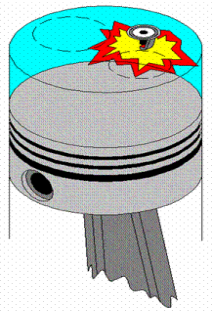
CZECH TECHNICAL UNIVERSITY IN PRAGUE

U 12120.3 - Josef Božek Research Center of Engine and Automotive Engineering

MATHIEU ALLORY (*mathieu.allory@eleves.ec-nantes.fr*)

AUGUST 2006

A COMPUTATIONAL CODE FOR EVALUATION OF PREMIXED FLAME PARAMETERS BASED ON A THREE-ZONES MODEL



RAPPORT DE TRAVAIL DE FIN D'ETUDES

ECOLE CENTRALE DE NANTES – OPTION INFORMATIQUE

THANKS

I would like to thank everybody at the Josef Božek Research Center for their great welcome, their kindness and their availability. Everyone did his best to help me settling in, which allowed me to work in good conditions.

Thanks to Pr. Ing. MACEK for giving me the chance to do my traineeship in his department, and also for putting me in charge of an interesting and exciting subject. He made himself available every time I needed to explain the subject and to follow the advance of the project.

Thanks to Dr. ACHTENOVA for helping me much with practical details at my arrival, especially for my accommodation and my student card.

Thanks to Ing. MIKULEC for his numerous and useful advices.

RÉSUMÉ

Mon travail de fin d'études (TFE), effectué au sein du laboratoire Joseph Bozek de l'Université Technique de Prague mêlait à la fois l'informatique, la mécanique, l'énergétique et les mathématiques.

Mon maître de stage, le Professeur Jan Macek, a mis au point une nouvelle méthode permettant d'évaluer les paramètres de flammes au sein d'un cylindre de moteur à combustion interne, et plus généralement d'y simuler une combustion, avec l'espoir qu'elle soit plus fiable et précise que les techniques utilisées actuellement. Jusqu'alors, les modèles de calculs considèrent souvent les grandeurs physiques (température, pression) comme homogènes au sein du cylindre. Or, depuis peu, de nouveaux moyens de mesure permettent de connaître avec précision la position du front de flamme et son déplacement durant le cycle. L'idée de cette nouvelle méthode est donc de découper le cylindre en trois zones : gaz neufs, front de flamme et gaz brûlés. Dès lors il devient possible, à l'aide des lois de la thermodynamique, de modéliser les échanges entre ces zones ainsi que les réactions chimiques, et de connaître les paramètres d'état qui en découlent.

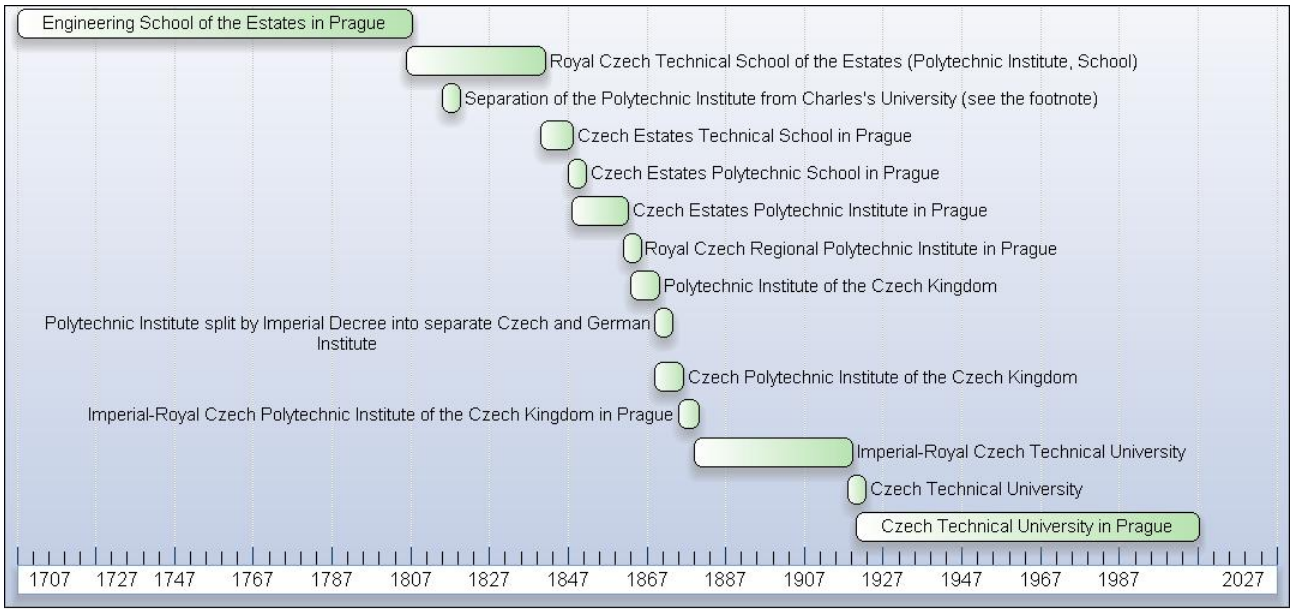
Cette approche trois zones, valide en théorie, n'a pour lors jamais été mise à l'épreuve et pour cause : il n'existait aucun code de calcul traduisant ces équations. Il m'a donc été demandé de mettre au point celui-ci, ou du moins une première ébauche, afin non pas d'avoir des résultats fiables de suite, mais surtout d'être certain que cette approche mérite d'être approfondie. Qui plus est, le code devait être clair, modulaire, facilement modifiable et évolutif.
Le langage Fortran 77, que j'ai appris durant les premières semaines de mon stage, bien que d'un âge avancé, permettait au code d'être compris facilement et rapidement par mes collègues, et surtout rendait possible une intégration facile du modèle avec les autres codes du laboratoire, eux aussi écrits en Fortran.
En plus du modèle lui même, il m'a été demandé d'écrire des codes périphériques : un modèle géométrique (qui peut sembler simple mais dont la transcription informatique est en fait assez complexe), un modèle des conditions initiales, des solveurs et autres outils mathématiques.

Les premiers résultats, bien que loin encore de permettre une exploitation du modèle en production, sont encourageants : la tendance des courbes s'approche de plus en plus de la réalité, et même certains résultats numériques sont plausibles. Mon travail a donc prouvé que cette méthode a sans aucun doute un avenir, et le code, une fois calibré et purgé des quelques erreurs qui pourraient encore y résider, pourra bien être utilisé en simulation.

Part I

A short presentation of the Czech Technical University in Prague

Milestones of the History of Czech Technical University



Milestones of the history of CTU

Footnote : Emperor Joseph II by a decree of the Study Commission of the Court ultimately separated the Polytechnic Institute from Charles's University to which the Engineering School of the Estates was affiliated from 1787 by an Imperial Decree as one of its departments. It remained to be a department of the Faculty of Philosophy even after its transformation into a polytechnic in 1806, however only formally. The debate on whether this incorporation should continue or not reached its climax in 1814-1815. It ended with the complete separation of the School of the Estates in Prague.

CZECH TECHNICAL UNIVERSITY NOWADAYS

- **The aftermath of the Velvet revolution**

The Czech technical University in Prague started a new era of its existence after the "velvet revolution" in November 17, 1989 which ended the 40 years long communist totalitarian regime. It then started embarking on the gradual process of its transformation in the spirit of democratic traditions and in harmony with the demands contemporary technology and progressive development made on technical intelligentsia in all fields cultivated within the university, both old and new.

- **The faculties**

Since 1993, when the Faculty of Transportation was established, the CTU comprises 6 faculties: Civil Engineering, Mechanical Engineering, Electrical Engineering, Nuclear and Physical Engineering and Transportation. It also comprises all-university departments: the Masaryk Institute of Higher Studies, Klokner Institute, Computer and Information Center, Entrepreneurial and Innovation Center, Administration of Operating Departments, CTU Publishing House and Institute of Biomedical Engineering.
New branches of the Faculty of Transportation and the Faculty of Nuclear and Physical Engineering were opened in Děčín in 1995. The Research Center of Industrial Heritage and the Institute of Technical and Experimental Physics of the CTU were established in 2002. The Center for Radiochemistry and Radiation Chemistry was established in 2003.
Currently the University employs 3300 people and has around 23000 students.

THE FACULTY OF MECHANICAL ENGINEERING

- **History and purposes**

The Faculty of Mechanical Engineering of the CTU in Prague is a constituent of the oldest civil technical university in Central Europe established in 1707. Mechanical engineering as an independent field of study began to be taught at this school in 1864 and consequently this year is considered as the date of the establishment of the Faculty of Mechanical Engineering. The contemporary Faculty of Mechanical Engineering of the CTU in Prague provides technical education on a university level and educates specialists in many fields of mechanical engineering. The aim of the contemporary faculty is to be a peak educational and research establishment recognized both in the Czech Republic and abroad.

- **The study programs**

The Bachelor study program of the faculty offers the following branches: Transportation and Handling technology, Information and Automation Technology, Environmental Engineering, Thermal Power and Process Engineering, Production Engineering, Applied Mechanics for Bachelors, Manufacturing Technology and Management.
The Master study program and Master study program for Bachelor graduates of the faculty offer the following branches : Environmental Engineering, Production Engineering, Power Engineering, Instrumentation and Control Engineering, Transportation and Handling equipment, Engineering Mechanics and Mechatronics, Process Engineering, Production Machines and Equipment, Aerospace Engineering, Enterprise Management and Economics, Biomedical and Rehabilitation Engineering, Materials Engineering and Mathematical Modeling in Engineering.

- **National and international cooperation**

The faculty is engaged in international educational cooperation in many spheres. Besides the mobility of students and the academic staff both executed on a contractual basis concerned are namely scientific, research and educational activities including participation in scientific and technical seminars and conferences namely in the EU. International cooperation in science and research is implemented by direct involvement in particular EU or other international programs. Moreover, it is involved in the process of harmonization of the European Educational and Research Area in order to be fully compatible with the European system and also to be attractive to its students and to comply with demands of the society.
The faculty is also engaged in the work of other research centers. It cooperates with the Academy of Sciences of the Czech Republic, Engineering Academy of the Czech Republic, Association of Research Institutes, Association of Industry and Transport, Association of Producers of Mechanical

Engineering Technology and with a number of large, medium and small industrial companies, like Siemens, Porsche, Škoda Auto, Volkswagen, Hydrosystem Olomouc etc.

- **Resources**

The premises of the faculty are in four different localities in Prague: Prague 6-Dejvice (where my office was), the ancient faculty building on Charles'Square and buildings on Horska Street and Juliska. Since the academic year 2003/2004 a new detached consulting and educational center was opened at Sezimovo Usti. The faculty also has a number of other educational facilities.
Presently 32 professors, 79 senior lecturers, and 175 teaching assistants, who are also involved in numerous research and development projects, work in 14 departments and 2 research centers.

THE JOSEF BOŽEK RESEARCH CENTER OF ENGINE AND AUTOMOTIVE ENGINEERING

- **The organizational structure**

The Josef Božek Research Center based at the Czech Technical University in Prague has acquired high international reputation as the country's leading research body focused on automotive technology. The Center links research workers and postgraduate students of the following institutions:

- Faculty of Mechanical Engineering & Faculty of Electrical Engineering, Czech Technical University in Prague
- Faculty of Mechanical Engineering, Technical University in Liberec
- Faculty of Mechanical Engineering, Technical University in Brno
- Faculty of Mechanical Engineering, Mining School - Technical University in Ostrava
- Ricardo Prague
- TÜV – UVMV

- **Research purposes**

The main focus of the Research Center is research and development of spark ignition engines (gasoline, gas, alternative fuels) and diesel engines for cars and heavy-duty vehicles (this is what I was involved in). The engine research is focused on thermodynamics, internal flow aerodynamics, turbocharging and supercharging of engines using conventional and emerging technologies, emission reduction and post-treatment, engine management by intelligent controllers, engine dynamics and structural strength of components applied to the design optimization.
The Center further provides R&D results for vehicle transmission design and powertrain optimization (mechanical, hydraulic, electrical powertrain), vehicle suspension design (including active mechatronic elements and their control), body aerodynamics and passive safety issues.

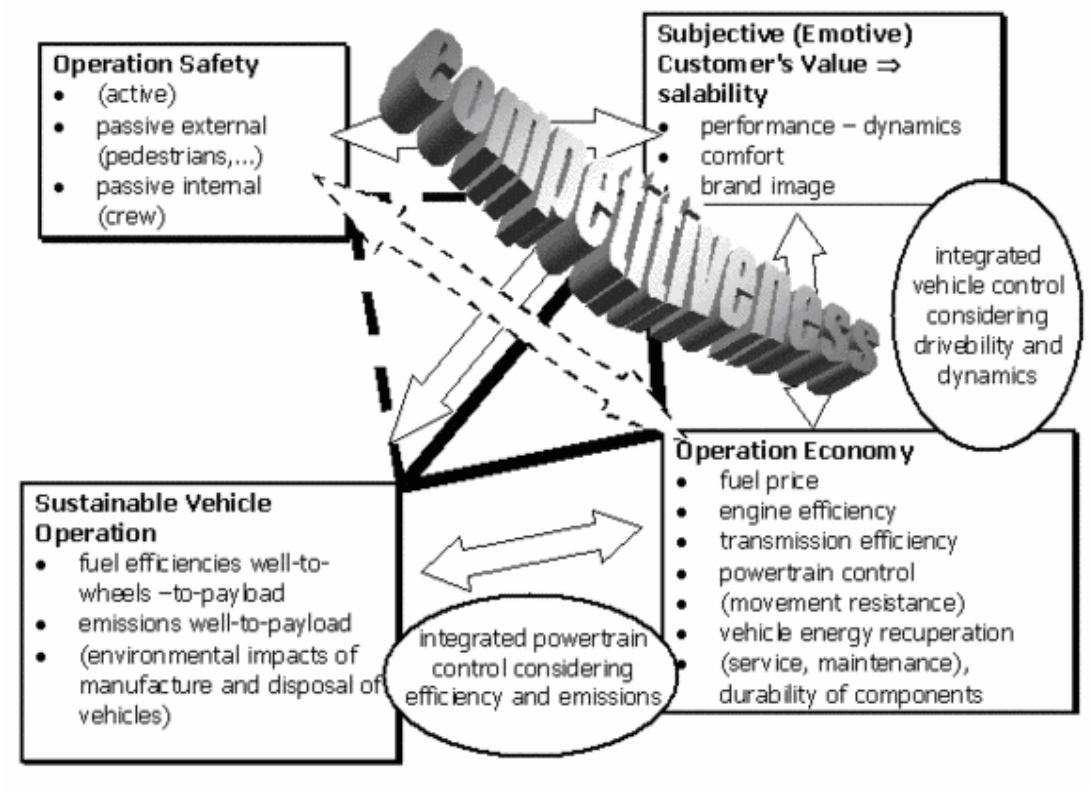


Diagram of Josef Božek Research Centre research activities

The Center will be active in finding tools and solutions for sustainable mobility by means of road vehicles, and of sustainable energy resources for transport and de-centralised power supply, based on internal combustion engines. The conditions of increased use of alternative fuels based on renewable resources will be investigated. The activity of the Center also covers the applied research of the automotive electronics for vehicle systems control and the introduction of electric power components, especially for hybrid and fuel cell vehicles.

The research are focused on the following domains:

- the concepts of powertrains and their components, especially those of internal combustion engines (conceptual optimization), including control systems and utilization of electric transmissions.
- the detailed optimization of chosen concepts mentioned above (parametrical optimization), which is necessary for the determination of innovative potential of different concepts.
- the research of simulation tools based on calibrated models (algorithms, databases), the means of experimental research and the evaluation of its results using inverse algorithms.
- the research of algorithms, processes and electronic hardware of the automatic control of vehicles, both as entities and as subsystems – engines, drivelines and chassis, based on predictive concepts. This is the domain I was involved in.
- automotive electronics – the applied research of distributed automotive systems.

- **Publications and partnerships**

The Josef Božek Research Center regularly publishes its results at world congresses such as SAE World Congress in Detroit and in the in-house published, international peer-reviewed MECCA journal (Middle European Journal for Construction of Cars), which is where the theoretical basis of my work had been published.
The Center is an official partner of Gamma Technologies, Inc., the leader in specialized engine simulation software. The Center is linked to the automotive and engine industry by its Research Board consisting of representatives of OEMs and component suppliers, as well as other research and development facilities (e.g. Mercedes-Benz Engineering, Škoda Auto, ČZ Strakonice and others). The Center is involved as a partner in several European integrated projects of EU FP 6 (NICE, GREEN and Roads2HYCOM). It is also a member of European Automotive Research Partners Association (EARPA).
Apart from R&D activities, the Center conducts the Master in Automotive Engineering, a two-year international course taught in cooperation with ENSIETA Brest, France, and HAN University, Arnhem, The Netherlands.

Part II
Report

TABLE OF CONTENTS

I. PRESENTATION OF THE SUBJECT

1. From a simple model to a more complicated one

For a long time, maybe since computers exist, people tried to design models describing more and more precisely the way car engines work. However, significant changes appeared in the last few years bringing more knowledges in what is happening inside the combustion chamber.

Till now, models had only a few data sources to work with, mainly pressure and temperature inside the combustion chamber, heat transfer to walls and known geometrical data. All those ones are and were easily measurable, that is why they have been used for a long time, combined with assumptions on state equations, to estimate rate of heat release (ROHR). This traditional model, known as 'one-zone model', is not really precise as it assumes that data are uniform inside the cylinder.

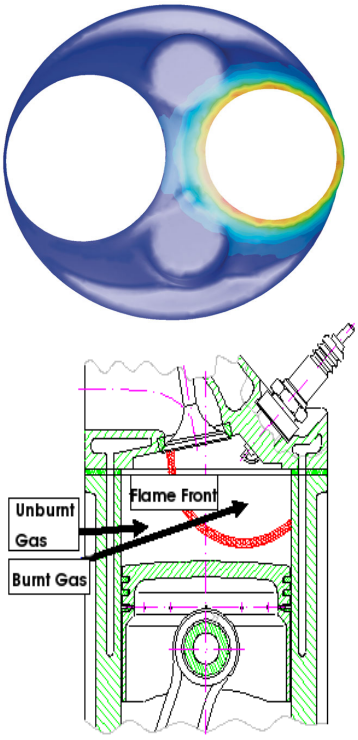


Figure 1: The chamber split in three zones & Real zones in a Daimler engine

Fortunately, another data source is now available, due to the improvement of optical measurement capabilities. We are now able to know precisely the evolution of the flame front, i.e. Flame front position (width) and speed *(Figure 1, top).* As a major consequence, some multi-zone models have been imagined.

The one I had to write the code for is one of them, designed by Pr. Jan MACEK. More precisely, it is a three-zones model, involving not only a unburnt gas zone and a burnt gas zone, which is quite usual, but also a flame front zone which is not considered just as an interface but as a true zone with a considerable thickness and three interfaces (front, rear and walls).

The theoretical work had almost been entirely done when I started to build the code, except boundary conditions of integration (in time, not in space), which we will talk about later (see *The initial model*).

However, instead of using geometrical data (chamber and zones volumes) from measurement, it has been asked to build a geometrical model from scratch, which would be a simple cylinder with a centered spark plug. This aims at working on a geometry which would be simple enough to test and debug the computational code itself without taking in consideration a real -and complicated- flame front shape.

2. Theoretical set of equations

The evolution of combustion inside the chamber is described by a set of equations mainly based on mass conservation and state equations. We are not going to view the whole process of establishing those equations here, so please study Pr. Jan Macek's article "Zone Approach Used For Determination of Premixed Flame Parameters" to learn more about them.

A) Assumptions

Before exposing the equations themselves, we have to formulate a few assumptions, in order to simplify the code. In fact most of them could be overcame, even quite easily by rewriting parts of the code.

- We simulate a petrol engine. For a diesel engine, equations are slightly different.
- The reactions involve 5 species, which are : Dioxygen, Fuel, Carbon dioxide, Steam water and the couple (Nitrogen, Argon) considered as one specie as it does not react.
- Pressure is considered homogeneous inside the cylinder.
- Temperatures are considered homogeneous inside a zone.
- International metric units will be used everywhere.
- The state equation used is the perfect gas law $P \cdot V = \{{}_5m\}^T \cdot \{{}_5r\} \cdot T$.
- Zones are numbered the following way : $\begin{Bmatrix} Zone\,I : Unburnt\ gas \\ Zone\,II : Flame\ front \\ Zone\,III : Burnt\ gas \end{Bmatrix}$
- The whole mixture is already inside the cylinder when ignition occurs, i.e. when computation starts (it wouldn't be interesting to start integration before ignition).

Moreover, the following assumptions, which are really much more parameters for the code, have been done :

- Propane will stand as the fuel used : $C_3H_8, \begin{pmatrix} x=3 \\ y=8 \end{pmatrix}$
- Only the standard burning reaction occurs : $C_xH_y + (x + \frac{y}{4})O_2 \rightarrow xCO_2 + (\frac{x}{2})H_2O$. It would be very easy to implement other reactions by modifying the reaction matrix. Corrective terms in equation (1), (2) and (3) should also be taken in consideration, which is not the case now. Once again, it is not difficult to do.
- No heat transfer occurs between the zones and the walls. This is just a coefficient to change to enable it.

B) Relative flame velocity

There is a tricky thing here : those velocities do not represent flame front growth speed, which is an input (from a geometrical model or from optical measurements). In fact it represents how fast the flame front consumes unburnt gas ($w_{f,front}$) and releases burnt gas ($w_{f,back}$).

$$w_{f,front} = \frac{V_I}{\left(\frac{K}{K-1}\right)_I T_I A_{f,I,II} \{r\}^T \{m_I\}} \cdot \begin{bmatrix} -\dot{Q}_{I,walls} - \left[(c_p m_I)\frac{T_I}{p} - V_I\right]\frac{dp}{dt} - p\left(\frac{K}{K-1}\right)_I \\ -\left[\{\boldsymbol{i_I}\}^T - \{\boldsymbol{H_u}\}^T - \left(\frac{\boldsymbol{K}}{\boldsymbol{K-1}}\right)_I \boldsymbol{T_I}\{\boldsymbol{r}\}^T\right]\{\dot{\boldsymbol{r}}_{\boldsymbol{I,corr}}\} \end{bmatrix} \quad (1)$$

$$w_{f,back} = \frac{V_{II}}{\left[\{i_{III}\}^T - \{i_{II}\}^T - \left(\frac{K}{K-1}\right)_{III} T_{III}\{r\}^T\right] A_{f,II,III}\left(\{m_{II}\} - \{m_{II,fuel}\}_{O_2}\right)} \begin{bmatrix} -\dot{Q}_{III,walls} - \left[(c_p m_{III})\frac{T_{III}}{p} - V_{III}\right]\frac{dp}{dt} - p\left(\frac{K}{K-1}\right)_{III} \\ -\left[\{\boldsymbol{i_{III}}\}^T - \{\boldsymbol{H_u}\}^T - \left(\frac{\boldsymbol{K}}{\boldsymbol{K-1}}\right)_{III} \boldsymbol{T_{III}}\{\boldsymbol{r}\}^T\right]\{\dot{\boldsymbol{r}}_{\boldsymbol{III,corr}}\} \end{bmatrix} \quad (2)$$

Notice that corrective terms (in bold) are not implemented yet. For explanations about notations, see *Appendix 1*.

C) Rate of burning

The following term is not exactly rate of burning : to obtain ROB, the corrective term (in bold) should be removed. As it is not implemented in this version of the code, we can thus consider $\dot{r}_{II,fuel}$ as the rate of burning (in $kg.s^{-1}$).

$$\dot{r}_{II,fuel} = \frac{1}{\left[\{i_{II}\}^T - \{H_u\}^T - \left(\frac{K}{K-1}\right)_{II} T_{II}\{r\}^T\right]\{_5 C_B\}} \begin{bmatrix} -\dot{Q}_{II,walls} - \left[\{\boldsymbol{i_{II}}\}^T - \{\boldsymbol{H_u}\}^T - \left(\frac{\boldsymbol{K}}{\boldsymbol{K-1}}\right)_{II} \boldsymbol{T_{II}}\{\boldsymbol{r}\}^T\right]\{\dot{\boldsymbol{r}}_{\boldsymbol{II,corr}}\} \\ -\left[(c_p m_{II})\frac{T_{II}}{p} - V_{II}\right]\frac{dp}{dt} - p\left(\frac{K}{K-1}\right)_{II} \\ -\left[\{i_{II}\}^T - \{i_I\}^T - \left(\frac{K}{K-1}\right)_I T_I\{r\}^T\right] A_{f,I,II}\frac{\{m_I\}}{V_I} w_{f,front} \\ -\left(\frac{K}{K-1}\right)_{II} T_{II}\{r\}^T A_{f,II,III}\frac{\{m_{II}\} - \{m_{II,fuel}\}_{O_2}}{V_{II}} w_{f,back} \end{bmatrix} \quad (3)$$

D) Mass transfer between zones

From the ROB and flame velocities we can define mass transfer through the interfaces between zone (I and II) and (II and III). Of course, no mass transfer between the zones and the walls happens, as we consider all mixture is already inside the cylinder when ignition occurs.

$$\{\dot{m}_{I,II}\} = \frac{A_{f,I,II}\, w_{f,front}}{V_I}\{m_I\} \quad (4)$$

$$\{\dot{m}_{II,III}\} = \frac{A_{f,II,III}\, w_{f,back}}{V_{II}}\left(\{m_{II}\} - \{m_{II,fuel}\}_{O_2}\right) \quad (5)$$

(4) models the advance of the border between flame front and unburnt gas : unburnt gas pass from zone I to the flame front to be consumed.

(5) models how residues of combustion (i.e. CO_2 and H_2O) are expelled from zone II to zone III when the flame front advances.

$$\left\{\dot{r}_{II}\right\} = \left\{{}_5C_B\right\} r_{II,fuel}^{\cdot} + \left\{r_{II,Corr}^{\cdot}\right\} \quad (6)$$

(6) models chemical reactions in zone II (flame front). In our case, only the burning reaction is taken in consideration (which means $\left\{r_{II,Corr}^{\cdot}\right\}$ is null).

$$\left\{r_{II,Corr}^{\cdot}\right\} = \left\|{}_{5*r-1}C_{corr}\right\| \left\{{}_{r-1}\dot{r}_{II,corr}\right\} \quad (7)$$

(7) allows including non burning reactions. $\left({}_{5*r-1}C_{corr}\right)$ is the reaction matrix without the column referring to the combustion reaction (so that if the dimension of reaction matrix is (5,r) the dimension of this matrix is (5,r-1)).

E) Mass conservation

Using the previous equations, it is now possible to write the main differential system based on mass conservation principle.

$$\begin{aligned} \frac{d\left\{m_I\right\}}{dt} &= -\left\{\dot{m}_{I,II}\right\} \\ \frac{d\left\{m_{II}\right\}}{dt} &= \left\{\dot{m}_{I,II}\right\} - \left\{\dot{m}_{II,III}^{\cdot}\right\} + \left\{\dot{r}_{II}\right\} \\ \frac{d\left\{m_{III}\right\}}{dt} &= \left\{\dot{m}_{II,III}^{\cdot}\right\} \end{aligned} \quad (8)$$

The unknowns are the vectors $\left\{m_I\right\}, \left\{m_{II}\right\}, \left\{m_{III}\right\}$, which represent 15 time-dependent scalar functions. Once all equations are compiled, a classical set of differential equations is obtained, under the form $\frac{d\left\{{}_5m_i(t)\right\}}{dt} = F\left(\left\{{}_5m_i(t)\right\}, t\right) \quad i \in (I, II, III) \quad t \geq t_{start\ of\ ignition}$. The purpose of the computational code will be to solve it.

3. Implementation

A) The choice of Fortran

The first step of my work was to choose a programming language to implement the code, into a set of three : Matlab, VBE or Fortran. VBE was off-side from the beginning as I am developing under a Linux environment. Moreover it is very slow and has an awfully dirty syntax. About Matlab, first I don't really like it and moreover it would have been more complicated to interface with existing computational codes.

Even if it is sometimes considered obsolete, Fortran still has many advantages for numerical computation : it is very fast, has powerful built-in functions and an impressive number of reliable libraries. It can easily be compiled with numerous compilers under numerous operating systems for numerous platforms. It has been chosen to use only the Fortran 77 specifications to ensure a maximum compatibility with existing codes and to limit errors that can be done when using pointers. However we will see it is still possible to produce a lot of bugs and mistakes, which are often hard to track.

I also had to write a bunch of C routines to bypass some limits of Fortran, especially concerning the maximum length of records. The system is not able to write more than 73 characters per line in a file, which isn't enough in our case. So that outputs to files are done thought a C library instead of using 'WRITE(1,*) Vector' for example.

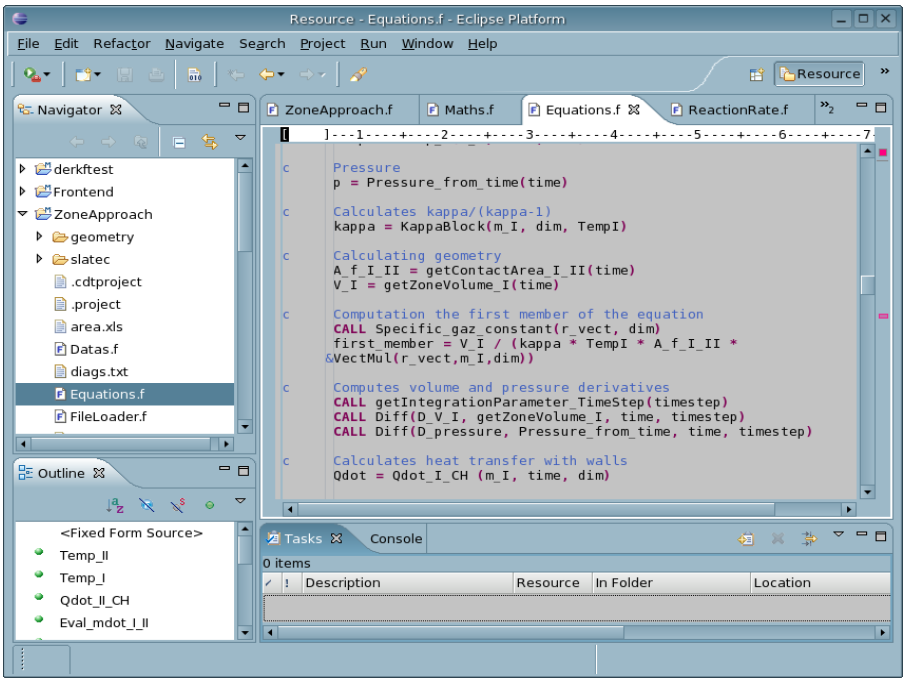


Figure 2: Photran IDE main screen

The whole development has been done using Eclipse Photran, a free IDE for Fortran and C programming, based on the well-known Eclipse Project (*see snapshot, Figure 3*).

The compiler used was the **GNU Fortran (GCC) 3.4.6**, a free compiler recognized as one of the most fast and reliable. The entire development has been done under Linux on a PC. However, I compiled the code under Windows without any problem using the Win32 version of the above compiler. Moreover, as the standard F77 specifications have been respected, it should be possible to use another compiler. In this case, the interface with IO library which is written in C should be carefully watched out.

B) Rules of programming and common mistakes

The make the software clearer and less buggy, I asserted a few rules :

- To make the code as modular as possible. This has been explicitly asked by Pr. Jan MACEK, in order to make any future change easier. Of course global efficiency is affected, but it really helps making the code clearer.
- Not to use COMMON. In fact I cheated one time in the geometrical model, but when this one will be replaced by measurement there won't be any COMMON anymore.
- Always to initialize variables at the beginning of a routine, in order to avoid segmentation faults. For this purpose, the assumption IMPLICIT NONE will always be used.

Although I tried not to do mistakes while programming, it is very easy to write buggy things in Fortran. Indeed, the compiler does not check many things as the language specifications are quite permissive. The grammatical analysis is very basic, it does not detect even obvious bugs.

- Typing mistake : Don't forget the '.' after a numeric if the assigned variable is REAL.

You write	*Result*
REAL A A = 10 WRITE(6,*) A	0.
REAL A A = 10. WRITE(6,*) A	10.

- Never forget to initialize your variables.

You write	*Result*
REAL A WRITE(6,*) A	Segmentation fault (during execution, lost in thousands of lines of code !)
REAL A A = 10. WRITE(6,*) A	10.

Those examples are just a few of the dozen pervert errors I encountered. The major problem is that they occur during execution. In the best case, the program just prints 'Segmentation Fault' on screen an then exits, but it often just returns wrong results without a word. To analyze buggy routines in order to locate problems, I used GDB, a command line debugger included inside the GCC suite.

II. THE COMPUTATIONAL CODE

1. Overall diagram

The way the whole system works is quite complicated : not only the core system but also a geometrical model, a first-step model, mathematical and IO routines have been written. So that it is useful to represent the whole program by an overall diagram.

Some parts of the code, especially the geometrical model, should be replaced in the future. To ensure compatibility, some routines we will call “interfaces” ought to be present.

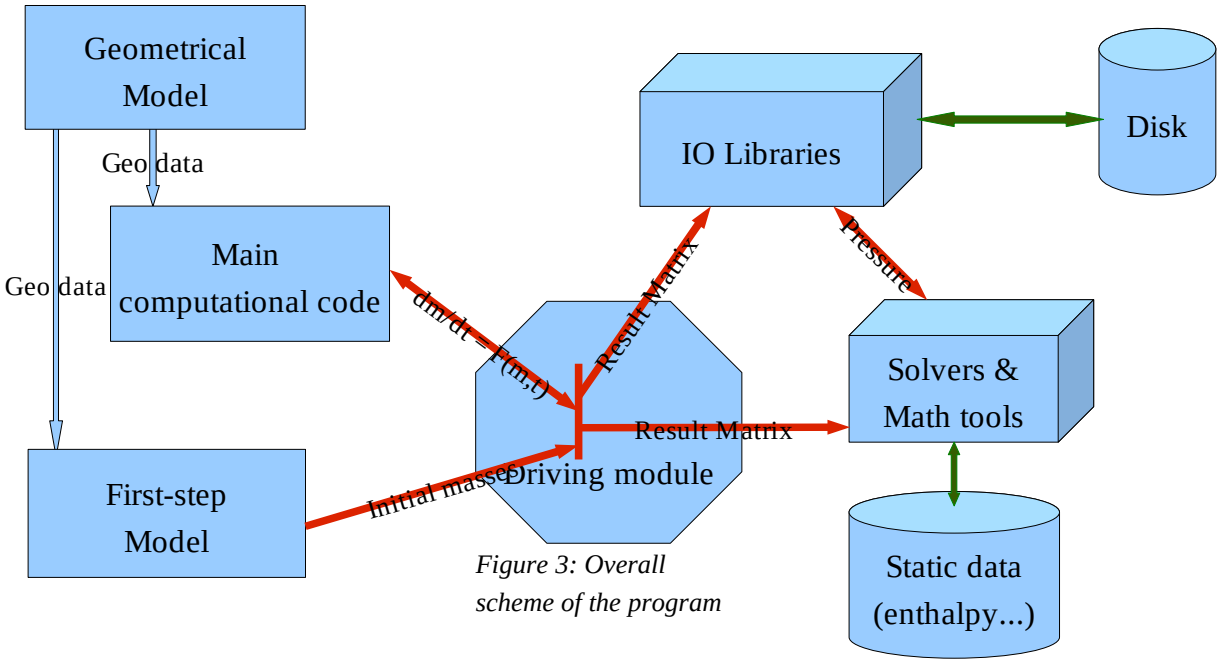


Figure 3: Overall scheme of the program

The scheme upon *(Figure 3)* is quite simplified, as it does not show the names of the functions. A complete list describing each file per module is available at *Appendix 2*.
Some connections between the First-step model, the Main code and the Geometrical model are missing : those three modules use math tools such as differentiation or linear approximation, in particular to access to static data such as enthalpy.

We will now examine every module separately, but keep in mind that each of them is highly dependent to each others, except the geometrical model that could be compiled separately. As a consequence, every change in another part of the code have to be done carefully, even if the large use of subroutines should make the things easier.

2. The geometrical model

A) Introduction

This part of the software, as is it designed only for test purposes, wasn't supposed to take me much time to program. However, I was still a beginner with Fortran, and it took me a couple of month to code it.

Although the geometry is pretty simple, it hasn't been so easy to translate it to Fortran, mainly because all the possible interactions between the zones, the walls, and the piston (whose movement has to be calculated too) had to be imagined and coded. Moreover, I focused on writing a code as modular as possible, as Prof. MACEK requested. And then the source code, which I won't explain here, can be seen as quite complicated for such a simple geometry.

Nevertheless, this part of the code is in my opinion reliable enough to be trusted and should not need to many patches in the future.

B) A very simple dummy geometry

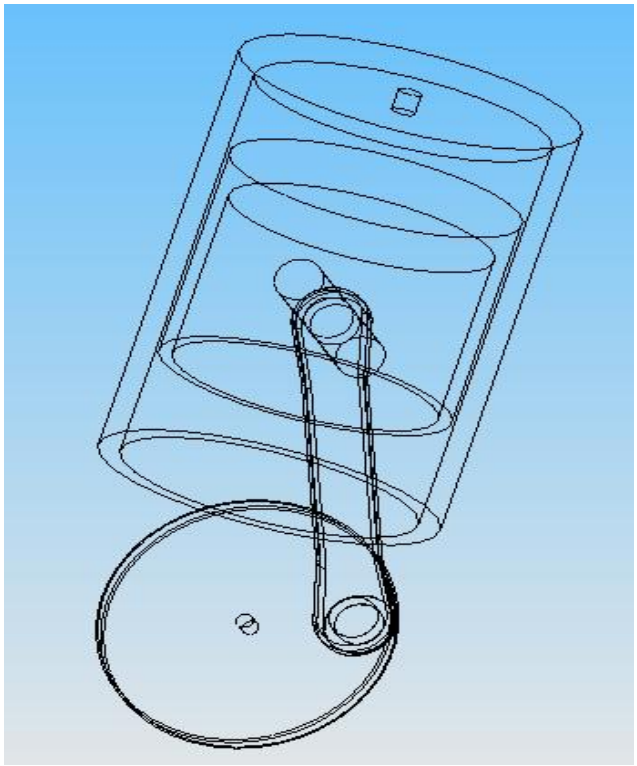


Figure 4: 3D view of the emulated geometry

This geometrical model is a very simple one, which is far from reality. It only aims at testing, debugging and calibrating the main computational code. However, even a real geometry module using measurement instead of emulated calculations should be able to return the same data as this simple one does.

It consists in a simple regular cylinder to represent the combustion chamber, and another one standing for the piston. The spark plug is at the top center of the chamber. Neither the valves nor the injector, nor the chamber or piston irregularities are taken in consideration.

Figures 4, 5, 6 and 7 should help in representing what this code emulates.

C) Configuring the model

First of all, the model has to be configured (i.e. configuring some geometrical properties like for example the length of the conrod) to be as close as possible to the real engine that has to be simulated. The process has been simplified since the first version of the model and is no more like explained in my preliminary report entitled "DUMMY GEOMETRY FOR PREMIXED FLAME PARAMETERS EVALUATION CODE".

Something else has to be taken in consideration : the angle of reference I used is not the most common one : **Top Dead Center occurs when crank angle is 360°** (not 0° like usually) so that the point of reference is the beginning of a new cycle, which stands from 0° to 719°. Data referring to crank angle (such as pressure) might be referenced that way.

Parameters are still hard-coded constants, in order to speed up the program, avoiding a disk access operation each time the library is called (which occurs a huge number of times). However, all the constants have been put together inside a single routine, so that it could be replaced by a reading operation or a variation law very easily. The new subroutine would only have to respect the existing prototype. Of course, each time something (including the following parameters) is changed inside the source code, the geometrical library has to be recompiled.

This routine is the first one situated in the file `GeoDatas.f` and has the following prototype.

```
SUBROUTINE GeoDatas(CylinderRadius, CylinderHeight, CrankRadius, ConrodLength,
flameFrontWidth, radius_expension_coef)
```

The default values are set to reproduce a Skoda 781.135 Z92-15 engine, that can be found on the Skoda 135 Favorit (1992).

Parameters' meaning :

Type-Unit	*Variable*	*Formal Parameter*	*Default Value*	*See figure*
REAL (m)	`CrankRadius`	Crank Radius	36.10^{-3}	5
REAL (m)	`ConrodLength`	Rod Size	130.10^{-3}	5
REAL (m)	`CylinderHeight`	Maximum height of the chamber (at bottom head center)	$81,6.10^{-3}$	7
REAL (m)	`CylinderRadius`	Chamber radius	$37,75.10^{-3}$	5
REAL (m)	`flameFrontWidth`	Width of the flame front	0.003	6
REAL (m/s)	`radius_expension_coef`	Speed of the flame front	25	6

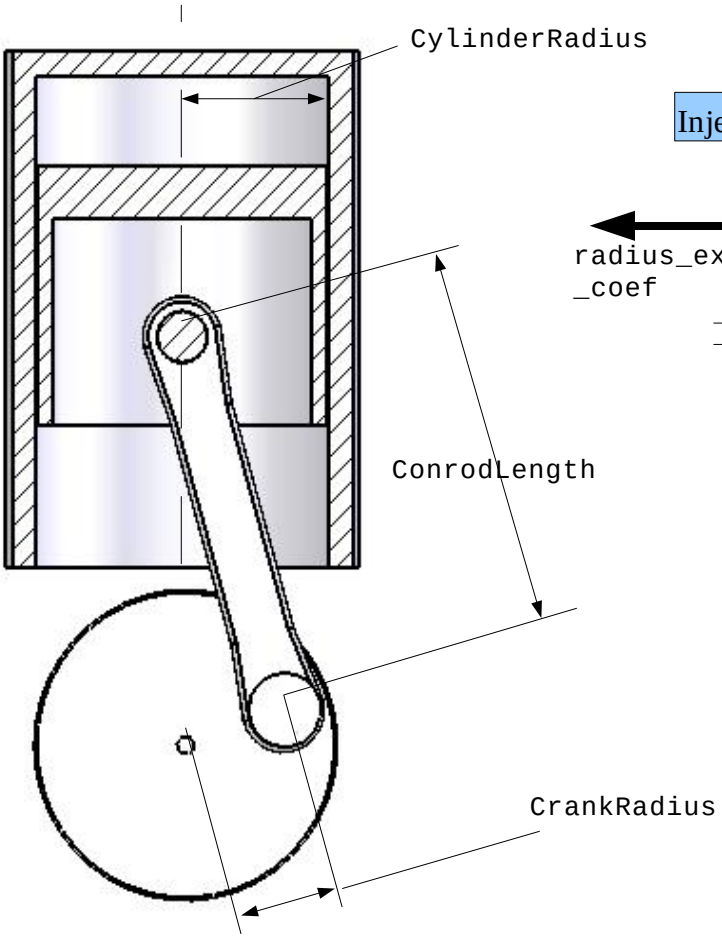


Figure 5: Parameters for describing the cylinder

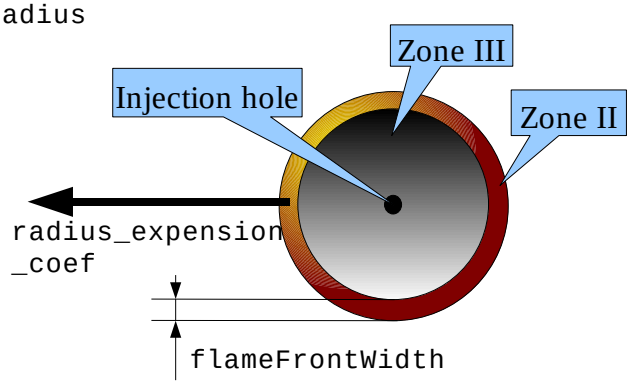


Figure 6: Zones evolution parameters

Height of the chamber. `CylinderHeight` is reached at bottom dead center (on this picture, crank angle is unspecified)

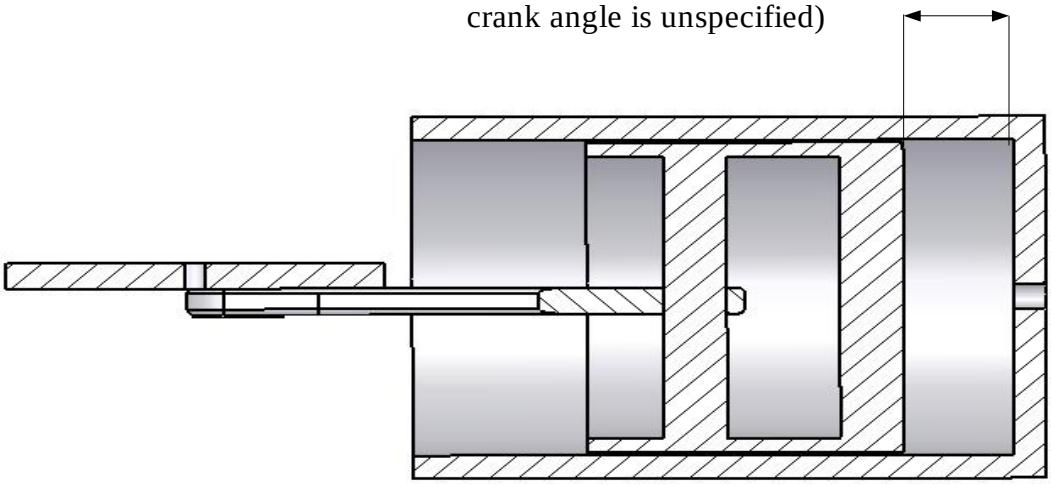


Figure 7: How to set CylinderHeight parameter

The model also calls two external routines to retrieve parameters that should be defined in the calling program (the main code or a test program). They have to be present somewhere, otherwise the model will compile but linking will fail.

Routine prototype	*Parameter type*	*Formal Parameter*	*Default Value*
SUBROUTINE getIntegrationParameter_angle_SOI(angle_SOI)	REAL	The angle when the integration starts, which means the ignition (when the spark occurs), in degree.	340
SUBROUTINE getParameter_engine_speed(speed)	REAL	Engine speed, in rpm.	3000

For example, they could be implemented like this (and this is the way it is done) :

```
c     The angle when the integration starts
      SUBROUTINE getIntegrationParameter_angle_SOI(angle_SOI)
      IMPLICIT NONE
      REAL angle_SOI
      angle_SOI = 340.
      RETURN
      END
```

The file `GeoDatas.f` also contains two useful routines to calculate and display informations about the compiled geometrical model :

- `REAL FUNCTION calcCompRatio()` which returns the compression ratio of the virtual engine.
- `SUBROUTINE printGeoModelInfos()` which displays on screen hard-coded parameters.

In the case of our Skoda engine, here is the summary displayed using this subroutine when the program starts :

```
##########  DUMMY ENGINE GEOMETRY  ##########
##########       Informations      ##########
Cylinder Radius :  0.037750002m
Cylinder Height :  0.0816000029m
Crank Radius :  0.0360000022m
Conrod Length :  0.13000001m
Volume at BDC :  0.365320474L
Compression ratio :  8.49999428

Flame front width :  0.00300000003m
Flame front speed :  25.m/s
################################################
```

D) Implementation

The model consists in five source files (plus the test program). Each file contains routines relative to the name of the file. Those routines are supposed to be independent one from each other and then can be modified, replaced or improved separately (as long as the prototype of the routine remains the same). In order to make the code clearer, I tried not to use COMMON blocks, only two routines use one to communicate indirectly.

The content of the source files is organized as follow :

- `CrankAngle.f` : routines for evaluating the crank angle from time (and the reverse operation).
- `GeoPiston.f` : evaluation of the piston position
- `GeoCylinder.f` : evaluation of the volume of the combustion chamber
- `GeoZones.f` : computation the volume and area of the zones. This can be seen as the main file, as it contains most of the « interface » routines.
- `GeoDatas.f` : virtual engine parameters (see *II.2.C) Configuring the model*)

Only a few routines are called by the main code, and can be qualified as « interface » routines. On the opposite most of the routines can be considered as « internal » ones. The following table shows which routines have to be absolutely present.

File	*Routine*	*Type*	*Must return*
For the core computational model			
GeoZones.f	getZoneVolume_I(t)	REAL	Volume of zone I at any time
GeoZones.f	getZoneVolume_II(t)	REAL	Volume of zone II at any time
GeoZones.f	getZoneVolume_III(t)	REAL	Volume of zone III at any time
GeoZones.f	getContactArea_I_II(t)	REAL	Contact area between zones I and II at any time
GeoZones.f	getContactArea_II_III(t)	REAL	Contact area between zones II and III at any time
GeoZones.f	getSurface_I_CHAMBER(t)	REAL	Contact area between zone I and the walls of the chamber at any time
GeoZones.f	getSurface_II_CHAMBER(t)	REAL	Contact area between zone II and the walls of the chamber at any time
GeoZones.f	getSurface_III_CHAMBER(t)	REAL	Contact area between zone III and the walls of the chamber at any time
CrankAngle.f	getAngle(t)	REAL	Crank angle from time, the reference is the beginning of a cycle
CrankAngle.f	getTime(angle)	REAL	Time from crank angle, the reference is the start of ignition

File	Routine	Type	Must return
For the initial model (InitialModel.f)			
GeoCylinder.f	getChamberVolume(angle)	REAL	Volume of the combustion chamber for any angle
GeoZones.f	getRadius_III(t)	REAL	Radius of zone III at any time
GeoZones.f	getZoneVolume_II(t)	REAL	Volume of zone II at any time
GeoZones.f	getZoneVolume_III(t)	REAL	Volume of zone III at any time

Of course, the values can be entirely computed on-the-fly (like this model does) or loaded from a file (for example containing data from optical measurements) and interpolated to t, which should be the future way to do.

About the way the model does to calculate zone volumes and surfaces, details can be found inside the source code, but from a general point of view, we can say the cycle is divided into four steps, described on *Figure 8.*

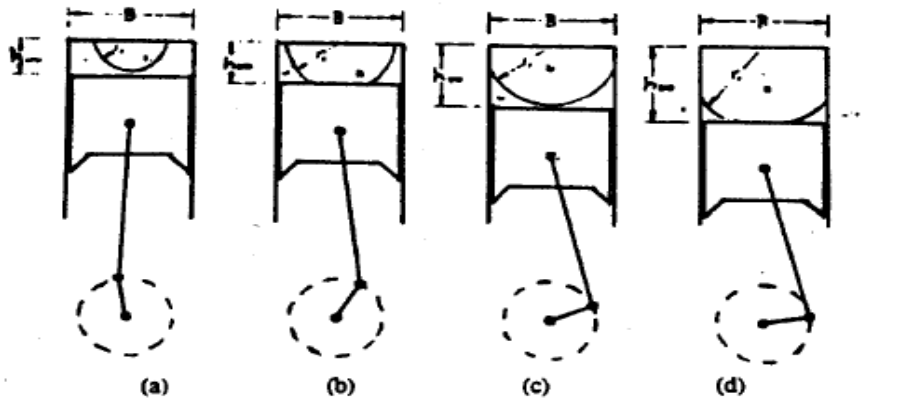


Figure 8: The four cases of geometric interaction inside the combustion chamber. Drawing from Dr. M.R. MODARRES RAZAVI.

Although *Figure 8* was originally designed for a two-zones model, our problem is quite the same except there is one more zone, which means two times more calculations. However, the evolution of the zones is driven by radius of zone III, which means that the same integration routines are used for calculation of volume and contact surfaces of zone II and III, just by replacing the value of the radius.

a) The zone grows freely.

b) The zone meets the piston only. Integration window has to be limited (between z=0 and z=piston_position)

c) The zone meets the walls only. The function returning the radius in the plan of the piston (from the radius in the spherical system) has to be limited to the radius of the cylinder

d) Zone meets both the piston and the walls. Both b) and c) have to be applied.

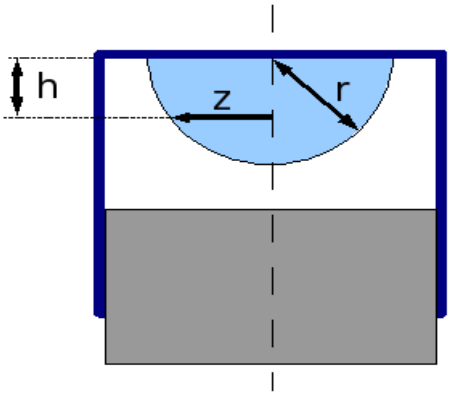


Figure 9: Functions used to calculate zone volume

Then a function giving the radius in the plan of the piston $z_{r(t),step}(h) : h \rightarrow z$ is built (see *Figure 9*) and integrated between the correct bounds. For volume, it yields : $V_{zone} = \int_{h=0}^{h=h_{max}} \pi \; z_{r(t),step}(h)^2 \; dh$.

For contact surface calculations, integrals have been symbolically solved and the results have been hard coded in the program, which is less beautiful but more effective.

To perform the numerical integrations needed, I used a well-known Fortran library called SLATEC. The function itself is called QK61. Of course SLATEC library has to be present, otherwise the linking operation will fail. As SLATEC is also used by other parts of the code, the routines needed have been included in the archive provided with this report. See *Appendix 3* for more details about SLATEC.

E) Results and evaluation

- Running the test program

For evaluation purposes, the geometrical model is provided with a test program, which can be compiled separately from the other modules.

To execute it, assuming you are in *geometry* directory, compile it by typing `make all`, then run `./dummygeom` which should create two files:

- `ffsurf_chart.csv` which contains the flame front surface depending on the crank angle and the flame sphere radius. It allows to plot the chart and to compare it with reality. Those data are not really relevant because the radius is driven by crank angle in this model. However, the model output scale (0 – 10000mm²) is of the same order than the real one (0-7000 mm²).
- `dummygeom.csv` which contains the values of zone volumes and contact areas depending on crank angle (time) for several engine speeds (from 500 rpm to 4500 rpm).

Unfortunately, Fortran does not seem to be able to write more than 72 characters on a line, and at the time I wrote this test program, I hadn't created *iolib.c* yet. So that you should probably type the following commands to make the file *dummygeom.csv* readable with your software :

```
tr -d '\n' < dummygeom.csv > buffer
sed -e 's/ENDLINE/\n/g' buffer > dummygeom.csv
```

- Zone III Volume (burnt gas) at different speeds

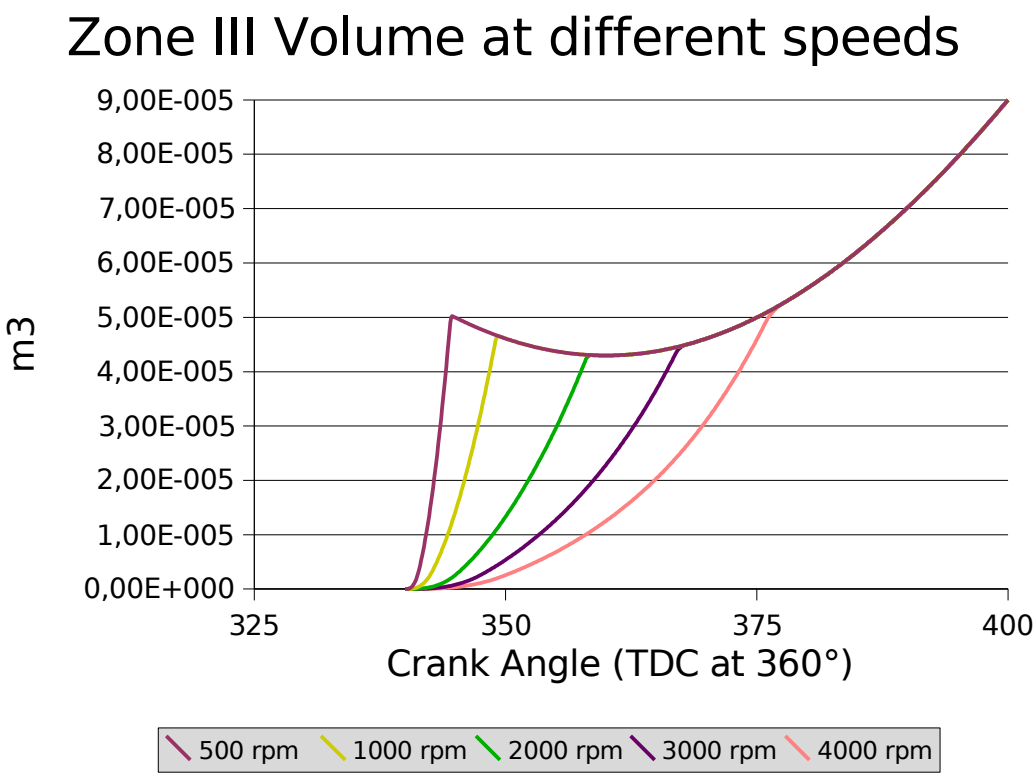


Figure 10: Burnt gas zone volume at different speeds

Figure 10 shows the evolution of the volume of zone III (burnt gas) for a bunch of engine speeds. To make the plot clearer, only zone III is represented, but the evolution for the other zones is similar and details will be given later.

Even if I have no real data to compare with, numerical values seem to be in a correct range. Moreover, the tendency observed here is compliant with the one expected : as the zone grows with a constant speed (25 m/s here), when the engine speed raises, the zone meets the piston later during the cycle (before TDC at 500, 1000 and 2000rpm, after TDC at 3000 and 4000rpm). But for each speed, burnt gas hopefully end up filling the cylinder, faster at a lower engine speed because the relative growth velocity of the sphere-zone is bigger.

- Zone volumes at 3000 rpm

Figure 11 shows volume of the three zones and of the entire cylinder for a specific engine speed (3000 rpm), which is the one chosen to operate the whole model. On this chart we can observe some of the four phases described above :

- 340° : Ignition occurs. Radius of zones II and III grows freely ($V_{\substack{zone\ II \\ zone\ III}} = f(r^3),\ r = f(t)$).

- Around 366,6° : Zone III reaches the piston, or the walls, then both of them and decreases. Volume of zone II (unburnt gas) reaches 0, there is no more fuel to burn. Then volume of the flame front decreases too.
- Around 368,8° : Zone III (burnt gas) fills all the cylinder, combustion is finished for this cycle.

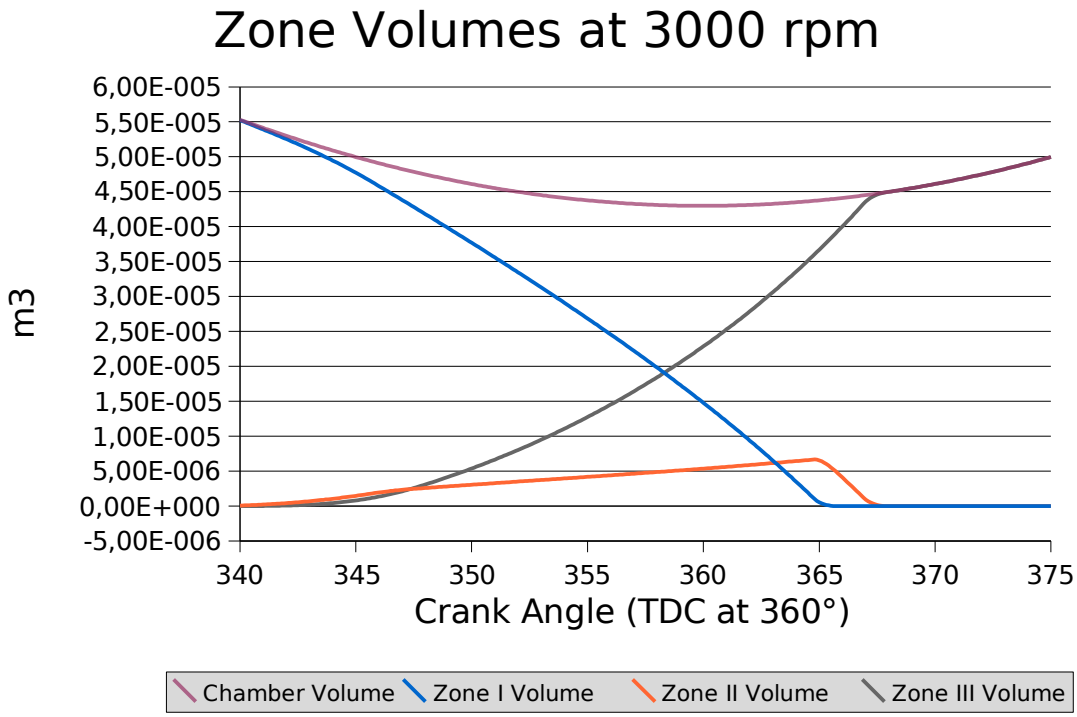


Figure 11: Zone volumes at 3000 rpm

Moreover we can observe that :

- The sum of the volumes of the three zones equals the volume of the combustion chamber at any time. This can be verified numerically and, as the contrary would obviously be wrong, is a way to track errors.
- The volume of zones II and III (i.e. flame front and unburnt gas) is non-zero when t=0 (i.e. When ignition occurs), typically slightly more than zero. This is part of the initial conditions of the problem : even if it is theoretically wrong, it is the best solution to avoid divisions by zero during the first step, especially in the calculation of the rate of burning (see equation 3).

- Flame front surface at 3000 rpm

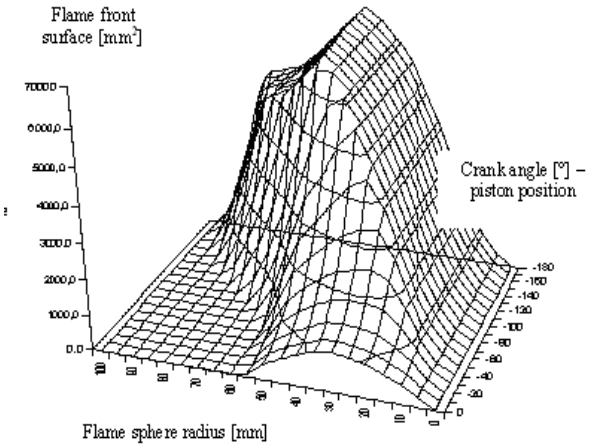


Figure 12: Measured hemispheric flame surface as a function of a piston position and radius of a flame.

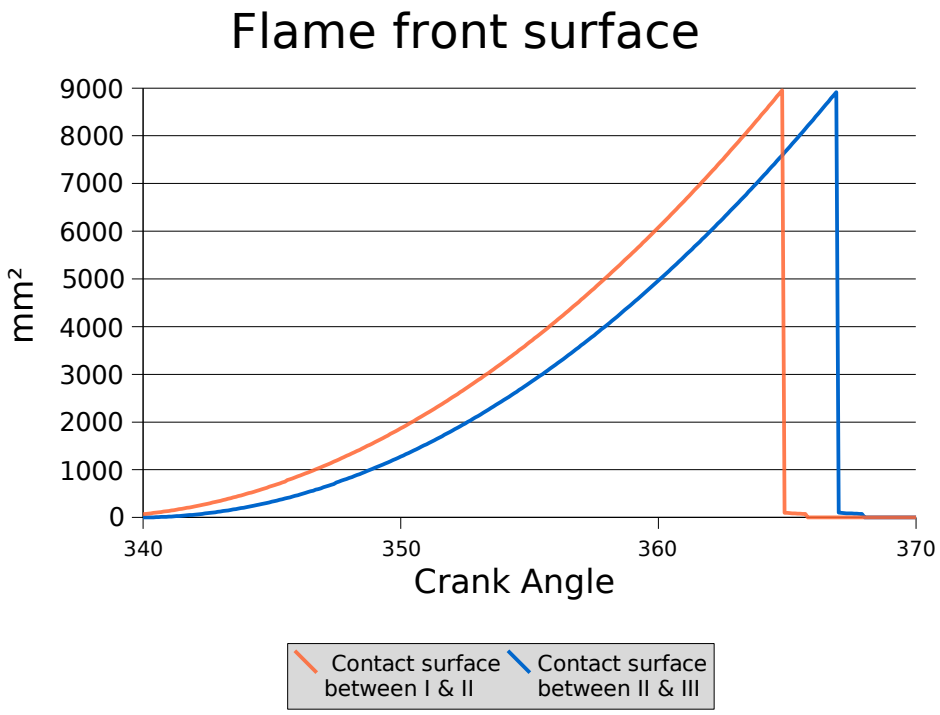


Figure 13: Flame front and back surface at 3000 rpm

Flame front surface is interesting in that we have measurement data to compare with. The chart *Figure 12* plots the real flame front surface for a cylinder with similar geometrical characteristics than our virtual one. Of course, the plot is much more tortuous than the virtual one because of the irregularities of the combustion chamber that are not modeled (valves, complex piston shape...).

However, the range of the values (0-7000 mm²) is similar to the one from this model, showed on Figure 13, where the values are between 0 and 9000mm².

3. The mathematical toolbox and the DE solvers

A) The mathematical toolbox

I had to write various mathematical tools, especially to work with vectors as Fortran does not natively supports arithmetical operations with vectors, whereas most of our equations are not scalar ones. Most of these routines are totally independent and autonomous.

They are all gathered in the file `Maths.f`, and could be eventually reused in another program.

Routine Name	*Arguments*	*Description*
Linear algebra with vectors		
VectMulNum (SUBROUTINE)	REAL out(*) : output vector REAL Vector(*) : input vector REAL Numeric : input numeric INTEGER dim : dimension of out and Vector	Multiply a vector of dimension 'dim' with a numeric : $\left\{_{dim} out\right\} = Numeric \times \left\{_{dim} Vector\right\}$
VectAdd (SUBROUTINE)	REAL out(*) : output vector REAL V1(*) : first input vector REAL V2(*) : second input vector INTEGER dim : dimension of the above vectors	Add two vectors of the same dimension 'dim' : $\left\{_{dim} out\right\} = \left\{_{dim} V1\right\} + \left\{_{dim} V2\right\}$
VectAddNum (SUBROUTINE)	REAL out(*) : output vector REAL V1(*) : input vector REAL num : input numeric INTEGER dim : dimension of out and V1	Add a given numeric to each component of an input vector : $\left\{_{dim} out\right\} = num \times \left\{_{dim} 1\right\} + \left\{_{dim} Vector\right\}$
VectCopy (SUBROUTINE)	REAL out(*) : output vector REAL in(*) : input vector INTEGER dim : dimension of out, in	Perform a carbon copy of a vector : $\left\{_{dim} out\right\} = \left\{_{dim} \text{in}\right\}$
VectMul (FUNCTION)	REAL : output numeric REAL Vcol(*) : first input vector REAL Vlig(*) : second input vector INTEGER dim : dimension of Vcol and Vlig	Multiply to vectors together : $VectMul = \left\{_{dim} Vlig\right\}^{T} \times \left\{_{dim} Vcol\right\}$

Routine Name	*Arguments*	*Description*
Linear algebra with matrix		
Norm (FUNCTION)	REAL : output vector REAL matrix(row,col) : input matrix INTEGER row : number of rows of the matrix INTEGER col : number of columns of the matrix	Cartesian norm of a matrix : $Norm = \sqrt{\sum_{i=1}^{row}\sum_{j=1}^{col}(\text{matrix})_{i,j}^2}$
getColFromMatrix (SUBROUTINE)	REAL out(*) : output vector REAL in(nb_row, *) : input matrix INTEGER col : number of columns of the matrix INTEGER nb_row : number of rows if the matrix and dimension of out	Extract the column 'col' from the matrix 'in'. 'out' must be at least of dimension 'nb_row', and 'in' must have at least 'col' columns, otherwise unpredictable results could occur.
Routines for numerical evaluation		
Diff (SUBROUTINE)	REAL out : result REAL EXTERNAL F : the function to differentiate REAL t : the moment when to differentiate REAL timestep : differentiation timestep	Perform the numerical differentiation $out = \left(\frac{dF(x)}{dx}\right)_{x=t}$ using the simple algorithm of Lagrange : $out = \frac{1}{timestep}[F(t+timestep/2) - F(t-timestep/2)]$ with the following assumption : $(t-timestep/2)<0 \rightarrow (t-timestep/2)$ forced to 0
Interpolate (SUBROUTINE)	REAL out(5) : result REAL known_values(nb_samples,6) : known discrete values INTEGER nb_samples : number of known values REAL T: the parameter to interpolate to, typically temperature	Return the value of the vectorial function described by the discrete values 'known_values' for the X-coordinate T, using a linear interpolation if $\text{known\_values}(1,1)<T<\text{known\_values}(\text{nb\_samples},1)$ (the first line of known_values contains X-coordinates). Elsewhere the result is truncated to the last (resp. the first) available value.

B) The differential equations solvers

`Maths.f` also contain the two solvers provided with the program. Their prototypes are fully compliant, i.e. they take the same arguments in the same order so that it is very easy to switch from one to the other.

For test purposes, `Volterra.f` contains the well known prey/predator problem :

$$(P) \quad \begin{cases} \frac{du}{dt}(t) = \frac{1}{2}u(t)(v(t)-1), & t \in \mathbb{R}_+, \\ \frac{dv}{dt}(t) = v(t)(1-u(t)), \\ u(0) = 2, \\ v(0) = 2 \end{cases}$$

- **Variable order Adams-Bashforth solver**

This solver is in fact just an overhead routine for Slatec DEABM solver, recommended by Slatec documentation for non-stiff to mid-stiff problems, when a huge number of computation points is requested. It is the one in Slatec packages that seems to suit best with our problem.

Extract of Slatec documentation abstract :

```
C         DEABM is a variable order (one through twelve) Adams code.
C         Its complexity lies somewhere between that of DERKF and DEBDF.
C         DEABM is primarily designed to solve non-stiff and mildly stiff
C         differential equations when derivative evaluations are
C         expensive, high accuracy results are needed or answers at
C         many specific points are required.  DEABM attempts to discover
C         when it is not suitable for the task posed.
```

The goal of the overhead routine is to drive the solver :

- Initialize the solver (first iteration)
- Determine the suitable tolerances
- Iterate the solving process for each point requested
- Store the results and forward them to the calling program

DEABM is able to determine by itself the order to use, from the 1st to the 12th order.

Adams-Bashforth is an explicit method for solving differential equations. Its formula of order p is obtained by integrating the polynomial $P(t)$ that interpolates f_{i+1-j} at t_{i+1-j} for $j=1..p$ in place of f, which yields : $y_{i+1}=y_i+h\sum_{j=1}^{p}\alpha_{p,j}f_{i+1-j}$

For example, the formula giving the value of a function F for a 3rd order AB solver is : $F(x,t)\approx\frac{1}{12}\left[5F_{i+1}+8F_i-F_{i-1}\right]$

The prototype of this routine is :

```
SUBROUTINE DEABMSolver(F, NEQ, T_INI, T_FIN, TIMESTEP, Y0,RESULT, DIMRES, DIAG)
```

For a short evaluation of SLATEC solvers, see my preliminary report entitled A short evaluation of the effectiveness of two Fortran DE solvers.

- **2nd order Runge-Kutta solver**

From Pr. Macek experience, a 2nd order Runge-Kutta solver seems to be one of the best choices for our problem. Indeed, an RK solver of an higher order could introduce instabilities and oscillations that could make the system diverge.

SLATEC only includes a 4th order Runge-Kutta solver, so that this routine implements the whole solving algorithm instead of using the one from SLATEC. The formula used, called “enhanced RK2” or “Heun schematic” are :

$$k_1 = \Delta t \cdot F(x_i, t_i)$$

$$k_2 = \Delta t \cdot F(x_i + \frac{k_1}{2}, t_i + \frac{\Delta t}{2})$$

$$x_{i+1} = x_i + k_2$$

As we can see, this method is a fixed time step, explicit one. The prototype of the routine is :

```
SUBROUTINE RK2Solver(F, NEQ, T_INI, T_FIN, TIMESTEP, Y0,RESULT, DIMRES, DIAG)
```

- **Parameters**

As those two solvers have the same prototype, the following description of their parameters is valid for them two.

Parameters	*Type*	*I/O*	*Description*
F	EXTERNAL REAL	I	The equations to integrate. See the following paragraph.
NEQ	INTEGER	I	The dimension of the vector issued by F, i.e. the number of equations in the problem.
T_INI	REAL	I	The time when the integration starts, in seconds. In our problem, it is zero, i.e. when the ignition occurs.
T_FIN	REAL	I	The time when to stop the integration, in seconds.
TIMESTEP	REAL	I	The step between two iterations, in second.
Y0	REAL(NEQ)	I	The vector containing the values of the unknown vector for t=t_ini, i.e. the initial conditions, here in kg.
RESULT	REAL(NEQ+1,*)	O	The table where the results are stored. There is one column more than the number of equations because time is stored in the first one, as a reference. Be sure that the table is large enough for the results to be stored, otherwise unpredictable things could occur (probably a segfault). The minimum number of rows should be ((T_FIN – T_INI)/TIMESTEP+1)
DIMRES	INTEGER	O	The number of significant rows in the RESULT and DIAG tables. Could also be seen as the number of iterations performed.
DIAG	REAL(12,*)	I/O	This table is passed from/to the function F through the solver without any change. Can be useful to pass initialization data. In our case, it is used to retrieve peripheral informations such as pressure, ROB... About the number of rows, the same recommendations than for the RESULT table apply.

- **Equations format**

The equations are provided to the solver by a given subroutine. This one must follow a definite prototype.

```
SUBROUTINE F(T, U, UPRIME, RPAR, IPAR)
```

- *REAL T* : The independent parameter t_i . Here it is time for current iteration.
- *REAL U(NEQ)* : the value of x_i , provided by the solver i.e. the dependent parameter. This vector has as many component than the number of equations NEQ.
- *REAL UPRIME(NEQ)* : The result of the equations ($F(x_i, t_i)$). This vector has as many component than the number of equations NEQ.
- *REAL RPAR(12,*)* : The same as DIAG : see the table below.
- *INTEGER IPAR(1)* : It has the same use than RPAR, but for integer values. It is not implemented in those versions of the solvers but has to be present for compatibility reasons.

As a consequence, the subroutine F could be represented in this way :

$$\left[_{\mathrm{NEQ}} UPRIME\right] = F_{\substack{RPAR \\ IPAR}}\left(\left[_{\mathrm{NEQ}} U\right], T\right)$$

- **Example**

As an example, let's imagine that we want to solve the following simple system of differential equations :

$$x'_1(t) = x_1(t) - x_2(t) + 1$$
$$x'_2(t) = 2x_1(t) - 4x_2(t) + e^t$$
$$x_1(0) = x_2(0) = 0$$

The function F will be the following :

```
      SUBROUTINE F(T, U, UPRIME, RPAR, IPAR)
      INTEGER IPAR(1)
      REAL X, RPAR(12), U(2), UPRIME(2)
      UPRIME(1) = U(1) - U(2) + 1.
      UPRIME(2) = 2.*U(1) + 4.*U(2) + EXP(T)
      RETURN
      END
```

Then, just call for example the RK2 solver to solve the problem :

```
      INTEGER lines
      EXTERNAL F
      REAL RESULT(2,1000), DIAG(12,*), U0(2)
      DATA U0 / 0., 0./
      CALL RK2Solver(F, 2, 0., 10., 0.1, U0, RESULT, lines, diag)
```

4. The Input/Output libraries

A) The output library IOlib

As a 'prehistorical' language, Fortran 77 has an intrinsic limitation when writing a file : it is not able to write more than 73 characters on the same line, either on screen or in a file.

In our case, this can be very tedious, in that our result vectors are very big (at least 16 components of 13 digits each per line). So that using the Fortran primitive 'WRITE(6,*) Result' is not possible, at least if we wish to load the data with Excel.

As a consequence, I had to find a trick to bypass this limitation. This has been done by writing an external library, in C. The file `iolib.c` contains two routines :

```
void writeresults_(int *lines, float table[10000][16])
```

```
void writediags_(int *lines, float table[10000][12])
```

The first one is used to store the table RESULT, the second one, the table DIAGS. Some remarks have to be done :

- The names of the routines are suffixed with an underscore. This is compiler dependent. If you do not want to use the GNU GCC compiler, read the documentation of your compiler to know how to suffix the external C routines.
- Those underscores have to be removed when calling the routines from within the Fortran program : **CALL** writeresults(lines, **RESULT**)
- The indexes of the tables are reversed compared with Fortran. This is due to the way Fortran organizes the memory.
- The maximum number of iterations (here 10000) should be enough, but do not forget to change it here too, if you modify it in the calling Fortran code.
- The header or the file contains the names of the output files, respectively "*results.txt*" and "*diags.txt*".
- In the output files, the values are separated by a tabulation (\t), and the decimal separator is the point (.).

The files created by this library should look like to :

```
#Results for premixed flame parameters simulation

#Time   Zone I, O2      Zone I, N2Ar    Zone I, CO2     Zone I, H2O     Zone I, Fuel    Zone II,
O2     Zone II, N2Ar   Zone II, CO2    Zone II, H2O    Zone II, Fuel   Zone III, O2    Zone III,
N2Ar  Zone III, CO2   Zone III, H2O   Zone III, Fuel
0.0000000e+00    1.4669987e-04    4.9102446e-04    1.0263005e-07    6.3646535e-06    4.0342460e-05
3.5527137e-15    1.0110965e-07    2.4916305e-08    1.3590711e-08    0.0000000e+00    1.7347235e-18
7.6022293e-11   1.8734063e-11   1.0218580e-11   0.0000000e+00
```

B) The data loader

In this chapter we will consider only pressure as an input. Other 'static' data, i.e. which are hard coded inside the program, will be treated in chapter II.6. with the computational model, as they are strictly dependent to it.

There are three routines that are in charge of loading pressure data. This could be seen as quite complicated. That is because, at the beginning of the project, I wanted to implement a caching mechanism in order to speed up the computation. Finally, I gave up mainly because it would have increased the complexity of the code by using a huge COMMON table (which is not recommended), and moreover the gain wasn't significant enough. However, it is still possible to enable this feature quite easily.

- `SUBROUTINE FileLoader (PressureData, TableDepth)` in the file `FileLoader.f`

  This is the real input part of the mechanisms. It returns the `REAL PressureData(2,*)` table filled with data contained in the file `pressure.dat`. This must be a text file, tabulation separated, containing the crank angle values on the first column and the associated pressure values on the second one. `INTEGER TableDepth` gives the real significant value of the table second index.

  Pressure might be in Bar and the crank angle should vary from 0 to 719 degree, following the convention described by the geometrical model.

- `REAL FUNCTION Pressure (angle)` & `REAL FUNCTION Pressure_from_time(time)` in the file `Pressure.f`

  These are the main routines called by the other parts of the program. After retrieving the table `PressureDatas` by calling the routine bellow, they linearly interpolate the pressure from the two nearest known crank angle values, after converting from time if requested.

- `SUBROUTINE getPressureDatas(PressureDatas, TableDepth)` in the file `ZoneApproach.f`

  This routine is the link between the preceding ones. In this implementation, it just calls `FileLoader` to retrieve pressure values each time it is called by `Pressure`. But `FileLoader` could also be called once when the program starts, then `PressureDatas` would be stored in a COMMON, and this COMMON would be used by `getPressureDatas` instead of reading the file again, which would speed up the whole process.

To make the things clearer, the following schematic summarizes the way it works presently (i.e. without implementing any caching mechanisms).

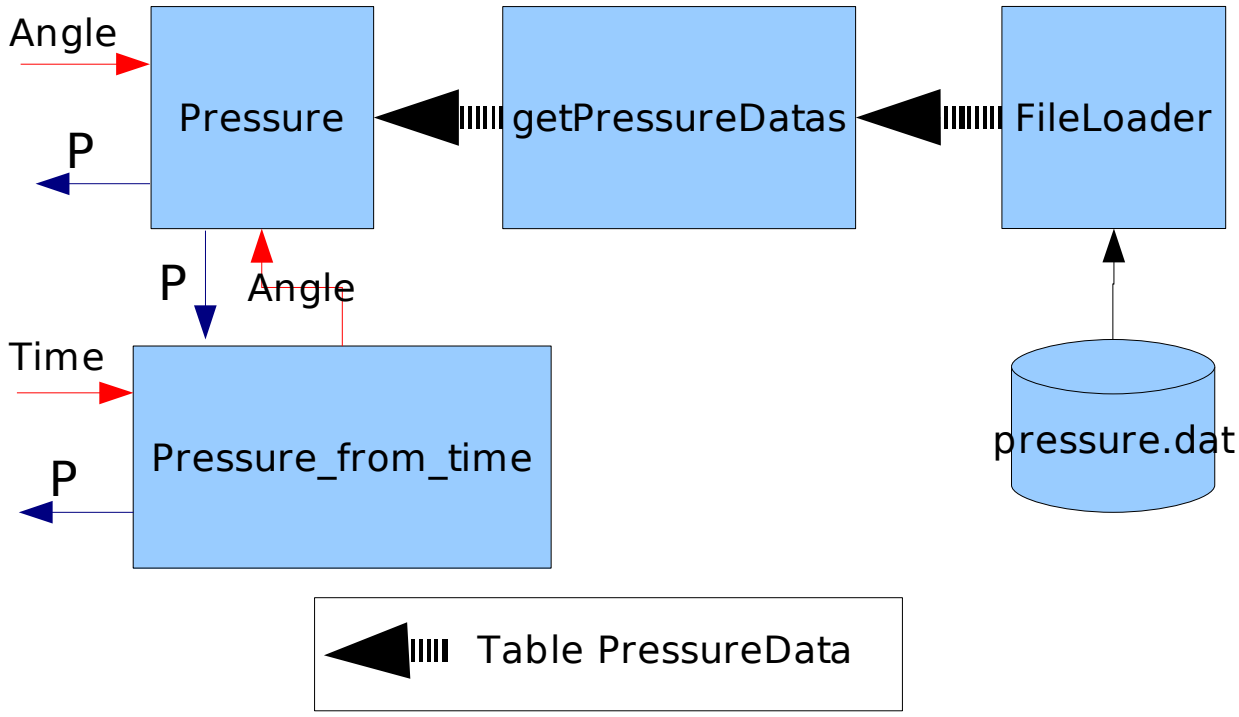


Figure 14: How the pressure data loader works

5. The Initial Model

A) Theory

Just like every differential problem, ours needs initial conditions. Here, these conditions relate to initial masses in the three zones. Indeed, the model itself is not able to work with null masses, or null volumes : this would lead to divisions by zero and other unwanted mathematical artifacts.

The way we decided to do is to consider that when the integration starts, the three zones are already created, even if zones II and III are really tiny.

So that this can be summarized by the assumption $radius_{III}(t=0)>0$.

For the standard burning reaction $C_xH_y+(x+\frac{y}{4})O_2 \rightarrow xCO_2+(\frac{x}{y})H_2O$ (assuming $\lambda=1$), the ratio $O_t=\frac{32(x+\frac{y}{4})}{12x+y}$ represents the quantity of fuel that is necessary to reach the stoichiometric proportions with oxygen. Then, using the standard repartition of components in air, it yields the repartition of the components inside the cylinder :

$$\begin{array}{c} X_{O_2}=0.23 \\ X_{CO_2}=0.0002 \\ X_{N_2+Ar}=0.77 \\ X_{H_2O}=0.01 \\ X_{fuel}=\frac{X_{O_2}}{O_t} \end{array} \longrightarrow \begin{array}{l} MC_{O_2}=\frac{X_{O_2}}{X_{O_2}+X_{CO_2}+X_{N_2+Ar}+X_{H_2O}+X_{fuel}} \\ MC_{N_2+Ar}=\frac{X_{N_2+Ar}}{X_{O_2}+X_{CO_2}+X_{N_2+Ar}+X_{H_2O}+X_{fuel}} \\ MC_{CO_2}=\frac{X_{CO_2}}{X_{O_2}+X_{CO_2}+X_{N_2+Ar}+X_{H_2O}+X_{fuel}} \\ MC_{H_2O}=\frac{X_{H_2O}}{X_{O_2}+X_{CO_2}+X_{N_2+Ar}+X_{H_2O}+X_{fuel}} \\ MC_{fuel}=\frac{X_{fuel}}{X_{O_2}+X_{CO_2}+X_{N_2+Ar}+X_{H_2O}+X_{fuel}} \end{array}$$

Then a traditional One-Zone model is used to determine the amount of species inside the cylinder, simply using the perfect gas law :

$$[m_{OneZone}]=[MC]\frac{p_{Inlet\ Closing}*V_{Inlet\ Closing}}{T_{Inlet\ Closing}[MC]^T[r]}$$

As we assumed that $radius_{III}(t=0)>0$, which implies we have two 'fake' zones, II and III. Their non zero initial volume will be called respectively V_{II0} and V_{III0} . Then the components can be distributed to those two zones proportionally to their volume, but taking in consideration the stoichiometric proportions have to be respected :

$$Mp_{O_2,II}=[m_{OneZone}]_{O_2}\frac{V_{II0}}{V_{InletClosing}}$$
$$Mp_{fuel,II}=[m_{OneZone}]_{fuel}\frac{V_{II0}}{V_{Inlet\ Closing}}$$

$Mp_{O_2,II}$ and $Mp_{fuel,II}$ are intermediate quantities to simplify writing.

Now we have two possibilities : oxygen or fuel could be in excess, which is not possible in zone II and III as we assumed we should respect the stoichiometric proportions. So that the excess is artificially 'burnt' :

$$\left[Mp_{O_2,II} - Mp_{fuel,II} * O_t\right] > 0 \rightarrow \begin{cases} \text{YES (O2 in excess)} & M_{II0,O_2} = Mp_{O_2,II} - Mp_{fuel,II} * O_t \\ & M_{II0,fuel} = 0 \\ \text{NO (Fuel in excess)} & M_{II0,O_2} = 0 \\ & M_{II0,fuel} = Mp_{fuel,II} - \dfrac{Mp_{O_2,II}}{O_t} \end{cases}$$

Masses of carbon dioxide and steam are pretty easy to guess using the burning equation, and nitrogen is simply distributed geometrically, as a non reacting specie :

$$M_{II0,CO_2} = Mp_{fuel,II} \frac{44x}{12y + x}$$

$$M_{II0,H_2O} = Mp_{fuel,II} \frac{18\frac{x}{2}}{12y + x}$$

$$M_{II0,N_2+Ar} = \left\{m_{OneZone,N_2+Ar}\right\} \frac{V_{II0}}{V_{Inlet\ Closing}}$$

Now our initial vector $\{m_{II0}\}$ is complete. The process is exactly the same for $\{m_{III0}\}$. Eventually, $\{m_{I0}\}$ contain what remains outside the two other zones :

$$\{m_{I0}\} = \{m_{OneZone}\} - \{m_{II0}\} - \{m_{III0}\}$$

B) The Fortran module

This module uses two files : one setting the initial radius of zone III, and the other one implementing the equations above.

- `InitialGeom.f`

This file contains the routine **`FUNCTION`** `getRadius_III_Initial ()` just returning the initial radius of zone III (burnt gas).

Currently, it should be **0.3 mm**.

- `InitialModel.f`

In this file there is also only one routine **`SUBROUTINE`** `Initial_Model(Masses)` returning a **`REAL(15)`** vector following the convention defined in chapter II.6 :

$m_{O_2,I}$	$m_{N_2+Ar,I}$	$m_{CO_2,I}$	$m_{H_2O,I}$	$m_{fuel,I}$	$m_{O_2,II}$	$m_{N_2+Ar,II}$	$m_{CO_2,II}$	$m_{H_2O,II}$	$m_{fuel,II}$	$m_{O_2,III}$	$m_{N_2+Ar,III}$	$m_{CO_2,III}$	$m_{H_2O,III}$	$m_{fuel,III}$
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Masses are in **kg**. This vector is designed to be directly injected in the solver as the initial conditions. Moreover, the routine displays a report after execution. Notice that this model consider (hard coded data) that the temperature at inlet close is **350K**.

########## FIRST STEP APPENDIX MODEL ##########

########## Informations ##########

Pressure at inlet close : 1.04200006 bar

Mean temperature at inlet close : 350. K

Chamber volume at inlet close : 0.365320474 L

One zone model :

 Total mass of mixture : 0.684673727 g

 Per specie in grams : (O2 - N2Ar - CO2 - H2O - fuel)

  0.146699861 0.491125613 0.000127565087 0.00637825439 0.0403424613

One zone model :

 Total mass of mixture : 0.384230167 g

 Per specie in grams : (O2 - N2Ar - CO2 - H2O - fuel)

  0.0823260918 0.275613427 7.15879069E-05 0.00357939536 0.0226396751

Three fake zones model :

 Zone I (Unburnt gas)

 Per specie in grams : (O2 - N2Ar - CO2 - H2O - fuel)

  0.0823260918 0.275556624 5.75946797E-05 0.0035717627 0.0226396751

 Zone II (Flame front)

 Per specie in grams : (O2 - N2Ar - CO2 - H2O - fuel)

  0. 5.6741439E-05 1.39827125E-05 7.62693389E-06 0.

 Zone III (Burnt gas)

 Per specie in grams : (O2 - N2Ar - CO2 - H2O - fuel)

  0. 4.26627373E-08 1.05133182E-08 5.73453729E-09 4.33680869E-16

##

6. The main computational model

A) Adjustments to the theoretical model

The first results have led to make some adjustments to the theoretical model. Assumptions and limits have been added, in order to prevent situations that could seem obviously impossible when solving 'by hand', but that the code cannot guess by itself ! This set of equations is not present in Pr. Macek theoretical work : most of them are typical of the problems that can happen during the transition from a working theory to its equivalent numerical model.

This is probably the most debatable point of my work and those assumptions should be tested again and validated later, because I unfortunately had neither the time nor the sufficient knowledge to carry out the right adjustments. However, I am pretty sure they all have to be present, otherwise some situations could be a nonsense, and others should probably be added in the future.

- ***Limitation of the rate of burning***

Considering the equation $\left\{\dot{r}_{II}\right\}=\left\{{}_5C_B\right\}\dot{r}_{II,fuel}+\left\{\dot{r}_{II,Corr}\right\}$, as the coefficients of $\left\{{}_5C_B\right\}$ concerning dioxygen and fuel are negative, a too high value of the ROB $\dot{r}_{II,fuel}$ could lead to negative masses in zone II (flame front). So that I propose the following equation to ensure the code won't burn more than he has :

$$\dot{r}_{II,fuel} \le MIN\left(\frac{m_{I,fuel}+m_{I,fuel}\frac{A_{f,I,II}}{V_I}w_{f,front}\Delta t}{\Delta t\,|C_b^{fuel}|} \;;\; \frac{m_{I,O_2}+m_{I,O_2}\frac{A_{f,I,II}}{V_I}w_{f,front}\Delta t}{\Delta t\,|C_b^{O_2}|} \right)$$

This equation tries to discover the maximum amount of dioxygen and fuel that should be present in zone II at the beginning of the next iteration, and then deduces the maximum rate of burning. The problem here is that it uses Δt which is the time step between two 'effective' steps, the only one which is easily known. In fact between two of these steps, the equations are called several times, depending of the integration method employed and then the 'real' time step between two of these calls is pretty hard to guess.

This could explain why we still have negative masses values sometimes in zone II, and why they are not at same place depending on the solver used (RK2 or DEABM). Anyway, even with this defect, it improves in a significant way the quality of the results. But it is still perfectible.

- ***Limitation of flame front speed value***

It it necessary to force the code to consider the following obvious assumption :

$$w_{f,front} \geq 0$$
$$w_{f,back} \geq 0$$

It could also be interesting to impose a top border to avoid for example artifacts in numerical resolution that could produce extremely high values of $w_{f,back}$ and then throw off the balance all the remaining iterations : due to equation *(5)* masses could then become negative. It could also be considered to force $w_{f,front}$ to be higher than a certain value, not to encounter the case where the flame front wouldn't be supplied anymore. Then the assumption would become :

$$0 < a \leq w_{f,front}$$
$$0 \leq w_{f,back} \leq b$$

- ***Force the homogeneity of the temperatures when*** $t = 0$

Following Pr. Macek recommendation, the following assumption is applied :

$$t = 0 \rightarrow \left\{ \begin{array}{l} T_{II} = T_I(0) \\ T_{III} = T_I(0) \end{array} \right\}$$

This is because as zone II and III are really tiny, the calculated temperature inside those two zones wouldn't be really significant. Moreover, as the ignition is just occurring and that zone II and III have just been created, it is not inadmissible to think that the temperature in the cylinder is still homogeneous.

- ***Limitation of the temperatures*** (not implemented)

In the future, it could be considered to limit the temperature in the flame front, for example to the adiabatic flame temperature, which should yield something like (for a stoichiometric mixture) :

$$\forall t, T_{II}(t) \leq 2301.63 + 13.70 * \ln\left(\frac{P}{P_0}\right), P_0 = 1\,atm = 101.32\,kPa$$

B) How the code works

It it divided into three files :

- `Datas.f` : Contains routines to provide static data to the equations (such as the specific gas constants).
- `ReactionRate.f` : Routines to evaluate the reaction rate into the three zones. Could eventually be merged with `Equations.f`.
- `Equations.f` : Contains the most important part of the equations, from the 'entry point' routine for the solver to the evaluation of zone temperatures.

C) Data needed by the equations : `Datas.f`

- ***The way static data are input***

In this version of the code, most of the data and informations requested by the code are directly encoded inside the source code.

This way of proceeding has pros and cons : it increases the mean speed as necessary data are directly packed inside the executable module. If those parameters were read from inside a file, each call of one of these routines would have triggered a disk operation (as there are no global variables in Fortran, and that I do not want to use COMMON, there is no possibility to read the parameters once for all at the beginning). Moreover, it was obviously the easiest way to do.

On a second hand, If you want to modify even one of those parameters, you will have to edit the source code, and then to recompile the whole program, which can be quite tiresome, at least the first time you do it.

However, all the following routines, which are necessary for the program to work, are quite easy to understand. Sometimes it is even only an affectation :

```
      SUBROUTINE getIntegrationParameter_angle_SOI(angle_SOI)
      IMPLICIT NONE
      REAL angle_SOI
      angle_SOI = 340.
      RETURN
      END
```

Such a routine could be for example rewritten in C to use global variables, whose input would be done from a file or from the keyboard instead of directly from inside the source code.

- ***Table of the routines***

This table describes all the routines inside `Datas.f` and the output they should provide. Some are more complicated, but luckily they are also less likely to be modified.

Notice that all of them are SUBROUTINE.

Routine Name	*Arguments*	*Description*
Walls_Temp_Params	REAL angle : (input) angle, in degree REAL alpha_coef : output alpha coefficient REAL temp_wall : output wall temperature, in Kelvin	Returns the two coefficients for the law describing the temperature of the walls of the combustion chamber used in the formula $\dot{Q}_{i,CH} = S_{i,CH} * \text{alpha\_coef}(T_i - \text{temp\_wall})$ The current angle is provided if needed, but coefficients can obviously also be constants.
getIntegrationParameter_Timestep	REAL time_step : output, in seconds	Defines the time between two iterations, Δt used in the derivation routine, by the driver when calling

Routine Name	*Arguments*	*Description*
		the solver, and by the routine guessing the maximum ROB.
getIntegrationParameter_angle_SOI	REAL angle_SOI : output, in degrees	Defines the angle when the integration begins (Start Of Integration), which is also the value of the crank angle when the ignition occurs (remember that TDC = 360°).
getParameters_engine_speed	REAL speed : output, in RPM	The speed of the virtual engine. Assumed to be constant in this version.
getFuelCompo	INTEGER m : output INTEGER n : output	Sets the nature of the fuel, where m and n are : $C_m H_n$
correction_vector	REAL Vrr_corr(5) : output correction vector REAL t : input time, in seconds	The routine returning the correction vector in equation (6) : $\left\{\dot{r}_{II}\right\}=\left\{{}_5C_B\right\}\dot{r}_{II,fuel}+\left\{\dot{r}_{II,Corr}\right\}$. Time is provided if needed.
reaction_matrix	REAL rm(5,1) : output the reaction matrix INTEGER row : number of rows in the matrix (here 5) INTEGER col : number of columns in the matrix (here 1)	The matrix representing the occurring reactions. Here only the burning reaction is taken in consideration, depending on the nature of the fuel.
Cp	REAL Cp_vect(*) : output vector, in J/Kg/K INTEGER dim : input dimension of the vector REAL T : input temperature, in kelvin	Returns the values of Cp for each specie at the requested temperature. Values are interpolated from a bunch of known values. Here “dim” should be 5, and the length of “Cp_vect” should be at least 5, or funny things would occur.
Enthalpy	REAL i_vect(*) : output vector, in J/Kg INTEGER dim : input dimension of the vector REAL T : input temperature, in kelvin	Returns the values of the specific enthalpy for each specie at the requested temperature. Values are interpolated from a bunch of known values. Here “dim” should be 5, and the length of “i_vect” should be at least 5, or funny things would occur.
Specific_gaz_constant	REAL r_vect(*) : output vector, in J/Kg/K INTEGER dim : input dimension of the vector	Returns the values of the specific gas constant for each specie. Here “dim” should be 5, and the length of “r_vect” should be at least 5, or funny things would occur.

- ***Default values of those parameters***

The following table summarizes the constants parameters applied in this version of the code.

Parameter	*Value*	*Comment*
alpha_coef	0	Cooling disabled in the version.
temp_wall	0	
time_step	0.00555589423 ms	Corresponds to 0.1° of crank angle at 3000rpm
angle_SOI	340°	
speed	3000 rpm	
m	3	Fuel used is C_3H_8
n	8	
rm	{-5, 0, 3, 4, -1}	Corresponds to the coefficient of the burning reaction $C_3H_8+5\,O_2\rightarrow 3\,CO_2+4\,H_2O$
Vrr_corr	{0, 0, 0, 0, 0}	Disabled : till now, only the burning reaction is taken in consideration
r_vect	{259.827, 295.434, 188.965, 461.915, 188.965}	The precision (only 10^{-3}) is probably too weak and should be increased.

Cp and specific enthalpies are temperature dependent (see *Figure 15 & 16* respectively). The routines do not compute them on the fly : I calculated a bunch of values before (one value each 100K), them the routines only use the “Interpolate” tool. Here the precision is only 10^{-1} which is quite weak especially especially compared to Cp values. This could be interesting to use more known points with a better precision.

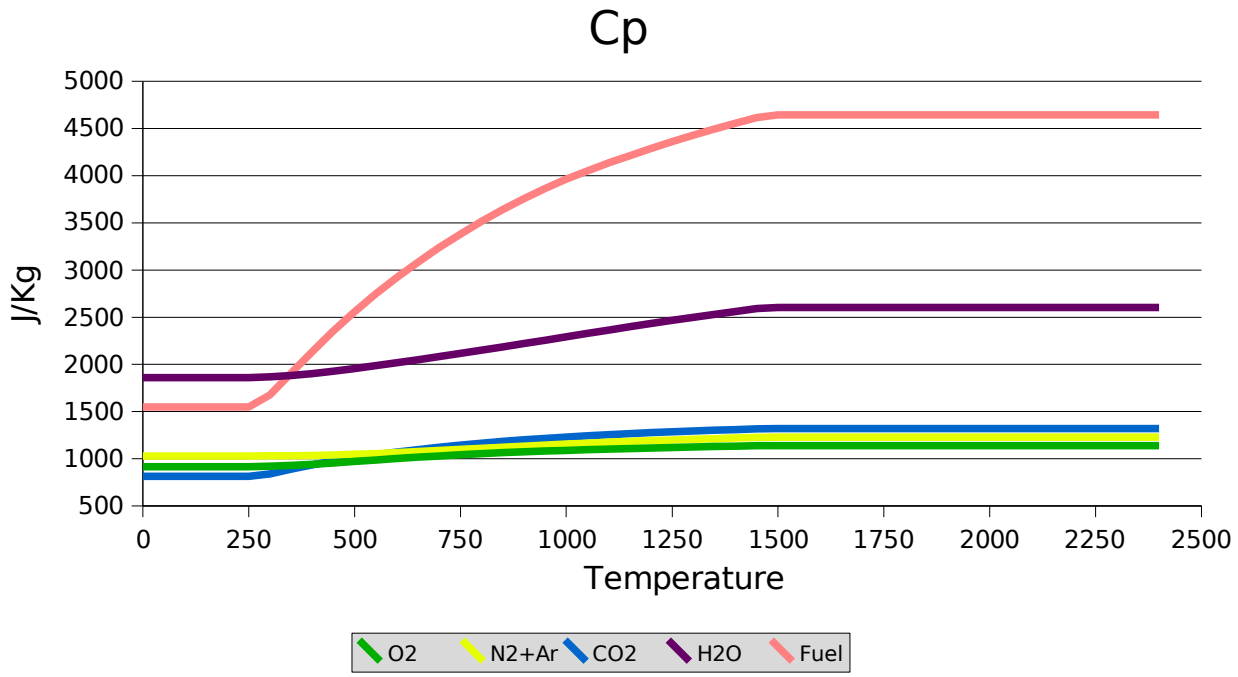


Figure 15: Default values for Cp

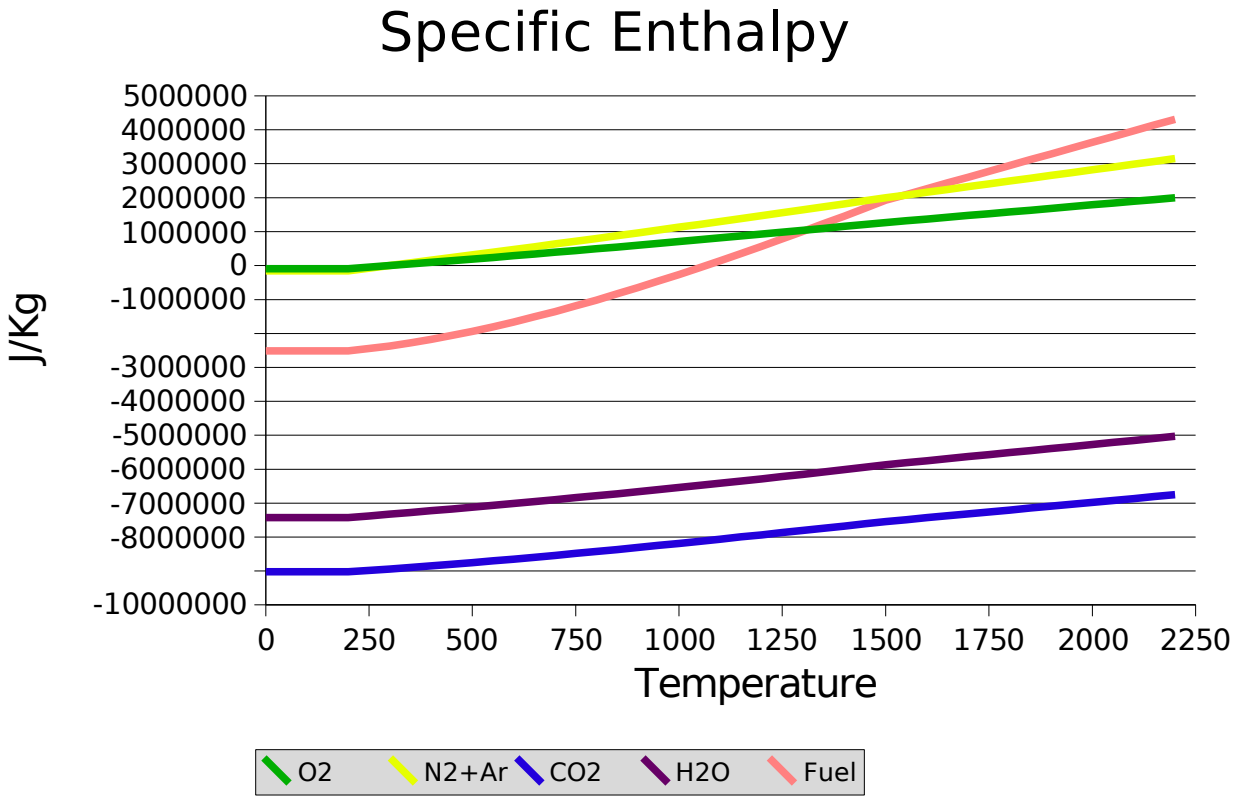


Figure 16: Default values for Specific Enthalpy

D) The core of the model : `Equations.f` & `ReactionRate.f`

Those two files contain most of the equations exposed in chapter I.2). In that, they are the ones which really host the computational translation of the three-zones model.

To make the files more understandable for further work, I tried not to use only one routine (what would have been possible), but to split it into several routines, each of them standing for more or less one -or a group of- equation described above. So that it is not necessary to modify and check to whole file (which is probably the longer of the whole program) when applying a minor patch to a single equation. It could also be possible to use several versions of a routine for the same purpose , for example another state equation could be easily implemented to evaluate temperatures (*Figure 17*).

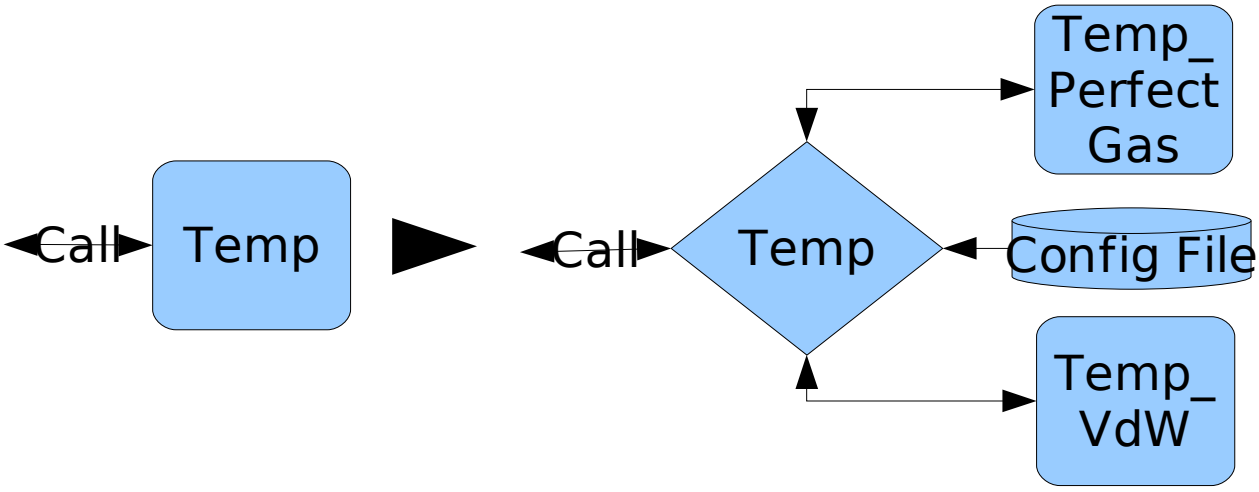


Figure 17: Example showing the replacement of the present simple model to a more complicated one for temperature evaluation

As a consequence, some parts or routines could seem to be quite complicated, or less efficient than they could be. Of course it would certainly have been much more efficient if all the equations had been encoded in the same routine : for example the temperature would have been evaluated once rather than one time for each equation or so. But then the code would have been far less modular and easy to modify. Moreover, modularity of the code was part of the initial waitings.

With the same purpose, most of time I named the variables in a way that make them easily to find out in the equations. $A_{f,I,II}$ will become `A_f_I_II`. Lower case characters are used even if Fortran is not case sensitive.

As we previously did, we will describe all the routines' prototypes in a synthesis table, then we will go further about the main routine.

- ***Synthesis table of the routines***

Routine Name	*Arguments*	*Equivalent Equation / Comment*
In ReactionRate.f		
rr_I (SUBROUTINE)	O REAL(5) reaction_rate : output vector O INTEGER dim : size of reaction_rate vector I REAL t : current time	Computes reaction rate vector for zone I. In this version, the returned vector is always full of zeros as the isn't any reaction occurring in zone I.
rr_II (SUBROUTINE)	O REAL(5) reaction_rate : output vector O INTEGER dim : size of reaction_rate vector I REAL t : current time I REAL(*) m_I : masses in zone I I REAL(*) m_II: masses in zone II I REAL(*) m_III: masses in zone III	Computes reaction rate vector for zone II. $\left\{\dot{r}_{II}\right\}=\left\{{}_5C_B\right\}\dot{r}_{II,fuel}+\left\{\dot{r}_{II,Corr}\right\}$ (6) $\left\{\dot{r}_{II,Corr}\right\}=\left\|{}_{5*r-1}C_{corr}\right\|\left\{{}_{r-1}\dot{r}_{II,corr}\right\}$ (7) Notice that the corrective vector IS implemented, even if it is always zero in this version (as said in Datas.f description).
rr_III (SUBROUTINE)	O REAL(5) reaction_rate : output vector O INTEGER dim : size of reaction_rate vector I REAL t : current time	Computes reaction rate vector for zone III. In this version, the returned vector is always full of zeros as the isn't any reaction occurring in zone III.
In Equations.f		
MassTransfert (SUBROUTINE)	O REAL(*) out : output vector (the left side of the equation) I REAL(*) mdot_ip_i I REAL(*) mdot_i_if I REAL(*) rdot_i I INTEGER dim : the size of the above vectors	Computes one of the equations of the system : $\frac{d\left\{m_I\right\}}{dt}=-\left\{\dot{m}_{I,II}\right\}$ $\frac{d\left\{m_{II}\right\}}{dt}=\left\{\dot{m}_{I,II}\right\}-\left\{\dot{m}_{II,III}\right\}+\left\{\dot{r}_{II}\right\}$ (8) $\frac{d\left\{m_{III}\right\}}{dt}=\left\{\dot{m}_{II,III}\right\}$ More precisely : $\left\{{}_{dim}out\right\}=\left\{{}_{dim}mdot\_ip\_i\right\}-\left\{{}_{dim}mdot\_i\_if\right\}+\left\{{}_{dim}rdot\_i\right\}$
Core_Equations_System (SUBROUTINE)	I REAL T : time, in second I REAL(5) U : known mass values O REAL(5) UPRIME : calculated derivatives of mass values O REAL(15) RPAR : control data INTERGER(1) IPAR : unused	The entry point, in a format understandable by the solver. See below.

Routine Name	*Arguments*	*Equivalent Equation / Comment*
Eval_mdot_I_II (SUBROUTINE)	O REAL(*) mdot_I_II : output vector I REAL A_f_I_II I REAL V_I I REAL w_f_front I REAL m_I(*) I INTEGER dim : size of the above vectors	$\{\dot{m}_{I,II}\} = \frac{A_{f,I,II}\, w_{f,front}}{V_I} \{m_I\}$ (5)
Eval_mdot_II_III (SUBROUTINE)	O REAL(*) mdot_II_III : output vector I REAL A_f_II_III I REAL V_II I REAL w_f_back I REAL m_II(*) I INTEGER dim : size of the above vectors	$\{\dot{m}_{II,III}\} = \frac{A_{f,II,III}\, w_{f,back}}{V_{II}} \left(\{m_{II}\} - \left\{ m_{II,\,fuel\atop O_2} \right\} \right)$ (6) The evaluate the term $\left(\{m_{II}\} - \left\{ m_{II,\,fuel\atop O_2} \right\} \right)$, the routine just replace m_II(1) and m_II(dim) by 0, so that it is necessary that those two positions contain masses of dioxygen and fuel.
w_flame_front (SUBROUTINE)	O REAL out : result, in m/s I REAL(*) m_I : masses in zone I, in kg I REAL TempI : Temperature in zone I, in kelvin I INTEGER dim : size of the vector m_I I REAL time : current time, in seconds	Computes the equation (1), which is too big to fit here. ● Also implements a limitation mechanism, as described above (negative values are replaced by 0) ● The corrective term of the equation is not implemented
w_flame_back (SUBROUTINE)	O REAL out : result, in m/s I REAL(*) m_II : masses in zone II, in kg I REAL(*) m_III : masses in zone III, in kg I REAL TempII : Temperature in zone II, in kelvin I REAL TempIII : Temperature in zone III, in kelvin I INTEGER dim : size of the vectors m_II and m_III	Computes the equation (2), which is too big to fit here. ● Also implements a limitation mechanism, as described above (negative values are replaced by 0) ● The corrective term of the equation is not implemented

Routine Name	*Arguments*	*Equivalent Equation / Comment*
	I REAL time : current time, in seconds	
rdot_II_fuel (SUBROUTINE)	O REAL out : result, in kg/s I REAL(*) m_I : masses in zone I I REAL(*) m_II : masses in zone II I REAL(*) m_III : masses in zone III I REAL time : current time in seconds I INTEGER dim : size of masses vectors	Computes the equation (3), which is too big to fit here, and the limitation of the ROB described in chapter II.6.A). For safety reasons, the possible negative values are replaced by 0. It is equivalent to the Rate of Burning, as the corrective term is not implemented.
Temp (FUNCTION)	O REAL : Current temperature in requested zone, in kelvin I REAL(*) m1_vect : masses in zone I I REAL(*) m2_vect : masses in zone II I REAL(*) m3_vect : masses in zone III I REAL time : current time in seconds I INTEGER zone : the zone you want to evaluate temperature for. I INTEGER dim : size of masses vectors	Implements the state equation (the perfect gas law), and the initial condition : $$T_i = \begin{cases} t=0 \rightarrow T_i = T_I \\ t>0 \rightarrow \dfrac{p\,V_i}{\left(r\right)^T \left(m_i\right)} \end{cases}$$ So that, if you are sure that time > 0, the masses vectors of the two zones you do not requested the evaluation can be dummy vectors (they won't be used). To obtain temperature in zone I, m1_vect is always the only significant vector (and then m2_vect and m3_vect can be dummy ones).
Qdot_I_CH (FUNCTION)	O REAL : result, heat transferred to the wall for zone I I REAL M_vect : masses vector in zone I I REAL temp : current temperature in zone I, in kelvin I REAL time : current time, in seconds I INTEGER dim : size of M_vect	Evaluates the heat transferred to the walls by zone I : $\dot{Q}_{I,CH} = S_{I,CH} * \alpha \left(T_I - T_{CH} \right)$ α and T_{CH} are retrieved by calling `Walls_Temp_Params` in `Datas.f` As cooling is disabled in `Datas.f`, the routine presently always returns 0.
Qdot_II_CH (FUNCTION)	O REAL : result, heat transferred to the wall for zone II I REAL M_vect : masses vector in zone II	Evaluates the heat transferred to the walls by zone II : $\dot{Q}_{II,CH} = S_{II,CH} * \alpha \left(T_{II} - T_{CH} \right)$

Routine Name	*Arguments*	*Equivalent Equation / Comment*
	I REAL temp : current temperature in zone II, in kelvin I REAL time : current time, in seconds I INTEGER dim : size of M_vect	α and T_{CH} are retrieved by calling `Walls_Temp_Params` in `Datas.f` As cooling is disabled in `Datas.f`, the routine presently always returns 0.
Qdot_III_CH (FUNCTION)	O REAL : result, heat transferred to the wall for zone III I REAL M_vect : masses vector in zone III I REAL temp : current temperature in zone III, in kelvin I REAL time : current time, in seconds I INTEGER dim : size of M_vect	Evaluates the heat transferred to the walls by zone III: $\dot{Q}_{III,CH} = S_{III,CH} * \alpha (T_{III} - T_{CH})$ α and T_{CH} are retrieved by calling `Walls_Temp_Params` in `Datas.f` As cooling is disabled in `Datas.f`, the routine presently always returns 0.
Cp_Mi (FUNCTION)	O REAL : result I REAL(*) Mi_vect : masses vector I INTEGER dim : size of vector Mi_vect I REAL T : temperature in kelvin	A routine to evaluate $(c_p m_i) = \{c_p(T_i)\}^T \{m_i\}$
KappaBlock (FUNCTION)	O REAL : result I REAL(*) Mi_vect : masses vector I INTEGER dim : size of vector Mi_vect I REAL T : temperature in kelvin	A routine to evaluate $\left(\frac{\kappa}{\kappa - 1}\right)_i = \frac{\{c_p(T_i)\}^T \{m_i\}}{\{r\}^T \{m_i\}}$

- ***The main entry point :*** **`Core_Equation_System`**

Although the other routines can be called separately (for example to evaluate the temperature or the ROB for a given amount of species), **`Core_Equation_System`** is certainly the one you will have to call if you try to use the code as a library. Just notice that here “main” does not signify the routine is a program directly runnable. This task is devoted to the driving module.

It is responsible for computing one iteration, i.e. evaluating the left members of the mass transfer equations (the derivatives of the masses vector). So that it is also the one called one -or several, depending on the method used- time by the solver.

As a consequence, I chose to use a prototype which is non obvious but that has the advantage to be in a standard form, as described by the SLATEC library. Then it is possible, if needed, to use easily another solver, or reciprocally, to use my RK2 solver for another purpose as it follows SLATEC standards for describing a problem (see II.3.B).

To summarize, we could say this routine composes the system of mass transfer equations (8), calling the **`MassTransfer`** routine for each zone with the right parameters. The exact steps are the following :

- Unpacking the U(15) vector into three masses vector (in kg) m_I(5), m_II(5), m_III(5). Vector U contains the masses stored with the following format :

$m_{O_2,I}$	$m_{N_2+Ar,I}$	$m_{CO_2,I}$	$m_{H_2O,I}$	$m_{fuel,I}$	$m_{O_2,II}$	$m_{N_2+Ar,II}$	$m_{CO_2,II}$	$m_{H_2O,II}$	$m_{fuel,II}$	$m_{O_2,III}$	$m_{N_2+Ar,III}$	$m_{CO_2,III}$	$m_{H_2O,III}$	$m_{fuel,III}$
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

- Evaluating the geometrical parameters for the current iteration.
- Computing the preliminary equations (1), (2) and (3).
- Calculating the mass transfer equation for each zone.
- Repacking the result into the UPRIME(15) vector (Same format as U, in kg/s).
- Sending back useful informations to the solver then to a file through RPAR(12)

The RPAR vector, returned after each iteration and stored in a table by the solver, contains essential informations :

```
RPAR(1) = Time
RPAR(2) = Angle
RPAR(3) = V_I
RPAR(4) = V_II
RPAR(5) = V_III
RPAR(6) = pressure
RPAR(7) = Temp_I
RPAR(8) = Temp_II
RPAR(9) = Temp_III
RPAR(10) = w_f_front
RPAR(11) = w_f_back
RPAR(12) = ROB
```

7. The driving module: ZoneApproach.f

The driving module is in fact, from the programmer's view, the executable routine (called “main” in C). As a consequence, it is the only file containing a PROGRAM block :

```
PROGRAM ZoneApproach
...
STOP
END
```

That is there everything begins when you call the executable file from your favorite operating system.

Operations processed by ZoneApproach :

- Declares variables to store the results
- Displays a welcome message
- Displays informations about geometry by calling printGeoModelInfos
- Calls the initial model to compute the initial masses vector :

```
CALL Initial_Model(Y0)
```

- Calls the solver (see below)
- Stores the result by calling routines contained in IOlib.c

- ***Default parameters passed to the solver***

As we previously said, two solvers are included in the program (and more could be used). In the version distributed with this report, the program should use the RK2 solver. But if you want to switch to the DEABM solver, uncomment the line :

```
c      CALL DEABMSolver(Core_Equations_System, 15, getTime(340.),
c     &getTime(365.6), timestep, Y0, RESULT, lines, diag)
```

and comment :

```
      CALL RK2Solver(Core_Equations_System, 15, getTime(340.),
     &getTime(365.6), timestep, Y0, RESULT, lines, diag)
```

Two last parameters are modifiable here :

- getTime(340.) defines the time when the integration starts
- getTime(365.6) defines when the integration stops

III. RESULTS OF A COMPUTATION

1. Compiling and running the program

- ***Compilation***

As Fortran is not an interpreted language, the first thing to do after each modification of the source code is to compile it. Except the C Iolib library, the entire project is supposed to be fully Fortran 77 compatible, so that every Fortran compiler should be able to produce a binary.

Anyway, I recommend to use the GNU GCC suite. It is a free and reliable toolbox including not only a Fortran compiler but also the famous gcc C compiler. Moreover, to make the things easier, I wrote a makefile that should make the process completely automated.

- Note for MS-Windows users

  Although the whole project have been designed under a Linux environment, it is possible to compile and run the program using MS-Windows. To do so, you will have to install Cygwin, which is a Linux-like shell. It can be found at the following address : http://www.cygwin.com
  When you install it, make sure to select (at least) the following packages : *gcc*, *gcc-core*, *gcc-g77*, *binutils*, *make*. Then the only thing you have to do is to run Cygwin, and a beautiful linux-like console should appear in front of you. Your hard drive is reachable in the directory */cygdrive/c*

  Anyway, once the program compiled, you do not need cygwin anymore to run it, just the windows dynamic library cygwin.dll (which means if you want to execute the binary on a computer where cygwin isn't installed you have to copy not only zone.exe but also cygwin.dll)

- Note for *nix users

  The compiler should already be installed on your computer if you are using a Linux/Unix environment. If it is not, ask you system administrator to do so. Moreover, if you are using the very last version of gcc (>3.4.6), the fortran compiler has been renamed from g77 to gfortran. As a consequence you will have to replace the line "*CC=g77*" by "*CC=gfortran*" inside all the makefiles, or to create a symlink from gfortran to g77 (probably the easiest way to do).

Now we can assume that the development environment is properly installed and configured. To compile the program, just go to the directory containing the source files (should be "*ZoneApproach*") and type "***make***", which should display the following report on the screen :

```
g77 -c ZoneApproach.f -Wall -g
g77 -c ReactionRate.f -Wall -g
g77 -c Datas.f -Wall -g
g77 -c Equations.f -Wall -g
g77 -c FileLoader.f -Wall -g
g77 -c Pressure.f -Wall -g
g77 -c Maths.f -Wall -g
g77 -c InitialModel.f -Wall -g
gcc -Wall -g   -c -o iolib.o iolib.c
make[1]: Entering directory `/mnt/fat32/tfe/Project/ZoneApproach'
gcc -c iolib.c -Wall -g
make[1]: Leaving directory `/mnt/fat32/tfe/Project/ZoneApproach'
make[1]: Entering directory `/mnt/fat32/tfe/Project/ZoneApproach/slatec'
g77 -c deabm.f -Wall -g
g77 -c defehl.f -Wall -g
g77 -c des.f -Wall -g
g77 -c hstart.f -Wall -g
g77 -c hvnrm.f -Wall -g
g77 -c qage.f -Wall -g
g77 -c qag.f -Wall -g
g77 -c qk15.f -Wall -g
g77 -c qk21.f -Wall -g
g77 -c qk31.f -Wall -g
g77 -c qk41.f -Wall -g
g77 -c qk51.f -Wall -g
g77 -c qk61.f -Wall -g
g77 -c qpsrt.f -Wall -g
g77 -c sintrp.f -Wall -g
g77 -c slatec-util.f -Wall -g
g77 -c steps.f -Wall -g
ar rc slatec_x86.a deabm.o defehl.o des.o hstart.o hvnrm.o qage.o qag.o qk15.o qk21.o qk31.o qk41.o
qk51.o qk61.o qpsrt.o sintrp.o slatec-util.o steps.o
rm -f *.o
********************************************************************************
Slatec library archive for x86 (slatec_x86.a) built ended.
This partial built of Slatec should be used with ZoneApproch Code only.
********************************************************************************
make[1]: Leaving directory `/mnt/fat32/tfe/Project/ZoneApproach/slatec'
make[1]: Entering directory `/mnt/fat32/tfe/Project/ZoneApproach/geometry'
g77 -c GeoCylinder.f -Wall -g
g77 -c GeoPiston.f -Wall -g
g77 -c GeoZones.f -Wall -g
g77 -c CrankAngle.f -Wall -g
g77 -c InitialGeom.f -Wall -g
g77 -c GeoDatas.f -Wall -g
ar rc geometry.a GeoCylinder.o GeoPiston.o GeoZones.o CrankAngle.o InitialGeom.o GeoDatas.o
../slatec/slatec_x86.a
rm -f *.o
********************************************************************************
This geometry is very simple, for test purposes.
It modelizes a simple cylinder and piston with purely spherical zones
********************************************************************************
make[1]: Leaving directory `/mnt/fat32/tfe/Project/ZoneApproach/geometry'
g77 -o zone ZoneApproach.o ReactionRate.o Datas.o Equations.o FileLoader.o Pressure.o Maths.o
InitialModel.o iolib.o geometry/geometry.a slatec/slatec_x86.a -Wall -g -Wl,-
rpath=/usr/local/lib64/
rm -f *.o
```

This will build the whole program : the main model, the IO library, the geometry and the SLATEC library. If you are not using and IBM-PC compatible computer, see *Appendix 3* to know how to configure the SLATEC library.

- ***Running a computation***

The compiler should have created an executable binary, either "*zone*" on a Linux system, or "*zone.exe"* on a Windows system, in the current directory.

To start a computation, be sure that the file *"pressure.dat"*, defining the values of the pressure depending on the crank angle is present in the current directory, and run the above binary. No command line parameter is admitted, as they are all defined inside the source code.

A short report of computation like the following one should be printed on screen.

```
[root@thew ZoneApproach]# ./zone
 ********************************************
    Zone Approach model for determination of
   premixed flame parameters computational code
                    .:::.
   Josef Bozek Research Center of Engine and
            Automotive Engineering
          July 2006 - Mathieu ALLORY
 ********************************************


 ##########   DUMMY ENGINE GEOMETRY   ##########
 ##########         Informations      ##########
 Cylinder Radius :   0.037750002m
 Cylinder Height :   0.0816000029m
 Crank Radius :  0.0360000022m
 Conrod Length :  0.13000001m
 Volume at BDC :  0.365320474L
 Compression ratio :  8.49999428

 Flame front width :  0.00300000003m
 Flame front speed :   25.m/s
 ################################################

 ########## FIRST STEP APPENDIX MODEL ##########
 ##########         Informations      ##########
 Pressure at inlet close :  1.04200006 bar
 Mean temperature at inlet close :  350. K
 Chamber volume at inlet close :  0.365320474 L

 One zone model :
   Total mass of mixture :  0.384230167 g
   Per specie in grams : (O2 - N2Ar - CO2 - H2O - fuel)
     0.0823260918  0.275613427  7.15879069E-05  0.00357939536  0.0226396751

 Three fake zones model :
   Zone I (Unburnt gas)
   Per specie in grams : (O2 - N2Ar - CO2 - H2O - fuel)
     0.0823260918  0.275556624  5.75946797E-05  0.0035717627  0.0226396751
   Zone II (Flame front)
   Per specie in grams : (O2 - N2Ar - CO2 - H2O - fuel)
     0.  5.6741439E-05  1.39827125E-05  7.62693389E-06  0.
   Zone III (Burnt gas)
   Per specie in grams : (O2 - N2Ar - CO2 - H2O - fuel)
     0.  4.26627373E-08  1.05133182E-08  5.73453729E-09  4.33680869E-16
 ################################################

 Starting integration process with a time step of   0.00555589423 ms
 [RK2SOLVER] Entering iteration  2(T=  0.)
 [RK2SOLVER] Entering iteration  3(T=  5.5558944E-06)
... One line displayed per iteration (cut here) ...
 [RK2SOLVER] Entering iteration  256(T=  0.00141120143)
 [RK2SOLVER] Entering iteration  257(T=  0.00141675735)
 [RK2SOLVER] Computation ended (  1.41675735ms)

 *** C output library for F77 : Saving results to results.txt
 *** Writting 259 lines.

 *** C output library for F77 : Saving results to diags.txt
 *** Writting 259 lines.
```

2. Analysis of the results

- ***The analysis sheet & general considerations***

A spreadsheet file is provided ("*results.ods*" for OpenOffice 2.0, "*results.xls*" for Excel), containing already defined graphs. You just have to open output files and to paste the values into the right tab to obtain up-to-date plots. Many of the following graphs are available in this file.

Moreover, except when it is specified, the following analysis and graphs have been made with the default values of the parameters, which have been defined in chapter *II.6.C)*.

Values for pressure are taken from measurements on a Skoda motor (*Figure 18*).

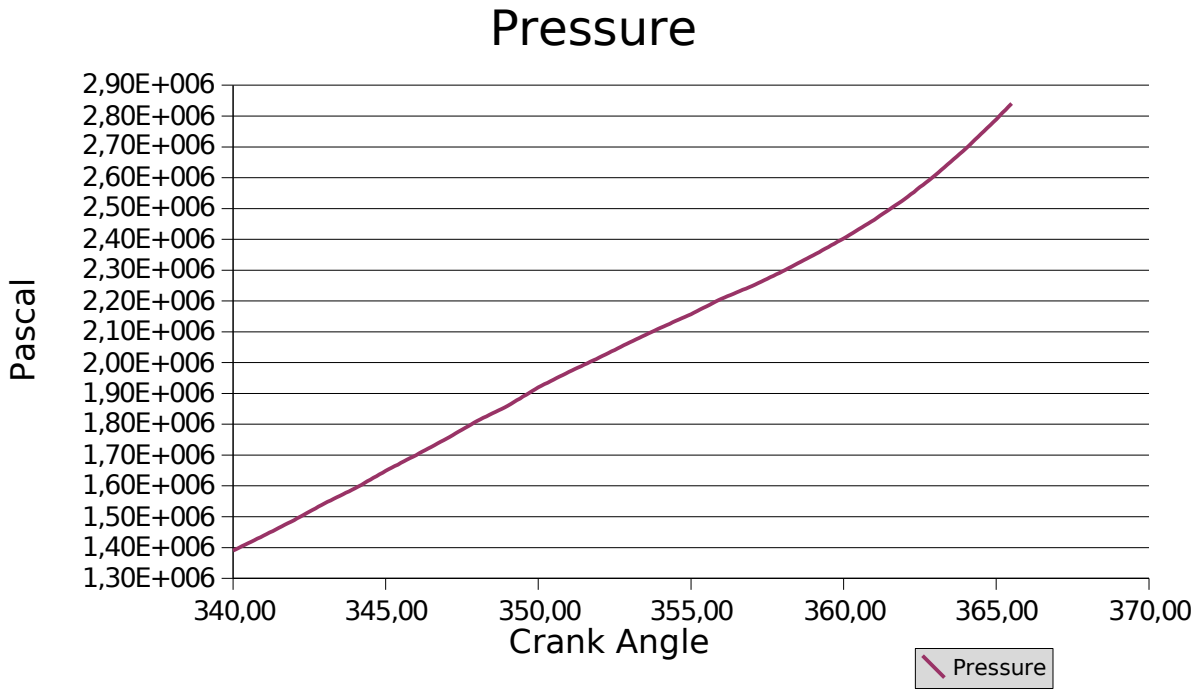


Figure 18: Pressure (input, from measurements)

- ***Without any reaction***

When disabling the ROB (by adding an assumption in `rdot_II_fuel`) we can observe on *Figure 19* that, just as expected, components are going mainly from zone I to zone III, passing by zone II, depending on the evolution of their volumes. As a consequence, the form of the diagram is close to the one representing the volume of the zones (*Figure 11*). That confirms the geometrical part of the model works : zones are correctly "fed" with components depending on their growth.

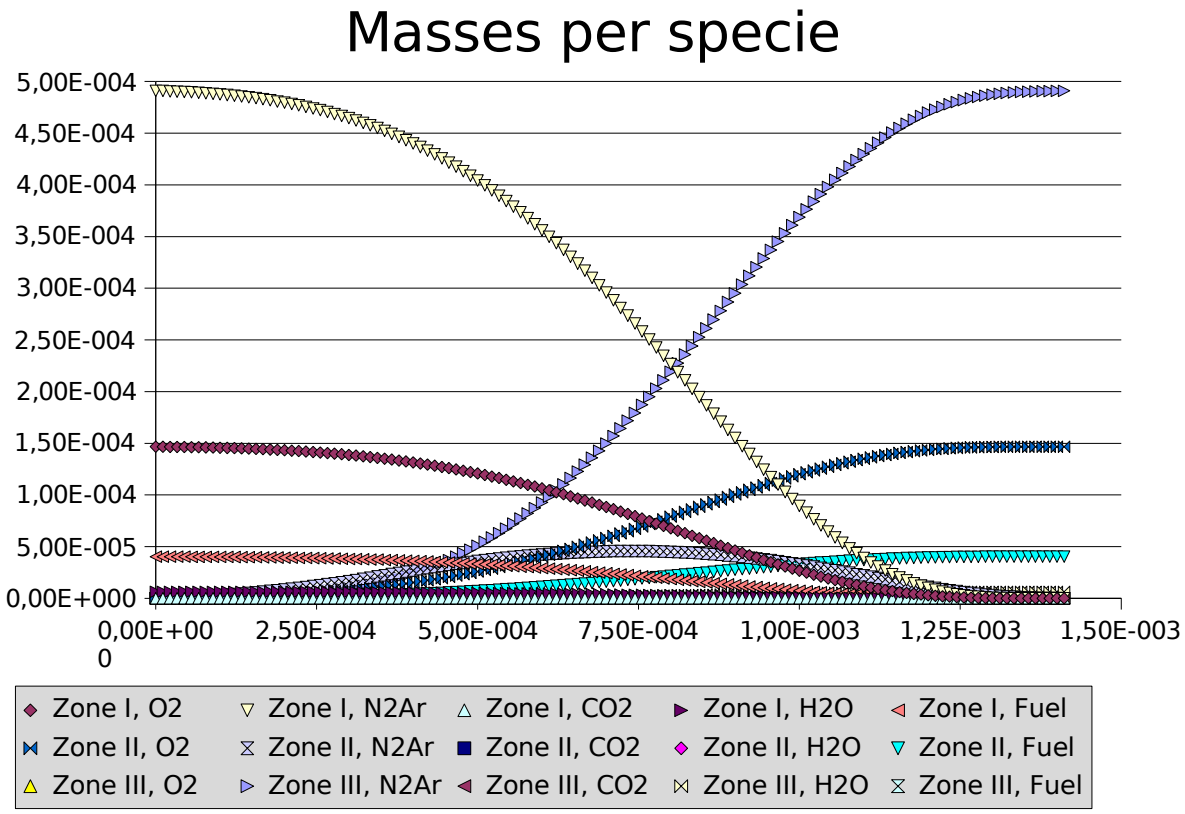


Figure 19: Masses of species in the three zones without any reaction

- ***Rate of burning***

The shape of the computed rate of burning (*Figure 20*) seems to be correct, as soon as we consider that the spark plug is situated in the center of the roof of the cylinder.

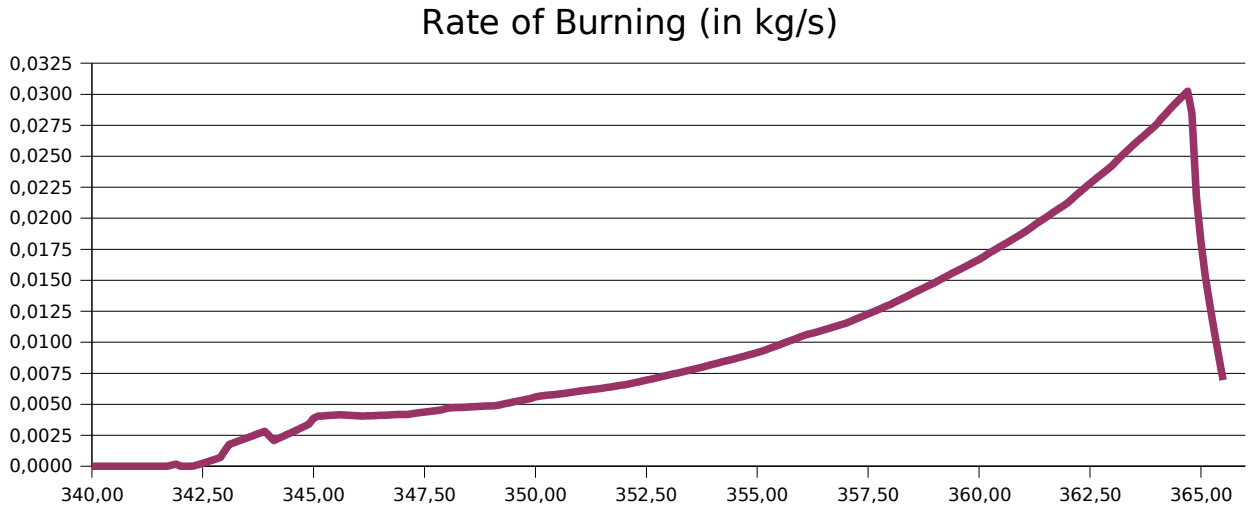


Figure 20: Rate of burning

As a comparison, we can consider the *Figure 21* : it has been plot using measurements from an Iranian Paykan engine. Even if it is not exactly the same engine, the cylinders have the same volume (1600cc), and the same speed is applied (3000rpm).

The curve to consider is marked with “0%” (degree of eccentricity of the spark plug).

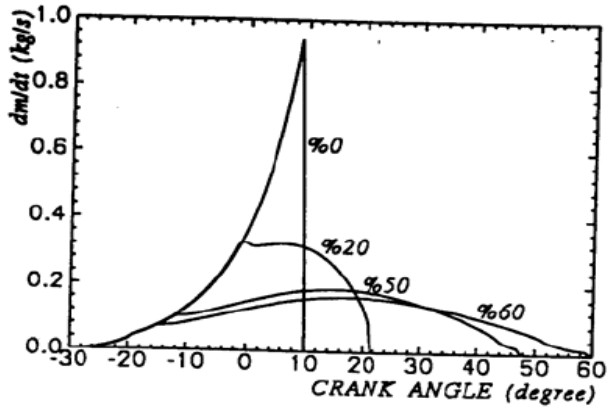


Figure 21: ROB in a Paykan engine

Quantitatively, even if the values have no reason for being exactly the same, the order can be compared : considering *Figure 20* is for only one cylinder in a four cylinders engine, we can observe that the difference is approximately a 10 factor. I do not know where this difference come from, but we can calculate that in our case it is most likely to be closer to the values calculated by our model. Anyway I cannot exclude there could be an error in the routine or formula computing the rate of burning.

Moreover, at the beginning of the computation ($342° < \alpha < 345°$) some unexplained variations appear, whereas the curve should be smoother. It seems to be due to numerical computation, as the same artifacts can be seen when solving with the other solver (DEABM), but not at the same place

Studying the rate of heat release is useless as ROB=ROHR here (the burning reaction is the only one enabled).

- ***Relative flame front velocity***

Figure 22 represents the evolution of front and back relative flame front velocity, i.e. $w_{f,front}, w_{f,back}$. What is interesting here is that quantitatively the order of the values is right (speed is in meters per second). This curve have been rescaled, and we can see that the value of $w_{f,back}$ at the very beginning is far too big (and at the very end for $w_{f,front}$, when $\alpha \approx 364.90°$, which is not visible here). At the beginning, values of (m_{II}) and $A_{f,II,III}$ are very low, even close to 0 so that could explain such a high value for $w_{f,back}$, as both are present at the denominator. At the end, zones I and II do not even exist anymore (zone I disappears when $\alpha \approx 365°$, zone II when $\alpha \approx 367°$) so the values here are not really significant.

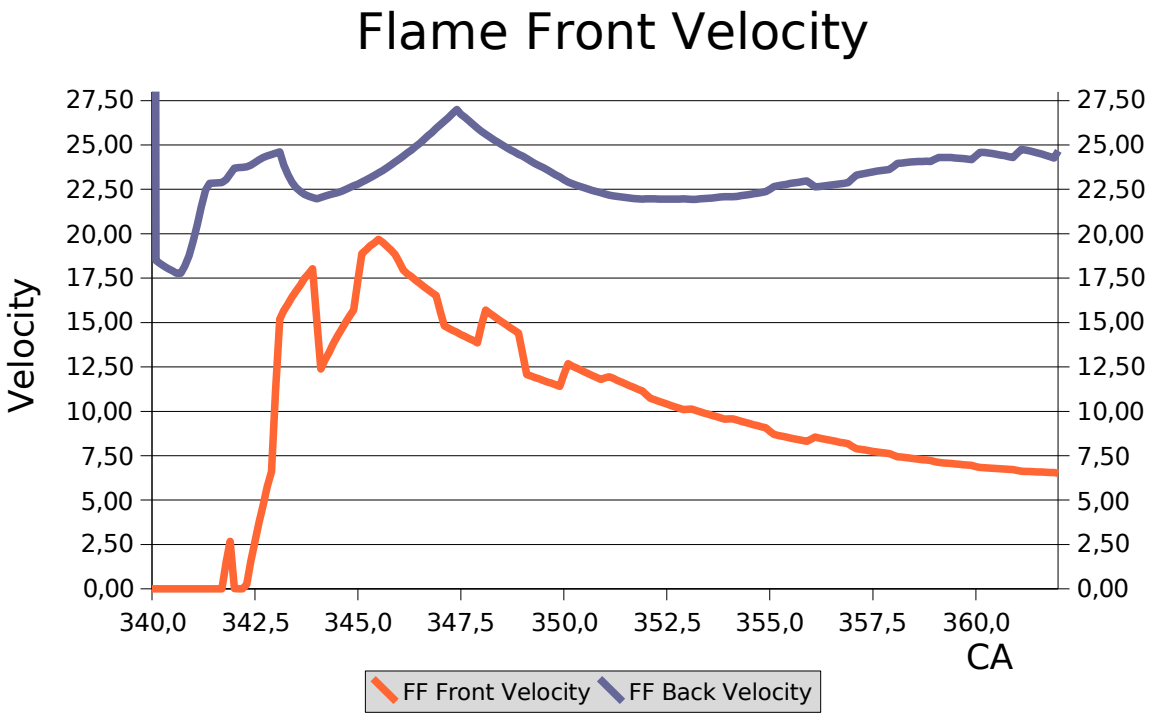


Figure 22: Relative flame front velocity

Moreover, the artifacts observed for the ROB are also present here, and even amplified especially for $w_{f,front}$.

- ***Temperatures***

Presently, as we can see on *Figure 23* and *Figure 24*, the temperature in zone one is pretty good. $706K < T_I < 830K$, which yields an increasing ratio of 18%, so that it is conform with what was expected.

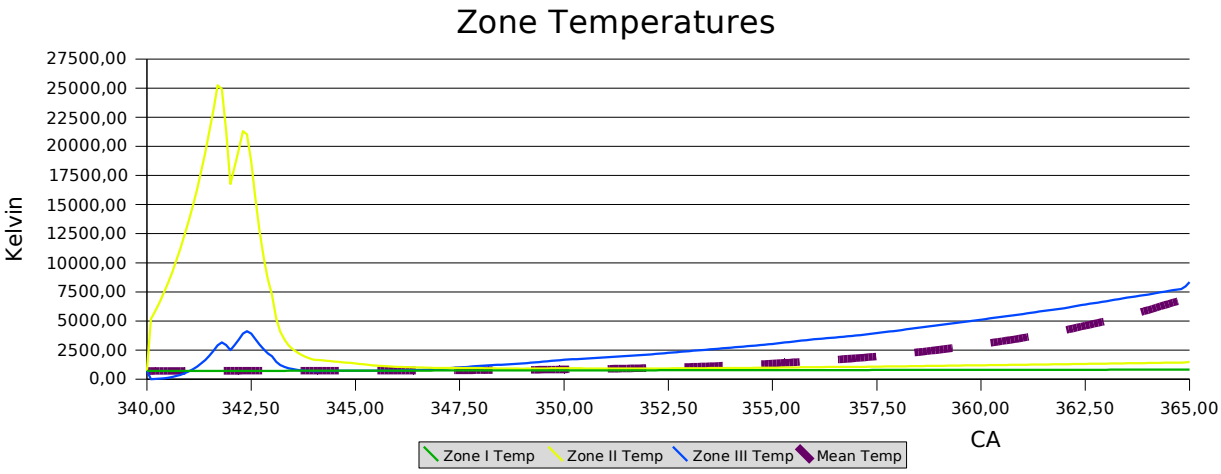


Figure 23: Zone temperatures

For zones II and III, there is obviously a problem. Even if we tried to lock them to T_I for the first iteration, they rapidly grow, reaching crazy values that are not significant at all. I had no time to deepen this problem, but it is probably here something have to be done first, as temperatures are used almost everywhere in the code.

However, I can say that as the same routine (then the same formula) is used for three zones, the error probably does not come from there, otherwise the temperature in zone I should be affected too (and it is not the case). Maybe the mess comes from the formula itself : $T_i = \frac{p V_i}{\{r\}^T \{m_i\}}$. At the beginning, as well V_i as m_i tend to zero in zones II and III, it yields an undetermined result. As solving is an iterative process, initial errors are then amplified more and more.

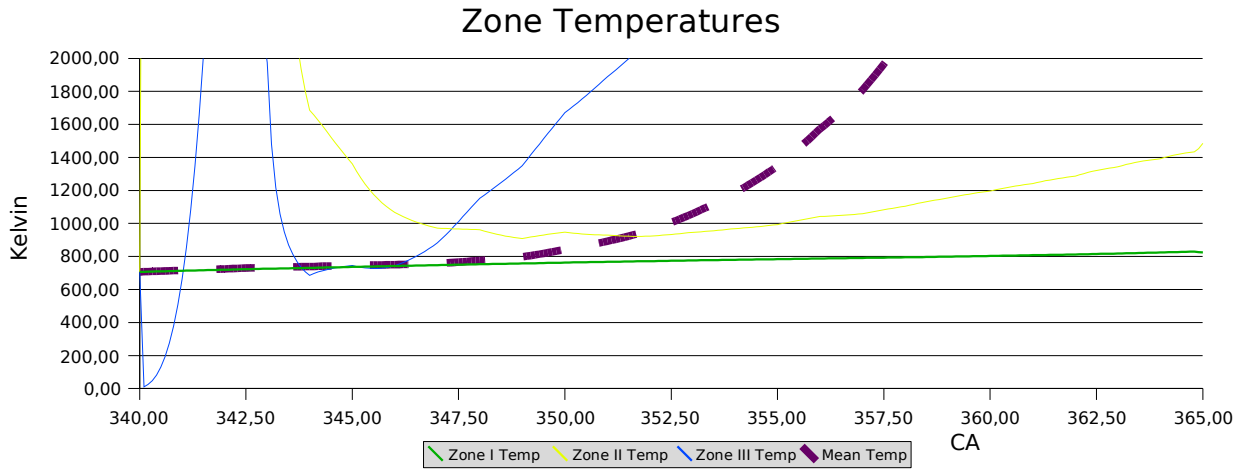


Figure 24: Zone temperatures (rescaled)

- ***Influence of the time step***

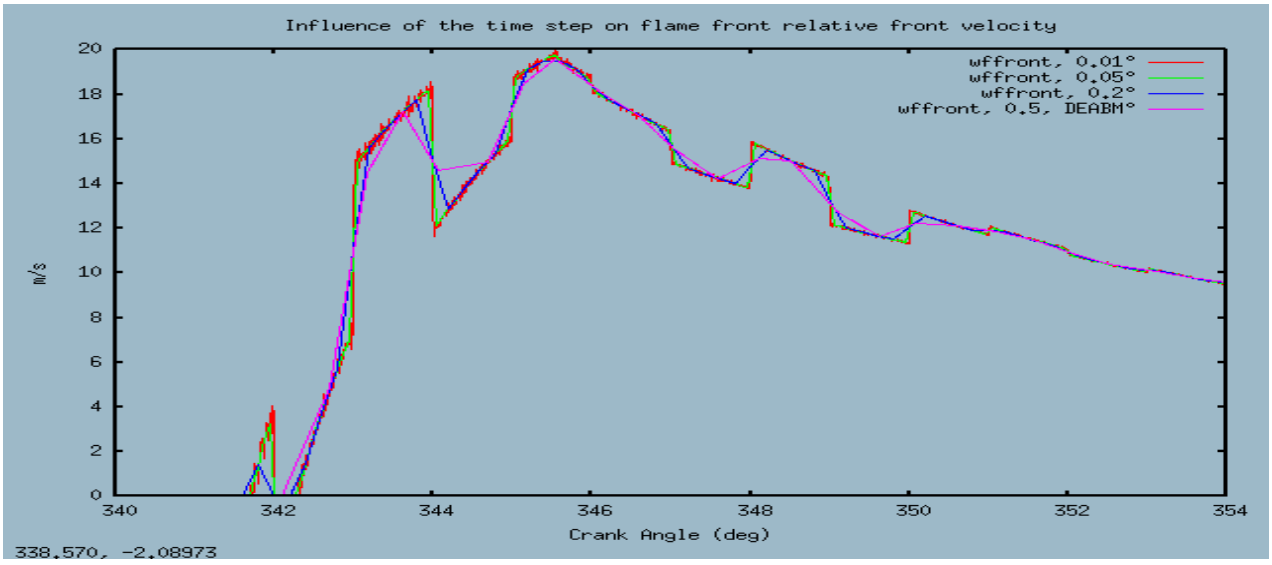


Figure 25.: Influence of the time step on flame front relative front velocity

Like every numerical problem, the time step chosen has an influence upon the results. When it is too big, the precision is decreased. When it is too small, the evolution between to iteration is not significant enough according to the precision of the input values (here the pressure).

Figure 25 and 26 show that nevertheless it does not seem to fundamentally change the results, in that it does not drastically improve or decrease the precision of the results. However, the chosen time step should be at least 0.05°, in that we can clearly observe that below this value, oscillations are occurring.

With a time step of 0.2° or greater, the curves seem to be smoother, but in fact we only have less points to plot. The present time step of 0.1° seem to be a right choice, at least with those solvers. Moreover, as opposed to what I thought, the influence of the time step do not allow to explain the presence of crenelations on *Figure 22*.

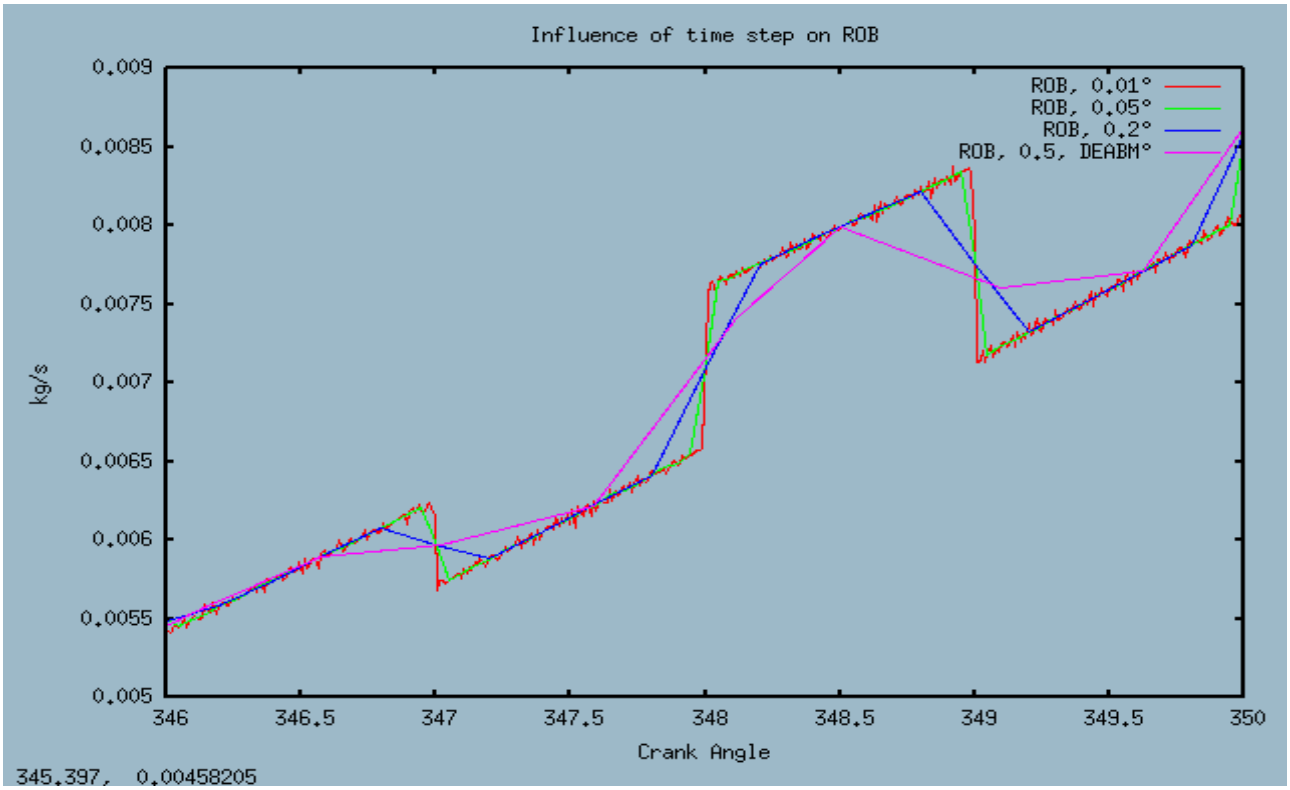


Figure 26.: Influence of the time step on the ROB

IV. FURTHER DEVELOPMENTS

As I said previously, the work on this this computational code is far from being completed. However, significant progresses have been made since I tried the program for the first time. Even if some bugs probably remain, we slowly moved from completely absurd results to something which is starting to look like more and more to an engine.

But now as I am not a specialist about internal combustion, I am reaching the limits of my basic knowledges. Nevertheless, I have identified a few points that could be improved, and some tracks to go further, either with the code or the model itself.

- Fix the problem with temperatures in zone II and III. This is probably where to start from, as almost every equations are highly temperature-dependent. Presently I do not know where the problem comes from, but I think it is more likely to come from the model than from the code. A first try could be done by adding a top limitation (corresponding to the adiabatic flame temperature), described in chapter II.6.A)

- Create a more realistic model for the flame front speed (the geometrical one, not $w_{f,front}$ & $w_{f,back}$). Presently, a constant value is used, which is maybe too far from reality. Moreover, modifying the code to introduce a time-dependent speed shouldn't be too difficult (by modifying `GeoDatas.f`).

- Study the influence of the initial size of zones II and III : even if it is obviously a nonsense to consider these zones are existing before ignition occurs, too small values of the surfaces, masses and volumes disturb the calculation by tending either to indeterminate forms, or to non significant numbers, even if this scenario seems not really relevant to me (the range allowed by REAL, single precision, type allows to store values from 10^{-37} to 10^{38}, with 7 significant digits).

- For the same reasons, it seems to me that the very first steps are introducing most of the errors. So that it could be interesting to imagine a kind of “multi-model”. For the first steps, the code would use a one-zone or two-zones model, then the three zone models would be applied. I do not really know what to expect from this idea but technically, it shouldn't be too difficult to implement, by adding at the very beginning of `core_equations_system` :

```
c   below limit_angle, another model is used
    IF ( T .LT. getTime(limit_angle) ) THEN
      CALL otherModel(T, U, UPRIME, RPAR, IPAR)
    ELSE
c   present code
    ...
    ENDIF
```

- Once the model is working, it will be necessary to implement the corrective reactions. Part of the work is already done but not enabled, and I tried to make the remaining part easily feasible by indicating inside the code (comments) where to script the necessary equations. Nevertheless, it will be necessary to study and to understand the code before.

- A theoretical study should also be done upon the precision needed for the input data, i.e. the pressure, and the data encoded inside `datas.f`. Presently, data used for evaluating Cp and Specific enthalpy are only 4 decimals precise in KJ, which means only 1 decimal precise in J, and I have no idea of neither the precision needed, nor even the precision requested for the output. However, between inputs and output, there shouldn't be any precision problem as the precision of REAL type is quite large (see above).

V. PERSONAL CONCLUSION

When I started this traineeship, of course I was really enjoying working and living in Prague. But I also knew that the task would not be easy. Indeed, If the theoretical part had already been studied before, the computational translation had to be built entirely from scratch. Moreover, when I addressed the project, first I did not know much about Fortran so I had to learn, and as I am not an engine specialist, I also had to update my knowledges in energetics.

In spite of -or perhaps thanks to- these difficulties, I found this project really interesting and motivating. What I appreciated more was the fact that many fields of knowledge were mixed in the same subject : computer science and programming of course, but also numerical mathematics, energetics and project management, to make sure that the things were done in the best way. About this last point, I must add that the organization of such a project is a task by itself. Indeed, there were so many things to do and in the right order, that it was necessary to plan the development in advance : for example, the geometrical library had to be programmed before the general code, but the Runge Kutta solver could be designed concurrently.

Another thing that was very exciting was the feeling of creating something new, of doing something that had never been done before. Although this could have been wrong, it helped me to find the motivation to go ahead. And at the end I must admit this is very gratifying to obtain something that -almost- works.

To conclude this report, I can say that I think that the project, and the contacts I had consequently with Pr. Jan MACEK, really improved my capability to quickly seize the important points in scientific documents which I do not always understand completely. Moreover he took time to explain me the concepts of thermodynamics which I had forgotten or badly understood, so that without being a specialist about internal combustion, I think that I now know more about it than ever. Finally, this traineeship reinforced my ability to organize myself and my work in a complete autonomy, which I had already learned from previous professional experiences.

TABLE OF ROUTINES

APPENDICES

APPENDIX I : NOTATIONS

The following notations are almost the same as those of the source theoretical document, written by Pr. MACEK.

- ***Symbols***

$\boldsymbol{A}$	[m^2]	surface area
$\|\|\boldsymbol{C}\|\|$	[1]	matrix of stoichiometric coefficients normalized acccording to the main reaction component
$\boldsymbol{c_p}$	[$J.kg^{-1}.K^{-1}$]	isobaric specific heat capacity
$\boldsymbol{H_u}$	[$J.kg^{-1}$]	calorific value of fuel
$\boldsymbol{i}$	[$J.kg^{-1}$]	specific enthalpy
$\boldsymbol{L_{O2}}$	[1]	stoichiometric mixing ratio for oxygen/fuel
$\boldsymbol{m}$	[kg]	mass, mass of specie
$\boldsymbol{p}$	[Pa]	pressure
$\boldsymbol{Q}$	[J]	transferred heat
$\boldsymbol{r}$	[$J.kg^{-1}.K^{-1}$]	specific gas constant
$\dot{r}$	[$kg.s^{-1}$]	reaction rate
$\boldsymbol{S}$	[m^2]	heat transfer surface
$\boldsymbol{T}$	[K]	temperature
$\boldsymbol{t}$	[s]	time
$\boldsymbol{V}$	[m^3]	volume
$\boldsymbol{w}$	[$m.s^{-1}$]	velocity of convective or diffusion flow
α	[$W.m^{-2}.K^{-1}$]	heat transfer coefficient
κ	[1]	isentropic exponent

- ***Indices***

back	backward face of flame
f	flame
CH	cooling; heat transfer to cooled wall
i or I, II, III	current zone
j	neighboring zone
r	all reactions
s	all species
{}^{T}	transposed (row) vector
x	arbitrarily specie
$\dot{\bullet}$	time derivative; flux
$\{_N V\}$	column vector of size N
{}	column vector, of default size (here five)

- ***Abbreviations***

ROB	Rate Of Burning (rate of fuel entering combustion reaction)
ROHR	Rate Of Heat Release
RK2	Runge Kutta of the 2nd order
DEABM	Differential Equations Adams Bashforth Method
TDC	Top dead center
BDC	Bottom dead center

Appendix II : Files per module

Geometrical Model

- ✔ Inside the directory : sources/geometry

```
2050        jui 13 11:25     CrankAngle.f
3700        jui 14 13:38     GeoCylinder.f
2737        aoû 14 19:05     GeoDatas.f
2392        jui 14 13:23     GeoPiston.f
21694       aoû 15 13:59     GeoZones.f
927         jui 21 16:19     InitialGeom.f
4046        aoû 14 18:53     Test_geom.f
```

Main computational code

- ✔ Inside the directory : sources/

```
22718       aoû 24 17:46     Equations.f
2727        jui 26 16:30     ReactionRate.f
```

First-step Model

- ✔ Inside the directory : sources/

```
7145        aoû 22 18:07     InitialModel.f
```

IO Libraries

- ✔ Inside the directory : sources/

```
1851        aoû  4 13:54     iolib.c
1488        jui 13 11:25     FileLoader.f
3140        jui 13 11:25     Pressure.f
```

Static data

- ✔ Inside the directory : sources/

```
7985        sep  2 21:14     Datas.f
```

Solvers & Math tools

- ✔ Inside the directory : sources/

```
14310       aoû 24 14:34     Maths.f
```

Driving module

- ✔ Inside the directory : sources/

```
2194        sep  2 21:14     ZoneApproach.f
```

SLATEC : see *Appendix 3*

APPENDIX III : SLATEC

- ***What is SLATEC ?***

SLATEC is a bunch of high-portable, high-performance mathematical routines written in Fortran 77 : *"The SLATEC CML is written in FORTRAN 77 (ANSI standard FORTRAN as defined by ANSI X3.9-1978, reference [6]) and contains general purpose mathematical and statistical routines."* Moreover, the entire library is the public domain. The first version has been released by (and is an acronym for) the Sandia, Los Alamos, (Air Force Weapons Laboratory) Technical Exchange Committee.

Several distributions can be found on the Internet :

- http://www.netlib.org/slatec/
- http://www.ictp.trieste.it/cgi-bin/ICTPslatec/www-slatec.pl : also contains useful informations about how to use the library

- ***This distribution of SLATEC***

The whole library contains more than a hundred routines and as many files. As I only use three routines, I built a "special" version of the library containing only the files needed to satisfy the dependencies. I also created a `makefile` which builds the library and stores it inside an archive file.

Here is the list of the files that can be found in the directory `sources/slatec` :

`deabm.f`
`defehl.f`
`des.f`
`hstart.f`
`hvnrm.f`
`qage.f`
`qag.f`
`qk15.f`
`qk21.f`
`qk31.f`
`qk41.f`
`qk51.f`
`qk61.f`
`qpsrt.f`
`sintrp.f`
`steps.f`
	Mathematical routines

`slatec-util.f`	Machine dependent constants
`slatec_x86.a`	Archive file containing the compiled version of the library
`makefile`	Make file

- ***Configuration***

The library can be compiled on a large range of machines. However, the version provided here is configured to be compiled and executed with an Intel x86 compatible processor (including 32bits and 64bits series).

If you want to use another machine, like a Sparc, an Apple with a PowerPC processor, or even oldies (VAX an others), you will have to modify `slatec-util.f` according to your architecture. Otherwise, the SLATEC routines could return wrong results.

The source file is well explained, and there are three blocks to modify (lines 218, 588 & 1218). For example, if you want to use a Sun Sparc, comment the following lines :

```
C     MACHINE CONSTANTS FOR THE IBM PC

      DATA SMALL(1) / 1.18E-38      /
      DATA LARGE(1) / 3.40E+38      /
      DATA RIGHT(1) / 0.595E-07     /
      DATA DIVER(1) / 1.19E-07      /
      DATA LOG10(1) / 0.30102999566 /
```

And uncomment :

```
C     MACHINE CONSTANTS FOR THE SUN

C     DATA RMACH(1) / Z'00800000' /
C     DATA RMACH(2) / Z'7F7FFFFF' /
C     DATA RMACH(3) / Z'33800000' /
C     DATA RMACH(4) / Z'34000000' /
C     DATA RMACH(5) / Z'3E9A209B' /
```

And do the same at line 588 and 1218.

Moreover, if you want to use a version of SLATEC downloaded from the Internet, do not forget to check that the library if configured for your machine. Even if PCs are very widespread nowadays, the version I downloaded was initially configured for a DEC RISC computer !

APPENDIX IV : SOURCE CODE

Date of the following source codes : September 21th, 2006

Table of the available source codes :

The geometrical model : directory /ZoneApproach/sources/geometry

CrankAngle.f

```
      REAL FUNCTION CrankAngle (time)

c=======================================================================
c  FUNCTION CrankAngle
c=======================================================================
c  Author : Mathieu ALLORY - CVUT U2201 - April 2006
c
c  Abstract : returns the crank angle in radius from time. This is a
c             simple implementation assuming the rotation speed is
c             constant during a cycle
c
c  Inputs :
c    REAL time : the time of which you want the crank angle
c  Outputs :
c    RETURN REAL : the crank angle value in radius
c  Parameters :
c    REAL speed : the speed of the motor, in cycles per second
c
c=======================================================================
c     7 Start Column
      IMPLICIT NONE

c     Variables
      REAL time
      REAL deltaD
      REAL speed
      REAL angle_SOI

c     Retreives engine speed
      CALL getParameter_engine_speed(speed)
      CALL getIntegrationParameter_angle_SOI(angle_SOI)

      deltaD = time*speed/60.*360.

      CrankAngle = MOD(deltaD + angle_SOI, 720.)

c     If we have a negative value, it need to be readjusted
      IF (CrankAngle .LT. 0.) THEN
            CrankAngle = 720. + CrankAngle
      ENDIF

      RETURN
      END


c     Inverse function (mainly for test purposes)
c     Which means that angle = getAngle(getTime(angle))
c     It allows to trace diagrams driven by angle value instead of time
      REAL FUNCTION getTime(angle)
      IMPLICIT NONE
      REAL angle
      REAL speed
```

CrankAngle.f

```
      REAL angle_SOI

c     Retreives engine speed
      CALL getParameter_engine_speed(speed)
      CALL getIntegrationParameter_angle_SOI(angle_SOI)
      getTime = (angle - angle_SOI) / 360. / speed * 60.
      RETURN
      END


c     Just an alias
      FUNCTION getAngle(time)
      REAL time
      getAngle = CrankAngle(time)
      RETURN
      END
```

GeoCylinder.f

```
      SUBROUTINE GeoCylinder (R, H)

c=======================================================================
c  SUBROUTINE GeoCylinder
c=======================================================================
c  Author : Mathieu ALLORY - CVUT U2201 - April 2006
c
c  Abstract : Specifications and routines for the geometry
c             of the Cylinder
c
c  Inputs :
c  Outputs :
c     REAL R : Chamber radius (in meters)
c     REAL H : Maximum height of the chamber (in meters), ie when
c              Angle = 180deg (piston is down)
c  Parameters
c
c=======================================================================
c     7 Start Column
      IMPLICIT NONE
      REAL R,H
      REAL CrankRadius, ConrodLength,
     &flameFrontWidth, radius_expension_coef

      CALL GeoDatas(R, H, CrankRadius,
     &ConrodLength, flameFrontWidth, radius_expension_coef)

      RETURN
      END


      REAL FUNCTION getChamberVolume(Angle)
c=======================================================================
c  FUNCTION getPistonPosition
c=======================================================================
c  Author : Mathieu ALLORY - CVUT U2201 - April 2006
c
c  Abstract : returns the actual volume of the combustion chamber
c             This is a very simple case where the combustion chamber
c             is considered as a simple cylinder
c
c  Inputs :
c     REAL Angle : Crank angle (in degrees)
c  Outputs :
c     REAL : The volume of the combustion chamber
c  Parameters :
c     Pi
c
c=======================================================================
c     7 Start Column
      IMPLICIT NONE
      REAL H, Angle, R, Pi, V, Dummy
      REAL getCurrentChamberHeigth

      PARAMETER (Pi = 3.14159265)

      CALL GeoCylinder(R, Dummy)
```

GeoCylinder.f

```
      H = getCurrentChamberHeigth(Angle)

c     The volume of a cylinder... so easy !
      V = H * Pi * R**2

      getChamberVolume = V
      RETURN
      END

C=======================================================================
c  FUNCTION getMinCylinderHeigth
C=======================================================================
c  Author : Mathieu ALLORY - CVUT U2201 - July 2006
c
c  Abstract : returns the heigth of the combustion chamber
c                     when BDC is reached (when piston stroke is max)
c
c  Outputs :
c     REAL : Heigth of the chamber at BDC
c
C=======================================================================
c     7 Start Column
      FUNCTION getMinCylinderHeigth()
      IMPLICIT NONE
      REAL getMinCylinderHeigth
      REAL R,H
      REAL getPistonStroke
      CALL GeoCylinder(R, H)
      getMinCylinderHeigth  = H - getPistonStroke(180.)
      RETURN
      END

C=======================================================================
c  FUNCTION getCurrentChamberHeigth
C=======================================================================
c  Author : Mathieu ALLORY - CVUT U2201 - July 2006
c
c  Abstract : returns current heigth of combustion chamber.
c                    ie heigth(TDC) + piston stroke
c
c  Outputs :
c     REAL : Current heigth of the combustion chamber
c
C=======================================================================
c     7 Start Column
      FUNCTION getCurrentChamberHeigth(Angle)
      IMPLICIT NONE
      REAL getCurrentChamberHeigth
      REAL getMinCylinderHeigth
      REAL getPistonStroke
      REAL Angle
      getCurrentChamberHeigth  = getMinCylinderHeigth() +
     &getPistonStroke(Angle)
      RETURN
      END
```

GeoDatas.f

```
c=======================================================================
c  SUBROUTINE GeoDatas
c=======================================================================
c  Author : Mathieu ALLORY - CVUT U2201 - April 2006
c
c  Abstract : Specifications for the geometry of the dummy cylinder
c             model. Those CAN BE MODIFIED, and are taken from Skoda
c             781.135 Z92-15 engine specifications
c
c
c=======================================================================
      SUBROUTINE GeoDatas(CylinderRadius, CylinderHeight, CrankRadius,
     &ConrodLength, flameFrontWidth, radius_expension_coef)
      REAL CylinderRadius, CrankRadius, ConrodLength, CylinderHeight,
     &flameFrontWidth, radius_expension_coef

c     Cylinder Specifications (in meters)
      CylinderRadius = 75.5 / 2. * 10.**(-3.)
      CylinderHeight = 0.0816

c     Crank & Conrod specifications (in meters)
      CrankRadius = 36. * 10.**(-3.)
      ConrodLength = 130. * 10.**(-3.)

c     Flame front modeling
c     in meters
      flameFrontWidth = 0.003
c     in m/s
      radius_expension_coef = 25.

      RETURN
      END


c     This function calculate the engine compression ratio.
      FUNCTION calcCompRatio()
      IMPLICIT NONE
      REAL calcCompRatio
      REAL getChamberVolume
c     In this model, don't forget that BDC occurs at 180deg and
c     TDC at 360deg
      calcCompRatio = getChamberVolume(180.) / getChamberVolume(360.)
      RETURN
      END


      SUBROUTINE printGeoModelInfos()
      IMPLICIT NONE
      REAL CylinderRadius, CrankRadius, ConrodLength, CylinderHeigth,
     &flameFrontWidth, radius_expension_coef
      REAL calcCompRatio,  getChamberVolume

c     Retreives engine settings
      CALL GeoDatas(CylinderRadius, CylinderHeigth, CrankRadius,
     &ConrodLength, flameFrontWidth, radius_expension_coef)
      WRITE(6,*)''
```

GeoDatas.f

```
      WRITE(6,*)'##########   DUMMY ENGINE GEOMETRY   ##########'
      WRITE(6,*)'##########        Informations       ##########'
      WRITE(6,*)'Cylinder Radius : ', CylinderRadius, 'm'
      WRITE(6,*)'Cylinder Height : ', CylinderHeigth, 'm'
      WRITE(6,*)'Crank Radius :', CrankRadius, 'm'
      WRITE(6,*)'Conrod Length :', ConrodLength, 'm'
      WRITE(6,*)'Volume at BDC :', getChamberVolume(180.) * 10. ** (3.)
     &, 'L'
      WRITE(6,*)'Compression ratio :', calcCompRatio()
      WRITE(6,*)''
      WRITE(6,*)'Flame front width :', flameFrontWidth, 'm'
      WRITE(6,*)'Flame front speed : ', radius_expension_coef, 'm/s'
      WRITE(6,*)'################################################'

      RETURN
      END
```

GeoPiston.f

```
      SUBROUTINE GeoPiston (G, R)

c=======================================================================
c  SUBROUTINE GeoPiston
c=======================================================================
c  Author : Mathieu ALLORY - CVUT U2201 - April 2006
c
c  Abstract : Specifications for the geometry of the piston
c
c  Inputs :
c  Outputs :
c     REAL G : Crank radius (in meters)
c     REAL R : Rod size (in meters)
c  Parameters
c
c=======================================================================
c     7 Start Column
      IMPLICIT NONE
      REAL G, R
      REAL CylinderRadius, CylinderHeigth,
     &flameFrontWidth, radius_expension_coef

      CALL GeoDatas(CylinderRadius, CylinderHeigth, G, R,
     &flameFrontWidth, radius_expension_coef)

      RETURN
      END


      REAL FUNCTION getPistonPosition(Angle)
c=======================================================================
c  FUNCTION getPistonPosition
c=======================================================================
c  Author : Mathieu ALLORY - CVUT U2201 - April 2006
c
c  Abstract : returns piston position from the crank angle.
c             This example uses a very simple geometry that should be
c             rewritten
c             Crank gear is considered as a circle linked to the piston
c             with a simple rod
c
c  Inputs :
c     REAL Angle : Crank angle (in degrees)
c  Outputs :
c     REAL : The distance of the piston between the point where Angle
c            is 180deg (BDC)  and now
c  Parameters :
c     Pi
c=======================================================================
c     7 Start Column

      REAL G, d, Angle, AngleRad, R, Pi
      REAL Beta, H, Dummy

      PARAMETER (Pi = 3.1415926)

      CALL GeoPiston(G, R)
```

GeoPiston.f

```
      CALL GeoCylinder(Dummy, H)

c     Angle has to be converted in radians
      AngleRad = (Angle) * Pi / 180.0

      Beta = - ASIN((G * SIN(AngleRad)) / R)
      d = G * COS(AngleRad) + R * COS(Beta)
      getPistonPosition = d
      RETURN
      END


      REAL FUNCTION getPistonStroke(Angle)
      IMPLICIT NONE
      REAL Angle
      REAL getPistonPosition
      getPistonStroke = getPistonPosition(360.) -
     &getPistonPosition(Angle)

      RETURN
      END
```

GeoZones.f

```
c=======================================================================
c  Library GeoZones
c=======================================================================
c  Author : Mathieu ALLORY - CVUT U2201
c  Date : May, 19th 2006, updated July, 14th 2006
c=======================================================================
c  Abstract : Those routines compute zone geometry (volume, area...)
c  Needs : getIntegrationParameters_angle_SOI
c  Summary :
c    * Internal
c           computeSphereVolume(Angle, r)
c           calculateSphereArea(r, hCh, rCh)
c           toIntegrate(z)
c           calculateSphereZRadius(z)
c           linear(z)
c    * Used by the code (must be present on another version of the
c      library)
c           getZoneVolume_I(t)
c           getZoneVolume_II(t)
c           getZoneVolume_III(t)
c           getContactArea_I_II(t)
c           getContactArea_II_III(t)
c           getSurface_I_CHAMBER(t)
c           getSurface_II_CHAMBER(t)
c           getSurface_III_CHAMBER(t)
c=======================================================================
c Parameters for this file (pre-computed or on-the-fly) :
c     REAL flameFrontWidth : The width of the flamme front, considered
c                            constant (in meters)
c     REAL radius_expension_coef : the speed of the flamme front (m/s)
c=======================================================================
      SUBROUTINE GeoZones (flameFrontWidth, radius_expension_coef)
      IMPLICIT NONE
      REAL flameFrontWidth
      REAL radius_expension_coef
      REAL CylinderRadius, CrankRadius, ConrodLength, CylinderHeigth

      CALL GeoDatas(CylinderRadius, CylinderHeigth, CrankRadius,
     &ConrodLength, flameFrontWidth, radius_expension_coef)

      RETURN
      END


c=======================================================================
c  FUNCTIONS getSurface_I_CHAMBER, getSurface_II_CHAMBER
c            getSurface_III_CHAMBER
c=======================================================================
c  Author : Mathieu ALLORY - CVUT U2201 - April 2006
c
c  Abstract : Returns the contact surface (in square meters) between
c             zones and the combustion chamber (excluding the piston)
c
c  Inputs :
c     REAL t : time when you want the surface
c  Outputs :
```

GeoZones.f

```
c     REAL : Contact surface in m?
c
c=======================================================================
c     7 Start Column
      REAL FUNCTION getSurface_III_CHAMBER(time)
      IMPLICIT NONE
c     Input
      REAL time
c     Internals
      REAL ChamberRadius, MaxHeigthChamber, r, hCh, h
      REAL result_r, result_w
      REAL Pi
c     Functions
      REAL getRadius_III, getAngle
      REAL getCurrentChamberHeigth
      PARAMETER (Pi = 3.14159265)

      result_r = 0.
      result_w = 0.
      r = getRadius_III(time)
      CALL GeoCylinder(ChamberRadius, MaxHeigthChamber)
      hCh = getCurrentChamberHeigth(getAngle(time))

c     The contact surface is the sum of contact surface with the roof
c     (disc) and with the walls (cylinder)
c     First part : The roof (always present)
      IF (r .GT. ChamberRadius) THEN
        result_r = Pi * ChamberRadius**2
      ELSE
        result_r = Pi * r**2
      ENDIF

c     Second part if necessary : the walls
      IF (r .GT. ChamberRadius) THEN
        h = SQRT(r**2 - ChamberRadius**2)
c     Of course, the heigth of the wall cannot be greater than the
c     heigth of the chamber
        IF (h .GT. hCh) THEN
          h = hCH
        ENDIF
        result_w = 2.*Pi*ChamberRadius*h
      ENDIF

      getSurface_III_CHAMBER = result_r + result_w
      RETURN
      END


      REAL FUNCTION getSurface_II_CHAMBER(time)
      IMPLICIT NONE
c     Input
      REAL time
c     Internals
      REAL ChamberRadius, MaxHeigthChamber, r2, r3, hCh, h
      REAL result_r, result_w
      REAL Pi
```

GeoZones.f

```
c     Functions
      REAL getRadius_III, getRadius_II, getAngle
      REAL getCurrentChamberHeigth
      PARAMETER (Pi = 3.14159265)

      result_r = 0.
      result_w = 0.
      r3 = getRadius_III(time)
      r2 = getRadius_II(time)
      CALL GeoCylinder(ChamberRadius, MaxHeigthChamber)
      hCh = getCurrentChamberHeigth(getAngle(time))

c     The contact surface is the sum of contact surface with the roof
c     (disc) and with the walls (cylinder)

c     First part : The roof
      IF (r2 .GT. ChamberRadius) THEN
        IF (r3. GE. ChamberRadius) THEN
c         In this case, zone 2 does not even touch the roof
          result_r = 0.
        ELSE
          result_r = Pi * ChamberRadius**2 - Pi * r3**2
        ENDIF
      ELSE
c       We must substract the surface occupied by zone 3
        result_r = Pi * r2**2 - Pi * r3**2
      ENDIF

c     Second part if necessary : the walls
      IF (r2 .GT. ChamberRadius) THEN
        IF (r3 .GT. ChamberRadius) THEN
          h = SQRT(r2**2-ChamberRadius**2)-SQRT(r3**2-ChamberRadius**2)
        ELSE
          h = SQRT(r2**2-ChamberRadius**2)
        ENDIF
c     Of course, the heigth of the wall cannot be greater than the
c     heigth of the chamber
        IF (SQRT(r2**2-ChamberRadius**2) .GT. hCh) THEN
           IF (SQRT(r3**2-ChamberRadius**2) .GT. hCh) THEN
c            In this case, zone 2 does not exist anymore or so
             h = 0.
           ELSE
c            Else, its borders are those of the chamber
             h = hCh - SQRT(r3**2-ChamberRadius**2)
           ENDIF
        ENDIF
        result_w = 2.*Pi*ChamberRadius*h
      ENDIF

      getSurface_II_CHAMBER = result_r + result_w
      RETURN
      END


      REAL FUNCTION getSurface_I_CHAMBER(time)
      IMPLICIT NONE
```

GeoZones.f

```
      REAL time
c     Internals
      REAL s3, s2
      REAL ChamberRadius, MaxHeigthChamber, hCh
      REAL Pi
c     Functions
      REAL getSurface_III_CHAMBER, getSurface_II_CHAMBER
      REAL getAngle
      REAL getCurrentChamberHeigth

      PARAMETER (Pi = 3.14159265)
      s3 = getSurface_III_CHAMBER(time)
      s2 = getSurface_II_CHAMBER(time)
      CALL GeoCylinder(ChamberRadius, MaxHeigthChamber)
      hCh = getCurrentChamberHeigth(getAngle(time))
c     The exchange surface for zone 1 is just the free cylinder surface
c     minus exchange surfaces of zones 2 & 3
      getSurface_I_CHAMBER = Pi * ChamberRadius**2
     &+ 2.*Pi*ChamberRadius*hCh
     &- s2 - s3
      RETURN
      END



c=======================================================================
c  FUNCTIONS getZoneVolume_I, getZoneVolume_II, getZoneVolume_III
c=======================================================================
c  Author : Mathieu ALLORY - CVUT U2201 - April 2006
c
c  Abstract : Returns the volume of a zone (I, II or III)
c
c  Inputs :
c     REAL t : time when you want the volume of the zone
c  Outputs :
c     REAL : The volume of the zone
c
c=======================================================================
c     7 Start Column
      REAL FUNCTION getZoneVolume_III(t)
      IMPLICIT NONE
      REAL t
      REAL computeSphereVolume, getAngle, getRadius_III
      getZoneVolume_III = computeSphereVolume(getRadius_III(t),
     &getAngle(t))
      RETURN
      END


      REAL FUNCTION getZoneVolume_II(t)
      IMPLICIT NONE
      REAL t
      REAL computeSphereVolume, getAngle, getRadius_II
      REAL getZoneVolume_III
      getZoneVolume_II = computeSphereVolume(getRadius_II(t),
     & getAngle(t)) - getZoneVolume_III(t)
```

GeoZones.f

```
      RETURN
      END


      REAL FUNCTION getZoneVolume_I(t)
      IMPLICIT NONE
      REAL t
      REAL getAngle, getChamberVolume
      REAL getZoneVolume_III, getZoneVolume_II
      getZoneVolume_I = getChamberVolume(getAngle(t))
     &- getZoneVolume_II(t) - getZoneVolume_III(t)
      RETURN
      END


      REAL FUNCTION getContactArea_Chart(r,t)
      IMPLICIT NONE
      REAL ChamberRadius, MaxHeigthChamber, hCh
      REAL*8 calculateSphereArea
      REAL ContactArea_Chart
      REAL r, t, getAngle
      REAL getCurrentChamberHeigth

c     Calculation of the heigth of the combustion chamber from its
c     parameters (r if it is less than the heigth of the chamber,
c     this heigth otherwise)
      CALL GeoCylinder(ChamberRadius, MaxHeigthChamber)
      hCh = getCurrentChamberHeigth(getAngle(t))

c     The radius of the sphere standing for the interface between zones
c     I and II is getRadius_I(t)
      ContactArea_Chart = SNGL(calculateSphereArea(
     &DBLE(r), DBLE(hCh), DBLE(ChamberRadius)))

      getContactArea_Chart = ContactArea_Chart

      RETURN
      END


      REAL FUNCTION getContactArea_I_II(t)
      IMPLICIT NONE
      REAL ChamberRadius, MaxHeigthChamber, hCh
      REAL*8 calculateSphereArea
      REAL getRadius_II, getCurrentChamberHeigth, getAngle
      REAL t

c     Calculation of the heigth of the combustion chamber from its
c     parameters (r if it is less than the heigth of the chamber,
c     this heigth otherwise)
      CALL GeoCylinder(ChamberRadius, MaxHeigthChamber)
      hCh = getCurrentChamberHeigth(getAngle(t))

c     The radius of the sphere standing for the interface between zones
c     I and II is getRadius_II(t)
      getContactArea_I_II = SNGL(calculateSphereArea(
```

GeoZones.f

```
     &DBLE(getRadius_II(t)), DBLE(hCh), DBLE(ChamberRadius)))

      RETURN
      END


      REAL FUNCTION getContactArea_II_III(t)
      IMPLICIT NONE
      REAL ChamberRadius, MaxHeigthChamber, hCh, t
      REAL getAngle
      REAL*8 calculateSphereArea
      REAL getRadius_III, getCurrentChamberHeigth

c     Calculation of the heigth of the combustion chamber from its
c     parameters (r if it is less than the heigth of the chamber,
c     this heigth otherwise)
      CALL GeoCylinder(ChamberRadius, MaxHeigthChamber)
      hCh = getCurrentChamberHeigth(getAngle(t))

c     The radius of the sphere standing for the interface between zones
c     II and III is getRadius_III(t)
      getContactArea_II_III = SNGL(calculateSphereArea(
     &DBLE(getRadius_III(t)), DBLE(hCh), DBLE(ChamberRadius)))

      RETURN
      END


c=======================================================================
c  FUNCTIONS getRadius_II(t), getRadius_III(t)
c=======================================================================
c  Author : Mathieu ALLORY - CVUT U2201 - April 2006, July 2006
c
c  Abstract : Returns the theorical radius of a zone. Please consider :
c     * width of zone II is (r_II - r_III)
c     * Volume IS NOT Pi*r?, intersections with the cylinder and piston
c       have to be checked (it is done later during integration)
c
c  Inputs :
c     REAL t : time when you want the radius of the zone
c  Outputs :
c     REAL : The radius of the zone
c
c=======================================================================
c     7 Start Column
      REAL FUNCTION getRadius_III(t)
      IMPLICIT NONE
      REAL t, angle, radius_expension_coef, dummy, angle_SOI
c     Fuctions
      REAL getRadius_III_Initial
      REAL getAngle
      angle = getAngle(t)
c     A very simple law (linear) defines the zones expension
c     First, retreive the angle where the integration starts (spark
c     ignition)
      CALL getIntegrationParameter_angle_SOI(angle_SOI)
```

GeoZones.f

```
      CALL GeoZones(dummy, radius_expension_coef)
c     Return the radius value (or zero if the ignition hasn't started)
      IF (angle .LE. angle_SOI) THEN
        radius_expension_coef = 0.
      ENDIF
c     Of course we take the initial value in consideration
      getRadius_III = getRadius_III_Initial()
     &+ radius_expension_coef * t
      RETURN
      END


      REAL FUNCTION getRadius_II(t)
      IMPLICIT NONE
      REAL t
      REAL getRadius_III, getAngle
      REAL angle_SOI
      REAL flameFrontWidth, dummy
      CALL GeoZones(flameFrontWidth, dummy)
c     Zone II exists only if ignition has happened and if it hasn't
c     reached the wall
c     The second case will be tested later during integration
      CALL getIntegrationParameter_angle_SOI(angle_SOI)
      IF (getAngle(t) .GE. angle_SOI) THEN
c     In this implementation of the geometry, the width of the flame
c     front is assumed to be a constant
            getRadius_II = getRadius_III(t) + flameFrontWidth
      ELSE
            getRadius_II = 0.
      ENDIF
      RETURN
      END


      REAL FUNCTION computeSphereVolume(r, Angle)
c=======================================================================
c  FUNCTION getZoneVolume
c=======================================================================
c  Author : Mathieu ALLORY - CVUT U2201 - April 2006
c
c  Abstract : Calculate the volume of half a sphere, according to the
c             position of the piston and the size of the chamber.
c             Integration uses a 61-POINT GAUSS-KRONROD RULES algo.
c             SLATEC library (QK61) is required.
c
c  Inputs :
c     REAL Angle : Crank angle (in degrees)
c     REAL r : the sphere theorical radius, without calculation of the
c              intersection with the piston or chamber
c  Outputs :
c     REAL : An approximation of the volume of the sphere
c  Parameters :
c     For all those parameters that NEED to be correctly set up, please
c     see SLATEC library documentation, file qk61.f.html
c     REAL Precision : the absolute precision to check
c
```

GeoZones.f

```
C=======================================================================
c     7 Start Column
      IMPLICIT NONE

c     For integration purpose (see Slatec doc)
      REAL Result, AbsErr
      REAL ResAbs, ResAsc
      REAL Precision
      PARAMETER (Precision = 0.000001)

      REAL Angle, r, rCOMMON, Hp, IntLimit
      REAL ChamberRadius, MaxHeigthChamber
      REAL toIntegrate
      REAL getCurrentChamberHeigth

c     The function returning the radius, that we will integrate
      EXTERNAL toIntegrate

c     We have to use a COMMON to give r and rCh to
c     calculateSphereZRadius, which must have only one parameter
c     (cf QAG specifications in Slatec documentation)
      COMMON rCOMMON, ChamberRadius
      rCOMMON = r

c     Retreives the usefull informations from the other subroutines
      Hp = getCurrentChamberHeigth(Angle)

c     Calculation of the heigth of the combustion chamber from its
c     parameters (r if it is less than the heigth of the chamber,
c     this heigth otherwise)
      CALL GeoCylinder(ChamberRadius, MaxHeigthChamber)
      IntLimit = MIN(r, Hp)

c     We are using Slatec library to compute the integral
      CALL QK61 (toIntegrate, REAL(0), IntLimit, Result,
     & AbsErr, ResAbs, ResAsc)

c     Dealing with precision
      IF (AbsErr .GT. Precision) THEN
        WRITE (6,*) 'WARNING : Accuracy lower than requested when comput
     &ing zone volume.'
        WRITE (6,*) 'Trace : Angle=', Angle, ' r=',r, ' Error=',AbsErr,
     & ' (See slatec doc)'
      ENDIF

      computeSphereVolume = Result
      RETURN
      END


      REAL*8 FUNCTION calculateSphereArea(r, hCh, rCh)
C=======================================================================
c  FUNCTION calculateSphereArea
C=======================================================================
c  Author : Mathieu ALLORY - CVUT U2201 - April 2006
c
```

GeoZones.f

```
c  Abstract : Calculates the area of a zone, depending on the its size
c             and the size of the cylinder. This is an internal
c             function used by getContactArea.
c             It assumes that the spark is in the middle of the cylinder
c             Integrals have been symbolicaly solved.
c
c  Inputs :
c     REAL DOUBLE r : The radius of the sphere (meters)
c     REAL DOUBLE hCh : Current heigth of the chamber (meters)
c     REAL DOUBLE rCh : The radius of the combustion chamber (meters)
c
c  Outputs :
c     REAL DOUBLE : The area of the sphere
c  Parameters :
c     REAL DOUBLE Pi : Pi constant
c
c=======================================================================
c     7 Start Column

      IMPLICIT NONE
      REAL*8 r, hCh, rCh
      REAL*8 d, h, buff1, buff2
      REAL*8 out
      REAL*8 Pi
      PARAMETER (Pi = 3.14159265358979323846264338327950288419716)

      out = 0.0

c     Three cases can happen :
c     * hCh >= r && rCh >= r : the area is a simple half sphere
      IF ((hCH .GE. r) .AND. (rCH .GE. r)) THEN
c     CORRIGE
        out = 2. * Pi * (r**2)
      ENDIF

c     * hCH < r && rCH >= r : the area is the area of half a sphere minus
c     the intersection with the piston
      IF ((hCH .LT. r) .AND. (rCH .GE. r)) THEN
c        out = (1./2.)*(r**2)*Pi + 3.*hCh*DSQRT(r**2-hCh**2)+(r**2)
c     &  *DASIN(DSQRT(r**2-hCh**2)/r)
         d = 2. * DSQRT(r**2 - hCh**2)
         h = r - rCh
         out = 2. * Pi * r**2 - Pi * h**2 * (3.*d**2 + 4.*h**2)/(24.*h)
      ENDIF

c     * hCh >= r && rCh < r : the area is the area of half a sphere
c     minus the interstion with the walls of the chamber
      IF ((hCH .GE. r) .AND. (rCH .LT. r)) THEN
c         out = rCh*DSQRT(r**2-rCh**2) + (r**2)*DASIN(rCh/r)
c     CORRIGE
          d = 2. * rCh
          h = r - DSQRT(r**2 - rCh**2)
          out = Pi * h**2 * (3.*d**2 + 4.*h**2)/(24.*h)
      ENDIF

c     * hCh < r && rCh < r : the area is the area of half a sphere
```

GeoZones.f

```
c     minus the interstion with the walls of the chamber and the piston
      IF ((hCH .LT. r) .AND. (rCH .LT. r)) THEN
c        The case (obviously not very relevant) where
c        the zone filfull all the chamber
         IF (r .GE. DSQRT(rCh**2 + hCh**2)) THEN
            out = 0.
         ELSE
c             out = hCh*DSQRT(r**2 - hCh**2) + r**2 * DASIN(rCh/r) +
c     &       rCh * DSQRT(r**2 - rCh**2) - r**2 *
c     &       DASIN(SQRT(r**2-hCh**2)/r)
            d = 2. * rCh
            h = r - DSQRT(r**2 - rCh**2)
            buff1 = Pi * h**2 * (3.*d**2 + 4.*h**2)/(24.*h)
            d = 2. * DSQRT(r**2 - hCh**2)
            h = r - rCh
            buff2 = Pi * h**2 * (3.*d**2 + 4.*h**2)/(24.*h)
            out = buff1 - buff2
         ENDIF
      ENDIF

      calculateSphereArea = out
      RETURN
      END


      REAL FUNCTION toIntegrate(z)
c=======================================================================
c  FUNCTION calculateSphereZRadius
c=======================================================================
c  Author : Mathieu ALLORY - CVUT U2201 - April 2006
c
c  Abstract : just returns Pi*r(z)^2 for integration
c
c  Inputs :
c     REAL z   : The altitude where to computer the radius
c  Inputs using COMMON :
c     REAL r : The radius of the sphere (meters)
c     REAL rCh : The radius of the combustion chamber (meters)
c  Parameters :
c     REAL Pi : Pi constant
c  Outputs :
c     REAL : The function to integrate
c
c=======================================================================
c     7 Start Column
      IMPLICIT NONE

      REAL Pi, z
      REAL calculateSphereZRadius
      PARAMETER (Pi = 3.14159265)

      toIntegrate = Pi * calculateSphereZRadius(z)**2
      RETURN
      END
```

GeoZones.f

```
      REAL FUNCTION calculateSphereZRadius(z)
c=======================================================================
c  FUNCTION calculateSphereZRadius
c=======================================================================
c  Author : Mathieu ALLORY - CVUT U2201 - April 2006
c
c  Abstract : Calculates the projection of the radius of a zone,
c             depending on the altitude you ask for.
c             function used by getZoneVolume.
c             It assumes that the spark is in the middle of the cylinder
c
c  Inputs :
c     REAL z   : The altitude where to computer the radius
c  Inputs using COMMON :
c     REAL r : The radius of the sphere (meters)
c     REAL rCh : The radius of the combustion chamber (meters)
c
c  Outputs :
c     REAL : The radius of the sphere
c
c=======================================================================
c     7 Start Column

      IMPLICIT NONE
      REAL r, rCh, z
      REAL out

c     Retreiving r and rCh from common memory
      COMMON r, rCh

      out = 0.0

      IF (rCh .GE. r) THEN
        out = SQRT(r**2 - z**2)
      ELSE
        IF (z .GT. SQRT(r**2 - rCh**2)) THEN
          out = SQRT(r**2 - z**2)
        ELSE
          out = rCh
        ENDIF
      ENDIF

      calculateSphereZRadius = out
      RETURN
      END

c=======================================================================
c  Debugging functions
c=======================================================================
      REAL FUNCTION linear(z)
      IMPLICIT NONE
      REAL z
      linear = z
      RETURN
      END
```

InitialGeom.f

```
c=======================================================================
c  Author : Mathieu ALLORY - CVUT U2201 - July 2006
c
c  Abstract : This subroutines return initial values of zone radius.
c             It has to be done this way because the code is unable to
c             calculate with null volume values (causes division by 0)
c             Of course it is purely theorical but it seems to be the
c             only way...
c
c=======================================================================
c     7 Start Column

c     Returns the initial value of zone III radius (burnt gas) in meters
      FUNCTION getRadius_III_Initial ()
      IMPLICIT NONE
      REAL getRadius_III_Initial
      getRadius_III_Initial = 0.3 * 10.**(-3.)
      RETURN
      END

c     There is no need to define any initial radius for the other zones
c     because they depend on zone III radius.
```

Test_geom.f

```
      PROGRAM Test_geom

      REAL a,b,sp
      REAL t, angle, radius, getContactArea_Chart
      REAL ContactArea
c      REAL getAngle
c      REAL getRadius_I, getRadius_II
      REAL getRadius_III
      REAL getZoneVolume_I, getZoneVolume_II, getZoneVolume_III
      COMMON a,b,sp

      OPEN(unit=10, file='dummygeom.csv', form='FORMATTED')
c     Write titles
      WRITE (10,*) 'Speed;', 'Angle;', 'Time;', 'Chamber Volume;',
     & 'Zone I Volume;', 'Zone II Volume;', 'Zone III Volume;',
     & 'Contact surface between I & II;',
     & 'Contact surface between II & III;', 'ENDLINE'

      WRITE (6,*) '************************************************'
      WRITE (6,*) '* This is a debugging program for the Dummy    *'
      WRITE (6,*) '* Geometry library. It will store values for   *'
      WRITE (6,*) '* speed from 500rpm to 4000rpm to the file     *'
      WRITE (6,*) '* dummygeom.csv                                *'
      WRITE (6,*) '* Please wait a moment ...                     *'
      WRITE (6,*) '************************************************'
      WRITE (6,*) ''


c     Writing a file to allow drawing of the chart of the page 7
      sp = 3000.
      OPEN(unit=11, file='ffsurf_chart.csv', form='FORMATTED')
      WRITE (11,*) '# Radius(mm) Angle(°) Flame front surface(mm2)'
      DO angle = 180., 360., 0.1
         DO radius = 0., 0.1, 0.001
c           WRITE (11,*) radius*10**3, angle, 10**6 *getContactArea_Chart
c     &(radius, getTime(angle))
         ENDDO
      ENDDO
      CLOSE(11)
      WRITE (6,*) 'ffsurf_chart.csv succesfully written.'


c     Writing a file to allow drawing of the chart of the page 7
      sp = 3000.
      OPEN(unit=11, file='ffsurf_chart2.csv', form='FORMATTED')
      WRITE (11,*) '# Radius(mm) Angle(°) Flame front surface(mm2)'
      DO angle = 365., 370.5, 0.0001
c          DO radius = 0., 0.05, 0.001
           radius = getRadius_III(getTime(angle))
           ContactArea = getContactArea_Chart(radius, getTime(angle))
c           WRITE (11,*) radius*10**3, angle, 10**6 *ContactArea
          WRITE (11,*) angle, 10**6 *ContactArea
c          ENDDO
      ENDDO
      CLOSE(11)
      WRITE (6,*) 'ffsurf_chart2.csv succesfully written.'
```

Test_geom.f

```
      WRITE (6,*) 'Computing dummygeom.csv ...'
c     Engine speed, passed through a common to make it vary (for test
c     purposes, in the code, it is constant)
      DO sp = 500., 4000., 100.

c     The loop is driven by angle...
        DO angle = 300., 400., 0.1
c     ... so we have to retreive time
           t = getTime(angle)

c       Storing results in a file

            WRITE (10,*)
     &    sp,';', angle,';', t,';', getChamberVolume(angle),';'
     &    , getZoneVolume_I(t),';', getZoneVolume_II(t),';'
     &    ,getZoneVolume_III(t), ';', getContactArea_I_II(t),';'
     &    ,getContactArea_II_III(t), ';', 'ENDLINE'

        ENDDO
      ENDDO

      WRITE (6,*) 'Computation ended.'
      WRITE (6,*) ''
      WRITE (6,*) 'Fortran is not able to write long enough lines so'
      WRITE (6,*) 'you may have to type (Unix system) :'
      WRITE (6,*) 'tr -d \'\\n\' < dummygeom.csv > buffer'
      WRITE (6,*) 'sed -e \'s/ENDLINE/\\n/g\' buffer > dummygeom.csv'
      WRITE (6,*) 'to use it with another software.'
      CLOSE(10)

      STOP
      END



c     Those routines override those defines in the code (Datas.f)
c     They provide the informations needed for test purposes

c     The angle when the integration starts
      SUBROUTINE getIntegrationParameter_angle_SOI(angle_SOI)
      REAL angle_SOI
      angle_SOI = 340.
      RETURN
      END


c     Engine speed
      SUBROUTINE getParameter_engine_speed(speed)
      REAL speed
      REAL a,b,sp
      COMMON a,b,sp
      speed = sp
      RETURN
      END
```

makefile

```
CC=g77

AR= ar
ARFLAGS= rc
CFLAGS=-Wall -g
LDFLAGS=-Wall -g -Wl,-rpath=/usr/local/lib64/

SLATEC=../slatec/slatec_x86.a

OUT_LIB= geometry.a
SRC_LIB= GeoCylinder.f GeoPiston.f GeoZones.f CrankAngle.f InitialGeom.f
GeoDatas.f

SRC_PRG= Test_geom.f
EXEC=dummygeom

OBJ_LIB= $(SRC_LIB:.f=.o)
OBJ_PRG= $(SRC_PRG:.f=.o)


all:
        @( $(MAKE) library)
        @( $(MAKE) test)

library: $(OBJ_LIB)
        $(AR) $(ARFLAGS) $(OUT_LIB) $^ $(SLATEC)
        rm -f *.o
        @echo
"************************************************************************"
        @echo "This geometry is very simple, for test purposes."
        @echo "It modelizes a simple cylinder and piston with purely spherical
zones"
        @echo
"************************************************************************"

test: $(OBJ_PRG) $(OUT_LIB) $(SLATEC)
        $(CC) -o $(EXEC) $^ $(LDFLAGS)
        @echo "Test program ($(EXEC)) built finished."

%.o: %.f
        $(CC) -c $^ $(CFLAGS)

clean:
        rm -f *.o
```

The other models and files : directory /ZoneApproach/sources/

Datas.f

```
c=======================================================================
c  Author : Mathieu ALLORY - CVUT U2201
c  Date : 22 Mai 2006
c=======================================================================
c  Abstract : This file is supposed to contain datas from the problem
c=======================================================================


c     Integration and differentiation timestep
      SUBROUTINE getIntegrationParameter_Timestep(time_step)
      IMPLICIT NONE
      REAL time_step
      REAL angleSOI
      REAL getTime
      CALL getIntegrationParameter_angle_SOI(angleSOI)
      time_step = getTime(angleSOI + 0.1)
      RETURN
      END


c     Returns the temperature of th walls in Kelvin
      SUBROUTINE Walls_Temp_Params(angle, alpha_coef, temp_wall)
      IMPLICIT NONE
      REAL angle
      REAL alpha_coef, temp_wall
c     In this version we assume that the wall cooling is disabled
c      temp_wall = 500.
c      alpha_coef = 1.
      temp_wall = 0.
      alpha_coef = 0.
      RETURN
      END


c     Return the specific gaz constants for each component
c     (J/kg/K)
      SUBROUTINE Specific_gaz_constant(r_vect,dim)
      IMPLICIT NONE
      INTEGER dim
      REAL r_vect(*)
      REAL data_r(5)
c     Specific gaz constants (as the result of R/M) for :
c     O2 - N2Ar - CO2 - H2O - C3H8
      DATA data_r / 259.827, 295.434, 188.965, 461.915, 188.965 /
      CALL VectCopy (r_vect, data_r,dim)
      RETURN
      END


c     This subroutine calculate Cp as an approximation from values known
c     for some temperatures in J/Kg
      SUBROUTINE Enthalpy(i_vect, dim, T)
```

Datas.f

```
      IMPLICIT NONE
      INTEGER dim
      REAL i_vect(5), buffer(5), mbuffer(5), zbuffer(5), buffer2(5)
      REAL T
      INTEGER samples
      DATA samples / 15 /
      REAL known_values_enthalpy (15,6)
c     Values below are in KJ/KG/Celsius
c     Known values of Enthalpy. First, the temperatures (celsius)
      DATA known_values_enthalpy / -00073.1500, 00025.0000, 00026.8500,
     &00126.8500, 00226.8500, 00326.8500, 00426.8500, 00526.8500,
     &00626.8500, 00726.8500, 00826.8500, 00926.8500, 01026.8500,
     &01126.8500, 01226.8500,
c     Values for 02
     &-00089.3125, 00000.0000, 00001.6875, 00094.6250, 00190.2813,
     &00288.9688, 00390.6250, 00494.9063, 00601.3750, 00709.6250,
     &00819.1563, 00929.8125, 01041.5000, 01154.1875, 01267.8438,
c     Values for N2+Ar
     &-00153.3282, 00000.0000, 00002.8532, 00159.0039, 00316.0214,
     &00474.6095, 00635.2324, 00798.2470, 00963.7247, 01131.4513,
     &01301.2127, 01472.6518, 01645.6973, 01820.1350, 01995.8221,
c     Values for CO2
     &-09021.2273, -08943.4545, -08941.8864, -08852.4773, -08754.7273,
     &-08650.1136, -08539.8409, -08424.9773, -08306.2955, -08184.3636,
     &-08059.6364, -07932.9773, -07804.4318, -07674.1818, -07544.7045,
c     Values for H2O
     &-07427.1818, -07328.0000, -07326.0909, -07223.3636, -07118.1515,
     &-07009.8788, -06898.1515, -06782.7576, -06663.6061, -06540.6364,
     &-06413.6970, -06282.9394, -06148.6061, -06010.8788, -05870.0303,
c     Values for Fuel (C3H8)
     &-02513.2778, -02373.9007, -02370.7939, -02178.4190, -01941.8227,
     &-01666.2433, -01356.4434, -01016.7770, -00651.1445, -00263.0612,
     &00144.4569, 00568.8245, 01007.8646, 01459.8537, 01923.4991 /


      IF ((T-273.15) .GT. known_values_enthalpy (15,1)) THEN
c     Temperature is out of range
        zbuffer(1) = known_values_enthalpy (1,2)
        zbuffer(2) = known_values_enthalpy (1,3)
        zbuffer(3) = known_values_enthalpy (1,4)
        zbuffer(4) = known_values_enthalpy (1,5)
        zbuffer(5) = known_values_enthalpy (1,6)
        mbuffer(1) = known_values_enthalpy (15,2)
        mbuffer(2) = known_values_enthalpy (15,3)
        mbuffer(3) = known_values_enthalpy (15,4)
        mbuffer(4) = known_values_enthalpy (15,5)
        mbuffer(5) = known_values_enthalpy (15,6)
        CALL VectMulNum(buffer, zbuffer,-1., dim)
        CALL VectAdd(buffer2, mbuffer, buffer,dim)
        CALL VectMulNum(mbuffer, buffer2, (T- 273.15 -
     &known_values_enthalpy (1,1))/(known_values_enthalpy (15,1) -
     &known_values_enthalpy (1,1)), dim)
        CALL VectAdd(buffer, mbuffer, zbuffer,dim)
      ELSE
c     Temperature is in the range, interpolation between the two nearest
c     values
```

Datas.f

```
        CALL Interpolate (buffer, known_values_enthalpy, samples,
     &T - 273.15)
       ENDIF
c      Now we convert to J/KG/K (-273 just behind is for celsius to
c      kelvin translation)
       CALL VectMulNum(i_vect, buffer, 1000., dim)

       RETURN
       END


c      This subroutine calculate Cp as an approximation from values known
c      for some temperatures, in J/KG/K
       SUBROUTINE Cp(Cp_vect, dim, T)
       IMPLICIT NONE
       INTEGER dim
       REAL Cp_vect(5), buffer(5)
       REAL T
       INTEGER samples
       DATA samples / 13 /
       REAL known_values_cp (13,6)
c      Values below are in KJ/KG/Celsius
c      Known values of Cp. First, the temperatures (Celsius)
       DATA known_values_cp / 0., 100., 200., 300., 400., 500., 600.,
     &700., 800., 900., 1000., 1100., 1200.,
c      Values for 02
     &0.9148, 0.9337, 0.9630, 0.9948, 1.0237, 1.0484, 1.0689, 1.0856,
     &1.0999, 1.1120, 1.1229, 1.1317, 1.1401,
c      Values for N2+Ar
     &1.0304, 1.0333, 1.0427, 1.0600, 1.0819, 1.1053, 1.1287, 1.1506,
     &1.1703, 1.1876, 1.2033, 1.2168, 1.2288,
c      Values for CO2
     &0.8148, 0.9136, 0.9927, 1.0567, 1.1103, 1.1547, 1.1920, 1.2230,
     &1.2493, 1.2715, 1.2900, 1.3059, 1.3197,
c      Values for H2O
     &1.8594, 1.8903, 1.9406, 2.0005, 2.0645, 2.1319, 2.2014, 2.2730,
     &2.3450, 2.4154, 2.4824, 2.5456, 2.6042,
c      Values for Fuel (C3H8)
     &1.5495, 2.0168, 2.4581, 2.8345, 3.1610, 3.4487, 3.6974, 3.9159,
     &4.0926, 4.2500, 4.3945, 4.5263, 4.6448 /

       CALL Interpolate (buffer, known_values_cp, samples,
     &T - 273.15)
c      Now we convert to J/KG/K (-273 just behind is for celsius to
c      kelvin translation)
       CALL VectMulNum(Cp_vect, buffer, 1000., dim)
       RETURN
       END

c      The angle when the integration starts
       SUBROUTINE getIntegrationParameter_angle_SOI(angle_SOI)
       IMPLICIT NONE
       REAL angle_SOI
       angle_SOI = 340.
       RETURN
       END
```

Datas.f

```
c     Engine speed
      SUBROUTINE getParameter_engine_speed(speed)
      IMPLICIT NONE
      REAL speed
      speed = 3000.
      RETURN
      END

c     We assume that there is only burning reaction, so correction
c     vector is null
      SUBROUTINE correction_vector(Vrr_corr, t)
      IMPLICIT NONE
      REAL t
      REAL Vrr_corr(5)
      Vrr_corr(1) = 0.
      Vrr_corr(2) = 0.
      Vrr_corr(3) = 0.
      Vrr_corr(4) = 0.
      Vrr_corr(5) = 0.
      RETURN
      END

      SUBROUTINE reaction_matrix(rm, row, col)
      IMPLICIT NONE
      REAL rm(5,1)
      REAL x,y
      INTEGER row, col
c     Matrix dimensions
      row = 5
      col = 1
      CALL getFuelCompo(x, y)
c     Only one reaction : burning
c     O2
      rm(1,1) = -(x+y/4.)
c     N2 + Ar
      rm(2,1) = 0.
c     CO2
      rm(3,1) = x
c     H2O
      rm(4,1) = (x/2.)
c     fuel
      rm(5,1) = -1.
      RETURN
      END

c     Returns fuel composition (CmHn)
      SUBROUTINE getFuelCompo(m, n)
      IMPLICIT NONE
      REAL m, n
c     CmHn (Here C3H8)
      m = 3.
      n = 8.
      RETURN
      END
```

Equations.f

```
c=======================================================================

c  Library Equations
c=======================================================================
c  Author : Mathieu ALLORY - CVUT U2201
c  Date : 22 Mai 2006 - July 2006
c=======================================================================
c  Abstract : Defines the equations used by the core program.
c             They are taken from Pr. J.Macek "Zone Approach Used for
c             Determination of Premixed Turbulent Flame Parameters",
c             assuming we are using a 3 zones model
c  Summary :
c
c=======================================================================


c=======================================================================
c  Mass transfer equations
c=======================================================================
c     out = d{s mi}/dt
      SUBROUTINE MassTransfert(out, mdot_ip_i, mdot_i_if, rdot_i,dim)
        IMPLICIT NONE
        REAL out (*)
        REAL mdot_ip_i(*)
        REAL mdot_i_if(*)
        REAL rdot_i(*)
        INTEGER dim
        INTEGER i
        i = 1
        DO WHILE (i .LE. dim)
          out(i) = mdot_ip_i(i) - mdot_i_if(i) + rdot_i(i)
          i = i + 1
        ENDDO
      RETURN
      END


c=======================================================================
c  This subroutine formats the equations in a comprehensive way for the
c  solver (cf. Maths.f - DiffSolver)
c  This is the core equations routine
c=======================================================================
c     U(1)....U(Neq) stand for the time dependant functions
c     UPRIME(1)....UPRIME(Neq) are their derivatives

c     Here, U(n) stands for :
c     U(1) : (O2)mI(t)
c     U(2) : (N2)mI(t)
c     U(3) : (CO2)mI(t)
c     U(4) : (H20)mI(t)
c     U(5) : (fuel)mI(t)
c     Zone II, n=6..10
c     U(6) : (O2)mII(t)
c     ...
c     Zone III, n=11..15
c     ...
c     UPRIME(n) are assigned in the same way
```

Equations.f

```
      SUBROUTINE Core_Equations_System(T, U, UPRIME, RPAR, IPAR)
      IMPLICIT NONE
      REAL U(15), UPRIME(15)
c     RPAR & IPAR are variables used to pass arguments through the
c     solver. Here they are dummy variables, cause we will use
c     subroutines calls instead.
c     Update : We will use RPAR to send back informations to be stored
c     (volumes, temps...) to the main calling routine through the
c     solver
      INTEGER IPAR(1)
      REAL RPAR(12)

c     It's T Time...
      REAL T
      REAL buffer(5)
      INTEGER I, DummyI

c     The variables we will be used to store mass transfert values,
c     reaction rates and all the intermediate variables
c     (right member of the equation)
c     We assume there are 5 components
      REAL mdot_null(5)
      REAL mdot_I_II(5)
      REAL mdot_II_III(5)
      REAL rdot_I(5)
      REAL rdot_II(5)
      REAL rdot_III(5)
c     Contact areas
      REAL A_f_I_II
      REAL A_f_II_III
c     Zone volumes
      REAL V_I
      REAL V_II
      REAL V_III
c     Speed of the flame front
      REAL w_f_front
      REAL w_f_back
c     Mass variation
      REAL m_I(5)
      REAL m_II(5)
      REAL m_III(5)
c     Temperatures
      REAL Temp_I, Temp_II, Temp_III
c     Functions
      REAL getZoneVolume_I, getZoneVolume_II, getZoneVolume_III
      REAL getContactArea_I_II, getContactArea_II_III
      REAL Temp
c     Functions used for diagnostic
      REAL getAngle, pressure

c     Filling the null vectors with null values (instead of reserving
c     useless memory for mdot_Wall_I, mdot_III_wall)
      DO I = 1, 5, 1
        mdot_null(I) = 0.
      ENDDO
```

Equations.f

```
c     Recopying from U (last step results) to m
      DO I = 1, 5, 1
        m_I(I) = U(I)
        m_II(I) = U(I + 5)
        m_III(I) = U(I + 10)
      ENDDO

c     Retreiving the geometry
c     Those routines are call from an external library
      A_f_I_II = getContactArea_I_II(T)
      A_f_II_III = getContactArea_II_III(T)
      V_I = getZoneVolume_I(T)
      V_II = getZoneVolume_II(T)
      V_III = getZoneVolume_III(T)

c     Evaluation of current temperatures
      Temp_I = Temp(m_I, m_II, m_II, T, 1, 5)
      Temp_II = Temp(m_I, m_II, m_II, T, 2, 5)
      Temp_III = Temp(m_I, m_II, m_II, T, 3, 5)

c     Evaluation of flame front speed
      CALL w_flame_front (w_f_front, m_I, Temp_I,5, T)
      CALL w_flame_back (w_f_back, m_II, m_III, Temp_II, Temp_III,5, T)

c     Evaluation of mdot vectors
      CALL Eval_mdot_I_II(mdot_I_II, A_f_I_II, V_I, w_f_front, m_I,5)
      CALL Eval_mdot_II_III(mdot_II_III, A_f_II_III, V_II, w_f_back,
     & m_II, 5)

c     Computing equations for zone I.
      CALL rr_I(rdot_I, DummyI, T)
      CALL MassTransfert(buffer, mdot_null, mdot_I_II, rdot_I,5)
c     Storing the output values
      DO I = 1, 5, 1
        UPRIME(I) = buffer(I)
      ENDDO

c     Computing equations for zone II.
      CALL rr_II(rdot_II, DummyI, T, m_I, m_II, m_III)
      CALL MassTransfert(buffer, mdot_I_II, mdot_II_III, rdot_II,5)
c     Storing the output values
      DO I = 1, 5, 1
        UPRIME(I + 5) = buffer(I)
      ENDDO

c     Computing equations for zone III.
      CALL rr_III(rdot_III, DummyI, T)
      CALL MassTransfert(buffer, mdot_II_III, mdot_null, rdot_III,5)
c     Storing the output values
      DO I = 1, 5, 1
        UPRIME(I + 10) = buffer(I)
      ENDDO

c     Sending back informations about the current iteration
      RPAR(1) = T
```

Equations.f

```
      RPAR(2) = getAngle(T)
      RPAR(3) = V_I
      RPAR(4) = V_II
      RPAR(5) = V_III
      RPAR(6) = pressure(getAngle(T))
      RPAR(7) = Temp_I
      RPAR(8) = Temp_II
      RPAR(9) = Temp_III
      RPAR(10) = w_f_front
      RPAR(11) = w_f_back
      CALL rdot_II_fuel (RPAR(12), m_I, m_II, m_III, T, 5)

      RETURN
      END


c=======================================================================
c  Equations for evaluation of mass exchange between zones
c=======================================================================
      SUBROUTINE Eval_mdot_I_II(mdot_I_II, A_f_I_II, V_I, w_f_front, m_I
     &, dim)
      IMPLICIT NONE
      REAL mdot_I_II(*)
      REAL m_I(*)
      REAL A_f_I_II
      REAL V_I
      REAL w_f_front
      INTEGER dim
      REAL Numeric
      Numeric = A_f_I_II * w_f_front / V_I
      CALL VectMulNum (mdot_I_II, m_I, Numeric,dim)
      RETURN
      END


      SUBROUTINE Eval_mdot_II_III(mdot_II_III, A_f_II_III, V_II,
     & w_f_back, m_II, dim)
      IMPLICIT NONE
c     We assume that the last component of vector m DOES concern fuel
c     (Usually it should be the fifth one) and the first is about O2
      REAL mdot_II_III(*)
      REAL m_II(*)
      INTEGER dim
      REAL m_temp(dim)
      REAL A_f_II_III
      REAL V_II
      REAL w_f_back
      REAL Numeric
      Numeric = A_f_II_III * w_f_back / V_II
c     As parameters are passed by reference, we have to use a temporary
c     copy, not to alterate m_II
      CALL VectCopy(m_temp, m_II, dim)
      m_temp(dim) = 0.
      m_temp(1) = 0.
      CALL VectMulNum (mdot_II_III, m_temp, Numeric,dim)
      RETURN
```

Equations.f

```
      END


c=======================================================================
c  Speed of the flame front equations  (Still something TODO)
c=======================================================================
      SUBROUTINE w_flame_front (out, m_I, TempI,dim, time)
      IMPLICIT NONE
c     Output
      REAL out
c     Inputs
      INTEGER dim
      REAL m_I(*)
      REAL time
c     Internals
      REAL first_member
      REAL kappa
      REAL A_f_I_II
      REAL V_I
      REAL r_vect(5)
      REAL timestep
      REAL D_V_I
      REAL D_pressure
      REAL Qdot
      REAL CpMi
      REAL TempI
      REAL p
c     Functions
      REAL getContactArea_I_II, getZoneVolume_I, KappaBlock
      REAL VectMul, Pressure_from_time, Cp_Mi, Qdot_I_CH
      EXTERNAL getZoneVolume_I
      EXTERNAL Pressure_from_time

c     Pressure
      p = Pressure_from_time(time)

c     Calculates kappa/(kappa-1)
      kappa = KappaBlock(m_I, dim, TempI)

c     Calculating geometry
      A_f_I_II = getContactArea_I_II(time)
      V_I = getZoneVolume_I(time)

c     Computation the first member of the equation
      CALL Specific_gaz_constant(r_vect,dim)
      first_member = V_I / (kappa * TempI * A_f_I_II *
     &VectMul(r_vect,m_I,dim))

c     Computes volume and pressure derivatives
      CALL getIntegrationParameter_TimeStep(timestep)
      CALL Diff(D_V_I, getZoneVolume_I, time, timestep)
      CALL Diff(D_pressure, Pressure_from_time, time, timestep)

c     Calculates heat transfer with walls
      Qdot = Qdot_I_CH (m_I, TempI, time,dim)
```

Equations.f

```
c     Calculates {Cp}{M_I}
      CpMi = Cp_Mi (m_I, dim, TempI)

c     ======================================================
c     TODO : Include corrective term (now considered null)
c     ======================================================

c     Main Formula
      out = first_member * (-Qdot - (CpMi * TempI / p - V_I) *D_pressure
     &- p * kappa * D_V_I)

c     Protection against negative values
      IF (out .LT. 0.) THEN
       out = 0.
      ENDIF

      RETURN
      END


      SUBROUTINE w_flame_back (out, m_II, m_III, TempII, TempIII,
     &dim, time)
      IMPLICIT NONE
c     Output
      REAL out
c     Inputs
      INTEGER dim
      REAL m_II(*)
      REAL m_III(*)
      REAL time
c     Internals
      REAL p
      REAL kappa
      REAL A_f_II_III
      REAL V_II, V_III
      REAL r_vect(5)
      REAL timestep
      REAL D_V_III
      REAL D_pressure
      REAL Qdot
      REAL CpMi
      REAL Enthalpy_II(5)
      REAL Enthalpy_III(5)
      REAL TempII, TempIII
c     Internals (buffers)
      REAL first_member
      REAL vBuff1(5)
      REAL vBuff2(5)
      REAL vBuff3(5)
c     Functions
      REAL getContactArea_II_III
      REAL getZoneVolume_II, getZoneVolume_III, KappaBlock
      REAL VectMul, Pressure_from_time, Qdot_III_CH
      EXTERNAL getZoneVolume_III
      EXTERNAL Pressure_from_time
```

Equations.f

```
c     Pressure
      p = Pressure_from_time(time)

c     Calculating geometry
      A_f_II_III = getContactArea_II_III(time)
      V_II = getZoneVolume_II(time)
      V_III = getZoneVolume_III(time)

c     Calculates kappa/(kappa-1) for zone III
      kappa = KappaBlock(m_III, dim, TempIII)

c     Evaluation of enthalpy for zone 2 & 3
      CALL Enthalpy(Enthalpy_II, dim, TempII)
      CALL Enthalpy(Enthalpy_III, dim, TempIII)

c     Computation the first member of the equation
      CALL Specific_gaz_constant(r_vect,dim)
      CALL VectMulNum(vBuff1, r_vect, TempIII* kappa, dim)
      CALL VectAdd(vBuff2, vBuff1, Enthalpy_II,dim)
      CALL VectMulNum(vBuff3, vBuff2, -1., dim)
      CALL VectAdd(vBuff1, Enthalpy_III, vBuff3,dim)
c     vBuff1 now contains what's between brackets
c     We are to create mII,fuel vector from mII in vBuff2 assuming that
c     the fuel component is in last position (dim)
      CALL VectCopy(vBuff2, m_II, dim)
      vBuff2(dim) = 0.
      vBuff2(1) = 0.
      first_member = V_II / (VectMul(vBuff1, vBuff2, dim)*A_f_II_III)

c     Computes volume and pressure derivatives
      CALL getIntegrationParameter_TimeStep(timestep)
      CALL Diff(D_V_III, getZoneVolume_III, time, timestep)
      CALL Diff(D_pressure, Pressure_from_time, time, timestep)

c     Calculates heat transfer with walls
      Qdot = Qdot_III_CH (m_III, TempIII, time,dim)

c     Calculates {Cp}{M_III}
      CpMi = VectMul(r_vect,m_III,dim)

c     ======================================================
c     TODO : Include corrective term (now considered null)
c     ======================================================

c     Main Formula
      out = (-Qdot - (CpMi*TempIII/p - V_III)*D_pressure -
     &p*kappa*D_V_III)*first_member

c     Protection against negative values
      IF (out .LT. 0.) THEN
       out = 0.
      ENDIF
c     Protection against extremely high values
c      IF (out .GT. 50.) THEN
c       out = 50.
c      ENDIF
```

Equations.f

```
      RETURN
      END


c=======================================================================
c Reaction rate evaluation for zone 2 (flame front)
c
c Note : large parts of this routine are similar to w_flamme_front and
c        w_flamme_back. Look at those routines for more informations.
c
c TODO : Include corrective terms
c=======================================================================
      SUBROUTINE rdot_II_fuel (out, m_I, m_II, m_III, time,dim)
      IMPLICIT NONE
c     Outputs
      REAL out
c     Inputs
      REAL m_I(*), m_II(*), m_III(*)
      REAL time
      INTEGER dim
c     Internals
      REAL vBuff1(5), vBuff2(5), vBuff3(5)
      REAL TempI, TempII, TempIII
      REAL Qdot
      REAL r_vect(5)
      REAL kappa_I, kappa_II
      REAL A_f_II_III, A_f_I_II
      REAL V_I, V_II, D_V_II
      REAL w_f_front, w_f_back
      REAL p, D_pressure
      REAL CpMII
      REAL Cb_vect(5)
      REAL Enthalpy_I(5)
      REAL Enthalpy_II(5)
      REAL rm(5,1)
      INTEGER rm_col, rm_row
      REAL timestep
      REAL first_member, sec_mem, sec_mem_front, sec_mem_back
      REAL fuel_limit, o2_limit
      INTEGER I
c     Functions
      REAL getContactArea_I_II, getContactArea_II_III
      REAL getZoneVolume_II, getZoneVolume_I, KappaBlock, getAngle
      REAL VectMul, Pressure_from_time, Cp_Mi, Qdot_II_CH, Temp
      EXTERNAL getZoneVolume_II
      EXTERNAL Pressure_from_time

c     Evaluation of usefull quantities (see w_flame_xxx for more
c     detailled explanations)
      A_f_I_II = getContactArea_I_II(time)
      A_f_II_III = getContactArea_II_III(time)
      V_I = getZoneVolume_I(time)
      V_II = getZoneVolume_II(time)
      TempI = Temp(m_I, m_II, m_III, time, 1, dim)
      TempII = Temp(m_I, m_II, m_III, time, 2, dim)
```

Equations.f

```
      TempIII = Temp(m_I, m_II, m_III, time, 3, dim)
      kappa_I = KappaBlock(m_I, dim, TempI)
      kappa_II = KappaBlock(m_II, dim, TempII)
      Qdot = Qdot_II_CH (m_II, TempII, time, dim)
      CpMII = Cp_Mi (m_II, dim, TempII)
      p = Pressure_from_time(time)
      CALL Specific_gaz_constant(r_vect, dim)
      CALL Enthalpy(Enthalpy_II, dim, TempII)
      CALL Enthalpy(Enthalpy_I, dim, TempI)
      CALL w_flame_front (w_f_front, m_I, TempI, dim, time)
      CALL w_flame_back (w_f_back, m_II, m_III, TempII, TempIII,
     &dim, time)
      GOTO 20
      WRITE (6,*) '====================================================='
      WRITE (6,*) ' DEBUG INFORMATIONS (Routine rdot_II_fuel)'
      WRITE (6,*) '====================================================='
      WRITE (6,*) ' Time = ', time, ' (',getAngle(time),')'
      WRITE (6,*) ' m_I = ', (m_I(I), I=1, 5)
      WRITE (6,*) ' m_II = ', (m_II(I), I=1, 5)
      WRITE (6,*) ' m_III = ', (m_III(I), I=1, 5)
      WRITE (6,*) ' kappa_I = ', kappa_I
      WRITE (6,*) ' kappa_II = ', kappa_II
      WRITE (6,*) ' A_f_I_II = ', A_f_I_II
      WRITE (6,*) ' A_f_II_III = ', A_f_II_III
      WRITE (6,*) ' V_I = ', V_I
      WRITE (6,*) ' V_II = ', V_II
      WRITE (6,*) ' TempI = ', TempI
      WRITE (6,*) ' TempII = ', TempII
      WRITE (6,*) ' Qdot = ', Qdot
      WRITE (6,*) ' p = ', p
      WRITE (6,*) '====================================================='
   20 CONTINUE

c     Computing the fist member of the equation
      CALL VectMulNum(vBuff3, r_vect, -1. * TempII * kappa_II, dim)
      CALL VectAdd(vBuff1, Enthalpy_II, vBuff3, dim)
c     vBuff1 now contains what's in the brackets. We have to multiply it
c     with {sCb} that we retreive from the reaction matrix, assuming the
c     burning reaction is the first column of the matrix
      CALL reaction_matrix(rm, rm_row, rm_col)
      CALL getColFromMatrix(Cb_vect, rm, 1, rm_row)
      first_member = 1. / VectMul(vBuff1, Cb_vect, dim)

c     Computes volume and pressure derivatives
      CALL getIntegrationParameter_TimeStep(timestep)
      CALL Diff(D_V_II, getZoneVolume_II, time, timestep)
      CALL Diff(D_pressure, Pressure_from_time, time, timestep)

c     Computes the second member term containing w,f,front
      CALL VectMulNum(vBuff3, Enthalpy_I, -1., dim)
      CALL VectAdd(vBuff2, Enthalpy_II, vBuff3, dim)
      CALL VectMulNum(vBuff3, r_vect, -1. * kappa_I * TempI, dim)
      CALL VectAdd(vBuff1, vBuff2, vBuff3, dim)
      sec_mem_front = VectMul(vBuff1, m_I, dim) * A_f_I_II * w_f_front /
     &V_I
```

Equations.f

```
c     Computes the second member term containing w,f,back
c     (similar to first_member in w_flame_back)
      CALL VectCopy(vBuff2, m_II, dim)
      vBuff2(dim) = 0.
      vBuff2(1) = 0.
      sec_mem_back = kappa_II * TempII * A_f_II_III * w_f_back / V_II *
     &VectMul(r_vect, vBuff2, dim)

c     What remains...
      sec_mem = (CpMII * TempII / p - V_II) * D_pressure +
     &p*kappa_II* D_V_II

c     ======================================================
c     TODO : Include corrective term (now considered null)
c     ======================================================

      out = first_member * (-Qdot - sec_mem - sec_mem_front -
     &sec_mem_back)

c     Checking if the value is not to big
c     (not to burn more than we have)
      fuel_limit = (m_II(5)+m_I(5)*A_f_I_II/V_I*w_f_front*timestep)/
     &(timestep * ABS(Cb_vect(5)))
      o2_limit = (m_II(1)+m_I(1)*A_f_I_II/V_I*w_f_front*timestep)/
     &(timestep * ABS(Cb_vect(1)))

      IF (out .GT. MIN(fuel_limit, o2_limit)) THEN
        out = MIN(fuel_limit, o2_limit)
      ENDIF

c     Of course ROB can't be negative
      IF (out .LT. 0.) THEN
        out = 0.
      ENDIF

      RETURN
      END


c=======================================================================
c State Equations (to evaluate temperature)
c=======================================================================
c     Calculates the temperature for each zone using the perfect gaz
c     state equation
c     Zone could have been passed through a variable,but three functions
c     are easier too read.
      REAL FUNCTION Temp(m1_vect, m2_vect, m3_vect, time, zone,dim)
      IMPLICIT NONE
      REAL m1_vect(*)
      REAL m2_vect(*)
      REAL m3_vect(*)
      REAL time
      INTEGER dim, zone
      REAL r_vect(100)
      REAL Pressure, VolumeI, VolumeII, VolumeIII, Press
      REAL VectMul, getZoneVolume_I, getAngle, getZoneVolume_II
```

Equations.f

```
     &, getZoneVolume_III
c     Retreiving datas (Pressure, Gaz constants, Volume)
      Press = Pressure (getAngle(time))
      CALL Specific_gaz_constant(r_vect,dim)
      VolumeI = getZoneVolume_I(time)
      VolumeII = getZoneVolume_II(time)
      VolumeIII = getZoneVolume_III(time)
c     The state equation
      Temp = 0.
      IF (zone .EQ. 3) THEN
        Temp = Press * VolumeIII / VectMul(r_vect, m3_vect,dim)
      ENDIF
      IF (zone .EQ. 2) THEN
        Temp = Press * VolumeII / VectMul(r_vect, m2_vect,dim)
      ENDIF
      IF ((zone .EQ. 1).OR.(time .LE. 0.)) THEN
        Temp = Press * VolumeI / VectMul(r_vect, m1_vect,dim)
      ENDIF
      IF ((zone .NE. 1).AND.(zone .NE. 2).AND.(zone.NE.3)) THEN
        WRITE(6,*) '[ERROR] Wrong zone number in routine TEMP :',zone
      ENDIF
      RETURN
      END


c=======================================================================
c  Heat transfert to walls equations
c=======================================================================
      REAL FUNCTION Qdot_I_CH (M_vect, temp, time,dim)
      IMPLICIT NONE
      REAL time, M_vect(*)
      INTEGER dim
      REAL temp, tempW, alpha
      REAL getAngle, getSurface_I_Chamber

c     Retreives wall temperature
      CALL Walls_Temp_Params(getAngle(time), alpha, tempW)

c     Formulae
      Qdot_I_CH = alpha * (temp - tempW) * getSurface_I_Chamber(time)

      RETURN
      END

      REAL FUNCTION Qdot_II_CH (M_vect, temp, time,dim)
      IMPLICIT NONE
      REAL time, M_vect(*)
      INTEGER dim
      REAL temp, tempW, alpha
      REAL getAngle, getSurface_II_Chamber

c     Retreives wall temperature
      CALL Walls_Temp_Params(getAngle(time), alpha, tempW)

c     Formulae
      Qdot_II_CH = alpha * (temp - tempW) * getSurface_II_Chamber(time)
```

Equations.f

```
      RETURN
      END

      REAL FUNCTION Qdot_III_CH (M_vect, temp, time,dim)
      IMPLICIT NONE
      REAL time, M_vect(*)
      INTEGER dim
      REAL temp, tempW, alpha
      REAL getAngle, getSurface_III_Chamber

c     Retreives wall temperature
      CALL Walls_Temp_Params(getAngle(time), alpha, tempW)

c     Formulae
      Qdot_III_CH = alpha * (temp - tempW) *
     &getSurface_III_Chamber(time)

      RETURN
      END

c=======================================================================
c  Secondary but usefull equations
c=======================================================================
      REAL FUNCTION Cp_Mi (Mi_vect, dim, T)
      IMPLICIT NONE
      REAL Cp_vect(100)
      REAL Mi_vect(*)
      REAL T
      REAL VectMul
      INTEGER dim
      CALL Cp(Cp_vect, dim, T)
      Cp_Mi = VectMul(Cp_vect, Mi_vect, dim)
      RETURN
      END

c     Calculates Kappa/(Kappa - 1)
      REAL FUNCTION KappaBlock (Mi_vect, dim, T)
      IMPLICIT NONE
      REAL Mi_vect(*)
      INTEGER dim
      REAL T
      REAL Cp_vect(dim)
      REAL r_vect(dim)
      REAL VectMul
      CALL Specific_gaz_constant(r_vect,dim)
      CALL Cp(Cp_vect, dim, T)

c     After retreiving r and Cp from datas, the evaluation uses M (which
c     is provided by the solver) to compute the scalar value
      KappaBlock = VectMul(Cp_vect, Mi_vect, dim) /
     &VectMul(r_vect, Mi_vect, dim)
      RETURN
      END
```

FileLoader.f

```
      SUBROUTINE FileLoader (PressureData, TableDepth)

c=======================================================================
c  SUBROUTINE FileLoader
c=======================================================================
c  Author : Mathieu ALLORY - CVUT U2201 - April 2006
c
c  Abstract : in order to speed up the computation, datas has to be
c             preloaded from files into memory
c
c  Inputs :
c    none
c  Outputs :
c    TABLE REAL PressureData : the table containing pressure values
c    INTEGER TableDepth : the significant size of the table
c  Parameters
c    CHARACTER xxx : the names of the files that have to be preloaded
c
c=======================================================================
c     7 Start Column
      IMPLICIT NONE

c     Filenames parameters
      CHARACTER*12 FileName_PressureData
      PARAMETER (FileName_PressureData = 'pressure.dat')

c     Output variables
c     Pressure datas is a table containing angle and value
      REAL PressureData(2,*)
      INTEGER i, TableDepth

c     Open the file containing pressure datas and loads it into memory
      OPEN(9,FILE=FileName_PressureData,STATUS='old',FORM='formatted')
      i = 0
      DO WHILE (1 .LT. 3)
       i = i + 1
       READ (9,*,ERR=1000,END=1000)
     & PressureData(1,i), PressureData(2,i)
      ENDDO

1000  CONTINUE
      TableDepth = i - 1
      CLOSE(9)

      RETURN
      END
```

InitialModel.f

```
c=======================================================================
c  Author : Mathieu ALLORY - CVUT U2201 - July 2006
c
c  Abstract : The purpose of those routines is to compute the initial
c             values of masses in the cylinder.
c             This is purely theorical and the relievance of the method
c             used has to be evaluated. However, it is obvious that
c             starting the integration with null values would lead to
c             divisions-by-0 or dummy results.
c
c=======================================================================
c     7 Start Column

      SUBROUTINE Initial_Model(Masses)
      IMPLICIT NONE
c     Output, in a convenient way for the solver
      REAL Masses(15)
c     Internals
c     Geometry
      REAL V_II0, V_III0
c     Nature of fuel (CmHn)
      REAL n, m, Ot
c     For mixture composition
      REAL MixtureCompo(5)
      REAL X_O2, X_CO2, X_N2AR, X_H2O, X_Fuel
c     Integration parameter
      REAL t_ini
c     Datas at inlet closing
      REAL p_IC, V_IC, T_IC
c     Thermo constants
      REAL r_vect(5)
c     Intermediate variables
      INTEGER I
      REAL Sigma_mOZ
      REAL M_OneZone(5), M_I0(5), M_II0(5), M_III0(5)
      REAL vBuff1(5), vBuff2(5), vBuff3(5)
      REAL Mp_fuel_III, Mp_fuel_II, Mp_O2_III, Mp_O2_II

c     Functions
      REAL Pressure, getChamberVolume
      REAL VectMul
      REAL getZoneVolume_II
      REAL getZoneVolume_III, getRadius_III

c     REMINDER : Composent order in vectors:
c     O2 - N2Ar - CO2 - H2O - C3H8
      t_ini = 0.

c     Retreiving fuel composition
      CALL getFuelCompo(m, n)

c     Ot computation
      Ot = 32.*(m + n/4.) / (12.*m + n)

c     Composition of the mixture in the cylinder
      X_O2 = 0.23
```

InitialModel.f

```
      X_CO2 = 0.0002
      X_N2AR = 0.77
      X_H2O = 0.01
      X_Fuel = X_O2/Ot

      MixtureCompo(1) = X_O2 / (X_O2 + X_CO2 + X_N2AR + X_H2O + X_Fuel)
      MixtureCompo(2) = X_N2AR / (X_O2 + X_CO2 + X_N2AR + X_H2O +X_Fuel)
      MixtureCompo(3) = X_CO2 / (X_O2 + X_CO2 + X_N2AR + X_H2O + X_Fuel)
      MixtureCompo(4) = X_H2O / (X_O2 + X_CO2 + X_N2AR + X_H2O + X_Fuel)
      MixtureCompo(5) = X_Fuel / (X_O2 + X_CO2 + X_N2AR + X_H2O +X_Fuel)


c     Computing datas at inlet closing
c     /!\ T_IC in KELVIN !
c     p_IC and V_IC are taken when angle = 0deg (BDC)
      T_IC = 350.
      p_IC = Pressure(180.)
      V_IC = getChamberVolume(180.)
      CALL Specific_gaz_constant(r_vect,5)
      Sigma_mOZ = p_IC * V_IC / (VectMul(MixtureCompo, r_vect,5)* T_IC)


c     Now we are able to calculate M_OneZone, which is of course not
c     real at all but represent the 'simulated' values of masses in a
c     one zone model (before ignition, zones 2 and 3 do not exist).
      CALL VectMulNum (M_OneZone, MixtureCompo, Sigma_mOZ,5)

c     Let's evaluate 'fake' zones 2 and 3 to avoid division-by-0
c     problems during the first iteration of the general model
c     solving process
c     NOTICE : We assume that the geometric model takes
c     in consideration that zone III radius is not null when t=0
      IF (getRadius_III(t_ini) .LE. 0.) THEN
            WRITE (6,*) ''
            WRITE (6,*) '[WARNING](First Step Model)'
            WRITE (6,*) 'Zone radius seems to be 0. at initial step.'
            WRITE (6,*) '(Value :',getRadius_III(t_ini),'m)'
            WRITE(6,*) 'Results may be wrong !'
            WRITE(6,*) 'Check your geometric datas/model !'
      ENDIF

      V_II0 = getZoneVolume_II(t_ini)
      V_III0 = getZoneVolume_III(t_ini)

c     Reference mass of fuel (& O2), distributed function of
c     the fake volume of the fake zones
      Mp_fuel_III = M_OneZone(5) * V_III0 / V_IC
      Mp_fuel_II = M_OneZone(5) * V_II0 / V_IC
      Mp_O2_III = M_OneZone(1) * V_III0 / V_IC
      Mp_O2_II = M_OneZone(1) * V_II0 / V_IC

c     There could be fuel or O2 in excess
      IF ((Mp_O2_III - Mp_fuel_III * Ot) .GT. 0.) THEN
        M_III0(1) = Mp_O2_III - Mp_fuel_III * Ot
        M_III0(5) = 0.
      ELSE
```

InitialModel.f

```
        M_III0(1) = 0.
        M_III0(5) = Mp_fuel_III - Mp_O2_III / Ot
      ENDIF
c     N2+Ar, CO2, then H2O    (N2 TO CHECK !!!)
c     Stoechiometric proportions
      M_III0(2) = M_OneZone(2) * V_III0 / V_IC
      M_III0(3) = Mp_fuel_III * (m*44.) / (12.*m + n)
      M_III0(4) = Mp_fuel_III * (n/2.*18.) / (12.*m + n)

      IF ((Mp_O2_II - Mp_fuel_II * Ot) .GT. 0.) THEN
        M_II0(1) = Mp_O2_II - Mp_fuel_II * Ot
        M_II0(5) = 0.
      ELSE
        M_II0(1) = 0.
        M_II0(5) = Mp_fuel_II - Mp_O2_II / Ot
      ENDIF
c     N2+Ar, CO2, then H2O    (N2 TO CHECK !!!)
c     Stoechiometric proportions
      M_II0(2) = M_OneZone(2) * V_II0 / V_IC
      M_II0(3) = Mp_fuel_II * (m*44.) / (12.*m + n)
      M_II0(4) = Mp_fuel_II * (n/2.*18.) / (12.*m + n)

c     Now we are able to calculate M_I0 :
c     the masses for a 1 zone model minus the
c     masses of components in the 2 'fake zones'
      CALL VectMulNum(vBuff1, M_III0, -1., 5)
      CALL VectMulNum(vBuff2, M_II0, -1., 5)
      CALL VectAdd(vBuff3, vBuff1, vBuff2,5)
      CALL VectAdd(M_I0, M_OneZone, vBuff3,5)

c     Storing results in the output variable
      DO I = 1, 5, 1
        Masses(I) = M_I0(I)
        Masses(I + 5) = M_II0(I)
        Masses(I + 10) = M_III0(I)
      ENDDO

c     Conversion from kg to grams (for printing purposes only)
      CALL VectMulNum(vBuff1, M_OneZone,10.**3., 5)

c     Printing informations
      WRITE(6,*)''
      WRITE(6,*)'########## FIRST STEP APPENDIX MODEL ##########'
      WRITE(6,*)'##########         Informations      ##########'
      WRITE(6,*)'Pressure at inlet close :', p_IC * 10.**(-5.), ' bar'
      WRITE(6,*)'Mean temperature at inlet close :', T_IC, ' K'
      WRITE(6,*)'Chamber volume at inlet close :', V_IC * 10.**(3.),' L'
      WRITE(6,*)''
      WRITE(6,*)'One zone model :'
      WRITE(6,*)'  Total mass of mixture :', Sigma_mOZ*10.**3., ' g'
      WRITE(6,*)'  Per specie in grams : (O2 - N2Ar - CO2 - H2O - fuel)'
      WRITE(6,*)'  ', vBuff1
      WRITE(6,*)''
      WRITE(6,*)'Three fake zones model :'
c     Conversion from kg to grams (for printing purposes only)
      CALL VectMulNum(vBuff1, M_I0, 10.**3., 5)
```

InitialModel.f

```
      WRITE(6,*)'  Zone I (Unburnt gas)'
      WRITE(6,*)'  Per specie in grams : (O2 - N2Ar - CO2 - H2O - fuel)'
      WRITE(6,*)'  ', vBuff1
c     Conversion from kg to grams (for printing purposes only)
      CALL VectMulNum(vBuff1, M_II0, 10.**3., 5)
      WRITE(6,*)'  Zone II (Flame front)'
      WRITE(6,*)'  Per specie in grams : (O2 - N2Ar - CO2 - H2O - fuel)'
      WRITE(6,*)'  ', vBuff1
c     Conversion from kg to grams (for printing purposes only)
      CALL VectMulNum(vBuff1, M_III0, 10.**3., 5)
      WRITE(6,*)'  Zone III (Burnt gas)'
      WRITE(6,*)'  Per specie in grams : (O2 - N2Ar - CO2 - H2O - fuel)'
      WRITE(6,*)'  ', vBuff1
      WRITE(6,*)'################################################'

      END
```

Maths.f

```
c==============================================================================
c  Library Maths
c==============================================================================
c  Author : Mathieu ALLORY - CVUT U2201
c  Date : 22 Mai 2006
c==============================================================================
c  Abstract : Usefull mathematic routines
c  Summary :
c
c==============================================================================


c==============================================================================
c     Vectorial routines
c==============================================================================
c     Product of 2 vectors of dimension dim
      REAL FUNCTION VectMul (Vcol, Vlig, dim)
      IMPLICIT NONE
      REAL Vcol(*)
      REAL Vlig(*)
      INTEGER dim
      REAL sum, i, buf
      sum = 0.
      i = 1
      DO WHILE (i .LE. dim)
        buf = sum + Vcol(i)*Vlig(i)
        sum = buf
        i = i + 1
      END DO
      VectMul = sum
      RETURN
      END


c     Multiplication of a vector with a numeric
      SUBROUTINE VectMulNum (out, Vector, Numeric,Dim)
      IMPLICIT NONE
      REAL Vector(*)
      REAL out(*)
      REAL Numeric
      INTEGER i, Dim
      DO i=0, Dim, 1
        out(i) = Vector(i) * Numeric
      END DO
      RETURN
      END


c     Addition of two vectors
      SUBROUTINE VectAdd (out, V1, V2, Dim)
      IMPLICIT NONE
      REAL V1(*), V2(*)
      REAL out(*)
      INTEGER i, Dim
      i = 0
      DO i = 1, Dim, 1
```

Maths.f

```
        out(i) = V1(i) + V2(i)
      END DO
      RETURN
      END


c     Addition of a vector and a numeric
      SUBROUTINE VectAddNum (out, V1, num, Dim)
      IMPLICIT NONE
      REAL V1(*), num
      REAL out(*)
      INTEGER i, Dim
      i = 0
      DO i = 1, Dim, 1
        out(i) = V1(i) + num
      END DO
      RETURN
      END


c     Carbon copy of a vector
      SUBROUTINE VectCopy(Out, In, Dim)
      IMPLICIT NONE
      INTEGER Dim, I
      REAL Out(*), In(*)
      DO I = 1, Dim, 1
        Out(I) = In(I)
      ENDDO
      RETURN
      END

c=======================================================================
c     Numeric routines
c=======================================================================
c     Numerical derivation of a function taken in t
c     applying the very simple algorithm of Lagrange
      SUBROUTINE Diff(out, F, t, timestep)
      IMPLICIT NONE
      EXTERNAL F
      REAL F
      REAL t, timestep, out
      REAL t_prec
      t_prec = t-timestep/2.
c     ======= DEBUG TAG =======
      IF ((t_prec) .LE. 0.) THEN
c       WRITE(6,*) '[DIFF] Warning : Derivation while t-timestep/2.<0 !'
        t_prec = 0.
      ENDIF

      out = (F(t+timestep/2.)-F(t_prec))
     &/ ((t + timestep/2.)-(t_prec))
      RETURN
      END


c=======================================================================
```

Maths.f

```
c     Matrix routines
c=======================================================================
c     Cartesian norm of a matrix
      REAL FUNCTION Norm(matrix, row, col)
      IMPLICIT NONE
      INTEGER row, col
      REAL matrix(row,col)
      INTEGER i,j
      REAL sum
      i = 1
      j = 1
      sum = 0.
      DO WHILE (i .LE. row)
        DO WHILE (j .LE. col)
          sum = sum + matrix(i,j)**2.
          j = j + 1
        ENDDO
        i = i + 1
      ENDDO
      Norm = SQRT(sum)
      RETURN
      END


c     Routine to extract a column in a matrix
c     out : the extracted column
c     in :  the input matrix
c     col : the column to extract (integer)
c     nb_row : number of rows of the matrix (int)
c     Please ensure the dimension of out vector is large enough
      SUBROUTINE getColFromMatrix(out, in, col, nb_row)
      IMPLICIT NONE
c     Inputs
      INTEGER col, nb_row
      REAL in(nb_row, col)
c     Outputs
      REAL out(*)
c     Internals
      INTEGER I
      DO I = 1, nb_row, 1
        out(I) = in(I, col)
      ENDDO
      RETURN
      END

c=======================================================================
c     Solver routine
c=======================================================================

c     The differential equations solver
c     This is an overhead routine for DEABM solver from SLATEC library
c     (the Adams-Bashforth solver from SLATEC, recommended for the
c     computation of a large number of points)
c     It could be modified to use another algorithm, following the
c     prototype described below
      SUBROUTINE DEABMSolver(F, NEQ, T_INI, T_FIN, TIMESTEP, Y0,
```

Maths.f

```
     & RESULT, DIMRES, DIAG)

c      SUBROUTINE RK2Solver(F, NEQ, T_INI, T_FIN, TIMESTEP, Y0,
c     &results, lines, diag)

      IMPLICIT NONE
c     The subroutine defining the system
      EXTERNAL F
c     The number of equations in the system
      INTEGER NEQ
c     The initial conditions on Y
      REAL Y0(*)
c     Time parameters
      REAL T_INI, T_FIN, TIMESTEP
c     Tolerances
      REAL RTOL, ATOL

c     Internal variables
      INTEGER I, J, IDID, INFO(5)
      REAL T, TOUT, R_RTOL, R_ATOL

c     Out
      INTEGER DIMRES, COMM
      REAL RESULT(NEQ+1,*)
      REAL DIAG(12, *)

c     Dummy parameters (used to pass arguments to F)
      INTEGER IPAR(1)
      REAL RPAR(12)

c     RWORK & IWORK are buffers for DERKF
      INTEGER LRW, LIW
      REAL RWORK(10000), IWORK(10000)
      LRW = 10000
      LIW = 10000

      INFO(1) = 0
      INFO(2) = 0
      INFO(3) = 0
      INFO(4) = 0

c     Timestep tolerances RTOL & ATOL are now arbitrary set
      RTOL = 0.000000001
      ATOL = 0.000000001

c     Preparing parameters for first iteration
      T = T_INI
      TOUT = T_INI + TIMESTEP
      INFO(1) = 0
      INFO(2) = 0
      INFO(3) = 0
      INFO(4) = 0
      R_ATOL = ATOL
      R_RTOL = RTOL
      COMM = 0
      I=1
```

Maths.f

```
      WRITE (6,*) '[DEABMSolver] Calibrating solver...'
c     Initialization of the problem
      CALL DEABM (F, NEQ, T, Y0, TOUT, INFO, RTOL, ATOL, IDID,
     +  RWORK, LRW, IWORK, LIW, RPAR, IPAR)


      IF (IDID .EQ. -2) THEN
        WRITE (6,*) 'WARNING : Requested tolerance for DE solving is to
     &stringent.'
        WRITE (6,*) 'Slatec determined new values.'
        WRITE (6,*) 'Req. ATOL :', R_ATOL, ' Used ATOL :', ATOL
        WRITE (6,*) 'Req. RTOL :', R_RTOL, ' Used RTOL :', RTOL
        COMM = 1
      ENDIF

      IF ((IDID .LT. 0) .AND. (IDID .NE. -2)) THEN
        WRITE (6,*) 'ERROR : An error occured during DE solving (init).'
        WRITE (6,*) 'Results may be inacurrate. See SLATEC doc.'
        WRITE (6,*) 'SLATEC error code : IDID=',IDID
        COMM = 2
      ENDIF

c     Storing result
      RESULT(1,I) = TOUT
      J = 1
      DO WHILE (J .LE. NEQ)
        RESULT(J+1,I) = Y0(J)
        J = J + 1
      ENDDO
      DO J=1, 12, 1
        diag(J, I) = RPAR(J)
      ENDDO


      I = I + 1

      IF (T .EQ. TOUT) THEN
        TOUT = TOUT + TIMESTEP
        WRITE (6,*) '[DEAMBSolver] TOUT reached during first iteration'
        WRITE (6,*) '[DEAMBSolver] Calibration succeded'
      ELSE
        WRITE (6,*) '[DEAMBSolver] Cannot reach TOUT during first
     &iteration'
      ENDIF

c     Indicates we want to continue solving the same problem
      INFO(1)=1

c     Iterate call of DEABM
      DO WHILE (T .LE. T_FIN)
         WRITE (6,*) '[DEABMSolver] Entering iteration ', I,
     & '(T=', T,')'

        CALL DEABM (F, NEQ, T, Y0, TOUT, INFO, RTOL, ATOL, IDID,
     +  RWORK, LRW, IWORK, LIW, RPAR, IPAR)
```

Maths.f

```
        IF ((IDID .LT. 0) .AND. (IDID .NE. -2)) THEN
        WRITE (6,*) 'ERROR : An error occured during DE solving at T=',T
        WRITE (6,*) 'Results may be inacurrate. See SLATEC doc.'
        WRITE (6,*) 'SLATEC error code : IDID=',IDID
        COMM = 3
        INFO(1) = 1
        ENDIF
c     Storing results
        RESULT(1,I) = TOUT
        J = 1
        DO WHILE (J .LE. NEQ)
          RESULT(J+1,I) = Y0(J)
          J = J + 1
        ENDDO
        DO J=1, 12, 1
          diag(J, I) = RPAR(J)
        ENDDO
        I = I + 1

c     Going a step ahead...
        TOUT = TOUT + TIMESTEP
      ENDDO

      DIMRES = I - 1

      TOUT = TOUT - TIMESTEP
      WRITE (6,*)'[DEABMSolver] Computation ended (', TOUT*10.**3.,'ms)'

1000  CONTINUE
      RETURN
      END


c     A simple second order Runge-Kutta solver
c     This can be used instead of Slatec Adams-Bashforth
c     routine. It is probably less accurate but faster and better
c     for testing the code as it is everything but a black-box.
c     ***********************************************************
c     SPECIFICATIONS :
c     Inputs :
c     * REAL EXTERNAL F : The function to solve
c     * INTEGER NEQ : The number of equations in the system
c     * REAL T_INI : Initial value of independant parameter T
c     * REAL T_FIN : Final value of independant parameter T
c     * REAL TIMESTEP : The timestep
c     * REAL Y0(NEQ) : Vector of initial values for time
c                                  dependant parameter Y
c     Outputs :
c     * REAL results(NEQ+1,*) : The table where the results will
c                                  be saved. Has to be large
c                                  enougth, or a sigfault will
c                                  occur.
c     * REAL diag(11,*) : Returns intermediate values for
c                           diagnostic purposes (volumes, pressure...)
c     * INTEGER lines : Number of iteration processed
c     ***********************************************************
```

Maths.f

```
      SUBROUTINE RK2Solver(F, NEQ, T_INI, T_FIN, TIMESTEP, Y0,
     &results, lines, diag)
      IMPLICIT NONE
c     The subroutine defining the system
      EXTERNAL F
c     The number of equations in the system
      INTEGER NEQ
c     The initial conditions on Y
      REAL Y0(*)
c     Time parameters
      REAL T_INI, T_FIN, TIMESTEP

c     Results
      REAL results(NEQ+1, *)
      REAL diag(12, *)
      INTEGER lines

c     Internals
      INTEGER I
      REAL T
      REAL YN(NEQ), Y(NEQ), YP(NEQ), vBuff1(NEQ), vBuff2(NEQ)
      REAL K1(NEQ), K2(NEQ)
      REAL DummyR(12)
      REAL CurDiag(12)
      INTEGER DummyI
      INTEGER Iteration

      Iteration = 1
      T = T_INI
c     First step : Copying initial values
      results(1, Iteration) = T
      DO I=2, NEQ+1, 1
            results(I, Iteration)= Y0(I-1)
            YN(I-1) = Y0(I-1)
      ENDDO

      Iteration = Iteration + 1
c      T = T + TIMESTEP

c     Computation loop
      DO WHILE (T .LE. T_FIN)

c     Message
            WRITE (6,*) '[RK2SOLVER] Entering iteration ', Iteration,
     & '(T=', T,')'

c     K1 evaluation
            CALL F(T, YN, YP, CurDiag, DummyI)
            CALL VectMulNum(K1, YP, TIMESTEP, NEQ)

c     K2 evaluation
            CALL VectMulNum (vBuff1, K1, 0.5, NEQ)
            CALL VectAdd(vBuff2, vBuff1, YN, NEQ)
            CALL F(T + TIMESTEP / 2., vBuff2, YP,
     &DummyR, DummyI)
            CALL VectMulNum(K2, YP, TIMESTEP, NEQ)
```

Maths.f

```
c     Y contains the result of the iteration
            CALL VectAdd(Y, YN, K2, NEQ)

             T = T + TIMESTEP

c     Storing results and YN for next iteration
            results(1, Iteration) = T
            DO I=2, NEQ+1, 1
                   results(I, Iteration)= Y(I-1)
                   YN(I-1) = Y(I-1)
            ENDDO
c     Storing diagnostic values
            DO I=1, 12, 1
                  diag(I, Iteration)= CurDiag(I)
            ENDDO

            Iteration = Iteration + 1
c            T = T + TIMESTEP

       ENDDO

      lines = Iteration - 1
      T = T - TIMESTEP
      WRITE (6,*) '[RK2SOLVER] Computation ended (', T*10.**3.,'ms)'

      RETURN
      END


c=======================================================================
c     Interpolation routines
c=======================================================================

      SUBROUTINE Interpolate(out, known_values, nb_samples, T)
      IMPLICIT NONE
c     Output
      REAL out(5)
c     Inputs
      INTEGER nb_samples
      REAL known_values(nb_samples,6)
      REAL T
c     Internals
      INTEGER i,j

      IF (T .LE. known_values(1,1)) THEN
            out(1) = known_values(1,2)
            out(2) = known_values(1,3)
            out(3) = known_values(1,4)
            out(4) = known_values(1,5)
            out(5) = known_values(1,6)
      ENDIF

      IF (T .GE. known_values(nb_samples, 1)) THEN
            out(1) = known_values(nb_samples, 2)
            out(2) = known_values(nb_samples, 3)
            out(3) = known_values(nb_samples, 4)
```

Maths.f

```
            out(4) = known_values(nb_samples, 5)
            out(5) = known_values(nb_samples, 6)
      ENDIF

      IF ((T .GT. known_values(1,1)) .AND.
     &(T .LT. known_values(nb_samples, 1))) THEN
         DO i = 1, nb_samples - 1, 1
           IF ((known_values(i, 1).LT.T)
     &        .AND.(known_values(i+1,1).GE.T)) THEN
            DO j = 2, 6, 1
c     Cp is linearly interpolated
              out(j-1) = ((known_values(i+1,j)-known_values(i, j))/
     &        (known_values(i+1, 1)-known_values(i, 1))) *
     &        (T - known_values(i, 1)) + known_values(i, j)
            ENDDO
           ENDIF
         ENDDO
      ENDIF

      RETURN
      END
```

Pressure.f

```
      REAL FUNCTION Pressure (angle)

c=======================================================================
c  FUNCTION Pressure
c=======================================================================
c  Author : Mathieu ALLORY - CVUT U2201 - April 2006
c
c  Abstract : returns the pressure, considered as uniform in the
c             cylinder, from the crank angle
c
c  Inputs :
c    REAL angle : the crank of which you want the pressure (in degree)
c  Outputs :
c    RETURN REAL : the pressure value
c
c=======================================================================
c     7 Start Column
      IMPLICIT NONE

c     Variables
      REAL angle, out
      REAL PressureDatas(2,10000)
      INTEGER index, TableDepth, i
      LOGICAL found

      out = 0.
      found = .FALSE.

c    TABLE REAL PressureDatas : the table that contains the pressure
c                               values for different values of the
c                               crank angle
c    INTEGER TableDepth : the real significant size of the table.
      CALL  getPressureDatas(PressureDatas, TableDepth)

c     Assuming that we have a value of the pressure for each entire
c     value of angle between 0 and 719 degree, we will first consider
c     the index and the first column of the table are equals

      index = MOD(INT(angle), 719) + 1

      IF (MOD(INT(angle), 719) .EQ. PressureDatas(1, index)) THEN
c     Assuming that we have a value of the pressure for each entire
c     value of angle between 0 and 719 degree, we will first consider
c     the index and the first column of the table are equals
       found = .TRUE.

      ELSE
c      Otherwise the entire table has to be scanned to determine where
c      to take the values
       i = 1
       DO WHILE (i .LT. TableDepth - 1)
        IF (PressureDatas(1, i) .LE. angle) THEN
         IF (PressureDatas(1, i+1) .GE. angle) THEN
            index = i
            found = .TRUE.
         ENDIF
        ENDIF
```

Pressure.f

```
        i = i + 1
       ENDDO

      ENDIF

c     The requested value has to be calculated using linear
c     interpolation between the two nearest known points
      out = PressureDatas(2, index) +
     & (angle - PressureDatas(1, index))*
     & (PressureDatas(2, index + 1) - PressureDatas(2, index))/
     & (PressureDatas(1, index + 1) - PressureDatas(1, index))

      IF (found .EQV. .FALSE.) THEN
       WRITE (6,*) 'Error parsing pressure value for crank angle ';angle
       WRITE (6,*) 'Returned value ', out, ' might be wrong.'
      ENDIF

c     Pressure is read in Bar, we need to convert it to Pascal
      Pressure = out * 10.**5.

      RETURN
      END

c     An alias giving pressure from time instead of crank angle
c     (usefull mainly for derivation)
      REAL FUNCTION Pressure_from_time(time)
      IMPLICIT NONE
      REAL time, Pressure, getAngle
      Pressure_from_time = Pressure (getAngle(time))
      RETURN
      END
```

ReactionRate.f

```
c=======================================================================
c  Library ReactionRate
c=======================================================================
c  Author : Mathieu ALLORY - CVUT U2201
c  Date : 22 Mai 2006
c=======================================================================
c  Abstract : These routines are used to get reaction rate rr from datas
c  Summary :
c
c=======================================================================

c     We assume that the reaction rate is non zero only in the flame
c     zone

      SUBROUTINE rr_I(reaction_rate, dim, t)
      IMPLICIT NONE
      REAL reaction_rate(5)
      REAL t
      INTEGER dim
      dim = 5
      reaction_rate(1) = 0.
      reaction_rate(2) = 0.
      reaction_rate(3) = 0.
      reaction_rate(4) = 0.
      reaction_rate(5) = 0.
      RETURN
      END


      SUBROUTINE rr_III(reaction_rate, dim, t)
      IMPLICIT NONE
      REAL reaction_rate(5)
      REAL t
      INTEGER dim
      dim = 5
      reaction_rate(1) = 0.
      reaction_rate(2) = 0.
      reaction_rate(3) = 0.
      reaction_rate(4) = 0.
      reaction_rate(5) = 0.
      RETURN
      END


      SUBROUTINE rr_II(reaction_rate, dim, t, m_I, m_II, m_III)
      IMPLICIT NONE
      REAL reaction_rate(5)
      REAL t
      INTEGER dim, col_rm, row_rm
      REAL m_I(*), m_II(*), m_III(*)
      REAL rm(5,1)
      REAL Vbr_rm(5)
      REAL Vrr_corr(5)
      REAL first_member(5)
      REAL second_member(5)
      REAL rr_fuel
```

ReactionRate.f

```
c     The functions we are going to use
      REAL Norm
      dim = 5
c     Retreives the reaction matrix
      CALL reaction_matrix(rm, row_rm, col_rm)

c     Build the burning reaction vector
      Vbr_rm(1) = rm(1,1)
      Vbr_rm(2) = rm(2,1)
      Vbr_rm(3) = rm(3,1)
      Vbr_rm(4) = rm(4,1)
      Vbr_rm(5) = rm(5,1)

c     First member : only burning reaction
      CALL rdot_II_fuel (rr_fuel, m_I, m_II, m_III, t,dim)
      CALL VectMulNum(first_member, Vbr_rm, rr_fuel,5)

c     Second Member : Corrective reactions
      CALL correction_vector(Vrr_corr, t)
      CALL VectMulNum(second_member, Vrr_corr, Norm(rm,1), 5)

c     Addind the two members
      reaction_rate(1) = first_member(1) + second_member(1)
      reaction_rate(2) = first_member(2) + second_member(2)
      reaction_rate(3) = first_member(3) + second_member(3)
      reaction_rate(4) = first_member(4) + second_member(4)
      reaction_rate(5) = first_member(5) + second_member(5)

      RETURN
      END
```

ZoneApproach.f

```
      PROGRAM ZoneApproach
      IMPLICIT NONE

      REAL timestep
      EXTERNAL Core_Equations_System

c     Tables to store the results
      REAL RESULT(16,10000)
      REAL diag(12,10000)
      REAL Y0(15)
      INTEGER lines
c     Functions
      REAL getTime


      WRITE (6,*) '************************************************'
      WRITE (6,*) '   Zone Approach model for determination of'
      WRITE (6,*) '  premixed flame parameters computational code'
      WRITE (6,*) '                      .::.'
      WRITE (6,*) '  Josef Bozek Research Center of Engine and '
      WRITE (6,*) '           Automotive Engineering'
      WRITE (6,*) '         July 2006 - Mathieu ALLORY '
      WRITE (6,*) '************************************************'
      WRITE (6,*)

c     Prints information about the geometric model
      CALL printGeoModelInfos()

c     Computation of initial conditions using the alternate
c     initial model
c     O2, N2+Ar, C02, H2O, Fuel
      DATA Y0 / 0., 0., 0., 0., 0.,
     &0., 0., 0., 0., 0.,
     &0., 0., 0., 0., 0. /
      CALL Initial_Model(Y0)
      CALL getIntegrationParameter_Timestep(timestep)

c     Starting integration process
      WRITE (6,*) ''
      WRITE (6,*) 'Starting integration process with a time step of ',
     &timestep * 10.**3.,' ms'

c      GOTO 10
      CALL RK2Solver(Core_Equations_System,15, getTime(340.),
     &getTime(365.6), timestep, Y0, RESULT, lines, diag)
c      CALL DEABMSolver(Core_Equations_System, 15, getTime(340.),
c     &getTime(365.6), timestep, Y0, RESULT, lines, diag)


c     Saving results to disk using the C library iolib.c
      CALL writeresults(lines, RESULT)
      CALL writediags(lines, diag)
10    CONTINUE

      STOP
      END
```

ZoneApproach.f

```
c     This subroutine retreives preloaded datas for pressure in a common
c     to send them to the interpolation routine
      SUBROUTINE getPressureDatas(PressureDatas, TableDepth)
      IMPLICIT NONE
      REAL PressureDatas(2, 1000)
      INTEGER TableDepth
      CALL FileLoader (PressureDatas, TableDepth)
      RETURN
      END
```

iolib.c

```
// Standard outputs library
#include <stdio.h>

// Index translation between C and Fortran
#define TABLE(j,i) table[i][j]

char* message = "Saving results to";
char* filename = "results.txt";
char* header = "#Results for premixed flame parameters simulation\n#Time\tZone
I, O2\tZone I, N2Ar\tZone I, CO2\tZone I, H2O\tZone I, Fuel\tZone II, O2\tZone
II, N2Ar\tZone II, CO2\tZone II, H2O\tZone II, Fuel\tZone III, O2\tZone III,
N2Ar\tZone III, CO2\tZone III, H2O\tZone III, Fuel\n";
char* diagfilename = "diags.txt";
char* diagheader = "#Diagnostic values\n#Time\tAngle\tZone I Volume\tZone II
Volume\tZone III Volume\tPressure\tZone I Temp\tZone II Temp\tZone III Temp\tFF
Front Velocity\tFF Back Velocity\tROB\n";

// Format and write results on disk, to bypass fortran restrictions
void writeresults_(int *lines, float table[10000][16])
        {
                FILE *file;
                int i=0;
                int j=0;
                printf("\n *** C output library for F77 : %s %s\n",message,
filename);
                // Opening file
                file = fopen (filename,"wt");
                printf(" *** Writting %d lines.\n", *lines+2);
                // Writting header
                fprintf(file, "%s", header);
                // Writting values
                for (i=0; i < *lines; i++)
                {
                        for (j=0; j < 16; j++)
                        {
                                fprintf(file,"%.7e\t", table[i][j]);
                        }

                        fprintf(file,"\n");
                }
                // Closing file
                fclose(file);
        }


void writediags_(int *lines, float table[10000][12])
        {
                FILE *file;
                int i=0;
                int j=0;
                printf("\n *** C output library for F77 : %s %s\n",message,
diagfilename);
                // Opening file
                file = fopen (diagfilename,"wt");
                printf(" *** Writting %d lines.\n", *lines+2);
                // Writting header
```

iolib.c

```
                fprintf(file, "%s", diagheader);
                // Writting values
                for (i=0; i < *lines; i++)
                {
                        for (j=0; j < 12; j++)
                        {
                                fprintf(file,"%.7e\t", table[i][j]);
                        }

                        fprintf(file,"\n");
                }
                // Closing file
                fclose(file);
        }
```

makefile

```
CCF=g77

CC=gcc
CFLAGS=-Wall -g
LDFLAGS=-Wall -g -Wl,-rpath=/usr/local/lib64/
EXEC= zone
SLATECDIR= slatec
GEOMDIR= geometry

SRC= ZoneApproach.f ReactionRate.f Datas.f Equations.f FileLoader.f Pressure.f
Maths.f InitialModel.f
SLATEC = $(SLATECDIR)/slatec_x86.a
GEOMETRY = $(GEOMDIR)/geometry.a
OBJ= $(SRC:.f=.o) iolib.o $(GEOMETRY) $(SLATEC)

all: $(OBJ)
        @( $(MAKE) iolib)
        @(cd $(SLATECDIR) && $(MAKE))
        @(cd $(GEOMDIR) && $(MAKE) library)
        $(CCF) -o $(EXEC) $^ $(LDFLAGS)
        rm -f *.o

%.o: %.f
        $(CCF) -c $^ $(CFLAGS)

iolib:
        gcc -c iolib.c $(CFLAGS)

clean:
        rm -f *.o
```