Usage of Automatic Differentiation in Some Practical Problems of Celestial Mechanics

Dmitry Pavlov

Laboratory of Ephemeris Astronomy
Institute of Applied Astronomy of the Russian Academy of Sciences
St. Petersburg, Russia

April 21, 2018

Problem formulation: propagation of uncertainty

Solar system bodies are modeled with a differential equation:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t))$$

(Actually it is more complex, but let us leave it at that for now.)

There are:

- Initial conditions: $\mathbf{x}(t_0) = \mathbf{x}_0$
- Parameters $\{p_i\}$ (masses, gravity field coefficients, etc).

IC and parameters are determined from astronomical observations and have their uncertainties and correlations.

Let $\mathbf{P} = (\mathbf{x}_0^{(1)}, \ldots, \mathbf{x}_0^{(n)}, p_1, \ldots, p_m)$.

Covariance matrix $\mathrm{cov}(\mathbf{P})$ is estimated from observations.

How do we estimate $\mathrm{cov}(\mathbf{x}(t))$?

Linearization

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}\mathbf{P}} = \begin{bmatrix} \frac{\mathrm{d}x^{(1)}}{\mathrm{d}x_0^{(1)}} & \cdots & \frac{\mathrm{d}x^{(1)}}{\mathrm{d}x_0^{(n)}} & \frac{\mathrm{d}x^{(1)}}{\mathrm{d}p_1} & \cdots & \frac{\mathrm{d}x^{(1)}}{\mathrm{d}p_m} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \frac{\mathrm{d}x^{(n)}}{\mathrm{d}x_0^{(1)}} & \cdots & \frac{\mathrm{d}x^{(n)}}{\mathrm{d}x_0^{(n)}} & \frac{\mathrm{d}x^{(n)}}{\mathrm{d}p_1} & \cdots & \frac{\mathrm{d}x^{(n)}}{\mathrm{d}p_m} \end{bmatrix}$$

$$\mathrm{cov}(\mathbf{x}) \approx \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}\mathbf{P}}\ \mathrm{cov}(\mathbf{P})\ \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}\mathbf{P}}^T$$

$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}\mathbf{P}}$ (isochronous derivative) is now part of the dynamic equation:

$$\left(\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}\mathbf{P}}\right)^{\cdot} = \frac{\mathrm{d}\mathbf{f}}{\mathrm{d}\mathbf{P}} = \frac{\partial \mathbf{f}}{\partial \mathbf{x}} \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}\mathbf{P}}$$

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}\mathbf{P}}(t_0) = \begin{bmatrix} 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 1 & 0 & \cdots & 0 \end{bmatrix}$$

Differentiation

How to calculate $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}$?

- *Numericalally* (can as well calculate $\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}\mathbf{P}}$ directly)
  Most simple, but suffers from numerical noise and requires extra computing time. (Though calculation of $\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}\mathbf{P}}$ can be run in parallel and the results can be reused).
- *Symbolically*
  Most precise, but requires much time and memory to expand expressions and can be inefficent. Requires CAS software. Hardly compatible with the way programs are written (conditions, loops etc).
- *Automatically*
  Most convenient for complex programs. Uses a minimal set of "primitive" derivatives plus the chain rule. Requires code analysis tools or programming language support.

Automatic differentiation with dual numbers

The idea: to each value x in the program, we attach an x'. Instead of arithmetics on x we have arithmetics on $(x, x + x'\varepsilon)$. $\varepsilon^2 = 0$. Some examples borrowed online:

$$\langle u, u' \rangle + \langle v, v' \rangle = \langle u + v, u' + v' \rangle$$

$$\langle u, u' \rangle * \langle v, v' \rangle = \langle uv, u'v + uv' \rangle$$

$$\langle u, u' \rangle / \langle v, v' \rangle = \left\langle \frac{u}{v}, \frac{u'v - uv'}{v^2} \right\rangle \quad (u \neq 0)$$

$$\sin \langle u, u' \rangle = \langle \sin(u), u' \cos(u) \rangle$$

$$\langle u, u' \rangle^k = \left\langle u^k, ku^{k-1}u' \right\rangle \quad (u \neq 0)$$

Dual number are easily extendable to multiple derivatives: from dual numbers to triple numbers to $(1 + m + n)$-numbers.

Case for a domain-specific language

In order to be usable for the task, program code must

- Be restrictive enough (no fancy memory access, no conditions involving state variables
- Nevertheless be expressive enough to define complex systems (loops, conditions involving non-state variables)
- Not have external calls (or have them properly annotated)
- Have information about what state variables are isochronous derivatives
- Provide interface to be called from a generic numerical integrator as a “black box”

The most convenient choice is to create a DSL.

LANDAU: LANguage for Dynamical systems with AUtomatic differentiation

- Explicitly typed; has reals, integers, and arrays of integers or reals
- Functions, loops, conditions
- Global constants, function arguments (immutable), and temporary variables (mutable)
- "Parameters": functions do not have them directly as arguments, but have derivatives w.r.t them in the state vector
- Annotations for derivatives in the state vector (input) and returned vector (output)
- Option to manually discard automatic derivatives where they are not significant

Implemented in Racket platform (Felleisen et al. 2018).
Currently LANDAU compiles to Racket, C code generation is planned.

Code example (part 1)

```
const int N = 3 # Sun, Earth-Moon barycenter, Moon (geocentric)

# Annotated parameters: geocentric Moon initial position and velocity
parameter initial[6]

real[N * 6 + 36 + N * 6] xdot (
  real[N * 6 + 36 + N * 6] x, # N-body state + derivatives
  real[N] GM)                 # masses
{
  # annotating derivatives in the state vector
  x[12:18] ' initial[0 : 6] = x[N * 6      : N * 6 + 36]
  x[12:18] ' GM[2]          = x[N * 6 + 36 : N * 6 + 36 + N * 6]

  real[N * 6] xb, xdotb # like x and xdot, but barycentric

  # Neglect the effect of the Moon's orbit and mass to the Sun
  discard xdotb[3 : 6] ' initial[0 : 6]
  discard xdotb[3 : 6] ' GM[2]

  xb[0 : 6] = x[0 : 6] # Sun is already barycentric

  # Transfer time derivatives from x to their xdot counterparts
  for j = [0 : N]
    xdot[j * 6 : j * 6 + 3] = x[j * 6 + 3: j * 6 + 3 + 6]
```

Code example (part 2)

```
  for k = [0 : 6] {  # Calculate barycentric Earth and Moon
    xb[6 + k] = x[6 + k] - x[12 + k] * 1 / (1 + GM[1] / GM[2])
    xb[12 + k] = xb[6 + k] + x[12 + k]
  }
  for i = [0 : N]   # Apply Newtonian laws
    for j = [0 : N]
      if i != j {
        real dist2 = sqr(xb[6 * i] - xb[6 * j]) +
                     sqr(xb[6 * i + 1] - xb[6 * j + 1]) +
                     sqr(xb[6 * i + 2] - xb[6 * j + 2])
        real dist3inv = 1 / (dist2 * sqrt(dist2))

        for k = [0 : 3]
          xdotb[6*i + 3 + k] += GM[j] * (xb[6*j + k] - xb[6*i + k]) * dist3inv
      }
  xdot[0:6] = xdotb[0:6] # Back to EMB + geocentric Moon coordinates for xdot
  for k = [0 : 6] {
    xdot[12+k] = xdotb[12+k] - xdotb[6+k]
    xdot[6+k]  = (xdotb[6+k] * GM[1] / GM[2] + xdotb[12+k]) / (1 + GM[1] / GM[2])
  }
  # Annotating the parameter derivatives in xdot
  xdot[N * 6      : N * 6 + 36]         = xdot[12:18] ' initial[0 : 6]
  xdot[N * 6 + 36 : N * 6 + 36 + N * 6] = xdot[12:18] ' GM[2]
}
```

Summary of the proposed solution

- Single run of Adams–Bashforth–Moulton numerical integrator for the orbits and isochronous derivatives
- Equations of motion (right-hand side for the integrator) coded in LANDAU and compiled by Racket with automatic differentiation via dual numbers
- Propagation of uncertainty by linearized estimation from the initial covariance matrix

No polynomial algebra yet.

Estimation of Moon's orbit accuracy

- From linear approximation

$$\sigma(x_i)^2 = \sum_{j=1}^{n+m} \sum_{k=1}^{n+m} \frac{\mathrm{d}x_i}{\mathrm{d}P_j} \frac{\mathrm{d}x_i}{\mathrm{d}P_k} \mathrm{cov}(\mathbf{P})_{jk}$$

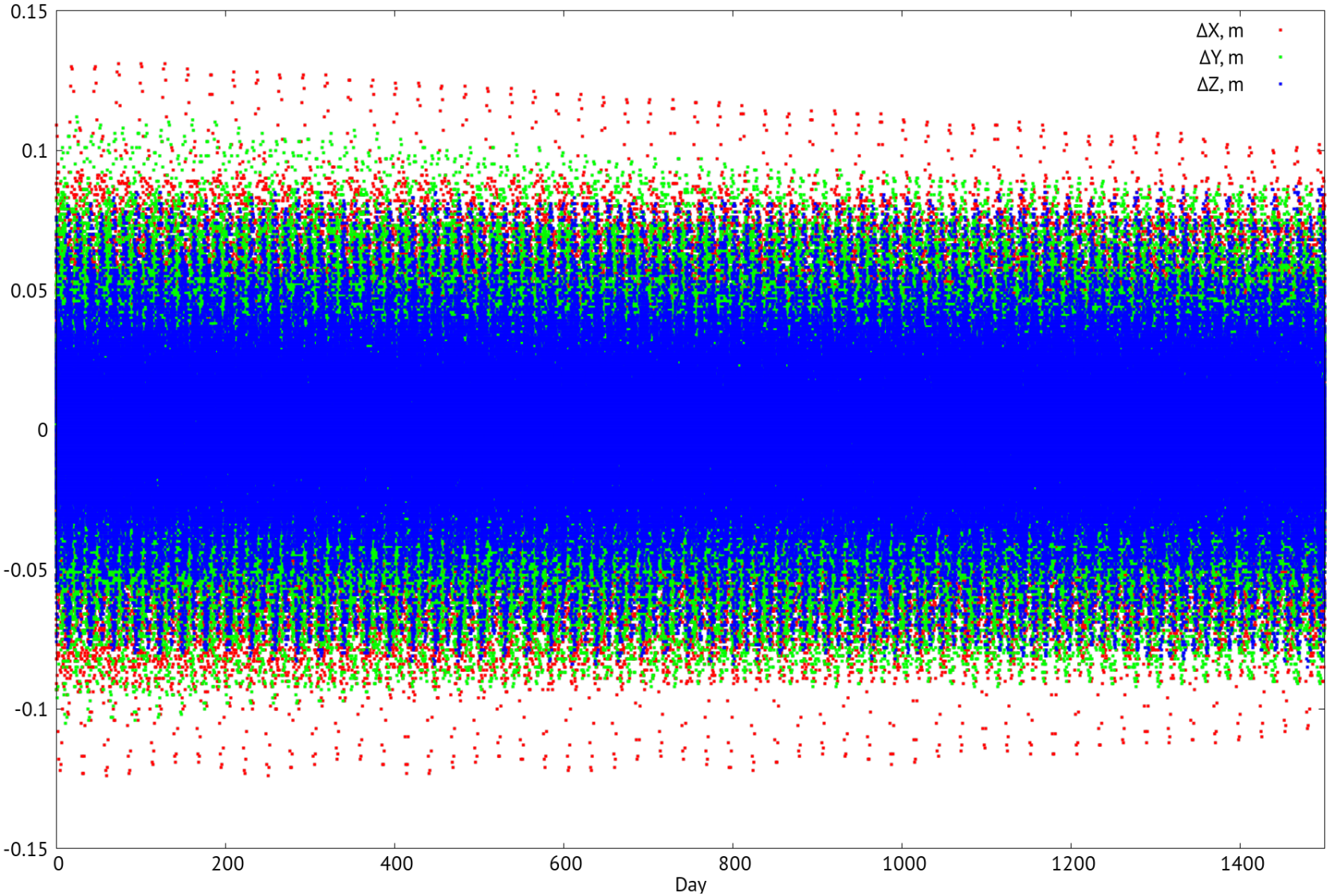
- From Monte-Carlo simulation

  Cholesky decomposition: $\mathrm{cov}(\mathbf{P}) = L^T L$, where $L_{(n+m)\times(n+m)}$ is a lower triangular matrix

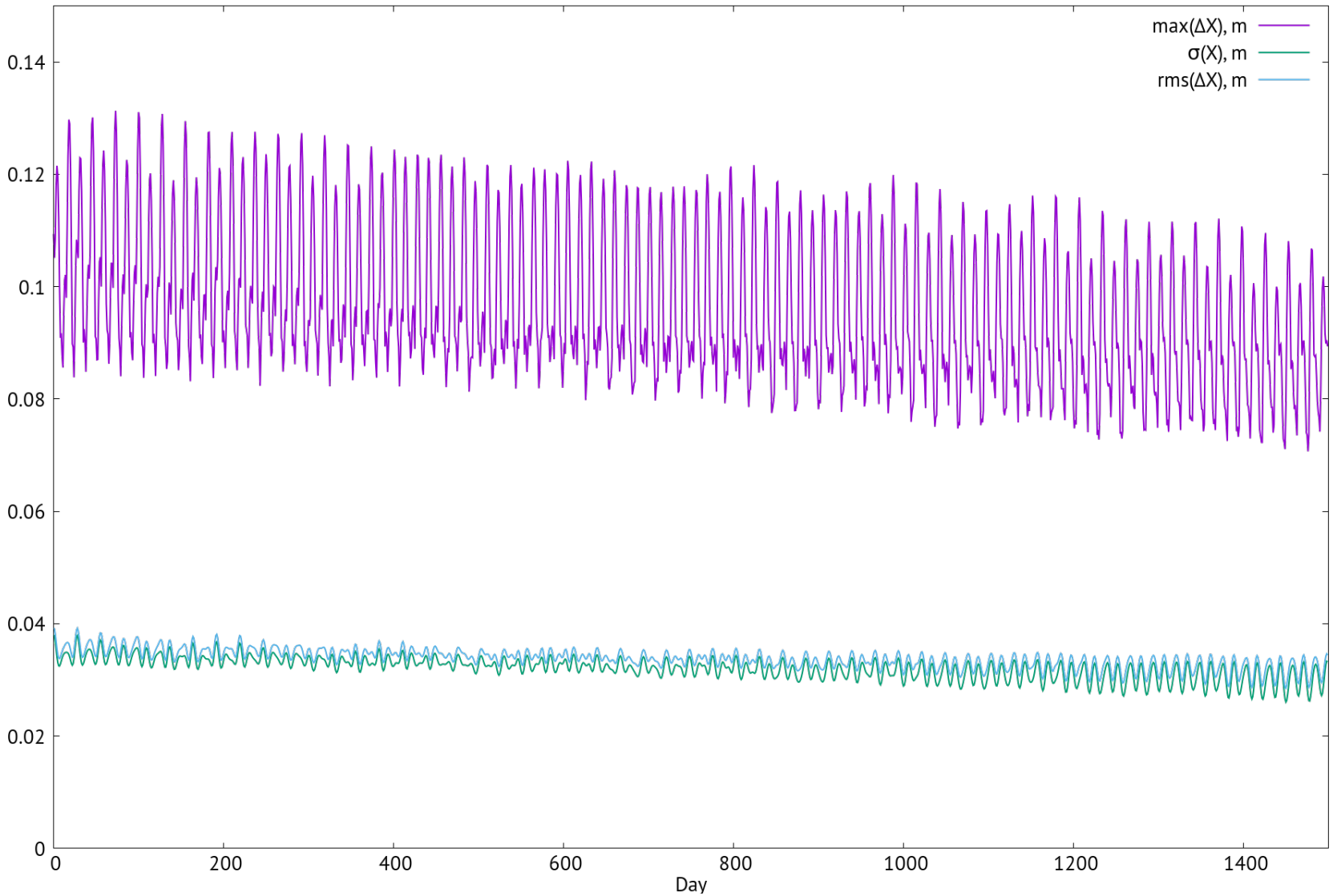
  Sample $\mathbf{x}_{\mathrm{random}} = L^T \mathbf{y}$, where y_i $(1 \le i \le n+m)$ are uncorrelated $\mathcal{N}(0,1)$ random variables.

  In the following test,
  $\mathbf{P} = \{x_0, y_0, z_0, \dot{x}_0, \dot{y}_0, \dot{z}_0, \omega_{z0}, \beta, \gamma, k_v/c_T, \tau, \tau_{R1}, \tau_{R2}\}$

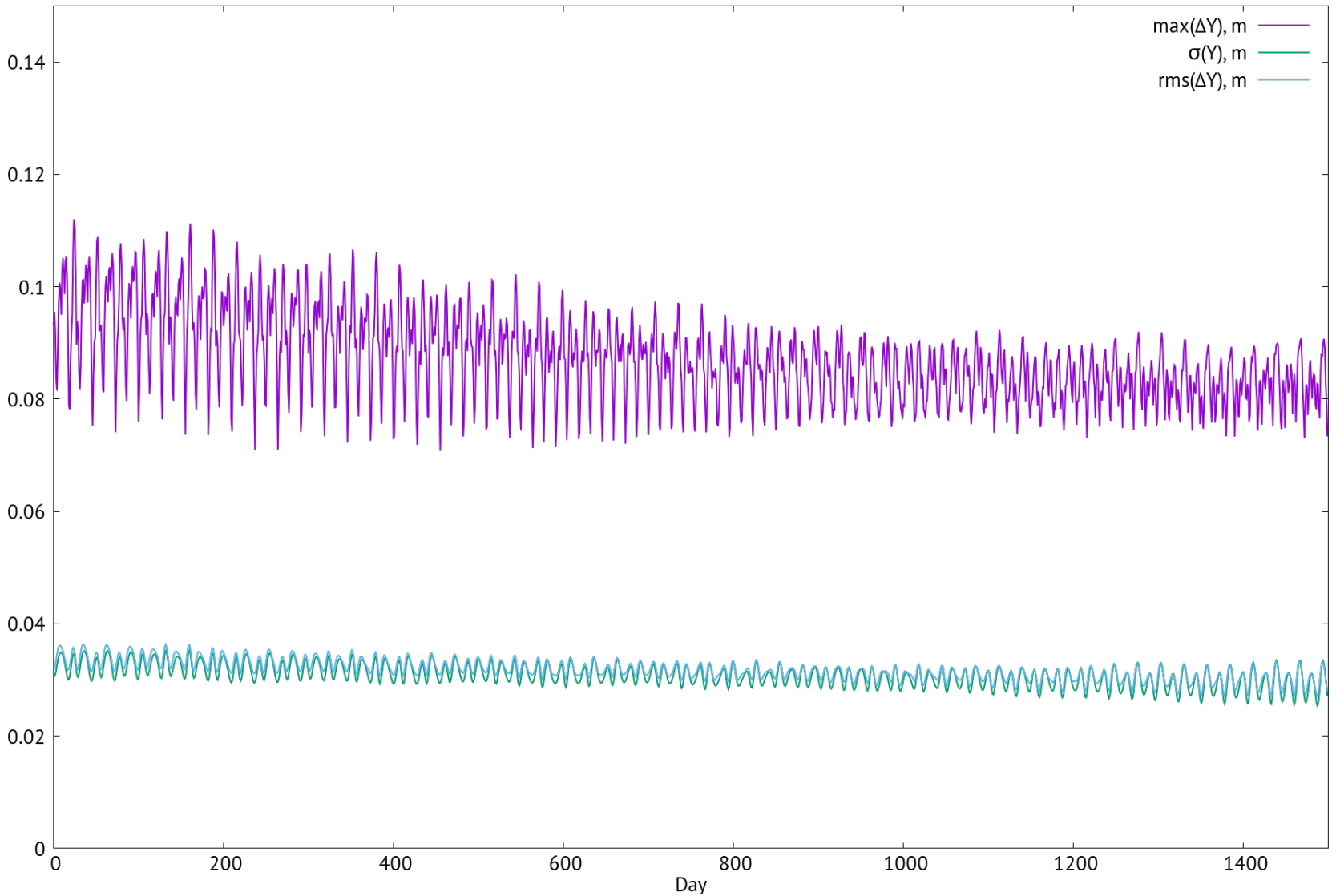
Uncertainty propagation by Monte-Carlo



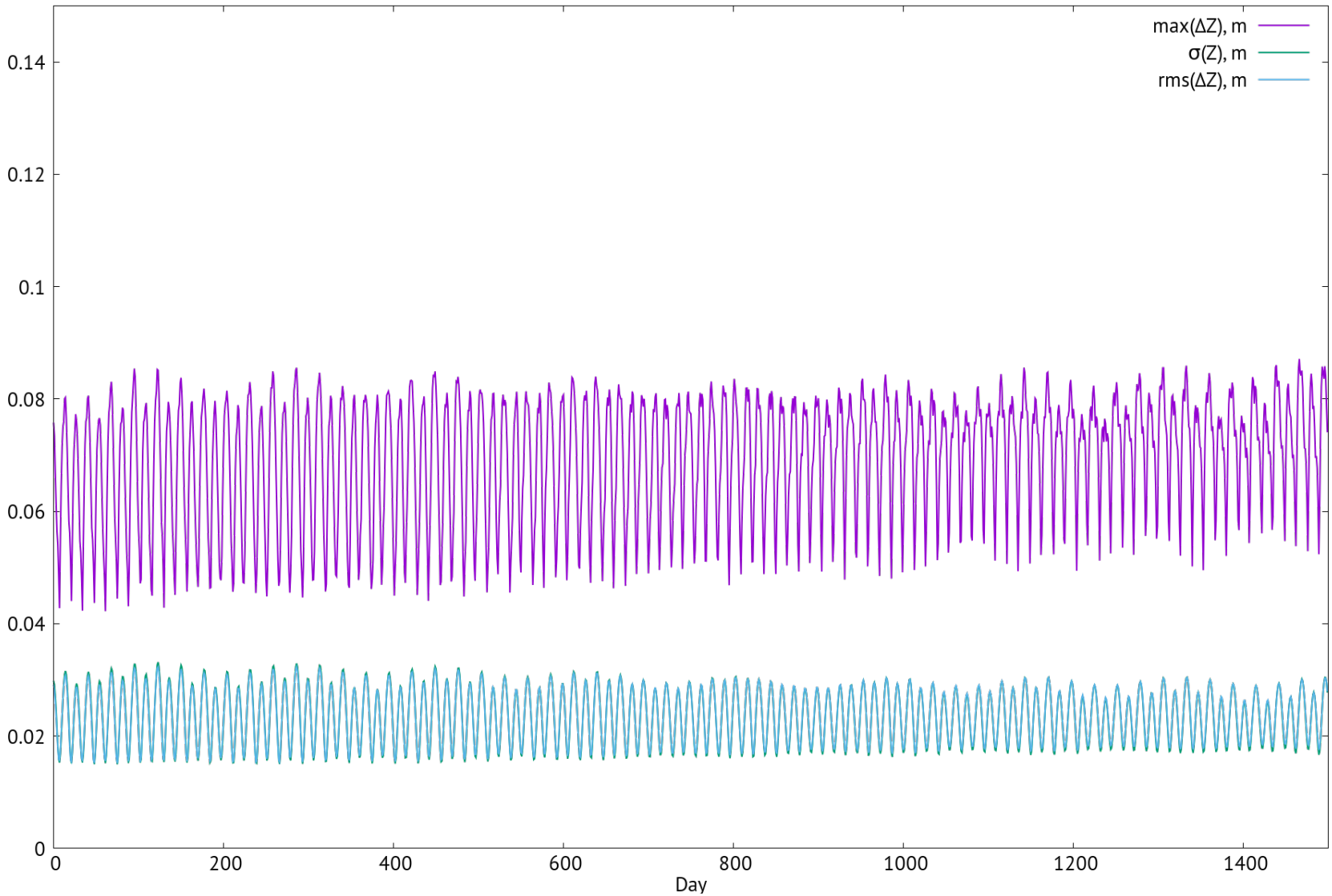
Comparison: X coordinate of the Moon



Comparison: Y coordinate of the Moon



Comparison: Z coordinate of the Moon



Other approaches

Taylor method of numerical integration (Jorba and Zou, 2004)

- Approximates $x(t)$ as a Taylor expansion
- Heavily relies on automatic differentiation and code generation
- Taylor expansion can be used to propagate uncertainty
- Requires higher order derivatives, so dual numbers will not work

Jet transport (Pérez-Palau, Masdemont, Gomez, 2013)

Chebyshev polynomial algebra (Riccardi, Tardioli, Vasile, 2015)