Methods and Algorithms for the Minimization of the Energy Consumed by an Electrical Vehicle

Riadh Omheni

Faculty of Sciences and Technology, University of Limoges
ENSEEIHT-IRIT, Toulouse

Department of Mathematics and Computer Science
M2 Research ACSYON

Supervisor: **Frédéric MESSINE**

Methods and Algorithms for the Minimization of the Energy Consumed by an Electrical Vehicle

Riadh OMHENI
<riadh.omheni@gmail.com>

Limoges, July 7, 2011

Acknowledgements

I would like to thank my supervisor, Professor Frédéric Messine, for his continual support, guidance, assistance, and encouragement throughout the internship and the writing of this report.

I would also like to thank Abdelkader Merakeb for his assistance at the beginning of this internship. Thank you to all APO team members who welcomed me in the IRIT-ENSEEIHT.

I will be always grateful to all the teaching staff of the Faculty of Sciences and Technology of Limoges involved in the formation of Master ACSYON for delivering a high quality program for us. In particular, I would like to thank Professors Samir Adly and Ihsen Yengui for providing me the opportunity to pursue my studies by this master.

Last, but not at all least, I would like to thank all my family. I am grateful to my parents for their support throughout my study.

Contents

List of Figures

List of Tables

Introduction

I have done my internship in the APO team (Parallel Algorithm and Optimization) of IRIT (Institut of Research in Informatic of Toulouse) under the supervision of Frédéric Messine, Associate Professor in INPT-ENSEEIHT.

Increasing concerns about environmental issues, such as global warming and greenhouse gas emissions, as well as the predicted scarcity of oil supplies (and geopolitical issues related to oil suppliers) have made energy efficiency and reduced emissions a primary selling point for automobiles, and a concern for many governments. Because of this, electrical vehicles become extremely popular (even though not very widespread) and represent an icon of "good" technology. Indeed, electric motors are more efficient than internal combustion engines in converting stored energy into driving a vehicle, and electric drive vehicles do not consume energy while at rest or coasting, and some of the energy lost when braking is captured and reused through regenerative braking, which captures as much as one fifth of the energy normally lost during braking. Typically, conventional gasoline engines effectively use only 15% of the fuel energy content to move the vehicle or to power accessories, and diesel engines can reach on-board efficiencies of 20%, while electric drive vehicles have on-board efficiency of around 80%.

In recent years, the electrical vehicle energy management has been an important subject and a lot of research has been carried out in this field. However, satisfactory solutions or evident answers for this kind of research do not exist up to now. Indeed, this problem is modeled by a Bang-Bang optimal control problem. The presence of a constraint on a state variable and the Bang-Bang structure of the control make this problem very hard to solve; i.e., the classical optimal control techniques including indirect methods based on Pontryaguine's maximum principle and direct methods based on discretization cannot lead to satisfactory results.

In order to solve efficiently this problem, an original methodology was developed by Abdelkader Merakeb and Frédéric Messine [3,4]. They try to reformulate the initial problem

following the construction of a current regulator technique to obtain a global optimization problem which is first discretized and then solved using a Branch and Bound algorithm.

In this internship, we focus on the problem of minimization of the energy consumption of an electrical vehicle achievable on a given driving cycle. Our work is divided into 3 chapters. Chapter 1 presents a detailed description of the method developed in [3,4] to solve the electrical energy management problem. Chapter 2 is devoted to describe the software architecture of our Fortran 90 code. The benefits of this approach are discussed on some examples. In Chapter 3, we extend the original method to study some extensions. We begin with proposing new heuristics to approach the exact bounds used in the Branch and Bound algorithm. This is followed by studying problems of long travels and of slopes management.

Chapter 1

A Branch and Bound method for minimizing the energy consumption of an electrical vehicle

1.1 Introduction

An electrical vehicle uses an electrical energy source for its displacement. This energy source can be reversible, in the sense that it can be recovered. The problem of energy management of electrical vehicles can be expressed as an optimal control problem which is hard to solve. The main classical approaches have been studied but they did not provide satisfactory solutions. In fact, despite its accuracy and precision, the implementation of indirect methods using shooting methods may deal with some difficulties. Indeed, it transforms the initial problem by solving a nonlinear system. In this case, the numerical solution will be extremely sensitive to the choice of the initial point. Note that the presence of a constraint on a state variable increases the complexity of its use. On the other hand, the use of direct methods based on partial or total discretizations lead to a large scale optimization problem. It should be noted also that these methods are less suitable for certain special cases, including problems with a Bang-Bang structure yielding a large number of switches (as in our case).

An innovative way to solve this problem was developed by Abdelkader Merakeb, in his thesis under the supervising of Frédéric Messine [3,4]. This new methodology is based on a discretization technique associated with a Branch and Bound algorithm.

In this chapter, we recall this new methodology of resolution. We begin with a simple presentation of the studied problem and its modeling. This is followed by giving the

method to approximate the optimal control problem as a global optimization problem by considering a system of current regulator. Finally, we present the method developed in [3,4] to solve, using a Branch and Bound algorithm, the electrical vehicle energy management problem during an imposed time and displacement.

1.2 Model of electrical vehicle

The total electrical energy consummated during the displacement on the cycle of time $[0, t_f]$ (where t_f denotes the final time) is given by the following formula:

$$E(t_f, i_m, u) = \int_0^{t_f} (u(t) i_m(t) V_{alim} + R_{bat} u^2(t) i_m^2(t)) dt \tag{1.1}$$

The quadratic term reflects the losses due to the internal resistance of the battery.
The electrical vehicle energy management can be reformulated as an optimal control problem with a Bang–Bang structure:

$$\begin{cases} \min_{im(t),\Omega(t),pos(t),u(t)} \quad E(t_f, i_m, u) \\ s.t. \\ \quad \dot{i}_m(t) = \frac{u(t)V_{alim} - R_m i_m(t) - K_m \Omega(t)}{L_m} \\ \quad \dot{\Omega}(t) = \frac{1}{J}\left(K_m i_m(t) - \frac{r}{K_r}\left(MgK_f + \frac{1}{2}\rho S C_x \left(\frac{\Omega(t) r}{K_r} \right)^2 \right) \right) \\ \quad \dot{pos}(t) = \frac{\Omega(t) \times r}{K_r} \\ \\ \quad |i_m(t)| \leq 150 \\ \quad u(t) \in \{-1, +1\} \\ \\ \quad (i_m(0), \Omega(0), pos(0)) = (i_m^0, \Omega^0, pos^0) \in \mathbb{R}^3 \\ \quad (i_m(t_f), \Omega(t_f), pos(t_f)) \in \Gamma \subseteq \mathbb{R}^3 \end{cases} \tag{1.2}$$

The differential equations describing the constraints come from mechanical and electromechanical laws. The state variables are:

– i_m the current inside the motor,
– Ω the angular velocity,
– pos the position of the vehicle.

The control u is in {-1,1} (Bang-Bang type). In this problem, we have a constraint on a state variable: $|i_m(t)| \leq 150$A in order to restrict the current in the motor to discard the possibility to destroy it.

The other terms are fixed parameters and represent some physical things: $K_r = 10$, the coefficient of reduction; $\rho = 1.293kg/m^3$, the air density; $C_x = 0.4$, the aerodynamic coefficient; $S = 2m^2$, the area in the front of the vehicle; $r = 0.33m$, the radius of the wheel; $K_f = 0.03$, the constant representing the friction of the wheels on the road; $K_m = 0.27$, the coefficient of the motor torque; $R_m = 0.03\ Ohms$, the inductor resistance; $L_m = 0.05H$, the inductance of the rotor; $M = 250kg$, the mass; $g = 9.81m.s^{-2}$, the gravity constant; $J = M \times r^2/K_r^2$; $Valim = 150V$, the battery voltage; $R_{bat} = 0.05\ Ohms$, the resistance of the battery.

This problem is subject to boundary conditions. The initial conditions are given with the initial point (i_m^0, Ω^0, pos^0) at the starting time $t_0 = 0$ which is always fixed to (0,0,0) in this study, but the target set Γ is free and depends on the instances of the problem (see next chapters).

1.3 Approximation of Problem (1.2)

On the basis of its properties, Problem (1.2) can be approximated as an optimization problem. In fact, the formula (1.1) of the energy is only a function of the current i_m and the control u. That is why it is just required to search the trajectory of the current in the motor that minimize the energy consumption. If we discretize the interval of time $[0, t_f]$ by fixing the value of the control u, it is necessary to have very small steps ($\leq 10^{-3}$) in order to be able to control the current in the motor because the current changes too roughly (about 3 amps per 10^{-3} seconds). This will generate a very huge mixed integer global optimization problem which is difficult to solve with direct methods of optimal control.

In order to ovoid these problems, the idea is to impose during some short time the value of the current in the electrical motor of the vehicle which is possible using the control parameter $u(t)$ and a reference current.

$$\begin{cases} u(t) := -1 \text{ if } i_m(t) > iref + \frac{\Delta}{2}, \\ u(t) := +1 \text{ if } i_m(t) < iref - \frac{\Delta}{2}\ , \\ \text{else } u(t) \text{ takes its previous value.} \end{cases}$$

The control u switches between two values when the current i_m in the motor exceeds the

value of $iref$ with respect to a tolerance Δ. This technique is a way to build a current regulator which is a first step before building a velocity control for an electrical car. Thus, in this way, the system of differential equations to be solved is the following:

$$VS_{t_0,iref}(t) := \begin{cases} \dot{E}(t) = u(t)i_m(t)V_{alim} + R_{bat}u^2(t)i_m^2(t) \\ \dot{i}_m(t) = \frac{u(t)V_{alim} - R_m i_m(t) - K_m\Omega(t)}{L_m} \\ \dot{\Omega}(t) = \frac{1}{J}\left(K_m i_m(t) - \frac{r}{K_r}\left(MgK_f + \frac{1}{2}\rho SC_x\left(\frac{\Omega(t)r}{K_r}\right)^2\right)\right) \\ \dot{pos}(t) = \frac{\Omega(t)\times r}{K_r} \\ u(t) := \begin{cases} -1 \text{ if } i_m(t) > iref + \frac{\Delta}{2} \\ +1 \text{ if } i_m(t) < iref - \frac{\Delta}{2} \\ u(t) \text{ else.} \end{cases} \\ (E(t_0), i_m(t_0), \Omega(t_0), pos(t_0)) = (E^{t_0}, i_m^{t_0}, \Omega^{t_0}, pos^{t_0}) \in \mathbb{R}^4 \\ u(t_0) = 1; \end{cases} \tag{1.3}$$

where t_0 is the initial time which is not necessary equal to 0.

This system of differential equations can be efficiently solved using classical numerical integrator such as $Euler$, $RK2$, $RK4$ with a step of time less than 10^{-3} (else too many numerical errors to control i_m are generated). In our work, we use a step which is equal to 10^{-4}. The function $VS_{t_0,iref}(t)$ will compute all values of $E(t)$, $i_m(t)$, $\Omega(t)$, $pos(t)$, for all $t \in [t_0, t_f]$ but, in practice only discretized times $t_i \in [t_0, t_f]$ will be taken.

Here, we are only interested by final values of state variables, consequently, we define the function:

$$VSF(iref, t_0, t_f) := (E(t_f), i_m(t_f), \Omega(t_f), pos(t_f)) \in \mathbb{R}^4.$$

The computations are performed using the function $VS_{t_0,iref}(t)$ which solves the system of differential equations (1.3) under the initial conditions $(E^{t_0}, i_m^{t_0}, \Omega^{t_0}, pos^{t_0})$.

The approximation of Problem (1.2) using the system (1.3) is generated by subdividing the cycle of time $[0, t_f]$ to P sub-intervals. In each sample of time $[t_{k-1}, t_k]$ with $k \in \{1, \ldots, P\}$ ($t_k = k \times \frac{t_f}{P}$), we apply a reference current $iref_k$, which takes its values in $[-150, 150]$ to satisfy directly the constraint on the state variable i_m in Problem (1.2).

Hence, we focus on the resolution of the following global optimization problem:

$$\begin{cases} \min_{iref \in [-150,150]^P} \sum_{k=1}^{P} E_k \\ s.t. \\ \qquad (E_k, i_k, \Omega_k, pos_k) := VSF(iref_k, t_{k-1}, t_k) \\ \qquad (E_0, i_0, \Omega_0, pos_0) := (E^0, i_m^0, \Omega^0, pos^0) \\ \qquad (i_p, \Omega_p, pos_p) \in \Gamma \subseteq \mathbb{R}^3 \end{cases} \tag{1.4}$$

Problem (1.4) is a good approximation of Problem (1.2).

1.4 A Branch and Bound based method

For the moment, we are not able to solve exactly the global optimization problem (1.4). Thus, we need also to discretize the possible values of the reference current: $iref \in \{-150, -150+s, -150+2\times s, \ldots, 150\}^P$; we will take values for s that divide exactly the interval $[-150, 150]$. Therefore, the set of solutions becomes finite and could be enumerated. However, in order to have a good approximation for the resolution of Problem (1.4), we need to discretize into small sample of time. In this case, the set of possible solutions becomes rapidly very huge (about $(\frac{300}{s}+1)^P$). The idea will be to use a Branch and Bound algorithm.

1.4.1 Bounding techniques

To be able to use a Branch and Bound algorithm, we need to elaborate a technique that allows us to compute bounds of the 4 parameters: E_k, i_k, Ω_k, pos_k over all the discrete box $IREF \subseteq \{-150, -150+s, -150+2\times s, \ldots, 150\}^P$. In order to be more efficient, we compute 4 matrices M_E, M_{i_m}, M_Ω, M_{pos} where the columns correspond to values where $iref$ is fixed with $i_m^{t_0} = iref$ and the rows provide values when a velocity Ω^{t_0} is given (we discretize also the possible values of the velocity).

	$iref = -150 + (j-1)s$
	$\vdots$
$\Omega^{t_{k-1}} = (i-1)st\_V$	$\ldots \quad m_\Theta(i,j)$

st_V is the discretization step of the velocity values and Θ represents one of the symbols E, Ω, i_m or pos.

$e_E = (1,0,0,0)$, $e_{i_m} = (0,1,0,0)$, $e_\Omega = (0,0,1,0)$ and $e_{pos} = (0,0,0,1)$ are the vectors of the canonical basis of $\mathbb{R}^4$ which serve in computing the element $m_\Theta(i,j)$ using the following formula:

$$m_\Theta(i,j) =< VSF(iref, t_{k-1}, t_k), e_\Theta >,$$

where $< .,. >$ denotes the scalar product.
$m_\Theta(i,j)$ is obtained using the function $VS_{t_{k-1},iref}(t)$ over a sample of time $[t_{k-1}, t_k]$ under the following initial conditions:

$$(E^{t_{k-1}}, i_m^{t_{k-1}}, \Omega^{t_{k-1}}, pos^{t_{k-1}}) = (0, iref, \Omega^{t_{k-1}}, 0).$$

For example, $m_E(i,j)$ represents the value of the energy which is consummated in the interval $[t_{k-1}, t_k]$, where $iref$ corresponds to the j-th components of the set $\{-150, -150+s, -150+2\times s, \ldots, 150\}$ with $i_m^{t_0} = iref$ and the $i-$th line corresponds to the discretized value of the velocity , the other initial values are taken equal to zero: $E^{t_{k-1}} = pos^{t_{k-1}} = 0$. When a discrete box $IREF$ is considered, we compute the bounds of E, i_m, Ω and pos using the following method. Firstly, we compute the set of indices $I = \{i_1, i_2, \ldots, i_n\}$ and $J = \{j_1, j_2, \ldots, j_m\}$ corresponding respectively to the possible values of the speed at the previous sample and the possible values of $iref$. Then, we compute bounds corresponding to the maximal and minimal values of $m_E(i,j), m_{i_m}(i,j), m_\Omega(i,j), m_{pos}(i,j)$ with $(i,j) \in I \times J$. To obtain the final value of E and pos, we sum all the lower and upper bounds.

The rest of the Branch and Bound algorithm use the following principle:

(i) subdivision of the box $IREF$ in two distinct parts,

(ii) the upper bound is updating by taking the middle of $IREF$ if the constraints are satisfied and if its value is better than the previous one (we begin with $+\infty$),

(iii) we branch following the heuristic of lowest lower bound of the energy.

1.4.2 Alternative heuristics for computation of bounds

Our Branch and Bound code uses the data corresponding to the computations of the matrices M_Θ. The exact method EM for computing bounds, is expensive in CPU-time, then we were interested by the following 4 heuristics $H1$, $H2$, $H3$ and $H4$:

- For $H1$, the values of each sub-matrices at the first row and first column $m_E(i_1, j_1)$, $m_\Omega(i_1, j_1)$, $m_{pos}(i_1, j_1)$ are taken as lower bounds and we take for upper bounds,

the values corresponding to the last rows and last columns $m_E(i_n, j_m)$, $m_\Omega(i_n, j_m)$, $m_{pos}(i_n, j_m)$.

- For $H2$, we keep the same bounds as heuristic $H1$ for position. However, considering the energy and the velocity, we compute the minimum value on the first row for lower bounds and as upper bounds, the maximum value on the last row:
  - Lower bounds: $\min_{j\in J} m_E(i_1, j)$, $\min_{j\in J} m_\Omega(i_1, j)$, $m_{pos}(i_1, j_1)$.
  - Upper bounds: $\max_{j\in J} m_E(i_n, j)$, $\max_{j\in J} m_\Omega(i_n, j)$, $m_{pos}(i_n, j_m)$.
- For $H3$, we keep the same bounds as heuristic $H1$ for position and velocity. However, for the energy, the value of the sub-matrix at the last row and the first column $m_E(i_n, j_1)$ is taken as lower bound and respectively $m_E(i_1, j_m)$ as upper bound.
- For $H4$, we keep the same bounds as the heuristic $H2$ for position and velocity. Nevertheless, for the energy, as lower bound, we compute the minimum value on the last row and as upper bound, the maximum value on the first row:
  - Lower bounds: $\min_{j\in J} m_E(i_n, j)$, $\min_{j\in J} m_\Omega(i_1, j)$, $m_{pos}(i_1, j_1)$.
  - Upper bounds: $\max_{j\in J} m_E(i_1, j)$, $\max_{j\in J} m_\Omega(i_n, j)$, $m_{pos}(i_n, j_m)$

1.4.3 Branch and Bound Algorithm

1. Initialization: Let $L :=$ the initial domain in which the minimum is searched, $Emin := \infty$ the upper bound of the minimum (L is a list), $st\_Iref := s$ discretization step of reference current, $posmin := 0$ the initial position, $solmin := 0$ the initial solution, $st\_V :=$ the discretization step of the velocity (in km/h), $st\_tf := \frac{P}{t_f}$ denotes the length of the sample time.
2. Compute matrices: $Posf$, Ef, Vf.
3. **While** $L \neq \emptyset$ **do**
4. Extract an element X of L.
5. **for** i = 1 to 2 **do**
6. Bisect X into two discrete sub-boxes $X^{(1)}$ and $X^{(2)}$ following the largest edge of the convex hull of X.
7. Compute lower bounds lbE, $lbpos$, lbV and upper bounds $ubpos$, ubV of $X^{(i)}$ using $H1$, $H2$, $H3$, $H4$ or EM.

8. **if** $(ubpos \geq posf)$ and $(lbpos \leq posf)$ and $(lbE < Emin)$ **then**
9. $midi :=$ midpoint of the subbox $X^{(i)}$
10. $sol := [midi/st\_Iref] * st\_Iref$
11. Compute bounds $Esol$, $possol$, $Vsol$
12. **if** $(possol \geq posf)$ and $(Esol < Emin)$ **then**
13. $Emin = Esol$, $posmin = possol$, $solmin = sol$
14. **end if**
15. **if** $X^{(i)}$ is not reduced to a point **then**
16. Insert $X^{(i)}$ in L
17. **end if**
18. **end if**
19. **end for**
20. **end While**
21. Results: $Emin$, $posmin$, $Vsol$.

where [.] returns the closest integer value of a real number.

The general structure of this algorithm can be explained schematically in four stages: (i) extraction, (ii) division, (iii) analysis and (iv) insertion. The list L can have a LIFO or FIFO management, for example. In this work, we used FIFO which is much more efficient. For example, using Matlab and the LIFO management and taking the case of a displacement of 100 meters, a cycle time $t_f = 10$ seconds and a free final velocity, we obtain the refined solution $iref = (150, 139, 97, 85, 30, 30, -16, -31, -101, -131)$ corresponding to the energy $22718J$ after $4792s$ of time computation. However, using the FIFO management, we need only $323s$ to have the solution $iref = (150, 147, 104, 63, 21, 20, 0, -21, -80, -136)$ which corresponds to the energy $22648J$. This example is a good illustration of the efficiency of using FIFO management in terms of time and solution quality. Indeed, comparing to LIFO and using FIFO, the CPU time is divided by **14** with a gain of 0.31% on the quality of the global optimum.

In step 6 of the algorithm, the technique to bisect X into two discret sub-boxes proceeds by a selection of the component with the largest width of the convex hull of X and by bisecting it by the middle of this component. The loop **for** in line 5 is the main part of the Branch and Bound algorithm. Lines 7-8 concern the verification of constraint by computing bounds. This is the stage of analysis that uses one of the bounding techniques $H1$, $H2$, $H3$, $H4$ or EM. If the condition of line 8 holds, then the global minimum may occur in

the discrete subbox $X^{(i)}$ which justifies its insertion into the list (line 16), otherwise this subbox is directly discarded.

Finding a good feasible solution consists in evaluating the objective function at the midpoint of X. If this solution satisfies the constraints, we compare it to the best current solution of the problem *Emin*. This one is updated if it improves the current solution (line 13). The stopping criterion condition of the algorithm occurs when the whole list L is empty (line 3). This algorithm has an exponential complexity in time and in memory (as all discrete Branch and Bound algorithms).

Remark 1. *It should be noted that this algorithm can take into account constraints on the velocity state variable and the final velocity.*

i)- ***Null final velocity case:***
In this case, the final velocity is equal to 0 km/h. *The boundary conditions are written* $(i_m(0), \Omega(0), pos(0)) = (0, 0, 0)$; $(i_m(t_f), \Omega(t_f), pos(t_f)) \in \Gamma = \mathbb{R} \times \{0\} \times \{100\}$.
Our algorithm is modified in line 8 and 12. Line 8 becomes $(ubpos \geq posf)$ *and* $(lbpos \leq posf)$ *and* $(lbE < Emin)$ *and* $(lbV(ns+1) \leq Vf)$ *and* $(ubV(ns+1) \geq Vf)$. *However, Line 12 becomes* $(possol \geq posf)$ *and* $(Esol < Emin)$ *and* $(Vsol(ns+1) \leq Vf)$. Vf *refers to the final velocity and* ns *refers to the length of the list* L.

ii)- ***Case with constraints on the velocity state variable and on the final velocity:***
To take into account the constraint on the velocity parameter, we add the constraint $\Omega(t) \leq \frac{K_r}{3.6 \times r} \times 50$ *which indicates that a velocity limit is given to perform the displacement. The final velocity is fixed but in this case it has to be equal to* 50 km/h. *The boundary conditions for this case are written* $(i_m(0), \Omega(0), pos(0)) = (0, 0, 0)$; $(i_m(t_f), \Omega(t_f), pos(t_f)) \in \Gamma = \mathbb{R} \times \{\frac{K_r}{3.6 \times r} \times 50\} \times \{100\}$.
Line 8 becomes $(ubpos \geq posf)$ *and* $(lbpos \leq posf)$ *and* $(lbE < Emin)$ *and* $(all(lbV(1 : ns) \leq V(t)))$ *and* $(lbV(ns+1) \leq Vf)$ *and* $(ubV(ns+1) \geq Vf)$. *Line 12* $(possol \geq posf)$ *and* $(Esol < Emin)$ *and* $(all(Vsol(1 : ns) \leq V(t)))$ *and* $(Vsol(ns+1) \geq V(t))$. $V(t)$ *refers to the velocity state variable.*

1.5 Conclusion

The presence of a constraint on a state variable and the Bang-Bang structure of the control make the energy management problem very difficult to solve. The new way described in this chapter based on discretization and a Branch and Bound method to solve the

approximation of the optimal control problem presents a good approach to solve these difficulties.

Chapter 2

Implementation in Fortran 90

2.1 Introduction

The electrical vehicle energy management can be expressed as a Bang-Bang optimal control problem which is difficult to solve by conventional methods including direct methods (based on discretization) and indirect ones (based on Pontryagin's maximum principle). This problem has been studied by Abdelkader Merakeb in his thesis under the supervising of Frédéric Messine [3,4] and an original way to solve this problem was developed. This new method is based on a discretization technique associated with a Branch and Bound algorithm. This first study was performed using the high-level array programming language MATLAB which is widely used for prototyping algorithms and applications of scientific computations. However, its dynamically-typed nature, which means that MATLAB programs are usually executed via an interpreter, leads to poor performance in CPU time. An alternative approach would be the use of compiled programming language such as C++ or Fortran 90. In fact, we need in our algorithm, to manage lists where its length and complexity was not a priori known. That is why we emphasize on the use of C++ or Fortran 90 which are characterized by their simplicity in the manipulation of these data structures.

In this chapter, we will argue our choice to use Fortran 90 on the basis of some numerical tests performed on the fourth-order Runge-Kutta method comparing Fortran 90 versus C++. Then, we present the software architecture of our implementation: a detailed description of the implemented modules will be given. The final section is devoted to numerical tests performed for a displacement of 100 meters and a cycle of time $t_f = 10$ seconds. These results will be compared to those obtained using Matlab in [3,4].

2.2 Implementation of the fourth-order Runge-Kutta method

The optimal control problem describing the electrical vehicle energy management is composed of three elements:

- a cost function that depends on state variables,
- a set of differential equations describing the paths of the control and state variables that minimize the cost function,
- conditions that specify the initial and final values of the state variables.

The system of differential equations representing the constraints, can be efficiently solved using a classical differential integrator such as $RK4$ with a step of time less than 10^{-3}. First, we implemented this method in Fortran 90 and C++. The preliminary numerical tests show a strong sensitivity to the floating point variables which are used. In fact, to just use the simple precision floating point number in Fortran or C++ can lead to inaccurate results. Indeed, sometimes the single floating point data type provides less precision than required and therefore provided rounding errors. Whereas in MATLAB, this problem does not arise because all variables are treated in double floating point precision and there is no distinction between integer and real variables, nor between real and complex variables.

The loss of precision and significance due to the way where numbers are represented and the way where mathematical operations are performed, can at the end lead to totally inaccurate results (for example final current I_f) as shown in Table 2.1. In Table 2.1, we present the results by testing the $RK4$ method with the solution $irefT = (150, 90, 30, -30, -110)$.

	$F90$ / $C++$ (*single precision*)	$Matlab$ / $F90$ / $C++$ (*double precision*)
E_f (J)	23313	23228
Pos_f (m)	99.3595	99.3696
V_f (km/h)	11.13	11.16
I_f $(amps)$	-94.19	-110.26

Table 2.1: The influence of the chosen precision on the performance of results.

Using Fortran 90 and C++ with single floating point representation and comparing to MATLAB, we generate an error of 0.36% for the energy (E_f), 0.01% for the position (Pos_f),

14.57% for the current (I_f) and 0.03 km/h for the velocity (V_f). However, with double precision, Fortran and C++ succeed to determine the solution obtained using Matlab in a very short time. In fact, they converge to the solution after $0.04s$ of computations. It should be noticed that these computations use the pre-processing for computing matrices which needs $348s$.

Finally, we conclude that the chosen precision is important and the single precision floating point representation is in general not good enough in scientific computing. That is why, we will always recommend to employ double precision floating point representation in all computations with real numbers.

Why we use Fortran 90 ?

The choice to work with Fortran 90 (FORmula TRANslation) and not C++ is motivated by various reasons. Firstly, its syntax similar to that of mathematical language makes it a language particularly suitable for the treatment of scientific problems.

Besides, Fortran 90 is characterized by its simplicity. In fact, the objective during its design was to create a programming language that would be: simple to learn, suitable for a wide variety of applications, machine independent, and would allow complex mathematical expressions to be stated similarly to regular algebraic notation.

Next, the superiority of Fortran manifests itself mainly in the area of numerical, scientific and technical application which is the case of our Branch and Bound algorithm.

2.3 Software architecture

Our Fortran program is presented in the form of a structured set of modules written one after the others (see Figure 2.1). The advantage of using modules is that the main program can be kept relatively simple and easy to follow, while nitty-gritty calculations and complex procedures are shuffled off to various subroutines, each performing a specific task.

In order to be faster and more efficient, we try to compute four matrices (energy, position, velocity, current). Each element of these matrices corresponds to the solution of differential system (1.3) on a sample of time $[t_{k-1}, t_k]$ using the numerical integrator $RK4$. These matrices are stored in memory and used via the module **compute_bounds**. These

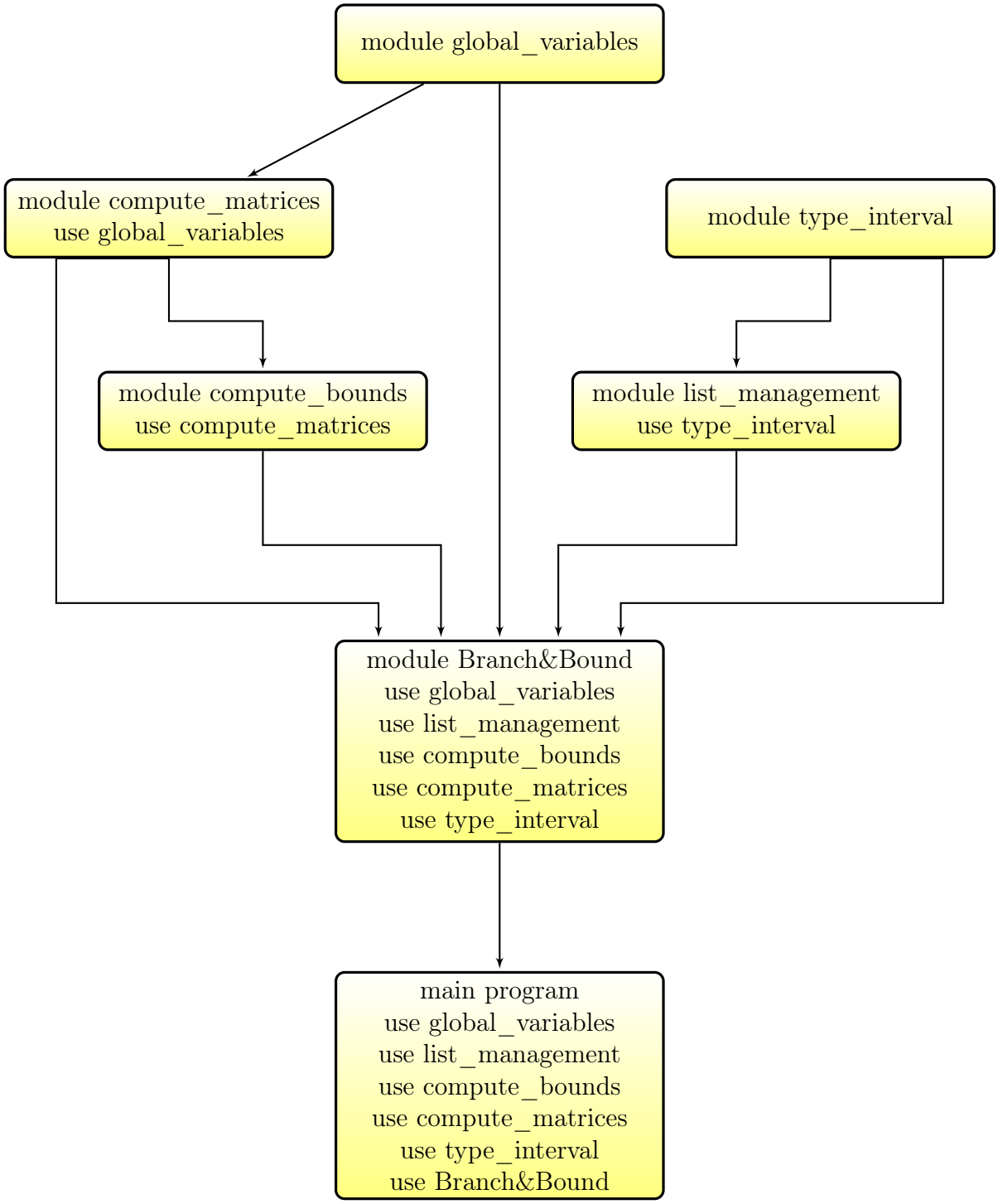

Figure 2.1: Software architecture of our code in Fortran 90.

computations use the module **global_variables** which contains the physical parameters of the problem and the simulation parameters.

As for modern programming languages (C and C++), Fortran 90 provides programmers the possibility to create and manipulate new data type like the intrinsic data types (integer, real, and logical). These are called derived data type in Fortran and structures in C and C++. This is the purpose of the module **type_interval** which allows us to create a new type which is essential for making our method more readable and easier to manipulate.

In our algorithm, we need to manage lists, collection of structures ordered by logical links where the complexity and the length are not a priori known. It is the role of the module **list_management** including the various possible operations on a list of elements (creation, insertion, extraction, deletion, display...). It should be noted that we used in implementing this module, some parts of the code IBBA which is dedicated to solve global optimization problems. It has been developed by Messine since the beginning of his thesis in 1997 [5] and he never ceased improving it [6,7].

The module **compute_bound** consists of two sub-programs:

1. When a block $IREF$ is considered, the first subroutine computes the bounds corresponding to the minimum and the maximum values of the energy, position, speed and current.

2. When a solution is considered, the second subroutine computes the final values of the state and control variables of the problem.

Using the various modules already mentioned, the module **Branch&Bound** implements our algorithm described in the previous chapter.

The **main program** is the compilation unit.

2.4 Numerical results

The algorithm developed to solve our global optimization problem is based on a Branch and Bound method whose complexity in the worst case is about $(\frac{300}{s}+1)^P$. In order to be faster and more efficient, four heuristics have been developed and after a series of tests, the heuristic $H4$ has been validated in the case where the final position is fixed to $100m$ and the final velocity is kept free [4].

When we evaluate our algorithm for parameters $P = 10$ and $s = 1$ using $H4$, we obtain the solution after 3 hours and 45 minutes of computations: $iref = (150, 140, 110, 70, 30, 20, -10, -40, -70, -140)$, which corresponds to 2 130 045 196 iterations of the Branch and

Bound algorithm; the corresponding minimum energy $Emin^*(10)$ is equal to 22772 J and position $pos^*(10)$ is equal 100.08 m (it should be noted that we are not able to have a satisfactory result for this instance using the Matlab code).

Nevertheless, one can reasonably expect that a notable reduction of the total computing time will be obtained through appropriate strategies for decreasing the time spent in exploring all the list of currrent. That is why, the idea is to run the Branch and Bound code iteratively by defining more specific areas around the exact solutions obtained in a previous step, by increasing the parameter P and decreasing parameter s. Using the solution $iref = (145, 86, 34, -16, -112)$ over a sampling intervals of $2s$, $(P = 5,\ s = 10)$, we construct $iref = (145, 145, 86, 86, 34, 34, -16, -16, -112, -112)$ over the sampling intervals of $1s$, $(P = 10,\ s = 10)$. Spreading it on a maximum range of 40 Amps, we generate the following box:

$$IREF = [125, 150] \times [125, 150] \times [66, 106] \times [66, 106] \times [14, 54] \times [14, 54] \times$$

$$[-36, 4] \times [-36, 4] \times [-132, -92] \times [-132, -92]$$

where the solution will be searched. The complexity in the worst case of our algorithm is reduced: $(\frac{40}{s} + 1)^P$.

We can iterate this process to refine the solutions progressively using the solutions obtained in the previous steps. In Table 2.2, we give the refined solutions obtained after 3 iterative runs of the Branch and Bound algorithm where P is increased and s is decreased by using previous solution to define a smallest domain of research.

Instance	*iref* (*amps*)	E_{min} (J)	*range* (*amps*)	*CPU* (*s*)		*Iter.*	
				Fortran	*Matlab*	*Fortran*	*Matlab*
$P = 5$ $s = 10$	(150, 90, 30, -30, -110)	23274	300	4×10^{-2}	2.65	23 074	23 069
$P = 10$ $s = 10$	(150,140,110,70,30, 20,-10,-30,-90,-130)	22813	40	0.79	11.48	264 374	264 406
$P = 10$ $s = 5$	(150,145,105,65,20, 20,0,-20,-80,-135)	22698	20	2.12	12.94	702 154	702 222
$P = 10$ $s = 1$	(150,147,104,63,21, 20,0,-21,-80,-136)	22648	4	4.70	12.73	1 405 661	1 410 307

Table 2.2: Table of refined solutions: free final velocity.

In Table 2.2, we remark a significant gain in time using Fortran 90 compared to Matlab.

Indeed, the CPU time is divided by **2** in the worst case and **66** in the best case, although the number of iterations is of the same order.

The last refined solution (last row in Table 2.2) which is represented on Figure 2.2 allows us to obtain a gain of 2.69% on the quality of the global optimum for 7.65 seconds of additional CPU-time. By validating this solution directly using $RK4$ (without using matrices), this provides an energy $\overline{E}min = 22876J$. The calculation error is about 0.99% for the energy.

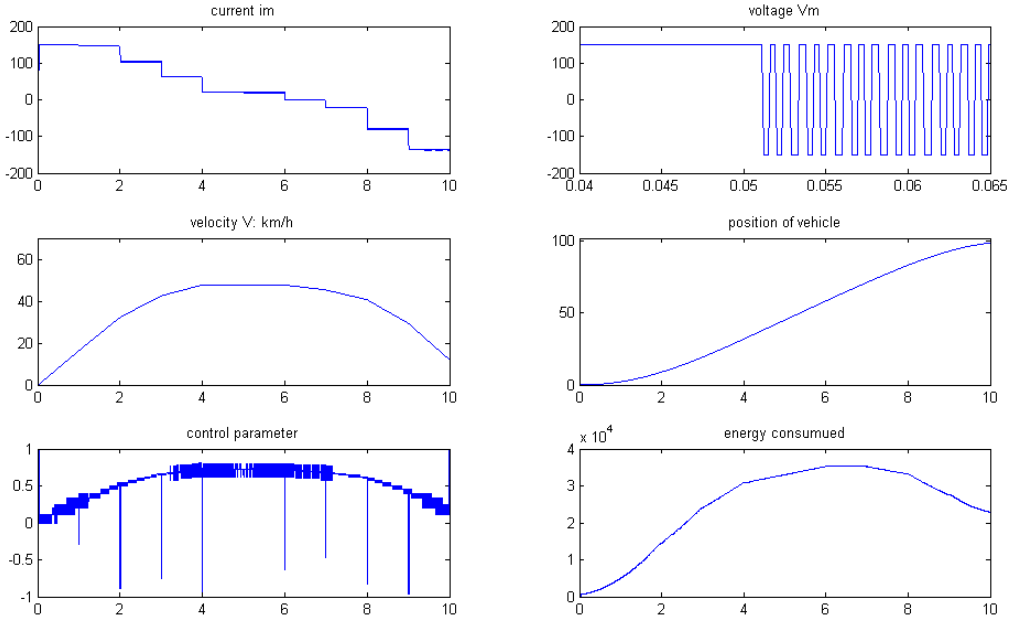

Figure 2.2: $Case : P = 10;\ s = 1;\ posf = 100m.$

On Figure 2.2, the scheme of the current takes its maximum value early in the cycle, decreases as far as reaching negatives values which corresponds to the phase of recovery. Indeed, the curve of the energy decreases at the end of the cycle because this corresponds to the phase of recovery. We remark also that the current i_m remains trapped around $iref$ with respect to the tolerance Δ. The values of u switch many times between -1 and +1; this is due to the fact that the current in the motor increases too quickly (about $3A$ every 10^{-3} seconds). The scheme representing the voltage V_m illustrates perfectly these switching phases. Its values are taken between -150 and +150 (the figure has been enlarged for the visualization).

Remark 2.
It should be noted that our algorithm can take into account constraints on the velocity state variable and on the final speed which is difficult for the indirect methods.
For more numerical results, see Appendix C.

2.5 Conclusion

Our primary goal in this chapter was to use Fortran 90 to speed up computations of our algorithm based on a Branch and Bound method. We feel this goal has been accomplished. Our secondary goal will to extend this Fortran code and this method to consider some other managements of the electrical vehicle energy. This point will be the subject of the next chapter.

Chapter 3

Extension of the method dedicated to long travels and travels with slopes

3.1 Introduction

As we saw in Chapter 2, the use of Fortran 90 gives us a very important gain in CPU-time compared to Matlab. This represents a real motivation to study more complex cases: long travels and management of travels including slopes.

We begin this chapter with a detailed description of two new heuristics which are developed after a study based on the structure of the used matrices (energy, position, velocity). This will be followed by giving the first extension. Numerical tests and some discussions for long travels ($1km$) will be given in Section 3.3. The last section will be devoted to the management of slopes. We supply a detailed algorithm that is used for studying this extension.

3.2 Heuristics $H5$ and $H6$

The evaluation of our method for a displacement of 100 meters with different conditions on the final velocity and a cycle of time $t_f = 10$ seconds, shows that the percentage of success on the solution of the heuristics described in Chapter 1 ($H1$, $H2$, $H3$ and $H4$) depends on the studied instance. In Table 3.1 (given later), we show this strong dependency. For example (from Table 3.1), Heuristic $H2$ seems very efficient in the case where the final position ($Posf$) is fixed to $100m$, the final velocity (Vf) is equal to 50 km/h and the state velocity ($V(t)$) is less or equal than 50 km/h. However, considering the free final velocity case, $H2$ is sometimes far from determining the exact solution and may generate an error

about 1.5% (see Table 3.1).

Therefore, we propose new heuristics that will be able to approach the solutions independently of the studied instance. These will be developed after a study on the behavior of the energy, the speed and the position matrices.

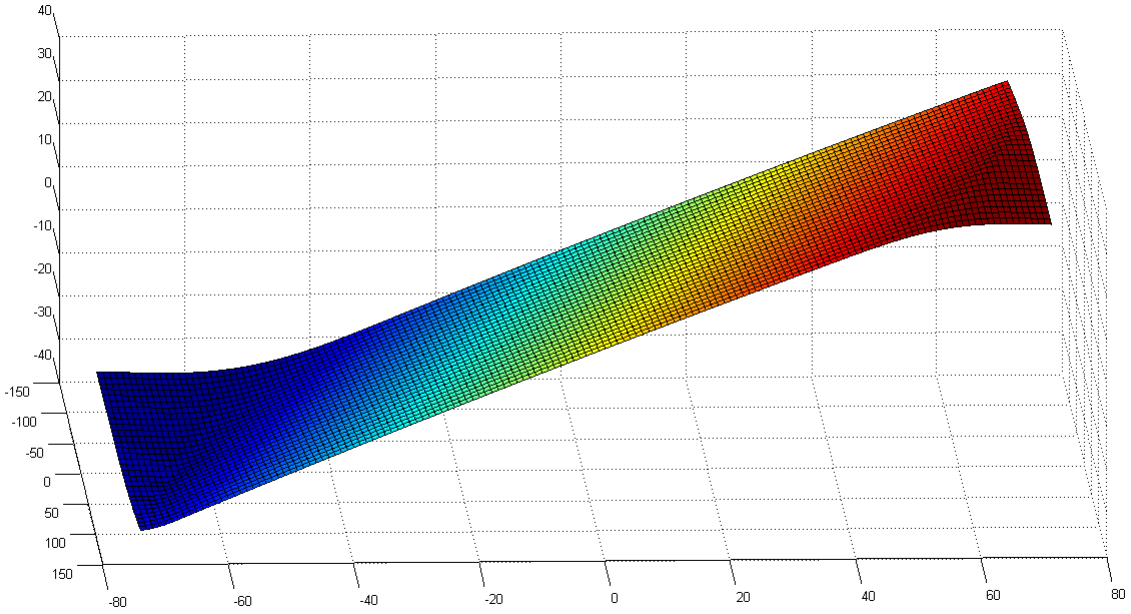

Figure 3.1: Behavior of the position matrix.

The position matrix has a growing behavior (see Figure 3.1). That is why, we keep the same bounds used in developing the heuristics of Chapter 1: the value of the sub-matrix at the first row and first column $m_{pos}(i_1, j_1)$ is taken as lower bound and we take for upper bound, the value corresponding to the last row and last column $m_{pos}(i_n, j_m)$.

However, the energy matrix has two concavities, see Figure 3.2. Therefore, it is just required to consider the minimum of the vertices of the rectangle where the energy is studied as a lower bound.

One of the reasons of the failure of the heuristics already developed is its vagueness in the computation of the velocity bounds, which is clear from the complicated behavior of the velocity matrix shown in Figure 3.3. Indeed, this structure presents three parts: a linear part and two others which have a complex behavior. Therefore, we need first to explore the whole sub-matrix to find the exact bounds. But, using the preliminary numerical results, we remark that we can look for bounds of the velocity only on some parts of the sub-matrix

(see $H6$).

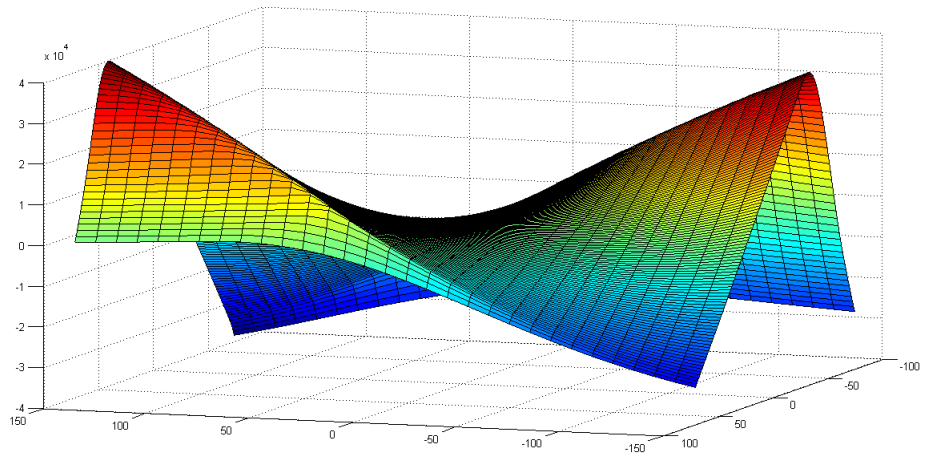

Figure 3.2: Behavior of the energy matrix.

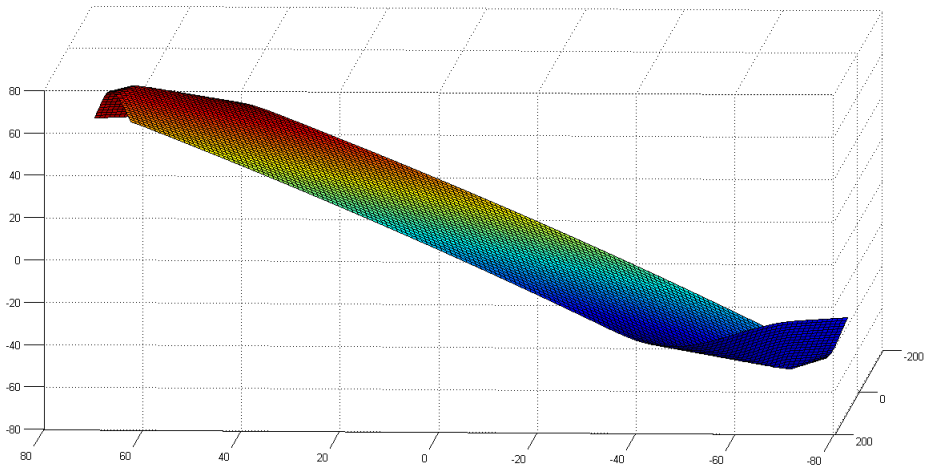

Figure 3.3: Behavior of the velocity matrix.

Based on the remarks mentioned above, we can describe the two new heuristics as follows:

- For $H5$: for the position, we keep the same bounds as heuristic $H1$, described in Chapter 1. As lower bound for the energy, we compute the minimum value of the

4 elements located on the vertices of the induced sub-matrix. To compute the lower (resp. upper) bound for the velocity, we consider the minimum (resp. the maximum) over all the sub-matrix.

  - Lower bounds: $\min(m_E(i_1, j_1), m_E(i_1, j_m), m_E(i_n, j_1), m_E(i_n, j_m))$, $\min\limits_{i \in I, j \in J} m_\Omega(i, j)$, $m_{pos}(i_1, j_1)$.
  - Upper bounds: $\max\limits_{i \in I, j \in J} m_\Omega(i, j)$; $m_{pos}(i_n, j_m)$.

- For $H6$: we keep the same bounds as heuristic $H5$ for the position and the energy. Nevertheless, for the velocity, as lower bound, we compute the minimum value on the first column and as upper bound, the value corresponding to the last row and last column of the induced sub-matrix.

  - Lower bounds: $\min(m_E(i_1, j_1), m_E(i_1, j_m), m_E(i_n, j_1), m_E(i_n, j_m))$, $\min\limits_{i \in I} m_\Omega(i, j_1)$, $m_{pos}(i_1, j_1)$.
  - Upper bounds: $m_\Omega(i_n, j_m)$, $m_{pos}(i_n, j_m)$.

Remark 3. *For the sets I and J, see Sub-section 1.4.1.*

In order to test the effectiveness of these two new heuristics $H5$ and $H6$, we performed 101 tests in which the final position varies from $20m$ to $120m$ in increment of $1m$. In Table 3.1, we present the percentage of success of different heuristics. $H5$ and $H6$ are completely reliables on these tests. Indeed, these heuristics are able to give the same solution as using the exact method EM independently of the studied instance without generating any error contrary to other heuristics.

$profile$	$H1$	$H2$	$H3$	$H4$	$H5$	$H6$
Vf free	0%	67%	97%	100%	100%	100%
$Vf = 0$	33%	98%	94%	94%	100%	100%
$Vf = 50$ $V(t) \leq 50$	76%	100%	77%	78%	100%	100%

Table 3.1: Percentage of success of different heuristics with respect to the studied instance.

Moreover, to validate the use of Heuristics $H5$ and $H6$, we compare their performance in CPU-time as presented in Table 3.2. This table is obtained by performing the same tests used in Table 3.1.

$Profile$	$H5$	$H6$	EM
Vf free	$0.5s$	$0.13s$	$2.04s$
$Vf = 0$	$0.22s$	$0.05s$	$1.03s$
$Vf = 50$ $V(t) \leq 50$	$0.24s$	$0.05s$	$1.04s$

Table 3.2: Performance of heuristics $H5$ and $H6$ in CPU-time.

From Table 3.2, Heuristic $H6$ is faster than $H5$. In fact, considering for example the zero final velocity case, comparing to EM and using $H6$, the average CPU-time is divided by **20**. However, it is only divided by **4** when using $H5$. This important gain in time for $H6$ compared to EM is also clear from the scheme given in Figure 3.4 (which is obtained using the last performed tests in the case of a free final velocity).

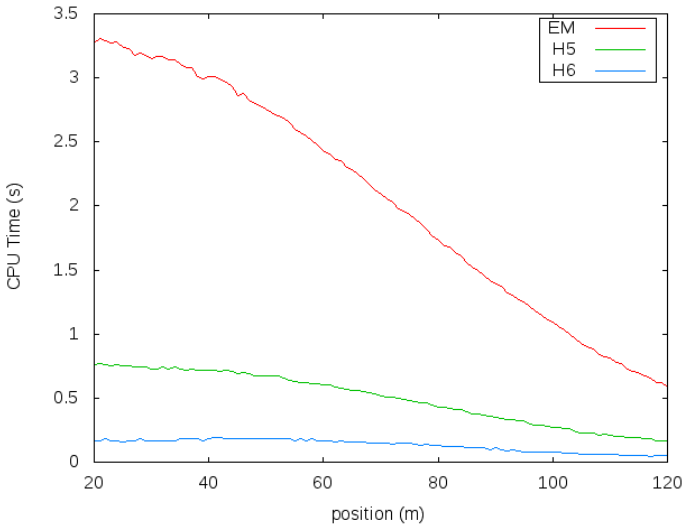

Figure 3.4: CPU-time, case: $P = 5$; $s = 10$.

For the case of $20m$, we remark that the exact method EM is very slow comparing to $H5$ and $H6$. This is due to the fact that the set of possible solutions is very large which is not the case where the final distance is $120m$.

3.3 Case of long travels

To have a good approximation of the global optimum of Problem (1.4), we have to discretize the interval $[0, t_f]$ of time into small samples. Since the final time t_f is large, the finite set

of possible solutions becomes too huge. In order to avoid this problem, the idea is to use variable steps of discretization instead of constant ones. Indeed, we can simply divide the first and last ten seconds with steps of $2s$ and the remainder part with uniform steps of $10s$.
Returning to the algorithm described in Chapter 1 and taking into account the above remarks, we obtain the following numerical results.

3.3.1 Case without constraint on velocity

We evaluate our method for a displacement of $1000m$ and a cycle of time $t_f = 70s$: $(i_m(0), \Omega(0), pos(0)) = (0, 0, 0)$; $(i_m(t_f), \Omega(t_f), pos(t_f)) \in \Gamma = \mathbb{R} \times \mathbb{R} \times \{1000\}$. Testing our algorithm over a sampling intervals of $10s$, we obtain the exact solution $iref = (140, 10, 30, 20, 30, 20, -20)$ which is equivalent to $(140, 140, 140, 140, 140, 10, 30, 20, 30, 20, -20, -20, -20, -20, -20, -20)$ (see method above). Spreading it on a range of 20 amps, we generate a new box where the solution will be searched. By repeating this process with a range of 4 amps to refine the solutions, we obtain the results presented in Table 3.3.

Instance	*iref*	E_{min} (J)	*posf* (m)	Vf km/h	CPU (s)	*range* $(amps)$	*Iter.*
$P = 7$ $s = 10$	(140, 10, 30, 20, 30, 20, -20)	206 945	1000.02	12.93	12	300	3 354 323
$P = 15$ $s = 10$	(140,130,130,130,150, 20,20,20,30,20, -10,-10,-10,-30,-30)	204 987	1000.17	13.63	4	20	683 753
$P = 15$ $s = 1$	(141,128,128,128,149, 18,21,21,28,22, -10,-8,-8,-29,-32)	203 934	1000.01	14.86	3671	4	627 133 367

Table 3.3: Refined solutions: free final velocity.

With the improved solutions, we obtain a gain of 1.45% on the quality of the optimum for 3687 seconds of additional CPU-time. The curves on Figure 3.5 are drawn using the latest refined solution (last row in Table 3.3). The computation of this solution, by simulation using $RK4$, provides an energy $\overline{E}_{min} = 207353\ J$ and a position $\overline{pos} = 1006.6m$. The error is about 1.65% on the energy and 0.65% on the position.

The current i_m takes its maximum values in the first ten seconds to ensure the starting phase, decreases as far as reaching a "steady" value around 21 *amps* and ends with negative values which corresponds to the phase of recovery. In fact, the decreasing of the curve of

the energy at the end of the cycle (last ten minutes) corresponds to this phase. The current i_m remains trapped around $iref$ with respect to the tolerance Δ. The values of the control u switch many times between -1 and +1; this is due to the fact that the current varies too quickly in the motor (about 3 amps every 10^{-3} seconds). The curve of the voltage V_m illustrates these switching phases; it takes its values between -150 and +150. The curve of the control represents the values of u synthesized to smooth signal by the PWM (Pulse-Width Modulation) technique on a succession of discrete states (+1,-1) during a cycle of two switches; the assigned value is the mean over all values.

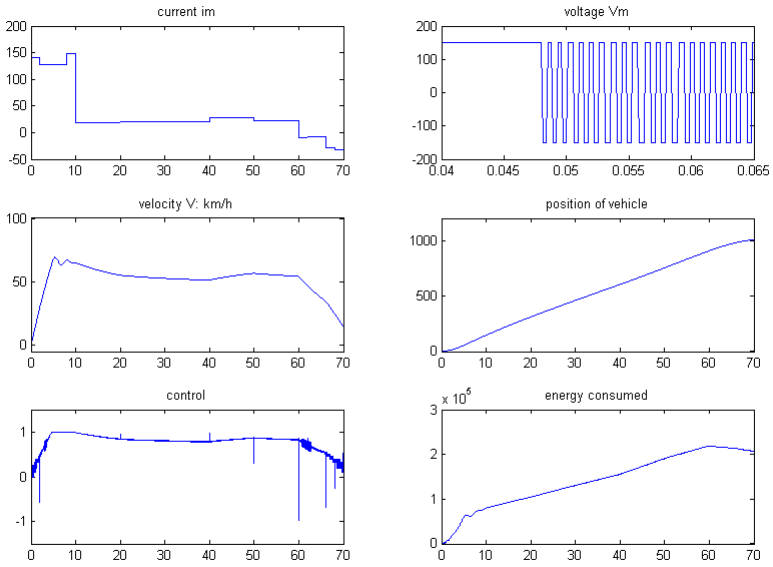

Figure 3.5: $Case$: $P = 15$; $s = 1$; $posf = 1000m$.

3.3.2 Case with null final velocity

The final velocity on Figure 3.5 is not null. Nevertheless, our method with successive refinement can take into account this parameter. To compare solutions, we simulate last instances with a constraint on the final velocity; i.e., a displacement of $1000m$ and a cycle of time $t_f = 70$ seconds with a null final velocity: $(i_m(0), \Omega(0), pos(0)) = (0, 0, 0)$; $(i_m(t_f), \Omega(t_f), pos(t_f)) \in \Gamma = \mathbb{R} \times \{0\} \times \{1000\}$. The corresponding refined solutions are presented in Table 3.4.

$Instance$	$iref$	E_{min} (J)	$posf$ (m)	Vf km/h	CPU (s)	$range$ $(amps)$	$Iter.$
$P = 7$ $s = 10$	(130, 10, 70, 20, 20, 20, -30)	214761	1000.43	-0.69	5	300	1 337 855
$P = 15$ $s = 10$	(130,120,120,120,120, 20,60,10,20,30, -20,-30,-40,-40,-40)	212591	1000.22	-0.46	3.17	20	705 279
$P = 15$ $s = 1$	(132,118,119,121,122, 18,58,10,21,29, -18,-28,-38,-41,-42)	211295	1000.03	-0.02	3622	4	615 528 064

Table 3.4: Refined solutions with null final velocity.

The last refined solution is obtained using a total CPU-time about $3630s$. Using the numerical integrator $RK4$, the calculation error is about 0.36% for the energy ($\overline{E}_{min} = 212052\ J$) and 0.008% for the position ($\overline{pos} = 999.95m$).

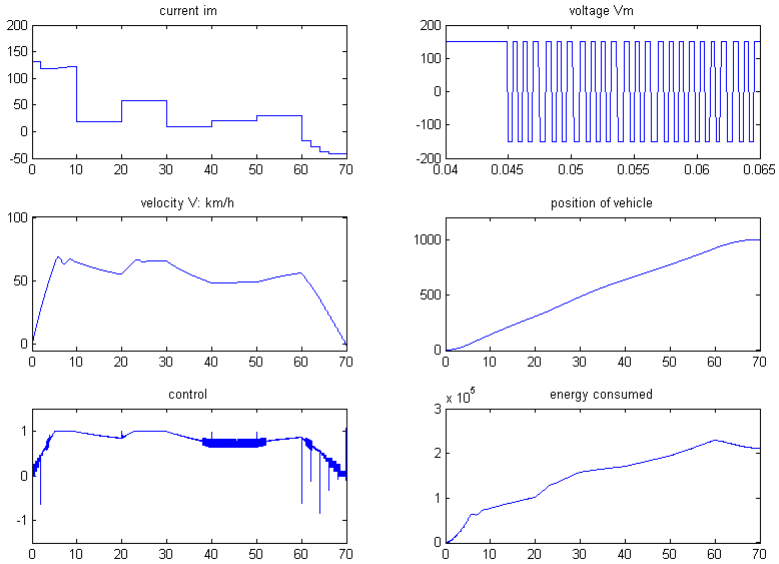

Figure 3.6: $Case$: $P = 15$; $s = 1$; $posf = 1000m$; $Vf = 0\ km/h$.

Because the vehicle starts and ends with a velocity which is equal to zero, it necessarily goes through a phase of deceleration corresponding to the recovery phase of electrical energy. The current is to the bottom at the first moments and ends with its minimum

value to be able to stop the vehicle (null final velocity). The curve of the control in Figure 3.6 follows exactly the velocity one.

3.3.3 Case with constraints on the velocity state variable and on the final velocity

If we test our algorithm with constraints on velocity state ($V(t) \leq 50km/h$) and final velocity ($Vf = 50km/h$), our simulations show that it is not possible to perform the displacement of $1km$ in a limited time equal to $70s$. That is why, we limited the final distance to $800m$. The solutions obtained using successive runs and refinement of our Branch and Bound code, are reported in Table 3.5 and drawn in Figure 3.7.

$Instance$	$iref$	E_{min} (J)	$posf$ (m)	Vf km/h	CPU (s)	$range$ $(amps)$	$Iter.$
$P = 7$ $s = 10$	(50, 20, 10, 20, 20, 20, 40)	167 522	803.22	63.59	0.06	300	13 697
$P = 15$ $s = 10$	(60,60,50,50,50, 20,20,20,10,10, 40,30,30,30,40)	150 391	800.67	51.04	0.17	20	31 314
$P = 15$ $s = 1$	(62, 62, 50, 48, 51, 18, 18, 22, 12, 10, 38, 29, 28, 28, 38)	148 242	800.03	50.05	2963	4	466 010 273

Table 3.5: Refined solutions: constraints on the velocity state and final velocity ($Vf = 50km/h$).

To validate the solution obtained in Table 3.5, we use directly $RK4$. This provides an energy $\overline{E}min = 148056J$, a position $\overline{pos} = 799.10m$ and a final velocity $Vf = 50.3\ km/h$. The error is about 0.1% for the energy, 0.1% for the position and $\pm 0.25\ km/h$ for the velocity.

The curve of the current on Figure 3.7 is in its maximum value at the beginning of the cycle of time, then decreases to maintain the velocity at its desired value and finally, increases in order to make the vehicle able to finish with a velocity of $50km/h$. In this profile, there is no recovery of the electrical energy which can be seen on the energy curve.

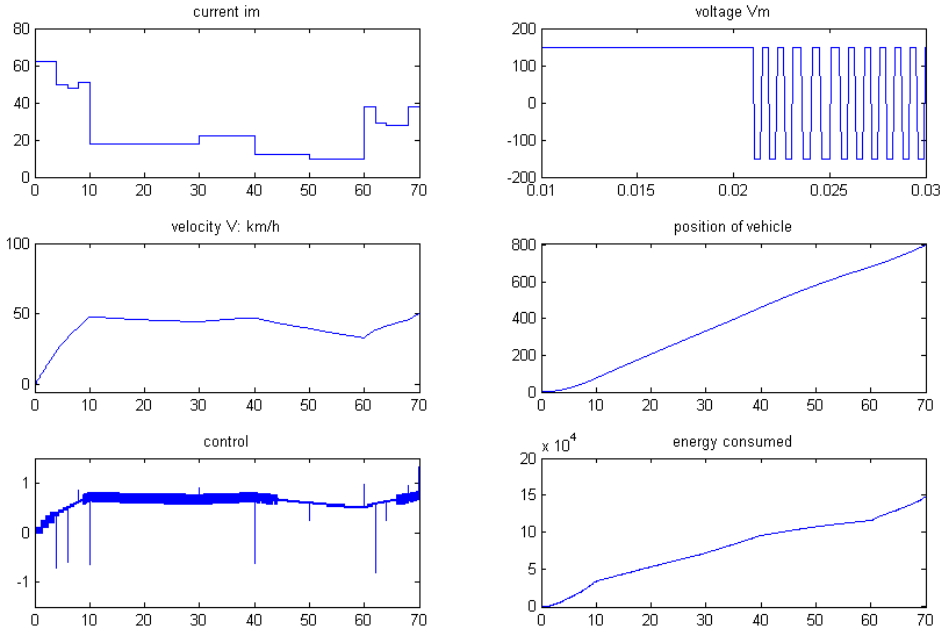

Figure 3.7: *Case*: $P = 15$; $s = 1$; $posf = 800m$; $V(t) \leq 50\ km/h$; $Vf = 50\ km/h$.

3.3.4 Conclusion

Based on the numerical tests already done, we can remark that:

i) Using successive runs and refinement of the Branch and Bound code, we obtain an important gain on the quality of the global optimum which needs a very long CPU-time.

ii) The refined solutions obtained with the different studied cases have an identical behavior at the beginning of the interval of time, i.e. i_m takes its maximum value early in the cycle to ensure the starting phase. This is not the case at the end of the cycle. Indeed, considering the free or null final velocity case, the curve of current i_m ends with negative currents which corresponds to the recovery phase. However, considering a non-zero final velocity and a limit on the speed (sub-section 3.3.3), i_m increases in the last ten minutes to maintain the final velocity and in this case, there is no recovery phase.

3.4 Management of slopes

Until now, we just study cases where the travels are flat (no slope is considered). However, the modeling of the problem can take it into account.

$$\begin{cases} \min_{im(t),\Omega(t),pos(t),u(t)} \quad E(t_f, i_m, u) \\ s.t. \\ \quad \dot{i}_m(t) = \frac{u(t)V_{alim} - R_m i_m(t) - K_m \Omega(t)}{L_m} \\ \quad \dot{\Omega}(t) = \frac{1}{J}\left(K_m i_m(t) - \frac{r}{K_r}\left(MgK_f + \frac{1}{2}\rho S C_x \left(\frac{\Omega(t) r}{K_r}\right)^2 + Mg\ sin(\frac{\pi}{180}\theta)\right)\right) \\ \quad \dot{pos}(t) = \frac{\Omega(t) \times r}{K_r} \\ \quad |i_m(t)| \leq 150 \\ \quad u(t) \in \{-1,\ +1\} \\ \quad (i_m(0), \Omega(0), pos(0)) = (i_m^0, \Omega^0, pos(0)) \in \mathbb{R}^3 \\ \quad (i_m(t_f), \Omega(t_f), pos(t_f)) \in \Gamma \subset \mathbb{R}^3 \end{cases} \tag{3.1}$$

where θ refers to the angle of the slope. This is the same model as (1.2) but $-\frac{1}{J}\frac{r}{K_r} Mg\ sin(\frac{\pi}{180}\theta)$ is added to penalize the velocity if some slopes in degree have to be performed at time t_*.

3.4.1 Algorithm

The difficulty to solve Problem (3.1) using our algorithm comes essentially from the computation of bounds. In fact, we are not able to determine exactly the optimal time t_*, when the vehicle reaches the position of change of the slope. That is why, we just try to determine the sample time in which that distance having a slope is reached ($50m$ in our case). Once found, we fall to the beginning of this sample which will be again subdivided using a step of $0.5s$ in order to obtain a greater accuracy.

These computations require additional pre-processing for computing matrices over a sample time of length $0.5s$ taking into account the different slopes encountered in our travel. Different steps described above can be summarized in these five points:

1) search the sample time in which the vehicle reaches the position of the change of slope,

2) return to the beginning of this sample time,

3) reinitialize the state variables by taking their values in the previous sample,

4) subdivide it with a step of $0.5s$,

5) research the sample time of length $0.5m$ (included in the one found in step 1) in which the vehicle reaches the position when the slope changes.

Using these steps and including them in our algorithm (in line 11 of the algorithm presented in Chapter 1), we will be able to manage travels including slopes.

3.4.2 Numerical results

Testing our algorithm for a displacement of $100m$ including a slope of $+3°$ using the "classical" method for computing bounds, we notice that we may have some solutions which present important stalls. For example, considering a free final velocity case, we obtain the solution $iref = (150, 150, -150, 100, -70)$ producing an energy of $24338J$. In order to avoid such situations, the idea will be to consider constraints on the acceleration. To perform this, we proceed as follows:
the linear speed (in m/s) of the vehicle obtained from the motor speed is written:

$$V = \frac{r}{K_r}\ \Omega$$

and consequently, the acceleration is

$$A = \dot{V} = \frac{r}{K_r}\ \dot{\Omega}$$

This acceleration is measured in m/s^2. It can be converted to "g" (the gravity of Earth) if we prefer to compare it to the acceleration of gravity which is $1g = 9,81m/s^2$. Indeed, $1g$ means that an object of $1kg$ accelerates, in a crash for example, with $9,81m/s$ per second (we say $9,81\ m.s^{-2}$) or also, we add every second about $35,3km/h$ to our initial velocity, i.e., we reach the velocity of $105,9km/h$ after the first three seconds, which can be unbearable by the person inside the electrical vehicle. Thus, the objective of this conversion is to be able to control and determine the acceleration that is supported by the vehicle and the driver.

Based on the above remarks, we can add this new constraint:

$$|A| \leq \beta\ g \tag{3.2}$$

where $\beta \in]0, 1]$.

To take into account this constraint, we need to compute a new matrix which is that of the acceleration. It should be noticed that the smallest is β, the smoothest is the solution,

as presented in Table 3.6.

$coefficient$	$iref$	E_{min} (J)	$posf$ (m)	Vf km/h	CPU (s)	$Iter.$
$\beta = 1$	(150,150,-150,100,-70)	24338	100.74	21.71	42.07	1 553 280
$\beta = 0.5$	(150,150,-120,70,-90)	25223	100.22	21.99	37.22	1 527 197
$\beta = 0.4$	(100,80,130,-50,-90)	25898	100.09	22.70	38.26	1 527 268
$\beta = 0.3$	(100,100,70,20,-90)	28954	100.18	29.24	39.14	1 528 126

Table 3.6: Influence of the choice of β on the quality of solutions.

In Table 3.6, we present the solutions of Problem (3.1) by taking into account the new constraint (3.2). These numerical results correspond to a distance of 100m including a slope of $+3°$ (at $50m$) with a free final velocity.

Although the choice of $\beta = 1$ gives the minimum value for the energy consumed by an electrical vehicle compared to other choices, we will not consider this solution for the reasons already mentioned. It should be noticed that this solution is also the absolute one when constraint (3.2) is removed. However, the maximum acceleration (resp. minimum) corresponding to the choice of $\beta = 0.3$ is about $3m.s^{-2}$ $(-3m.s^{-2})$ which is equivalent to reach speed of 34.77 km/h in 3 seconds which can be accepted in a driving located at the limit of "quiet" on dry ground.

Testing our algorithm for a displacement of $100m$ including a slope of $+3°$ (at $50m$) with different constraints on the state velocity and the final speed, we find the results presented in Table 3.7.

$Instance$	$iref$	E_{min} (J)	$posf$ (m)	Vf km/h	CPU (s)	$Iter.$
$Posf = 100$ Vf free	(100,100,70,20,-90)	28 954	100.18	29.24	39.14	1 528 126
$Posf = 100$ $Vf = 0$	(150,120,20,-50,-130)	29 618	100.00	1.21	2.09	13 499
$Posf = 100$ $Vf = 50$ $V(t) \leq 50$	(90,90,70,20,50)	48 944	100.07	52.01	9.81	234 807

Table 3.7: Solutions for a displacement of $100m$ including a slope $+3°$.

Considering the free final velocity case, we note that the minimum energy consumption is about 22% greater than in the case without including slope. The calculation of the solution (first row), by simulating using $RK4$ directly, provides an energy $\overline{E}min = 28341J$ and a position $\overline{pos} = 100.99m$. The error is about 2.16% for the energy and 0.8% for the position.

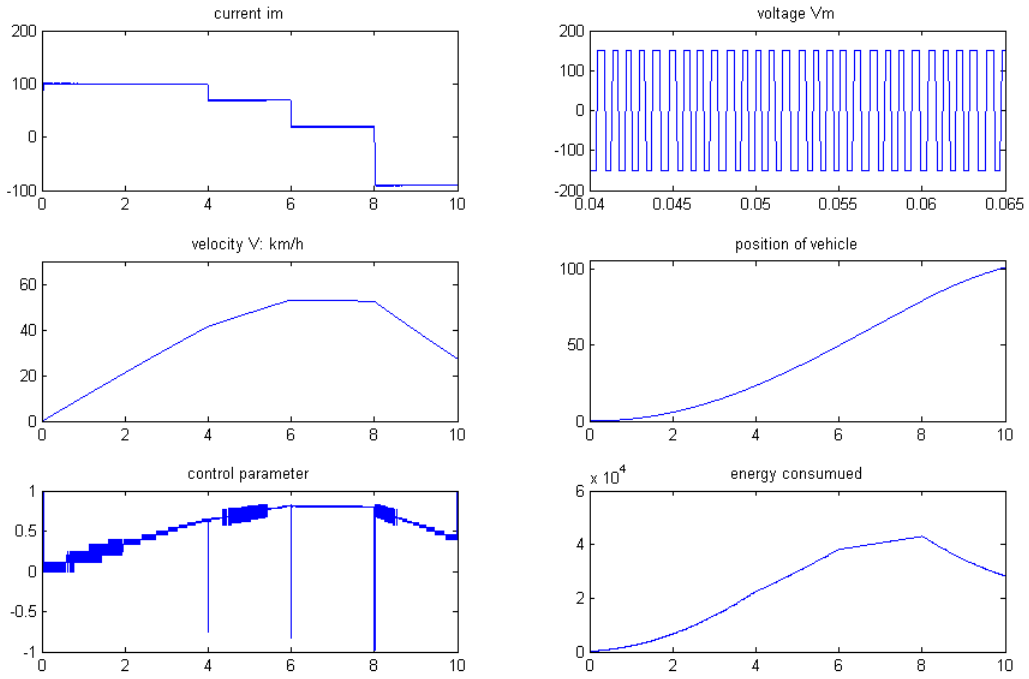

Figure 3.8: Free final velocity case with a slope of $+3°$.

Despite the presence of a positive slope, we remark that there is a recovery phase. In fact, the curve of the energy decreases at the end of the cycle (the last two seconds).

Our algorithm can also take into account negative slope as presented in Table 3.8.

Instance	*iref*	E_{min} (J)	*posf* (m)	*Vf* km/h	*CPU* (s)	*Iter.*
$Posf = 100$ Vf free	(120,100,20,10,-70)	22 187	100.14	29.24	64.59	1 888 871
$Posf = 100$ $Vf = 0$	(150,120,10,-60,-140)	24 277	100.03	0.21	23.6	568 987
$Posf = 100$ $Vf = 50$ $V(t) \leq 50$	(90,90,70,20,20)	41 796	100.54	52.01	11.76	279 732

Table 3.8: Solutions for a displacement of $100m$ including a slope $-3°$.

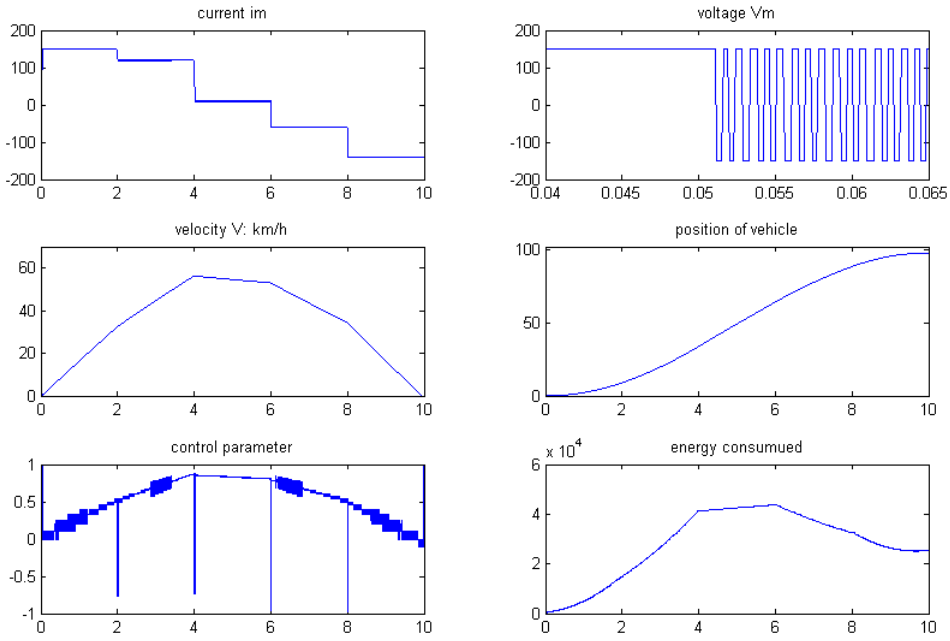

Figure 3.9: Null final velocity case with a slope of $-3°$.

Figure 3.9 represents the obtained solution in the null final velocity case with a slope of $-3°$. Using the solution $iref = (150, 120, 10, -60, -140)$, the electrical vehicle reaches the position $50m$ presenting the slope $-3°$ in the third sample of time, i.e. between $4s$ and $6s$, which is clear from the curve of the velocity. Indeed, this sample corresponds to the beginning of the phase of deceleration corresponding to the recovery phase of electrical energy.

We consider now the case of the non-zero velocity and a limit on the speed (last row in Table 3.8). The minimum energy consumption for this case is about 17.1% smaller than in the case with a slope of $+3°$. By validating this solution using $RK4$, this provides an energy of $\overline{E}min = 41474J$ and a position $\overline{pos} = 99.09m$. The error is about 0.77% for the energy and 1.46% for the position. The current takes its maximum at the beginning of the cycle and ends with constant value, 20 amps in the last four seconds. This allows maintaining the velocity at its desired value. In this case, there is no recovery phase despite the presence of negative slope. This is due to the fact that the final velocity must be equal to $50km/h$.

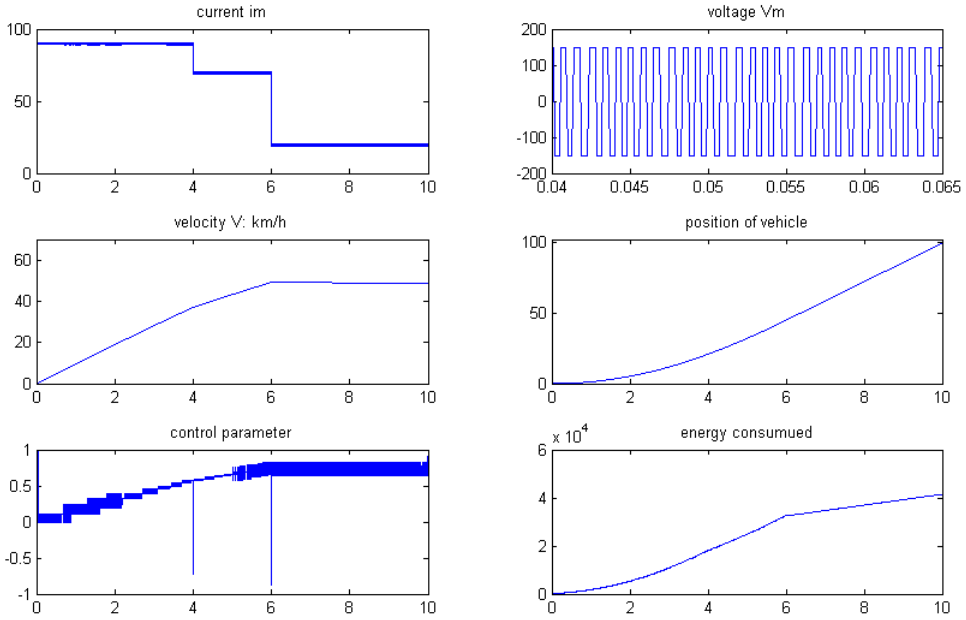

Figure 3.10: Non-zero final velocity and a limit on the speed with a slope of $-3°$.

3.4.3 Conclusion

As we saw in this section, we can extend our original algorithm to solve the slope management problem. The preliminary tests give too exaggerated solutions in terms of acceleration. That is why, we add a new constraint on the acceleration. Taking into account this condition, our code provides new solutions which are smoother than that given before and more expensive in terms of consumed energy.

Remark 4. *Studying this extension, we remark that, in some other cases (for example*

combining negative and positive slope in the same travel), our method can generate large errors which come essentially from the use of 4 matrices computed over a sample time of length 0.5s to recover that of 2s (see Appendix B).

3.5 Conclusion

In this chapter, we proposed new heuristics which are more reliable than that already developed in Chapter 1. This allows us to study some extensions of the electrical energy management problem. We begun with the study of the extension to long travels and we finished by giving a detailed description of different steps for dealing with the slopes management problem.

Conclusion

The energy management problem of an electrical vehicle can be formulated as an optimal problem with a Bang-Bang control which is actually difficult to solve using classical methods. An innovative methodology to solve this problem was developed in [3,4]. We started by presenting this method which is the basis of our work.

The first goal of this internship is to provide a compiled code in Fortran 90 which was for us the best target language. The numerical results show that this compiled program performs better than their MATLAB executions and the speedups vary, ranging from **2** to **66** depending on the studied instance. We also observe that the heuristics already developed are not reliable enough to approach the global optimum. That is why, we constructed two new heuristics ($H5$ and $H6$) which are very powerful and able to determine the exact solution independently of the studied profile. These significant gains in terms of computation time without loss in quality solution encouraged us to study some extensions of electrical vehicle energy management problems which are the study of long travels and the management of slopes.

The methodology already developed is based on discretization of the possible values of the current and velocity. Nevertheless, if we want to have a good approximation of the global optimum of the initial optimal control problem, we have to discretize into small samples which is very expensive in terms of memory. In order to be more efficient (by discarding the use of large matrices), the idea will be to use interpolation functions for position, velocity and energy. This new approach can help us to move from the study of discrete case to the continuous one and then, to have a better approximation of the optimal control problem. Furthermore, after the realization of a current regulator, we also want to construct a velocity regulator of an electrical vehicle. These approaches could be the subject of a futur work.

Bibliography

[1] E. Trélat, *Contrôle optimal: théorie et applications*, Vuibert, Collection: "Mathématiques Concrètes", 2005.

[2] R. Vinter, *Optimal Control, Systems and Control: Foundations and Applications* Birkhuser Boston, Inc, Boston, MA, 2000.

[3] A. Merakeb and F. Messine, *On Minimizing the Energy Consumption of an Electrical Vehicle*, preprint in Optimization Online, N° 2997, 2011.

[4] A. Merakeb, *Optimisation Multicritères en contrôle Optimal: Application au Véhicule Électrique*, Thèse de doctorat, Université Mouloud Mammeri, Tizi-Ouzou, Algerie, 2011.

[5] F. Messine, *Méthode d'Optimisation Globale basée sur l'Analyse d'Intervalles pour la Résolution de Problèmes avec Contraintes*. Thèse de doctorat, Institut National Polytechnique de Toulouse, 1997.

[6] F. Messine, *L'Optimisation Globale par Intervalles: de l'Étude Théorique aux Applications*, Habilitation à Diriger des Recherches, Institut National Polytechnique de Toulouse, 2006.

[7] J. Ninin, *Optimisation Globale basée sur l'Analyse d'Intervalles: Relaxation Affine et Limitation de la Mémoire*. Thèse de doctorat, Institut National Polytechnique de Toulouse, 2010.

Appendix A

Interpolation

In order to use our Branch and Bound algorithm, we need a pre-processing to compute matrices of energy, velocity and position. To have a good approximation of Problem (1.4), these matrices should be of a large size to recover the maximum possible cases which is very expensive in terms of memory especially when we study long travels, as we presented in Chapter 3. In order to avoid such situations, our idea will be to use interpolation functions: i.e, for each matrix of the three ones, we search a function that passes as close as possible to some elements. As a first step, we will try it with polynomial interpolation.

To simplify the procedure, let us take the energy matrix as an example. We propose to determine a polynomial P of 2 variables of degree deg having the following shape:

$$P(i_m, V) = \sum_{\substack{i,j=0 \\ i+j \leq deg}}^{deg} a_{ij} V^i i_m^j$$

where i_m and V refer respectively to the possible value of the current and velocity.

The goal will be to find the coefficients (a_{ij}) of the polynomial P that minimize the interpolation error. Thus, the interpolation problem can be formulated as follows:

$$\min_{a_{ij} \in \mathbb{R}} \sum_{(i,j) \in I \times J} \big(m_E(i,j) - P(i,j)\big)^2$$

where $I = \{-150, -150+p, -150+2\times p, \ldots, 150\}$ with $p = 30$ and $J = \{-10, -10+ q, -10+ 2 \times q, \ldots, 70\}$ with $q = 10$. $m_E(i,j)$ refers to the element of the energy matrix when applying an initial velocity equal to i and a reference current equal to j. It should be noticed also that the used matrices are computed over a sample of time of length $1s$.

The preliminary results show that this approach is ca be used in the case of the matrix

of position. In fact, we succeed to interpolate it with two polynomials: the first is of degree 1 corresponding to the linear part and the second one is of degree 2 which corresponds to the remaining part. The interpolation error is negligible.

$$P(i_m, V) = \begin{cases} 0.0077 + 0.2707V + 0.0164i_m & \text{if } (i_m, V) \in \Sigma \\ 0.0048 + 0.2670V + 0.0243i_m - 0.0002i_m V & \text{if } (i_m, V) \in (I \times J)\backslash\Sigma \end{cases}$$

where $\Sigma = I \times \{-10, 0, 10, 20, 30, 40\}$. When $(i_m, V) \in \Sigma$, the sum of the square of the absolute error is equal to 1.0012 and in the other case, it is about 3.709.

However, up to now, we are not able to do the same with the energy and velocity matrices which comes from their complicated behavior, see Figure 3.2 and 3.3.

Once this approach is accomplished even with other approximation functions, we will be able to extend our study from the discrete case to the continuous one and consequently, we will have better approximations of Problem (1.4).

Appendix B

Errors due to stalls

B.1 Introduction

In this appendix, we show that significant errors can occur when the matrices E, Pos and V are computed over a sample time of length $2s$ using those computed over a sample of length $1s$: i.e., considering the sample of length $2s$ as the "sum" of two samples, each one of length $1s$ can generate large errors.

B.2 Presentation of the problem

The approximation of the optimal control problem via the system of differential equations (1.3) is generated by subdividing the cycle of time $[0, t_f]$ into P subintervals. In each sample of ime $[t_{k-1}, t_k]$, we apply a reference current $iref$ which takes its values between -150 and +150. If the final time t_f is large, the subintervals may not have the same length and hence, this will require the computation of new matrices for each subinterval which is very expensive in terms of time and memory. In order to simplify the pre-processing for computing these matrices of energy, velocity and position, we just compute 3 matrices over a sample time of length $1s$ and reuse them to generate the other matrices over long samples using the following formulas:

$$E_{2s}(Ve, Iref) = E_{1s}(Ve, Iref) + E_{1s}(V_{1s}(Ve, Iref), Iref) \quad \text{(B.1)}$$

$$Pos_{2s}(Ve, Iref) = Pos_{1s}(Ve, Iref) + Pos_{1s}(V_{1s}(Ve, Iref), Iref) \quad \text{(B.2)}$$

$$V_{2s}(Ve, Iref) = V_{1s}(V_{1s}(Ve, Iref), Iref) \quad \text{(B.3)}$$

E_{2s}, Pos_{2s} and V_{2s} (resp. E_{1s}, Pos_{1s} and V_{1s}) refer respectively to the matrices of energy, position and velocity computed over a sample of length $2s$ (resp. $1s$). $Iref$ and Ve vary respectively from -150A to +150A and from $-74km/h$ to $71km/h$ with fixed steps.

B.3 Numerical results

Generating our matrices using the two methods, taking the case of energy matrix and by computing the relative error, we remark the presence of errors strongly dependent on the initial velocity and the reference current. These errors seem very important when applying high absolute values of the current and the velocity, see Figure B.1.

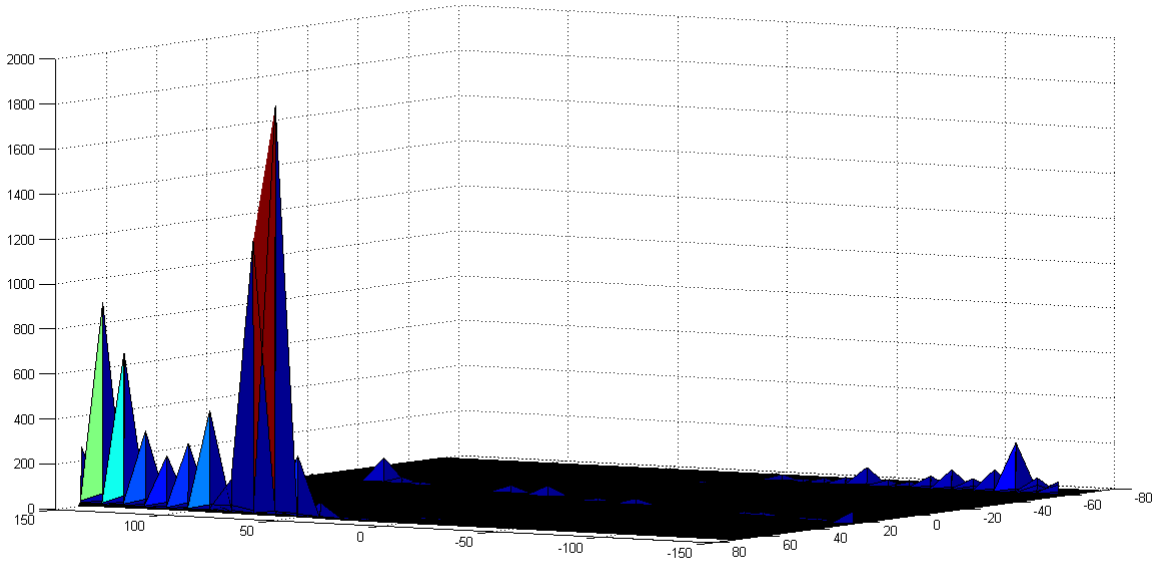

Figure B.1: Relative error on the energy matrix

To test the running of these methods, we take an element of each matrix corresponding to the same coordinates and we simulate it by drawing the curves of $im(t)$, $Pos(t)$, $V(t)$, $E(t)$ and $u(t)$ over a sample of length $2s$ using these two methods. We begin by taking the element of the matrix corresponding to a velocity of $70.9km/h$ and a current of 60A.

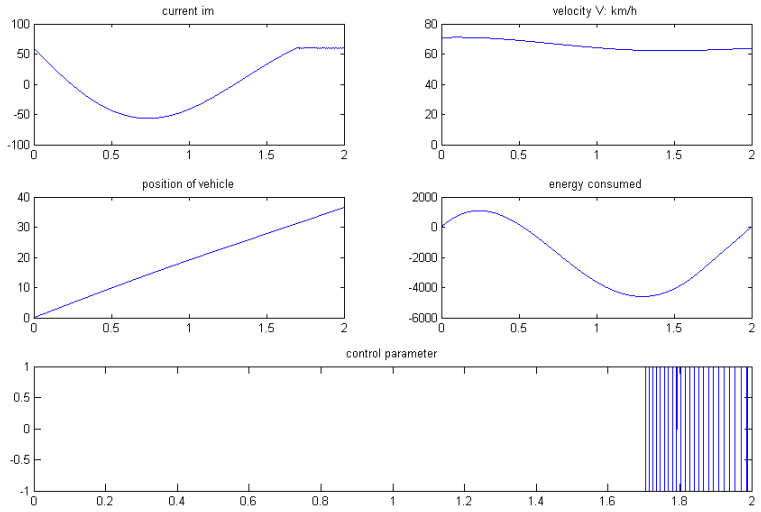

Figure B.2: Numerical simulation results of element ($70.9km/h$, 60A) over a sample of length $2s$ using "classical" method.

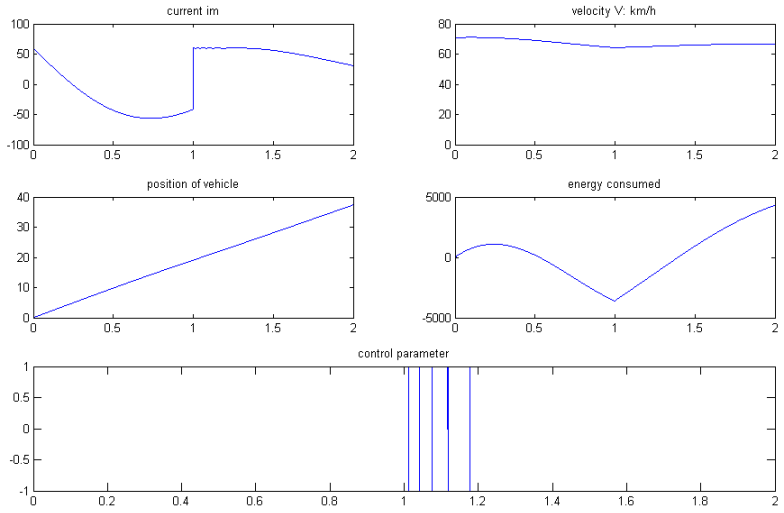

Figure B.3: Numerical simulation results of element ($70.9km/h$, 60A) over a sample of length $2s$ using formulas (B.1), (B.2) and (B.3).

On Figure B.3, the scheme representing the current i_m takes its maximum value at the first moments and its minimum one at the end of the first second. A sudden increase in the current i_m takes place at the beginning of the second second. This sudden change is

justified by the fact that i_m is not trapped around $iref$ with a tolerance Δ ($\Delta = 1$ in our case). In this case, using two time the sample of length $1s$ to obtain the matrices over a sample of length $2s$ leads to large errors: the computational error is about 1731.4% for $E(2s)$, 2.43% for $Pos(2s)$, 4.81% for $V(2s)$ and 48.30% for $im(2s)$.

Now, we take another element that belongs to the region where the error is small (see Figure B.1); it is the element corresponding to velocity of $20km/h$ and a reference current equal to 20A.

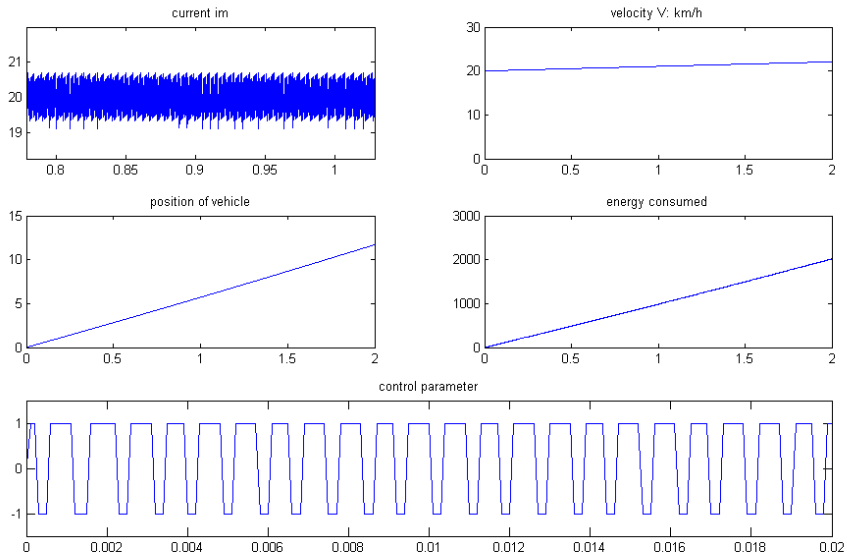

Figure B.4: Numerical simulation results of element (20 km/h, 20A) over a sample of length $2s$ using the two methods.

In this case, the two methods lead to the same result. It is clear that the current i_m remains trapped around the reference current $iref$. The values of control u switch intensively between -1 and +1 to maintain this trap. In Figure B.4, curves of the current and command u have been enlarged for visualization.

B.4 Conclusion

Our method consists in building a current regulator which allows to trap the current i_m around a given reference current $iref$. Nevertheless, we remark that there is some extreme cases where the current may not be controlled by our regulator. These extreme cases must be treated differently.

Appendix C

More numerical results

In Section 2.4, we just gave the numerical results corresponding to a displacement of $100m$ in a cycle of time $t_f = 10s$ with a free final velocity using only Heuristic $H4$. Indeed, this case illustrates very well the gain in CPU-time using Fortran 90. In this appendix, using the most reliable Heuristic $H6$, we provide the numerical results with the same previous instances ($Posf = 100m$, $t_f = 10s$) but with different constraints on the sate velocity and the final speed. For each case, the refined solution corresponding to the last row is simulated using the numerical integrator $RK4$ and then drawn.

$Instance$	$iref$	E_{min} (J)	$posf$ (m)	Vf km/h	CPU (s)	$porte$ $(amps)$	$Iter.$
$P = 5$ $s = 10$	(150, 110, 40, -70, -150)	26517	100.72	-1.17	10^{-2}	300	3 076
$P = 10$ $s = 10$	(150,150,130,90,30, 20,-50,-60,-150,-150)	25646	100.00	-0.89	0.4	40	95 722
$P = 10$ $s = 5$	(150,145,130,90,35, 20,-50,-65,-140,-150)	25199	100.00	-0.10	0.59	20	140 098
$P = 10$ $s = 1$	(150,146,130,88,36, 20,-51,-65,-138,-150)	25156	100.02	0.00	6.36	4	1 512 512

Table C.1: Null final velocity case.

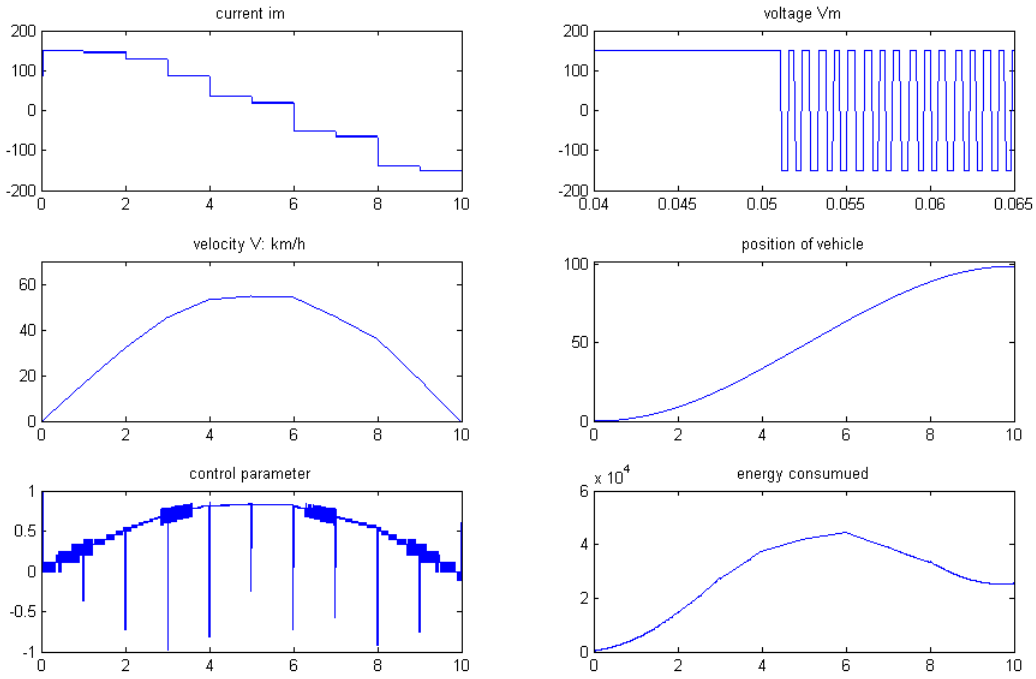

Figure C.1: $Case : P = 10;\ s = 1;\ posf = 100m;\ Vf = 0km/h$

Instance	*iref*	E_{min} (J)	*posf* (m)	*Vf* km/h	*CPU* (s)	*porte* $(amps)$	*Iter.*
$P = 5$ $s = 10$	(120, 60, 50, 30, 40)	43836	100.10	51.49	$2{\times}10^{-2}$	300	9 718
$P = 10$ $s = 10$	(120,100,80,70,50, 40,30,40,30,30)	42496	100.03	50.24	1.86	40	483 577
$P = 10$ $s = 5$	(120,95,85,65,55, 45,35,30,30,30)	42469	100.03	50.05	3.65	20	910 202
$P = 10$ $s = 1$	(121,96,83,63,54, 43,35,30,32,29)	42096	100.00	50.03	11.90	4	2 739 102

Table C.2: Constraints on the velocity state $(V(t) \leq 50km/h)$ and the final velocity $(Vf = 50km/h)$.

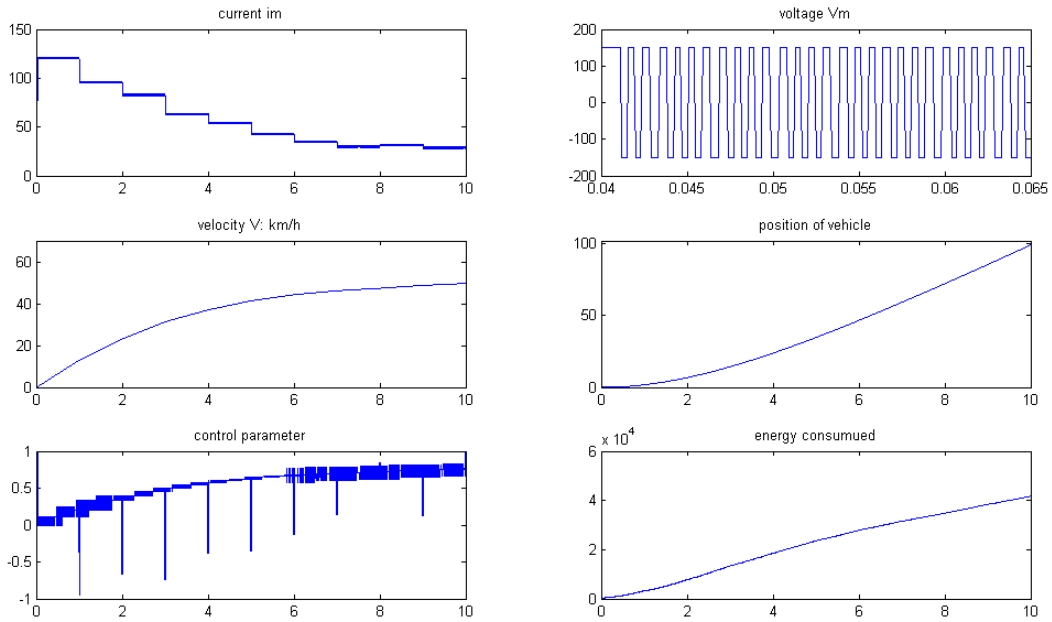

Figure C.2: $Case : P = 10;\ s = 1;\ posf = 100m;\ V(t) \leq 50km/h;\ Vf = 50km/h.$

$Instance$	$iref$	E_{min} (J)	$posf$ (m)	Vf km/h	CPU (s)	$porte$ $(amps)$	$Iter.$
$P = 5$ $s = 10$	(150, 80, 20, -30, -150)	21285	90.82	-1.65	10^{-2}	300	2 915
$P = 10$ $s = 10$	(150,140,100,60,30, 0,-20,-30,-130,-150)	20194	90.13	-0.10	0.4	40	97 912
$P = 10$ $s = 5$	(150,140,95,65,25, 10,-20,-40,-125,-150)	20044	90.15	0.00	1.43	20	369 436
$P = 10$ $s = 1$	(150,138,95,63,27, 11,-19,-38,-127,-150)	19950	90.01	0.00	4.97	4	1 719 702

Table C.3: Constraints on the velocity state $(V(t) \leq 50km/h)$ and the final velocity $(Vf = 0km/h)$.

In this case, the displacement is limited to $90m$. In fact, our simulations show that it is not possible to keep the same previous instances (described in Table C.2) with a null final velocity.

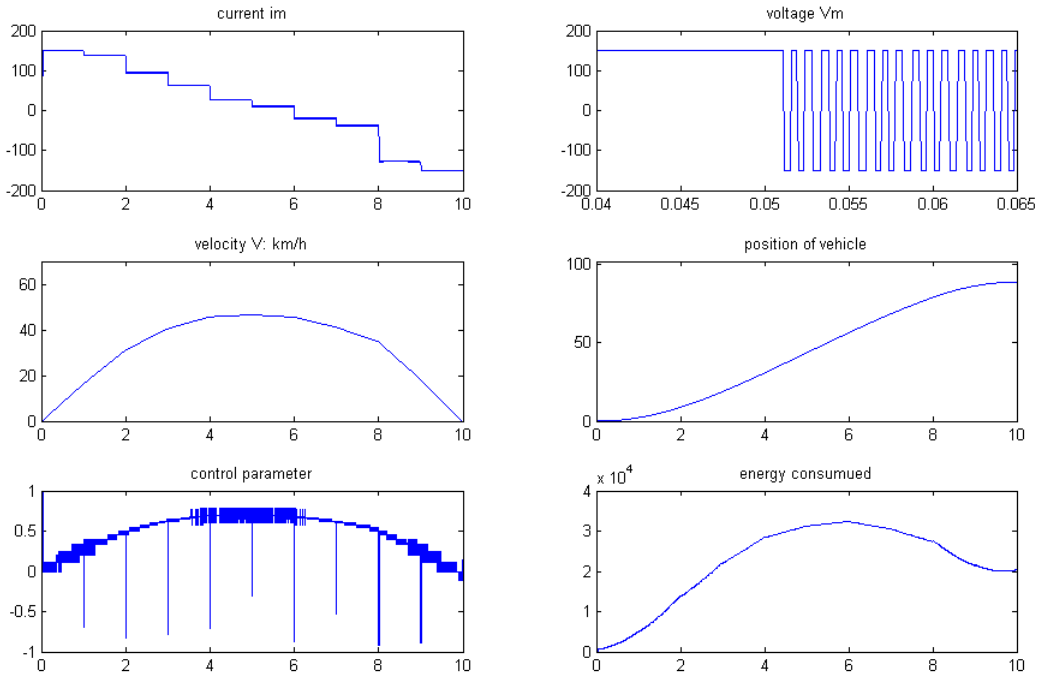

Figure C.3: $Case: P = 10;\ s = 1;\ posf = 90m;\ \ V(t) \leq 50km/h;\ Vf = 0km/h.$