Projet 2 d'algorithmique numérique

Generated by Doxygen 1.8.17

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Namespace Documentation

3.1 partie1_q1 Namespace Reference

Functions

- def calc_ti (A, T, i)
- def calc_tji (A, T, i, j)
- def facto_cholesky_dense_REC (A, T, i)
- def facto_cholesky_dense (A)
- def main_test (name, entree, res, hope)
- def print_summary (function_name, test_result, number_of_test)

Variables

- list function_name = ["calc_ti","calc_tji","facto_cholesky_dense_REC","facto_cholesky_dense"]
- dictionary test_result = {}
- dictionary number_of_test = {}
- A = np.array([[1,2],[2,3]])
- T = np.zeros([2,2])
- int i = 0
- def res = calc_ti(A,T,i)
- int hope = 1
- int j = 0

3.1.1 Function Documentation

3.1.1.1 calc_ti()

```
def partie1_q1.calc_ti (
              A,
              T,
              i )
```

```
calc_ti(...)

returns the value of tii of the dense Cholesky factorization

Parameters
----------

A : positive definite square matrix

T : the result of dense Cholesky factorization

i : positive integer between 0 and the dimension of A-1

Returns
-------

out : the value of tii

Time complexity
---------------

O(2i-1)

Space complexity
----------------

O(1)
```

3.1.1.2 calc_tji()

```
def partie1_q1.calc_tji (
              A,
              T,
              i,
              j )
```

```
calc_tji(...)

returns the value of tji of the dense Cholesky factorization

Parameters
----------

A : positive definite square matrix

T : the result of dense Cholesky factorization

i : positive integer between 0 and the dimension of A-1

j : positive integer between 0 and i

Returns
-------
```

```
out : the value of tji

Time complexity
---------------

O(2i-1)

Space complexity
----------------

O(1)
```

3.1.1.3 facto_cholesky_dense()

```
def partie1_q1.facto_cholesky_dense (
              A )
```

```
facto_cholesky_dense(...)

returns the matrix of the dense Cholesky factorization

Parameters
----------

A : positive definite square matrix

Returns
-------

out : the matrix of the dense Cholesky factorization

Time complexity
---------------

O(1/3*n**3)

Space complexity
----------------

O(n**2)
```

3.1.1.4 facto_cholesky_dense_REC()

```
def partie1_q1.facto_cholesky_dense_REC (
              A,
              T,
              i )
```

```
facto_cholesky_dense_REC(...)

completes T with the value of tii and tji for j from 0 to i-1 of the dense Cholesky factorization and returns

Parameters
----------

A : positive definite square matrix
```

```
T : the result of dense Cholesky factorization

i : positive integer between 0 and the dimension of A-1

Returns
-------

out : itself with a call on next index

Time complexity
---------------

O((n-i+1)(2i-1))

Space complexity
----------------

O(1)
```

3.1.1.5 main_test()

def partie1_q1.main_test (
name,
entree,
res,
hope)

3.1.1.6 print_summary()

def partie1_q1.print_summary (
function_name,
test_result,
number_of_test)

3.1.2 Variable Documentation

3.1.2.1 A

```
partie1_q1.A = np.array([[1,2],[2,3]])
```

3.1.2.2 function_name

```
list partie1_q1.function_name = ["calc_ti","calc_tji","facto_cholesky_dense_REC","facto_cholesky_dense"]
```

3.1.2.3 hope

```
int partie1_q1.hope = 1
```

3.1.2.4 i

```
int partie1_q1.i = 0
```

3.1.2.5 j

```
int partie1_q1.j = 0
```

3.1.2.6 number_of_test

```
dictionary partie1_q1.number_of_test = {}
```

3.1.2.7 res

```
def partie1_q1.res = calc_ti(A,T,i)
```

3.1.2.8 T

```
partie1_q1.T = np.zeros([2,2])
```

3.1.2.9 test_result

```
dictionary partie1_q1.test_result = {}
```

3.2 partie1_q3 Namespace Reference

Functions

- def calc_rand_non_zero ()
- def generate_SPDSM (n, i)
- def print_summary (function_name, test_result, number_of_test)

Variables

- list function_name = ["calc_rand_non_zero","generate_SPDSM"]
- dictionary test_result = {}
- dictionary number_of_test = {}
- int nb_test = 22
- def f = calc_rand_non_zero
- def x = f()
- int n = 5
- int i = 2
- def A = f(n,i)
- bool check = True

3.2.1 Function Documentation

3.2.1.1 calc_rand_non_zero()

```
def partie1_q3.calc_rand_non_zero ( )
```

```
calc_rand_non_zero(...)

returns a random non zero number

Parameters
----------

None

Returns
-------

out : returns a random non zero number
```

3.2.1.2 generate_SPDSM()

```
def partie1_q3.generate_SPDSM (
              n,
              i )
```

```
generate_SPDSM(...)

returns a symetrical positive definite sparse matrix of nXn dim with i non zero extra-diagonal coefficents

Parameters
----------

n : positive integer wich is the size of the square matrix

i : even number, between 0 and n(n-1), of non zero extra-diagonal coefficients

Returns
-------

out : a symetrical positive definite sparse matrix of nXn dim with i non zero extra-diagonal coefficents
```

3.2.1.3 print_summary()

```
def partie1_q3.print_summary (
              function_name,
              test_result,
              number_of_test )
```

3.2.2 Variable Documentation

3.2.2.1 A

```
def partie1_q3.A = f(n,i)
```

3.2.2.2 check

```
bool partie1_q3.check = True
```

3.2.2.3 f

```
def partie1_q3.f = calc_rand_non_zero
```

3.2.2.4 function_name

```
list partie1_q3.function_name = ["calc_rand_non_zero","generate_SPDSM"]
```

3.2.2.5 i

```
int partie1_q3.i = 2
```

3.2.2.6 n

```
int partie1_q3.n = 5
```

3.2.2.7 nb_test

```
int partie1_q3.nb_test = 22
```

3.2.2.8 number_of_test

```
dictionary partie1_q3.number_of_test = {}
```

3.2.2.9 test_result

```
dictionary partie1_q3.test_result = {}
```

3.2.2.10 x

```
def partie1_q3.x = f()
```

3.3 partie1_q4 Namespace Reference

Functions

- def facto_cholesky_incomplete_REC (A, T, i)
- def facto_cholesky_incomplete (A)
- def compare_function (f1, f2)
- def main_test (name, entree, res, hope)
- def print_summary (function_name, test_result, number_of_test)

Variables

- list function_name = ["facto_cholesky_incomplete"]
- dictionary test_result = {}
- dictionary number_of_test = {}
- A = generate_SPDSM(5,2)
- T = np.zeros([4,4])
- int i = 0
- timer = time.time()
- int nb_timer = 10
- tuple timer_in = (time.time()-timer)/nb_timer
- timer_de = time.time()
- list res = [facto_cholesky_incomplete(A),timer]
- list hope = [facto_cholesky_dense(A),True]

3.3.1 Function Documentation

3.3.1.1 compare_function()

```
def partie1_q4.compare_function (
              f1,
              f2 )
```

```
compare_function(...)

print a graph with the curve of time execution of function f1 and f2 applied to a symetrical positive definite

Parameters
----------

f1 : function 1 which can work on an symetrical positive definite sparse matrix

f2 : function 2 which can work on an symetrical positive definite sparse matrix

Returns
-------

out : None
```

3.3.1.2 facto_cholesky_incomplete()

```
def partie1_q4.facto_cholesky_incomplete (
              A )
```

```
facto_cholesky_incomplete(...)

returns the matrix of the incomplete Cholesky factorization

Parameters
----------

A : positive definite square matrix

Returns
-------

out : the matrix of the incomplete Cholesky factorization

Time complexity
---------------

O(2*i**2)

Space complexity
----------------

O(n**2)
```

3.3.1.3 facto_cholesky_incomplete_REC()

```
def partie1_q4.facto_cholesky_incomplete_REC (
              A,
              T,
              i )
```

```
facto_cholesky_incomplete_REC(...)

completes T with the value of tii and tji for j from 0 to i-1 of the incomplete Cholesky factorization and ret

Parameters
----------

A : positive definite square matrix

T : the result of incomplete Cholesky factorization

i : positive integer between 0 and the dimension of A-1

Returns
-------

out : itself with a call on next index
```

3.3.1.4 main_test()

```
def partie1_q4.main_test (
              name,
              entree,
              res,
              hope )
```

3.3.1.5 print_summary()

```
def partie1_q4.print_summary (
              function_name,
              test_result,
              number_of_test )
```

3.3.2 Variable Documentation

3.3.2.1 A

```
partie1_q4.A = generate_SPDSM(5,2)
```

3.3.2.2 function_name

```
list partie1_q4.function_name = ["facto_cholesky_incomplete"]
```

3.3.2.3 hope

```
list partie1_q4.hope = [facto_cholesky_dense(A),True]
```

3.3.2.4 i

```
int partie1_q4.i = 0
```

3.3.2.5 nb_timer

```
int partie1_q4.nb_timer = 10
```

3.3.2.6 number_of_test

```
dictionary partie1_q4.number_of_test = {}
```

3.3.2.7 res

```
list partie1_q4.res = [facto_cholesky_incomplete(A),timer]
```

3.3.2.8 T

```
partie1_q4.T = np.zeros([4,4])
```

3.3.2.9 test_result

```
dictionary partie1_q4.test_result = {}
```

3.3.2.10 timer

```
tuple partie1_q4.timer = time.time()
```

3.3.2.11 timer_de

```
partie1_q4.timer_de = time.time()
```

3.3.2.12 timer_in

```
tuple partie1_q4.timer_in = (time.time()-timer)/nb_timer
```

3.4 partie1_q5 Namespace Reference

Functions

- def calc_inverse_matrice_T (T)
- def est_bon_preconditionneur (T, A)
- def test_est_bon_preconditionneur ()

3.4.1 Function Documentation

3.4.1.1 calc_inverse_matrice_T()

```
def partie1_q5.calc_inverse_matrice_T (
              T )
```

3.4.1.2 est_bon_preconditionneur()

```
def partie1_q5.est_bon_preconditionneur (
              T,
              A )
```

```
est_bon_preconditionneur(...)

checks if T is a good preconditioner of A

Parameters
----------

A : positive definite square matrix

T : the preconditioner

Returns
-------

out : boolean telling if T is a good preconditioner of A
```

3.4.1.3 test_est_bon_preconditionneur()

```
def partie1_q5.test_est_bon_preconditionneur ( )
```

3.5 partie2 Namespace Reference

Functions

- def prod (A, B)
- def trans (A)
- def conjugate_gradient (A, b, x)
- def triangular_solve_down (A, b)
- def triangular_solve_up (A, b)
- def solve (T, Tt, r)
- def conjugate_gradient_preconditioned (A, b, x)
- def test ()

Variables

- int n = 10

  End of program's function.
- A = generate_SPDSM(n,20)
- b = np.ones((n,1))
- x = np.zeros((n,1))
- def sol = conjugate_gradient(A,b,x)
- def sol2 = conjugate_gradient_preconditioned(A,b,x)
- int threshold = 10∗∗(-6)

  Tests.
- def res = prod(A,x)
- int flag = 0
- T = facto_cholesky_incomplete(A)
- def Tt = trans(T)
- y1
- y2
- fig = plt.subplot()
- linestyle
- label

3.5.1 Function Documentation

3.5.1.1 conjugate_gradient()

```
def partie2.conjugate_gradient (
              A,
              b,
              x )
```

```
conjugate_gradient(...)

returns the solution to the equation Ax = b

Parameters
----------

A : positive definite square matrix
b : vector solution of the equation Ax = b
x : initial estimation of the searched vector

Returns
-------

out : The vector x, solution to the equation Ax = b

Time complexity
---------------

inferior to n**3

Space complexity
----------------

O(n)
```

3.5.1.2 conjugate_gradient_preconditioned()

```
def partie2.conjugate_gradient_preconditioned (
              A,
              b,
              x )
```

```
conjugate_gradient_preconditioned(...)

returns the solution to the equation Ax = b

Parameters
----------

A : positive definite square matrix
b : vector solution of the equation Ax = b
x : initial estimation of the searched vector

Returns
-------

out : The vector x, solution to the equation Ax = b

Time complexity
---------------

inferior to n**3

Space complexity
----------------

O(n**2)
```

3.5.1.3 prod()

```
def partie2.prod (
              A,
              B )
```

```
prod(...)

returns the transpose of a matrix.

Parameters
----------

A : a matrix of dimension n,m
B : a matrix of dimension n',m'


Returns
-------

out : The product of the two given matrix

Time complexity
---------------

O(n*m')

Space complexity
----------------

O(1)
```

3.5.1.4 solve()

```
def partie2.solve (
              T,
              Tt,
              r )
```

```
solve(...)

returns the solution x to the equation T*Tt*x = b

Parameters
----------

T : a lower triangular matrix
Tt : a upper triangular matrix
b : vector solution of the equation T*Tt*x = b
x : initial estimation of the searched vector

Returns
-------

out : The vector x, solution to the equation T*Tt*x = b

Time complexity
---------------

O(n**2)

Space complexity
----------------

O(n)
```

3.5.1.5 test()

```
def partie2.test ( )
```

3.5.1.6 trans()

```
def partie2.trans (
              A )
```

```
trans(...)

returns the transpose of a matrix.

Parameters
----------

A : a matrix

Returns
-------

out : the transpose of the matrix

Time complexity
---------------

O(1)

Space complexity
----------------

O(1)
```

3.5.1.7 triangular_solve_down()

```
def partie2.triangular_solve_down (
              A,
              b )
```

```
triangular_solve_down(...)

returns the solution to the equation Ax = b where A is a lower triangular matrix

Parameters
----------

A : a lower triangular matrix
b : a column vector solution of the equation Ax = b

Returns
-------

out : The vector x, solution to the equation Ax = b

Time complexity
---------------

O(n**2)

Space complexity
----------------

O(n)
```

3.5.1.8 triangular_solve_up()

```
def partie2.triangular_solve_up (
              A,
              b )
```

```
triangular_solve_up(...)

returns the solution to the equation Ax = b where A is a upper triangular matrix

Parameters
----------

A : a upper triangular matrix
b : a column vector solution of the equation Ax = b

Returns
-------

out : The vector x, solution to the equation Ax = b

Time complexity
---------------

O(n**2)

Space complexity
----------------

O(n)
```

3.5.2 Variable Documentation

3.5.2.1 A

```
partie2.A = generate_SPDSM(n,20)
```

3.5.2.2 b

```
partie2.b = np.ones((n,1))
```

3.5.2.3 fig

```
partie2.fig = plt.subplot()
```

3.5.2.4 flag

```
int partie2.flag = 0
```

3.5.2.5 label

```
partie2.label
```

3.5.2.6 linestyle

```
partie2.linestyle
```

3.5.2.7 n

```
int partie2.n = 10
```

End of program's function.

Execution of all algorithms

3.5.2.8 res

```
def partie2.res = prod(A,x)
```

3.5.2.9 sol

```
def partie2.sol = conjugate_gradient(A,b,x)
```

3.5.2.10 sol2

```
def partie2.sol2 = conjugate_gradient_preconditioned(A,b,x)
```

3.5.2.11 T

```
partie2.T = facto_cholesky_incomplete(A)
```

3.5.2.12 threshold

```
int partie2.threshold = 10**(-6)
```

Tests.

3.5.2.13 Tt

```
def partie2.Tt = trans(T)
```

3.5.2.14 x

```
def partie2.x = np.zeros((n,1))
```

3.5.2.15 y1

```
partie2.y1
```

3.5.2.16 y2

```
partie2.y2
```

3.6 partie3_q1 Namespace Reference

Functions

- def genere_matrice_N (N)
- def print_summary (function_name, test_result, number_of_test)

Variables

- list function_name = ["genere_matrice_N"]
- dictionary test_result = {}
- dictionary number_of_test = {}
- int error_threshold = 10∗∗(-6)
- int N = 1
- def f = genere_matrice_N
- A
- b
- hope = np.diag([-4 for k in range(N∗∗2)])+np.diag([int((k+1)%N!=0) for k in range(N∗∗2-1)],1)+np.↩ diag([int((k+1)%N!=0) for k in range(N∗∗2-1)],-1)+np.diag([1 for k in range(N∗∗2-N)],-N)+np.diag([1 for k in range(N∗∗2-N)],N)

3.6.1 Function Documentation

3.6.1.1 genere_matrice_N()

```
def partie3_q1.genere_matrice_N (
              N )
```

```
genere_matrice_N(...)

returns the matrix A and b of the linear system of the 2D heat diffusion problem on a N by N plan

Parameters
----------

N : positive integer which is the size of the plan

Returns
-------

out : a tuple with the matrix A as first element and the vector b as second element

Space complexity
----------------

O(N**4)
```

3.6.1.2 print_summary()

```
def partie3_q1.print_summary (
              function_name,
              test_result,
              number_of_test )
```

3.6.2 Variable Documentation

3.6.2.1 A

```
partie3_q1.A
```

3.6.2.2 b

```
partie3_q1.b
```

3.6.2.3 error_threshold

```
int partie3_q1.error_threshold = 10**(-6)
```

3.6.2.4 f

```
def partie3_q1.f = genere_matrice_N
```

3.6.2.5 function_name

```
list partie3_q1.function_name = ["genere_matrice_N"]
```

3.6.2.6 hope

```
tuple partie3_q1.hope = np.diag([-4 for k in range(N**2)])+np.diag([int((k+1)%N!=0) for k in
range(N**2-1)],1)+np.diag([int((k+1)%N!=0) for k in range(N**2-1)],-1)+np.diag([1 for k in
range(N**2-N)],-N)+np.diag([1 for k in range(N**2-N)],N)
```

3.6.2.7 N

```
int partie3_q1.N = 1
```

3.6.2.8 number_of_test

```
dictionary partie3_q1.number_of_test = {}
```

3.6.2.9 test_result

```
dictionary partie3_q1.test_result = {}
```

3.7 partie3_q2_et_q3 Namespace Reference

Functions

- def probleme_mur_chaud (N, T_mur_chaud)
- def probleme_radiateur_centre (N, T_radiateur)
- def affiche_image (T)
- def main_test (name, entree, res, hope)
- def print_summary (function_name, test_result, number_of_test)

Variables

- list function_name = ["probleme_mur_chaud","probleme_radiateur_centre"]
- dictionary test_result = {}
- dictionary number_of_test = {}
- int N = 11
- int Temp = 50
- def f = probleme_mur_chaud
- def T = f(N,Temp)

3.7.1 Function Documentation

3.7.1.1 affiche_image()

```
def partie3_q2_et_q3.affiche_image (
              T )
```

```
affiche_image(...)

plot the temperature color map from the heat diffusion result matrix

Parameters
----------

T : the heat diffusion result matrix

Returns
-------

out : None
```

3.7.1.2 main_test()

```
def partie3_q2_et_q3.main_test (
              name,
              entree,
              res,
              hope )
```

3.7.1.3 print_summary()

```
def partie3_q2_et_q3.print_summary (
              function_name,
              test_result,
              number_of_test )
```

3.7.1.4 probleme_mur_chaud()

```
def partie3_q2_et_q3.probleme_mur_chaud (
              N,
              T_mur_chaud )
```

```
probleme_mur_chaud(...)

returns the coefficients of the matrix T of the problem of heat diffusion in a 2D plan of size N by N with a w

Parameters
----------

N : size of the 2D plan

T_mur_chaud : the temperature of the warm wall

Returns
-------

out : the matrix T of heat diffusion
```

3.7.1.5 probleme_radiateur_centre()

```
def partie3_q2_et_q3.probleme_radiateur_centre (
              N,
              T_radiateur )
```

```
probleme_radiateur_centre(...)

returns the coefficients of the matrix T of the problem of heat diffusion in a 2D plan of size N by N with a r

Parameters
----------

N : size of the 2D plan

T_radiateur : the temperature of the radiator

Returns
-------

out : the matrix T of heat diffusion result
```

3.7.2 Variable Documentation

3.7.2.1 f

```
def partie3_q2_et_q3.f = probleme_mur_chaud
```

3.7.2.2 function_name

```
list partie3_q2_et_q3.function_name = ["probleme_mur_chaud","probleme_radiateur_centre"]
```

3.7.2.3 N

```
int partie3_q2_et_q3.N = 11
```

3.7.2.4 number_of_test

```
dictionary partie3_q2_et_q3.number_of_test = {}
```

3.7.2.5 T

```
def partie3_q2_et_q3.T = f(N,Temp)
```

3.7.2.6 Temp

```
int partie3_q2_et_q3.Temp = 50
```

3.7.2.7 test_result

```
dictionary partie3_q2_et_q3.test_result = {}
```

Chapter 4

File Documentation

4.1 /home/nomprenom/IS104/is104-p2-12417/partie_1/partie1_q1.py File Reference

Namespaces

- partie1_q1

Functions

- def partie1_q1.calc_ti (A, T, i)
- def partie1_q1.calc_tji (A, T, i, j)
- def partie1_q1.facto_cholesky_dense_REC (A, T, i)
- def partie1_q1.facto_cholesky_dense (A)
- def partie1_q1.main_test (name, entree, res, hope)
- def partie1_q1.print_summary (function_name, test_result, number_of_test)

Variables

- list partie1_q1.function_name = ["calc_ti","calc_tji","facto_cholesky_dense_REC","facto_cholesky_dense"]
- dictionary partie1_q1.test_result = {}
- dictionary partie1_q1.number_of_test = {}
- partie1_q1.A = np.array([[1,2],[2,3]])
- partie1_q1.T = np.zeros([2,2])
- int partie1_q1.i = 0
- def partie1_q1.res = calc_ti(A,T,i)
- int partie1_q1.hope = 1
- int partie1_q1.j = 0

4.2 /home/nomprenom/IS104/is104-p2-12417/partie_1/partie1_q3.py File Reference

Namespaces

- partie1_q3

Functions

- def partie1_q3.calc_rand_non_zero ()
- def partie1_q3.generate_SPDSM (n, i)
- def partie1_q3.print_summary (function_name, test_result, number_of_test)

Variables

- list partie1_q3.function_name = ["calc_rand_non_zero","generate_SPDSM"]
- dictionary partie1_q3.test_result = {}
- dictionary partie1_q3.number_of_test = {}
- int partie1_q3.nb_test = 22
- def partie1_q3.f = calc_rand_non_zero
- def partie1_q3.x = f()
- int partie1_q3.n = 5
- int partie1_q3.i = 2
- def partie1_q3.A = f(n,i)
- bool partie1_q3.check = True

4.3 /home/nomprenom/IS104/is104-p2-12417/partie_1/partie1_q4.py File Reference

Namespaces

- partie1_q4

Functions

- def partie1_q4.facto_cholesky_incomplete_REC (A, T, i)
- def partie1_q4.facto_cholesky_incomplete (A)
- def partie1_q4.compare_function (f1, f2)
- def partie1_q4.main_test (name, entree, res, hope)
- def partie1_q4.print_summary (function_name, test_result, number_of_test)

Variables

- list partie1_q4.function_name = ["facto_cholesky_incomplete"]
- dictionary partie1_q4.test_result = {}
- dictionary partie1_q4.number_of_test = {}
- partie1_q4.A = generate_SPDSM(5,2)
- partie1_q4.T = np.zeros([4,4])
- int partie1_q4.i = 0
- partie1_q4.timer = time.time()
- int partie1_q4.nb_timer = 10
- tuple partie1_q4.timer_in = (time.time()-timer)/nb_timer
- partie1_q4.timer_de = time.time()
- list partie1_q4.res = [facto_cholesky_incomplete(A),timer]
- list partie1_q4.hope = [facto_cholesky_dense(A),True]

4.4 /home/nomprenom/IS104/is104-p2-12417/partie_1/partie1_q5.py File Reference

Namespaces

- partie1_q5

Functions

- def partie1_q5.calc_inverse_matrice_T (T)
- def partie1_q5.est_bon_preconditionneur (T, A)
- def partie1_q5.test_est_bon_preconditionneur ()

4.5 /home/nomprenom/IS104/is104-p2-12417/partie_2/partie2.py File Reference

Namespaces

- partie2

Functions

- def partie2.prod (A, B)
- def partie2.trans (A)
- def partie2.conjugate_gradient (A, b, x)
- def partie2.triangular_solve_down (A, b)
- def partie2.triangular_solve_up (A, b)
- def partie2.solve (T, Tt, r)
- def partie2.conjugate_gradient_preconditioned (A, b, x)
- def partie2.test ()

Variables

- int partie2.n = 10

  End of program's function.
- partie2.A = generate_SPDSM(n,20)
- partie2.b = np.ones((n,1))
- partie2.x = np.zeros((n,1))
- def partie2.sol = conjugate_gradient(A,b,x)
- def partie2.sol2 = conjugate_gradient_preconditioned(A,b,x)
- int partie2.threshold = 10∗∗(-6)

  Tests.
- def partie2.res = prod(A,x)
- int partie2.flag = 0
- partie2.T = facto_cholesky_incomplete(A)
- def partie2.Tt = trans(T)
- partie2.y1
- partie2.y2
- partie2.fig = plt.subplot()
- partie2.linestyle
- partie2.label

4.6 /home/nomprenom/IS104/is104-p2-12417/partie_3/partie3_q1.py File Reference

Namespaces

- partie3_q1

Functions

- def partie3_q1.genere_matrice_N (N)
- def partie3_q1.print_summary (function_name, test_result, number_of_test)

Variables

- list partie3_q1.function_name = ["genere_matrice_N"]
- dictionary partie3_q1.test_result = {}
- dictionary partie3_q1.number_of_test = {}
- int partie3_q1.error_threshold = 10∗∗(-6)
- int partie3_q1.N = 1
- def partie3_q1.f = genere_matrice_N
- partie3_q1.A
- partie3_q1.b
- partie3_q1.hope = np.diag([-4 for k in range(N∗∗2)])+np.diag([int((k+1)%N!=0) for k in range(N∗∗2-1)],1)+np.diag([int((k+1)%N!=0) for k in range(N∗∗2-1)],-1)+np.diag([1 for k in range(N∗∗2-N)],-N)+np.diag([1 for k in range(N∗∗2-N)],N)

4.7 /home/nomprenom/IS104/is104-p2-12417/partie_3/partie3_q2_et_↩ q3.py File Reference

Namespaces

- partie3_q2_et_q3

Functions

- def partie3_q2_et_q3.probleme_mur_chaud (N, T_mur_chaud)
- def partie3_q2_et_q3.probleme_radiateur_centre (N, T_radiateur)
- def partie3_q2_et_q3.affiche_image (T)
- def partie3_q2_et_q3.main_test (name, entree, res, hope)
- def partie3_q2_et_q3.print_summary (function_name, test_result, number_of_test)

Variables

- list partie3_q2_et_q3.function_name = ["probleme_mur_chaud","probleme_radiateur_centre"]
- dictionary partie3_q2_et_q3.test_result = {}
- dictionary partie3_q2_et_q3.number_of_test = {}
- int partie3_q2_et_q3.N = 11
- int partie3_q2_et_q3.Temp = 50
- def partie3_q2_et_q3.f = probleme_mur_chaud
- def partie3_q2_et_q3.T = f(N,Temp)

Index