DAST: Nonlinear Uncertainty Propagation using Differential Algebra

Hands-on Demo Session

22nd September 2015

DINAMICA srl
Registered Office:
Piazza della Repubblica, 10 - 20121 - Milano (Italy)
Operational Headquarters:
Via Morghen, 13 - 20156 - Milano (Italy)
Phone +39 02 8342 2930
Fax +39 02 3206 6679
e-mail: dinamica@dinamicatech.com
website: www.dinamicatech.com

Outline

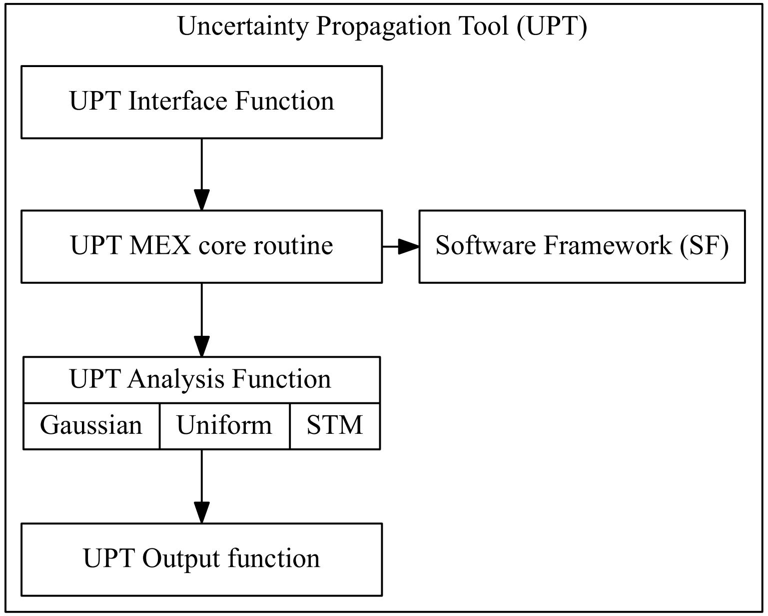
- **Uncertainty Propagation Tool (UPT)**
  - General Architecture
  - Matlab Routine
  - Hands-on session
    - Interplanetary Satellite
- **DA Computational Engine (DACE)**
  - Overview
  - General Architecture
  - Hands-on session
    - Single Variable Functions
    - Multivariable Functions

Uncertainty Propagation Tool

General Architecture

The purpose of the UPT is to allow users to perform uncertainty propagations, based on Taylor differential algebra, directly within MATLAB.

- **UPT Architecture Design**
  - **UPT Interface Function**: interface between Matlab and UPT.
  - **UPT MEX Core Routine**: to set up the DA environment and perform DA propagation (interface with SF routines)
  - **UPT Analysis Function**: to perform the required analyses on the results.
  - **UPT Output Function**: to easily handle the results.



Uncertainty Propagation Tool

Matlab Routine

1 UPTmodel
Matlab function for dynamical model definition

The user must provide a Matlab structure (model structure) containing all information for the setup of the dynamical model

```
model = UPTmodel('param1',value1,'param2', value2,...);
```

2 UPTmethod
Matlab function for propagation method definition

The user must provide a Matlab structure (method structure) containing all information for the setup of the uncertainty propagation method

```
method = UPTmethod('param1',value1,'param2',value2, ...);
```

3 UPTrun
MEX file to perform DA propagations

Once the method and model structures are defined, the user can start the simulation using the routine UPTrun

```
[UPToutput,UPTinput] = UPTrun('Model',model,'Method',method);
```

4 UPTeval
Matlab function to be used for additional evaluations of the final DA map

The user must provide the information on the covariance (or state interval), the sample distribution and number of samples

```
[xf_distr,x0_distr,p0_distr] = UPTeval(UPToutput,...
                     'Distribution',nsamples);
```

Uncertainty Propagation Tool

Matlab Routine

1 — UPTmodel
Matlab function for dynamical model definition

The user must provide a Matlab structure (model structure) containing all information for the setup of the dynamical model

```
model = UPTmodel('param1',value1,'param2', value2,...);
```

2 — UPTmethod
Matlab function for propagation method definition

The user must provide a Matlab structure (method structure) containing all information for the setup of the uncertainty propagation method

```
method = UPTmethod('param1',value1,'param2',value2, ...);
```

3 — UPTrun
MEX file to perform DA propagations

Once the method and model structures are defined, the user can start the simulation using the routine UPTrun

```
[UPToutput,UPTinput] = UPTrun('Model',model,'Method',method);
```

4 — UPTeval
Matlab function to be used for additional evaluations of the final DA map

The user must provide the information on the covariance (or state interval), the sample distribution and number of samples

```
[xf_distr,x0_distr,p0_distr] = UPTeval(UPToutput,...
                        'Distribution',nsamples);
```

Uncertainty Propagation Tool

Matlab Routine

Let's see how it works with a simple example…

1 — UPTmodel
Matlab function for dynamical model definition

The user must provide a Matlab structure (model structure) containing all information for the setup of the dynamical model

```
model = UPTmodel('param1',value1,'param2', value2,...);
```

2 — UPTmethod
Matlab function for propagation method definition

The user must provide a Matlab structure (method structure) containing all information for the setup of the uncertainty propagation method

```
model = UPTmethod('param1',value1,'param2',value2, ...);
```

3 — UPTrun
MEX file to perform DA propagations

Once the method and model structures are defined, the user can start the simulation using the routine UPTrun

```
[UPToutput,UPTinput] = UPTrun('Model',model,'Method',method);
```

4 — UPTeval
Matlab function to be used for additional evaluations of the final DA map

The user must provide the information on the covariance (or state interval), the sample distribution and number of samples

```
[xf_distr,x0_distr,p0_distr] = UPTeval(UPToutput,...
                    'Distribution',nsamples);
```

Let us consider an interplanetary satellite. Given an uncertainty on the initial state vector, the UPT serves the purpose of determining the statistics at final instant time, t_f

Initial State & Simulation Epochs

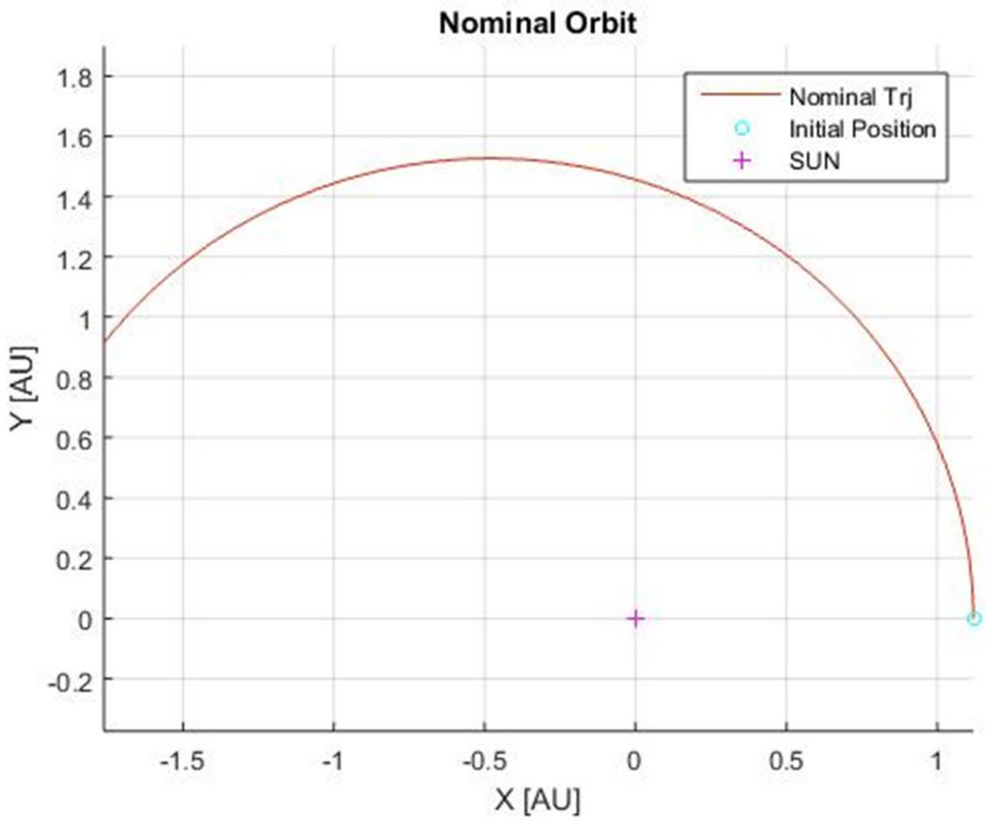
Orbital Parameter	
Semi-major axis [AU]	1.6
Inclination [deg]	0
RAAN [deg]	0
Argument of perigee [deg]	0
Eccentricity	0.3
True anomaly [deg]	0

$$t_0 =' 2009-06-17T00:00:00' \\ t_f =' 2010-03-17T00:00:00'$$

Cartesian State	
X [AU]	1.12
Y [AU]	0
Z [AU]	0
Vx [AU/day]	0
Vy [AU/day]	0.018532930835363
Vz [AU/day]	0

$$et_0 = 2.984688661844962e+08\ sec \\ et_f = 3.220560661855782e+08\ \text{sec}$$

Two-Body Dynamical Model



$$et_0 = 2.984688661844962e + 08\ sec$$
$$et_f = 3.220560661855782e + 08\ sec$$

Cartesian State

X [AU]	1.12
Y [AU]	0
Z [AU]	0
Vx [AU/day]	0
Vy [AU/day]	0.018532930835363
Vz [AU/day]	0

Uncertainties on initial state

Uncertainty Propagation Tool

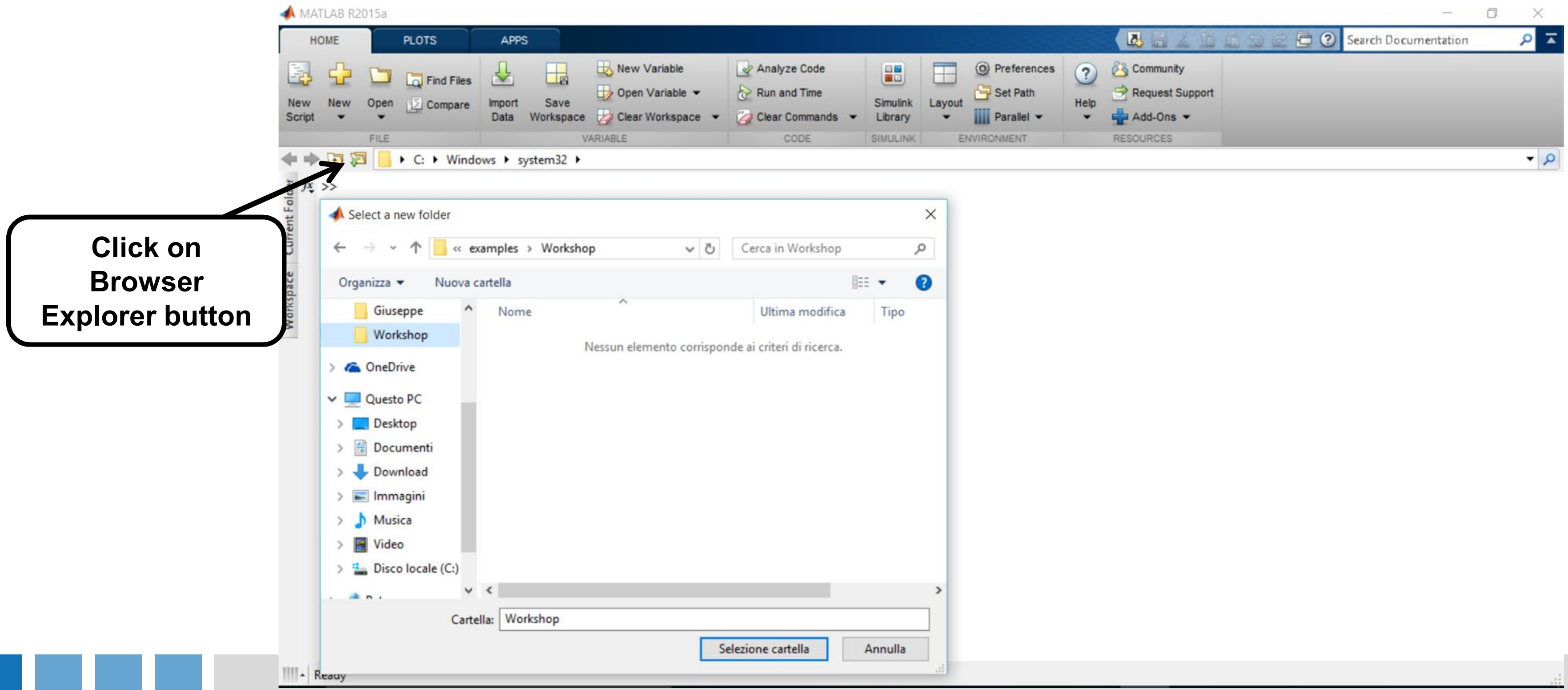
Exercises Summary

- **What will we do??**
  - Given the uncertainties on initial state, we compute the statistics at t_f using the DA-based Monte Carlo Simulation method
    - EX. 1-2: Gaussian Initial Distribution / Two-body Model / Order 1
    - EX. 3: Gaussian Initial Distribution / Two-body Model / Order 3
    - EX. 5: Gaussian Initial Distribution / N-body Model / Order 3
    - EX. 6: Uniform Initial Distribution / N-body Model / Order 3
  - Given the uncertainties on initial state, we compute the statistics at t_f using the Linearized Dynamics method
    - EX. 4: Gaussian Initial Distribution / Two-body Model
  - Given the uncertainties on initial state, we determine the upper and lower bounders of final uncertainties using Polynomial Bounder method
    - EX. 7: Uniform Initial Distribution / N-body Model / Order 3

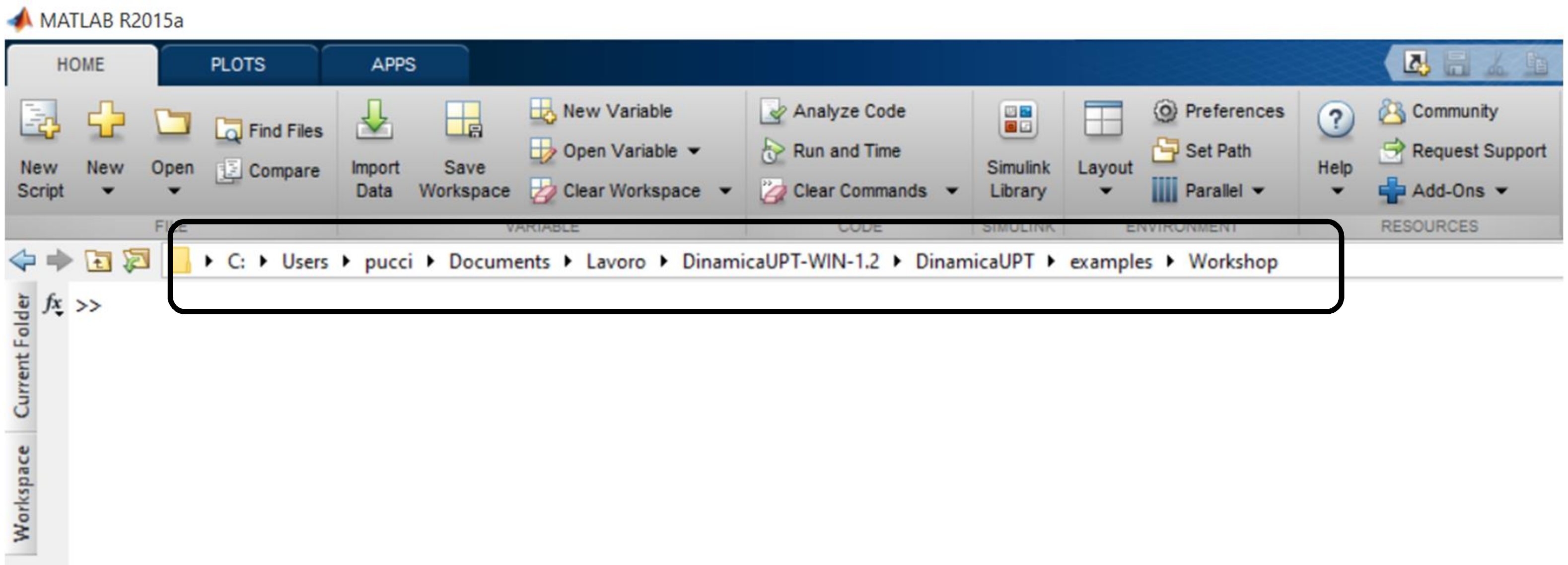
Uncertainty Propagation Tool

Before Starting...

- **Matlab**
  - Open Matlab
  - Change the current folder to Workshop in the address field of the current folder toolbar of Matlab



- **Matlab**
  - Open Matlab
  - Change the current folder to Workshop in the address field of the current folder toolbar of Matlab



Matlab

- Add the lib, matlab, examples, and Workshop folders (included in the *DinamicaUPT*) to Matlab path.

```
% Add needed path for UPT
DinamicaUPT_folder = pwd;

addpath(fullfile(DinamicaUPT_folder,'matlab'));
addpath(fullfile(DinamicaUPT_folder,'lib'));
addpath(fullfile(DinamicaUPT_folder,'examples'));
addpath(fullfile(DinamicaUPT_folder,'examples','Workshop'));

warning off
```

- Change the current folder to run folder in the address field of the current folder toolbar of Matlab

```
cd (fullfile(pwd,'run'));
```

Run the UPTpath.m

Initial State & Simulation Epochs

- Run the InitialState.m to set the initial nominal conditions and simulation interval or type the following script in Matlab command window

```
% Initial State
state = [1.1200, 0, 0, 0, 0.018532930835363, 0];

% Initial epoch: t0 = '2009-06-17T00:00:00';
et0 = 2.984688661844962e+08;
% Final epoch: tf = '2010-03-17T00:00:00';
etf = 3.220560661855782e+08;
dt_sec = etf - et0;
```

Cartesian State	
X [AU]	1.12
Y [AU]	0
Z [AU]	0
Vx [AU/day]	0
Vy [AU/day]	0.018532930835363
Vz [AU/day]	0

$$et_0 = 2.984688661844962e+08\ sec$$
$$et_f = 3.220560661855782e+08\ \text{sec}$$

- **EX. 1:** Perform a DA-based Monte Carlo Simulation assuming an expansion order equal to 1 (referred to as DAMC-G1). A Gaussian distribution is considered for each initial state (the covariance matrix Cov must be defined). The uncertainties are propagate through the two-body dynamics.

$$Cov = diag\left(\left[\sigma_{xx}, \sigma_{yy}, \sigma_{zz}, \sigma_{\dot{x}\dot{x}}, \sigma_{\dot{y}\dot{y}}, \sigma_{\dot{z}\dot{z}}\right]\right)$$
$$\sigma_{xx} = \sigma_{yy} = \sigma_{zz} = 1e-04$$
$$\sigma_{\dot{x}\dot{x}} = \sigma_{\dot{y}\dot{y}} = \sigma_{\dot{z}\dot{z}} = 1e-10$$

```
% Covariance Matrix
Cov = diag([1e-4*ones(1,3), 1e-10*ones(1,3)]);
```

Uncertainty Propagation Tool

Example: Interplanetary Satellite

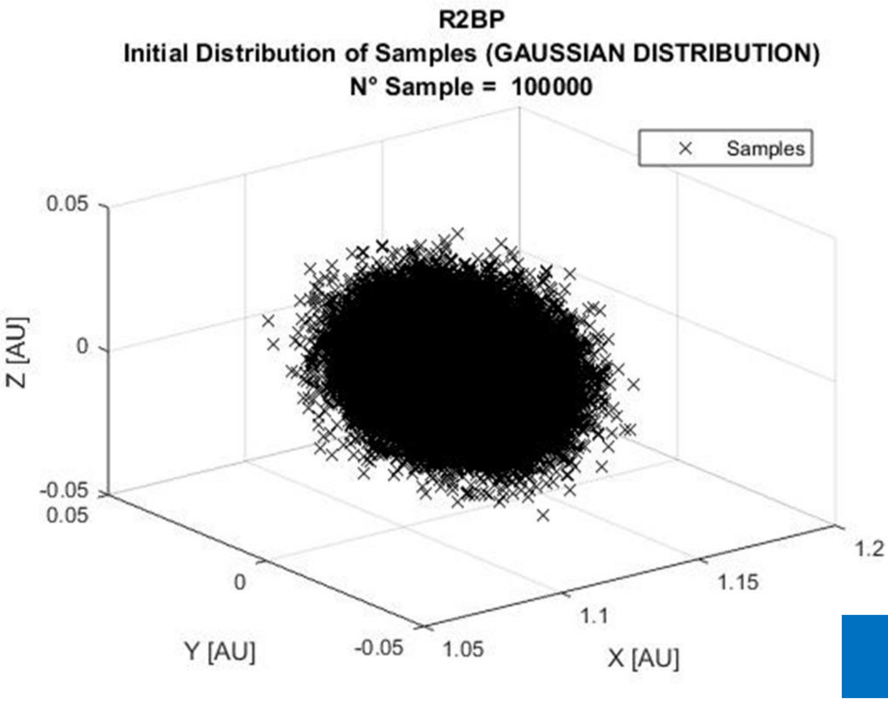
- **EX. 1:** Perform a DA-based Monte Carlo Simulation assuming an expansion order equal to 1 (referred to as DAMC-G1). A Gaussian distribution is considered for each initial state (the covariance matrix Cov must be defined). The uncertainties are propagated through the two-body dynamics.

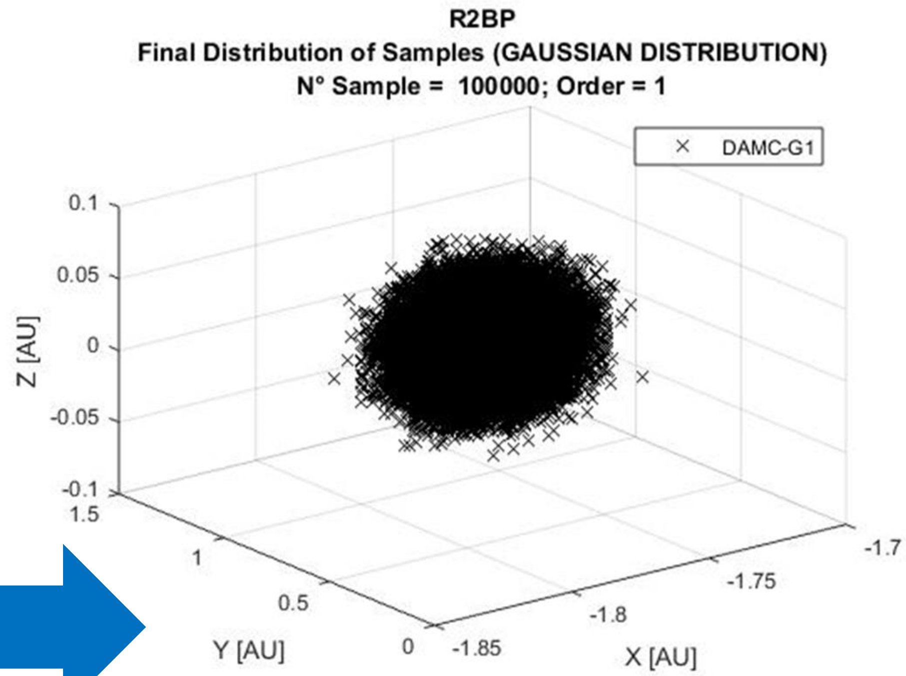
```
% Define the dynamical model by UPTmodel routine
model_R2BP = UPTmodel('Model', 'R2BP', 'MainAttractor', 'SUN', 'InitialState', state, ...
                 'Coordinate', 'RECTANGULAR', 'Frame', 'ECLIPJ2000','FrameCenter', 'SUN', ...
                 'InitialEpoch', t0, 'FinalEpoch', tf, 'LengthUnits', 'AU', ...
                 'TimeUnits', 'DAY', 'AngleUnits', 'RAD', 'Tolerance',1e-12);
% Define Covariance Matrix
Cov   = diag([1e-4*ones(1,3),1e-10*ones(1,3)]);
% Define the uncertainty propagation method by UPTmethod routine
a_x       = [1 1 1 1 1 1]; nsample = 1e5; order = 1;
method_DAMCG1    = UPTmethod('Method', 'DAMC', 'Distribution','GAUSSIAN',...
                    'CovarianceMatrix', Cov, 'UncertainStates', a_x,'Samples', nsamples,...
                     'Order', order);
% Propagate the initial uncertainties by UPTrun routine
[UPToutput_DAMCG1, UPTinput_DAMCG1] = UPTrun( 'Model', model_R2BP, 'Method', method_DAMCG1);
x0_distr_DAMCG   = UPToutput_DAMCG1.x0_distr;
xf_distr_DAMCG1 = UPToutput_DAMCG1.xf_distr;
LB0_DAMCG   = min(x0_distr_DAMCG,[],2);
UB0_DAMCG   = max(x0_distr_DAMCG,[],2);

COV_DAMCG1   = UPToutput_DAMCG1.finalcov;
mean_DAMCG1 = UPToutput_DAMCG1.finalmean;
```

Uncertainty Propagation Tool

Example: Interplanetary Satellite

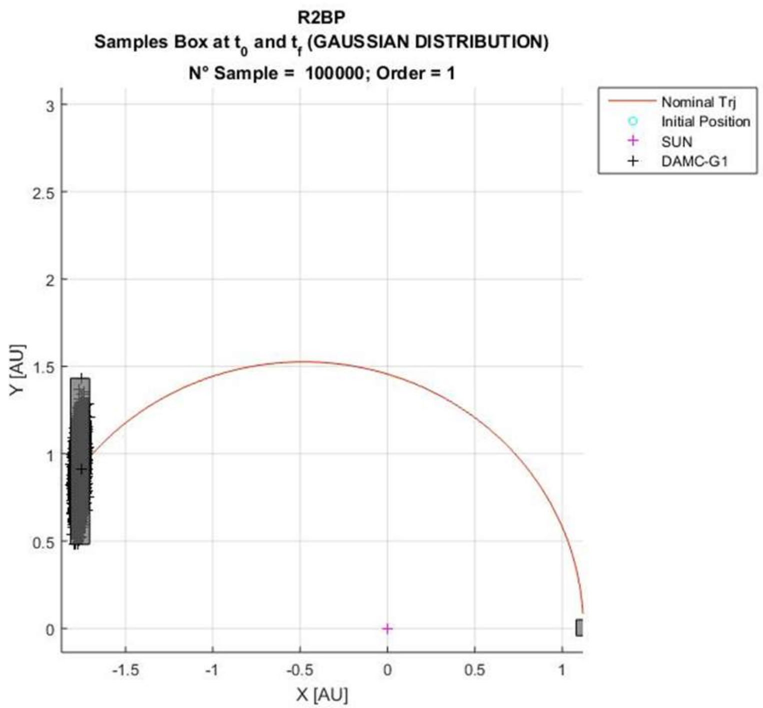
- **EX. 1:** Perform a DA-based Monte Carlo Simulation assuming an expansion order equal to 1 (referred to as DAMC-G1). A Gaussian distribution is considered for each initial state (the covariance matrix Cov must be defined). The uncertainties are propagated through the two-body dynamics.

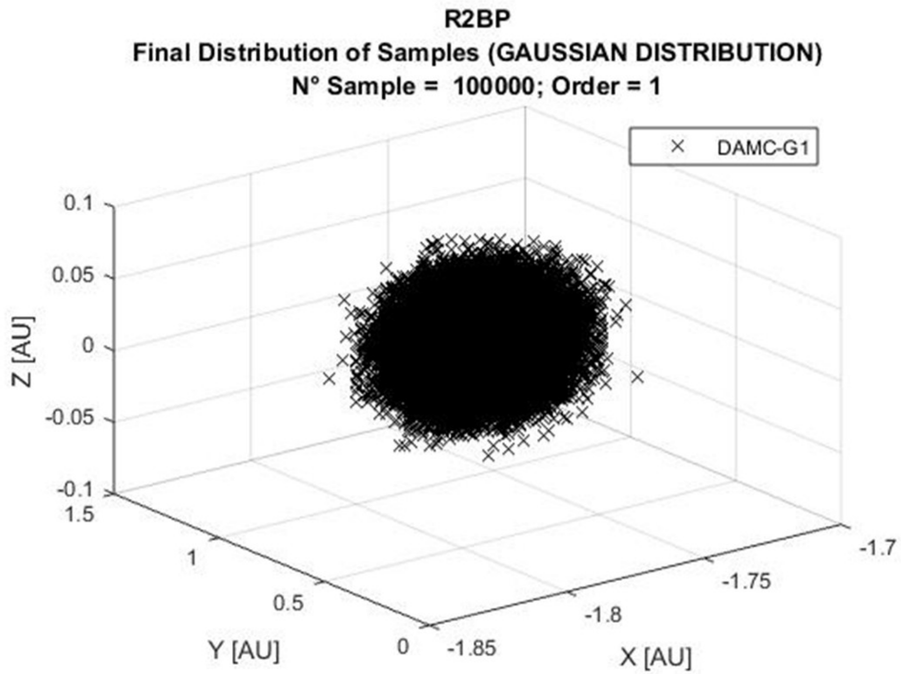




$$\tau_{DAMC-G1} = 0.186\ sec$$

- **EX. 1:** Perform a DA-based Monte Carlo Simulation assuming an expansion order equal to 1 (referred to as DAMC-G1). A Gaussian distribution is considered for each initial state (the covariance matrix Cov must be defined). The uncertainties are propagated through the two-body dynamics.





- **EX. 2:** Compare DAMC-G1 results with Standard Monte Carlo (referred to as sMC) ones. The same initial Gaussian distribution of EX. 1 is used. The uncertainties are propagated through the two-body dynamics.

Uncertainty Propagation Tool

Example: Interplanetary Satellite

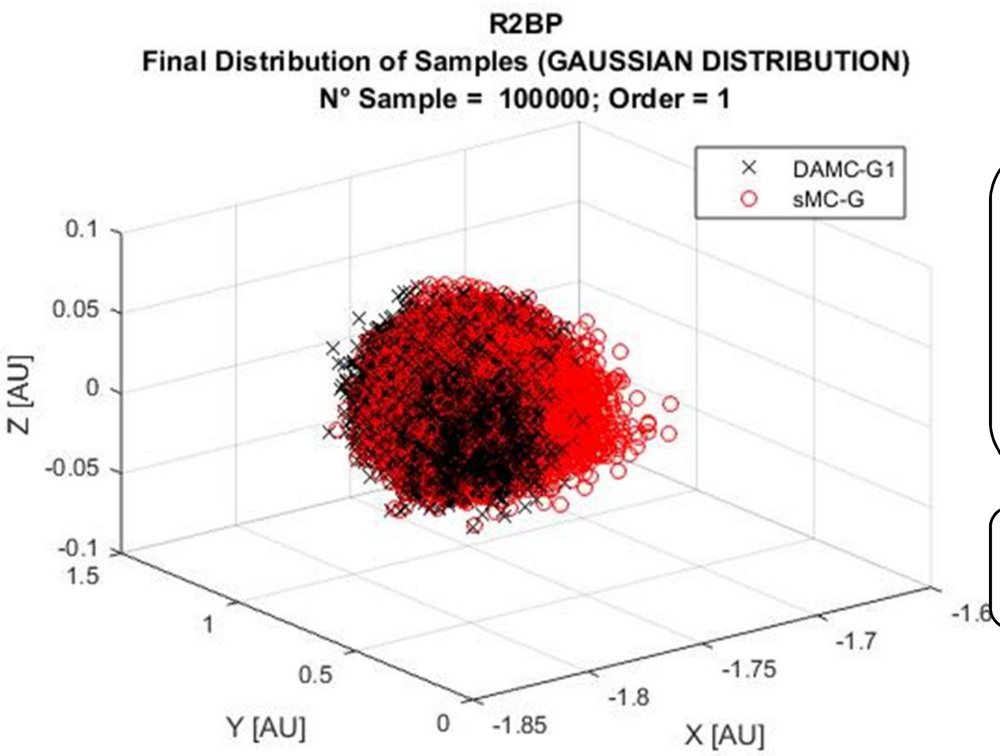
- **EX. 2:** Compare DAMC-G1 results with Standard Monte Carlo (referred to as sMC) ones. The same initial Gaussian distribution of EX. 1 is used. The uncertainties are propagated through the two-body dynamics.

```
% Standard Monte Carlo Simulation
xf_sMC = zeros(6,size(x0_distr_DAMCG,2));
tic
for i = 1:size(x0_distr_DAMCG,2)
    % Solve the Kepler Equation
    [r, v] = keplerUniversal(x0_distr_DAMCG(1:3,i)*AU, x0_distr_DAMCG(4:6,i)*AU/day,dt_sec,mu);
    xf_sMC(1:3,i) = r/AU;
    xf_sMC (4:6,i) = v*day/AU;
end
computational_time.sMC = toc;
COV_sMC  = cov(xf_sMC');
mean_sMC = mean(xf_sMC,2);
```

Uncertainty Propagation Tool

Example: Interplanetary Satellite

- **EX. 2:** Compare DAMC-G1 results with Standard Monte Carlo (referred to as sMC) ones. The same initial Gaussian distribution of EX. 1 is used. The uncertainties are propagated through the two-body dynamics.



$$\varepsilon_{r,max} = max(\|\boldsymbol{r}_{DAMC-} \quad - \boldsymbol{r}_{sMC-G}\|) = 0.065747\ [AU]$$

$$\varepsilon_{v,max} = ma\ \ (\|\boldsymbol{v}_{DAMC-} \quad - \boldsymbol{v}_{sMC} \quad \|) = 7.581394e-04[\frac{AU}{day}]$$

$$\tau_{DAMC-G1} = 0.186\ [sec]$$
$$\tau_{sMC-G} = 38.11\ [sec]$$

- **EX. 3:** Perform a DAMC-G3 simulation and compare with sMC. The same initial Gaussian distribution of EX. 1 is used. The uncertainties are propagated through the two-body dynamics.

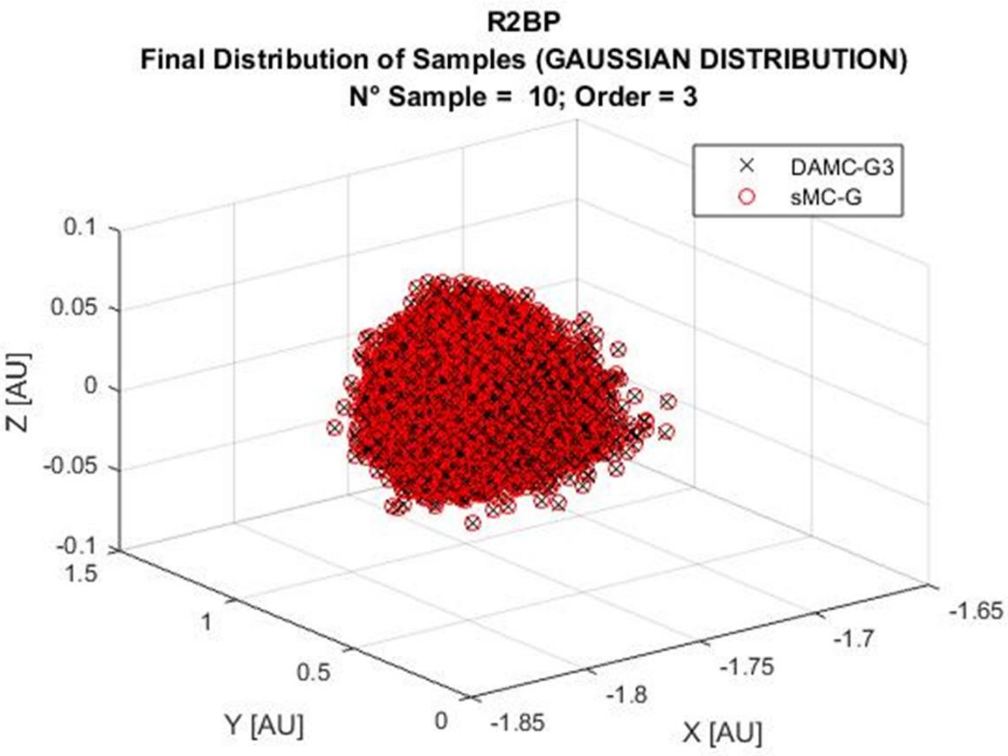
Uncertainty Propagation Tool

Example: Interplanetary Satellite

- **EX. 3:** Perform a DAMC-G3 simulation and compare with sMC. The same initial Gaussian distribution of EX. 1 is used. The uncertainties are propagated through the two-body dynamics.

```
% Define the uncertainty propagation method by UPTmethod routine
a_x       = [1 1 1 1 1 1]; nsample = 1e5; order = 3;
method_DAMCG3    = UPTmethod('Method', 'DAMC', 'Distribution','GAUSSIAN',...
                    'CovarianceMatrix', Cov, 'UncertainStates', a_x,'Samples', 1e1,...
                     'Order', order);

% Propagate the initial uncertainties by UPTrun routine
tic;
[UPToutput_DAMCG3, UPTinput_DAMCG3] = UPTrun( 'Model', model_R2BP, 'Method', method_DAMCG3);
[ xf_distr_DAMCG3 ] = UPTeval( UPToutput_DAMCG3, x0_distr_DAMCG, nsample );
computationalime.DAMCG3 = toc;

COV_DAMCG3  = cov(xf_distr_DAMCG3');
mean_DAMCG3 = mean(xf_distr_DAMCG3,2);
```

- **EX. 3:** Perform a DAMC-G3 simulation and compare with sMC. The same initial Gaussian distribution of EX. 1 is used. The uncertainties are propagated through the two-body dynamics.



$$\varepsilon_{r,max} = max(\|\boldsymbol{r}_{DAMC-} \quad - \boldsymbol{r}_{sMC-G}\|) = 0.001574[AU]$$

$$\varepsilon_{v,max} = max(\|\boldsymbol{v}_{DAMC-G3} - \boldsymbol{v}_{sMC-G}\|) = 2.689792e-05\ [\frac{AU}{day}]$$

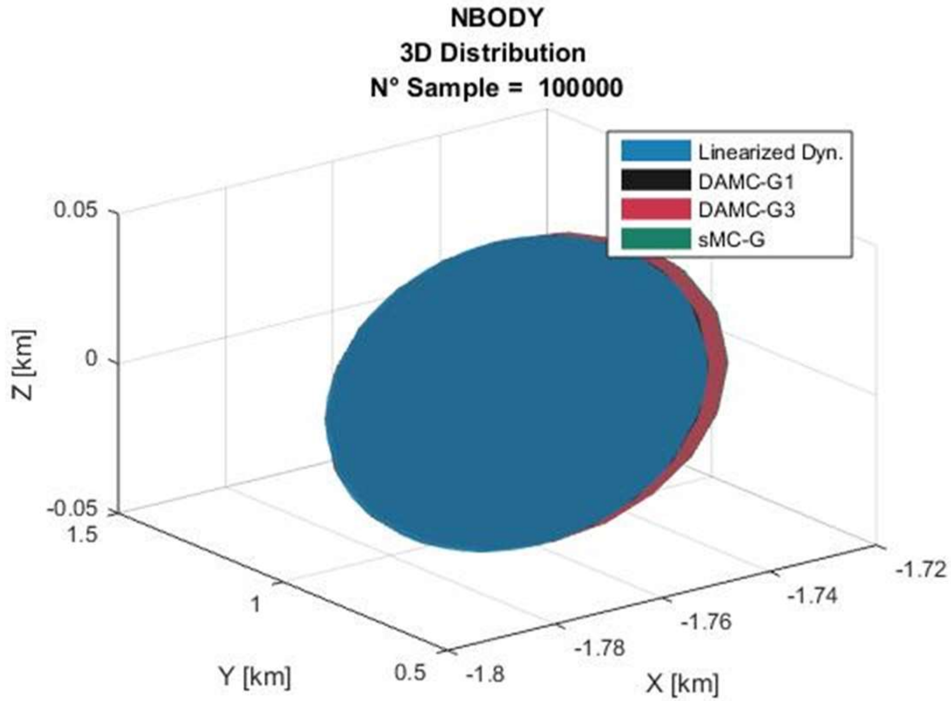
$$\tau_{DAMC-G3} = 0.482\ [sec$$
$$\tau_{sMC-G} = 38.11\ [sec]$$

- **EX. 4:** Compute the final covariance matrix through the Linearized Dynamics method (referred to as LD) and compare the results with those obtained by DAMC-G1, DAMC-G3, and sMC. The same initial Gaussian distribution of EX. 1 is used. The uncertainties are propagated through the two-body dynamics.

- **EX. 4:** Compute the final covariance matrix through the Linearized Dynamics method (referred to as LD) and compare the results with those obtained by DAMC-G1, DAMC-G3, and sMC. The same initial Gaussian distribution of EX. 1 is used. The uncertainties are propagated through the two-body dynamics.

```
% Define the uncertainty propagation method by UPTmethod routine
a_x       = [1 1 1 1 1 1];
method_LD = UPTmethod('Method','LINEARIZED_DYNAMICS', 'UncertainStates', a_x, ...
                       'CovarianceMatrix', Cov);

% Propagate the initial uncertainties by UPTrun routine
tic;
[[UPToutput_LD, UPTinput_LD] = UPTrun( 'Model', model_R2BP, 'Method', method_LD );
computationalime.LD = toc;

COV_LD    = UPToutput_LD.finalcov;      % Extract the covariance matrix
mean_LD   = UPToutput_LD.finalmean;
```

- **EX. 4:** Compute the final covariance matrix through the Linearized Dynamics method (referred to as LD) and compare the results with those obtained by DAMC-G1, DAMC-G3, and sMC. The same initial Gaussian distribution of EX. 1 is used. The uncertainties are propagated through the two-body dynamics.



- **EX. 5:** Change the dynamical model for the uncertainties propagation from 2BP to N-body. A new Gaussian distribution is generated with the same covariance of EX. 1. Perform a DAMC-G3 simulation.

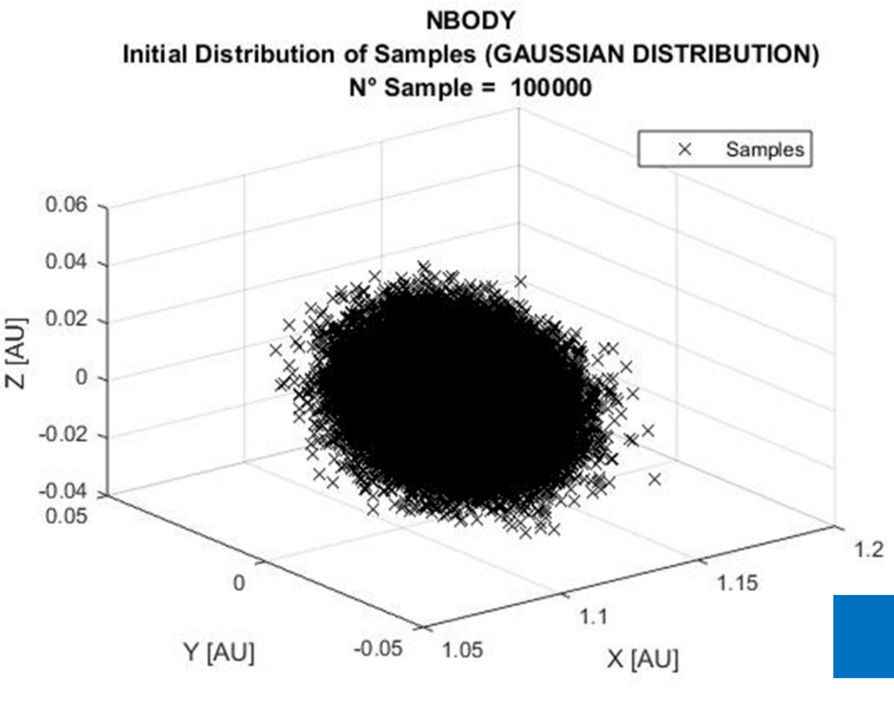
Uncertainty Propagation Tool

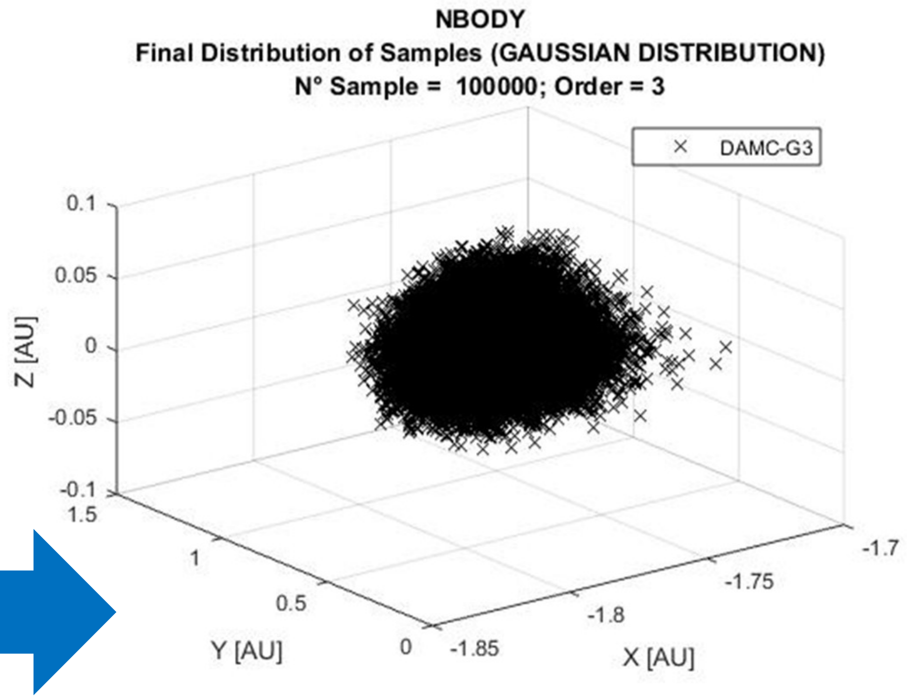
Example: Interplanetary Satellite

- **EX. 5:** Change the dynamical model for the uncertainties propagation from 2BP to N-body. A new Gaussian distribution is generated with the same covariance of EX. 1. Perform a DAMC-G3 simulation.

```
% Define the dynamical model by UPTmodel routine
model_NBP = UPTmodel('Model', 'NBODY', 'MainAttractor', 'SUN', 'InitialState', state, ...
                'Coordinate', 'RECTANGULAR', 'Frame', 'ECLIPJ2000', 'FrameCenter', 'SUN', ...
                'InitialEpoch', t0, 'FinalEpoch', tf, 'LengthUnits', 'AU', ...
                'TimeUnits', 'DAY', 'AngleUnits', 'RAD', 'Tolerance',1e-12);

% Define the uncertainty propagation method by UPTmethod routine
a_x      = [1 1 1 1 1 1]; nsample = 1e5; order = 3;
method_DAMCG3 = UPTmethod('Method', 'DAMC', 'Distribution','GAUSSIAN',...
                 'CovarianceMatrix', Cov ,'UncertainStates', a_x,...
                 'Samples', nsamples, 'Order', order);

% Propagate the initial uncertainties by UPTrun routine
UPToutput_DAMCG3  = UPTrun( 'Model', model_NBP, 'Method', method_DAMCG3);

xf_distr_DAMCG3 = UPToutput_DAMCG3.xf_distr;
x0_distr_DAMCG  = UPToutput_DAMCG3.x0_distr;
LB0_DAMCG   = min(x0_distr_DAMCG,[],2);
UB0_DAMCG   = max(x0_distr_DAMCG,[],2);

COV_DAMCG3   = UPToutput_DAMCG3.finalcov;
mean_DAMCG3 = UPToutput_DAMCG3.finalmean;
```

- **EX. 5:** Change the dynamical model for the uncertainties propagation from 2BP to N-body. A new Gaussian distribution is generated with the same covariance of EX. 1. Perform a DAMC-G3 simulation.





- **EX. 6:** Generate an uniform initial distribution of samples (through standard Matlab routine) and propagate it through the N-body dynamics. The interval for each uncertain state is determined computing the max and min limits of distribution defined in EX. 5. Perform a DAMC-U3 simulation.

Uncertainty Propagation Tool

Example: Interplanetary Satellite

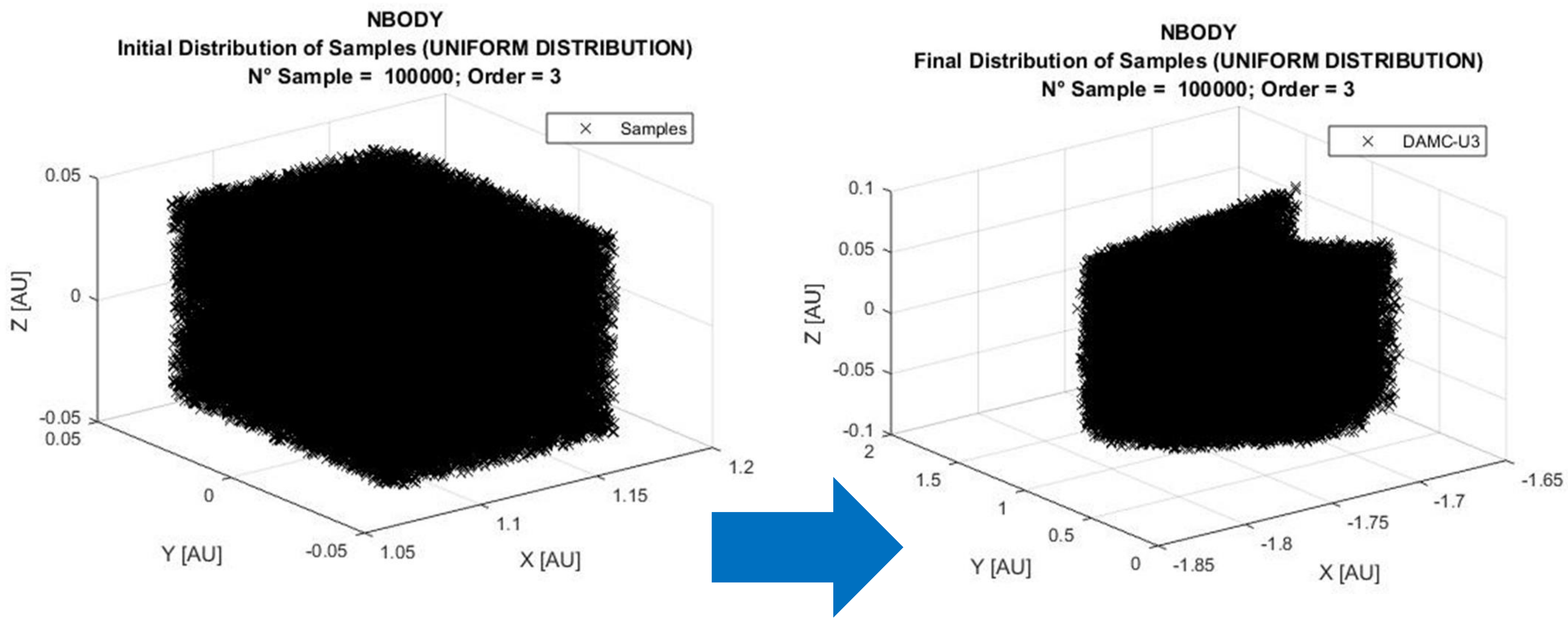
- **EX. 6:** Generate an uniform initial distribution of samples (through standard Matlab routine) and propagate it through the N-body dynamics. The interval for each uncertain state is determined computing the max and min limits of distribution defined in EX. 5. Perform a DAMC-U3 simulation.

```
% Generate a new uniform distribution using standard Matlab routine
nsamples = method_DAMCG3.samples;
samples = unifrnd(-1,1,nsamples,6);
sigma_x = abs(UB0_DAMCG-LB0_DAMCG)/2;
for i = 1:size(sigma_x,1)
    x0_distr_DAMCU(i,:) = state(i) + samples(:,i)'*sigma_x(i);
end
% Run the UPTeval routine
tic
[xf_distr_DAMCU3,x0_distr_DAMCU]    = UPTeval(UPToutput_DAMCG3,x0_distr_DAMCU,nsamples);
computationalTime.DAMCU3 = toc;

LBf_DAMCU3  = min(xf_distr_DAMCU3,[],2);
UBf_DAMCU3  = max(xf_distr_DAMCU3,[],2);
LB0_DAMCU3  = min(x0_distr_DAMCU,[],2);
UB0_DAMCU3  = max(x0_distr_DAMCU,[],2);
```

Uncertainty Propagation Tool

Example: Interplanetary Satellite

- **EX. 6:** Generate an uniform initial distribution of samples (through standard Matlab routine) and propagate it through the N-body dynamics. The interval for each uncertain state is determined computing the max and min limits of distribution defined in EX. 5. Perform a DAMC-U3 simulation.



- **EX. 7:** Compute the final upper and lower bounders (approximation) through the Polynomial Bounder method (referred to as PB) and compare the results with those obtained by DAMC-U3. The same interval for each uncertain state defined in EX. 6 is used. The N-body dynamics is used for PB simulation.

Uncertainty Propagation Tool

Example: Interplanetary Satellite

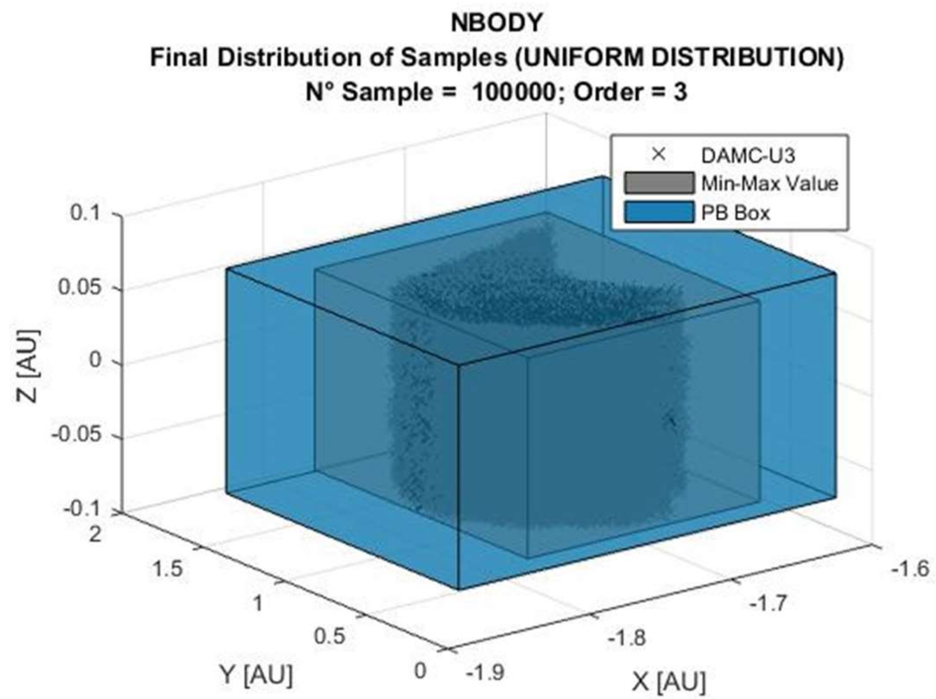
- **EX. 7:** Compute the final upper and lower bounders (approximation) through the Polynomial Bounder method (referred to as PB) and compare the results with those obtained by DAMC-U3. The same interval for each uncertain state defined in EX. 6 is used here. The N-body dynamics is used for PB simulation.

```
% Define the uncertainty propagation method by UPTmethod routine
IntervalState = abs(UB0_DAMCU3'-LB0_DAMCU3')/2;
nsamples = 1e5;              % N° of sample
order    = 3;                % Taylor expansion order = 3
method_PB = UPTmethod('Method', 'POLYNOMIAL_BOUNDER', 'Order', order, ...
                   'UncertainStates', a_x, 'IntervalStates',IntervalState);

% Propagate the initial uncertainties by UPTrun routine
[UPToutput_PB, UPTinput_PB] = UPTrun( 'Model', model_NBP, 'Method', method_PB );

UBf_PB = UPToutput_PB.bounds.ub;
LBf_PB = UPToutput_PB.bounds.lb;
```

Uncertainty Propagation Tool

Example: Interplanetary Satellite

- **EX. 7:** Compute the final upper and lower bounders (approximation) through the Polynomial Bounder method (referred to as PB) and compare the results with those obtained by DAMC-U3. The same interval for each uncertain state defined in EX. 6 is used here. The N-body dynamics is used for PB simulation.



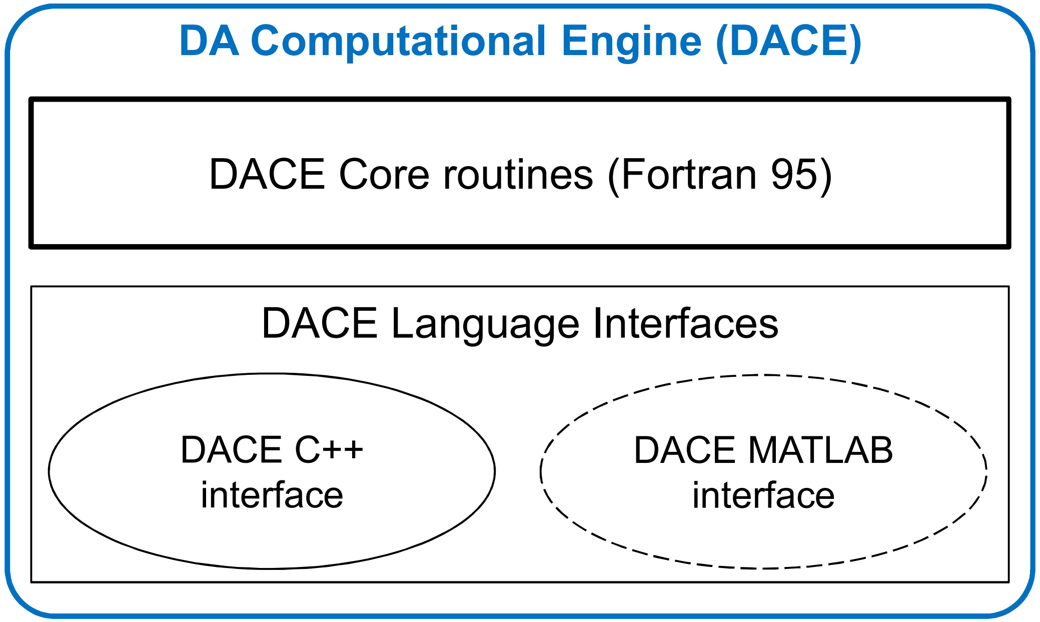
The DA Computational Engine (DACE) is an implementation of the basic DA routines

DACE

- Each DA routine approximates the result of an operation by its Taylor expansion around 0
- After each operation one obtains an approximation, yielding eventually to the Taylor expansion of arbitrarily complex expressions
- The DACE provides a user interface to use the DA routine such that
  1. It allows writing mathematical expressions in typical computer programming way
  2. It allows evaluating them using DA and double precision numbers

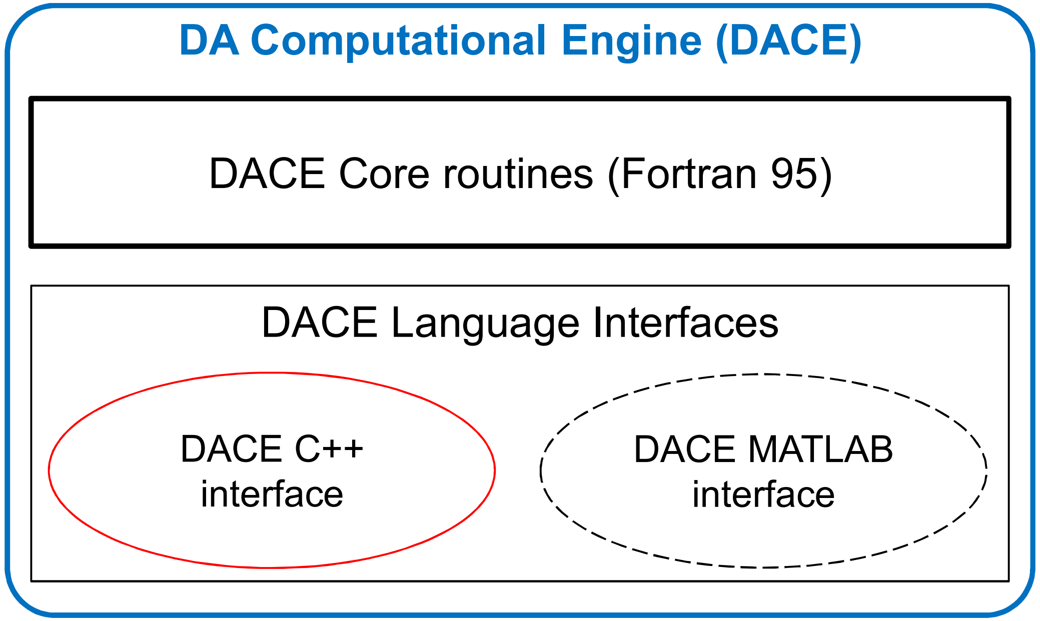
DACE Architecture Design

- DA core routines implemented in Fortran 95
- Powerful C++ interface directly to Fortran 95 routines
- MATLAB interface directly to Fortran 95 routines (beta version)



DACE Architecture Design

- DA core routines implemented in Fortran 95
- Powerful C++ interface directly to Fortran 95 routines
- MATLAB interface directly to Fortran 95 routines (beta version)



DA Comutational Engine

Before Starting...

- **Bootable USB keys**
  - Complete Linux Development environment
  - Dace Library already included
- **KDevelop Project (ESA_WORKSHOP)**
  - Modify CMakeLists.txt to add DACE Library

```
include_directories(.)
find_library(DACE_LIBRARY dace PATHS .)

add_executable(exe1 main1.cpp)
target_link_libraries(exe1 ${DACE_LIBRARY})
```

- **Include the DA header**

```
#include <DA/dace.h>
#include <iostream>
#include <cmath>
#include <fstream>
#include <iomanip>

using namespace std;
using namespace DACE;
```

- **What will we do??**
  - Use the DACE to compute Taylor expansion of following single variable functions
    - EX. 1: $y = sin(x)$ around $(0,0)$
    - EX. 2: $y = sin^2(x) + cos^2(x)$ around $(0,0)$
    - EX. 3: $dy = \frac{d}{dx}(sin(x))$ around $(0,0)$
  - Use the DACE to compute Taylor expansion of following multivariable functions
    - EX. 4: *Sombrero function* around $(0,0)$
    - EX. 5: *Sombrero function* around $(2,3)$
    - EX. 6 - 7: *Gradient of sombrero function* around $(2,3)$

EX. 1: $y = sin(x)$

1. Initialize DACE to perform 20-th order computations

```
DA::init( 20, 1 );
```

2. Initialize x as a DA number

```
DA x = DA(1);
```

3. Compute

```
DA y = sin(x);
```

4. Print to screen

```
cout << "x" << endl << x << endl;
cout << "sin(x)" << endl << sin(x);
```

- **EX. 1:** $y = sin(x)$
  - Compare with analytical Taylor expansion

$$\mathcal{T}_y(x) = \sum_{i=0}^{\infty} a_i x^i = \sum_{j=0}^{\infty} \frac{(-1)^j}{(2j+1)!} x^{2j+1}$$

$$a_0 = a_2 = a_4 = \cdots = 0$$

$$a_1 = 1$$

$$a_3 = -{}^1/_6 = -0.166666\ldots$$

$$a_5 = {}^1/_{20} = 0.0083\ldots$$

$$a_7 = {}^1/_{5040} = 0.00019841269841$$

EX. 2: Verify that $sin^2(x) + cos^2(x) = 1$

1. Initialize DACE to perform 20-th order computations

```
DA::init( 20, 1 );
```

2. Initialize x as a DA number

```
DA x = DA(1);
```

3. Compute

```
DA y1 = sqr(sin(x));
DA y2 = sqr(cos(x));
```

4. Print to screen

```
cout << "sin(x)^2+cos(x)^2 << endl;
Cout << y1 + y2 << endl;
```

EX. 3:

Compute $\mathrm{d}y = \frac{d}{dx}(sin(x))$

1. Initialize DACE to perform 20-th order computations in one variable

```
DA::init( 20, 1 );
```

2. Initialize x as a DA number and compute $sin(x)$

```
DA x = DA(1);
DA y = sin(x);
```

3. Compute $\mathrm{d}y = \frac{d}{dx}(sin(x))$

```
DA dy = y.deriv(1);
```

4. Print to screen

```
cout << "d[sin(x)]/dx" << endl << dy << endl;
cout << "cos(x)" << endl << cos(x) << endl;
```

5. Verify that it is equal to $\cos(x)$ (find the difference and explain 🙂)

- **EX. 3:** Compute $\mathrm{d}y = \frac{d}{dx}(sin(x))$

1. Initialize DACE to perform 20-th order computations in one variable

```
DA::init( 20, 1 );
```

2. Initialize x as a DA number and compute $sin(x)$

```
DA x = DA(1);
DA y = sin(x);
```

3. Compute $\mathrm{d}y = \frac{d}{dx}(sin(x))$

```
DA dy = y.deriv(1);
```

Note that the integral of $sin(x)$ function can be easily computed through the DACE (→ `y.integ(1)`)

4. Print to screen

```
cout << "d[sin(x)]/dx" << endl << dy << endl;
cout << "cos(x)" << endl << cos(x) << endl;
```

5. Verify that it is equal to $\cos(x)$ (find the difference and explain 🙂)

- **EX. 4:** *Sombrero Function*: $z = sin(\sqrt{(x_1^2 + x_2^2)})/\sqrt{(x_1^2 + x_2^2)}$

1. Initialize DACE to perform 10-th order computations in 2 variables

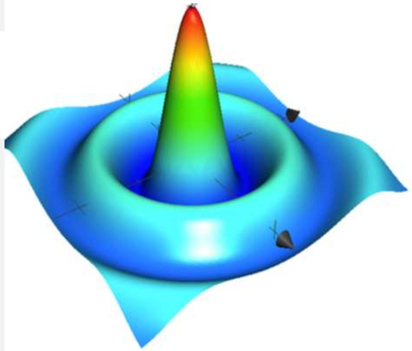
```
DA::init( 10, 2 );
```

2. Initialize x as a two-dimensional vector of DA numbers (Taylor expansion around the point $(0,0)$)

```
AlgebraicVector<DA> x(2);
x[0] = DA(1);
x[1] = DA(2);
```

3. Evaluate *sombrero function*

```
DA z = somb(x);
cout << "Sombrero Function" << endl;
cout << z << endl;
```



DA Comutational Engine

Multivariate Function

- **EX. 5:** *Sombrero Function*: $z = sin(\sqrt{(x_1^2 + x_2^2)})/\sqrt{(x_1^2 + x_2^2)}$

1. Initialize DACE to perform 10-th order computations in 2 variables

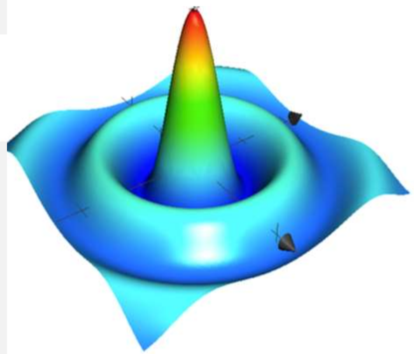
```
DA::init( 10, 2 );
```

2. Initialize x as a two-dimensional vector of DA numbers (Taylor expansion around the point $(2{,}3)$)

```
AlgebraicVector<DA> x(2);
x[0] = 2.0 + DA(1);
x[1] = 3.0 + DA(2);
```

3. Evaluate *sombrero function*

```
DA z = somb(x);
cout << "Sombrero Function" << endl;
cout << z << endl;
```



- **EX. 6:** Gradient of *sombrero function*

1. Initialize DACE to perform 1-st order computations in 2 variables

```
DA::init( 1, 2 );
```

2. Compute the 1-st order Taylor expansion of the sombrero function around the point $(2,3)$ (See EX.5)

3. Compute the *gradient sombrero function* around the point $(2,3)$

```
AlgebraicVector<DA> grad_z(2);
grad_z = z.gradient();
```

4. Verify that the obtained result is equal to the aalytical solution, that is

```
cout << "Grad. of sombrero function" << endl;
cout << grad_z << endl;
```

$$\partial x / \partial y = -0.1184886$$

$$\partial z / \partial y = -0.1777329$$

- **EX. 7:** Gradient of *sombrero function*

1. Initialize DACE to perform 5-th order computations in 2 variables

```
DA::init( 5, 2 );
```

2. Compute the 5-th order Taylor expansion of the sombrero function around the point $(2,3)$ (See EX. 5)

3. Compute the *gradient sombrero function* around the point $(2,3)$

```
AlgebraicVector<DA> grad_z(2);
grad_z = z.gradient();
```

4. Verify that the obtained result is equal to the aalytical solution, that is

```
cout << "Grad. of sombrero function" << endl;
cout << grad_z << endl;
```

DAST: Nonlinear Uncertainty Propagation using Differential Algebra

Hands-on Demo Session

22nd September 2015

DINAMICA srl
Registered Office:
Piazza della Repubblica, 10 - 20121 - Milano (Italy)
Operational Headquarters:
Via Morghen, 13 - 20156 - Milano (Italy)
Phone +39 02 8342 2930
Fax +39 02 3206 6679
e-mail: dinamica@dinamicatech.com
website: www.dinamicatech.com