1 Introduction

2 Some hints on the implementation of the FD-FMM

3 The semi-Lagrangian FM method

4 Other acceleration techniques

5 Recent extensions of the FMM
   - Characteristics driven Fast Marching method (CFM)
   - BFM method for anisotropic front propagation problems
   - Pursuit-Evasion games
Outline

1. Introduction

2. Some hints on the implementation of the FD-FMM

3. The semi-Lagrangian FM method

4. Other acceleration techniques

5. Recent extensions of the FMM
   - Characteristics driven Fast Marching method (CFM)
   - BFM method for anisotropic front propagation problems
   - Pursuit-Evasion games
In the previous lecture we have presented the classical Fast Marching Method (FMM) based on a local finite difference scheme and we have outlined some features of this method. Now we are going to give more details on the FMM implementation. Then we will examine another Fast Marching Method based on the DP scheme (or semi-Lagrangian scheme). This is a good example to see how we can extend the FM idea to other approximation schemes.
Then we are going to examine **other types of acceleration techniques**: sweeping, group marching.

The goal is to compare these techniques with the FMM and get an hint on their advantages/disadvantages with respect to the following features:

- accuracy
- computational cost
- flexibility
The main questions are always the following:

- what is the initial configuration of the front in our numerical scheme?
- what is the position of the front at every iteration?
- which nodes will be necessary to compute the front configuration at the following time step?
- does the procedure converge to the correct viscosity solution?
- how many operations will be needed to get the right solution?
Outline

1. Introduction
2. Some hints on the implementation of the FD-FMM
3. The semi-Lagrangian FM method
4. Other acceleration techniques
5. Recent extensions of the FMM
   - Characteristics driven Fast Marching method (CFM)
   - BFM method for anisotropic front propagation problems
   - Pursuit-Evasion games
Partitioning the nodes

At every iteration (but the last one) we will have three sets of nodes:

- the **Accepted nodes**, where the values have been already computed and fixed
- the **Narrow Band nodes**, where the algorithm is computing
- the **Far nodes**, where the algorithm will compute in the next iterations

We will denote by $A(k)$, $NB(k)$ and $F(k)$ these subsets at the $k$-th iteration.

**Definition**

At the iteration $k$ the $NB(k)$ is the set of nodes which are first neighbors of the nodes in the Accepted region of the previous iteration, i.e. $A(k - 1)$. 

Let us get back to the FMM picture
The FD discretization

Let us write equation eikonal equation as

\[ T_x^2 + T_y^2 = \frac{1}{c^2(x, y)}. \]

The standard up-wind first order FD approximation is

\[
\left( \max \left\{ \max \left\{ \frac{T_{i,j} - T_{i-1,j}}{\Delta x}, 0 \right\}, -\min \left\{ \frac{T_{i+1,j} - T_{i,j}}{\Delta x}, 0 \right\} \right\} \right)^2 + \\
+ \left( \max \left\{ \max \left\{ \frac{T_{i,j} - T_{i,j-1}}{\Delta y}, 0 \right\}, -\min \left\{ \frac{T_{i,j+1} - T_{i,j}}{\Delta y}, 0 \right\} \right\} \right)^2 = \frac{1}{c_{i,j}^2},
\]

where, as usual, \( T_{i,j} = T(x_i, y_j) \).
The Basic Local Rule

The points involved in this formula are the stencil of the scheme and they are the "first neighbors" of the node where we are computing, i.e.

\[ N_{FD}(x_{ij}) = \{x_{i+1,j}, x_{i,j-1}, x_{i-1,j}, x_{i,j+1}\} \]
The Basic Local Rule

One can write the approximation in an explicit form as

\[ T_{ij} = S_{FD}(T_{i+1,j}, T_{i,j-1}, T_{i-1,j}, T_{i,j+1}) \]

This formula will be applied just on the Narrow Band nodes and will define our local operator \( S_{FD} \).
Initialization

Let us consider the simple case of a structured uniform grid in \( \mathbb{R}^2 \),

\[ Z \equiv \{ x_{ij} : x_{ij} = (x_i, y_j), x_i = i\Delta x, y_j = j\Delta y \} \]

we compute on \( Q \cap Z \) where \( \Omega_0 \subset Q \).

We set

- \( T_{ij} = 0 \) for every \( x_{ij} \in \Omega_0 \)
- \( T_{ij} = +\infty \) elsewhere
FMM Algorithm

The algorithm step-by-step, initialization

1. The nodes belonging to the initial front $\Gamma_0$ are located and labeled as *Accepted*. They form the set $\tilde{\Gamma}_0$. The value of $T$ of these nodes is set to 0.

2. $NB(1)$ is defined as the set of the nodes belonging to $N_{FD}(\Gamma_0)$, external to $\Gamma_0$.

3. Set $T_{i,j} = +\infty$ for any $(i,j) \in NB(1)$.

4. The remaining nodes are labeled as *Far*, their value is set to $T = +\infty$. 

---

M. Falcone
Fast Marching Methods for Front Propagation
The algorithm step-by-step, main cycle

Repeat

1. Compute \( T_{i,j} \) by the FD scheme on \( NB(k) \).
2. Find the minimum value of \( T_{i,j} \) in \( NB(k) \).
3. Label \((i,j)\) as Accepted, i.e. \( A(k+1) = A(k) \cup \{x_{ij}\} \).
4. Remove \((i,j)\) from \( NB(k) \).
5. Up-date the \( NB(k) \) to \( NB(k+1) \) adding the first neighbors of the NEW accepted node.
6. Set \( k = k+1 \).

Until ALL the nodes have been accepted.
Min-Heap Data Structure

At the $k$-th iteration we must compute the minimum value on the nodes belonging to the $NB(k)$. This can be done in many ways but can be rather expensive. An efficient procedure consists in the construction of a ”complete binary tree” with the property that the value at any given node is less or equal to the values of its children. This structure must be up-dated at every iteration (moving the values up and down along the tree). Since this keeps the minimum value always at the root of the tree we just pay the price of the up-date which is $O(\ln H)$ if there are $H$ nodes in the heap (delicate implementation).
Gauss-Seidel acceleration

The standard iterative method is

\[ T^{k+1} = S(T^k) \]

for a given \( T_0 \). The \( S \) operator is defined locally by our rule ad will be applied sequentially on the nodes \( NB(k) \). We can accelerate convergence up-dating the values of \( T \) as far as they are computed, i.e. in the 1 dimensional case

\[ T_{i}^{k+1} = S(T_{i-1}^{k+1}, T_{i}^{k}), \text{ for any } k, i \]

However, the improvement obtained by this ”re-alimentation” algorithm is very limited.
Boundary conditions

In every PDE problem we must set up boundary conditions. We have a natural Dirichelet boundary condition on $T = \Omega_0$, $T(x) = 0$. We do NOT have a "natural" boundary condition on $\partial Q$. There are two solutions:

- Use a **Neumann homogeneous** boundary condition, i.e. $\partial T(x)/\partial \eta = 0$ for $x \in \partial Q$
- Use a **transparent** boundary condition on $\partial Q$

Transparent boundary conditions corresponds to *state constraints* boundary conditions (delicate implementation with FD).
Outline

1. Introduction
2. Some hints on the implementation of the FD-FMM
3. The semi-Lagrangian FM method
4. Other acceleration techniques
5. Recent extensions of the FMM
   - Characteristics driven Fast Marching method (CFM)
   - BFM method for anisotropic front propagation problems
   - Pursuit-Evasion games
Introduction
Some hints on the implementation of the FD-FMM
The semi-Lagrangian FM method
Other acceleration techniques
Recent extensions of the FMM

The semi-Lagrangian discretization

Let us examine the Fast Marching method based on the semi-Lagrangian discretization. It is useful to change variable via the Kružkov transform

\[ v(x) = 1 - e^{-T(x)}. \]

This allows to treat easily the problem with state constraints (where c vanishes). The SL-scheme is

\[
\begin{cases}
  v_h(x_{ij}) = \min_{a \in B(0,1)} \{ e^{-h} v_h(x_{ij} - hc(x_{ij})a) \} + 1 - e^{-h} & x_{ij} \in \Omega_0^c \cap \mathbb{Z} \\
  v_h(x_{ij}) = 0 & x \in \partial \Omega_0
\end{cases}
\]
We have coupled the FM idea with the SL-scheme. The new method consists in calculating $v_h$ at every node using the SL discretization shown above following the particular ordering of the nodes given by the FM technique.

Problem

Is the FM technique compatible with the new (and bigger) stencil?
We have coupled the FM idea with the SL-scheme. The new method consists in calculating $v_h$ at every node using the SL discretization shown above following the particular ordering of the nodes given by the FM technique.

**Problem**

Is the FM technique compatible with the new (and bigger) stencil?
The sets of nodes for the SL-FM

The $NB_{SL}$ includes MORE nodes (NSEW and the diagonals for every A node).

Let the following assumptions hold true:
- $c \in Lip(\mathbb{R}^n) \cap L^\infty(\mathbb{R}^n)$
- $c(x) \geq 0$

Then, SL-FM method computes an approximate solution of eikonal equation. Moreover, this solution is exactly the same solution of SL classical iterative scheme.

Remark

NO conditions on $c$ and/or $\Delta x$ are required!
On a finite grid this gives the solution after a finite number of operations, the solution coincides with the numerical solution of the global scheme.

How much it costs?

We have to compute the solution at every point at most 8 times and we have to search for the minimum in $NB$ at every iteration. Using a heap-sort method this search costs $O(\ln(N_{nb}))$. The global cost is dominated by $O(N\ln(N))$ ($N$ represents the total number of nodes in the grid).
The convergence of FD-FM and SL-FM comes from the equivalence of the FM approximate solutions with respect to the fixed point solutions (which have been shown to converge to the correct viscosity solution).
Distance from the origin. $\Omega = \{(0, 0)\}$, $c(x, y) \equiv 1$

Exact solution: $T(x, y) = \sqrt{x^2 + y^2}$
**TEST 1. Numerical comparison**

Distance from the origin. $\Omega = \{(0,0)\}$, $c(x, y) \equiv 1$

Exact solution: $T(x, y) = \sqrt{x^2 + y^2}$

<table>
<thead>
<tr>
<th>method</th>
<th>$\Delta x$</th>
<th>$L^\infty$ error</th>
<th>$L^1$ error</th>
<th>CPU time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL (46 its.)</td>
<td>0.08</td>
<td>0.0329</td>
<td>0.3757</td>
<td>8.4</td>
</tr>
<tr>
<td>FM-FD</td>
<td>0.08</td>
<td>0.0875</td>
<td>0.7807</td>
<td>0.5</td>
</tr>
<tr>
<td>FM-SL</td>
<td>0.08</td>
<td>0.0329</td>
<td>0.3757</td>
<td>0.7</td>
</tr>
</tbody>
</table>
TEST 1. Numerical comparison

Distance from the origin. $\Omega = \{(0, 0)\}$, $c(x, y) \equiv 1$

Exact solution: $T(x, y) = \sqrt{x^2 + y^2}$

<table>
<thead>
<tr>
<th>method</th>
<th>$\Delta x$</th>
<th>$L^\infty$ error</th>
<th>$L^1$ error</th>
<th>CPU time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL (46 its.)</td>
<td>0.08</td>
<td>0.0329</td>
<td>0.3757</td>
<td>8.4</td>
</tr>
<tr>
<td>FM-FD</td>
<td>0.08</td>
<td>0.0875</td>
<td>0.7807</td>
<td>0.5</td>
</tr>
<tr>
<td>FM-SL</td>
<td>0.08</td>
<td>0.0329</td>
<td>0.3757</td>
<td>0.7</td>
</tr>
</tbody>
</table>
TEST 1. Numerical comparison

Distance from the origin. \( \Omega = \{(0,0)\}, \quad c(x,y) \equiv 1 \)

Exact solution: \( T(x,y) = \sqrt{x^2 + y^2} \)

<table>
<thead>
<tr>
<th>method</th>
<th>( \Delta x )</th>
<th>( L^\infty ) error</th>
<th>( L^1 ) error</th>
<th>CPU time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL (46 its.)</td>
<td>0.08</td>
<td>0.0329</td>
<td>0.3757</td>
<td>8.4</td>
</tr>
<tr>
<td>FM-FD</td>
<td>0.08</td>
<td><strong>0.0875</strong></td>
<td>0.7807</td>
<td><strong>0.5</strong></td>
</tr>
<tr>
<td>FM-SL</td>
<td>0.08</td>
<td><strong>0.0329</strong></td>
<td>0.3757</td>
<td><strong>0.7</strong></td>
</tr>
</tbody>
</table>
TEST 2. Minimum time problem

Presence of obstacles
TEST 2. Minimum time problem

Presence of obstacles
In the SL method it is rather easy to implement "state constraint" boundary conditions. This allows to deal with obstacles, $O$. In fact, the algorithm look for a minimum

$$v_h(x_{ij}) = \min_{a \in B(0,1)} \left\{ e^{-h} v_h(x_{ij} - hc(x_{ij})a) \right\} + 1 - e^{-h}$$

for $x_{ij} \in \Omega_0^C \cap Z \setminus O$ so it is sufficient to impose on the boundaries $\partial Q$ and $\partial O$ a value $\hat{\nu}$

$$\hat{\nu} > \max_{x \in Q \setminus O} v_h(x)$$
Trasparent boundary conditions

The results are quite accurate
Outline

1. Introduction
2. Some hints on the implementation of the FD-FMM
3. The semi-Lagrangian FM method
4. Other acceleration techniques
5. Recent extensions of the FMM
   - Characteristics driven Fast Marching method (CFM)
   - BFM method for anisotropic front propagation problems
   - Pursuit-Evasion games
Sweeping

It has been observed that for the eikonal equation there is a simple way to speed-up the computations instead of looking for a fixed point iteration and search for the information driven by the characteristics at every node. The sweeping method sweeps the informations along the directions of the axis until the solution does not change.

- The first iteration sweeps to the North every point
- The second iteration sweeps to the West every point
- The third iteration sweeps to the South every point
- The forth iteration sweeps to the Est every point
This algorithm continues until convergence. In fact, for the eikonal equation only 8 iterations are needed, so the cost is very limited.

The sweeping method has been applied to more general Hamilton-Jacobi equations, f.e. to Bellman equations. The Lax-Friedrichs scheme or the Godunov scheme have been used as local rules [Kao-Osher-Tsai (2005)].
Sweeping

Advantages

The methods is rather simple to implement and does not require
dynamic structures
Convergence is obtained in finite number of iterations (for the
eikonal equation)

Disadvantages

It is difficult to determine its complexity It does not converge in a
finite number of iterations for more general equations
The idea is to select a group of points at every iteration and make them advance at the same time [Kim (2001)]. This requires a correction-by-iteration strategy but avoids the search for the minimum value in the Narrow Band. Every iteration cost a bit more than the standard FMM. This algorithm continues until all the nodes have been Accepted.
Group Marching

Advantages

It has as an $O(N)$ complexity
Convergence is obtained in finite number of iterations (for the eikonal equation)

Disadvantages

Specifically designed for the eikonal equation
Difficult to apply for more general equations
Outline

1. Introduction
2. Some hints on the implementation of the FD-FMM
3. The semi-Lagrangian FM method
4. Other acceleration techniques
5. Recent extensions of the FMM

- Characteristics driven Fast Marching method (CFM)
- BFM method for anisotropic front propagation problems
- Pursuit-Evasion games
As in the group marching method we would like to accelerate convergence accepting more nodes at every iteration. The difficulty is to control the algorithm so that we do not loose important informations.

We follow the front propagation for a given (variable) $\Delta t$ and see how many points are reached by the front. We will accept all the nodes that are reached by the front.

The crucial point is to determine the node which has maximum velocity in the NB. This determines the current $\Delta t$. 
CFM: main idea

As in the group marching method we would like to accelerate convergence accepting more nodes at every iteration. The difficulty is to control the algorithm so that we do not loose important informations.

We follow the front propagation for a given (variable) $\Delta t$ and see how many points are reached by the front. We will accept all the nodes that are reached by the front.

The crucial point is to determine the node which has maximum velocity in the NB. This determines the current $\Delta t$. 
CFM: main idea

As in the group marching method we would like to accelerate convergence accepting more nodes at every iteration. The difficulty is to control the algorithm so that we do not loose important informations.

We follow the front propagation for a given (variable) $\Delta t$ and see how many points are reached by the front. We will accept all the nodes that are reached by the front.

The crucial point is to determine the node which has maximum velocity in the NB. This determines the current $\Delta t$. 
The algorithm step-by-step, initialization

1. The nodes belonging to the initial front \( \Gamma_0 \) are located and labeled as accepted. They form the set \( \tilde{\Gamma}_0 \). The value of \( \nu \) of these nodes is set to 0.

2. \( NB \) is defined as the set of the nodes belonging to \( N_{SL}(\tilde{\Gamma}_0) \), external to \( \Gamma_0 \).

3. Set \( t_{i,j}^{loc} = 0 \) for any \( (i,j) \in NB \).

4. The remaining nodes are labeled as far, their value is set to \( \nu = 1 \) (corresponding to \( T = +\infty \)).
**CFM Algorithm**

The algorithm step-by-step, main cycle

1. Compute $c_{\text{max}} = \max\{c_{i,j} : (i,j) \in NB\}$ and set $\Delta t := \Delta x / c_{\text{max}}$.

2. For any $(i,j) \in NB$:
   - Update the local time: $t_{i,j}^{\text{loc}} := t_{i,j}^{\text{loc}} + \Delta t$.
   - If $t_{i,j}^{\text{loc}} \cdot c_{i,j} \geq \Delta x$ then
     1. Compute $T_{i,j}$ by the SL scheme.
     2. Check if $T(i,j)$ is computed using only accepted nodes. If yes, set flag=1, else flag=0.
     3. If flag=1 then
        - Label $(i,j)$ as accepted.
        - Remove $(i,j)$ from $NB$.
        - Define $FN$ as the set of the far neighbors of $(i,j)$. Include $FN$ in $NB$ and set $t_{k,m}^{\text{loc}} = 0$ for any $(k,m) \in FN$.
   3. If not all nodes are accepted go back to 1.
CFM Algorithm, Numerical results. Test 1

MATLAB 7.0, Processor Intel dual core 2x2.80 GHz, 1 GB RAM.
\( \Gamma_0 = (0, 0), \quad c(x, y) \equiv 1, \quad \text{Solution: } T(x, y) = \sqrt{x^2 + y^2} \).

<table>
<thead>
<tr>
<th>method</th>
<th>( \Delta x )</th>
<th>( L^\infty ) error</th>
<th>( L^1 ) error</th>
<th>CPU time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFM</td>
<td>0.08</td>
<td>0.0329</td>
<td>0.3757</td>
<td>0.27</td>
</tr>
<tr>
<td>FM-SL</td>
<td>0.08</td>
<td>0.0329</td>
<td>0.3757</td>
<td>0.58</td>
</tr>
<tr>
<td>SL (46 it)</td>
<td>0.08</td>
<td>0.0329</td>
<td>0.3757</td>
<td>9.7</td>
</tr>
<tr>
<td>FS-SL</td>
<td>0.08</td>
<td>0.0329</td>
<td>0.3757</td>
<td>0.86</td>
</tr>
<tr>
<td>CFM</td>
<td>0.04</td>
<td>0.0526</td>
<td>0.4762</td>
<td>1.14</td>
</tr>
<tr>
<td>FM-SL</td>
<td>0.04</td>
<td>0.0204</td>
<td>0.2340</td>
<td>2.44</td>
</tr>
<tr>
<td>SL (86 it)</td>
<td>0.04</td>
<td>0.0204</td>
<td>0.2340</td>
<td>70.95</td>
</tr>
<tr>
<td>FS-SL</td>
<td>0.04</td>
<td>0.0204</td>
<td>0.2340</td>
<td>3.37</td>
</tr>
<tr>
<td>CFM</td>
<td>0.02</td>
<td>0.0309</td>
<td>0.2834</td>
<td>4.9</td>
</tr>
<tr>
<td>FM-SL</td>
<td>0.02</td>
<td>0.0122</td>
<td>0.1406</td>
<td>10.56</td>
</tr>
<tr>
<td>SL (162 it)</td>
<td>0.02</td>
<td>0.0122</td>
<td>0.1406</td>
<td>530.56</td>
</tr>
<tr>
<td>FS-SL</td>
<td>0.02</td>
<td>0.0122</td>
<td>0.1406</td>
<td>13.33</td>
</tr>
</tbody>
</table>
**CFM Algorithm, Test 2: discontinuous vector field**

\[ \Gamma_0 = (0, 0), \quad c(x, y) = \begin{cases} 0.4 & (x, y) \in [0.5, 1] \times [0.5, 1] \\ 1 & \text{elsewhere} \end{cases} \]

<table>
<thead>
<tr>
<th>method</th>
<th>( \Delta x )</th>
<th>CPU time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFM</td>
<td>0.04</td>
<td>1.12</td>
</tr>
<tr>
<td>FM-SL</td>
<td>0.04</td>
<td>2.39</td>
</tr>
<tr>
<td>CFM</td>
<td>0.02</td>
<td>5</td>
</tr>
<tr>
<td>FM-SL</td>
<td>0.02</td>
<td>10.56</td>
</tr>
</tbody>
</table>
CFM Algorithm. Test 4

\[ \Gamma_0 = (0, 0), \quad c(x, y) = x + 3 \]

- **CFM** method:
  - \( \Delta x \): 0.04
  - CPU time (sec): 1.19
- **FM-SL** method:
  - \( \Delta x \): 0.04
  - CPU time (sec): 2.34
- **CFM** method:
  - \( \Delta x \): 0.02
  - CPU time (sec): 5.20
- **FM-SL** method:
  - \( \Delta x \): 0.02
  - CPU time (sec): 10.44
CFM Algorithm. Test 5

CFM (left) and SL-FM (right)

<table>
<thead>
<tr>
<th>method</th>
<th>$\Delta x$</th>
<th>CPU time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFM</td>
<td>0.04</td>
<td>1.05</td>
</tr>
<tr>
<td>FM-SL</td>
<td>0.04</td>
<td>2.12</td>
</tr>
<tr>
<td>CFM</td>
<td>0.02</td>
<td>5.08</td>
</tr>
<tr>
<td>FM-SL</td>
<td>0.02</td>
<td>9.53</td>
</tr>
</tbody>
</table>
Cristal growth

The corresponding equation is

\[ v(x) = \min_{a \in B(0,1)} \{ c(x, a) a \cdot \nabla v(x) \} + 1 \quad \text{in } Q \setminus \Omega_0 \]

A rather simple case is when we know the Wulff shape so that we can easily adapt the SL-scheme
In general, the problem is more complex. In the anisotropic front propagation problem a value $v$ at a node can depend on values greater than $v$ itself and then the gradient and the characteristic directions are quite different. So the update procedure of FM method is not suitable for its numerical solution nor for differential games.
BFM: main idea

In the BFM method the node in the *narrow band* with the minimum value is not directly accepted BUT it is moved in a buffer. All the nodes in the buffer are recomputed until their value is stabilized.
BFM Algorithm

The algorithm step-by-step, initialization

1. The nodes belonging to the initial front $\Gamma_0$ are located and labeled as *accepted*. They form the set $\tilde{\Gamma}_0$. The value of $v$ of these nodes is set to 0.

2. $NB$ is defined as the set of the nodes belonging to $N_{SL}(\tilde{\Gamma}_0)$, external to $\Gamma_0$.

3. The values at nodes in $NB$ are computed by the SL scheme.

4. The remaining nodes are labeled as *far*, their value is set to $v = 1$ (corresponding to $T = +\infty$).
BFM Algorithm

The algorithm step-by-step, main cycle

1. We search for the node in $NB$ with the minimum value for $v$. That node is removed from $NB$ and labeled as B(uffered).

2. All the B nodes are computed by the SL scheme.

3. We search again for the minimum value of $v$, $v_{min}$. We set $v_{i,j} = v_{min}$ for $(i,j) \in NB$ then we compute the solution in the buffer as in the classical iterative scheme.

4. We remove from B and label as A those nodes whose value is not changed in the previous step.

5. The far nodes in $N_{SL}(A)$ are moved in $NB$ and the nodes in $N_{SL}(A)$ which are not accepted are (re)computed by the SL scheme.
Pursuit-Evasion games with state constraints in $\mathbb{R}^2 \times \mathbb{R}^2$

\[
\begin{align*}
\dot{y}(t) &= f(y(t), a(t), b(t)), \quad t > 0 \\
y(0) &= x
\end{align*}
\]

where

\[
x = (x_P, x_E), \quad y = (y_P, y_E), \quad a \in A, b \in B
\]

\[
f(x, a, b) = f(x_P, x_E, a, b) = \begin{pmatrix} f_P(x_P, a) \\ f_E(x_E, b) \end{pmatrix}, \quad f_P, f_E \in \mathbb{R}^2
\]

\[
x \in \Omega_1, \quad y \in \Omega_2 \quad \text{state constraints}
\]

Target: $\mathcal{T} = \{(x_P, x_E) \in \mathbb{R}^4 : |x_P - x_E| \leq \varepsilon\}, \quad \varepsilon \geq 0$
Pursuit-Evasion games with state constraints in $\mathbb{R}^2 \times \mathbb{R}^2$

\[
\begin{align*}
\dot{y}(t) &= f(y(t), a(t), b(t)), \quad t > 0 \\
y(0) &= x
\end{align*}
\]

where

\[\begin{align*}
x &= (x_P, x_E), \quad y &= (y_P, y_E), \quad a \in A, b \in B \\
f(x, a, b) &= f(x_P, x_E, a, b) = \begin{pmatrix} f_P(x_P, a) \\ f_E(x_E, b) \end{pmatrix}, \quad f_P, f_E \in \mathbb{R}^2
\end{align*}\]

\[x \in \Omega_1, \quad y \in \Omega_2 \quad \text{state constraints}\]

Target: $\mathcal{T} = \{(x_P, x_E) \in \mathbb{R}^4 : |x_P - x_E| \leq \varepsilon\}, \quad \varepsilon \geq 0$
Pursuit-Evasion games with state constraints in $\mathbb{R}^2 \times \mathbb{R}^2$

\[
\begin{align*}
\left\{ \begin{array}{l}
\dot{y}(t) = f(y(t), a(t), b(t)), \quad t > 0 \\
y(0) = x
\end{array} \right.
\]

where

\[
x = (x_P, x_E), \quad y = (y_P, y_E), \quad a \in A, b \in B
\]

\[
f(x, a, b) = f(x_P, x_E, a, b) = \begin{pmatrix} f_P(x_P, a) \\ f_E(x_E, b) \end{pmatrix}, \quad f_P, f_E \in \mathbb{R}^2
\]

\[
x \in \Omega_1, \quad y \in \Omega_2 \quad \text{state constraints}
\]

Target: $T = \{(x_P, x_E) \in \mathbb{R}^4 : |x_P - x_E| \leq \varepsilon\}, \quad \varepsilon \geq 0$
**Isaacs equation**

\[ T(x) := \text{capture time under optimal (non-anticipating) strategies of both players.} \]

The function \( v(x) = 1 - e^{-T(x)} \) is the unique viscosity solution of the **Hamilton-Jacobi-Isaacs equation**

\[
v(x) + \min_{b \in B(x)} \max_{a \in A(x)} \{-f(x, a, b) \cdot \nabla v(x)\} - 1 = 0 \quad x \in \overline{\Omega \setminus T}
\]

complemented with Dirichlet boundary condition \( v(x) = 0 \) on \( \partial T \) [Koike, 1995].
We build a regular triangulation of $\overline{\Omega}$ denoting by:

$X$ the set of its nodes $x_i$, $i = 1, \ldots, N$

$S$ the set of simplices $S_j$, $j = 1, \ldots, L$

$k := \max_j \{diam(S_j)\}$

\[
\begin{align*}
    v^k_h(x_i) &= \max_{b \in B_h(x_i)} \min_{a \in A_h(x_i)} \{ \beta v^k_h(x_i + hf(x_i, a, b)) \} + 1 - \beta & x_i \in (X \setminus \mathcal{T}) \\
    v^k_h(x_i) &= 0 & x_i \in \mathcal{T} \cap X \\
    v^k_h(x) &= \sum_j \lambda_j(x) v^k_h(x_j), & 0 \leq \lambda_j(x) \leq 1, \quad \sum_j \lambda_j(x) = 1 & x \in \overline{\Omega}
\end{align*}
\]

$\beta = e^{-h}$. 

M. Falcone

Fast Marching Methods for Front Propagation
BFM Algorithm. Test 1: anisotropic front propagation

\[ \Gamma_0 = (0, 0), \quad c(x, y, a) = (1 + (\lambda a_1 + \mu a_2)^2)^{-\frac{1}{2}}, \quad \lambda = \mu = 5 \]

Exact solution:

\[ T(x, y) = \sqrt{(1 + \lambda^2)x^2 + (1 + \mu^2)y^2 + 2\lambda\mu xy} \]

Exact solution (left) and FM-SL (right)
BFM Algorithm. Test 1: anisotropic front propagation

SL iterative (left) and BFM (right)
BFM Algorithm. Test 1: anisotropic front propagation

The $L^1$ error is computed with respect to the solution of the iterative algorithm rather than the exact solution. 16 controls, $tol = 10^{-6}$.

<table>
<thead>
<tr>
<th>method</th>
<th>$\Delta x$</th>
<th>$L^1$ error</th>
<th>CPU time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL (45 it)</td>
<td>0.08</td>
<td>-</td>
<td>75</td>
</tr>
<tr>
<td>BFM</td>
<td>0.08</td>
<td>0.018</td>
<td>30.45</td>
</tr>
<tr>
<td>FM-SL</td>
<td>0.08</td>
<td>1.68</td>
<td>8</td>
</tr>
<tr>
<td>SL (84 it)</td>
<td>0.04</td>
<td>-</td>
<td>551</td>
</tr>
<tr>
<td>BFM</td>
<td>0.04</td>
<td>0.0046</td>
<td>128</td>
</tr>
<tr>
<td>FM-SL</td>
<td>0.04</td>
<td>1.54</td>
<td>31.8</td>
</tr>
</tbody>
</table>
BFM Algorithm. Test 2: differential games with state constraints

\[ f(x, y, a, b) = (v_P a, v_E b), \quad a \in [-1, 1], \quad b \in [-1, 1] \]

Exact solution (left) and FM-SL (right)
BFM Algorithm. Test 2: differential games with state constraints

SL iterative (left) and BFM (right)
BFM Algorithm. Test 2: differential games with state constraints

The $L^1$ error is computed with respect to the solution of the iterative algorithm rather than the exact solution. 2x2 controls, $tol = 10^{-6}$.

<table>
<thead>
<tr>
<th>method</th>
<th>$\Delta x$</th>
<th>$L^1$ error</th>
<th>CPU time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL (70 it)</td>
<td>0.08</td>
<td>-</td>
<td>21.5</td>
</tr>
<tr>
<td>BFM</td>
<td>0.08</td>
<td>0.002</td>
<td>9.16</td>
</tr>
<tr>
<td>FM-SL</td>
<td>0.08</td>
<td>0.23</td>
<td>1.59</td>
</tr>
<tr>
<td>SL (130 it)</td>
<td>0.04</td>
<td>-</td>
<td>157</td>
</tr>
<tr>
<td>BFM</td>
<td>0.04</td>
<td>0.002</td>
<td>60</td>
</tr>
<tr>
<td>FM-SL</td>
<td>0.04</td>
<td>0.42</td>
<td>6.17</td>
</tr>
</tbody>
</table>
Another extension

Dislocation dynamics

Non-local unsigned velocity of the front \( c(\Gamma_t) \geq 0 \)

In this case the difficulty is due to the non local term which is obtained by convolution (Carlini, Forcadel, Monneau [2007, in progress]).
Some references


