Automatic differentiation for propagation of orbit uncertainties on Orekit

Antolino Andrea, Luc Maisonobe

1 Automatic Differentiation

Automatic differentiation has been developed as a mathematical tool to avoid the calculations of the derivatives of long equations. The interesting part is that the resulting computation is accurate to the precision of the computer system and the computation is equivalent of calculating f(x) without having to find the analytical expression.

So where is the difference with the others systems of differentiation? The analytical, or symbolic, way takes an equation f(x), calculates the analytical form of the derivative f(x) and applying it to a particular value finds the derivative with respect of the symbol it has been chosen. The problem of this method is that for higher orders, complicated equations or equations spread over a large number of functions in some programming language, it is hard to implement. The advantages of this method is that its very precise, rapid and efficient.

The numerical way just calculates in the surrounding of x a few values for f(x) then evaluates the derivative with a precision related to the number of points taken by the first part of the calculation. This procedure has been widely used when the analytical way was too hard to implement, the useful part is that it is adaptable to a lot of situations, it just needs an implementation of f(x), various points to evaluate the derivative and few more things. The drawbacks are that the precision relies on both the number of points and the step size in such a way finding the proper settings is often a trial and error process, and the computation cost is directly proportional to the step size. Another less obvious drawback is that if the implementation of f(x) retains some intermediate results in a cache, typically to speed up long computation,

Antolino Andrea
C-S Systemes d'information, 5 rue Brindejong des Moulinais e-mail: andrea.antolino@c-s.fr

Maisonobe Luc
C-S Systemes d'information, 5 rue Brindejong des Moulinais e-mail: luc.maisonobe@c-s.fr

this has a side effect on finite differences evaluation and may imply adapting the caching feature.

An interesting part of the automatic differentiation is that once the low level differentiation framework is available, its behavior and use is exactly the same as a normal real number. Of course the calculations will be slower because instead of calculating real numbers we are dealing with vectors of reals [2] but the resulting accuracy will be exactly the same as the analytical model. So it could be harder to write the differentiation framework but, once this operation has been done, the utilization is immediate, and in every moment we will have access to the derivatives needed.

2 Differentiation Framework

The framework we used for this purpose is provided by the Hipparchus library [4] . This library is a free software mathematical library published under the Apache V2 license. It is the successor of the Apache Commons Math library.

This library has already implemented completely an Automatic Differentiation framework thanks to its DerivativeStructure (DS) class. This class allow computation of derivatives with respect to any number of parameters, and to any derivation order. The DS implements all the methods used by a double: the operations, the exponentials, the trigonometrical functions, etc. In any algorithm using primitive double numbers, these numbers can be replaced by DS instances without changing the structure of the algorithm. One nice feature of Hipparchus differentiation framework is that the DS-aware algorithm implementation does in fact not depend on the number of parameters or derivation order. The code is written (or rather translated from the primitive double implementation) once, and the number of parameters and derivation orders will be selected at run time later on. The exact same algorithm can be run using only one parameter and first order derivative with respect to this parameter, or using 6 parameters and all partial derivatives up to order 3. The settings are selected at initialization of the first DS instances.

The initialization of the DS can be done in various ways, but the simplest and most used one is based on the definition of the canonical variables, i.e. the ones with respect to which all partial derivatives will be computed. As an example if we need to compute the derivatives with respect to time t and position (x, y, z), first of all this 4 parameters will be defined, telling to the DS their values, that there are in total 4 independent parameters, and the order of derivation it is needed for the program. Then these four canonical variables will be used to feed all the rest of the algorithm and intermediate variables will be created and used and they will propagate derivatives up to the final result. Thanks to this kind of initialization it is possible to choose whatever parameter as a derivative parameter and it is easy to change order of derivation, if needed, without a change in the complex core algorithms. Different studies can be performed, and general propagation libraries can be set up so users can tailor them to their own needs.

Hipparchus also provides different integrators for solving Ordinary Differential Equations. All of them are already able to integrate systems of differentials equations based on DSs. This Java library has already have been thoroughly tested. The differentiation framework is based on the doubly recursive multivariate method[1] for the computation of DS, with an additional compilation step that folds the computation rules found through recursion into extremely fast single pass iterative rules using indirection tables.

3 Orekit

Orekit [5] is an open source Java library of tools for the orbital mechanics initiated by Communication and Systems and developed by a community including external committers. Orekit is intended to be integrated into higher level projects, **Fig.** 1, but it also proposes a set of concepts turnkey to respond to more advanced needs.

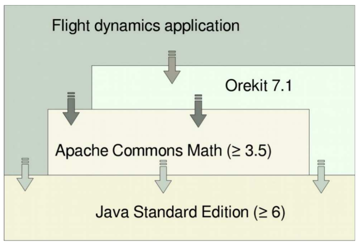
Flight dynamics application
Orekit 7.1
Apache Commons Math (≥ 3.5)
Java Standard Edition (≥ 6)

Fig. 1 Orekit implementation in a flight dynamics application.

As a Java project it is based on classes and is highly modularized(**Fig.**2). It is impractical to apply an analytical way of derivation, because every term inside would add a contribution in the derivation and the derivatives terms would have to be chosen before the utilization. As an instance if the final user wants to see the influence of a moons position error on the evolution of the spacecraft, it would need to add as a derivative term the position of the moon. It could be possible to derivate all the equations with respect of the moons position, but its definitely impossible to take in account all the possible needs of the users and derivate the equations with the respect any variable in the environment.

Choosing a numerical differentiation could be more flexible from the users point of view, but the the computation time would increase tremendously, especially if a lot of derivation parameters are desired. Furthermore if an high precision is required it is necessary to have smaller steps which are hard to choose because it could be a factor 1E7 1E9 between the different values. Talking about val-

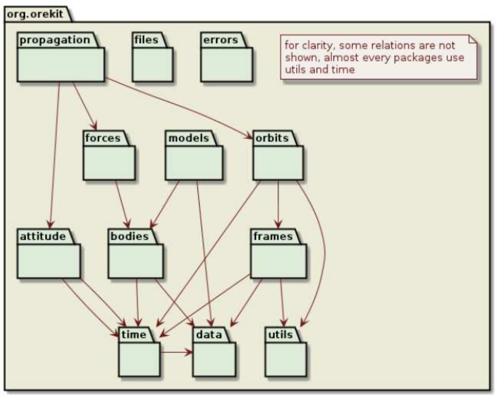
org.orekit
propagation
files
errors
for clarity, some relations are not shown, almost every packages use utils and time
forces
models
orbits
attitude
bodies
frames
time
data
utils

Fig. 2 Orekit main classes and structure.

ues, the limit of the validity domain also adds problems, because the numerical calculations of the derivatives around the boundaries would be one-sided which would strongly decrease the precision.

The main advantages of the automatic differentiation is in the between of the two precedent ways. It has the precision of an analytical method, being of course slower in the computations, and the adaptability of the numerical, without problems on the boundaries, and the different scale of its values are irrelevant. Thanks to this approach it is possible to choose any value as a derivative term and to derivate to whatever order is needed, keeping in mind that the growth in computational time is a binomial coefficient taking in account the number of parameters and the order of derivation [2].

The translation from the primitive double number-based Orekit to the DS-based Orekit has covered all the different types of orbits representation, some of the attitudes, some of the force models, all the analytical propagators and the numerical propagator.

The force models for the numerical propagator already translated include the main body gravity field, with all the tesseral and zonal terms, the isotropic drag, the third body attraction and the solar radiation pressure. If needed also all the other force models on Orekit may be translated, but in order to ease the validation of the global project it has been chosen to translate only in case of need. The remaining force models will be translated in the future (solid tides, ocean tides, relativity, non-isotropic non-conservative forces,).

The correctness of this work has been proven comparing the propagation of a sphere around the initial point [x0+dx; y0+dy; z0+dz] in the real number based Orekit to the Taylors expansion of (dx,dy,dz) at the order N around the propagated point (xt,yt,zt)DS = fDS(x0,y0,z0) of the DS based Orekit. As expected at the order 1 the DS approximation adds an important error, because its equivalent to the propagation of the state error covariance matrix using the state transition matrix [3], which is linear (fig.1). But increasing the order of the Taylor approximation, the curve takes the form of the orbit and the error between the DS propagation and the real one tends to zero.

One of the feature of the DS is that it is possible to choose any free parameter, and study their effect on the totality of what Orekit offers. So the results may be offered in any reference frame (terrestrial, inertial, etc.), or even in representation of these frames, as orbital elements, Cartesian coordinates or geodetic points.

4 Test

Orekit is been operational since 2002 and it's been already deeply tested and proved. To test the improvement and the correctness of the additional features we gave to Orekit. First of all we did the same tests, adapted to the new environment, just using a DS with 0 derivatives and 0 parameters (it means it's just a real number). It has been done to prove the correctness of the algorithms, and also compare the efficacity of the DS treated as a Real number, which resulted being 4 times slower, due to the optimization added.

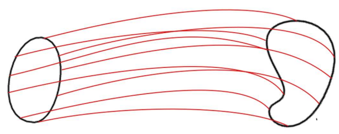
Fig. 3 Sketch to illustrate a Montecarlo simulation .

Secondly we did tests for the propagation of the uncertainty comparing the results from the original Orekit with the DS-based one. First we compared the error evolution calculated using the state transition matrix with the equivalent in the DS system, which is achieved doing the Taylor's expansion at the first order for the DS. After that, to compare higher order approximation, we compared the evolution of a "cloud of points" around the initial point with the same "cloud" evolved from the DS-initial point.

Starting from a point X we evolved X and then applied the transformation for dx compared this result with the original Orekit propagating $\mathbf{X}+\mathbf{dx}$ from the starting point. It resulted that starting from a 4-th order we can accomplish for an error in the order of the km a relative error of 10e-8. The relative error has been calculated normalizing with the initial error.

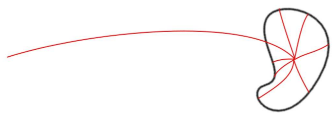
Fig. 4 Sketch to illustrate a DS Montecarlo simulation .

5 Applications

Once the DS had been implemented and tested in Orekit, it had its firsts applications. It has been used in an internal project of C-S to validate the orbit restitution of Orekit. The goal of this application was to see how the error on some of the orbital elements would evolve in 48h. The derivative parameters were the semi-axis (a), the inclination (i) and the longitude of ascending node (Ω) and they were used to evaluate the long track and cross track standard deviation of the position after 48h (**Fig.** 5) has been calculated using an error of (da, di, dΩ) on the initial orbital elements. In order to do so a Montecarlo Simulation, as Armellin et

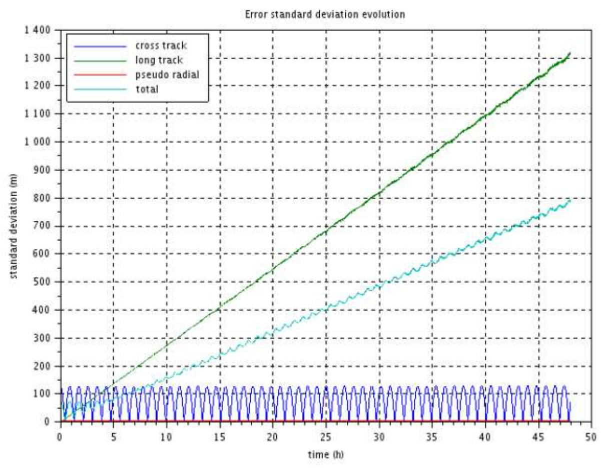

Fig. 5 Evolution of an initial error on orbital elements displayed on the long track and cross track standard deviation.

al. [1], applied on the final propagated vector has been used. Thanks to the Taylors expansion of the DS it has been possible to use a pool of 100k samples of da, di, d around the initial value with a fixed standard deviation for each of the independent parameters.

6 Performances

To show the performances of the DS propagation we chose to compare it with the propagation time of a Real Based propagation to show how the DS propagation slows the procedure. The table 1 is showing that in case of a MC simulation its worth using the DS propagation if the number of samples needed for the Montecarlo simulation, is above the number wrote in the table. Meaning that if it's needed a number of samples above 4 thousands, its worth using 6 parameters at the 6th order, because doing 4thousand times the real-based Orekit is going to take more time than the DS 6-6 Orekit with at the end the Taylors sampling.

Table 1 Performances of the DS with respect to the time of a single propagation on the real-based Orekit. $\tau = 0.043s$

par/ord	1	2	3	4	5	6
1	9.53	10.46	10.69	11.86	12.30	12.79
2	9.76	10.93	11.16	16.51	20.46	22.09
3	10.46	14.65	25.81	55.81	160.5	296.7
4	10.69	25.81	44.18	103.2	298.8	648.3
5	12.32	49.53	87.21	205.5	786.2	2474
6	12.79	68.13	116.2	435.5	1335	3737

References

1. Armellin, R. , Di Lizia, P. , Bernelli-Zazzera, F., Berz, M. *Asteroid close encounters characterization using differential algebra: the case of Apophis*, Springer Science+Business Media (2010)
2. Kalman, D. *Doubly Recursive Multivariate Automatic Differentiation*. American University Washington, D.C. (2002)
3. Tapley, B. D., Schutz, B.E., Born, G.H. *Statistical Orbit Determination*, pg. 405 (2004)
4. Hipparchus https://hipparchus.org/
5. Orekit https://www.orekit.org/