On cherche à fabriquer un toboggan partant du point $A = (x_A, y_A)$ et arrivant au point $B = (x_B, y_B)$ de telle sorte que le temps mis par une bille qui part sans vitesse du point A arrive le plus vite possible au point B. On suppose que cette bille n'est soumise qu'à la gravitation et à la réaction du toboggan, sans frottement. Si la forme du toboggan est donnée par la courbe d'équation $y = f(x)$, le temps mis par la boule pour aller de A vers B est

$$T(f) = \int_{x_A}^{x_B} \sqrt{\frac{1 + (f'(x))^2}{2\,g\,(y_A - f(x))}}\, dx$$

o`u $g = 9,8 m\, s^{-2}$ est l'accélération de la pesanteur. Dans la suite on suppose $x_A = 0$ et $B = (1, 0)$, on cherche donc une solution du problème

$$(P) \qquad \inf \{T(f) : \ f : [0,1] \to \mathbb{R}, f(0) = y_A, f(1) = 0\}\,.$$

Afin de résoudre ce problème de manière approchée, on le discrétise par la méthode des éléments finis. Pour $N \geq 1$ fixé, on considère l'ensemble $\mathcal{A}_N$ des fonctions affines par morceaux $f : [0,1] \to \mathbb{R}$ telles que $f(0) = y_A, f(1) = 0$ et

$$\forall n \in \{0, \dots, N\}, \qquad f \text{ affine sur } \left[\frac{n}{N+1}, \frac{n+1}{N+1}\right].$$

Toute fonction $f \in \mathcal{A}_N$ est alors caractérisée par le vecteur $y \in \mathbb{R}^{N+2}$ donné par

$$\forall n \in \{0, \dots, N+1\}, \qquad y_n = f\left(\frac{n}{N+1}\right).$$

L'ensemble des vecteurs ainsi obtenus est alors

$$\mathcal{B}_N = \{y \in \mathbb{R}^{N+2} \ : \ y_0 = y_A,\ y_{N+1} = 0\}.$$

On doit alors résoudre le problème :

$$(P_N) \qquad \inf\{J(y) \ : \ y \in \mathcal{B}_N\}$$

où $J(y) = T(f)$ pour la fonction affine $f \in \mathcal{A}_N$ associée au vecteur y.

Pour commencer, on va donc déclarer les variables du problème discrétisé sous Octave, et afficher les valeurs pour l'utilisateur. On définit donc N, y_A et y_B :

In [47]:
```
N=9;
printf("nombre de points de discretisation : N = %d \n",N
);
yA=0.2;
yB=0;
printf("valeurs de y_A et y_B : y_A = %f , y_B = %f\n\n",
yA, yB);
```

```
nombre de points de discretisation : N = 9
valeurs de y_A et y_B : y_A = 0.200000 , y_B = 0.000000
```

On aura aussi besoin d'un vecteur initial pour la méthode de la plus grande pente, on prend l'interpolation affine entre les valeurs y_A et y_B sur les $N+2$ points $\frac{i}{N+1}$ pour $i \in \{0, \dots, N+1\}$. Ainsi le vecteur y est de taille $N+2$, avec $y_1 = y_A$ et $y_{N+2} = y_B$ fixés, et les autres points sont obtenus par interpolation :

In [48]:
```
y=zeros(N+2,1);
i=1;
while (i <= N+2)
  y(i) = ((N+2-i)*yA+(i-1)*yB)/(N+1);
  i=i+1;
endwhile
y
```

```
y =

   0.20000
   0.18000
   0.16000
   0.14000
   0.12000
   0.10000
   0.08000
   0.06000
   0.04000
   0.02000
   0.00000
```

Désormais on définit les paramètres ρ (le pas) et ε (la précision) pour la méthode de la plus grande pente :

In [49]:
```
rho=10^(-2);
epsilon=10^(-2);
eps = epsilon^2;
```

On va maintenant déclarer une fonction funcJ pour calculer le temps $T_N(y)$ associé au toboggan correspondant à la fonction affine dont les valeurs aux points de discrétisation sont contenues dans y. Pour cela, on s'appuie sur la formule

$$J(y) \;=\; T(f) \;=\; \sum_{n=0}^{N} \int_{\frac{n}{N+1}}^{\frac{n+1}{N+1}} \sqrt{\frac{1+(f'(x))^2}{y_A - f(x)}}\, dx,$$

où on oublie volontairement la constante $2g$. Chaque terme de cette somme ne dépend que de y_n et y_{n+1} et peut être calculé par la formule :

$$\int_{\frac{n}{N+1}}^{\frac{n+1}{N+1}} \sqrt{\frac{1+(f'(x))^2}{(y_A - f(x))}}\, dx = \frac{2\sqrt{1+((N+1)(y_{n+1}-y_n))^2}}{(N+1)(y_{n+1}-y_n)} \left(\sqrt{y_A - y_n} - \sqrt{y_A - y_{N+}}\right.$$

$$= \frac{2\sqrt{1+((N+1)(y_{n+1}-y_n))^2}}{(N+1)(\sqrt{y_A - y_n} + \sqrt{y_A - y_{N+1}})}.$$

Notons que cette dernière formule est bien définie même dans le cas $y_n = y_{n+1}$.

On obtient donc la fonction suivante en sommant ces N termes :

In [50]:
```
function r=funcJ(x)
  r=0.;
  n=length(x);
  yA=x(1);
  i=1;
  while (i <= n-1)
    a = x(i);
    b = x(i+1);
    r = r + 2*sqrt(1+((b-a)*(n-1))**2)/((n-1)*(sqrt(yA-a)
+ sqrt(yA-b)));
    i = i+1;
  endwhile
endfunction
```

On utilise ensuite une fonction gradJ pour la différenciation de J, dans laquelle on fait attention à ce que les dérivées partielles pour la première et la dernière coordonnée soient nulles : il ne faut pas faire varier les valeurs y_A et y_B dans le vecteur y !

Les dérivées partielles sont calculées grâce à la formule de différence finie :

$$\frac{\partial J}{\partial x_i} \;=\; \frac{J(x + h\, e_i) - J(x - h\, e_i)}{2}$$

où e_i est le i-ème vecteur de la base canonique de $\mathbb{R}^{N+2}$.

In [51]:
```
function z=gradJ(x)
  h=10^(-8);
  n=length(x);
  i=2;
  z=zeros(n,1);
  while (i <= (n-1))
    vecbase = zeros(n,1);
    vecbase(i)=1;
    z(i)=(funcJ(x+h*vecbase)-funcJ(x-h*vecbase))/(2*h);
    i=i+1;
  endwhile
endfunction
```

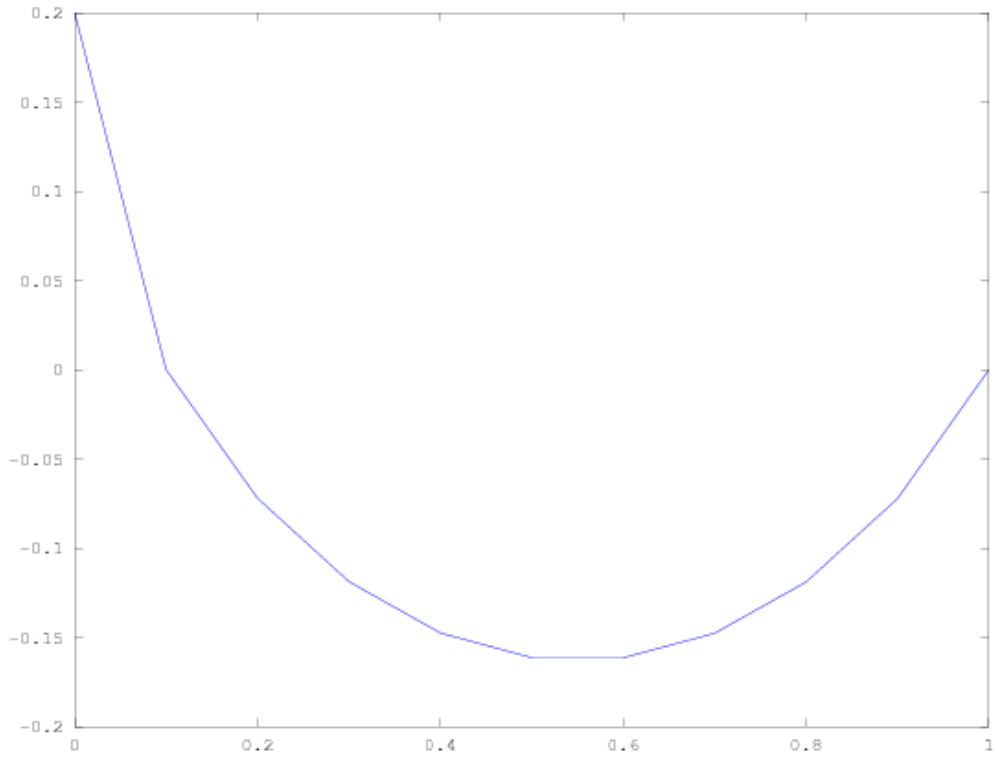
Enfin on applique la méthode de la plus grande pente et on affiche les résultats : le vecteur y et le nombre d'itérations k :

In [52]:
```
k=0; #compteur
while (gradJ(y)'*gradJ(y) > eps)
  y=y-rho*gradJ(y);
  k=k+1;
  #printf("test k=%d : %f\n", k,sqrt(gradJ(y)'*gradJ(y)))
#Afficher pour chaque iteration la valeur du test
  endwhile
printf("solution calculéee :\n");
y
printf("nombre d'iteration k = %d\n", k);
```

```
solution calculéee :
y =

   0.20000
   0.00008
  -0.07206
  -0.11881
  -0.14759
  -0.16143
  -0.16148
  -0.14774
  -0.11901
  -0.07227
   0.00000

nombre d'iteration k = 211
```

Et pour finir, on trace le toboggan obtenu. Pour cela, il faut définir le vecteur x des abscisses, puis tracer.

In [53]:
```
x=zeros(N+2,1);
i=1;
while (i <= N+2)
  x(i) = (i-1)/(N+1);
  i=i+1;
endwhile

plot(x,y);
```



On pourra tester les expériences suivantes :

- faire varier N dans $\{1, 5, 10, 20\}$,
- faire varier ρ dans $\{10^{-1}, 5 \times 10^{-2}, 10^{-2}, 5 \times 10^{-3}, 10^{-3}\}$
- faire varier ε dans $\{10^{-2}, 10^{-4}, 10^{-6}\}$

Pour pouvoir prendre des valeurs plus grandes de N, il faut faire appel à la recherche linéaire de Wolfe afin de calculer un meilleur pas que le pas constant à chaque iteration de la boucle principale.

Rappelons que cette méthode fait appel à deux fonctions : une fonction principale Wolfe_search qui cherche un intervalle $[t_g, t_d]$ dans lequel il existe au moins un pas qui satisfait les conditions de Wolfe, et une fonction Zoom_search qui cherche un pas ρ dans dans cet intervalle.

Rappelons que dans la méthode de Wolfe, il s'agit de trouver un pas ρ tel que

$$q(\rho) \leq q(0) + c_2\, q'(0)\, \rho \qquad \text{et} |q'(\rho)| \leq -c_1\, q'(0)$$

où $0 < c_1 < c_2 < 1$ sont des paramètres fixés, et q est la fonction définie sur $\mathbb{R}_+$ par

$$q(t) \;=\; J(y - t\, \nabla J(y)).$$

On rappelle que dans ce cas $q'(t) = -\nabla J(y - t\, \nabla J(y)) \cdot \nabla J(y)$, et en particulier $q'(0) = -\|\nabla J(y)\|^2$.

Dans les fonctions suivantes, on doit passer les paramètres suivants :

- le pas ρ utilisé à l'itération précédente;
- le vecteur y de l'itération précédente;
- $g = \nabla J(y)$;
- $qp_0 = q'(0) = -\|\nabla J(y)\|^2$;
- les paramètres c_1 et c_2, ainsi que le nombre maximal d'itérations dans la recherche de Wolfe.

In [54]:
```
function t=Wolfe_search(rho,y,g,qp_0,c1,c2,nmax)
  q_0=funcJ(y);
  c1qp=c1*qp_0;
  c2qp=c2*qp_0;
  test=0;
  tg=0.;
  td=2*rho;
  t=rho;
  while (test==0)
    qt=funcJ(y-t*g);
    if (qt>(q_0+c2qp*t))
      test =1;
      t = Zoom_search(tg,t,y,g,q_0,c1qp,c2qp,nmax);
    else
      qp=-gradJ(y-t*g)'*g;
      if (abs(qp) <= -c1qp)
        test=2;
      elseif (qp >=0)
        test=3;
        t = Zoom_search(tg,t,y,g,q_0,c1qp,c2qp,nmax);
      else
        tg=t;
        td=10*td;
        t=0.5*(tg+td);
      endif
    endif
  endwhile
endfunction

function t=Zoom_search(tg,td,y,g,q_0,c1qp,c2qp,nmax)
  n=0;
  testZ=0;
  while and(testZ==0,n<nmax)
    t=0.5*(tg+td);
    qt=funcJ(y-t*g);
    if (qt>(q_0+c2qp*t))
      td=t;
    else
      qp=-gradJ(y-t*g)'*g;
      if (abs(qp) <= -c1qp)
        testZ=1;
      elseif (qp >=0)
        td=t;
      else
        tg=t;
      endif
    endif
    n=n+1;
  endwhile
endfunction
```

A vous d'intégrer cette méthode dans une méthode de plus grande pente !