Simple Mat

Généré par Doxygen 1.9.1

Chapitre 1

Index des structures de données

1.1 Structures de données

Liste des structures de données avec une brève description :

Chapitre 2

Index des fichiers

2.1 Liste des fichiers

Liste de tous les fichiers avec une brève description :

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

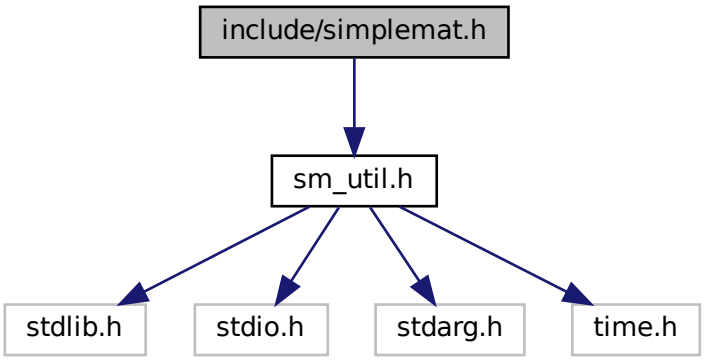
Chapitre 4

Documentation des fichiers

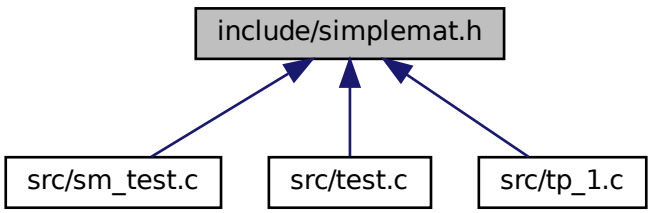
4.1 Référence du fichier include/simplemat.h

```
#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 )
```

Soustration 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<-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<-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 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

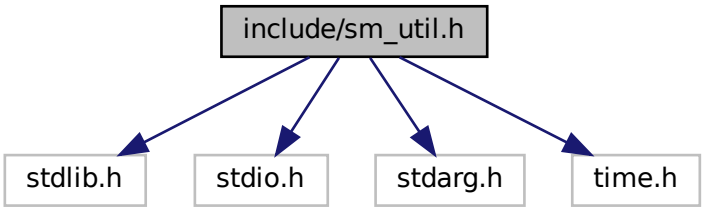
4.2.1.1 run_tests()

```
int run_tests (
            void  )
```

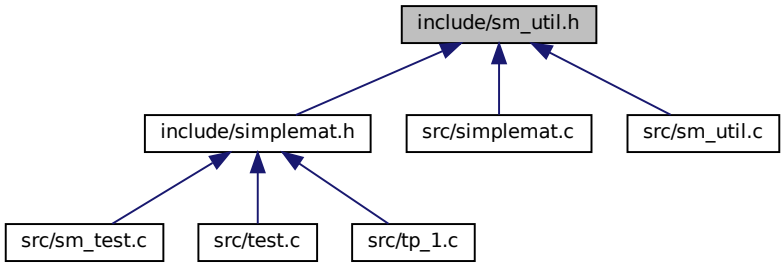
4.3 Référence du fichier include/sm_util.h

```
#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

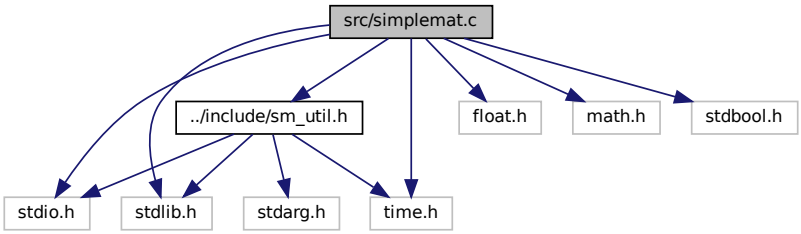
4.3.2.5 my_log()

```
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 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 )
```

Soustration 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<-A∗B

4.4.3.13 mat_mul_ext()

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

Multiplication par un scalaire

4.4.3.14 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<-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 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()

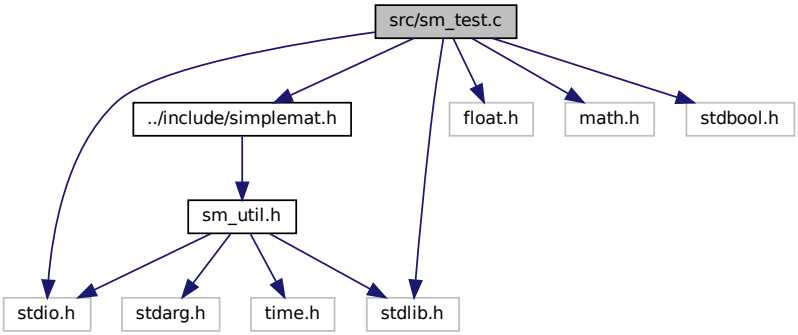
```
s_mat* transpose (
            s_mat * A )
```

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

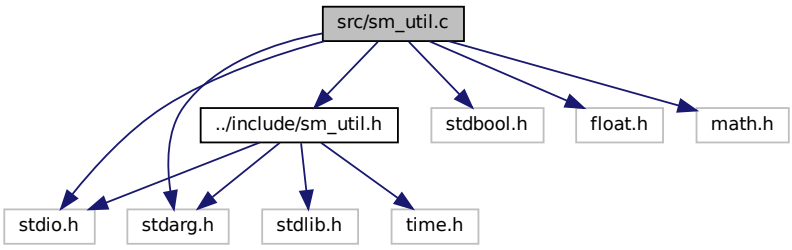
4.5.1.1 run_tests()

```
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()

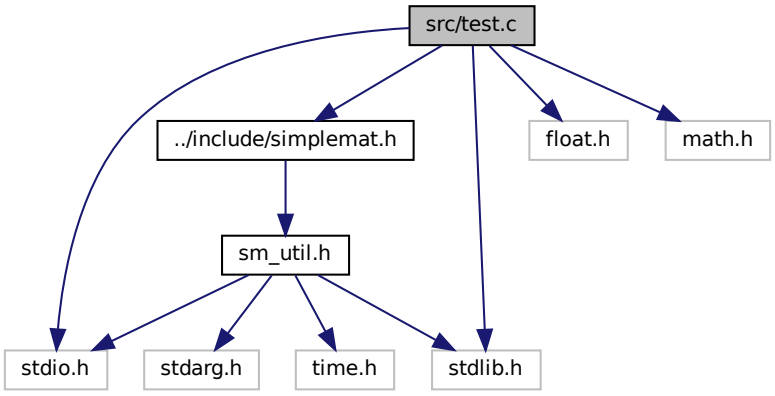
```
double min (
            double num1,
            double num2 )
```

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

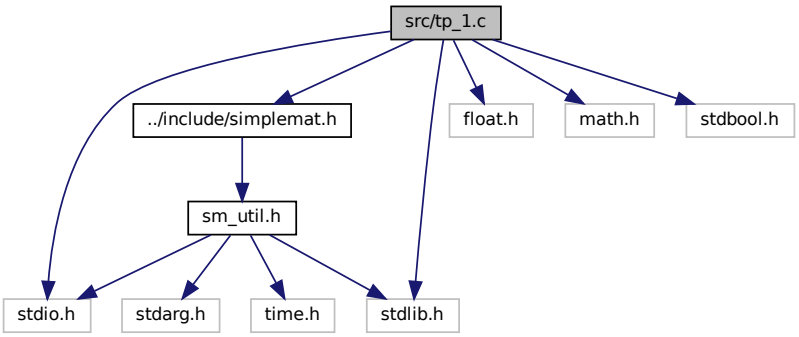
4.7.1.1 main()

```
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 x 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∗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<—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 x 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∗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