Gilles Teixidor
Laurent Chignac
1999-2000

DESS MSRO

OPTIMISATION NON LINEAIRE

Problème de la surface minimale

Table des matières

Chapitre 1

Analyse du problème - Formulation analytique

Le problème consiste à estimer la surface minimum qui s'appuie sur un contour donné. On discrétise la surface à l'aide d'un maillage formé de triangles.
Les variables du problème sont les hauteurs $(z_i)_{i=1..(N-2)^2}$ de chacun des points intérieurs du domaine. La surface ne dépends que de $Z = (z_i)$. Elle s'obtient en sommant la surface des triangles.

On notera :

- Δ : l' ensemble des triangles du maillage.
- S : l' application qui à un triangle τ associe sa surface.

La surface du maillage s'écrit alors :

$$f(Z) = \sum_{\tau \in \Delta} S(\tau)$$

avec $S(\tau) = \frac{\left\| \overrightarrow{AB} \wedge \overrightarrow{AC} \right\|}{2}$, si $\tau = (ABC)$

Il s'agit d'un problème d'optimisation non linéaire sans contraintes.

Chapitre 2

Etude numérique

La résolution numérique s'effectue par itérations successives à partir de la solution initiale $Z = \overrightarrow{0}$. On a choisi la méthode de Fletcher-Reeves pour résoudre ce problème de minimisation. Ce choix est justifié par une fonction objectif non quadratique.

2.1 La méthode de Fletcher-Reeves

La méthode s'inpspire du celle du gradient conjugué.

Algorithme :

1. choisir z_0.
   calculer $g_0 = \nabla f(z^0)$, et poser $d_0 = g_0; k = 0$.
2. Etape k.
   $z^{k+1} = z^k + \lambda_k d_k$ où λ_k minimise $g(\lambda) = f(z^k + \lambda d_k)$.
   $d_{k+1} = -g_{k+1} + \beta_k d_k$
   avec $\beta_k = \frac{\|g_{k+1}\|^2}{\|g_k\|^2}$ et $g_{k+1} = \nabla f(z^{k+1})$.

Une de nos difficultés a porté sur le calcul de ∇f :

$$\nabla f = \begin{pmatrix} \frac{\partial f}{\partial z_1} \\ \frac{\partial f}{\partial z_2} \\ . \\ . \\ . \\ \frac{\partial f}{\partial z_{(N-2)^2}} \end{pmatrix}$$

avec :

$$\frac{\partial f}{\partial z_i} = \frac{\partial \sum_{\tau \in \Delta} S(\tau)}{\partial z_i}$$

On examine les six triangles voisins du point z_i, la dérivée étant nulle ailleurs.
donc on a :

$$\frac{\partial f}{\partial z_i} = \frac{\partial \sum_{\tau \in V(i)} S(\tau)}{\partial z_i} = \sum_{\tau \in V(i)} \frac{\partial S(\tau)}{\partial z_i}$$

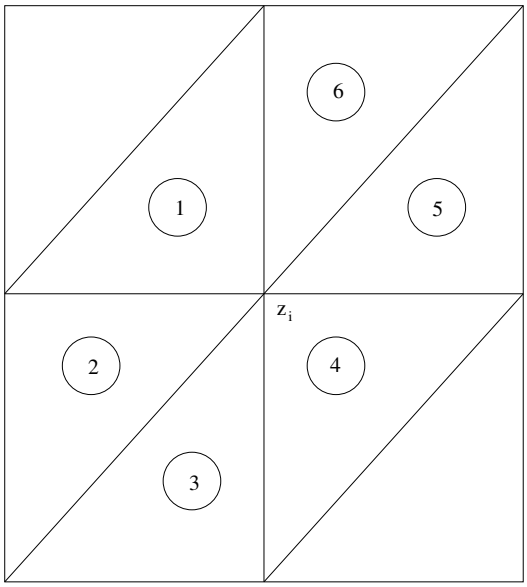

FIG. 2.1 – Triangles adjacents à z_i

où

$$S(\tau) = \frac{\left\|\overrightarrow{AB} \wedge \overrightarrow{AC}\right\|}{2} = \sqrt{u_1^2 + u_2^2 + u_3^2}$$

avec

$$u_1 = (y_B - y_A)(z_C - z_A) - (z_B - z_A)(y_C - y_A)$$
$$u_2 = (z_B - z_A)(x_C - x_A) - (x_B - x_A)(z_C - z_A)$$
$$u_3 = (x_B - x_A)(y_C - y_A) - (y_B - y_A)(x_C - x_A)$$

d'où

$$\frac{\partial S(\tau)}{\partial z_i} = \frac{\partial \|u\|}{\partial z_i} = \frac{\partial \sqrt{u_1^2 + u_2^2 + u_3^2}}{2\partial z_i}$$
$$\frac{\partial S(\tau)}{\partial z_i}(Z^k) = \frac{u_1(y_C - y_B) + u_2(x_B - x_C)}{2\sqrt{u_1^2 + u_2^2 + u_3^2}}$$

La formule est valable pour les six triangles voisins du point z_i.
Au total, on obtient $\frac{\partial f}{\partial z_i}(Z^k)$.

2.2 La méthode de dichotomie

A chaque itération k de l'algorithme de Fletcher-Reeves il est nécessaire de calculer le λ_k optimal, c'est à dire celui qui minimise la surface dans la direction de descente d_k.
On cherche ainsi à minimiser la fonction g définie par :

$$g(\lambda) = f(Z_k + \lambda d_k)$$

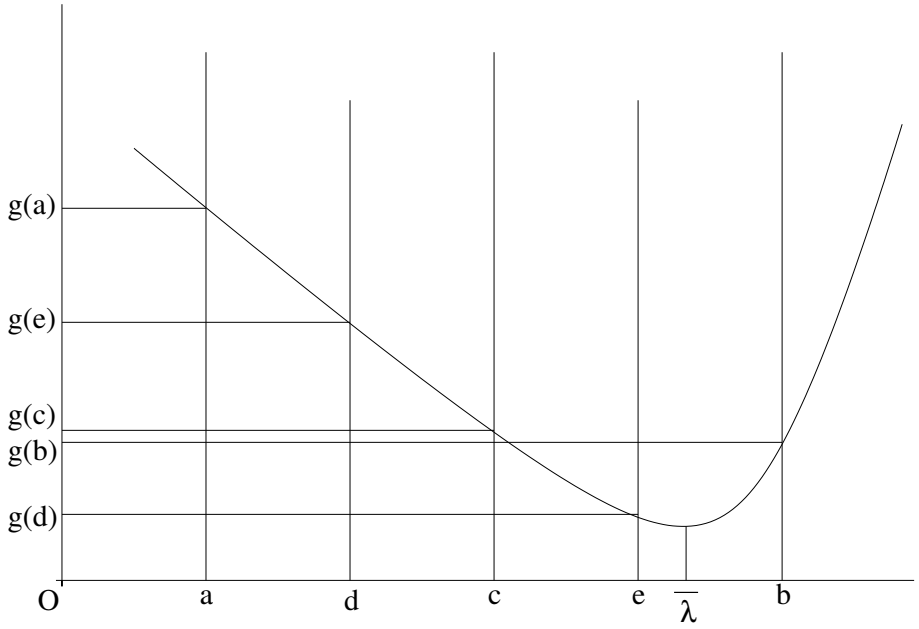

FIG. 2.2 – Exemple de fonction unimodale

On a choisi la méthode de dichotomie sans dérivée en supposant que g est unimodale. Principe de l'algorithme :

On utilise la propriété d'unimodalité de g, pour tous λ_1, λ_2 $\in$[A,B] tels que $\lambda_1 < \lambda_2$ on a :

$$g(\lambda_1) < g(\lambda_2) \Rightarrow \lambda_2 \geq \overline{\lambda}$$

$$g(\lambda_2) > g(\lambda_1) \Rightarrow \lambda_1 \leq \overline{\lambda}$$

A chaque itération, on cherche à éliminer deux des quatres intervalles :

- $[a, d]$
- $[d, c]$
- $[c, e]$
- $[e, d]$

Seulement deux tests sont nécessaires :

- si $g(c) > g(e)$ alors on conserve $[c, b]$
- si $g(d) < g(c)$ alors on conserve $[a, c]$
- sinon on conserve $[d, e]$

On arrête dès que la longeur de $[a, b]$ est inferieure à un ε choisi.

Code du source :

```
float minimisation1D(maillage xk,float dk[N-2][N-2]) {
  float lk,A=0.,B=5.;
  float a,b,c,d,e;
  float fc,fd,fe;
  float epsilon=0.01;
  a=A;b=B;c=0.5*(a+b);
  fc=CalculSurface(xk,c,dk);
```

```
  while ((b-a)>epsilon){
    d=0.5*(a+c); e=0.5*(c+b);
    fd=CalculSurface(xk,d,dk);
    fe=CalculSurface(xk,e,dk);
    if(fc>fe){
      a=c; c=e; fc=fe;
    }
    else {
      if(fd<fc){
b=c;c=d;fc=fd;
      }
      else {
a=d;b=e;
      }
    }
  }
  return(c);
}
```

Chapitre 3

Le programme

Afin d'utiliser les possibilités offertes par la programmation objet, nous avons choisi d'utiliser le langage C++ pour ce programme qui comporte 3 fichiers :

- Surface-Minimum.cpp
- mailleur.h
- mailleur.cpp

L'exécutable lit un fichier appelé Cond-Init ou sont stockées les hauteurs de six points particuliers du bord.
Il retourne en sortie le fichier Résultats qui contient les vecteurs X et Y ainsi que la matrice Z des hauteurs des points du maillage. Ce fichier, au format Scilab, permet de visualiser la surface en 3D.

L'algorithme de Fletcher-Reeves et la minimisation-1D sont implémentés dans le fichier Surface-Minimum.cpp.

La discrétisation du domaine est effectuée en début de d'exécution par le constructeur de la classe maillage, décrite dans "mailleur.cpp".
La classe maillage contient également :

- La méthode Refresh() qui fixe la solution initiale à zéro et interpôle les hauteurs du bord.
- La méthode surface() utilisée dans la minimisation-1D pour calculer la surface totale à un λ_k fixé.
- La méthode gradient() qui renvoie le vecteur, mis à jour uniquement après l'appel à la méthode calculer-gradient().
- La méthode calculer-gradient().

Chapitre 4

Code du source

4.1 Surface Minimum.cpp

```
                    Probleme de minimisation multidimentionnel
                          Gilles Texidor - Laurent Chignac
                          1999 - 2000
                          DESS MSRO
#include <stdio.h>
#include <iostream.h>
#include <stdlib.h>
#include <fstream.h>
#include "mailleur.h"
#include <math.h>



//*********************************************************************
float abs(float a){
  if(a>0) return(a); else return(-a);
}
//*********************************************************************
float CalculSurface(maillage xk,float lk,float dk[N-2][N-2]){
  float s;
  maillage xtemp(2.0);
  xtemp.refresh();
  for(int i=0;i<N-2;i++){
    for(int j=0;j<N-2;j++){
      xtemp.z[i+1][j+1]=xk.z[i+1][j+1]+lk*dk[i][j];
    }
  }
  s=xtemp.surface();
  return(s);
}
//*********************************************************************
float minimisation1D(maillage xk,float dk[N-2][N-2]) {
```

```
  float lk;
  float A=0.;
  float B=5.;
  float a,b,c,d,e;
  float fc,fd,fe;
  float epsilon=0.01;

  a=A;
  b=B;
  c=0.5*(a+b);
  fc=CalculSurface(xk,c,dk);

  while ((b-a)>epsilon){
    d=0.5*(a+c);
    e=0.5*(c+b);
    fd=CalculSurface(xk,d,dk);
    fe=CalculSurface(xk,e,dk);
    if(fc>fe){
      a=c;
      c=e;
      fc=fe;
    }
    else {
      if(fd<fc){
b=c;
c=d;
fc=fd;
      }
      else {
a=d;
b=e;
      }
    }
  }
  return(c);
}
//*********************************************************
//permet de calculer ||gradient||^2
//*********************************************************
float norme2_grad(float gradient[N-2][N-2]){
  float somme=0;
  for(int i=0;i<(N-2);i++){
    for(int j=0;j<(N-2);j++){
      somme+=gradient[i][j]*gradient[i][j];
    }
  }
  return(somme);
```

```
}
//*********************************************************
void Fletcher_Reeves(){
  float lk;
  float s1,s2,ds=2.;
  float epsilon=0.001;
  float dk[N-2][N-2];
  float bk;
  float norm2gk,norm2gkpun;
  int stop=0;

  maillage xk(2.0);
  xk.refresh_curv();


  xk.calculer_gradient();
  norm2gk=norme2_grad(xk.gradient);

  for(int i=0;i<N-2;i++){
    for(int j=0;j<N-2;j++){
  s1=xk.surface();
      dk[i][j]=-xk.gradient[i][j];
    }
  }
  s1=xk.surface();

  while (stop<2){
    lk=minimisation1D(xk,dk);
    for(int i=0;i<N-2;i++){
      for(int j=0;j<N-2;j++){
xk.z[i+1][j+1]=xk.z[i+1][j+1]+lk*dk[i][j];
      }
    }
    xk.calculer_gradient();
    norm2gkpun=norme2_grad(xk.gradient);
    bk=norm2gkpun/norm2gk;
    if(stop>=1) bk=0.;
    norm2gk=norm2gkpun;
    for(int i=0;i<N-2;i++){
      for(int j=0;j<N-2;j++){
dk[i][j]=-xk.gradient[i][j]+bk*dk[i][j];
      }
    }
    s2=xk.surface();
    ds=s1-s2;
    s1=s2;
    if (abs(ds)<epsilon) stop++; else stop=0;
```

```
  }
  xk.results();
}
//*********************************************************
//*********************************************************
main(){
  Fletcher_Reeves();
}
```

4.2 mailleur.h

```
#include <stdio.h>
#include <iostream.h>
#include <stdlib.h>
#include <fstream.h>

const
  int N=25;
const
  float X=2.,Y=2.;

struct point{
  float *x,*y,*z;
};

struct triangle{
  struct point a,b,c;
};
#ifndef maillage_H
#define maillage_H
//**************************declaration de la classe maillage*****************

class maillage
{
 private:
  struct triangle tabtri[2*(N-1)*(N-1)];
  float surface_triangle(int);
  float df(float,float,float,float,float,float,float,float,float);

 public:
  float x[N];
  float y[N];
  float z[N][N];
  float gradient[N-2][N-2];

  maillage(float);
  void results();
  void refresh();
  void refresh_curv();
  float surface();
  void calculer_gradient();
};
#endif
```

4.3 mailleur.cpp

```
#include <stdio.h>
#include <iostream.h>
#include <stdlib.h>
#include <fstream.h>
#include "mailleur.h"

typedef struct _vect{
  float xx,yy,zz;
} vect;


//*********************************************************************
//*********************************************************************

float norme(vect u){
  return(sqrt(u.xx*u.xx+u.yy*u.yy+u.zz*u.zz));

}
//*********definition du constructeur***********************************

maillage::maillage(float test){

 //init des vecteurs x et y
   for (int i=0;i<=N-1;i++){
     x[i]=X*i/(N-1);
     y[i]=Y*i/(N-1);
   }
   //init des triangles
   for(int i=0;i<=N-2;i++){
      for(int j=0;j<=N-2;j++){
       // premier triangle
       tabtri[2*i*(N-1)+2*(j+1)-2].a.x=&x[j];
       tabtri[2*i*(N-1)+2*(j+1)-2].a.y=&y[i+1];
       tabtri[2*i*(N-1)+2*(j+1)-2].a.z=&z[j][i+1];

       tabtri[2*i*(N-1)+2*(j+1)-2].b.x=&x[j];
       tabtri[2*i*(N-1)+2*(j+1)-2].b.y=&y[i];
       tabtri[2*i*(N-1)+2*(j+1)-2].b.z=&z[j][i];

       tabtri[2*i*(N-1)+2*(j+1)-2].c.x=&x[j+1];
       tabtri[2*i*(N-1)+2*(j+1)-2].c.y=&y[i+1];
       tabtri[2*i*(N-1)+2*(j+1)-2].c.z=&z[j+1][i+1];

       // deuxieme triangle
       tabtri[2*i*(N-1)+2*(j+1)-1].a.x=&x[j+1];
       tabtri[2*i*(N-1)+2*(j+1)-1].a.y=&y[i];
```

```
        tabtri[2*i*(N-1)+2*(j+1)-1].a.z=&z[j+1][i];

        tabtri[2*i*(N-1)+2*(j+1)-1].b.x=&x[j+1];
        tabtri[2*i*(N-1)+2*(j+1)-1].b.y=&y[i+1];
        tabtri[2*i*(N-1)+2*(j+1)-1].b.z=&z[j+1][i+1];

        tabtri[2*i*(N-1)+2*(j+1)-1].c.x=&x[j];
        tabtri[2*i*(N-1)+2*(j+1)-1].c.y=&y[i];
        tabtri[2*i*(N-1)+2*(j+1)-1].c.z=&z[j][i];
       }
   }
   //  cout << "nono\n";
}

//******Definition des fonctions menmbres de la classe point************

//*************************************************************************
float maillage::df(float x1,float y1,float z1,float x2,float y2,float z2
,float x3,float y3,float z3){
  float u,u1,u2,u3,u1prim,u2prim;
  u1=(y2-y1)*(z3-z1)-(z2-z1)*(y3-y1);
  u2=(z2-z1)*(x3-x1)-(x2-x1)*(z3-z1);
  u3=(x2-x1)*(y3-y1)-(y2-y1)*(x3-x1);
  u=u1*u1+u2*u2+u3*u3;
  u1prim=y3-y2;
  u2prim=x2-x3;
  return((u1prim*u1+u2prim*u2)/(2*sqrt(u)));
}
//*************************************************************************


float maillage::surface_triangle(int i){
vect ab,ac,u;
  ab.xx=*tabtri[i].b.x-*tabtri[i].a.x;
  ab.yy=*tabtri[i].b.y-*tabtri[i].a.y;
  ab.zz=*tabtri[i].b.z-*tabtri[i].a.z;
  ac.xx=*tabtri[i].c.x-*tabtri[i].a.x;
  ac.yy=*tabtri[i].c.y-*tabtri[i].a.y;
  ac.zz=*tabtri[i].c.z-*tabtri[i].a.z;
  u.xx=ab.yy*ac.zz-ab.zz*ac.yy;
  u.yy=ab.zz*ac.xx-ab.xx*ac.zz;
  u.zz=ab.xx*ac.yy-ab.yy*ac.xx;
  return(norme(u)/2.);
}


//*************************************************************************
```

```
//genere un fichier executable par scilab contenant
//x mat (1xN)      y mat (1xN)      z mat (NxN,XxY )
//*************************************************************************



void maillage::results()
{
  ofstream sortie("resultats",ios::out);

  sortie << "z1=[";
  for(int j=0;j<N-1;j++){
    sortie << z[0][j] << ",";
  }
  sortie << z[0][N-1] << "];" << "\n";
  sortie <<"z=z1;" << "\n";
  for(int i=1;i<N;i++){
    sortie << "z1=[";
    for(int j=0;j<N;j++){
      sortie << z[i][j];
      if(j<N-1) sortie<<",";
    }
  sortie << "];" << "\n";
  sortie << "z=[z;z1];" << "\n";
  }

  sortie << "x=[";
  for(int i=0;i<N;i++){
      sortie << X*i/(N-1);
      if(i<N-1) sortie<<",";
  }
  sortie << "];" << "\n";

  sortie << "y=[";
  for(int i=0;i<N;i++){
      sortie << Y*i/(N-1);
      if(i<N-1) sortie << ",";
  }
  sortie << "];" << "\n";
  sortie << "xbasc(0);" << "\n";
  sortie << "plot3d(x,y,z);" << "\n";

  sortie.close();
}
//*************************************************************************
void maillage::refresh_curv(){
  float delta=X/(N-1);
```

```
  float a=-4./(X*X*X);
  float b=-3.*a*X/2.;
  float temp;
  for(int i=0;i<N;i++) for(int j=0;j<N;j++) z[i][j]=0.0; //tout le mode a zero
  for(int i=0;i<N;i++){
    temp=a*(i*delta)*(i*delta)*(i*delta)+b*(i*delta)*(i*delta);
    z[0][i]=temp;
    z[i][0]=temp;
    z[N-1][N-i-1]=temp;
    z[N-1-i][N-1]=temp;
  }
}
//*************************************************************************
void maillage::refresh()
{
  for(int i=0;i<N;i++)
    for(int j=0;j<N;j++)
      z[i][j]=0.0;
  ifstream cond_init("cond_init",ios::in);
  cond_init >> z[0][0] >> z[(N-1)/2][0] >> z[N-1][0]
    >> z[0][(N-1)/2] >> z[N-1][(N-1)/2]
    >> z[0][N-1] >> z[(N-1)/2][N-1] >> z[N-1][N-1];
  cond_init.close();


  float delta;

  delta=(-z[0][0]+z[(N-1)/2][0])/(N-1)*2;
  for(int i=1;i<N-1;i++)
    if(i==(N-1)/2) delta=(-z[(N-1)/2][0]+z[N-1][0])/(N-1)*2;
    else z[i][0]=z[i-1][0]+delta;

  delta=(-z[0][N-1]+z[(N-1)/2][N-1])/(N-1)*2;
  for(int i=1;i<N-1;i++)
    if(i==(N-1)/2) delta=(-z[(N-1)/2][N-1]+z[N-1][N-1])/(N-1)*2;
    else z[i][N-1]=z[i-1][N-1]+delta;

  delta=(-z[0][0]+z[0][(N-1)/2])/(N-1)*2;
  for(int i=1;i<N-1;i++)
    if(i==(N-1)/2) delta=(-z[0][(N-1)/2]+z[0][N-1])/(N-1)*2;
    else z[0][i]=z[0][i-1]+delta;

   delta=(-z[N-1][0]+z[N-1][(N-1)/2])/(N-1)*2;
   for(int i=1;i<N-1;i++)
     if(i==(N-1)/2) delta=(-z[N-1][(N-1)/2]+z[N-1][N-1])/(N-1)*2;
     else z[N-1][i]=z[N-1][i-1]+delta;
```

```
}
//*************************************************************************
float maillage::surface(){
int i;
float s;
  s=0.;
  for(i=0;i<2*(N-1)*(N-1);i++) s=s+surface_triangle(i);
  return(s);
}
//*************************************************************************
void maillage::calculer_gradient(){
int i,j;
  for(i=0;i<N-2;i++){
   for(j=0;j<N-2;j++){
     gradient[i][j]=df(x[i+1],y[j+1],z[i+1][j+1],x[i],y[j+1]
,z[i][j+1],x[i],y[j],z[i][j]);
     gradient[i][j]+=df(x[i+1],y[j+1],z[i+1][j+1],x[i+1],y[j]
,z[i+1][j],x[i],y[j],z[i][j]);
     gradient[i][j]+=df(x[i+1],y[j+1],z[i+1][j+1],x[i+1],y[j]
,z[i+1][j],x[i+2],y[j+1],z[i+2][j+1]);
     gradient[i][j]+=df(x[i+1],y[j+1],z[i+1][j+1],x[i+2],y[j+2],
z[i+2][j+2],x[i+2],y[j+1],z[i+2][j+1]);
     gradient[i][j]+=df(x[i+1],y[j+1],z[i+1][j+1],x[i+2],y[j+2]
,z[i+2][j+2],x[i+1],y[j+2],z[i+1][j+2]);
     gradient[i][j]+=df(x[i+1],y[j+1],z[i+1][j+1],x[i],y[j+1],
z[i][j+1],x[i+1],y[j+2],z[i+1][j+2]);
   }
 }
}
```

Chapitre 5

Essais numérique

Nous avons effectué trois tests sur un pentium 450Mhz 64Mo RAM. Nous en présentons trois avec la représentation des surfaces en annexe. Le format du fichier Cond-Init est le suivant :

- coin inférieur gauche
- milieu arrête inférieur
- coin inférieur droit
- milieu arrête gauche
- milieu arrête droite
- coin supérieur gauche
- milieu arrête supérieure
- coin supérieur droit

Les surfaces sont présentées en annexe à la fin du rapport.

5.1 test 1

Données : le fichier Cond-Init contient :
0.0
2.0
0.0
0.0
0.0
0.0
2.0
0.0

5.2 test 2

Données : Le fichier Cond-Init contient :
1.0
2.0
3.0
0.0

2.0
0.5
5.0
4.0

5.3 test 3

Un dernier test est effectué à l'aide d'une procédure (methode refresh-curv() de la classe maillage) qui permet de calculer un bord curviligne.