Modelling flight dynamics for real time simulator applications

How Mathematica could be used for to design and delevop flight dynamics models

Laurent Farenc

On behalf of Airbus Flight dynamics simulation for Airbus design office

■ 1 − Abstract

Real time flight mechanics simulation requires the development of efficient time discrete models. These models allows to simulate, for instance:

- Flight mechanics equations
- Aircraft aerodynamic coefficients
- Aircraft flexibility
- Aircraft inertial properties (including fuel loading and displacements)
- Aircraft landing gears
- Aircraft engines
- Aircraft control surfaces
- Aircraft systems
- Etc ...

These models are integrated on cockpit real time simulators that underly aircraft development activities.

Up to recent years these models were often hand coded with C or Fortran language. Besides the cost, obvious issues are:

- Consistence of code with its documentation
- The validation
- Control of programming rules
- Etc ..

It becomes clear that new technologies need to be investigated to improve our modelling processes, and keep competitive. This presentation aims at presenting how *Mathematica* could be used to propose an original and innovative way to improve our modelling activities.

This presentation outlines the using of *Mathematica* envisaged within Airbus in the domain of real time flight dynamics simulation. Concerning this using a certain amount of results are now established, although a lot of objectives still need to be investigated, current progress has clearly demonstrated the

feasibility of a wide application of *Mathematica* in the frame of our activities.
This presentation does not state any remarkable results in terms of symbolic calculation, it is rather the relation of a detailed study for an industrial application of *Mathematica* by Airbus.

■ 2 − Quick presentation of our needs

At short term, our main need is to set up modelling tools:

- To design and develop discrete time models, in an efficient environment, and allowing the control of a fixed step numerical scheme.
- To automate C code generation
- To automate the production of the documentation compliant with the code

Mid term objectives are:

- To fit the model with features allowing its tuning against actual equipment tests
- To fit the models with features allowing to support flight control laws development, for instance linearization, stability margins, spectral performances, etc ..
- To set up code optimisation methods
- To set up numerical error assessment methods

Furthermore, we add the constraint of the ergonomy, allowing the engineer to describe its models using a classical mathematical typesetting

To meet these objectives, an analysis of existing technologies has been launched.

■ 3 − Which technology ?

□ 3.1 − Short discusion

Our approach has been to start with an analysis of existing technologies to develop efficient modelling tools. It is not useful here to make the inventory of technologies explored, let say simply that *Mathematica* quickly appeared as a serious competitor, mainly at the beginning for its capability to interpret an elaborated mathematical typesetting, allowing to envisage to describe physical assumptions of our models under a "natural" formalism for engineers.

The most serious rivals are the "Scientific Calculation Interpretors" (that we we will later on oppose to "Symbolic Calculation Interpretors"), like MatLab, SciLab, MathCad, etc ..., with or without a block set interface like Simulink, AMESim, etc ....
These scientific calculation interpretors are already widely used by Airbus and its suppliers, they often natively feature C code generation (like the famous Real Time Workshop (RTW) of The MathWorks).

The qualities of scientific interpretors are well known, and it is not necessary to insist here, more interesting is to point out some deficiencies:

- Nearly lack of capability to interpret an elaborated mathematical typesetting. Conversely, an elaborated mathematical typesetting is the *Mathematica* natural interface with its users.
- Concerning automatic C code generation features associated to scientific interpretors, when it exist, recurrent criticisms are addressed about performances, the lack of readability of the code, the difficulties to access internal variables.

- Concerning documentation generation, if we reject the block−set formalism, the potential of scientific interpretors to generate a satisfying documentation is very limited.

To sum up, in favour of symbolic interpretors, and especially *Mathematica,* the mathematical typesetting of the front end, attractive alternative to the block−set formalism, the transformational paradigm allowing to seriously envisage the transformation of physical equations into C code, is the cocktail that has encouraged us to assess *Mathematica* potential for our activities.

To go further a feasibility study, based around an aircraft landing gear model is presented here after.

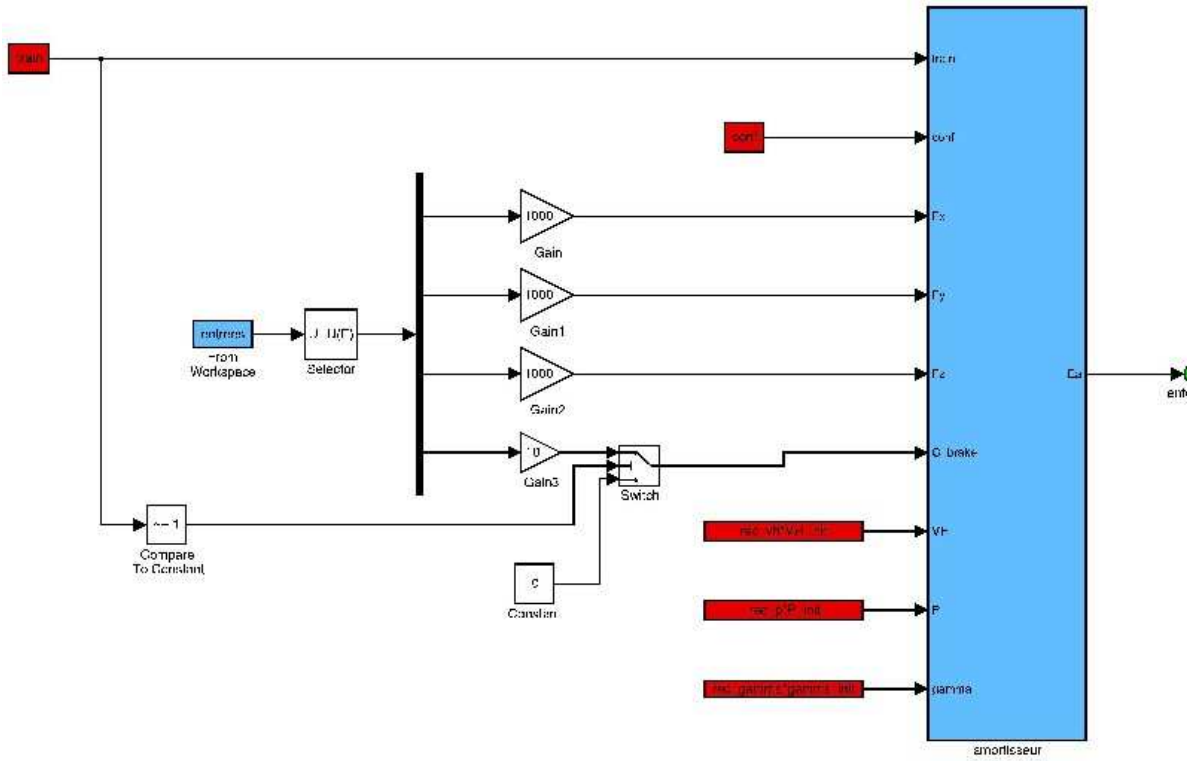
□ 3.2 − Modelling an aircraft landing gear

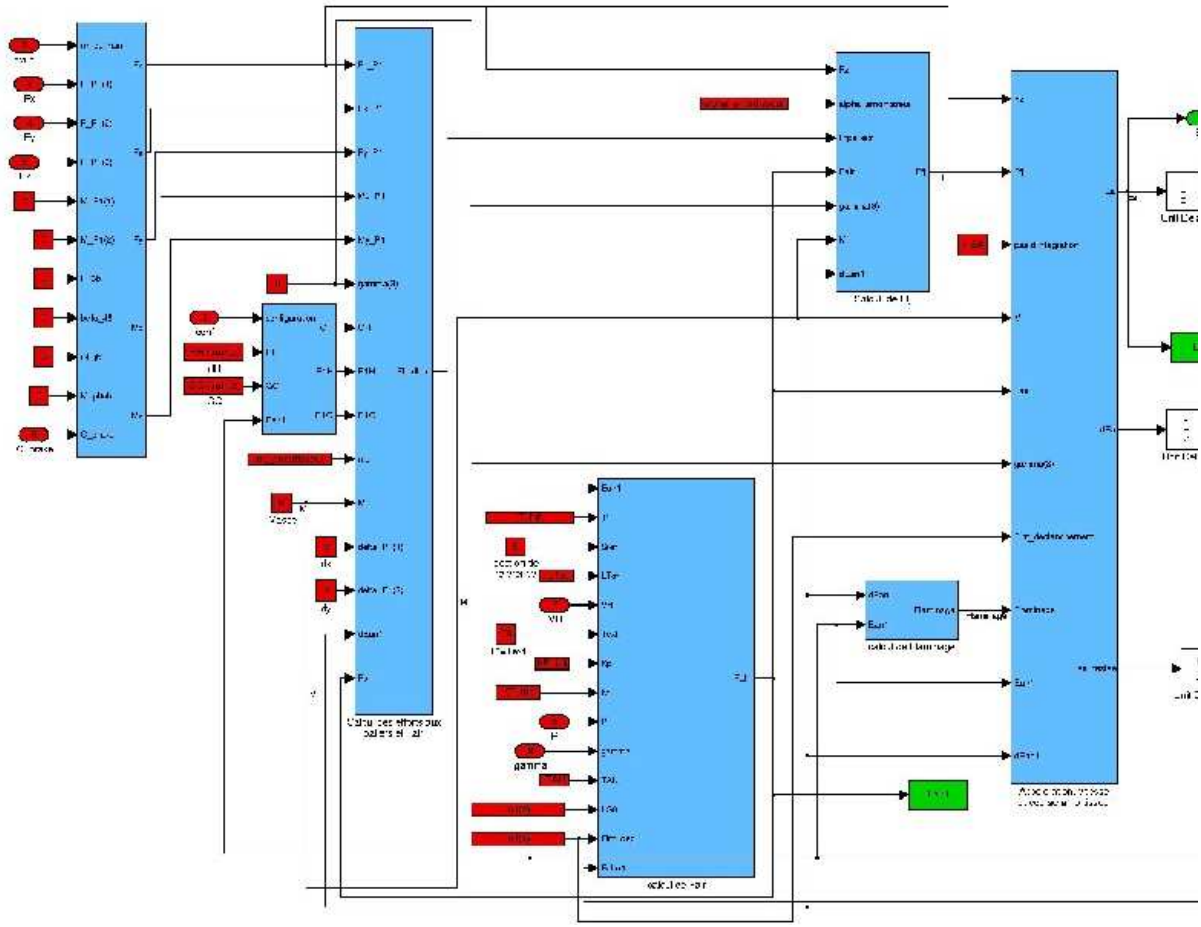
3.2.1 − With Mathematica

The model presented here after is widely simplified for confidentiality reasons.

Hand coded *Mathematica* landing gear model

3.2.2 − With a block−set interface





Note : these schemes aims only at illustrating the complexity of the block set formalism on a similar model, but does not exactly match the *Mathematica* model previously presented.

3.2.3 − Discusion

The first result is that feasibility of modelling a complex mechanical system, like a landing gear, with *Mathematica,* and with acceptable performances to simulate a time sequence, is demonstrated.
Concerning the modelling formalism, a quick survey in the opinion of mechanical engineers gives a strong advantage in favour of *Mathematica,* compared to scientific interpretors, for its conciseness and clarity, the possibility offered to describe physics assumptions with an appropriate mathematical typesetting.

The strong points, with the using of *Mathematica,* to be mentioned are:

- The declarative paradigm, allowing to free the engineer from an order to describe physics assumptions, and also allowing to individually test/evaluate/visualize physics assumptions without affecting global integrity of the modelling.
- The using of memory modules, allowing to test easily and individually the discrete numerical schemes.

To balance, we must notice that in terms of performances it seems that there is a strong advantage in favour of scientific interpretors.

□ 3.3 − Outcome

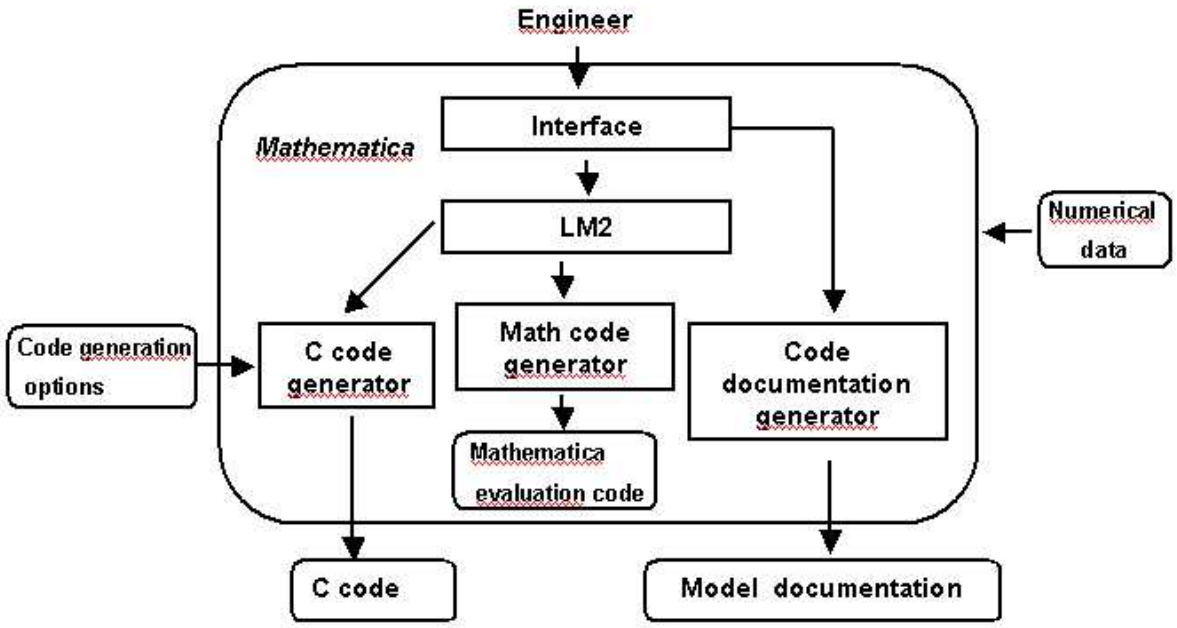
The exercise showed clearly the feasibility of a modelling tool based on *Mathematica*. The next section present a prototype of the modelling tool envisaged.

■ 4. Prototype

□ 4.1 − General principle

The feasability of the use of *Mathematica* established, we decided to develop a prototype for the modelling tool of our dreams.
For this purpose we requested the support of INRIA (French national research institute: "Institut National de Recherche en Informatique et en Automatique"), which helped us to develop this prototype.

The general principle is summed up hereafter.



The engineer describes its model with a *Mathematica* interface allowing to generate what we called hereafter an LM2 representation, for the French acronym "Langage de Modelisation en *Mathematica*" ("*Mathematica* Modelling Language").
From the LM2 representation, the model can be transformed using

- The Math Code Generator package, allowing to evaluate and validate the model with the *Mathematica* kernel.
- The C Code Generator package, allowing to generate a real time C code

From the interface code, it will be also possible to generate the code documentation (likely with an XML format)

Finally, the Math Code Generator and the C Code generator are associated with a modelling data model to handle numerical data.

The strong idea is that from a unique representation, the Interface code, it is possible to generate a consistent set of *Mathematica* code for direct evaluation, the C code and its documentation.

□ 4.2 − The LM2 representation

4.2.1 − Principle

On the basis of the landing gear model, presented here above, we have identified the minimum basic concepts we need to design our models

- To define the constants, i.e., variables that do not vary with time
- To define continuous formulas that describe the system to be modelled, i.e., kinematic relations, physical descriptions, etc ...
- To define discrete algorithms to solve the dynamics of the system modelled (time differential equations), and associated with these algorithms, the discrete variables for which time history need to be memorised.
- And to be complete, we also defined a modelling data model, to handle numerical data: a format for numerical data and a set of interpolation tools.

4.2.2 − Example

LM2 interface code for the landing gear model

□ 4.3 − The C code generation

4.3.1 − Discusion

The main question is: why develop a C code generator while it already exists an off the shelf C code generator, that we could apply on the *Mathematica* evaluation code, generated from the LM2 representation?
There are a few good reasons for this choice:

- A first point is that the durability of off the shelf *Mathematica* to C translators is not ensured.
- Another good reason is that we would like to avoid dealing with issues raised by a non specialised code generator (like RTW for instance): performances, readability, variable naming, access to internal variables, etc. Code performances are of major importance, the models are intensively used to perform parametric studies that require markedly better performances than real time.
- Furthermore, we would like to ensure future extensions: code instrumentation to support aircraft systems development, code instrumentation for error assessment, etc ...

4.3.2 − Principle

The code generator is composed of two parts. The first part is a C wrapper, allowing to translate a *Mathematica* representation of a C code, into a real C code. This wrapper is not specific to our code generator, and can be used for other purposes, it has been developed by INRIA.
The second part, allow to calculate a C code representation from the LM2 representation of the model

4.3.3 − Example

LM2 interface code for the landing gear model

■ 5. Conclusions

The prototype developed with the support of INRIA is extremely promising. The feasibility of an industrialisation of the prototype is now ensured. The use of the package already developed to lead research activities is also envisaged.