

Simple Mat

Généré par Doxygen 1.9.1



<b>1 Index des structures de données</b>	<b>1</b>
1.1 Structures de données	1
<b>2 Index des fichiers</b>	<b>3</b>
2.1 Liste des fichiers	3
<b>3 Documentation des structures de données</b>	<b>5</b>
3.1 Référence de la structure <code>simplemat_s</code>	5
3.1.1 Documentation des champs	5
3.1.1.1 <code>data</code>	5
3.1.1.2 <code>num_cols</code>	5
3.1.1.3 <code>num_rows</code>	5
<b>4 Documentation des fichiers</b>	<b>7</b>
4.1 Référence du fichier <code>include/simplemat.h</code>	7
4.1.1 Documentation des macros	8
4.1.1.1 <code>INVALID_CARREE</code>	9
4.1.1.2 <code>INVALID_COLS</code>	9
4.1.1.3 <code>INVALID_DIM</code>	9
4.1.1.4 <code>INVALID_DIM_ADD</code>	9
4.1.1.5 <code>INVALID_DIM_MUL</code>	9
4.1.1.6 <code>INVALID_DIM_MUL_D</code>	9
4.1.1.7 <code>INVALID_DIM_MUL_G</code>	9
4.1.1.8 <code>INVALID_INV</code>	10
4.1.1.9 <code>INVALID_INVERSION</code>	10
4.1.1.10 <code>INVALID_LIGNE</code>	10
4.1.1.11 <code>INVALID_ROWS</code>	10
4.1.1.12 <code>INVALID_SYS_DIM</code>	10
4.1.2 Documentation des définitions de type	10
4.1.2.1 <code>s_mat</code>	10
4.1.3 Documentation des fonctions	10
4.1.3.1 <code>afficher_mat()</code>	11
4.1.3.2 <code>echange_deux_lignes()</code>	11
4.1.3.3 <code>est_carree()</code>	11
4.1.3.4 <code>est_egal()</code>	11
4.1.3.5 <code>est_id()</code>	11
4.1.3.6 <code>est_nulle()</code>	11
4.1.3.7 <code>mat_add()</code>	12
4.1.3.8 <code>mat_alea()</code>	12
4.1.3.9 <code>mat_alea_entier()</code>	12
4.1.3.10 <code>mat_minus()</code>	12
4.1.3.11 <code>mat_mul()</code>	12
4.1.3.12 <code>mat_mul_droite()</code>	13

4.1.3.13 mat_mul_ext()	13
4.1.3.14 mat_mul_gauche()	13
4.1.3.15 mat_oppose()	13
4.1.3.16 s_mat_carree()	13
4.1.3.17 s_mat_coef()	13
4.1.3.18 s_mat_copie()	14
4.1.3.19 s_mat_e()	14
4.1.3.20 s_mat_free()	14
4.1.3.21 s_mat_h()	14
4.1.3.22 s_mat_id()	14
4.1.3.23 s_mat_l()	14
4.1.3.24 s_mat_new()	15
4.1.3.25 trace()	15
4.1.3.26 transpose()	15
4.2 Référence du fichier include/sm_test.h	15
4.2.1 Documentation des fonctions	15
4.2.1.1 run_tests()	15
4.3 Référence du fichier include/sm_util.h	16
4.3.1 Documentation des macros	17
4.3.1.1 ALLOC_CHECK	17
4.3.1.2 DEBUG_TRUE	17
4.3.1.3 LOG_ERROR	17
4.3.1.4 RESET	17
4.3.1.5 ROUGE	17
4.3.1.6 VERT	17
4.3.1.7 VIOLET	18
4.3.2 Documentation des fonctions	18
4.3.2.1 egalD()	18
4.3.2.2 error_log()	18
4.3.2.3 max()	18
4.3.2.4 min()	18
4.3.2.5 my_log()	18
4.4 Référence du fichier src/simplemat.c	19
4.4.1 Documentation des macros	20
4.4.1.1 INVALID_CARREE	20
4.4.1.2 INVALID_COLS	20
4.4.1.3 INVALID_DIM	20
4.4.1.4 INVALID_DIM_ADD	20
4.4.1.5 INVALID_DIM_MUL	21
4.4.1.6 INVALID_DIM_MUL_D	21
4.4.1.7 INVALID_DIM_MUL_G	21
4.4.1.8 INVALID_INVERSION	21

4.4.1.9 INVALID_LIGNE	21
4.4.1.10 INVALID_ROWS	21
4.4.2 Documentation des définitions de type	21
4.4.2.1 s_mat	21
4.4.3 Documentation des fonctions	22
4.4.3.1 afficher_mat()	22
4.4.3.2 echange_deux_lignes()	22
4.4.3.3 est_carree()	22
4.4.3.4 est_egal()	22
4.4.3.5 est_id()	22
4.4.3.6 est_nulle()	22
4.4.3.7 mat_add()	23
4.4.3.8 mat_alea()	23
4.4.3.9 mat_alea_entier()	23
4.4.3.10 mat_minus()	23
4.4.3.11 mat_mul()	23
4.4.3.12 mat_mul_droite()	23
4.4.3.13 mat_mul_ext()	24
4.4.3.14 mat_mul_gauche()	24
4.4.3.15 mat_oppose()	24
4.4.3.16 s_mat_carree()	24
4.4.3.17 s_mat_coef()	24
4.4.3.18 s_mat_copie()	24
4.4.3.19 s_mat_e()	25
4.4.3.20 s_mat_free()	25
4.4.3.21 s_mat_h()	25
4.4.3.22 s_mat_id()	25
4.4.3.23 s_mat_l()	25
4.4.3.24 s_mat_new()	25
4.4.3.25 trace()	26
4.4.3.26 transpose()	26
4.5 Référence du fichier src/sm_test.c	26
4.5.1 Documentation des fonctions	26
4.5.1.1 run_tests()	27
4.6 Référence du fichier src/sm_util.c	27
4.6.1 Documentation des fonctions	27
4.6.1.1 egalD()	27
4.6.1.2 error_log()	28
4.6.1.3 max()	28
4.6.1.4 min()	28
4.7 Référence du fichier src/test.c	28
4.7.1 Documentation des fonctions	29

4.7.1.1 main()	29
4.8 Référence du fichier src/tp_1.c	29
4.8.1 Documentation des fonctions	30
4.8.1.1 determinant()	30
4.8.1.2 echelon_passage()	30
4.8.1.3 echelonner()	30
4.8.1.4 est_inversible()	30
4.8.1.5 inverse()	30
4.8.1.6 LU()	31
4.8.1.7 main()	31
4.8.1.8 pivot_sous_mat_j()	31
4.8.1.9 rang()	31
4.8.1.10 solve_triangulaire_inf()	31
4.8.1.11 solve_triangulaire_sup()	31
<b>Index</b>	<b>33</b>

# Chapitre 1

## Index des structures de données

### 1.1 Structures de données

Liste des structures de données avec une brève description :

<a href="#">simplemat_s</a> . . . . .	5
---------------------------------------	---





## Chapitre 2

# Index des fichiers

### 2.1 Liste des fichiers

Liste de tous les fichiers avec une brève description :

include/ <a href="#">simplemat.h</a> . . . . .	7
include/ <a href="#">sm_test.h</a> . . . . .	15
include/ <a href="#">sm_util.h</a> . . . . .	16
src/ <a href="#">simplemat.c</a> . . . . .	19
src/ <a href="#">sm_test.c</a> . . . . .	26
src/ <a href="#">sm_util.c</a> . . . . .	27
src/ <a href="#">test.c</a> . . . . .	28
src/ <a href="#">tp_1.c</a> . . . . .	29



## Chapitre 3

# Documentation des structures de données

### 3.1 Référence de la structure `simplemat_s`

```
#include <simplemat.h>
```

#### Champs de données

- unsigned int `num_rows`
- unsigned int `num_cols`
- double \*\* `data`

#### 3.1.1 Documentation des champs

##### 3.1.1.1 `data`

```
double ** simplemat_s::data
```

##### 3.1.1.2 `num_cols`

```
unsigned int simplemat_s::num_cols
```

##### 3.1.1.3 `num_rows`

```
unsigned int simplemat_s::num_rows
```

La documentation de cette structure a été générée à partir du fichier suivant :

- `include/simplemat.h`
- `src/simplemat.c`



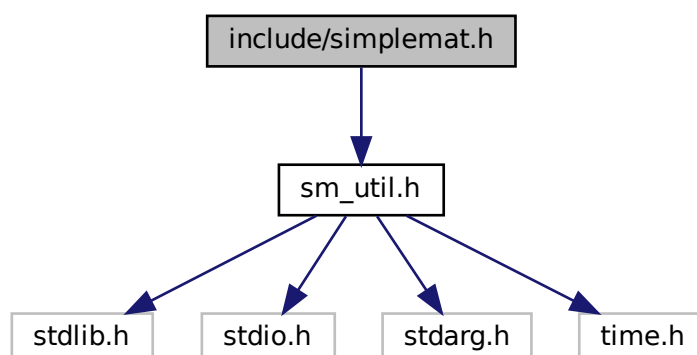
## Chapitre 4

# Documentation des fichiers

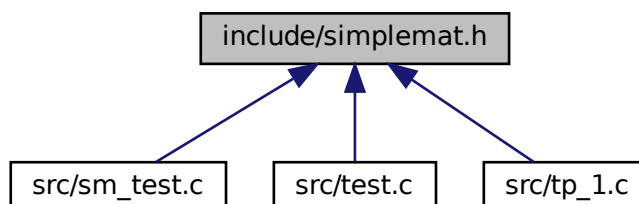
### 4.1 Référence du fichier include/simplemat.h

```
#include "sm_util.h"
```

Graphe des dépendances par inclusion de simplemat.h:



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



## Structures de données

— struct [simplemat\\_s](#)

## Macros

— #define [INVALID\\_ROWS](#) "Le nombre de ligne ne peut être nul"  
 — #define [INVALID\\_COLS](#) "Le nombre de colonne ne peut être nul"  
 — #define [INVALID\\_DIM](#) "La dimension ne peut être nulle"  
 — #define [INVALID\\_SYS\\_DIM](#) "Les matrices doivent avoir le même nombre de lignes"  
 — #define [INVALID\\_DIM\\_ADD](#) "Les dimensions des deux matrices doivent être identiques pour en faire la somme"  
 — #define [INVALID\\_DIM\\_MUL](#) "Le nombre de colonnes de la première matrice doit être égal au nombre de ligne de la seconde pour effectuer le produit de deux matrices"  
 — #define [INVALID\\_LIGNE](#) "L'indice de ligne ou colonne doit être inférieur à la taille de la matrice"  
 — #define [INVALID\\_INVERSION](#) "Pour les matrices d'interversion ou de type L(n,i,j,a) les indices doivent être différents"  
 — #define [INVALID\\_CARREE](#) "La matrice doit être carrée"  
 — #define [INVALID\\_INV](#) "La matrice n'est pas inversible"  
 — #define [INVALID\\_DIM\\_MUL\\_D](#) "La matrice de droite n'est pas carrée"  
 — #define [INVALID\\_DIM\\_MUL\\_G](#) "La matrice de gauche n'est pas carrée"

## Définitions de type

— typedef struct [simplemat\\_s](#) [s\\_mat](#)

## Fonctions

— [s\\_mat](#) \* [s\\_mat\\_new](#) (unsigned int num\_row, unsigned int num\_cols)  
 — [s\\_mat](#) \* [s\\_mat\\_carree](#) (unsigned int n)  
 — [s\\_mat](#) \* [s\\_mat\\_copie](#) ([s\\_mat](#) \*matrice)  
 — [s\\_mat](#) \* [s\\_mat\\_id](#) (unsigned int n)  
 — [s\\_mat](#) \* [s\\_mat\\_coef](#) (unsigned int n, unsigned int m, double valeurs[], unsigned int nbVal)  
 — [s\\_mat](#) \* [mat\\_alea\\_entier](#) (unsigned int n, unsigned int m, int [min](#), int [max](#))  
 — [s\\_mat](#) \* [mat\\_alea](#) (unsigned int n, unsigned int m, int [min](#), int [max](#))  
 — [s\\_mat](#) \* [s\\_mat\\_e](#) (unsigned int n, unsigned int i, unsigned j)  
 — [s\\_mat](#) \* [s\\_mat\\_h](#) (unsigned int n, unsigned int i, double a)  
 — [s\\_mat](#) \* [s\\_mat\\_l](#) (unsigned int n, unsigned int i, unsigned int j, double a)  
 — [s\\_mat](#) \* [mat\\_add](#) ([s\\_mat](#) \*A, [s\\_mat](#) \*B)  
 — [s\\_mat](#) \* [mat\\_minus](#) ([s\\_mat](#) \*A, [s\\_mat](#) \*B)  
 — [s\\_mat](#) \* [mat\\_mul\\_ext](#) ([s\\_mat](#) \*matrice, double scalaire)  
 — [s\\_mat](#) \* [mat\\_oppose](#) ([s\\_mat](#) \*matrice)  
 — [s\\_mat](#) \* [mat\\_mul](#) ([s\\_mat](#) \*A, [s\\_mat](#) \*B)  
 — void [mat\\_mul\\_droite](#) ([s\\_mat](#) \*A, [s\\_mat](#) \*B)  
 — void [mat\\_mul\\_gauche](#) ([s\\_mat](#) \*A, [s\\_mat](#) \*B)  
 — void [s\\_mat\\_free](#) ([s\\_mat](#) \*mat)  
 — double [trace](#) ([s\\_mat](#) \*A)  
 — [s\\_mat](#) \* [transpose](#) ([s\\_mat](#) \*A)  
 — \_Bool [est\\_carree](#) ([s\\_mat](#) \*matrice)  
 — \_Bool [est\\_egal](#) ([s\\_mat](#) \*A, [s\\_mat](#) \*B)  
 — \_Bool [est\\_id](#) ([s\\_mat](#) \*A)  
 — \_Bool [est\\_nulle](#) ([s\\_mat](#) \*A)  
 — void [afficher\\_mat](#) ([s\\_mat](#) \*matrix)  
 — void [echange\\_deux\\_lignes](#) ([s\\_mat](#) \*A, unsigned int i, unsigned int j)

### 4.1.1 Documentation des macros

#### 4.1.1.1 INVALID\_CARREE

```
#define INVALID_CARREE "La matrice doit être carrée"
```

#### 4.1.1.2 INVALID\_COLS

```
#define INVALID_COLS "Le nombre de colonne ne peut être nul"
```

#### 4.1.1.3 INVALID\_DIM

```
#define INVALID_DIM "La dimension ne peut être nulle"
```

#### 4.1.1.4 INVALID\_DIM\_ADD

```
#define INVALID_DIM_ADD "Les dimensions des deux matrices doivent être identiques pour en faire la somme"
```

#### 4.1.1.5 INVALID\_DIM\_MUL

```
#define INVALID_DIM_MUL "Le nombre de colonnes de la première matrice doit être égal au nombre de ligne de la seconde pour effectuer le produit de deux matrices"
```

#### 4.1.1.6 INVALID\_DIM\_MUL\_D

```
#define INVALID_DIM_MUL_D "La matrice de droite n'est pas carrée"
```

#### 4.1.1.7 INVALID\_DIM\_MUL\_G

```
#define INVALID_DIM_MUL_G "La matrice de gauche n'est pas carrée"
```

#### 4.1.1.8 INVALID\_INV

```
#define INVALID_INV "La matrice n'est pas inversible"
```

#### 4.1.1.9 INVALID\_INVERSION

```
#define INVALID_INVERSION "Pour les matrices d'interversion ou de type L(n,i,j,a) les indices  
doivent être différents"
```

#### 4.1.1.10 INVALID\_LIGNE

```
#define INVALID_LIGNE "L'indice de ligne ou colonne doit être inférieur à la taille de la  
matrice"
```

#### 4.1.1.11 INVALID\_ROWS

```
#define INVALID_ROWS "Le nombre de ligne ne peut être nul"
```

#### 4.1.1.12 INVALID\_SYS\_DIM

```
#define INVALID_SYS_DIM "Les matrices doivent avoir le même nombre de lignes"
```

### 4.1.2 Documentation des définitions de type

#### 4.1.2.1 s\_mat

```
typedef struct simplemat\_s s_mat
```

### 4.1.3 Documentation des fonctions



#### 4.1.3.1 afficher\_mat()

```
void afficher_mat (
    s_mat * matrix )
```

Affiche une matrice à l'écran

#### 4.1.3.2 echange\_deux\_lignes()

```
void echange_deux_lignes (
    s_mat * A,
    unsigned int i,
    unsigned int j )
```

Échange deux lignes

#### 4.1.3.3 est\_carree()

```
_Bool est_carree (
    s_mat * matrice )
```

La matrice est-elle carrée?

#### 4.1.3.4 est\_egal()

```
_Bool est_egal (
    s_mat * A,
    s_mat * B )
```

Deux matrices sont-elles égales?

#### 4.1.3.5 est\_id()

```
_Bool est_id (
    s_mat * A )
```

Renvoie vrai si la matrice est la matrice identité

#### 4.1.3.6 est\_nulle()

```
_Bool est_nulle (
    s_mat * A )
```

Renvoie vrai si la matrice est nulle

#### 4.1.3.7 mat\_add()

```
s_mat* mat_add (
    s_mat * A,
    s_mat * B )
```

Structure d'espace vectoriel

Addition de deux matrices

#### 4.1.3.8 mat\_alea()

```
s_mat* mat_alea (
    unsigned int n,
    unsigned int m,
    int min,
    int max )
```

Matrice aléatoires de flottants

#### 4.1.3.9 mat\_alea\_entier()

```
s_mat* mat_alea_entier (
    unsigned int n,
    unsigned int m,
    int min,
    int max )
```

Matrice aléatoires

Matrice aléatoires d'entier

#### 4.1.3.10 mat\_minus()

```
s_mat* mat_minus (
    s_mat * A,
    s_mat * B )
```

Soustraction de deux matrices

#### 4.1.3.11 mat\_mul()

```
s_mat* mat_mul (
    s_mat * A,
    s_mat * B )
```

Structure d'algèbre

Multiplication de deux matrices Renvoie une matrice produit de A et de B

#### 4.1.3.12 mat\_mul\_droite()

```
void mat_mul_droite (
    s_mat * A,
    s_mat * B )
```

Multiplication de deux matrices multiplie la matrice A par une matrice carrée B  $A \leftarrow A * B$

#### 4.1.3.13 mat\_mul\_ext()

```
s_mat* mat_mul_ext (
    s_mat * matrice,
    double scalaire )
```

Multiplication par un scalaire

#### 4.1.3.14 mat\_mul\_gauche()

```
void mat_mul_gauche (
    s_mat * A,
    s_mat * B )
```

Multiplication de deux matrices multiplie la matrice B à gauche par la matrice carrée A  $B \leftarrow A * B$

#### 4.1.3.15 mat\_oppose()

```
s_mat* mat_oppose (
    s_mat * matrice )
```

Opposé d'une matrice

#### 4.1.3.16 s\_mat\_carree()

```
s_mat* s_mat_carree (
    unsigned int n )
```

Crée une matrice carrée

Matrice carrée

#### 4.1.3.17 s\_mat\_coef()

```
s_mat* s_mat_coef (
    unsigned int n,
    unsigned int m,
    double valeurs[],
    unsigned int nbVal )
```

Matrice coefficients donnés

#### 4.1.3.18 s\_mat\_copie()

```
s_mat* s_mat_copie (
    s_mat * matrice )
```

Copie une matrice

Copie d'une matrice

#### 4.1.3.19 s\_mat\_e()

```
s_mat* s_mat_e (
    unsigned int n,
    unsigned int i,
    unsigned int j )
```

Inversion de deux lignes

#### 4.1.3.20 s\_mat\_free()

```
void s_mat_free (
    s_mat * matrix )
```

Libère la mémoire occupée par une matrice

#### 4.1.3.21 s\_mat\_h()

```
s_mat* s_mat_h (
    unsigned int n,
    unsigned int i,
    double a )
```

Homothétie

#### 4.1.3.22 s\_mat\_id()

```
s_mat* s_mat_id (
    unsigned int n )
```

Crée une matrice identité

Matrice identité

#### 4.1.3.23 s\_mat\_l()

```
s_mat* s_mat_l (
    unsigned int n,
    unsigned int i,
    unsigned int j,
    double a )
```

#### 4.1.3.24 s\_mat\_new()

```
s_mat* s_mat_new (
    unsigned int num_rows,
    unsigned int num_cols )
```

Constructeurs Allocation de la mémoire d'une matrice allocation d'un pointeur vers un s\_mat

Allocation d'un tableau de pointeurs

Allocation de n\*m double

#### 4.1.3.25 trace()

```
double trace (
    s_mat * A )
```

Renvoie la trace d'une matrice

#### 4.1.3.26 transpose()

```
s_mat* transpose (
    s_mat * A )
```

Renvoie la transposée d'une matrice

## 4.2 Référence du fichier include/sm\_test.h

### Fonctions

— int [run\\_tests](#) (void)

#### 4.2.1 Documentation des fonctions

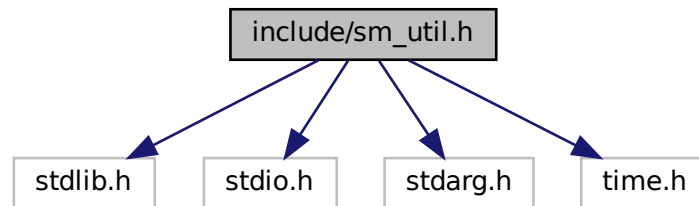
##### 4.2.1.1 run\_tests()

```
int run_tests (
    void )
```

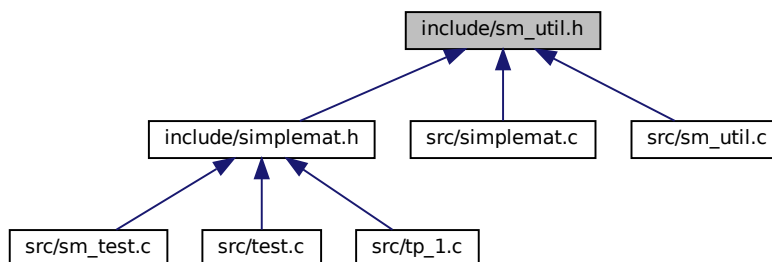
### 4.3 Référence du fichier include/sm\_util.h

```
#include <stdlib.h>
#include <stdio.h>
#include <stdarg.h>
#include <time.h>
```

Graphe des dépendances par inclusion de sm\_util.h:



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



### Macros

```
— #define DEBUG TRUE 1
— #define ROUGE "\033[0;31m"
— #define RESET "\033[0m"
— #define VERT "\033[0;32m"
— #define VIOLET "\033[0;35m"
— #define LOG_ERROR(fmt) error_log(stderr, __FILE__, __LINE__, ROUGE fmt RESET);
— #define ALLOC_CHECK(ptr)
```

### Fonctions

```
— void error_log (FILE *stream, const char *file_name, unsigned int line, const char *format)
— void my_log (FILE *stream, const char *file_name, unsigned int line, const char *format)
— _Bool egalD (double a, double b, double epsilon)
— double max (double num1, double num2)
— double min (double num1, double num2)
```

### 4.3.1 Documentation des macros

#### 4.3.1.1 ALLOC\_CHECK

```
#define ALLOC_CHECK(  
    ptr )
```

Valeur :

```
if (!ptr) { \  
    fprintf(stderr, "%s:%d NULL POINTER: %s n", \  
        __FILE__, __LINE__, (#ptr)); \  
    exit(-1); \  
} \  

```

#### 4.3.1.2 DEBUG\_TRUE

```
#define DEBUG_TRUE 1
```

#### 4.3.1.3 LOG\_ERROR

```
#define LOG_ERROR(  
    fmt ) error_log(stderr, __FILE__, __LINE__, ROUGE fmt RESET);
```

#### 4.3.1.4 RESET

```
#define RESET "\033[0m"
```

#### 4.3.1.5 ROUGE

```
#define ROUGE "\033[0;31m"
```

#### 4.3.1.6 VERT

```
#define VERT "\033[0;32m"
```

#### 4.3.1.7 VIOLET

```
#define VIOLET "\033[0;35m"
```

### 4.3.2 Documentation des fonctions

#### 4.3.2.1 egalD()

```
_Bool egalD (  
    double a,  
    double b,  
    double epsilon )
```

Renvoie vrai si deux doubles sont égaux à epsilon près

#### 4.3.2.2 error\_log()

```
void error_log (  
    FILE * stream,  
    const char * file_name,  
    unsigned int line,  
    const char * format )
```

#### 4.3.2.3 max()

```
double max (  
    double num1,  
    double num2 )
```

maximum

#### 4.3.2.4 min()

```
double min (  
    double num1,  
    double num2 )
```

minimum

#### 4.3.2.5 my\_log()

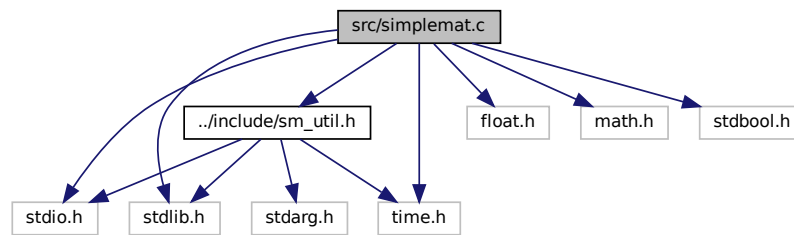
```
void my_log (  
    FILE * stream,  
    const char * file_name,  
    unsigned int line,  
    const char * format )
```



## 4.4 Référence du fichier src/simplemat.c

```
#include <stdio.h>
#include <stdlib.h>
#include <float.h>
#include <math.h>
#include <time.h>
#include <stdbool.h>
#include "../include/sm_util.h"
```

Graphe des dépendances par inclusion de simplemat.c:



## Structures de données

— struct [simplemat\\_s](#)

## Macros

- #define [INVALID\\_ROWS](#) "Le nombre de ligne ne peut être nul"
- #define [INVALID\\_COLS](#) "Le nombre de colonne ne peut être nul"
- #define [INVALID\\_DIM](#) "La dimension ne peut être nulle"
- #define [INVALID\\_DIM\\_ADD](#) "Les dimensions des deux matrices doivent être identiques pour en faire la somme"
- #define [INVALID\\_DIM\\_MUL](#) "Le nombre de colonnes de la première matrice doit être égal au nombre de ligne de la seconde pour effectuer le produit de deux matrices"
- #define [INVALID\\_DIM\\_MUL\\_D](#) "La matrice de droite n'est pas carrée"
- #define [INVALID\\_DIM\\_MUL\\_G](#) "La matrice de gauche n'est pas carrée"
- #define [INVALID\\_LIGNE](#) "L'indice de ligne ou colonne doit être inférieur à la taille de la matrice"
- #define [INVALID\\_INVERSION](#) "Pour les matrices d'interversion ou de type L(n,i,j,a) les indices doivent être différents"
- #define [INVALID\\_CARREE](#) "La matrice doit être carrée"

## Définitions de type

— typedef struct [simplemat\\_s s\\_mat](#)

## Fonctions

- `s_mat * s_mat_new` (unsigned int num\_rows, unsigned int num\_cols)
- `s_mat * s_mat_copie` (`s_mat *matrice`)
- `s_mat * s_mat_carree` (unsigned int n)
- `s_mat * s_mat_id` (unsigned int n)
- `s_mat * s_mat_h` (unsigned int n, unsigned int i, double a)
- `s_mat * s_mat_e` (unsigned int n, unsigned int i, unsigned int j)
- `s_mat * s_mat_l` (unsigned int n, unsigned int i, unsigned int j, double a)
- `s_mat * s_mat_coef` (unsigned int n, unsigned int m, double valeurs[], unsigned int nbVal)
- `s_mat * mat_alea_entier` (unsigned int n, unsigned int m, int min, int max)
- `s_mat * mat_alea` (unsigned int n, unsigned int m, int min, int max)
- `void s_mat_free` (`s_mat *matrix`)
- `void afficher_mat` (`s_mat *matrix`)
- `s_mat * mat_mul_ext` (`s_mat *matrice`, double scalaire)
- `s_mat * mat_add` (`s_mat *A`, `s_mat *B`)
- `s_mat * mat_minus` (`s_mat *A`, `s_mat *B`)
- `s_mat * mat_oppose` (`s_mat *matrice`)
- `s_mat * mat_mul` (`s_mat *A`, `s_mat *B`)
- `void mat_mul_gauche` (`s_mat *A`, `s_mat *B`)
- `void mat_mul_droite` (`s_mat *A`, `s_mat *B`)
- `_Bool est_carree` (`s_mat *matrice`)
- `_Bool est_egal` (`s_mat *A`, `s_mat *B`)
- `_Bool est_nulle` (`s_mat *A`)
- `_Bool est_id` (`s_mat *A`)
- `double trace` (`s_mat *A`)
- `s_mat * transpose` (`s_mat *A`)
- `void echange_deux_lignes` (`s_mat *A`, unsigned int i, unsigned int j)

### 4.4.1 Documentation des macros

#### 4.4.1.1 INVALID\_CARREE

```
#define INVALID_CARREE "La matrice doit être carrée"
```

#### 4.4.1.2 INVALID\_COLS

```
#define INVALID_COLS "Le nombre de colonne ne peut être nul"
```

#### 4.4.1.3 INVALID\_DIM

```
#define INVALID_DIM "La dimension ne peut être nulle"
```

#### 4.4.1.4 INVALID\_DIM\_ADD

```
#define INVALID_DIM_ADD "Les dimensions des deux matrices doivent être identiques pour en faire la somme"
```

#### 4.4.1.5 INVALID\_DIM\_MUL

```
#define INVALID_DIM_MUL "Le nombre de colonnes de la première matrice doit être égal au nombre  
de ligne de la seconde pour effectuer le produit de deux matrices"
```

#### 4.4.1.6 INVALID\_DIM\_MUL\_D

```
#define INVALID_DIM_MUL_D "La matrice de droite n'est pas carrée"
```

#### 4.4.1.7 INVALID\_DIM\_MUL\_G

```
#define INVALID_DIM_MUL_G "La matrice de gauche n'est pas carrée"
```

#### 4.4.1.8 INVALID\_INVERSION

```
#define INVALID_INVERSION "Pour les matrices d'interversion ou de type L(n,i,j,a) les indices  
doivent être différents"
```

#### 4.4.1.9 INVALID\_LIGNE

```
#define INVALID_LIGNE "L'indice de ligne ou colonne doit être inférieur à la taille de la  
matrice"
```

#### 4.4.1.10 INVALID\_ROWS

```
#define INVALID_ROWS "Le nombre de ligne ne peut être nul"
```

### 4.4.2 Documentation des définitions de type

#### 4.4.2.1 s\_mat

```
typedef struct simplemat_s s_mat
```

### 4.4.3 Documentation des fonctions

#### 4.4.3.1 afficher\_mat()

```
void afficher_mat (
    s_mat * matrix )
```

Affiche une matrice à l'écran

#### 4.4.3.2 echange\_deux\_lignes()

```
void echange_deux_lignes (
    s_mat * A,
    unsigned int i,
    unsigned int j )
```

Échange deux lignes

#### 4.4.3.3 est\_carree()

```
_Bool est_carree (
    s_mat * matrice )
```

La matrice est-elle carrée?

#### 4.4.3.4 est\_egal()

```
_Bool est_egal (
    s_mat * A,
    s_mat * B )
```

Deux matrices sont-elles égales?

#### 4.4.3.5 est\_id()

```
_Bool est_id (
    s_mat * A )
```

Renvoie vrai si la matrice est la matrice identité

#### 4.4.3.6 est\_nulle()

```
_Bool est_nulle (
    s_mat * A )
```

Renvoie vrai si la matrice est nulle

#### 4.4.3.7 mat\_add()

```
s_mat* mat_add (
    s_mat * A,
    s_mat * B )
```

Addition de deux matrices

#### 4.4.3.8 mat\_alea()

```
s_mat* mat_alea (
    unsigned int n,
    unsigned int m,
    int min,
    int max )
```

Matrice aléatoires de flottants

#### 4.4.3.9 mat\_alea\_entier()

```
s_mat* mat_alea_entier (
    unsigned int n,
    unsigned int m,
    int min,
    int max )
```

Matrice aléatoires d'entier

#### 4.4.3.10 mat\_minus()

```
s_mat* mat_minus (
    s_mat * A,
    s_mat * B )
```

Soustraction de deux matrices

#### 4.4.3.11 mat\_mul()

```
s_mat* mat_mul (
    s_mat * A,
    s_mat * B )
```

Multiplication de deux matrices Renvoie une matrice produit de A et de B

#### 4.4.3.12 mat\_mul\_droite()

```
void mat_mul_droite (
    s_mat * A,
    s_mat * B )
```

Multiplication de deux matrices multiplie la matrice A par une matrice carrée B  $A \leftarrow A * B$

#### 4.4.3.13 mat\_mul\_ext()

```
s_mat* mat_mul_ext (
    s_mat * matrice,
    double scalaire )
```

Multipliation par un scalaire

#### 4.4.3.14 mat\_mul\_gauche()

```
void mat_mul_gauche (
    s_mat * A,
    s_mat * B )
```

Multipliation de deux matrices multiplie la matrice B à gauche par la matrice carrée A  $B \leftarrow A * B$

#### 4.4.3.15 mat\_oppose()

```
s_mat* mat_oppose (
    s_mat * matrice )
```

Opposé d'une matrice

#### 4.4.3.16 s\_mat\_carree()

```
s_mat* s_mat_carree (
    unsigned int n )
```

Matrice carrée

#### 4.4.3.17 s\_mat\_coef()

```
s_mat* s_mat_coef (
    unsigned int n,
    unsigned int m,
    double valeurs[],
    unsigned int nbVal )
```

Matrice coefficients donnés

#### 4.4.3.18 s\_mat\_copie()

```
s_mat* s_mat_copie (
    s_mat * matrice )
```

Copie d'une matrice

#### 4.4.3.19 s\_mat\_e()

```
s_mat* s_mat_e (
    unsigned int n,
    unsigned int i,
    unsigned int j )
```

Inversion de deux lignes

#### 4.4.3.20 s\_mat\_free()

```
void s_mat_free (
    s_mat * matrix )
```

Libère la mémoire occupée par une matrice

#### 4.4.3.21 s\_mat\_h()

```
s_mat* s_mat_h (
    unsigned int n,
    unsigned int i,
    double a )
```

Homothétie

#### 4.4.3.22 s\_mat\_id()

```
s_mat* s_mat_id (
    unsigned int n )
```

Matrice identité

#### 4.4.3.23 s\_mat\_l()

```
s_mat* s_mat_l (
    unsigned int n,
    unsigned int i,
    unsigned int j,
    double a )
```

#### 4.4.3.24 s\_mat\_new()

```
s_mat* s_mat_new (
    unsigned int num_rows,
    unsigned int num_cols )
```

Constructeurs Allocation de la mémoire d'une matrice allocation d'un pointeur vers un s\_mat

Allocation d'un tableau de pointeurs

Allocation de n\*m double

#### 4.4.3.25 trace()

```
double trace (
    s_mat * A )
```

Renvoie la trace d'une matrice

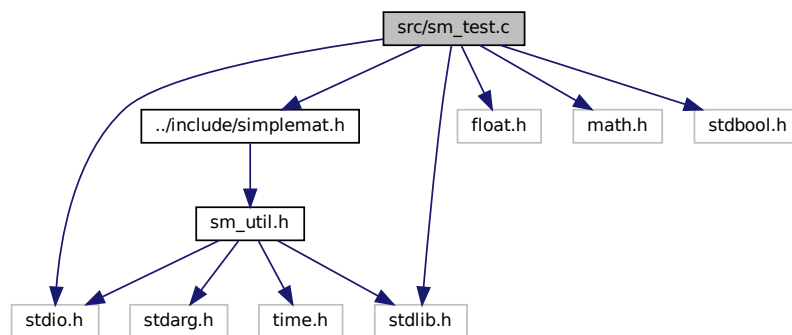
#### 4.4.3.26 transpose()

```
s_mat* transpose (
    s_mat * A )
```

Renvoie la transposée d'une matrice

## 4.5 Référence du fichier src/sm\_test.c

```
#include <stdio.h>
#include <stdlib.h>
#include <float.h>
#include <math.h>
#include <stdbool.h>
#include "../include/simplemat.h"
Graphe des dépendances par inclusion de sm_test.c:
```



## Fonctions

— int [run\\_tests](#) (void)

### 4.5.1 Documentation des fonctions



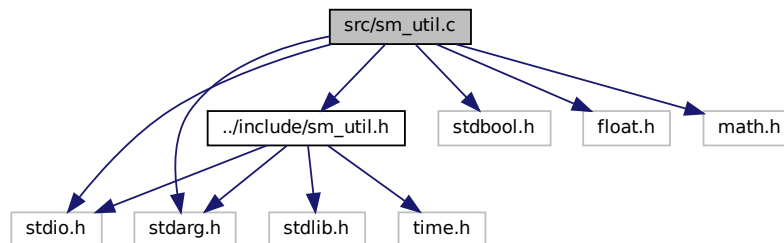
#### 4.5.1.1 run\_tests()

```
int run_tests (
    void )
```

## 4.6 Référence du fichier src/sm\_util.c

```
#include <stdio.h>
#include <stdarg.h>
#include <stdbool.h>
#include "../include/sm_util.h"
#include <float.h>
#include <math.h>
```

Graphe des dépendances par inclusion de sm\_util.c:



## Fonctions

- void [error\\_log](#) (FILE \*stream, const char \*file\_name, unsigned int line, const char \*format)
- double [max](#) (double num1, double num2)
- double [min](#) (double num1, double num2)
- \_Bool [egalD](#) (double a, double b, double epsilon)

### 4.6.1 Documentation des fonctions

#### 4.6.1.1 egalD()

```
_Bool egalD (
    double a,
    double b,
    double epsilon )
```

Renvoie vrai si deux doubles sont égaux à epsilon près

#### 4.6.1.2 error\_log()

```
void error_log (
    FILE * stream,
    const char * file_name,
    unsigned int line,
    const char * format )
```

#### 4.6.1.3 max()

```
double max (
    double num1,
    double num2 )
```

maximum

#### 4.6.1.4 min()

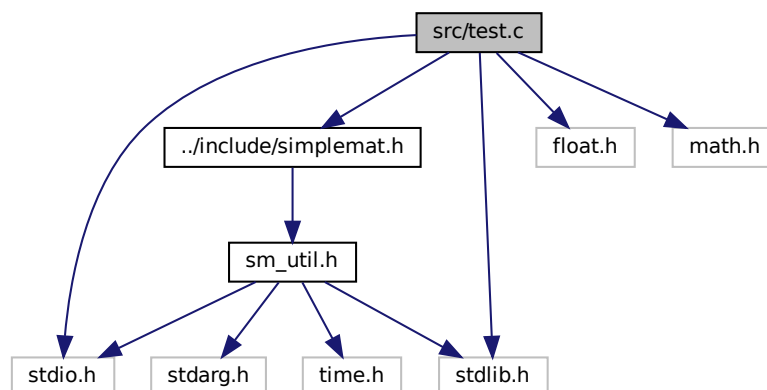
```
double min (
    double num1,
    double num2 )
```

minimum

## 4.7 Référence du fichier src/test.c

```
#include <stdio.h>
#include <stdlib.h>
#include <float.h>
#include <math.h>
#include "../include/simplemat.h"
```

Graphe des dépendances par inclusion de test.c:



## Fonctions

— int `main` (void)

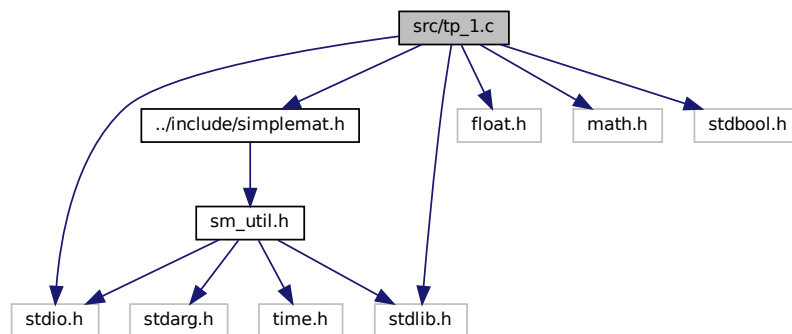
### 4.7.1 Documentation des fonctions

#### 4.7.1.1 `main()`

```
int main (
    void )
```

## 4.8 Référence du fichier src/tp\_1.c

```
#include <stdio.h>
#include <stdlib.h>
#include <float.h>
#include <math.h>
#include <stdbool.h>
#include "../include/simplemat.h"
Graphe des dépendances par inclusion de tp_1.c:
```



## Fonctions

— int `pivot_sous_mat_j` (`s_mat` \*matrice, unsigned int i, unsigned int j)  
 — void `echelon_passage` (`s_mat` \*matrice, `s_mat` \*echelonnee, `s_mat` \*passage)  
 — void `LU` (`s_mat` \*matrice, `s_mat` \*echelonnee, `s_mat` \*passage)  
 — `s_mat` \* `echelonner` (`s_mat` \*matrice)  
 — unsigned int `rang` (`s_mat` \*matrice)  
 — double `determinant` (`s_mat` \*matrice)  
 — `_Bool` `est_inversible` (`s_mat` \*matrice)  
 — `s_mat` \* `solve_triangulaire_sup` (`s_mat` \*U, `s_mat` \*B)  
 — `s_mat` \* `solve_triangulaire_inf` (`s_mat` \*L, `s_mat` \*B)  
 — `s_mat` \* `inverse` (`s_mat` \*matrice)  
 — int `main` (void)

## 4.8.1 Documentation des fonctions

### 4.8.1.1 determinant()

```
double determinant (
    s_mat * matrice )
```

Calcule le déterminant de la matrice

### 4.8.1.2 echelon\_passage()

```
void echelon_passage (
    s_mat * matrice,
    s_mat * echelonnee,
    s_mat * passage )
```

À partir d'une matrice (M) de taille  $n \times m$ , d'une copie de cette matrice (E:echelonnee) est de l'identité (P: passage) échelonne la matrice E et transforme la matrice P en une matrice de passage telle que  $P \cdot M = E$

La matrice P est triangulaire inférieure L La matrice E est triangulaire supérieure U

Il s'agit de la décomposition LU d'une matrice

### 4.8.1.3 echelonner()

```
s_mat* echelonner (
    s_mat * matrice )
```

Renvoie une matrice ligne équivalente à la matrice donnée en paramètre Les pivots sont calculés suivant la règle  $L_i \leftarrow L_i + \lambda L_k$

### 4.8.1.4 est\_inversible()

```
_Bool est_inversible (
    s_mat * matrice )
```

Renvoie vrai si la matrice est inversible

### 4.8.1.5 inverse()

```
s_mat* inverse (
    s_mat * matrice )
```

Résolution de  $AX=B$  où L triangulaire inférieure inversible et U triangulaire supérieure

```
s_mat* solve_lu(s_mat* L, s_mat* U, s_mat* B) {
```

```
} Retourne l'inverse d'une matrice carrée inversible. On commence par résoudre  $LX=I$  puis  $U Y= X$  Ainsi  $LU Y= LX$  ce qui équivaut à  $A Y=I$  Donc Y vaut  $A^{(-1)}$ 
```

#### 4.8.1.6 LU()

```
void LU (
    s_mat * matrice,
    s_mat * echelonnee,
    s_mat * passage )
```

À partir d'une matrice (M) de taille  $n \times m$ , d'une copie de cette matrice (E:echelonnee) est de l'identité (P: passage) échelonne la matrice E et transforme la matrice P en une matrice de passage telle que  $P \cdot E = M$

La matrice P est triangulaire inférieure L La matrice E est triangulaire supérieure U

Il s'agit de la décomposition LU d'une matrice

#### 4.8.1.7 main()

```
int main (
    void )
```

#### 4.8.1.8 pivot\_sous\_mat\_j()

```
int pivot_sous_mat_j (
    s_mat * matrice,
    unsigned int i,
    unsigned int j )
```

Renvoie la ligne du premier pivot de la sous matrices  $M[i:-1, j:-1]$  Si il n'y a pas de pivot renvoie -1

#### 4.8.1.9 rang()

```
unsigned int rang (
    s_mat * matrice )
```

#### 4.8.1.10 solve\_triangulaire\_inf()

```
s_mat* solve_triangulaire_inf (
    s_mat * L,
    s_mat * B )
```

Résolution de  $LX=B$  où L triangulaire inférieure et B a le même nombre de ligne que U. Renvoie une matrice de même taille que B où chaque colonne est solution de  $LX=B[:,i]$

#### 4.8.1.11 solve\_triangulaire\_sup()

```
s_mat* solve_triangulaire_sup (
    s_mat * U,
    s_mat * B )
```

Résolution de  $UX=B$  où U triangulaire supérieure et B a le même nombre de ligne que U. Renvoie une matrice de même taille que B où chaque colonne est solution de  $UX=B[:,i]$



# Index

afficher\_mat  
    simplemat.c, [22](#)  
    simplemat.h, [10](#)  
ALLOC\_CHECK  
    sm\_util.h, [17](#)  
  
data  
    simplemat\_s, [5](#)  
DEBUG\_TRUE  
    sm\_util.h, [17](#)  
determinant  
    tp\_1.c, [30](#)  
  
echange\_deux\_lignes  
    simplemat.c, [22](#)  
    simplemat.h, [11](#)  
echelon\_passage  
    tp\_1.c, [30](#)  
echelonner  
    tp\_1.c, [30](#)  
egalD  
    sm\_util.c, [27](#)  
    sm\_util.h, [18](#)  
error\_log  
    sm\_util.c, [27](#)  
    sm\_util.h, [18](#)  
est\_carree  
    simplemat.c, [22](#)  
    simplemat.h, [11](#)  
est\_egal  
    simplemat.c, [22](#)  
    simplemat.h, [11](#)  
est\_id  
    simplemat.c, [22](#)  
    simplemat.h, [11](#)  
est\_inversible  
    tp\_1.c, [30](#)  
est\_nulle  
    simplemat.c, [22](#)  
    simplemat.h, [11](#)  
  
include/simplemat.h, [7](#)  
include/sm\_test.h, [15](#)  
include/sm\_util.h, [16](#)  
INVALID\_CARREE  
    simplemat.c, [20](#)  
    simplemat.h, [8](#)  
INVALID\_COLS  
    simplemat.c, [20](#)  
    simplemat.h, [9](#)  
  
INVALID\_DIM  
    simplemat.c, [20](#)  
    simplemat.h, [9](#)  
INVALID\_DIM\_ADD  
    simplemat.c, [20](#)  
    simplemat.h, [9](#)  
INVALID\_DIM\_MUL  
    simplemat.c, [20](#)  
    simplemat.h, [9](#)  
INVALID\_DIM\_MUL\_D  
    simplemat.c, [21](#)  
    simplemat.h, [9](#)  
INVALID\_DIM\_MUL\_G  
    simplemat.c, [21](#)  
    simplemat.h, [9](#)  
INVALID\_INV  
    simplemat.h, [9](#)  
INVALID\_INVERSION  
    simplemat.c, [21](#)  
    simplemat.h, [10](#)  
INVALID\_LIGNE  
    simplemat.c, [21](#)  
    simplemat.h, [10](#)  
INVALID\_ROWS  
    simplemat.c, [21](#)  
    simplemat.h, [10](#)  
INVALID\_SYS\_DIM  
    simplemat.h, [10](#)  
inverse  
    tp\_1.c, [30](#)  
  
LOG\_ERROR  
    sm\_util.h, [17](#)  
LU  
    tp\_1.c, [30](#)  
  
main  
    test.c, [29](#)  
    tp\_1.c, [31](#)  
mat\_add  
    simplemat.c, [22](#)  
    simplemat.h, [11](#)  
mat\_alea  
    simplemat.c, [23](#)  
    simplemat.h, [12](#)  
mat\_alea\_entier  
    simplemat.c, [23](#)  
    simplemat.h, [12](#)  
mat\_minus  
    simplemat.c, [23](#)

- simplemat.h, 12
- mat\_mul
  - simplemat.c, 23
  - simplemat.h, 12
- mat\_mul\_droite
  - simplemat.c, 23
  - simplemat.h, 12
- mat\_mul\_ext
  - simplemat.c, 23
  - simplemat.h, 13
- mat\_mul\_gauche
  - simplemat.c, 24
  - simplemat.h, 13
- mat\_oppose
  - simplemat.c, 24
  - simplemat.h, 13
- max
  - sm\_util.c, 28
  - sm\_util.h, 18
- min
  - sm\_util.c, 28
  - sm\_util.h, 18
- my\_log
  - sm\_util.h, 18
- num\_cols
  - simplemat\_s, 5
- num\_rows
  - simplemat\_s, 5
- pivot\_sous\_mat\_j
  - tp\_1.c, 31
- rang
  - tp\_1.c, 31
- RESET
  - sm\_util.h, 17
- ROUGE
  - sm\_util.h, 17
- run\_tests
  - sm\_test.c, 26
  - sm\_test.h, 15
- s\_mat
  - simplemat.c, 21
  - simplemat.h, 10
- s\_mat\_carree
  - simplemat.c, 24
  - simplemat.h, 13
- s\_mat\_coef
  - simplemat.c, 24
  - simplemat.h, 13
- s\_mat\_copie
  - simplemat.c, 24
  - simplemat.h, 13
- s\_mat\_e
  - simplemat.c, 24
  - simplemat.h, 14
- s\_mat\_free
  - simplemat.c, 25
  - simplemat.h, 14
- s\_mat\_h
  - simplemat.c, 25
  - simplemat.h, 14
- s\_mat\_id
  - simplemat.c, 25
  - simplemat.h, 14
- s\_mat\_l
  - simplemat.c, 25
  - simplemat.h, 14
- s\_mat\_new
  - simplemat.c, 25
  - simplemat.h, 14
- simplemat.c
  - afficher\_mat, 22
  - echange\_deux\_lignes, 22
  - est\_carree, 22
  - est\_egal, 22
  - est\_id, 22
  - est\_nulle, 22
  - INVALID\_CARREE, 20
  - INVALID\_COLS, 20
  - INVALID\_DIM, 20
  - INVALID\_DIM\_ADD, 20
  - INVALID\_DIM\_MUL, 20
  - INVALID\_DIM\_MUL\_D, 21
  - INVALID\_DIM\_MUL\_G, 21
  - INVALID\_INVERSION, 21
  - INVALID\_LIGNE, 21
  - INVALID\_ROWS, 21
  - mat\_add, 22
  - mat\_alea, 23
  - mat\_alea\_entier, 23
  - mat\_minus, 23
  - mat\_mul, 23
  - mat\_mul\_droite, 23
  - mat\_mul\_ext, 23
  - mat\_mul\_gauche, 24
  - mat\_oppose, 24
  - s\_mat, 21
  - s\_mat\_carree, 24
  - s\_mat\_coef, 24
  - s\_mat\_copie, 24
  - s\_mat\_e, 24
  - s\_mat\_free, 25
  - s\_mat\_h, 25
  - s\_mat\_id, 25
  - s\_mat\_l, 25
  - s\_mat\_new, 25
  - trace, 25
  - transpose, 26
- simplemat.h
  - afficher\_mat, 10
  - echange\_deux\_lignes, 11
  - est\_carree, 11
  - est\_egal, 11
  - est\_id, 11



- est\_nulle, [11](#)
- INVALID\_CARREE, [8](#)
- INVALID\_COLS, [9](#)
- INVALID\_DIM, [9](#)
- INVALID\_DIM\_ADD, [9](#)
- INVALID\_DIM\_MUL, [9](#)
- INVALID\_DIM\_MUL\_D, [9](#)
- INVALID\_DIM\_MUL\_G, [9](#)
- INVALID\_INV, [9](#)
- INVALID\_INVERSION, [10](#)
- INVALID\_LIGNE, [10](#)
- INVALID\_ROWS, [10](#)
- INVALID\_SYS\_DIM, [10](#)
- mat\_add, [11](#)
- mat\_alea, [12](#)
- mat\_alea\_entier, [12](#)
- mat\_minus, [12](#)
- mat\_mul, [12](#)
- mat\_mul\_droite, [12](#)
- mat\_mul\_ext, [13](#)
- mat\_mul\_gauche, [13](#)
- mat\_oppose, [13](#)
- s\_mat, [10](#)
- s\_mat\_carree, [13](#)
- s\_mat\_coef, [13](#)
- s\_mat\_copie, [13](#)
- s\_mat\_e, [14](#)
- s\_mat\_free, [14](#)
- s\_mat\_h, [14](#)
- s\_mat\_id, [14](#)
- s\_mat\_l, [14](#)
- s\_mat\_new, [14](#)
- trace, [15](#)
- transpose, [15](#)
- simplemat\_s, [5](#)
  - data, [5](#)
  - num\_cols, [5](#)
  - num\_rows, [5](#)
- sm\_test.c
  - run\_tests, [26](#)
- sm\_test.h
  - run\_tests, [15](#)
- sm\_util.c
  - egalD, [27](#)
  - error\_log, [27](#)
  - max, [28](#)
  - min, [28](#)
- sm\_util.h
  - ALLOC\_CHECK, [17](#)
  - DEBUG\_TRUE, [17](#)
  - egalD, [18](#)
  - error\_log, [18](#)
  - LOG\_ERROR, [17](#)
  - max, [18](#)
  - min, [18](#)
  - my\_log, [18](#)
  - RESET, [17](#)
  - ROUGE, [17](#)
  - VERT, [17](#)
  - VIOLET, [17](#)
- solve\_triangulaire\_inf
  - tp\_1.c, [31](#)
- solve\_triangulaire\_sup
  - tp\_1.c, [31](#)
- src/simplemat.c, [19](#)
- src/sm\_test.c, [26](#)
- src/sm\_util.c, [27](#)
- src/test.c, [28](#)
- src/tp\_1.c, [29](#)
- test.c
  - main, [29](#)
- tp\_1.c
  - determinant, [30](#)
  - echelon\_passage, [30](#)
  - echelonner, [30](#)
  - est\_inversible, [30](#)
  - inverse, [30](#)
  - LU, [30](#)
  - main, [31](#)
  - pivot\_sous\_mat\_j, [31](#)
  - rang, [31](#)
  - solve\_triangulaire\_inf, [31](#)
  - solve\_triangulaire\_sup, [31](#)
- trace
  - simplemat.c, [25](#)
  - simplemat.h, [15](#)
- transpose
  - simplemat.c, [26](#)
  - simplemat.h, [15](#)
- VERT
  - sm\_util.h, [17](#)
- VIOLET
  - sm\_util.h, [17](#)