# Projet

Corentin Marcou Walid Abed

19 février 2023

## Table des matières



## 1 Variables globales

```
#ifndef DONNEES_DU_PROBLEME
#define DONNEES_DU_PROBLEME

extern double epsilon;
extern double gama;
extern double lambda;

extern double a;
extern double b;
extern int N;
extern int M;
```

```
extern double det;

#endif
```

## 2 Point d'entrée du programme

```
#define _USE_MATH_DEFINES
#include <cmath>
#include <fstream>
#include <iostream>
#include <vector>

#include "boite_a_outils.h"
#include "donnees_du_probleme.h"
#include "maillage.h"
#include "noeud.h"
#include "triangle.h"

using namespace std;

double epsilon = 1;
double gama = 1;
double lambda = 1;
double a = 1;
double b = 1;
int N = 12;
int M = 12;
double det = abs(2 * (a / N) * (b / M));

double u_g(double y) { return sin(M_PI * y); }
double u_gpp(double y) { return -M_PI * M_PI * sin(M_PI * y); }

double u_d(double y) { return 0; }
double u_dpp(double y) { return 0; }

double f_eta(double x, double y) {
  return f_second_membre(u_g, u_d, u_gpp, u_dpp, x, y);
}

int main(void) {
  int I = (N - 1) * (M - 1);
  Maillage maille;
  vector<double> B_eta = scd_membre(f_eta, maille);
  cout << "Le second membre est:" << endl;
```

```
for (int k = 0; k < I && k < 30; k++)
  cout << B_eta[k] << endl;
// w_eta_h est la solution approchée.
vector<double> w_eta_h = inv_syst(B_eta, maille, 5);
cout << endl << "La solution approchée est:" << endl;
for (int k = 0; k < I && k < 30; k++)
  cout << w_eta_h[k] << endl;
cout << endl << "Les erreurs sont:" << endl;
vector<double> erreur = erreurs(u_eta, w_eta_h, maille);
for (double err : erreur)
  cout << err << endl;
vector<double> solution_exacte;
for (int k = 0; k < I; k++) {
  vector<double> xy = maille.int_coord(k);
  solution_exacte.push_back(u_eta(xy[0], xy[1]));
}
vector<double> ecart = solution_exacte - w_eta_h;
cout << endl << "sol exacte\tsol approchee\tecart" << endl;
for (int k = 0; k < I && k < 30; k++)
  cout << solution_exacte[k] << "   \t" << w_eta_h[k] << "   \t" << ecart[k]
       << endl;
cout << endl;

// Ecriture des fichiers:
ofstream file;

// Ecriture des solutions exactes:
file.open("solution_exacte.txt");
for (double d : solution_exacte)
  file << d << endl;
file.close();

// Ecriture des solutions approchées:
file.open("solution_approchee.txt");
for (double d : w_eta_h)
  file << d << endl;
file.close();


vector<double> X0(B_eta.size(), 1);
vector<double> AX0 = mat_vec(X0, maille);
vector<double> AB_eta = mat_vec(B_eta, maille);
cout << "A * X0  \tA * B_eta" << endl;
for (int k = 0; k < I && k < 30; k++)
        cout << AX0[k] << "   \t" << AB_eta[k] << endl;
cout << endl;
```

```
  return 0;
}
```

# 3 Classe Noeud

## 3.1 header

```
#ifndef NOEUD_H
#define NOEUD_H

class Noeud {
  // Coordonnées du noeud
  double x, y;

public:
  // Constructeur par défaut.
  Noeud(void);
  // Constructeur.
  Noeud(double x, double y);

  // Getters.
  double get_x(void);
  double get_y(void);
};

#endif
```

## 3.2 implémentation

```
#include "noeud.h"

Noeud::Noeud(void) {}

Noeud::Noeud(double x, double y) {
  this->x = x;
  this->y = y;
}

double Noeud::get_x(void) { return x; }

double Noeud::get_y(void) { return y; }
```

# 4 Classe Triangle

## 4.1 header

```
#ifndef TRIANGLE_H
#define TRIANGLE_H

#include <vector>

#include "donnees_du_probleme.h"
#include "noeud.h"

using namespace std;

class Triangle {
  // sommets du triangle.
  vector<Noeud> noeuds;

public:
  // Constructeur par défaut.
  Triangle(void);
  // Constructeurs.
  Triangle(Noeud n0, Noeud n1, Noeud n2);
  Triangle(vector<Noeud> noeuds);

  // Getter.
  vector<Noeud> get_noeuds(void);

  // Question 31:
  // Retourne la matrice B_T du triangle.
  vector<vector<double>> calc_mat_BT(void);

  // Retourne la matrice B_T du triangle avec une permutation des sommets.
  vector<vector<double>> calc_mat_BT(vector<int> permut);

  // Retourne l'inverse de la matrice B_T.
  vector<vector<double>> inv_mat_BT(void);

  // Question 37:
  vector<vector<double>> diff_terme(void);
  vector<vector<double>> convect_terme(void);
  vector<vector<double>> react_terme(void);
};

#endif
```

## 4.2 implémentation

```cpp
#include <cmath>
#include <iostream>
#include <vector>

#include "triangle.h"

using namespace std;

Triangle::Triangle(void) {}

Triangle::Triangle(Noeud n0, Noeud n1, Noeud n2) {
  this->noeuds = {n0, n1, n2};
}

Triangle::Triangle(vector<Noeud> noeuds) { this->noeuds = noeuds; }

vector<Noeud> Triangle::get_noeuds(void) { return noeuds; }

vector<vector<double>> Triangle::calc_mat_BT() {
  return {{noeuds[1].get_x() - noeuds[0].get_x(),
           noeuds[2].get_x() - noeuds[0].get_x()},
          {noeuds[1].get_y() - noeuds[0].get_y(),
           noeuds[2].get_y() - noeuds[0].get_y()}};
}

vector<vector<double>> Triangle::calc_mat_BT(vector<int> permut) {
  return {{noeuds[permut[1]].get_x() - noeuds[permut[0]].get_x(),
           noeuds[permut[2]].get_x() - noeuds[permut[0]].get_x()},
          {noeuds[permut[1]].get_y() - noeuds[permut[0]].get_y(),
           noeuds[permut[2]].get_y() - noeuds[permut[0]].get_y()}};
}

vector<vector<double>> Triangle::inv_mat_BT(void) {
  vector<vector<double>> BT = calc_mat_BT();
  return {{BT[1][1] / det, -BT[0][1] / det}, {-BT[1][0] / det, BT[0][0] / det}};
}

vector<vector<double>> Triangle::diff_terme(void) {
  vector<vector<double>> BI = inv_mat_BT();
  double d = det / 2;
  double d1 = BI[0][0] + BI[1][0];
  double d2 = BI[0][1] + BI[1][1];
  double m00 = (d1 * d1 + d2 * d2) * d;
  double m01 = (-BI[0][0] * d1 - BI[0][1] * d2) * d;
```

```
  double m02 = (-BI[1][0] * d1 - BI[1][1] * d2) * d;
  double m11 = (BI[0][0] * BI[0][0] + BI[0][1] * BI[0][1]) * d;
  double m12 = (BI[0][0] * BI[1][0] + BI[0][1] * BI[1][1]) * d;
  double m22 = (BI[1][0] * BI[1][0] + BI[1][1] * BI[1][1]) * d;
  return {{m00, m01, m02}, {m01, m11, m12}, {m02, m12, m22}};
}

vector<vector<double>> Triangle::convect_terme(void) {
  double c = det / 6;
  return {{-c, -c, -c}, {c, c, c}, {0, 0, 0}};
}

vector<vector<double>> Triangle::react_terme(void) {
  double r1 = det / 12;
  double r2 = det / 24;
  return {{r1, r2, r2}, {r2, r1, r2}, {r2, r2, r1}};
}
```

# 5 Classe Maillage

## 5.1 header

```
#ifndef MAILLAGE_H
#define MAILLAGE_H

#include <vector>

#include "donnees_du_probleme.h"
#include "noeud.h"
#include "triangle.h"

using namespace std;

class Maillage {
  // Subdivisions.
  vector<Triangle> triangulation;

public:
  // Constructeur.
  Maillage(void);

  // Getters.
  vector<Triangle> get_triangulation(void);

  // Question 9:
```

```
  // Retourne une subdivision uniforme de [-a, a] en N + 1 points et de [-b, b]
  // en M + 1 points.
  static vector<double> sub_div(double largeur, int nb_divisions);

  // Question 11:
  // Retourne le numéro global associé aux indices i et j.
  int num_gb(int i, int j);

  // Question 13:
  // Retourne les indices i et j à partir du numéro global s.
  vector<int> inv_num_gb(int s);

  // Question 14:
  // Retourne le numéro intérieur associé aux indices i et j.
  int num_int(int i, int j);

  // Question 16:
  // Retourne les indices i et j à partir du numéro intérieur k.
  vector<int> inv_num_int(int k);

  // Question 17:
  // Retourne le numéro global à partir du numéro intérieur k.
  int num_int_gb(int k);

  // Question 18:
  // Retourne le numéro intérieur à partir du numéro global s.
  int num_gb_int(int s);

  // Retourne le numéro global d'un noeud dans le maillage.
  int num_gb_noeud(Noeud noeud);

  // Retourne le numéro intérieur d'un noeud dans le maillage.
  int num_int_noeud(Noeud noeud);

  // Vérifie si le noeud est sur le bord.
  bool est_sur_le_bord(Noeud noeud);

  // Donne les coordonnées x et y à partir du numéro intérieur du noeud.
  vector<double> int_coord(int k);

  // Question 20:
  // Initialise le tableau de triangle dont la l-ème ligne contient le triangle
  // T_l.
  void init_maillage_TR(void);
};
```

```
#endif
```

## 5.2 implémentation

```
#include <cmath>

#include "maillage.h"

Maillage::Maillage(void) { this->init_maillage_TR(); }

vector<Triangle> Maillage::get_triangulation(void) { return triangulation; }

vector<double> Maillage::sub_div(double largeur, int nb_divisions) {
  vector<double> xi;
  for (int i = 0; i <= nb_divisions; i++)
    xi.push_back(-largeur + (2 * i * largeur) / nb_divisions);
  return xi;
}

int Maillage::num_gb(int i, int j) { return (N + 1) * j + i; }

vector<int> Maillage::inv_num_gb(int s) {
  int i = s % (N + 1);
  int j = s / (N + 1);
  return {i, j};
}

int Maillage::num_int(int i, int j) { return (N - 1) * (j - 1) + (i - 1); }

vector<int> Maillage::inv_num_int(int k) {
  int i = k % (N - 1) + 1;
  int j = k / (N - 1) + 1;
  return {i, j};
}

int Maillage::num_int_gb(int k) {
  vector<int> ij = this->inv_num_int(k);
  return this->num_gb(ij[0], ij[1]);
}

int Maillage::num_gb_int(int s) {
  vector<int> ij = this->inv_num_gb(s);
  return this->num_int(ij[0], ij[1]);
}

bool Maillage::est_sur_le_bord(Noeud noeud) {
```

```
  int i = round(N * (noeud.get_x() + a) / (2 * a));
  int j = round(M * (noeud.get_y() + b) / (2 * b));
  return i == 0 || j == 0 || i == N || j == M;
}

int Maillage::num_gb_noeud(Noeud noeud) {
  int i = round(N * (noeud.get_x() + a) / (2 * a));
  int j = round(M * (noeud.get_y() + b) / (2 * b));
  return this->num_gb(i, j);
}

int Maillage::num_int_noeud(Noeud noeud) {
  int i = round(N * (noeud.get_x() + a) / (2 * a));
  int j = round(M * (noeud.get_y() + b) / (2 * b));
  return this->num_int(i, j);
}

vector<double> Maillage::int_coord(int k) {
  vector<int> ij = this->inv_num_int(k);
  return {(2 * ij[0] - N) * a / N, (2 * ij[1] - M) * b / M};
}

void Maillage::init_maillage_TR(void) {
  // On génère la matrice des noeuds.
  vector<vector<Noeud>> noeuds;
  for (int i = 0; i <= N; i++) {
    vector<Noeud> colonne;
    double x = (2 * i - N) * a / N;
    for (int j = 0; j <= M; j++) {
      double y = (2 * j - M) * b / M;
      colonne.push_back(Noeud(x, y));
    }
    noeuds.push_back(colonne);
  }
  for (int j = 0; j < M; j++) {
    for (int i = 0; i < N; i++) {
      Noeud SO = noeuds[i][j];
      Noeud SE = noeuds[i + 1][j];
      Noeud NO = noeuds[i][j + 1];
      Noeud NE = noeuds[i + 1][j + 1];
      // Il y a 2 configurations possibles en fonction de la position du
      // rectangle du maillage qui nous intéresse. Dans chaque cas, il
      // faut déterminer les sommets du rectangle et les placer dans un
      // certain ordre.
      if ((i ^ j) & 1) {
        triangulation.push_back(Triangle(SO, SE, NO));
```

```
        triangulation.push_back(Triangle(SE, NO, NE));
      } else {
        triangulation.push_back(Triangle(SO, NO, NE));
        triangulation.push_back(Triangle(SO, SE, NE));
      }
    }
  }
}
```

# 6 Boite à outils

## 6.1 header

```
#ifndef BOITE_A_OUTILS
#define BOITE_A_OUTILS

#include <vector>

#include "donnees_du_probleme.h"
#include "maillage.h"
#include "triangle.h"

using namespace std;

// On définit l'addition de 2 vecteur de double (ils doivent être de même
// taille).
vector<double> operator+(vector<double> A, vector<double> B);

// On définit la soustraction de 2 vecteur de double (ils doivent être de même
// taille).
vector<double> operator-(vector<double> A, vector<double> B);

// On définit le produit d'un vecteur de double avec un double.
vector<double> operator*(double scalaire, vector<double> B);

// On définit le produit scalaire de 2 vecteur de double (ils doivent être de
// même taille).
double operator*(vector<double> A, vector<double> B);

// Retourne la plus grande valeur absolue du vecteur.
double max(vector<double> A);

// Question 3:
// Retourne f_eta (x, y).
double f_second_membre(double (*u_g)(double), double (*u_d)(double),
```

```
                        double (*u_gpp)(double), double (*u_dpp)(double),
                        double x, double y);

// Question 43.d:
// Calcule la prolongation du vecteur des noeuds intérieurs sur tous les
// noeuds.
vector<double> extend_vec(vector<double> V_int);

// Question 43.e:
// Calcule la restriction du vecteur des noeuds globaux sur tous les noeuds
// intérieurs.
vector<double> int_vec(vector<double> V_glb);

// Question 46.i:
// Retourne la norme L2 de la fonction v associée au vecteur V de taille I.
double norme_L2(vector<double> V, Maillage maille);

// Question 46.j:
// Retourne la norme L2 grad de la fonction v associée au vecteur V de taille I.
double norme_L2_grad(vector<double> V, Maillage maille);

// Question 44:
// Retourne le produit vectoriel A_eta * V.
vector<double> mat_vec(vector<double> V, Maillage maille);

// Question 45:
// Retourne le second membre B_eta du système linéaire A_eta * X = B_eta.
vector<double> scd_membre(double (*rhfs)(double, double), Maillage maille);

// Partie 4:
// Retourne une solution approchée du système linéaire A_eta * X = B_eta.
vector<double> inv_syst(vector<double> B_eta, Maillage maille,
                        int max_iteration);

// Question 49:
// Retournes les trois erreurs relatives.
vector<double> erreurs(double (*sol_exa)(double, double),
                        vector<double> sol_appr, Maillage maille);

// TEMPORAIRE
double u_eta(double x, double y);

#endif
```

## 6.2 implémentation

```
#define _USE_MATH_DEFINES
#include <cmath>
#include <iostream>

#include "boite_a_outils.h"

using namespace std;

vector<double> operator+(vector<double> A, vector<double> B) {
  vector<double> C;
  for (size_t i = 0; i < B.size(); ++i)
    C.push_back(A[i] + B[i]);
  return C;
}

vector<double> operator-(vector<double> A, vector<double> B) {
  vector<double> C;
  for (size_t i = 0; i < B.size(); ++i)
    C.push_back(A[i] - B[i]);
  return C;
}

vector<double> operator*(double scalaire, vector<double> B) {
  vector<double> C;
  for (size_t i = 0; i < B.size(); ++i)
    C.push_back(scalaire * B[i]);
  return C;
}

double operator*(vector<double> A, vector<double> B) {
  double C;
  for (size_t i = 0; i < B.size(); ++i)
    C += A[i] * B[i];
  return C;
}

double max(vector<double> A) {
  double max = 0;
  for (double i : A)
    max = i > 0 ? i > max ? i : max : -i > max ? -i : max;
  return max;
}

double f_second_membre(double (*u_g)(double), double (*u_d)(double),
```

```
                        double (*u_gpp)(double), double (*u_dpp)(double),
                        double x, double y) {
  return (epsilon * (a - x) * u_gpp(y) + epsilon * (a + x) * u_dpp(y) +
          gama * u_g(y) - gama * u_d(y) - lambda * (a - x) * u_g(y) -
          lambda * (a + x) * u_d(y)) /
         (2 * a);
}

vector<double> extend_vec(vector<double> V_int) {
  vector<double> V_glb;
  for (int i = 0; i <= N; i++)
    V_glb.push_back(0);
  int numInt = 0;
  for (int j = 1; j < M; j++) {
    V_glb.push_back(0);
    for (int i = 1; i < N; i++) {
      V_glb.push_back(V_int[numInt]);
      numInt++;
    }
    V_glb.push_back(0);
  }
  for (int i = 0; i <= N; i++)
    V_glb.push_back(0);
  return V_glb;
}

vector<double> int_vec(vector<double> V_glb) {
  vector<double> V_int;
  for (int j = 1; j < M; j++) {
    for (int i = 1; i < N; i++)
      V_int.push_back(V_glb[(N + 1) * j + i]);
  }
  return V_int;
}

double norme_L2(vector<double> V, Maillage maille) {
  vector<double> V_glb = extend_vec(V);
  vector<double> WW((N + 1) * (M + 1), 0);
  for (Triangle triangle : maille.get_triangulation()) {
    vector<Noeud> noeuds = triangle.get_noeuds();
    for (int i = 0; i < 3; i++) {
      int s = maille.num_gb_noeud(noeuds[i]);
      double res = 0;
      for (int j = 0; j < 3; j++) {
        int r = maille.num_gb_noeud(noeuds[j]);
        res += V_glb[r] * triangle.react_terme()[j][i];
```

```
      }
      WW[s] += res;
    }
  }
  vector<double> AV = int_vec(WW);
  return AV * V;
}

double norme_L2_grad(vector<double> V, Maillage maille) {
  vector<double> V_glb = extend_vec(V);
  vector<double> WW((N + 1) * (M + 1), 0);
  for (Triangle triangle : maille.get_triangulation()) {
    vector<Noeud> noeuds = triangle.get_noeuds();
    for (int i = 0; i < 3; i++) {
      int s = maille.num_gb_noeud(noeuds[i]);
      double res = 0;
      for (int j = 0; j < 3; j++) {
        int r = maille.num_gb_noeud(noeuds[j]);
        res += V_glb[r] * triangle.diff_terme()[j][i];
      }
      WW[s] += res;
    }
  }
  vector<double> AV = int_vec(WW);
  return AV * V;
}

vector<double> mat_vec(vector<double> V, Maillage maille) {
  vector<double> VV = extend_vec(V);
  vector<double> WW((N + 1) * (M + 1), 0);
  for (Triangle triangle : maille.get_triangulation()) {
    vector<Noeud> noeuds = triangle.get_noeuds();
    for (int i = 0; i < 3; i++) {
      int s = maille.num_gb_noeud(noeuds[i]);
      double res = 0;
      for (int j = 0; j < 3; j++) {
        int r = maille.num_gb_noeud(noeuds[j]);
        double prod2 = epsilon * triangle.diff_terme()[j][i] +
                       gama * triangle.convect_terme()[j][i] +
                       lambda * triangle.react_terme()[j][i];
        res += VV[r] * prod2;
      }
      WW[s] += res;
    }
  }
  return int_vec(WW);
```

```
}

vector<double> scd_membre(double (*rhfs)(double, double), Maillage maille) {
  vector<double> B((N - 1) * (M - 1), 0);
  for (Triangle triangle : maille.get_triangulation()) {
    vector<Noeud> noeuds = triangle.get_noeuds();
    for (int i = 0; i < 3; i++) {
      if (!maille.est_sur_le_bord(noeuds[i])) {
        vector<vector<double>> BT =
            triangle.calc_mat_BT({i, (i + 1) % 3, (i + 2) % 3});
        double res = 0;
        // FT(1/2, 0) = BT * (1/2, 0) + (x0, y0):
        vector<double> FT = {BT[0][0] / 2 + noeuds[i].get_x(),
                             BT[1][0] / 2 + noeuds[i].get_y()};
        // wk(FT(1/2, 0)) = 1/2
        res += rhfs(FT[0], FT[1]) / 12;
        // FT(0, 1/2) = BT * (0, 1/2) + (x0, y0):
        FT = {BT[0][1] / 2 + noeuds[i].get_x(),
              BT[1][1] / 2 + noeuds[i].get_y()};
        // wk(FT(0, 1/2)) = 1/2
        res += rhfs(FT[0], FT[1]) / 12;
        B[maille.num_int_noeud(noeuds[i])] += res;
      }
    }
  }
  return B;
}

vector<double> inv_syst(vector<double> B_eta, Maillage maille,
                        int max_iteration) {
  vector<double> X0(B_eta.size(), 1);
  vector<double> R0 = B_eta - mat_vec(X0, maille);
  vector<double> R0_etoile = R0;
  vector<double> W0 = R0;
  for (int j = 0; j < max_iteration; j++) {
    vector<double> AW0 = mat_vec(W0, maille);
    double alpha0 = (R0 * R0_etoile) / (AW0 * R0_etoile);
    vector<double> S0 = R0 - (alpha0 * AW0);
    vector<double> AS0 = mat_vec(S0, maille);
    double omega0 = (AS0 * S0) / (AS0 * AS0);
    vector<double> X1 = X0 + (alpha0 * W0) + (omega0 * S0);
    vector<double> R1 = S0 - (omega0 * AS0);
    double beta0 = ((R1 * R0_etoile) / (R0 * R0_etoile)) * (alpha0 / omega0);
    vector<double> W1 = R1 + (beta0 * (W0 - (omega0 * AW0)));
    R0 = R1;
    W0 = W1;
```

```
    X0 = X1;
  }
  return X0;
}

vector<double> erreurs(double (*sol_exa)(double, double),
                       vector<double> sol_appr, Maillage maille) {
  vector<double> w;
  int I = (N - 1) * (M - 1);
  for (int k = 0; k < I; k++) {
    vector<double> xy = maille.int_coord(k);
    w.push_back(sol_exa(xy[0], xy[1]));
  }
  vector<double> erreur = w - sol_appr;
  return {norme_L2(erreur, maille) / norme_L2(w, maille),
          norme_L2_grad(erreur, maille) / norme_L2_grad(w, maille),
          max(erreur) / max(w)};
}

double u_eta(double x, double y) {
  double A = (gama / epsilon - sqrt(gama * gama / epsilon / epsilon +
                                    4 * M_PI * M_PI + 4 * lambda / epsilon)) /
             2;
  double B = (gama / epsilon + sqrt(gama * gama / epsilon / epsilon +
                                    4 * M_PI * M_PI + 4 * lambda / epsilon)) /
             2;
  double C_1 = 1 / (exp(-A) - exp(A - 2 * B));
  double C_2 = 1 / (exp(-B) - exp(B - 2 * A));
  double U_x = C_1 * exp(A * x) + C_2 * exp(B * x);
  return U_x * sin(M_PI * y);
}
```