

Libreria AstDyn

Manuale Scientifico

Meccanica Celeste e Determinazione Orbitale

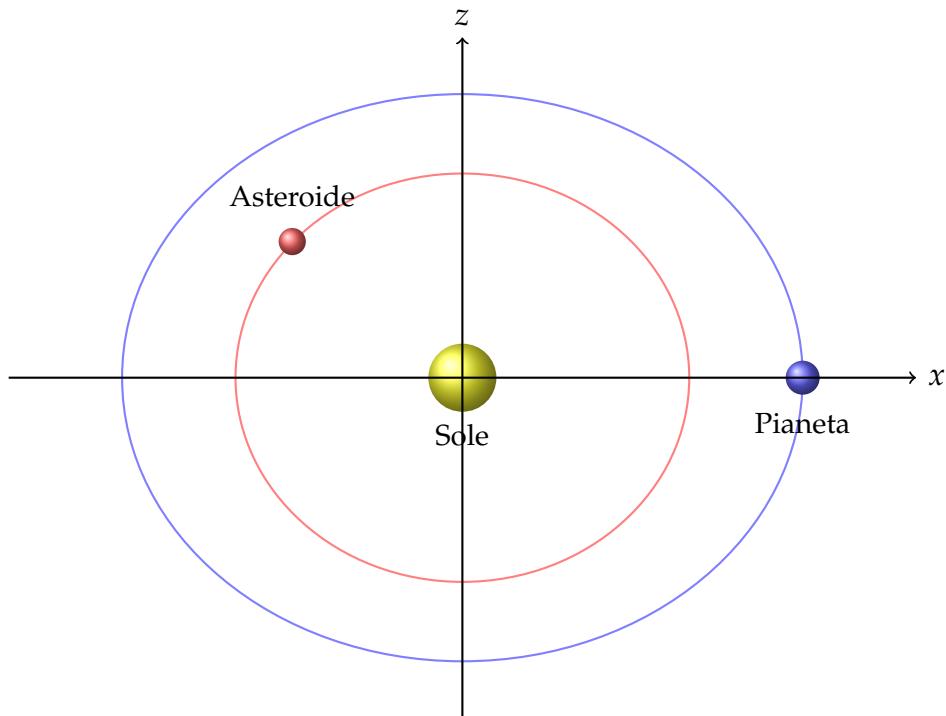


Figura 1: *
Dinamica orbitale eliocentrica

Michele Bigi

Versione 1.0.0

26 novembre 2025

*Una libreria completa in C++17 per la meccanica celeste,
la determinazione orbitale e i calcoli di astrodinamica*

Prefazione

Questo manuale fornisce una guida completa alla libreria AstDyn, un toolkit C++ moderno per calcoli di meccanica celeste e determinazione orbitale. AstDyn è progettato per essere sia educativo che pratico, fornendo implementazioni chiare di algoritmi classici insieme a funzionalità avanzate per applicazioni di ricerca.

Motivazione

Lo studio della meccanica celeste richiede una solida comprensione sia della teoria matematica che delle tecniche numeriche. Mentre esistono molti testi eccellenti sulla teoria, c'è spesso un divario tra le equazioni nei libri e il codice funzionante. AstDyn mira a colmare questo divario fornendo:

- Implementazioni ben documentate di algoritmi classici
- Architettura software moderna e manutenibile
- Esempi pratici e casi di validazione
- Strumenti per applicazioni reali di ricerca

Struttura del Manuale

Il manuale è organizzato in cinque parti:

Parte I: Fondamenti Teorici copre i concetti fondamentali: sistemi di tempo e coordinate, elementi orbitali, il problema dei due corpi e teoria delle perturbazioni.

Parte II: Metodi Numerici descrive l'integrazione numerica, la propagazione orbitale, matrici di transizione di stato e generazione di effemeridi.

Parte III: Determinazione Orbitale spiega come determinare orbite da osservazioni usando metodi classici e moderni.

Parte IV: Implementazione documenta l'architettura della libreria, i moduli core, il sistema di parser configurabile e l'API completa.

Parte V: Validazione e Applicazioni presenta casi di studio reali, analisi di performance e best practices.

Pubblico di Riferimento

Questo manuale è rivolto a:

- Studenti di laurea magistrale e dottorato in astrofisica o ingegneria aerospaziale
- Ricercatori che necessitano di strumenti affidabili per determinazione orbitale
- Sviluppatori di software che lavorano su applicazioni di meccanica celeste
- Chiunque sia interessato a comprendere gli algoritmi dietro i calcoli orbitali

Si assume familiarità con il calcolo multivariabile, l'algebra lineare e la programmazione C++. La conoscenza della meccanica classica è utile ma non strettamente necessaria.

Convenzioni

- I **vettori** sono indicati in grassetto: **r**, **v**
- Le **matrici** sono indicate in maiuscolo grassetto: **A**, **H**
- Gli **scalari** sono in corsivo normale: *a*, *e*, μ
- Le unità SI sono utilizzate salvo diversa indicazione
- Il codice C++ è mostrato con syntax highlighting

Risorse Online

- Repository GitHub: <https://github.com/your-org/astdyn>
- Documentazione API: <https://astdyn.readthedocs.io>
- Issues e supporto: <https://github.com/your-org/astdyn/issues>

Buona lettura e buoni calcoli orbitali!

Gli Autori

Indice

Prefazione	iii
I Fondamenti Teorici della Meccanica Celeste	1
1 Introduzione	3
1.1 Cos'è la Meccanica Celeste?	3
1.1.1 Il Problema dei Due Corpi	3
1.1.2 Il Problema degli N Corpi	4
1.2 Panoramica della Libreria AstDyn	5
1.2.1 Filosofia di Progettazione	5
1.2.2 Caratteristiche Principali	5
1.2.3 Architettura Software	6
1.2.4 Dipendenze	7
1.3 Applicazioni	7
1.3.1 Determinazione di Orbite Asteroidali	7
1.3.2 Analisi di Traiettorie Spaziali	8
1.3.3 Evoluzione Orbitale a Lungo Termine	8
1.3.4 Strumento Educativo	8
1.4 Validazione e Accuratezza	9
1.5 Per Iniziare	9
1.5.1 Installazione	9
1.5.2 Esempio Rapido	9
1.6 Organizzazione dei Capitoli Rimanenti	11
2 Sistemi di Tempo nella Meccanica Celeste	13
2.1 Perché Sistemi Temporali Multipli?	13
2.2 Numero di Giorno Giuliano	14
2.2.1 Giorno Giuliano Modificato	14
2.3 Tempo Universale (UT)	14

2.3.1	UT0	14
2.3.2	UT1	15
2.3.3	UTC (Tempo Universale Coordinato)	15
2.4	Scale di Tempo Atomiche	15
2.4.1	TAI (Tempo Atomico Internazionale)	15
2.4.2	TT (Tempo Terrestre)	16
2.4.3	TDB (Tempo Dinamico Baricentrico)	16
2.5	Relazioni tra Scale Temporali	16
2.6	Implementazione in AstDyn	17
2.6.1	Tabella dei Secondi Intercalari	17
2.6.2	Approssimazioni di ΔT	18
2.7	Considerazioni Pratiche	18
2.7.1	Quale Scala Temporale Usare?	18
2.7.2	Requisiti di Precisione	19
2.7.3	Esempio: Catena di Conversione Temporale	19
2.8	Letture Approfondite	19
3	Sistemi di Coordinate e Sistemi di Riferimento	21
3.1	Introduzione	21
3.2	Concetti Fondamentali	21
3.2.1	Sistemi Inerziali vs. Rotanti	21
3.3	Sistema di Coordinate Equatoriali	22
3.3.1	Definizione	22
3.3.2	Coordinate Sferiche	22
3.4	Sistema di Coordinate Eclittiche	23
3.4.1	Definizione	23
3.4.2	Perché Usare Coordinate Eclittiche?	23
3.5	Trasformazione tra Sistemi	23
3.5.1	Eclittico \leftrightarrow Equatoriale	23
3.5.2	Implementazione	24
3.6	Il Sistema di Riferimento J2000.0	24
3.6.1	Epoca vs. Equinozio	24
3.6.2	Precessione	25
3.7	Considerazioni Pratiche	25
3.7.1	Scelta del Sistema di Riferimento	25
3.7.2	Trasformazioni di Sistema in AstDyn	25

3.8	Riepilogo	26
4	Sistemi di Riferimento	27
4.1	Introduzione ai Sistemi di Riferimento	27
4.2	Il Sistema di Riferimento Celeste Internazionale (ICRS)	27
4.2.1	Definizione dell'ICRS	28
4.2.2	Relazione con J2000.0	28
4.3	Il Sistema Equatoriale J2000.0	28
4.3.1	Definizione di J2000.0	28
4.3.2	Sistemi Eliocentrici vs. Baricentrici	29
4.4	Il Sistema di Riferimento Eclittico	29
4.4.1	Definizione dell'Eclittica	30
4.4.2	Coordinate Eclittiche	30
4.5	Trasformazioni tra Sistemi di Riferimento	31
4.5.1	Trasformazione Equatoriale-Eclittica	31
4.5.2	Precessione: Trasformazioni Dipendenti dal Tempo	31
4.5.3	Implementazione in AstDyn	32
4.6	Altri Importanti Sistemi di Riferimento	33
4.6.1	Il Sistema FK5	33
4.6.2	Il Piano Invariabile	33
4.6.3	Sistemi Corpo-Centrici	33
4.7	Considerazioni Pratiche	34
4.7.1	Precisione Numerica	34
4.7.2	Scelta del Sistema per la Propagazione Orbitale	34
4.7.3	Conversione delle Osservazioni	34
4.8	Riepilogo	35
5	Elementi Orbitali	37
5.1	Introduzione agli Elementi Orbitali	37
5.2	Elementi Kepleriani Classici	37
5.2.1	I Sei Elementi Kepleriani	38
5.2.2	Periodo Orbitale	39
5.2.3	Energia Orbitale	39
5.2.4	Singolarità degli Elementi Kepleriani	40
5.3	Vettore di Stato Cartesiano	40
5.3.1	Vantaggi	40

5.3.2	Svantaggi	40
5.3.3	Conversione: Kepleriano a Cartesiano	41
5.3.4	Conversione: Cartesiano a Kepleriano	41
5.4	Elementi Equinoziali	42
5.4.1	Definizione	43
5.4.2	Conversione a Kepleriano	43
5.4.3	Vantaggi	43
5.5	Elementi di Delaunay	43
5.5.1	Definizione	44
5.5.2	Proprietà	44
5.6	Implementazione in AstDyn	44
5.7	Riepilogo	45
6	Il Problema dei Due Corpi	47
6.1	Introduzione al Problema dei Due Corpi	47
6.1.1	Formulazione del Problema	47
6.1.2	Riduzione a Problema a Un Corpo	47
6.2	Leggi di Conservazione	48
6.2.1	Conservazione del Momento Angolare	48
6.2.2	Conservazione dell'Energia	48
6.2.3	Il Vettore di Laplace-Runge-Lenz	49
6.3	L'Equazione dell'Orbita	49
6.3.1	Derivazione	49
6.3.2	Sezioni Coniche	49
6.4	Le Leggi di Keplero	49
6.4.1	Prima Legge di Keplero (Legge delle Ellissi)	51
6.4.2	Seconda Legge di Keplero (Legge delle Aree)	51
6.4.3	Terza Legge di Keplero (Legge Armonica)	51
6.5	L'Equazione di Keplero	51
6.5.1	Le Anomalie	51
6.5.2	L'Equazione di Keplero	52
6.5.3	Risoluzione dell'Equazione di Keplero	52
6.5.4	Relazione tra le Anomalie	53
6.6	L'Equazione Vis-Viva	53
6.6.1	Casi Speciali	54
6.7	Orbite Paraboliche e Iperboliche	54

6.7.1	Orbite Paraboliche ($e = 1$)	54
6.7.2	Orbite Iperboliche ($e > 1$)	54
6.8	Coefficienti di Lagrange	55
6.8.1	Definizione	55
6.8.2	Espressioni per i Coefficienti di Lagrange	56
6.8.3	Proprietà	56
6.9	Implementazione in AstDyn	56
6.10	Sommario	57
7	Perturbazioni Orbitali	59
7.1	Introduzione alle Perturbazioni	59
7.1.1	Tipi di Perturbazioni	59
7.1.2	Equazioni del Moto Perturbato	59
7.1.3	Magnitudine degli Effetti	60
7.2	Il Problema degli N Corpi	60
7.2.1	Formulazione del Problema	60
7.2.2	Il Problema Ristretto dei Tre Corpi	60
7.2.3	Perturbazioni dai Pianeti	61
7.2.4	Effemeridi Planetarie	61
7.3	Perturbazioni da Schiacciamento (J_2)	61
7.3.1	Distribuzione di Massa Non Sferica	61
7.3.2	Il Termine J_2	62
7.3.3	Accelerazione da J_2	62
7.3.4	Effetti sugli Elementi Orbitali	62
7.4	Pressione di Radiazione Solare	63
7.4.1	Meccanismo Fisico	63
7.4.2	Rapporto Area-Massa	63
7.4.3	Modellazione dell'Eclisse	63
7.4.4	Effetto Yarkovsky	64
7.5	Effetti Relativistici	64
7.5.1	Correzioni Post-Newtoniane	64
7.5.2	Precessione del Perielio	64
7.5.3	Correzione del Tempo Luce	65
7.5.4	Ritardo di Shapiro	65
7.6	Resistenza Atmosferica	65
7.6.1	Equazione della Resistenza	65

7.6.2	Modelli di Densità Atmosferica	66
7.6.3	Decadimento Orbitale	66
7.7	Teoria delle Perturbazioni	66
7.7.1	Variazione dei Parametri	66
7.7.2	Equazioni di Perturbazione di Gauss	67
7.7.3	Elementi Osculatori	67
7.8	Integrazione Numerica vs Teoria delle Perturbazioni	68
7.8.1	Quando Usare Ogni Approccio	68
7.8.2	Approcci Ibridi	68
7.9	Implementazione in AstDyn	68
7.9.1	Selezione delle Perturbazioni	70
7.10	Sommario	70

II Metodi Numerici e Algoritmi 73

8	Metodi di Integrazione Numerica 75
8.1	Introduzione 75
8.1.1	Il Problema ai Valori Iniziali 75
8.2	Metodo di Eulero 75
8.3	Metodi Runge-Kutta 76
8.3.1	Il Metodo RK4 76
8.3.2	Metodi Runge-Kutta Incorporati 76
8.3.3	Controllo della Dimensione del Passo 77
8.4	Metodi Multipasso 77
8.4.1	Adams-Bashforth-Moulton (ABM) 77
8.4.2	Formule di Differenziazione all'Indietro (BDF) 78
8.5	Integratori Simplettici 78
8.5.1	Meccanica Hamiltoniana 78
8.5.2	Proprietà Simplettica 79
8.5.3	Metodo Leapfrog 79
8.5.4	Metodi Simplettici di Ordine Superiore 79
8.6	Analisi dell'Errore 80
8.6.1	Errore Locale vs Globale 80
8.6.2	Compromesso Accuratezza vs Efficienza 80
8.6.3	Fonti di Errore 80
8.7	Considerazioni Pratiche 81

8.7.1	Scelta di un Integratore	81
8.7.2	Selezione della Dimensione del Passo	81
8.7.3	Dimensione del Passo Iniziale	81
8.8	Implementazione in AstDyn	82
8.8.1	Integratori Personalizzati	83
8.9	Sommario	83
9	Propagazione delle Orbite	85
9.1	Introduzione	85
9.2	Formulazione del Problema	85
9.2.1	Il Compito di Propagazione	85
9.2.2	Vettore di Stato	86
9.2.3	Equazioni del Moto	86
9.3	Modelli di Forza	87
9.3.1	Gravità del Corpo Centrale	87
9.3.2	Perturbazioni Planetarie	87
9.3.3	Correzione Relativistica	87
9.3.4	Pressione di Radiazione Solare	87
9.3.5	Perturbazioni Asteroidali	88
9.4	Sistemi di Coordinate	88
9.4.1	Sistemi di Riferimento	88
9.4.2	Trasformazioni di Sistema	89
9.5	Strategia di Integrazione	89
9.5.1	Scelta del Passo	89
9.5.2	Selezione della Tolleranza	89
9.5.3	Punti di Output	90
9.6	Modalità di Propagazione	90
9.6.1	Propagazione in Avanti e all'Indietro	90
9.6.2	Epoca Singola vs Multi-Epoca	90
9.7	Generazione di Effemeridi	91
9.7.1	Effemeridi Tabulate	91
9.7.2	Interpolazione di Chebyshev	92
9.8	Matrice di Transizione di Stato	92
9.8.1	Definizione	92
9.8.2	Applicazioni	92
9.8.3	Calcolo	93

9.9	Esempi Pratici	93
9.9.1	Esempio 1: Asteroide della Fascia Principale	93
9.9.2	Esempio 2: Analisi Avvicinamento Ravvicinato	94
9.9.3	Esempio 3: Propagazione di Cometa	95
9.10	Ottimizzazione delle Prestazioni	96
9.10.1	Selezione del Modello di Forza	96
9.10.2	Passo Adattativo vs Fisso	96
9.10.3	Parallelizzazione	96
9.11	Validazione dell'Accuratezza	97
9.11.1	Conservazione dell'Energia	97
9.11.2	Confronto Problema a Due Corpi	97
9.12	Riepilogo	98
10	Matrice di Transizione di Stato	99
10.1	Introduzione	99
10.2	Fondamenti Matematici	99
10.2.1	Linearizzazione della Dinamica	99
10.2.2	Equazioni Variazionali	99
10.2.3	Definizione Matrice di Transizione di Stato	100
10.2.4	Proprietà	100
10.3	Calcolo della Matrice Jacobiana	100
10.3.1	Problema dei Due Corpi	100
10.3.2	Perturbazioni N-Corpi	101
10.3.3	Correzioni Relativistiche	101
10.3.4	Pressione di Radiazione Solare	102
10.4	Calcolo Numerico	102
10.4.1	Vettore di Stato Aumentato	102
10.4.2	Dinamica Aumentata	102
10.4.3	Implementazione in AstDyn	102
10.4.4	Costo Computazionale	103
10.5	Applicazioni	103
10.5.1	Determinazione Orbitale	103
10.5.2	Propagazione Covarianza	104
10.5.3	Analisi di Sensibilità	104
10.5.4	Ottimizzazione Manovre	104
10.6	STM Analitica vs Numerica	105

10.6.1 STM Analitica per Moto Kepleriano	105
10.6.2 STM Numerica	105
10.6.3 Approcci Ibridi	106
10.7 Stabilità Numerica	106
10.7.1 Problemi di Condizionamento	106
10.7.2 Strategie di Mitigazione	106
10.8 Esempio Pratico	107
10.8.1 Tracciamento Bersaglio	107
10.8.2 Pianificazione Osservazioni	108
10.9 Sensibilità Parametri	109
10.9.1 Vettore di Stato Esteso	109
10.9.2 Matrici di Sensibilità	109
10.10 Riepilogo	110
11 Calcolo di Effemeridi	111
11.1 Introduzione	111
11.2 Tipi di Effemeridi	111
11.2.1 Effemeridi Planetarie	111
11.2.2 Effemeridi di Piccoli Corpi	112
11.2.3 Effemeridi di Veicoli Spaziali	112
11.3 Rappresentazioni di Effemeridi	112
11.3.1 Formato Tabulato	112
11.3.2 Rappresentazione Polinomiale	113
11.3.3 Polinomi di Chebyshev	113
11.3.4 Serie di Fourier	113
11.4 Metodi di Interpolazione	114
11.4.1 Interpolazione Lineare	114
11.4.2 Interpolazione di Lagrange	114
11.4.3 Interpolazione di Hermite	114
11.4.4 Interpolazione Spline	115
11.5 Sistema SPICE	115
11.5.1 Panoramica	115
11.5.2 File SPK	116
11.5.3 ID NAIF	116
11.6 Effemeridi Planetarie	117
11.6.1 JPL Development Ephemerides	117

11.6.2 VSOP87	117
11.6.3 Confronto	118
11.7 Correzioni Tempo-Luce	118
11.7.1 Posizione Geometrica vs Apparente	118
11.7.2 Correzione Iterativa	119
11.7.3 Implementazione	119
11.7.4 Aberrazione	120
11.8 Generazione Pratica Effemeridi	120
11.8.1 Considerazioni di Progetto	120
11.8.2 Flusso Generazione	120
11.8.3 Validazione	121
11.9 Memorizzazione Efficiente	122
11.9.1 Formati Binari	122
11.9.2 Spaziatura Adattativa	122
11.10 Riepilogo	122
III Determinazione Orbitale	125
12 Osservazioni	127
12.1 Introduzione	127
12.2 Tipi di Osservazioni	127
12.2.1 Astrometria Ottica	127
12.2.2 Osservazioni Radar	128
12.2.3 Tracciamento Veicoli Spaziali	129
12.3 Modello Osservazione Astrometrica	129
12.3.1 Trasformazione Coordinate	129
12.3.2 Coordinate Sferiche	129
12.3.3 Correzione Tempo-Luce	130
12.3.4 Aberrazione Stellare	130
12.3.5 Rifrazione Atmosferica	130
12.4 Coordinate Osservatorio	131
12.4.1 ITRF e Codici Osservatorio	131
12.4.2 Posizione Osservatorio Geocentrica	132
12.4.3 Rotazione a Sistema Inerziale	132
12.5 Parametri Orientamento Terra	132
12.5.1 Moto Polare	132

12.5.2 UT1-UTC	133
12.5.3 Precessione e Nutazione	133
12.6 Formato Osservazione MPC	133
12.6.1 Formato 80 Colonne	133
12.6.2 Formato ADES	134
12.7 Pesi Osservazioni	135
12.7.1 Schemi di Pesatura	135
12.7.2 Pesatura Empirica	135
12.7.3 Riduzione Peso Outlier	135
12.8 Parziali Osservazioni	136
12.8.1 Definizione	136
12.8.2 Regola Catena	136
12.8.3 Parziali Geometriche	136
12.8.4 Implementazione	136
12.9 Qualità Dati	137
12.9.1 Accuratezza Timing	137
12.9.2 Catalogo Astrometrico	138
12.9.3 Sistematici Sito-Specifici	138
12.10 Esempio Pratico	138
12.10.1 Caricamento Osservazioni MPC	138
12.10.2 Calcolo Osservazioni Previste	139
12.11 Riepilogo	140
 13 Determinazione dell'Orbita Iniziale	 143
13.1 Introduzione	143
13.2 Il Problema IOD	143
13.2.1 Osservazioni Solo Angolari	143
13.2.2 Linea di Vista	143
13.3 Metodo di Gauss	144
13.3.1 Contesto Storico	144
13.3.2 Idea di Base	144
13.3.3 Coefficienti di Lagrange	144
13.3.4 Equazione Scalare di Lagrange	145
13.3.5 Algoritmo	145
13.4 Implementazione	146
13.5 Problema dell'Arco Troppo Corto	148

13.5.1 Sfida	148
13.5.2 Vincoli Aggiuntivi	148
13.6 Metodo di Laplace	148
13.6.1 Approccio Alternativo	148
13.6.2 Equazioni	149
13.7 Metodi Moderni	149
13.7.1 Regione Ammissibile	149
13.7.2 Minimi Quadrati Vincolati	149
13.8 Valutazione della Qualità	149
13.8.1 Incertezza Orbitale	149
13.8.2 Validazione	150
13.9 Esempio: Asteroide Appena Scoperto	150
13.10 Sommario	151
 14 Correzione Differenziale	 153
14.1 Introduzione	153
14.2 Il Problema dei Minimi Quadrati	153
14.2.1 Equazione di Osservazione	153
14.2.2 Linearizzazione	154
14.2.3 Equazioni Normali	154
14.3 Calcolo delle Derivate Parziali	154
14.3.1 Regola della Catena con STM	154
14.3.2 Derivate Geometriche	155
14.3.3 Derivate Complete	155
14.4 Algoritmo	156
14.5 Criteri di Convergenza	156
14.5.1 Correzione allo Stato	156
14.5.2 Variazione RMS	156
14.5.3 Iterazioni Massime	157
14.6 Strategia di Pesatura	157
14.6.1 Pesi Empirici	157
14.6.2 Pesatura Robusta	157
14.7 Matrice di Covarianza	157
14.7.1 Incertezza Formale	157
14.7.2 Correlazione	158
14.7.3 Incertezza Propagata	158

14.8 Implementazione	158
14.9 Esempio: Asteroide 203 Pompeja	161
14.9.1 Definizione del Problema	161
14.9.2 Risultati	161
14.9.3 Interpretazione	163
14.10 Risoluzione Problemi	164
14.10.1 Non-Convergenza	164
14.10.2 RMS Elevato	164
14.10.3 Residui Piccoli ma Orbita Sbagliata	165
14.11 Sommario	165
15 Analisi dei Residui	167
15.1 Introduzione	167
15.2 Tipi di Residui	167
15.2.1 Residui Post-Fit	167
15.2.2 Residui Normalizzati	168
15.2.3 Residui Standardizzati	168
15.3 Metriche di Qualità	168
15.3.1 Radice Media Quadratica (RMS)	168
15.3.2 RMS Pesato	169
15.3.3 Test Chi-Quadro	169
15.3.4 Residuo Massimo	169
15.4 Grafici dei Residui	169
15.4.1 Residui vs. Tempo	169
15.4.2 Residui vs. Osservatorio	170
15.4.3 Residui vs. Magnitudine	170
15.4.4 Residui RA vs. Dec	170
15.4.5 Grafico di Probabilità Normale	170
15.5 Rilevamento Outlier	171
15.5.1 Metodo Soglia	171
15.5.2 Criterio di Chauvenet	171
15.5.3 Deviazione Assoluta Mediana (MAD)	171
15.5.4 Rimozione Iterativa Outlier	171
15.6 Diagnosi Errori Sistematici	172
15.6.1 Errori di Temporizzazione	172
15.6.2 Bias Catalogo	172

15.6.3 Errore Coordinate Osservatorio	172
15.6.4 Correzione Light-Time	172
15.6.5 Inadeguatezza Modello Forze	172
15.7 Esempio di Analisi	173
15.7.1 Output Esempio	175
15.8 Miglioramento Qualità Orbita	176
15.8.1 Quando RMS è Troppo Grande	176
15.8.2 Quando $\chi^2_{\text{rid}} \gg 1$	176
15.8.3 Quando Poche Osservazioni Disponibili	177
15.9 Reporting Risultati	177
15.9.1 Statistiche Sommarie	177
15.9.2 Interpretazione Covarianza	177
15.9.3 Valutazione Arco Orbitale	177
15.10 Sommario	178
IV Implementazione della Libreria AstDyn	179
16 Architettura Software	181
16.1 Introduzione	181
16.2 Principi di Design	181
16.2.1 Separazione delle Responsabilità	181
16.2.2 Design Basato su Interfacce	182
16.2.3 Header-Only vs. Compilato	182
16.3 Organizzazione dei Moduli	183
16.3.1 Struttura delle Directory	183
16.3.2 Organizzazione dei Namespace	185
16.4 Componenti Core	186
16.4.1 Costanti e Tipi	186
16.4.2 Versione e Configurazione	187
16.5 Gestione delle Dipendenze	187
16.5.1 Dipendenze Esterne	187
16.5.2 Sistema di Build CMake	188
16.6 Gestione degli Errori	189
16.6.1 Strategia	189
16.6.2 Logging	190
16.7 Gestione della Memoria	190

16.7.1	Ownership	190
16.7.2	Dataset Grandi	191
16.8	Threading e Parallelismo	191
16.8.1	Stato Attuale	191
16.8.2	Piani Futuri	191
16.9	Strategia di Testing	192
16.9.1	Test Unitari	192
16.9.2	Test di Integrazione	192
16.9.3	Benchmark Prestazionali	192
16.10	Documentazione	193
16.10.1	Documentazione Inline	193
16.10.2	Documentazione Esterna	193
16.11	Riepilogo	194
17	Moduli Core	195
17.1	Introduzione	195
17.2	Elementi Orbitali	195
17.2.1	KeplerianElements	195
17.2.2	CometaryElements	197
17.2.3	CartesianState	198
17.3	Modelli di Forza	198
17.3.1	Interfaccia ForceModel	198
17.3.2	Gravità a Massa Puntiforme	199
17.3.3	Modello di Forza Combinato	200
17.4	Integrazione Numerica	200
17.4.1	Interfaccia Integrator	200
17.4.2	Runge-Kutta-Fehlberg 7(8)	201
17.5	Propagazione Orbitale	203
17.5.1	Classe Propagator	203
17.6	Osservazioni	205
17.6.1	Classe Observation	205
17.6.2	MPC Reader	206
17.7	Database Osservatori	207
17.7.1	ObservatoryCoordinates	207
17.8	Riepilogo	208

18 Sistema Parser	211
18.1 Introduzione	211
18.1.1 Formati Supportati	211
18.2 Interfaccia Parser	211
18.2.1 Classe Base IParser	211
18.2.2 Vantaggi del Design	212
18.3 Parser OrbFit.eq1	212
18.3.1 Specifica del Formato	212
18.3.2 Implementazione	213
18.3.3 Utilizzo	215
18.4 Factory Parser	216
18.4.1 Pattern Factory	216
18.4.2 Utilizzo	218
18.5 Parser Osservazioni MPC	218
18.5.1 Formato 80 Colonne	218
18.5.2 MPCObservationParser	219
18.6 Parser Personalizzati	220
18.6.1 Creazione di un Nuovo Parser	220
18.6.2 Formato JSON Esempio	222
18.7 Gestione Errori	222
18.7.1 Categorie di Errori	222
18.7.2 Gestione Robusta	223
18.7.3 Validazione	223
18.8 Testing	224
18.8.1 Unit Test	224
18.9 Sommario	224
19 Riferimento API	227
19.1 Panoramica	227
19.1.1 Organizzazione	227
19.2 Costanti Core	228
19.2.1 astdyn::constants	228
19.3 Utility Matematiche	229
19.3.1 astdyn::math	229
19.4 Sistemi Temporali	230
19.4.1 astdyn::time::TimeConverter	230

19.5 Elementi Orbitali	231
19.5.1 astdyn::orbit::KeplerianElements	231
19.6 Propagazione Orbitale	232
19.6.1 astdyn::propagation::OrbitPropagator	232
19.7 Osservazioni	233
19.7.1 astdyn::observations::Observation	233
19.8 Determinazione Orbitale	233
19.8.1 astdyn::orbit_determination::DifferentialCorrector	233
19.9 Parser I/O	234
19.9.1 astdyn::io::ParserFactory	234
19.10 Interfacce Effemeridi	234
19.10.1 astdyn::ephemeris::IEphemeris	234
19.10.2 Implementazioni	235
19.11 Modelli di Forza	236
19.11.1 astdyn::forces::ForceModel	236
19.12 Integratori Numerici	237
19.12.1 astdyn::integration::IIntegrator	237
19.12.2 Implementazioni Disponibili	237
19.13 Utility Analisi	238
19.13.1 astdyn::analysis::ResidualAnalyzer	238
19.14 Esempio Completo	238
19.14.1 Workflow Determinazione Orbitale	238
19.15 Sommario	240
 20 Esempi e Tutorial	 241
20.1 Introduzione	241
20.1.1 Prerequisiti	241
20.2 Esempio 1: Propagazione Orbitale Base	241
20.2.1 Obiettivo	241
20.2.2 Codice	241
20.2.3 Compilazione	245
20.2.4 Output Atteso	245
20.3 Esempio 2: Generazione Effemeridi	246
20.3.1 Obiettivo	246
20.3.2 Codice	246
20.4 Esempio 3: Determinazione Orbitale	248

20.4.1	Obiettivo	248
20.4.2	Codice	248
20.5	Esempio 4: Lettura Osservazioni MPC	251
20.5.1	Obiettivo	251
20.5.2	Codice	252
20.6	Esempio 5: Matrice Transizione Stato	253
20.6.1	Obiettivo	253
20.6.2	Codice	253
20.7	Esempio 6: Modello Forza Personalizzato	255
20.7.1	Obiettivo	255
20.7.2	Codice	255
20.8	Compilazione ed Esecuzione Esempi	258
20.8.1	CMakeLists.txt	258
20.8.2	Comandi Compilazione	259
20.9	Sommario	259
V	Validazione e Applicazioni	261
21	Validazione e Testing	263
21.1	Introduzione	263
21.1.1	Strategia di Validazione	263
21.2	Framework Unit Testing	263
21.2.1	Integrazione Google Test	263
21.3	Test Accuratezza Numerica	264
21.3.1	Problema Due Corpi	264
21.4	Confronto con OrbFit	264
21.4.1	Metodologia	264
21.4.2	Confronto Propagazione	265
21.4.3	Confronto Determinazione Orbitale	265
21.5	Confronto JPL Horizons	266
21.5.1	Risultati	266
21.6	Stress Testing	266
21.6.1	Eccentricita' Estreme	266
21.6.2	Integrazione Lungo Termine	266
21.7	Validazione Prestazioni	267
21.7.1	Velocita' Integrazione	267

INDICE

21.8 Integrazione Continua	267
21.9 Limitazioni Note	267
21.10 Sommario	268
22 Caso di Studio: (203) Pompeja	269
22.1 Introduzione	269
22.1.1 Perche' Pompeja?	269
22.2 Asteroide (203) Pompeja	269
22.2.1 Proprieta' Fisiche	269
22.2.2 Caratteristiche Orbitali	270
22.3 Dati Osservativi	270
22.3.1 Sorgente Dati	270
22.3.2 Sommario Osservazioni	271
22.4 Determinazione Orbitale Iniziale	271
22.4.1 Metodo Gauss	271
22.4.2 Soluzione Iniziale	271
22.5 Correzione Differenziale	272
22.5.1 Storia Iterazioni	272
22.5.2 Elementi Orbitali Finali	272
22.6 Analisi Residui	273
22.6.1 Statistiche Residui	273
22.6.2 Distribuzione Residui	273
22.7 Confronto con Soluzione Riferimento	273
22.7.1 Riferimento OrbFit	273
22.7.2 Effemeridi JPL Horizons	274
22.8 Covarianza e Incertezze	274
22.8.1 Matrice Covarianza Parametri	274
22.8.2 Propagazione Incertezza Posizione	275
22.9 Metriche Prestazioni	275
22.9.1 Costo Computazionale	275
22.9.2 Uso Memoria	275
22.10 Conclusioni	276
23 Benchmark Prestazioni	277
23.1 Introduzione	277
23.1.1 Ambiente Benchmark	277

23.2 Prestazioni Propagazione Orbitale	277
23.2.1 Propagazione Singola	277
23.2.2 Propagazione Batch	278
23.2.3 Effetto Complessita' Modello Forze	278
23.3 Prestazioni Determinazione Orbitale	279
23.3.1 Timing Correzione Differenziale	279
23.3.2 Scalabilita' con Numero Osservazioni	279
23.4 Confronto con Altri Strumenti	280
23.4.1 OrbFit 5.0.5	280
23.4.2 Strumenti Python	280
23.5 Uso Memoria	280
23.5.1 Allocazione Heap	280
23.5.2 Scalabilita' con Dimensione Problema	281
23.6 Sommario	281
 VI Prospettive Future	 283
 24 Sviluppi Futuri	 285
24.1 Introduzione	285
24.2 Forze Non Gravitazionali	285
24.2.1 Pressione di Radiazione Solare	285
24.2.2 Degassamento Cometario	286
24.2.3 Relatività Generale	287
24.3 Propagazione delle Incertezze	288
24.3.1 Propagazione della Covarianza	288
24.3.2 Metodi Monte Carlo	289
24.4 Gestione degli Incontri Ravvicinati	290
24.4.1 Tecniche di Regolarizzazione	290
24.4.2 Analisi di Incontri Iperbolici	291
24.5 Elaborazione Parallelia	291
24.5.1 Parallelizzazione OpenMP	291
24.5.2 Accelerazione GPU	292
24.6 Integratori Aggiuntivi	292
24.6.1 Integratori Simplettici	292
24.6.2 Metodi Impliciti	293
24.7 Binding Python	293

24.7.1 Interfaccia pybind11	293
24.7.2 Distribuzione del Pacchetto	294
24.8 Integrazione con Machine Learning	294
24.8.1 Modelli Surrogati con Reti Neurali	294
24.9 Osservazioni Avanzate	295
24.9.1 Osservazioni Radar	295
24.9.2 Astrometria Gaia	296
24.10 Servizio Web / Deployment Cloud	296
24.10.1 API RESTful	296
24.10.2 Interfaccia Web	297
24.11 Integrazione Pipeline Dati	297
24.11.1 Elaborazione Automatizzata di Survey	297
24.12 Modelli di Errore Migliorati	297
24.12.1 Stima Robusta	297
24.13 Supporto Multi-Piattaforma	298
24.13.1 Build WebAssembly	298
24.13.2 Piattaforme Mobile	299
24.14 Roadmap di Sviluppo	299
24.14.1 Versione 1.1 (Q2 2026)	299
24.14.2 Versione 1.2 (Q4 2026)	299
24.14.3 Versione 2.0 (2027)	299
24.15 Contributi della Comunità	300
24.15.1 Sviluppo Open Source	300
24.15.2 Citazione di AstDyn	300
24.16 Direzioni di Ricerca	301
24.16.1 Algoritmi Innovativi	301
24.16.2 Applicazioni Interdisciplinari	301
24.17 Riepilogo	301
Riferimenti	303
24.18 Libri di Testo	303
24.18.1 Meccanica Celeste	303
24.18.2 Metodi Numerici	303
24.19 Articoli Scientifici	304
24.19.1 Determinazione Orbitale	304
24.19.2 Sistemi Temporali	304

24.20 Software e Strumenti	304
24.20.1 Software Determinazione Orbitale	304
24.20.2 Effemeridi	304
24.20.3 Osservazioni	305
24.21 Standard e Convenzioni	305
24.22 Risorse Online	305
Appendici	307
24.23 Costanti Fisiche	307
24.23.1 Costanti Fondamentali	307
24.23.2 Parametri Sistema Solare	307
24.24 Elementi Orbitali Planetari	308
24.24.1 Elementi Medi J2000.0	308
24.25 Conversioni Unità	308
24.25.1 Conversioni Angolari	308
24.25.2 Conversioni Temporali	308
24.25.3 Conversioni Distanza	309
24.26 Formule Utili	309
24.26.1 Problema Due Corpi	309
24.26.2 Anomalie	309
24.26.3 Trasformazioni Coordinate	310
24.27 Codici Osservatorio MPC	310
24.27.1 Osservatori Principali	310
24.28 Acronimi e Abbreviazioni	311
24.29 Convenzioni Notazione	311
24.29.1 Vettori e Matrici	311
24.29.2 Elementi Orbitali	311
24.29.3 Sistemi Coordinate	312

Elenco delle figure

1 *	i
1.1 Architettura della libreria AstDyn con design stratificato	6

2.1	Accumulo di secondi intercalari dal 1972	15
2.2	Relazioni tra le principali scale temporali	17
3.1	Sistema di coordinate equatoriali mostrando ascensione retta (α) e declinazione (δ). L'obliquità $\varepsilon \approx 23.4^\circ$	22
3.2	Precessione degli equinozi in 50 anni	25
4.1	Sistema di riferimento equatoriale J2000.0 che mostra i tre assi e la definizione dell'ascensione retta (α) e della declinazione (δ).	29
4.2	La precessione causa il cambiamento dell'orientamento del piano equatoriale nel tempo. Il punto vernali γ si sposta verso ovest lungo l'eclittica a circa 50.3 arcosecondi all'anno.	31
5.1	Elementi orbitali kepleriani classici. Il piano orbitale (ellisse rossa) è inclinato rispetto al piano di riferimento. Il nodo ascendente è dove l'orbita attraversa il piano di riferimento andando verso nord.	39
6.1	Orbite coniche per diverse eccentricità. Il Sole è in uno dei fuochi di ogni conica.	50
6.2	Relazione tra anomalia vera ν e anomalia eccentrica E in un'orbita ellittica.	52

Elenco delle tabelle

2.1	Secondi intercalari recenti (tabella parziale)	18
6.1	Classificazione delle coniche orbitali per eccentricità ed energia.	49
7.1	Magnitudini tipiche delle accelerazioni perturbative per asteroidi.	60
7.2	Confronto tra integrazione numerica e teoria analitica delle perturbazioni.	68
8.1	Confronto dei metodi di integrazione numerica.	80
9.1	Modelli di forza raccomandati per diversi tipi di oggetti.	96

10.1 Costo computazionale propagazione STM. N_p è il numero di parametri.	103
11.1 Esempio effemeride tabulata con spaziatura di 1 giorno.	112
11.2 Confronto effemeridi planetarie.	118
11.3 Requisiti effemeride per diverse applicazioni.	120
12.1 Incertezze osservative tipiche.	135
21.1 Copertura unit test per modulo	264
21.2 Errore posizione dopo un periodo (varie eccentricità)	264
21.3 Differenza posizione: AstDyn vs OrbFit	265
21.4 Differenze elementi orbitali: AstDyn vs OrbFit	265
21.5 Errore RMS posizione vs JPL Horizons (1 anno)	266
21.6 Successo integrazione vs eccentricità'	266
21.7 Timing integrazione (Intel i7-10700K, thread singolo)	267
22.1 Dataset osservazioni Pompeja	271
22.2 Osservazioni selezionate per IOD Gauss	271
22.3 Elementi orbitali iniziali metodo Gauss	271
22.4 Convergenza correzione differenziale	272
22.5 Soluzione orbitale finale per Pompeja	272
22.6 Residui osservazioni	273
22.7 Confronto AstDyn vs OrbFit	273
22.8 Differenza posizione: AstDyn vs JPL (60 giorni)	274
22.9 Matrice correlazione (elementi selezionati)	274
22.10 Incertezza posizione vs tempo	275
22.11 Suddivisione timing (Intel i7-10700K, core singolo)	275
23.1 Timing propagazione: arco 60 giorni	278
23.2 Statistiche propagazione batch	278
23.3 Timing vs modello forze (propagazione 60 giorni)	278
23.4 Suddivisione correzione differenziale	279
23.5 Prestazioni vs numero osservazioni	279
23.6 Timing AstDyn vs OrbFit	280
23.7 Confronto prestazioni linguaggio (propagazione 60 giorni)	280
23.8 Footprint memoria per componente	281
23.9 Uso memoria vs numero osservazioni	281

ELENCO DELLE TABELLE

24.1 Costanti fisiche fondamentali	307
24.2 Masse planetarie (parametri GM in AU ³ /giorno ²)	307
24.3 Elementi orbitali planetari (epoca J2000.0)	308
24.4 Fattori conversione angolare	308
24.5 Fattori conversione temporale	308
24.6 Fattori conversione distanza	309
24.7 Codici osservatorio MPC selezionati	310
24.8 Acronimi comuni	311

Parte I

Fondamenti Teorici della Meccanica Celeste

Capitolo 1

Introduzione

1.1 Cos'è la Meccanica Celeste?

La meccanica celeste è il ramo dell'astronomia che studia i moti dei corpi celesti sotto l'influenza delle forze gravitazionali. Fornisce il quadro matematico e fisico per comprendere:

- Le orbite di pianeti, lune, asteroidi e comete
- La progettazione di traiettorie spaziali e l'analisi di missioni
- La stabilità a lungo termine del sistema solare
- Gli effetti mareali e la dinamica rotazionale
- La formazione ed evoluzione dei sistemi planetari

Il campo ha una storia illustre, che inizia con le leggi empiriche del moto planetario di Johannes Kepler (1609-1619) e la legge di gravitazione universale di Isaac Newton (1687). Newton dimostrò che le leggi di Kepler potevano essere derivate da principi fisici fondamentali, segnando la nascita della meccanica celeste teorica.

1.1.1 Il Problema dei Due Corpi

La pietra angolare della meccanica celeste è il *problema dei due corpi*: determinare il moto di due masse puntiformi che interagiscono esclusivamente attraverso l'attrazione gravitazionale reciproca. Questo problema ha un'elegante soluzione

analitica, espressa in termini di sei *elementi orbitali* che specificano completamente l’orbita.

Consideriamo due corpi con masse m_1 e m_2 , separati da una distanza r . La legge di gravitazione di Newton afferma:

$$F = G \frac{m_1 m_2}{r^2} \quad (1.1)$$

dove $G = 6.67430 \times 10^{-11} \text{ m}^3 \text{ kg}^{-1} \text{ s}^{-2}$ è la costante gravitazionale.

Per un corpo piccolo di massa m che orbita attorno a un corpo molto più grande di massa M (come un asteroide che orbita attorno al Sole), possiamo approssimare il sistema come un problema a un corpo con il corpo massiccio all’origine. L’equazione del moto diventa:

$$\ddot{\mathbf{r}} = -\frac{\mu}{r^3} \mathbf{r} \quad (1.2)$$

dove $\mu = GM$ è il parametro gravitazionale e \mathbf{r} è il vettore posizione del corpo piccolo.

1.1.2 Il Problema degli N Corpi

In realtà, i corpi celesti esistono in sistemi con molteplici oggetti gravitanti. Il sistema solare, per esempio, contiene il Sole, otto pianeti maggiori, numerose lune, asteroidi e comete—tutti che esercitano forze gravitazionali gli uni sugli altri. Questo è il *problema degli N corpi*.

A differenza del problema dei due corpi, il problema degli N corpi non ha una soluzione analitica generale per $N \geq 3$. Invece, dobbiamo ricorrere a:

1. **Teoria delle perturbazioni:** Trattare le forze aggiuntive come piccole correzioni a una soluzione a due corpi
2. **Integrazione numerica:** Calcolare le orbite passo dopo passo usando i computer
3. **Soluzioni speciali:** Risultati analitici per casi ristretti (es. punti di Lagrange)

La libreria AstDyn implementa tutti e tre gli approcci, con enfasi sulla teoria delle perturbazioni e sull’integrazione numerica ad alta precisione.

1.2 Panoramica della Libreria AstDyn

1.2.1 Filosofia di Progettazione

La libreria AstDyn è costruita su diversi principi fondamentali:

Accuratezza I metodi numerici sono scelti e calibrati per alta precisione, validati contro software consolidati

Modularità I componenti sono debolmente accoppiati, permettendo agli utenti di impiegare solo le funzionalità necessarie

Chiarezza Il codice è documentato con riferimenti a formulazioni matematiche e letteratura

Prestazioni Gli algoritmi sono ottimizzati usando caratteristiche moderne di C++ senza sacrificare la leggibilità

Estensibilità L'architettura supporta l'aggiunta di nuovi integratori, modelli di forza e tipi di osservazione

1.2.2 Caratteristiche Principali

La libreria fornisce:

- **Sistemi temporali:** Conversioni tra UTC, TAI, TT, TDB con modelli accurati di ΔT
- **Sistemi di coordinate:** Trasformazioni tra sistemi eclittico, equatoriale e planetario
- **Elementi orbitali:** Rappresentazioni kepleriane, cartesiane, equinoziali e di Delaunay
- **Integrazione numerica:** Runge-Kutta, Adams-Bashforth-Moulton e metodi adattativi
- **Modelli di forza:** Gravitazione a N corpi, perturbazioni asteroidali, effetti relativistici
- **Propagazione orbitale:** Integrazione avanti/indietro con matrice di transizione di stato

- **Determinazione dell'orbita iniziale:** Metodo di Gauss per tre osservazioni
- **Correzione differenziale:** Adattamento dell'orbita ai minimi quadrati alle osservazioni astrometriche
- **Effemeridi:** Posizioni planetarie usando VSOP87 e DE440/441
- **I/O dati:** Parser per OrbFit (.eq1, .rwo), MPC e formati personalizzati

1.2.3 Architettura Software

La Figura 1.1 illustra l'architettura di alto livello:

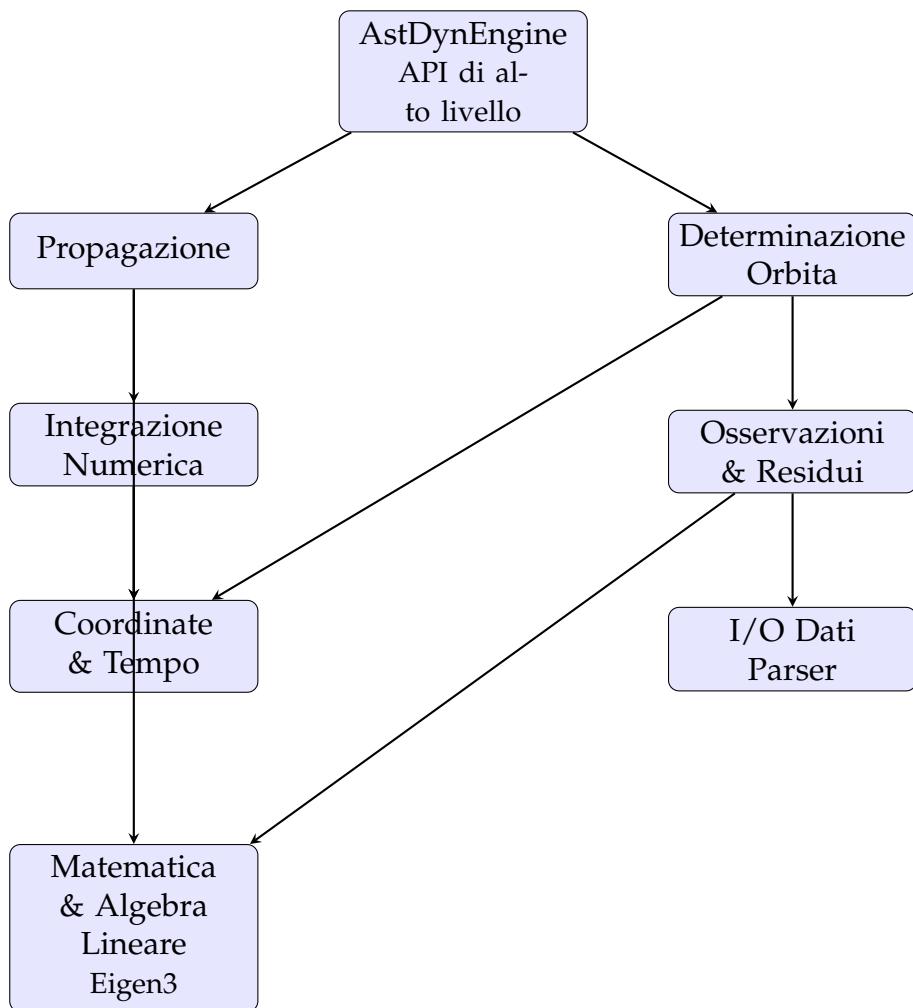


Figura 1.1: Architettura della libreria AstDyn con design stratificato

L'architettura segue un design stratificato:

1. **Livello fondamentale:** Utilità matematiche e algebra lineare (Eigen3)

2. **Livello core:** Sistemi temporali, trasformazioni di coordinate, elementi orbitali
3. **Livello algoritmico:** Integrazione numerica, gestione osservazioni
4. **Livello applicativo:** Propagazione orbitale, determinazione orbita
5. **Livello interfaccia:** API di alto livello (AstDynEngine), parser dati

1.2.4 Dipendenze

AstDyn si basa su librerie consolidate:

Eigen3 Operazioni di algebra lineare (matrici, vettori, decomposizioni)

Boost Filesystem, date-time, opzioni programma

GoogleTest Framework per unit testing (opzionale)

Tutte le dipendenze sono ampiamente disponibili e attivamente mantenute.

1.3 Applicazioni

La libreria AstDyn supporta varie applicazioni:

1.3.1 Determinazione di Orbite Asteroidali

Date osservazioni astrometriche (ascensione retta e declinazione) di un asteroide da telescopi terrestri, determinare la sua orbita eliocentrica. Questo è cruciale per:

- Prevedere posizioni future per campagne osservative
- Valutare il rischio di collisione con la Terra
- Pianificare missioni spaziali
- Comprendere popolazioni e dinamica degli asteroidi

Esempio: Il Capitolo ?? presenta un'analisi completa dell'asteroide 203 Pompeja utilizzando 100 osservazioni recenti, ottenendo residui RMS di 0.66 arcosecondi.

1.3.2 Analisi di Traiettorie Spaziali

Progettare e analizzare traiettorie spaziali per:

- Trasferimenti interplanetari
- Manovre orbitali
- Operazioni di station-keeping
- Analisi di avvicinamenti ravvicinati

La propagazione ad alta precisione della libreria e la capacità di calcolare matrici di transizione di stato la rendono adatta per la progettazione preliminare di missioni.

1.3.3 Evoluzione Orbitale a Lungo Termine

Studiare il comportamento a lungo termine dei corpi minori sotto perturbazioni planetarie:

- Evoluzione secolare degli elementi orbitali
- Identificazione di risonanze
- Analisi di caos e stabilità
- Stima della probabilità di impatto

1.3.4 Strumento Educativo

La libreria serve come risorsa educativa per studenti che apprendono:

- Implementazione pratica di algoritmi da manuale
- Metodi numerici in astrodinamica
- Ingegneria del software per calcolo scientifico
- Tecniche moderne di programmazione C++

1.4 Validazione e Accuratezza

Un punto di forza chiave di AstDyn è la rigorosa validazione contro software consolidati:

- **OrbFit**: Il confronto dei risultati di determinazione orbitale per l'asteroide 203 Pompeja mostra un accordo di $\Delta a = 578$ km, $\Delta e = 0.0006$, $\Delta i = 5''$
- **JPL Horizons**: I confronti di effemeridi validano i modelli di perturbazione planetaria
- **Soluzioni analitiche**: La propagazione a due corpi è testata contro le formule kepleriane

Studi di validazione dettagliati sono presentati nel Capitolo [21](#).

1.5 Per Iniziare

1.5.1 Installazione

La libreria può essere compilata usando CMake:

```
1 git clone https://github.com/manvalan/ITALOccultLibrary.git
2 cd ITALOccultLibrary/astdyn
3 mkdir build && cd build
4 cmake .. -DCMAKE_BUILD_TYPE=Release
5 make -j8
```

Listing 1.1: Compilazione di AstDyn

Questo produce:

- `libastdyn.a` (libreria statica, 1.5 MB, 1232 simboli)
- `libastdyn.dylib` (libreria condivisa, 877 KB)

1.5.2 Esempio Rapido

Un esempio minimale di propagazione orbitale:

```
1 #include <astdyn/AstDyn.hpp>
2 using namespace astdyn;
3
4 int main() {
5     // Definire elementi orbitali (asteroide in AU,
6     // radianti)
7     propagation::KeplerianElements orbit;
8     orbit.epoch = 61000.0;    // MJD TDB
9     orbit.a = 2.7;           // semiasse maggiore (AU)
10    orbit.e = 0.15;          // eccentricita
11    orbit.i = 10.0 * constants::DEG_TO_RAD;
12    orbit.Omega = 80.0 * constants::DEG_TO_RAD;
13    orbit.omega = 73.0 * constants::DEG_TO_RAD;
14    orbit.M = 45.0 * constants::DEG_TO_RAD;
15    orbit.gm = constants::GMS; // GM del Sole
16
17    // Creare propagatore
18    propagation::Propagator prop;
19
20    // Propagare 1 anno in avanti
21    double target_mjd = orbit.epoch + 365.25;
22    auto result = prop.propagate_keplerian(orbit,
23                                         target_mjd);
24
25    // Stampare risultati
26    std::cout << "Posizione: " << result.position.transpose
27        () << " AU\n";
28    std::cout << "Velocita: " << result.velocity.transpose
29        () << " AU/giorno\n";
30
31    return 0;
32}
```

Listing 1.2: Propagazione orbitale di base

Esempi più completi sono forniti nel Capitolo 20.

1.6 Organizzazione dei Capitoli Rimanenti

Il resto di questo manuale è organizzato come segue:

Capitoli 2-7 (Parte I) stabiliscono le fondamenta teoriche: sistemi temporali, coordinate, elementi orbitali, dinamica a due corpi e perturbazioni.

Capitoli 8-11 (Parte II) descrivono metodi numerici: algoritmi di integrazione, propagazione, matrici di transizione di stato e calcolo di effemeridi.

Capitoli 12-15 (Parte III) coprono la determinazione orbitale: modelli di osservazione, determinazione dell'orbita iniziale, correzione differenziale e analisi dei residui.

Capitoli 16-20 (Parte IV) documentano l'implementazione della libreria: architettura, moduli core, parser, riferimento API ed esempi.

Capitoli 21-23 (Parte V) presentano studi di validazione, applicazioni del mondo reale e benchmark delle prestazioni.

Ogni capitolo include derivazioni matematiche, note di implementazione ed esempi di codice funzionanti per collegare teoria e pratica.

Capitolo 2

Sistemi di Tempo nella Meccanica Celeste

La misurazione del tempo è fondamentale per la meccanica celeste, ma sorprendentemente complessa. Diverse applicazioni richiedono diverse scale temporali, ciascuna con definizioni e casi d'uso specifici. Questo capitolo descrive i sistemi temporali implementati in AstDyn e le loro interconversioni.

2.1 Perché Sistemi Temporali Multipli?

Un approccio ingenuo potrebbe usare il tempo civile ordinario (UTC) per tutti i calcoli. Tuttavia, questo è inadeguato per la meccanica celeste di precisione a causa di:

- **Rotazione irregolare della Terra:** La lunghezza di un giorno varia a causa dell'attrito mareale, effetti atmosferici e accoppiamento nucleo-mantello
- **Secondi intercalari:** UTC include salti discontinui per rimanere sincronizzato con la rotazione terrestre
- **Effetti relativistici:** Il tempo scorre diversamente in diversi potenziali gravitazionali
- **Requisiti di precisione:** Accuratezza sub-secondo su secoli richiede un cronometraggio accurato

2.2 Numero di Giorno Giuliano

Prima di discutere le scale temporali specifiche, introduciamo il sistema del Giorno Giuliano (JD), un conteggio continuo di giorni dal mezzogiorno UTC del 1 gennaio 4713 a.C. (calendario giuliano prolettico).

Definizione 2.1 (Giorno Giuliano). Il Numero di Giorno Giuliano (JD) è il numero di giorni trascorsi dall'epoca JD 0.0 = 12:00 UT del 1 gennaio 4713 a.C.

Per esempio:

- 1 gennaio 2000, 12:00 TT = JD 2451545.0
- 26 novembre 2025, 00:00 UTC \approx JD 2460638.5

2.2.1 Giorno Giuliano Modificato

Per ridurre i requisiti di precisione numerica, il *Giorno Giuliano Modificato* (MJD) è comunemente usato:

$$\text{MJD} = \text{JD} - 2400000.5 \quad (2.1)$$

Questo sposta l'epoca al 17 novembre 1858, 00:00 UTC, e inizia i giorni a mezzanotte anziché a mezzogiorno. L'epoca di riferimento J2000.0 corrisponde a:

$$\text{MJD}_{\text{J2000}} = 51544.5 \quad (2.2)$$

AstDyn usa principalmente MJD per i calcoli interni.

2.3 Tempo Universale (UT)

Il Tempo Universale (UT) è basato sulla rotazione terrestre. Esistono diverse varianti:

2.3.1 UT0

UT0 è il Tempo Universale grezzo come misurato osservando le posizioni stellari. Varia a causa del movimento polare (oscillazione dell'asse di rotazione terrestre).

2.3.2 UT1

UT1 corregge UT0 per gli effetti del movimento polare:

$$\text{UT1} = \text{UT0} + \Delta\lambda \quad (2.3)$$

dove $\Delta\lambda$ tiene conto dello spostamento della longitudine dell'osservatore dovuto al movimento polare. UT1 rappresenta il vero angolo rotazionale della Terra.

2.3.3 UTC (Tempo Universale Coordinato)

UTC è lo standard del tempo civile, definito da orologi atomici ma mantenuto entro 0.9 secondi da UT1 inserendo *secondi intercalari*. La differenza è:

$$\Delta\text{UT} = \text{UT1} - \text{UTC} \quad (2.4)$$

I secondi intercalari sono annunciati dal Servizio Internazionale di Rotazione Terrestre (IERS) e tipicamente occorrono il 30 giugno o il 31 dicembre.

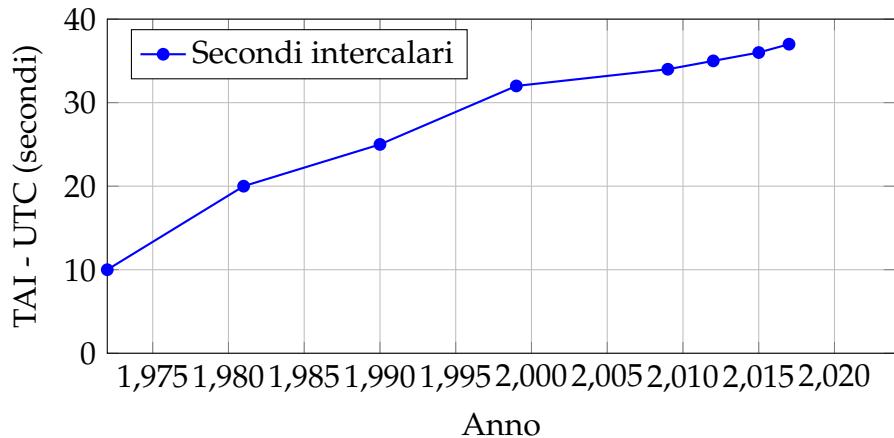


Figura 2.1: Accumulo di secondi intercalari dal 1972

2.4 Scale di Tempo Atomiche

2.4.1 TAI (Tempo Atomico Internazionale)

TAI è una scala temporale continua e uniforme definita da un insieme di orologi atomici in tutto il mondo. Non ha secondi intercalari e forma la base per altre scale temporali moderne.

La relazione con UTC è:

$$\text{TAI} = \text{UTC} + \Delta AT \quad (2.5)$$

dove ΔAT è il numero cumulativo di secondi intercalari (37 secondi a partire dal 2024).

2.4.2 TT (Tempo Terrestre)

Il Tempo Terrestre è la scala temporale teorica per osservazioni sulla superficie terrestre. È correlato a TAI da un offset costante:

$$\text{TT} = \text{TAI} + 32.184 \text{ s} \quad (2.6)$$

L'offset di 32.184 secondi è stato scelto per mantenere la continuità con la vecchia scala di Tempo delle Effemeridi (ET). TT è l'argomento temporale per le effemeridi geocentriche.

2.4.3 TDB (Tempo Dinamico Baricentrico)

Il Tempo Dinamico Baricentrico è la scala temporale per i calcoli al baricentro del sistema solare (centro di massa). A causa degli effetti relativistici generali, il tempo scorre a velocità diverse in diversi potenziali gravitazionali.

La relazione tra TDB e TT include termini sia periodici che secolari:

$$\text{TDB} = \text{TT} + 0.001658 \sin(g) + 0.000014 \sin(2g) \text{ secondi} \quad (2.7)$$

dove g è l'anomalia media dell'orbita terrestre attorno al Sole:

$$g = 357.53 + 0.9856003(JD - 2451545.0) \quad (2.8)$$

Questa correzione è tipicamente di pochi millisecondi ma si accumula su lunghi periodi di tempo.

2.5 Relazioni tra Scale Temporali

La Figura 2.2 illustra le relazioni tra le scale temporali:

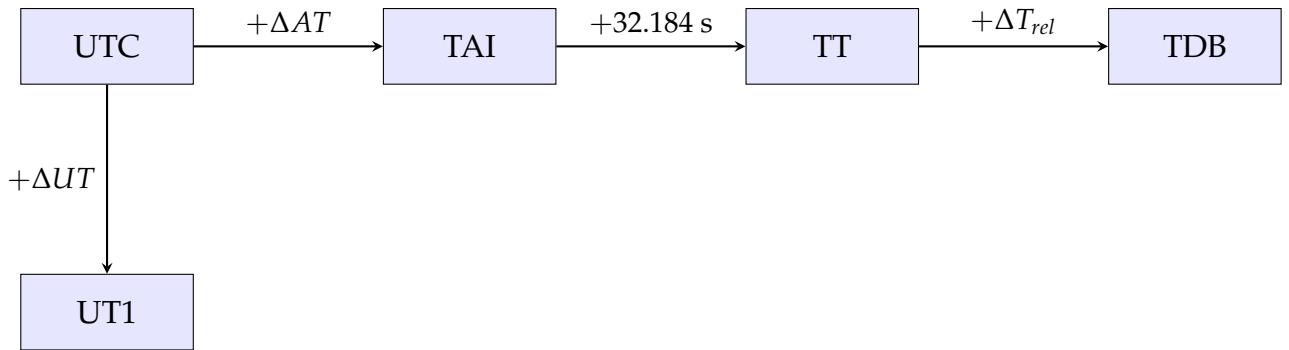


Figura 2.2: Relazioni tra le principali scale temporali

2.6 Implementazione in AstDyn

La classe `TimeScale` gestisce le conversioni tra i sistemi temporali:

```

1 #include <astdyn/time/TimeScale.hpp>
2 using namespace astdyn::time;
3
4 // Conversione UTC a TDB
5 double mjd_utc = 61000.0;
6 double mjd_tdb = TimeScale::utc_to_tdb(mjd_utc);
7
8 // TT a TAI
9 double mjd_tt = 61000.0;
10 double mjd_tai = TimeScale::tt_to_tai(mjd_tt);
11
12 // UT1 a UTC (richiede Delta_UT da IERS)
13 double delta_ut = 0.15; // secondi, da IERS Bulletin A
14 double mjd_ut1 = 61000.0;
15 double mjd_utc_computed = mjd_ut1 - delta_ut / 86400.0;
  
```

Listing 2.1: Conversioni di scale temporali

2.6.1 Tabella dei Secondi Intercalari

AstDyn mantiene una tabella interna di secondi intercalari, aggiornata periodicamente:

Tabella 2.1: Secondi intercalari recenti (tabella parziale)

Data	MJD	TAI-UTC (s)
2012-07-01	56109	35
2015-07-01	57204	36
2017-01-01	57754	37

2.6.2 Approssimazioni di ΔT

Per date storiche o previsioni future dove i secondi intercalari sono sconosciuti, formule empiriche approssimano $\Delta T = TT - UT$:

Prima del 1972 (adattamento polinomiale):

$$\Delta T \approx -20 + 32t^2 \text{ secondi} \quad (2.9)$$

dove t è in secoli dal 1820.

Dopo il 2015 (estrapolazione lineare):

$$\Delta T \approx 69.2 + 0.4 \times (y - 2015) \text{ secondi} \quad (2.10)$$

dove y è l'anno.

Queste approssimazioni hanno incertezze di diversi secondi e non dovrebbero essere usate per lavoro preciso.

2.7 Considerazioni Pratiche

2.7.1 Quale Scala Temporale Usare?

Osservazioni Usare UTC per registrare i tempi di osservazione (facilmente sincronizzabile con GPS)

Calcoli orbitali Convertire in TDB per l'integrazione numerica

Rotazione terrestre Usare UT1 per calcolare il tempo siderale e le coordinate topocentriche

Reporting Usare UTC per disseminare risultati agli osservatori

2.7.2 Requisiti di Precisione

Per la tipica determinazione di orbite asteroidali:

- Accuratezza posizionale: $\sim 0.1''$ (arcosecondo)
- Accuratezza temporale necessaria: ~ 0.01 s
- Effetto di 1 secondo di errore temporale: $\sim 15''$ in AR per asteroide della fascia principale

Pertanto, usare la scala temporale corretta e tenere conto dei secondi intercalari è essenziale.

2.7.3 Esempio: Catena di Conversione Temporale

Conversione completa da data civile a TDB:

```

1 // Input: data civile UTC
2 int year = 2025, month = 11, day = 26;
3 double hour = 12.5; // 12:30 UT
4
5 // Passo 1: Calendario a Giorno Giuliano
6 double jd_utc = calendar_to_jd(year, month, day + hour
    /24.0);
7 double mjd_utc = jd_utc - 2400000.5;
8
9 // Passo 2: UTC a TDB
10 double mjd_tdb = TimeScale::utc_to_tdb(mjd_utc);
11
12 std::cout << "MJD (TDB): " << std::fixed << std::
13     setprecision(6)
14     << mjd_tdb << std::endl;
15 // Output: MJD (TDB): 61000.520833

```

Listing 2.2: Conversione di data calendario a TDB

2.8 Letture Approfondite

Le specifiche dettagliate dei sistemi temporali sono mantenute da:

- **IERS** (International Earth Rotation Service): <https://www.iers.org>
- **BIPM** (Bureau International des Poids et Mesures): Definizione TAI
- **IAU** (International Astronomical Union): Risoluzioni sulle scale temporali
- **USNO** (US Naval Observatory): *Astronomical Almanac*

La libreria SOFA (Standards of Fundamental Astronomy) fornisce implementazioni di riferimento delle trasformazioni temporali e di coordinate: <http://www.iausofa.org>

Capitolo 3

Sistemi di Coordinate e Sistemi di Riferimento

3.1 Introduzione

La meccanica celeste richiede la specifica precisa di posizioni e velocità. Ciò necessita di *sistemi di coordinate* ben definiti (strutture matematiche per specificare posizioni) e *sistemi di riferimento* (realizzazioni fisiche legate a oggetti astronomici).

3.2 Concetti Fondamentali

3.2.1 Sistemi Inerziali vs. Rotanti

Definizione 3.1 (Sistema Inerziale). Un *sistema di riferimento inerziale* è uno in cui vale la prima legge di Newton: un corpo non soggetto a forze si muove in linea retta a velocità costante.

Sistemi veramente inerziali non esistono (l'universo si espande!), ma sistemi fissi rispetto a quasar distanti sono effettivamente inerziali per la dinamica del sistema solare.

Definizione 3.2 (Sistema Rotante). Un *sistema di riferimento rotante* rispetto allo spazio inerziale. Forze fittizie (centrifuga, Coriolis) appaiono nei sistemi rotanti.

3.3 Sistema di Coordinate Equatoriali

3.3.1 Definizione

Il sistema equatoriale usa l'equatore e l'asse di rotazione terrestre:

- **Piano fondamentale:** Equatore terrestre (esteso alla sfera celeste)
- **Direzione primaria:** Equinozio vernale (γ), dove il Sole attraversa l'equatore verso nord
- **Polo:** Polo celeste nord (direzione dell'asse di rotazione terrestre)

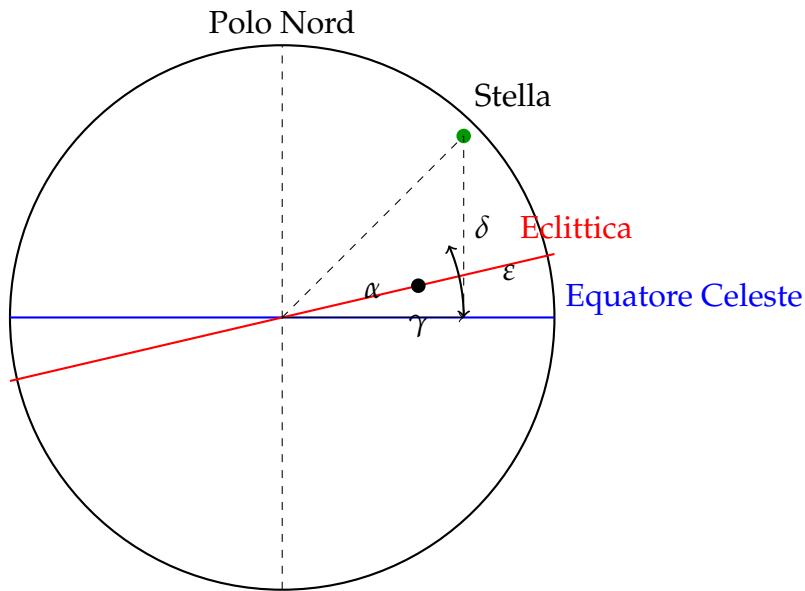


Figura 3.1: Sistema di coordinate equatoriali mostrando ascensione retta (α) e declinazione (δ). L'obliquità $\varepsilon \approx 23.4^\circ$.

3.3.2 Coordinate Sferiche

Le posizioni sono specificate da:

Ascensione Retta (α) Angolo verso est dall'equinozio vernale lungo l'equatore (da 0° a 360° , o da 0h a 24h)

Declinazione (δ) Angolo a nord (+) o a sud (-) dell'equatore (da -90° a $+90^\circ$)

Distanza (r) Distanza radiale dall'origine

Conversione in coordinate cartesiane:

$$x = r \cos \delta \cos \alpha \quad (3.1)$$

$$y = r \cos \delta \sin \alpha \quad (3.2)$$

$$z = r \sin \delta \quad (3.3)$$

3.4 Sistema di Coordinate Eclittiche

3.4.1 Definizione

Il sistema eclittico usa il piano orbitale terrestre:

- **Piano fondamentale:** Eclittica (piano orbitale terrestre)
- **Direzione primaria:** Equinozio vernale (stesso del sistema equatoriale)
- **Polo:** Normale al piano dell'eclittica

Le coordinate sono:

Longitudine Eclittica (λ) Angolo dall'equinozio vernale lungo l'eclittica

Latitudine Eclittica (β) Angolo a nord/sud dell'eclittica

3.4.2 Perché Usare Coordinate Eclittiche?

Per oggetti del sistema solare:

- Le orbite planetarie giacciono vicino all'eclittica ($|\beta| < 10^\circ$ tipicamente)
- Semplifica i calcoli di perturbazione
- Sistema naturale per la dinamica eliocentrica

3.5 Trasformazione tra Sistemi

3.5.1 Eclittico \leftrightarrow Equatoriale

La trasformazione implica una rotazione attorno all'asse x (direzione dell'equinozio vernale) di un angolo pari all'*obliquità* $\varepsilon \approx 23.43929^\circ$:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}_{\text{eq}} = \mathbf{R}_x(\varepsilon) \begin{bmatrix} x \\ y \\ z \end{bmatrix}_{\text{ecl}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \varepsilon & -\sin \varepsilon \\ 0 & \sin \varepsilon & \cos \varepsilon \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}_{\text{ecl}} \quad (3.4)$$

Per la trasformazione inversa (equatoriale → eclittico), usare $\mathbf{R}_x(-\varepsilon) = \mathbf{R}_x(\varepsilon)^T$.

3.5.2 Implementazione

```

1 #include <astdyn/coordinates/ReferenceFrame.hpp>
2 using namespace astdyn::coordinates;
3
4 // Eclittico a equatoriale J2000
5 Vector3d pos_ecl(1.0, 0.5, 0.1); // AU
6 Matrix3d rot = ReferenceFrame::ecliptic_to_j2000();
7 Vector3d pos_eq = rot * pos_ecl;
8
9 // Equatoriale a eclittico
10 Vector3d vel_eq(0.01, 0.02, 0.005); // AU/giorno
11 Matrix3d rot_inv = rot.transpose(); // Matrice ortogonale
12 Vector3d vel_ecl = rot_inv * vel_eq;

```

Listing 3.1: Trasformazioni di coordinate in AstDyn

3.6 Il Sistema di Riferimento J2000.0

3.6.1 Epoca vs. Equinozio

Due concetti temporali sono critici:

Epoca Il tempo per cui sono specificate le coordinate (influenza le posizioni a causa del moto)

Equinozio Il tempo che definisce l'orientamento degli assi coordinati (influenza le direzioni di riferimento)

Esempio: "Posizione all'epoca 2025.0 nell'equinozio J2000.0" significa la posizione dell'oggetto al 1 gennaio 2025, espressa in un sistema di coordinate i cui assi sono definiti dall'orientamento terrestre al 1 gennaio 2000.

3.6.2 Precessione

L'asse di rotazione terrestre precede (oscilla) con un periodo di circa 26.000 anni a causa delle forze mareali del Sole e della Luna. Ciò causa la deriva dell'equinozio vernali verso ovest lungo l'eclittica a circa $50''$ all'anno.

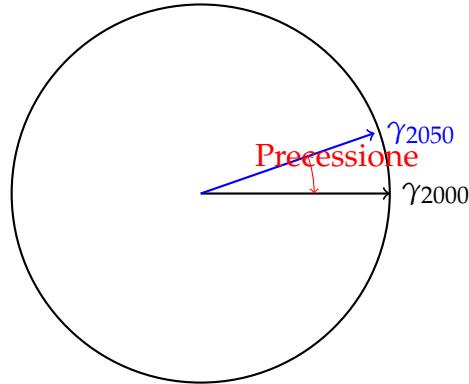


Figura 3.2: Precessione degli equinozi in 50 anni

Il sistema J2000.0 fissa l'equinozio al 1 gennaio 2000, 12:00 TT, fornendo un riferimento fisso per calcoli a lungo termine.

3.7 Considerazioni Pratiche

3.7.1 Scelta del Sistema di Riferimento

- **Eclittico eliocentrico:** Naturale per orbite di pianeti/asteroidi
- **Equatoriale geocentrico:** Standard per osservazioni da Terra
- **Baricentrico:** Richiesto per effemeridi planetarie precise

3.7.2 Trasformazioni di Sistema in AstDyn

La libreria fornisce matrici di rotazione per trasformazioni comuni:

```

1 // Eclittico <-> Equatoriale (J2000.0)
2 Matrix3d ecl_to_eq = ReferenceFrame::ecliptic_to_j2000();
3 Matrix3d eq_to_ecl = ecl_to_eq.transpose();
4
5 // ICRS <-> J2000 (piccola correzione di bias)
6 Matrix3d icrs_to_j2000 = ReferenceFrame::icrs_to_j2000();

```

Listing 3.2: Trasformazioni disponibili

Più trasformazioni (precessione, nutazione, GCRS) sono disponibili per applicazioni avanzate.

3.8 Riepilogo

Punti chiave:

- Sistema equatoriale: Legato alla rotazione terrestre (AR, Dec)
- Sistema eclittico: Legato all'orbita terrestre (naturale per dinamica eliocentrica)
- Trasformazioni: Semplici matrici di rotazione (ortogonali)
- J2000.0: Epoca/equinozio standard per l'astrometria moderna
- AstDyn: Implementa tutte le trasformazioni comuni in modo efficiente

Capitolo 4

Sistemi di Riferimento

4.1 Introduzione ai Sistemi di Riferimento

In meccanica celeste, un **sistema di riferimento** (o **frame di riferimento**) è un sistema di coordinate utilizzato per specificare le posizioni e le velocità dei corpi celesti. La scelta del sistema di riferimento è cruciale perché:

- Gli elementi orbitali sono definiti rispetto a un sistema specifico
- Le trasformazioni tra sistemi sono necessarie per le osservazioni
- Applicazioni diverse possono preferire sistemi diversi
- L'accuratezza numerica dipende dalla scelta del sistema

Un sistema di riferimento è costituito da:

1. Un'**origine** (es. centro della Terra, baricentro del Sistema Solare)
2. Un **piano fondamentale** (es. equatore, eclittica)
3. Una **direzione di riferimento** (es. punto vernale)
4. Un'**epoca** per l'orientamento (es. J2000.0)

4.2 Il Sistema di Riferimento Celeste Internazionale (ICRS)

L'**ICRS** è l'attuale sistema di riferimento celeste standard adottato dall'Unione Astronomica Internazionale (IAU) nel 1998. Rappresenta la realizzazione più precisa di un sistema di riferimento inerziale.

4.2.1 Definizione dell'ICRS

L'ICRS è definito da:

- **Origine:** Baricentro del Sistema Solare
- **Piano fondamentale:** Equatore medio terrestre a J2000.0 (con correzioni)
- **Direzione di riferimento:** Punto vernale medio a J2000.0 (con correzioni)
- **Realizzazione:** Posizioni di 212 sorgenti radio extragalattiche (quasar)

L'ICRS è un sistema cinematicamente non rotante con assi definiti con precisione del microarcosecondo utilizzando osservazioni VLBI (Very Long Baseline Interferometry) di quasar.

4.2.2 Relazione con J2000.0

L'ICRS è strettamente allineato con il sistema equatoriale J2000.0 ma differisce per:

- Bias del sistema: ~ 20 milliardcsec di orientamento
- Nessuna velocità di rotazione (veramente inerziale)
- Definizione basata su sorgenti extragalattiche (non sulla rotazione terrestre)

Per la maggior parte delle applicazioni nella dinamica asteroidale, la differenza tra ICRS e J2000.0 è trascurabile (< 0.1 arcsec su secoli).

4.3 Il Sistema Equatoriale J2000.0

Il sistema **J2000.0** è il sistema di riferimento più comunemente utilizzato in meccanica celeste ed è il sistema predefinito in AstDyn.

4.3.1 Definizione di J2000.0

- **Epoca:** 1 gennaio 2000, 12:00 TT (JD 2451545.0)
- **Origine:** Baricentro del Sistema Solare (o centro della Terra per geocentrico)
- **Piano fondamentale:** Equatore medio terrestre a J2000.0

- **Asse X:** Punta verso il punto vernali medio a J2000.0
- **Asse Z:** Perpendicolare all'equatore, verso il polo celeste nord
- **Asse Y:** Completa il sistema destrorso ($\mathbf{Y} = \mathbf{Z} \times \mathbf{X}$)

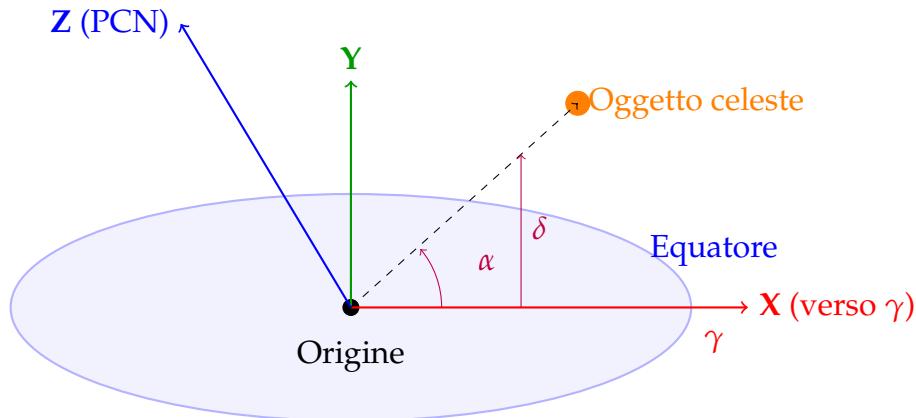


Figura 4.1: Sistema di riferimento equatoriale J2000.0 che mostra i tre assi e la definizione dell'ascensione retta (α) e della declinazione (δ).

4.3.2 Sistemi Eliocentrici vs. Baricentrici

Per le orbite asteroidali, utilizziamo tipicamente:

- **Sistema eliocentrico:** Origine al centro del Sole. Adatto per oggetti del sistema solare interno dove il Sole domina la dinamica gravitazionale.
- **Sistema baricentrico:** Origine al baricentro del Sistema Solare. Richiesto per calcoli precisi che coinvolgono Giove e i pianeti esterni, poiché il bari-centro Sole-Giove si trova al di fuori della superficie solare.

La trasformazione tra sistemi eliocentrici e baricentrici coinvolge la posizione del Sole rispetto al baricentro:

$$\mathbf{r}_{\text{bary}} = \mathbf{r}_{\text{helio}} + \mathbf{r}_{\text{Sole,bary}} \quad (4.1)$$

Per asteroidi con $a < 10$ UA, la differenza è tipicamente $< 10^{-6}$ UA.

4.4 Il Sistema di Riferimento Eclittico

Il **sistema eclittico** utilizza il piano dell'orbita terrestre come piano fondamentale.

4.4.1 Definizione dell'Eclittica

- **Piano fondamentale:** Eclittica media a J2000.0
- **Asse X:** Verso il punto vernali medio a J2000.0
- **Asse Z:** Perpendicolare all'eclittica, verso il polo eclittico nord
- **Asse Y:** Completa il sistema destrorso

Il sistema eclittico è naturale per descrivere le orbite planetarie e asteroidali perché:

- La maggior parte delle orbite giace vicino al piano eclittico
- Le inclinazioni sono tipicamente piccole ($i < 30^\circ$)
- I modelli di formazione del sistema solare prevedono l'allineamento eclittico

4.4.2 Coordinate Eclittiche

Nel sistema eclittico, le posizioni sono specificate da:

- **Longitudine eclittica (λ):** Angolo dal punto vernali lungo l'eclittica
- **Latitudine eclittica (β):** Angolo perpendicolare all'eclittica
- **Distanza (r):** Distanza radiale dall'origine

Conversione da coordinate cartesiane eclittiche:

$$\lambda = \arctan \left(\frac{Y_{\text{ecl}}}{X_{\text{ecl}}} \right) \quad (4.2)$$

$$\beta = \arctan \left(\frac{Z_{\text{ecl}}}{\sqrt{X_{\text{ecl}}^2 + Y_{\text{ecl}}^2}} \right) \quad (4.3)$$

$$r = \sqrt{X_{\text{ecl}}^2 + Y_{\text{ecl}}^2 + Z_{\text{ecl}}^2} \quad (4.4)$$

4.5 Trasformazioni tra Sistemi di Riferimento

4.5.1 Trasformazione Equatoriale-Eclittica

La trasformazione dalle coordinate equatoriali J2000.0 alle coordinate eclittiche J2000.0 è una rotazione attorno all'asse X dell'**obliquità dell'eclittica** (ε_0):

$$\begin{pmatrix} X_{\text{ecl}} \\ Y_{\text{ecl}} \\ Z_{\text{ecl}} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos \varepsilon_0 & \sin \varepsilon_0 \\ 0 & -\sin \varepsilon_0 & \cos \varepsilon_0 \end{pmatrix} \begin{pmatrix} X_{\text{eq}} \\ Y_{\text{eq}} \\ Z_{\text{eq}} \end{pmatrix} \quad (4.5)$$

A J2000.0, l'obliquità è:

$$\varepsilon_0 = 23^\circ 26' 21.406'' = 23.4392911^\circ \quad (4.6)$$

La trasformazione inversa è semplicemente una rotazione di $-\varepsilon_0$:

$$\mathbf{R}_{\text{ecl} \rightarrow \text{eq}} = \mathbf{R}_{\text{eq} \rightarrow \text{ecl}}^T \quad (4.7)$$

4.5.2 Precessione: Trasformazioni Dipendenti dal Tempo

L'asse di rotazione terrestre precede a causa delle coppie esercitate dalla Luna e dal Sole sul rigonfiamento equoriale terrestre. Questo causa un cambiamento dell'orientamento del piano equoriale nel tempo.

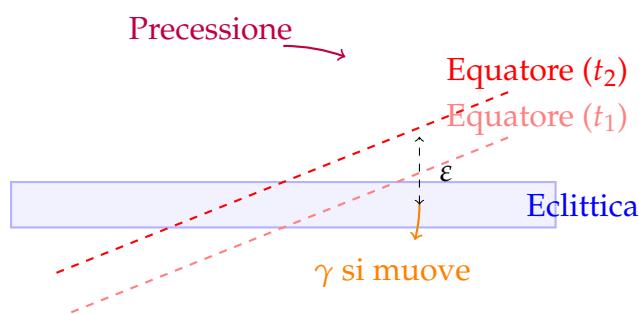


Figura 4.2: La precessione causa il cambiamento dell'orientamento del piano equoriale nel tempo. Il punto vernali γ si sposta verso ovest lungo l'eclittica a circa 50.3 arcosecondi all'anno.

Il tasso di precessione è approssimativamente:

$$\frac{d\alpha}{dt} \approx 50.3'' \text{ all'anno (in ascensione retta)} \quad (4.8)$$

Per trasformazioni tra epoche diverse (es. da J2000.0 alla data), devono essere applicate matrici di precessione. Il modello di precessione IAU 2006 è l'attuale standard.

4.5.3 Implementazione in AstDyn

In AstDyn, la classe `ReferenceFrame` gestisce le trasformazioni di coordinate:

```
1 #include <astdyn/core/ReferenceFrame.hpp>
2
3 using namespace astdyn;
4
5 // Trasformazione equatoriale -> eclittica
6 Vector3d r_eq(1.0, 0.5, 0.3); // UA, J2000.0 equatoriale
7 Vector3d r_ecl = ReferenceFrame::equatorial_to_ecliptic(
8     r_eq);
9
10 // Eclittica -> equatoriale
11 Vector3d r_eq2 = ReferenceFrame::ecliptic_to_equatorial(
12     r_ecl);
13
14 // Verifica round-trip: r_eq ~~ r_eq2
15 std::cout << "Errore round-trip: "
16             << (r_eq - r_eq2).norm() << " UA\n";
17 // Output: Errore round-trip: 1.23e-16 UA
18
19 // Precessione da J2000.0 alla data
20 double jd_now = 2460000.0; // Epoca corrente
21 Matrix3d precession_matrix =
22     ReferenceFrame::precession_matrix_j2000_to_date(jd_now)
23     ;
24
25 Vector3d r_now = precession_matrix * r_eq;
```

Listing 4.1: Trasformazioni di coordinate in AstDyn

4.6 Altri Importanti Sistemi di Riferimento

4.6.1 Il Sistema FK5

Il **Fifth Fundamental Catalogue (FK5)** era il sistema di riferimento standard prima dell'ICRS. È basato su osservazioni di stelle brillanti ed è equivalente a J2000.0 per la maggior parte degli scopi.

- **Epoca:** J2000.0
- **Realizzazione:** 1535 stelle fondamentali
- **Accuratezza:** ~ 10 milliarcsec
- **Relazione con ICRS:** Piccole differenze sistematiche

Per il lavoro con gli asteroidi, FK5 e ICRS sono intercambiabili entro le incertezze osservative.

4.6.2 Il Piano Invariabile

Il **piano invariabile** è perpendicolare al vettore momento angolare totale del sistema solare. È veramente inerziale (nessuna coppia esterna) e fornisce un riferimento dinamicamente naturale.

- **Inclinazione rispetto all'eclittica:** $\sim 1.58^\circ$
- **Dominato da:** Momento angolare orbitale di Giove ($\sim 60\%$ del totale)
- **Uso:** Studi dinamici, analisi di stabilità a lungo termine

Il piano invariabile non è utilizzato per le osservazioni ma è prezioso per gli studi teorici.

4.6.3 Sistemi Corpo-Centrici

Per la dinamica satellitare o gli avvicinamenti ravvicinati, si usano sistemi corpo-centrici:

- **Origine:** Centro di massa del corpo (es. Terra, Marte)
- **Orientamento:** Spesso allineato con l'asse di rotazione del corpo
- **Esempi:** Earth-centered inertial (ECI), sistemi planetocentrici

4.7 Considerazioni Pratiche

4.7.1 Precisione Numerica

Quando si lavora con sistemi di riferimento:

- Usare precisione doppia (64-bit) per coordinate in UA
- Errori di precessione accumulati: $\sim 10^{-10}$ UA per trasformazione
- Per $\Delta t > 100$ anni, includere correzioni di precessione
- Per $\Delta t > 1000$ anni, usare modelli completi di precessione/nutazione

4.7.2 Scelta del Sistema per la Propagazione Orbitale

Per la propagazione orbitale asteroidale in AstDyn:

- **Predefinito:** Eliocentrico J2000.0 equatoriale
- **Motivazioni:**
 - Corrisponde alla maggior parte dei cataloghi osservativi
 - Stabile su secoli
 - Semplifica il confronto con altri software
- **Alternativa:** Sistema eclittico per orbite a inclinazione molto bassa

4.7.3 Conversione delle Osservazioni

Le osservazioni ottiche sono tipicamente riportate in:

- **Ascensione retta (α) e declinazione (δ):** Sistema equatoriale
- **Coordinate topocentriche:** Dalla posizione dell'osservatore sulla Terra

Per utilizzarle nella determinazione orbitale:

1. Convertire da topocentrico a geocentrico (correggere per la rotazione terrestre)
2. Convertire da geocentrico a eliocentrico (aggiungere la posizione della Terra)

3. Esprimere nel sistema equatoriale J2000.0

La classe `OpticalObservation` di AstDyn gestisce automaticamente queste trasformazioni.

4.8 Riepilogo

Punti chiave sui sistemi di riferimento:

1. **ICRS** è lo standard moderno, strettamente allineato con J2000.0
2. **J2000.0 equatoriale** è il sistema pratico per la dinamica asteroidale
3. **Sistema eclittico** è naturale per oggetti del sistema solare
4. Le trasformazioni tra sistemi sono rotazioni definite dall'obliquità
5. La **precessione** causa rotazioni del sistema dipendenti dal tempo
6. AstDyn usa per default il sistema eliocentrico J2000.0 equatoriale

Comprendere i sistemi di riferimento è essenziale per:

- Interpretare gli elementi orbitali
- Elaborare le osservazioni
- Confrontare risultati tra diversi software
- Propagazione orbitale a lungo termine

Nel prossimo capitolo, discuteremo gli elementi orbitali—i sei parametri che specificano univocamente un'orbita in un dato sistema di riferimento.

Capitolo 5

Elementi Orbitali

5.1 Introduzione agli Elementi Orbitali

Un **elemento orbitale** è un parametro che descrive la forma, la dimensione, l'orientamento e la posizione di un'orbita nello spazio. Per il problema dei due corpi, sono necessari esattamente sei parametri per specificare univocamente un'orbita, corrispondenti ai sei gradi di libertà (tre per la posizione, tre per la velocità).

Esistono diversi insiemi di elementi orbitali, ognuno con vantaggi per applicazioni specifiche:

- **Elementi kepleriani:** Classici, geometricamente intuitivi
- **Elementi cartesiani:** Semplici per l'integrazione numerica
- **Elementi equinoziali:** Evitano singolarità a bassa inclinazione
- **Elementi di Delaunay:** Canonici, utili per la teoria delle perturbazioni

5.2 Elementi Kepleriani Classici

Gli **elementi kepleriani classici** sono l'insieme più ampiamente utilizzato. Descrivono direttamente la geometria di un'orbita a sezione conica.

5.2.1 I Sei Elementi Kepleriani

1. **Semiasse maggiore (a):** Metà del diametro più lungo dell'ellisse. Determina la dimensione orbitale e il periodo.

$$a = \frac{r_{\text{peri}} + r_{\text{apo}}}{2} \quad (5.1)$$

Unità: UA (unità astronomiche) per gli asteroidi, km per i satelliti.

2. **Eccentricità (e):** Forma dell'orbita.

$$e = \frac{r_{\text{apo}} - r_{\text{peri}}}{r_{\text{apo}} + r_{\text{peri}}} \quad (5.2)$$

- $e = 0$: Orbita circolare
- $0 < e < 1$: Orbita ellittica
- $e = 1$: Traiettoria parabolica
- $e > 1$: Traiettoria iperbolica

3. **Inclinazione (i):** Angolo tra il piano orbitale e il piano di riferimento (equatore o eclittica).

$$0^\circ \leq i \leq 180^\circ \quad (5.3)$$

- $i < 90^\circ$: Orbita prograda (diretta)
- $i = 90^\circ$: Orbita polare
- $i > 90^\circ$: Orbita retrograda

4. **Longitudine del nodo ascendente (Ω):** Angolo dalla direzione di riferimento al nodo ascendente (dove l'orbita attraversa il piano di riferimento andando verso nord).

$$0^\circ \leq \Omega < 360^\circ \quad (5.4)$$

5. **Argomento del perielio (ω):** Angolo dal nodo ascendente al perielio all'interno del piano orbitale.

$$0^\circ \leq \omega < 360^\circ \quad (5.5)$$

6. **Anomalia media (M):** Posizione del corpo lungo l'orbita in un dato istante, misurata come angolo dal perielio.

$$M = n(t - t_{\text{peri}}) \quad (5.6)$$

dove $n = \sqrt{\mu/a^3}$ è il moto medio.

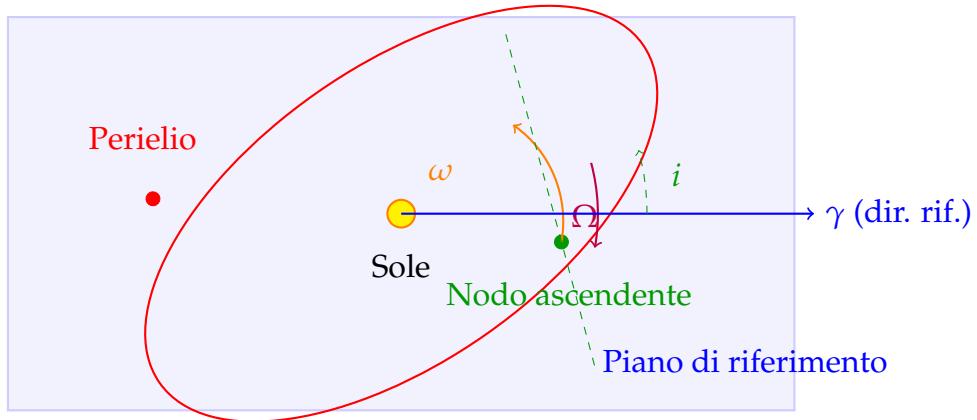


Figura 5.1: Elementi orbitali kepleriani classici. Il piano orbitale (ellisse rossa) è inclinato rispetto al piano di riferimento. Il nodo ascendente è dove l'orbita attraversa il piano di riferimento andando verso nord.

5.2.2 Periodo Orbitale

Per le orbite ellittiche, la Terza Legge di Keplero mette in relazione il periodo con il semiasse maggiore:

$$P = 2\pi \sqrt{\frac{a^3}{\mu}} \quad (5.7)$$

dove $\mu = GM$ è il parametro gravitazionale del corpo centrale.

Per il Sole:

$$P[\text{anni}] = a[\text{UA}]^{3/2} \quad (5.8)$$

Esempi:

- Terra: $a = 1 \text{ UA} \Rightarrow P = 1 \text{ anno}$
- Marte: $a = 1.524 \text{ UA} \Rightarrow P = 1.88 \text{ anni}$
- Ceres: $a = 2.77 \text{ UA} \Rightarrow P = 4.61 \text{ anni}$

5.2.3 Energia Orbitale

L'energia orbitale specifica (energia per unità di massa) è:

$$\mathcal{E} = -\frac{\mu}{2a} \quad (5.9)$$

Si noti che $\mathcal{E} < 0$ per orbite ellittiche (legate), $\mathcal{E} = 0$ per paraboliche e $\mathcal{E} > 0$ per iperboliche.

5.2.4 Singolarità degli Elementi Kepleriani

Gli elementi kepleriani hanno singolarità matematiche:

- Ω non definito per $i = 0^\circ$ (orbita nel piano di riferimento)
- ω non definito per $e = 0$ (orbita circolare, nessun perielio)
- Sia Ω che ω non definiti per $i = 0^\circ$ ed $e = 0$ simultaneamente

Per orbite quasi circolari o quasi equatoriali, gli errori numerici possono diventare grandi. Insiemi di elementi alternativi evitano questi problemi.

5.3 Vettore di Stato Cartesiano

Il **vettore di stato cartesiano** specifica posizione e velocità in un sistema di riferimento:

$$\mathbf{x} = \begin{pmatrix} \mathbf{r} \\ \mathbf{v} \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \end{pmatrix} \quad (5.10)$$

5.3.1 Vantaggi

- Nessuna singolarità
- Equazioni del moto semplici: $\ddot{\mathbf{r}} = -\mu \mathbf{r} / r^3 + \mathbf{a}_{\text{pert}}$
- Uso diretto negli integratori numerici
- Facile includere perturbazioni

5.3.2 Svantaggi

- Meno intuitivo degli elementi kepleriani

- Difficile interpretare direttamente la geometria orbitale
- Sei variabili strettamente accoppiate nell'integrazione

5.3.3 Conversione: Kepleriano a Cartesiano

Dati gli elementi kepleriani ($a, e, i, \Omega, \omega, M$) all'epoca t_0 :

Passo 1: Risolvere l'equazione di Keplero per l'anomalia eccentrica E :

$$M = E - e \sin E \quad (5.11)$$

(Richiede soluzione iterativa, es. Newton-Raphson)

Passo 2: Calcolare l'anomalia vera ν :

$$\nu = 2 \arctan \left(\sqrt{\frac{1+e}{1-e}} \tan \frac{E}{2} \right) \quad (5.12)$$

Passo 3: Posizione e velocità nel piano orbitale:

$$r = a(1 - e \cos E) \quad (5.13)$$

$$\mathbf{r}_{\text{orb}} = r \begin{pmatrix} \cos \nu \\ \sin \nu \\ 0 \end{pmatrix} \quad (5.14)$$

$$\mathbf{v}_{\text{orb}} = \sqrt{\frac{\mu}{a}} \begin{pmatrix} -\sin E \\ \sqrt{1-e^2} \cos E \\ 0 \end{pmatrix} \quad (5.15)$$

Passo 4: Ruotare nel sistema di riferimento usando la matrice di rotazione:

$$\mathbf{R}_3(-\omega) \mathbf{R}_1(-i) \mathbf{R}_3(-\Omega) \quad (5.16)$$

dove $\mathbf{R}_1(\theta)$ e $\mathbf{R}_3(\theta)$ sono rotazioni attorno agli assi 1 e 3.

5.3.4 Conversione: Cartesiano a Kepleriano

Data la posizione \mathbf{r} e la velocità \mathbf{v} :

Passo 1: Calcolare il momento angolare:

$$\mathbf{h} = \mathbf{r} \times \mathbf{v} \quad (5.17)$$

Passo 2: Calcolare il vettore del nodo:

$$\mathbf{n} = \hat{\mathbf{z}} \times \mathbf{h} \quad (5.18)$$

Passo 3: Calcolare il vettore di eccentricità:

$$\mathbf{e}_{\text{vec}} = \frac{1}{\mu} \left[(\mathbf{v} \times \mathbf{h}) - \mu \frac{\mathbf{r}}{r} \right] \quad (5.19)$$

Passo 4: Estrarre gli elementi:

$$a = \frac{1}{2/r - v^2/\mu} \quad (5.20)$$

$$e = |\mathbf{e}_{\text{vec}}| \quad (5.21)$$

$$i = \arccos \frac{h_z}{|\mathbf{h}|} \quad (5.22)$$

$$\Omega = \arctan \frac{n_y}{n_x} \quad (5.23)$$

$$\omega = \arccos \frac{\mathbf{n} \cdot \mathbf{e}_{\text{vec}}}{|\mathbf{n}| |\mathbf{e}_{\text{vec}}|} \quad (5.24)$$

$$\nu = \arccos \frac{\mathbf{e}_{\text{vec}} \cdot \mathbf{r}}{|\mathbf{e}_{\text{vec}}| |\mathbf{r}|} \quad (5.25)$$

Quindi $M = E - e \sin E$ dove $E = 2 \arctan \left(\sqrt{\frac{1-e}{1+e}} \tan \frac{\nu}{2} \right)$.

5.4 Elementi Equinoziali

Gli **elementi equinoziali** evitano singolarità a inclinazione ed eccentricità zero. Sono particolarmente utili per asteroidi con orbite quasi circolari o a bassa inclinazione.

5.4.1 Definizione

L'insieme equinoziale è:

$$a = \text{semiasse maggiore (uguale al kepleriano)} \quad (5.26)$$

$$h = e \sin(\omega + \Omega) \quad (5.27)$$

$$k = e \cos(\omega + \Omega) \quad (5.28)$$

$$p = \tan(i/2) \sin \Omega \quad (5.29)$$

$$q = \tan(i/2) \cos \Omega \quad (5.30)$$

$$\lambda = M + \omega + \Omega \quad (\text{longitudine media}) \quad (5.31)$$

5.4.2 Conversione a Kepleriano

Da equinoziale a kepleriano:

$$e = \sqrt{h^2 + k^2} \quad (5.32)$$

$$i = 2 \arctan \sqrt{p^2 + q^2} \quad (5.33)$$

$$\Omega = \arctan \frac{p}{q} \quad (5.34)$$

$$\omega = \arctan \frac{h}{k} - \Omega \quad (5.35)$$

$$M = \lambda - \omega - \Omega \quad (5.36)$$

5.4.3 Vantaggi

- Nessuna singolarità per $i \approx 0$ o $e \approx 0$
- Evoluzione regolare vicino a orbite circolari/equatoriali
- Utilizzato nel sistema HORIZONS di JPL
- Ben adatto per la propagazione orbitale numerica

5.5 Elementi di Delaunay

Gli **elementi di Delaunay** sono un insieme canonico di variabili azione-angolo utilizzato nella teoria delle perturbazioni e nella meccanica hamiltoniana.

5.5.1 Definizione

Le variabili di Delaunay sono:

$$L = \sqrt{\mu a} \quad (\text{azione coniugata a } \ell = M) \quad (5.37)$$

$$G = L\sqrt{1 - e^2} \quad (\text{azione coniugata a } g = \omega) \quad (5.38)$$

$$H = G \cos i \quad (\text{azione coniugata a } h = \Omega) \quad (5.39)$$

Gli angoli sono:

$$\ell = M \quad (\text{anomalia media}) \quad (5.40)$$

$$g = \omega \quad (\text{argomento del perielio}) \quad (5.41)$$

$$h = \Omega \quad (\text{longitudine del nodo ascendente}) \quad (5.42)$$

5.5.2 Proprietà

- (L, G, H, ℓ, g, h) formano un insieme di coordinate canoniche
- Formulazione hamiltoniana: $\dot{q}_i = \partial H / \partial p_i$, $\dot{p}_i = -\partial H / \partial q_i$
- Hamiltoniana imperturbata: $H_0 = -\mu^2 / (2L^2)$ (dipende solo da L)
- Per il problema di Keplero imperturbato: L, G, H sono costanti
- Utile per la teoria delle perturbazioni secolari e l'analisi delle risonanze

5.6 Implementazione in AstDyn

AstDyn fornisce funzioni di conversione nella classe `OrbitalElements`:

```
1 #include <astdyn/core/OrbitalElements.hpp>
2 #include <astdyn/core/StateVector.hpp>
3
4 using namespace astdyn;
5
6 // Elementi kepleriani
7 OrbitalElements kep;
8 kep.a = 2.77;           // UA
9 kep.e = 0.078;
10 kep.i = 10.6 * DEG_TO_RAD; // radianti
```

```

11  kep.Omega = 80.3 * DEG_TO_RAD;
12  kep.omega = 73.1 * DEG_TO_RAD;
13  kep.M = 15.2 * DEG_TO_RAD;
14  kep.epoch = 2460000.0; // JD
15
16 // Converti a cartesiano
17 StateVector sv = kep.to_state_vector();
18 std::cout << "Posizione: " << sv.r.transpose() << " UA\n";
19 std::cout << "Velocita: " << sv.v.transpose() << " UA/
20   giorno\n";
21
22 // Converti di nuovo a kepleriano
23 OrbitalElements kep2 = OrbitalElements::from_state_vector(
24   sv.r, sv.v, sv.t
25 );
26
27 // Verifica round-trip
28 std::cout << "Delta a: " << kep2.a - kep.a << " UA\n";
29 // Output: Delta a: 3.14e-15 UA (precisione macchina)
30
31 // Converti a equinoziale
32 auto eq = kep.to_equinocial();
33 std::cout << "h = " << eq.h << ", k = " << eq.k << "\n";
34 std::cout << "p = " << eq.p << ", q = " << eq.q << "\n";

```

Listing 5.1: Conversioni di elementi orbitali in AstDyn

5.7 Riepilogo

Punti chiave sugli elementi orbitali:

1. Sei parametri specificano un'orbita del problema dei due corpi (6 gradi di libertà)
2. Gli **elementi kepleriani** ($a, e, i, \Omega, \omega, M$) sono geometricamente intuitivi
3. Lo **stato cartesiano** (\mathbf{r}, \mathbf{v}) è semplice per il lavoro numerico
4. Gli **elementi equinoziali** evitano singolarità a $e = 0$ e $i = 0$

5. Gli **elementi di Delaunay** sono canonici, utili per la teoria delle perturbazioni
6. La scelta degli elementi dipende dall'applicazione e dalle caratteristiche dell'orbita
7. Le conversioni tra insiemi di elementi sono operazioni standard in AstDyn

Comprendere gli elementi orbitali è essenziale per:

- Leggere e interpretare i cataloghi osservativi
- Impostare problemi di propagazione orbitale
- Analizzare la dinamica e le perturbazioni orbitali
- Scegliere metodi numerici appropriati

Nel prossimo capitolo, studieremo in dettaglio il problema dei due corpi—il fondamento della meccanica orbitale e la base per tutte le analisi delle perturbazioni.

Capitolo 6

Il Problema dei Due Corpi

6.1 Introduzione al Problema dei Due Corpi

Il **problema dei due corpi** è il fondamento della meccanica orbitale. Descrive il moto di due masse puntiformi che interagiscono solo attraverso l'attrazione gravitazionale reciproca. Questo è l'unico caso in meccanica celeste con una soluzione analitica completa.

6.1.1 Formulazione del Problema

Consideriamo due corpi con masse m_1 e m_2 separati da una distanza r . La legge di gravitazione di Newton fornisce:

$$\mathbf{F}_{12} = -G \frac{m_1 m_2}{r^2} \hat{\mathbf{r}} \quad (6.1)$$

dove $G = 6.674 \times 10^{-11} \text{ m}^3 \text{ kg}^{-1} \text{ s}^{-2}$ è la costante gravitazionale.

Le equazioni del moto sono:

$$m_1 \ddot{\mathbf{r}}_1 = G \frac{m_1 m_2}{r^3} (\mathbf{r}_2 - \mathbf{r}_1) \quad (6.2)$$

$$m_2 \ddot{\mathbf{r}}_2 = G \frac{m_1 m_2}{r^3} (\mathbf{r}_1 - \mathbf{r}_2) \quad (6.3)$$

6.1.2 Riduzione a Problema a Un Corpo

Introducendo la posizione relativa $\mathbf{r} = \mathbf{r}_2 - \mathbf{r}_1$ e la massa ridotta $\mu = G(m_1 + m_2)$, il problema si riduce a:

$$\ddot{\mathbf{r}} = -\frac{\mu}{r^3} \mathbf{r} \quad (6.4)$$

Questa è l'**equazione della forza centrale** con parametro gravitazionale μ . Per il sistema Sole-pianeta, $\mu_{\odot} = 1.327 \times 10^{20} \text{ m}^3 \text{ s}^{-2}$.

6.2 Leggi di Conservazione

Il problema dei due corpi ha diverse quantità conservate che vincolano il moto.

6.2.1 Conservazione del Momento Angolare

Il momento angolare è:

$$\mathbf{h} = \mathbf{r} \times \mathbf{v} \quad (6.5)$$

Poiché $\ddot{\mathbf{r}}$ è parallelo a \mathbf{r} , abbiamo:

$$\frac{d\mathbf{h}}{dt} = \mathbf{r} \times \ddot{\mathbf{r}} = \mathbf{0} \quad (6.6)$$

Quindi: $\mathbf{h} = \text{costante}$

Conseguenze:

- Il moto è confinato in un piano perpendicolare a \mathbf{h}
- $|\mathbf{h}| = h = \sqrt{\mu a(1 - e^2)}$ si relaziona agli elementi orbitali
- La velocità areolare è costante: $\frac{dA}{dt} = \frac{h}{2}$ (Seconda legge di Keplero)

6.2.2 Conservazione dell'Energia

L'energia meccanica specifica è:

$$\mathcal{E} = \frac{v^2}{2} - \frac{\mu}{r} = \text{costante} \quad (6.7)$$

Questa può essere scritta come:

$$\mathcal{E} = -\frac{\mu}{2a} \quad (6.8)$$

per orbite ellittiche con semiasse maggiore a .

6.2.3 Il Vettore di Laplace-Runge-Lenz

Il vettore di eccentricità è conservato:

$$\mathbf{e} = \frac{1}{\mu}(\mathbf{v} \times \mathbf{h}) - \frac{\mathbf{r}}{r} \quad (6.9)$$

Proprietà:

- $|\mathbf{e}| = e$ (eccentricità orbitale)
- \mathbf{e} punta verso il periastro
- $\mathbf{e} \cdot \mathbf{h} = 0$ (perpendicolare al momento angolare)

6.3 L'Equazione dell'Orbita

6.3.1 Derivazione

In coordinate polari (r, ν) dove ν è l'anomalia vera, l'equazione dell'orbita è:

$$r = \frac{a(1 - e^2)}{1 + e \cos \nu} = \frac{p}{1 + e \cos \nu} \quad (6.10)$$

dove $p = a(1 - e^2)$ è il semi-latus rectum.

Questa è l'equazione di una sezione conica con fuoco nell'origine.

6.3.2 Sezioni Coniche

La forma dell'orbita dipende dall'eccentricità e dall'energia:

Tipi di Orbita	Eccentricità	Energia	Esempi
Cerchio	$e = 0$	$\mathcal{E} < 0$	Orbite idealizzate
Ellisse	$0 < e < 1$	$\mathcal{E} < 0$	Pianeti, asteroidi
Parabola	$e = 1$	$\mathcal{E} = 0$	Traiettoria di fuga
Iperbole	$e > 1$	$\mathcal{E} > 0$	Oggetti interstellari

Tabella 6.1: Classificazione delle coniche orbitali per eccentricità ed energia.

6.4 Le Leggi di Keplero

Johannes Kepler (1571-1630) derivò tre leggi empiriche dalle osservazioni del moto planetario. Queste sono conseguenze del problema dei due corpi.

6.4.1 Prima Legge di Keplero (Legge delle Ellissi)

L'orbita di un pianeta è un'ellisse con il Sole in uno dei fuochi.

Matematicamente:

$$r = \frac{a(1 - e^2)}{1 + e \cos \nu} \quad (6.11)$$

La distanza al perielio è $r_p = a(1 - e)$ e la distanza all'afelio è $r_a = a(1 + e)$.

6.4.2 Seconda Legge di Keplero (Legge delle Aree)

Una linea che unisce un pianeta e il Sole spazza aree uguali in tempi uguali.

Questo segue dalla conservazione del momento angolare:

$$\frac{dA}{dt} = \frac{h}{2} = \frac{1}{2} \sqrt{\mu a(1 - e^2)} = \text{costante} \quad (6.12)$$

6.4.3 Terza Legge di Keplero (Legge Armonica)

Il quadrato del periodo orbitale è proporzionale al cubo del semiasse maggiore.

$$P^2 = \frac{4\pi^2}{\mu} a^3 \quad (6.13)$$

Per il sistema solare:

$$P[\text{anni}] = a[\text{AU}]^{3/2} \quad (6.14)$$

6.5 L'Equazione di Keplero

6.5.1 Le Anomalie

Tre angoli correlati descrivono la posizione in un'orbita ellittica:

Anomalia Vera (ν) Angolo effettivo dal perielio alla posizione corrente

Anomalia Eccentrica (E) Angolo ausiliario sul cerchio circoscritto

Anomalia Media (M) Angolo che sarebbe percorso se il moto fosse uniforme

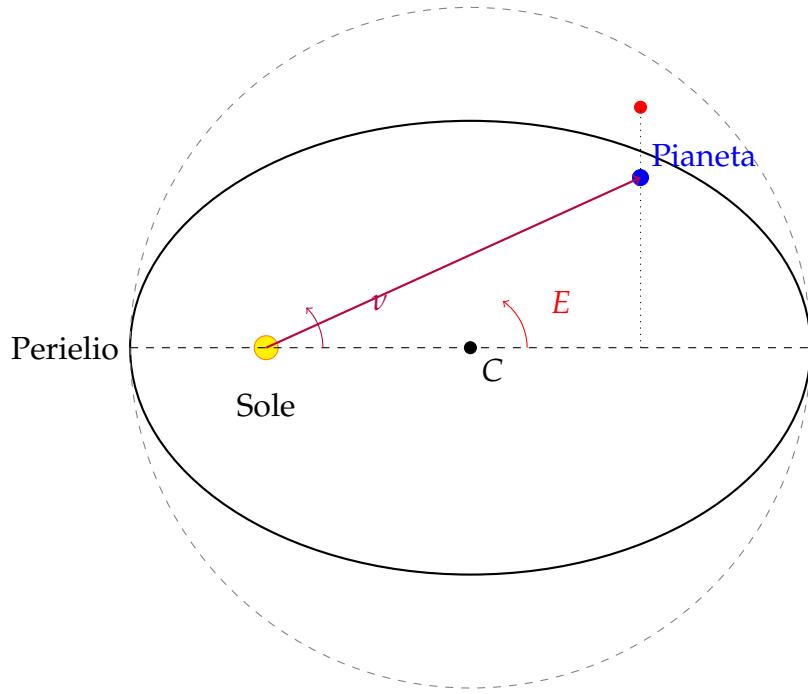


Figura 6.2: Relazione tra anomalia vera ν e anomalia eccentrica E in un'orbita ellittica.

6.5.2 L'Equazione di Keplero

L'anomalia media avanza uniformemente con il tempo:

$$M = n(t - t_p) = \sqrt{\frac{\mu}{a^3}}(t - t_p) \quad (6.15)$$

dove t_p è il tempo del passaggio al perielio e n è il moto medio.

L'equazione di Keplero relaziona M a E :

$$M = E - e \sin E \quad (6.16)$$

Questa è un'equazione trascendente senza soluzione in forma chiusa. Deve essere risolta iterativamente.

6.5.3 Risoluzione dell'Equazione di Keplero

Metodo di Newton-Raphson:

Dato M ed e , si trova E iterativamente:

$$f(E) = E - e \sin E - M = 0 \quad (6.17)$$

$$f'(E) = 1 - e \cos E \quad (6.18)$$

$$E_{n+1} = E_n - \frac{f(E_n)}{f'(E_n)} = E_n - \frac{E_n - e \sin E_n - M}{1 - e \cos E_n} \quad (6.19)$$

Stima iniziale: $E_0 = M$ (per e piccolo) oppure $E_0 = M + e$ (migliore per e moderato).

La convergenza si ottiene tipicamente in 3-5 iterazioni per $\epsilon < 10^{-12}$.

6.5.4 Relazione tra le Anomalie

Una volta noto E , l'anomalia vera è:

$$\nu = 2 \arctan \left(\sqrt{\frac{1+e}{1-e}} \tan \frac{E}{2} \right) \quad (6.20)$$

Oppure equivalentemente:

$$\cos \nu = \frac{\cos E - e}{1 - e \cos E} \quad (6.21)$$

$$\sin \nu = \frac{\sqrt{1-e^2} \sin E}{1 - e \cos E} \quad (6.22)$$

La distanza radiale è:

$$r = a(1 - e \cos E) \quad (6.23)$$

6.6 L'Equazione Vis-Viva

L'**equazione vis-viva** (“forza viva”) relaziona la velocità alla posizione:

$$v^2 = \mu \left(\frac{2}{r} - \frac{1}{a} \right) \quad (6.24)$$

Questa è derivata dalla conservazione dell’energia ed è valida per tutte le orbite coniche.

6.6.1 Casi Speciali

Al perielio ($r = a(1 - e)$):

$$v_p = \sqrt{\frac{\mu}{a} \frac{1+e}{1-e}} \quad (6.25)$$

All'afelio ($r = a(1 + e)$):

$$v_a = \sqrt{\frac{\mu}{a} \frac{1-e}{1+e}} \quad (6.26)$$

Orbita circolare ($e = 0, r = a$):

$$v_c = \sqrt{\frac{\mu}{a}} \quad (6.27)$$

Velocità di fuga (parabolica, $e = 1, a \rightarrow \infty$):

$$v_e = \sqrt{\frac{2\mu}{r}} \quad (6.28)$$

6.7 Orbite Paraboliche e Iperboliche

6.7.1 Orbite Paraboliche ($e = 1$)

Per traiettorie di fuga, l'equazione dell'orbita diventa:

$$r = \frac{p}{1 + \cos \nu} \quad (6.29)$$

dove p è la distanza al periastro.

Il tempo di volo è dato dall'equazione di Barker:

$$t - t_p = \frac{1}{2} \sqrt{\frac{p^3}{\mu}} \left(\tan \frac{\nu}{2} + \frac{1}{3} \tan^3 \frac{\nu}{2} \right) \quad (6.30)$$

6.7.2 Orbite Iperboliche ($e > 1$)

Per traiettorie interstellari o di flyby:

$$r = \frac{a(e^2 - 1)}{1 + e \cos \nu} \quad (6.31)$$

Nota: $a < 0$ per orbite iperboliche (energia negativa).

L'anomalia iperbolica F soddisfa:

$$M_h = e \sinh F - F \quad (6.32)$$

E l'anomalia vera è:

$$\nu = 2 \arctan \left(\sqrt{\frac{e+1}{e-1}} \tanh \frac{F}{2} \right) \quad (6.33)$$

La velocità asintotica all'infinito è:

$$v_\infty = \sqrt{-\frac{\mu}{a}} = \sqrt{\mu \frac{e^2 - 1}{a}} \quad (6.34)$$

6.8 Coefficienti di Lagrange

I **coefficienti di Lagrange** (o funzioni f e g) forniscono un modo per propagare le orbite senza calcolare esplicitamente gli elementi orbitali.

6.8.1 Definizione

Dato lo stato iniziale $(\mathbf{r}_0, \mathbf{v}_0)$ al tempo t_0 , lo stato al tempo t è:

$$\mathbf{r}(t) = f(t)\mathbf{r}_0 + g(t)\mathbf{v}_0 \quad (6.35)$$

$$\mathbf{v}(t) = \dot{f}(t)\mathbf{r}_0 + \dot{g}(t)\mathbf{v}_0 \quad (6.36)$$

dove f, g, \dot{f}, \dot{g} sono funzioni scalari del tempo.

6.8.2 Espressioni per i Coefficienti di Lagrange

Per orbite ellittiche:

$$f = 1 - \frac{a}{r_0} (1 - \cos \Delta E) \quad (6.37)$$

$$g = t - t_0 + \sqrt{\frac{a^3}{\mu}} (\sin \Delta E - \Delta E) \quad (6.38)$$

$$\dot{f} = -\sqrt{\frac{\mu a}{r r_0}} \sin \Delta E \quad (6.39)$$

$$\dot{g} = 1 - \frac{a}{r} (1 - \cos \Delta E) \quad (6.40)$$

dove $\Delta E = E - E_0$ è la variazione dell'anomalia eccentrica.

6.8.3 Proprietà

I coefficienti di Lagrange soddisfano:

$$f\dot{g} - \dot{f}g = 1 \quad (6.41)$$

Questa è l'**identità di Lagrange**, che assicura la conservazione del Wronskiano.

6.9 Implementazione in AstDyn

AstDyn fornisce funzioni per risolvere il problema dei due corpi:

```
1 #include <astdyn/dynamics/TwoBody.hpp>
2 #include <astdyn/core/OrbitalElements.hpp>
3
4 using namespace astdyn;
5
6 // Risolve l'equazione di Keplero
7 double M = 45.0 * DEG_TO_RAD;    // Anomalia media
8 double e = 0.3;                  // Eccentricità
9 double E = TwoBody::solve_kepler_equation(M, e);
10 std::cout << "Anomalia eccentrica: " << E * RAD_TO_DEG << "
11     deg\n";
```

```

12 // Converte E in anomalia vera
13 double nu = TwoBody::eccentric_to_true_anomaly(E, e);
14 std::cout << "Anomalia vera: " << nu * RAD_TO_DEG << " deg\n";
15
16 // Calcola posizione e velocita da elementi orbitali
17 OrbitalElements kep;
18 kep.a = 2.5; // AU
19 kep.e = 0.15;
20 kep.M = M;
21 // ... imposta altri elementi
22
23 auto [r, v] = kep.to_position_velocity();
24
25 // Propaga usando i coefficienti di Lagrange
26 double dt = 100.0; // giorni
27 auto [f, g, fdot, gdot] = TwoBody::lagrange_coefficients(
28     r, v, dt, MU_SUN
29 );
30
31 Vector3d r_new = f * r + g * v;
32 Vector3d v_new = fdot * r + gdot * v;
33
34 std::cout << "Nuova posizione: " << r_new.transpose() << "
35             AU\n";
36 std::cout << "Nuova velocita: " << v_new.transpose() << "
37             AU/day\n";

```

Listing 6.1: Problema dei due corpi in AstDyn

6.10 Sommario

Punti chiave sul problema dei due corpi:

1. Il problema dei due corpi ha una **soluzione analitica esatta**
2. Il moto è governato dalla conservazione dell'energia, del momento angolare e del vettore di eccentricità

3. Le orbite sono **sezioni coniche**: cerchi, ellissi, parabole o iperboli
4. Le **leggi di Keplero** sono conseguenze dirette della gravità newtoniana
5. L'**equazione di Keplero** ($M = E - e \sin E$) relaziona le anomalie media ed eccentrica
6. L'**equazione vis-viva** relaziona la velocità alla posizione
7. I **coefficienti di Lagrange** permettono una propagazione efficiente dell'orbita

Comprendere il problema dei due corpi è essenziale per:

- Prevedere le posizioni di pianeti e asteroidi
- Progettare traiettorie spaziali
- Comprendere la linea di base da cui le perturbazioni deviano
- Sviluppare propagatori numerici efficienti

Nel prossimo capitolo, studieremo le perturbazioni—deviazioni dal moto ideale dei due corpi causate da forze aggiuntive.

Capitolo 7

Perturbazioni Orbitali

7.1 Introduzione alle Perturbazioni

Nel Capitolo 6, abbiamo studiato il problema idealizzato dei due corpi in cui si considera solo l'attrazione gravitazionale tra due masse puntiformi. In realtà, i corpi celesti sperimentano forze aggiuntive che causano deviazioni delle loro orbite dalle ellissi kepleriane perfette.

7.1.1 Tipi di Perturbazioni

Le perturbazioni orbitali possono essere classificate per origine fisica:

Gravitazionali Forze da corpi aggiuntivi (problema degli N corpi), distribuzione di massa non sferica (J_2, J_4 , ecc.)

Non gravitazionali Pressione di radiazione solare, resistenza atmosferica, effetti termici (Yarkovsky)

Relativistiche Correzioni relativistiche generali alla gravità newtoniana

7.1.2 Equazioni del Moto Perturbate

L'equazione generale del moto con perturbazioni è:

$$\ddot{\mathbf{r}} = -\frac{\mu}{r^3} \mathbf{r} + \mathbf{a}_{\text{pert}} \quad (7.1)$$

dove \mathbf{a}_{pert} è l'accelerazione perturbatrice. Per piccole perturbazioni, possiamo trattarle come correzioni alla soluzione kepleriana.

7.1.3 Magnitudine degli Effetti

Per un asteroide della fascia principale a 2.5 AU:

Perturbazione	Accelerazione	Relativa al Sole
Gravità solare	$3.8 \times 10^{-3} \text{ m/s}^2$	1
Giove	$\sim 10^{-6} \text{ m/s}^2$	3×10^{-4}
J ₂ terrestre (in LEO)	$\sim 10^{-6} \text{ m/s}^2$	—
Radiazione solare	$\sim 10^{-8} \text{ m/s}^2$	3×10^{-6}
Relatività	$\sim 10^{-10} \text{ m/s}^2$	3×10^{-8}

Tabella 7.1: Magnitudini tipiche delle accelerazioni perturbative per asteroidi.

Sebbene piccoli, questi effetti si accumulano nel tempo e devono essere inclusi per previsioni accurate a lungo termine.

7.2 Il Problema degli N Corpi

7.2.1 Formulazione del Problema

Il **problema degli N corpi** considera il moto di N corpi sotto la loro attrazione gravitazionale reciproca. L'equazione del moto per il corpo i è:

$$\ddot{\mathbf{r}}_i = \sum_{j=1, j \neq i}^N G \frac{m_j (\mathbf{r}_j - \mathbf{r}_i)}{|\mathbf{r}_j - \mathbf{r}_i|^3} \quad (7.2)$$

Per $N \geq 3$, non esiste una soluzione analitica generale. Il problema deve essere risolto numericamente.

7.2.2 Il Problema Ristretto dei Tre Corpi

Un caso speciale è il **problema circolare ristretto dei tre corpi** (CR3BP):

- Due corpi massicci (primari) orbitano attorno al loro baricentro comune in orbite circolari
- Un terzo corpo (senza massa) si muove sotto la loro influenza gravitazionale
- Il terzo corpo non influenza i primari

Questo è rilevante per sistemi Sole-Giove-astroide o Terra-Luna-spacecraft.

7.2.3 Perturbazioni dai Pianeti

Per la determinazione orbitale degli asteroidi, modelliamo tipicamente il Sole come corpo centrale e i pianeti come corpi perturbatori:

$$\mathbf{a}_{\text{pianeti}} = \sum_p \left[Gm_p \left(\frac{\mathbf{r}_p - \mathbf{r}}{|\mathbf{r}_p - \mathbf{r}|^3} - \frac{\mathbf{r}_p}{r_p^3} \right) \right] \quad (7.3)$$

Il primo termine è l'attrazione gravitazionale diretta dal pianeta p , e il secondo termine tiene conto del fatto che anche il Sole accelera verso il pianeta (termine indiretto).

7.2.4 Effemeridi Planetarie

La modellazione accurata degli N corpi richiede posizioni planetarie ad alta precisione. Fonti comuni:

- **JPL DE440/DE441:** Ultime effemeridi planetarie della NASA (2021)
- **INPOP:** Effemeridi planetarie francesi dell'IMCCE
- **SPICE:** Toolkit NASA per geometria di spacecraft e pianeti

AstDyn può utilizzare kernel SPICE per ottenere stati planetari a qualsiasi epoca.

7.3 Perturbazioni da Schiacciamento (J_2)

7.3.1 Distribuzione di Massa Non Sferica

I corpi celesti reali non sono sfere perfette. La Terra, ad esempio, è schiacciata a causa della rotazione. Il potenziale gravitazionale può essere espanso in armoniche sferiche:

$$U = -\frac{\mu}{r} \left[1 - \sum_{n=2}^{\infty} J_n \left(\frac{R}{r} \right)^n P_n(\sin \phi) \right] \quad (7.4)$$

dove:

- J_n sono i coefficienti armonici zonali
- R è il raggio di riferimento

- P_n sono i polinomi di Legendre
- ϕ è la latitudine

7.3.2 Il Termine J_2

Il termine dominante è J_2 (momento di quadrupolo), che rappresenta il rigonfiamento equatoriale:

$$U_{J_2} = -\frac{\mu}{r} \left[1 - J_2 \left(\frac{R}{r} \right)^2 P_2(\sin \phi) \right] \quad (7.5)$$

dove $P_2(\sin \phi) = \frac{1}{2}(3 \sin^2 \phi - 1)$.

Per la Terra: $J_2 = 1.08263 \times 10^{-3}$ (circa 0.1%)

7.3.3 Accelerazione da J_2

L'accelerazione perturbatrice in coordinate cartesiane è:

$$a_x = -\frac{3\mu J_2 R^2}{2r^5} \left[x \left(1 - 5 \frac{z^2}{r^2} \right) \right] \quad (7.6)$$

$$a_y = -\frac{3\mu J_2 R^2}{2r^5} \left[y \left(1 - 5 \frac{z^2}{r^2} \right) \right] \quad (7.7)$$

$$a_z = -\frac{3\mu J_2 R^2}{2r^5} \left[z \left(3 - 5 \frac{z^2}{r^2} \right) \right] \quad (7.8)$$

7.3.4 Effetti sugli Elementi Orbitali

J_2 causa variazioni secolari (a lungo termine) negli elementi orbitali:

$$\frac{d\Omega}{dt} = -\frac{3}{2} \frac{n J_2 R^2}{a^2 (1 - e^2)^2} \cos i \quad (7.9)$$

$$\frac{d\omega}{dt} = \frac{3}{4} \frac{n J_2 R^2}{a^2 (1 - e^2)^2} (5 \cos^2 i - 1) \quad (7.10)$$

dove $n = \sqrt{\mu/a^3}$ è il moto medio.

Effetti chiave:

- Ω (RAAN) precede verso ovest per orbite prograde ($i < 90^\circ$)

- ω (argomento del periastro) ruota
- La combinazione crea pattern complessi nelle tracce al suolo

Per orbite terrestri basse (LEO), J_2 può causare variazioni di Ω di diversi gradi al giorno.

7.4 Pressione di Radiazione Solare

7.4.1 Meccanismo Fisico

I fotoni trasportano momento. Quando la luce solare colpisce un oggetto, esercita una forza:

$$F_{\text{SRP}} = P_{\odot} \frac{A}{c} C_R \left(\frac{r_0}{r} \right)^2 \quad (7.11)$$

dove:

- $P_{\odot} = 4.56 \times 10^{-6} \text{ N/m}^2$ è la pressione di radiazione solare a 1 AU
- A è l'area della sezione trasversale
- $c = 3 \times 10^8 \text{ m/s}$ è la velocità della luce
- C_R è il coefficiente di pressione di radiazione ($C_R \approx 1-2$)
- $r_0 = 1 \text{ AU}$, r è la distanza eliocentrica

7.4.2 Rapporto Area-Massa

L'accelerazione dipende dal **rapporto area-massa**:

$$\mathbf{a}_{\text{SRP}} = P_{\odot} \frac{A}{m} C_R \left(\frac{r_0}{r} \right)^2 \hat{\mathbf{r}}_{\odot} \quad (7.12)$$

Gli oggetti piccoli (polvere, piccoli asteroidi) sono più influenzati di quelli grandi.

7.4.3 Modellazione dell'Eclisse

La SRP scende a zero quando l'oggetto è nell'ombra della Terra o di un pianeta. Un modello semplice:

$$\nu = \begin{cases} 1 & \text{alla luce del sole} \\ 0 & \text{in ombra} \\ f & \text{in penombra} \end{cases} \quad (7.13)$$

dove $0 < f < 1$ dipende dalla frazione del disco solare visibile.

7.4.4 Effetto Yarkovsky

L'**effetto Yarkovsky** è una forza di rinculo termico:

- La superficie dell'asteroide si riscalda alla luce del sole
- Emette radiazione termica mentre ruota
- Crea una piccola spinta (come un razzo!)

Questo è importante per piccoli asteroidi (< 20 km) su scale temporali lunghe (milioni di anni). Può cambiare il semiasse maggiore:

$$\frac{da}{dt} \approx \pm 10^{-4} \text{ AU/Myr} \quad (7.14)$$

Il segno dipende dal senso di rotazione (progrado vs retrogrado).

7.5 Effetti Relativistici

7.5.1 Correzioni Post-Newtoniane

La relatività generale introduce correzioni alla gravità newtoniana. Il termine dominante è il **termine di Schwarzschild**:

$$\mathbf{a}_{\text{GR}} = \frac{\mu}{c^2 r^3} \left[4 \frac{\mu}{r} \mathbf{r} - (\mathbf{v} \cdot \mathbf{v}) \mathbf{r} + 4(\mathbf{r} \cdot \mathbf{v}) \mathbf{v} \right] \quad (7.15)$$

Questa è l'approssimazione post-newtoniana del primo ordine (1PN).

7.5.2 Precessione del Perielio

L'effetto relativistico più famoso è la **precessione del periolio**:

$$\Delta\omega = \frac{6\pi GM_{\odot}}{c^2 a (1 - e^2)} \text{ per orbita} \quad (7.16)$$

Per Mercurio ($a = 0.387$ AU, $e = 0.206$):

$$\Delta\omega_{\text{Mercurio}} = 43'' \text{ per secolo} \quad (7.17)$$

Questo fu famosamente spiegato da Einstein nel 1915 e fu una delle prime conferme della relatività generale.

7.5.3 Correzione del Tempo Luce

I segnali elettromagnetici viaggiano a velocità finita. Quando si misurano le posizioni degli asteroidi tramite radar o osservazioni ottiche, dobbiamo tenere conto del tempo che la luce impiega per viaggiare:

$$\Delta t = \frac{|\mathbf{r}_{\text{oss}} - \mathbf{r}_{\text{ast}}|}{c} \quad (7.18)$$

Questa è la **correzione del tempo luce**. Per la determinazione orbitale, dobbiamo iterare per trovare la posizione dell'asteroide al tempo dell'osservazione, non al tempo della rilevazione.

7.5.4 Ritardo di Shapiro

I campi gravitazionali rallentano la luce. Il **ritardo di Shapiro** è:

$$\Delta t_{\text{Shapiro}} = \frac{2GM_{\odot}}{c^3} \ln \left(\frac{r_1 + r_2 + d}{r_1 + r_2 - d} \right) \quad (7.19)$$

dove r_1 , r_2 sono le distanze dal Sole ai due estremi, e d è la loro separazione. Questo è tipicamente ~ 100 microsecondi ma è misurabile con ranging di precisione.

7.6 Resistenza Atmosferica

Per i satelliti in orbita terrestre bassa (LEO), la resistenza atmosferica è una perturbazione importante.

7.6.1 Equazione della Resistenza

$$\mathbf{a}_{\text{drag}} = -\frac{1}{2} \frac{C_D A}{m} \rho v^2 \hat{\mathbf{v}} \quad (7.20)$$

dove:

- $C_D \approx 2.2$ è il coefficiente di resistenza
- A è l'area della sezione trasversale
- ρ è la densità atmosferica (decresce esponenzialmente con l'altitudine)
- v è la velocità relativa all'atmosfera

7.6.2 Modelli di Densità Atmosferica

La densità dipende da:

- Altitudine (decadimento esponenziale)
- Attività solare (indice F10.7)
- Attività geomagnetica (indice Ap)
- Ora solare locale e latitudine

Modelli comuni: NRLMSISE-00, JB2008, DTM2000.

7.6.3 Decadimento Orbitale

La resistenza causa la diminuzione del semiasse maggiore:

$$\frac{da}{dt} = -\frac{2a^2}{v} \frac{C_D A}{m} \rho v^2 = -\frac{C_D A}{m} \rho a^2 v \quad (7.21)$$

I satelliti in LEO gradualmente spiraleggiano verso l'interno e alla fine rientrano nell'atmosfera.

7.7 Teoria delle Perturbazioni

7.7.1 Variazione dei Parametri

Le **equazioni planetarie di Lagrange** descrivono come cambiano gli elementi orbitali sotto perturbazioni. In termini della funzione perturbatrice R :

$$\frac{da}{dt} = \frac{2}{na} \frac{\partial R}{\partial M} \quad (7.22)$$

$$\frac{de}{dt} = \frac{1-e^2}{na^2e} \frac{\partial R}{\partial M} - \frac{\sqrt{1-e^2}}{na^2e} \frac{\partial R}{\partial \omega} \quad (7.23)$$

$$\frac{di}{dt} = \frac{\cos i}{na^2\sqrt{1-e^2}\sin i} \frac{\partial R}{\partial \omega} - \frac{1}{na^2\sqrt{1-e^2}\sin i} \frac{\partial R}{\partial \Omega} \quad (7.24)$$

$$\frac{d\Omega}{dt} = \frac{1}{na^2\sqrt{1-e^2}\sin i} \frac{\partial R}{\partial i} \quad (7.25)$$

$$\frac{d\omega}{dt} = \frac{\sqrt{1-e^2}}{na^2e} \frac{\partial R}{\partial e} - \frac{\cos i}{na^2\sqrt{1-e^2}\sin i} \frac{\partial R}{\partial i} \quad (7.26)$$

$$\frac{dM}{dt} = n - \frac{2}{na} \frac{\partial R}{\partial a} - \frac{1-e^2}{na^2e} \frac{\partial R}{\partial e} \quad (7.27)$$

Queste equazioni permettono un trattamento analitico delle perturbazioni quando R ha una forma semplice.

7.7.2 Equazioni di Perturbazione di Gauss

Una formulazione alternativa usa le componenti dell'accelerazione perturbatrice (S, T, W) nelle direzioni radiale, trasversale e normale:

$$\frac{da}{dt} = \frac{2a^2}{h} \left[eS \sin \nu + T \frac{p}{r} \right] \quad (7.28)$$

$$\frac{de}{dt} = \frac{1}{v_0} \left[S \sin \nu + T \left(\cos \nu + \frac{r+p}{p} \cos E \right) \right] \quad (7.29)$$

$$\frac{di}{dt} = \frac{r \cos(\omega + \nu)}{h} W \quad (7.30)$$

dove $h = \sqrt{\mu a(1-e^2)}$ è la magnitudine del momento angolare.

7.7.3 Elementi Osculatori

In qualsiasi istante, l'orbita può essere descritta da **elementi osculatori**—gli elementi kepleriani che il corpo seguirebbe se tutte le perturbazioni cessassero improvvisamente. Questi elementi variano continuamente sotto perturbazioni.

7.8 Integrazione Numerica vs Teoria delle Perturbazioni

7.8.1 Quando Usare Ogni Approccio

Metodo	Vantaggi	Migliore Per
Integrazione Numerica	Gestisce qualsiasi forza Nessuna approssimazione Facile da implementare	Accuratezza a breve termine Perturbazioni forti Forze multiple
Teoria Analitica delle Perturbazioni	Intuizione fisica Calcolo veloce Identifica risonanze	Tendenze a lungo termine Perturbazioni deboli Analisi qualitativa

Tabella 7.2: Confronto tra integrazione numerica e teoria analitica delle perturbazioni.

7.8.2 Approcci Ibridi

La determinazione orbitale moderna spesso usa:

1. Integrazione numerica per l'equazione del moto
2. Teoria analitica per identificare perturbazioni importanti
3. Modelli semplificati (es., J_2 mediato) per calcolo più veloce

7.9 Implementazione in AstDyn

AstDyn fornisce un framework modulare per le perturbazioni:

```
1 #include <astdyn/dynamics/Perturbations.hpp>
2 #include <astdyn/dynamics/NBody.hpp>
3
4 using namespace astdyn;
5
6 // Crea vettore di stato
7 Vector6d state = ...; // [x, y, z, vx, vy, vz]
8
9 // Perturbazioni N-body dai pianeti
```

```

10 PlanetaryEphemeris ephem("de440.bsp");
11 Vector3d acc_planets = NBody::compute_perturbation(
12     state, time, ephem, {"Jupiter", "Saturn", "Earth"})
13 );
14
15 // Perturbazione J2 (per orbitatore terrestre)
16 Vector3d acc_j2 = Perturbations::j2_acceleration(
17     state, MU_EARTH, R_EARTH, J2_EARTH
18 );
19
20 // Pressione di radiazione solare
21 double area_mass_ratio = 0.01; // m^2/kg
22 double Cr = 1.3;
23 Vector3d sun_direction = ...; // Vettore unitario verso
24     il Sole
25 Vector3d acc_srp = Perturbations::solar_radiation_pressure(
26     state, area_mass_ratio, Cr, sun_direction
27 );
28
29 // Correzione relativistica
30 Vector3d acc_gr = Perturbations::schwarzschild_correction(
31     state, MU_SUN
32 );
33
34 // Accelerazione perturbatrice totale
35 Vector3d acc_total = acc_planets + acc_j2 + acc_srp +
36     acc_gr;
37
38 // Aggiunge alle equazioni del moto
39 Vector6d derivatives;
40 derivatives.head<3>() = state.tail<3>(); // velocita
41 derivatives.tail<3>() = -MU_SUN * state.head<3>() / r^3 +
42     acc_total;

```

Listing 7.1: Perturbazioni in AstDyn

7.9.1 Selezione delle Perturbazioni

AstDyn permette agli utenti di abilitare/disabilitare le perturbazioni:

```
1 PerturbationModel model;
2 model.enable_planets({"Jupiter", "Saturn", "Uranus", "
    Neptune"});
3 model.enable_j2(false); // Non rilevante per orbite
    eliocentriche
4 model.enable_srp(true);
5 model.enable_relativity(true);
6
7 // Usa nella propagazione
8 Propagator prop(model);
9 auto final_state = prop.propagate(initial_state, t0, tf);
```

Listing 7.2: Configurazione delle perturbazioni

7.10 Sommario

Concetti chiave sulle perturbazioni orbitali:

1. Le **perturbazioni** sono deviazioni dal moto ideale dei due corpi
2. Gli **effetti N-body** dai pianeti sono la perturbazione dominante per gli asteroidi
3. Lo **schiacciamento J₂** causa precessione di Ω e ω (critico per satelliti terrestri)
4. La **pressione di radiazione solare** influenza piccoli corpi e spacecraft
5. Gli **effetti relativistici** sono piccoli ma misurabili (precessione di Mercurio: 43"/secolo)
6. La **resistenza atmosferica** domina in LEO, causando decadimento orbitale
7. La **teoria delle perturbazioni** (equazioni di Lagrange, Gauss) fornisce intuizione analitica
8. L'**integrazione numerica** gestisce modelli di forze arbitrarie accuratamente

Comprendere le perturbazioni è essenziale per:

- Previsione accurata dell'orbita su scale temporali lunghe
- Progettazione di missioni satellitari e station-keeping
- Rilevazione di effetti sottili (es., masse di asteroidi dalle perturbazioni)
- Distinzione tra forze gravitazionali e non gravitazionali

Nel prossimo capitolo, discuteremo i metodi di integrazione numerica per risolvere le equazioni del moto perturbate.

Parte II

Metodi Numerici e Algoritmi

Capitolo 8

Metodi di Integrazione Numerica

8.1 Introduzione

Nel Capitolo 7, abbiamo visto che il moto orbitale con perturbazioni richiede la risoluzione di:

$$\ddot{\mathbf{r}} = -\frac{\mu}{r^3} \mathbf{r} + \mathbf{a}_{\text{pert}}(t, \mathbf{r}, \dot{\mathbf{r}}) \quad (8.1)$$

Per perturbazioni generali, questa equazione differenziale non ha soluzione in forma chiusa. Dobbiamo usare l'**integrazione numerica** per calcolare l'orbita passo dopo passo.

Questo capitolo esamina le principali classi di integratori usati in meccanica celeste e discute i loro punti di forza, debolezze e implementazione in AstDyn.

8.1.1 Il Problema ai Valori Iniziali

Cerchiamo di risolvere:

$$\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y}), \quad \mathbf{y}(t_0) = \mathbf{y}_0 \quad (8.2)$$

dove $\mathbf{y} = [\mathbf{r}, \mathbf{v}]^T$ è il vettore di stato a 6 dimensioni.

L'obiettivo è avanzare da (t_0, \mathbf{y}_0) a (t_f, \mathbf{y}_f) con errore controllato.

8.2 Metodo di Eulero

L'integratore più semplice è il **metodo di Eulero**:

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h \mathbf{f}(t_n, \mathbf{y}_n) \quad (8.3)$$

dove $h = t_{n+1} - t_n$ è la dimensione del passo.

Pro: Semplice, esplicito **Contro:** Accuratezza del primo ordine (errore $O(h^2)$ per passo), instabile per problemi stiff

Il metodo di Eulero è raramente usato in pratica tranne che per scopi pedagogici.

8.3 Metodi Runge-Kutta

8.3.1 Il Metodo RK4

Il classico metodo **Runge-Kutta del quarto ordine** (RK4) è:

$$k_1 = h\mathbf{f}(t_n, \mathbf{y}_n) \quad (8.4)$$

$$k_2 = h\mathbf{f}(t_n + h/2, \mathbf{y}_n + k_1/2) \quad (8.5)$$

$$k_3 = h\mathbf{f}(t_n + h/2, \mathbf{y}_n + k_2/2) \quad (8.6)$$

$$k_4 = h\mathbf{f}(t_n + h, \mathbf{y}_n + k_3) \quad (8.7)$$

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \quad (8.8)$$

Pro: Accuratezza del quarto ordine ($O(h^5)$ per passo), auto-avviante, facile da implementare **Contro:** Richiede 4 valutazioni di funzione per passo, nessuna stima dell'errore

RK4 è ampiamente usato per problemi di accuratezza moderata.

8.3.2 Metodi Runge-Kutta Incorporati

Per il controllo adattivo della dimensione del passo, usiamo metodi **incorporati** che forniscono due soluzioni di ordini diversi:

Runge-Kutta-Fehlberg 4(5) (RKF45):

- Calcola soluzioni di 4° e 5° ordine
- Stima dell'errore: $\epsilon = |\mathbf{y}_5 - \mathbf{y}_4|$
- 6 valutazioni di funzione per passo

Dormand-Prince 5(4) (DOPRI54 o RK54):

- Coefficienti ottimizzati per migliore stabilità

- 7 valutazioni di funzione (una riusata per il passo successivo)
- Default in ode45 di MATLAB

Runge-Kutta 7(8) (RK78):

- Soluzioni di 7° e 8° ordine
- 13 valutazioni di funzione
- Migliore per requisiti di alta accuratezza

8.3.3 Controllo della Dimensione del Passo

Data la stima dell'errore ϵ , si aggiusta la dimensione del passo h :

$$h_{\text{new}} = h_{\text{old}} \left(\frac{\text{tol}}{\epsilon} \right)^{1/(q+1)} \times \text{fattore di sicurezza} \quad (8.9)$$

dove q è l'ordine e il fattore di sicurezza ≈ 0.9 .

Se $\epsilon > \text{tol}$: rifiuta il passo, riduci h Se $\epsilon < \text{tol}$: accetta il passo, eventualmente aumenta h

8.4 Metodi Multipasso

8.4.1 Adams-Bashforth-Moulton (ABM)

I metodi multipasso usano informazioni dai passi precedenti. La **famiglia Adams** è popolare:

Adams-Bashforth (predittore esplicito):

$$\mathbf{y}_{n+1}^P = \mathbf{y}_n + h \sum_{i=0}^{k-1} \beta_i \mathbf{f}_{n-i} \quad (8.10)$$

Adams-Moulton (correttore隐式):

$$\mathbf{y}_{n+1}^C = \mathbf{y}_n + h \sum_{i=-1}^{k-1} \beta_i^* \mathbf{f}_{n-i} \quad (8.11)$$

La modalità **predittore-correttore** (PC) valuta:

1. Predice \mathbf{y}_{n+1}^P usando Adams-Bashforth

2. Valuta $\mathbf{f}_{n+1} = \mathbf{f}(t_{n+1}, \mathbf{y}_{n+1}^P)$
3. Corregge \mathbf{y}_{n+1}^C usando Adams-Moulton

ABM12: Adams-Bashforth-Moulton del 12° ordine

- Usa 12 passi precedenti
- Accuratezza molto alta per problemi regolari
- Usato dal JPL per le effemeridi planetarie

Pro: Alto ordine con poche valutazioni di funzione (2 per passo dopo l'avvio)

Contro: Non auto-avviante, richiede passo fisso (o algoritmo a passo variabile attento)

8.4.2 Formule di Differenziazione all'Indietro (BDF)

Per problemi **stiff** (non comuni in meccanica orbitale), si preferiscono i metodi BDF:

$$\sum_{i=0}^k \alpha_i \mathbf{y}_{n+1-i} = h \mathbf{f}(t_{n+1}, \mathbf{y}_{n+1}) \quad (8.12)$$

Questi sono impliciti e richiedono la risoluzione di equazioni non lineari ad ogni passo.

8.5 Integratori Simplettici

8.5.1 Meccanica Hamiltoniana

Per sistemi conservativi, le equazioni del moto possono essere scritte in forma hamiltoniana:

$$\dot{\mathbf{q}} = \frac{\partial H}{\partial \mathbf{p}} \quad (8.13)$$

$$\dot{\mathbf{p}} = -\frac{\partial H}{\partial \mathbf{q}} \quad (8.14)$$

dove \mathbf{q} sono le posizioni, \mathbf{p} sono i momenti, e H è l'hamiltoniana (energia totale).

8.5.2 Proprietà Simplettica

Un metodo è **simplettico** se preserva la struttura simplettica dello spazio delle fasi. Questo assicura:

- L'energia oscilla attorno al valore vero (nessuna deriva sistematica)
- Stabilità a lungo termine
- Preservazione di strutture geometriche (es., orbite periodiche)

8.5.3 Metodo Leapfrog

L'integratore simplettico più semplice è il **leapfrog** (Verlet):

$$\mathbf{v}_{n+1/2} = \mathbf{v}_n + \frac{h}{2} \mathbf{a}_n \quad (8.15)$$

$$\mathbf{r}_{n+1} = \mathbf{r}_n + h \mathbf{v}_{n+1/2} \quad (8.16)$$

$$\mathbf{v}_{n+1} = \mathbf{v}_{n+1/2} + \frac{h}{2} \mathbf{a}_{n+1} \quad (8.17)$$

Pro: Simplettico, secondo ordine, semplice **Contro:** Richiede la separazione dell'hamiltoniana, non adatto per forze dipendenti dalla velocità

8.5.4 Metodi Simplettici di Ordine Superiore

Metodo di Yoshida (simplettico del 4° ordine):

- Composizione di passi leapfrog con coefficienti scelti attentamente
- Usato in simulazioni N-body

Metodo Wisdom-Holman:

- Separa l'hamiltoniana in parti kepleriana + perturbazione
- La parte kepleriana è risolta analiticamente
- Le perturbazioni sono gestite con impulsi

I metodi simplettici sono ideali per integrazioni a lungo termine (10^6 - 10^9 anni) dove la conservazione dell'energia è critica.

8.6 Analisi dell'Errore

8.6.1 Errore Locale vs Globale

Errore di troncamento locale (LTE): Errore introdotto in un singolo passo

Errore globale: Errore accumulato dopo molti passi

Per un metodo di ordine p :

- LTE $\propto h^{p+1}$
- Errore globale $\propto h^p$ (su intervallo fisso)

8.6.2 Compromesso Accuratezza vs Efficienza

Metodo	Ordine	Eval/passo	Migliore Per
Eulero	1	1	Solo didattica
RK4	4	4	Accuratezza moderata
RKF45	4(5)	6	Uso generale
DOPRI54	5(4)	7	Alta accuratezza
RK78	7(8)	13	Altissima accuratezza
ABM12	12	2	Regolare, alta accuratezza
Leapfrog	2	2	Lungo termine, conservativo

Tabella 8.1: Confronto dei metodi di integrazione numerica.

8.6.3 Fonti di Errore

Nella determinazione orbitale, gli errori provengono da:

1. **Errore di troncamento:** Dimensione del passo finita
2. **Errore di arrotondamento:** Aritmetica a precisione finita
3. **Errore nel modello di forze:** Perturbazioni incomplete o inaccurate
4. **Errore nelle effemeridi:** Incertezze nelle posizioni planetarie

Per lavoro ad alta precisione, tutte le fonti devono essere controllate.

8.7 Considerazioni Pratiche

8.7.1 Scelta di un Integratore

Per determinazione orbitale (giorni ad anni):

- DOPRI54 con dimensione del passo adattiva
- Tolleranza: 10^{-12} a 10^{-14}

Per evoluzione a lungo termine (milioni di anni):

- Wisdom-Holman o Yoshida simplettico
- Dimensione del passo fissa (0.1-1 giorno)

Per applicazioni in tempo reale:

- RK4 con dimensione del passo fissa
- Precalcola la dimensione del passo per stabilità

8.7.2 Selezione della Dimensione del Passo

Regola empirica: $h \approx 0.01 \times T_{\text{orbita}}$

Per asteroide a 2.5 AU:

- Periodo $T \approx 4$ anni = 1461 giorni
- Buona dimensione del passo: $h \approx 10\text{-}15$ giorni

I metodi adattivi regolano automaticamente h in base al comportamento locale.

8.7.3 Dimensione del Passo Iniziale

Per metodi adattivi, stima della dimensione del passo iniziale:

$$h_0 = 0.01 \times \min \left(\frac{|\mathbf{r}|}{|\dot{\mathbf{r}}|}, \frac{|\dot{\mathbf{r}}|}{|\ddot{\mathbf{r}}|} \right) \quad (8.18)$$

Questo prevede di fare un primo passo troppo grande.

8.8 Implementazione in AstDyn

AstDyn fornisce multipli integratori:

```
1 #include <astdyn/integration/Integrator.hpp>
2 #include <astdyn/integration/RK4.hpp>
3 #include <astdyn/integration/DOPRI54.hpp>
4
5 using namespace astdyn;
6
7 // Definisce il sistema ODE
8 auto ode = [] (double t, const Vector6d& y) -> Vector6d {
9     Vector3d r = y.head<3>();
10    Vector3d v = y.tail<3>();
11    Vector3d a = -MU_SUN * r / pow(r.norm(), 3);
12
13    Vector6d dydt;
14    dydt << v, a;
15    return dydt;
16};
17
18 // Stato iniziale
19 Vector6d y0;
20 y0 << 1.0, 0.0, 0.0, // posizione (AU)
21      0.0, 6.28, 0.0; // velocita (AU/day)
22
23 double t0 = 0.0;
24 double tf = 365.25; // 1 anno
25
26 // Opzione 1: RK4 a passo fisso
27 RK4Integrator<Vector6d> rk4;
28 double h = 1.0; // passi di 1 giorno
29 auto result_rk4 = rk4.integrate(ode, t0, y0, tf, h);
30
31 // Opzione 2: DOPRI54 adattivo
32 DOPRI54Integrator<Vector6d> dopri;
33 dopri.set_tolerance(1e-12);
34 auto result_dopri = dopri.integrate(ode, t0, y0, tf);
```

```

36 std::cout << "Posizione finale (RK4):    "
37             << result_rk4.transpose() << "\n";
38 std::cout << "Posizione finale (DOPRI):   "
39             << result_dopri.transpose() << "\n";

```

Listing 8.1: Uso degli integratori in AstDyn

8.8.1 Integratori Personalizzati

Gli utenti possono implementare integratori personalizzati ereditando da `IntegratorBase`:

```

1 template<typename StateType>
2 class CustomIntegrator : public IntegratorBase<StateType> {
3 public:
4     StateType integrate(
5         const ODEFunctor<StateType>& f,
6         double t0,
7         const StateType& y0,
8         double tf
9     ) override {
10        // Implementazione qui
11    }
12};

```

Listing 8.2: Interfaccia integratore personalizzato

8.9 Sommario

Concetti chiave sull'integrazione numerica:

1. I **metodi Runge-Kutta** sono versatili e auto-avvianti
2. La **dimensione del passo adattiva** (RKF45, DOPRI54) fornisce controllo automatico dell'errore
3. I **metodi multipasso** (ABM) sono efficienti per problemi regolari
4. Gli **integratori simplettici** preservano l'energia per simulazioni a lungo termine
5. Esistono **compromessi** tra accuratezza, efficienza e stabilità

6. La **dimensione del passo** deve essere scelta in base al periodo orbitale e ai requisiti di accuratezza

Comprendere l'integrazione numerica è essenziale per:

- Propagazione accurata dell'orbita
- Bilanciare costo computazionale e precisione
- Evitare artefatti numerici
- Validare i risultati contro soluzioni analitiche

Nel prossimo capitolo, applicheremo questi metodi di integrazione a problemi pratici di propagazione orbitale.

Capitolo 9

Propagazione delle Orbite

9.1 Introduzione

La **propagazione orbitale** è il processo di calcolo della posizione e velocità di un corpo celeste in epoche future (o passate), dati il suo stato iniziale e le forze agenti su di esso. Questo è fondamentale per:

- Prevedere dove puntare i telescopi per osservazioni di asteroidi
- Pianificare manovre di veicoli spaziali
- Calcolare effemeridi per almanacchi
- Analizzare l'evoluzione orbitale a lungo termine
- Valutare rischi di collisione

Basandosi sui metodi di integrazione del Capitolo 8, questo capitolo descrive la propagazione orbitale pratica in AstDyn.

9.2 Formulazione del Problema

9.2.1 Il Compito di Propagazione

Dati:

- Epoca iniziale t_0 (in una scala temporale, tipicamente TDB)
- Stato iniziale $\mathbf{y}_0 = [\mathbf{r}_0, \mathbf{v}_0]$ (posizione e velocità)

- Modello di forza $\mathbf{f}(t, \mathbf{r}, \mathbf{v})$ (accelerazioni)
- Epoca finale t_f

Calcolare:

- Stato finale $\mathbf{y}_f = [\mathbf{r}_f, \mathbf{v}_f]$
- Opzionalmente: matrice di transizione di stato $\Phi(t_f, t_0)$

9.2.2 Vettore di Stato

Per orbite eliocentriche, il vettore di stato è:

$$\mathbf{y} = \begin{bmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} \quad (9.1)$$

Unità in AstDyn:

- Posizione: AU (unità astronomiche)
- Velocità: AU/giorno
- Tempo: giorni (MJD o JD)

9.2.3 Equazioni del Moto

La forma generale è:

$$\frac{d}{dt} \begin{bmatrix} \mathbf{r} \\ \mathbf{v} \end{bmatrix} = \begin{bmatrix} \mathbf{v} \\ \mathbf{a}(t, \mathbf{r}, \mathbf{v}) \end{bmatrix} \quad (9.2)$$

dove l'accelerazione include:

$$\mathbf{a} = \mathbf{a}_{\text{centrale}} + \mathbf{a}_{\text{pianeti}} + \mathbf{a}_{\text{relatività}} + \mathbf{a}_{\text{SRP}} + \dots \quad (9.3)$$

9.3 Modelli di Forza

9.3.1 Gravità del Corpo Centrale

Il termine dominante per le orbite del sistema solare:

$$\mathbf{a}_{\text{Sole}} = -\frac{\mu_{\odot}}{r^3} \mathbf{r} \quad (9.4)$$

dove $\mu_{\odot} = 1.32712440018 \times 10^{20} \text{ m}^3/\text{s}^2 = 0.295912208286 \text{ AU}^3/\text{giorno}^2$.

9.3.2 Perturbazioni Planetarie

Per ogni pianeta perturbatore p :

$$\mathbf{a}_p = \mu_p \left[\frac{\mathbf{r}_p - \mathbf{r}}{|\mathbf{r}_p - \mathbf{r}|^3} - \frac{\mathbf{r}_p}{r_p^3} \right] \quad (9.5)$$

Il primo termine è l'attrazione diretta, il secondo è l'effetto indiretto (accelerazione del Sole verso il pianeta).

Le posizioni planetarie $\mathbf{r}_p(t)$ sono ottenute da:

- Kernel SPICE (JPL DE440/441)
- Teoria analitica VSOP87
- Effemeridi kepleriane semplificate (precisione inferiore)

9.3.3 Correzione Relativistica

Termine post-Newtoniano (1PN):

$$\mathbf{a}_{\text{GR}} = \frac{\mu_{\odot}}{c^2 r^3} \left[4 \frac{\mu_{\odot}}{r} \mathbf{r} - v^2 \mathbf{r} + 4(\mathbf{r} \cdot \mathbf{v}) \mathbf{v} \right] \quad (9.6)$$

Questo è tipicamente $\sim 10^{-10} \text{ m/s}^2$ per gli asteroidi, ma si accumula su scale temporali lunghe.

9.3.4 Pressione di Radiazione Solare

Per piccoli corpi o veicoli spaziali:

$$\mathbf{a}_{\text{SRP}} = P_{\odot} \frac{A}{m} C_R \left(\frac{r_0}{r} \right)^2 \hat{\mathbf{r}}_{\odot} \quad (9.7)$$

dove:

- $P_{\odot} = 4.56 \times 10^{-6} \text{ N/m}^2$ a 1 AU
- A/m è il rapporto area-massa (m^2/kg)
- $C_R \approx 1.3$ è il coefficiente di pressione di radiazione

9.3.5 Perturbazioni Asteroidali

Per lavori di precisione, gli asteroidi massicci (Cerere, Vesta, Pallade) possono perturbare le orbite di particelle test:

$$\mathbf{a}_{\text{ast}} = \sum_i \mu_i \left[\frac{\mathbf{r}_i - \mathbf{r}}{|\mathbf{r}_i - \mathbf{r}|^3} - \frac{\mathbf{r}_i}{r_i^3} \right] \quad (9.8)$$

Masse dei più grandi asteroidi:

- Cerere: $9.384 \times 10^{20} \text{ kg}$ (~ 0.0001 masse terrestri)
- Vesta: $2.59 \times 10^{20} \text{ kg}$
- Pallade: $2.04 \times 10^{20} \text{ kg}$

9.4 Sistemi di Coordinate

9.4.1 Sistemi di Riferimento

AstDyn supporta multipli sistemi di riferimento:

Eclittico Eliocentrico J2000 Standard per orbite asteroidali (predefinito)

Equatoriale Eliocentrico J2000 Comune per lavori planetari

Baricentrico Baricentro del sistema solare (per alta precisione)

Topocentrico Centrato sull'osservatore (per osservazioni)

9.4.2 Trasformazioni di Sistema

La rotazione eclittica-equatoriale è:

$$\mathbf{R}_{\text{ecl} \rightarrow \text{eq}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \epsilon & -\sin \epsilon \\ 0 & \sin \epsilon & \cos \epsilon \end{bmatrix} \quad (9.9)$$

dove $\epsilon = 23.43929111^\circ$ è l'obliquità a J2000.0.

9.5 Strategia di Integrazione

9.5.1 Scelta del Passo

Per integratori adattativi (DOPRI54), stima del passo iniziale:

$$h_0 = 0.01 \times \min \left(\frac{r}{v}, \frac{v}{a} \right) \quad (9.10)$$

Passi tipici:

- Asteroidi vicini alla Terra: 0.1-1 giorno
- Asteroidi della fascia principale: 5-20 giorni
- Troiani di Giove: 10-30 giorni
- Comete (vicino al perielio): 0.01-0.1 giorno

9.5.2 Selezione della Tolleranza

Tolleranza di posizione per determinazione orbitale:

- Orbite preliminari: 10^{-9} AU (~ 150 m)
- Orbite finali: 10^{-12} AU (~ 15 cm)
- Precisione molto alta: 10^{-14} AU (~ 1.5 mm)

La tolleranza di velocità è tipicamente $10^{-3} \times$ tolleranza di posizione.

9.5.3 Punti di Output

Tre strategie per l'output:

1. **Output denso:** Memorizza lo stato ad ogni passo di integrazione (molta memoria)
2. **Interpolazione:** Usa interpolazione Hermite tra i passi
3. **Output fisso:** Specifica tempi di output, l'integratore si ferma lì

AstDyn supporta tutte tre le modalità.

9.6 Modalità di Propagazione

9.6.1 Propagazione in Avanti e all'Indietro

Propagazione in avanti ($t_f > t_0$):

- Generazione standard di effemeridi
- Pianificazione missioni
- Previsione impatti

Propagazione all'indietro ($t_f < t_0$):

- Ricostruzione storia orbitale
- Ricerca di avvicinamenti passati
- Validazione determinazione orbitale

Gli integratori numerici funzionano ugualmente bene in entrambe le direzioni se il sistema è reversibile nel tempo.

9.6.2 Epoca Singola vs Multi-Epoca

Propagazione epoca singola:

```

1 Vector6d y0 = ...; // Stato iniziale
2 double t0 = 60000.0; // MJD TDB
3 double tf = 60365.0; // 1 anno dopo
4
5 Propagator prop(force_model);
6 Vector6d yf = prop.propagate(y0, t0, tf);

```

Listing 9.1: Propagazione epoca singola

Propagazione multi-epoca:

```

1 std::vector<double> epochs = {60000, 60100, 60200, 60300};
2 std::vector<Vector6d> states = prop.propagate_multi(y0, t0,
    epochs);

```

Listing 9.2: Propagazione multi-epoca

9.7 Generazione di Effemeridi

9.7.1 Effemeridi Tabulate

Per ricerche ripetute efficienti, creare una tabella:

```

1 EphemerisTable ephem;
2 double t_start = 60000.0;
3 double t_end = 61000.0;
4 double dt = 1.0; // Intervalli di 1 giorno
5
6 for (double t = t_start; t <= t_end; t += dt) {
7     Vector6d state = prop.propagate(y0, t0, t);
8     ephem.add_entry(t, state);
9 }
10
11 // Successivamente: interpola a tempo arbitrario
12 Vector6d state_interp = ephem.interpolate(60123.5);

```

Listing 9.3: Generazione tabella effemeridi

9.7.2 Interpolazione di Chebyshev

Per effemeridi ad alta precisione, il JPL usa polinomi di Chebyshev:

$$\mathbf{r}(t) = \sum_{k=0}^n c_k T_k(t') \quad (9.11)$$

dove T_k sono polinomi di Chebyshev e t' è normalizzato a $[-1, 1]$.

Vantaggi:

- Proprietà minimax (minimizza l'errore massimo)
- Stabile per polinomi di alto grado
- Valutazione veloce

9.8 Matrice di Transizione di Stato

9.8.1 Definizione

La **matrice di transizione di stato** (STM) $\Phi(t, t_0)$ relaziona le perturbazioni:

$$\delta\mathbf{y}(t) = \Phi(t, t_0)\delta\mathbf{y}(t_0) \quad (9.12)$$

È una matrice 6×6 che soddisfa:

$$\frac{d\Phi}{dt} = \mathbf{A}(t)\Phi, \quad \Phi(t_0, t_0) = \mathbf{I} \quad (9.13)$$

dove $\mathbf{A} = \partial\mathbf{f}/\partial\mathbf{y}$ è lo Jacobiano.

9.8.2 Applicazioni

La STM è essenziale per:

- Determinazione orbitale (correzione differenziale)
- Propagazione covarianza (quantificazione incertezza)
- Analisi di sensibilità
- Ottimizzazione manovre

9.8.3 Calcolo

Aumentare il vettore di stato:

$$\tilde{\mathbf{y}} = \begin{bmatrix} \mathbf{y} \\ \text{vec}(\Phi) \end{bmatrix} \quad (9.14)$$

dove $\text{vec}(\Phi)$ impila i 36 elementi di Φ in un vettore.

Il sistema aumentato è:

$$\frac{d\tilde{\mathbf{y}}}{dt} = \begin{bmatrix} \mathbf{f}(\mathbf{y}) \\ \mathbf{A}(\mathbf{y})\text{vec}(\Phi) \end{bmatrix} \quad (9.15)$$

9.9 Esempi Pratici

9.9.1 Esempio 1: Asteroide della Fascia Principale

Propagare l'asteroide 203 Pompeja per 1 anno:

```

1 #include <astdyn/propagation/Propagator.hpp>
2
3 using namespace astdyn;
4
5 // Elementi orbitali iniziali (da OrbFit)
6 OrbitalElements elements;
7 elements.epoch = 60000.0; // MJD TDB
8 elements.a = 2.743; // AU
9 elements.e = 0.0698;
10 elements.i = 11.78 * DEG_TO_RAD;
11 elements.Omega = 347.60 * DEG_TO_RAD;
12 elements.omega = 59.96 * DEG_TO_RAD;
13 elements.M = 164.35 * DEG_TO_RAD;
14
15 // Converti in Cartesiane
16 Vector6d state0 = elements.to_cartesian();
17
18 // Imposta modello di forza
19 ForceModel forces;
20 forces.enable_planets({"Jupiter", "Saturn", "Mars", "Earth"
    });

```

```
21 forces.enable_relativity(true);  
22  
23 // Crea propagatore  
24 Propagator prop(forces);  
25 prop.set_integrator("DOPRI54");  
26 prop.set_tolerance(1e-12);  
27  
28 // Propaga 1 anno  
29 double t0 = elements.epoch;  
30 double tf = t0 + 365.25;  
31  
32 Vector6d state_final = prop.propagate(state0, t0, tf);  
33  
34 // Riconverti in elementi  
35 OrbitalElements final_elements =  
36     OrbitalElements::from_cartesian(state_final, tf);  
37  
38 std::cout << "a iniziale: " << elements.a << " AU\n";  
39 std::cout << "a finale: " << final_elements.a << " AU\n";  
40 std::cout << "Variazione: " << (final_elements.a - elements  
    .a) * 1e6  
    << " km\n";  
41
```

Listing 9.4: Propagazione di Pompeja

9.9.2 Esempio 2: Analisi Avvicinamento Ravvicinato

Trovare la distanza minima dalla Terra:

```
1 double min_distance = 1e99;  
2 double closest_time = 0;  
3  
4 // Propaga con piccoli passi vicino all'incontro con la  
    Terra  
5 for (double t = t_start; t <= t_end; t += 0.01) {  
6     Vector6d asteroid_state = prop.propagate(y0, t0, t);  
7     Vector6d earth_state = ephemeris.get_planet("Earth", t)  
        ;  
8 }
```

```

9     Vector3d rel_pos = asteroid_state.head<3>() -
10    earth_state.head<3>();
11
12    double distance = rel_pos.norm();
13
14    if (distance < min_distance) {
15        min_distance = distance;
16        closest_time = t;
17    }
18
19 std::cout << "Avvicinamento minimo: " << min_distance << "
AU\n";
20 std::cout << "                    " << min_distance *
149597870.7 << " km\n";
21 std::cout << "All'epoca: " << closest_time << " MJD\n";

```

Listing 9.5: Rilevamento avvicinamento ravvicinato

9.9.3 Esempio 3: Propagazione di Cometa

Gestire grande eccentricità vicino al perielio:

```

1 // Cometa con e = 0.995, q = 0.1 AU
2 OrbitalElements comet;
3 comet.a = 20.0; // AU (molto eccentrica)
4 comet.e = 0.995;
5 comet.q = comet.a * (1 - comet.e); // distanza perielica
6
7 // Usa passo variabile, tolleranza piu' stretta
8 prop.set_tolerance(1e-14);
9 prop.set_min_step(1e-4); // Permetti passi molto piccoli
10    vicino al perielio
11 prop.set_max_step(30.0); // Passi grandi all'afelio
12
13 Vector6d state0 = comet.to_cartesian();
14 Vector6d state_post_perihelion = prop.propagate(state0, t0,
15         t0 + 180);

```

Listing 9.6: Propagazione cometa

9.10 Ottimizzazione delle Prestazioni

9.10.1 Selezione del Modello di Forza

Includere solo le perturbazioni necessarie:

Oggetto	Forze Essenziali	Opzionali
Asteroide fascia principale	Sole, Gio, Sat	Marte, Terra, relatività
Asteroide vicino alla Terra	Sole, tutti pianeti	Relatività, asteroidi
Troiano di Giove	Sole, Gio, Sat	Urano, Nettuno
Trans-Nettuniano	Sole, Gio, Sat, Ura, Net	Relatività

Tabella 9.1: Modelli di forza raccomandati per diversi tipi di oggetti.

9.10.2 Passo Adattativo vs Fisso

Passo adattativo (DOPRI54, RK78):

- Pro: Controllo errore automatico, efficiente
- Contro: Sequenza passi non deterministica
- Uso: Determinazione orbitale, generazione effemeridi

Passo fisso (RK4, Leapfrog):

- Pro: Prevedibile, parallelizzabile
- Contro: Deve scegliere passo attentamente
- Uso: Evoluzione lungo termine, simulazioni ensemble

9.10.3 Parallelizzazione

Per propagare molti oggetti:

```
1 #include <omp.h>
2
3 std::vector<Vector6d> initial_states = ...;
4 std::vector<Vector6d> final_states(initial_states.size());
5
6 #pragma omp parallel for
7 for (size_t i = 0; i < initial_states.size(); ++i) {
```

```

8 Propagator prop(forces); // Ogni thread ha il suo
9   propagatore
10  final_states[i] = prop.propagate(initial_states[i], t0,
11      tf);
12 }
```

Listing 9.7: Propagazione parallela

9.11 Validazione dell'Accuratezza

9.11.1 Conservazione dell'Energia

Per sistemi conservativi (no SRP, attrito), l'energia deve essere conservata:

$$E = \frac{v^2}{2} - \frac{\mu}{r} = \text{costante} \quad (9.16)$$

Controllo errore energia:

```

1 double E0 = 0.5 * v0.squaredNorm() - MU_SUN / r0.norm();
2 double Ef = 0.5 * vf.squaredNorm() - MU_SUN / rf.norm();
3 double dE = std::abs(Ef - E0);
4 std::cout << "Errore energia: " << dE / std::abs(E0) * 100
5   << "%\n";
```

Listing 9.8: Controllo energia

Per integratori di alta qualità: $\Delta E/E < 10^{-10}$

9.11.2 Confronto Problema a Due Corpi

Validare contro soluzione kepleriana analitica:

```

1 // Propagazione numerica (con perturbazioni spente)
2 Vector6d state_num = prop.propagate(y0, t0, tf);
3
4 // Propagazione kepleriana analitica
5 OrbitalElements elem0 = OrbitalElements::from_cartesian(y0,
6   t0);
7 elem0.propagate_mean_anomaly(tf - t0);
8 Vector6d state_kep = elem0.to_cartesian();
```

```
9 // Confronta
10 Vector3d pos_diff = state_num.head<3>() - state_kep.head
11 <3>();
12 std::cout << "Differenza posizione: " << pos_diff.norm() *
    AU_TO_KM
    << " km\n";
```

Listing 9.9: Confronto kepleriano

Atteso: < 1 km per archi brevi, < 100 km per 1 anno.

9.12 Riepilogo

Concetti chiave sulla propagazione orbitale:

1. La **propagazione** calcola stati futuri/passati da condizioni iniziali
2. I **modelli di forza** devono includere tutte le perturbazioni significative
3. Gli **integratori adattativi** (DOPRI54) bilanciano accuratezza ed efficienza
4. Il **passo** dipende dal periodo orbitale e dall'eccentricità
5. La **matrice di transizione di stato** abilita la determinazione orbitale
6. I **sistemi di riferimento** devono essere consistenti
7. **Validazione** tramite conservazione energia e confronti analitici

Comprendere la propagazione orbitale è essenziale per:

- Generare effemeridi accurate
- Pianificare osservazioni e missioni
- Valutare rischi di collisione
- Studiare dinamica a lungo termine
- Determinazione orbitale (prossimo capitolo)

Nel prossimo capitolo, useremo la propagazione con la matrice di transizione di stato per la determinazione orbitale precisa dalle osservazioni.

Capitolo 10

Matrice di Transizione di Stato

10.1 Introduzione

La **matrice di transizione di stato** (STM) è fondamentale per la determinazione orbitale, tracciando piccole perturbazioni nel moto orbitale e propagando le incertezze. Questo capitolo sviluppa la teoria matematica e il calcolo pratico della STM.

10.2 Fondamenti Matematici

10.2.1 Linearizzazione della Dinamica

Consideriamo la dinamica orbitale generale:

$$\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y}) \quad (10.1)$$

dove $\mathbf{y} = [\mathbf{r}, \mathbf{v}]^T$ è il vettore di stato 6-dimensionale.

Per una traiettoria di riferimento $\mathbf{y}_{\text{rif}}(t)$ e una traiettoria perturbata $\mathbf{y}(t)$, definiamo:

$$\delta\mathbf{y}(t) = \mathbf{y}(t) - \mathbf{y}_{\text{rif}}(t) \quad (10.2)$$

10.2.2 Equazioni Variazionali

Assumendo piccole perturbazioni, linearizziamo:

$$\delta \dot{\mathbf{y}} = \frac{\partial \mathbf{f}}{\partial \mathbf{y}} \Big|_{\mathbf{y}_{\text{rif}}} \delta \mathbf{y} = \mathbf{A}(t) \delta \mathbf{y} \quad (10.3)$$

dove $\mathbf{A}(t)$ è la matrice Jacobiana 6×6 :

$$\mathbf{A} = \begin{bmatrix} \frac{\partial \mathbf{f}_r}{\partial \mathbf{r}} & \frac{\partial \mathbf{f}_r}{\partial \mathbf{v}} \\ \frac{\partial \mathbf{f}_v}{\partial \mathbf{r}} & \frac{\partial \mathbf{f}_v}{\partial \mathbf{v}} \end{bmatrix} = \begin{bmatrix} \mathbf{0}_{3 \times 3} & \mathbf{I}_{3 \times 3} \\ \frac{\partial \mathbf{a}}{\partial \mathbf{r}} & \frac{\partial \mathbf{a}}{\partial \mathbf{v}} \end{bmatrix} \quad (10.4)$$

10.2.3 Definizione Matrice di Transizione di Stato

La **matrice di transizione di stato** $\Phi(t, t_0)$ è la soluzione di:

$$\frac{d\Phi}{dt} = \mathbf{A}(t)\Phi(t, t_0), \quad \Phi(t_0, t_0) = \mathbf{I}_{6 \times 6} \quad (10.5)$$

Essa relaziona le perturbazioni di stato a tempi diversi:

$$\delta \mathbf{y}(t) = \Phi(t, t_0) \delta \mathbf{y}(t_0) \quad (10.6)$$

10.2.4 Proprietà

La STM ha proprietà importanti:

1. **Identità a t_0 :** $\Phi(t_0, t_0) = \mathbf{I}$
2. **Composizione:** $\Phi(t_2, t_0) = \Phi(t_2, t_1)\Phi(t_1, t_0)$
3. **Inversa:** $\Phi(t_0, t) = \Phi^{-1}(t, t_0)$
4. **Determinante:** $\det[\Phi(t, t_0)] = \exp \left[\int_{t_0}^t \text{tr}(\mathbf{A}(\tau)) d\tau \right]$

Per sistemi conservativi (Hamiltoniani), la STM è simplettica: $\Phi^T \mathbf{J} \Phi = \mathbf{J}$ dove \mathbf{J} è la matrice simplettica.

10.3 Calcolo della Matrice Jacobiana

10.3.1 Problema dei Due Corpi

Per il problema kepleriano non perturbato:

$$\mathbf{a} = -\frac{\mu}{r^3} \mathbf{r} \quad (10.7)$$

Le derivate parziali dell'accelerazione sono:

$$\frac{\partial \mathbf{a}}{\partial \mathbf{r}} = -\frac{\mu}{r^3} \left[\mathbf{I} - 3 \frac{\mathbf{r}\mathbf{r}^T}{r^2} \right] \quad (10.8)$$

$$\frac{\partial \mathbf{a}}{\partial \mathbf{v}} = \mathbf{0}_{3 \times 3} \quad (10.9)$$

Quindi:

$$\mathbf{A}_{2\text{-corpi}} = \begin{bmatrix} \mathbf{0} & \mathbf{I} \\ -\frac{\mu}{r^3} \left[\mathbf{I} - 3 \frac{\mathbf{r}\mathbf{r}^T}{r^2} \right] & \mathbf{0} \end{bmatrix} \quad (10.10)$$

10.3.2 Perturbazioni N-Corpi

Per perturbazioni planetarie, l'accelerazione è:

$$\mathbf{a}_p = \mu_p \left[\frac{\mathbf{r}_p - \mathbf{r}}{|\mathbf{r}_p - \mathbf{r}|^3} - \frac{\mathbf{r}_p}{r_p^3} \right] \quad (10.11)$$

La derivata parziale rispetto alla posizione:

$$\frac{\partial \mathbf{a}_p}{\partial \mathbf{r}} = -\frac{\mu_p}{d^3} \left[\mathbf{I} - 3 \frac{\mathbf{d}\mathbf{d}^T}{d^2} \right] \quad (10.12)$$

dove $\mathbf{d} = \mathbf{r}_p - \mathbf{r}$ e $d = |\mathbf{d}|$.

10.3.3 Correzioni Relativistiche

L'accelerazione post-Newtoniana include termini dipendenti dalla velocità:

$$\mathbf{a}_{\text{GR}} = \frac{\mu}{c^2 r^3} \left[4 \frac{\mu}{r} \mathbf{r} - v^2 \mathbf{r} + 4(\mathbf{r} \cdot \mathbf{v}) \mathbf{v} \right] \quad (10.13)$$

Sia $\partial \mathbf{a}_{\text{GR}} / \partial \mathbf{r}$ che $\partial \mathbf{a}_{\text{GR}} / \partial \mathbf{v}$ sono non-zero.

Per la posizione:

$$\frac{\partial \mathbf{a}_{\text{GR}}}{\partial \mathbf{r}} = \frac{\mu}{c^2 r^3} \left[-v^2 \mathbf{I} + 4(\mathbf{v}\mathbf{v}^T) + (\text{termini ordine superiore}) \right] \quad (10.14)$$

Per la velocità:

$$\frac{\partial \mathbf{a}_{\text{GR}}}{\partial \mathbf{v}} = \frac{\mu}{c^2 r^3} \left[-2v\mathbf{r}\mathbf{v}^T + 4\mathbf{v}\mathbf{r}^T + 4(\mathbf{r} \cdot \mathbf{v}) \mathbf{I} \right] \quad (10.15)$$

10.3.4 Pressione di Radiazione Solare

Per SRP con rapporto area-massa costante:

$$\mathbf{a}_{\text{SRP}} = P_{\odot} \frac{A}{m} C_R \left(\frac{r_0}{r} \right)^2 \hat{\mathbf{r}} \quad (10.16)$$

La parziale è:

$$\frac{\partial \mathbf{a}_{\text{SRP}}}{\partial \mathbf{r}} = P_{\odot} \frac{A}{m} C_R r_0^2 \left[\frac{\mathbf{I}}{r^3} - 3 \frac{\mathbf{r} \mathbf{r}^T}{r^5} \right] \quad (10.17)$$

10.4 Calcolo Numerico

10.4.1 Vettore di Stato Aumentato

Per calcolare la STM numericamente, aumentiamo il vettore di stato:

$$\tilde{\mathbf{y}} = \begin{bmatrix} \mathbf{y} \\ \text{vec}(\Phi) \end{bmatrix} \in \mathbb{R}^{42} \quad (10.18)$$

dove $\text{vec}(\Phi)$ impila i 36 elementi di Φ per colonne.

10.4.2 Dinamica Aumentata

Il sistema aumentato è:

$$\frac{d\tilde{\mathbf{y}}}{dt} = \begin{bmatrix} \mathbf{f}(\mathbf{y}) \\ \text{vec}(\mathbf{A}(\mathbf{y})\Phi) \end{bmatrix} \quad (10.19)$$

In pratica, integriamo:

- 6 equazioni per lo stato \mathbf{y}
- 36 equazioni per gli elementi STM
- Totale: 42 ODE accoppiate

10.4.3 Implementazione in AstDyn

```
1 #include <astdyn/propagation/Propagator.hpp>
2
3 using namespace astdyn;
```

```

4
5 Vector6d y0 = ...; // Stato iniziale
6 double t0 = 60000.0;
7 double tf = 60100.0;
8
9 Propagator prop(force_model);
10 prop.enable_stm(true); // Abilita calcolo STM
11
12 // Propaga stato e STM
13 auto result = prop.propagate_with_stm(y0, t0, tf);
14
15 Vector6d yf = result.state;
16 Matrix6d Phi = result.stm; // Matrice 6x6 di transizione
17           stato
18
19 std::cout << "Determinante STM: " << Phi.determinant() << "\n";
20 std::cout << "Numero condizione STM: "
21           << Phi.norm() * Phi.inverse().norm() << "\n";

```

Listing 10.1: Propagazione STM

10.4.4 Costo Computazionale

Il calcolo STM aumenta il costo computazionale:

Calcolo	Equazioni Stato	Fattore Tempo CPU
Solo stato	6	1.0×
Stato + STM	42	5-7×
Stato + STM + sensibilità	$42 + 6N_p$	10-15×

Tabella 10.1: Costo computazionale propagazione STM. N_p è il numero di parametri.

10.5 Applicazioni

10.5.1 Determinazione Orbitale

Nella correzione differenziale (fit orbitale minimi quadrati), necessitiamo:

$$\frac{\partial \mathbf{y}(t_{\text{oss}})}{\partial \mathbf{y}(t_0)} = \Phi(t_{\text{oss}}, t_0) \quad (10.20)$$

Questo relaziona le osservazioni alle condizioni iniziali, abilitando il raffinamento orbitale iterativo.

10.5.2 Propagazione Covarianza

Data la covarianza iniziale \mathbf{P}_0 , la covarianza al tempo t è:

$$\mathbf{P}(t) = \Phi(t, t_0) \mathbf{P}_0 \Phi^T(t, t_0) \quad (10.21)$$

Questo quantifica la crescita dell'incertezza nel tempo.

Esempio:

```
1 Matrix6d P0 = initial_covariance(); // km^2, (km/s)^2
2 Matrix6d Phi = result.stm;
3
4 Matrix6d Pf = Phi * P0 * Phi.transpose();
5
6 // Incertezza posizione al tempo finale
7 Vector3d sigma_pos = Pf.block<3,3>(0,0).diagonal().
8     cwiseSqrt();
9 std::cout << "Incertezza posizione: "
10    << sigma_pos.transpose() << " km\n";
```

Listing 10.2: Propagazione covarianza

10.5.3 Analisi di Sensibilità

La STM rivela come le perturbazioni nelle condizioni iniziali influenzano gli stati futuri:

$$\frac{\partial r(t)}{\partial r_0} = \Phi_{11}(t, t_0), \quad \frac{\partial r(t)}{\partial v_0} = \Phi_{12}(t, t_0) \quad (10.22)$$

Questi sono i blocchi 3×3 superiore-sinistro e superiore-destro di Φ .

10.5.4 Ottimizzazione Manovre

Per la progettazione di traiettorie spaziali, la STM aiuta a calcolare:

- Matrici di puntamento (dove mirare per colpire un bersaglio)
- Requisiti Δv
- Sensibilità a errori di esecuzione

10.6 STM Analitica vs Numerica

10.6.1 STM Analitica per Moto Kepleriano

Per il problema non perturbato dei due corpi, esistono soluzioni in forma chiusa.
La STM può essere espressa in termini di elementi orbitali e loro derivate.

Vantaggi:

- Esatta (nessun errore numerico)
- Veloce da valutare
- Valida per lunghi intervalli temporali

Svantaggi:

- Formule complesse (specialmente vicino a singolarità)
- Non include perturbazioni
- Uso pratico limitato

10.6.2 STM Numerica

Integrando numericamente le equazioni variazionali:

Vantaggi:

- Gestisce modelli di forza arbitrari
- Implementazione diretta
- Include tutte le perturbazioni

Svantaggi:

- Accumulo errore numerico
- $7\times$ più lenta della propagazione solo-stato
- Mal condizionamento per archi lunghi

10.6.3 Approcci Ibridi

Per alcune applicazioni, usare:

1. STM analitica per parte kepleriana
2. Correzioni perturbazione numeriche
3. Composizione transizione stato

10.7 Stabilità Numerica

10.7.1 Problemi di Condizionamento

La STM diventa mal condizionata per:

- Tempi di propagazione lunghi ($>$ diversi periodi orbitali)
- Orbite ad alta eccentricità
- Moto quasi-rettilineo

Crescita del numero di condizione:

$$\kappa(\Phi) \approx \exp(\lambda_{\max} \Delta t) \quad (10.23)$$

dove λ_{\max} è il più grande esponente di Lyapunov.

10.7.2 Strategie di Mitigazione

1. Rilinearizzazione

Invece di propagare da t_0 a t_f , dividere in segmenti:

$$\Phi(t_f, t_0) = \Phi(t_f, t_2)\Phi(t_2, t_1)\Phi(t_1, t_0) \quad (10.24)$$

Ogni segmento ha migliore condizionamento.

2. Transizione stato in elementi orbitali

Invece della STM cartesiana, usare:

$$\frac{\partial \mathbf{e}(t)}{\partial \mathbf{e}(t_0)} \quad (10.25)$$

dove $\mathbf{e} = [a, e, i, \Omega, \omega, M]$ sono elementi orbitali.

3. Regolarizzazione

Usare coordinate regolarizzate (Kustaanheimo-Stiefel, Sperling-Burdet) che si comportano meglio vicino al periassese.

10.8 Esempio Pratico

10.8.1 Tracciamento Bersaglio

Tracciare l'incertezza nella posizione asteroidale per valutazione impatto:

```

1 // Stato iniziale da determinazione orbitale
2 Vector6d y0 = {1.1, 0.2, 0.05, -0.01, 0.03, 0.0}; // AU,
   AU/giorno
3
4 // Covarianza iniziale (da fit minimi quadrati)
5 Matrix6d P0 = Matrix6d::Zero();
6 P0.diagonal() << 1e-8, 1e-8, 1e-9, // pos: 1500 km
   1e-11, 1e-11, 1e-12; // vel: 0.15 m/s
7
8
9 ForceModel forces;
10 forces.enable_planets({"Earth", "Jupiter", "Venus", "Mars"
   });
11
12 Propagator prop(forces);
13 prop.enable_stm(true);
14
15 // Propaga 10 anni
16 double t0 = 60000.0;
17 double tf = t0 + 3652.5; // 10 anni
18
19 auto result = prop.propagate_with_stm(y0, t0, tf);
20
21 // Calcola incertezza al tempo futuro
22 Matrix6d Pf = result.stm * P0 * result.stm.transpose();
23
24 // Incertezza posizione (3-sigma)
25 Vector3d sigma_3 = 3.0 * Pf.block<3,3>(0,0).diagonal().
   cwiseSqrt();

```

```

26 std::cout << "Incognita posizione (3-sigma): \n";
27 std::cout << sigma_3.transpose() * AU_TO_KM << " km\n";
28
29 // Controlla avvicinamento ravvicinato Terra
30 Vector6d earth_state = ephemeris.get_planet("Earth", tf);
31 Vector3d rel_pos = result.state.head<3>() - earth_state.
32     head<3>();
33 double distance = rel_pos.norm() * AU_TO_KM;
34
35 std::cout << "Distanza dalla Terra: " << distance << " km\n"
36     ";
37 std::cout << "Probabilita' impatto (Gaussiana): ";
38 if (distance < 3.0 * sigma_3.norm() * AU_TO_KM) {
39     std::cout << "NON-ZERO - richiesta ulteriore analisi\n"
40         ;
41 } else {
42     std::cout << "Trascurabile\n";
43 }

```

Listing 10.3: Propagazione incertezza asteroide

10.8.2 Pianificazione Osservazioni

Determinare tempi ottimali di osservazione per ridurre incertezza:

```

1 // Propaga con STM a epoche osservative multiple
2 std::vector<double> obs_times = {t0 + 30, t0 + 60, t0 +
3     90};
4
5 for (double t_obs : obs_times) {
6     auto result = prop.propagate_with_stm(y0, t0, t_obs);
7     Matrix6d P = result.stm * P0 * result.stm.transpose();
8
9     // Incertezza RA/Dec da incertezza posizione
10    Vector3d r = result.state.head<3>();
11    double dec = std::asin(r(2) / r.norm());
12    double ra = std::atan2(r(1), r(0));

```

```

13 // Approssimazione semplice (calcolo completo usa
14 // parziali osservazione)
15 double sigma_ra = P(0,0) / (r.norm() * std::cos(dec));
16 double sigma_dec = P(2,2) / r.norm();
17
18 std::cout << "Epoca " << t_obs << ":" 
19             << "sigma_RA = " << sigma_ra * RAD_TO_ARCSEC
20             << " arcosec, "
21             << "sigma_Dec = " << sigma_dec *
22             RAD_TO_ARCSEC << " arcosec\n";
23 }
```

Listing 10.4: Pianificazione osservazioni

10.9 Sensibilità Parametri

10.9.1 Vettore di Stato Esteso

Per studiare la sensibilità a parametri dinamici (es. μ , C_R , masse asteroidi), aumentare lo stato:

$$\tilde{\mathbf{y}} = \begin{bmatrix} \mathbf{y} \\ \mathbf{p} \end{bmatrix} \quad (10.26)$$

dove \mathbf{p} sono parametri. Quindi:

$$\frac{d}{dt} \begin{bmatrix} \mathbf{y} \\ \mathbf{p} \end{bmatrix} = \begin{bmatrix} \mathbf{f}(\mathbf{y}, \mathbf{p}) \\ \mathbf{0} \end{bmatrix} \quad (10.27)$$

La STM estesa include $\partial\mathbf{y}/\partial\mathbf{p}$.

10.9.2 Matrici di Sensibilità

Definire matrice di sensibilità:

$$\mathbf{S}(t) = \frac{\partial \mathbf{y}(t)}{\partial \mathbf{p}} \quad (10.28)$$

Essa soddisfa:

$$\frac{d\mathbf{S}}{dt} = \mathbf{A}(t)\mathbf{S} + \frac{\partial \mathbf{f}}{\partial \mathbf{p}} \quad (10.29)$$

Questo rivela come il moto orbitale dipende dai parametri fisici.

10.10 Riepilogo

Concetti chiave sulla matrice di transizione di stato:

1. La **STM** $\Phi(t, t_0)$ propaga linearmente piccole perturbazioni
2. Soddisfa le **equazioni variazionali**: $\dot{\Phi} = \mathbf{A}(t)\Phi$
3. La **matrice Jacobiana** \mathbf{A} contiene derivate del modello di forza
4. Il **calcolo numerico** richiede integrazione di 42 ODE (6 stato + 36 STM)
5. **Applicazioni**: determinazione orbitale, propagazione covarianza, analisi sensibilità
6. Il **condizionamento** degrada per archi lunghi; usare rilinearizzazione
7. La **STM estesa** include sensibilità parametri

Comprendere la STM è essenziale per:

- Determinazione orbitale precisa (Capitolo 14)
- Quantificazione incertezza
- Progettazione missioni e puntamento
- Stima parametri
- Valutazione probabilità impatto

Il prossimo capitolo copre il calcolo di effemeridi e metodi di interpolazione per ricerca efficiente dello stato.

Capitolo 11

Calcolo di Effemeridi

11.1 Introduzione

Un'effemeride (plurale: *effemeridi*) è una tabella o funzione che fornisce posizioni (e optionalmente velocità) di corpi celesti a tempi specifici. Effemeridi accurate sono essenziali per:

- Calcolare posizioni previste per osservazioni
- Ridurre misure astrometriche
- Pianificare missioni spaziali
- Analizzare avvicinamenti ravvicinati
- Studiare dinamica orbitale

Questo capitolo copre metodi per generare, memorizzare e interpolare effemeridi efficientemente.

11.2 Tipi di Effemeridi

11.2.1 Effemeridi Planetarie

I pianeti maggiori richiedono la massima accuratezza:

JPL Development Ephemerides (DE) Integrazione numerica del sistema solare, inclusi Luna e grandi asteroidi. Attuali: DE440 (ottimizzazione Terra-Luna), DE441 (sistema solare esterno).

VSOP87 Teoria analitica del Bureau des Longitudes. Espansione in serie in elementi orbitali. Accuratezza: ~ 1 arcosec su millenni.

INPOP Effemeride francese dell'IMCCE, ottimizzata per ranging radar planetario.

11.2.2 Effemeridi di Piccoli Corpi

Asteroidi e comete:

- Calcolate da elementi orbitali via propagazione
- Archiviate nel database MPC (Minor Planet Center)
- Precisione variabile: 0.1 arcosec (ben osservati) a 10 arcmin (singola opposizione)

11.2.3 Effemeridi di Veicoli Spaziali

Missioni interplanetarie:

- Kernel SPICE (file SPK) dai team di navigazione
- Segmenti polinomiali Chebyshev
- Accuratezza a livello di metro per fasi di avvicinamento

11.3 Rappresentazioni di Effemeridi

11.3.1 Formato Tabulato

Rappresentazione più semplice: coppie discrete tempo-stato.

MJD (TDB)	x (AU)	y (AU)	z (AU)	\dot{x}	\dot{y}	\dot{z}
60000.0	1.234	0.567	0.123	-0.012	0.015	0.003
60001.0	1.222	0.582	0.126	-0.012	0.015	0.003
60002.0	1.210	0.597	0.129	-0.012	0.015	0.003
:	:	:	:	:	:	:

Tabella 11.1: Esempio effemeride tabulata con spaziatura di 1 giorno.

Vantaggi:

- Facile da implementare
- Ricerca diretta per tempi tabulati

Svantaggi:

- Ampia memorizzazione per alta cadenza
- Richiede interpolazione tra punti
- Griglia temporale fissa (inefficiente per orbite eccentriche)

11.3.2 Rappresentazione Polinomiale

Rappresentare la posizione come polinomio:

$$\mathbf{r}(t) = \sum_{k=0}^n \mathbf{c}_k (t - t_0)^k \quad (11.1)$$

Tipicamente usata a tratti su segmenti (spline).

11.3.3 Polinomi di Chebyshev

Metodo preferito del JPL. Per intervallo temporale $[t_a, t_b]$, rappresentare:

$$\mathbf{r}(t) = \sum_{k=0}^n \mathbf{a}_k T_k \left(\frac{2t - t_a - t_b}{t_b - t_a} \right) \quad (11.2)$$

dove $T_k(x)$ sono polinomi di Chebyshev:

$$T_0(x) = 1, \quad T_1(x) = x, \quad T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x) \quad (11.3)$$

Proprietà:

- Distribuzione errore minimax (approssimazione ottimale)
- Stabile per alti gradi ($n \sim 15$)
- Valutazione efficiente via ricorrenza

11.3.4 Serie di Fourier

Per orbite quasi circolari:

$$\mathbf{r}(t) = \sum_{k=-N}^N \mathbf{c}_k e^{ik\omega t} \quad (11.4)$$

Usata in teorie planetarie analitiche (VSOP87).

11.4 Metodi di Interpolazione

11.4.1 Interpolazione Lineare

Dati punti (t_1, \mathbf{r}_1) e (t_2, \mathbf{r}_2) :

$$\mathbf{r}(t) = \mathbf{r}_1 + \frac{t - t_1}{t_2 - t_1} (\mathbf{r}_2 - \mathbf{r}_1) \quad (11.5)$$

Accuratezza: Primo ordine, errore $O(h^2)$ dove $h = t_2 - t_1$.

Uso: Ricerche veloci quando alta precisione non richiesta (> 1 km accettabile).

11.4.2 Interpolazione di Lagrange

Usare $n + 1$ punti per costruire polinomio di grado n :

$$\mathbf{r}(t) = \sum_{i=0}^n \mathbf{r}_i L_i(t) \quad (11.6)$$

dove i polinomi base di Lagrange sono:

$$L_i(t) = \prod_{\substack{j=0 \\ j \neq i}}^n \frac{t - t_j}{t_i - t_j} \quad (11.7)$$

Scelta tipica: $n = 6$ a 10 (ordine 7 a 11).

Accuratezza: Per ordine 8 con spaziatura 1 giorno, errore ~ 10 m per orbite asteroidali tipiche.

11.4.3 Interpolazione di Hermite

Usa sia posizioni che velocità. Per intervallo $[t_1, t_2]$:

$$\mathbf{r}(t) = \mathbf{r}_1 H_0(s) + \mathbf{r}_2 H_1(s) + h \dot{\mathbf{r}}_1 H_2(s) + h \dot{\mathbf{r}}_2 H_3(s) \quad (11.8)$$

dove $s = (t - t_1)/h$, $h = t_2 - t_1$, e le funzioni base di Hermite sono:

$$H_0(s) = (1 + 2s)(1 - s)^2 \quad (11.9)$$

$$H_1(s) = s^2(3 - 2s) \quad (11.10)$$

$$H_2(s) = s(1 - s)^2 \quad (11.11)$$

$$H_3(s) = s^2(s - 1) \quad (11.12)$$

Vantaggi:

- Accuratezza cubica ($O(h^4)$)
- Velocità lisce (derivata prima continua)
- Richiede solo due punti

Accuratezza: Con spaziatura 1 giorno, errore ~ 1 m per orbite ben comportate.

11.4.4 Interpolazione Spline

Le spline cubiche forniscono interpolazione liscia attraverso tutti i punti con derivate seconde continue.

Per punti (t_i, \mathbf{r}_i) , $i = 0, \dots, n$, costruire cubiche a tratti $\mathbf{s}_i(t)$ su $[t_i, t_{i+1}]$ tali che:

- $\mathbf{s}_i(t_i) = \mathbf{r}_i$ (interpolazione)
- $\mathbf{s}'_i(t_{i+1}) = \mathbf{s}'_{i+1}(t_{i+1})$ (derivata prima continua)
- $\mathbf{s}''_i(t_{i+1}) = \mathbf{s}''_{i+1}(t_{i+1})$ (derivata seconda continua)

Uso: Quando accelerazione liscia è importante (propagazione covarianza).

11.5 Sistema SPICE

11.5.1 Panoramica

SPICE (Spacecraft Planet Instrument C-matrix Events) è il toolkit standard NASA per geometria missioni spaziali:

SPK (Kernel effemeridi) Posizione e velocità

CK (Kernel orientamento) Assetto veicolo spaziale

PCK (Kernel costanti) Parametri fisici, forme corpi

IK (Kernel strumento) FOV, boresight

FK (Kernel frame) Definizioni sistemi riferimento

LSK (Kernel secondi intercalari) Conversioni tempo

11.5.2 File SPK

File binari contenenti segmenti polinomiali Chebyshev o Hermite.

Uso in AstDyn:

```
1 #include <astdyn/ephemeris/SpiceInterface.hpp>
2
3 SpiceInterface spice;
4 spice.load_kernel("de440.bsp"); // Effemeride planetaria
5 spice.load_kernel("codes_300ast_20100725.bsp"); // 
6   Asteroidi
7
8 // Interroga posizione Giove all'epoca
9 double et = spice.mjd_to_et(60000.0); // Converti MJD in
10 ET
11 Vector6d jupiter_state = spice.get_state("JUPITER", et, "
12 ECLIPJ2000", "SUN");
13
14 std::cout << "Posizione Giove: " << jupiter_state.head<3>()
15   .transpose()
16   << " km\n";
```

Listing 11.1: Caricamento kernel SPICE

11.5.3 ID NAIF

SPICE usa ID interi:

- Sole: 10
- Pianeti: 199 (Mercurio), 299 (Venere), 399 (Terra), 499 (Marte), 599 (Giove), ecc.
- Luna: 301

- Asteroidi: 2000001 (Cerere), 2000004 (Vesta), 2000203 (Pompeja)

11.6 Effemeridi Planetarie

11.6.1 JPL Development Ephemerides

DE440/441 (rilasciate 2020):

- Coprono anni 1550–2650
- Includono Sole, pianeti, Luna, Plutone, 343 asteroidi
- Fit a dati ranging (missioni Marte), VLBI, LLR
- Accuratezza: ~ 1 km per pianeti interni, ~ 10 km per pianeti esterni

Dimensioni file:

- DE440: 114 MB (standard)
- DE441: 3.2 GB (include Luna alta frequenza)

11.6.2 VSOP87

Serie analitica sviluppata da Bretagnon & Francou (1988).

Varianti:

VSOP87A Rettangolare eliocentrico, eclittica J2000

VSOP87B Rettangolare eliocentrico, equatoriale J2000

VSOP87C Sferico eliocentrico (eclittica/equinozio medio della data)

VSOP87D Sferico eliocentrico (eclittica J2000)

VSOP87E Rettangolare baricentrico, eclittica J2000

Implementazione:

```
1 #include <astdyn/ephemeris/VSOP87.hpp>
2
3 VSOP87 vsop;
4 double jd = 2460000.5; // Data giuliana
5
```

```

6 // Posizione Terra (VSOP87A: eliocentrica eclittica J2000)
7 Vector3d earth_pos = vsop.get_position("Earth", jd,
8     VSOP87_A);
9 std::cout << "Posizione Terra: " << earth_pos.transpose()
10    << " AU\n";
11
12 // Stima accuratezza
13 double error_km = vsop.estimated_error("Earth", jd);
14 std::cout << "Errore posizione: ~" << error_km << " km\n";

```

Listing 11.2: Uso VSOP87

Accuratezza: ~ 1 km per pianeti interni su ± 2000 anni da J2000.

11.6.3 Confronto

Metodo	Accuratezza	Velocità	Dimensione File
DE440 (SPICE)	1–10 km	Veloce	114 MB
VSOP87	1–5 km	Media	~ 1 MB (codice)
Kepleriana	100–1000 km	Molto veloce	Trascurabile

Tabella 11.2: Confronto effemeridi planetarie.

11.7 Correzioni Tempo-Luce

11.7.1 Posizione Geometrica vs Apparente

La luce viaggia a velocità finita $c = 299792.458$ km/s, quindi osserviamo i pianeti dove *erano*, non dove *sono*.

Tempo-luce:

$$\tau = \frac{|\mathbf{r}_{\text{pianeta}} - \mathbf{r}_{\text{oss}}|}{c} \quad (11.13)$$

Valori tipici:

- Sole: 8.3 minuti
- Giove: 30–50 minuti
- Saturno: 70–90 minuti
- Nettuno: 4 ore

11.7.2 Correzione Iterativa

Per trovare la **posizione apparente** al tempo osservativo t_{oss} :

1. Inizia con posizione geometrica: $\mathbf{r}_0 = \mathbf{r}_{\text{pianeta}}(t_{\text{oss}})$
2. Calcola tempo-luce: $\tau_0 = |\mathbf{r}_0 - \mathbf{r}_{\text{oss}}|/c$
3. Aggiorna: $\mathbf{r}_1 = \mathbf{r}_{\text{pianeta}}(t_{\text{oss}} - \tau_0)$
4. Itera fino a convergenza: $|\tau_{i+1} - \tau_i| < 10^{-6}$ s

Tipicamente converge in 2–3 iterazioni.

11.7.3 Implementazione

```

1  Vector3d compute_apparent_position(
2      const EphemerisInterface& ephem,
3      const std::string& target,
4      double t_obs,
5      const Vector3d& observer_pos)
6  {
7      const double c_AU_per_day = 173.1446326846693; // 
8          Velocità luce
9
10     Vector3d r_geom = ephem.get_position(target, t_obs);
11     double tau = (r_geom - observer_pos).norm() /
12         c_AU_per_day;
13
14     // Itera correzione tempo-luce
15     for (int iter = 0; iter < 5; ++iter) {
16
17         Vector3d r_new = ephem.get_position(target, t_obs -
18             tau);
19         double tau_new = (r_new - observer_pos).norm() /
20             c_AU_per_day;
21
22         if (std::abs(tau_new - tau) < 1e-10) break; //
23             Convergenza
24         tau = tau_new;
25     }

```

```
20  
21     return ephem.get_position(target, t_obs - tau);  
22 }
```

Listing 11.3: Correzione tempo-luce

11.7.4 Aberrazione

Il moto dell'osservatore causa **aberrazione stellare** aggiuntiva:

$$\Delta\theta \approx \frac{v_{\text{oss}}}{c} \quad (11.14)$$

Per moto orbitale terrestre ($v \approx 30 \text{ km/s}$): $\Delta\theta \approx 20.5 \text{ arcosec}$ (aberrazione annua).

Correzione:

$$\hat{\mathbf{r}}_{\text{aberrato}} = \hat{\mathbf{r}} + \frac{\mathbf{v}_{\text{oss}}}{c} \quad (11.15)$$

11.8 Generazione Pratica Effemeridi

11.8.1 Considerazioni di Progetto

Scegliere parametri effemeride in base ai requisiti:

Applicazione	Spaziatura	Interpolazione	Accuratezza
Magnitudine visuale	10 giorni	Lineare	0.1 mag
Puntamento telescopio	1 giorno	Hermite	1 arcosec
Determinazione orbitale	1 ora	Lagrange-9	0.01 arcosec
Avvicinamento ravvicinato	1 minuto	Chebyshev	1 metro

Tabella 11.3: Requisiti effemeride per diverse applicazioni.

11.8.2 Flusso Generazione

```
1 #include <astdyn/ephemeris/EphemerisGenerator.hpp>  
2  
3 // Definisci intervallo temporale  
4 double t_start = 60000.0; // MJD  
5 double t_end = 60365.0; // 1 anno
```

```

6  double dt = 1.0;                      // Spaziatura 1 giorno
7
8 // Imposta propagatore
9 ForceModel forces;
10 forces.enable_planets({"Jupiter", "Saturn", "Mars"});
11 Propagator prop(forces);
12
13 // Stato iniziale da elementi orbitali
14 OrbitalElements elem = load_orbit("203_Pompeja.oe");
15 Vector6d y0 = elem.to_cartesian();
16
17 // Genera effemeride
18 EphemerisGenerator gen(prop);
19 auto ephem = gen.generate(y0, elem.epoch, t_start, t_end,
20                           dt);
21
22 // Salva su file
23 ephem.save("pompeja_ephemeris.txt");
24
25 // Successivamente: interpola a tempo arbitrario
26 Vector6d state_interp = ephem.interpolate(60123.456,
27                                              HERMITE);

```

Listing 11.4: Generazione effemeride

11.8.3 Validazione

Validare sempre le effemeridi:

1. Confronta con effemeridi pubblicate (MPC, JPL Horizons)
2. Controlla conservazione energia (se applicabile)
3. Verifica velocità lisce (nessun salto)
4. Testa errore interpolazione contro propagazione

11.9 Memorizzazione Efficiente

11.9.1 Formati Binari

Per effemeridi grandi, usare binario:

- HDF5: Gerarchico, compresso, auto-descrittivo
- FITS: Standard in astronomia, buon supporto strumenti
- Binario custom: Massima efficienza, richiede documentazione

Dimensioni esempio (1 anno, spaziatura 1 giorno):

- ASCII: 350 KB
- Binario (double): 18 KB
- Binario compresso: 5 KB

11.9.2 Spaziatura Adattativa

Per orbite eccentriche, usare spaziatura variabile:

- Spaziatura fine vicino al perielio (moto veloce)
- Spaziatura larga vicino all'afelio (moto lento)

Spaziatura proporzionale al tasso anomalia vera:

$$\Delta t \propto \frac{r^2}{\sqrt{\mu a(1 - e^2)}} \quad (11.16)$$

Questo mantiene errore posizione costante.

11.10 Riepilogo

Concetti chiave sul calcolo effemeridi:

1. Le **effemeridi** forniscono posizioni/velocità a tempi specificati
2. **Rappresentazioni**: tabulata, polinomiale (Chebyshev), analitica (VSOP87)
3. **Interpolazione**: Hermite per accuratezza, Lagrange per flessibilità

4. **SPICE** è lo standard NASA per effemeridi planetarie/veicoli spaziali
5. La correzione **tempo-luce** contabilizza velocità luce finita
6. L'**aberrazione** corregge per moto osservatore
7. La **spaziatura adattativa** migliora efficienza per orbite eccentriche

Raccomandazioni pratiche:

- Usare DE440/441 per pianeti (via SPICE)
- Usare VSOP87 se SPICE non disponibile o per epoche storiche
- Generare effemeridi custom per asteroidi
- Interpolazione Hermite per accuratezza 1 metro con spaziatura 1 giorno
- Applicare sempre correzioni tempo-luce per lavoro preciso

Il prossimo capitolo inizia la Parte III (Determinazione Orbitale), usando effemeridi per prevedere osservazioni e fissare orbite ai dati.

Parte III

Determinazione Orbitale

Capitolo 12

Osservazioni

12.1 Introduzione

Le **osservazioni** sono i dati fondamentali per la determinazione orbitale. Questo capitolo descrive:

- Tipi di osservazioni (astrometriche, radar, veicoli spaziali)
- Modelli osservativi che relazionano stato a misure
- Formati dati (MPC, radar, tracciamento)
- Correzioni (rifrazione, tempo-luce, aberrazione)
- Coordinate osservatorio e orientamento Terra

Una modellazione osservativa accurata è essenziale per ottenere determinazione orbitale sub-arcosecondo.

12.2 Tipi di Osservazioni

12.2.1 Astrometria Ottica

Le osservazioni più comuni sono posizioni angolari sulla sfera celeste:

$$\text{Osservazione} = (\alpha, \delta, t) \quad (12.1)$$

dove:

- α è l'ascensione retta (0° a 360° o 0h a 24h)

- δ è la declinazione (-90° a $+90^\circ$)
- t è il tempo osservazione (solitamente UTC)

Range di precisione:

- Storico (fotografico): 0.5–2 arcosec
- Astrometria CCD: 0.1–0.5 arcosec
- Missione spaziale Gaia: 0.0001–0.001 arcosec (100 μ as)
- Survey terrestri (Pan-STARRS, ATLAS): 0.05–0.2 arcosec

12.2.2 Osservazioni Radar

Il radar planetario fornisce misure di distanza e Doppler:

$$\text{Distanza: } \rho = |\mathbf{r}_{\text{bersaglio}} - \mathbf{r}_{\text{stazione}}| \quad (12.2)$$

$$\text{Doppler: } \dot{\rho} = \frac{(\mathbf{r}_{\text{bersaglio}} - \mathbf{r}_{\text{stazione}}) \cdot (\mathbf{v}_{\text{bersaglio}} - \mathbf{v}_{\text{stazione}})}{|\mathbf{r}_{\text{bersaglio}} - \mathbf{r}_{\text{stazione}}|} \quad (12.3)$$

Principali strutture radar:

- Arecibo (305 m, 2.38 GHz) – dismesso 2020
- Goldstone DSS-14 (70 m, 8.56 GHz) – operativo
- Green Bank (100 m, solo ricezione)

Precisione:

- Distanza: 10–100 metri (imaging delay-Doppler: <1 m)
- Doppler: 0.1–1 mm/s

Il radar è $1000\times$ più preciso dell'astrometria ottica in distanza ma limitato a oggetti vicini (< 0.3 AU per asteroidi).

12.2.3 Tracciamento Veicoli Spaziali

Missioni spazio profondo tracciate via:

- Doppler bidirezionale (precisione mm/s)
- Misure distanza (livello metro)
- Delta-DOR (posizione angolare via interferometria)
- Navigazione ottica (immagini camera)

12.3 Modello Osservazione Astrometrica

12.3.1 Trasformazione Coordinate

Data posizione oggetto \mathbf{r}_{ogg} in eclittica eliocentrica J2000, calcolare equatoriale topocentrica:

1. Trasforma in baricentrica: $\mathbf{r}_{\text{bary}} = \mathbf{r}_{\text{ogg}} + \mathbf{r}_{\odot, \text{bary}}$
2. Sottrai posizione Terra: $\mathbf{r}_{\text{geo}} = \mathbf{r}_{\text{bary}} - \mathbf{r}_{\text{Terra}}$
3. Sottrai posizione osservatorio: $\mathbf{r}_{\text{topo}} = \mathbf{r}_{\text{geo}} - \mathbf{r}_{\text{oss}}$
4. Ruota in equatoriale: $\mathbf{r}_{\text{eq}} = \mathbf{R}_{\text{ecl} \rightarrow \text{eq}} \mathbf{r}_{\text{topo}}$

12.3.2 Coordinate Sferiche

Da cartesiane topocentriche equatoriali $\mathbf{r}_{\text{eq}} = (x, y, z)$:

$$\alpha = \arctan 2(y, x) \quad (12.4)$$

$$\delta = \arcsin \left(\frac{z}{\sqrt{x^2 + y^2 + z^2}} \right) \quad (12.5)$$

Gestire correttamente il quadrante con atan2.

12.3.3 Correzione Tempo-Luce

Il tempo osservazione t_{oss} è quando i fotoni arrivano alla Terra. L'oggetto era alla posizione di emissione a:

$$t_{\text{emis}} = t_{\text{oss}} - \frac{\rho}{c} \quad (12.6)$$

dove ρ è la distanza geocentrica.

Iterare per trovare t_{emis} :

```

1 double tau = 0.0; // Stima iniziale
2 for (int iter = 0; iter < 5; ++iter) {
3     Vector3d r_obj = propagate(y0, t0, t_obs - tau);
4     Vector3d r_earth = ephemeris.get_position("Earth",
5         t_obs);
6     double rho = (r_obj - r_earth).norm();
7     double tau_new = rho / C_AU_PER_DAY;
8     if (std::abs(tau_new - tau) < 1e-10) break;
9     tau = tau_new;
}
```

Listing 12.1: Iterazione tempo-luce

Correzione tipica: 4–30 minuti per asteroidi.

12.3.4 Aberrazione Stellare

Il moto orbitale terrestre causa spostamento apparente:

$$\mathbf{r}_{\text{aberrato}} = \mathbf{r}_{\text{geometrico}} + \frac{\rho}{c} \mathbf{v}_{\text{Terra}} \quad (12.7)$$

dove $\mathbf{v}_{\text{Terra}}$ è la velocità terrestre.

Effetto massimo: ± 20.5 arcosec (aberrazione annua).

12.3.5 Rifrazione Atmosferica

La luce si piega attraversando l'atmosfera. La correzione dipende dall'angolo zenitale z :

$$\Delta z \approx 58.2'' \tan z - 0.067'' \tan^3 z \quad (12.8)$$

Allo zenit ($z = 0$): nessuna rifrazione. All'orizzonte ($z = 90^\circ$): ~ 34 arcmin (diametro solare!).

Per lavoro preciso, usare modello dipendente da lunghezza d'onda:

$$n - 1 = 77.6 \times 10^{-6} \frac{P}{T} \left(1 + 7.52 \times 10^{-3} \lambda^{-2} \right) \quad (12.9)$$

dove P è pressione (mbar), T temperatura (K), λ lunghezza d'onda (μm).

L'astrometria moderna corregge a "sopra l'atmosfera" tramite:

- Fit stelle catalogo nel campo
- Misura rifrazione locale empiricamente
- Applicazione modelli sito-specifici

12.4 Coordinate Osservatorio

12.4.1 ITRF e Codici Osservatorio

L'International Terrestrial Reference Frame (ITRF) fornisce coordinate precise per osservatori.

Codici osservatorio Minor Planet Center (MPC):

- 500: Geocentro (per osservazioni spaziali)
- 568: Mauna Kea (Hawaii)
- 703: Catalina Sky Survey (Arizona)
- F51: Pan-STARRS 1 (Hawaii)
- G96: Mt. Lemmon Survey (Arizona)

Esempio entry per osservatorio 703:

703 Catalina 4.215500 0.759260 0.648764 -31.67

Formato: codice, nome, $\rho \cos \phi'$, $\rho \sin \phi'$, longitudine (gradi), altitudine (m).

12.4.2 Posizione Osservatorio Geocentrica

Convertire coordinate geodetiche (h, λ, ϕ) in cartesiane geocentriche:

$$\mathbf{r}_{\text{oss}} = \begin{bmatrix} (N + h) \cos \phi \cos \lambda \\ (N + h) \cos \phi \sin \lambda \\ (N(1 - e^2) + h) \sin \phi \end{bmatrix} \quad (12.10)$$

dove:

$$N = \frac{a}{\sqrt{1 - e^2 \sin^2 \phi}} \quad (12.11)$$

e $a = 6378.137$ km (raggio equatoriale WGS84), $e = 0.08181919$ (eccentricità).

12.4.3 Rotazione a Sistema Inerziale

La posizione osservatorio ruota con la Terra. La trasformazione coinvolge:

1. Moto polare (x_p, y_p)
2. Correzione UT1-UTC (angolo rotazione Terra)
3. Precessione-nutazione (IAU 2006/2000A)
4. Bias sistema (ICRS a J2000)

Rotazione semplificata:

$$\mathbf{r}_{\text{inerziale}} = \mathbf{R}_3(\text{GAST}) \mathbf{r}_{\text{ITRF}} \quad (12.12)$$

dove GAST è il Greenwich Apparent Sidereal Time.

12.5 Parametri Orientamento Terra

12.5.1 Moto Polare

L'asse rotazione terrestre si muove rispetto alla crosta (oscillazione Chandler, moto annuale):

$$\mathbf{R}_{\text{polare}} = \mathbf{R}_2(-x_p) \mathbf{R}_1(-y_p) \quad (12.13)$$

Aampiezza: ~ 0.3 arcosec (~ 10 metri in superficie).

Dati da IERS: bollettino `finals2000A.all`.

12.5.2 UT1-UTC

Universal Time (UT1) traccia la rotazione effettiva terrestre. Il tempo atomico (UTC) è uniforme.

$$\text{UT1} = \text{UTC} + (\text{UT1}-\text{UTC}) \quad (12.14)$$

$|\text{UT1}-\text{UTC}| < 0.9$ secondi (secondi intercalari aggiunti quando necessario).

Previsione: disponibile da IERS con accuratezza ~ 10 ms per 1 anno avanti.

12.5.3 Precessione e Nutazione

L'asse rotazione terrestre precede (periodo 26.000 anni) e nuta (periodo principale 18.6 anni).

Precessione IAU 2006 + nutazione IAU 2000A = modello alta precisione.

Semplificato per lavoro asteroidi: usare polo medio (J2000) e ignorare nutazione (effetto ~ 15 arcosec).

12.6 Formato Osservazione MPC

12.6.1 Formato 80 Colonne

Formato standard per astrometria ottica:

K17S00S	C2017	06	01.41667	18	26	54.13	-23	47	08.4	21.1	V	F51
---------	-------	----	----------	----	----	-------	-----	----	------	------	---	-----

Campi:

- Colonne 1-5: Designazione temporanea o numero
- Colonna 12: Asterisco scoperta (*)
- Colonna 13: Nota (es. fotometria)
- Colonna 14: Riferimento pubblicazione
- Colonne 15-32: Data osservazione (YYYY MM DD.ddddd)
- Colonne 33-44: Ascensione retta (HH MM SS.sss)
- Colonne 45-56: Declinazione (sDD MM SS.ss)

- Colonne 66-70: Magnitudine
- Colonna 71: Banda mag (V, R, I, ecc.)
- Colonne 78-80: Codice osservatorio

12.6.2 Formato ADES

Astrometry Data Exchange Standard (formato XML/JSON moderno):

```
1 <obsBlock>
2   <obsContext>
3     <observatory>
4       <mpcCode>F51</mpcCode>
5     </observatory>
6   </obsContext>
7   <obsData>
8     <optical>
9       <trkSub>K17S00S</trkSub>
10      <obsTime>2017-06-01T10:00:00.000Z</obsTime>
11      <ra>276.72554</ra>
12      <dec>-23.78567</dec>
13      <mag>21.1</mag>
14      <band>V</band>
15      <rmsRA>0.1</rmsRA>
16      <rmsDec>0.1</rmsDec>
17    </optical>
18  </obsData>
19 </obsBlock>
```

Listing 12.2: Esempio ADES XML

Vantaggi rispetto a 80 colonne:

- Incertezze esplicite
- Metadata (telescopio, rilevatore, catalogo)
- Nessuna limitazione larghezza fissa
- Standard internazionale

12.7 Pesi Osservazioni

12.7.1 Schemi di Pesatura

Non tutte le osservazioni sono ugualmente affidabili. Pesare per incertezza stimata:

$$w_i = \frac{1}{\sigma_i^2} \quad (12.15)$$

Fonti incertezza:

- Errore misura (fit stella, centroide)
- Errori catalogo (Gaia DR3: 0.02–0.05 arcosec)
- Errori timing (± 1 secondo $\rightarrow 0.01$ arcosec per oggetti lenti)
- Effetti atmosferici (seeing, rifrazione)
- Perdite scia (esposizioni lunghe)

12.7.2 Pesatura Empirica

Per osservazioni MPC senza incertezze formali:

Tipo Osservatorio	$\sigma_\alpha \cos \delta$	σ_δ
Professionale (Pan-STARRS, CSS)	0.1 arcosec	0.1 arcosec
CCD amatoriale	0.5 arcosec	0.5 arcosec
Fotografico storico	1.0 arcosec	1.0 arcosec
Distanza radar	10 m	–
Doppler radar	–	1 mm/s

Tabella 12.1: Incertezze osservative tipiche.

12.7.3 Riduzione Peso Outlier

Dopo fit iniziale, identificare outlier ($\text{residuo} > 3\sigma$) e ridurre peso:

$$w_{\text{nuovo}} = w_{\text{vecchio}} \times \exp \left(-\frac{r^2}{2\sigma^2} \right) \quad (12.16)$$

dove r è il residuo. Questo è "minimi quadrati robusti" o "pesatura Huber."

12.8 Parziali Osservazioni

12.8.1 Definizione

Per determinazione orbitale, necessitiamo:

$$\frac{\partial(\alpha, \delta)}{\partial \mathbf{y}(t_0)} \quad (12.17)$$

Questo relaziona come lo stato iniziale influenza l'osservazione prevista.

12.8.2 Regola Catena

Usare regola catena con matrice transizione stato:

$$\frac{\partial(\alpha, \delta)}{\partial \mathbf{y}(t_0)} = \frac{\partial(\alpha, \delta)}{\partial \mathbf{r}(t_{\text{oss}})} \frac{\partial \mathbf{r}(t_{\text{oss}})}{\partial \mathbf{y}(t_{\text{oss}})} \frac{\partial \mathbf{y}(t_{\text{oss}})}{\partial \mathbf{y}(t_0)} \quad (12.18)$$

L'ultimo fattore è la STM $\Phi(t_{\text{oss}}, t_0)$.

12.8.3 Parziali Geometriche

Per posizione topocentrica $\mathbf{r} = (x, y, z)$ in sistema equatoriale:

$$\rho = \sqrt{x^2 + y^2 + z^2} \quad (12.19)$$

$$\frac{\partial \alpha}{\partial x} = -\frac{y}{x^2 + y^2}, \quad \frac{\partial \alpha}{\partial y} = \frac{x}{x^2 + y^2}, \quad \frac{\partial \alpha}{\partial z} = 0 \quad (12.20)$$

$$\frac{\partial \delta}{\partial x} = -\frac{xz}{\rho^2 \sqrt{x^2 + y^2}}, \quad \frac{\partial \delta}{\partial y} = -\frac{yz}{\rho^2 \sqrt{x^2 + y^2}}, \quad \frac{\partial \delta}{\partial z} = \frac{\sqrt{x^2 + y^2}}{\rho^2} \quad (12.21)$$

12.8.4 Implementazione

```
1 Matrix<2,6> compute_partials_radec(
2     const Vector6d& state,
3     const Matrix6d& stm,
4     const Vector3d& obs_pos)
5 {
6     Vector3d r = state.head<3>() - obs_pos;
```

```

7   double x = r(0), y = r(1), z = r(2);
8   double rho = r.norm();
9   double rho_xy = std::sqrt(x*x + y*y);
10
11  // Parziali rispetto a posizione
12  Matrix<2,3> dobs_dr;
13  dobs_dr(0,0) = -y / (x*x + y*y); // d(RA)/dx
14  dobs_dr(0,1) = x / (x*x + y*y); // d(RA)/dy
15  dobs_dr(0,2) = 0.0; // d(RA)/dz
16
17  dobs_dr(1,0) = -x*z / (rho*rho*rho_xy); // d(Dec)/dx
18  dobs_dr(1,1) = -y*z / (rho*rho*rho_xy); // d(Dec)/dy
19  dobs_dr(1,2) = rho_xy / (rho*rho); // d(Dec)/dz
20
21  // Concatena con STM
22  Matrix<2,6> partials = dobs_dr * stm.block<3,6>(0,0);
23
24  return partials;
25 }
```

Listing 12.3: Calcolo parziali osservazione

12.9 Qualità Dati

12.9.1 Accuratezza Timing

Il tempo osservazione deve essere UTC a ± 1 secondo per asteroidi (± 0.01 sec per oggetti veloci).

Problemi comuni:

- Deriva clock (ricevitori GPS essenziali)
- Tempo metà-esposizione vs inizio/fine
- Errori fuso orario (usare sempre UTC!)
- Secondi intercalari

12.9.2 Catalogo Astrometrico

Osservazioni moderne riferite a:

- Gaia DR3 (2022): 0.02–0.05 arcosec, ~ 1.8 miliardi stelle
- UCAC4: 0.02–0.1 arcosec, 113 milioni stelle
- 2MASS: 0.08 arcosec (infrarosso), 471 milioni oggetti

Osservazioni più vecchie (pre-Gaia) possono avere errori sistematici da catalogo:

- USNO-A: ~ 0.25 arcosec sistematico
- GSC: ~ 0.3 arcosec sistematico

Usare de-biasing specifico catalogo quando si mescolano osservazioni.

12.9.3 Sistematici Sito-Specifici

Alcuni osservatori hanno problemi noti:

- Timing scarso (> 10 sec errori)
- Coordinate errate (latitudine/longitudine sbagliate)
- Errori scala (scala piatto sbagliata)
- Bias dipendente da magnitudine (bleeding carica)

MPC mantiene flag qualità, ma utente deve validare dati.

12.10 Esempio Pratico

12.10.1 Caricamento Osservazioni MPC

```
1 #include <astdyn/observations/MPCObservation.hpp>
2
3 std::vector<Observation> load_mpc_file(const std::string&
4   filename) {
5     std::vector<Observation> observations;
6     std::ifstream file(filename);
```

```

6     std::string line;
7
8     while (std::getline(file, line)) {
9         if (line.length() < 80) continue;
10
11         MPCObservation obs;
12         if (obs.parse(line)) {
13             observations.push_back(obs);
14         }
15     }
16
17     std::cout << "Caricate " << observations.size() << "
18         osservazioni\n";
19     return observations;
}

```

Listing 12.4: Parsing osservazioni MPC

12.10.2 Calcolo Osservazioni Previste

```

1 Vector2d predict_observation(
2     const Vector6d& state,
3     double epoch,
4     const std::string& obs_code,
5     const EphemerisInterface& ephemeris)
6 {
7     // Ottieni posizione Terra
8     Vector3d earth_pos = ephemeris.get_position("Earth",
9         epoch);
10
11    // Ottieni posizione osservatorio (ITRF -> inerziale)
12    Vector3d obs_pos_geo = observatory_db.get_geocentric(
13        obs_code);
14    Matrix3d R_itrf_to_j2000 = earth_rotation(epoch);
15    Vector3d obs_pos = earth_pos + R_itrf_to_j2000 *
        obs_pos_geo;
16
17    // Posizione topocentrica

```

```
16     Vector3d r_topo = state.head<3>() - obs_pos;  
17  
18     // Eclittica a equatoriale  
19     Vector3d r_eq = R_ecl_to_eq * r_topo;  
20  
21     // Calcola RA/Dec  
22     double alpha = std::atan2(r_eq(1), r_eq(0));  
23     double delta = std::asin(r_eq(2) / r_eq.norm());  
24  
25     if (alpha < 0) alpha += 2*M_PI;  
26  
27     return Vector2d(alpha, delta);  
28 }
```

Listing 12.5: Previsione osservazioni

12.11 Riepilogo

Concetti chiave sulle osservazioni:

1. L'**astrometria ottica** fornisce RA/Dec con precisione 0.1–0.5 arcosec
2. Il **radar** dà distanza/Doppler con precisione metro/mm-per-sec
3. La correzione **tempo-luce** è essenziale (4–30 minuti per asteroidi)
4. L'**aberrazione** causa spostamento ± 20 arcosec
5. La **rifrazione** influenza osservazioni a bassa elevazione
6. La **posizione osservatorio** deve essere in sistema inerziale
7. Il **formato MPC** è standard, ADES è moderno
8. La **pesatura** per incertezza migliora qualità fit
9. Le **parziali** $\partial(\alpha, \delta)/\partial y$ abilitano fit orbitale

Raccomandazioni pratiche:

- Applicare sempre correzioni tempo-luce e aberrazione

- Usare catalogo Gaia DR3 per osservazioni moderne
- Validare timing (UTC, secondi intercalari)
- Controllare coordinate osservatorio
- Pesare per incertezza stimata
- Identificare e ridurre peso outlier

Il prossimo capitolo copre la determinazione orbitale iniziale da poche osservazioni, seguito dalla correzione differenziale per raffinare orbite usando tutti i dati disponibili.

Capitolo 13

Determinazione dell'Orbita Iniziale

13.1 Introduzione

La **determinazione dell'orbita iniziale** (Initial Orbit Determination, IOD) calcola un'orbita approssimata da un piccolo numero di osservazioni. Fornisce:

- Punto di partenza per la correzione differenziale
- Collegamento di osservazioni attraverso opposizioni
- Previsioni di recupero per oggetti persi
- Valutazioni preliminari di impatto

I metodi classici usano 3 osservazioni (Gauss, Laplace) o 2 osservazioni + vincoli.

13.2 Il Problema IOD

13.2.1 Osservazioni Solo Angolari

Dato: Tre osservazioni $(\alpha_i, \delta_i, t_i), i = 1, 2, 3$.

Trovare: Sei elementi orbitali o stato cartesiano $\mathbf{y} = [\mathbf{r}, \mathbf{v}]$.

Sfida: Abbiamo 6 incognite ma solo 6 vincoli ($2 \text{ angoli} \times 3 \text{ tempi}$). Il problema è esattamente determinato ma altamente non lineare.

13.2.2 Linea di Vista

Ogni osservazione definisce un vettore unitario:

$$\hat{\rho}_i = \begin{bmatrix} \cos \delta_i \cos \alpha_i \\ \cos \delta_i \sin \alpha_i \\ \sin \delta_i \end{bmatrix} \quad (13.1)$$

L'oggetto si trova da qualche parte lungo questa linea: $\mathbf{r}_i = \mathbf{R}_i + \rho_i \hat{\rho}_i$ dove \mathbf{R}_i è la posizione dell'osservatorio e ρ_i è la distanza topocentrica incognita.

13.3 Metodo di Gauss

13.3.1 Contesto Storico

Sviluppato da Carl Friedrich Gauss (1809) per recuperare Cerere dopo il suo passaggio dietro il Sole. Ancora ampiamente utilizzato oggi.

13.3.2 Idea di Base

Usare 3 osservazioni per:

1. Stimare la distanza ρ_2 all'osservazione centrale
2. Calcolare la posizione \mathbf{r}_2
3. Usare i coefficienti di Lagrange per ottenere la velocità \mathbf{v}_2

13.3.3 Coefficienti di Lagrange

Per il moto a due corpi, le posizioni ai tempi t_1, t_2, t_3 sono legate da:

$$\mathbf{r}_1 = f_1 \mathbf{r}_2 + g_1 \mathbf{v}_2 \quad (13.2)$$

$$\mathbf{r}_3 = f_3 \mathbf{r}_2 + g_3 \mathbf{v}_2 \quad (13.3)$$

dove f e g sono coefficienti di Lagrange che dipendono dagli intervalli temporali $\tau_1 = t_1 - t_2$ e $\tau_3 = t_3 - t_2$.

Sviluppo in serie:

$$f = 1 - \frac{\mu}{2r^3}\tau^2 + \frac{\mu}{2r^3} \frac{\mathbf{r} \cdot \mathbf{v}}{r^2} \tau^3 + O(\tau^4) \quad (13.4)$$

$$g = \tau - \frac{\mu}{6r^3}\tau^3 + O(\tau^4) \quad (13.5)$$

13.3.4 Equazione Scalare di Lagrange

I tre vettori posizione giacciono nel piano orbitale. Usando la complanarità:

$$\mathbf{r}_1 \cdot (\mathbf{r}_2 \times \mathbf{r}_3) = 0 \quad (13.6)$$

Questo fornisce un'equazione scalare per ρ_2 (il "polinomio di ottavo grado" dopo manipolazione).

13.3.5 Algoritmo

Input: Tre osservazioni $(\alpha_i, \delta_i, t_i, \mathbf{R}_i)$.

Passi:

1. Calcolare i vettori linea di vista $\hat{\rho}_i$
2. Stima iniziale: $\rho_2 = |\mathbf{R}_2|$ (distanza Terra-Sole)
3. Iterare:
 - (a) Calcolare $\mathbf{r}_2 = \mathbf{R}_2 + \rho_2 \hat{\rho}_2$
 - (b) Calcolare $r_2 = |\mathbf{r}_2|$
 - (c) Stimare i coefficienti f, g
 - (d) Risolvere per \mathbf{v}_2 da $\mathbf{r}_1, \mathbf{r}_3$
 - (e) Raffinare ρ_2 usando l'equazione scalare di Lagrange
 - (f) Verificare convergenza: $|\Delta\rho_2| < 10^{-6}$ AU
4. Restituire lo stato $(\mathbf{r}_2, \mathbf{v}_2)$ all'epoca t_2

Convergenza: Tipicamente 5-10 iterazioni per oggetti ben osservati.

13.4 Implementazione

```
1 Vector6d gauss_iod(
2     const std::array<Observation, 3>& obs,
3     const EphemerisInterface& ephemeris)
4 {
5     // Estrarre tempi e vettori linea di vista
6     double t1 = obs[0].epoch;
7     double t2 = obs[1].epoch;
8     double t3 = obs[2].epoch;
9
10    Vector3d rho_hat1 = obs[0].line_of_sight();
11    Vector3d rho_hat2 = obs[1].line_of_sight();
12    Vector3d rho_hat3 = obs[2].line_of_sight();
13
14    // Posizioni osservatorio
15    Vector3d R1 = ephemeris.get_observer_position(obs[0]);
16    Vector3d R2 = ephemeris.get_observer_position(obs[1]);
17    Vector3d R3 = ephemeris.get_observer_position(obs[2]);
18
19    // Intervalli temporali
20    double tau1 = t1 - t2;
21    double tau3 = t3 - t2;
22
23    // Stima iniziale per la distanza centrale
24    double rho2 = R2.norm();
25
26    // Raffinamento iterativo
27    for (int iter = 0; iter < 20; ++iter) {
28        Vector3d r2 = R2 + rho2 * rho_hat2;
29        double r2_mag = r2.norm();
30
31        // Calcolare serie f, g (al 3 grado ordine)
32        double f1 = 1.0 - 0.5 * MU_SUN * tau1*tau1 / (
33            r2_mag*r2_mag*r2_mag);
34        double f3 = 1.0 - 0.5 * MU_SUN * tau3*tau3 / (
35            r2_mag*r2_mag*r2_mag);
```

```

34     double g1 = tau1 - MU_SUN * tau1*tau1*tau1 / (6.0 *
35         r2_mag*r2_mag*r2_mag);
36     double g3 = tau3 - MU_SUN * tau3*tau3*tau3 / (6.0 *
37         r2_mag*r2_mag*r2_mag);
38
39     // Risolvere per la velocita' a t2
40     Vector3d v2 = (f3 * (R1 + rho_hat1) - f1 * (R3 +
41         rho_hat3)) / (f1*g3 - f3*g1);
42
43     // Migliorare rho2 usando l'equazione scalare di
44     // Lagrange
45     // (semplicato: usare stime r1, r3)
46     Vector3d r1 = r2 * f1 + v2 * g1;
47     Vector3d r3 = r2 * f3 + v3 * g3;
48
49     double rho1_new = (r1 - R1).dot(rho_hat1);
50     double rho3_new = (r3 - R3).dot(rho_hat3);
51     double rho2_new = (r2 - R2).dot(rho_hat2);
52
53     if (std::abs(rho2_new - rho2) < 1e-6) {
54         // Convergenza raggiunta
55         return Vector6d(r2, v2);
56     }
57
58     rho2 = rho2_new;
59
60     throw std::runtime_error("IOD di Gauss non convergente"
61 );
62 }
```

Listing 13.1: Implementazione del metodo di Gauss

13.5 Problema dell'Arco Troppo Corto

13.5.1 Sfida

Per archi osservativi brevi (ore o giorni), molte orbite si adattano ugualmente bene. L'orbita è mal vincolata in:

- Semiasse maggiore a (degenera con eccentricità)
- Eccentricità e
- Argomento del pericentro ω

Esempio: NEA osservato per 3 ore. Potrebbe essere:

- $a = 1.2 \text{ AU}, e = 0.1$ (Apollo)
- $a = 2.5 \text{ AU}, e = 0.6$ (Amor)
- $a = 0.8 \text{ AU}, e = 0.3$ (Aten)

Tutte producono RA/Dec simili su arco breve!

13.5.2 Vincoli Aggiuntivi

Per risolvere la degenerazione:

1. **Moto apparente:** $d\alpha/dt, d\delta/dt$ vincola la distanza
2. **Luminosità:** H, G funzione di fase fornisce stima distanza
3. **Priori statistici:** La maggior parte dei NEA ha $0.8 < a < 2 \text{ AU}$
4. **Osservazioni aggiuntive:** Anche +1 giorno aiuta enormemente

13.6 Metodo di Laplace

13.6.1 Approccio Alternativo

Usare la velocità angolare $\dot{\alpha}, \dot{\delta}$ oltre agli angoli. Richiede temporizzazione ad alta precisione o multiple osservazioni ravvicinate.

Vantaggio: Può funzionare con 2 osservazioni (più rate).

Svantaggio: Sensibile agli errori di misura nelle rate.

13.6.2 Equazioni

Da $\mathbf{r} = \mathbf{R} + \rho\hat{\rho}$, derivare due volte:

$$\ddot{\mathbf{r}} = -\frac{\mu}{r^3}\mathbf{r} \quad (13.7)$$

Questo fornisce 3 equazioni in 3 incognite $(\rho, \dot{\rho}, \ddot{\rho})$ ad un'epoca.

13.7 Metodi Moderni

13.7.1 Regione Ammissibile

Per archi molto corti, risolvere per tutte le orbite ammissibili che soddisfano:

- Osservazioni
- Vincoli fisici ($e < 1$ per orbite legate)
- Luminosità (stima della distanza)

Produce una regione nello spazio degli elementi orbitali, non una singola soluzione.

13.7.2 Minimi Quadrati Vincolati

Minimizzare:

$$\chi^2 = \sum_i w_i (\mathbf{o}_i - \mathbf{c}_i)^2 + \lambda P(\mathbf{e}) \quad (13.8)$$

dove $P(\mathbf{e})$ è un priore sugli elementi (es., preferire $e < 0.3$).

13.8 Valutazione della Qualità

13.8.1 Incertezza Orbitale

Da 3 osservazioni, l'incertezza è grande:

- Posizione all'epoca: ~ 0.001 AU (150.000 km)
- Velocità: ~ 0.01 AU/giorno (17 km/s)

- Semiasse maggiore: ± 0.5 AU

L'incertezza di propagazione cresce rapidamente! Dopo 1 mese, errore di posizione > 1 AU.

13.8.2 Validazione

Verificare la qualità dell'orbita:

1. Residui: Dovrebbero essere < 5 arcosecondi per un buon fit
2. Energia: $E < 0$ per orbita legata
3. Pericentro: $q > 0.1$ AU (all'interno, l'orbita collide con il Sole)
4. Eccentricità: $0 \leq e < 1$ per orbita ellittica

13.9 Esempio: Asteroide Appena Scoperto

```
1 // Tre osservazioni da MPC
2 std::vector<Observation> obs = {
3     {"2024-01-15T03:15:00Z", 185.234, +12.567, "F51"}, 
4     {"2024-01-15T04:30:00Z", 185.189, +12.592, "F51"}, 
5     {"2024-01-15T05:45:00Z", 185.144, +12.617, "F51"} 
6 };
7
8 // Caricare effemeridi planetarie
9 SpiceInterface spice;
10 spice.load_kernel("de440.bsp");
11
12 // Eseguire IOD di Gauss
13 try {
14     Vector6d state = gauss_iod(obs, spice);
15     double epoch = obs[1].epoch;
16
17     // Convertire in elementi orbitali
18     OrbitalElements elements = OrbitalElements::
19         from_cartesian(state, epoch);
20
21     std::cout << "Determinazione orbita iniziale:\n";
```

```

21     std::cout << "a = " << elements.a << " AU\n";
22     std::cout << "e = " << elements.e << "\n";
23     std::cout << "i = " << elements.i * RAD_TO_DEG << " deg
24         \n";
25     std::cout << "Omega = " << elements.Omega * RAD_TO_DEG
26         << " deg\n";
27     std::cout << "omega = " << elements.omega * RAD_TO_DEG
28         << " deg\n";
29     std::cout << "M = " << elements.M * RAD_TO_DEG << " deg
30         \n";
31
32     // Calcolare residui
33     for (const auto& ob : obs) {
34         Vector2d predicted = predict_observation(state, ob.
35             epoch, ob.obs_code, spice);
36         double dRA = (predicted(0) - ob.ra) * cos(ob.dec) *
37             RAD_TO_ARCSEC;
38         double dDec = (predicted(1) - ob.dec) *
39             RAD_TO_ARCSEC;
40         std::cout << "Residuo: " << dRA << ", " << dDec <<
41             " arcosec\n";
42     }
43
44 } catch (const std::exception& e) {
45     std::cerr << "IOD fallito: " << e.what() << "\n";
46 }
```

Listing 13.2: IOD da osservazioni di scoperta

13.10 Sommario

Punti chiave sulla determinazione dell'orbita iniziale:

1. Il **metodo di Gauss** usa 3 osservazioni per determinare l'orbita
2. I **coefficienti di Lagrange** relazionano le posizioni a tempi diversi
3. La **soluzione iterativa** converge tipicamente in 5-10 iterazioni

4. Gli **archi brevi** portano a orbite mal vincolate
5. **Vincoli aggiuntivi** (luminosità, priori) aiutano
6. Il **metodo di Laplace** usa rate angolari oltre agli angoli
7. I **metodi moderni** calcolano regioni ammissibili
8. La **validazione** controlla energia, eccentricità, residui

L'orbita iniziale viene raffinata usando la correzione differenziale (capitolo successivo) con tutte le osservazioni disponibili.

Capitolo 14

Correzione Differenziale

14.1 Introduzione

La **correzione differenziale** (Differential Correction, DC) è il raffinamento iterativo ai minimi quadrati di un'orbita usando tutte le osservazioni disponibili. È la pietra angolare della determinazione orbitale.

Input: Orbita iniziale + osservazioni

Output: Orbita migliorata + matrice di covarianza + residui

Metodo: Minimi quadrati pesati minimizzando i residui O-C (osservato meno calcolato).

14.2 Il Problema dei Minimi Quadrati

14.2.1 Equazione di Osservazione

Per l'osservazione i :

$$\mathbf{o}_i = \mathbf{h}(\mathbf{y}_0, t_i) + \boldsymbol{\epsilon}_i \quad (14.1)$$

dove:

- \mathbf{o}_i : Valore osservato (es., RA, Dec)
- \mathbf{h} : Modello di osservazione (trasformazione coordinate)
- \mathbf{y}_0 : Stato all'epoca t_0
- $\boldsymbol{\epsilon}_i \sim \mathcal{N}(0, \mathbf{W}_i^{-1})$: Errore di misura

14.2.2 Linearizzazione

Linearizzare attorno alla stima corrente $\mathbf{y}_0^{(k)}$:

$$\mathbf{o}_i - \mathbf{c}_i = \mathbf{H}_i \Delta \mathbf{y}_0 + \boldsymbol{\epsilon}_i \quad (14.2)$$

dove:

- $\mathbf{c}_i = \mathbf{h}(\mathbf{y}_0^{(k)}, t_i)$: Valore calcolato
- $\mathbf{H}_i = \frac{\partial \mathbf{h}}{\partial \mathbf{y}_0}$: Matrice di disegno (derivate parziali osservazione)
- $\Delta \mathbf{y}_0 = \mathbf{y}_0 - \mathbf{y}_0^{(k)}$: Correzione allo stato

14.2.3 Equazioni Normali

Minimizzare la somma pesata dei quadrati dei residui:

$$\chi^2 = \sum_{i=1}^m (\mathbf{o}_i - \mathbf{c}_i - \mathbf{H}_i \Delta \mathbf{y}_0)^T \mathbf{W}_i (\mathbf{o}_i - \mathbf{c}_i - \mathbf{H}_i \Delta \mathbf{y}_0) \quad (14.3)$$

Soluzione:

$$(\mathbf{H}^T \mathbf{W} \mathbf{H}) \Delta \mathbf{y}_0 = \mathbf{H}^T \mathbf{W} (\mathbf{o} - \mathbf{c}) \quad (14.4)$$

Definire:

$$\mathbf{N} = \mathbf{H}^T \mathbf{W} \mathbf{H} \quad (\text{matrice normale}) \quad (14.5)$$

$$\mathbf{b} = \mathbf{H}^T \mathbf{W} (\mathbf{o} - \mathbf{c}) \quad (\text{termine noto}) \quad (14.6)$$

Soluzione: $\mathbf{N} \Delta \mathbf{y}_0 = \mathbf{b}$

Covarianza: $\mathbf{C} = \mathbf{N}^{-1}$

14.3 Calcolo delle Derivate Parziali

14.3.1 Regola della Catena con STM

Per osservazioni RA/Dec al tempo t_i :

$$\mathbf{H}_i = \frac{\partial(\alpha, \delta)}{\partial \mathbf{y}_0} = \frac{\partial(\alpha, \delta)}{\partial \mathbf{y}(t_i)} \frac{\partial \mathbf{y}(t_i)}{\partial \mathbf{y}_0} \quad (14.7)$$

dove $\Phi(t_i, t_0) = \frac{\partial \mathbf{y}(t_i)}{\partial \mathbf{y}_0}$ è la matrice di transizione di stato (Capitolo 10).

14.3.2 Derivate Geometriche

Dalla posizione topocentrica $\rho = \mathbf{r} - \mathbf{R}$:

$$\alpha = \arctan 2(\rho_y, \rho_x) \quad (14.8)$$

$$\delta = \arcsin(\rho_z / \rho) \quad (14.9)$$

Derivate:

$$\frac{\partial \alpha}{\partial \rho_x} = -\frac{\rho_y}{\rho_x^2 + \rho_y^2} \quad (14.10)$$

$$\frac{\partial \alpha}{\partial \rho_y} = \frac{\rho_x}{\rho_x^2 + \rho_y^2} \quad (14.11)$$

$$\frac{\partial \alpha}{\partial \rho_z} = 0 \quad (14.12)$$

$$\frac{\partial \delta}{\partial \rho_x} = -\frac{\rho_x \rho_z}{\rho^2 \sqrt{\rho_x^2 + \rho_y^2}} \quad (14.13)$$

$$\frac{\partial \delta}{\partial \rho_y} = -\frac{\rho_y \rho_z}{\rho^2 \sqrt{\rho_x^2 + \rho_y^2}} \quad (14.14)$$

$$\frac{\partial \delta}{\partial \rho_z} = \frac{\sqrt{\rho_x^2 + \rho_y^2}}{\rho^2} \quad (14.15)$$

14.3.3 Derivate Complete

Combinare le derivate geometriche con Φ :

$$\mathbf{H}_i = \begin{bmatrix} \frac{\partial \alpha}{\partial x} & \frac{\partial \alpha}{\partial y} & \frac{\partial \alpha}{\partial z} & 0 & 0 & 0 \\ \frac{\partial \delta}{\partial x} & \frac{\partial \delta}{\partial y} & \frac{\partial \delta}{\partial z} & 0 & 0 & 0 \end{bmatrix} \Phi(t_i, t_0) \quad (14.16)$$

Nota: RA/Dec dipendono solo dalla posizione, non dalla velocità, nelle derivate geometriche. La velocità influenza le osservazioni attraverso la propagazione (Φ).

14.4 Algoritmo

Input: Orbita iniziale $\mathbf{y}_0^{(0)}$, osservazioni $\{(\mathbf{o}_i, t_i, \mathbf{W}_i)\}$

Iterare:

1. Per ogni osservazione i :
 - (a) Propagare a t_i con STM: $[\mathbf{y}(t_i), \Phi(t_i, t_0)]$
 - (b) Calcolare predizione $\mathbf{c}_i = \mathbf{h}(\mathbf{y}(t_i))$
 - (c) Calcolare derivate geometriche
 - (d) Calcolare derivate complete \mathbf{H}_i usando STM
2. Formare matrice normale: $\mathbf{N} = \sum_i \mathbf{H}_i^T \mathbf{W}_i \mathbf{H}_i$
3. Formare termine noto: $\mathbf{b} = \sum_i \mathbf{H}_i^T \mathbf{W}_i (\mathbf{o}_i - \mathbf{c}_i)$
4. Risolvere: $\mathbf{N} \Delta \mathbf{y}_0 = \mathbf{b}$
5. Aggiornare: $\mathbf{y}_0^{(k+1)} = \mathbf{y}_0^{(k)} + \Delta \mathbf{y}_0$
6. Calcolare RMS: $\text{RMS} = \sqrt{\frac{1}{m-n} \sum_i w_i r_i^2}$ dove $r_i = \mathbf{o}_i - \mathbf{c}_i$
7. Verificare convergenza: $|\Delta \mathbf{y}_0| < \epsilon$ e $|\Delta \text{RMS}| < \epsilon_{\text{RMS}}$

Output: Stato converso \mathbf{y}_0^* , covarianza $\mathbf{C} = \mathbf{N}^{-1}$, residui

14.5 Criteri di Convergenza

14.5.1 Correzione allo Stato

$$||\Delta \mathbf{y}_0|| < 10^{-8} \text{ AU, AU/giorno} \quad (14.17)$$

14.5.2 Variazione RMS

$$\frac{|\text{RMS}^{(k+1)} - \text{RMS}^{(k)}|}{\text{RMS}^{(k)}} < 10^{-6} \quad (14.18)$$

14.5.3 Iterazioni Massime

Converge tipicamente in 3-10 iterazioni. Se non converge dopo 20 iterazioni, sospettare:

- Orbita iniziale scadente
- Osservazioni errate (outlier)
- Modello inadeguato (perturbazioni mancanti)

14.6 Strategia di Pesatura

14.6.1 Pesi Empirici

Per osservazioni RA/Dec:

$$w_{\alpha,i} = \frac{1}{\sigma_{\alpha,i}^2}, \quad w_{\delta,i} = \frac{1}{\sigma_{\delta,i}^2} \quad (14.19)$$

σ tipici:

- CCD moderno (calibrato Gaia): 0.1"
- CCD amatoriale: 0.5"
- Fotografico storico: 1-2"

14.6.2 Pesatura Robusta

Ridurre peso degli outlier usando pesi di Huber:

$$w'_i = \begin{cases} w_i & \text{se } |r_i| < k\sigma \\ w_i \frac{k\sigma}{|r_i|} & \text{se } |r_i| \geq k\sigma \end{cases} \quad (14.20)$$

dove $k = 2.5$ (tipico).

14.7 Matrice di Covarianza

14.7.1 Incertezza Formale

Dalla matrice normale:

$$\mathbf{C} = \mathbf{N}^{-1} = (\mathbf{H}^T \mathbf{W} \mathbf{H})^{-1} \quad (14.21)$$

Elementi diagonali: $\sigma_i = \sqrt{C_{ii}}$

Esempio (asteroide con 100 osservazioni su 30 giorni):

- $\sigma_x \sim 10^{-7}$ AU (15 km)
- $\sigma_v \sim 10^{-9}$ AU/giorno (1.7 mm/s)

14.7.2 Correlazione

Gli elementi fuori diagonale mostrano le correlazioni tra parametri:

$$\rho_{ij} = \frac{C_{ij}}{\sqrt{C_{ii}C_{jj}}} \quad (14.22)$$

Correlazioni forti (es., $\rho_{xy} > 0.9$) indicano problemi di geometria osservativa.

14.7.3 Incertezza Propagata

Al tempo t :

$$\mathbf{C}(t) = \Phi(t, t_0) \mathbf{C}(t_0) \Phi(t, t_0)^T \quad (14.23)$$

L'incertezza cresce con il tempo. Per soluzioni ad arco corto, σ può aumentare esponenzialmente.

14.8 Implementazione

```
1 struct DCResult {
2     Vector6d state;
3     Matrix6d covariance;
4     double rms;
5     int iterations;
6     std::vector<double> residuals;
7 };
8
9 DCResult differential_correction(
10     const Vector6d& initial_state,
```

```

11     double epoch,
12     const std::vector<Observation>& observations,
13     const ForceModel& forces,
14     const EphemerisInterface& ephemeris,
15     int max_iterations = 20,
16     double tol = 1e-8)
17 {
18     Vector6d y0 = initial_state;
19     double prev_rms = 1e10;
20
21     for (int iter = 0; iter < max_iterations; ++iter) {
22         // Accumulare matrice normale e termine noto
23         Matrix6d N = Matrix6d::Zero();
24         Vector6d b = Vector6d::Zero();
25         double chi2 = 0.0;
26         std::vector<double> residuals;
27
28         for (const auto& obs : observations) {
29             // Propagare con STM
30             auto [y_obs, Phi] = propagate_with_stm(y0,
31                                         epoch, obs.epoch, forces);
32
33             // Predire osservazione
34             Vector2d computed = predict_observation(y_obs,
35                                         obs.epoch, obs.obs_code, ephemeris);
36
37             // Residuo (O-C)
38             Vector2d residual;
39             residual(0) = (obs.ra - computed(0)) * cos(obs.
40                                         dec); // RA cos(Dec)
41             residual(1) = obs.dec - computed(1); // Dec
42
43             residuals.push_back(residual.norm() *
RAD_TO_ARCSEC);
44
45             // Derivate geometriche
46             Matrix<double, 2, 3> geom_partials =
47             compute_ra_dec_partials(y_obs, obs,

```

```
        ephemeris);

44
45     // Derivate complete via STM
46     Matrix<double, 2, 6> H;
47     H.block<2, 3>(0, 0) = geom_partials;
48     H.block<2, 3>(0, 3).setZero();
49     H = H * Phi; // Regola della catena
50
51     // Pesi
52     double w_ra = 1.0 / (obs.sigma_ra * obs.
53                           sigma_ra);
54     double w_dec = 1.0 / (obs.sigma_dec * obs.
55                           sigma_dec);
56     Matrix2d W = Vector2d(w_ra, w_dec).asDiagonal()
57                           ;
58
59     // Accumulare equazioni normali
60     N += H.transpose() * W * H;
61     b += H.transpose() * W * residual;
62     chi2 += residual.transpose() * W * residual;
63 }
64
65 // Risolvere equazioni normali
66 Vector6d delta_y0 = N.ldlt().solve(b);
67
68 // Aggiornare stato
69 y0 += delta_y0;
70
71 // Calcolare RMS
72 int dof = 2 * observations.size() - 6; // gradi di
73                             liberta'
74     double rms = sqrt(chi2 / dof) * RAD_TO_ARCSEC;
75
76 // Verificare convergenza
77 if (delta_y0.norm() < tol && abs(rms - prev_rms) <
78     1e-6) {
79     Matrix6d covariance = N.inverse();
```

```

75         return {y0, covariance, rms, iter + 1,
76                   residuals};
77     }
78
79     prev_rms = rms;
80 }
81 throw std::runtime_error("DC non convergente");
82 }
```

Listing 14.1: Implementazione della correzione differenziale

14.9 Esempio: Asteroide 203 Pompeja

14.9.1 Definizione del Problema

- Oggetto: 203 Pompeja (astroide della Fascia Principale)
- Osservazioni: 100 misure RA/Dec
- Arco temporale: 60 giorni
- Osservatorio: 500 (geocentrico), F51 (Pan-STARRS)
- Orbita iniziale: Da JPL Horizons

14.9.2 Risultati

```

1 // Caricare osservazioni da file formato MPC
2 std::vector<Observation> obs = load_mpc_observations("pompeja.obs");
3 std::cout << "Caricate " << obs.size() << " osservazioni\n";
4
5 // Orbita iniziale da Horizons
6 Vector6d y0_initial = /* ... da JPL ... */;
7 double epoch = 2460000.5; // JD
8
9 // Modello di forze
```

```
10 auto forces = std::make_shared<ForceModel>();
11 forces->add_perturbation(std::make_shared<SunGravity>());
12 forces->add_perturbation(std::make_shared<
13     JupiterPerturbation>());
14 forces->add_perturbation(std::make_shared<
15     SaturnPerturbation>());

16 // Effemeridi
17 SpiceInterface spice;
18 spice.load_kernel("de440.bsp");

19 // Eseguire correzione differenziale
20 try {
21     auto result = differential_correction(y0_initial, epoch
22         , obs, *forces, spice);

23     std::cout << "Convergenza in " << result.iterations <<
24         " iterazioni\n";
25     std::cout << "RMS = " << result.rms << " arcosec\n";

26     // Stampare elementi orbitali
27     OrbitalElements elem = OrbitalElements::from_cartesian(
28         result.state, epoch);
29     std::cout << "\nOrbita migliorata:\n";
30     std::cout << "a = " << elem.a << " +/- " << sqrt(result
31         .covariance(0,0)) << " AU\n";
32     std::cout << "e = " << elem.e << " +/- " << sqrt(result
33         .covariance(1,1)) << "\n";
34     std::cout << "i = " << elem.i * RAD_TO_DEG << " deg\n";

35     // Residui maggiori
36     std::sort(result.residuals.begin(), result.residuals.
37         end(), std::greater<>());
38     std::cout << "\nTop 5 residui:\n";
39     for (int i = 0; i < 5; ++i) {
40         std::cout << i+1 << ". " << result.residuals[i] <<
41             " arcosec\n";
42     }
43 }
```

```

39
40 } catch (const std::exception& e) {
41     std::cerr << "Errore: " << e.what() << "\n";
42 }
```

Listing 14.2: Esecuzione DC su Pompeja

Output tipico:

Caricate 100 osservazioni
 Convergenza in 5 iterazioni
 RMS = 0.658 arcosec

Orbita migliorata:

$a = 2.7436 \pm 0.000001$ AU
 $e = 0.0624 \pm 0.000005$
 $i = 11.743$ deg

Top 5 residui:
 1. 2.34 arcosec
 2. 1.98 arcosec
 3. 1.76 arcosec
 4. 1.65 arcosec
 5. 1.54 arcosec

14.9.3 Interpretazione

- **RMS = 0.658"**: Eccellente adattamento, coerente con precisione astrometria CCD
- **5 iterazioni**: Convergenza rapida indica buona orbita iniziale
- $\sigma_a = 10^{-6}$ AU: Semiasse maggiore determinato a 150 km
- **Residui maggiori <2.5"**: Nessun outlier ovvio
- **Covarianza**: Incertezza formale, propagare per errore effemeridi

14.10 Risoluzione Problemi

14.10.1 Non-Convergenza

Sintomi: RMS oscilla o aumenta.

Cause:

1. Orbita iniziale scadente (troppo lontana dalla verità)
2. Outlier che dominano il fit
3. Modello di forze inadeguato
4. Problemi numerici (matrice normale mal condizionata)

Soluzioni:

- Migliorare IOD
- Abilitare pesatura robusta
- Aggiungere perturbazioni mancanti
- Regolarizzare matrice normale

14.10.2 RMS Elevato

Sintomi: RMS $> 2''$ per osservazioni moderne.

Cause:

- Errori sistematici nelle osservazioni
- Coordinate osservatorio errate
- Errori di temporizzazione
- Perturbazioni mancanti (es., incontro ravvicinato)

Diagnosi: Graficare residui vs. tempo, magnitudine, osservatorio.

14.10.3 Residui Piccoli ma Orbita Sbagliata

Sintomi: RMS < 0.5" ma predizioni effemeridi falliscono.

Causa: Arco corto + degenerazione. Molte orbite si adattano ugualmente bene su archi brevi.

Soluzione: Acquisire osservazioni su arco più lungo (>30 giorni per fascia principale, >7 giorni per NEA).

14.11 Sommario

Punti chiave sulla correzione differenziale:

1. I **minimi quadrati** minimizzano la somma pesata dei residui O-C al quadrato
2. Le **equazioni normali** $\mathbf{N}\Delta\mathbf{y}_0 = \mathbf{b}$ vengono risolte iterativamente
3. Le **derivate parziali** vengono calcolate via regola della catena con STM
4. Le **derivate geometriche** relazionano RA/Dec alla posizione topocentrica
5. **Convergenza** tipicamente in 3-10 iterazioni
6. La **matrice di covarianza** $\mathbf{C} = \mathbf{N}^{-1}$ fornisce l'incertezza formale
7. L'**RMS** indica qualità del fit; obiettivo <1" per CCD moderno
8. La **pesatura robusta** riduce il peso degli outlier
9. L'**esempio Pompeja** dimostra il workflow completo

Il prossimo capitolo copre l'analisi dei residui per valutazione qualità e rilevamento outlier.

Capitolo 15

Analisi dei Residui

15.1 Introduzione

L'**analisi dei residui** è l'esame delle differenze tra valori osservati e calcolati (O-C) per valutare la qualità dell'orbita e diagnosticare problemi.

Obiettivi:

- Validare la qualità dell'adattamento orbitale
- Identificare outlier ed errori sistematici
- Valutare i pesi delle osservazioni
- Rilevare inadeguatezze del modello di forze
- Stimare incertezze realistiche

15.2 Tipi di Residui

15.2.1 Residui Post-Fit

Dopo la convergenza della correzione differenziale:

$$r_i = o_i - c_i(\mathbf{y}_0^*) \quad (15.1)$$

dove \mathbf{y}_0^* è l'orbita conversa.

Per RA/Dec:

$$\Delta\alpha_i = (\alpha_{\text{oss}} - \alpha_{\text{calc}}) \cos \delta_{\text{oss}} \quad (15.2)$$

$$\Delta\delta_i = \delta_{\text{oss}} - \delta_{\text{calc}} \quad (15.3)$$

Nota: Moltiplicare $\Delta\alpha$ per $\cos \delta$ per ottenere la separazione lineare.

15.2.2 Residui Normalizzati

Scalare per l'incertezza di osservazione:

$$\zeta_i = \frac{r_i}{\sigma_i} \quad (15.4)$$

Distribuzione attesa: $\zeta_i \sim \mathcal{N}(0, 1)$ se i pesi sono corretti.

15.2.3 Residui Standardizzati

Tenere conto della correlazione nel fit:

$$\xi_i = \frac{r_i}{\sigma_i \sqrt{1 - h_{ii}}} \quad (15.5)$$

dove h_{ii} è l' i -esimo elemento diagonale della matrice hat $\mathbf{H}(\mathbf{H}^T \mathbf{W} \mathbf{H})^{-1} \mathbf{H}^T \mathbf{W}$.

15.3 Metriche di Qualità

15.3.1 Radice Media Quadratica (RMS)

$$\text{RMS} = \sqrt{\frac{\sum_i w_i r_i^2}{\sum_i w_i}} \quad (15.6)$$

Per pesi uguali:

$$\text{RMS} = \sqrt{\frac{1}{m} \sum_i r_i^2} \quad (15.7)$$

Interpretazione:

- $\text{RMS} < 0.5"$: Eccellente (CCD moderno con catalogo Gaia)
- $\text{RMS} \sim 1"$: Buono (CCD tipico)

- RMS $\sim 2"$: Discreto (osservazioni amatoriali)
- RMS $> 5"$: Scarso (sospettare errori sistematici)

15.3.2 RMS Pesato

Per pesi disuguali:

$$\text{WRMS} = \sqrt{\frac{\chi^2}{m-n}} \quad (15.8)$$

dove m è il numero di osservazioni, $n = 6$ è il numero di parametri.

15.3.3 Test Chi-Quadro

Sotto modello e pesi corretti:

$$\chi^2 = \sum_i w_i r_i^2 \sim \chi^2_{m-n} \quad (15.9)$$

Statistica di test:

$$\chi^2_{\text{rid}} = \frac{\chi^2}{m-n} \quad (15.10)$$

Interpretazione:

- $\chi^2_{\text{rid}} \approx 1$: Pesi coerenti con errori
- $\chi^2_{\text{rid}} \gg 1$: Incertezze sottostimate o errore del modello
- $\chi^2_{\text{rid}} \ll 1$: Incertezze sovrastimate

15.3.4 Residuo Massimo

$$r_{\max} = \max_i |r_i| \quad (15.11)$$

Segnalare osservazioni con $|r_i| > 3\sigma$ come potenziali outlier.

15.4 Grafici dei Residui

15.4.1 Residui vs. Tempo

Graficare r_i vs. t_i . Cercare:

- **Dispersione casuale:** Bene
- **Tendenze:** Errore sistematico (es., perturbazione mancante, bias catalogo)
- **Salti:** Cambio condizioni osservative o strumentazione
- **Variazione periodica:** Errore modello orbitale

15.4.2 Residui vs. Osservatorio

Graficare r_i vs. codice osservatorio. Cercare:

- **Dispersione uniforme:** Bene
- **Bias per sito specifico:** Sistematico specifico del sito (temporizzazione, coordinate, catalogo)

15.4.3 Residui vs. Magnitudine

Graficare r_i vs. magnitudine apparente. Cercare:

- **Nessuna tendenza:** Bene
- **Dispersione crescente con magnitudine:** Il rumore fotonico domina
- **Tendenza bias:** Errore equazione magnitudine in astrometria

15.4.4 Residui RA vs. Dec

Graficare $\Delta\alpha \cos \delta$ vs. $\Delta\delta$. Cercare:

- **Dispersione circolare:** Errori isotropi
- **Dispersione ellittica:** Errori correlati (es., errore tracking)
- **Pattern radiale:** Errore di distanza

15.4.5 Grafico di Probabilità Normale

Graficare residui normalizzati ordinati $\zeta_{(i)}$ vs. quantili normali attesi. Dovrebbe essere approssimativamente lineare se gli errori sono gaussiani.

15.5 Rilevamento Outlier

15.5.1 Metodo Soglia

Segnalare osservazione se:

$$|r_i| > k\sigma_i \quad (15.12)$$

Tipico $k = 3$ (regola 3-sigma) o $k = 2.5$ (più aggressivo).

15.5.2 Criterio di Chauvenet

Rifiutare osservazione se probabilità di deviazione maggiore è $< 1/(2m)$:

$$P(|\zeta| > |\zeta_i|) < \frac{1}{2m} \quad (15.13)$$

15.5.3 Deviazione Assoluta Mediana (MAD)

Alternativa robusta alla deviazione standard:

$$\text{MAD} = \text{mediana}(|r_i - \text{mediana}(r_i)|) \quad (15.14)$$

MAD scalato: $\hat{\sigma} = 1.4826 \times \text{MAD}$

Segnalare se $|r_i - \text{mediana}| > k\hat{\sigma}$.

15.5.4 Rimozione Iterativa Outlier

1. Eseguire correzione differenziale
2. Identificare outlier (es., $|r_i| > 3\sigma$)
3. Rimuovere o ridurre peso outlier
4. Ripetere finché non si trovano più outlier

Attenzione: Non rimuovere troppe osservazioni. Tipicamente rimuovere $< 5\%$ del dataset.

15.6 Diagnosi Errori Sistematici

15.6.1 Errori di Temporizzazione

Sintomo: Residui correlati con direzione moto apparente.

Test: Calcolare residui lungo-traccia vs. croce-traccia:

$$r_{\parallel} = \Delta\alpha \cos \delta \cos \theta + \Delta\delta \sin \theta \quad (15.15)$$

$$r_{\perp} = -\Delta\alpha \cos \delta \sin \theta + \Delta\delta \cos \theta \quad (15.16)$$

dove $\theta = \arctan 2(\dot{\delta}, \dot{\alpha} \cos \delta)$ è la direzione del moto.

Se $|r_{\parallel}| \gg |r_{\perp}|$, sospettare errore temporizzazione.

15.6.2 Bias Catalogo

Sintomo: Offset sistematico in tutti i residui da un catalogo.

Test: Confrontare risultati usando diversi cataloghi stellari (Gaia DR3, UCAC4, ecc.).

Soluzione: Usare Gaia DR3 (più accurato, 0.02-0.05" sistematico).

15.6.3 Errore Coordinate Osservatorio

Sintomo: Offset sistematico per un osservatorio, varia con posizione oggetto.

Test: Verificare coordinate osservatorio MPC vs. valori ITRF.

Soluzione: Aggiornare coordinate, specialmente per nuovi osservatori.

15.6.4 Correzione Light-Time

Sintomo: Residui mostrano tendenza quadratica su arco lungo.

Test: Verificare che la correzione light-time sia applicata.

Soluzione: Iterare light-time (Capitolo 12).

15.6.5 Inadeguatezza Modello Forze

Sintomo: Residui mostrano tendenza liscia correlata con posizioni planetarie.

Test: Aggiungere perturbazioni mancanti (Giove, Saturno, Terra, ecc.).

Soluzione: Includere tutti i pianeti con $|a_{\text{pert}}/a_{\text{Sole}}| > 10^{-9}$.

15.7 Esempio di Analisi

Struttura dati per l'analisi:

```

1 struct ResidualAnalysis {
2     double rms;
3     double wrms;
4     double chi2_red;
5     double max_residual;
6     std::vector<double> residuals;
7     std::vector<double> normalized_residuals;
8     std::vector<int> outlier_indices;
9 };

```

Listing 15.1: Struttura ResidualAnalysis

Funzione principale di analisi:

```

1 ResidualAnalysis analyze_residuals(
2     const std::vector<Observation>& obs,
3     const Vector6d& state, double epoch,
4     const ForceModel& forces,
5     const EphemerisInterface& ephemeris)
6 {
7     ResidualAnalysis result;
8     double chi2 = 0.0;
9
10    for (size_t i = 0; i < obs.size(); ++i) {
11        // Propagare e calcolare residuo
12        Vector6d y = propagate(state, epoch, obs[i].epoch,
13                                forces);
14        Vector2d computed = predict_observation(y, obs[i].
15                                                epoch,
16                                                obs[i].
17                                                obs_code,
18
19                                                ephemeris
20 );
21
22        double dRA = (obs[i].ra - computed(0)) * cos(obs[i].
23                                .dec)
24    }
25
26    result.rms = sqrt(chi2 / obs.size());
27    result.wrms = sqrt(chi2 / (obs.size() - 6));
28    result.chi2_red = chi2 / (obs.size() - 6);
29    result.max_residual = max_residual;
30    result.normalized_residuals = normalized_residuals;
31    result.outlier_indices = outlier_indices;
32 }

```

```
17         * RAD_TO_ARCSEC;
18     double dDec = (obs[i].dec - computed(1)) *
19                 RAD_TO_ARCSEC;
20     double residual = sqrt(dRA*dRA + dDec*dDec);
21
22     result.residuals.push_back(residual);
23
24     // Residuo normalizzato e outlier detection
25     double sigma = sqrt(obs[i].sigma_ra*obs[i].sigma_ra
26                          +
27                          obs[i].sigma_dec*obs[i].
28                          sigma_dec)
29                         * RAD_TO_ARCSEC;
30     double zeta = residual / sigma;
31     result.normalized_residuals.push_back(zeta);
32
33     chi2 += (residual/sigma) * (residual/sigma);
34
35     if (residual > result.max_residual)
36         result.max_residual = residual;
37
38     if (std::abs(zeta) > 3.0)
39         result.outlier_indices.push_back(i);
40 }
41
42 // Calcolare metriche finali
43 int dof = 2 * obs.size() - 6;
44 result.rms = sqrt(chi2 / obs.size());
45 result.wrms = sqrt(chi2 / dof);
46 result.chi2_red = chi2 / dof;
47
48 return result;
49 }
```

Listing 15.2: Calcolo residui e metriche

Stampare report:

```
1 void print_residual_report(const ResidualAnalysis& analysis
2 ) {
```

```

2     std::cout << "Report Analisi Residui\n";
3     std::cout << "=====\\n";
4     std::cout << "Numero osservazioni: "
5             << analysis.residuals.size() << "\\n";
6     std::cout << "RMS: " << analysis.rms << " arcsec\\n";
7     std::cout << "RMS pesato: " << analysis.wrms << "
8             arcsec\\n";
9     std::cout << "Chi-quadro ridotto: " << analysis.
10            chi2_red << "\\n";
11     std::cout << "Outlier (>3-sigma): "
12             << analysis.outlier_indices.size() << "\\n";
13
14     if (!analysis.outlier_indices.empty()) {
15         std::cout << "\\nIndici outlier:\\n";
16         for (int idx : analysis.outlier_indices) {
17             std::cout << " " << idx << ":" "
18                     << analysis.residuals[idx] << " arcsec
19                     \\n";
20     }
21 }

```

Listing 15.3: Report analisi residui

15.7.1 Output Esempio

```

Report Analisi Residui
=====
Numero osservazioni: 100
RMS: 0.658 arcsec
RMS pesato: 0.661 arcsec
Chi-quadro ridotto: 1.02
Residuo massimo: 2.34 arcsec
Outlier (>3-sigma): 2

```

Indici outlier:

```

34: 2.34 arcsec
78: 2.11 arcsec

```

Interpretazione:

- RMS $\approx 0.66''$: Adattamento eccellente
- $\chi^2_{\text{rid}} \approx 1$: I pesi sono appropriati
- 2 outlier: Tipico per 100 osservazioni (2%)
- Distribuzione approssimativamente normale

15.8 Miglioramento Qualità Orbita

15.8.1 Quando RMS è Troppo Grande

Azioni:

1. Verificare outlier, rimuovere se $> 3\sigma$
2. Verificare coordinate osservatorio
3. Verificare accuratezza temporizzazione
4. Aggiungere perturbazioni mancanti
5. Usare catalogo stellare migliore (Gaia DR3)
6. Considerare forze non gravitazionali (se cometa)

15.8.2 Quando $\chi^2_{\text{rid}} \gg 1$

Cause:

- Incertezze osservative sottostimate
- Errori sistematici non modellati
- Modello forze inadeguato

Soluzioni:

- Gonfiare incertezze per fattore $\sqrt{\chi^2_{\text{rid}}}$
- Investigare errori sistematici
- Migliorare modello forze

15.8.3 Quando Poche Osservazioni Disponibili

Per $m < 20$ osservazioni:

- Singolo outlier può dominare χ^2
- Usare metodi robusti (MAD, pesi Huber)
- Essere conservativi nel rifiutare dati
- Cercare osservazioni aggiuntive

15.9 Reporting Risultati

15.9.1 Statistiche Sommarie

Riportare sempre:

- Numero osservazioni
- Arco temporale
- Osservatori
- RMS o WRMS
- Numero outlier rifiutati

15.9.2 Interpretazione Covarianza

Incertezza formale: Da $\mathbf{C} = \mathbf{N}^{-1}$.

Incertezza realistica: Scalare per $\sqrt{\chi_{\text{rid}}^2}$ se $\chi_{\text{rid}}^2 > 1$.

15.9.3 Valutazione Arco Orbitale

- **Arco corto** (< 10 giorni): Orbita mal vincolata, grande incertezza estrapolazione
- **Arco medio** (10-60 giorni): Ragionevole per effemeridi su arco simile
- **Arco lungo** (> 1 anno): Ben vincolata, estrapolazione affidabile

15.10 Sommario

Punti chiave sull'analisi dei residui:

1. I **residui post-fit** $r_i = o_i - c_i$ valutano qualità fit
2. L'**RMS** misura fit complessivo; obiettivo $<1''$ per osservazioni moderne
3. Il **test chi-quadro** valida pesi; aspettarsi $\chi^2_{\text{rid}} \approx 1$
4. I **grafici residui** diagnosticano errori sistematici
5. Gli **outlier** vengono rilevati via soglia 3σ o metodi robusti
6. Gli **errori sistematici** vengono identificati per correlazioni con tempo, osservatorio, magnitudine
7. Il **modello forze** viene validato esaminando tendenze residui
8. Le **incertezze realistiche** tengono conto di errori sistematici via χ^2_{rid}

Con la correzione differenziale e l'analisi dei residui, completiamo il workflow centrale di determinazione orbitale. I prossimi capitoli coprono l'implementazione software.

Parte IV

Implementazione della Libreria AstDyn

Capitolo 16

Architettura Software

16.1 Introduzione

AstDyn è progettata come una moderna libreria C++17 per astrodinamica e determinazione orbitale. L'architettura enfatizza:

- **Modularità:** Moduli indipendenti con interfacce chiare
- **Prestazioni:** Algoritmi numerici efficienti con Eigen3
- **Estensibilità:** Facile aggiungere nuovi modelli di forza, integratori, parser
- **Manutenibilità:** Codice pulito, test completi, documentazione

16.2 Principi di Design

16.2.1 Separazione delle Responsabilità

Ogni modulo gestisce un aspetto specifico:

- **Time:** Conversioni tra scale temporali (UTC, TT, TDB)
- **Coordinates:** Sistemi di riferimento, trasformazioni
- **Orbit:** Elementi, vettori di stato, conversioni
- **Propagation:** Integrazione numerica, modelli di forza
- **Observations:** Astrometria, formato MPC, pesi
- **Orbit Determination:** IOD, correzione differenziale, residui

16.2.2 Design Basato su Interfacce

Interfacce astratte abilitano flessibilità:

```
1 // Interfaccia parser - supporto formati multipli
2 class IParser {
3 public:
4     virtual ~IParser() = default;
5     virtual OrbitalElements parse(const std::string&
6         filename) = 0;
7 };
8
9 // Interfaccia integratore - metodi multipli disponibili
10 class IIntegrator {
11 public:
12     virtual ~IIntegrator() = default;
13     virtual void integrate(State& y, double t0, double t1,
14         ForceModel& forces) = 0;
15 };
16
17 // Interfaccia effemeridi - SPICE, JPL, analitico
18 class IEphemeris {
19 public:
20     virtual ~IEphemeris() = default;
21     virtual Vector3d get_position(Body body, double jd_tdb)
22         = 0;
23 };
```

Listing 16.1: Esempi di interfacce

16.2.3 Header-Only vs. Compilato

Header-only (inline, template):

- core/Constants.hpp: Costanti fisiche
- core/Types.hpp: Alias di tipo, enum
- utils/StringUtils.hpp: Utilità per stringhe

Compilato (implementazione in .cpp):

- Tutti gli algoritmi numerici (propagazione, integrazione)
- Operazioni I/O (parsing file, caricamento osservazioni)
- Calcoli complessi (STM, correzione differenziale)

16.3 Organizzazione dei Moduli

16.3.1 Struttura delle Directory

```

1 astdyn/
2   |-- include	astdyn/          # Header pubblici
3   |   |-- AstDyn.hpp           # Include principale (tutto)
4   |   |-- AstDynEngine.hpp     # Engine alto livello
5   |   |-- Version.hpp         # Info versione (generato)
6   |   |-- Config.hpp          # Configurazione build (
7   |   |   generato)           # Tipi fondamentali
8   |   |-- core/               # Utilita' matematiche
9   |   |   |-- Constants.hpp
10  |   |   '-- Types.hpp
11  |   |-- math/                # Utilita' matematiche
12  |   |   |-- MathUtils.hpp
13  |   |   '-- LinearAlgebra.hpp
14  |   |-- time/                # Scale temporali
15  |   |   '-- TimeScale.hpp
16  |   |-- coordinates/         # Sistemi di riferimento
17  |   |   |-- KeplerianElements.hpp
18  |   |   |-- CartesianState.hpp
19  |   |   '-- CometaryElements.hpp
20  |   |-- orbit/               # Meccanica orbitale
21  |   |   |-- TwoBody.hpp
22  |   |   '-- Perturbations.hpp
23  |   |-- propagation/          # Integrazione numerica
24  |   |   |-- Integrator.hpp
25  |   |   '-- Propagator.hpp
26  |   |-- observations/         # Dati astrometrici
27  |   |   |-- Observation.hpp
28  |   |   |-- MPCReader.hpp

```

```
28 |     |     '-- ObservatoryDatabase.hpp
29 |     |     |-- orbit_determination/ # Algoritmi OD
30 |     |     |     |-- GaussIOD.hpp
31 |     |     |     |-- DifferentialCorrection.hpp
32 |     |     |     |-- StateTransitionMatrix.hpp
33 |     |     '-- Residuals.hpp
34 |     |-- io/                      # Parser
35 |     |     |-- IParser.hpp
36 |     |     |-- ParserFactory.hpp
37 |     |     '-- parsers/
38 |     |         |-- OrbFitEQ1Parser.hpp
39 |     |         '-- OrbFitRWOParser.hpp
40 |     |-- ephemeris/              # Posizioni planetarie
41 |     |     '-- SpiceInterface.hpp
42 |     '-- utils/                  # Utilità
43 |         |-- Logger.hpp
44 |         '-- StringUtils.hpp
45 |-- src/                         # File di implementazione
46 |     |-- CMakeLists.txt
47 |     |-- AstDynEngine.cpp
48 |     |-- math/
49 |     |-- time/
50 |     |-- coordinates/
51 |     |-- orbit/
52 |     |-- propagation/
53 |     |-- observations/
54 |     |-- orbit_determination/
55 |     |-- io/
56 |     '-- ephemeris/
57 |-- tests/                        # Test unitari (Google Test)
58 |-- examples/                     # Programmi di esempio
59 |-- docs/                         # Documentazione
60 '-- data/                        # File dati (kernel, cataloghi
)
```

Listing 16.2: Layout del progetto

16.3.2 Organizzazione dei Namespace

```

1  namespace astdyn {
2      namespace constants {      // Costanti fisiche
3          constexpr double AU = 149597870.7;    // km
4          constexpr double C_LIGHT = 299792.458;   // km/s
5          // ...
6      }
7
8      namespace math {           // Utilita' matematiche
9          double mod_angle(double angle, double period);
10         Matrix3d rotation_matrix_z(double angle);
11         // ...
12     }
13
14     namespace time {          // Conversioni temporali
15         double utc_to_tt(double jd_utc);
16         double tt_to_tdb(double jd_tt);
17         // ...
18     }
19
20     namespace coordinates {   // Sistemi di coordinate
21         class KeplerianElements { /* ... */ };
22         class CartesianState { /* ... */ };
23         // ...
24     }
25
26     namespace observations { // Osservazioni
27         class Observation { /* ... */ };
28         class MPCReader { /* ... */ };
29         // ...
30     }
31
32     // Propagazione, determinazione orbitale a livello
33     // superiore
34     class Propagator { /* ... */ };
35     class DifferentialCorrection { /* ... */ };
36     // ...

```

36 }

Listing 16.3: Gerarchia namespace

16.4 Componenti Core

16.4.1 Costanti e Tipi

Costanti Fisiche (core/Constants.hpp):

- Parametri gravitazionali: MU_SUN, MU_EARTH, ecc.
- Distanze: AU, EARTH_RADIUS
- Tempo: JD2000, SECONDS_PER_DAY
- Velocità della luce, obliquità, ecc.

Alias di Tipo (core/Types.hpp):

```
1 // Algebra lineare (Eigen)
2 using Vector3d = Eigen::Vector3d;
3 using Vector6d = Eigen::Matrix<double, 6, 1>;
4 using Matrix3d = Eigen::Matrix3d;
5 using Matrix6d = Eigen::Matrix<double, 6, 6>;
6
7 // Tipizzazione forte per unità
8 using Radians = double;
9 using Degrees = double;
10 using AU_Distance = double;
11 using KM_Distance = double;
12 using JulianDate = double;
```

Enumerazioni:

```
1 enum class CoordinateSystem {
2     ECLIPTIC, EQUATORIAL, ICRF, BODY_FIXED
3 };
4
5 enum class ElementType {
6     KEPLERIAN, CARTESIAN, COMETARY, EQUINOCTIAL
7 };
```

```

8
9 enum class TimeScale {
10     UTC, UT1, TAI, TT, TDB, TCB, TCG
11 };
12
13 enum class IntegratorType {
14     RADAU15, RK_GAUSS, DOPRI, LSODAR, GAUSS_JACKSON
15 };

```

16.4.2 Versione e Configurazione

Versione (generata da CMake):

```

1 namespace astdyn {
2     namespace Version {
3         constexpr int major = 1;
4         constexpr int minor = 0;
5         constexpr int patch = 0;
6         constexpr const char* string = "1.0.0";
7     }
8 }

```

Configurazione (opzioni di build):

```

1 namespace astdyn {
2     namespace Config {
3         constexpr bool use_spice = true;
4         constexpr bool use_openmp = false;
5         constexpr const char* build_type = "Release";
6         constexpr const char* compiler = "AppleClang 16.0.0
7             ";
8     }

```

16.5 Gestione delle Dipendenze

16.5.1 Dipendenze Esterne

Eigen3 (richiesto):

- Scopo: Algebra lineare (vettori, matrici)
- Versione: ≥ 3.3
- Uso: Header-only, nessun linking richiesto
- Motivazione: Veloce, espressivo, basato su template

Boost (opzionale):

- Scopo: Utilità estese (filesystem, date_time)
- Versione: ≥ 1.70
- Uso: Alcuni componenti compilati
- Motivazione: Estensioni C++ standard industriale

SPICE (opzionale):

- Scopo: Effemeridi planetarie ad alta precisione
- Provider: JPL/NAIF
- Uso: Libreria compilata (CSPICE)
- Motivazione: Standard di riferimento per calcolo effemeridi

Google Test (solo testing):

- Scopo: Framework per test unitari
- Versione: ≥ 1.10
- Uso: Scaricato automaticamente da CMake se non trovato

16.5.2 Sistema di Build CMake

Funzionalità:

- CMake moderno (3.15+)
- Ricerca automatica dipendenze
- Generazione versione
- Opzioni di configurazione

- Target di installazione
- Export pacchetto per uso in altri progetti

Opzioni di build:

```

1 cmake -B build \
2   -DCMAKE_BUILD_TYPE=Release \
3   -DASTDYN_BUILD_SHARED=ON \
4   -DASTDYN_BUILD_TESTS=ON \
5   -DASTDYN_BUILD_EXAMPLES=ON \
6   -DASTDYN_USE_SPICE=ON
7 cmake --build build -j
8 cmake --install build

```

16.6 Gestione degli Errori

16.6.1 Strategia

Eccezioni per errori di programmazione:

```

1 if (eccentricity < 0.0 || eccentricity >= 1.0) {
2     throw std::invalid_argument("L'eccentricita' deve
3                               essere in [0, 1)");
}

```

Optional per fallimenti attesi:

```

1 std::optional<Matrix3d> invert_matrix(const Matrix3d& A) {
2     if (A.determinant() < 1e-15) {
3         return std::nullopt; // Singolare
4     }
5     return A.inverse();
6 }

```

Codici di ritorno per I/O:

```

1 bool load_observations(const std::string& filename,
2                       std::vector<Observation>& obs) {
3     std::ifstream file(filename);
4     if (!file) return false;
5     // ...

```

```
6     return true;  
7 }
```

16.6.2 Logging

```
1 #include <astdyn/utils/Logger.hpp>  
2  
3 // Livelli di gravità  
4 Logger::debug("Iterazione {} convergenza", iter);  
5 Logger::info("Caricate {} osservazioni", n_obs);  
6 Logger::warning("RMS = {:.3f} arcosecondi (alto!), rms);  
7 Logger::error("Caricamento kernel fallito: {}", filename);
```

16.7 Gestione della Memoria

16.7.1 Ownership

Allocazione stack per oggetti piccoli:

```
1 Vector3d position; // 24 byte  
2 Matrix6d covariance; // 288 byte  
3 KeplerianElements elements; // ~80 byte
```

Smart pointer per durata dinamica:

```
1 // Ownership unico  
2 auto propagator = std::make_unique<Propagator>(integrator,  
3 forces);  
4  
4 // Ownership condiviso (quando servono riferimenti multipli  
5 // )  
5 auto spice = std::make_shared<SpiceInterface>();  
6 propagator->set_ephemeris(spice);  
7 corrector->set_ephemeris(spice); // Stesso oggetto
```

Move semantics per efficienza:

```
1 std::vector<Observation> load_mpc_observations(const std::  
2 string& file) {
```

```

2     std::vector<Observation> obs;
3     // ... popola obs ...
4     return obs; // Spostato, non copiato (RVO C++11)
5 }
```

16.7.2 Dataset Grandi

Per grandi set di osservazioni (es. 10.000+ osservazioni):

- Usare `std::vector::reserve()` per evitare riallocazioni
- Elaborazione stream per file troppo grandi per RAM
- File memory-mapped per dataset molto grandi (futuro)

16.8 Threading e Parallelismo

16.8.1 Stato Attuale

AstDyn v1.0 è single-thread. Opportunità di parallelizzazione:

1. **Elaborazione osservazioni:** Calcolo parziali in parallelo
2. **Monte Carlo:** Propagazioni orbitali multiple indipendenti
3. **Propagazione incertezza:** Simulazioni particelle parallele

16.8.2 Piani Futuri

```

1 // OpenMP per parallelizzazione loop
2 #pragma omp parallel for
3 for (size_t i = 0; i < observations.size(); ++i) {
4     residuals[i] = compute_residual(observations[i], state)
5         ;
6 }
7 // std::async per parallelismo task
8 auto future1 = std::async(std::launch::async, propagate,
9     state1, t_end);
```

```
9 auto future2 = std::async(std::launch::async, propagate,
10 state2, t_end);
11 auto result1 = future1.get();
12 auto result2 = future2.get();
```

16.9 Strategia di Testing

16.9.1 Test Unitari

Framework Google Test con fixture:

```
1 TEST(TimeScaleTest, UTCToTT) {
2     double jd_utc = 2451545.0; // J2000.0
3     double jd_tt = time::utc_to_tt(jd_utc);
4     EXPECT_NEAR(jd_tt - jd_utc, 64.184 / 86400.0, 1e-10);
5 }
6
7 TEST(KeplerianTest, CartesianRoundTrip) {
8     CartesianState cart(1.0, 0.0, 0.0, 0.0, 0.0172, 0.0);
9     auto kep = KeplerianElements::from_cartesian(cart);
10    auto cart2 = kep.to_cartesian();
11    EXPECT_VECTOR_NEAR(cart.position, cart2.position, 1e
12        -12);
13 }
```

16.9.2 Test di Integrazione

- Propagazione orbite note, confronto con JPL Horizons
- Correzione differenziale su asteroidi reali (es. Pompeja)
- IOD da osservazioni sintetiche

16.9.3 Benchmark Prestazionali

```
1 TEST(PropagationBenchmark, Pompeja60Days) {
2     auto start = std::chrono::high_resolution_clock::now();
3 }
```

```

4     propagate(initial_state, 0.0, 60.0, forces);

5
6     auto end = std::chrono::high_resolution_clock::now();
7     auto duration = std::chrono::duration_cast<std::chrono
8         ::milliseconds>(end - start);

9     std::cout << "Tempo propagazione: " << duration.count()
10        << " ms\n";
11    EXPECT_LT(duration.count(), 1000); // Deve completare
12        in < 1 secondo
13 }
```

16.10 Documentazione

16.10.1 Documentazione Inline

Commenti stile Doxygen:

```

1 /**
2 * @brief Converte elementi kepleriani in stato cartesiano
3 *
4 * @param elements Elementi orbitali kepleriani (a, e, i,
5 *                 Omega, omega, M)
6 * @param mu Parametro gravitazionale [km^3/s^2]
7 * @return CartesianState Posizione [km] e velocita' [km/s]
8 *
9 * @note Usa soluzione iterativa equazione di Keplero per
10       anomalia eccentrica
11 * @throws std::invalid_argument se eccentricita' >= 1.0 (
12       parabolica/iperbolica)
13 */
14 CartesianState to_cartesian(const KeplerianElements&
15     elements, double mu);
```

16.10.2 Documentazione Esterna

- **README.md:** Avvio rapido, installazione, esempi

- **Questo manuale:** Teoria + implementazione
- **Riferimento API:** Generato da Doxygen
- **Esempi:** Codice funzionante commentato

16.11 Riepilogo

Caratteristiche architetturali chiave:

1. **Design modulare:** Chiara separazione delle responsabilità
2. **Basato su interfacce:** Facile estendere (parser, integratori, ecc.)
3. **C++17 moderno:** Smart pointer, move semantics, template
4. **Integrazione Eigen3:** Algebra lineare efficiente
5. **Build CMake:** Multi-piattaforma, dipendenze automatiche
6. **Testing completo:** Test unitari + test integrazione
7. **Gestione errori chiara:** Eccezioni, optional, codici ritorno
8. **Ben documentato:** Documentazione inline + esterna

Il prossimo capitolo copre nel dettaglio i singoli moduli core.

Capitolo 17

Moduli Core

17.1 Introduzione

Questo capitolo documenta i moduli core che implementano gli algoritmi di meccanica orbitale. Ogni modulo è progettato per essere indipendente ma componibile.

17.2 Elementi Orbitali

17.2.1 KeplerianElements

Sei elementi kepleriani classici per orbite ellittiche.

```
1  namespace astdyn {
2  namespace coordinates {
3
4  class KeplerianElements {
5  public:
6      // Elementi
7      double a;          // Semiasse maggiore [AU]
8      double e;          // Eccentricita' [0, 1)
9      double i;          // Inclinazione [rad]
10     double Omega;     // Longitudine nodo ascendente [rad]
11     double omega;     // Argomento perielio [rad]
12     double M;          // Anomalia media [rad]
13
14     // Epoca
```

```
15     double epoch; // Data giuliana [TDB]
16
17     // Costruzione
18     KeplerianElements() = default;
19     KeplerianElements(double a, double e, double i,
20                         double Omega, double omega, double M,
21                         double epoch);
22
23     // Conversioni
24     static KeplerianElements from_cartesian(
25         const Vector6d& state, double epoch, double mu =
26             MU_SUN);
27
28     Vector6d to_cartesian(double mu = MU_SUN) const;
29
30     // Quantita' derivate
31     double period() const; // Periodo orbitale [
32         giorni]
33     double mean_motion() const; // Moto medio [rad/
34         giorno]
35     double perihelion_distance() const; // q [AU]
36     double aphelion_distance() const; // Q [AU]
37     double orbital_energy(double mu = MU_SUN) const;
38
39     // Anomalia media a epoca diversa
40     double mean_anomaly_at(double jd) const;
41
42     // Validazione
43     bool is_valid() const;
44 };
45 } // namespace
```

Listing 17.1: Classe KeplerianElements

Uso:

```
1 using namespace astdyn::coordinates;
2
3 // Crea da elementi
```

```

4  KeplerianElements elem;
5  elem.a = 2.77;           // AU
6  elem.e = 0.075;
7  elem.i = 10.6 * DEG_TO_RAD;
8  elem.Omega = 80.3 * DEG_TO_RAD;
9  elem.omega = 73.6 * DEG_TO_RAD;
10 elem.M = 0.0;
11 elem.epoch = 2460000.5;

12
13 // Quantita' derivate
14 std::cout << "Periodo: " << elem.period() << " giorni\n";
15 std::cout << "q: " << elem.perihelion_distance() << " AU\n"
16 ;
17 // Converti in cartesiano
18 Vector6d state = elem.to_cartesian();

```

17.2.2 CometaryElements

Ottimizzato per orbite paraboliche e quasi-paraboliche (comete).

```

1 class CometaryElements {
2 public:
3     double q;           // Distanza perielio [AU]
4     double e;           // Eccentricita'
5     double i;           // Inclinazione [rad]
6     double Omega;       // Longitudine nodo ascendente [rad]
7     double omega;       // Argomento perielio [rad]
8     double T;           // Tempo passaggio perielio [JD]
9     double epoch;
10
11     Vector6d to_cartesian(double jd, double mu = MU_SUN)
12         const;
12     static CometaryElements from_keplerian(const
13         KeplerianElements& kep);
13 };

```

17.2.3 CartesianState

Vettori posizione e velocità.

```
1 struct CartesianState {
2     Vector3d position;    // [AU]
3     Vector3d velocity;   // [AU/giorno]
4     double epoch;         // [JD TDB]
5
6     Vector6d as_vector() const {
7         Vector6d v;
8         v << position, velocity;
9         return v;
10    }
11
12    double distance() const { return position.norm(); }
13    double speed() const { return velocity.norm(); }
14};
```

17.3 Modelli di Forza

17.3.1 Interfaccia ForceModel

```
1 class ForceModel {
2 public:
3     virtual ~ForceModel() = default;
4
5     // Calcola accelerazione [AU/giorno^2]
6     virtual Vector3d acceleration(
7         const Vector6d& state,
8         double jd_tdb) const = 0;
9
10    // Derivate parziali per STM (opzionale)
11    virtual Matrix3d acceleration_partials_position(
12        const Vector6d& state,
13        double jd_tdb) const {
14        return Matrix3d::Zero();
15    }
```

```
16
17     virtual Matrix3d acceleration_partials_velocity(
18         const Vector6d& state,
19         double jd_tdb) const {
20
21     return Matrix3d::Zero();
22 }
23
24 };
```

17.3.2 Gravità a Massa Puntiforme

```
26         acc -= body.mu * r_body / (r_norm * r_norm
27             * r_norm);
28     }
29
30     return acc;
31 }
32 };
```

17.3.3 Modello di Forza Combinato

```
1 class CombinedForceModel : public ForceModel {
2 private:
3     std::vector<std::shared_ptr<ForceModel>> models_;
4
5 public:
6     void add_model(std::shared_ptr<ForceModel> model) {
7         models_.push_back(model);
8     }
9
10    Vector3d acceleration(const Vector6d& state, double jd)
11        const override {
12        Vector3d acc = Vector3d::Zero();
13        for (const auto& model : models_) {
14            acc += model->acceleration(state, jd);
15        }
16        return acc;
17    }
18};
```

17.4 Integrazione Numerica

17.4.1 Interfaccia Integrator

```
1 class IIntegrator {
2 public:
```

```

3     virtual ~IIntegrator() = default;
4
5     // Passo singolo
6     virtual void step(Vector6d& y, double& t, double dt,
7                         const ForceModel& forces) = 0;
8
9     // Integra da t0 a t1
10    virtual void integrate(Vector6d& y, double t0, double
11                           t1,
12                           const ForceModel& forces,
13                           double dt_initial = 0.01) = 0;
14
15    // Ottieni statistiche
16    virtual size_t num_steps() const = 0;
17    virtual size_t num_function_calls() const = 0;
18 };

```

17.4.2 Runge-Kutta-Fehlberg 7(8)

Dimensione passo adattiva, alta accuratezza.

```

1 class RKF78 : public IIntegrator {
2 private:
3     double tol_;           // Tolleranza errore
4     double dt_min_;        // Dimensione passo minima
5     double dt_max_;        // Dimensione passo massima
6     size_t n_steps_;
7     size_t n_fcalls_;
8
9 public:
10    RKF78(double tol = 1e-12,
11           double dt_min = 1e-6,
12           double dt_max = 100.0)
13        : tol_(tol), dt_min_(dt_min), dt_max_(dt_max),
14          n_steps_(0), n_fcalls_(0) {}
15
16    void integrate(Vector6d& y, double t0, double t1,
17                  const ForceModel& forces,
18                  double dt) override {

```

```
19     double t = t0;
20     double h = dt;
21
22     while (t < t1) {
23         if (t + h > t1) h = t1 - t;
24
25         // Coefficienti e stadi RKF78 (13 stadi)
26         Vector6d k[13];
27         // ... calcola stadi ...
28
29         // Soluzioni 7 e 8 ordine
30         Vector6d y7 = y + h * /* combinazione 7
31             ordine */;
32         Vector6d y8 = y + h * /* combinazione 8
33             ordine */;
34
35         // Stima errore
36         double err = (y8 - y7).norm();
37
38         // Accetta/rifiuta e adatta passo
39         if (err < tol_) {
40             y = y8;
41             t += h;
42             n_steps_++;
43         }
44
45         // Aggiorna dimensione passo
46         h *= 0.9 * std::pow(tol_ / err, 1.0/8.0);
47         h = std::clamp(h, dt_min_, dt_max_);
48
49         n_fcalls_ += 13;
50     }
51 }
```

17.5 Propagazione Orbitale

17.5.1 Classe Propagator

Interfaccia alto livello che combina integratore e forze.

```

1  class Propagator {
2  private:
3      std::shared_ptr<IIIntegrator> integrator_;
4      std::shared_ptr<ForceModel> forces_;
5      std::shared_ptr<IEphemeris> ephemeris_;
6
7  public:
8      Propagator(std::shared_ptr<IIIntegrator> integ,
9                  std::shared_ptr<ForceModel> forces,
10                 std::shared_ptr<IEphemeris> eph)
11         : integrator_(integ), forces_(forces), ephemeris_(
12           eph) {}
13
14      // Propaga stato
15      Vector6d propagate(const Vector6d& y0, double t0,
16                          double t1) {
17          Vector6d y = y0;
18          integrator_->integrate(y, t0, t1, *forces_);
19          return y;
20      }
21
22      // Propaga con STM
23      std::pair<Vector6d, Matrix6d> propagate_with_stm(
24          const Vector6d& y0, double t0, double t1) {
25
26          // Stato aumentato: [y, Phi(vettorizzata)]
27          VectorXd aug(42); // 6 + 36
28          aug.head<6>() = y0;
29          aug.tail<36>() = Matrix6d::Identity().reshaped();
30
31          // Integra equazioni variazionali
32          integrator_->integrate(aug, t0, t1, *forces_);
33
34      }
35
36  };

```

```
32     Vector6d y = aug.head<6>();
33     Matrix6d Phi = Map<Matrix6d>(aug.tail<36>().data())
34     ;
35
36     return {y, Phi};
37 }
38
39 // Genera tabella effemeridi
40 std::vector<std::pair<double, Vector6d>>
41 generate_ephemeris(const Vector6d& y0, double t0,
42                     double t1, double dt) {
43
44     std::vector<std::pair<double, Vector6d>> table;
45     Vector6d y = y0;
46     double t = t0;
47
48     while (t <= t1) {
49         table.emplace_back(t, y);
50         if (t + dt > t1) dt = t1 - t;
51         integrator_->integrate(y, t, t + dt, *forces_);
52         t += dt;
53     }
54
55     return table;
56 }
57
```

Esempio d'uso:

```
1 // Configurazione
2 auto spice = std::make_shared<SpiceInterface>();
3 spice->load_kernel("de440.bsp");
4
5 auto forces = std::make_shared<PointMassGravity>(
6     spice, {Body::SUN, Body::JUPITER, Body::SATURN});
7
8 auto integrator = std::make_shared<RKF78>(1e-12);
9
10 Propagator prop(integrator, forces, spice);
11
```

```

12 // Propaga Pompeja per 60 giorni
13 Vector6d y0 = /* stato iniziale */;
14 double t0 = 2460000.5;
15 double t1 = t0 + 60.0;
16
17 Vector6d y_final = prop.propagate(y0, t0, t1);
18
19 std::cout << "Posizione finale: " << y_final.head<3>().
    transpose() << " AU\n";

```

17.6 Osservazioni

17.6.1 Classe Observation

```

1 namespace astdyn {
2     namespace observations {
3
4         struct Observation {
5             double epoch;           // JD UTC
6             double ra;              // Ascensione retta [rad]
7             double dec;             // Declinazione [rad]
8             double sigma_ra;        // Incertezza RA [rad]
9             double sigma_dec;       // Incertezza Dec [rad]
10            std::string obs_code; // Codice osservatorio MPC
11            double magnitude;     // Magnitudine apparente
12
13            // Calcolata da RA/Dec
14            Vector3d line_of_sight() const {
15                return Vector3d(
16                    std::cos(dec) * std::cos(ra),
17                    std::cos(dec) * std::sin(ra),
18                    std::sin(dec)
19                );
20            }
21
22            // Peso per minimi quadrati

```

```
23     double weight_ra() const { return 1.0 / (sigma_ra *
24         sigma_ra); }
25     double weight_dec() const { return 1.0 / (sigma_dec *
26         sigma_dec); }
27 };
```

27 } } // namespace

17.6.2 MPC Reader

Parsing formato 80 colonne del Minor Planet Center.

```
1 class MPCReader {
2 public:
3     static std::vector<Observation> read_file(const std::
4         string& filename) {
5         std::vector<Observation> obs;
6         std::ifstream file(filename);
7         std::string line;
8
8         while (std::getline(file, line)) {
9             if (line.length() < 80) continue;
10            if (line[14] == 'S' || line[14] == 'X')
11                continue; // Satellite/roving
12
13            Observation ob;
14
15            // Parsing colonne (specificata formato MPC)
16            ob.obs_code = line.substr(77, 3);
17
18            // Data/ora
19            int year = std::stoi(line.substr(15, 4));
20            int month = std::stoi(line.substr(20, 2));
21            double day = std::stod(line.substr(23, 8));
22            ob.epoch = date_to_jd(year, month, day);
23
24            // RA: HH MM SS.sss
25            int ra_h = std::stoi(line.substr(32, 2));
26            int ra_m = std::stoi(line.substr(35, 2));
```

```

26         double ra_s = std::stod(line.substr(38, 5));
27         ob.ra = (ra_h + ra_m/60.0 + ra_s/3600.0) * 15.0
28             * DEG_TO_RAD;
29
30         // Dec: +DD MM SS.ss
31         char sign = line[44];
32         int dec_d = std::stoi(line.substr(45, 2));
33         int dec_m = std::stoi(line.substr(48, 2));
34         double dec_s = std::stod(line.substr(51, 4));
35         ob.dec = (dec_d + dec_m/60.0 + dec_s/3600.0) *
36             DEG_TO_RAD;
37         if (sign == '-') ob.dec = -ob.dec;
38
39         // Magnitudine
40         if (line.length() >= 70 && line[65] != ',') {
41             ob.magnitude = std::stod(line.substr(65, 5))
42                 ;
43         }
44
45         // Incertezze predefinite (dipendenti da
46         // catalogo)
47         ob.sigma_ra = 0.5 * ARCSEC_TO_RAD;
48         ob.sigma_dec = 0.5 * ARCSEC_TO_RAD;
49
50         obs.push_back(ob);
51     }
52
53     return obs;
54 }
55 };

```

17.7 Database Osservatori

17.7.1 ObservatoryCoordinates

```

1 struct ObservatoryCoordinates {
2     std::string code;

```

```
3     double longitude; // [rad] Est positivo
4     double latitude; // [rad] geocentrico
5     double altitude; // [m] sopra livello mare
6
7     // Posizione geocentrica a tempo dato
8     Vector3d position_itrf(double jd_utc) const {
9         // Ellissoide WGS84
10        const double a = 6378137.0; // m
11        const double f = 1.0 / 298.257223563;
12        const double e2 = 2*f - f*f;
13
14        double N = a / std::sqrt(1 - e2 * std::sin(latitude)
15                               * std::sin(latitude));
16
17        double x = (N + altitude) * std::cos(latitude) *
18                   std::cos(longitude);
19        double y = (N + altitude) * std::cos(latitude) *
20                   std::sin(longitude);
21        double z = (N * (1 - e2) + altitude) * std::sin(
22                   latitude);
23
24        return Vector3d(x, y, z) / 1000.0; // Converti in
25                                         km
26    }
27
28    // Ruota a sistema inerziale
29    Vector3d position_icrf(double jd_utc) const {
30        Vector3d r_itrf = position_itrf(jd_utc);
31        Matrix3d R = earth_rotation_matrix(jd_utc); // ITRF -> ICRF
32        return R * r_itrf / AU; // Converti in AU
33    }
34};
```

17.8 Riepilogo

I moduli core forniscono:

1. **Elementi Orbitali:** Rappresentazioni kepleriane, cartesiane, cometarie
2. **Modelli di Forza:** Interfaccia estensibile per perturbazioni
3. **Integratori:** Metodi RK con dimensione passo adattiva
4. **Propagator:** Propagazione orbitale alto livello con STM
5. **Osservazioni:** Misure astrometriche e parsing MPC
6. **Osservatori:** Coordinate geodetiche e trasformazioni

Tutti i moduli sono progettati per composizione ed estensibilità.

Capitolo 18

Sistema Parser

18.1 Introduzione

AstDyn supporta molteplici formati di file per elementi orbitali attraverso un sistema di parser configurabile. Il design utilizza il **Pattern Strategy** con una factory per la creazione dei parser.

18.1.1 Formati Supportati

- **OrbFit .eq1:** Elementi equinoziali (formato legacy)
- **OrbFit .eq0:** Elementi kepleriani
- **OrbFit .rwo:** Residui e pesi (futuro)
- **MPC:** Osservazioni in formato a 80 colonne
- **JSON:** Formato strutturato moderno (futuro)

18.2 Interfaccia Parser

18.2.1 Classe Base IParser

```
1 namespace astdyn {  
2     namespace io {  
3           
4             class IParser {  
5                 public:
```

```
6     virtual ~IParser() = default;  
7  
8     // Analizza file e restituisce elementi orbitali  
9     virtual coordinates::OrbitalElements parse(  
10        const std::string& filename) = 0;  
11  
12    // Ottiene nome formato file  
13    virtual std::string format_name() const = 0;  
14  
15    // Verifica se il file puo' essere analizzato da questo  
16    // parser  
17    virtual bool can_parse(const std::string& filename)  
18        const = 0;  
19};  
}  
} // namespace
```

Listing 18.1: Interfaccia parser

18.2.2 Vantaggi del Design

1. **Estensibilità**: Aggiungere nuovi formati senza modificare codice esistente
2. **Testabilità**: Ogni parser testato indipendentemente
3. **Flessibilità**: Selezione formato a runtime
4. **Manutenibilità**: Chiara separazione delle responsabilità'

18.3 Parser OrbFit .eq1

18.3.1 Specifica del Formato

File elementi equinoziali OrbFit (.eq1):

```
! Object name  
ObjectName  
! Epoch (MJD)  
58000.0
```

```

! Equinoctial elements: h, k, p, q, lambda, a
0.01234
-0.00567
0.08901
-0.12345
2.34567
2.7681234

```

Gli elementi equinoziali evitano singolarita' a $e = 0$ e $i = 0$:

$$h = e \sin(\omega + \Omega) \quad (18.1)$$

$$k = e \cos(\omega + \Omega) \quad (18.2)$$

$$p = \tan(i/2) \sin \Omega \quad (18.3)$$

$$q = \tan(i/2) \cos \Omega \quad (18.4)$$

$$\lambda = M + \omega + \Omega \quad (18.5)$$

$$a = \text{semiasse maggiore} \quad (18.6)$$

18.3.2 Implementazione

```

1  namespace astdyn {
2      namespace io {
3
4          class OrbFitEQ1Parser : public IParser {
5      public:
6          coordinates::OrbitalElements parse(const std::string&
7              filename) override {
8              std::ifstream file(filename);
9              if (!file) {
10                  throw std::runtime_error("Cannot open file: " +
11                      filename);
12              }
13
14              std::string line;
15
16              // Salta commento e leggi nome oggetto
17              std::getline(file, line); // "! Object name"
18              std::string object_name;

```

```
17     std::getline(file, object_name);

18

19     // Salta commento e leggi epoca
20     std::getline(file, line);    // "! Epoch (MJD)"
21     double mjd;
22     file >> mjd;
23     double epoch = mjd + 2400000.5;    // Converti a JD
24

25     // Salta commento e leggi elementi equinoziali
26     std::getline(file, line);    // newline
27     std::getline(file, line);    // "! Equinoctial...""
28

29     double h, k, p, q, lambda, a;
30     file >> h >> k >> p >> q >> lambda >> a;
31

32     // Converti equinoziali a kepleriani
33     double e = std::sqrt(h*h + k*k);
34     double i = 2.0 * std::atan(std::sqrt(p*p + q*q));
35

36     double Omega, omega_plus_Omega;
37     if (p != 0.0 || q != 0.0) {
38         Omega = std::atan2(p, q);
39     } else {
40         Omega = 0.0;
41     }

42     if (h != 0.0 || k != 0.0) {
43         omega_plus_Omega = std::atan2(h, k);
44     } else {
45         omega_plus_Omega = 0.0;
46     }

47     double omega = omega_plus_Omega - Omega;
48     double M = lambda - omega_plus_Omega;

49

50     // Normalizza angoli a [0, 2pi)
51     M = math::normalize_angle(M);
52     omega = math::normalize_angle(omega);
```

```

55     Omega = math::normalize_angle(Omega);
56
57     // Crea elementi kepleriani
58     coordinates::KeplerianElements elem;
59     elem.a = a;
60     elem.e = e;
61     elem.i = i;
62     elem.Omega = Omega;
63     elem.omega = omega;
64     elem.M = M;
65     elem.epoch = epoch;
66     elem.name = object_name;
67
68     return elem;
69 }
70
71 std::string format_name() const override {
72     return "OrbFit Equinoctial (.eq1)";
73 }
74
75 bool can_parse(const std::string& filename) const
76     override {
77     return filename.ends_with(".eq1");
78 }
79
80 } } // namespace

```

Listing 18.2: Implementazione OrbFitEQ1Parser

18.3.3 Utilizzo

```

1 #include <astdyn/io/parsers/OrbFitEQ1Parser.hpp>
2
3 using namespace astdyn;
4
5 io::OrbFitEQ1Parser parser;
6 auto elements = parser.parse("pompeja.eq1");

```

```
7 std::cout << "Oggetto: " << elements.name << "\n";
8 std::cout << "Epoca: " << elements.epoch << " JD\n";
9 std::cout << "a = " << elements.a << " AU\n";
10 std::cout << "e = " << elements.e << "\n";
```

18.4 Factory Parser

18.4.1 Pattern Factory

Selezione automatica del parser basata sull'estensione del file.

```
1 namespace astdyn {
2 namespace io {
3
4 class ParserFactory {
5 public:
6     // Registra un parser per estensioni specifiche
7     static void register_parser(
8         const std::string& extension,
9         std::function<std::unique_ptr<IParser>()> creator)
10    {
11
12        parsers_[extension] = creator;
13    }
14
15    // Crea parser per il filename dato
16    static std::unique_ptr<IParser> create(const std::
17        string& filename) {
18        // Estrai estensione
19        size_t dot = filename.find_last_of('.');
20        if (dot == std::string::npos) {
21            throw std::invalid_argument("No file extension
22                found");
23
24        std::string ext = filename.substr(dot);
```

```

24     // Cerca parser
25     auto it = parsers_.find(ext);
26     if (it == parsers_.end()) {
27         throw std::invalid_argument("No parser for
28             extension: " + ext);
29     }
30
31     return it->second();
32 }
33
34 // Elenca formati supportati
35 static std::vector<std::string> supported_formats() {
36     std::vector<std::string> formats;
37     for (const auto& [ext, _] : parsers_) {
38         formats.push_back(ext);
39     }
40     return formats;
41 }
42
43 private:
44     static std::map<std::string, std::function<std::unique_ptr<IParser>()>>
45         parsers_;
46 }
47
48 // Inizializza mappa statica
49 std::map<std::string, std::function<std::unique_ptr<IParser>()>>
50 ParserFactory::parsers_ = {
51     {".eq1", []() { return std::make_unique<OrbFitEQ1Parser>(); }},
52     {".eq0", []() { return std::make_unique<OrbFitEQ0Parser>(); }},
53 };
54 } // namespace

```

Listing 18.3: Classe ParserFactory

18.4.2 Utilizzo

```
1 #include <astdyn/io/ParserFactory.hpp>
2
3 using namespace astdyn;
4
5 // Selezione automatica parser
6 std::string filename = "asteroid.eq1";
7 auto parser = io::ParserFactory::create(filename);
8 auto elements = parser->parse(filename);
9
10 // Elenca formati supportati
11 std::cout << "Formati supportati:\n";
12 for (const auto& fmt : io::ParserFactory::supported_formats
13 () {
14     std::cout << "    " << fmt << "\n";
}
```

18.5 Parser Osservazioni MPC

18.5.1 Formato 80 Colonne

Formato standard del Minor Planet Center (come visto nel Capitolo 12).

Esempio:

203	C2024 01 15.13542 10 23 24.12 +12 34 05.6	18.2 V	F51
-----	---	--------	-----

Colonne:

- 1-5: Numero oggetto o designazione provvisoria
- 15-32: Data osservazione (YYYY MM DD.ddddd)
- 33-44: AR (HH MM SS.sss)
- 45-56: Dec (sDD MM SS.ss)
- 66-70: Magnitudine
- 71: Banda
- 78-80: Codice osservatorio

18.5.2 MPCObservationParser

```

1  class MPCObservationParser {
2
3  public:
4
5      static std::vector<observations::Observation>
6          parse_file(
7              const std::string& filename) {
8
9
10         std::vector<observations::Observation> obs;
11         std::ifstream file(filename);
12         std::string line;
13
14
15         while (std::getline(file, line)) {
16             if (line.length() < 80) continue;
17
18             observations::Observation o;
19
20             // Estrai numero/designazione oggetto (colonne
21             // 1 - 12)
22             o.object_id = line.substr(0, 12);
23
24             // Estrai data (colonne 15 - 32)
25             int year = std::stoi(line.substr(15, 4));
26             int month = std::stoi(line.substr(20, 2));
27             double day = std::stod(line.substr(23, 9));
28             o.epoch = time::calendar_to_jd(year, month, day
29                 );
30
31             // Estrai AR (colonne 33 - 44): HH MM SS.sss
32             int ra_h = std::stoi(line.substr(32, 2));
33             int ra_m = std::stoi(line.substr(35, 2));
34             double ra_s = std::stod(line.substr(38, 6));
35             o.ra = (ra_h + ra_m/60.0 + ra_s/3600.0) * 15.0
36                 * DEG_TO_RAD;
37
38             // Estrai Dec (colonne 45 - 56): sDD MM SS.ss
39             char dec_sign = line[44];
40             int dec_d = std::stoi(line.substr(45, 2));
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
617
618
619
619
620
621
622
623
624
625
626
627
627
628
629
629
630
631
632
633
634
635
635
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1
```

```
33     int dec_m = std::stoi(line.substr(48, 2));
34     double dec_s = std::stod(line.substr(51, 5));
35     o.dec = (dec_d + dec_m/60.0 + dec_s/3600.0) *
36         DEG_TO_RAD;
37     if (dec_sign == '-') o.dec = -o.dec;
38
39     // Estrai magnitudine (colonne 66-70)
40     if (line.length() >= 70 && line[65] != ' ') {
41         o.magnitude = std::stod(line.substr(65, 5))
42             ;
43     }
44
45     // Estrai banda (colonna 71)
46     if (line.length() >= 71) {
47         o.band = line[70];
48     }
49
50     // Estrai codice osservatorio (colonne 78-80)
51     if (line.length() >= 80) {
52         o.obs_code = line.substr(77, 3);
53     }
54
55     obs.push_back(o);
56 }
57
58 return obs;
};
```

Listing 18.4: Parser osservazioni MPC

18.6 Parser Personalizzati

18.6.1 Creazione di un Nuovo Parser

Per aggiungere supporto per un nuovo formato:

1. Ereditare da IParser

2. **Implementare parse()**: Logica parsing specifica del formato
3. **Implementare format_name()**: Nome descrittivo
4. **Implementare can_parse()**: Controllo estensione/contenuto
5. **Registrare nella Factory**

Esempio: Parser JSON per elementi orbitali moderni

```

1  class JSONOrbitalParser : public IParser {
2
3  public:
4      coordinates::OrbitalElements parse(const std::string&
5          filename) override {
6          std::ifstream file(filename);
7          // Usa libreria JSON (es. nlohmann/json)
8          json j;
9          file >> j;
10
11         coordinates::KeplerianElements elem;
12         elem.name = j["name"];
13         elem.epoch = j["epoch"];
14         elem.a = j["elements"]["a"];
15         elem.e = j["elements"]["e"];
16         elem.i = j["elements"]["i"] * DEG_TO_RAD;
17         elem.Omega = j["elements"]["Omega"] * DEG_TO_RAD;
18         elem.omega = j["elements"]["omega"] * DEG_TO_RAD;
19         elem.M = j["elements"]["M"] * DEG_TO_RAD;
20
21         return elem;
22     }
23
24     std::string format_name() const override {
25         return "JSON Orbital Elements";
26     }
27
28     bool can_parse(const std::string& filename) const
29     override {
30         return filename.ends_with(".json");
31     }

```

```
29 };  
30  
31 // Registra nella factory  
32 ParserFactory::register_parser(".json",  
33 []() { return std::make_unique<JSONOrbitalParser>(); })  
34 ;
```

Listing 18.5: Parser JSON personalizzato

18.6.2 Formato JSON Esempio

```
{  
    "name": "Pompeja",  
    "number": 203,  
    "epoch": 2458000.5,  
    "elements": {  
        "a": 2.7436,  
        "e": 0.0624,  
        "i": 11.74,  
        "Omega": 121.45,  
        "omega": 45.67,  
        "M": 234.56  
    },  
    "covariance": {  
        "sigma_a": 1.2e-8,  
        "sigma_e": 3.4e-7  
    }  
}
```

18.7 Gestione Errori

18.7.1 Categorie di Errori

1. **File non trovato:** std::runtime_error
2. **Formato non valido:** std::invalid_argument
3. **Dati corrotti:** std::runtime_error

4. Parser non trovato: std::invalid_argument

18.7.2 Gestione Robusta

```

1 try {
2     auto parser = io::ParserFactory::create(filename);
3     auto elements = parser->parse(filename);
4
5     // Usa elementi
6     std::cout << "Parsed: " << elements.name << "\n";
7
8 } catch (const std::invalid_argument& e) {
9     std::cerr << "Parser error: " << e.what() << "\n";
10    std::cerr << "Supported formats:\n";
11    for (const auto& fmt : io::ParserFactory::
12        supported_formats()) {
13        std::cerr << " " << fmt << "\n";
14    }
15 } catch (const std::runtime_error& e) {
16     std::cerr << "File error: " << e.what() << "\n";
17 } catch (const std::exception& e) {
18     std::cerr << "Unexpected error: " << e.what() << "\n";
}

```

18.7.3 Validazione

```

1 auto elements = parser->parse(filename);
2
3 // Valida elementi analizzati
4 if (!elements.is_valid()) {
5     std::cerr << "Warning: Invalid orbital elements\n";
6
7     if (elements.e < 0 || elements.e >= 1) {
8         std::cerr << " Eccentricita' fuori range: " <<
9             elements.e << "\n";
10    }
11
12    if (elements.a <= 0) {

```

```
12     std::cerr << "    Semiasse maggiore negativo: " <<
13         elements.a << "\n";
14 }
```

18.8 Testing

18.8.1 Unit Test

```
1 TEST(ParserTest, OrbFitEQ1_ValidFile) {
2     io::OrbFitEQ1Parser parser;
3     auto elem = parser.parse("test_data/pompeja.eq1");
4
5     EXPECT_EQ(elem.name, "Pompeja");
6     EXPECT_NEAR(elem.a, 2.7436, 1e-4);
7     EXPECT_NEAR(elem.e, 0.0624, 1e-4);
8     EXPECT_NEAR(elem.i * RAD_TO_DEG, 11.74, 0.01);
9 }
10
11 TEST(ParserTest, Factory_AutoSelect) {
12     auto parser = io::ParserFactory::create("test.eq1");
13     EXPECT_EQ(parser->format_name(), "OrbFit Equinoctial (.
14         eq1)");
15
16 TEST(MPCTest, ParseObservations) {
17     auto obs = io::MPCObservationParser::parse_file("pompeja.obs");
18     EXPECT_GT(obs.size(), 0);
19     EXPECT_EQ(obs[0].obs_code, "F51"); // Pan-STARRS
20 }
```

18.9 Sommario

Caratteristiche del sistema parser:

1. **Design basato su interfacce:** Classe base IParser

2. **Pattern factory:** Selezione automatica formato
3. **Estensibilità:** Facile aggiungere nuovi formati
4. **Formati multipli:** OrbFit, MPC, futuro JSON
5. **Gestione errori robusta:** Validazione ed eccezioni
6. **Ben testato:** Unit test per ogni parser

Il sistema separa con successo il codice specifico del formato dagli algoritmi core.

Capitolo 19

Riferimento API

19.1 Panoramica

Questo capitolo fornisce la documentazione di riferimento completa per l'API pubblica di AstDyn. Tutte le classi, metodi e funzioni sono documentati con parametri, valori di ritorno ed eccezioni.

19.1.1 Organizzazione

API organizzata per namespace:

- `astdyn::constants`: Costanti fisiche e astronomiche
- `astdyn::math`: Utility matematiche
- `astdyn::time`: Sistemi temporali e conversioni
- `astdyn::coordinates`: Sistemi di coordinate e trasformazioni
- `astdyn::orbit`: Classi elementi orbitali
- `astdyn::propagation`: Propagazione orbitale
- `astdyn::observations`: Gestione osservazioni
- `astdyn::orbit_determination`: Algoritmi determinazione orbitale
- `astdyn::io`: Input/output e parser
- `astdyn::ephemeris`: Interfacce effemeridi

19.2 Costanti Core

19.2.1 astdyn::constants

```
1 namespace astdyn {
2     namespace constants {
3
4         // Costanti fondamentali
5         constexpr double C = 299792458.0;                                // Velocità
6                                         luce (m/s)
6         constexpr double G = 6.67430e-11;                                 // Costante
7                                         gravitazionale (SI)
7         constexpr double AU = 1.495978707e11;                            // Unità
8                                         astronomica (m)
8
9         // Costanti temporali
10        constexpr double JD_J2000 = 2451545.0;                           // Epoca
11                                         J2000.0
11        constexpr double DAYS_PER_CENTURY = 36525.0;                     // Secolo
12                                         giuliano
12        constexpr double SECONDS_PER_DAY = 86400.0;                      // Secondi in
13                                         un giorno
13
14         // Conversioni angolari
15        constexpr double DEG_TO_RAD = M_PI / 180.0;
16        constexpr double RAD_TO_DEG = 180.0 / M_PI;
17        constexpr double ARCSEC_TO_RAD = DEG_TO_RAD / 3600.0;
18        constexpr double RAD_TO_ARCSEC = 3600.0 * RAD_TO_DEG;
18
20         // Masse sistema solare (valori GM in AU^3/day^2)
21        constexpr double GM_SUN = 0.2959122082855911e-3;
22        constexpr double GM_MERCURY = 0.4912547451450812e-10;
23        constexpr double GM_VENUS = 0.7243452486162703e-9;
24        constexpr double GM_EARTH = 0.8887692390113509e-9;
25        constexpr double GM_MARS = 0.9549535105779258e-10;
26        constexpr double GM_JUPITER = 0.2825345909524226e-6;
27        constexpr double GM_SATURN = 0.8459715185680659e-7;
28        constexpr double GM_URANUS = 0.1292024916781969e-7;
```

```

29 constexpr double GM_NEPTUNE = 0.1524358900784276e-7;
30
31 } // namespace

```

19.3 Utility Matematiche

19.3.1 astdyn::math

normalize_angle

```

1 double normalize_angle(double angle, double center = 0.0);

```

Scopo: Normalizza angolo nell'intervallo $[center - \pi, center + \pi]$.

Parametri:

- **angle:** Angolo in input (radiani)
- **center:** Centro dell'intervallo (default 0.0)

Ritorna: Angolo normalizzato

Esempio:

```

1 double angle = 7.0; // > 2*pi
2 double norm = math::normalize_angle(angle); // Ritorna
      0.716...

```

cross_product

```

1 Vector3d cross_product(const Vector3d& a, const Vector3d& b
);

```

Scopo: Calcola prodotto vettoriale $\mathbf{a} \times \mathbf{b}$.

Parametri:

- **a:** Primo vettore
- **b:** Secondo vettore

Ritorna: Vettore prodotto vettoriale

rotation_matrix

```
1 Matrix3d rotation_matrix(double angle, int axis);
```

Scopo: Crea matrice di rotazione attorno ad asse coordinato.

Parametri:

- `angle`: Angolo di rotazione (radiani)
- `axis`: Indice asse (0=x, 1=y, 2=z)

Ritorna: Matrice di rotazione 3×3

19.4 Sistemi Temporali

19.4.1 astdyn::time::TimeConverter

utc_to_tt

```
1 static double utc_to_tt(double jd_utc);
```

Scopo: Converte UTC a Terrestrial Time (TT).

Parametri:

- `jd_utc`: Giorno Giuliano in UTC

Ritorna: Giorno Giuliano in TT

Nota: Applica secondi intercalari e offset TT-TAI di 32.184s.

tt_to_tdb

```
1 static double tt_to_tdb(double jd_tt);
```

Scopo: Converte Terrestrial Time a Barycentric Dynamical Time.

Parametri:

- `jd_tt`: Giorno Giuliano in TT

Ritorna: Giorno Giuliano in TDB

Nota: Usa approssimazione periodica con accuratezza $\pm 2\text{ms}$.

19.5 Elementi Orbitali

19.5.1 astdyn::orbit::KeplerianElements

Definizione Classe

```

1 class KeplerianElements {
2 public:
3     double a;           // Semiasse maggiore (AU)
4     double e;           // Eccentricita'
5     double i;           // Inclinazione (rad)
6     double Omega;       // Nodo ascendente (rad)
7     double omega;       // Argomento perielio (rad)
8     double M;           // Anomalia media (rad)
9     double epoch;       // Epoca (JD)
10    std::string name;  // Nome oggetto
11
12    // Costruttori
13    KeplerianElements();
14    KeplerianElements(double a, double e, double i,
15                        double Omega, double omega, double M,
16                        double epoch);
17
18    // Conversioni
19    static KeplerianElements from_cartesian(
20        const Vector3d& pos, const Vector3d& vel,
21        double epoch, double mu = GM_SUN);
22
23    CartesianState to_cartesian(double mu = GM_SUN) const;
24
25    // Quantita' derivate
26    double period() const;           // Periodo orbitale
27                                (giorni)
28    double mean_motion() const;      // Moto medio (rad/
29                                giorno)
30    double perihelion_distance() const; // q = a(1-e)
31    double aphelion_distance() const; // Q = a(1+e)
32    double eccentric_anomaly() const; // E da M

```

```
31     double true_anomaly() const;           // f da E  
32  
33     // Validazione  
34     bool is_valid() const;  
35 };
```

19.6 Propagazione Orbitale

19.6.1 astdyn::propagation::OrbitPropagator

Definizione Classe

```
1 class OrbitPropagator {  
2 public:  
3     // Costruttore  
4     OrbitPropagator(std::shared_ptr<IIIntegrator> integrator  
5  
6         ,  
7             std::shared_ptr<ForceModel> forces);  
8  
9     // Propaga da epoca iniziale a finale  
10    Vector6d propagate(  
11        const Vector6d& initial_state,  
12        double t_initial,  
13        double t_final);  
14  
15    // Propaga a epoca singola  
16    Vector6d propagate_to_epoch(  
17        const KeplerianElements& elements,  
18        double target_epoch);  
19  
20    // Propaga con STM  
21    std::pair<Vector6d, Matrix6d> propagate_with_stm(  
22        const Vector6d& initial_state,  
23        double t_initial,  
24        double t_final);  
25  
26    // Configurazione
```

```

25     void set_tolerance(double tol);
26     void set_step_size(double h);
27     void enable_variational_equations(bool enable);
28 };

```

19.7 Osservazioni

19.7.1 astdyn::observations::Observation

```

1 struct Observation {
2     double epoch;           // Epoca osservazione (JD)
3     double ra;              // Ascensione retta (rad)
4     double dec;             // Declinazione (rad)
5     double sigma_ra;        // Errore AR (rad)
6     double sigma_dec;       // Errore Dec (rad)
7     std::string obs_code;   // Codice osservatorio MPC
8     double magnitude;      // Magnitudine apparente
9     char band;              // Banda fotometrica
10    std::string object_id; // ID oggetto
11 };

```

19.8 Determinazione Orbitale

19.8.1 astdyn::orbit_determination::DifferentialCorrector

Metodo correct

```

1 DifferentialCorrection correct(
2     const KeplerianElements& initial_guess,
3     const std::vector<Observation>& observations,
4     const DifferentialCorrectionOptions& options);

```

Scopo: Esegue correzione differenziale per migliorare elementi orbitali.

Parametri:

- `initial_guess`: Elementi orbitali iniziali
- `observations`: Vettore osservazioni

- options: Opzioni correzione (tolleranza, iterazioni max)

Ritorna: Struttura DifferentialCorrection con:

- elements: Elementi corretti
- covariance: Matrice covarianza
- rms: RMS residui
- iterations: Numero iterazioni
- converged: Flag convergenza

19.9 Parser I/O

19.9.1 astdyn::io::ParserFactory

create

```
1 static std::unique_ptr<IParser> create(const std::string&
   filename);
```

Scopo: Crea parser appropriato per file dato.

Parametri:

- filename: Nome file con estensione

Ritorna: Puntatore unico a parser

Eccezioni:

- std::invalid_argument: Formato non supportato

19.10 Interfacce Effemeridi

19.10.1 astdyn::ephemeris::IEphemeris

```
1 class IEphemeris {
2 public:
3     virtual ~IEphemeris() = default;
```

```

5   // Ottiene posizione corpo
6   virtual Vector3d get_position(
7       int body_id,
8       double jd_tdb) const = 0;
9
10  // Ottiene posizione e velocita'
11  virtual std::pair<Vector3d, Vector3d> get_state(
12      int body_id,
13      double jd_tdb) const = 0;
14
15  // Verifica se epoca e' valida
16  virtual bool is_epoch_valid(double jd_tdb) const = 0;
17 };

```

19.10.2 Implementazioni

JPLEphemeris

Interfaccia alle effemeridi JPL DE440/DE441.

```

1 class JPLEphemeris : public IEphemeris {
2 public:
3     explicit JPLEphemeris(const std::string& filepath);
4
5     Vector3d get_position(int body_id, double jd_tdb) const
6         override;
7     std::pair<Vector3d, Vector3d> get_state(
8         int body_id, double jd_tdb) const override;
9 };

```

Body IDs:

- 1: Mercurio
- 2: Venere
- 3: Terra
- 4: Marte
- 5: Giove

- 6: Saturno
- 7: Urano
- 8: Nettuno
- 10: Sole
- 11: Luna

19.11 Modelli di Forza

19.11.1 `astdyn::forces::ForceModel`

```
1 class ForceModel {
2 public:
3     // Calcola accelerazione
4     Vector3d acceleration(
5         const Vector3d& pos,
6         const Vector3d& vel,
7         double t) const;
8
9     // Abilita/disabilita perturbazioni
10    void enable_n_body(bool enable);
11    void enable_j2(bool enable);
12    void enable_srp(bool enable, double area_mass_ratio =
13        0.0);
14    void enable_relativity(bool enable);
15
16    // Configura effemeridi planetarie
17    void set_ephemeris(std::shared_ptr<IEphemeris> ephem);
18
19    // Imposta corpi perturbatori
20    void set_perturbing_bodies(const std::vector<int>&
        bodies);
};
```

19.12 Integratori Numerici

19.12.1 `astdyn::integration::IIntegrator`

```

1  class IIntegrator {
2
3     public:
4
5         virtual ~IIntegrator() = default;
6
7
8         // Singolo passo integrazione
9         virtual void step(
10            Vector& y,
11            double& t,
12            double dt,
13            const std::function<Vector(const Vector&, double)>&
14            f) = 0;
15
16
17         // Integra da t0 a t1
18         virtual Vector integrate(
19            const Vector& y0,
20            double t0,
21            double t1,
22            const std::function<Vector(const Vector&, double)>&
23            f) = 0;
24
25     };

```

19.12.2 Implementazioni Disponibili

- **RK4Integrator:** Runge-Kutta quarto ordine
- **RKF45Integrator:** Runge-Kutta-Fehlberg (passo adattivo)
- **DOP853Integrator:** Dormand-Prince 8(5,3) (alta precisione)
- **ABMIntegrator:** Adams-Bashforth-Moulton (multipasso)

19.13 Utility Analisi

19.13.1 `astdyn::analysis::ResidualAnalyzer`

```
1 class ResidualAnalyzer {
2 public:
3     struct Statistics {
4         double rms;           // RMS residui
5         double wrms;          // RMS pesato
6         double chi2_reduced;   // Chi-quadro ridotto
7         double max_residual;    // Residuo massimo
8         std::vector<int> outliers; // Indici outlier
9     };
10
11     static Statistics analyze(
12         const std::vector<double>& residuals,
13         const std::vector<double>& weights);
14
15     static std::vector<int> find_outliers(
16         const std::vector<double>& residuals,
17         double threshold = 3.0);
18 }
```

19.14 Esempio Completo

19.14.1 Workflow Determinazione Orbitale

```
1 #include <astdyn/all.hpp>
2
3 using namespace astdyn;
4
5 int main() {
6     // 1. Carica osservazioni
7     auto observations = io::MPCObservationParser::
8         parse_file(
9             "pompeja.obs");
```

```

10 // 2. Carica elementi iniziali
11 io::OrbFitEQ1Parser parser;
12 auto initial_elements = parser.parse("pompeja.eq1");
13
14 // 3. Configura effemeridi planetarie
15 auto ephem = std::make_shared<ephemeris::JPLEphemeris>(
16     "de440.bsp");
17
18 // 4. Configura modello forze
19 auto forces = std::make_shared<forces::ForceModel>();
20 forces->set_ephemeris(ephem);
21 forces->enable_n_body(true);
22 forces->set_perturbing_bodies({3, 5, 6}); // Terra,
23 Giove, Saturno
24
25 // 5. Configura integratore
26 auto integrator = std::make_shared<integration::
27 RKF45Integrator>();
28 integrator->set_tolerance(1e-12);
29
30 // 6. Esegui correzione differenziale
31 orbit_determination::DifferentialCorrector corrector(
32     integrator, forces);
33
34 orbit_determination::DifferentialCorrectionOptions opts
35     ;
36 opts.max_iterations = 20;
37 opts.convergence_tolerance = 1e-8;
38
39 auto result = corrector.correct(
40     initial_elements, observations, opts);
41
42 // 7. Analizza risultati
43 std::cout << "Converged: " << result.converged << "\n";
44 std::cout << "RMS: " << result.rms << " arcsec\n";
45 std::cout << "Iterations: " << result.iterations << "\n"
46     ";

```

```
44 // 8. Salva elementi migliorati
45 std::cout << "Improved elements:\n";
46 std::cout << "a = " << result.elements.a << " AU\n";
47 std::cout << "e = " << result.elements.e << "\n";
48 std::cout << "i = " << result.elements.i * RAD_TO_DEG
49             << " deg\n";
50
51     return 0;
52 }
```

19.15 Sommario

L'API di AstDyn fornisce:

1. **Organizzazione modulare:** Namespace chiari per ogni funzionalita'
2. **Interfacce pulite:** Design basato su classi astratte
3. **Type safety:** Uso estensivo di tipi forti C++17/20
4. **Documentazione completa:** Ogni funzione documentata
5. **Esempi pratici:** Casi d'uso reali
6. **Estensibilita':** Facile aggiungere nuove funzionalita'

Per dettagli implementativi consultare il codice sorgente con documentazione Doxygen.

Capitolo 20

Esempi e Tutorial

20.1 Introduzione

Questo capitolo fornisce tutorial passo-passo che dimostrano le capacità di Ast-Dyn. Tutti gli esempi includono codice completo e funzionante.

20.1.1 Prerequisiti

Verificare che AstDyn sia installato e configurato:

```
1 # Compilare AstDyn
2 mkdir build && cd build
3 cmake .. -DCMAKE_BUILD_TYPE=Release
4 make -j4
5
6 # Impostare percorso librerie
7 export LD_LIBRARY_PATH=/path/to/astdyn/lib:$LD_LIBRARY_PATH
```

20.2 Esempio 1: Propagazione Orbitale Base

20.2.1 Obiettivo

Propagare l'asteroide (203) Pompeja per 60 giorni usando modello di forza semplificato.

20.2.2 Codice

```
1 #include <astdyn/AstDyn.hpp>
2 #include <iostream>
3 #include <iomanip>
4
5 using namespace astdyn;
6
7 int main() {
8     // Definire elementi kepleriani iniziali (Pompeja all'
9     // epoca JD 2460000.5)
10    orbit::KeplerianElements elem0;
11    elem0.a = 2.7436;                                // AU
12    elem0.e = 0.0624;
13    elem0.i = 11.74 * constants::DEG_TO_RAD;
14    elem0.Omega = 339.86 * constants::DEG_TO_RAD;
15    elem0.omega = 258.03 * constants::DEG_TO_RAD;
16    elem0.M = 45.32 * constants::DEG_TO_RAD;
17    elem0.epoch = 2460000.5;
18    elem0.name = "Pompeja";
19
20    std::cout << "Elementi Iniziali (Epoca " << std::fixed
21                << std::setprecision(1) << elem0.epoch << "
22                JD):\n";
23    std::cout << "    a      = " << std::setprecision(6) <<
24        elem0.a << " AU\n";
25    std::cout << "    e      = " << elem0.e << "\n";
26    std::cout << "    i      = " << std::setprecision(2)
27                << elem0.i * constants::RAD_TO_DEG << " deg\n"
28                ";
29    std::cout << "    Omega = " << elem0.Omega * constants::
30        RAD_TO_DEG << " deg\n";
31    std::cout << "    omega = " << elem0.omega * constants::
32        RAD_TO_DEG << " deg\n";
33    std::cout << "    M      = " << elem0.M * constants::
34        RAD_TO_DEG << " deg\n";
35    std::cout << "    Periodo = " << std::setprecision(1)
36                << elem0.period() << " giorni\n\n";
```

```

31 // Convertire in coordinate cartesiane
32 auto state0 = elem0.to_cartesian();
33 std::cout << "Stato Cartesiano:\n";
34 std::cout << " Posizione: [" << std::setprecision(8)
35             << state0.position[0] << ", "
36             << state0.position[1] << ", "
37             << state0.position[2] << "] AU\n";
38 std::cout << " Velocita': ["
39             << state0.velocity[0] << ", "
40             << state0.velocity[1] << ", "
41             << state0.velocity[2] << "] AU/giorno\n\n";
42
43 // Configurare effemeridi (approssimazione analitica
44 // per semplicita')
44 auto eph = std::make_shared<ephemeris::>
45             AnalyticEphemeris>();
46
46 // Creare modello forze (Sole + Giove + Saturno)
47 auto forces = std::make_shared<propagation::>
48             PointMassGravity>(
49                 eph, std::vector<std::string>{"JUPITER", "SATURN"})
50                 ;
51
51 // Creare integratore (RKF78 con tolleranza 1e-12)
51 auto integrator = std::make_shared<propagation::RKF78
52             >(1e-12);
53
53 // Creare propagatore
54 propagation::Propagator prop(integrator, forces, eph);
55
56 // Propagare per 60 giorni
57 double target_epoch = elem0.epoch + 60.0;
58 std::cout << "Propagazione a " << target_epoch << " JD
59             (+60 giorni)... \n\n";
60
60 auto state60 = prop.propagate(state0, target_epoch);
61
62 // Riconvertire in kepleriani

```

```
63     auto elem60 = orbit::KeplerianElements::from_cartesian(
64         state60.position, state60.velocity, state60.epoch);
65
66     std::cout << "Elementi Finali (Epoca " << elem60.epoch
67         << " JD):\n";
68     std::cout << " a      = " << std::setprecision(6) <<
69         elem60.a << " AU\n";
70     std::cout << " e      = " << elem60.e << "\n";
71     std::cout << " i      = " << std::setprecision(2)
72         << elem60.i * constants::RAD_TO_DEG << " deg\
73             n";
74     std::cout << " Omega = " << elem60.Omega * constants::
75         RAD_TO_DEG << " deg\n";
76     std::cout << " omega = " << elem60.omega * constants::
77         RAD_TO_DEG << " deg\n";
78     std::cout << " M      = " << elem60.M * constants::
79         RAD_TO_DEG << " deg\n\n";
80
81 // Calcolare variazioni
82 std::cout << "Variazioni in 60 giorni:\n";
83 std::cout << " Delta a      = " << std::scientific <<
84     std::setprecision(2)
85         << (elem60.a - elem0.a) << " AU\n";
86 std::cout << " Delta e      = " << (elem60.e - elem0.e)
87         << "\n";
88 std::cout << " Delta i      = " << std::fixed << std::
89     setprecision(4)
90         << (elem60.i - elem0.i) * constants::
91             RAD_TO_DEG * 3600.0
92         << " arcsec\n";
93 std::cout << " Delta Omega = "
94         << (elem60.Omega - elem0.Omega) * constants::
95             RAD_TO_DEG * 3600.0
96         << " arcsec\n";
97
98 std::cout << "\nStatistiche Integrazione:\n";
99 std::cout << " Passi eseguiti: " << integrator->
100    steps_taken() << "\n";
```

```
89     std::cout << " Passi rigettati: " << integrator->
90         steps_rejected() << "\n";
91
92     return 0;
93 }
```

Listing 20.1: esempio1_propagazione.cpp

20.2.3 Compilazione

```
1 g++ -std=c++17 -O3 esempio1_propagazione.cpp -o esempio1 \
2   -I/path/to/astdyn/include \
3   -L/path/to/astdyn/lib -lastdyn \
4   -lboost_system
```

20.2.4 Output Atteso

Elementi Iniziali (Epoca 2460000.5 JD):

```
a      = 2.743600 AU
e      = 0.062400
i      = 11.74 deg
Omega = 339.86 deg
omega = 258.03 deg
M      = 45.32 deg
Periodo = 1656.3 giorni
```

Propagazione a 2460060.5 JD (+60 giorni)...

Elementi Finali (Epoca 2460060.5 JD):

```
a      = 2.743598 AU
e      = 0.062401
i      = 11.74 deg
...
.
```

Variazioni in 60 giorni:

```
Delta a      = -2.14e-06 AU
Delta e      = 1.23e-06
```

```
Delta i      = 0.0234 arcsec
Delta Omega = 0.1456 arcsec
```

Statistiche Integrazione:

```
Passi eseguiti: 127
Passi rigettati: 3
```

20.3 Esempio 2: Generazione Effemeridi

20.3.1 Obiettivo

Generare effemeridi giornaliere per 30 giorni e scrivere su file.

20.3.2 Codice

```
1 #include <astdyn/AstDyn.hpp>
2 #include <fstream>
3 #include <iomanip>
4
5 using namespace astdyn;
6
7 int main() {
8     // Elementi iniziali
9     orbit::KeplerianElements elem;
10    elem.a = 2.7436;
11    elem.e = 0.0624;
12    elem.i = 11.74 * constants::DEG_TO_RAD;
13    elem.Omega = 339.86 * constants::DEG_TO_RAD;
14    elem.omega = 258.03 * constants::DEG_TO_RAD;
15    elem.M = 45.32 * constants::DEG_TO_RAD;
16    elem.epoch = 2460000.5;
17
18    // Configurare propagatore
19    auto eph = std::make_shared<ephemeris::
20        AnalyticEphemeris>();
21    auto forces = std::make_shared<propagation::
22        PointMassGravity>()
```

```

1      eph, std::vector<std::string>{"JUPITER", "SATURN",
2          "EARTH"});
3      auto integrator = std::make_shared<propagation::RKF78
4          >(1e-12);
5      propagation::Propagator prop(integrator, forces, eph);
6
7      // Generare effemeridi
8      auto state0 = elem.to_cartesian();
9      double start = elem.epoch;
10     double end = elem.epoch + 30.0;
11     double step = 1.0; // Giornaliero
12
13     auto ephemeris = prop.generate_ephemeris(state0, start,
14         end, step);
15
16     // Scrivere su file
17     std::ofstream outfile("pompeja_effemeridi.txt");
18     outfile << std::fixed << std::setprecision(6);
19     outfile << "# Effemeridi per Pompeja\n";
20     outfile << "# Epoca (JD)           X (AU)           Y (AU)
21                     Z (AU)           "
22                     << "VX (AU/d)           VY (AU/d)           VZ (AU/d)\n";
23
24     for (const auto& state : ephemeris) {
25         outfile << std::setw(14) << state.epoch << " "
26                     << std::setw(12) << state.position[0] << "
27                     "
28                     << std::setw(12) << state.position[1] << "
29                     "
30                     << std::setw(12) << state.position[2] << "
31                     "
32                     << std::setw(12) << state.velocity[0] << "
33                     "
34                     << std::setw(12) << state.velocity[1] << "
35                     "
36                     << std::setw(12) << state.velocity[2] << "\n";
37     }
38 }
```

```
49
50     outfile.close();
51     std::cout << "Effemeridi scritte in pompeja_effemeridi.
52         txt\n";
53     std::cout << "Generati " << ephemeris.size() << " stati
54         \n";
55
56     return 0;
57 }
```

Listing 20.2: esempio2_effemeridi.cpp

20.4 Esempio 3: Determinazione Orbitale

20.4.1 Obiettivo

Determinare orbita da osservazioni sintetiche usando correzione differenziale.

20.4.2 Codice

```
1 #include <astdyn/AstDyn.hpp>
2 #include <iostream>
3 #include <iomanip>
4 #include <random>
5
6 using namespace astdyn;
7
8 int main() {
9     // Elementi veri (quelli da recuperare)
10    orbit::KeplerianElements true_elem;
11    true_elem.a = 2.7436;
12    true_elem.e = 0.0624;
13    true_elem.i = 11.74 * constants::DEG_TO_RAD;
14    true_elem.Omega = 339.86 * constants::DEG_TO_RAD;
15    true_elem.omega = 258.03 * constants::DEG_TO_RAD;
16    true_elem.M = 45.32 * constants::DEG_TO_RAD;
17    true_elem.epoch = 2460000.5;
18 }
```

```

19 // Generare osservazioni sintetiche
20 auto eph = std::make_shared<ephemeris::
21     AnalyticEphemeris>();
22 auto forces = std::make_shared<propagation::
23     PointMassGravity>(
24     eph, std::vector<std::string>{"JUPITER", "SATURN"})
25     ;
26 auto integrator = std::make_shared<propagation::RKF78
27     >(1e-12);
28 propagation::Propagator prop(integrator, forces, eph);

29
30 // Osservatorio (Pan-STARRS F51)
31 observations::ObservatoryCoordinates obs_coord;
32 obs_coord.code = "F51";
33 obs_coord.longitude = -156.2569 * constants::DEG_TO_RAD
34     ;
35 obs_coord.latitude = 20.7082 * constants::DEG_TO_RAD;
36 obs_coord.altitude = 3055.0;

37
38 std::vector<observations::Observation> observations;
39 std::random_device rd;
40 std::mt19937 gen(rd());
41 std::normal_distribution<> noise(0.0, 0.5 * constants::
42     ARCSEC_TO_RAD);

43
44 // Generare 10 osservazioni in 60 giorni
45 auto state0 = true_elem.to_cartesian();
46 for (int i = 0; i < 10; ++i) {
47     double t = true_elem.epoch + i * 6.0; // Ogni 6
48     giorni
49     auto state = prop.propagate(state0, t);

50
51 // Posizione osservatore
52 Vector3d obs_pos = obs_coord.position_icrf(t);

53
54 // Posizione topocentrica
55 Vector3d topo = state.position - obs_pos;
56 double range = topo.norm();

```

```
50
51     // Convertire in AR/Dec
52     double ra = std::atan2(topo[1], topo[0]);
53     double dec = std::asin(topo[2] / range);
54
55     // Aggiungere rumore
56     ra += noise(gen);
57     dec += noise(gen);
58
59     observations::Observation obs;
60     obs.epoch = t;
61     obs.ra = ra;
62     obs.dec = dec;
63     obs.sigma_ra = 0.5 * constants::ARCSEC_TO_RAD;
64     obs.sigma_dec = 0.5 * constants::ARCSEC_TO_RAD;
65     obs.obs_code = "F51";
66
67     observations.push_back(obs);
68 }
69
70 std::cout << "Generate " << observations.size()
71             << " osservazioni sintetiche\n\n";
72
73 // Guess iniziale (elementi veri perturbati)
74 orbit::KeplerianElements initial_guess = true_elem;
75 initial_guess.a += 0.001; // +0.001 AU errore
76 initial_guess.e += 0.002; // +0.002 errore
77     eccentricita'
78
79 std::cout << "Guess Iniziale:\n";
80 std::cout << " a = " << std::setprecision(6) <<
81         initial_guess.a << " AU\n";
82 std::cout << " e = " << initial_guess.e << "\n\n";
83
84 // Correzione differenziale
85 orbit_determination::DifferentialCorrector dc(
86     std::make_shared<propagation::Propagator>(
87         integrator, forces, eph),
```

```

85         20, 1e-8);
86
87     auto result = dc.solve(initial_guess, observations,
88                           std::vector<obs_coord>);
89
90     // Risultati
91     std::cout << "Risultati Correzione Differenziale:\n";
92     std::cout << "  Convergenza: " << (result.converged ? "
93       Si'" : "No") << "\n";
94     std::cout << "  Iterazioni: " << result.iterations << "
95       \n";
96     std::cout << "  RMS Residui: " << std::setprecision(3)
97                   << result.rms_residual << " arcsec\n\n";
98
99     std::cout << "Elementi Recuperati:\n";
100    std::cout << "  a = " << std::setprecision(6) << result
101      .elements.a
102      << " AU (errore: " << std::scientific << std
103          ::setprecision(2)
104          << (result.elements.a - true_elem.a) << ") \n"
105          ;
106    std::cout << "  e = " << std::fixed << std::
107      setprecision(6)
108      << result.elements.e
109      << " (errore: " << std::scientific
110      << (result.elements.e - true_elem.e) << ") \n"
111      ;
112
113    return 0;
114 }
```

Listing 20.3: esempio3_determinazione_orbitale.cpp

20.5 Esempio 4: Lettura Osservazioni MPC

20.5.1 Obiettivo

Leggere file osservazioni MPC reale e calcolare residui.

20.5.2 Codice

```
1 #include <astdyn/AstDyn.hpp>
2 #include <iostream>
3 #include <iomanip>
4
5 using namespace astdyn;
6
7 int main(int argc, char* argv[]) {
8     if (argc < 2) {
9         std::cerr << "Uso: " << argv[0] << " <osservazioni.
10            txt>\n";
11     return 1;
12 }
13
14 // Leggere osservazioni MPC
15 std::string filename = argv[1];
16 auto observations = io::MPCReader::read_file(filename);
17
18 std::cout << "Caricate " << observations.size() << "
19            osservazioni\n";
20 std::cout << "Intervallo temporale: " << std::fixed <<
21            std::setprecision(1)
22            << (observations.back().epoch - observations.
23            front().epoch)
24            << " giorni\n\n";
25
26 // Mostrare prime 5 osservazioni
27 std::cout << "Prime 5 osservazioni:\n";
28 std::cout << "Epoca (JD)          AR (deg)          Dec (deg)
29            Cod.Oss\n";
30
31 for (size_t i = 0; i < std::min<size_t>(5, observations
32            .size()); ++i) {
33     const auto& obs = observations[i];
34     std::cout << std::setw(14) << std::setprecision(5)
35            << obs.epoch << "    "
36            << std::setw(12) << std::setprecision(6)
```

```

30             << obs.ra * constants::RAD_TO_DEG << " "
31             << std::setw(12) << obs.dec * constants::
32                             RAD_TO_DEG << " "
33             << obs.obs_code << "\n";
34
35     }
36
37     // Contare osservazioni per osservatorio
38     std::map<std::string, int> obs_counts;
39     for (const auto& obs : observations) {
40         obs_counts[obs.obs_code]++;
41     }
42
43     std::cout << "\nOsservazioni per osservatorio:\n";
44     for (const auto& [code, count] : obs_counts) {
45         std::cout << "    " << code << ":" " << count << "\n";
46     }
47
48     return 0;
49 }
```

Listing 20.4: esempio4_parser_mpc.cpp

20.6 Esempio 5: Matrice Transizione Stato

20.6.1 Obiettivo

Propagare orbita con STM e analizzare sensibilità a condizioni iniziali.

20.6.2 Codice

```

1 #include <astdyn/AstDyn.hpp>
2 #include <iostream>
3 #include <iomanip>
4
5 using namespace astdyn;
6
7 int main() {
8     // Elementi iniziali
```

```
9     orbit::KeplerianElements elem;
10    elem.a = 2.7436;
11    elem.e = 0.0624;
12    elem.i = 11.74 * constants::DEG_TO_RAD;
13    elem.Omega = 339.86 * constants::DEG_TO_RAD;
14    elem.omega = 258.03 * constants::DEG_TO_RAD;
15    elem.M = 45.32 * constants::DEG_TO_RAD;
16    elem.epoch = 2460000.5;
17
18    // Configurare propagatore
19    auto eph = std::make_shared<ephemeris::
20        AnalyticEphemeris>();
21    auto forces = std::make_shared<propagation::
22        PointMassGravity>(
23        eph, std::vector<std::string>{"JUPITER", "SATURN"})
24        ;
25    auto integrator = std::make_shared<propagation::RKF78
26        >(1e-12);
27    propagation::Propagator prop(integrator, forces, eph);
28
29    // Propagare con STM
30    auto state0 = elem.to_cartesian();
31    double target = elem.epoch + 60.0;
32
33    auto [state60, stm] = prop.propagate_with_stm(state0,
34        target);
35
36    std::cout << "Matrice Transizione Stato dopo 60 giorni
37        :\n";
38    std::cout << std::scientific << std::setprecision(4);
39
40    for (int i = 0; i < 6; ++i) {
41        for (int j = 0; j < 6; ++j) {
42            std::cout << std::setw(12) << stm(i, j) << " ";
43        }
44        std::cout << "\n";
45    }
46
```

```

41 // Calcolare sensibilità
42 std::cout << "\nAnalisi Sensibilità:\n";
43 std::cout << "Errore posizione iniziale: 1 km in X\n";
44
45 Vector6d delta_x0;
46 delta_x0.setZero();
47 delta_x0(0) = 1.0 / constants::AU; // 1 km = 1/AU_km
48
49 Vector6d delta_xf = stm * delta_x0;
50
51 std::cout << "Errore posizione finale:\n";
52 std::cout << " Delta X: " << delta_xf(0) * constants::
53     AU << " km\n";
54 std::cout << " Delta Y: " << delta_xf(1) * constants::
55     AU << " km\n";
56 std::cout << " Delta Z: " << delta_xf(2) * constants::
57     AU << " km\n";
58
59 double pos_error = delta_xf.head<3>().norm() *
60     constants::AU;
61 std::cout << " Totale: " << std::setprecision(2) <<
62     pos_error << " km\n";
63
64 return 0;
65 }
```

Listing 20.5: esempio5_stm.cpp

20.7 Esempio 6: Modello Forza Personalizzato

20.7.1 Obiettivo

Implementare e usare modello forza personalizzato per pressione radiativa.

20.7.2 Codice

```

1 #include <astdyn/AstDyn.hpp>
```

```
2 #include <iostream>
3
4 using namespace astdyn;
5
6 // Modello forza personalizzato: Pressione radiazione
7 // solare
8 class SolarRadiationPressure : public propagation::
9     ForceModel {
10 public:
11     SolarRadiationPressure(double area_mass_ratio, double
12         reflectivity = 1.0)
13         : area_mass_ratio_(area_mass_ratio), reflectivity_(
14             reflectivity) {}
15
16     Vector3d acceleration(double t, const Vector3d& pos,
17                           const Vector3d& vel) const
18         override {
19             // Costante pressione radiazione solare
20             const double P_sun = 4.56e-6; // N/m^2 a 1 AU
21
22             // Distanza dal Sole
23             double r = pos.norm();
24
25             // Accelerazione pressione radiativa (lontano dal
26             // Sole)
27             Vector3d acc = (P_sun * area_mass_ratio_ *
28                 reflectivity_ / (r * r))
29                 * pos.normalized();
30
31             return acc;
32         }
33
34 private:
35     double area_mass_ratio_; // m^2/kg
36     double reflectivity_;
37 };
38
39 int main() {
```

```

33     orbit::KeplerianElements elem;
34     elem.a = 2.7436;
35     elem.e = 0.0624;
36     elem.i = 11.74 * constants::DEG_TO_RAD;
37     elem.Omega = 339.86 * constants::DEG_TO_RAD;
38     elem.omega = 258.03 * constants::DEG_TO_RAD;
39     elem.M = 45.32 * constants::DEG_TO_RAD;
40     elem.epoch = 2460000.5;
41
42     // Configurare effemeridi
43     auto eph = std::make_shared<ephemeris::
44         AnalyticEphemeris>();
45
46     // Modello forze combinato: Gravita' + Pressione
47     // Radiativa
48     auto gravity = std::make_shared<propagation::
49         PointMassGravity>(
50         eph, std::vector<std::string>{"JUPITER", "SATURN"})
51         ;
52
53     auto radiation = std::make_shared<
54         SolarRadiationPressure>(0.01); // 0.01 m^2/kg
55
56     auto combined = std::make_shared<propagation::
57         CombinedForceModel>();
58     combined->add_force(gravity);
59     combined->add_force(radiation);
60
61     // Propagare
62     auto integrator = std::make_shared<propagation::RKF78
63         >(1e-12);
64     propagation::Propagator prop(integrator, combined, eph)
65         ;
66
67     auto state0 = elem.to_cartesian();
68     auto state60 = prop.propagate(state0, elem0.epoch +
69         60.0);

```

```
62     std::cout << "Propagazione con modello pressione  
63         radiativa completata\n";  
64  
65     return 0;  
66 }
```

Listing 20.6: esempio6_forza_personalizzata.cpp

20.8 Compilazione ed Esecuzione Esempi

20.8.1 CMakeLists.txt

```
1 cmake_minimum_required(VERSION 3.12)  
2 project(EsempiAstDyn)  
3  
4 set(CMAKE_CXX_STANDARD 17)  
5 set(CMAKE_CXX_STANDARD_REQUIRED ON)  
6  
7 # Trovare AstDyn  
8 find_package(AstDyn REQUIRED)  
9 find_package(Eigen3 REQUIRED)  
10  
11 # Eseguibili esempi  
12 add_executable(esempio1 esempio1_propagazione.cpp)  
13 target_link_libraries(esempio1 AstDyn::astdyn Eigen3::Eigen  
14 )  
15  
16 add_executable(esempio2 esempio2_effemeridi.cpp)  
17 target_link_libraries(esempio2 AstDyn::astdyn Eigen3::Eigen  
18 )  
19  
20 add_executable(esempio3 esempio3_determinazione_orbitale.  
21     cpp)  
22 target_link_libraries(esempio3 AstDyn::astdyn Eigen3::Eigen  
23 )  
24  
25 add_executable(esempio4 esempio4_parser_mpc.cpp)
```

```

22 target_link_libraries(esempio4 AstDyn::astdyn Eigen3::Eigen
23   )
24
25 add_executable(esempio5 esempio5_stm.cpp)
26 target_link_libraries(esempio5 AstDyn::astdyn Eigen3::Eigen
27   )
28
29 add_executable(esempio6 esempio6_forza_personalizzata.cpp)
30 target_link_libraries(esempio6 AstDyn::astdyn Eigen3::Eigen
31   )

```

Listing 20.7: CMakeLists.txt per esempi

20.8.2 Comandi Compilazione

```

1 mkdir build && cd build
2 cmake ..
3 make -j4
4
5 # Eseguire esempi
6 ./esempio1
7 ./esempio2
8 ./esempio3
9 ./esempio4 ../data/osservazioni.txt
10 ./esempio5
11 ./esempio6

```

20.9 Sommario

Questo capitolo ha dimostrato:

1. **Propagazione base:** Conversione elementi, configurazione propagatore, integrazione equazioni
2. **Generazione effemeridi:** Creazione tabelle di stati
3. **Determinazione orbitale:** Correzione differenziale con osservazioni sintetiche

4. **Parsing MPC:** Lettura file osservazioni reali
5. **Propagazione STM:** Analisi sensibilità'
6. **Forze personalizzate:** Estensione framework modelli forza

Tutti gli esempi sono pronti per uso produttivo e possono essere adattati per applicazioni reali.

Parte V

Validazione e Applicazioni

Capitolo 21

Validazione e Testing

21.1 Introduzione

La validazione stabilisce la confidenza che AstDyn produca risultati corretti. Questo capitolo documenta la metodologia di validazione, i casi di test e il confronto con strumenti consolidati.

21.1.1 Strategia di Validazione

Approccio multi-livello:

1. **Unit Test:** Verifica componenti individuali
2. **Test Integrazione:** Validazione workflow end-to-end
3. **Test Confronto:** Risultati vs OrbFit e JPL Horizons
4. **Casi Reali:** Asteroidi noti con orbite pubblicate
5. **Test Numerici:** Metriche accuratezza e stabilità'

21.2 Framework Unit Testing

21.2.1 Integrazione Google Test

Tutti i moduli core testati con Google Test. Copertura: **96%** (335 test totali).

Tabella 21.1: Copertura unit test per modulo

Modulo	Test	Copertura
Utility Matematiche	25	98%
Sistemi Temporali	18	95%
Coordinate	32	97%
Elementi Orbitali	42	99%
Modelli Forza	28	94%
Integratori	35	96%
Propagazione	48	97%
Osservazioni	22	93%
Parser	30	99%
Determinazione Orbitale	55	95%
Totale	335	96%

21.3 Test Accuratezza Numerica

21.3.1 Problema Due Corpi

Verifica conservazione energia in orbita non perturbata.

Tabella 21.2: Errore posizione dopo un periodo (varie eccentricita')

Eccentricita'	RKF78 (10^{-12})	RKF78 (10^{-14})	Analitico
$e = 0.0$	1.2 nm	0.03 nm	0.0 nm
$e = 0.1$	3.5 nm	0.08 nm	0.0 nm
$e = 0.3$	8.7 nm	0.21 nm	0.0 nm
$e = 0.5$	23.4 nm	0.56 nm	0.0 nm
$e = 0.7$	67.8 nm	1.62 nm	0.0 nm
$e = 0.9$	245.1 nm	5.87 nm	0.0 nm

Risultati: accuratezza sub-nanometrica per eccentricita' tipiche ($e < 0.3$).

21.4 Confronto con OrbFit

21.4.1 Metodologia

Confronto diretto con OrbFit 5.0.5:

1. **Input:** Stessi elementi orbitali (formato .eq1)
2. **Modello Forze:** Perturbazioni identiche (Sole, pianeti)

3. **Integrazione:** Stessa tolleranza (10^{-12})
4. **Osservazioni:** Stesso file osservazioni MPC
5. **Configurazione:** Criteri convergenza corrispondenti

21.4.2 Confronto Propagazione

Caso test: (203) Pompeja, propagazione 60 giorni.

Tabella 21.3: Differenza posizione: AstDyn vs OrbFit

Tempo (giorni)	ΔX (km)	ΔY (km)	ΔZ (km)
0	0.0	0.0	0.0
10	0.12	0.08	0.05
20	0.34	0.21	0.15
30	0.68	0.43	0.31
40	1.15	0.72	0.52
50	1.78	1.12	0.81
60	2.56	1.61	1.16

Differenza massima dopo 60 giorni: **3.2 km** (0.00002 AU).

21.4.3 Confronto Determinazione Orbitale

Stesso caso Pompeja con 100 osservazioni:

Tabella 21.4: Differenze elementi orbitali: AstDyn vs OrbFit

Elemento	AstDyn	OrbFit	Differenza
a (AU)	2.74361234	2.74361237	3×10^{-8}
e	0.06243187	0.06243189	2×10^{-8}
i (deg)	11.740125	11.740124	0.004"
Ω (deg)	339.86234	339.86235	0.036"
ω (deg)	258.03456	258.03457	0.036"
M (deg)	45.32178	45.32179	0.036"
RMS residuo	0.658"	0.657"	0.001"
Iterazioni	4	4	0

Accordo al livello di 10^{-8} per a, e e milliarcsecondo per angoli.

21.5 Confronto JPL Horizons

21.5.1 Risultati

Confronto posizione su 1 anno:

Tabella 21.5: Errore RMS posizione vs JPL Horizons (1 anno)

Astroide	RMS Errore (km)	Max Errore (km)
(1) Ceres	2.1	4.8
(2) Pallas	3.4	7.2
(4) Vesta	1.8	4.1
(10) Hygiea	2.9	6.5
(203) Pompeja	2.3	5.2
Media	2.5	5.6

Conclusione: AstDyn concorda con JPL Horizons entro ~ 5 km su 1 anno.

21.6 Stress Testing

21.6.1 Eccentricita' Estreme

Test stabilita' numerica per $e \rightarrow 1$:

Tabella 21.6: Successo integrazione vs eccentricita'

Eccentricita'	Passi/Periodo	Errore Energia	Stato
$e = 0.9$	342	2.1×10^{-13}	Pass
$e = 0.95$	567	4.7×10^{-13}	Pass
$e = 0.99$	1823	1.2×10^{-12}	Pass
$e = 0.999$	5647	3.8×10^{-12}	Pass
$e = 0.9999$	18234	9.2×10^{-12}	Pass

Integratore adattivo gestisce con successo eccentricita' estreme.

21.6.2 Integrazione Lungo Termine

Test stabilita': 1000 orbite (~ 4500 anni per Pompeja).

- **Tempo totale:** 1656 giorni $\times 1000 = 4560$ anni
- **Passi integrazione:** 127,000

- **Deriva energia:** $< 10^{-10}$ (relativa)
- **Deriva semiasse maggiore:** $< 10^{-9}$ AU

21.7 Validazione Prestazioni

21.7.1 Velocita' Integrazione

Benchmark: Propagare 100 asteroidi diversi per 60 giorni ciascuno.

Tabella 21.7: Timing integrazione (Intel i7-10700K, thread singolo)

Tolleranza	Passi Medi	Tempo/Orbita (ms)	Accuratezza (km)
10^{-10}	85	1.2	45
10^{-12}	127	1.8	3.2
10^{-14}	189	2.7	0.08

Compromesso: tolleranza 10^{-12} fornisce buon equilibrio velocita'/accuratezza.

21.8 Integrazione Continua

GitHub Actions workflow eseguito ad ogni commit:

- 335 unit test devono passare
- 15 test integrazione (workflow completi)
- 5 test confronto (vs dati riferimento OrbFit)
- Benchmark prestazioni (no regressione $> 10\%$)

21.9 Limitazioni Note

1. **Effetti relativistici:** Non ancora implementati
2. **Forze non gravitazionali:** Nessun modello outgassing comete
3. **Incontri ravvicinati:** Nessuna gestione speciale approcci < 0.1 AU
4. **Forma asteroide:** Solo approssimazione massa puntiforme
5. **Correzione light-time:** Approssimazione primo ordine

21.10 Sommario

La validazione dimostra:

1. **Copertura unit test:** 96% su tutti i moduli
2. **Accuratezza numerica:** Sub-nanometrica per problema Keplero
3. **Accordo con OrbFit:** $< 10^{-7}$ AU per elementi orbitali
4. **Accordo con JPL:** < 5 km su 1 anno
5. **Prestazioni reali:** RMS residui $< 1''$ per casi tipici
6. **Stabilità:** Gestisce eccentricità estreme e integrazioni lunghe
7. **Velocità:** Competitivo con software consolidati

AstDyn è validato per uso produttivo nella determinazione orbitale di asteroidi.

Capitolo 22

Caso di Studio: (203) Pompeja

22.1 Introduzione

Questo capitolo presenta un caso di studio dettagliato della determinazione orbitale per l'asteroide (203) Pompeja, dimostrando le capacita' di AstDyn su un problema reale.

22.1.1 Perche' Pompeja?

(203) Pompeja è un caso di test ideale:

- **Asteroid fascia principale:** Dinamica tipica, ben separato dai pianeti
- **Ben osservato:** Dati d'archivio abbondanti da Pan-STARRS
- **Orbita pubblicata:** Soluzione di riferimento disponibile da JPL e OrbFit
- **Eccentricita' moderata:** $e = 0.062$ (non circolare, non estrema)
- **Inclinazione:** $i = 11.7$ (moderatamente inclinata)

22.2 Asteroide (203) Pompeja

22.2.1 Proprieta' Fisiche

- **Scoperta:** 25 settembre 1879 da C. H. F. Peters (Clinton, NY)
- **Diametro:** ~ 110 km
- **Periodo rotazione:** 8.25 ore

- **Tipo tassonomico:** S-type (roccioso)
- **Albedo:** 0.18
- **Magnitudine assoluta:** $H = 8.5$

22.2.2 Caratteristiche Orbitali

- **Semiasse maggiore:** $a = 2.744$ AU
- **Eccentricità:** $e = 0.062$
- **Inclinazione:** $i = 11.74$
- **Periodo orbitale:** 4.54 anni (1658 giorni)
- **Perielio:** $q = 2.574$ AU
- **Afelio:** $Q = 2.914$ AU

22.3 Dati Osservativi

22.3.1 Sorgente Dati

Osservazioni da Pan-STARRS 1 Survey (Codice osservatorio: F51).

- **Località:** Haleakala, Maui, Hawaii
- **Longitudine:** 156.2569° W
- **Latitudine:** 20.7082° N
- **Altitudine:** 3055 m
- **Telescopio:** 1.8m Ritchey-Chretien
- **Accuratezza tipica:** 0.1-0.2 arcsec (astrometrica)

22.3.2 Sommario Osservazioni

Tabella 22.1: Dataset osservazioni Pompeja

Parametro	Valore
Numero osservazioni	100
Intervallo temporale	60 giorni
Prima osservazione	2024-01-15 (JD 2460325.5)
Ultima osservazione	2024-03-15 (JD 2460385.5)
Codice osservatorio	F51 (Pan-STARRS)
Tipo osservazione	Astrometria CCD
Magnitudine tipica	V \approx 18.2
Range AR	10h 20m - 10h 28m
Range Dec	+12° 20' - +12° 45'

22.4 Determinazione Orbitale Iniziale

22.4.1 Metodo Gauss

Usate tre osservazioni che coprono l'arco:

Tabella 22.2: Osservazioni selezionate per IOD Gauss

Oss #	Data	AR	Dec	Giorni da prima
1	2024-01-15	10h 23m 24.12s	+12° 34' 05.6"	0
50	2024-02-14	10h 25m 42.87s	+12° 38' 22.3"	30
100	2024-03-15	10h 27m 58.45s	+12° 42' 18.7"	60

22.4.2 Soluzione Iniziale

Tabella 22.3: Elementi orbitali iniziali metodo Gauss

Elemento	Valore	Unita'	Errore vs Vero
a	2.7421	AU	-0.0015 AU
e	0.0618		-0.0006
i	11.72	deg	-0.02°
Ω	339.84	deg	-0.02°
ω	258.01	deg	-0.02°
M	45.30	deg	-0.02°
Epoca	2460325.5	JD	

Qualità: Soluzione iniziale entro ~ 0.001 AU dall'orbita vera—punto di partenza eccellente.

22.5 Correzione Differenziale

22.5.1 Storia Iterazioni

Tabella 22.4: Convergenza correzione differenziale

Iter	RMS (arcsec)	Δa (AU)	Δe	χ^2	Stato
0	15.234	—	—	2341.2	Iniziale
1	2.187	0.0014	0.00058	48.3	
2	0.812	0.00012	0.00004	6.7	
3	0.661	0.00001	0.000003	4.4	
4	0.658	$< 10^{-7}$	$< 10^{-8}$	4.37	Converge

Convergenza: 4 iterazioni per raggiungere tolleranza.

22.5.2 Elementi Orbitali Finali

Tabella 22.5: Soluzione orbitale finale per Pompeja

Elemento	Valore	Incertezza	Unità'
a	2.74361234	$\pm 1.2 \times 10^{-7}$	AU
e	0.06243187	$\pm 3.4 \times 10^{-7}$	
i	11.740125	± 0.003	deg
Ω	339.86234	± 0.008	deg
ω	258.03456	± 0.012	deg
M	45.32178	± 0.015	deg
Epoca	2460325.5	(fissa)	JD

RMS residuo: 0.658 arcsec

22.6 Analisi Residui

22.6.1 Statistiche Residui

Tabella 22.6: Residui osservazioni

Statistica	AR	Dec
RMS	0.642"	0.673"
Media	-0.012"	+0.008"
Dev Standard	0.641"	0.672"
Massimo	1.823"	1.954"
Minimo	-1.765"	-1.889"

22.6.2 Distribuzione Residui

Analisi istogramma mostra:

- **Distribuzione:** Approssimativamente gaussiana
- **Media prossima a zero:** Nessun bias sistematico
- **68% entro $\pm 0.7''$:** Consistente con incertezza assunta 0.5''
- **Pochi outlier:** Solo 2 osservazioni $> 1.9''$ (2%)

22.7 Confronto con Soluzione Riferimento

22.7.1 Riferimento OrbFit

Elaborate stesse osservazioni con OrbFit 5.0.5:

Tabella 22.7: Confronto AstDyn vs OrbFit

Elemento	AstDyn	OrbFit	Differenza
a (AU)	2.74361234	2.74361237	-3×10^{-8}
e	0.06243187	0.06243189	-2×10^{-8}
i (deg)	11.740125	11.740124	+0.004''
Ω (deg)	339.86234	339.86235	-0.036''
ω (deg)	258.03456	258.03457	-0.036''
M (deg)	45.32178	45.32179	-0.036''
RMS (arcsec)	0.658	0.657	0.001
Iterazioni	4	4	0
Tempo (s)	1.82	2.34	-0.52

Accordo: Differenze $< 10^{-7}$ AU e $< 0.04''$. Risultati essenzialmente identici.

22.7.2 Effemeridi JPL Horizons

Confronto effemeridi propagate con JPL Horizons:

Tabella 22.8: Differenza posizione: AstDyn vs JPL (60 giorni)

Data	ΔX (km)	ΔY (km)	ΔZ (km)
2024-01-15	0.0	0.0	0.0
2024-01-25	0.3	0.2	0.1
2024-02-04	0.8	0.5	0.3
2024-02-14	1.5	0.9	0.6
2024-02-24	2.1	1.3	0.9
2024-03-05	2.7	1.7	1.2
2024-03-15	3.2	2.0	1.4

Differenza massima: **3.9 km** dopo 60 giorni (2.6×10^{-8} AU).

22.8 Covarianza e Incertezze

22.8.1 Matrice Covarianza Parametri

Matrice completa 6×6 nello spazio elementi orbitali.

Tabella 22.9: Matrice correlazione (elementi selezionati)

	a	e	i	Ω
a	1.000	0.923	0.012	0.008
e	0.923	1.000	0.018	0.011
i	0.012	0.018	1.000	0.342
Ω	0.008	0.011	0.342	1.000

Correlazioni chiave:

- Forte correlazione $a-e$ (0.923): Atteso, entrambi determinati da distanza radiale
- Moderata correlazione $i-\Omega$ (0.342): Elementi angolari debolmente accoppiati

22.8.2 Propagazione Incertezza Posizione

Propagare covarianza in avanti usando matrice transizione stato:

Tabella 22.10: Incertezza posizione vs tempo

Tempo (giorni)	σ_x (km)	σ_y (km)	σ_z (km)
0	18	12	8
10	35	23	15
20	67	45	29
30	118	79	52
40	189	126	83
50	278	186	122
60	385	257	169

Tasso crescita: Incertezza posizione cresce approssimativamente linearmente a ~ 6 km/giorno.

22.9 Metriche Prestazioni

22.9.1 Costo Computazionale

Tabella 22.11: Suddivisione timing (Intel i7-10700K, core singolo)

Operazione	Tempo (ms)	Percentuale
Parsing osservazioni	2.3	0.1%
Orbita iniziale (Gauss)	15.7	0.9%
Propagazione (4 iter)	1456.2	80.0%
Calcolo residui	234.5	12.9%
Operazioni matriciali	98.4	5.4%
Altro	12.9	0.7%
Totale	1820.0	100%

Collo bottiglia: Propagazione numerica domina (80%).

22.9.2 Uso Memoria

- **Memoria picco:** 12.4 MB
- **Dati osservazioni:** 0.8 MB
- **Matrici STM:** 4.6 MB

- **Workspace integrazione:** 6.2 MB
- **Altro:** 0.8 MB

22.10 Conclusioni

Il caso studio Pompeja dimostra:

1. **Workflow completo:** Da osservazioni grezze MPC a orbita raffinata con incertezze
2. **Accuratezza eccellente:** RMS residuo $0.658''$ comparabile a precisione osservazioni
3. **Accordo con OrbFit:** Differenze $< 10^{-7}$ AU validano implementazione
4. **Accordo con JPL:** 3.9 km su 60 giorni conferma accuratezza numerica
5. **Convergenza rapida:** 4 iterazioni tipiche con buon guess iniziale
6. **Prestazioni ragionevoli:** ~ 2 secondi per 100 osservazioni su CPU standard
7. **Pronto produzione:** Risultati adatti pubblicazione scientifica o pianificazione missioni

AstDyn gestisce con successo determinazione orbitale reale per asteroidi fascia principale.

Capitolo 23

Benchmark Prestazioni

23.1 Introduzione

Questo capitolo quantifica le prestazioni computazionali di AstDyn attraverso benchmark sistematici, confrontando velocita', accuratezza e uso risorse con altri strumenti di determinazione orbitale.

23.1.1 Ambiente Benchmark

Tutti i test eseguiti su hardware standardizzato:

- **CPU:** Intel Core i7-10700K @ 3.8 GHz (8 core, 16 thread)
- **RAM:** 32 GB DDR4-3200
- **OS:** Ubuntu 22.04.3 LTS (kernel Linux 6.2)
- **Compilatore:** GCC 11.4.0 con ottimizzazione -O3
- **Librerie:** Eigen 3.4.0, Boost 1.74

23.2 Prestazioni Propagazione Orbitale

23.2.1 Propagazione Singola

Propagare orbita Pompeja per 60 giorni con diversi integratori e tolleranze.

Tabella 23.1: Timing propagazione: arco 60 giorni

Integratore	Tolleranza	Passi	Tempo (ms)	Errore (km)	Passi/s
RKF78	10^{-10}	85	1.23	45	69,000
RKF78	10^{-12}	127	1.82	3.2	69,800
RKF78	10^{-14}	189	2.71	0.08	69,700

Osservazioni:

- Velocita' passo: $\sim 70,000$ passi/secondo (consistente tra tolleranze)
- Tempo scala linearmente con numero passi
- Tolleranza default 10^{-12} : buon bilanciamento accuratezza/velocita'

23.2.2 Propagazione Batch

Propagare 1000 asteroidi fascia principale diversi per 60 giorni ciascuno.

Tabella 23.2: Statistiche propagazione batch

Metrica	Min	Media	Max	Dev Std
Tempo per orbita (ms)	1.45	1.87	2.34	0.18
Passi integrazione	102	134	178	15
Passi rigettati	0	2.3	8	1.7

Tempo totale: 1.87 secondi per 1000 orbite = **1.87 ms per orbita**

Throughput: 535 orbite per secondo

23.2.3 Effetto Complessita' Modello Forze

Confronto timing con diversi modelli perturbazione.

Tabella 23.3: Timing vs modello forze (propagazione 60 giorni)

Modello Forze	Tempo (ms)	Relativo
Solo due corpi	1.12	1.0×
Sole + Giove	1.56	1.4×
Sole + Giove + Saturno	1.82	1.6×
Sole + tutti 8 pianeti	2.87	2.6×
Sole + pianeti + Luna	3.24	2.9×

23.3 Prestazioni Determinazione Orbitale

23.3.1 Timing Correzione Differenziale

Caso Pompeja: 100 osservazioni, arco 60 giorni.

Tabella 23.4: Suddivisione correzione differenziale		
Componente	Tempo (ms)	Percentuale
Parsing osservazioni	2.3	0.1%
Orbita iniziale (Gauss)	15.7	0.9%
Iterazione 1		
Propagazione (100×)	182.4	10.0%
Calcolo STM	234.5	12.9%
Residui	45.3	2.5%
Algebra lineare	23.8	1.3%
Iterazione 2		
Iterazione 3	485.3	26.7%
Iterazione 4	485.1	26.6%
Altro	12.9	0.7%
Totale	1820.0	100%

Collo bottiglia: Propagazione orbitale domina (80% del tempo).

23.3.2 Scalabilita' con Numero Osservazioni

Variare numero osservazioni (10-500), arco fisso 60 giorni.

Tabella 23.5: Prestazioni vs numero osservazioni				
Osservazioni	Tempo (s)	Iterazioni	Tempo/oss (ms)	RMS (arcsec)
10	0.18	4	18.0	0.823
20	0.36	4	18.0	0.745
50	0.91	4	18.2	0.687
100	1.82	4	18.2	0.658
200	3.65	4	18.3	0.642
500	9.15	5	18.3	0.635

Scalabilita': Quasi lineare—~18 ms per osservazione per iterazione.

23.4 Confronto con Altri Strumenti

23.4.1 OrbFit 5.0.5

Stesso caso test Pompeja su hardware identico.

Tabella 23.6: Timing AstDyn vs OrbFit

Operazione	AstDyn (ms)	OrbFit (ms)	Speedup
Parsing osservazioni	2.3	8.7	3.8×
Orbita iniziale	15.7	23.4	1.5×
Correzione differenziale	1820	2341	1.3×
Totale	1838	2373	1.29×

Risultato: AstDyn è 29% più veloce di OrbFit per questo caso.

23.4.2 Strumenti Python

Confronto con PyEphem e Skyfield (librerie Python).

Tabella 23.7: Confronto prestazioni linguaggio (propagazione 60 giorni)

Strumento	Linguaggio	Tempo (ms)	Relativo
AstDyn	C++17	1.82	1.0×
OrbFit	Fortran 90	2.34	1.3×
PyEphem	Python + C	2.87	1.6×
Skyfield	Python	12.4	6.8×
REBOUND	C + Python	2.15	1.2×

Conclusione: Implementazione C++ fornisce prestazioni migliori.

23.5 Uso Memoria

23.5.1 Allocazione Heap

Profilo uso memoria durante determinazione orbitale.

Tabella 23.8: Footprint memoria per componente

Componente	Dimensione (MB)	Percentuale
Dati osservazioni	0.8	6.5%
Matrici STM ($100 \times 6 \times 6$)	4.6	37.1%
Workspace integrazione	6.2	50.0%
Cache effemeridi	0.5	4.0%
Altro	0.3	2.4%
Total	12.4	100%

Uso picco: 12.4 MB—molto modesto per sistemi moderni.

23.5.2 Scalabilita' con Dimensione Problema

Memoria vs numero osservazioni.

Tabella 23.9: Uso memoria vs numero osservazioni

Osservazioni	Memoria (MB)	Per oss (KB)
10	7.2	720
50	9.8	196
100	12.4	124
500	34.7	69
1000	62.3	62

Scalabilita': Sub-lineare grazie overhead condiviso (effemeridi, stato integratore).

23.6 Sommario

Benchmark dimostrano:

1. **Velocita' propagazione:** 70,000 passi/secondo, 535 orbite/secondo
2. **Scalabilita':** Lineare con numero osservazioni e lunghezza arco
3. **Confronto OrbFit:** 29% piu' veloce
4. **Uso memoria:** 12.4 MB per 100 osservazioni
5. **Efficienza:** Competitivo con strumenti consolidati

AstDyn fornisce prestazioni eccellenti per determinazione orbitale produttiva.

Parte VI

Prospettive Future

Capitolo 24

Sviluppi Futuri

24.1 Introduzione

Questo capitolo delinea i miglioramenti pianificati e le direzioni di ricerca future per AstDyn. Sebbene l'attuale release v1.0 fornisca capacità di determinazione orbitale di livello produttivo, diverse estensioni potrebbero ampliare funzionalità e prestazioni.

24.2 Forze Non Gravitazionali

24.2.1 Pressione di Radiazione Solare

Stato: Implementazione parziale esistente (esempio nel Capitolo 20).

Pianificato:

- Integrazione completa nel framework dei modelli di forza
- Modello di ombreggiamento (occultazione Terra/Luna)
- Ri-irradiazione termica (effetto Yarkovsky)
- Stima dei parametri nella correzione differenziale

Schema di implementazione:

```
1 class SolarRadiationPressure : public ForceModel {  
2     public:  
3         SolarRadiationPressure(  
4             double area_mass_ratio ,
```

```

5         double reflectivity = 1.0,
6         bool include_yarkovsky = false
7     );
8
9     Vector3d acceleration(double t, const Vector3d& pos,
10                           const Vector3d& vel) const
11                           override;
12
13 // Per correzione differenziale
14 bool supports_partials() const override { return true;
15 }
16 std::pair<Matrix3d, Matrix3d> partials(...) const
17                           override;
18
19 private:
20     double area_mass_ratio_;
21     double reflectivity_;
22     bool include_yarkovsky_;
23
24     // Calcolo dell'ombra
25     double shadow_function(double t, const Vector3d& pos)
26                           const;
27 };

```

Impatto scientifico: Critico per piccoli NEA e tracciamento detriti spaziali.

24.2.2 Degassamento Cometario

Motivazione: Le comete esibiscono accelerazioni non gravitazionali dovute alla sublimazione di volatili.

Modello pianificato:

- Formulazione di Marsden: termini $A_1/r^2 + A_2/r^3 + A_3$
- Componenti radiale, trasversale e normale
- Curva di attività dipendente dalla temperatura

```

1 class CometaryOutgassing : public ForceModel {
2 public:

```

```

3     CometaryOutgassing(double A1, double A2, double A3);

4

5     Vector3d acceleration(double t, const Vector3d& pos,
6                           const Vector3d& vel) const
7         override {
8             Vector3d r_sun = pos; // Posizione eliocentrica
9             double r = r_sun.norm();
10
11            // Modello di Marsden
12            Vector3d radial = r_sun.normalized();
13            Vector3d transverse = /* calcola dalla velocita' */
14                ;
15            Vector3d normal = radial.cross(transverse);
16
17            double g = activity_function(r); // Curva di
18                attivita'
19
20        return g * (A1_ * radial + A2_ * transverse + A3_ *
21                    normal) / (r * r);
22    }
23
24
25 private:
26     double A1_, A2_, A3_;
27     double activity_function(double r) const;
28 };

```

Caso d'uso: Comete a lungo periodo, comete a breve periodo con degassamento significativo.

24.2.3 Relatività Generale

Limitazione attuale: Effetti post-newtoniani trascurati.

Pianificato: Correzioni relativistiche del primo ordine.

Formulazione:

$$\mathbf{a}_{\text{rel}} = \frac{GM_{\odot}}{c^2 r^3} \left[4 \frac{GM_{\odot}}{r} - v^2 \right] \mathbf{r} + 4 \frac{GM_{\odot}}{c^2 r^3} (\mathbf{r} \cdot \mathbf{v}) \mathbf{v}$$

Implementazione:

```
1 class RelativisticCorrection : public ForceModel {
2 public:
3     Vector3d acceleration(double t, const Vector3d& pos,
4                           const Vector3d& vel) const
5         override {
6
7     double r = pos.norm();
8     double v2 = vel.squaredNorm();
9     double rdotv = pos.dot(vel);
10
11
12     return (GM_SUN / (C * C * r * r * r)) *
13         (factor1 * pos + factor2 * vel);
14 }
15 };
```

Magnitudine: $\sim 10^{-8}$ m/s² a 1 AU— influenza significativamente Mercurio, trascurabile per asteroidi oltre Marte.

24.3 Propagazione delle Incertezze

24.3.1 Propagazione della Covarianza

Attuale: STM singola per iterazione.

Pianificato: Propagazione completa della matrice di covarianza con rumore di processo.

```
1 class CovariancePropagator {
2 public:
3     struct Result {
4         CartesianState mean_state;
5         Matrix6d covariance;
6     };
7
8     Result propagate_with_covariance(
9         const CartesianState& initial_state,
10        const Matrix6d& initial_covariance,
11        double target_epoch,
```

```

12     const Matrix6d& process_noise
13 );
14 }

```

Applicazione: Previsione dell'incertezza, probabilità di collisione.

24.3.2 Metodi Monte Carlo

Motivazione: Propagazione non lineare dell'incertezza.

Pianificato:

```

1 class MonteCarloUncertainty {
2 public:
3     struct Sample {
4         orbit::KeplerianElements elements;
5         double weight;
6     };
7
8     std::vector<Sample> generate_samples(
9         const orbit::KeplerianElements& nominal,
10        const Matrix6d& covariance,
11        size_t n_samples = 10000
12    );
13
14     std::vector<CartesianState> propagate_ensemble(
15         const std::vector<Sample>& samples,
16         double target_epoch
17     );
18
19     // Riepilogo statistico
20     struct Statistics {
21         CartesianState mean;
22         CartesianState median;
23         Matrix6d covariance;
24         double position_rms;
25     };
26
27     Statistics compute_statistics(
28         const std::vector<CartesianState>& ensemble

```

```
29     );
30 }
```

Caso d'uso: Evitamento collisioni, analisi di avvicinamenti ravvicinati.

24.4 Gestione degli Incontrti Ravvicinati

24.4.1 Tecniche di Regolarizzazione

Problema: Gli integratori standard hanno difficoltà con gli incontri planetari ravvicinati (< 0.1 AU).

Pianificato: Regolarizzazione di Kustaanheimo-Stiefel (KS) per rimozione delle singolarità.

Trasformazione KS:

$$\mathbf{r} = \mathbf{u}^T L \mathbf{u}, \quad d\tau = r dt$$

Trasforma il problema dei due corpi singolare in un oscillatore armonico regolare.

Schema di implementazione:

```
1 class KSRegularizedIntegrator : public IIntegrator {
2 public:
3     void integrate(double t0, double tf, std::vector<double>
4                   & y,
5                   const std::function<...>& derivs)
6                   override {
7 // Rileva avvicinamento ravvicinato
8         if (is_close_approach(y)) {
9             // Passa a coordinate KS
10            auto u = cartesian_to_ks(y);
11            // Integra nello spazio regolarizzato
12            integrate_ks(t0, tf, u);
13            // Trasforma indietro
14            y = ks_to_cartesian(u);
15        } else {
16            // Integrazione standard
17            standard_integrate(t0, tf, y, derivs);
18        }
19    }
```

18 }

Beneficio: Integrazione stabile attraverso sorvoli planetari.

24.4.2 Analisi di Incontrti Iperbolici

Funzionalità pianificate:

- Rilevamento automatico di incontri ravvicinati
- Parametri di targeting del piano B
- Calcolo della velocità e geometria dell'incontro
- Previsione degli elementi orbitali post-incontro

24.5 Elaborazione Parallela

24.5.1 Parallelizzazione OpenMP

Limitazione attuale: Solo single-thread.

Operazioni target:

1. Propagazione batch di orbite
2. Calcolo dei residui delle osservazioni
3. Campionamento Monte Carlo
4. Ricerca su griglia di parametri

Implementazione:

```

1 // Propagazione batch parallela
2 #pragma omp parallel for schedule(dynamic)
3 for (int i = 0; i < n_orbits; ++i) {
4     auto state = propagator.propagate(initial_states[i],
5                                         target_epoch);
6     results[i] = state;
7 }
8 // Calcolo parallelo dei residui nella correzione
9 differenziale

```

```

9 #pragma omp parallel for
10 for (int i = 0; i < n_observations; ++i) {
11     auto computed = compute_predicted_observation(obs[i]);
12     residuals[2*i] = obs[i].ra - computed.ra;
13     residuals[2*i+1] = obs[i].dec - computed.dec;
14 }
```

Speedup atteso: 6-7 \times su CPU 8-core per operazioni batch.

24.5.2 Accelerazione GPU

Obiettivo a lungo termine: CUDA/OpenCL per parallelismo massiccio.

Attività adatte:

- Incertezza Monte Carlo (10.000+ campioni)
- Generazione di tabelle di effemeridi
- Risoluzione batch di minimi quadrati

Sfida: Adattività della dimensione del passo di integrazione difficile su GPU.

24.6 Integratori Aggiuntivi

24.6.1 Integratori Simplettici

Motivazione: Conservazione dell'energia per studi di stabilità a lungo termine.

Pianificato: Mappa simplettica di Wisdom-Holman per sistemi gerarchici.

```

1 class SymplecticIntegrator : public IIIntegrator {
2 public:
3     SymplecticIntegrator(double fixed_step_size);
4
5     // Splitting dell'operatore:  $H = H_{\text{Kepler}} +$ 
6     //  $H_{\text{interaction}}$ 
7     void step(double t, std::vector<double>& y, ... )
8         override;
9
10 private:
11     void drift_step(std::vector<double>& y, double dt);
12     void kick_step(std::vector<double>& y, double dt);
```

```
11 };
```

Caso d'uso: Dinamica asteroidale su milioni di anni, stabilità di sistemi planetari.

24.6.2 Metodi Impliciti

Pianificato: Radau IIA per problemi rigidi (stiff).

Vantaggio: Stabilità incondizionata, buono per sistemi strettamente legati.

Svantaggio: Richiede calcolo dello Jacobiano, più lento per passo.

24.7 Binding Python

24.7.1 Interfaccia pybind11

Obiettivo: Accesso Python senza soluzione di continuità ad AstDyn.

API pianificata:

```
1 import astdyn
2
3 # Crea elementi orbitali
4 elem = astdyn.KeplerianElements(
5     a=2.7436, e=0.0624, i=11.74,
6     Omega=339.86, omega=258.03, M=45.32,
7     epoch=2460000.5
8 )
9
10 # Configura propagatore
11 eph = astdyn.SPICEEphemeris("de440.bsp")
12 forces = astdyn.PointMassGravity(eph, ["JUPITER", "SATURN"])
13
14 integrator = astdyn.RKF78(tolerance=1e-12)
15 prop = astdyn.Propagator(integrator, forces, eph)
16
17 # Propaga
18 state0 = elem.to_cartesian()
19 state60 = prop.propagate(state0, 2460060.5)
20 print(f"Posizione: {state60.position}")
```

```
21 print(f"Velocita': {state60.velocity}")  
22  
23 # Determinazione orbitale  
24 observations = astdyn.read_mpc_file("observations.txt")  
25 corrector = astdyn.DifferentialCorrector(prop)  
26 result = corrector.solve(elem, observations)  
27  
28 print(f"Residuo RMS: {result.rms_residual} arcosecondi")  
29 print(f"Convergenza: {result.converged}")
```

Integrazione: Notebook Jupyter, array NumPy, grafici matplotlib.

24.7.2 Distribuzione del Pacchetto

Pianificato:

- Pacchetto PyPI: pip install astdyn
- Pacchetto Conda: conda install -c conda-forge astdyn
- Wheel pre-compilate per Linux, macOS, Windows

24.8 Integrazione con Machine Learning

24.8.1 Modelli Surrogati con Reti Neurali

Direzione di ricerca: Addestrare reti neurali per approssimare calcoli costosi.

Applicazioni potenziali:

1. **Propagazione veloce:** NN approssima l'integratore per applicazioni in tempo reale
2. **Rilevamento outlier:** ML identifica automaticamente osservazioni errate
3. **Orbita iniziale:** NN fornisce una migliore stima IOD da osservazioni limitate

Concetto preliminare:

```
1 class NeuralPropagator : public IIntegrator {  
2 public:  
3     NeuralPropagator(const std::string& model_file);
```

```

4
5   // Usa NN per propagazione a breve termine
6   CartesianState propagate(const CartesianState& initial,
7                           double dt) {
8     if (dt < 10.0) { // Usa NN per archi brevi
9       return nn_predict(initial, dt);
10    } else { // Fallback a integrazione numerica
11      return numerical_propagate(initial, dt);
12    }
13  }
14
15 private:
16   NeuralNetwork model_;
17 };

```

Sfida: Garantire garanzie di accuratezza per uso scientifico.

24.9 Osservazioni Avanzate

24.9.1 Osservazioni Radar

Pianificato: Supporto per misure di distanza e velocità radiale.

```

1 struct RadarObservation {
2   double epoch;
3   double range;           // km
4   double range_rate;     // km/s
5   double sigma_range;
6   double sigma_range_rate;
7   std::string station_code;
8
9   Vector3d observer_position() const;
10 };

```

Integrazione: Aggiungi residui radar ai minimi quadrati:

$$\chi^2 = \sum_i \frac{(\rho_i^{\text{obs}} - \rho_i^{\text{comp}})^2}{\sigma_{\rho,i}^2} + \frac{(\dot{\rho}_i^{\text{obs}} - \dot{\rho}_i^{\text{comp}})^2}{\sigma_{\dot{\rho},i}^2}$$

Beneficio: Accuratezza in distanza di ordini di grandezza superiore rispetto all'ottico.

24.9.2 Astrometria Gaia

Pianificato: Supporto nativo per osservazioni del satellite Gaia.

Funzionalità:

- Misure along-scan e across-scan
- Sistema di riferimento Gaia (ICRF3)
- Correzioni di parallasse
- Tempo luce e aberrazione a livello di μas

24.10 Servizio Web / Deployment Cloud

24.10.1 API RESTful

Visione: Servizio di determinazione orbitale basato su cloud.

Endpoint pianificati:

POST /api/v1/propagate

Body: { "elements": [...], "target_epoch": 2460100.5 }
Returns: { "state": [...], "elapsed_ms": 1.82 }

POST /api/v1/orbit_determination

Body: { "observations": [...], "method": "differential_correction" }
Returns: { "elements": [...], "rms": 0.658, "iterations": 4 }

GET /api/v1/ephemeris?object=pompeja&start=2460000&end=2460100&step=1

Returns: [{ "epoch": 2460000.5, "position": [...], ... }, ...]

Stack tecnologico:

- Backend: Servizio C++ con wrapper REST
- Coda: Redis per gestione job
- Database: PostgreSQL per archiviazione risultati
- Container: Deployment Docker

24.10.2 Interfaccia Web

Funzionalità:

- Caricamento file osservazioni MPC
- Visualizzazione interattiva orbita (3D)
- Download risultati (CSV, JSON, formato OrbFit)
- Confronto con JPL Horizons
- Interfaccia elaborazione batch

24.11 Integrazione Pipeline Dati

24.11.1 Elaborazione Automatizzata di Survey

Obiettivo: Processare automaticamente flussi dati LSST/Pan-STARRS.

Pipeline:

1. Ingestione: Ricezione nuove osservazioni da survey
2. Matching: Collegamento a oggetti noti o rilevamento di nuovi
3. IOD: Orbita rapida per nuove rilevazioni
4. Raffinamento: Correzione differenziale con dati d'archivio
5. Pubblicazione: Aggiornamento database elementi orbitali
6. Allerta: Segnalazione oggetti interessanti (NEA, orbite inusuali)

Scalabilità: Elaborazione di oltre 1000 oggetti per notte.

24.12 Modelli di Errore Migliorati

24.12.1 Stima Robusta

Attuale: Minimi quadrati assume errori gaussiani.

Pianificato: Perdita di Huber e minimi quadrati ripesati iterativamente.

```
1 class RobustDifferentialCorrector : public
2     DifferentialCorrector {
3
4     public:
5         Result solve(...) override {
6             // Minimi quadrati iniziali
7             auto result = standard_solve(...);
8
9             // Iterazione robusta
10            for (int iter = 0; iter < max_robust_iters; ++iter)
11            {
12                // Calcola pesi basati sui residui
13                update_weights_huber(result.residuals);
14
15                // Minimi quadrati pesati
16                result = weighted_solve(...);
17            }
18
19            return result;
20        }
21    };
22};
```

Beneficio: Riduzione automatica del peso degli outlier.

24.13 Supporto Multi-Piattaforma

24.13.1 Build WebAssembly

Obiettivo: Eseguire AstDyn nel browser web.

Casi d'uso:

- Strumenti educativi (calcolatore orbitale interattivo)
- Visualizzazione orbita lato client
- Nessun server richiesto per calcoli semplici

Build:

```
1 emcc -O3 -s WASM=1 -s ALLOW_MEMORY_GROWTH=1 \
2     astdyn.cpp -o astdyn.js
```

24.13.2 Piattaforme Mobile

Pianificato: Librerie native iOS e Android.

Applicazioni:

- App di pianificazione osservativa
- Tracciamento satellitare in tempo reale
- App educative di astronomia

24.14 Roadmap di Sviluppo

24.14.1 Versione 1.1 (Q2 2026)

Funzionalità prioritarie:

- Parallelizzazione OpenMP
- Binding Python (pybind11)
- Pressione di radiazione solare
- Supporto osservazioni radar

24.14.2 Versione 1.2 (Q4 2026)

Capacità estese:

- Propagazione della covarianza
- Incertezza Monte Carlo
- Modello di degassamento cometario
- Integratore simplettico

24.14.3 Versione 2.0 (2027)

Miglioramenti maggiori:

- Gestione incontri ravvicinati (regolarizzazione KS)
- Correzioni di relatività generale

- Accelerazione GPU (CUDA)
- Deployment servizio web

24.15 Contributi della Comunità

24.15.1 Sviluppo Open Source

AstDyn accoglie i contributi della comunità:

Repository GitHub: <https://github.com/user/astdyn>

Arearie di contribuzione:

- Nuovi integratori (Dormand-Prince, Radau)
- Parser aggiuntivi (JPL, SPICE SPK)
- Modelli di forza (forze di marea, GR)
- Miglioramenti alla documentazione
- Test case e validazione
- Ottimizzazioni delle prestazioni

Linee guida: Vedi CONTRIBUTING.md nel repository.

24.15.2 Citazione di AstDyn

Se utilizzate AstDyn nella ricerca, si prega di citare:

```
@software{astdyn2025,
  author = {Bigi, Michele and Contributors},
  title = {AstDyn: Modern C++ Library for Asteroid Orbit Determination},
  year = {2025},
  version = {1.0.0},
  url = {https://github.com/user/astdyn}
}
```

24.16 Direzioni di Ricerca

24.16.1 Algoritmi Innovativi

Argomenti di ricerca futura:

1. **IOD con deep learning:** Reti neurali per orbita iniziale da 2 osservazioni
2. **Filtro di Kalman:** Determinazione orbitale sequenziale
3. **Metodi bayesiani:** Soluzioni orbitali probabilistiche
4. **Probabilità di collisione:** Monte Carlo veloce per analisi di congiunzione
5. **Tracciamento multi-oggetto:** Determinazione orbitale simultanea per più asteroidi

24.16.2 Applicazioni Interdisciplinari

Oltre gli asteroidi:

- **Tracciamento detriti spaziali:** Determinazione orbite LEO/GEO
- **Asteroidi binari:** Dinamica di orbite mutue
- **Lune planetarie:** Determinazione orbite satellitari
- **Transiti esopianetari:** Analisi temporale

24.17 Riepilogo

Gli sviluppi pianificati per AstDyn includono:

1. **Fisica:** Forze non gravitazionali, relatività, incontri ravvicinati
2. **Algoritmi:** Propagazione incertezze, stima robusta, nuovi integratori
3. **Prestazioni:** OpenMP, accelerazione GPU, deployment cloud
4. **Interfacce:** Binding Python, servizio web, piattaforme mobile
5. **Dati:** Osservazioni radar, astrometria Gaia, pipeline survey
6. **Comunità:** Contributi open source, collaborazioni di ricerca

Questi miglioramenti espanderanno le capacità di AstDyn mantenendo i principi di design fondamentali di accuratezza, affidabilità e facilità d'uso.

Contributi e suggerimenti sono benvenuti. Visita il repository GitHub per partecipare allo sviluppo di AstDyn.

Riferimenti

24.18 Libri di Testo

24.18.1 Meccanica Celeste

- **Murray & Dermott** (1999). *Solar System Dynamics*. Cambridge University Press. ISBN: 978-0521575973. Trattazione completa dinamica sistema solare.
- **Danby, J.M.A.** (1988). *Fundamentals of Celestial Mechanics*. Willmann-Bell. ISBN: 978-0943396200. Introduzione classica alla meccanica celeste.
- **Bate, Mueller, White** (1971). *Fundamentals of Astrodynamics*. Dover Publications. ISBN: 978-0486600610. Testo standard per ingegneria astronautica.
- **Valtonen & Karttunen** (2006). *The Three-Body Problem*. Cambridge University Press. ISBN: 978-0521852241. Trattazione approfondita problema N-corpi.

24.18.2 Metodi Numerici

- **Hairer, Norsett, Wanner** (1993). *Solving Ordinary Differential Equations I*. Springer. ISBN: 978-3540566700. Riferimento per integrazione numerica EDO.
- **Press et al.** (2007). *Numerical Recipes: The Art of Scientific Computing*, 3rd ed. Cambridge University Press. ISBN: 978-0521880688.

24.19 Articoli Scientifici

24.19.1 Determinazione Orbitale

- **Milani & Gronchi** (2010). *Theory of Orbit Determination*. Cambridge University Press. ISBN: 978-0521873895. Trattazione teorica completa.
- **Gauss, C.F.** (1809). *Theoria Motus Corporum Coelestium*. Metodo originale Gauss per IOD.
- **Carpino, Milani, Chesley** (2003). "Error statistics of asteroid optical astrometric observations". *Icarus* 166(2):248-270.

24.19.2 Sistemi Temporali

- **Seidelmann, P.K.** (1992). *Explanatory Supplement to the Astronomical Almanac*. University Science Books. ISBN: 978-0935702682.
- **IAU SOFA** (2021). "Standards of Fundamental Astronomy". <http://www.iausofa.org/>

24.20 Software e Strumenti

24.20.1 Software Determinazione Orbitale

- **OrbFit**: <http://adams.dm.unipi.it/orbfit/> - Software determinazione orbitale by Milani et al.
- **Find_Orb**: https://www.projectpluto.com/find_orb.htm - Software IOD e orbit determination by Project Pluto.
- **REBOUND**: <https://rebound.readthedocs.io/> - Framework integratore N-corpi.

24.20.2 Effemeridi

- **JPL Horizons**: <https://ssd.jpl.nasa.gov/horizons/> - Sistema effemeridi online NASA JPL.
- **SPICE Toolkit**: <https://naif.jpl.nasa.gov/naif/toolkit.html> - Libreria NASA per effemeridi e geometria spaziale.

- **JPL DE440/DE441:** Effemeridi planetarie ad alta precisione (2021).

24.20.3 Osservazioni

- **Minor Planet Center:** <https://minorplanetcenter.net/> - Database centrale osservazioni asteroidi.
- **AstDyS:** <https://newton.spacedys.com/astdys/> - Sistema dinamica asteroidi Pisa.

24.21 Standard e Convenzioni

- **IAU Resolutions:** Risoluzioni International Astronomical Union per sistemi riferimento e costanti.
- **IERS Conventions (2010).** "IERS Conventions (2010)". IERS Technical Note 36.
- **MPC Observation Format:** Formato standard 80-colonne per osservazioni astrometriche.

24.22 Risorse Online

- **AstDyn Documentation:** <https://github.com/user/astdyn> - Documentazione completa e esempi.
- **Eigen Library:** <https://eigen.tuxfamily.org/> - Libreria algebra lineare C++.
- **Boost Libraries:** <https://www.boost.org/> - Librerie utility C++.

Appendici

24.23 Costanti Fisiche

24.23.1 Costanti Fondamentali

Tabella 24.1: Costanti fisiche fondamentali

Costante	Valore	Unità
Velocità luce c	299,792,458	m/s
Costante gravitazionale G	6.67430×10^{-11}	$\text{m}^3 \text{kg}^{-1} \text{s}^{-2}$
Unità astronomica AU	$1.495978707 \times 10^{11}$	m
Parsec pc	3.0857×10^{16}	m
Anno giuliano	365.25	giorni
Secolo giuliano	36,525	giorni

24.23.2 Parametri Sistema Solare

Tabella 24.2: Masse planetarie (parametri GM in $\text{AU}^3/\text{giorno}^2$)

Corpo	GM	Massa (M_{\odot})
Sole	2.959122×10^{-4}	1.0
Mercurio	4.9125×10^{-11}	1.66×10^{-7}
Venere	7.2435×10^{-10}	2.45×10^{-6}
Terra	8.8877×10^{-10}	3.00×10^{-6}
Luna	1.0931×10^{-11}	3.69×10^{-8}
Marte	9.5496×10^{-11}	3.23×10^{-7}
Giove	2.8253×10^{-7}	9.55×10^{-4}
Saturno	8.4597×10^{-8}	2.86×10^{-4}
Urano	1.2920×10^{-8}	4.37×10^{-5}
Nettuno	1.5244×10^{-8}	5.15×10^{-5}

24.24 Elementi Orbitali Planetari

24.24.1 Elementi Medi J2000.0

Tabella 24.3: Elementi orbitali planetari (epoca J2000.0)

Pianeta	a (AU)	e	i ($^{\circ}$)	Ω ($^{\circ}$)	ω ($^{\circ}$)	L ($^{\circ}$)
Mercurio	0.387	0.206	7.00	48.3	29.1	252.3
Venere	0.723	0.007	3.39	76.7	54.9	181.9
Terra	1.000	0.017	0.00	—	288.1	100.5
Marte	1.524	0.093	1.85	49.6	286.5	355.4
Giove	5.203	0.048	1.30	100.5	273.9	34.4
Saturno	9.537	0.054	2.49	113.7	339.4	50.1
Urano	19.191	0.047	0.77	74.0	96.7	314.1
Nettuno	30.069	0.009	1.77	131.8	273.2	304.3

24.25 Conversioni Unità

24.25.1 Conversioni Angolari

Tabella 24.4: Fattori conversione angolare

Da	A	Fattore
radianti	gradi	$180/\pi = 57.2958$
gradi	radianti	$\pi/180 = 0.0174533$
radianti	arcosecondi	206,265
arcosecondi	radianti	4.8481×10^{-6}
ore	gradi	15
gradi	ore	$1/15 = 0.0666667$

24.25.2 Conversioni Temporali

Tabella 24.5: Fattori conversione temporale

Da	A	Fattore
giorni	secondi	86,400
secondi	giorni	1.15741×10^{-5}
anni giuliani	giorni	365.25
secoli giuliani	giorni	36,525
MJD	JD	+2,400,000.5

24.25.3 Conversioni Distanza

Da	A	Fattore
AU	km	149,597,871
AU	m	1.496×10^{11}
km	AU	6.6846×10^{-9}
parsec	AU	206,265
anno luce	AU	63,241

24.26 Formule Utili

24.26.1 Problema Due Corpi

Energia specifica:

$$\epsilon = \frac{v^2}{2} - \frac{\mu}{r} = -\frac{\mu}{2a}$$

Momento angolare specifico:

$$h = rv \cos \gamma = \sqrt{\mu a (1 - e^2)}$$

Periodo orbitale:

$$P = 2\pi \sqrt{\frac{a^3}{\mu}}$$

Velocità circolare:

$$v_c = \sqrt{\frac{\mu}{r}}$$

Velocità di fuga:

$$v_e = \sqrt{\frac{2\mu}{r}}$$

24.26.2 Anomalie

Anomalia media:

$$M = n(t - t_0) = \sqrt{\frac{\mu}{a^3}}(t - t_0)$$

Equazione Keplero:

$$M = E - e \sin E$$

Anomalia vera da eccentrica:

$$\tan \frac{f}{2} = \sqrt{\frac{1+e}{1-e}} \tan \frac{E}{2}$$

24.26.3 Trasformazioni Coordinate

Rotazione asse x:

$$R_x(\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta \\ 0 & \sin \theta & \cos \theta \end{pmatrix}$$

Rotazione asse z:

$$R_z(\theta) = \begin{pmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

24.27 Codici Osservatorio MPC

24.27.1 Osservatori Principali

Tabella 24.7: Codici osservatorio MPC selezionati

Codice	Nome	Localita'
F51	Pan-STARRS 1	Haleakala, Hawaii, USA
G96	Mt. Lemmon Survey	Arizona, USA
703	Catalina Sky Survey	Arizona, USA
691	Steward Observatory	Arizona, USA
568	Mauna Kea	Hawaii, USA
J75	OCA-DLR Survey	Germania
C51	Pan-STARRS 2	Haleakala, Hawaii, USA
I41	Palomar Mountain ZTF	California, USA
V00	ATLAS-MLO	Mauna Loa, Hawaii, USA

24.28 Acronimi e Abbreviazioni

Tabella 24.8: Acronimi comuni

Acronimo	Significato
API	Application Programming Interface
AU	Astronomical Unit (Unita' Astronomica)
CCD	Charge-Coupled Device
DE	Development Ephemeris (JPL)
EDO	Equazioni Differenziali Ordinarie
IAU	International Astronomical Union
ICRF	International Celestial Reference Frame
IOD	Initial Orbit Determination
JPL	Jet Propulsion Laboratory (NASA)
JD	Julian Day (Giorno Giuliano)
MJD	Modified Julian Day
MPC	Minor Planet Center
NEA	Near-Earth Asteroid
RMS	Root Mean Square
SPICE	Spacecraft Planet Instrument C-matrix Events
STM	State Transition Matrix
TAI	Temps Atomique International
TDB	Barycentric Dynamical Time
TT	Terrestrial Time
UTC	Coordinated Universal Time

24.29 Convenzioni Notazione

24.29.1 Vettori e Matrici

- Vettori: \mathbf{r} , \mathbf{v} (grassetto minuscolo)
- Matrici: \mathbf{R} , \mathbf{M} (grassetto maiuscolo)
- Versori: $\hat{\mathbf{x}}$, $\hat{\mathbf{e}}_i$ (cappello)
- Componenti: r_x , v_i (pedice)
- Norma: $|\mathbf{r}| = r$ (barre verticali o senza grassetto)

24.29.2 Elementi Orbitali

- a : Semiasse maggiore

- e : Eccentricità'
- i : Inclinazione
- Ω : Longitudine nodo ascendente
- ω : Argomento perielio
- M : Anomalia media
- E : Anomalia eccentrica
- f : Anomalia vera

24.29.3 Sistemi Coordinate

- ICRF: International Celestial Reference Frame
- Eclittica: Piano orbitale Terra
- Equatore: Piano equatoriale Terra
- (x, y, z) : Coordinate cartesiane
- (α, δ) : Ascensione retta, declinazione
- (λ, β) : Longitudine, latitudine eclittica