Memoria del Proyecto de Taller Transversal I

Ekaitz Arriola Garcia

27 de junio de 2025

Índice

1 Introducción

Este proyecto aborda la migración sistemática de una librería de astrodynamica desde MATLAB hacia C++, preservando la precisión numerica y funcionalidad del código original. La librería implementa algoritmos fundamentales para determinación orbital, propagación de trayectorias satelitales y conversiones entre sistemas de coordenadas, utilizando una clase Matrix personalizada que replica el comportamiento matricial nativo de MATLAB.
El desarrollo incluye la traducción de funciones críticas como Gibbs para determinación orbital inicial, cálculo de elementos orbitales keplerianos, y transformaciones geodésicas, manteniendo compatibilidad directa con los algoritmos originales mediante indexación base-1 y operaciones vectoriales equivalentes. La implementación garantiza la integridad de los resultados através de un framework de testing exhaustivo que valida cada función traducida contra casos conocidos y ejemplos de referencia en mecánica orbital.

Todos los recursos nombrados en la memoria se podrá acceder desde Github.

2 Tiempos del proyecto

Tarea	Tiempo estimado	Tiempo real	Comentario
Matrix	**190 min**	**145 min**	
zeros(row, column)	5 min	5 min	Tiempo estimado basado en experiencia previa en tareas similares.
operator (Matrix)	5 min	5 min	Tiempo estimado basado en experiencia previa en tareas similares.
operator (Matrix)	0 min	0 min	No completado.
operator / (Matrix)	5 min	5 min	Tiempo estimado basado en experiencia previa en tareas similares.
operator () (row, column)	5 min	5 min	Tiempo estimado basado en experiencia previa en tareas similares.
operator + (Matrix)	5 min	5 min	Tiempo estimado basado en experiencia previa en tareas similares.
inv(Matrix)	5 min	5 min	Tiempo estimado basado en experiencia previa en tareas similares.
transpose(Matrix)	5 min	5 min	Tiempo estimado basado en experiencia previa en tareas similares.
eye(n)	5 min	5 min	Tiempo estimado basado en experiencia previa en tareas similares.
operator = (Matrix)	10 min	5 min	No me acordaba de como definir este operador.
Matrix(row, column)	5 min	5 min	Tiempo estimado basado en experiencia previa en tareas similares.
operator * (Matrix)	5 min	5 min	Tiempo estimado basado en experiencia previa en tareas similares.
operator + (Matrix)	5 min	5 min	Tiempo estimado basado en experiencia previa en tareas similares.
operator - (Matrix)	5 min	5 min	Tiempo estimado basado en experiencia previa en tareas similares.
operator * (double)	5 min	5 min	Tiempo estimado basado en experiencia previa en tareas similares.
operator + (double)	5 min	5 min	Tiempo estimado basado en experiencia previa en tareas similares.
operator - (double)	5 min	5 min	Tiempo estimado basado en experiencia previa en tareas similares.
Matrix(n)	5 min	5 min	Tiempo estimado basado en experiencia previa en tareas similares.

Tarea	**Tiempo estimado**	**Tiempo real**	**Comentario**
operator () (n)	5 min	5 min	Tiempo estimado basado en experiencia previa en tareas similares.
zeros(n)	5 min	5 min	Tiempo estimado basado en experiencia previa en tareas similares.
norm	10 min	5 min	Tiempo estimado basado en experiencia previa en tareas similares.
dot	10 min	5 min	Tiempo estimado basado en experiencia previa en tareas similares.
cross	10 min	5 min	Tiempo estimado basado en experiencia previa en tareas similares.
extract_vector	10 min	5 min	Tiempo estimado basado en experiencia previa en tareas similares.
union_vector	10 min	10 min	Gestionar vecticales y horizontales.
extract_row	10 min	5 min	Tiempo estimado basado en experiencia previa en tareas similares.
extract_column	10 min	5 min	Tiempo estimado basado en experiencia previa en tareas similares.
assign_row	10 min	5 min	Tiempo estimado basado en experiencia previa en tareas similares.
assign_column	10 min	5 min	Tiempo estimado basado en experiencia previa en tareas similares.
Iteration 1	**240 min**	**350 min**	
AccelPointMass	15 min	20 min	Tiempo estimado basado en expreriencia previa, junto a realización de los test.
Cheb3D	15 min	20 min	Tiempo estimado basado en expreriencia previa, junto a realización de los test.
EccAnom	15 min	20 min	Tiempo estimado basado en expreriencia previa, junto a realización de los test.
Frac	15 min	20 min	Tiempo estimado basado en expreriencia previa, junto a realización de los test.

Tarea	**Tiempo estimado**	**Tiempo real**	**Comentario**
MeanObliquity	15 min	20 min	Tiempo estimado basado en expreriencia previa, junto a realización de los test.
Mjday	15 min	20 min	Tiempo estimado basado en expreriencia previa, junto a realización de los test.
Mjday_TBD	15 min	20 min	Tiempo estimado basado en expreriencia previa, junto a realización de los test.
Position	15 min	20 min	Tiempo estimado basado en expreriencia previa, junto a realización de los test.
R_x	5 min	10 min	Tiempo estimado basado en expreriencia previa, junto a realización de los test.
R_y	5 min	10 min	Tiempo estimado basado en expreriencia previa, junto a realización de los test.
R_z	5 min	10 min	Tiempo estimado basado en expreriencia previa, junto a realización de los test.
SAT_Const	0 min	0 min	Ya estaba hecho.
sign_	15 min	20 min	Tiempo estimado basado en expreriencia previa, junto a realización de los test.
timediff	15 min	20 min	Tiempo estimado basado en expreriencia previa, junto a realización de los test.
AzElPa	15 min	20 min	Tiempo estimado basado en expreriencia previa, junto a realización de los test.
IERS	15 min	20 min	Tiempo estimado basado en expreriencia previa, junto a realización de los test.

Tarea	**Tiempo estimado**	**Tiempo real**	**Comentario**
Legendre	15 min	20 min	Tiempo estimado basado en expreriencia previa, junto a realización de los test.
NutAngles	15 min	20 min	Tiempo estimado basado en expreriencia previa, junto a realización de los test.
TimeUpdate	15 min	20 min	Tiempo estimado basado en expreriencia previa, junto a realización de los test.
Iteration 2	**160 min**	**160 min**	
AccelHarmonic	20 min	20 min	Tiempo estimado basado en expreriencia previa, junto a realización de los test.
EqnEquinox	20 min	20 min	Tiempo estimado basado en expreriencia previa, junto a realización de los test.
JPL_Eph_DE430	20 min	20 min	Tiempo estimado basado en expreriencia previa, junto a realización de los test.
LTC	20 min	20 min	Tiempo estimado basado en expreriencia previa, junto a realización de los test.
NutMatrix	20 min	20 min	Tiempo estimado basado en expreriencia previa, junto a realización de los test.
PoleMatrix	20 min	20 min	Tiempo estimado basado en expreriencia previa, junto a realización de los test.
PrecMatrix	20 min	20 min	Tiempo estimado basado en expreriencia previa, junto a realización de los test.
gmst	20 min	20 min	Tiempo estimado basado en expreriencia previa, junto a realización de los test.

Tarea	**Tiempo estimado**	**Tiempo real**	**Comentario**
Iteration 3	**120 min**	**220 min**	
gast	20 min	20 min	Tiempo estimado basado en expreriencia previa, junto a realización de los test.
MeasUpdate	20 min	20 min	Tiempo estimado basado en expreriencia previa, junto a realización de los test.
G_AccelHarmonic	20 min	40 min	Tiempo estimado basado en expreriencia previa, junto a realización de los test.
GHAMatrix	20 min	20 min	Tiempo estimado basado en expreriencia previa, junto a realización de los test.
Accel	20 min	60 min	Tiempo estimado basado en expreriencia previa, junto a realización de los test.
VarEqn	20 min	60 min	Tiempo estimado basado en expreriencia previa, junto a realización de los test.
Iteration 4	**90 min**	**140 min**	
DEInteg	40 min	60 min	Tiempo estimado basado en expreriencia previa, junto a realización de los test.
EKF_GEOS3	50 min	80 min	Tiempo estimado basado en expreriencia previa, junto a realización de los test.

Tarea	Tiempo estimado	Tiempo real	Comentario
Extra	**140 min**	**210 min**	
Geodetic	20 min	20 min	Tiempo estimado basado en expreriencia previa, junto a realización de los test.
angl	20 min	20 min	Tiempo estimado basado en expreriencia previa, junto a realización de los test.
unit	20 min	20 min	Tiempo estimado basado en expreriencia previa, junto a realización de los test.
anglesg	20 min	50 min	Tiempo estimado basado en expreriencia previa, junto a realización de los test.
elements	20 min	40 min	Tiempo estimado basado en expreriencia previa, junto a realización de los test.
gibbs	20 min	30 min	Tiempo estimado basado en expreriencia previa, junto a realización de los test.
hgibbs	20 min	30 min	Tiempo estimado basado en expreriencia previa, junto a realización de los test.
Total	**960 min**	**1245 min**	

2.1 Horas totales reales

20 horas y 45 minutos en total.

3 Tiempos de ejecución

Implementación	Matlab (original)	C++
Tiempo (segundos)	511.273	1.142

Cuadro 1: Comparativa de tiempos de ejecución EKF_GEOS3

Matlab es muy lento. La implementación en cpp funciona casi 500 veces más veloz. Esto se debe a que Matlab es un lenguaje interpretado que ejecuta código a través de su máquina virtual, mientras que C++ se compila directamente a código máquina.

4 Problemas y dificultades durante el proyecto

Estos fueron los problemas con más ocurrencias y más mayores dificultades durante la realización del proyecto:

4.1 Indexación base-1 vs base-0:

La diferencia de indexación entre MATLAB (base 1) y C++ (base 0) causó errores sutiles pero críticos cuando me despistaba. La clase Matrix funciona en base 1 de indexación, por lo que esos errores me los ahorré, pero en el resto del código tuve que tener cuidado de ello.

4.2 Orientación de vectores:

MATLAB maneja vectores fila y columna de forma implícita, pero en C++ hay que ser explícito. Los problemas de concatenación vertical vs horizontal no fueron muy dificultosos, pero molestaron bastante.

4.3 Propagación silenciosa de errores:

El mayor problema fue que al tener funciones que dependen unas de otras (como `anglesg` que llama a `gibbs`, que usa `angl` y `unit`), un pequeño error en una función base se propagaba en cascada a través de toda la cadena de cálculos. Los errores se acumulaban y se manifestaban como resultados aparentemente correctos pero numericamente incorrectos varios niveles arriba, haciendo el debugging extremadamente complejo porque el fallo real estaba enterrado en funciones de bajo nivel. A nivel test unitario, quizas pasaba con mucha permisividad, y el error se va acumulando.

4.4 Precisión numérica:

Diferencias mínimas en cálculos de punto flotante entre MATLAB y C++ causaron divergencias en algoritmos iterativos.

4.5 Condiciones de error:

Las cadenas de error exactas (' ok' vs 'not coplanar') tenían que coincidir al caracter, incluyendo espacios. Los algoritmos como `gibbs` dependían de estas comparaciones exactas.

4.6 Testing y validación:

Verificar que cada función traducida produjera exactamente los mismos resultados que MATLAB requirió crear casos de prueba exhaustivos y frameworks de comparación robustos.

5 Test unitarios y de integración

El proyecto implementa un framework de testing robusto para validar la correcta migración desde MATLAB. Cada función traducida incluye tests unitarios exhaustivos que verifican múltiples escenarios:

Tests unitarios: Se declaran directamente en el mismo archivo .cpp de cada función, validando casos típicos (órbitas LEO, GEO, elípticas), casos límite (vectores colineales, magnitudes extremas) y verificación de propiedades físicas (conservación de energía, perpendicularidad en órbitas circulares). Se utilizan assertions con tolerancias numéricas apropiadas para comparar resultados contra valores de referencia conocidos. El framework utiliza macros REGISTER_TEST para ejecutar automáticamente todos los tests.

Tests de integración: Verifican el comportamiento de sistemas completos como EKF_GEOS3_test, que valida la implementación completa del filtro de Kalman extendido aplicado al satelite GEOS3, integrando determinación orbital, propagación y estimación de estados. También incluyen funciones compuestas como anglesg, que combina transformaciones de coordenadas, el método de Gibbs y cálculos iterativos. Estos tests detectan la propagación de errores a través de la cadena de dependencias y validan que los resultados finales coincidan con los obtenidos en MATLAB.

La estrategia de testing fue fundamental para detectar errores sutiles causados por diferencias entre MATLAB y C++ en precisión numérica y manejo de casos especiales.

6 Mejoras en el código

Para garantizar la calidad y rendimiento se implementó un proceso sistemático de análisis y optimización del código mediante herramientas especializadas de profiling y análisis estático.

6.1 Metodología de análisis

El proceso de mejora se ejecutó en tres fases diferenciadas:

1. **Análisis estático pre-ejecución**: Utilizando CppCheck para identificar problemas potenciales en el código fuente sin necesidad de compilación o ejecución.

2. **Análisis de rendimiento**: Mediante Gprof para identificar cuellos de botella y patrones de uso de funciones durante la ejecución real del programa.

3. **Análisis de memoria**: Con Valgrind para detectar memory leaks, buffer overflows y otros problemas relacionados con la gestión dinámica de memoria.

Cada herramienta proporcionó perspectivas complementarias sobre diferentes aspectos del código, permitiendo una evaluación integral de la calidad del software desarrollado.

6.2 Resultados de los análisis

Los informes detallados de cada herramienta se presentan a continuación, incluyendo las métricas específicas, problemas identificados y mejoras implementadas.

6.2.1. Informe de Análisis del Proyecto con Cppcheck

Hallazgos críticos:

doubleFree en hgibbs.cpp: Se detectó un problema grave de liberación doble de memoria en las líneas 83 y 105, donde la variable `v2` es liberada dos veces. Este tipo de error puede causar comportamiento indefinido y crashes en tiempo de ejecución.

- **Acción tomada**: Se ha decidido ignorar temporalmente debido a que `v2` se utiliza como "puntero recipienteçon llamadas a `free()` controladas en los test.
- **Riesgo**: Alto - Puede causar corrupción de memoria y crashes.

Hallazgos de estilo y optimización

Problemas de Const

Se identificaron múltiples oportunidades para mejorar la inmutabilidad del código:

- **constParameter** (Cheb3D.cpp:8): Parámetros de array que pueden declararse como const
- **constVariable** (EKF_GEOS3.cpp:386): Array `Y_true` puede ser const
- **constVariablePointer** (EKF*GEOS3*test.cpp:271): Puntero `results` puede ser const
- ...

Estado: Corregidos durante la revisión de código.

Hallazgos ignorados por mantemer máxima similitud con la implementación original de MATLAB

knownConditionTrueFalse (DEInteg.cpp:84): Condición State_ >DE*INVPARAM* *siempre evalúa a false debido a la inicialización de State* como DE_INIT (valor 1).

duplicateExpression (G_AccelHarmonic.cpp:23): Expresión 1.0 / d donde d = 1.0, resultando en una operación innecesaria.

variableScope: Variables cuyo scope podría reducirse para mejorar la encapsulación.

shadowVariable: Variables que ocultan otras variables del mismo nombre en scopes superiores.

unreadVariable: Variables declaradas pero nunca leídas en el código.

duplicateAssignExpression: Expresiones de asignación duplicadas que podrían simplificarse.

Estado: Tambien ignorados intencionalmente para mantener parecido con implementación original de MATLAB.

Código Unused

- **unusedFunction** (anglesg.cpp:370): Función `debug_anglesg_complete()` nunca utilizada.

Estado: Mantenida para propósitos de debugging futuro.

6.2.2. Análisis de Rendimiento usando Gprof

El programa ejecutado tiene un tiempo total de ejecución de **0.72 segundos**. El análisis revela los principales cuellos de botella y patrones de uso.

Funciones más Costosas (Tiempo de CPU)

Las funciones que más tiempo consumen son:

- `IERS()` - 26.39% del tiempo total (0.19s) con 4,988 llamadas
- `Matrix::operator()(int) const` - 15.28% (0.11s) con 64,445,860 llamadas
- `AccelHarmonic()` - 15.28% (0.11s) con 19,182 llamadas
- `Legendre()` - 13.89% (0.10s) con 19,182 llamadas
- `Matrix::operator()(int, int) const` - 6.94% (0.05s) con 77,455,943 llamadas

Funciones más Llamadas

Las operaciones matriciales dominan completamente:

- `Matrix::operator()(int) const` - **64,445,860 llamadas** (acceso a elementos 1D)

- `Matrix::operator()(int, int) const` - **77,455,943 llamadas** (acceso a elementos 2D)
- `Matrix::initMatrix()` - 552,022 llamadas
- `Matrix::~Matrix()` - 552,005 llamadas

Estructura del Call Graph

El programa sigue esta jerarquía principal:

1. `main()` → `EKF_GEOS3()` (98.6% del tiempo total)
2. `EKF_GEOS3()` ejecuta:
   - `DEInteg()` - 94 veces (91% del tiempo)
   - Filtro de Kalman con `MeasUpdate()` y `TimeUpdate()`
3. `DEInteg()` llama intensivamente a:
   - `VarEqn()` - 2,374 veces
   - `Accel()` - 2,564 veces

Cuellos de Botella Identificados

1. Acceso a Matrices (21.22% tiempo total)

- 141+ millones de accesos a elementos de matriz
- Overhead significativo de la indexación base-1

2. Transformaciones Astronómicas (26.39%)

- `IERS()` para correcciones Earth Orientation Parameters
- Cálculos repetitivos de precesión/nutación

3. Cálculos Gravitacionales (29.17%)

- `AccelHarmonic()` + `Legendre()` para el modelo gravitacional
- 19,182 evaluaciones de polinomios de Legendre

Observaciones Técnicas

- **Overhead de la clase Matrix**: La implementación de matrix hecha para asemejarse al funcionamiento de matlab genera muchas llamadas
- **Algoritmos astronómicos**: Muy intensivos computacionalmente pero necesarios para precisión
- **Gestión de memoria**: 552K creaciones/destrucciones de matrices sin memory leaks aparentes
- **Filtro de Kalman**: Funcionando correctamente con 138 actualizaciones de medidas

Limitaciones del Análisis

El análisis muestra un sistema bien estructurado donde el tiempo se gasta apropiadamente en cálculos científicos complejos, no en ineficiencias obvias del código, aparte de Matrix.

6.2.3. Análisis de Rendimiento y Memoria Usando Valgrind

Herramientas utilizadas: Valgrind 3.22.0 (Memcheck, Callgrind, Cachegrind)
Sistema: Linux WSL2 (Ubuntu 24.04.2)

El programa ha sido sometido a un análisis completo de memoria y rendimiento usando Valgrind. Los resultados revelan un memory leak menor en el programa principal, mientras que los tests unitarios muestran un comportamiento limpio.

Análisis de Memoria (Memcheck)

Estado General del Programa Principal

- **Errores detectados:** 1 error (memory leak)
- **Memory leaks:** 176 bytes perdidos (64 directos + 112 indirectos)
- **Gestión de memoria:** Mayormente correcta con una falla menor

Detalles del Heap - Main

```
Heap Summary:
- Bloques en uso al finalizar: 176 bytes en 9 bloques
- Uso total del heap: 69 allocaciones, 60 liberaciones
- Memoria total asignada: 84,930 bytes
```

Desglose de Memory Leaks

- **64 bytes definitivamente perdidos:** 1 bloque en `EKF_GEOS3()`
- **112 bytes indirectamente perdidos:** 8 bloques relacionados con `Matrix::initMatrix()`
- **Origen:** Todas las pérdidas se originan en la función `EKF_GEOS3()` llamada desde `main()`

Estado de Tests Unitarios

- **Errores detectados:** 0
- **Memory leaks:** Ninguno
- **Gestión de memoria:** Perfecta

Interpretación: El problema de memoria está localizado específicamente en la función `EKF_GEOS3()` del programa principal. Los tests unitarios funcionan perfectamente, lo que sugiere que el issue está en la lógica específica del filtro de Kalman extendido, no en las funciones base de la librería.

Análisis de Rendimiento (Callgrind/Cachegrind)

Observaciones de Rendimiento

- Los archivos de análisis de rendimiento no pudieron ser procesados completamente
- Se requiere instalación de herramientas adicionales (`callgrind_annotate`, `cg_annotate`)

Ubicación Específica del Memory Leak

El memory leak se origina en tres ubicaciones dentro de `EKF_GEOS3()`:

1. **Línea 0x121DBC:** Allocación de objeto con `operator new`
2. **Línea 0x121DD1:** Creación de matrices con `Matrix::initMatrix()`
3. **Línea 0x11E5FE:** Allocación principal que causa el problema

Interpretación Técnica

Impacto del Memory Leak

- **Severidad:** Baja (solo 176 bytes)
- **Frecuencia:** Una vez por ejecución del programa
- **Impacto operacional:** Mínimo para ejecuciones cortas

Calidad del Código

- **Tests unitarios:** Excelente gestión de memoria (0 leaks)
- **Funciones core:** La clase Matrix y algoritmos base funcionan correctamente
- **Problema localizado:** El problema está específicamente en el flujo principal del EKF

6.3 Impacto de las mejoras

La aplicación sistemática de estos análisis permitió identificar y resolver issues específicos, resultando en un código más robusto y eficiente que mantiene la precisión numérica requerida.
Aún así, fui incapaz de conseguir el valor exacto, habiendo un pequeñisimo error de redondeo que no logro encontrar desde donde ocurre.

7 Documentacion

Para la generación de documentación técnica se utilizó Doxygen. La configuración clave del Doxyfile incluyó:

```
EXTRACT_ALL = YES
EXTRACT_PRIVATE = YES
EXTRACT_STATIC = YES
SOURCE_BROWSER = YES
INLINE_SOURCES = YES
CALL_GRAPH = YES
CALLER_GRAPH = YES
CLASS_DIAGRAMS = YES
COLLABORATION_GRAPH = YES
```

Estas configuraciones permiten que Doxygen extraiga automáticamente la estructura completa de clases, funciones, parámetros y dependencias directamente del código. Se generaron diagramas de llamadas entre funciones y grafos de colaboración que resultan especialmente útiles para entender las interdependencias del proyecto.
Adicionalmente, se creó un archivo mainpage.md que proporciona una descripción general del proyecto, incluyendo funcionalidades principales, arquitectura del sistema, instrucciones de compilación y información del entorno de desarrollo. Este mainpage sirve como punto de entrada a la documentación generada.
La documentación resultante incluye navegación por código fuente, índices alfabéticos de funciones y una vista completa de la arquitectura del proyecto.