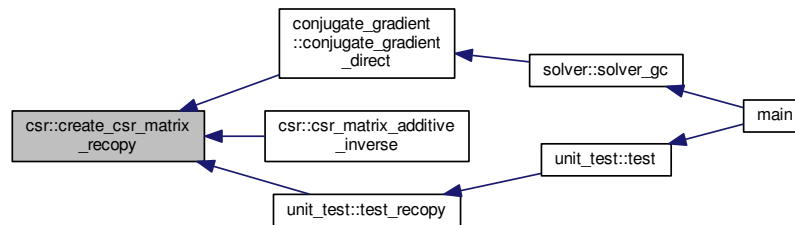
## Paramètres

<i>M</i>	Une matrice CSR que l'on souhaite recopier
----------	--

**Renvoie**

Une nouvelle matrice CSR avec les mêmes valeurs que la matrice passée en argument

Voici le graphe des appelants de cette fonction :

**5.2.2.4 create\_csr\_matrix\_square()**

```
type (csr_matrix) function csr::create_csr_matrix_square (
    integer n )
```

Cette fonction permet de créer une matrice carrée vide.

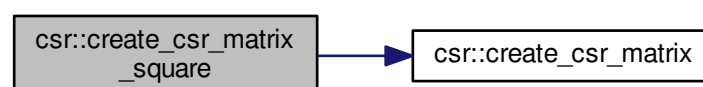
**Paramètres**

<i>n</i>	Taille d'un côté d'une matrice carrée
----------	---------------------------------------

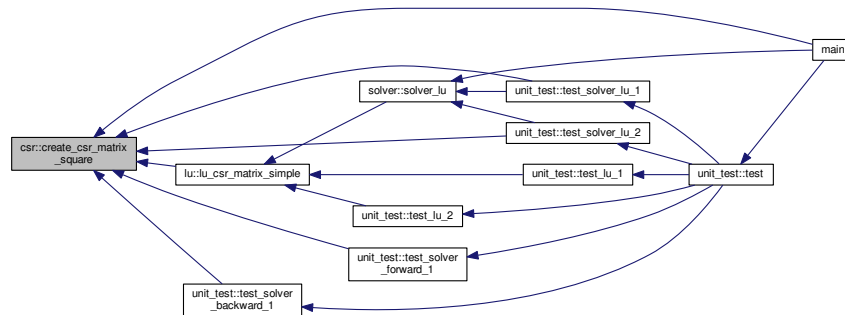
**Renvoie**

Matrice carrée de taille  $n * n$

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



### 5.2.2.5 csr\_matrix\_additive\_inverse()

```

type(csr_matrix) function csr::csr_matrix_additive_inverse (
    class(csr_matrix), intent(in) A )

```

Cette fonction calcule l'inverse associé à la loi additive d'une matrice.

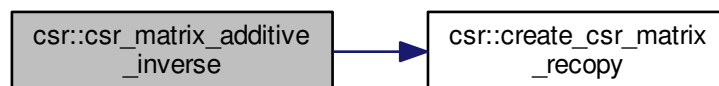
#### Paramètres

$A$	Matrice CSR
-----	-------------

#### Renvoie

$-A$

Voici le graphe d'appel pour cette fonction :



### 5.2.2.6 csr\_matrix\_equal()

```

logical function csr::csr_matrix_equal (
    type(csr_matrix), intent(in) A,
    type(csr_matrix), intent(in) B )

```

Cette fonction permet de vérifier si deux matrices sont égales.



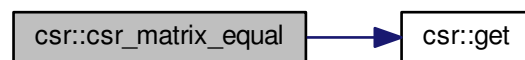
## Paramètres

$A$	Matrice CSR
$B$	Matrice CSR

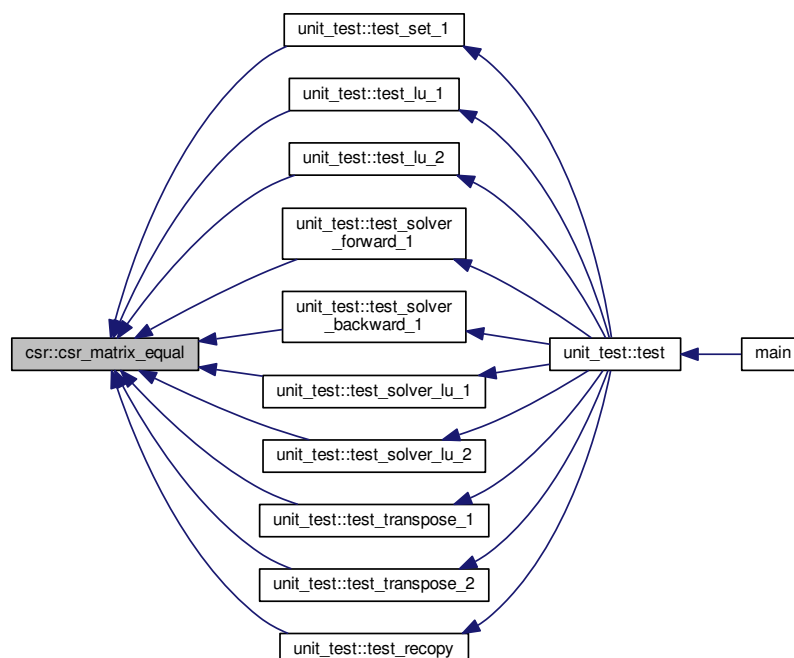
## Renvoie

$$A = B$$

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



#### 5.2.2.7 `csr_matrix_inner_product()`

```
real(kind=selected_real_kind(15, 307)) function csr::csr_matrix_inner_product (  
    type(csr_matrix), intent(in) A,  
    type(csr_matrix), intent(in) B )
```

Cette fonction calcule le produit scalaire de deux vecteurs.

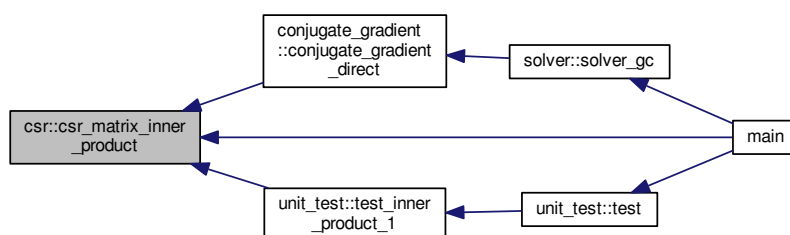
## Paramètres

<i>A</i>	Matrice CSR définissant un vecteur colonne
<i>B</i>	Matrice CSR définissant un vecteur colonne

## Renvoie

 $\langle A|B \rangle$ 

Voici le graphe des appelants de cette fonction :



## 5.2.2.8 csr\_matrix\_prod()

```

type(csr_matrix) function csr::csr_matrix_prod (
    type(csr_matrix), intent(in) A,
    type(csr_matrix), intent(in) B )

```

Cette fonction calcule le produit de deux matrices.

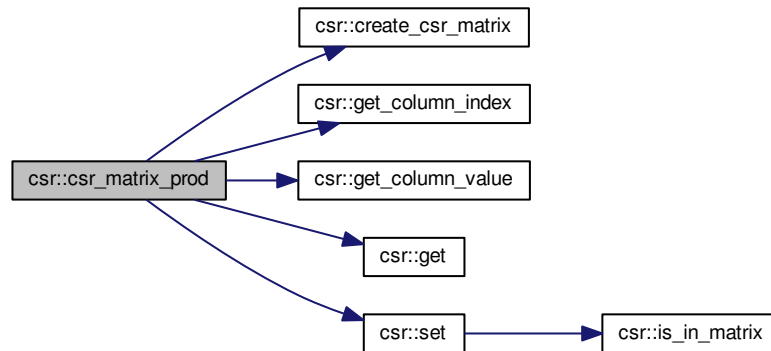
## Paramètres

<i>A</i>	Matrice CSR
<i>B</i>	Matrice CSR

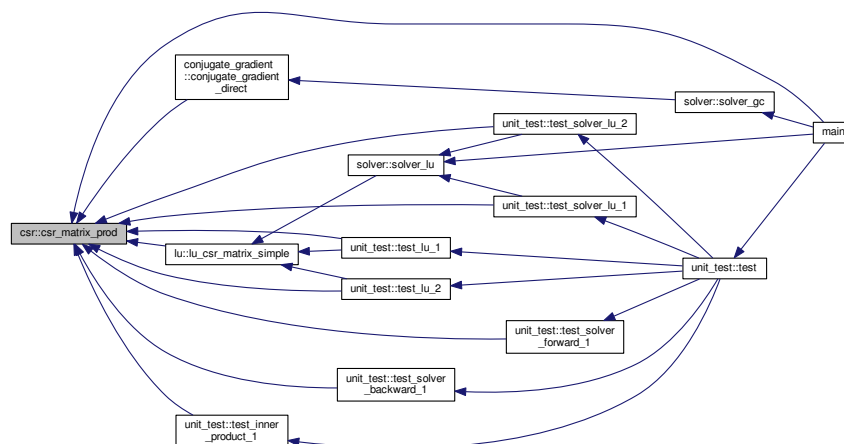
Renvoie

$$A * B$$

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



### 5.2.2.9 csr\_matrix\_save\_to\_file()

```

subroutine csr::csr_matrix_save_to_file (
    class (csr_matrix) M,
    character(len=*) , intent(in) f )
  
```

Cette subroutine permet de sauvegarder une matrice CSR dans un fichier.

## Paramètres

$M$	Matrice CSR à sauver dans un fichier
$f$	Fichier de sortie

## 5.2.2.10 csr\_matrix\_scalar\_prod()

```
type(csr_matrix) function csr::csr_matrix_scalar_prod (
    class(csr_matrix), intent(in) M,
    real(kind=selected_real_kind(15, 307)), intent(in) K )
```

Cette fonction calcule le produit entre un réel et une matrice.

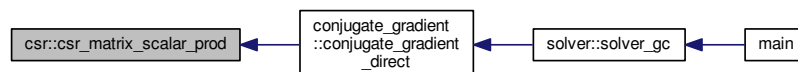
## Paramètres

$M$	Matrice CSR
$K$	un réel

## Renvoie

$$k * M$$

Voici le graphe des appelants de cette fonction :



## 5.2.2.11 csr\_matrix\_vector\_add()

```
type(csr_matrix) function csr::csr_matrix_vector_add (
    type(csr_matrix), intent(in) A,
    type(csr_matrix), intent(in) B )
```

Cette fonction calcule la somme de deux vecteurs colonnes.

## Paramètres

$A$	Matrice CSR définissant un vecteur colonne
$B$	Matrice CSR définissant un vecteur colonne

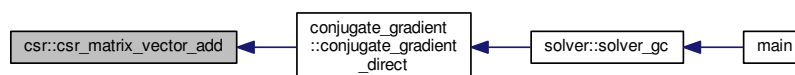
## Renvoie

$$A + B$$

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



### 5.2.2.12 csr\_matrix\_vector\_sub()

```

type(csr_matrix) function csr::csr_matrix_vector_sub (
    type(csr_matrix), intent(in) A,
    type(csr_matrix), intent(in) B )
  
```

Cette fonction calcule la différence de deux vecteurs colonnes.

#### Paramètres

<i>A</i>	Matrice CSR définissant un vecteur colonne
<i>B</i>	Matrice CSR définissant un vecteur colonne

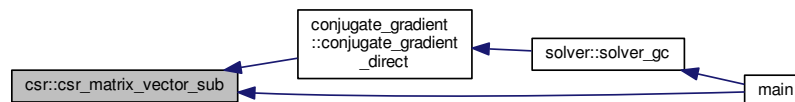
Renvoie

$$A - B$$

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



### 5.2.2.13 display()

```

subroutine csr::display (
    class(csr_matrix), intent(in) M )
  
```

Cette subroutine affiche une matrice CSR.

Paramètres

<i>M</i>	Matrice à afficher
----------	--------------------

Voici le graphe d'appel pour cette fonction :



#### 5.2.2.14 display\_debug()

```
subroutine csr::display_debug (
    class(csr_matrix), intent(in) M )
```

Cette subroutine affiche une matrice CSR avec des informations avancées.

##### Paramètres

<i>M</i>	Matrice à afficher
----------	--------------------

#### 5.2.2.15 free\_csr\_matrix()

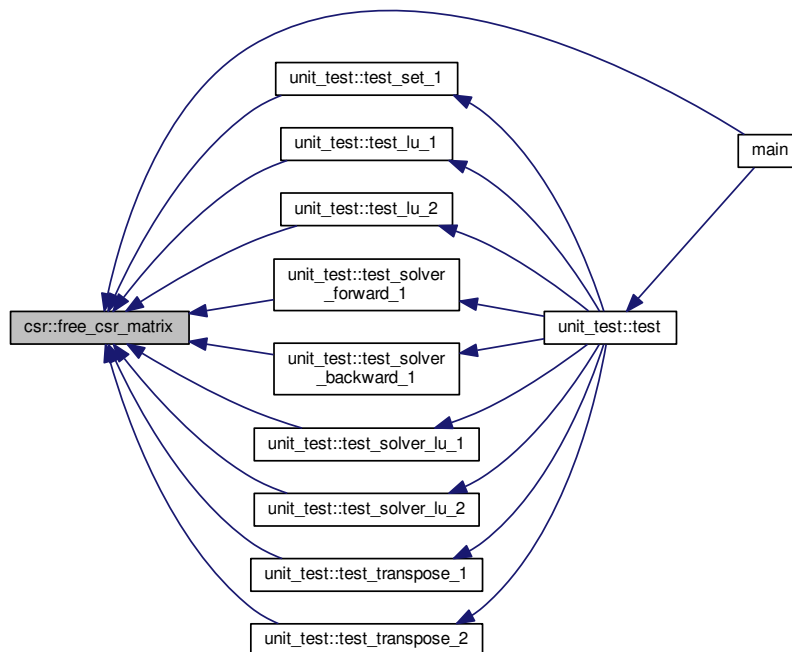
```
subroutine csr::free_csr_matrix (
    type(csr_matrix) M )
```

Cette subroutine permet de détruire une matrice CSR.

##### Paramètres

<i>M</i>	Matrice à désallouer
----------	----------------------

Voici le graphe des appelants de cette fonction :





## 5.2.2.16 get()

```
real(kind=selected_real_kind(15, 307)) function csr::get (
    class(csr_matrix), intent(in) M,
    integer, intent(in) row,
    integer, intent(in) col )
```

Cette fonction permet d'obtenir la valeur d'un élément d'une matrice.

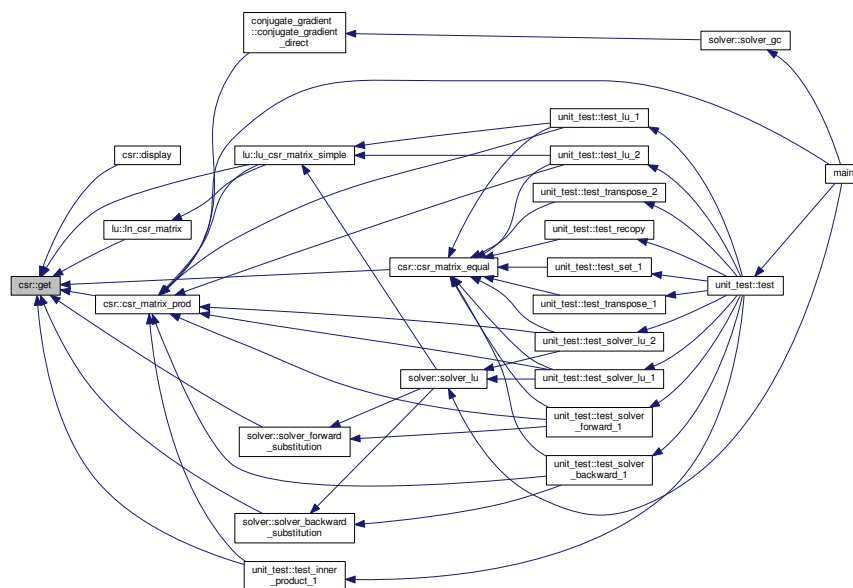
## Paramètres

<i>M</i>	Une matrice CSR
<i>row</i>	Ligne de la valeur souhaitée
<i>col</i>	Colonne de la valeur souhaitée

## Renvoie

Valeur contenu à la position  $(row, col)$ . Si La position est invalide retourne 0.

Voici le graphe des appelants de cette fonction :



## 5.2.2.17 get\_column\_index()

```
integer function, dimension(:), allocatable csr::get_column_index (
    class(csr_matrix), intent(in) M,
    integer, intent(in) row )
```

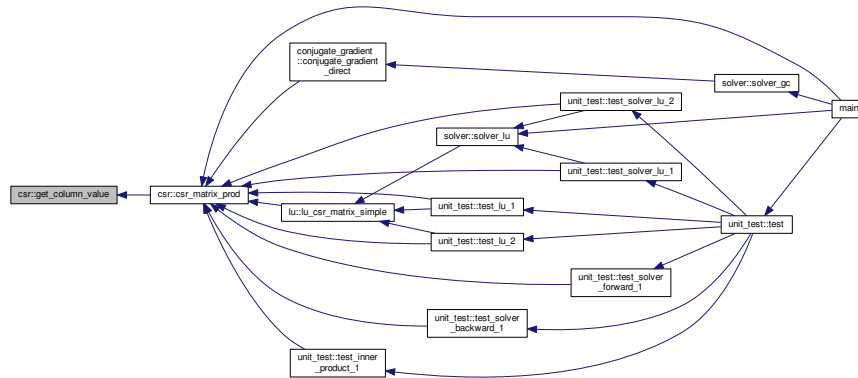
Cette fonction permet d'obtenir les colonnes non-nulles d'une ligne donnée.



**Renvoie**

Ensemble des valeurs non nulle

Voici le graphe des appelants de cette fonction :

**5.2.2.19 is\_column\_vector()**

```
logical function csr::is_column_vector (
    class(csr_matrix), intent(in) V )
```

Cette fonction permet de vérifier si une matrice est un vecteur colonne.

**Paramètres**

<i>V</i>	Une matrice CSR
----------	-----------------

**Renvoie**

Retourne VRAI si la matrice est un vecteur colonne

**5.2.2.20 is\_in\_matrix()**

```
logical function csr::is_in_matrix (
    class(csr_matrix), intent(in) M,
    integer, intent(in) row,
    integer, intent(in) col )
```

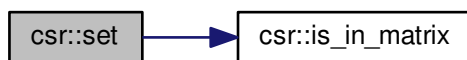
Cette fonction permet de vérifier que des indices sont bien valide pour une matrice donnée.

**Paramètres**

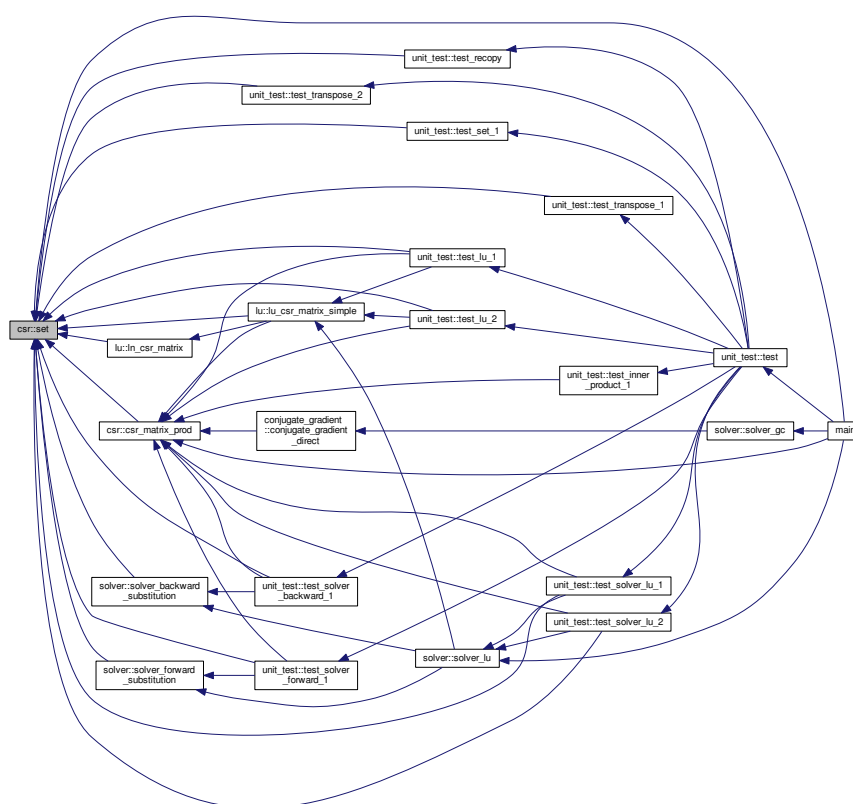
<i>M</i>	Matrice CSR
<i>row</i>	Ligne
<i>col</i>	Colonne



Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



### 5.2.2.22 tr()

```

type(csr_matrix) function csr::tr (
    class(csr_matrix), intent(in) M )
  
```

Cette fonction calcule la transposée d'une matrice.

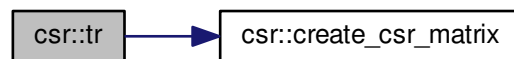
## Paramètres

$M$	Matrice CSR
-----	-------------

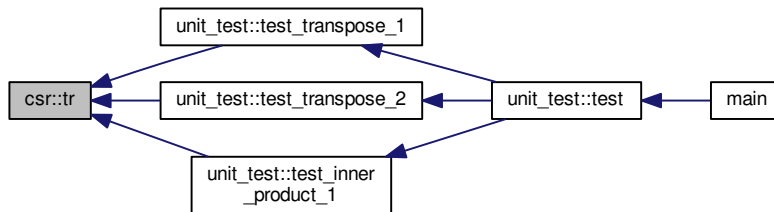
## Renvoie

$$M^T$$

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



## 5.3 Référence du module lu

Ce module permet d'obtenir la décomposition LU d'une matrice.

## Fonctions/Subroutines

- `type(csr_matrix) function, dimension(2) lu\_csr\_matrix\_simple (M)`
- Retourne la décomposition LU d'une matrice selon l'algorithme Doolittle sans permutation.*
- `type(csr_matrix) function ln\_csr\_matrix (A, n)`
- Fonction intermédiaire dans le calcul de LU.*

### 5.3.1 Description détaillée

Ce module permet d'obtenir la décomposition LU d'une matrice.

## Auteur

Mechineau Alexandre

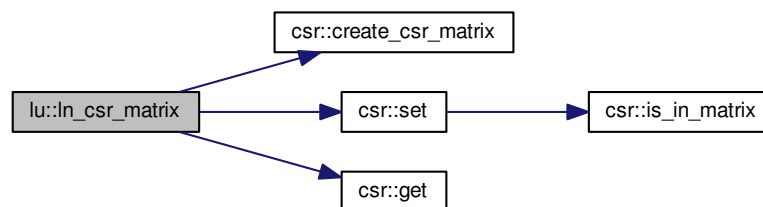
### 5.3.2 Documentation de la fonction/subroutine

#### 5.3.2.1 ln\_csr\_matrix()

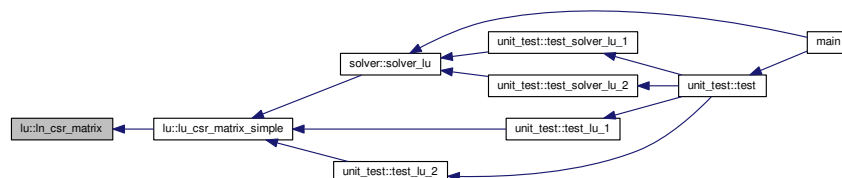
```
type(csr_matrix) function lu::ln_csr_matrix (
    type(csr_matrix), intent(in) A,
    integer, intent(in) n )
```

Fonction intermédiaire dans le calcul de LU.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



#### 5.3.2.2 lu\_csr\_matrix\_simple()

```
type(csr_matrix) function, dimension(2) lu::lu_csr_matrix_simple (
    type(csr_matrix), intent(in) M )
```

Retourne la décomposition LU d'une matrice selon l'algorithme Doolittle sans permutation.

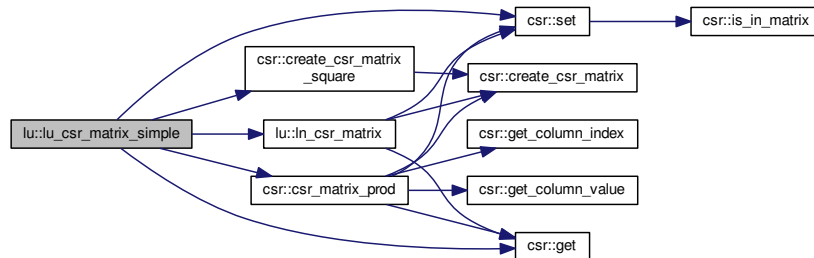
#### Paramètres

<i>M</i>	Matrice CSR
----------	-------------

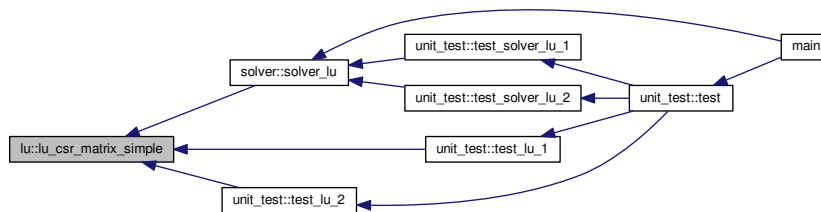
## Renvoie

Tableau de 2 matrice CSR (L, U)

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



## 5.4 Référence du module solver

Ce module permet de résoudre des systèmes linéaires en utilisant soit la décomposition LU, soit la méthode du gradient conjugué.

### Fonctions/Subroutines

- type(csr\_matrix) fonction [solver\\_lu](#) (A, B)  
*Résouds le système  $AX = B$  en utilisant la décomposition LU de A.*
- type(csr\_matrix) fonction [solver\\_gc](#) (A, B, eps)  
*Résouds le système  $AX = B$  en utilisant la méthode du gradient conjugué*
- type(csr\_matrix) fonction [solver\\_forward\\_substitution](#) (L, B)  
*Résouds le système  $LX = B$  un utilisant la méthode substitution.*
- type(csr\_matrix) fonction [solver\\_backward\\_substitution](#) (L, B)  
*Résouds le système  $LX = B$  un utilisant la méthode substitution inversée.*

### 5.4.1 Description détaillée

Ce module permet de résoudre des systèmes linéaires en utilisant soit la décomposition LU, soit la méthode du gradient conjugué.

#### Auteur

Mechineau Alexandre



## 5.4.2 Documentation de la fonction/subroutine

### 5.4.2.1 solver\_backward\_substitution()

```
type(csr_matrix) function solver::solver_backward_substitution (
    type(csr_matrix), intent(in) L,
    type(csr_matrix), intent(in) B )
```

Résouds le système  $LX = B$  en utilisant la méthode substitution inversée.

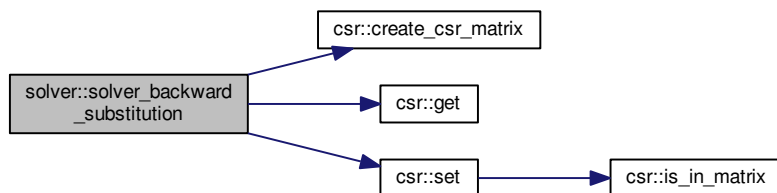
#### Paramètres

$L$	Matrice CSR triangulaire supérieur
$B$	Matrice CSR représentant un vecteur colonne

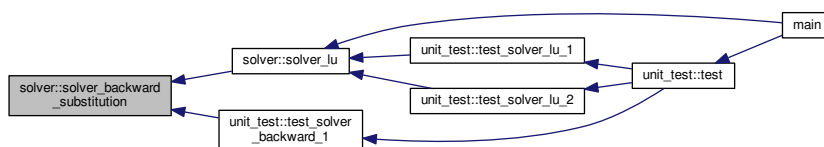
#### Renvoie

Solution du système

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



### 5.4.2.2 solver\_forward\_substitution()

```
type(csr_matrix) function solver::solver_forward_substitution (
    type(csr_matrix), intent(in) L,
    type(csr_matrix), intent(in) B )
```

Résouds le système  $LX = B$  en utilisant la méthode substitution.

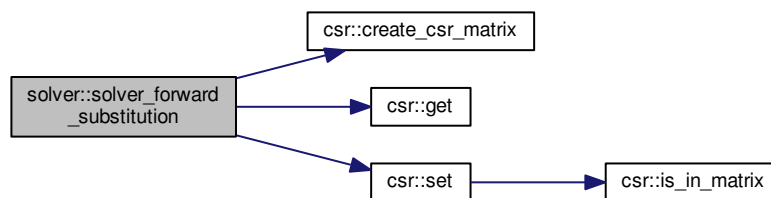
#### Paramètres

$L$	Matrice CSR triangulaire inférieur
$B$	Matrice CSR représentant un vecteur colonne

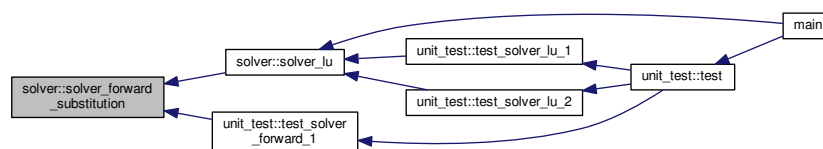
#### Renvoie

Solution du système

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



### 5.4.2.3 solver\_gc()

```
type(csr_matrix) function solver::solver_gc (
    type(csr_matrix), intent(in) A,
    type(csr_matrix), intent(in) B,
    real, intent(in), optional eps )
```

Résouds le système  $AX = B$  en utilisant la méthode du gradient conjugué

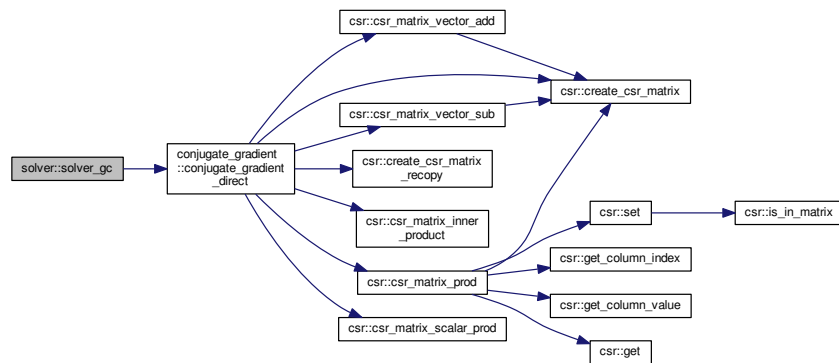
## Paramètres

<i>A</i>	Matrice CSR
<i>B</i>	Matrice CSR représentant un vecteur colonne
<i>eps</i>	Précision requise (Optionel)

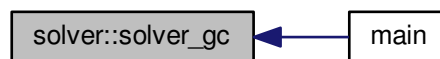
## Renvoie

Approximation, dans le pire des cas, de la solution du système

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



## 5.4.2.4 solver\_lu()

```

type(csr_matrix) function solver::solver_lu (
    type(csr_matrix), intent(in) A,
    type(csr_matrix), intent(in) B )
  
```

Résouds le système  $AX = B$  en utilisant la décomposition LU de A.

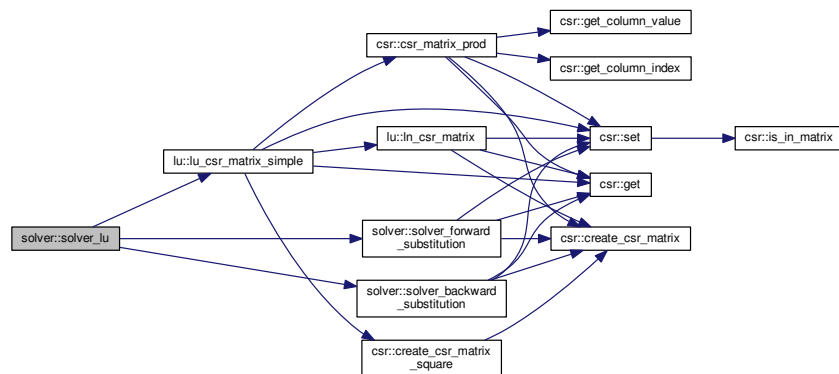
## Paramètres

<i>A</i>	Matrice CSR
<i>B</i>	Matrice CSR représentant un vecteur colonne

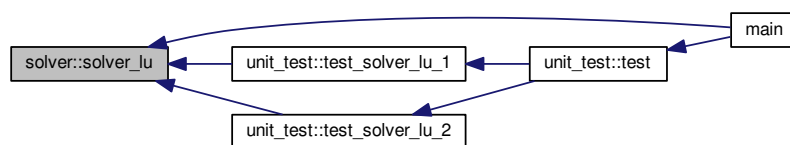
## Renvoie

Approximation, dans le pire des cas de la solution du système

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



## 5.5 Référence du module unit\_test

Ce module contient plusieurs test permettant d'assurer le bon fonctionnement des modules CSR, LU, CONJUGATE\_GRADIENT et SOLVER.

## Fonctions/Subroutines

- subroutine `test` ()  
*Cette subroutine appelle l'ensemble des tests implémentés et affiche chaque'un des résultats.*
- logical function `test_set_1` ()
- logical function `test_lu_1` ()
- logical function `test_lu_2` ()
- logical function `test_solver_forward_1` ()

- logical function `test_solver_backward_1` ()
- logical function `test_solver_lu_1` ()
- logical function `test_solver_lu_2` ()
- logical function `test_transpose_1` ()
- logical function `test_transpose_2` ()
- logical function `test_inner_product_1` ()
- logical function `test_recopy` ()

### 5.5.1 Description détaillée

Ce module contient plusieurs test permettant d'assurer le bon fonctionnement des modules CSR, LU, CONJUG↔ATE\_GRADIENT et SOLVER.

#### Auteur

Mechineau Alexandre

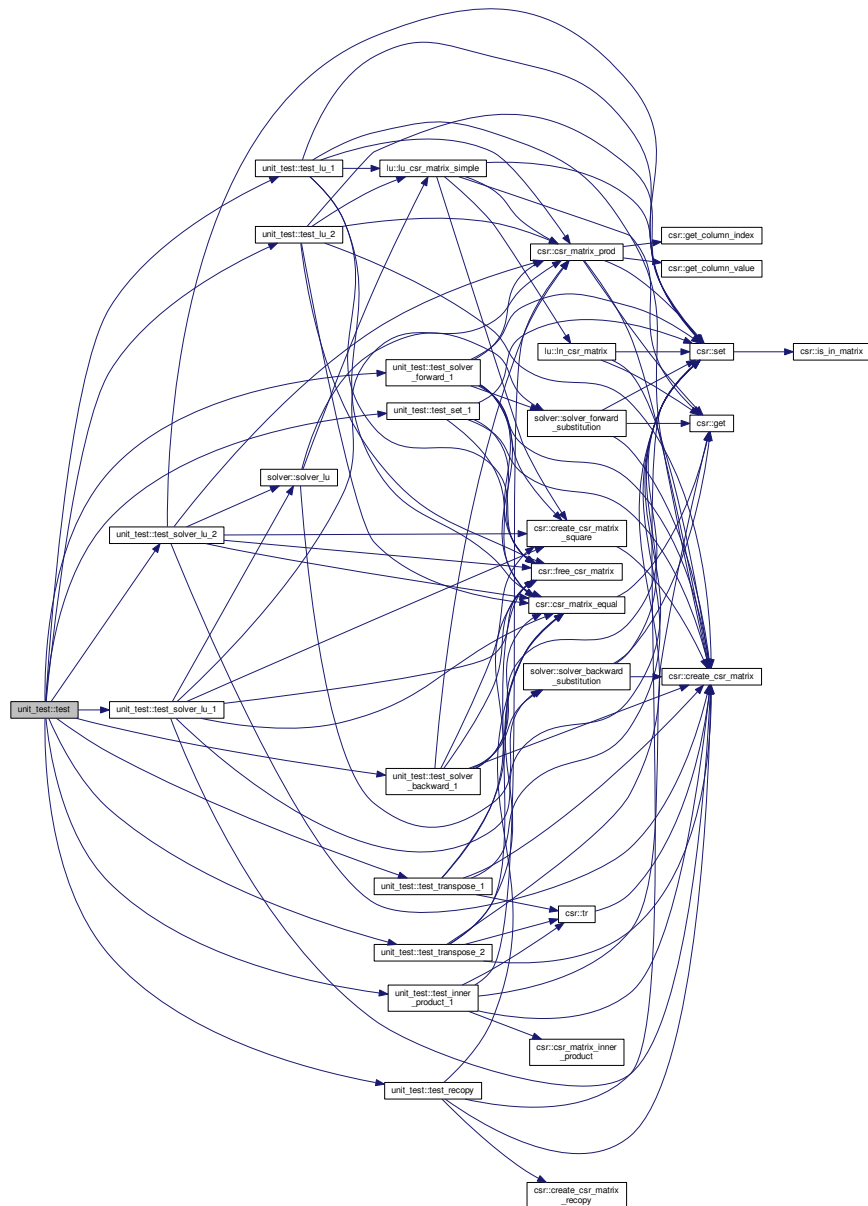
### 5.5.2 Documentation de la fonction/subroutine

#### 5.5.2.1 test()

```
subroutine unit_test::test ( )
```

Cette subroutine appelle l'ensemble des tests implémentés et affiche chaque'un des résultats.

Voici le graphe d'appel pour cette fonction :



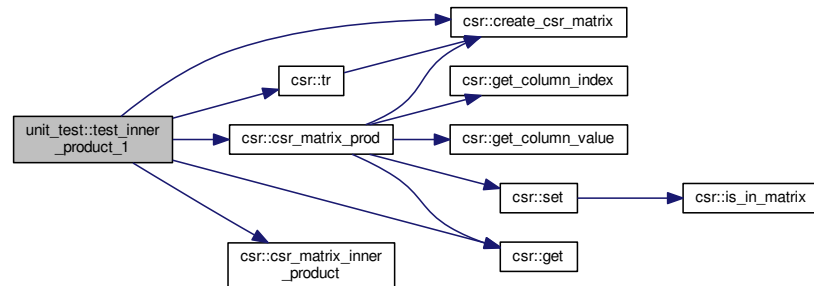
Voici le graphe des appelants de cette fonction :



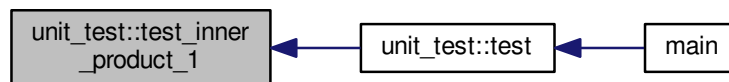
## 5.5.2.2 test\_inner\_product\_1()

```
logical function unit_test::test_inner_product_1 ( )
```

Voici le graphe d'appel pour cette fonction :



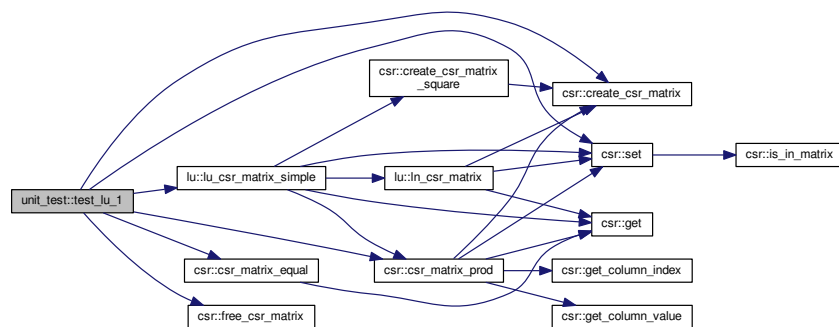
Voici le graphe des appelants de cette fonction :



## 5.5.2.3 test\_lu\_1()

```
logical function unit_test::test_lu_1 ( )
```

Voici le graphe d'appel pour cette fonction :



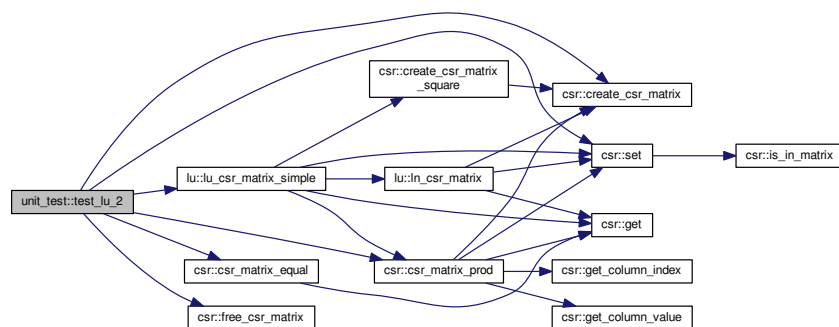
Voici le graphe des appelants de cette fonction :



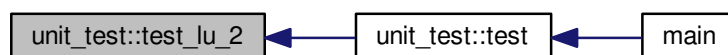
#### 5.5.2.4 test\_lu\_2()

logical function `unit_test::test_lu_2 ( )`

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :

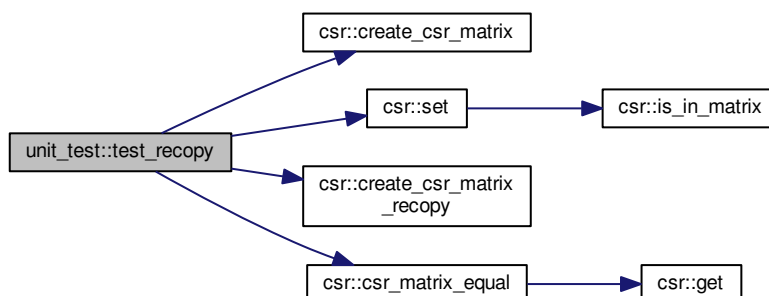




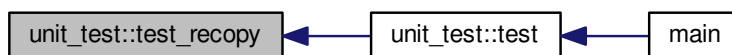
## 5.5.2.5 test\_recopy()

```
logical function unit_test::test_recopy ( )
```

Voici le graphe d'appel pour cette fonction :



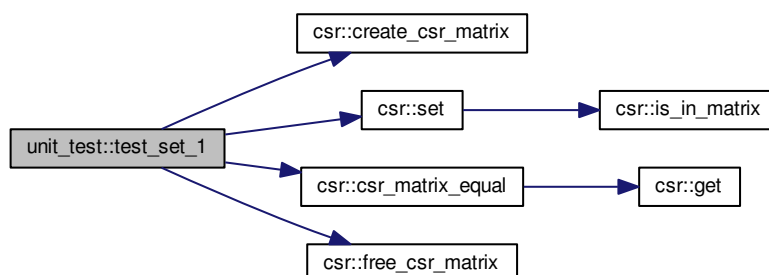
Voici le graphe des appelants de cette fonction :



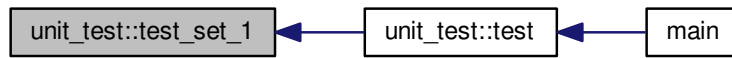
## 5.5.2.6 test\_set\_1()

```
logical function unit_test::test_set_1 ( )
```

Voici le graphe d'appel pour cette fonction :



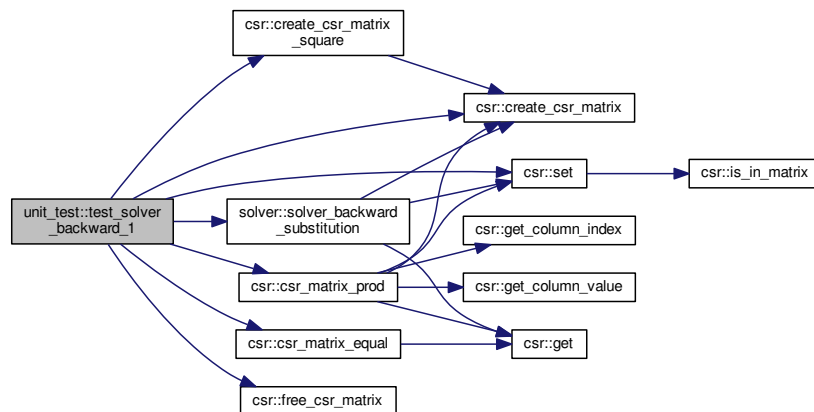
Voici le graphe des appelants de cette fonction :



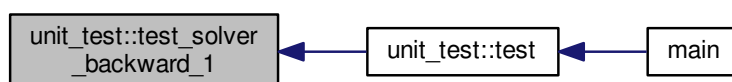
#### 5.5.2.7 test\_solver\_backward\_1()

```
logical function unit_test::test_solver_backward_1 ( )
```

Voici le graphe d'appel pour cette fonction :



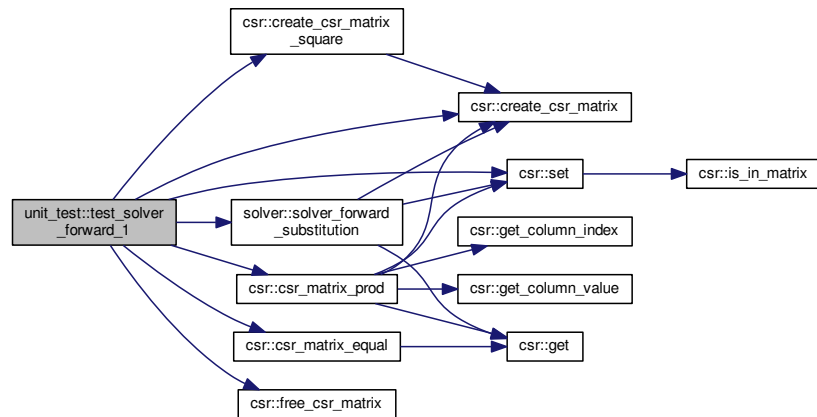
Voici le graphe des appelants de cette fonction :



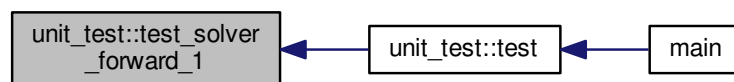
## 5.5.2.8 test\_solver\_forward\_1()

```
logical function unit_test::test_solver_forward_1 ( )
```

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



## 5.5.2.9 test\_solver\_lu\_1()

```
logical function unit_test::test_solver_lu_1 ( )
```



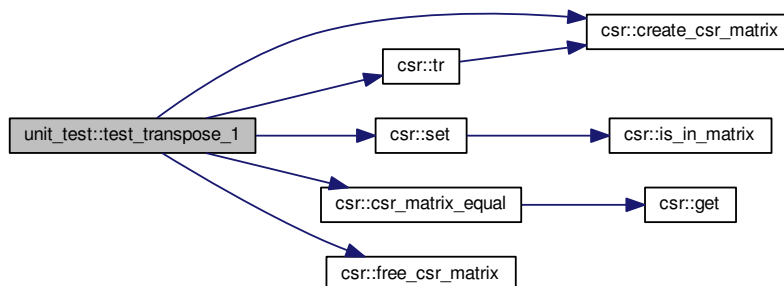
Voici le graphe des appelants de cette fonction :



#### 5.5.2.11 test\_transpose\_1()

logical function `unit_test::test_transpose_1 ( )`

Voici le graphe d'appel pour cette fonction :



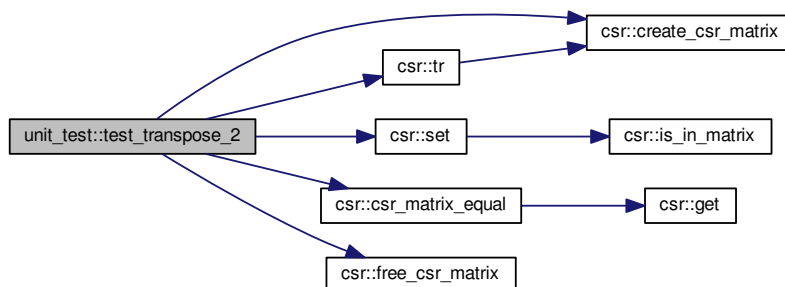
Voici le graphe des appelants de cette fonction :



### 5.5.2.12 test\_transpose\_2()

logical function unit\_test::test\_transpose\_2 ( )

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



## Chapitre 6

# Documentation du type de données

### 6.1 Référence du type csr : :csr\_matrix

Type décrivant une matrice CSR Attention m\_col stocke l'index des colonnes en commençant par 1. [CSR].

#### Fonctions membres publiques

- PROCEDURE [is\\_column\\_vector](#)
- [tr](#)
- [csr\\_matrix\\_scalar\\_prod](#)
- [is\\_in\\_matrix](#)
- [get](#)
- [get\\_column\\_index](#)
- [get\\_column\\_value](#)
- [set](#)
- [display](#)
- [display\\_debug](#)
- [csr\\_matrix\\_additive\\_inverse](#)
- [csr\\_matrix\\_save\\_to\\_file](#)

#### Attributs publics

- integer, dimension( :), allocatable [m\\_rowidx](#)  
*Tableau stockant l'index des lignes.*
- integer, dimension( :), allocatable [m\\_col](#)  
*Tableau stockant les colonnes non nulles.*
- real(kind=selected\_real\_kind(15, 307)), dimension( :), allocatable [m\\_val](#)  
*Tableau stockant les valeurs non nulles.*
- integer [m\\_m](#)  
*Nombre de colonnes de la matrice.*
- integer [m\\_n](#)  
*Nombre de lignes de la matrice.*

#### 6.1.1 Description détaillée

Type décrivant une matrice CSR Attention m\_col stocke l'index des colonnes en commençant par 1. [CSR].

#### 6.1.2 Documentation des fonctions/subroutines membres

#### 6.1.2.1 `csr_matrix_additive_inverse()`

```
csr::csr_matrix::csr_matrix_additive_inverse ( )
```

#### 6.1.2.2 `csr_matrix_save_to_file()`

```
csr::csr_matrix::csr_matrix_save_to_file ( )
```

#### 6.1.2.3 `csr_matrix_scalar_prod()`

```
csr::csr_matrix::csr_matrix_scalar_prod ( )
```

#### 6.1.2.4 `display()`

```
csr::csr_matrix::display ( )
```

#### 6.1.2.5 `display_debug()`

```
csr::csr_matrix::display_debug ( )
```

#### 6.1.2.6 `get()`

```
csr::csr_matrix::get ( )
```

#### 6.1.2.7 `get_column_index()`

```
csr::csr_matrix::get_column_index ( )
```

#### 6.1.2.8 `get_column_value()`

```
csr::csr_matrix::get_column_value ( )
```



#### 6.1.2.9 `is_column_vector()`

```
PROCEDURE csr::csr_matrix::is_column_vector ( )
```

#### 6.1.2.10 `is_in_matrix()`

```
csr::csr_matrix::is_in_matrix ( )
```

#### 6.1.2.11 `set()`

```
csr::csr_matrix::set ( )
```

#### 6.1.2.12 `tr()`

```
csr::csr_matrix::tr ( )
```

### 6.1.3 Documentation des données membres

#### 6.1.3.1 `m_col`

```
integer, dimension(:), allocatable csr::csr_matrix::m_col
```

Tableau stockant les colonnes non nulles.

#### 6.1.3.2 `m_m`

```
integer csr::csr_matrix::m_m
```

Nombre de colonnes de la matrice.

#### 6.1.3.3 m\_n

```
integer csr::csr_matrix::m_n
```

Nombre de lignes de la matrice.

#### 6.1.3.4 m\_rowidx

```
integer, dimension(:), allocatable csr::csr_matrix::m_rowidx
```

Tableau stockant l'index des lignes.

#### 6.1.3.5 m\_val

```
real(kind=selected_real_kind(15, 307)), dimension(:), allocatable csr::csr_matrix::m_val
```

Tableau stockant les valeurs non nulles.

La documentation de ce type a été générée à partir du fichier suivant :

— Source/[CSR.f90](#)

## Chapitre 7

# Documentation des fichiers

### 7.1 Référence du fichier Source/CONJUGATE\_GRADIENT.f90

#### Modules

- module `conjugate_gradient`  
*Ce module permet d'utiliser l'algorithme du gradient conjugué. Deux implémentations sont fournies, la première effectue le nombre maximal d'étape, la seconde s'arrête à une précision donnée.*

#### Fonctions/Subroutines

- `type(csr_matrix)` fonction `conjugate_gradient : :conjugate_gradient_direct` (A, B, eps)  
*Résouds le système  $AX = B$  en utilisant la méthode du gradient conjugué Source [https://www.ljll.fr/~math.upmc.fr/hecht/ftp/InfoSci/doc-pdf/Master\\_book\\_GC.pdf](https://www.ljll.fr/~math.upmc.fr/hecht/ftp/InfoSci/doc-pdf/Master_book_GC.pdf).*

### 7.2 Référence du fichier Source/CSR.f90

#### Les types de données

- type `csr : :csr_matrix`  
*Type décrivant une matrice CSR Attention `m_col` stocke l'index des colonnes en commençant par 1. [CSR].*

#### Modules

- module `csr`  
*Ce module permet de créer,stocker et manipuler une matrice sous la forme CSR. Plusieurs opérations courantes sont implémentés, avec certains raffinement pour le cas de vecteur.*

## Fonctions/Subroutines

- type(csr\_matrix) fonction `csr : :create_csr_matrix_from_file` (f)  
*Cette fonction permet de créer une matrice depuis un fichier. La matrice dans le fichier doit utiliser le style C-CSR modifié pour que les colonnes commencent à être indexées par 1. **On admet que le fichier passé est valide***
- subroutine `csr : :csr_matrix_save_to_file` (M, f)  
*Cette subroutine permet de sauvegarder une matrice CSR dans un fichier.*
- type(csr\_matrix) fonction `csr : :create_csr_matrix_square` (n)  
*Cette fonction permet de créer une matrice carrée vide.*
- type(csr\_matrix) fonction `csr : :create_csr_matrix_recopy` (M)  
*Cette fonction permet de dupliquer une matrice.*
- type(csr\_matrix) fonction `csr : :create_csr_matrix` (m, n)  
*Cette fonction crée une matrice de taille quelconque.*
- subroutine `csr : :free_csr_matrix` (M)  
*Cette subroutine permet de détruire une matrice CSR.*
- logical fonction `csr : :is_in_matrix` (M, row, col)  
*Cette fonction permet de vérifier que des indices sont bien valides pour une matrice donnée.*
- logical fonction `csr : :is_column_vector` (V)  
*Cette fonction permet de vérifier si une matrice est un vecteur colonne.*
- real(kind=selected\_real\_kind(15, 307)) fonction `csr : :get` (M, row, col)  
*Cette fonction permet d'obtenir la valeur d'un élément d'une matrice.*
- subroutine `csr : :set` (M, row, col, val)  
*Cette subroutine permet de modifier une valeur dans une matrice CSR.*
- subroutine `csr : :display_debug` (M)  
*Cette subroutine affiche une matrice CSR avec des informations avancées.*
- subroutine `csr : :display` (M)  
*Cette subroutine affiche une matrice CSR.*
- integer fonction, dimension( : ), allocatable `csr : :get_column_index` (M, row)  
*Cette fonction permet d'obtenir les colonnes non-nulles d'une ligne donnée.*
- real(kind=selected\_real\_kind(15, 307)) fonction, dimension( : ), allocatable `csr : :get_column_value` (M, row)  
*Cette fonction permet d'obtenir les valeurs non-nulles d'une ligne donnée.*
- type(csr\_matrix) fonction `csr : :csr_matrix_prod` (A, B)  
*Cette fonction calcule le produit de deux matrices.*
- logical fonction `csr : :csr_matrix_equal` (A, B)  
*Cette fonction permet de vérifier si deux matrices sont égales.*
- type(csr\_matrix) fonction `csr : :tr` (M)  
*Cette fonction calcule la transposée d'une matrice.*
- real(kind=selected\_real\_kind(15, 307)) fonction `csr : :csr_matrix_inner_product` (A, B)  
*Cette fonction calcule le produit scalaire de deux vecteurs.*
- type(csr\_matrix) fonction `csr : :csr_matrix_vector_add` (A, B)  
*Cette fonction calcule la somme de deux vecteurs colonnes.*
- type(csr\_matrix) fonction `csr : :csr_matrix_vector_sub` (A, B)  
*Cette fonction calcule la différence de deux vecteurs colonnes.*
- type(csr\_matrix) fonction `csr : :csr_matrix_additive_inverse` (A)  
*Cette fonction calcule l'inverse associé à la loi additive d'une matrice.*
- type(csr\_matrix) fonction `csr : :csr_matrix_scalar_prod` (M, K)  
*Cette fonction calcule le produit entre un réel et une matrice.*

## 7.3 Référence du fichier Source/LU.f90

### Modules

- module `lu`  
*Ce module permet d'obtenir la décomposition LU d'une matrice.*

### Fonctions/Subroutines

- type(csr\_matrix) fonction, dimension(2) `lu : :lu_csr_matrix_simple` (M)  
*Retourne la décomposition LU d'une matrice selon l'algorithme Doolittle sans permutation.*
- type(csr\_matrix) fonction `lu : :ln_csr_matrix` (A, n)  
*Fonction intermédiaire dans le calcul de LU.*

## 7.4 Référence du fichier Source/Projet.f90

### Fonctions/Subroutines

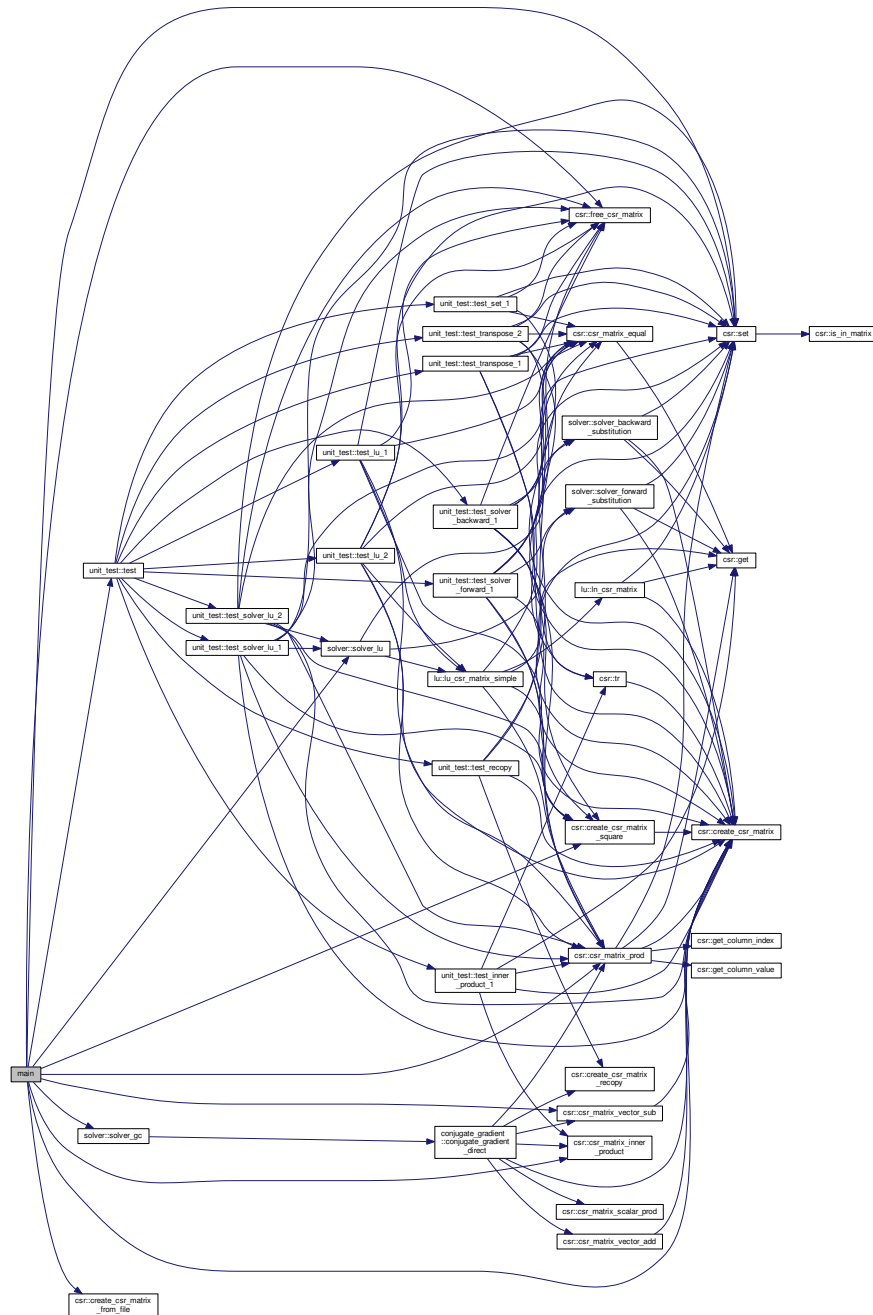
— program [main](#)

#### 7.4.1 Documentation de la fonction/subroutine

##### 7.4.1.1 main()

```
program main ( )
```

Voici le graphe d'appel pour cette fonction :



## 7.5 Référence du fichier Source/SOLVER.f90

### Modules

— module [solver](#)

*Ce module permet de résoudre des systèmes linéaires en utilisant soit la décomposition LU, soit la méthode du gradient conjugué.*

## Fonctions/Subroutines

- type(csr\_matrix) fonction `solver : :solver_lu` (A, B)  
*Résouds le système  $AX = B$  en utilisant la décomposition LU de A.*
- type(csr\_matrix) fonction `solver : :solver_gc` (A, B, eps)  
*Résouds le système  $AX = B$  en utilisant la méthode du gradient conjugué*
- type(csr\_matrix) fonction `solver : :solver_forward_substitution` (L, B)  
*Résouds le système  $LX = B$  un utilisant la méthode substitution.*
- type(csr\_matrix) fonction `solver : :solver_backward_substitution` (L, B)  
*Résouds le système  $LX = B$  un utilisant la méthode substitution inversée.*

## 7.6 Référence du fichier Source/UNIT\_TEST.f90

### Modules

- module `unit_test`  
*Ce module contient plusieurs test permettant d'assurer le bon fonctionnement des modules CSR, LU, CONJUGA↔  
TE\_GRADIENT et SOLVER.*

### Fonctions/Subroutines

- subroutine `unit_test : :test` ()  
*Cette subroutine appelle l'ensemble des tests implémentés et affiche chaqu'un des résultats.*
- logical function `unit_test : :test_set_1` ()
- logical function `unit_test : :test_lu_1` ()
- logical function `unit_test : :test_lu_2` ()
- logical function `unit_test : :test_solver_forward_1` ()
- logical function `unit_test : :test_solver_backward_1` ()
- logical function `unit_test : :test_solver_lu_1` ()
- logical function `unit_test : :test_solver_lu_2` ()
- logical function `unit_test : :test_transpose_1` ()
- logical function `unit_test : :test_transpose_2` ()
- logical function `unit_test : :test_inner_product_1` ()
- logical function `unit_test : :test_recopy` ()





# Index

- conjugate\_gradient, [11](#)
  - conjugate\_gradient\_direct, [11](#)
- conjugate\_gradient\_direct
  - conjugate\_gradient, [11](#)
- create\_csr\_matrix
  - csr, [13](#)
- create\_csr\_matrix\_from\_file
  - csr, [14](#)
- create\_csr\_matrix\_recopy
  - csr, [15](#)
- create\_csr\_matrix\_square
  - csr, [16](#)
- csr, [12](#)
  - create\_csr\_matrix, [13](#)
  - create\_csr\_matrix\_from\_file, [14](#)
  - create\_csr\_matrix\_recopy, [15](#)
  - create\_csr\_matrix\_square, [16](#)
  - csr\_matrix\_additive\_inverse, [17](#)
  - csr\_matrix\_equal, [17](#)
  - csr\_matrix\_inner\_product, [19](#)
  - csr\_matrix\_prod, [21](#)
  - csr\_matrix\_save\_to\_file, [22](#)
  - csr\_matrix\_scalar\_prod, [23](#)
  - csr\_matrix\_vector\_add, [23](#)
  - csr\_matrix\_vector\_sub, [24](#)
  - display, [25](#)
  - display\_debug, [25](#)
  - free\_csr\_matrix, [26](#)
  - get, [26](#)
  - get\_column\_index, [27](#)
  - get\_column\_value, [28](#)
  - is\_column\_vector, [29](#)
  - is\_in\_matrix, [29](#)
  - set, [30](#)
  - tr, [31](#)
- csr : :csr\_matrix, [49](#)
  - csr\_matrix\_additive\_inverse, [49](#)
  - csr\_matrix\_save\_to\_file, [50](#)
  - csr\_matrix\_scalar\_prod, [50](#)
  - display, [50](#)
  - display\_debug, [50](#)
  - get, [50](#)
  - get\_column\_index, [50](#)
  - get\_column\_value, [50](#)
  - is\_column\_vector, [50](#)
  - is\_in\_matrix, [51](#)
  - m\_col, [51](#)
  - m\_m, [51](#)
  - m\_n, [51](#)
  - m\_rowidx, [52](#)
  - m\_val, [52](#)
  - set, [51](#)
  - tr, [51](#)
- csr\_matrix\_additive\_inverse
  - csr, [17](#)
  - csr : :csr\_matrix, [49](#)
- csr\_matrix\_equal
  - csr, [17](#)
- csr\_matrix\_inner\_product
  - csr, [19](#)
- csr\_matrix\_prod
  - csr, [21](#)
- csr\_matrix\_save\_to\_file
  - csr, [22](#)
  - csr : :csr\_matrix, [50](#)
- csr\_matrix\_scalar\_prod
  - csr, [23](#)
  - csr : :csr\_matrix, [50](#)
- csr\_matrix\_vector\_add
  - csr, [23](#)
- csr\_matrix\_vector\_sub
  - csr, [24](#)
- display
  - csr, [25](#)
  - csr : :csr\_matrix, [50](#)
- display\_debug
  - csr, [25](#)
  - csr : :csr\_matrix, [50](#)
- free\_csr\_matrix
  - csr, [26](#)
- get
  - csr, [26](#)
  - csr : :csr\_matrix, [50](#)
- get\_column\_index
  - csr, [27](#)
  - csr : :csr\_matrix, [50](#)
- get\_column\_value
  - csr, [28](#)
  - csr : :csr\_matrix, [50](#)
- is\_column\_vector
  - csr, [29](#)
  - csr : :csr\_matrix, [50](#)
- is\_in\_matrix
  - csr, [29](#)
  - csr : :csr\_matrix, [51](#)

- ln\_csr\_matrix
  - lu, [33](#)
- lu, [32](#)
  - ln\_csr\_matrix, [33](#)
  - lu\_csr\_matrix\_simple, [33](#)
- lu\_csr\_matrix\_simple
  - lu, [33](#)
- m\_col
  - csr : :csr\_matrix, [51](#)
- m\_m
  - csr : :csr\_matrix, [51](#)
- m\_n
  - csr : :csr\_matrix, [51](#)
- m\_rowidx
  - csr : :csr\_matrix, [52](#)
- m\_val
  - csr : :csr\_matrix, [52](#)
- main
  - Projet.f90, [55](#)
- Projet.f90
  - main, [55](#)
- set
  - csr, [30](#)
  - csr : :csr\_matrix, [51](#)
- solver, [34](#)
  - solver\_backward\_substitution, [35](#)
  - solver\_forward\_substitution, [35](#)
  - solver\_gc, [36](#)
  - solver\_lu, [37](#)
- solver\_backward\_substitution
  - solver, [35](#)
- solver\_forward\_substitution
  - solver, [35](#)
- solver\_gc
  - solver, [36](#)
- solver\_lu
  - solver, [37](#)
- Source/CONJUGATE\_GRADIENT.f90, [53](#)
- Source/CSR.f90, [53](#)
- Source/LU.f90, [54](#)
- Source/Projet.f90, [55](#)
- Source/SOLVER.f90, [56](#)
- Source/UNIT\_TEST.f90, [57](#)
- test
  - unit\_test, [39](#)
- test\_inner\_product\_1
  - unit\_test, [40](#)
- test\_lu\_1
  - unit\_test, [41](#)
- test\_lu\_2
  - unit\_test, [42](#)
- test\_recopy
  - unit\_test, [42](#)
- test\_set\_1
  - unit\_test, [43](#)
- test\_solver\_backward\_1
  - unit\_test, [44](#)
- test\_solver\_forward\_1
  - unit\_test, [44](#)
- test\_solver\_lu\_1
  - unit\_test, [45](#)
- test\_solver\_lu\_2
  - unit\_test, [46](#)
- test\_transpose\_1
  - unit\_test, [47](#)
- test\_transpose\_2
  - unit\_test, [47](#)
- tr
  - csr, [31](#)
  - csr : :csr\_matrix, [51](#)
- unit\_test, [38](#)
  - test, [39](#)
  - test\_inner\_product\_1, [40](#)
  - test\_lu\_1, [41](#)
  - test\_lu\_2, [42](#)
  - test\_recopy, [42](#)
  - test\_set\_1, [43](#)
  - test\_solver\_backward\_1, [44](#)
  - test\_solver\_forward\_1, [44](#)
  - test\_solver\_lu\_1, [45](#)
  - test\_solver\_lu\_2, [46](#)
  - test\_transpose\_1, [47](#)
  - test\_transpose\_2, [47](#)