Landau: средство трансляции уравнений в программный код с применением автоматического дифференцирования

Иван Долгаков и Дмитрий Павлов

Лаборатория эфемеридной астрономии

5 декабря 2019

Задача

Динамическая система:

$$\begin{aligned} \dot{\mathbf{x}}(t) &= f(t, \mathbf{x}(t)), \\ \mathbf{x}(t_0) &= \mathbf{x}_0, \end{aligned}$$

Найти $\mathbf{x_0} = (x_0^{(0)}, \cdots, x_0^{(k)})$:

$$\sum_{i=1}^{i=N} \Big[\omega_i \big(\rho_{\mathrm{obs}i} - \rho_i(\mathbf{x}(t_i)) \big) \Big]^2 \to \min$$

ρ_i – функции редукции

$\rho_{\mathrm{obs}i}, \omega_i$ – наблюдаемая величина, вес

Для МНК нужно знать $\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}\mathbf{x_0}}$

Решение 1. Конечные разности

1. Проинтегрировать с известным $\mathbf{x_0}$

$$\mathbf{x}(T) = \texttt{integrate}(t_0, \mathbf{x_0}, f, T)$$

2. Проинтегрировать с k различными начальными условиями

$$\boldsymbol{\Delta}_j \in [\boldsymbol{\Delta}_1, ..., \boldsymbol{\Delta}_k]$$

$$\mathbf{x}'_j(T) = \texttt{integrate}(t_0, x_0 + \boldsymbol{\Delta}_j, f, T)$$

3.

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}\mathbf{x_0}^{(j)}} = \frac{\mathbf{x'}_j - \mathbf{x}}{h} + O(h)$$

Решение 2. Интегрирование изохронных производных

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}\mathbf{x_0}} = \frac{\mathrm{d}f}{\mathrm{d}\mathbf{x_0}} = \frac{\mathrm{d}f}{\mathrm{d}\mathbf{x}}\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}\mathbf{x_0}},$$

Новая система относительно $\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}\mathbf{x_0}}$:

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}\mathbf{x_0}}\right) &= \frac{\mathrm{d}f}{\mathrm{d}\mathbf{x}}\left(\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}\mathbf{x_0}}\right)\\ \frac{\mathrm{d}x_i}{\mathrm{d}x_0^{(j)}}(t_0) &= \delta_{ij}\end{aligned}$$

Можно интегрировать вместе с $\mathbf{x}$

Формализация задачи

$$\frac{\mathrm{d}x_i}{\mathrm{d}x_0^{(j)}} = J_{ij} \quad \text{(якобиан, полученный извне)}$$

$$f_i = 2\sin(x_i) + x_i$$

$$\frac{\mathrm{d}f_i}{\mathrm{d}x_0^{(j)}} = ?$$

```
parameter[k] x0
real[k * k] func(real[k] x, real[k * k] dxdx0){
  x[:] ' x0[:] = dxdx0[:]           # Annotation
  real[k] f = 2 * sin(x[:]) + x[:] # Body
  func[:] = f[:] ' x0[:]           # Write derivatives
}
```

Требования к средству задания уравнений

Тело функции может включать:

- переменные и константы,
- циклы,
- конструкции ветвления,
- вызовы функций.

LANDAU: LANguage for Dynamical systems with AUtomatic differentiation

- Дифференцируемый,
- статически типизированный,
- предсказуемый.

Автоматическое дифференцирование

Чуть более сложная функция:

```
real[k] g, f
g[:] = 2 * sin(x[:]) + x[:]
for i = [0 : k]
  f[i] = 2 * g[i] + g[i] * x[i]
```

Производные:

```
dgdx0[..] = 2 * cos(x[..]) * dxdx0[..] + dxdx0[..]
g[:] = 2 * sin(x[:]) + x[:]
for i = [0 : k]
  dfdx0[..] = 2 * dgdx0[..]
                + dgdx0[..] * x[i] + g[i] * dxdx0[..]
  f[i] = 2 * g[i] + g[i] * x[i]
```

Использование ранее полученных результатов вычислений

Реализация

Транслятор Landau написан на Racket

Синтаксический анализатор:

- Отслеживание зависимостей:
  - Вычисление только необходимых производных.
- Проверка корректности:
  - Проверка выхода за пределы массива:

```
real[4] x
for i = [0 : 5]
  x[i] = 1. # Ошибка на этапе компиляции
  ~~^   index 4 out of range [0, 3]
```

Кодогенератор:

- Генерация Racket, C
- Опциональное использование 80-битных чисел (расширенной точности)

Преимущества использования

- Быстрота разработки:
  - Высокий уровень, скорость C
  - Автоматическая проверки корректности
  - Простота отладки
  - Привычный синтаксис (Python, C)
- Простота поддержки кода
  - Производные генерируются компилятором
- Портируемость
  - Сгенерированный C код гарантированно работает на Windows, Linux, MacOS

Задача о спутниках Юпитера

- 4 спутника Юпитера
- Начальные положения и скорости каждого влияют на траекторию всех

Найти начальные положения и скорости

```
#lang landau

const int NSAT = 4
const int NPERTURB = 4
const int NGRAV = 7 # -, -, C20, -, C40, -, C60
const int DIM = NSAT * 6 + NSAT * 6 * NSAT * 6

const real reference_radius = 71492 / au_km # Опорный радиус модели гравитационного поля Юпитера

const real pi = 3.14159265358979323846

# Параметры вращения Юпитера
const real[2] pole_ra  = {268.056595, -0.006499}
const real[2] pole_dec = {64.495303 , 0.002413 }
const real[2] pm       = {284.95    , 870.536  }
const real[5] nut_prec_ra = {0.000117, 0.000938, 0.001432, 0.000030, 0.002150}
const real[5] nut_prec_dec = {0.000050, 0.000404, 0.000617, -0.000013, 0.000926}
const real[5] Jabcde_0 = { 99.360714, 175.895369, 300.323162, 114.012305, 49.511251 }
const real[5] Jabcde_T = { 4850.4046, 1191.9605, 262.5475, 6070.2476, 64.3000  }

const real au_km = 149597870.7
const real sec_day = 86400.0

# Jupiter gravity field estimated from the first two Juno orbits
# https://agupubs.onlinelibrary.wiley.com/doi/full/10.1002/2017GL073140
const real central_gm = 126686533.0 * sec_day * sec_day / (au_km * au_km * au_km)

const real[NGRAV] central_grav =
  { 0.0,
    0.0, -14696.514e-6 / sqrt(5.0),
    0.0,    586.623e-6 / sqrt(9.0),
    0.0,    -34.244e-6 / sqrt(13.0) }

# Начальные положения и скорости спутников
parameter[NSAT * 6] initial

# Матрица вращения Юпитера зависящая от времени
real[9] jupiter_rotation_matrix (real t)
{
  real T = t / 36525
  real alpha_0 = pole_ra[0]  + pole_ra[1] * T
  real delta_0 = pole_dec[0] + pole_dec[1] * T
  real W = (pm[0] + pm[1] * t) * pi / 180.0

  for i = [0 : 5]
  {
    real J = (Jabcde_0[i] + Jabcde_T[i] * T) * pi / 180.0
    alpha_0 += nut_prec_ra[i]  * sin(J)
    delta_0 += nut_prec_dec[i] * cos(J)
  }

  alpha_0 *= pi / 180.0
  delta_0 *= pi / 180.0

  # Rz(alpha + pi * 0.5) * Rx(pi * 0.5 - delta) * Rz(w)
  jupiter_rotation_matrix[0] = -sin(alpha_0) * cos(W) - cos(alpha_0) * sin(delta_0) * sin(W)
  jupiter_rotation_matrix[1] =  sin(alpha_0) * sin(W) - cos(alpha_0) * sin(delta_0) * cos(W)
  jupiter_rotation_matrix[2] =  cos(alpha_0) * cos(delta_0)
  jupiter_rotation_matrix[3] =  cos(alpha_0) * cos(W) - sin(alpha_0) * sin(delta_0) * sin(W)
  jupiter_rotation_matrix[4] = -cos(alpha_0) * sin(W) - sin(alpha_0) * sin(delta_0) * cos(W)
  jupiter_rotation_matrix[5] =  cos(delta_0) * sin(alpha_0)
  jupiter_rotation_matrix[6] =  cos(delta_0) * sin(W)
  jupiter_rotation_matrix[7] =  cos(delta_0) * cos(W)
  jupiter_rotation_matrix[8] =  sin(delta_0)
}

# Функция расчета производных системы Юпитера
real[DIM] jupsatsystem (real t, real[DIM] state_and_derivatives, real[3] central_pos,
                        real[NPERTURB] perturb_gms, real[3 * NPERTURB] perturb_pos,
                        real[NSAT] sat_gms)
{
  real[NSAT * 6] state = state_and_derivatives[:NSAT * 6]
  real[DIM - NSAT * 6] state_derivatives_initial = state_and_derivatives[NSAT * 6:]
  real[3 * NSAT] sat_acc
  real[3] central_acc

  # Аннотация производных
  # Спутники имеют производные по начальному состоянию каждого спутника
  for i = [0 : NSAT * 6]
    for j = [0 : NSAT * 6]
      state[i] ' initial[j] = state_derivatives_initial[i * NSAT * 6 + j]

  # Взаимодействие: Юпитер <-> спутники
  for i = [0 : NSAT]
  {
    real[3] r  = state[i * 6 : i * 6 + 3]
    real dist2 = (r r), dist3 = dist2 * sqrt(dist2)

    # a = GM \frac{r}{r^3}

    sat_acc[i * 3 : i * 3 + 3] = -central_gm * r[:] / dist3
    central_acc[:] += sat_gms[i] * r[:] / dist3
  }

  # Взаимодействие: Юпитер и спутники <- возмущающие тела
  for i = [0 : NPERTURB]
  {
    real[3] r = perturb_pos[i * 3 : i * 3 + 3] - central_pos[:]
    real dist2 = (r r)
    central_acc[:] += perturb_gms[i] * r[:] / dist2 / sqrt(dist2)

    for j = [0 : NSAT]
    {
      r[:] = perturb_pos[i * 3 : i * 3 + 3] - (state[j * 6 : j * 6 + 3] + central_pos[:])
      dist2 = (r r)
      sat_acc[j * 3 : j * 3 + 3] += perturb_gms[i] * r[:] / dist2 / sqrt(dist2)
    }
  }

  # Взаимодействие: спутники <-> спутники
  for i = [1 : NSAT]
    for j = [0 : i]
    {
      real[3] r = state[j * 6 : j * 6 + 3] - state[i * 6 : i * 6 + 3]
      real dist2 = (r r)

      sat_acc[i * 3 : i * 3 + 3] += sat_gms[j] * r[:] / dist2 / sqrt(dist2)
      sat_acc[j * 3 : j * 3 + 3] -= sat_gms[i] * r[:] / dist2 / sqrt(dist2)
    }

  # Вызов функции матрицы поворота Юпитера
  real[9] rot = jupiter_rotation_matrix(t)

  # Гравитационный потенциал, зональные гармоники
  for i = [0 : NSAT]
  {
    real x = state[i * 6 + 0] / reference_radius,
         y = state[i * 6 + 1] / reference_radius,
         z = state[i * 6 + 2] / reference_radius

    # Преобразование в  СК Юпитера
    real _x = rot[0] * x + rot[3] * y + rot[6] * z,
         _y = rot[1] * x + rot[4] * y + rot[7] * z
         _z = rot[2] * x + rot[5] * y + rot[8] * z

    real r2 = (r r), r = sqrt(r2), r3 = r * r2, r4 = r2 * r2, r5 = r4 * r

    real[NGRAV] v_0, dv_0_dx, dv_0_dy, dv_0_dz

    v_0[0] = 1 / r               # V00 = 1 / r
    v_0[1] = sqrt(3.0) * z / r3  # V10 = sqrt(3) z / r^3

    dv_0_dx[0] = -x / r3
    dv_0_dy[0] = -y / r3
    dv_0_dz[0] = -z / r3

    dv_0_dx[1] = sqrt(3.0) * -3 * z * x / r5
    dv_0_dy[1] = sqrt(3.0) * -3 * z * y / r5
    dv_0_dz[1] = sqrt(3.0) * (1 / r3 - 3 * z * z / r5)

    for n = [2 : NGRAV]
    {
       real coef1 = sqrt((2.0 * n - 1) * (2 * n + 1) / (n * n)),
            coef2 = sqrt((n - 1.0) * (n - 1) * (2 * n + 1) / (n * n * (2 * n - 3)))

       v_0[n] = coef1 * v_0[n - 1] * z / r2 - coef2 * v_0[n - 2] / r2

       # Производная ГП по x, y, z
       dv_0_dx[n] = coef1 * z * (dv_0_dx[n - 1] / r2 - v_0[n - 1] * 2 * x / r4) -
                    coef2 *     (dv_0_dx[n - 2] / r2 - v_0[n - 2] * 2 * x / r4)
       dv_0_dy[n] = coef1 * z * (dv_0_dy[n - 1] / r2 - v_0[n - 1] * 2 * y / r4) -
                    coef2 *     (dv_0_dy[n - 2] / r2 - v_0[n - 2] * 2 * y / r4)
       dv_0_dz[n] = coef1 * (dv_0_dz[n - 1] * z / r2 + v_0[n - 1] * (1 / r2 - 2 * z * z / r4)) -
                    coef2 * (dv_0_dz[n - 2] / r2 - v_0[n - 2] * 2 * z / r4)
    }

    real dpotential_dx = 0, dpotential_dy = 0, dpotential_dz = 0

    for n = [2 : NGRAV]
    {
       dpotential_dx += dv_0_dx[n] * central_grav[n]
       dpotential_dy += dv_0_dy[n] * central_grav[n]
       dpotential_dz += dv_0_dz[n] * central_grav[n]
    }

    real[3] grav_acc

    # Возврат в небесную СК
    for k = [0 : 3]
      grav_acc[k] = (rot[3 * k + 0] * dpotential_dx +
                     rot[3 * k + 1] * dpotential_dy +
                     rot[3 * k + 2] * dpotential_dz)

    sat_acc[i * 3 : i * 3 + 3] += central_gm * grav_acc[:] / sqr(reference_radius)
    central_acc[:] = central_acc[:] - sat_gms[i] * grav_acc[:] / sqr(reference_radius)
  }

  for i = [0 : NSAT]
  {
    real[3] acc = sat_acc[i * 3 : i * 3 + 3] - central_acc[:]
    # Запись сил и скоростей системы в выходной массив
    jupsatsystem[i * 6     : i * 6 + 3] = state[i * 6 + 3 : i * 6 + 6]
    jupsatsystem[i * 6 + 3 : i * 6 + 6] = acc[:]

    # Запись автоматических производных в выходной массив
    for j = [0 : 3]
    {
      # Производная по времени j-й координаты i-го тела относительно 24 начальных параметров
      jupsatsystem[NSAT * 6 + (i * 6 + j    ) * NSAT * 6 :
                   NSAT * 6 + (i * 6 + j + 1) * NSAT * 6] =
          state_and_derivatives[NSAT * 6 + (i * 6 + j + 3) * NSAT * 6 :
                                NSAT * 6 + (i * 6 + j + 4) * NSAT * 6]
      # Производная по времени j-й скорости i-го тела относительно 24 начальных параметров
      jupsatsystem[NSAT * 6 + (i * 6 + j + 3) * NSAT * 6 :
                   NSAT * 6 + (i * 6 + j + 4) * NSAT * 6] =
          acc[j] ' initial[0 : NSAT * 6]
    }
  }
}
```

Процесс разработки на Landau

1. Описание модели на Landau

   jup-sat.dau:

   ```
   real[DIM] jupsatsystem(...){
     ...
   }
   ```

2. Трансляция jup-sat.dau -> jup-sat.c

   jup-sat.c:

   ```
   int jupsatsystem(long double *jupsatsystem34279,
     ...
     return 0;
   }
   ```

3. Компиляция в динамическую библиотеку

   ```
   jup-sat.c -> jup-sat.dll,
                jup-sat.so,
                jup-sat.dylib.
   ```

Результаты

Интегрирование орбит 4 спутников Юпитера методом Эверхарта, на интервале 1891-2020 гг, с шагом 0.04 суток

Количество уравнений	**время, мин**
24 (Старая реализация, без производных)[1]	26
24 (Новая реализация, без производных)[2]	14
24 (Новая реализация, числ. производные)[3]	350
600 (Новая реализация, авт. производные)[4]	118

[1] 24 компоненты координат и скоростей (Racket)

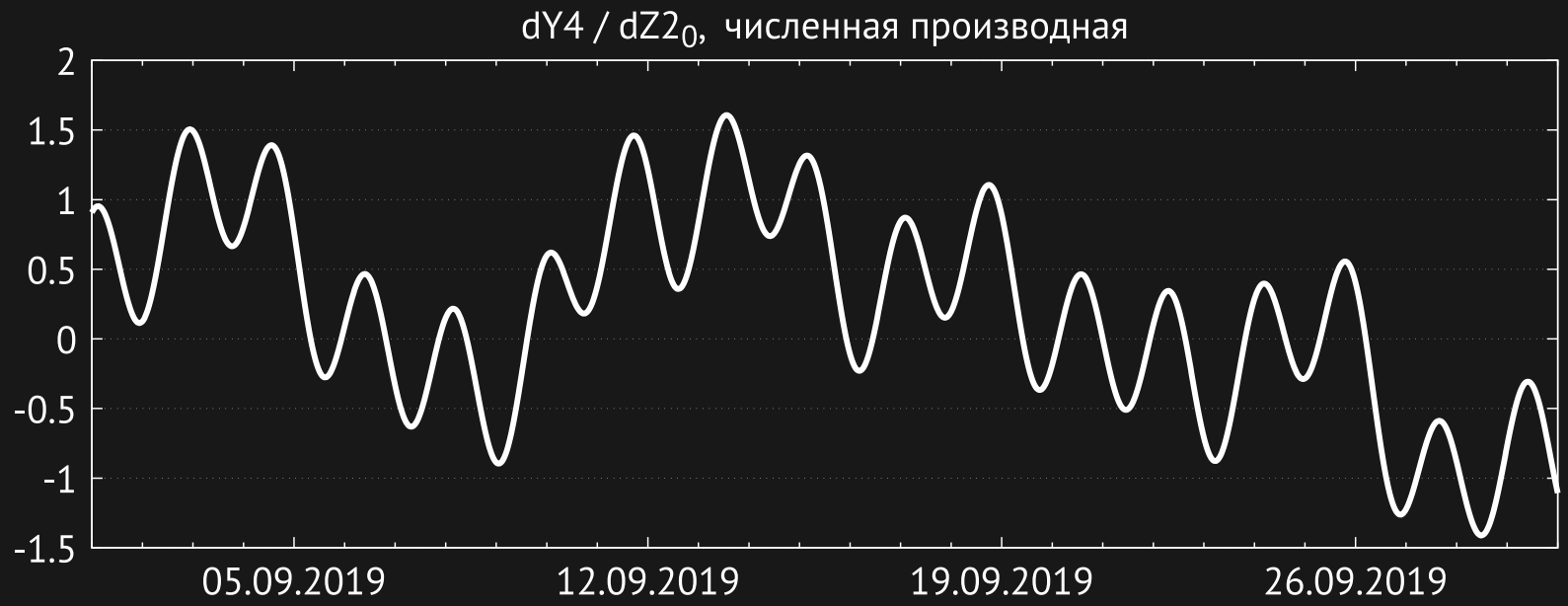
[2] 24 компоненты координат и скоростей (Landau C)

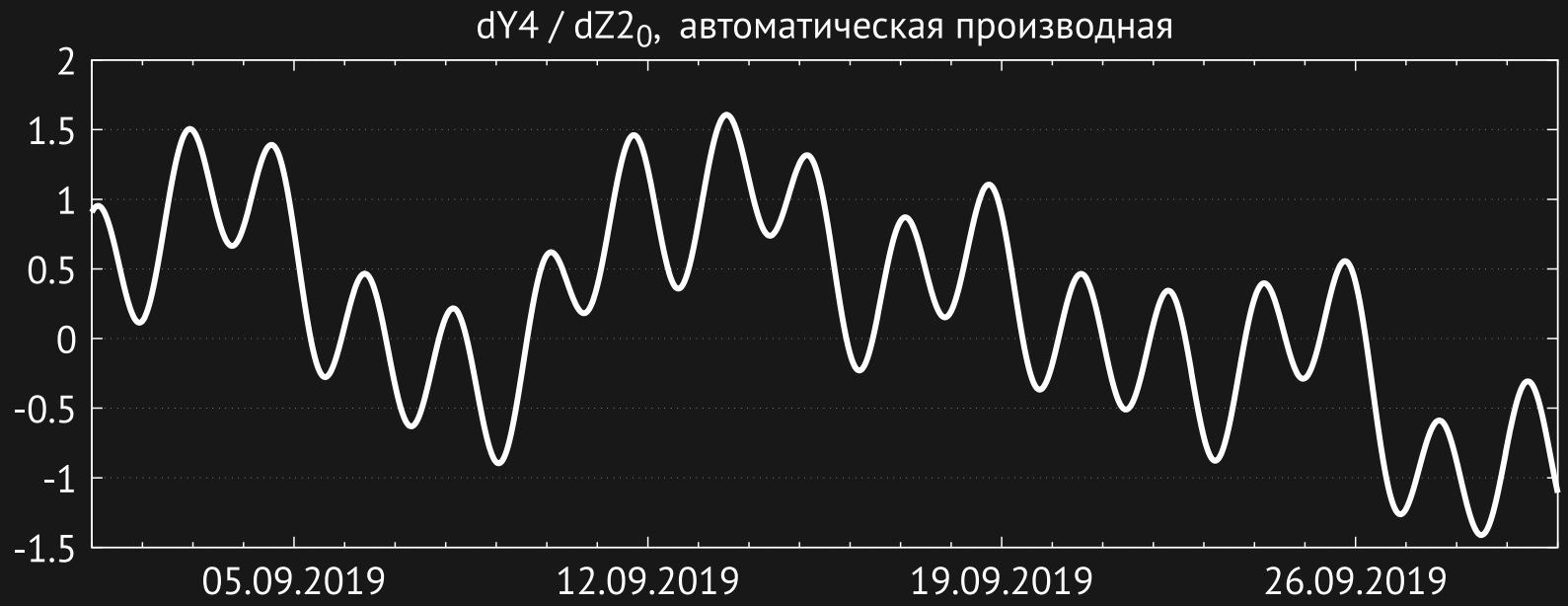
[3] 25 интегрирований

[3] 24 компоненты координат и скоростей + 24 * 24 производных, сгенерированных Landau


dX1 / dX1$_0$, численная производная
200
150
100
50
0
-50
-100
-150
-200
02.01.1958
09.01.1958
16.01.1958
23.01.1958
30.01.1958


dX1 / dX1$_0$, автоматическая производная
200
150
100
50
0
-50
-100
-150
-200
02.01.1958
09.01.1958
16.01.1958
23.01.1958
30.01.1958


dY4 / dZ2$_0$, численная производная
2
1.5
1
0.5
0
-0.5
-1
-1.5
05.09.2019
12.09.2019
19.09.2019
26.09.2019


dY4 / dZ2$_0$, автоматическая производная
2
1.5
1
0.5
0
-0.5
-1
-1.5
05.09.2019
12.09.2019
19.09.2019
26.09.2019

Уточнённые параметры[1]

Параметр	значение	ошибка
Jup1 X	304810.529	14.159 km
Jup1 Y	-264788.660	14.459 km
Jup1 Z	-121359.178	13.036 km
Jup1 VX	12031.154	0.598 m/s
Jup1 VY	11204.025	0.578 m/s
Jup1 VZ	5524.766	0.543 m/s
Jup2 X	35938.862	18.318 km
Jup2 Y	-607574.995	6.101 km
Jup2 Z	-293351.494	11.299 km
Jup2 VX	13622.680	0.067 m/s
Jup2 VY	617.682	0.331 m/s
Jup2 VZ	588.623	0.288 m/s
Jup3 X	-1021153.696	6.448 km
Jup3 Y	292563.471	15.079 km
Jup3 Z	123471.768	12.554 km
Jup3 VX	-3225.610	0.159 m/s
Jup3 VY	-9388.208	0.084 m/s
Jup3 VZ	-4477.346	0.120 m/s
Jup4 X	1850497.943	6.441 km
Jup4 Y	249003.542	25.267 km
Jup4 Z	141678.473	16.229 km
Jup4 VX	-1214.851	0.108 m/s
Jup4 VY	7373.750	0.035 m/s
Jup4 VZ	3490.343	0.053 m/s

[1] На основе наблюдательных данных, предоставленных Г. А. Космодамианским

Будущее применение Landau

- Совершенствование модели спутников Юпитера.
- Реализация модели движения Луны.
- Реализация динамических систем спутников Сатурна, Урана, Нептуна.
- Оценка точности эфемерид по матрице ковариации свободных параметров.
- Аналог Numpy для Racket.
- Другие задачи, требующие расчета производных или кодогенерации.

Будущее развитие языка

- Комплексные числа;
- Улучшения интерфейсов работы с массивами;
- Оптимизация генерируемого кода;
- FFI: C, Racket;
- Производные высших порядков.

Публикации и конференции

- *Landau: language for dynamical systems with automatic differentiation*

  Polynomial Computer Algebra ’2019

  15-20 апреля 2019

  Международный математический институт им. Леонарда Эйлера, Санкт-Петербург.

- *Using capabilities of Racket to implement a domain-specifc language*

  Computer Assisted Mathematics Conference CAM-2019

  22–24 июля 2019

  Санкт-Петербургский государственный электротехнический университет «ЛЭТИ».