Simulation d'un éboulement rocheux à Sechilienne

Rémi Galigné, Jean-Marie Roussel, Blaise Vergneaux

10 mars 2015

Table des matières

Introduction **3**

I. Modélisation du site d'étude **3**
1. Relevé des altitudes sur une carte IGN . . . 3
2. Lissage de la topographie par extrapolation linéaire . . . 4
3. Représentation informatique en 3 dimensions . . . 4

II. Extraction et représentation d'un plan de coupe **5**
1. Extraction des altitudes du plan de coupe . . . 5
2. L'algorithme de Bresenham . . . 6
   a. Principe de l'algorithme de Bresenham . . . 6
   b. Mise en oeuvre dans le code . . . 6
3. Représentation de la coupe . . . 6

III. Simulation de la chute par rebonds **7**
1. Evolution des vecteurs vitesse et position . . . 7
   a. Initialisation des vecteurs position, vitesse et accélération . . . 7
   b. Evolution temporelle - Méthode d'Euler . . . 7
2. Gestion des rebonds . . . 8
   a. Détection d'une intersection avec l'environnement . . . 8
   b. Modification de la trajectoire . . . 8
3. Modification dynamique du profil topographique par le bloc rocheux . . . 8

IV. Tests unitaires de fonctions, résultats de la simulation et limites du code **9**
1. Tests unitaires de fonctions sur un exemple simple . . . 9
   a. Test de coupediago . . . 9
   b. Test de environnement . . . 9
   c. Test de intersection . . . 9
2. Résultats de simulation . . . 10
3. Limites du code . . . 11

Annexes **12**
- Architecture du code . . . 12
- Listing des script et fonctions . . . 12
  - Script principal . . . 12
  - Fonctions annexes . . . 13

Introduction

Les ruines de Sechilienne sont une masse rocheuse fracturée surplombant la vallée de la Romanche à l'extrême sud du massif de Belledonne. Cette zone étant sismiquement active, car elle est à l'intersection de deux failles tectoniques, elle pourrait s'effondrer et s'écrouler vers le fond de la vallée. Selon les estimations, 25 à 100 millions de mètres cubes de roche pourraient être mis en mouvement par un éboulement. L'aléa géologique est couplé à un enjeu économique. Le site est en effet traversé par la route stratégique RD 1091 (anciennement RN 91), seule voie directe praticable en hiver qui relie Grenoble aux stations de l'Oisans, à Briançon et à Turin. Les ruines de séchilienne présentent donc un risque et plusieurs ouvrages de prévention ont déjà été réalisés. Un lit de secours pour la Romanche a été creusé et la population vivant à proximité du site a été priée d'évacuer. La RD 1091 a été déplacée une première fois du côté est (côté Taillefer) de la Romanche mais une récente simulation a montré que cette déviation ne serait pas suffisante. À ce jour, une nouvelle route est en construction sur le flanc du massif du Taillefer, surélevée d'une cinquantaine de mètres par rapport au fond de la vallée.

L'objectif du programme informatique que nous proposons est de simuler la chute d'un bloc rocheux sphérique sur la topographie de Sechilienne. Nous pourrons ensuite essayer de retrouver les résultats des études déjà menées sur le site, à savoir que la première déviation de la RD 1091 n'est pas suffisante, malgré les hypothèses simplificatrices que nous aurons prises. Le code de ce programme se divise en trois grandes étapes. Après avoir modélisé le site d'étude en 3 dimensions il faut en extraire un plan de coupe selon lequel on simulera la chute par rebonds.

I. Modélisation du site d'étude

1. Relevé des altitudes sur une carte IGN

L'institut Géographique National a cartographié la zone des ruines de Sechilienne. Le relevé des altitudes s'effectue en superposant un quadrillage de 209 carrés à cette carte et en notant toutes les altitudes situées au coin Nord-Est de chaque subdivision. Ces données sont ensuite recensées dans une matrice que nous noterons altitudes.

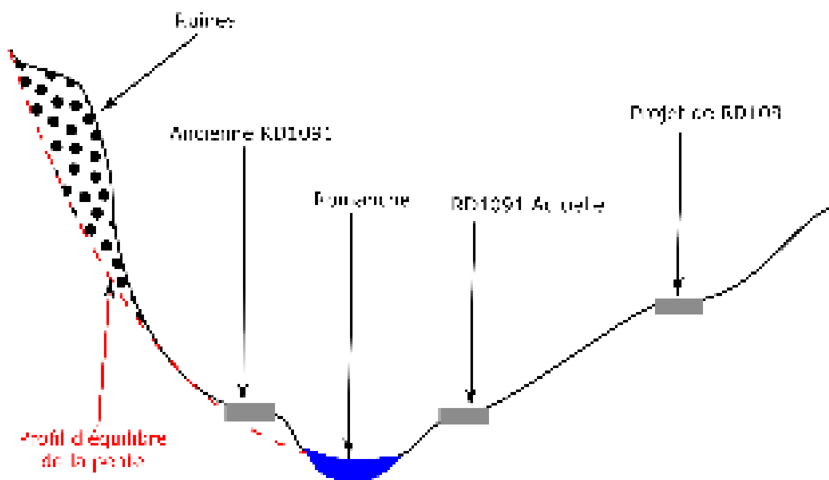


FIGURE 1 : Coupe topographique du site d'étude

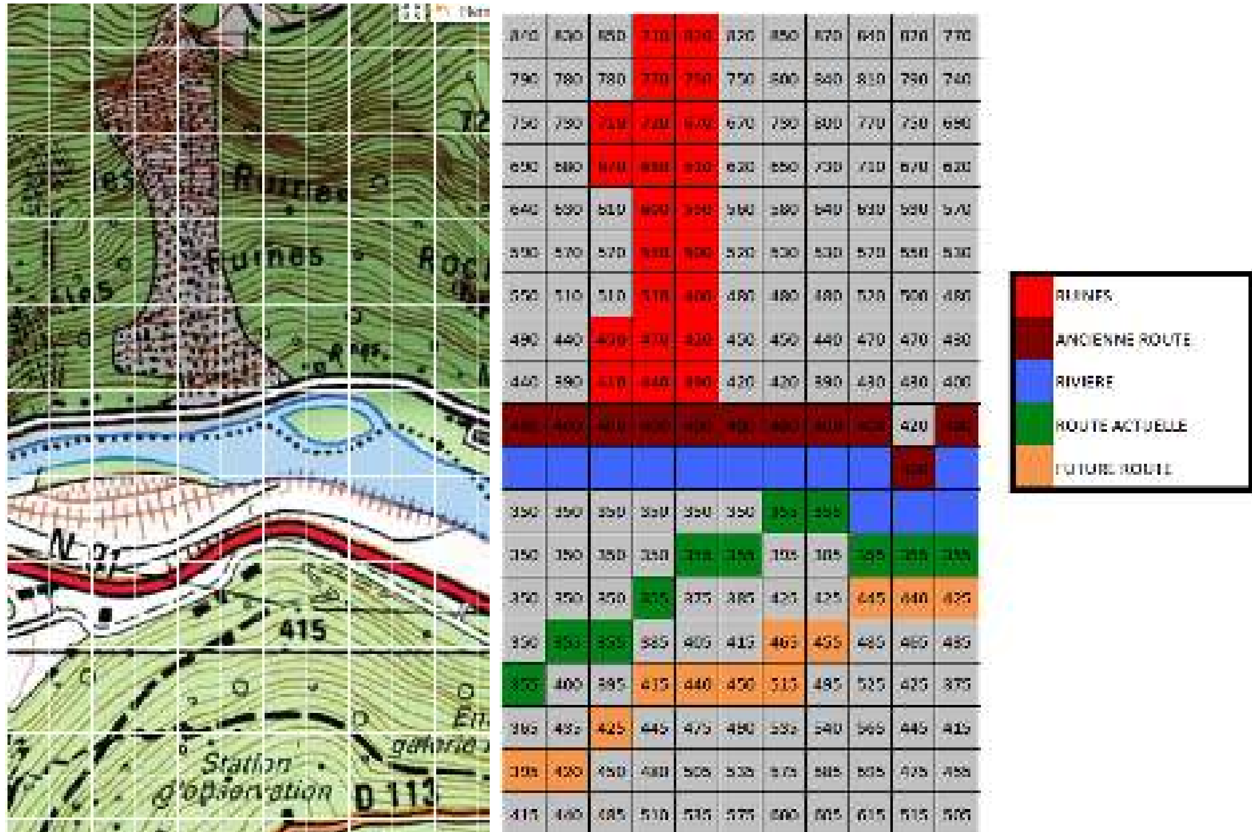


Figure 2 : Relevé des altitudes

2. Lissage de la topographie par extrapolation linéaire

Le trop faible nombre de points relevés ne nous permettant pas d'avoir un profil très lisse, nous avons décidé d'accroître la résolution de la modélisation de façon artificielle.

La fonction modif permet de multiplier la taille d'une matrice d'un facteur 2. altitudes étant de 11x19, modif(altitudes) donne une matrice de 22x38. Pour ce faire, nous considérons (de façon plus ou moins justifiée) que l'altitude évolue de façon linéaire entre deux points. On

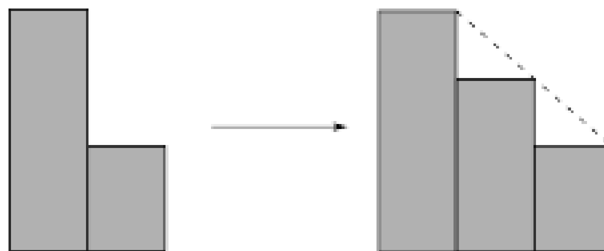


Figure 3 : Principe de l'extrapolation linéaire entre deux relevés topographiques

3. Représentation informatique en 3 dimensions

La représentation du site d'étude en 3 dimensions a été faite avec la fonction bar3, intégrée à Matlab. Cette fonction représente une matrice A par des barres tridimensionnelles dont la hauteur varie suivant la valeur des éléments de A, à l'image d'un histogramme.

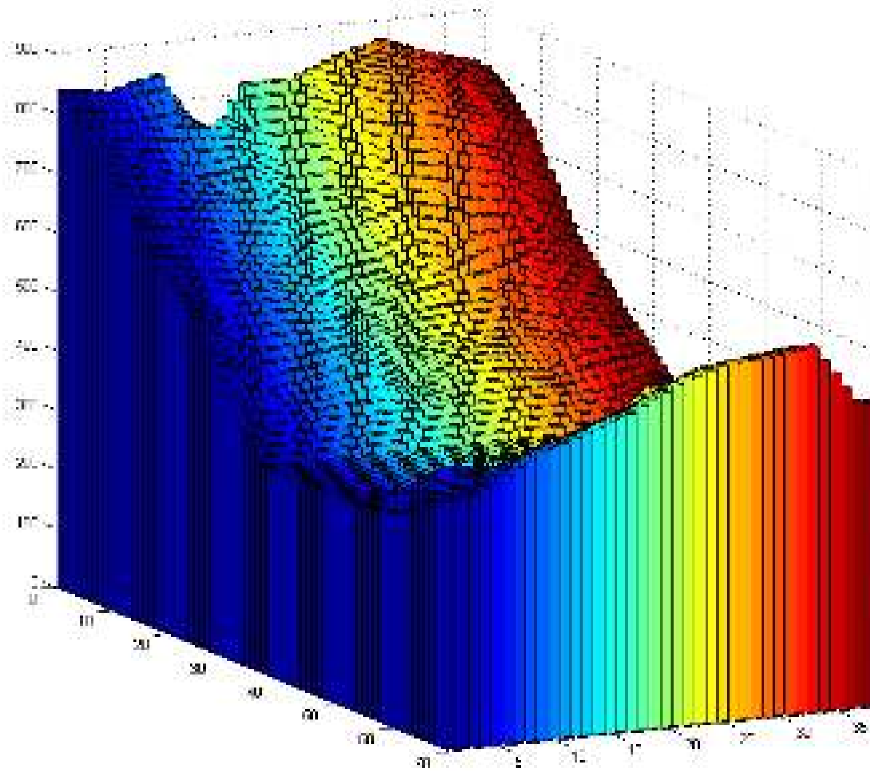


FIGURE 4 : Représentation tri-dimensionnelle du site de Sechilienne

II. Extraction et représentation d'un plan de coupe

Nous simulons la chute du bloque dans un univers en deux dimensions. Nous devons donc extraire une coupe, choisie par l'expérimentateur, de notre environnement en trois dimensions. Cette coupe doit pouvoir être choisie n'importe où à travers la matrice altitudes.

1. Extraction des altitudes du plan de coupe

Le profil de coupe, représenté par une matrice E (dont une ligne représente un segment) doit être extrait de la matrice altitudes. La construction de E, ensemble des segments symbolisant la coupe, se fait dans la fonction coupediago. On pose tout d'abord comme premier segment de E un segment vertical partant de l'origine du repère, ceci est la limite de notre coupe. Les lignes suivantes sont construites deux par deux avec, à chaque fois, un segment vertical et un segment horizontal.

Les segments horizontaux sont aux altitudes définies dans altitudes aux points par lesquels passe la coupe. Leur largeur est la moyenne des distances entre les points de la matrice altitudes.

2. L'algorithme de Bresenham

a. Principe de l'algorithme de Bresenham

Nous devions pouvoir faire des coupes dans notre matrice « altitude » suivant n'importe quelle direction. Hors une difficulté se posait car le nombre de case traversées par la coupe allait varier suivant la direction. Nous avons alors utilisé un algorithme pré-existant publié par Peter I. Corke de la MVTB sur une plateforme de partage MATLAB. Cette nous a permis d'obtenir les points aux coordonnées entières les plus proches de la coupe dans la matrice altitudes.

Bresenham2 utilise les coordonnées x0, y0 du point de départ et x1,y1 coordonnées du point d'arrivée. En sortie, ce programme nous donne un vecteur « points » composé des coordonnées des case de la matrice traversée par la coupe choisie.

b. Mise en oeuvre dans le code

Bresenham2 est ensuite utilisé dans la fonction coupediago. Coupediago à donc besoin des coordonnées du segment selon lequel nous allons faire la coupe car elles sont nécessaires pour le fonctionnement de Bresenham2. Ces coordonnées seront d'ailleur donnée par l'utilisateur pour le programme final,. elles constituerons les variables d'entrées de notre scripte crash2000. Coupediago a également besoin de la martice « altitude » pour connaître les hauteurs des différents points de coupe. Coupediago crée une matrice E composé d'autant de ligne qu'il y a des segments à tracer pour former le profil de notre coupe. Les quatre colonnes de E sont les coordonnées des chacun de ces segments.

E=coupediago(x0,y0,x1,y1,altitudes)

Bresenham2 intervient au début de la fonction coupediago. On obtient alors le vecteur « points ».

3. Représentation de la coupe

E est ensuite utilisé par la fonction environnement qui trace l'ensemble des segments contenus dans E.

La chute du bloque se fera alors par la suite sur cet environnement en deux dimensions.

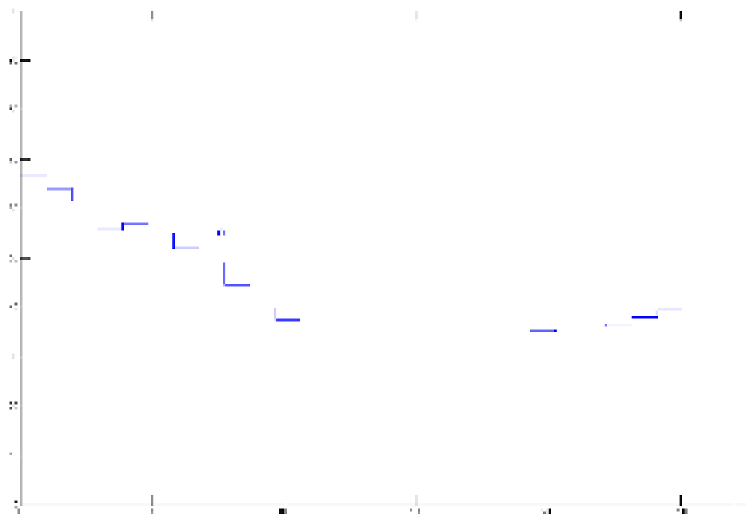


FIGURE 5 : Exemple d'une coupe localisée dans le site d'étude

III. Simulation de la chute par rebonds

1. Evolution des vecteurs vitesse et position

Une fois la coupe effectuée, nous avons un environnement qui correspond à une des directions qu'un bloc de roche est susceptible de prendre. Il faut donc maintenant modéliser le bloc par une boule et simuler sa chute pour pouvoir prévoir l'abscisse d'arrivée afin de la comparer à celle de la nouvelle route et vérifier qu'elle est hors danger.

Pour cela, il faut déterminer la position du bloc au cours du temps. On modélise le bloc par une boule de rayon r et de centre (x, y) par la fonction tracecercle.

a. Initialisation des vecteurs position, vitesse et accélération

Il faut trouver (x, y) qui varie au cours du temps. On initialise tout d'abord la position de la boule par un vecteur $T(0) = [x_i\ y_i\ v_{xi}\ v_{yi}\ a_{xi}\ a_{yi}]$ avec (x_i, y_i) la position initiale de la boule, (v_{xi}, v_{yi}) la vitesse initiale de la boule et (a_{xi}, a_{yi}) l'accélération initiale de la boule.

Les paramètres initiaux pourront être choisis pour coller au mieux à la réalité, on choisit ici

$$T(0) = \left[0\ \frac{altitudes(1,y0)}{10} + 6\ 20\ -10\ 0\ 10\right]$$

b. Evolution temporelle - Méthode d'Euler

On va ensuite utiliser la méthode d'Euler pour remonter jusqu'à la position de la boule au cours du temps (x,y). En effet, la fonction acceleration permet de sortir l'accélération de la boule que nous considérons ici constante et égale à l'accélération du champ de pesanteur. On utilise cette accélération dans la fonction vitesse et par la méthode d'Euler, avec un pas de temps égal à dt, on remonte jusqu'à la vitesse. Enfin, on fait de même avec la fonction position et on réunit ces trois résultats dans la fonction trajectoire où on applique ces trois fonctions au vecteur initiale T pour construire un vecteur T1 qui va décrire les caractéristiques de la boule au cours du temps.

La méthode d'Euler donne :

$$\begin{aligned}
a_x(t + dt) &= a_x(t) \\
a_y(t + dt) &= a_y(t) \\
v_x(t + dt) &= v_x(t) + a_x(t).dt \\
v_y(t + dt) &= v_y(t) + a_y(t).dt \\
x(t + dt) &= x(t) + v_x(t).dt \\
y(t + dt) &= x(t) + v_y(t).dt
\end{aligned}$$

On se rapporte alors à la fonction trace pour réunir ces derniers résultats. Celle-ci est soumise à une condition sur la norme de la vitesse : si $v < 0,1$ alors le programme met fin à l'expérience. Cette fonction va tout d'abord tester s'il y a intersection avec l'environnement ou pas. S'il n'y a pas d'intersection, il suffit d'appliquer tracecercle au vecteur T1 trouvé par la fonction trajectoire. S'il y a intersection avec l'environnement, il va falloir basculer vers une autre fonction, la fonction rebond.

2. Gestion des rebonds

Cette gestion intervient en deux étapes : le test des intersections avec l'environnement puis la modification de la trajectoire s'il y a intersection.

a. Détection d'une intersection avec l'environnement

Lorsque la boule rencontre un segment de l'environnement, il va falloir traiter plusieurs cas : c'est ce que fait la fonction intersection. Celle-ci va tester trois cas possibles : croisement avec un segment horizontal, croisement avec un segment vertical ou croisement avec un coin. La fonction intersection renseigne un vecteur V, par défaut $V = [0\ 0\ 0]$. S'il y a une intersection verticale, $V(1)$ prend la valeur 1. S'il y a une intersection, $V(2) = 1$. Enfin la troisième coordonnée de V retient le numéro (ligne de E) du segment avec lequel a lieu l'intersection.

b. Modification de la trajectoire

La fonction rebond a pour charge de modifier la trajectoire de la boule si une intersection est détectée. S'il s'agit d'une intersection avec un segment horizontal, on change juste le signe de l'ordonnée de la vitesse. Si le segment est vertical, on change juste le signe de l'abscisse de la vitesse. Enfin, pour un coin, on change le signe des deux composantes de la vitesse. On modélise également un amortissement par un facteur multiplicatif (< 1) qui simule les pertes énergétiques intervenant à chaque rebond dans la réalité.

3. Modification dynamique du profil topographique par le bloc rocheux

Cette fonctionnalité permet au bloc de "détruire" l'environnement s'il arrive avec une trop grande vitesse. Les dommages subis par l'environnement sont alors proportionnels à la vitesse du bloc. Cette gestion se trouve dans la fonction trace :

```
if V(2)==1
        v=sqrt(T(3)^2+T(4)^2)
        if v>10 %condition de vitesse
            E(V(3),2)=E(V(3),2)-0.1*v %destruction proportionnelle a la vitesse
            E(V(3),4)=E(V(3),4)-0.1*v
            E(V(3)-1,4)=E(V(3)-1,4)-0.1*v
            E(V(3)+1,2)=E(V(3)+1,2)-0.1*v
            environnement(E)
        end
    end
```

Cette modification de l'environnement n'intervient que si la vitesse du bloc est suffisante (arbitrairement on prend v>10).

IV. Tests unitaires de fonctions, résultats de la simulation et limites du code

1. Tests unitaires de fonctions sur un exemple simple

Pour vérifier leur bon fonctionnement, les fonctions de notre code ont été testées sur un exemple simple, un site caractérisé par la matrice suivante :

$$altitudes = \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}$$

On souhaite réaliser une coupe dans la première colonne de la matrice. Voici quelques exemples des tests que nous avons effectués à chaque étapes de l'écriture du code :

a. Test de coupediago

```
E=coupediago(1,1,1,2,altitudes)

E =

         0         0         0    1.0000
         0    0.1000    1.0000    1.0000
    1.0000    1.0000    1.0000    2.0000
    1.0000    2.0000    2.0000    2.0000
    2.0000    2.0000    2.0000         0
```

b. Test de environnement

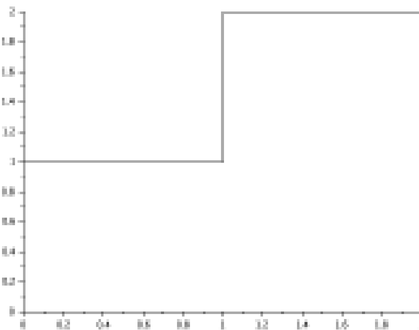


Figure 6 : Test de la fonction environnement

c. Test de intersection

On considère une boule de diamètre $0,3$ et on teste son intersection avec l'environnement dans deux configurations :

```
%intersection avec le segment numero 3 (vertical)

V=intersection(0.3, 0.7, 1.6,E)

V=
    1    0    3


%intersection avec le segment numero 2 (horizontal)

V=intersection(0.3, 0.5, 1.3)

V=
    0    1    2
```

2. Résultats de simulation

On cherche à savoir quelle distance horizontale un bloc lancé du haut des ruines de Séchilienne avec une vitesse initiale d'environ 14 m/s et orienté de 45° vers le bas.

La fonction resultat va réaliser une simulation de chute pour chaque tranche verticale de la matrice altitudes et va retenir à chaque fois l'abscisse maximale atteinte par le bloc.

On pourra ensuite comparer les abscisses maximales atteintes et les abscisses des routes (actuelle et future) dans la vallée de la Romanche.

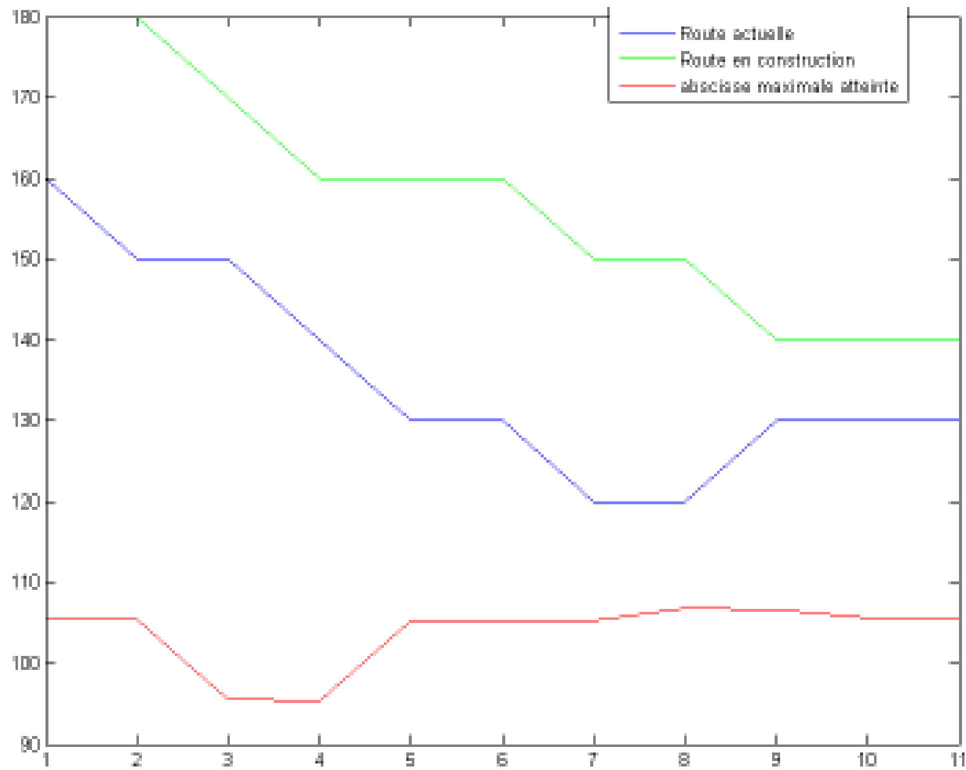


Figure 7 : Positions maximales atteintes par le bloc rocheux et tracés des routes actuelle et future

On voit dès lors que le bloc rocheux n'atteint ni la route actuelle (et a fortiori) ni la future route. Ces résultats, plus optimistes que ceux faits par des professionnels, sont cependant très dépendants des nombreux paramètres du programme (vitesse initiale, angle de lancement, facteurs d'amortissement, ...). Des phénomènes ont également été négligés comme les forces de frottements.

3. Limites du code

Notre code présente deux principales limites :

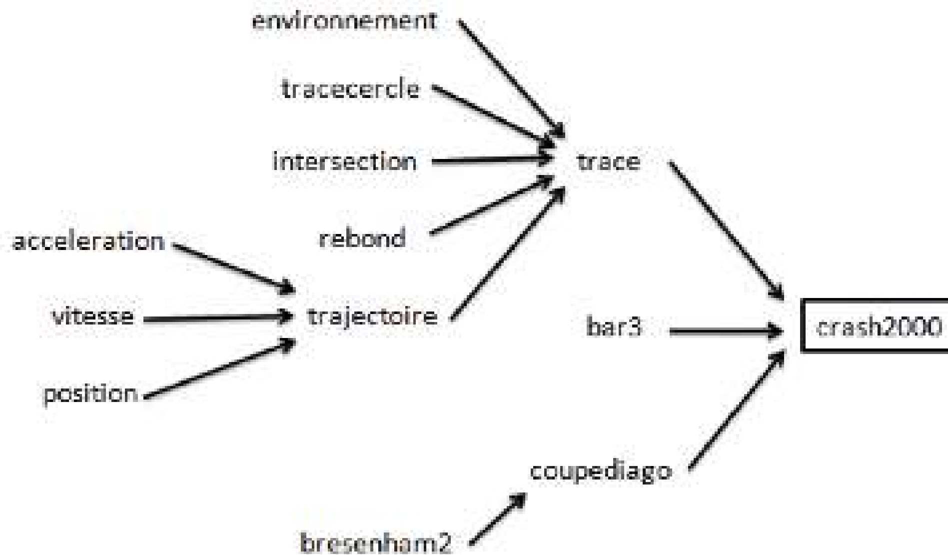
- Pour une vitesse trop élevée la boule peut traverser l'environnement. Cela est dû à un pas de temps trop élevé, la fonction intersection n'a alors pas le temps de 'voir' l'intersection.
- L'écart à la réalité est relativement important en raison du grand nombre de paramètres à régler (vitesse initiale, amortissement, ...) et des approximations (négligence des forces de frottement par exemple).

Conclusion

Ces quelques lignes de code nous ont permis d'obtenir une simulation, plus ou moins réaliste, de la chute d'un bloc rocheux sur les ruines de Sechilienne. Nous retenons du cours d'informatique que Matlab peut être très pratique dans la réalisation d'itérations numériques et qu'une certaine rigueur est à respecter dans l'écriture des commentaires sans quoi on se perd assez vite.

Annexes

Architecture du code



Listing des script et fonctions

Script principal

```
clf
disp('bienvenue sur notre programme de simulation crash 2000 !')
bar3(altitudes)

disp('voici une carte de Sechilienne')
disp('nous allons verifier si la future route est a l abris des eboulements futurs')
y0=input('choisissez une abscisse de depart de la coupe x0 ( entre 1 et 11 ): ');
x0=input('choisissez une ordonnÈe de depart de la coupe y0 ( entre 1 et 18 ) : ');
y1=input('choisissez une abscisse d''arrivÈe de la coupe x1 ( entre 1 et 11 ): ');
x1=input('choisissez une ordonnÈe d''arrivÈe de la coupe y1 ( entre 1 et 18 ): ');
clf
%conversion du script en fonction :
%function h=crash2000(altitudes,x0,y0,x1,y1)
E=coupediago(x0,y0,x1,y1,altitudes);
%on extrait un plan de coupe de la matrice altitudes
T=[0 (altitudes(1,y0))/10+6 20 -10 0 10];
%on initialise le depart de la boule (position et vitesse),
%l'acceleration est celle du champ de pesanteur (0,10)
h=trace(T,E);
```

Fonctions annexes

```
function E=coupediago(x0,y0,x1,y1,altitudes)
% E=coupediago(x0,y0,x1,y1,altitudes)
% x0,y0,x1,y1 coordonnÈes du segment de coupe dans la matrice "altitudes"
% E marice[size(points),4]
    altitudes=altitudes/10; % Èchelle verticale de 0 ‡ 100
    points=bresenham2(x0,y0,x1,y1)
% prend 2 pt (extremites du segment de coupe) et donne un vecteur "points"
%compose des coordonnees des points situes a proximite du segment
    E(1,:)=[0 0 0 altitudes(points(1,1),points(1,2))];
% initialisation ( premier segment vertical )
    d=sqrt((x0-x1)^2+(y0-y1)^2)/size(points,1);
    for i=1:size(points,1)-1
        E(2*i,:)=[E(2*i-1,3) E(2*i-1,4) 10*i*d altitudes(points(i,1),points(i,2))];
% reprend le point precedent puis donne le nouveau point
% pour former le segment vertical
        E(2*i+1,:)=[E(2*i,3) E(2*i,4) 10*i*d altitudes(points(i+1,1),points(i+1,2))];
% idem pour le segment horizontal
    end
    E(size(E,1)+1,:)=[E(size(E,1),3) E(size(E,1),4) E(size(E,1),3)+10 E(size(E,1),4)];
% dernier segment horizontal
    E(size(E,1)+1,:)=[E(size(E,1),3) E(size(E,1),4) E(size(E,1),3) 0];
% dernier segment vertical ( retombe jusqu'‡ 0 )
end
```

```
function h=trace(T,E)
% h=trace(T,E)
% trace simule la chute d'une boule de coordonnÈes (x,y),
% de vitesse (vx,xy) et d'acceleration (ax,ay)
% stockÈes dans T=[x y vx vy ax ay] sur l'environnement E (chaque ligne de
% E reprÈsente un segment par ses coordonnÈes x0 y0 x1 et y1.
% h donnera l'abscisse maximale atteinte par la boule lors de sa chute
h=0%initialisation de l'abscisse maximale
while sqrt(T(3)^2+T(4)^2)>0.1%condition sur la norme de la vitesse
    V=intersection(1,T(1),T(2),E);%teste l'intersection ‡ chaque pas de temps
    while V(1)==0 && V(2)==0%tant qu'il n'y a pas d'intersection,
% ni verticale ni horizontale
        T=trajectoire(T);
        tracecercle(1,T(1),T(2));
        hold on
        environnement(E);
        plot(T(1),T(2))
        axis([0 350 0 100]);
        pause(0.01);
        clf
        V=intersection(1,T(1),T(2),E);
        if T(1)>h%si on trouve une abscisse plus grande
            h=T(1)%on remplace l'abscisse maximale
        end
    end

    %modification dynamique de l'environnement, sous condition de vitesse
    %et proportionnellement ‡ la vitesse
    if V(2)==1
        v=sqrt(T(3)^2+T(4)^2)
        if v>10%condition de vitesse
            E(V(3),2)=E(V(3),2)-0.1*v%Ècrasement proportionnel ‡ la vitesse
            E(V(3),4)=E(V(3),4)-0.1*v
            E(V(3)-1,4)=E(V(3)-1,4)-0.1*v
            E(V(3)+1,2)=E(V(3)+1,2)-0.1*v
            environnement(E)
        end
    end
    T=rebond(V(1),V(2),T);
    T=trajectoire(T);
end
```

```
function T1=rebond(g,k,T)
%T1=rebond(g,k,T)
%rebond permet de simuler les rebonds de la balle sur le sol ou les cÙtÈs.
%On ajoute aussi un amortissement qui reprÈsente les pertes liÈes aux
%rebonds et aux intempÈries
%T est le vecteur contenant les vitesses et autres conditions initiales
%g correspond ‡ la prÈsence d'un segment vertical
%k correspond ‡ la prÈsence d'un segment horizontal
T1=T;
if g==1 &&g~=k %rebond vertical
    T1(3)=-T(3)*0.95; %on ajoute ici un amortissement selon x
    T1(4)=T(4)*0.75; %on ajoute ici un amortissement selon y
elseif k==1 && k~=g % rebond horizontal
    T1(4)=-T(4)*0.75;
    T1(3)=T(3)*0.95;
elseif k==1 && g==1 % coin
    T1(3)=-T(3)*0.95;
    T1(4)=-T(4)*0.75;
end
a=floor(rand*7);
if a==0
    disp(' ! BOUM ! ')
elseif a==1
    disp(' ! BAM ! ')
elseif a==2
    disp(' ! BANG ! ')
elseif a==3
    disp(' ! CRACK ! ')
elseif a==4
    disp(' ! PAF ! ')
elseif a==5
    disp(' ! BIM ! ')
elseif a==6
    disp(' !BADABOUM ! ')
end
end
```

```
function environnement(E)
%environnement(E)
% trace l'environnement E stockÈ sous forme de segments, E(i) code un
% segment vertical si i est impair, horizontal sinon
for i=1:size(E,1)
    X(2*i)=E(i,1);
    Y(2*i)=E(i,2);
    X(2*i+1)=E(i,3);
    Y(2*i+1)=E(i,4);
end
plot(X,Y)
```

```
function tracecercle(r,x,y)
%tracecercle(r,x,y)
%tracecercle trace notre boule de rayon r et de centre (x,y) qui varie au
%cours du temps
phi = [-pi:pi/24:pi];%on parcourt le cercle avec un pas de pi/24
m = x+r*cos(phi);
n = y+r*sin(phi);
plot(m,n)
end
```

```
function V=intersection(r,x,y,E)
%V=intersection(r,x,y,E)
% teste l'intersection entre l'environnement E et la boule de rayon r et de
% coordonnÈes (x,y)

V=[0 0 0];
%V(1)=1 si il y a intersection verticale
%V(2)=1 s'il y a intersection horizontale
%V(3) retient la ligne de E (c'est ‡ dire le segment) o˘ se produit
%l'intersection

for i=1:size(E,1)
    if E(i,1)==E(i,3)%si c'est un segment vertical
        if y-r<E(i,4) && y+r>E(i,2)
            if abs(x-E(i,1))<r
                V=[1,0,i];
            end
        end
    elseif E(i,2)==E(i,4)% si c'est un segment horizontal
        if x-r<E(i,3) && x+r>E(i,1)
            if abs(y-E(i,2))<r
                V=[0,1,i];
            end
        end
    end
end
```

```
function T1=trajectoire(T)
%T1=trajectoire(T)
%trajectoire permet de construire un vecteur T1 dans lequel sont reunis les
%caracteristiques de la balle : la position, la vitesse et l'acceleration
[T1(5),T1(6)]=acceleration(T(5),T(6)); %c'est le vecteur acceleration de la boule
[T1(3),T1(4)]=vitesse(T(3),T(4),T(5),T(6),0.01);
%c'est le vecteur vitesse de la boule
[T1(1),T1(2)]=position(T(1),T(2),T(3),T(4));
%T1(1) et T1(2) sont respectivement l'abcisse et l'ordonnee de la boule
end
```

```
function[c,d]=acceleration(ci,di)
%[c,d]=acceleration(ci,di)
%acceleration permet de calculer le vecteur acceleration
c=0;
d=-9.81;
end
```

```
function[a,b]=vitesse(ai,bi,ci,di,dt)
%[a,b]=vitesse(ai,bi,ci,di,dt)
%vitesse permet de calculer le vecteur vitesse par la mÈthode d'Euler
%ci et di sont les coordonnÈes du vecteur acceleration tirÈes de la
%fonction du mÍme nom et dt est le pas de temps
a=ai+dt*ci;
b=bi+dt*di;
end
```

```
function[x,y]=position(xi,yi,a,b)
%[x,y]=position(xi,yi,a,b)
%position nous permet de construire le vecteur position de la boule
% xi yi position initial, [a b] est le vecteur vitesse tirÈ de la fonction
% du mÍme nom
x=xi+0.01*a;
y=yi+0.01*b;
end
```

```
function resultat(altitudes)
%resultat(altitudes)
for i=1:size(altitudes,1)
    H(i)=crash2000(altitudes,1,1,size(altitudes,1),i)
%on releve les abscisses maximales atteintes
%pour chaque colonne de la matrice altitudes
end
X=[1:size(altitudes,1)]
plot(X,H)
end
```

```
function altitudes3=modif(altitudes)
%altitudes3=modif(altitudes)
%Ce programme permet d'amÈliorer la rÈsolution de l'environnement
%altitudes Ètant la matrice des altitudes relevÈes sur la carte des
%Seschiliennes
%altitudes3 est la nouvelle matrice des altitudes
for i=1:size(altitudes,1)
    for j=1:size(altitudes,2)-1
        altitudes2(i,2*(j-1)+1)=altitudes(i,j)
    %on garde les valeurs de altitudes dans les colonnes impaires de altitudes2
        altitudes2(i,2*(j-1)+2)=(altitudes(i,j)+altitudes(i,j+1))/2
    %dans les colonnes paires, on effectue la moyenne entre
    %les deux valeurs qui l'entourent
    end
end

for i=1:size(altitudes,1)
    for j=1:size(altitudes2,2)
        altitudes3(2*i-1,j)=altitudes2(i,j);
    %on garde les valeurs de altitudes2 dans les lignes impaires de altitudes3
    end
end

for i=1:size(altitudes3,1)/2
    for j=1:size(altitudes3,2)
        altitudes3(2*i,j)=(altitudes3(2*i-1,j)+altitudes3(2*i+1,j))/2
    %dans les lignes paires, on effectue la moyenne entre
    %les deux valeurs des lignes qui l'entourent
    end
end


%for j=1:size(altitudes2,2)
%for i=1:size(altitudes,1)
%altitudes2(2*i,j)=
%altitudes2(2*i+1,j)=altitudes2(


% altitudes3(:,size(altitudes3,2)+1)=altitudes3(:,size(altitudes3,2))
end
```