

距離関数

春日 悠

2022 年 9 月 24 日

目 次

1	Hamilton-Jacobi-Belman 方程式	1
2	Eikonal 方程式	2
3	距離関数の計算	3

1 Hamilton-Jacobi-Belman 方程式

Eikonal 方程式を導出するために, Hamilton-Jacobi-Belman 方程式について見る.

状態空間ベクトル $\mathbf{x}(t)$ を考え, これが入力ベクトル $\mathbf{u}(t)$ に応じて動くものとする. $\mathbf{x}(t)$ が動く時, あるコストがかかるとする. このときのコスト率の関数を $C(\mathbf{x}, \mathbf{u})$ とする. 「目的地」までのコストを評価する関数 $Q(\mathbf{x}, t)$ を考え, 時刻 t と $t + \Delta t$ の間のコストの変化量を次式で表す.

$$Q(\mathbf{x}, t + \Delta t) - Q(\mathbf{x}, t) = - \int_t^{t+\Delta t} C(\mathbf{x}(\tau), \mathbf{u}(\tau)) d\tau \quad (1)$$

右辺に負号がついているのは, 目的地に近い方が目的地に到るまでにかかるコストは小さいためである.

$\mathbf{u}$ に応じて $\mathbf{x}$ が動く時に, コストを最小化したい. 時刻 t と $t + \Delta t$ の間で考えると, 時刻 t における価値関数 $V(\mathbf{x}, t)$ を次式のように表す.

$$V(\mathbf{x}, t) = \min_{\mathbf{u}} \left\{ V(\mathbf{x}, t + \Delta t) + \int_t^{t+\Delta t} C(\mathbf{x}(\tau), \mathbf{u}(\tau)) d\tau \right\} \quad (2)$$

価値関数 $V(\mathbf{x}, t)$ は, コスト評価関数 $Q(\mathbf{x}, t)$ において, コストが最小になるような「経路」 $\mathbf{u}$ を選択したものである.

価値関数を $t + \Delta t$ で展開すると

$$V(\mathbf{x}, t + \Delta t) = V(\mathbf{x}, t) + \frac{\partial V(\mathbf{x}, t)}{\partial t} \Delta t + \nabla V(\mathbf{x}, t) \cdot \Delta \mathbf{x} + R \quad (3)$$

これを式 (2) に代入し、両辺を Δt で割った上で $\Delta t \rightarrow 0$ とすると、次式が得られる。

$$\frac{\partial V(\mathbf{x}, t)}{\partial t} + \min_{\mathbf{u}} \{ \nabla V(\mathbf{x}, t) \cdot \mathbf{F} + C(\mathbf{x}, \mathbf{u}) \} = 0 \quad (4)$$

ここで、 $\mathbf{F} = \dot{\mathbf{x}}$ である。これを Hamilton-Jacobi-Bellman 方程式という。

2 Eikonal 方程式

状態空間ベクトルを幾何学的空間内の位置と考え、コストを移動時間とする。位置 $\mathbf{x}$ の近傍の位置を $\mathbf{x} + \Delta \mathbf{x}$ とし、 $\mathbf{n}$ を単位ベクトルとして $\Delta \mathbf{x} = (\Delta l) \mathbf{n}$ と表す。 $\mathbf{x}$ での速度の大きさを $v(\mathbf{x})$ とすると、コストは $\Delta l / v(\mathbf{x})$ で表される。位置 $\mathbf{x}$ から目的地に到るまでの最短時間を $T(\mathbf{x})$ とすると

$$T(\mathbf{x}) = \min_{\Delta \mathbf{x}} \left\{ T(\mathbf{x} + \Delta \mathbf{x}) + \frac{\Delta l}{v(\mathbf{x})} \right\} \quad (5)$$

と表される。 $T(\mathbf{x} + \Delta \mathbf{x})$ を展開すると

$$T(\mathbf{x} + \Delta \mathbf{x}) = T(\mathbf{x}) + \nabla T(\mathbf{x}) \cdot \Delta \mathbf{x} + R \quad (6)$$

これを式 (5) に代入し、 $\Delta \mathbf{x} \rightarrow 0$ とすると、次式が得られる。

$$\min_{\mathbf{n}} \left\{ \nabla T(\mathbf{x}) \cdot \mathbf{n} + \frac{1}{v(\mathbf{x})} \right\} = 0 \quad (7)$$

これは、Hamilton-Jacobi-Bellman 方程式 (式 (4)) の定常版に相当する。

ここで、次のような関数 $\Pi(\mathbf{n})$ を考える。

$$\Pi(\mathbf{n}) = \nabla T(\mathbf{x}) \cdot \mathbf{n} + \frac{\mathbf{n} \cdot \mathbf{n}}{v(\mathbf{x})} \quad (8)$$

これを最小化したいので、変分をとって 0 とする。

$$\delta \Pi(\mathbf{n}) = \left(\nabla T(\mathbf{x}) + \frac{1}{v(\mathbf{x})} \mathbf{n} \right) \delta \mathbf{n} = 0 \quad (9)$$

$\delta \mathbf{n}$ は任意なので

$$\nabla T(\mathbf{x}) + \frac{1}{v(\mathbf{x})} \mathbf{n} = 0 \quad (10)$$

これより

$$\mathbf{n} = -v(\mathbf{x}) \nabla T(\mathbf{x}) \quad (11)$$

これを式 (7) に代入すると

$$\nabla T(\mathbf{x}) \cdot \nabla T(\mathbf{x}) = \frac{1}{(v(\mathbf{x}))^2} \quad (12)$$

これは、次のようにも表せる。

$$|\nabla T(\mathbf{x})|^2 = \frac{1}{(v(\mathbf{x}))^2} \quad (13)$$

あるいは

$$|\nabla T(\mathbf{x})| = \frac{1}{v(\mathbf{x})} \quad (14)$$

これを Eikonal 方程式という.

$T(\mathbf{x})$ は目的地までの移動時間であるが, 目的地というのは, $T(\mathbf{x}) = 0$ となる位置のことである. Eikonal 方程式を解く場合は, ある位置で $T(\mathbf{x}) = 0$ を設定する.

Eikonal 方程式において $v(\mathbf{x}) = 1$ とすると, $T(\mathbf{x})$ は距離を与える. これを $u(\mathbf{x})$ とすると

$$|\nabla u(\mathbf{x})| = 1 \quad (15)$$

である. ある位置に $u(\mathbf{x}) = 0$ を設定して解くと, 設定位置までの最短距離を与える. たとえば, ある形状の境界において $u(\mathbf{x}) = 0$ として方程式を解くと, $u(\mathbf{x})$ は境界までの距離の場を与える. このとき, $u(\mathbf{x})$ は距離関数 (distance function) と呼ばれる. また, 距離の符号を考えた方が便利ことがある. この場合の $u(\mathbf{x})$ は符号付き距離関数 (signed distance function: SDF) と呼ばれる.

3 距離関数の計算

距離関数 $u(\mathbf{x})$ を数値的に求める方法を考える. 距離関数を求める方法 (Eikonal 方程式を解く方法) としては, Fast Marching Method[1] が有名であるが, ここではその改良版である Fast Iterative Method[2] を示す.

簡単に, 2 次元の格子状の点群 $\mathbf{x}_{ij}$ を考える. 各点の距離関数の値の近似値を $u_{ij} \approx u(\mathbf{x}_{ij})$ とする. $\nabla u(\mathbf{x})$ を差分法で離散化する. 格子幅を h とし, 安定のため 1 次精度差分を用いると, x 方向成分は

$$\frac{u_{ij} - u_{i-1,j}}{h}, \quad \frac{u_{i+1,j} - u_{ij}}{h} \quad (16)$$

y 方向成分は

$$\frac{u_{ij} - u_{i,j-1}}{h}, \quad \frac{u_{i,j+1} - u_{ij}}{h} \quad (17)$$

である. 距離関数の値を $u_{ij} = 0$ の点群から順に計算していくことにすると, 計算が進んでいくほど値は大きくなるので, u_{ij} の隣接点の値は小さい方から取ればよい. 解くべき方程式を

$$|\nabla u(\mathbf{x})|^2 = 1 \quad (18)$$

とすると, 次のように表される.

$$\left(\frac{u_{ij} - u_x}{h}\right)^2 + \left(\frac{u_{ij} - u_y}{h}\right)^2 = 1 \quad (19)$$

ここで

$$\begin{aligned} u_x &= \min(u_{i-1,j}, u_{i+1,j}) \\ u_y &= \min(u_{i,j-1}, u_{i,j+1}) \end{aligned} \quad (20)$$

である。これを展開すると、 u_{ij} についての 2 次方程式が得られる。

$$u_{ij}^2 - (u_x + u_y)u_{ij} + \frac{1}{2}(u_x^2 + u_y^2 - h^2) = 0 \quad (21)$$

これを解いて

$$u_{ij} = \frac{u_x + u_y}{2} \pm \frac{1}{2}\sqrt{(u_x + u_y)^2 - 2(u_x^2 + u_y^2 - h^2)} \quad (22)$$

あるいは

$$u_{ij} = \frac{u_x + u_y}{2} \pm \frac{1}{2}\sqrt{2h^2 - (u_x - u_y)^2} \quad (23)$$

u_{ij} は隣接点の値より大きいはずなので、符号は正のほうをとる。

$$u_{ij} = \frac{u_x + u_y}{2} + \frac{1}{2}\sqrt{2h^2 - (u_x - u_y)^2} \quad (24)$$

平方根の中の値は正でなければならないので、 $|u_x - u_y| \leq \sqrt{2}h$ である必要がある。もし $|u_x - u_y| > \sqrt{2}h$ ならば、 u_{ij} は次式で (仮に) 計算する。

$$u_{ij} = \min(u_x + h, u_y + h) \quad (25)$$

上記の式を用いて u_{ij} を更新していく。まず、初期値として、境界に $u_{ij} = 0$ を設定し、それ以外に適当に大きな値を設定する。点の種類を fixed, front, free に分ける (呼び方はなんでもよい)。計算上は、値が決まったものを fixed リストに入れ、計算中のものを front リストに入れる。初期の段階では境界の点を fixed リストに入れ、その隣接点を front リストに入れる。

あとは Algorithm 1 の手順を繰り返す。

計算例を図 1, 2, 3 に示す。符号付き距離関数については、距離関数を求める時に内外の領域を区別しておき、内側の距離の符号を反転している。

3 次元の場合、 u_{ijk} は次式で計算すればよい。

$$u_{ijk} = \frac{u_x + u_y + u_z}{3} + \frac{1}{3}\sqrt{(u_x + u_y + u_z)^2 - 3(u_x^2 + u_y^2 + u_z^2 - h^2)} \quad (26)$$

参考文献

- [1] J.A. Sethian: A Fast Marching Level Set Method for Monotonically Advancing Fronts, Proceedings of the National Academy of Sciences, 93, 4 (1995).
- [2] W-K. Jeong nad R. Whitaker: A Fast Iterative Method for Eikonal Equations, SIAM Journal of Scientific Computing, Vol 30, Iss. 5 (2008).

Algorithm 1 Update u_{ij}

```
while  $front$  is not empty do
  for each  $(i, j)$  in  $front$  do
     $u_0 \leftarrow u_{i,j}$ 
     $u \leftarrow$  calculate  $u_{i,j}$ 
     $u_{i,j} \leftarrow u$ 
    if  $|u - u_0| < \epsilon$  then
      for each  $(k, l)$  in neighbours of  $(i, j)$  do
        if  $(k, l)$  is not in  $front$  then
           $u_0 \leftarrow u_{k,l}$ 
           $u \leftarrow$  calculate  $u_{k,l}$ 
          if  $u < u_0$  then
             $u_{k,l} \leftarrow u$ 
            add  $(k, l)$  to  $front$ 
          end if
        end if
      end for
      remove  $(i, j)$  from  $front$ 
    end if
  end for
end while
```

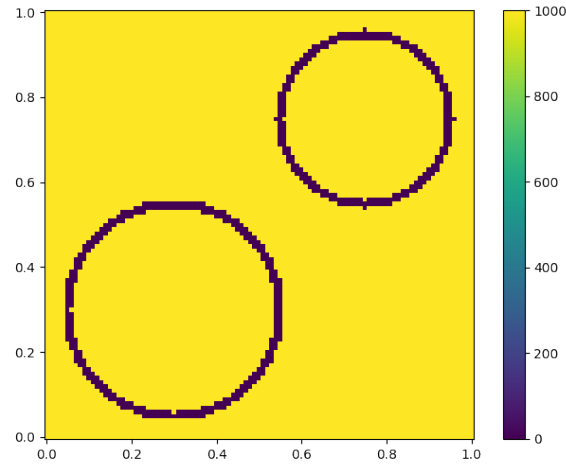


图 1: 境界

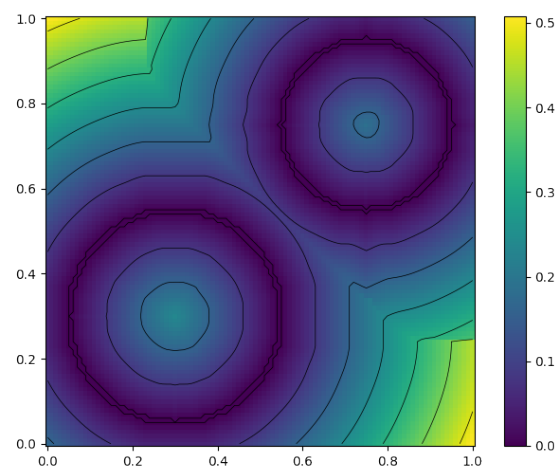


図 2: 距離関数

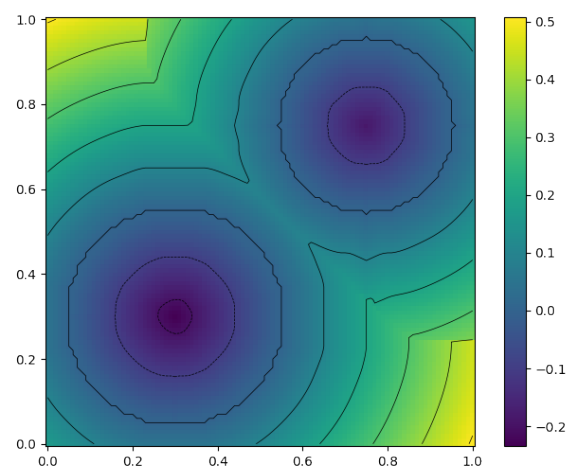


図 3: 符号付き距離関数