Differential Algebra Space Toolbox (DAST)

Workshop on Nonlinear Propagation of Uncertainties using Differential Algebra

September 22nd, 2015 *Escape Dance Room*, *ESTEC*

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

- Introduction
- DAST Overview
  - SF Architecture
  - UPT Architecture
- Applications: the Apophis case
- Advanced Applications
- Conclusions

Introduction

Differential Algebra

Mathematical foundation

- Substitution of classical real algebra with a new algebra of Taylor polynomials;
- Any function of n variables is expanded into its Taylor polynomial up to an arbitrary order k;

Implementation in a computerized environment (DACE)

- Efficient implementation of basic DA routines using FORTRAN 95;
- Powerful C++ interface (operator overloading, templates, etc.);
- Experimental Matlab interface;

How can be used for uncertainty propagations?

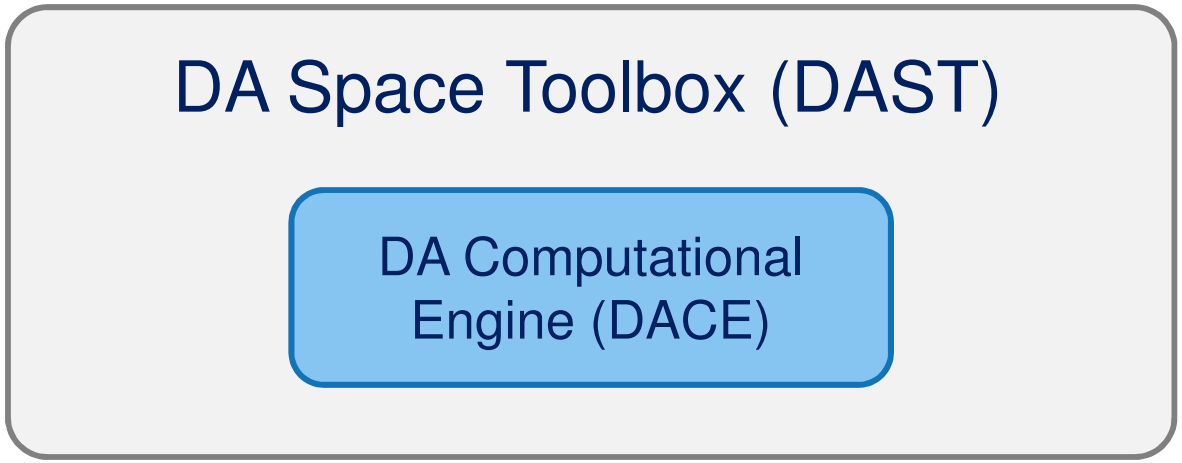
- Starting from the DA representation of the initial conditions, the flow of a given ODE can be obtained in terms of Taylor expansion;
- Resulting Taylor polynomials can then be used to compute the statistics of the propagated state through:
  a) DA-based Monte Carlo (DAMC);
  b) Linearized Dynamics (LD);
  c) Polynomial Bounder (PB);

Introduction

What if all these aspects are combined together in a tool for uncertainty analyses in space dynamics?

Differential Algebra Space Toolbox (DAST)

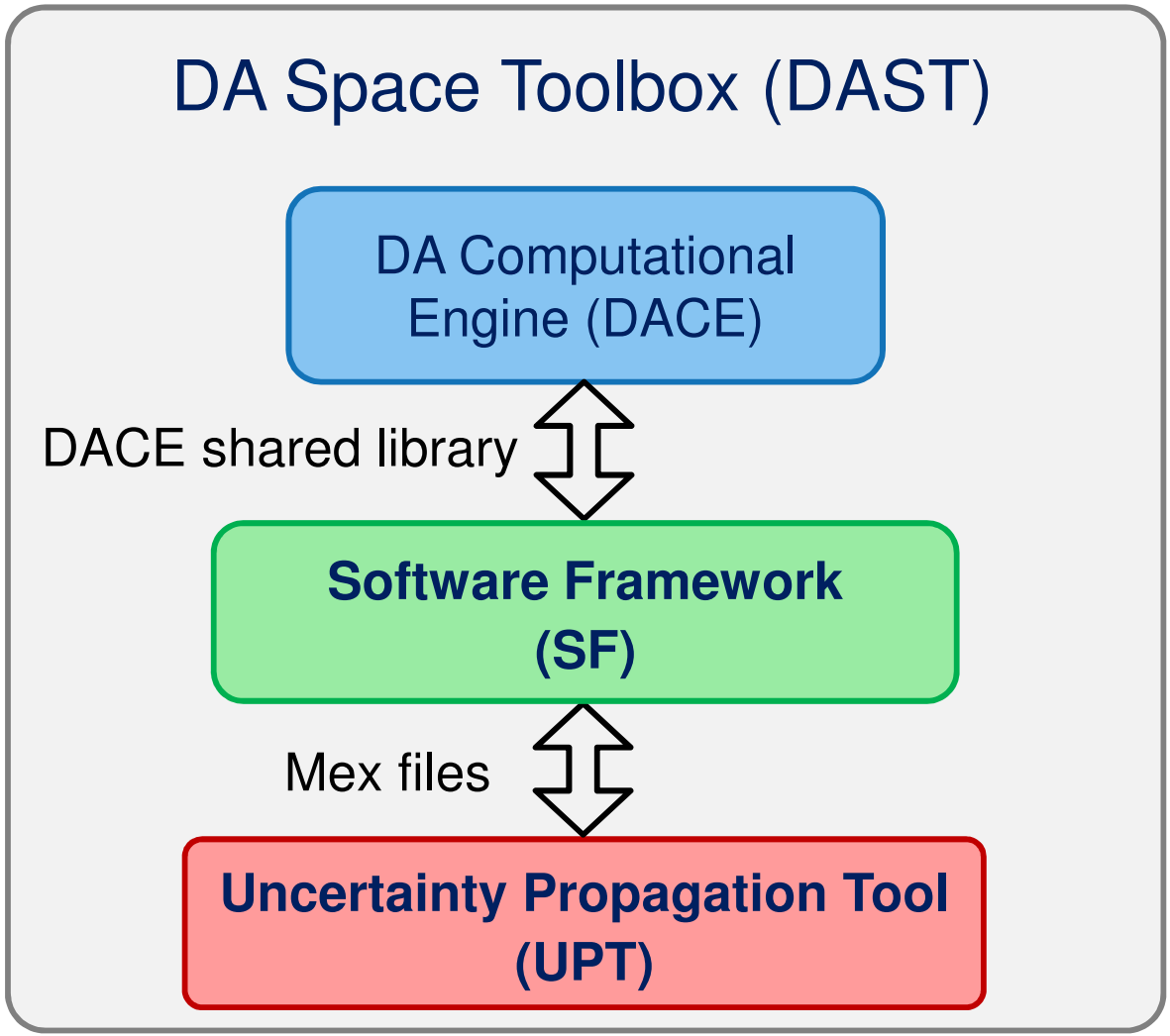
DAST Module Introduction



Inside DAST two sub-modules are introduced:

- Software Framework (SF)
- Uncertainty Propagation Tool (UPT)

DAST Module Overview



Possible Use-Cases of the Software

Standard user

Uncertainty Propagation Tool (UPT)

Uncertainty propagation in various dynamical systems

Developer

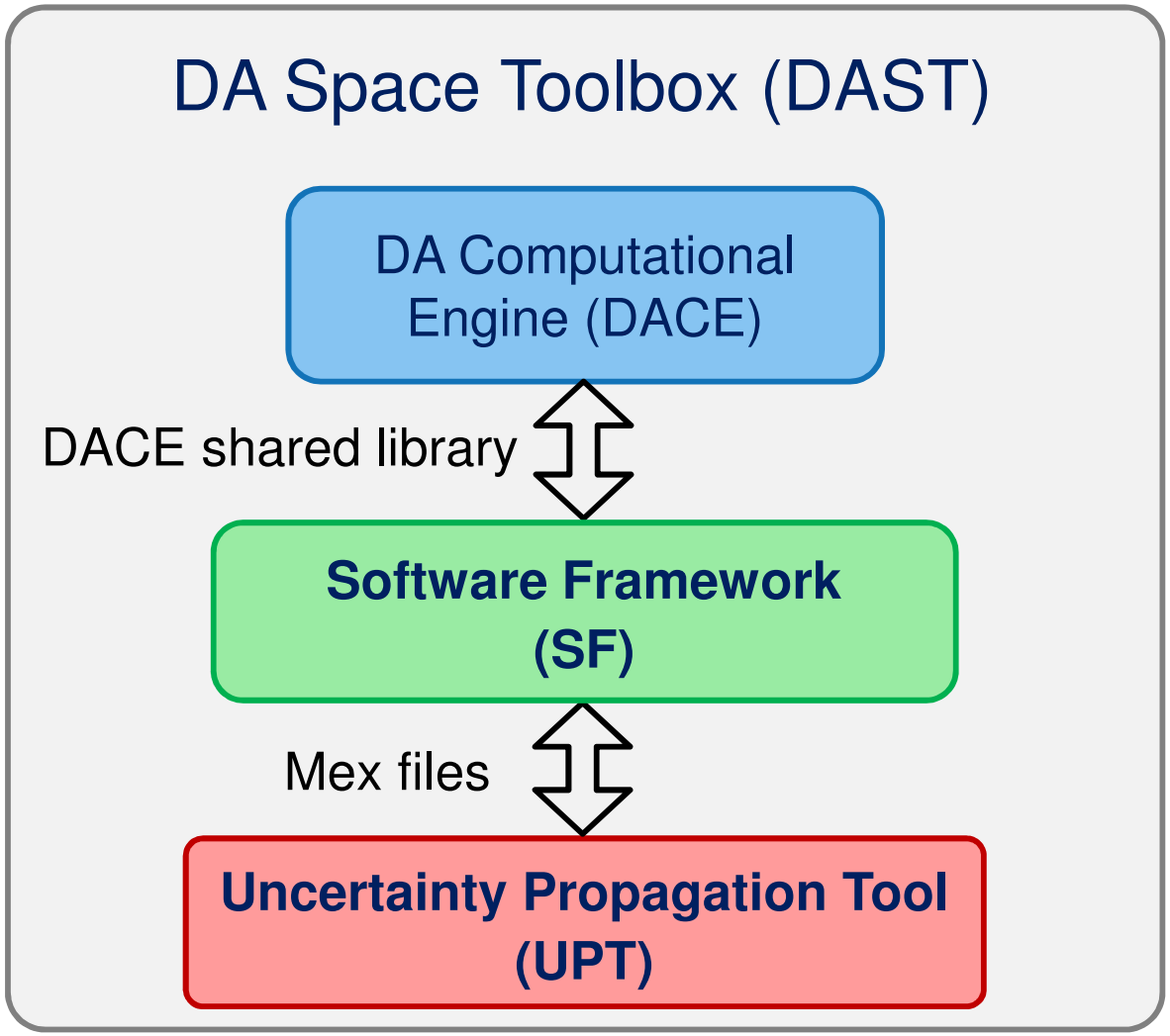
Software Framework (SF)

Implementation of custom DA-based astrodynamics applications

DACE

Use of DA operations for any custom application

DAST Module Overview

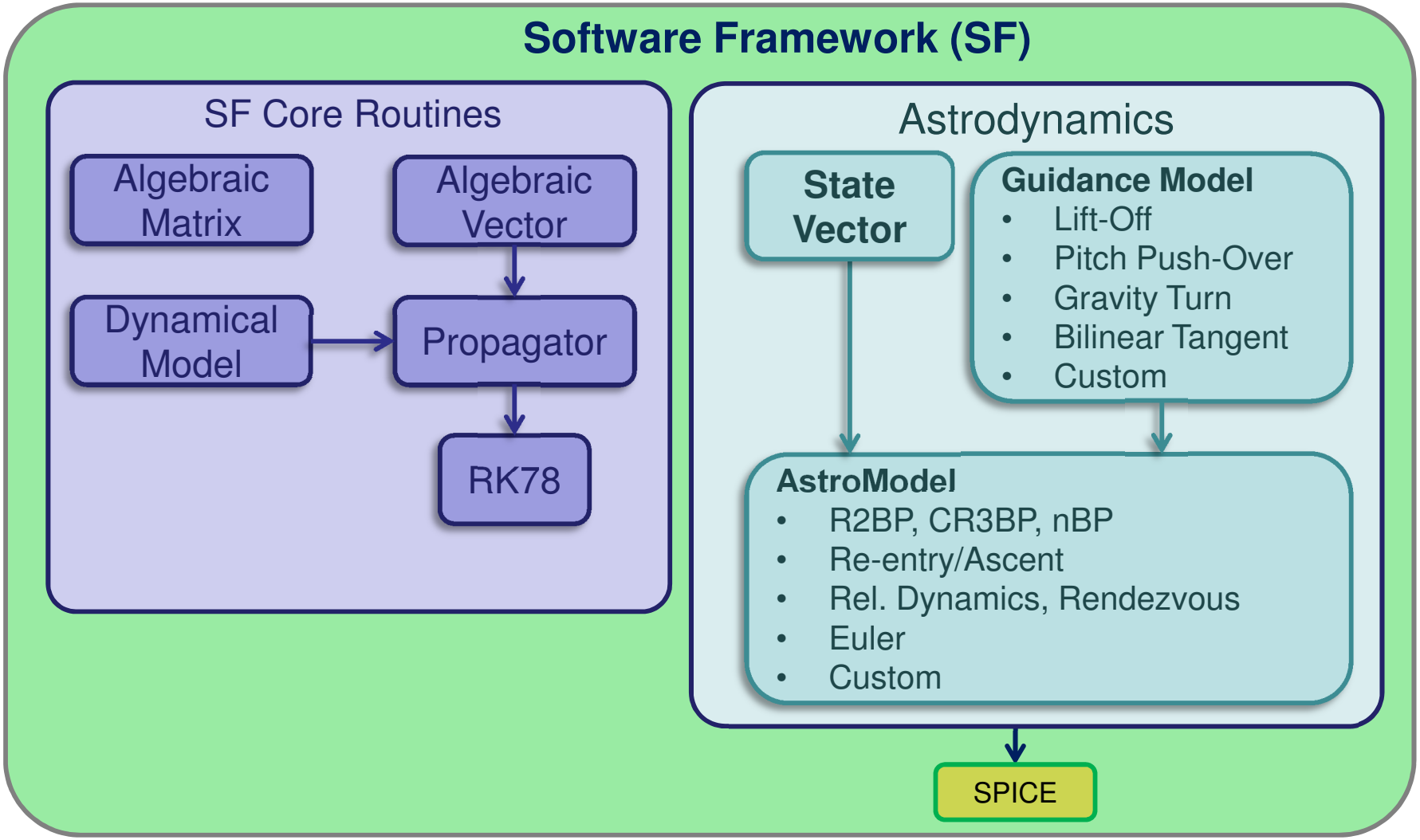


Purpose of SF Module

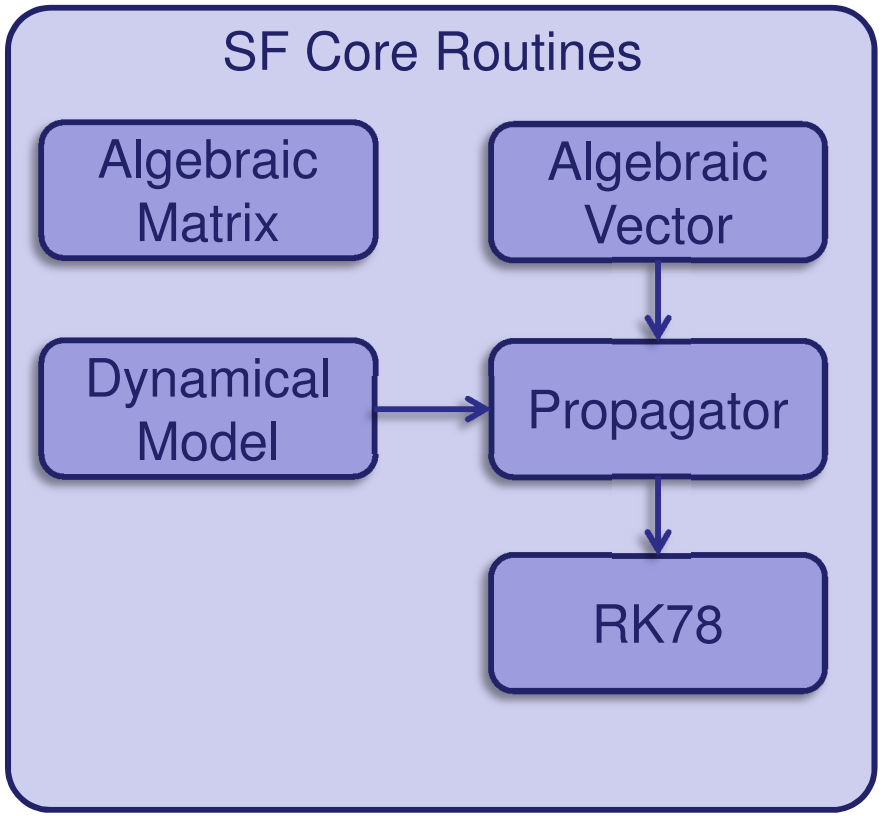
- The SF includes the implementation of more advanced features as:
  - DA operations between vectors and/or matrices of DA
  - DA implementation of propagation schemes (RK78)

- The SF also provides astrodynamics routines, which allow:
  - Defining a state vector at a certain epoch, with respect to given coordinate system, reference frame and units
  - Writing dynamical models and evaluating them using DA propagators.
  - Solving typical astrodynamics problems (DA Kepler solver)

SF Architecture Design



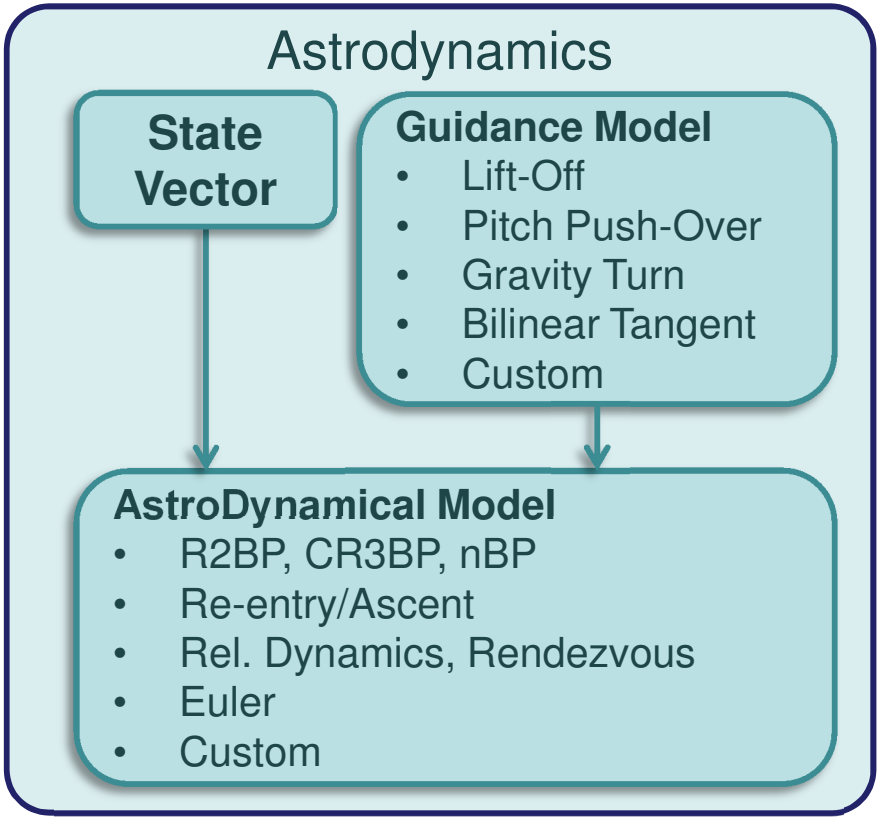
SF Core Routines



- Algebraic Vector
  - Any arithmetic type (double, DA, …)
  - Component-wise operations
  - Intrinsic functions evaluations
  - Vector norm, cross & dot products
- Algebraic Matrix
  - Any arithmetic type (double, DA, …)
  - Common matrix operations
- Dynamical Model
  - Abstract class, right-end-side (RHS) of ODEs
  - Evaluation of RHS of ODEs
- Propagator (variable and fixed step)
  - Abstract class
  - Propagate Dynamical Models
- RK78

SF Astrodynamics Routines

- State Vector
  - State (Algebraic Vector)
  - Epoch
  - Coordinate, Frame, Units
  - Unit and state transformations routines
- Guidance Model
  - Abstract class
  - Specialized guidance models
    - Lift-Off, Pitch Push-Over, Bilinear Tangent
    - Custom (user-supplied)
- AstroDynamical Model
  - Abstract class, derived from Dynamical Model
  - Dimension, parameters, frame, units
  - Specialized dynamical models
    - R2BP, CR3BP, nBP, Reentry, Rel. Dynamics, Euler, Ascent dynamics, Rendezvous
    - Custom (user-supplied)



SF Example

```
int main()
{
   // Global setup
   int ord   = 3;                                  // order of polynomials
   int nvar  = 6;                                  // number of DA variables
   DA::init( ord, nvar );                          // initialize DA environment

   // Define dynamical model
   R2BP<DA> dynR2BP( EARTH );                      // specify the center body

   // Set the gravitational parameter
   double GM = 398600.0;                           // user-defined value for GM
   dynR2BP.setFixParam( "GM", GM );                // set fixed parameter with label "GM"

   // Reference epoch:
   double t0 = mjd2et( 3456. );                    // convert to ephemeris time (used in SPICE)

   // Define uncertainties
   AlgebraicVector<double> sigma( 6, 0. );         // create a vector of double with 6 elements
   sigma[0] = 3.1681e-01;  sigma[1] = 1.2159e+00;  sigma[2] = 3.6776e-01;
   sigma[3] = 3.1216e-04;  sigma[4] = 1.5529e-05;  sigma[5] = 7.8670e-05;
```

SF Example

```
   // Define state vector
   StateVector<DA> X0( 6, t0, CARTESIAN, ECI, STDUNITS ); // create a state vector of DAs
   X0.state(0) = -6.86740024261e+03 + sigma[0]*DA(1);
   X0.state(1) = -5.31170788774e+02 + sigma[1]*DA(2);
   X0.state(2) =  8.76076142159e-02 + sigma[2]*DA(3);
   X0.state(3) =  5.79118186169e-01 + sigma[3]*DA(4);
   X0.state(4) = -7.57879982355e+00 + sigma[4]*DA(5);
   X0.state(5) =  3.27557629871e-01 + sigma[5]*DA(6);

   // Convert to Keplerian coordinates
   StateVector<DA> X0_kepl = ConvertState( X0, KEPLERIAN, ECI, STDUNITS );

   // Define propagation interval
   double sma = cons( X0_kepl.state(0) );      // retrieve semi-major axis
   double T   = 2*kPI*sqrt( sma*sma*sma/GM ); // compute the orbital period
   double tf  = t0 + 5.0*T;                   // define final propagation time

   // Propagation
   double tol = 1e-9;                         // specify the required tolerance
   StateVector<DA> XF = AstroPropagator( dynR2BP, X0, tf, tol );

   // Output
   cout << XF.getState() << endl;             // print final state

   return 0;
}
```

SF Typical Output

```
cout << XF.getState() << endl;
```

```
[[[ 6x1 vector
     I  COEFFICIENT                ORDER  EXPONENTS
     1   -6867.400242609914          0    0  0  0  0  0  0
     2    0.9889422361077381         1    1  0  0  0  0  0
     3    0.5940704972304844E-01     1    0  1  0  0  0  0
     4    0.9693864065624761E-03     1    0  0  1  0  0  0
     5   -0.5168487183344602E-01     1    0  0  0  1  0  0
     6    0.1114077641578412E-01     1    0  0  0  0  1  0
     7   -0.6605125264344823E-01     1    0  0  0  0  0  1
     8   -0.3809470814427352E-03     2    2  0  0  0  0  0
     9    0.6615685163868492E-02     2    1  1  0  0  0  0
    10   -0.1209848551284325E-01     2    0  2  0  0  0  0
    11   -0.7499541976119523E-04     2    1  0  1  0  0  0
    12    0.2997131588034893E-03     2    0  1  1  0  0  0
    13   -0.6351351944019880E-03     2    0  0  2  0  0  0
    14   -0.8082450777004574E-03     2    1  0  0  1  0  0
    15    0.3044487598938943E-02     2    0  1  0  1  0  0
     . . .
-------------------------------------------------------------
 . . .

]]]
```

Constant part of the polynomial

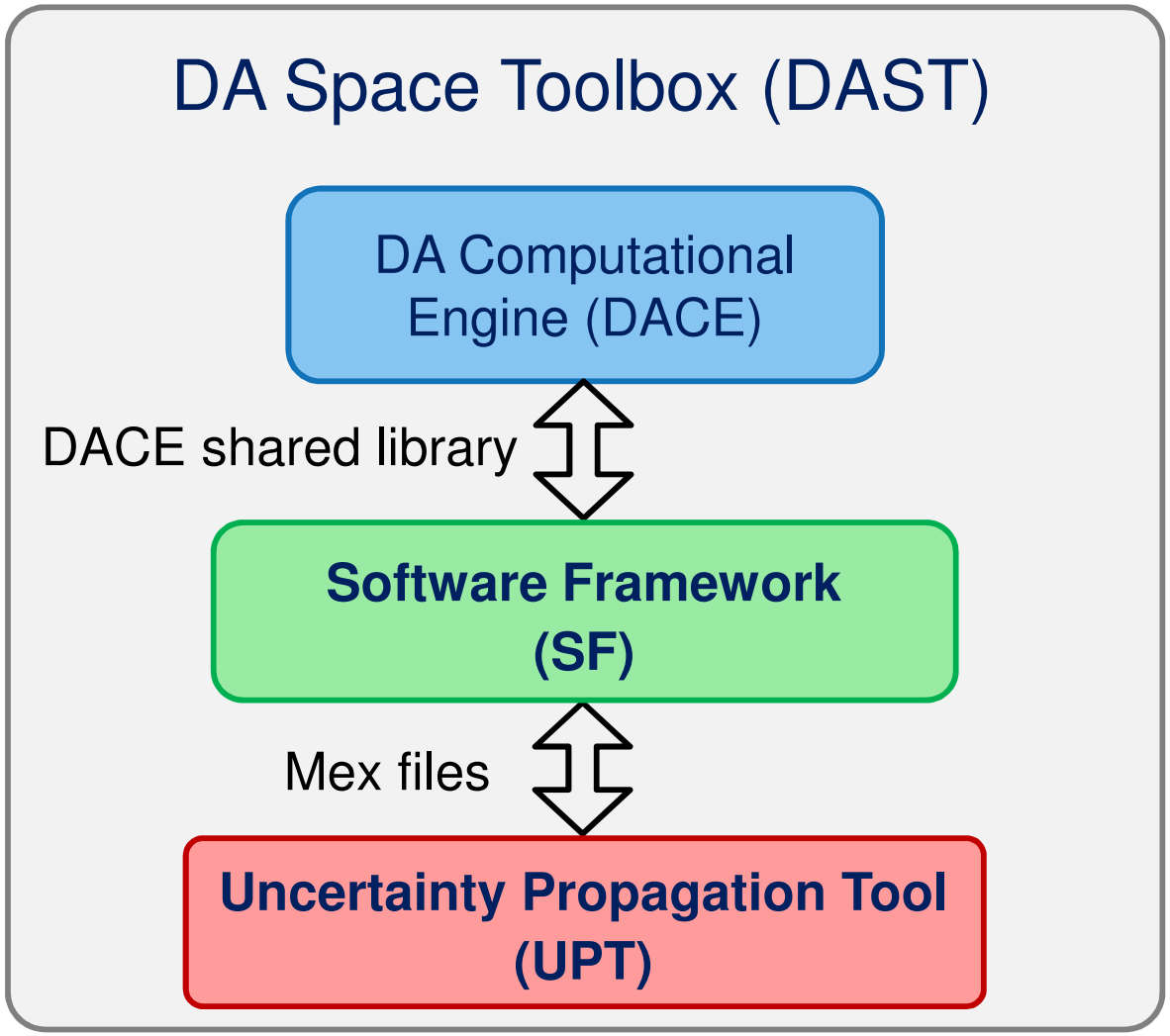
Other coefficients

1st row of STM

Higher orders

Other components of the final state vector

DAST Module Overview

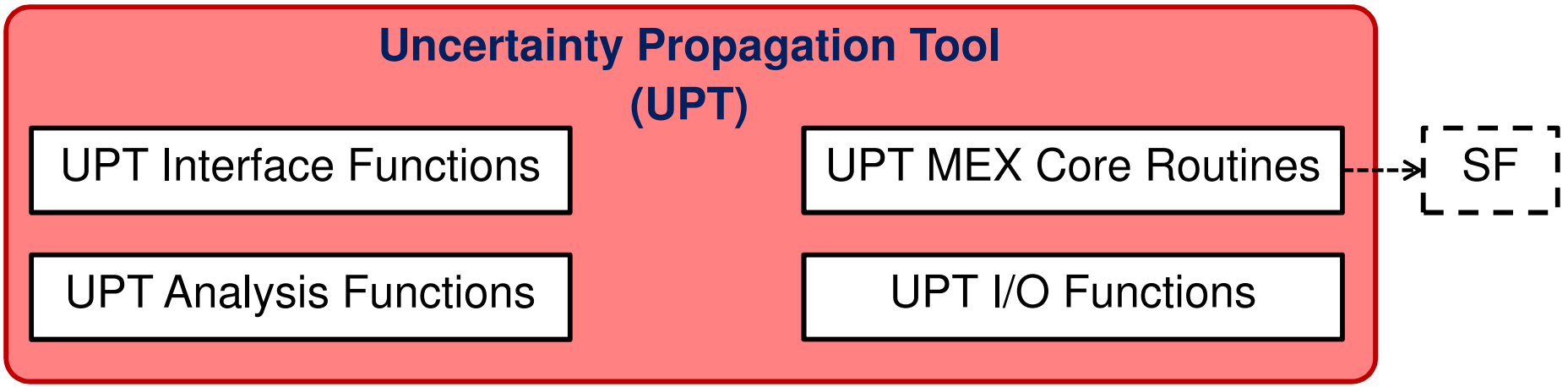


Purpose of UPT Module

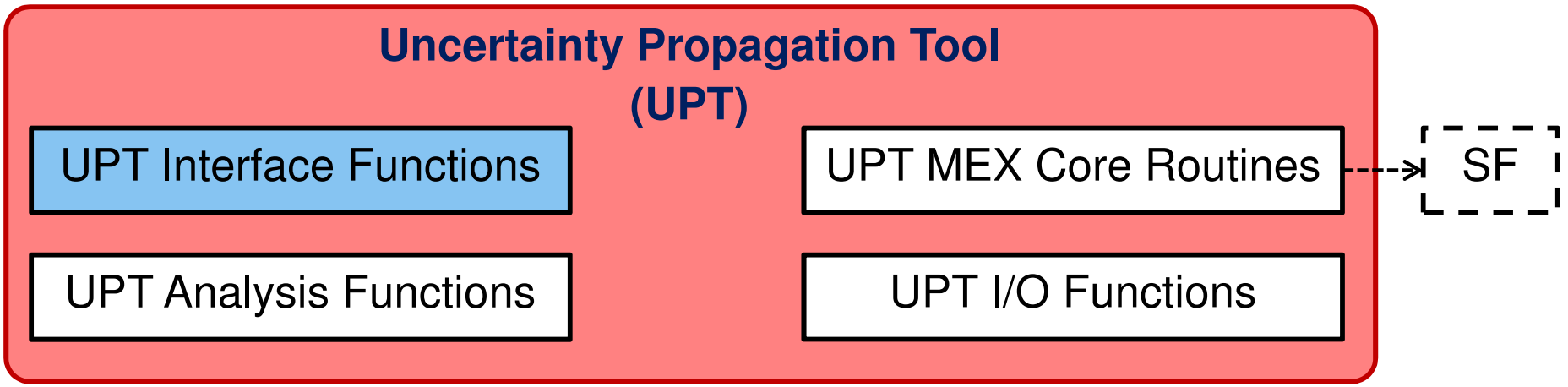
- The UPT is aimed at performing uncertainty propagations and statistical analyses.

- The UPT comes with a set of useful Matlab routines, which allow:
  - An easy interaction with the SF even for standard users (no need to write or compile code)
  - Managing simulation results
  - Easy graphical representations of the performed analyses

UPT Architecture Design

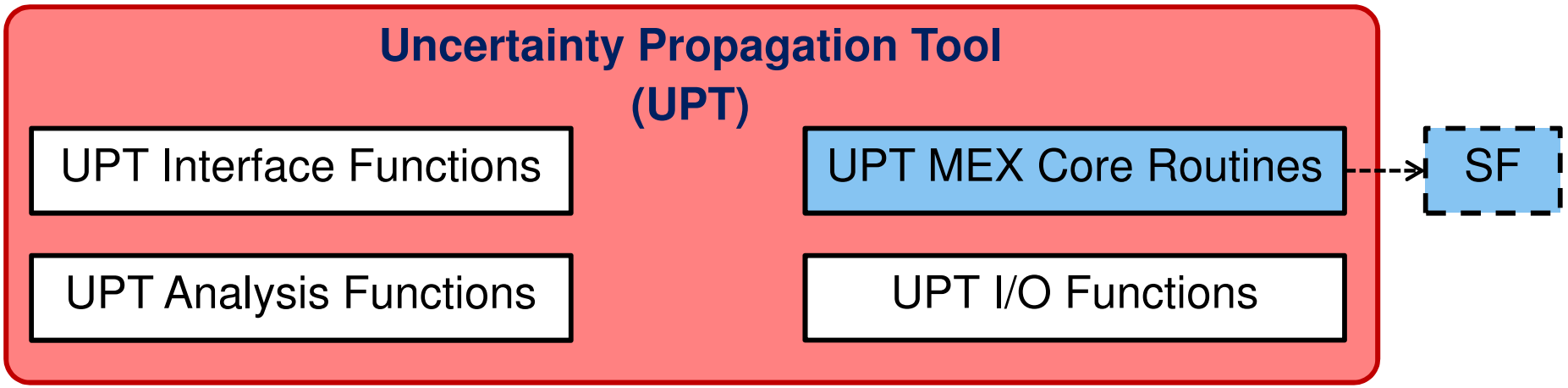


UPT Architecture Design



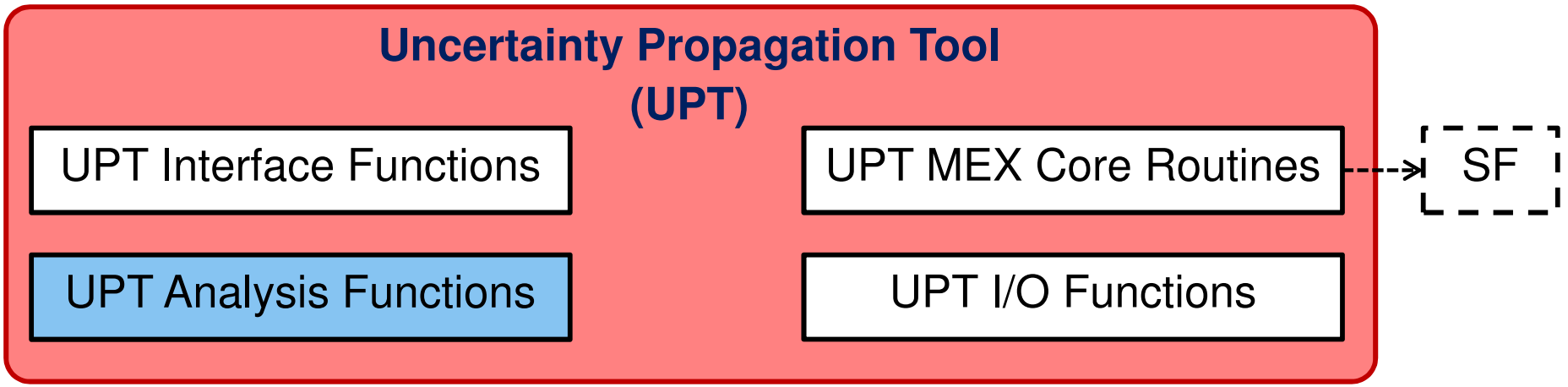
- **UPT Interface Functions**:
  - "`UPTmodel.m`" for dynamical model definition (model, reference frame, units, etc.)
  - "`UPTmethod.m`" for propagation method definition and all the required settings (order, number of samples, etc.)

UPT Architecture Design



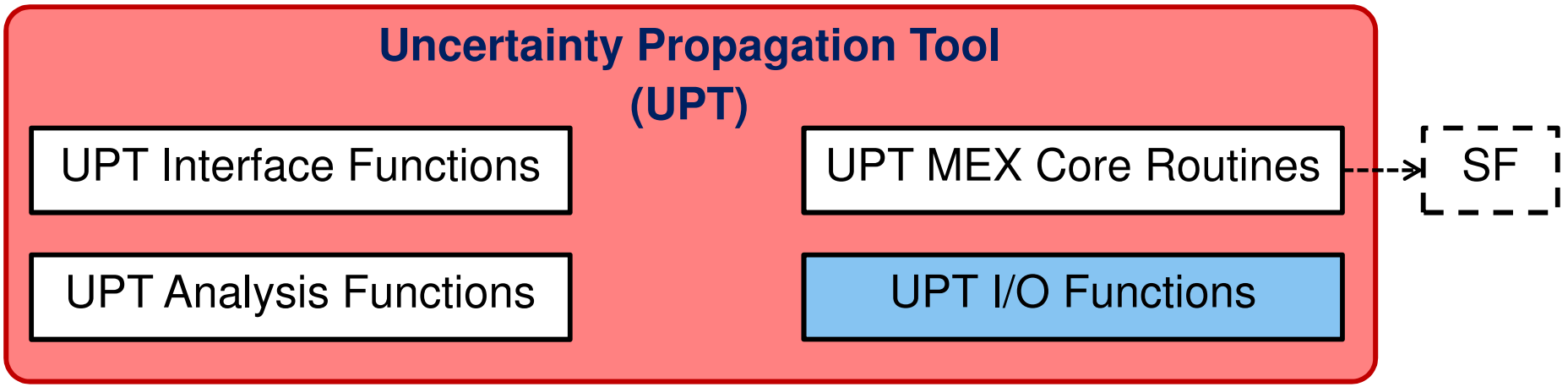
- **UPT MEX Core Routines**:
  - "`UPT.cpp`" to set up the DA environment and perform DA propagation
  - "`UPTaux.cpp`" contains auxiliary functions for conversions between C++ objects and Matlab variables

UPT Architecture Design



- **UPT Analysis Functions**:
  - "`UPTrun.m`" to collect all the inputs and call mex files.
  - "`UPTeval.m`" for additional evaluations of the final DA map (different distribution, number of samples).
  - "`evalCDA.m`" for fast evaluations of a compiled DA map (for advanced users).

UPT Architecture Design



- **UPT I/O Functions**:
  - "load_CDA.cpp" for loading a compiled DA from file (for advanced users)
  - "compiledDA.cpp" for saving compiled DA in an easy to handle format (for advanced users)
  - Other additional routines for plotting results ("drawCube.m", "plot_gaussian_ellipsoid.m", etc).

UPT Summary

- Summarizing, the main steps required for a typical analysis are:

1) Call to **UPTmodel** for dynamical model definition.

2) Call to **UPTmethod** for propagation method definition.

3) Call to **UPTrun** to perform DA propagations.

4) Eventually, call to **UPTeval** for additional evaluations of the final DA map.

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

UPT Example: Two-Body Dynamics

STEP 0 ➡ Problem variables declaration:

```
% Initial state of satellite. Units: km, km/s
state  = [ -6.867400242607589e+03;
           -5.311707887737306e+02;
            8.760761421587127e-02;
            5.791181861692615e-01;
           -7.578799823552207e+00;
            3.275576929871216e-01]';

% Initial covariance matrix. Units: km^2, km^2/s, km^2/s^2
Cov = [  1.1152e-02  -1.0863e-02   4.5946e-04   1.1469e-05  -1.0429e-05   5.1217e-07;
        -1.0863e-02   1.7486e-01  -6.9291e-03  -1.6280e-04  -1.5031e-05   1.1538e-07;
         4.5946e-04  -6.9291e-03   1.5302e-02   7.0795e-06   6.1989e-07   1.2962e-07;
         1.1469e-05  -1.6280e-04   7.0795e-06   1.6260e-07   1.3260e-08  -7.9236e-11;
        -1.0429e-05  -1.5031e-05   6.1989e-07   1.3260e-08   1.3692e-08   2.1382e-11;
         5.1217e-07   1.1538e-07   1.2962e-07  -7.9236e-11   2.1382e-11   1.4019e-08 ];

% Initial epoch
t0 = '2014-08-20T03:52:10.8397';

% Final epoch
tf = '2014-08-20T19:40:38.2680';
```

UPT Example: Two-Body Dynamics

STEP 1 ➡ Model declaration:

Performed using the *UPTmodel* function.

```
model = UPTmodel('Model', 'R2BP', ...
                 'MainAttractor', 'EARTH',...
                 'InitialState', state, ...
                 'Coordinate', 'RECTANGULAR', ...
                 'Frame', 'J2000',...
                 'FrameCenter', 'EARTH', ...
                 'InitialEpoch', t0, ...
                 'FinalEpoch', tf,...
                 'LengthUnits', 'KM', ...
                 'TimeUnits', 'SEC',...
                 'AngleUnits', 'RAD');
```

Additional fields may be required for different dynamical models

UPT Example: Two-Body Dynamics

STEP 2 ➡ Method declaration:

Performed using the *UPTmethod* function.

```
% Define masking vector.
a_x = [1 1 1 1 1 1];

% Define method
method = UPTmethod('Method','DAMC', ...
                   'Order', 5, ...
                   'UncertainStates', a_x,...
                   'CovarianceMatrix', Cov, ...
                   'Samples', 1e5 );
```

DA based Monte Carlo

Masking vector to switch on/off uncertain states and/or model parameters:
1 - enabled
0 - disabled

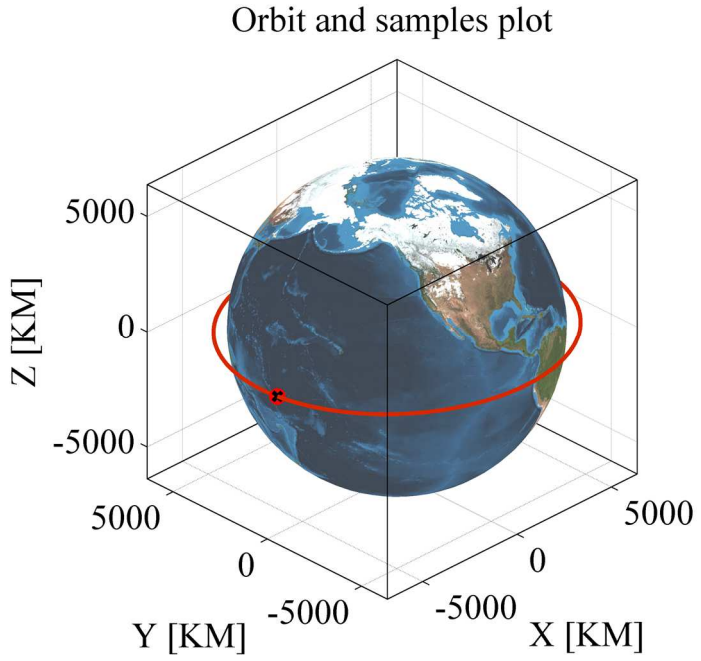
Additional fields may be required for different dynamical models

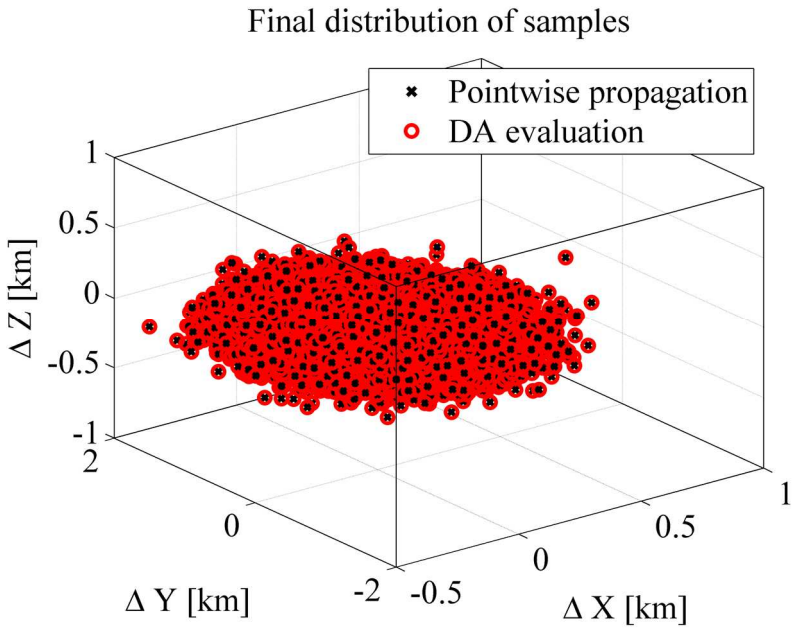
UPT Example: Two-Body Dynamics

STEP 3 ➡ Run UPT:

Performed using the *UPTrun* function.

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





UPT Example: Two-Body Dynamics

STEP 4 ➡ **Analysis of additional samples:**

If additional samples are desired, there is no need to re-run the whole UPT…

…just evaluate the map with new samples!!

```
% Generate and evaluate samples.
[ xf_distr, x0_distr ] = UPTeval( UPToutput, 'Gaussian', 1e4 );
```

Previous obtained results

Distribution

Number of samples

N.B: Samples can also be provided directly by the user, using any initial distribution (Matlab built-in functions can conveniently be exploited).

```
[ xf_distr, x0_distr ] = UPTeval( UPToutput, x0_distr );
```

UPT Example: Two-Body Dynamics

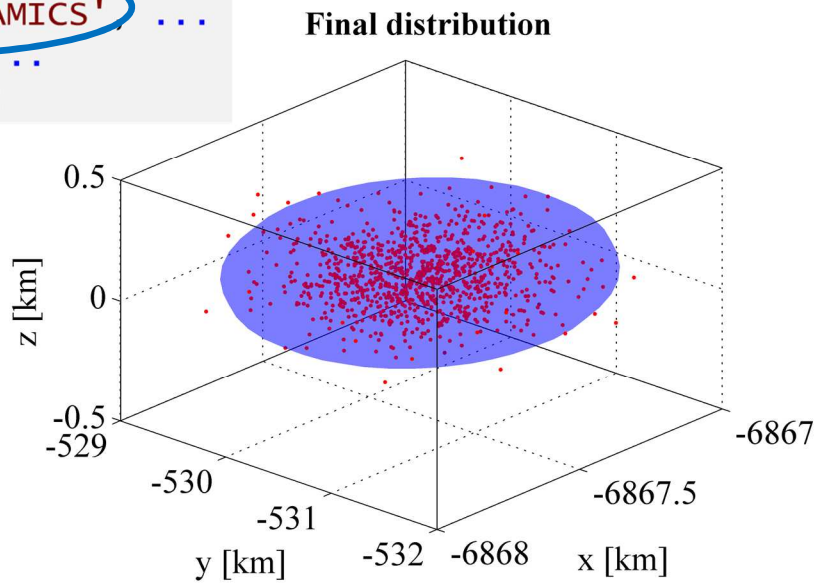
Other interesting features:

- **Linearized Dynamics (LD)**
  - Propagation of initial covariance through STM
  - The STM is directly the linear part of a DA final state
  - No need to integrate variational equations or compute finite difference

```
% Method definition
method = UPTmethod('Method','LINEARIZED_DYNAMICS', ...
                   'UncertainStates', a_x, ...
                   'CovarianceMatrix', Cov);
```

i. Covariance propagated with linearized dynamics propagation method

ii. Samples obtained using DAMC

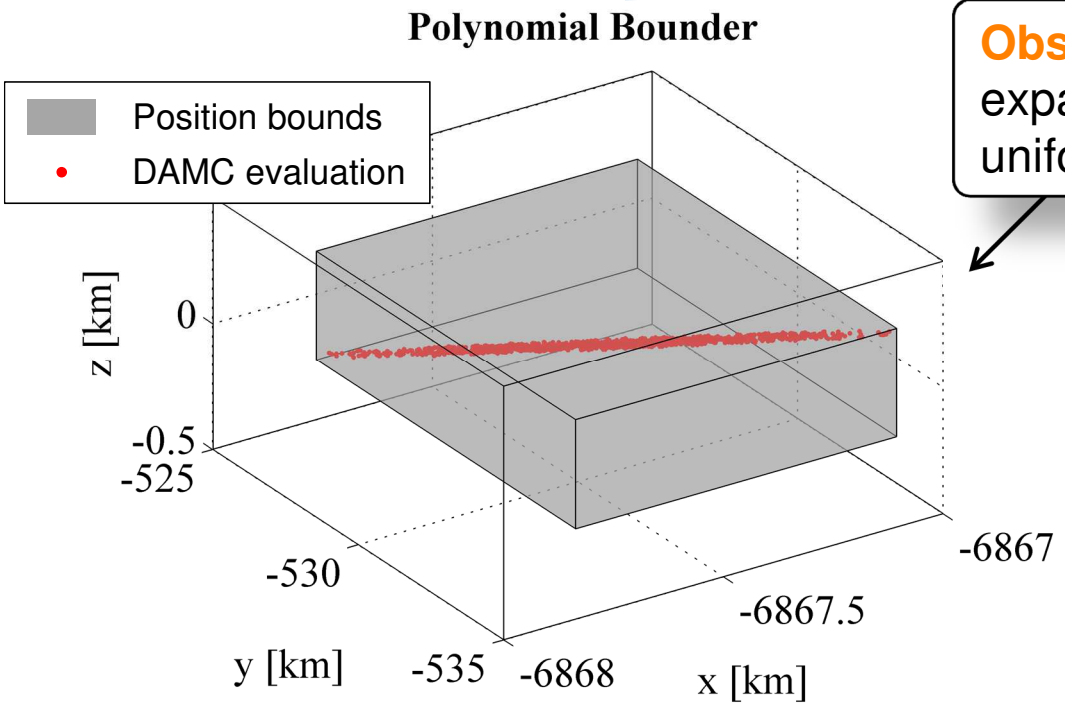


UPT Example: Two-Body Dynamics

Other interesting features:

- **Polynomial Bounder (PB)**

```
% Method definition
method = UPTmethod('Method','POLYNOMIAL_BOUNDER', ...
```

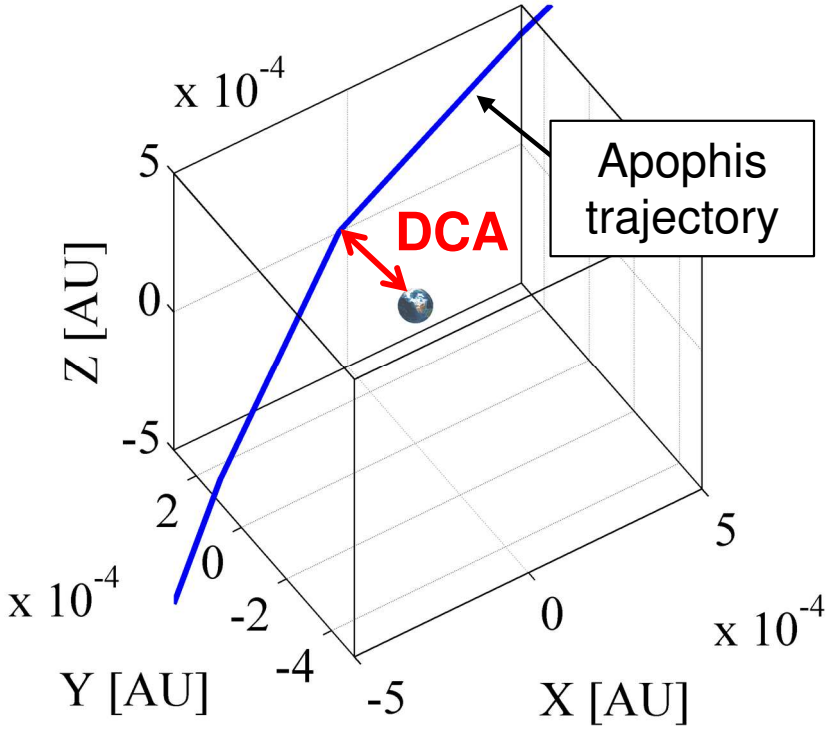


Obs. 4th order polynomial expansion, samples from uniform distribution

Applications: the Apophis case

Scenario

On its next approach on April 13, 2029, the asteroid Apophis will come within 35000 km of Earth.



DA can be used to describe how the distance of closest approach (DCA) changes due to uncertainties on the initial state.

Apophis Initial Conditions

- Apophis' ephemerides of June 18, 2009
  - Equinoctial elements
  - Diagonal covariance matrix

	Nominal value	σ	
a	0.922438242375914	2.29775 × 10–7	AU
P1	–0.093144699837425	3.26033 × 10–7	--
P2	0.166982492089134	7.05132 × 10–7	--
Q1	–0.012032857685451	5.39528 × 10–7	--
Q2	–0.026474053361345	1.83533 × 10–7	--
l	88.3150906433494	6.39035 × 10–4	deg

Apophis Closest Approach: DAMC

1. Initial condition initialized as DA and converted to cartesian ecliptic J2000 coordinates
2. Propagation forward in time to TCA, 13 April 2029 - 21:46 UTC using a n-body model
3. Taylor expansion of final state $[\boldsymbol{x}_f]$ used to compute Taylor expansion of DCA

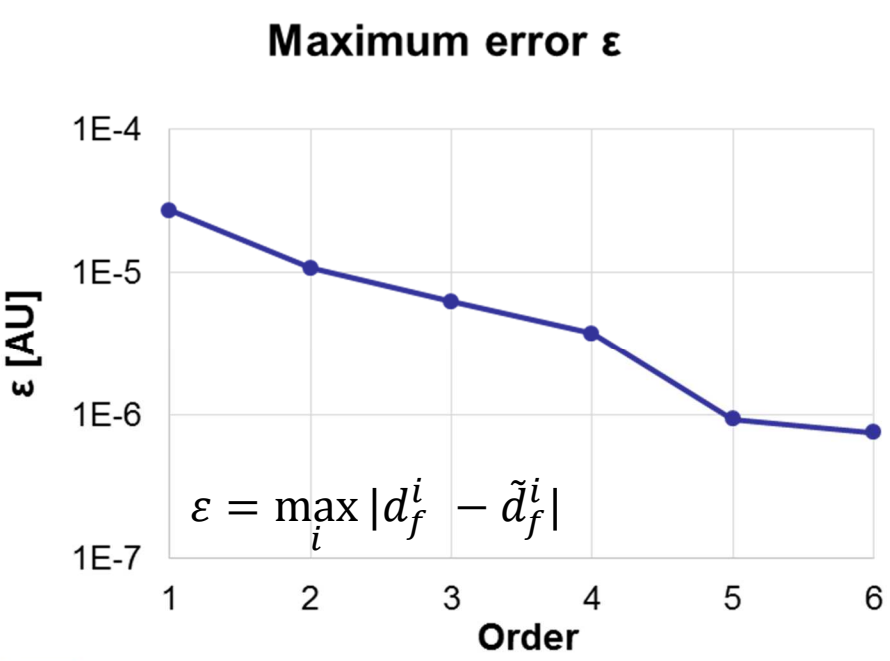
$$[\boldsymbol{x}_f] = \mathcal{T}_{\boldsymbol{x}_f}(\delta\boldsymbol{x}_0) \quad \longrightarrow \quad [DCA] = \mathcal{T}_{DCA}(\delta\boldsymbol{x}_0)$$

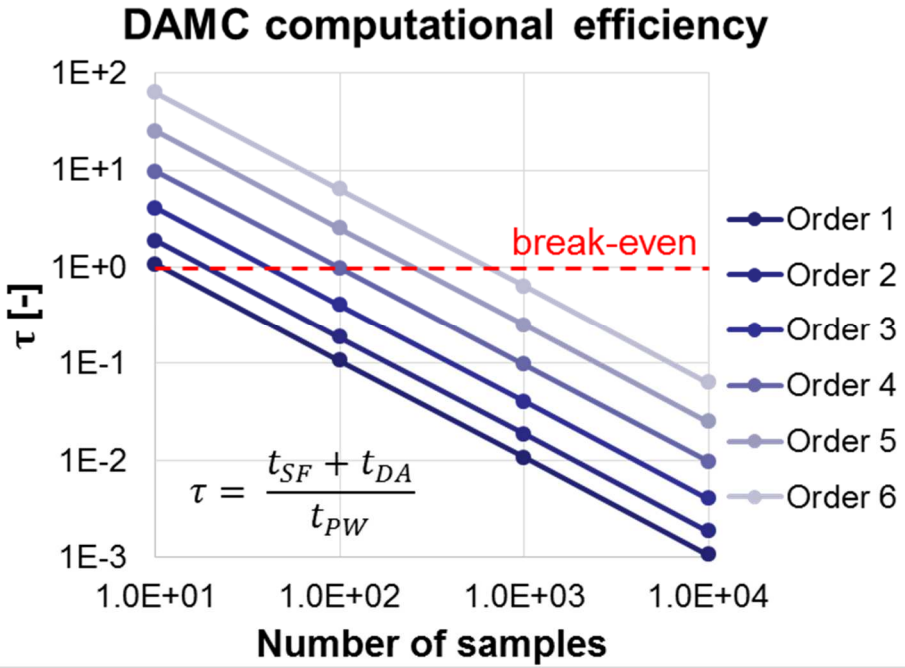
	DCA (AU)
NEODYS-2	0.00025
DAST	0.00025198

Obtained DCA is checked against NEODYS-2

Apophis Closest Approach: DAMC

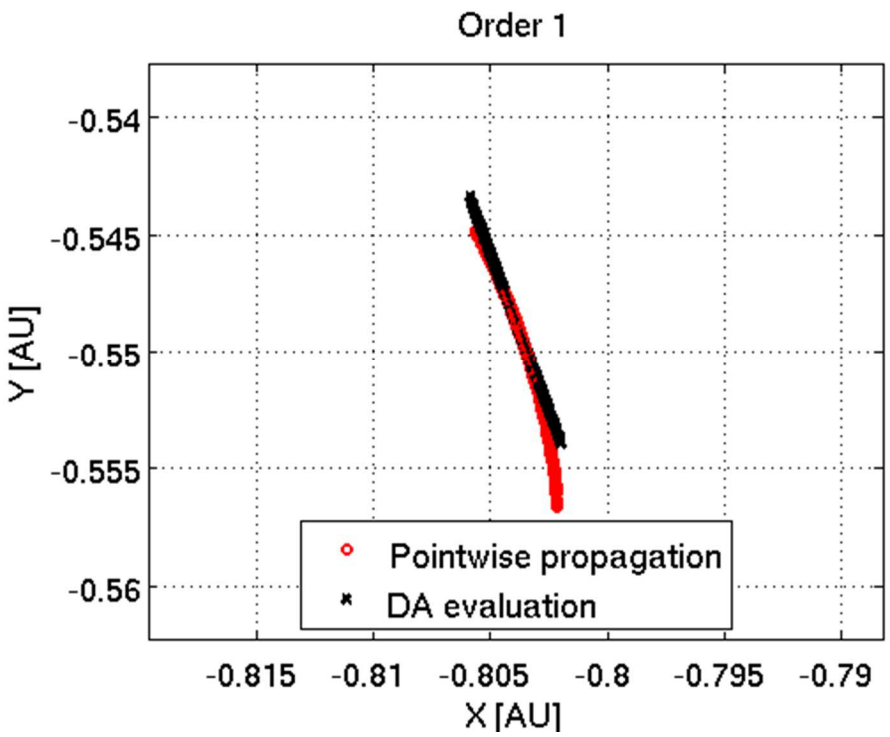
4. Propagation for further 10 days and computation of the Taylor expansion of the final distance wrt Earth $\mathcal{T}_{d_f}(\delta \boldsymbol{x}_0)$
5. Generation of N_T samples from initial statistic
   - MC Pointwise propagation, d_f^i
   - DAMC Polynomial evaluation of $[d_f]$ to compute approximation $\tilde{d}_f^i$

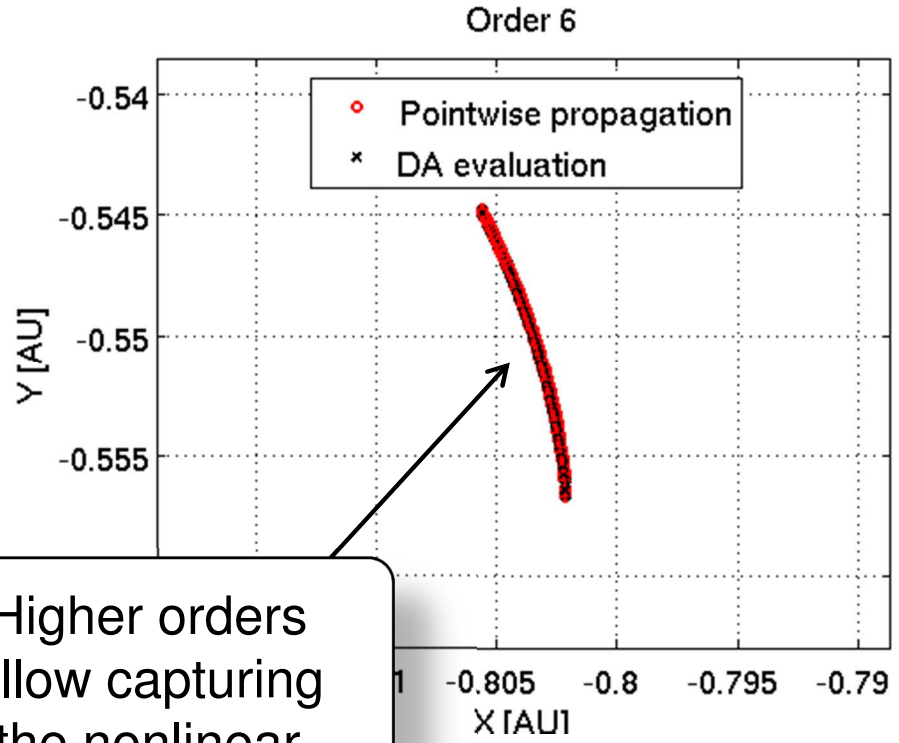




Apophis Closest Approach: DAMC

- Comparison standard MC and DA-based MC for increasing orders:





Higher orders allow capturing the nonlinear effects

Apophis Closest Approach: PB

- Polynomial Bounder (PB) propagation method:
  - Use polynomial bounder on $\mathcal{T}_{d_f}(\delta \boldsymbol{x}_0)$ to compute $[d_f^{LB}, d_f^{UB}]$

N.B. No need to generate samples and evaluate maps!!

Let's compare the results with those obtained with DAMC $[\tilde{d}_f^{min}, \tilde{d}_f^{max}]$:

Nsamples	$\varepsilon^{LB}\ [AU]$	$\varepsilon^{UB}\ [AU]$
1e3	-2.8108e-04	-5.6943e-04
1e6	2.6205e-05	-2.7087e-05

Minimum and maximum distance $\tilde{d}_f^i$ among all samples

Apophis Closest Approach: LD

- Linearized Dynamics (LD) propagation method:

a) Apophis' initial states initialized as **first order** DA variables

b) Propagation till TCA to compute $[\boldsymbol{x}_f] = \mathcal{T}^1_{\boldsymbol{x}_f}(\delta\boldsymbol{x}_0)$

c) Extraction of transition matrix Φ from $\mathcal{T}^1_{\boldsymbol{x}_f}(\delta\boldsymbol{x}_0)$

d) Compute final covariance matrix from initial covariance C_0

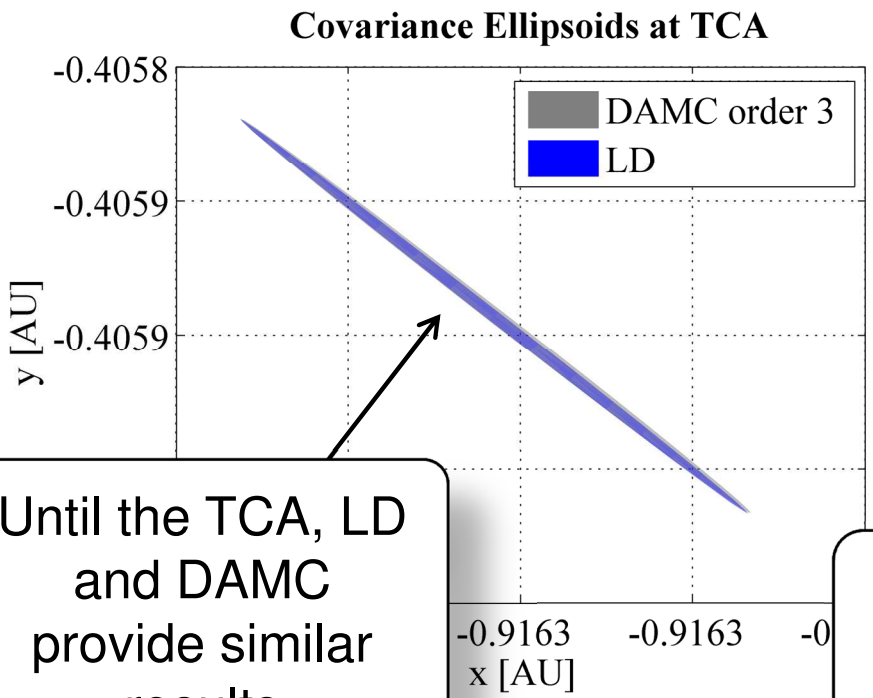
$$C_f^{UPT} = \Phi\, C_0\, \Phi^T$$

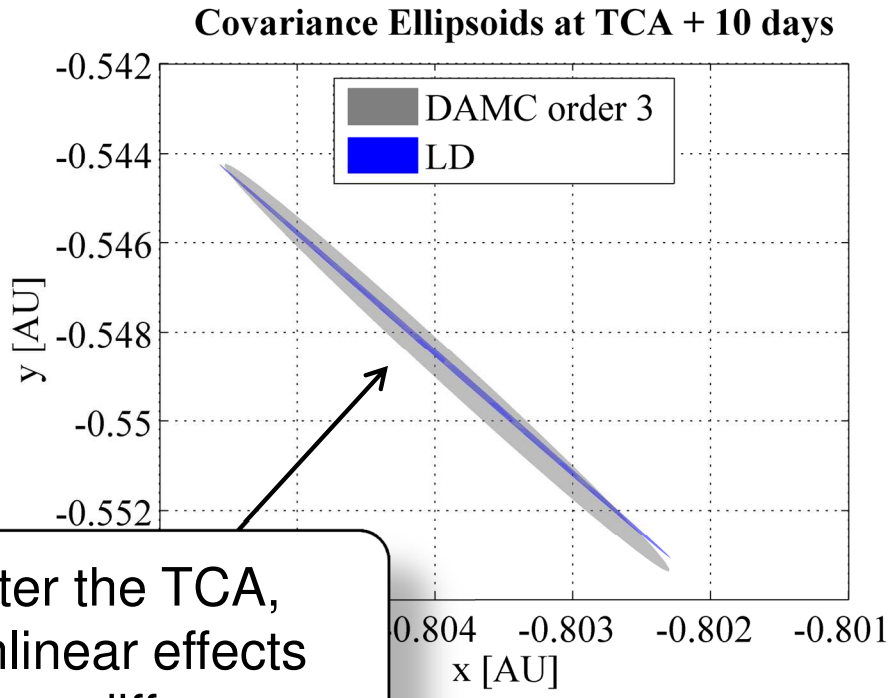
NB1: No need of variational equations!!

NB2: No need to generate samples and evaluate maps!!

Apophis Closest Approach: LD

- Comparison between DA-based MC and LD:





Until the TCA, LD and DAMC provide similar results

After the TCA, nonlinear effects produce differences in the covariance matrices

Advanced Applications

Scenario

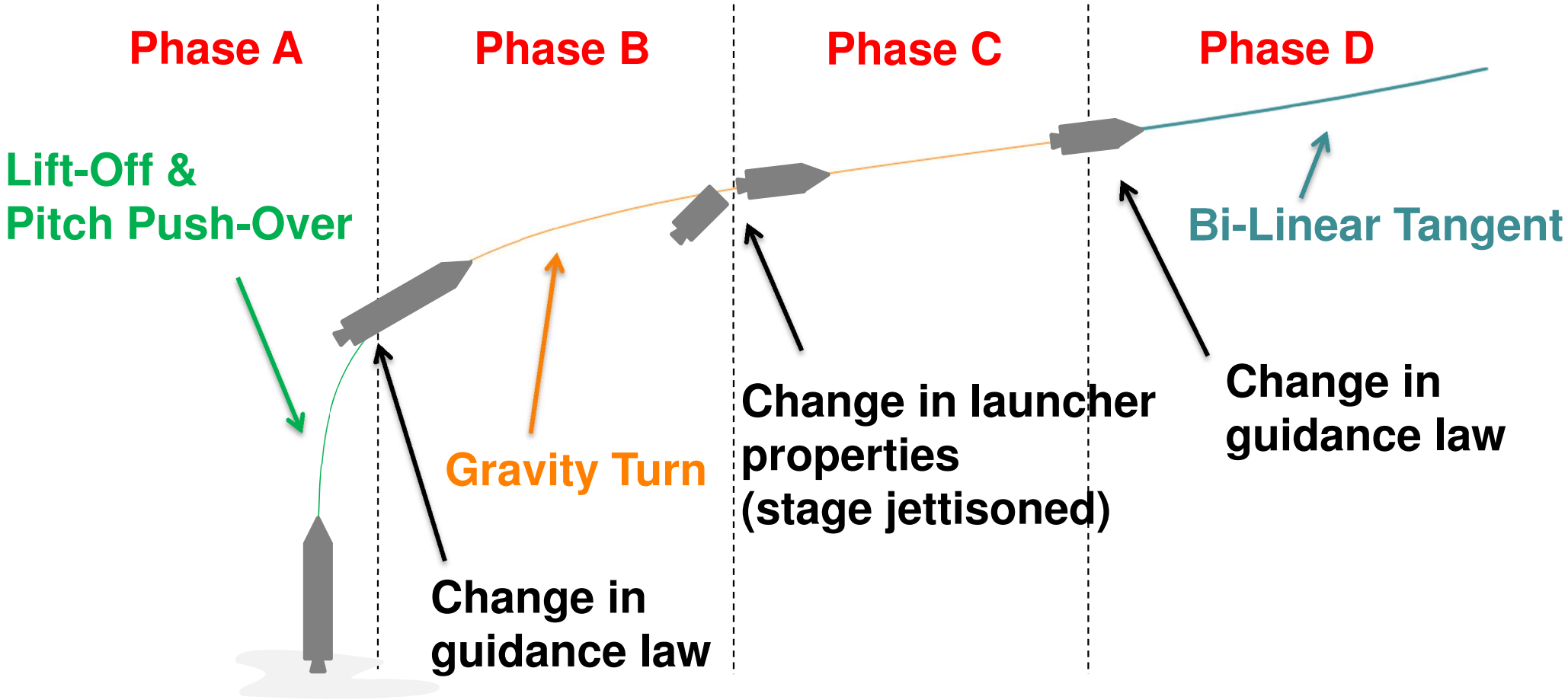
Suppose we want to assess the effects of uncertainties in the atmosphere model and/or in thrust magnitude and direction during the ascent trajectory of a Rocket Launch Vehicle (RLV).

Polynomials obtained with a DA approach could be used to:

- Assess uncertainties at orbit insertion
- Correct thrust to improve robustness

Main Assumptions

However…



Main Assumptions

- Within each phase:
  - Thrust magnitude does not depend on altitude (or atmospheric pressure) and is assumed constant (only direction changes);
  - Mass flow rate is constant;
  - Lift/Drag coefficients are constant (independent of Mach number);
  - Aerodynamic reference area is constant;

Possible uncertain model parameters

- Initial state (except for the first phase)
- C_L, C_d , ρ_0, T_{max}

+ Uncertain guidance parameters

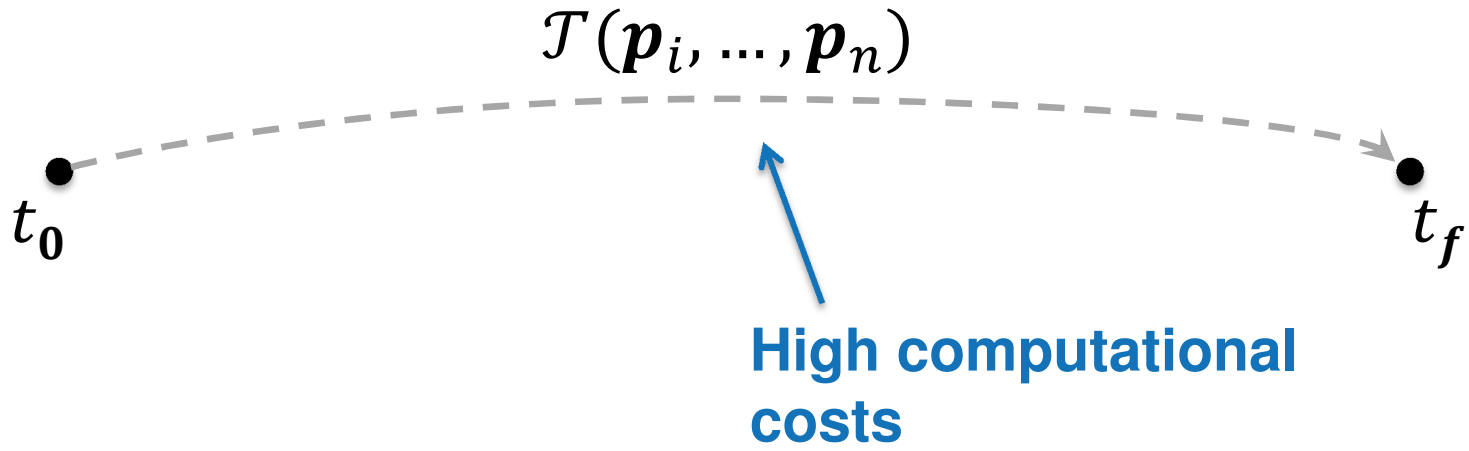
Main Assumptions

Summary:

Phase	Uncertain states	Model uncertain parameters	Guidance uncertain parameters	Total number of variables
A	-	$C_L, C_D, \rho_0, T_{max}$	$\Delta\Theta, \Delta\Psi, i_T$	7
B	$R, \lambda, \delta, V, \gamma, \chi$	$C_L, C_D, \rho_0, T_{max}$	-	10
C	$R, \lambda, \delta, V, \gamma, \chi$	$C_L, C_D, \rho_0, T_{max}$	-	10
D	$R, \lambda, \delta, V, \gamma, \chi$	$C_L, C_D, \rho_0, T_{max}$	Θ_0	11

Management of Multiple Phases

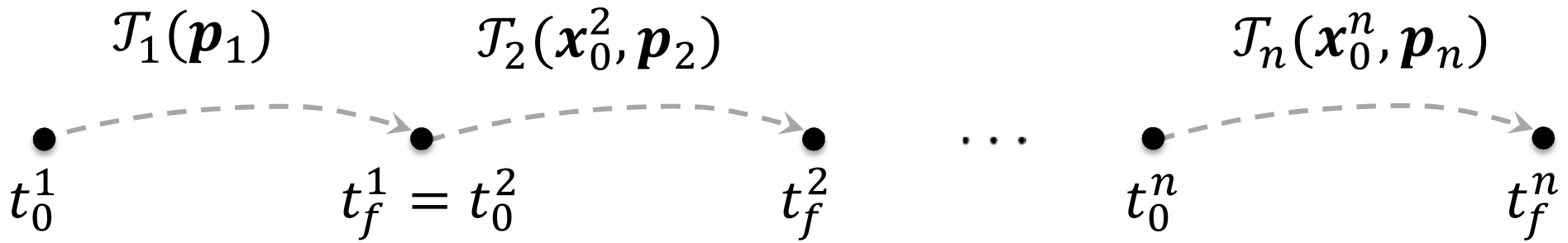
The expansion of the flow from t_0 to t_f yields to a Taylor polynomial which depends on a quite large number of variables (~35/40)



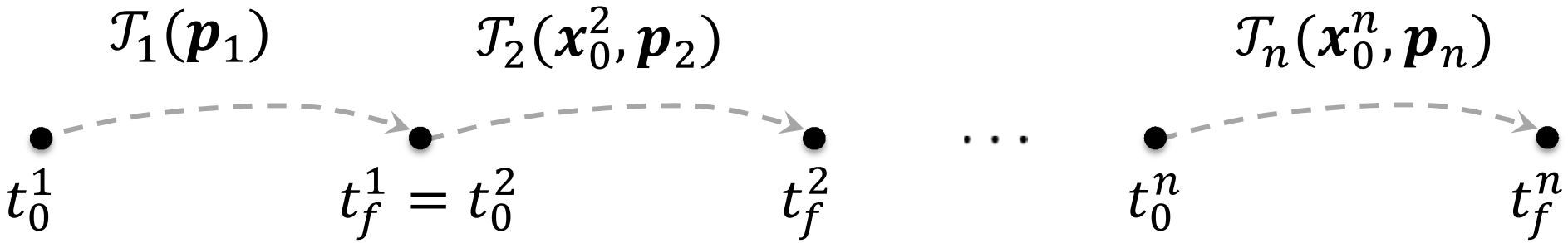
Management of Multiple Phases

The expansion of the flow from t_0 to t_f yields to a Taylor polynomial which depends on a quite large number of variables (~35/40)

The expansion of the flow is performed on each phase



Management of Multiple Phases



Pros:

- Optimization of the number of DA variables (comp. costs reduction);
- Possibility to Taylor expand each phase.

Cons:

- Need to properly manage the interactions between one phase and another.

VEGA Launcher: Target Orbit

- Consider a simplified launch trajectory of VEGA:

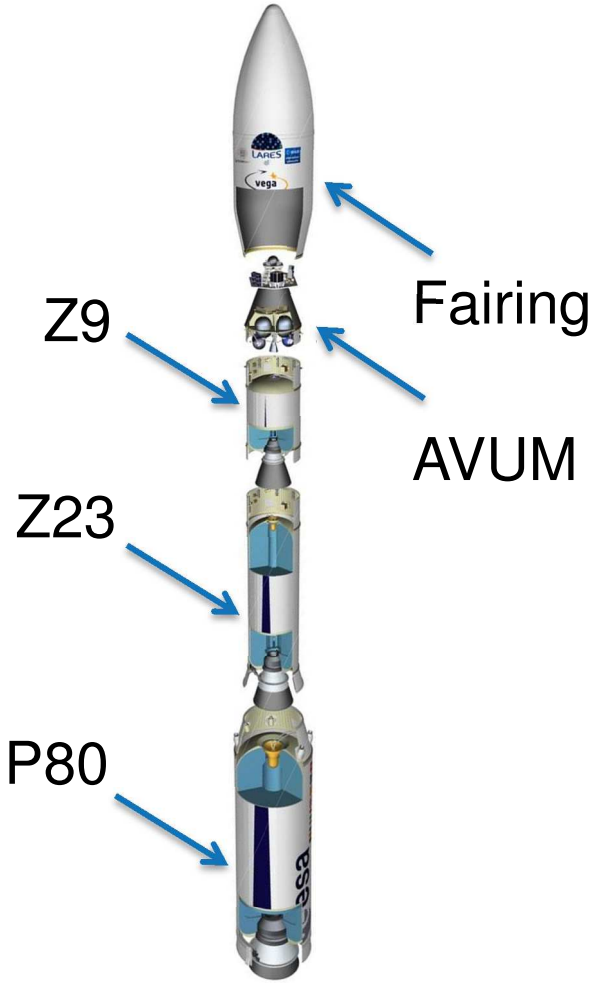
Departure site: Kourou launch site

Target orbit: Polar Earth Orbit (PEO)

$$h_f = 700 \text{ km}$$

$$e_f = 0$$

$$i_f = 90°$$

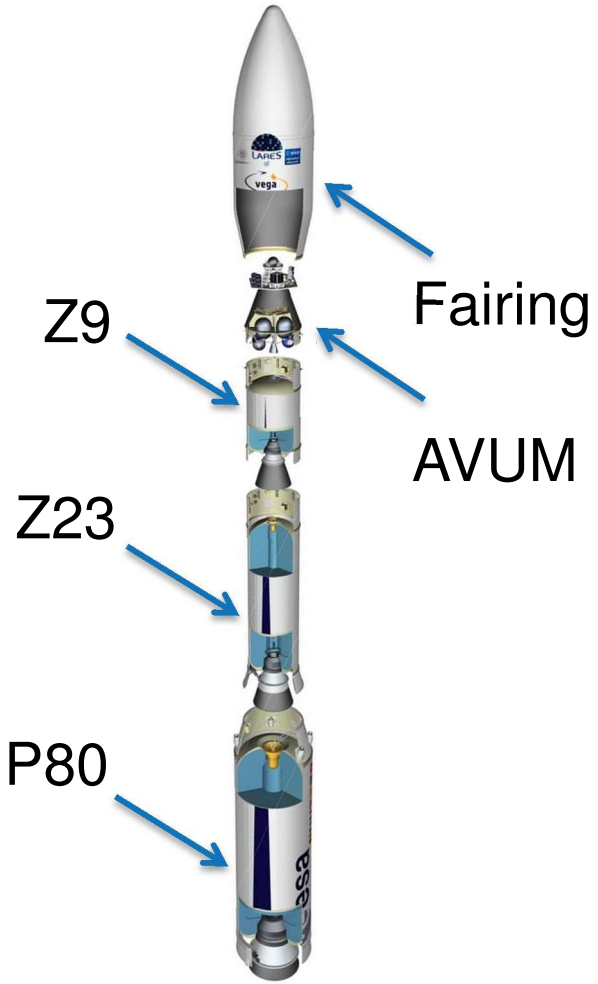


VEGA Launcher: Flight Sequence

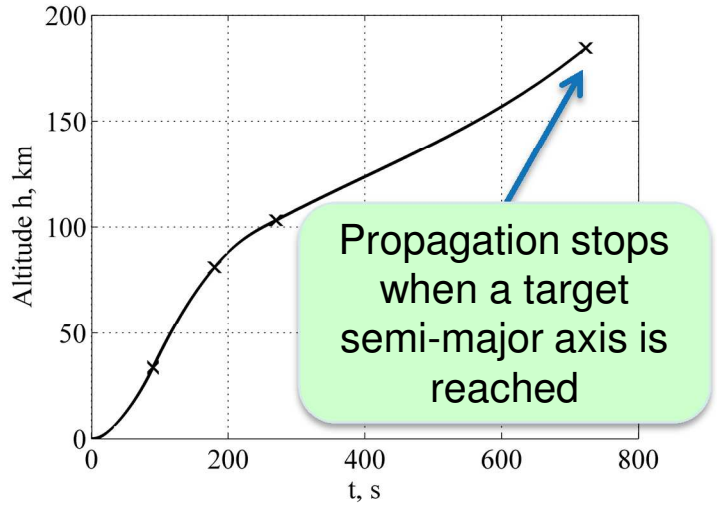
- Consider a simplified launch trajectory of VEGA:

Mission flight sequence:

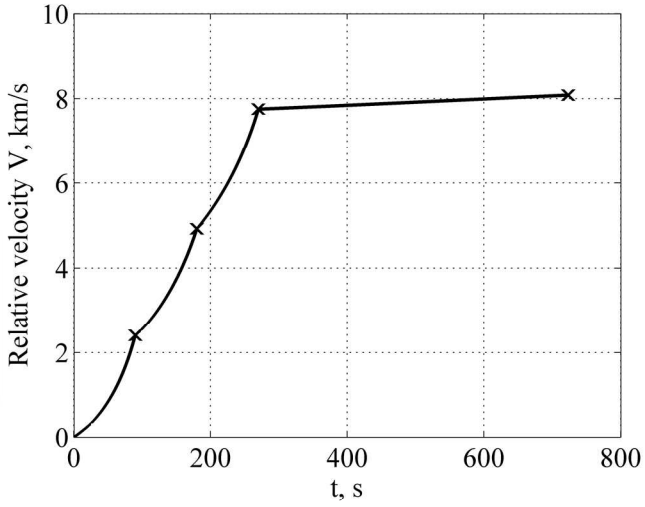
Phase	Duration [s]	Active propulsion	Guidance Law
A	0-90	P80	Lift-Off & Pitch Push-Over
B	90-180	Z23	Gravity Turn
C	180-270	Z9	Gravity Turn
D	270-920	AVUM	Bi-Linear Tangent

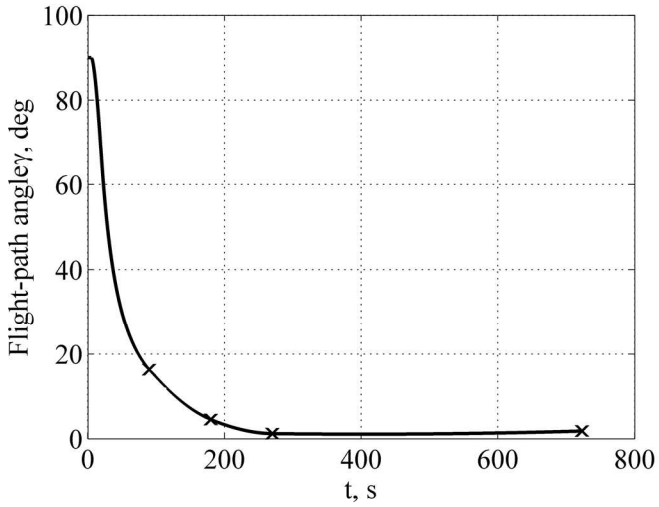


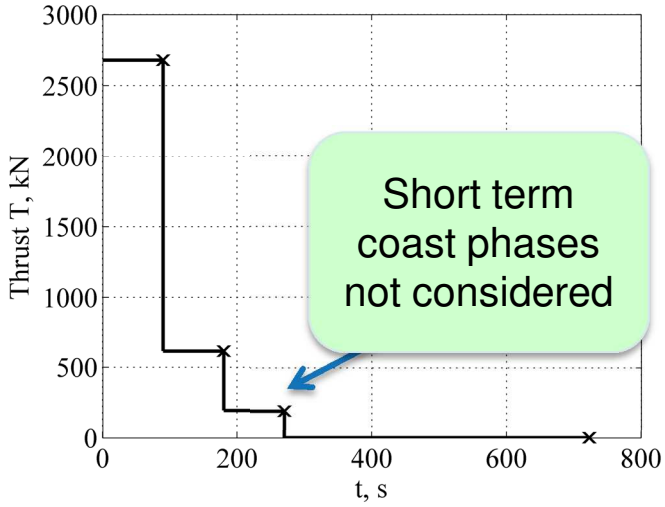
VEGA Launcher: Nominal Trajectory



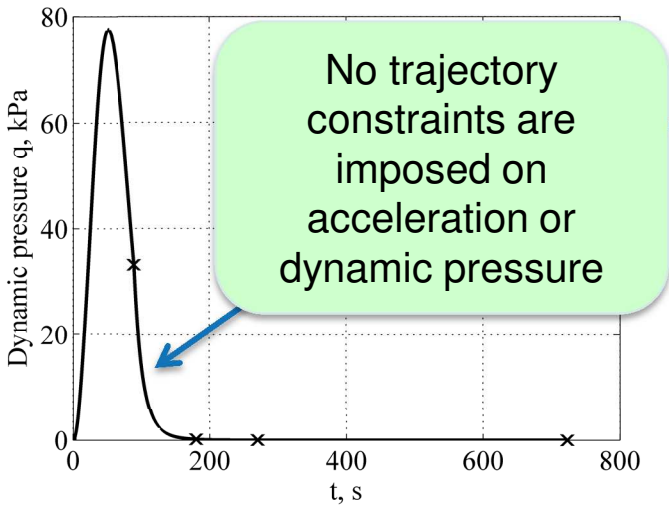
Propagation stops when a target semi-major axis is reached



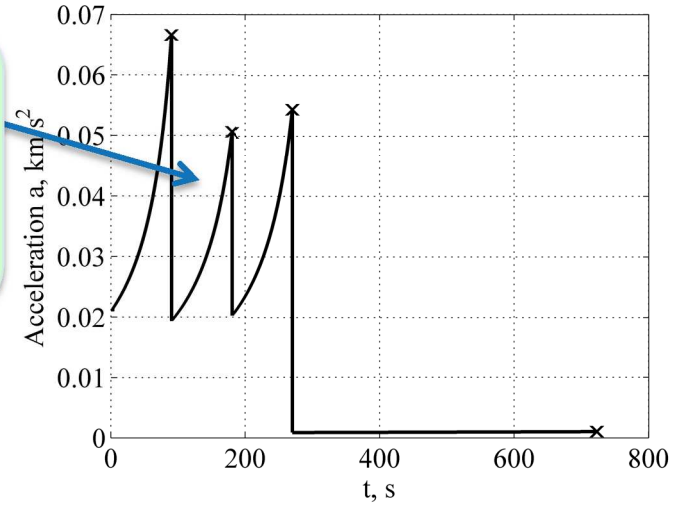




Short term coast phases not considered



No trajectory constraints are imposed on acceleration or dynamic pressure



UPT Example: RLV Ascent Dynamics

STEP 0 ➡ Problem variables declaration:

```
%**************************************
%     PHASE A
%**************************************
%% Initial state (ECEF J2000). Units: KM, KM/S
state  = [  6378.136600000000; -0.9210975127400074; 0.9138543963442310E-01;
            0.0000000000000000; 1.570778873502377; 0.0000000000000000 ]';

%% Time intervals [et]
t0A = 4.80427267185626e+08;
tfA = t01 + 90.0;

%% Fixed parameters
H        = 7.2;              % scale height [km]
area     = 1.5^2*pi*1e-6;    % reference area [km^2]
A_TARGET = 7078.1366;        % semi-major axis of the target orbit [km]

massA    = 128013;           % s/c mass at the beginning of the phase A [kg]
dotmassA = 975.167;          % propellant consumption during phase A [kg/s]

%% Nominal uncertain & control/guidance parameters
p0A = [0.2; 0.5; 1.225*1e9; 2.678e3]';          % model parameters [CL, CD, ρ0, T]
r0A = [90*pi/180; -1.9*pi/180; 16.03*pi/180]';  % control/guidance parameters [iT, ΔΨ, ΔΘ]

%% Uncertainties
sigma_pA = [p0A(1)*0.01, p0A(2)*0.01, p0A(3)*0.01, p0A(4)*1e-4]; % [σCL, σCD, σρ0, σT]
sigma_rA = [r0A(1)*0.01, r0A(2)*0.01, r0A(3)*0.01];              % [σiT, σΔΨ, σΔΘ]
```

UPT Example: RLV Ascent Dynamics

STEP 0 ➡ Problem variables declaration:

```
%***************************************
%     PHASE B
%***************************************
%% Time intervals [et]
t0B = tfA;
tfB = t0B + 90.0;

%% Fixed parameters
massB    = 31709;           % s/c mass at the beginning of the phase B [kg]
dotmassB = 217.545;         % propellant consumption during phase B [kg/s]

%% Nominal uncertain parameters
p0B = [0.2; 0.5; 1.225*1e9; 6.133e2]'; % model parameters [CL, CD, ρ0, T]

%% Uncertainties
sigma_pB = [p0B(1)*0.01, p0B(2)*0.01, p0B(3)*0.01, p0B(4)*1e-3]; % [σ_CL, σ_CD, σ_ρ0, σ_T]

%***************************************
%     PHASE C
%***************************************
...

%***************************************
%     PHASE D
%***************************************
...
```

Definition of parameters for phases C and D

UPT Example: RLV Ascent Dynamics

STEP 0 ➡ Problem variables declaration:

```
%**************************************
%     Generation of samples
%**************************************
nsamples = 1e5;

% PHASE A
x0A_distr = repmat(state, nsamples, 1); % dummy distribution (fixed initial state)

samplesA  = mvnrnd([p0A, r0A], ...
                   diag([sigma_pA.^2, sigma_rA.^2]), ...
                   nsamples);

p0A_distr = samplesA(:,1:4);            % distribution of model uncertain parameters
r0A_distr = samplesA(:,5:7);            % distribution of control/guidance uncertain parameters

% PHASE B
p0B_distr = mvnrnd(p0B, diag(sigma_pB.^2), nsamples); % distribution of model uncertain
                                                        parameters

% PHASE C
...
% PHASE D
...
```

UPT Example: RLV Ascent Dynamics

STEP 1 ➡ Model declaration:

Performed using the *UPTmodel* function:

```
% create model
model_PHA = UPTmodel('Model', 'ASCENT', 'MainAttractor','EARTH',...
                     'InitialState', state, 'InitialEpoch', t0A, 'FinalEpoch', tfA, ...
                     'LengthUnits', 'KM', 'TimeUnits', 'SEC', 'AngleUnits', 'RAD', ...
                     'Coordinate','LOCAL', 'Frame', 'IAU_EARTH', 'FrameCenter', 'EARTH', ...
                     'CL', p0A(1), 'CD', p0A(2), 'REFERENCE_DENSITY', p0A(3),...
                     'TMAX', p0A(4), 'MASS',massA, 'MASSRATE',dotmassA,...
                     'SCALE_HEIGHT',H, 'AREA',area, 'A_TARGET',A_TARGET,...
                     'Control','PITCH_PUSHOVER', 'INC_TARGET',r0A(1),...
                     'DYAW',r0A(2), 'DTHETA',r0A(3), ...
                     'ACCELERATION_TIME',5.0, 'PITCHOVER_ROTATION',14.5, 'PITCHOVER_TC',17.0,...
                     'Tolerance',1e-9);
```

Control/guidance model

UPT Example: RLV Ascent Dynamics

STEP 2 ➡ Method declaration:

Performed using the *UPTmethod* function:

```
% Masking vectors
a_xA = [0 0 0 0 0 0];
a_pA = [1 1 1 1];
a_rA = [1 1 1];

% create method
method_PHA = UPTmethod('Method','DAMC',...
                       'Samples',1e6,...
                       'Order', 5,...
                       'UncertainStates', a_xA,...
                       'CovarianceMatrix', diag(ones(1,6)),...
                       'UncertainParam', a_pA,...
                       'Sigma', sigma_pA,...
                       'UncertainCont', a_rA, ...
                       'SigmaCont', sigma_rA);
```

Fixed initial state

UPT Example: RLV Ascent Dynamics

STEP 3 ➡ Run UPT:

Performed using the *UPTrun* function.

```
% Run simulation
[UPToutput_PHA, UPTinput_PHA] = UPTrun( 'Model', model_PHA, 'Method', method_PHA );
```

STEP 4 ➡ Analysis of additional samples:

Use the *UPTeval* function to evaluate the polynomial with the distribution generated at the beginning:

```
% Evaluation with generated samples
[ xfA_distr, ~, ~ ] = UPTeval( UPToutput_PHA, x0A_distr', p0A_distr', r0A_distr' );

mean_fA = mean(xfA_distr,2); % compute mean
cov_fA = cov(xfA_distr');    % compute covariance
```

UPT Example: RLV Ascent Dynamics

For subsequent phase B:

```
% Masking vectors
a_xB = [1 1 1 1 1 1];
a_pB = [1 1 1 1];

% create model
model_PHB = UPTmodel('Model', 'ASCENT', 'MainAttractor','EARTH',...
                     'InitialState', UPToutput_PHA.xf_nominal',...
                     'InitialEpoch', t0B, 'FinalEpoch', tfB,...
                     'LengthUnits', 'KM', 'TimeUnits', 'SEC', 'AngleUnits', 'RAD', ...
                     'Coordinate','LOCAL', 'Frame', 'IAU_EARTH', 'FrameCenter', 'EARTH', ...
                     'CL', p0B(1), 'CD', p0B(2), 'REFERENCE_DENSITY', p0B(3),...
                     'TMAX', p0B(4), 'MASS',massB, 'MASSRATE',dotmassB,...
                     'SCALE_HEIGHT',H, 'AREA',area, 'A_TARGET',A_TARGET,...
                     'Control','GRAVITY_TURN', 'Tolerance',1e-9);

% create method
method_PHB = UPTmethod('Method','DAMC', 'Samples',1e6, 'Order', 5,...
                       'UncertainStates', a_xB, 'CovarianceMatrix', cov_fA,...
                       'UncertainParam', a_pB, 'Sigma', sigma_pB);

% Run simulation
[UPToutput_PHB, UPTinput_PHB] = UPTrun( 'Model', model_PHB, 'Method', method_PHB );

% Evaluation with samples obtained at the end of phase A
[ xfB_distr, ~, ~ ] = UPTeval( UPToutput_PHB, xfA_distr, p0B_distr' );
```

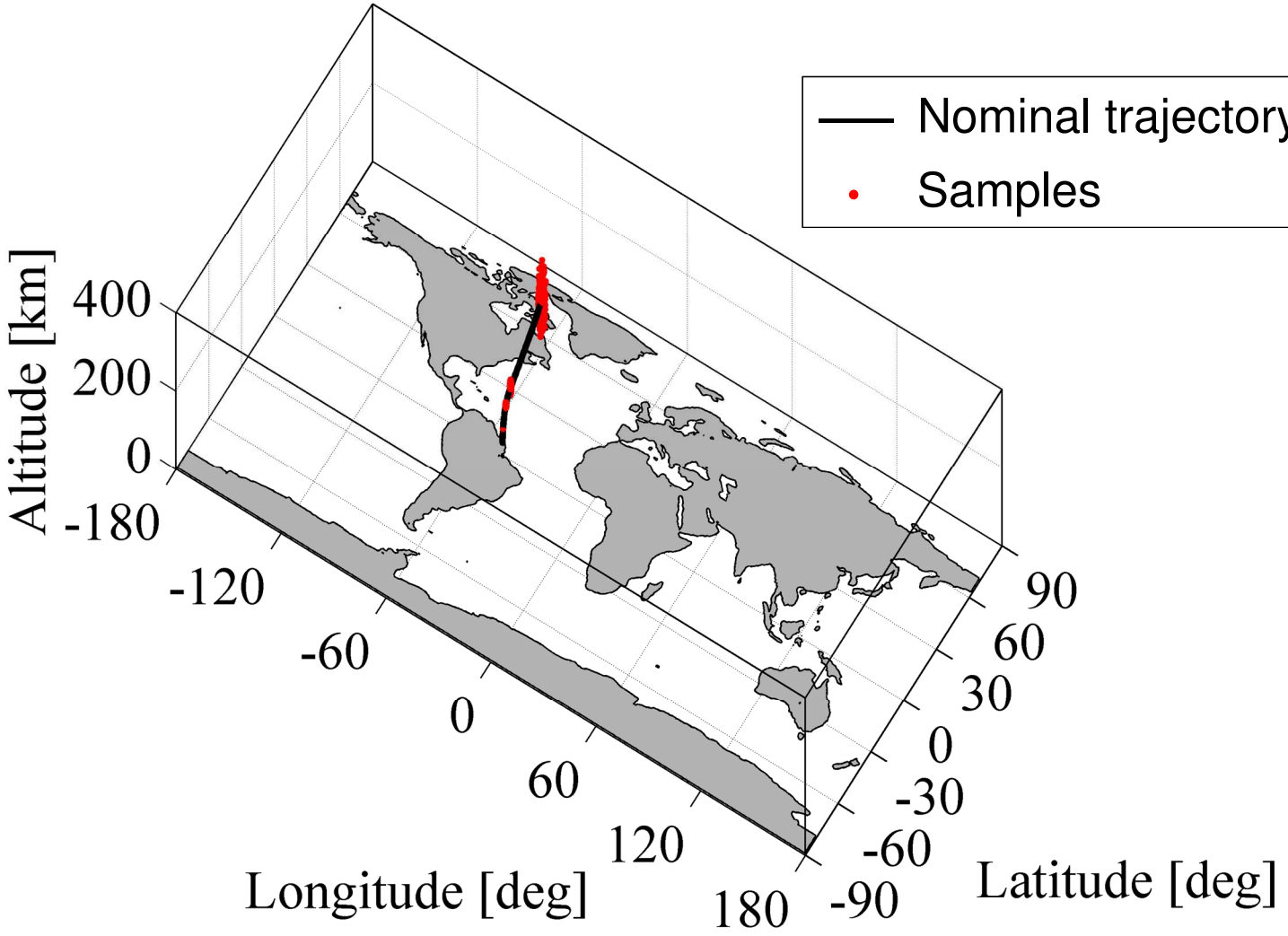
Resulting from phase A

New call to UPTrun

Call to UPTeval

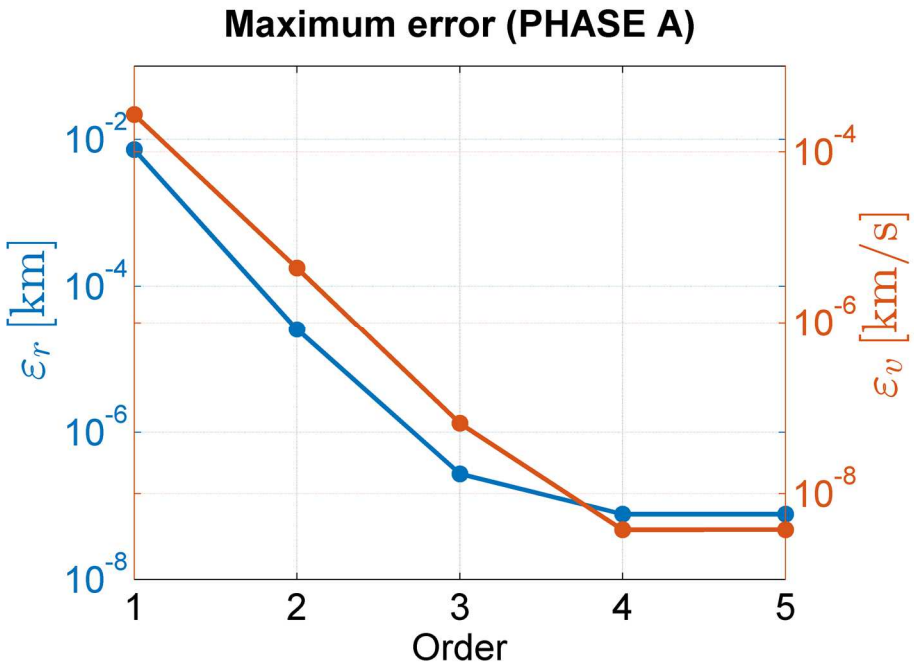
UPT Example: RLV Ascent Dynamics

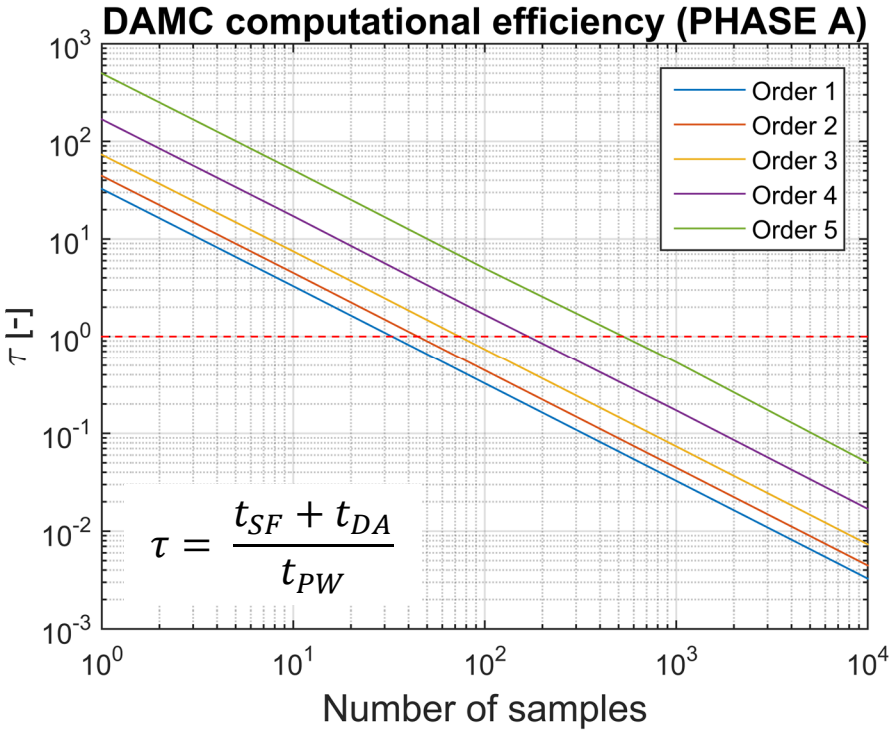
Results



DAMC Performance

- Comparison standard MC and DA-based MC (PHASE A)

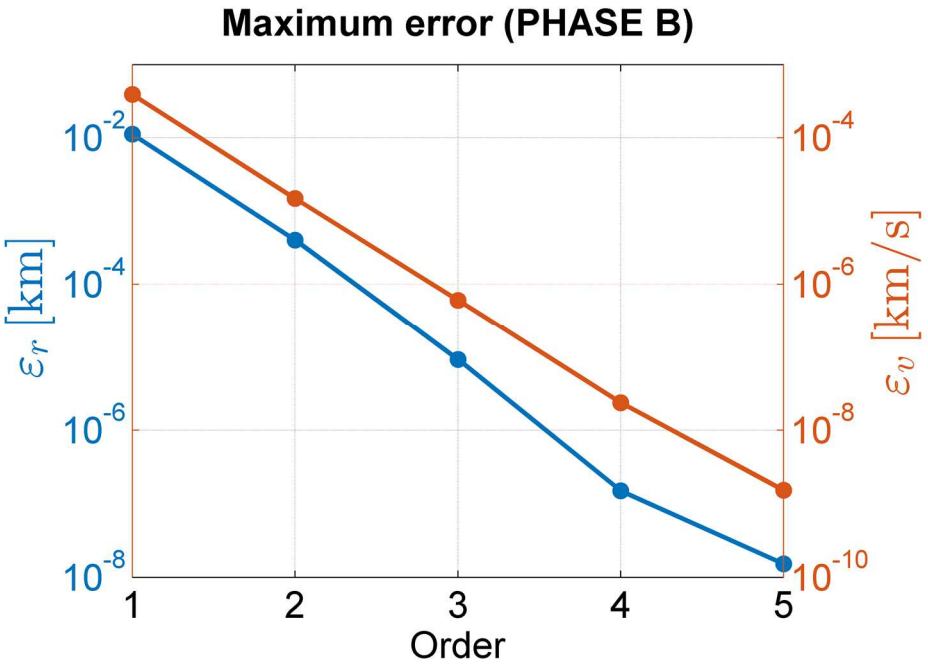


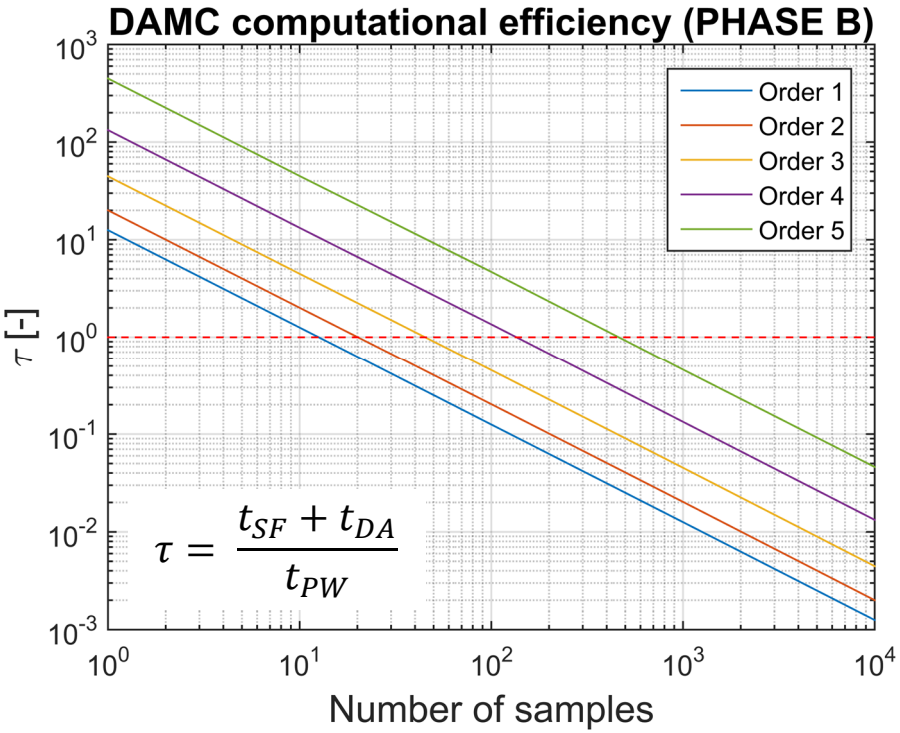


With 3rd order polynomials, the break-even point is reached with 70 samples!

DAMC Performance

- Comparison standard MC and DA-based MC (PHASE B)

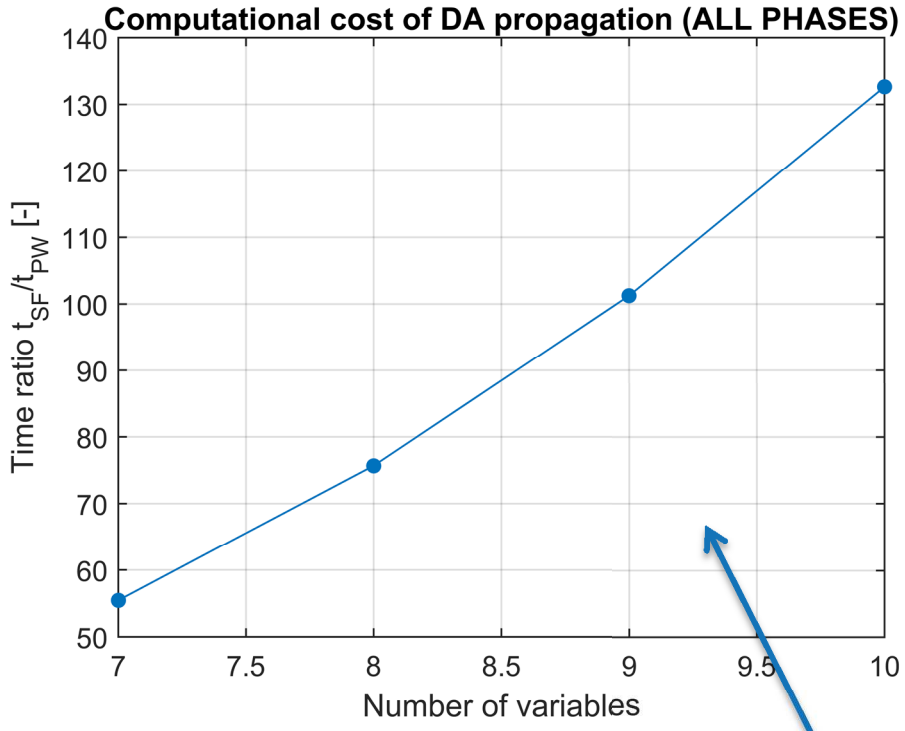


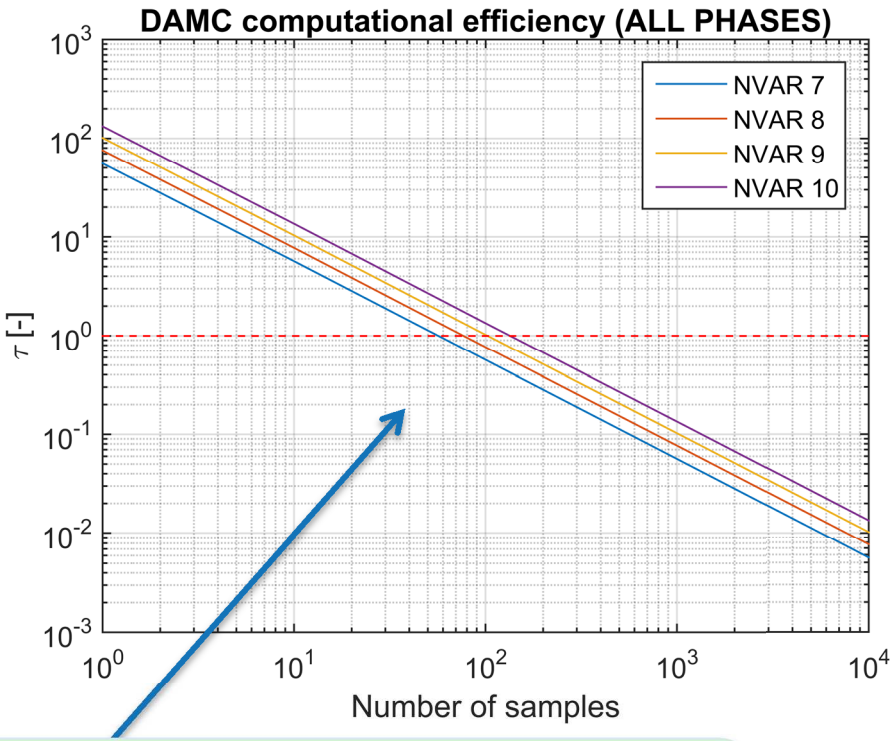


Using a 4th order polynomial, the break-even point is reached with 150 samples!

DAMC Performance

- DAMC computational efficiency vs number of variables

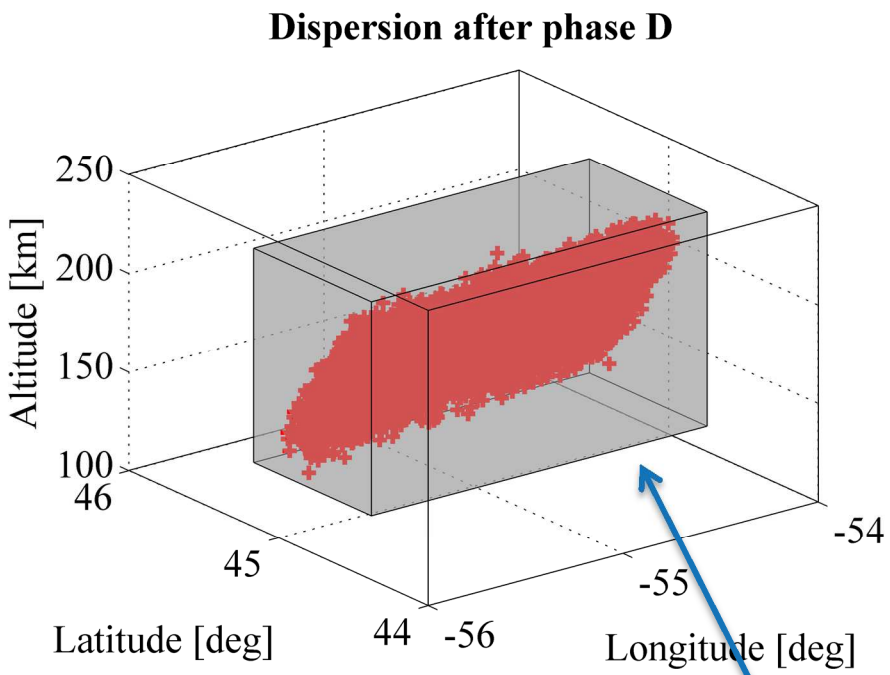


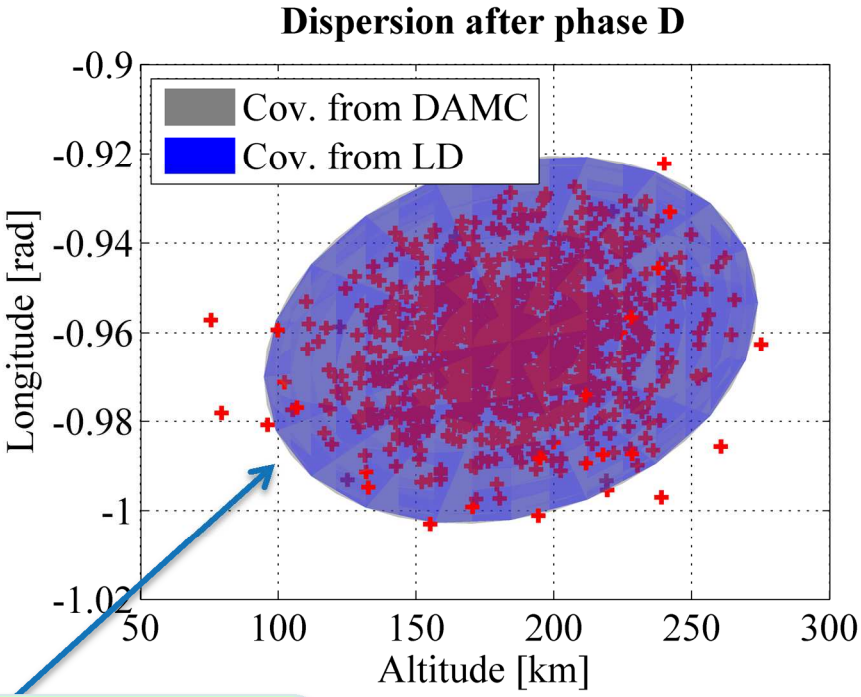


Computational burden increases not only with the polynomial order, but also with the number of variables.

Additional Analyses

- Polynomial Bounder (PB) and Linearized Dynamics (LD) methods can be used as well





Dispersion at orbit insertion

Conclusions (1/2)

- Two products have been implemented:
  - DACE: Efficient computation of Taylor expansions of arbitrary order
  - DAST: DA-based uncertainty propagations in astrodynamics
- For what concerns DAST:
  - Propagations can be run on several ready-to-use dynamical models and any DA-compatible custom dynamical model
  - Three different uncertainty propagation techniques:
    - DA-based Monte Carlo
    - Ranging estimation using a polynomial bounder
    - Linear covariance propagation

Conclusions (2/2)

- Main advantages:
  - More efficient than standard Monte Carlo for typical number of samples
  - No need of deriving and using variational equations
  - Obtained polynomials can be used to propagate different statistics

- Main consideration:
  - Uncertainty propagation is based on a Taylor approximation of the mapping function

  - Size of uncertainty set and order shall guarantee sufficient accuracy
  - Computational time increases for higher order and large number of variables.

Differential Algebra Space Toolbox (DAST)

Workshop on Nonlinear Propagation of Uncertainties using Differential Algebra

September 22nd, 2015 *Escape Dance Room*, *ESTEC*

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