Projet Informatique

Simulation d'une avalanche à Barèges

Romain BOULAUD - Freddy NUGIER - Noémie CLOU - Axel ROBIN

Groupe 1

Table des matières

Chapitre 1 : Analyse du problème

L'objectif de ce projet est de simuler une avalanche sur une montagne de Barèges, à l'aide du logiciel informatique Matlab.

En nous basant sur le travail que nous avons effectué en TD à propos de la chute d'une météorite dans l'atmosphère, nous avons divisé notre travail en trois grandes étapes. Tout d'abord, la première étape consiste à représenter la montagne de Barèges en 3D. Ensuite, la seconde étape a pour objectif de dessiner le profil 2D de cette montagne. Enfin, la troisième étape permet de transformer ce profil en matrice, afin de l'intégrer dans la fonction LBM.

Par ailleurs, nous allons tenter de remplir certains objectifs secondaires, d'une part en réalisant une avalanche qualitativement crédible, et d'autre part en programmant un critère pour stopper l'avalanche.

Chapitre 2 : Présentation des solutions retenues

A. Première étape : représentation 3D de la montagne

Afin de représenter la montagne de Barèges en 3D, nous avons d'abord relevé un certain nombre de points de cette montagne sur la carte, puis nous avons rentrés leurs coordonnées dans une matrice, appelée *Sommets*.

À l'aide de ces points, nous avons ensuite défini des triangles. La matrice contenant les coordonnées de chacun des sommets de ces triangles se nomme *Triangles*.

Pour terminer cette première étape, la fonction *DessineGeometrie_3D*, constituée d'un *plot3* intégré dans une boucle *for*, permet de représenter une vue 3D de la montagne de Barège.

Par ailleurs, la fonction *extrait_coordonnees_sommets* nous permet d'extraire les coordonnées des sommets des triangles. Cette fonction n'est pas nécessaire pour dessiner la montagne, mais elle pourrait nous être utile dans la suite de notre travail.

B. Seconde étape : dessin du profil de la montagne

Dans cette seconde étape, nous devons d'abord définir une droite, grâce aux coordonnées des deux points extrémités D et E, puis la fonction *segmentation* nous permet de diviser cette droite en cent sections de même longueur. On obtient alors une matrice U, contenant les coordonnées des points de segmentation de la droite, séparés par un pas donné.

Pour chacun de ces points, on cherche les coordonnées de l'intersection de la demi-droite verticale passant par ce point avec un éventuel triangle, à l'aide de la fonction *Calcul_Intersection_Triangle_Droite*. Cette fonction renvoi alors les coordonnées (x, y, z) de tous les points d'intersection, dans une matrice V.

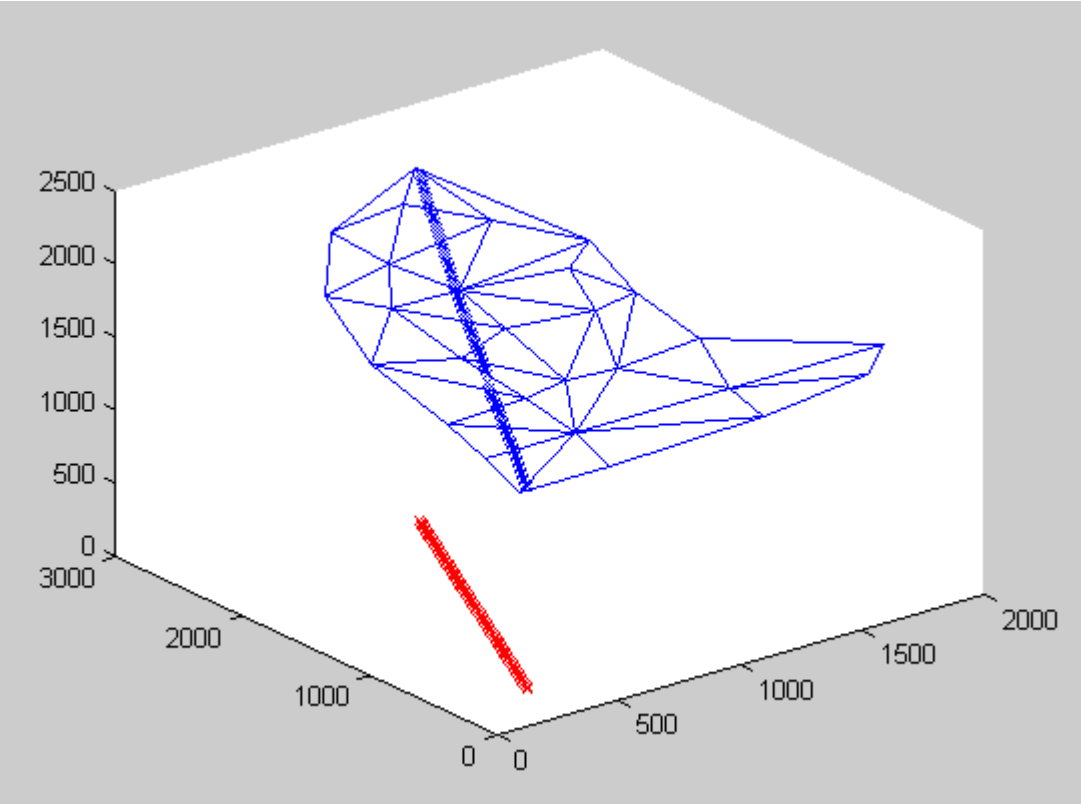


Schéma représentant la montagne en 3D (bleu), avec en rouge la droite segmentée et en bleu les points d'intersection des demi-droites verticales passant par ces points avec un éventuel triangle.

Enfin, la fonction *coordonnees_altitudes_profil_2D* permet de stocker toutes les altitudes des points d'intersection entre la surface et les demi-droites issues de la segmentation. Ainsi, nous pouvons dessiner le profil 2D de notre montagne, à partir de la matrice V. Ce profil comporte en abscisse un axe gradué de 0 à 100 correspondant aux points de la droite (DE), et en ordonnée l'altitude de chaque point d'intersection (coord. z).

C. Troisième étape : transformation du profil en matrice

Pour débuter cette troisième étape, la fonction *changement_repere* permet d'effectuer un changement de repère du profil, afin d'obtenir une échelle d'abscisse (respectivement, d'ordonnée) homogène à Lx (respectivement, à Ly) de LBM.

Nous définissons ensuite la fonction *matrice_obstacle* qui nous permet d'obtenir une matrice *obstacle*, de taille Lx - Ly, dont les termes sont égaux à 0 (si c'est de l'air) ou 1 (si c'est la montagne). Une fois définie, cette matrice sera un paramètre d'entrée de LBM.

De plus, nous avons créés une fonction *generation_newpart* qui ajoute des particules de neige dans l'avalanche à chaque instant, lorsque la vitesse du fluide au-dessus de la pente est supérieure à une vitesse seuil, fixée par l'opérateur.

Pour mettre en place cette fonction, il est nécessaire de savoir si une particule de la montagne, et qui pourrait éventuellement se détacher, est en contact avec une particule de neige de l'avalanche. Ainsi, la fonction *interface_montagne_atmosphere* nous renvoi la valeur 1 si la montagne est en contact avec une particule de l'avalanche, 0 sinon.

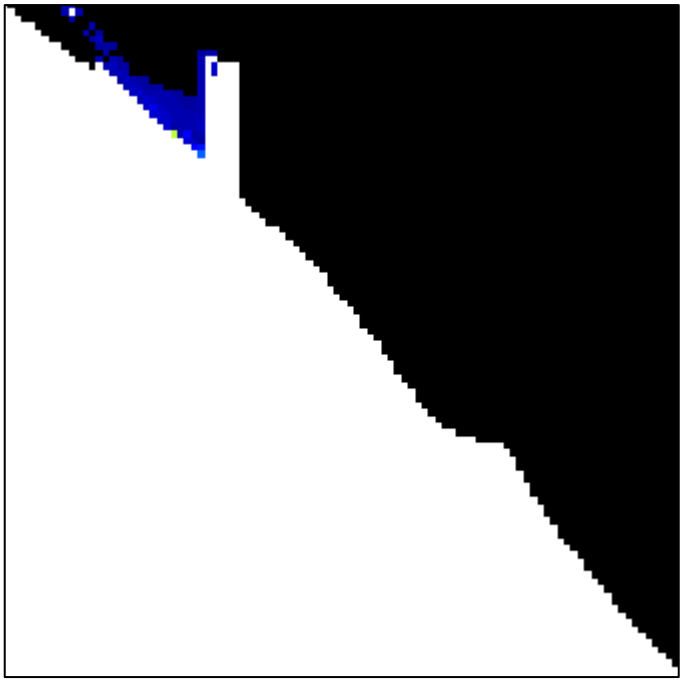
Pour finir, nous appliquons LBM à chaque instant, pour chacun de ces particules.

D. Quatrième étape : réalisation d'objectifs secondaires

D'une part, pour réaliser une avalanche qualitativement crédible, nous avons joué sur différents paramètres : la proportionnalité de la vitesse des nouvelles particules et de la vitesse du fluide, le pas d'Euler, ainsi que la vitesse seuil.

D'autre part, nous avons programmé un indice de risque permettant d'indiquer le caractère dangereux ou non de l'avalanche. Ainsi, si elle devient trop dangereuse, le message « Trop dangereux : évacuation des pistes » apparait.

Par ailleurs, nous avons aussi essayé de créer un mur anti-avalanche proche de la zone de déclanchement, afin de la stopper, mais celui-ci ne fonctionne pas car l'avalanche le franchit :



E. Problèmes rencontrés

Nous avons rencontré un problème dans la transformation du profil 2D en matrice obstacle. En effet, il a fallu trouver une solution pour ramener les altitudes à des valeurs admissibles dans LBM. Nous avons donc arrangé la fonction *changement_repere* afin de solutionner ce problème technique.

Puis, lors de la création de la matrice obstacles, nous avons passé beaucoup de temps à trouver comment retranscrire exactement le profil 2D dans LBM.

Le problème le plus important que nous avons rencontré est celui de la génération de nouvelles particules. Pour cela, il a fallu créer des boucles incluses dans d'autres boucles, afin d'analyser chacune des particules lors de chacune des nouvelles itérations dans le programme *LBM*. Nous avons dû créer deux nouvelles fonctions, l'une analysant la capacité d'une particule à se décoller à la surface de la montagne, l'autre générant les vecteurs contenant toutes les particules (coordonnées et vitesses).

Chapitre 3 : Réalisation du programme

Base_geometrie

```
%% Définition du profil 3D de la montagne
de BAREGES

% Sommets est la matrice des coordonnées
(x,y,z) dse points définissants le
% profil
Sommets = [0.9*333 1.2*333 1360 ;
    0.9*333 2.0*333 1480 ;
    0.9*333 2.9*333 1600 ;
    0.9*333 4.7*333 1760 ;
    0.9*333 5.8*333 2080 ;
    2.0*333 1.2*333 1360 ;
    3.9*333 1.2*333 1400 ;
    5.2*333 1.2*333 1480 ;
    2.0*333 2.0*333 1480 ;
    3.9*333 2.0*333 1480 ;
    5.8*333 2.0*333 1480 ;
    1.9*333 3.0*333 1600 ;
    2.5*333 3.2*333 1600 ;
    3.1*333 3.1*333 1600 ;
    4.2*333 3.3*333 1600 ;
    1.6*333 3.9*333 1800 ;
    2.4*333 4.4*333 1800 ;
    3.4*333 4.2*333 1800 ;
    4.0*333 4.4*333 1800 ;
    1.4*333 5.2*333 2000 ;
    2.3*333 5.3*333 1960 ;
    3.5*333 5.0*333 1960 ;
    1.8*333 6.0*333 2120 ;
    3.2*333 6.3*333 2160 ;
    1.5*333 6.8*333 2280 ;
    2.5*333 7.0*333 2280 ;
    2.9*333 7.5*333 2400 ;
    4.1*333 5.7*333 1960] ;

% Triangles est la matrice formant des
triangles à partir de trois points
% de la matrice Sommets
Triangles = [ 1 2 9 ;
    1 6 9 ;
    6 7 9 ;
    7 9 10 ;
    7 8 10 ;
    8 10 11 ;
    2 3 9 ;
    3 9 12 ;
    9 12 13 ;
    9 13 14 ;
    9 10 14 ;
    14 15 10 ;
    10 11 15 ;
    3 4 12 ;
    4 16 12 ;
    12 16 13 ;
    16 17 13 ;
    13 17 18 ;
    13 14 18 ;
    14 18 19 ;
    14 15 19 ;
    4 20 16 ;
    16 17 20 ;
    17 21 20 ;
    17 18 21 ;
    18 19 22 ;
    18 21 22 ;
    19 22 28 ;
    21 22 28 ;
    5 4 20 ;
    5 20 23 ;
    20 21 23 ;
    23 24 21 ;
    21 24 28 ;
    5 25 23 ;
    23 25 26 ;
    23 24 26 ;
    25 26 27 ;
    24 26 27 ;
    28 24 27];

%% Définition du segment servant de base
au plan de coupe

% [ED] segment servant de base à la
définition du plan de coupe de la
% montagne servant de base à la
définition du profil 2D
xE=1*333;
yE=1.2*333;
zE=0;
E=[xE,yE,zE];

xD=2.9*333;
yD=7.4*333;
zD=0;
D=[xD,yD,zD];

%% Définition du pas de segmentation de
la droite [ED]
n=101;
```

```
function DessineGeometrie_3D(Sommets,Triangles)
% function DessineGeometrie_3D(Sommets,Triangles)
% Représente en 3D le profil de la montagne défini à partir de la réunion
% des triangles (matrice Triangles) obtenu par réunion de trois sommets
% (matrice Sommets)

hold on ;

% Dessin de chacun des triangles
for j=1:size(Triangles,1)
    plot3([Sommets(Triangles(j,1:3),1)' Sommets(Triangles(j,1),1)], ...
        [Sommets(Triangles(j,1:3),2)' Sommets(Triangles(j,1),2)], ...
        [Sommets(Triangles(j,1:3),3)' Sommets(Triangles(j,1),3)]) ;
end
end
```

```
function
[Intersection,Z]=coordonnees_altitudes_profil_2D(Sommets,Triangles,D,E,n)
% [Intersection,Z]=coordonnees_altitudes_profil_2D(Sommets,Triangles,D,E,n)
% Récupère ls coordonnées des points définissant le profil 2D de la
% montagne selon le plan de coupe (ED,z)

%% Définition et segmentation de la droite ED
[S]=segmentation(D,E,n);

% Initialisation du vecteur des altitudes
Z=[];

% Dessin du profil 3D de la montagne
DessineGeometrie_3D(Sommets,Triangles);

%% Obtention des points d'intersection
% Parcours de points de ED
for i=1:n+1
    % Choix du numéro du triangle
    for j=1:size(Triangles,1)

        % Définition de la demi-droite
        O = [S(i,1) S(i,2) 0] ;
        R = [0 0 1] ;

        % Extraction des coordonnées des sommets du triangle
        [xA,yA,zA] = extrait_coordonnees_sommets(Sommets, Triangles, j, 1) ;
        A = [xA,yA,zA] ;
        [xB,yB,zB] = extrait_coordonnees_sommets(Sommets, Triangles, j, 2) ;
        B = [xB,yB,zB] ;
        [xC,yC,zC] = extrait_coordonnees_sommets(Sommets, Triangles, j, 3) ;
        C = [xC,yC,zC] ;

        % Calcul de l'intersection éventuelle
        Intersection = Calcul_Intersection_Triangle_Droite(A, B, C, O, R) ;

        % Dessin de la demi-droite
        hold on
        plot3(O(1), O(2), O(3), 'xr') ;
        plot3([O(1) O(1)+R(1)], [O(2) O(2)+R(2)], [O(3) O(3)+R(3)], '-r') ;
```

```
        % Si l'intersection existe
        if ~isempty(Intersection)
            % Dessin du point d'intersection
            plot3(Intersection(1), Intersection(2), Intersection(3), 'xb') ;
            Z=[Z Intersection(3)];
        end
    end
end
```

trace_profil_2D

```
% Script permettant le tracé du profil 2D de la montagne
% Les abscisses sont les points de segmentation de ED donc la demi-droite a
% une intersection avec la montagne.
% Les ordonnées sont les altitudes stockées dans Z

X=1:length(Z);
plot(X,Z);
```

```
function [P]=changement_repere(Z)
% function [P]=changement_repere(Z)
% permet de se placer dans le plan (ED,Z) utilisé pour la segmentation

% Initialisation de la matrice des points du plan
P=[];

% Parcours des altitudes stockées dans le vecteur Z
for i=1:length(Z);
    % Création des abscisses
    P(i,1)=i;
    % création des ordonnées telles que l'on puisse établir la matrice
    % obtacle correcte
    P(i,2)=floor((Z(i)-min(Z))/10);
end

end
```

```
function [Obstacles] = matrice_obstacle(P,Z)
% function [Obstacles] = matrice_obstacle(P,Z)
% P est la matrice contenant toutes les coordonnées des points du profil
% O est la matrice obstacle composée de 0 (air) et de 1 (montagne)

% Initialisation de la matrice Obstacles
Obstacles=zeros(length(Z),length(Z));

% Parcours lignes de Obstacles
for i=1:length(Z);
    % Parcours colonnes de Obstacles
    for j=1:length(Z);
```

```
        if length(Z)+1-j > P(i,2);
            Obstacles(i,j)=1;
        end
    end
end

end



%% Initialisation de la simulation
% Initialisation des valeurs

h=0.06; % Pas utilisé pour la méthode d'Euler
g=[9.81 0]; % Accélération de la pesanteur
m=0.1; % masse de la particule
f=0.02; % coefficient de frottement
p=[10 30]; % position initiale de la première particule
v=[200 0]; % vitese initiale de la première particule
vx=v(1); % vitesse horizontale initiale de la première particule
vy=v(2); % vitesse verticale initiale de la première particule
x=p(1); % abscisse initiale de la première particule
y=p(2); % ordonnée  initiale de la première particule
lx=length(Z); % longueur de la matrice LBM
ly=length(Z); % largeur de la matrice
seuil=0.01; % vitesse minimale pour le décollement d'une particule
v_arret_seuil=0.2; % vitesse au delà de laqulle l'avalanche devient dangereuse

% Nombre d'itérations de la simulation
nombre_iterations = 1000;

% Initialisation du champ de vitesse
% Ici, on donne au temps t=0 une vitesse de 1, horizontalement et verticalement,
% au fluide de la cellule située au milieu du domaine d'étude
[ux_init, uy_init] = Init_speed(lx, ly);

% Initialisation de la distribution LBM
fIn = LBM_Init_Distribution(ux_init, uy_init);

%% Boucle principale de simulation

%Ouverture d'une figure pour le dessin
figure(1);

% Boucle itérative (qui met à jour l'état de la simulation)
% Chaque itération correspond à un pas de temps
for n=1:nombre_iterations
    for i=1:length(x)
        v(1)=vx(i);
        v(2)=vy(i);
        p(1)=x(i);
        p(2)=y(i);

        [Sf]=somme_force(g,v,f);
        [a]=acceleration(Sf,m);
        [v]=Euler_explicite(v,a,h);
        [p]=Euler_explicite(p,v,h);

        vx(i)=v(1);
        vy(i)=v(2);
        x(i)=p(1);
```

```
        y(i)=p(2);

        if ~InDomain(lx, ly,  x(i),  y(i))
            x(i)=2;
            y(i)=2;
            v(1)=0;
            v(2)=0;
        end

    end

    % Etape de simulation du fluide : Calcul de l'état courant de la
    % distribution fIn à partir de son état précédent fIn_previous

    fIn = LBM_Step_Distribution_Obstacles(lx, ly, fIn, floor(x), floor(y), vx,
vy, Obstacles);

    % Calcul du champ de vitesses
    [ux, uy] = LBM_Compute_Macroscopic_Speed(lx, ly, fIn);

    % Calcul de la densité
    [rho] = LBM_Compute_Macroscopic_Density(lx, ly, fIn);

    % Affichage du module de la vitesse macroscopique
    Display_Speed_Obstacles(ux, uy, x, y, Obstacles);


    % Création de nouvelle particule
    cx = [ -1,   0,  1, 1,  1, -1,  0,  1];
    cy = [ 1,   1,  1,  0, 0, -1,  -1,  -1];

    %% Génération d'une éventuelle nouvlle particule avec condition sur la
vitesse du fluide
    % parcour des lignes de la matrice obstale
    for i=2:lx-1
        % parcour des colonnes de la matrice obstacle
        for j=2:ly-1
            [voisinage] = interface_montagne_atmosphere(i,j,Obstacles,cx,cy);
            if voisinage==1
                % Calcul de la norme de la vitesse du fluide
                u=sqrt(ux.*ux + uy.*uy);
                % décollment d'une partiul si la vitesse du fluide et supérieure
u seuil                 fixé
                if u(i,j) > seuil

[x,y,vx,vy]=generation_newpart(x,y,vx,vy,i,j,0.5*ux(i,j),0.5*uy(i,j));
                end
            end
        end
    end

    %% Détermination si l'avalanche peut etre considérée comme arrêtée
    for i=1:length(Z)
        % détermination de la condition sur la vitesse du fluide à la
        % sortie du domaine de simulation
        if norm(u(i,length(Z)-3))<v_arret_seuil
            disp( 'Avalanche est sans risques');
        else error('Trop dangereux : Evacuation des pistes');
        end
    end
end
```

Voici le script final que nous avons mis en place :

```
% Script permettant de simuler une avalanche à partir d'un profil de
% montagne

%% Effacement de la mémoire
clear all
close all

%% Création et dessin 3D de la montagne
base_geometrie;
DessineGeometrie_3D(Sommets,Triangles);

%% Récupération d'un profil 2D comme intersection du profil 3D et du plan de
coupe (ED,z)
[Intersection,Z]=coordonnees_altitudes_profil_2D(Sommets,Triangles,D,E,n);
trace_profil_2D;

%% Changement de repère et obtention de la matrice Obstacles
clf
[P]=changement_repere(Z);
[Obstacles] = matrice_obstacle(P,Z);

for i=1:length(Z)
    for j=70:75
      Obstacles(i,j)=1;
    end
end

%% Lancement de la simulation
LBM_getting_started
```

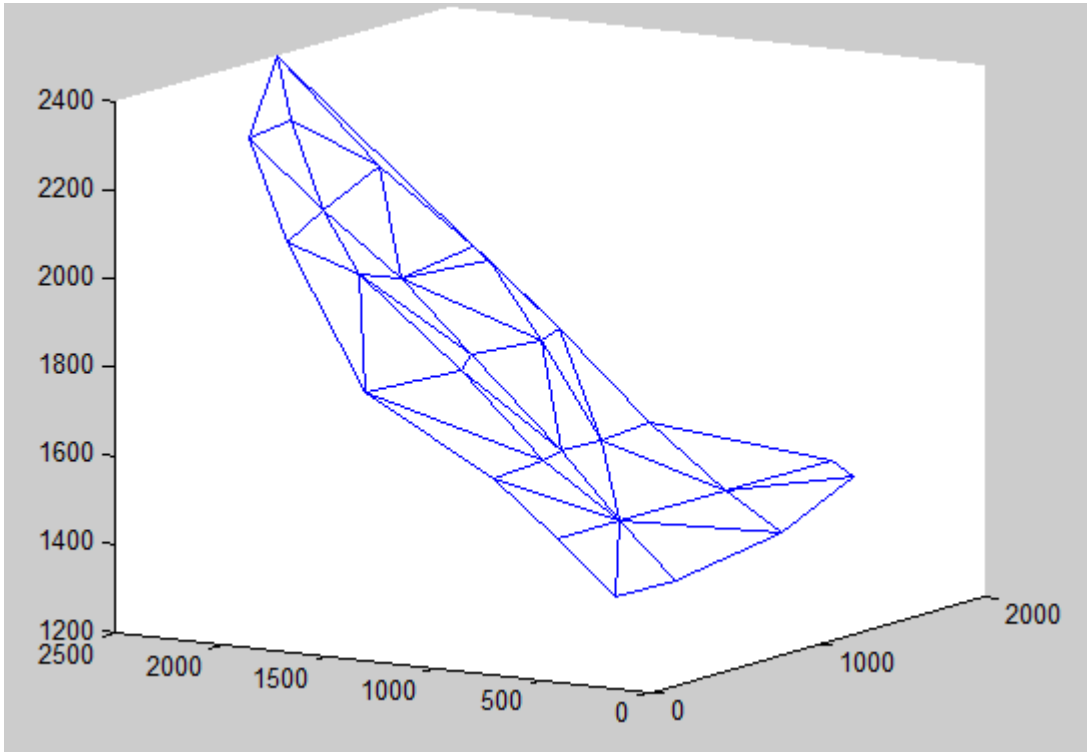
Remarque :

En **Annexe** (page 19), nous avons testé toutes les fonctions nécessaires à notre projet et que nous avons créées, à l'aide d'arguments d'entrée simples. Connaissant le résultat attendu, relativement simple, cela nous a permis de vérifier la validité de chacune de nos fonctions.

Chapitre 4 : Résultats

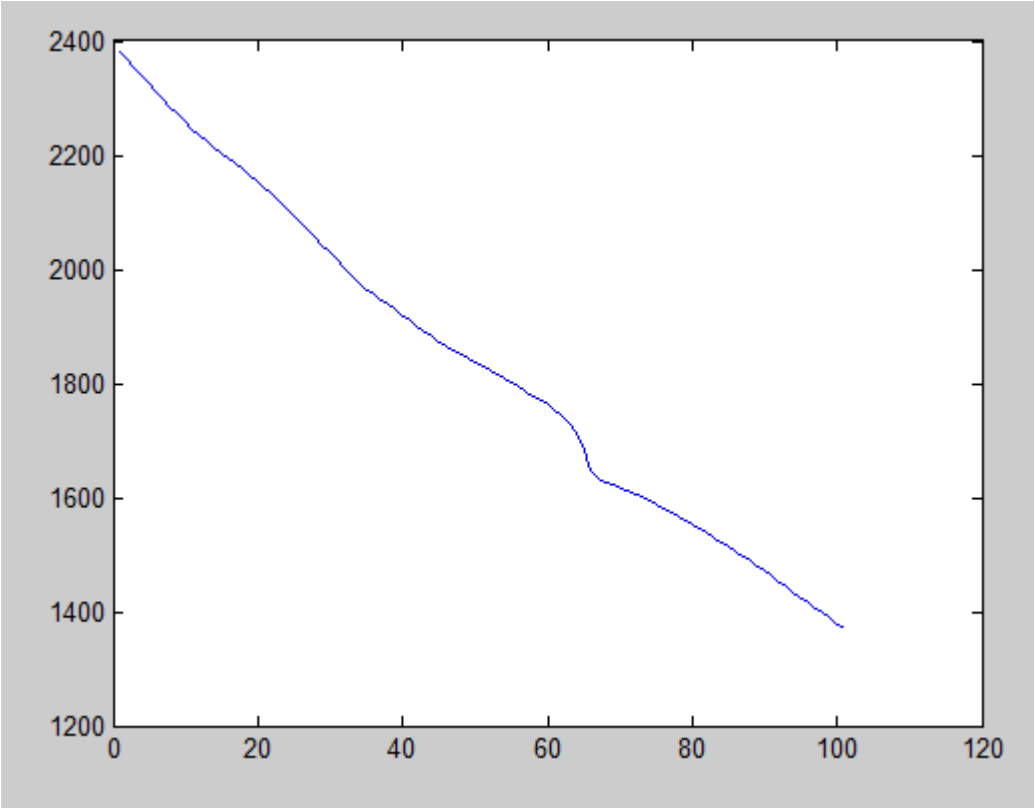
A. Représentation 3D de la montagne

La fonction *DessineGeometrie_3D* créée dans la première partie nous ont permis de dessiner la montagne de Barèges en 3D :



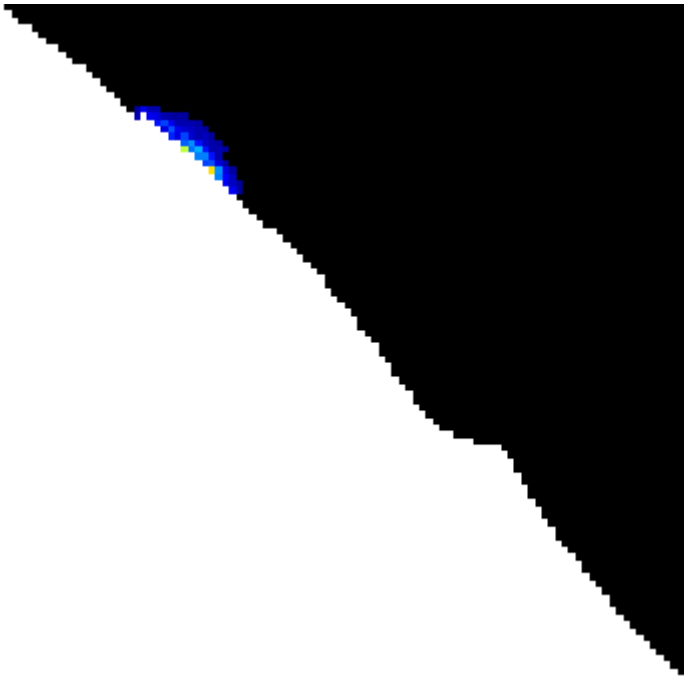
B. Profil de la montagne

Les fonctions mises en place dans la seconde partie nous permettent d'obtenir le profil 2D de notre montagne :

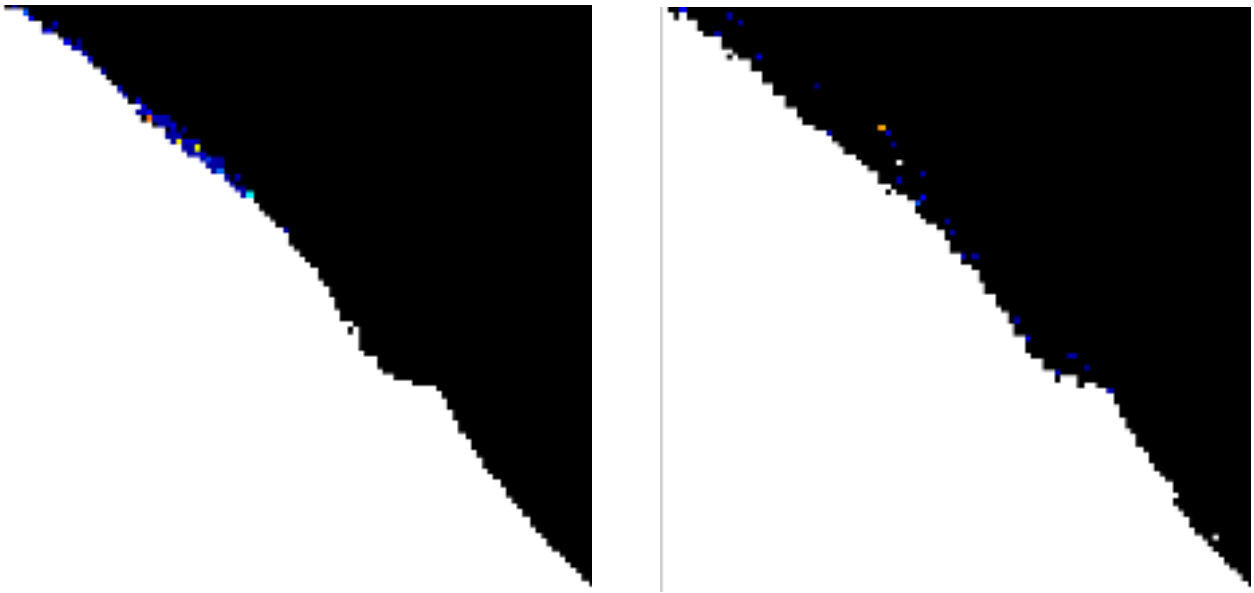


C. Avalanche

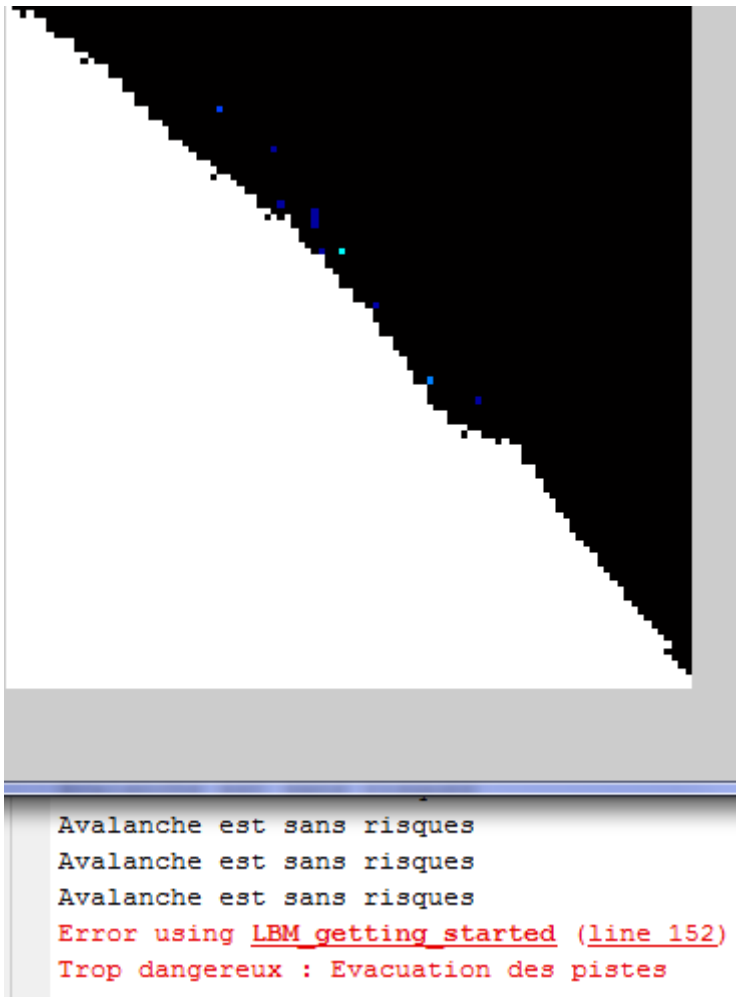
Nous avons finalement obtenus une avalanche qualitativement crédible, déclenchée par la chute d'une « météorite » en un point de la montagne et entretenue par le détachement de particules de neige de la montagne à chaque instant.



Déclanchement de l'avalanche



Avalanche en cours



Indice de risque

Annexes

Tests unitaires des fonctions utilisées

Sommaire :

Annexe 1 : *DessineGeometrie_3D*

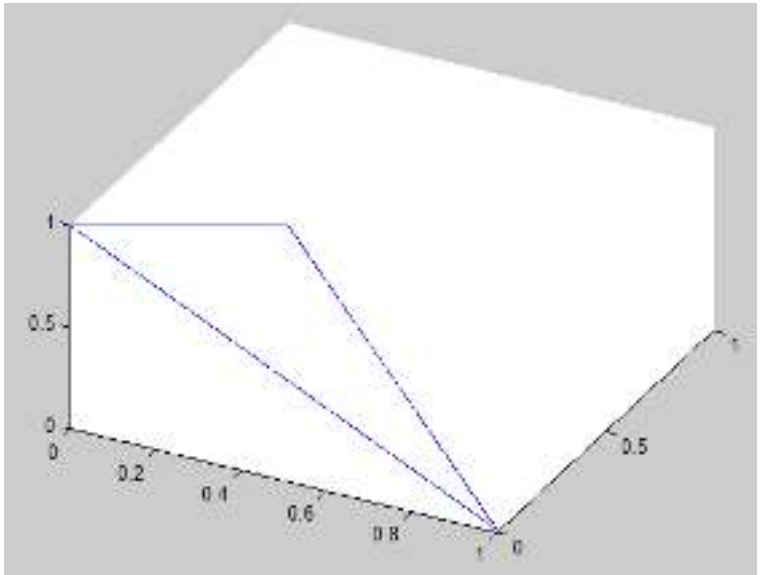
Cette fonction permet, à partir de points relevés sur une carte et stockés dans une matrice où chaque ligne contient les coordonnées (x, y, z), de représenter le triangle dont les sommets précédents sont reliés et stockés dans un nouvelle matrice.

En exécutant le script suivant :

```
% Définition de la matrice des sommets et de la matrice du triangle
Sommets = [0 0 1 ; 1 0 0 ; 0 1 0];
Triangles = [1 2 3];

% Dessin du triangle
DessineGeometrie_3D(Sommets,Triangles);
```

On obtient alors :



Ce graphique est bien conforme à la géométrie attendue en considérant ce cas trivial.

Annexe 2 : *segmentation*

Ce programme permet, à partir d'une droite donnée, de la subdiviser en intervalles de tailles égales.

Le script suivant est exécuté :

```
% Définition des points E et D
xD=0;
yD=0;
zD=0;
D=[xD yD zD];

xE=1;
yE=1;
zE=0;
E=[xE yE zE];

% Définition du pas de segmentation
n=10;

% Segmentation de la droite ED
S=segmentation(D,E,n)
```

On obtient alors le résultat suivant :

```
S =

0         0
0.1000    0.1000
0.2000    0.2000
0.3000    0.3000
0.4000    0.4000
0.5000    0.5000
0.6000    0.6000
0.7000    0.7000
0.8000    0.8000
0.9000    0.9000
1.0000    1.0000
```

On obtient ici encore un résultat conforme à nos attentes.

Annexe 3 : *extrait_coordonnees_sommets*

Cette fonction permet de récupérer les coordonnées des sommets d'un triangle donné.

On exécute pour cela le script suivant :

```
% Définition de la matrice des sommets et de la matrice du triangle
Sommets = [0 0 1 ; 1 0 0 ; 0 1 0];
Triangles = [1 2 3];

% Récupération des coordonnées des points des sommets du triangle
[xA,yA,zA] = extrait_coordonnees_sommets(Sommets, Triangles, 1, 1) ;
A = [xA,yA,zA]
[xB,yB,zB] = extrait_coordonnees_sommets(Sommets, Triangles, 1, 2) ;
B = [xB,yB,zB]
[xC,yC,zC] = extrait_coordonnees_sommets(Sommets, Triangles, 1, 3) ;
C = [xC,yC,zC]
```

On obtient par la suite ce résultat :

```
A =     0     0     1

B =     1     0     0

C =     0     1     0
```

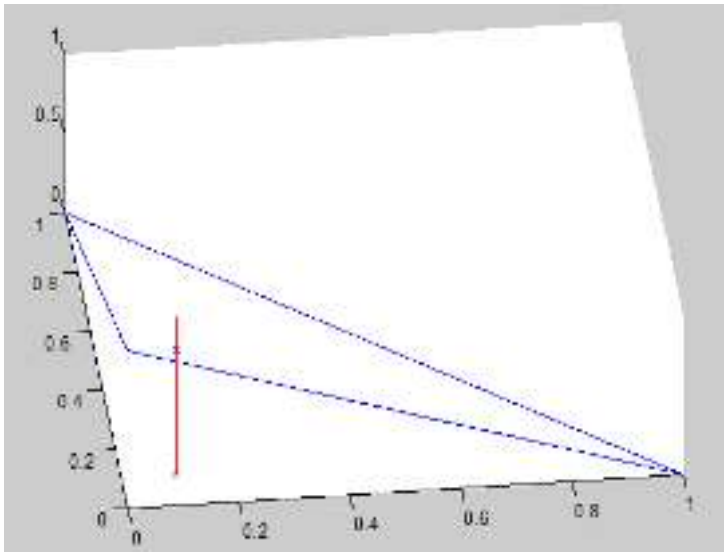
Annexe 4 : *Calcul_Intersection_Triangle_Droite*

Ce programme permet de déterminer le point d'intersection entre un triangle et une droite.

On réalise le test suivant (inspiré du poly) :

```
% Définition de la matrice des sommets et de la matrice du triangle
Sommets = [0 0 1 ; 1 0 0 ; 0 1 0] ;
Triangles = [1 2 3];
% Choix du numéro du triangle
j=1;
% Dessin du triangle
plot3([Sommets(Triangles(j,1:3),1)' Sommets(Triangles(j,1),1)], ...
[Sommets(Triangles(j,1:3),2)' Sommets(Triangles(j,1),2)], ...
[Sommets(Triangles(j,1:3),3)' Sommets(Triangles(j,1),3)])
% Définition de la demi-droite
O = [0.1 0.3 0] ;
R = [0 0 1] ;
% Extraction des coordonnées des sommets du triangle
[xA,yA,zA] = extrait_coordonnees_sommets(Sommets, Triangles, j, 1) ;
A = [xA,yA,zA] ;
[xB,yB,zB] = extrait_coordonnees_sommets(Sommets, Triangles, j, 2) ;
B = [xB,yB,zB] ;
[xC,yC,zC] = extrait_coordonnees_sommets(Sommets, Triangles, j, 3) ;
C = [xC,yC,zC] ;
% Calcul de l'intersection éventuelle
Intersection = Calcul_Intersection_Triangle_Droite(A, B, C, O, R) ;
% Dessin de la demi-droite
hold on
plot3(O(1), O(2), O(3), 'xr') ;
plot3([O(1) O(1)+R(1)], [O(2) O(2)+R(2)], [O(3) O(3)+R(3)], '-r') ;
% Si l'intersection existe
if ~isempty(Intersection)
% Dessin du point d'intersection
plot3(Intersection(1), Intersection(2), Intersection(3), 'xb') ;
end
```

On obtient alors le résultat suivant :




```
Intersection =    0.1000    0.1000    0.8000
```

Annexe 5 : *coordonnees_altitudes_profil_2D*

Ce programme permet de récupérer toutes les altitudes des points d'intersection entre la surface et les demi-droites issues de la segmentation.

On exécute le script suivant :

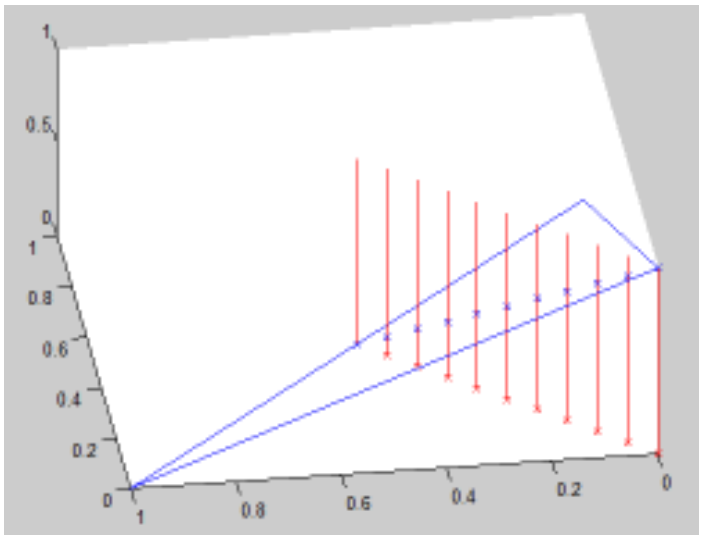
```
% Définition de la matrice des sommets et de la matrice du triangle
Sommets = [0 0 1 ; 1 0 0 ; 0 1 0];
Triangles = [1 2 3];

% Définition des points D et E de la matrice de segmentation
D=[0 0 0];
E=[0.5 0.5 0];

% Définition du pas de segmentation
n=10;

% Calcul des points d'intersection
[Intersection,Z]=coordonnees_altitudes_profil_2D(Sommets,Triangles,D,E,n)
```

On obtient alors le résultat suivant :




```
Z =     1.0000     0.9000     0.8000     0.7000     0.6000     0.5000     0.4000
0.3000    0.2000    0.1000    0.0000
```

Annexe 6 : *changement_repere*

Le script suivant est exécuté :

```
% Définition de la matrice des sommets et de la matrice du triangle
Sommets = [0 0 1 ; 1 0 0 ; 0 1 0];
Triangles = [1 2 3];

% Définition des points D et E de la matrice de segmentation
D=[0 0 0];
E=[0.5 0.5 0];

% Définition du pas de segmentation
n=10;

% Calcul des points d'intersection
[Intersection,Z]=coordonnees_altitudes_profil_2D(Sommets,Triangles,D,E,n);

% Calcul de la matrice permettant de changer de repère
[P]=changement_repere(Z)
```

On obtient alors le résultat suivant :

```
P =

1.0000    1.0000

2.0000    0.9000

3.0000    0.8000

4.0000    0.7000

5.0000    0.6000

6.0000    0.5000

7.0000    0.4000

8.0000    0.3000

9.0000    0.2000

10.0000    0.1000

11.0000         0
```

Annexe 7 : *matrice_obstacle*

Le script suivant est exécuté :

```
% Définition de la matrice des sommets et de la matrice du triangle
Sommets = [0 0 1 ; 1 0 0 ; 0 1 0];
Triangles = [1 2 3];

% Définition des points D et E de la matrice de segmentation
D=[0 0 0];
E=[0.5 0.5 0];

% Définition du pas de segmentation
n=10;

% Calcul des points d'intersection
[Intersection,Z]=coordonnees_altitudes_profil_2D(Sommets,Triangles,D,E,n);

% Calcul de la matrice permettant de changer de repère
[P]=changement_repere(Z) ;

% Détermination de la matrice obstacle
[Obstacles] = matrice_obstacle(P,Z)
```

On obtient alors le résultat suivant :

```
Obstacles =

     1     0     0     0     0     0     0     0     0     0     0

     1     1     0     0     0     0     0     0     0     0     0

     1     1     1     0     0     0     0     0     0     0     0

     1     1     1     1     0     0     0     0     0     0     0

     1     1     1     1     1     0     0     0     0     0     0

     1     1     1     1     1     1     0     0     0     0     0

     1     1     1     1     1     1     1     0     0     0     0

     1     1     1     1     1     1     1     1     0     0     0

     1     1     1     1     1     1     1     1     1     0     0

     1     1     1     1     1     1     1     1     1     1     0

     1     1     1     1     1     1     1     1     1     1     0
```

Annexe 8 : *interface_montagne_atmosphere*

Le script suivant est exécuté :

```
% Définition de la matrice des sommets et de la matrice du triangle
Sommets = [0 0 1 ; 1 0 0 ; 0 1 0];
Triangles = [1 2 3];

% Définition des points D et E de la matrice de segmentation
D=[0 0 0];
E=[0.5 0.5 0];

% Définition du pas de segmentation
n=10;

% Calcul des points d'intersection
[Intersection,Z]=coordonnees_altitudes_profil_2D(Sommets,Triangles,D,E,n);

% Calcul de la matrice permettant de changer de repère
[P]=changement_repere(Z) ;

% Détermination de la matrice obstacle
[Obstacles] = matrice_obstacle(P,Z) ;

% Détermination d'un voisinage (point Obstacles(4,5) ici)
cx = [ -1,   0,  1, 1,  1, -1,  0,  1];
cy = [ 1,   1,  1,  0, 0, -1,  -1,  -1];

[voisinage] = interface_montagne_atmosphere(4,5,Obstacles,cx,cy)
```

On obtient alors le résultat suivant :

```
voisinage =  1
```