AstDyn
Guida Rapida

Libreria per Determinazione Orbitale Asteroidi

Versione 1.0 – Novembre 2025

Sommario

Guida rapida alle funzionalità essenziali di AstDyn: installazione, configurazione e workflow base per la determinazione orbitale di asteroidi. Per dettagli completi consultare il manuale di 290 pagine.

1 Introduzione

AstDyn è una libreria C++17 per la determinazione orbitale di asteroidi che fornisce:

- **Propagazione orbitale** con integratori numerici ad alta precisione
- **Determinazione orbitale** tramite correzione differenziale
- **Parser multipli** (OrbFit .eq1, MPC, formati personalizzati)
- **Sistemi temporali** (UTC, TT, TDB) con conversioni automatiche
- **Validazione** completa vs OrbFit e JPL Horizons

Prestazioni: 535 orbite/secondo, 29% più veloce di OrbFit, uso memoria 12 MB.

2 Installazione Rapida

2.1 Dipendenze

```
# Ubuntu/Debian
sudo apt-get install build-essential cmake git
sudo apt-get install libeigen3-dev libboost-all-dev

# macOS
brew install cmake eigen boost
```

2.2 Compilazione

```
git clone https://github.com/user/astdyn.git
cd astdyn
mkdir build && cd build
cmake .. -DCMAKE_BUILD_TYPE=Release
make -j4
sudo make install
```

3 Quick Start

3.1 Esempio Minimo: Propagazione Orbita

```cpp
#include <astdyn/AstDyn.hpp>
using namespace astdyn;

int main() {
    // 1. Definire elementi kepleriani
    orbit::KeplerianElements elem;
    elem.a = 2.7436;                                    // AU
    elem.e = 0.0624;
    elem.i = 11.74 * constants::DEG_TO_RAD;
    elem.Omega = 339.86 * constants::DEG_TO_RAD;
    elem.omega = 258.03 * constants::DEG_TO_RAD;
    elem.M = 45.32 * constants::DEG_TO_RAD;
    elem.epoch = 2460000.5;                             // JD

    // 2. Configurare propagatore
    auto eph = std::make_shared<ephemeris::AnalyticEphemeris>();
    auto forces = std::make_shared<propagation::PointMassGravity>(
        eph, std::vector<std::string>{"JUPITER", "SATURN"});
    auto integrator = std::make_shared<propagation::RKF78>(1e-12);
    propagation::Propagator prop(integrator, forces, eph);

    // 3. Propagare 60 giorni
    auto state0 = elem.to_cartesian();
    auto state60 = prop.propagate(state0, elem.epoch + 60.0);

    // 4. Convertire risultato
    auto elem60 = orbit::KeplerianElements::from_cartesian(
        state60.position, state60.velocity, state60.epoch);

    std::cout << "a finale = " << elem60.a << " AU\n";
    return 0;
}
```

3.2 Compilazione Esempio

```
g++ -std=c++17 -O3 esempio.cpp -o esempio \
    -I/usr/local/include \
    -L/usr/local/lib -lastdyn -lboost_system
./esempio
```

4 Workflow Determinazione Orbitale

4.1 Passo 1: Caricare Osservazioni

```cpp
// Da file MPC (formato 80 colonne)
auto observations = io::MPCReader::read_file("pompeja.obs");

// Da file OrbFit .eq1
io::OrbFitEQ1Parser parser;
auto initial_elem = parser.parse("pompeja.eq1");
```

4.2 Passo 2: Orbita Iniziale (Gauss)

```
// Selezionare 3 osservazioni ben distribuite
auto obs1 = observations.front();
auto obs2 = observations[observations.size() / 2];
auto obs3 = observations.back();

// Metodo Gauss
GaussIOD gauss;
auto initial_guess = gauss.solve(obs1, obs2, obs3);
```

4.3 Passo 3: Correzione Differenziale

```
// Configurare correzione differenziale
orbit_determination::DifferentialCorrector corrector(
    propagator,
    20,       // max iterazioni
    1e-8      // tolleranza convergenza (AU)
);

// Eseguire
auto result = corrector.solve(
    initial_guess,
    observations,
    observatory_coords
);

// Verificare risultato
if (result.converged) {
    std::cout << "RMS residuo: " << result.rms_residual
              << " arcsec\n";
    std::cout << "a = " << result.elements.a << " AU\n";
    std::cout << "e = " << result.elements.e << "\n";
}
```

5 Configurazioni Raccomandate

5.1 Tolleranze Integrazione

Applicazione	Tolleranza	Accuratezza
Studio preliminare	10^{-10}	50 km / 60 giorni
Standard	10^{-12}	3 km / 60 giorni
Alta precisione	10^{-14}	0.08 km / 60 giorni

5.2 Modelli Forza per Tipo Oggetto

Fascia principale (tipico):

```
auto forces = std::make_shared<PointMassGravity>(
    eph, std::vector<std::string>{"JUPITER", "SATURN"});
```

Near-Earth Asteroids:

```
auto forces = std::make_shared<PointMassGravity>(
    eph, std::vector<std::string>{"EARTH", "JUPITER", "MARS"});
```

Sistema solare esterno:

```cpp
auto forces = std::make_shared<PointMassGravity>(
    eph, std::vector<std::string>{
        "JUPITER", "SATURN", "URANUS", "NEPTUNE"});
```

6 API Essenziale

6.1 Elementi Orbitali

```cpp
// Classe KeplerianElements
double a;          // Semiasse maggiore (AU)
double e;          // Eccentricita'
double i;          // Inclinazione (rad)
double Omega;      // Nodo ascendente (rad)
double omega;      // Argomento perielio (rad)
double M;          // Anomalia media (rad)
double epoch;      // Epoca (JD)

// Conversioni
auto state = elem.to_cartesian();
auto elem2 = KeplerianElements::from_cartesian(pos, vel, epoch);

// Quantita' derivate
double T = elem.period();              // Periodo (giorni)
double n = elem.mean_motion();         // Moto medio
double q = elem.perihelion_distance(); // Perielio
```

6.2 Propagazione

```cpp
// Propagazione base
auto state_final = propagator.propagate(state0, target_epoch);

// Con matrice transizione stato (STM)
auto [state, stm] = propagator.propagate_with_stm(
    state0, target_epoch);

// Generare effemeridi
auto ephemeris = propagator.generate_ephemeris(
    state0, start_epoch, end_epoch, step_days);
```

6.3 Sistemi Temporali

```cpp
// Conversioni temporali
double jd_tt = time::TimeConverter::utc_to_tt(jd_utc);
double jd_tdb = time::TimeConverter::tt_to_tdb(jd_tt);

// Epoche standard
constexpr double JD_J2000 = 2451545.0;
constexpr double MJD_OFFSET = 2400000.5;
```

7 Costanti Utili

```
// Costanti fisiche
constants::C          // Velocita' luce
constants::G          // Cost. grav.
constants::AU         // UA (m)

// Costanti temporali
constants::JD_J2000
constants::DAYS_PER_CENTURY
constants::SECONDS_PER_DAY
```

```
// Conversioni angolari
constants::DEG_TO_RAD
constants::RAD_TO_DEG
constants::ARCSEC_TO_RAD

// Masse planetarie (GM)
constants::GM_SUN
constants::GM_JUPITER
constants::GM_SATURN
constants::GM_EARTH
```

8 Troubleshooting Comune

8.1 Correzione Non Converge

Causa: Guess iniziale troppo lontano dalla soluzione vera.
Soluzione:

- Verificare elementi iniziali ($|\Delta a| < 0.1$ AU)
- Provare diverso terzetto osservazioni per Gauss
- Aumentare iterazioni max a 50
- Filtrare outlier (residui $> 3\sigma$)

8.2 NaN nei Risultati

Causa: Elementi orbitali invalidi o eccentricità parabola.
Soluzione:

```
if (!elem.is_valid()) {
    std::cerr << "Elementi invalidi\n";
    return;
}
if (elem.e >= 1.0) {
    std::cerr << "Orbita non ellittica\n";
    // Usare elementi cometari invece
}
```

8.3 Integrazione Lenta

Causa: Tolleranza troppo stretta o troppi corpi perturbatori.
Soluzione:

- Usare tolleranza 10^{-12} (non 10^{-14})
- Limitare a Giove + Saturno per fascia principale
- Verificare passo max non troppo piccolo

9 Validazione e Qualità

9.1 Accuratezza Numerica

- **Problema 2 corpi**: Accuratezza sub-nanometrica
- **vs OrbFit**: Accordo $< 10^{-7}$ AU elementi orbitali

- **vs JPL Horizons**: Differenza < 5 km su 1 anno
- **RMS residui tipici**: 0.5–0.7 arcsec per asteroidi ben osservati

9.2 Copertura Test

- **335 unit test**: Copertura 96% codice
- **15 test integrazione**: Workflow end-to-end
- **5 test confronto**: Dati riferimento OrbFit
- **CI/CD**: GitHub Actions su ogni commit

10 Esempio Completo: Pompeja

Caso studio completo per (203) Pompeja (vedi Capitolo 22 manuale completo):

- **Osservazioni**: 100 da Pan-STARRS, arco 60 giorni
- **RMS finale**: 0.658 arcsec
- **Iterazioni**: 4 (convergenza rapida)
- **Tempo calcolo**: 1.82 secondi
- **Accordo OrbFit**: Differenze $< 10^{-7}$ AU
- **Accuratezza posizione**: 3.9 km vs JPL dopo 60 giorni

Risultato: Soluzione production-ready adatta a pubblicazione scientifica.

11 Risorse

- **Manuale completo**: 290 pagine con teoria, API dettagliata, validazione
- **Esempi**: 6 programmi completi funzionanti (Capitolo 20)
- **Repository**: `github.com/user/astdyn`
- **Documentazione API**: Doxygen online
- **Supporto**: Issue tracker GitHub

11.1 Letture Consigliate

- **Murray & Dermott** (1999). *Solar System Dynamics*
- **Milani & Gronchi** (2010). *Theory of Orbit Determination*
- **Danby** (1988). *Fundamentals of Celestial Mechanics*

AstDyn v1.0 – Libreria C++17 per Determinazione Orbitale Asteroidi
Validato vs OrbFit e JPL Horizons – Pronto per uso produttivo
`https://github.com/user/astdyn`