An Efficient Data Structure to Solve Front Propagation Problems and its Application to the Climbing Problem

O. Bokanowski, E. Cristiani, H. Zidani

ENSTA (UMA), Paris, France

May 20, 2008

Outline

Outline

Purposes

We want to solve front propagation problems in dimension $d \geq 3$ reducing as mach as possible CPU and memory consuming.

These kind of problems appear in physics (d=1,2,3) and in optimal control problems ($d \geq 1$). In the latter case the dimension d corresponds to the number of the state variables involved in the problem.

In both cases the problems can be modeled by a first-order non-linear PDE. Computation of its solution is in general time and memory consuming.

The HJB equation

$\mathcal{T}$ = initial front (*final target*), f = dynamics,
$T(x)$ = first time the front reach the point x (*minimal time to reach the target*)

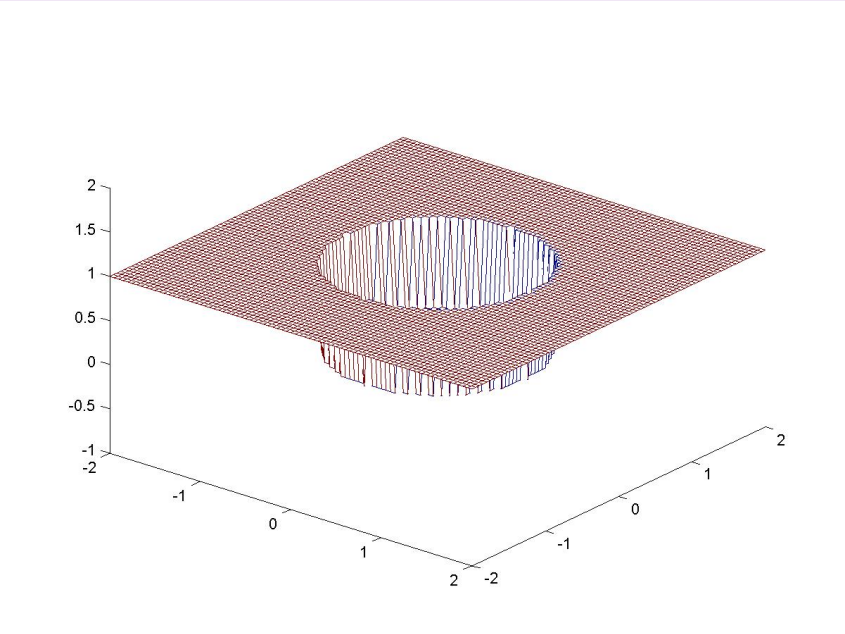
$$\begin{cases} \theta_t(t,x) + \max\limits_{\substack{a\in A,\\ \lambda\in\Lambda(x)}} \{-\lambda f(x,a)\cdot\theta_x(t,x)\} = 0 & x\in\mathbb{R}^d, t\in[0,T] \\ \theta(0,x) = \mathcal{X}_{\mathcal{T}}(x) & x\in\mathbb{R}^d \end{cases}$$

$$\Lambda(x) := \begin{cases} \{1\} & x\notin\mathcal{T} \\ [0,1] & x\in\mathcal{T} \end{cases}$$

Note that $\theta\in\{0,1\}$ and $T(x) = \inf\{t : \theta(t,x) = 0\}$

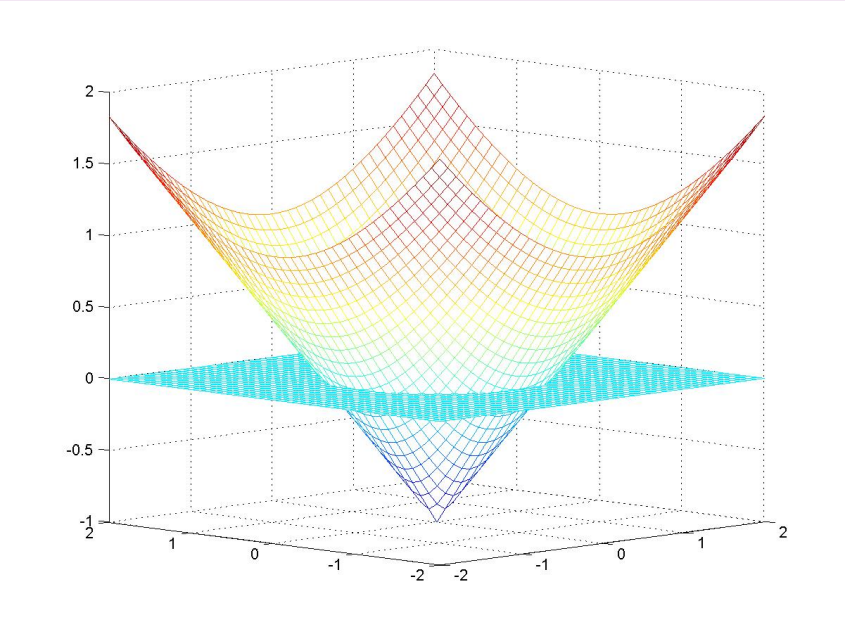
The function θ

$\theta(t,x) \in \{0,1\}$



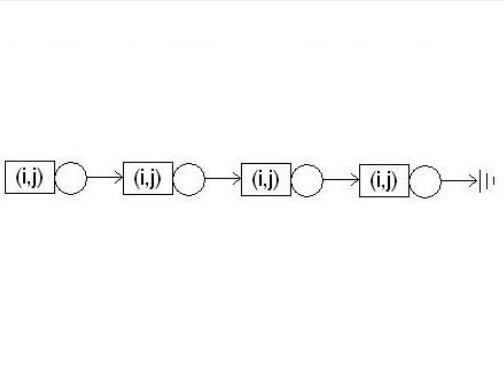
Starting point

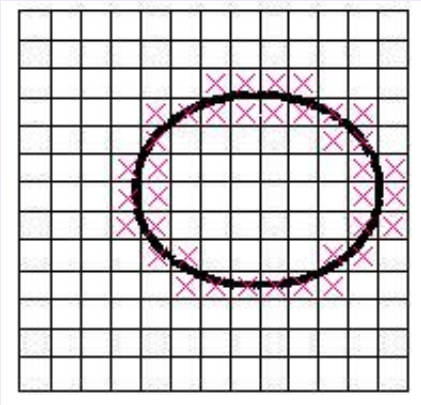
The starting point is the *Narrow Band* method (Sethian et al., 1995) for front propagation problems in the framework of level set methods. Here the goal is to reduce the numerical effort focusing the computation only around the front.



Classical implementation

To avoid computation on all the grid at each time step we can store a dynamic list containing the node of the *Narrow Band*.





1. Fast, computation is located to a small number of nodes
2. Difficulties in managing the NB
3. Memory consuming, especially if $d \geq 3$

Main goal

Memory consuming is crucial when the dimension d is greater than 3. If we use a fine grid or the front propagation problem is just a part of the computation, 2 GB RAM can be not sufficient.

Main goal

We want to focus the computation around the front and stock only the nodes involved in computation (to save memory), preserving the efficiency of the scheme.

Main difficulty

How to find the values of the neighbors during computation? The list is not ordered and it does not include all nodes!

Previous work

Sethian, Adalsteinsson, 1995: dynamic list

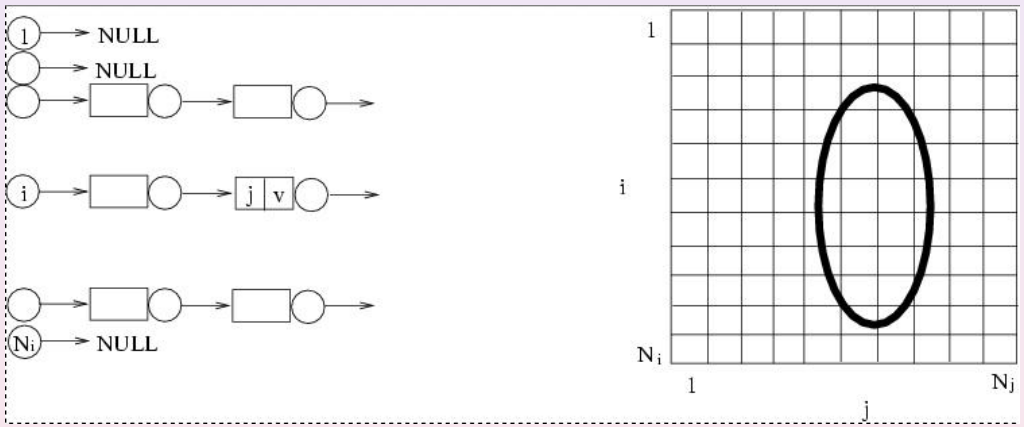
Peyré, 2007: hash table

Bokanowski, Megdich, Zidani, 2005-6-7: usage of a sparse matrix and a quad-tree approach. CPU time not linear with respect the number of nodes in the front.

Bokanowski, Cristiani, Zidani, 2007: special ordered dynamic list, abandoned.

First method proposed: SPARSE SEMI-DYNAMIC

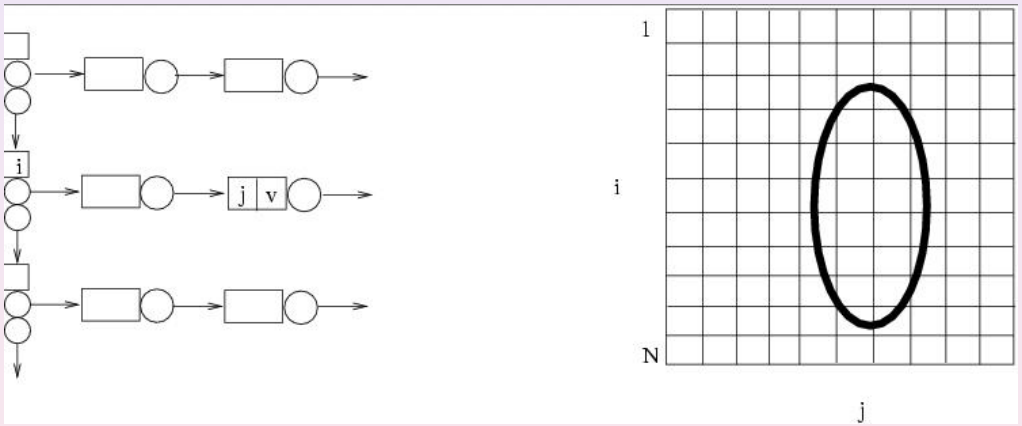
We construct the following semi-dynamic structure which contains the nodes of the NB.



We stock only nodes we are interested for and we have a direct access to each line of the original matrix. The vector of pointers is visited by a `for` cycle (nice for -O3 option).

Second method proposed: SPARSE DYNAMIC

We stock only pointers which does not point to NULL.



In order to maintain the direct access to each line of the original matrix we store a pointer at the beginning of the previous line we are visiting. HARDER TO BE IMPLEMENTED.

Extension to the dimension d

In dimension d the vector of pointers is substituted by a $(d-1)$-dimensional matrix of pointers.

The gain in memory requirements is very large. Using the Ultra Bee scheme we have

$$4 \times N^d \times size(double) \quad \longrightarrow \quad N^{d-1} \times size(pointer)$$

Ultra Bee scheme (Depres, Lagoutiere (01), B., Z. (07))

1D linear advection

$$\begin{cases} v_t + f(x)v_x = 0 & x \in \mathbb{R}, t \in [0, T] \\ v(0, x) = v_0(x) & x \in \mathbb{R}^n \end{cases}$$

$$V_i^{n+1} = V_i^n - \frac{\Delta t}{\Delta x} f(x_i)(V_{i+\frac{1}{2}}^{n,L} - V_{i-\frac{1}{2}}^{n,R})$$

$$V_{i+\frac{1}{2}}^{n,L} := \min(\max(V_{i+1}^n, b_i^+), B_i^+)$$

$$\text{If } f(x_i) > 0 \begin{cases} b_i^+ := \max(V_i^n, V_{i-1}^n) + \frac{\Delta x}{\Delta t\ f(x_i)}(V_i^n - \max(V_i^n, V_{i-1}^n)) \\ B_i^+ := \min(V_i^n, V_{i-1}^n) + \frac{\Delta x}{\Delta t\ f(x_i)}(V_i^n - \min(V_i^n, V_{i-1}^n)) \end{cases}$$

Ultra-Bee scheme /2

d-dimension linear advection

extension via the alternate direction method

HJB equations

$$V_i^{n+1} = \min_{\substack{k=1,\dots,N_\alpha \\ \lambda(x_i)}} (V_i^{n+1,UB}(\alpha_k, \lambda))$$

UB scheme has very good anti-diffusive properties

Managing the data structure

At each time step,

1. Enlarge the NB including all the neighbors
2. Make computation via the UB scheme
3. Removing nodes from the NB

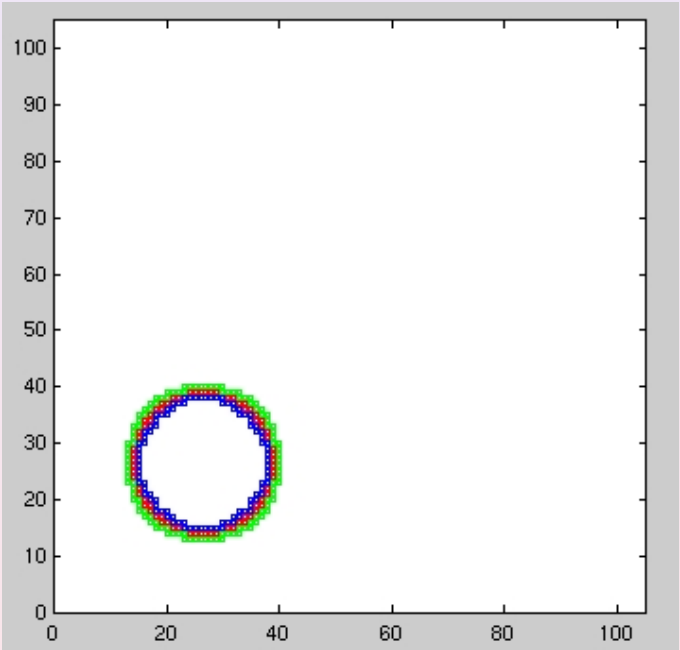
Outline

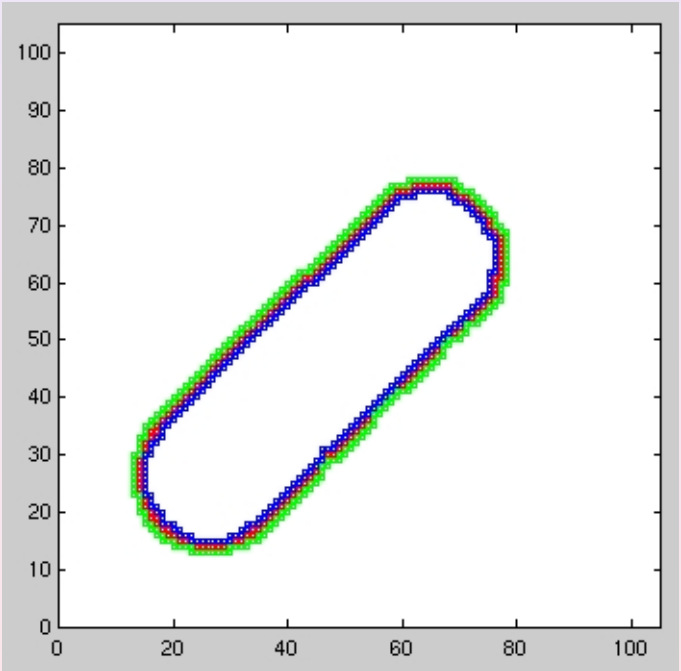
Numerical tests for $d = 2,\ 3,\ 4$

We tested and compared our methods on the *minimum time problem*

Domain$= [-2, 2]^d$
$\mathcal{T} = B_d(0, \frac{1}{2})$
$f = (-1, -1, \ldots, -1)$
$T = 0.5$
CFL$=0.9$
C++ on LINUX, optimized compilation with -O3 option

Results for 2D tests





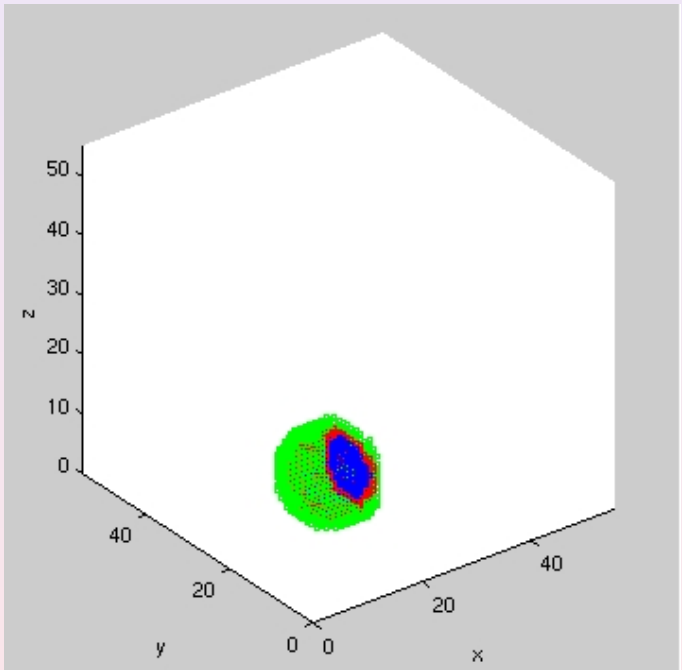
Tests in 2D

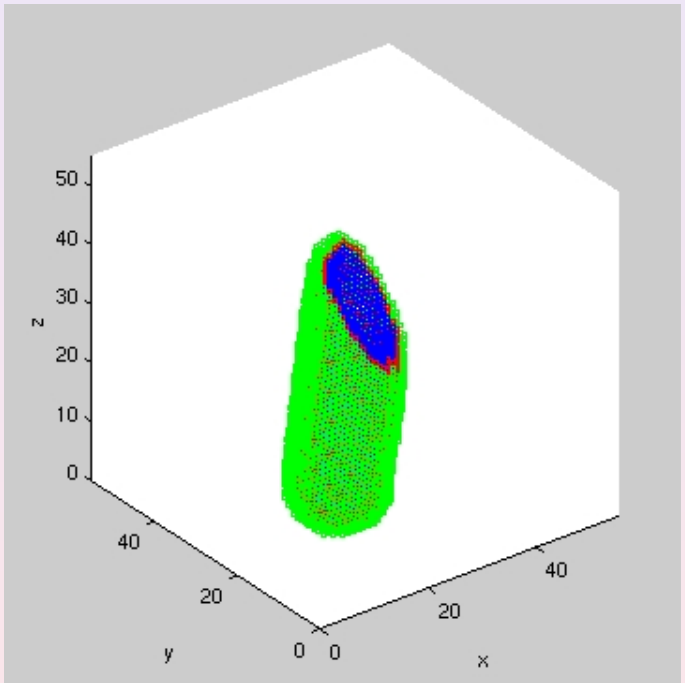
PC PIV 256MB RAM 512KB cache

nodes	FULL	SP SEMI-DYN	SP DYN	time steps	nodes in NB
50^2	0	0	0	7	132
100^2	0.04	0.02	0.01	14	272
200^2	0.35	0.06	0.05	28	532
400^2	2.86	0.24	0.21	56	1068
800^2	23.94	0.94	0.87	112	2124

nodes	FULL	SP SEMI-DYN	SP DYN	time steps	nodes in NB
4x	8.4x	3.9x	4.1x	2x	1.9x

Results for 3D tests





Tests in 3D

PC PIV 256MB RAM and AMD Opteron 8GB RAM 1MB cache

nodes	FULL 256MB	FULL 8GB	SP SEMI-DYN	SP DYN	ts	NB
25^3	0.02	0.01	0.02	0.02	4	432
50^3	0.38	0.24	*0.18	0.18	7	2016
100^3	6.10	4.39	1.51	1.85	14	8016
200^3	out of memory	64.29	14.41	22.55	28	31416
400^3	out of memory	1175.00	146.78	343.25	56	125968

* 0.6 secs if $\mathcal{T} = B_n(0,1)$.

nodes	FULL 256MB	FULL 8GB	SP SEMI-DYN	SP DYN	ts	NB
8x	∞x	18.3x	10.2x	15.2x	2x	4x

Tests in 4D

PC PIV 256MB RAM

total nodes	SP SEMI-DYN	time steps	nodes in NB
25^4	0.49	4	3024
50^4	*6.39	7	23552
100^4	123.97	14	179472

* 7.37 secs for 2 controls
* 9.04 secs for 4 controls.

total nodes	SP SEMI-DYN	time steps	nodes in NB
16x	19.4x	2x	7.6x

Outline

Fast Marching method

We want to apply the new data structure to solve the eikonal equation

$$\begin{cases} c(x)|\nabla T(x)| = 1, & x \in \mathbb{R}^d \backslash \mathcal{T} \\ T(x) = 0, & x \in \partial\mathcal{T} \end{cases}$$

using the Fast Marching method (Sethian, 1996) based on finite difference discretization.

Some modification

In this case we can not recover the value of a node if it not stocked, so the data structure contains the node around the front but also some *Accepted* nodes.

Removing nodes of the data structure is not done at every step because only few nodes should be removed.

The Sethian's method already focus computation around the front, so our method can not be expected to be faster than the classical one. The advantage is just in memory requirements.

Results

Intel dual core, 2 GB RAM

nodes	2D full	2D sparse
500^2	0.1	0.3
1000^2	0.4	1.4
2000^2	1.8	5.9
4000^2	8.2	25.9
8000^2	41.6	139.3
16000^2	out of mem	802.1

nodes	3D full	3D sparse
50^3	0.08	0.3
100^3	0.9	3.8
200^3	10.3	53.4
400^3	104.8	541.0
600^3	out of mem	2032.7

Outline

minimum time problem

Controlled dynamical system

$$\begin{cases} \dot{y} = f(y, \alpha) & t > 0 \\ y(0) = x & x \in \mathbb{R}^n \end{cases}$$

Goal

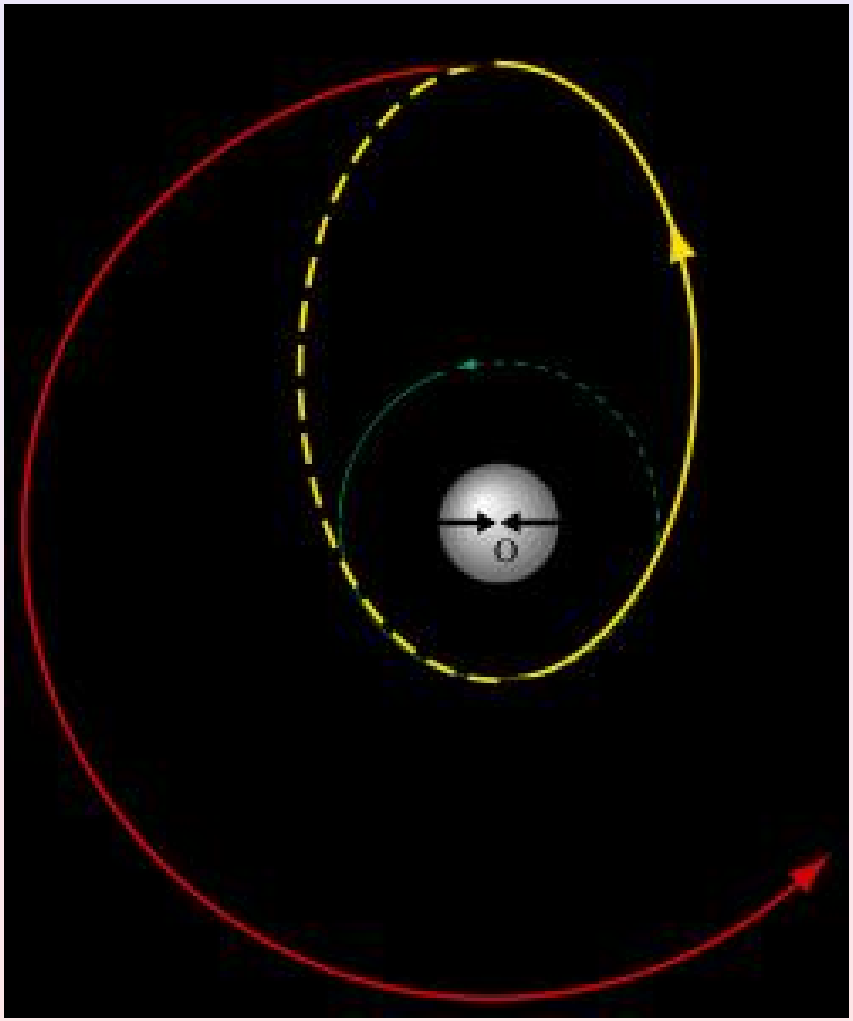
find $\alpha(t) \in \mathcal{A}$ such that the corresponding solution of the system $y_{x,\alpha}(t)$ hits a given target $\mathcal{T} \in \mathbb{R}^n$ in minimal time.

Solution

Once solved the associated HJB equations we can recover the optimal control in feedback form and the optimal trajectory.

Ariane V

GTO and GEO



The climbing problem - 1

GOAL

Maximize the charge utile *CU* carried in geostationary orbit by the european rocket ARIANE V.

We deal with 7 variables, the position of the rocket in the 3D space, its velocity and its mass.
Note that the variation of the mass is very big so it can not be considered as a constant during the experiment.

The forces acting on the rocket are GRAVITY (due to the earth), DRAG (due to the air), THRUST (due to the engine) and CORIOLIS (due to the rotation of the earth).

The climbing problem - 2

Fortunately, we can reduce the number of variables using some symmetries of the problem and considering that the starting point of the rocket is on the equator (this assumption is quite true for ARIANE V).

If the engine is always on, maximixe CU corresponds to minimize the arrival time on target, so the whole problem can be seen as a minimum time problem and then solved by HJB approach.

The related equation

New variables:

r=altitude

v=modulus of the velocity

γ=angle between the direction earth-rocket and the direction of the rocket's velocity.

Control:

α=angle between the thrust direction and the direction of the rocket's velocity.

$$\begin{cases} \dot{r} = v\cos\gamma \\ \dot{v} = -\frac{GM_E}{r^2}\cos\gamma - \frac{F_D(r,v)}{m} + \frac{F_T}{m}\cos\alpha + \Omega^2 r\cos\gamma \\ \dot{\gamma} = \sin\gamma\left(\frac{GM_E}{r^2 v} - \frac{v}{r}\right) - \frac{F_T}{vm}\sin\alpha - \Omega^2\frac{r}{v}\sin\gamma - 2\Omega \\ \dot{m} = -b \end{cases}$$

Other difficulties

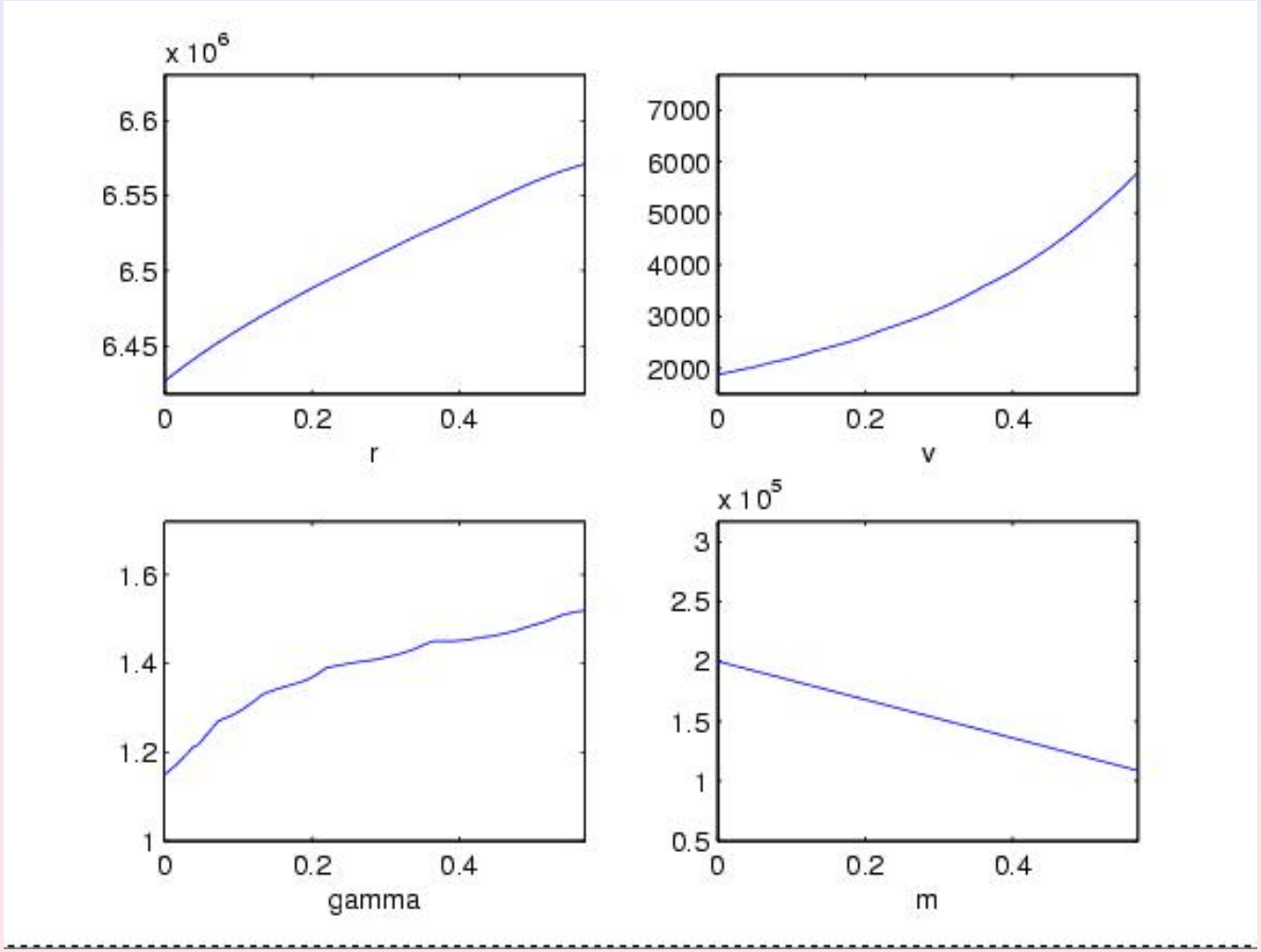
- F_D and F_T are given by tables of measured values.
- The control α is constrained by technical limitations and by a maximal admissible thermal flux.
- The thrust force F_T (and the mass consuming b) is not constant but varies in function of the phase:
  phase 1: main engine + 2 boosters.
  phase 2: main engine.
  phase 3: low-power engine.
- In the changes of phases, the empty envelopes are lost so we have a (not negligible) discontinuity in the rocket's mass.

A first try

We solved the 4D problem without constraints due to the thermal flux and not considering the change of phase, but we used real data given by CNES's tables and documents.

- The problem seems to be very ill-posed (the CFL conditions requires a huge number of time steps).
- Difficulties to reconstruct optimal trajectory because it is very sensible to initial data, number of discrete controls, target. Sometimes the trajectory passes near the border of the reachable set.
- In $\mathbb{R}^4$ it is hard to "see" the value function, find code's bugs and propose solutions.

A first result



A new approach

A new approach consists in *integrating the mass apart*, reversing time's flow to have a corrispondence with the front propagation problem.
In this way the mass is no longer a state variable, so the problem is set in $\mathbb{R}^3$ and we can deal with change of phases and discontinuity of the mass.
We find the minimal mass to transfer a given *CU*.

BUT..
the optimal trajectory is no longer in feedback form w.r.t. the mass.
(if, at some moment, the measured mass of the rocket is not equal to the precomputed optimal mass for that moment, we are no longer able to give the optimal control to reach the target. Otherwise, if the mass is exact but r, v or γ are not, we are able to compute the optimal control in real time.)

Outline

A test for a simple (2+1)D problem

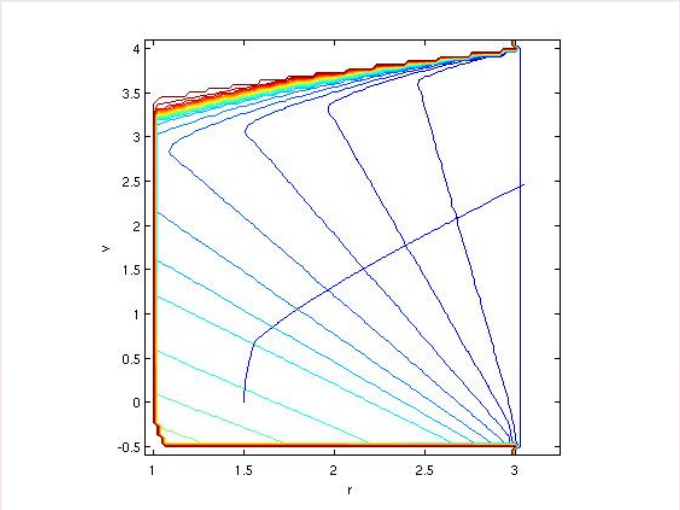
$$\begin{cases} \begin{cases} \dot{r} = v \\ \dot{v} = \frac{F_T}{m} \end{cases} \\ \dot{m} = -b \end{cases}$$

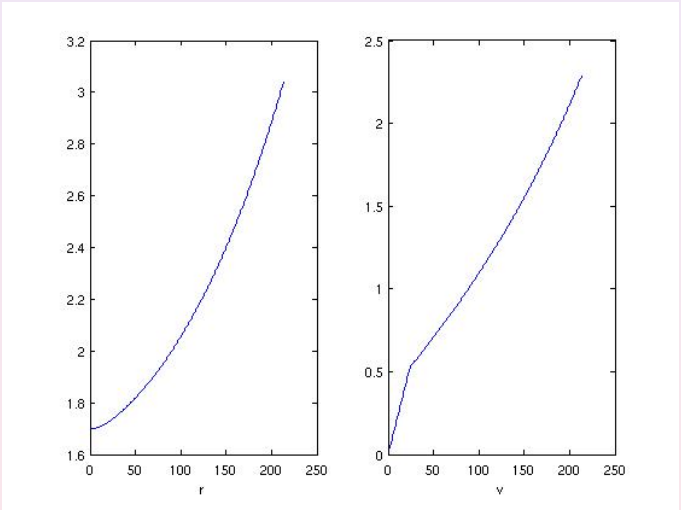
NB1: The dynamics is not controlled here.

NB2: We consider two phases for F_T and b.

Results for the simple (2+1)D problem - large target

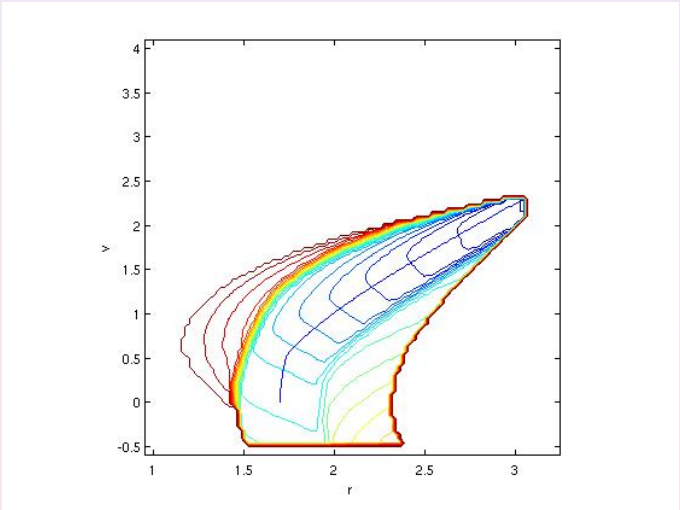
Target: $\{(r, v) : r > 3\}$

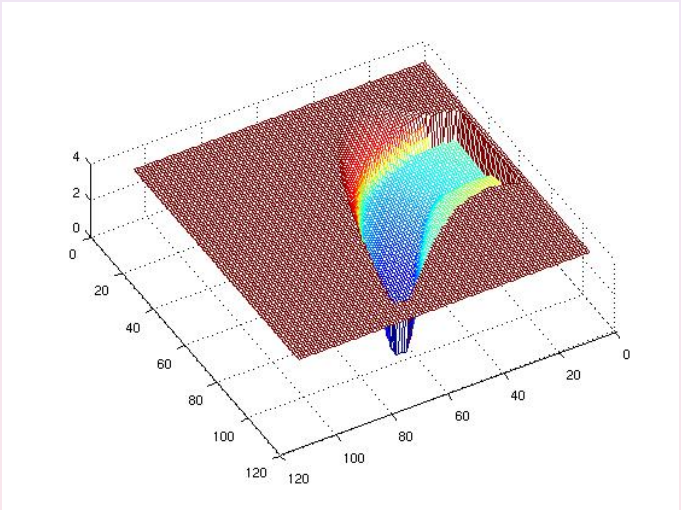




Results for the simple (2+1)D problem - small target

Target: $\{(r, v) \ : \ r > 3, \ 2.2 < v < 2.4\}$



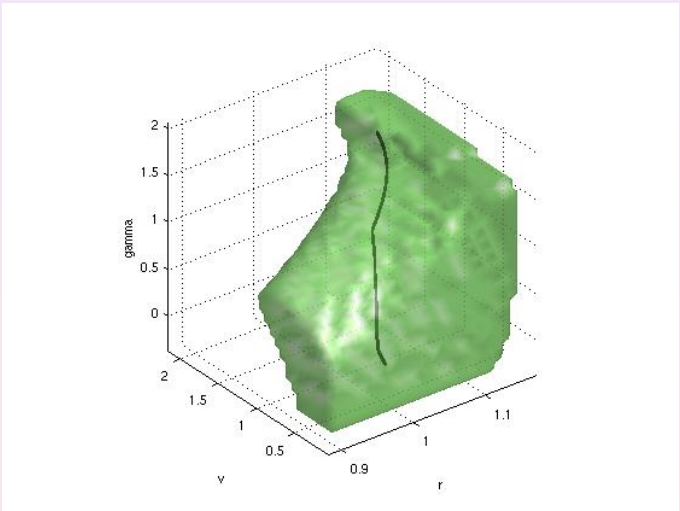


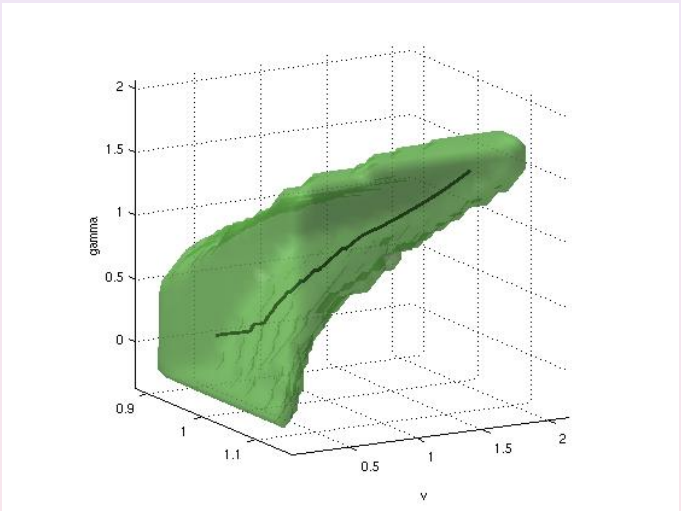
A test for the simplified CNES problem

$$\begin{cases} \begin{cases} \dot{r} = v\cos\gamma \\ \dot{v} = -\frac{g_0}{r^2}\cos\gamma + \frac{F_T}{m}\cos\alpha \\ \dot{\gamma} = \sin\gamma\left(\frac{g_0}{r^2 v}\right) - \frac{F_T}{vm}\sin\alpha \end{cases} \\ \dot{m} = -b \end{cases}$$

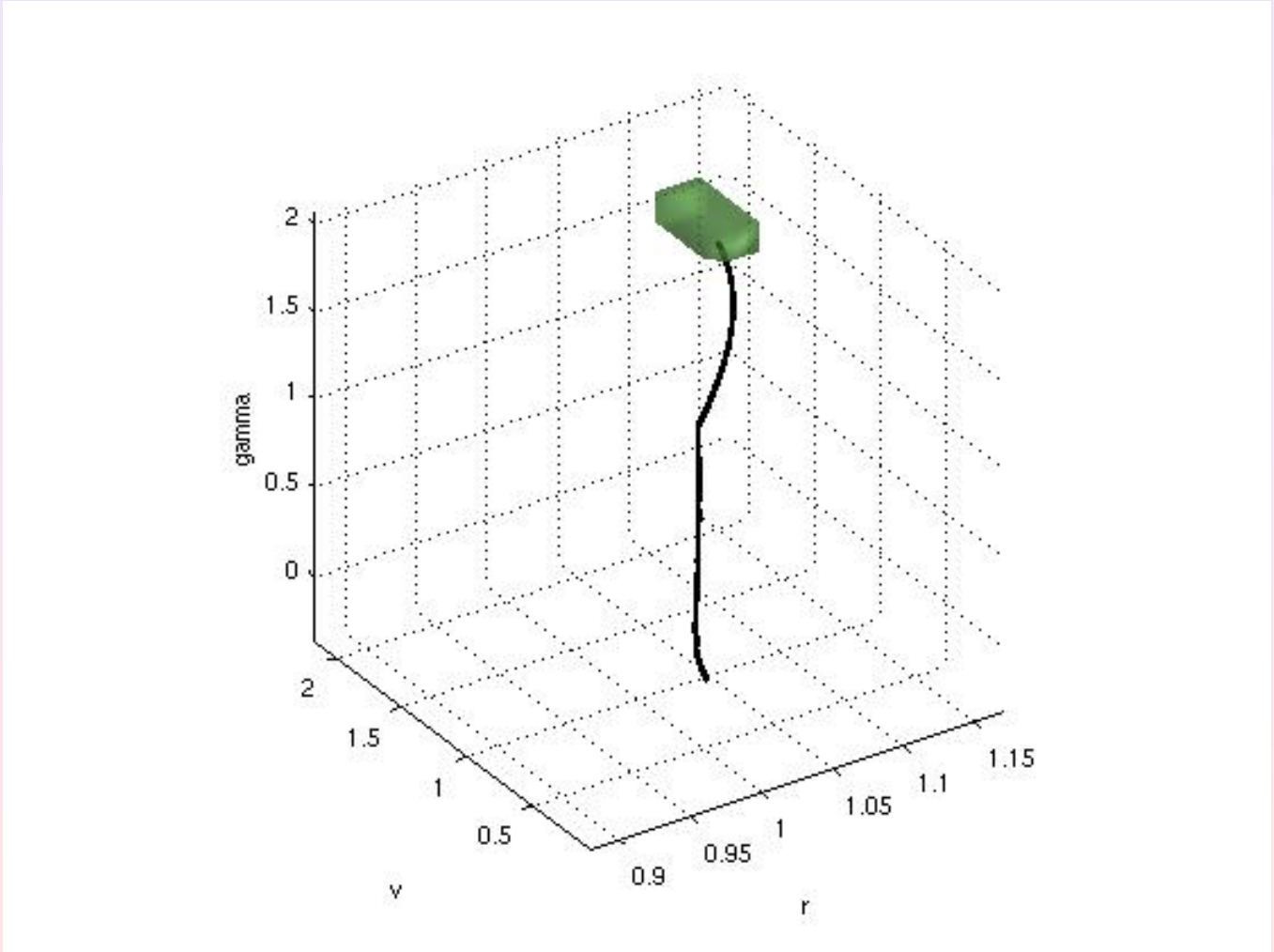
NB: No real data.

Results for the simplified CNES problem - 1

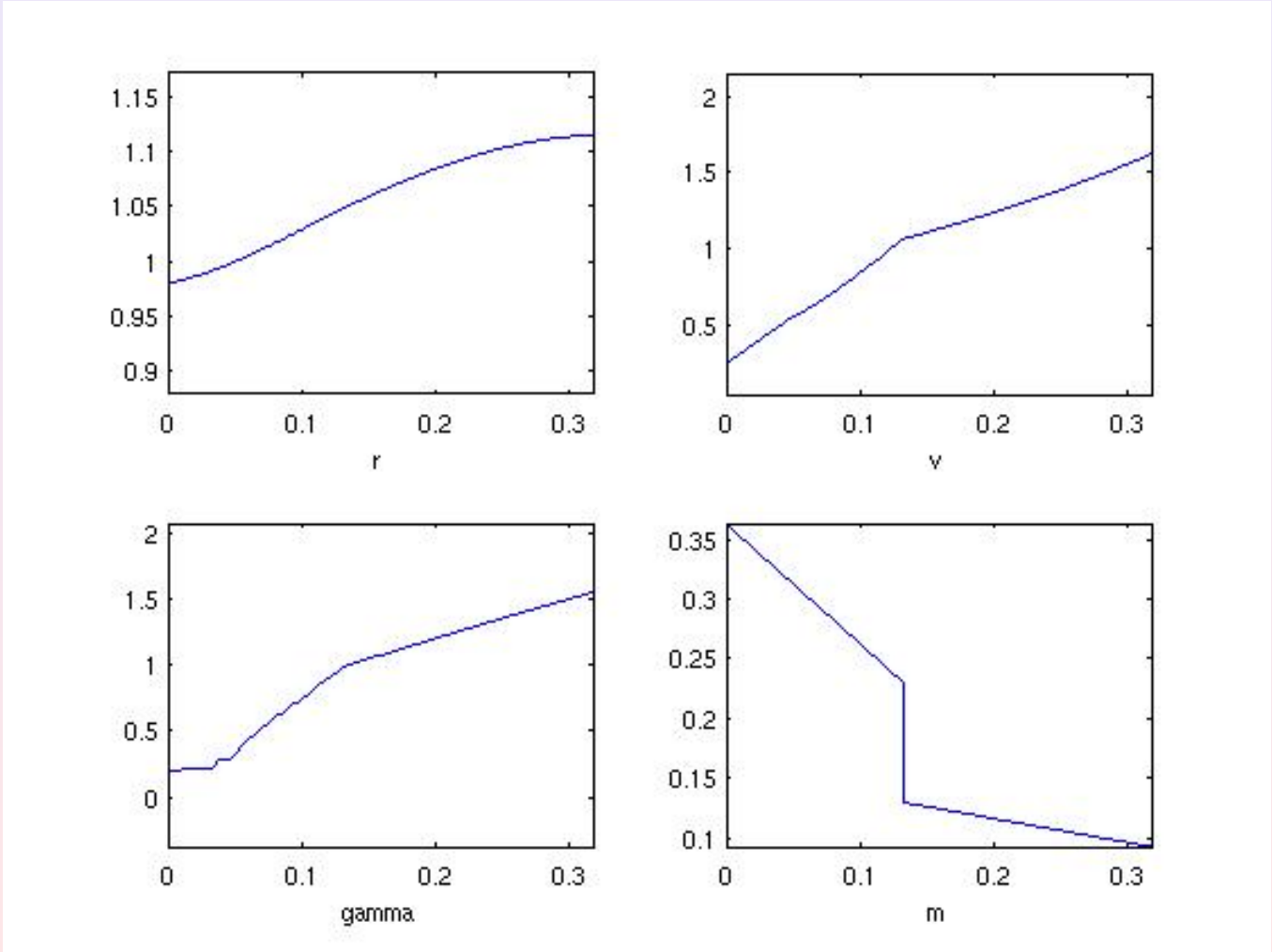




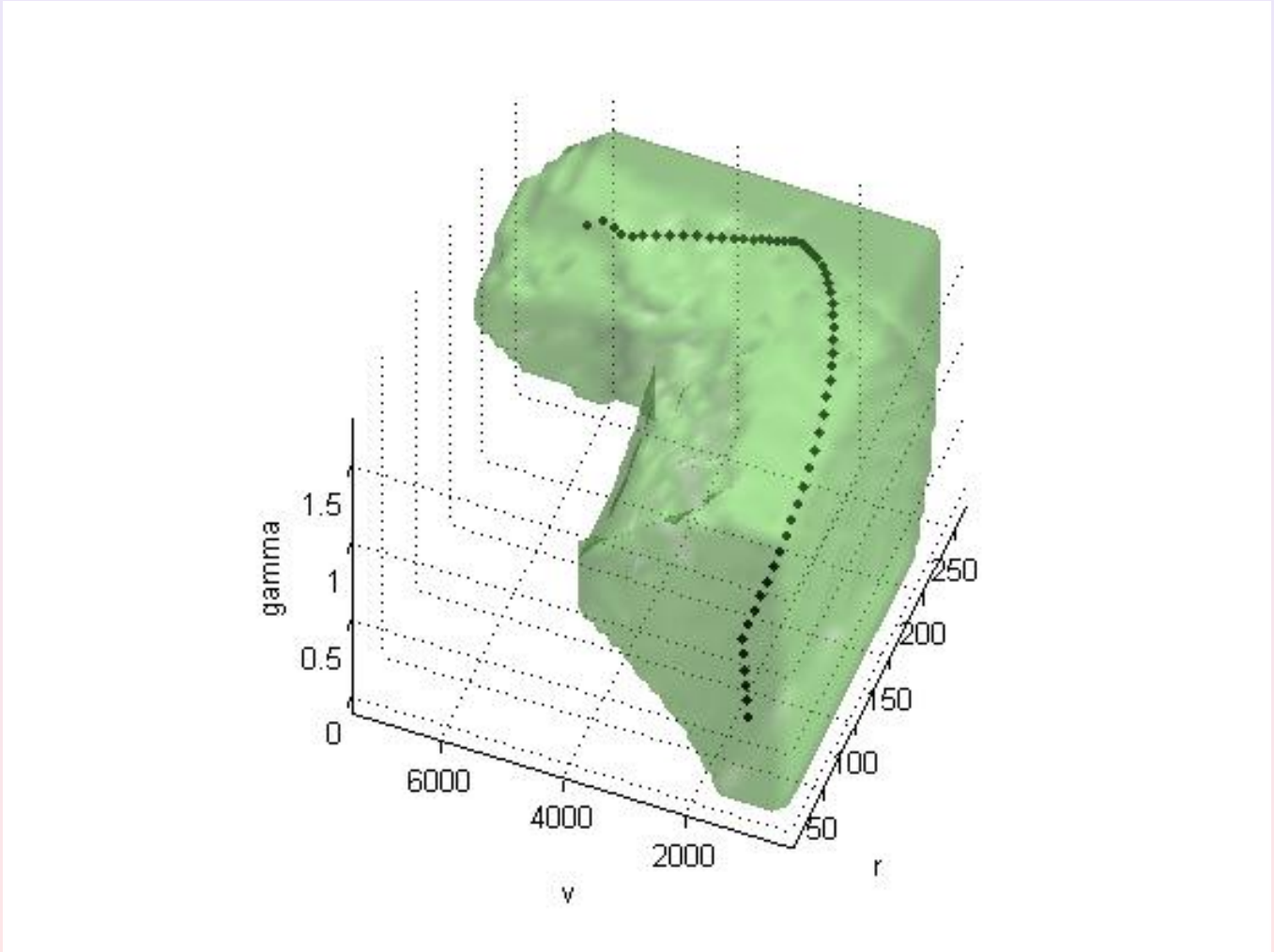
Results for the simplified CNES problem - 2



Results for the simplified CNES problem - 3



CNES problem, real data - 1



CNES problem, real data - 2