UNIVERSIDAD
POLITÉCNICA
DE MADRID

Ampliación de Matemáticas I

Programación

Mª Blanca Boado Cuartero
David Fernández Pulido
Ignacio García Guerrero

25 de enero de 2019

Máster Universitario en Sistemas Espaciales

Índice

1. Introducción **2**

2. Punteros **3**

3. Extreme programming **3**

4. Región de Estabilidad Absoluta **3**

5. El exponente de Lyapunov **3**

6. Mapa de Poincaré **4**

7. Paradigmas de la programación **4**

8. Subrutinas **5**

9. Código escrito **5**

9.1. Esquemas numéricos 5

9.1.1. Oscilador armónico 6

9.1.2. Órbita kepleriana 9

9.1.3. Problema N cuerpos 10

9.2. Error del esquema numérico 11

1. Introducción

El presente documento trata de explicar de manera sencilla los conocimientos adquiridos a lo largo de la asignatura de ampliación de matemáticas 1, integrada en el programa académico del Master Universitario en Sistemas Espaciales, MUSE, así como la presentación de casos prácticos donde aplicarlos.

El objetivo de esta asignatura es el manejo de herramientas y técnicas de programación útiles para el cálculo numérico presente en cualquier problema matemático que puede afrontar un ingeniero. En concreto en esta asignatura se han utilizado como lenguajes de programación Fortran-90, MATLAB, y Python, surgiendo numerosos debates en clase sobre ventajas y utilidades de cada uno según el problema a resolver o el algoritmo usado.

Por otro lado la metodología a utilizar es común en cualquiera de los casos,siendo el esquema a seguir:

- Idea
- Modelo matemático
- Algoritmo
- Implementación
- Resultados
- Validación

Este proceso se realiza de forma iterativa, hasta obtener resultados coherentes. Gracias al uso de este guión y a las técnicas que se mostrarán en este documento, hemos aprendido a enfrentarnos a cualquier problema que se pueda plantear así como el uso de las herramientas necesarias para su resolución.

2. Punteros

En programación, un puntero es un objeto que refiere a otro valor almacenado en otea parte de la memoria del ordenador utilizando su dirección. Los punteros a datos mejoran significativamente el rendimiento de las operaciones repetitivas tales como cadenas de desplazamiento, tablas de búsqueda, tablas de control y estructuras árbol.También se utilizan para mantener las direcciones de los puntos de entrada para las subrutinas (que se explicaran mas adelante) para llamadas en programación por procedimientos

En Fortran, los punteros contienen algo mas que una simple dirección de memoria. Un operador asociación se utiliza para asociar un puntero a una variable que tiene un atributo TARGET. En Fortran-90 también se puede usar la declaración ALLOCATE para asociar un puntero a un bloque de memoria. A continuación se puede ver un ejemplo de código:

```
column(:) => A(:,3)
column = column + 1
```

En Python no se usan punteros como tal con el propósito de hacer mas fácil y ágil su utilización. En Python todo es un objeto creado en la memoria dinámica. Cuando llamas a una función los argumentos son pasados mediante sus punteros. Es lo que se conoce como convención de llamada por objeto. De igual forma si asignas $a = b$, a lo que guarda es el puntero del objeto de b. Así que todas las variables son punteros a objetos, que son manejados implícitamente.

3. Extreme programming

Extreme programming (tambien conocida como XP) es un método de desarrollo de software diseñado para mejorar la calidad del software y su habilidad de adaptarse propiamente a las necesidades cambiantes del cliente. La *Extreme programming* mejora un proyecto de software de cinco maneras esenciales; Comunicación, simplicidad, feedback, respeto y coraje. Los programadores se comunican constantemente con sus clientes y compañeros. Mantienen su diseño simple y limpio. Reciben comentarios al probar su software desde el primer día. Entregan el sistema a los clientes lo antes posible e implementan los cambios como se sugiere. Cada pequeño éxito profundiza su respeto por las contribuciones únicas de cada miembro del equipo.Con esta base, los programadores pueden responder con valentía a los cambios en los requisitos y la tecnología.

La XP es exitosa porque enfatiza la satisfacción del cliente. En lugar de entregar todo lo que el cliente pueda querer en una fecha lejana en el futuro, este proceso entrega el software que necesita a medida que lo necesita. La XP le permite a sus programadores responder con confianza a los cambios en los requisitos del cliente.

4. Región de Estabilidad Absoluta

La región de estabilidad absoluta es una propiedad de las ecuaciones diferenciales ordinarias. A la hora de integrar numéricamente un problema interesa que la solución numérica tenga el mismo carácter de estabilidad que la solución analítica y saber para que valores del paso de integración se puede conseguir eso.

5. El exponente de Lyapunov

El exponente de Lyapunov de un sistema dinámico es una cantidad que caracteriza el grado de separación de dos trayectorias cercanas. Las dos trayectorias en el espacio-fase con una separación

inicial δZ_0 divergen a una velocidad dada por

$$|\delta Z(t)| \approx e^{\lambda t}|\delta Z_0| \tag{1}$$

donde λ es el exponente de Lyapunov.

La separación puede ser distinta para diferentes orientaciones del vector de separación inicial. Por ello que existe un espectro de exponentes de Lyapunov, igual en numero a las dimensiones del espacio-fase. Generalmente uno solo se refiere al exponente mas grande porque determina la predictibilidad de un sistema.

6. Mapa de Poincaré

En matemáticas, en especial en los sistemas dinámicos, el Mapa de Poincaré (nombrado así en honor de Henri Poincaré) o mapa de primera recurrencia, es la intersección de una órbita periódica en el estado de espacio de un sistema dinámico continuo con un cierto subespacio de dimensión inferior, llamada sección de Poincaré, transversal al flujo del sistema.

Uno considera una órbita periódica con condiciones iniciales dentro de una sección del espacio, que después deja esa sección, y se observa el punto en el que esta órbita regresa por primera vez a la sección. Luego, uno crea un mapa para enviar el primer punto al segundo, de ahí el nombre del primer mapa de recurrencia. La transversalidad de la sección de Poincaré significa que las órbitas periódicas que comienzan en el subespacio fluyen a través de él y no son paralelas a él.

7. Paradigmas de la programación

Un paradigma de programación indica un método de realizar cómputos y la manera en que se deben estructurar y organizar las tareas que debe llevar a cabo un programa. . Los lenguajes de programación, necesariamente, se encuadran en uno o varios paradigmas a la vez a partir del tipo de órdenes que permiten implementar, algo que tiene una relación directa con su sintaxis. Los principales tipos de paradigmas de programación son el imperativo, el declarativo, el lógico, el funcional y el orientado a objetos. Nos centraremos en los dos últimos.

La programación funcional es de un corte mas matemático. Los programas se componen en funciones, es decir, implementaciones de comportamiento que reciben un conjunto de datos de entrada y devuelven un valor de salida.

La programación orientada a objetos define los programas en términos de çlases de objetos". Expresa un programa como un conjunto de estos objetos, que colaboran entre ellos para realizar tareas. Está basada en varias técnicas, como el encapsulamiento.

A continuación un ejemplo de ambos (FORTRAN):

```
MATH DOMAIN (Paradigma funcional)
dU/dt = F(U;t)
U_0

PHYSICAL DOMAIN (paradigma de objetos)
type body
        real:: mass
        real:: r(3), v(3)
        real:: drdt(3), dvdt(3)
end type
```

```
type, extends(body):: satellite
        real:: engine
end type

type galaxy
        type(body), allocatable :: Planet(:)
        contains
                procedure :: ini => initialization
                procedure :: acceleration => acceleration-planet
end type
```

8. Subrutinas

En programación, una subrutina es una secuencia de instrucciones que realiza una tarea especifica en un programa, empaquetada en una unidad. Esta unidad se puede utilizar en programas donde se deba realizar esa tarea en particular. Se pueden definir dentro de los programas o en librerías separadas. Dependiendo del lenguaje de programación, a una subrutina se le puede llamar *procedure, function, routine o method.*

El contenido de una subrutina es su cuerpo. Se puede escribir una subrutina para que espere obtener uno o más valores de datos del programa que llama (para reemplazar sus parámetros o parámetros formales). La subrutina puede entonces devolver un valor computado o proveer de varios resultados o *outputs.*

En Python hay dos tipos de subrutinas, *procedures* y funciones. Un *procedure* simplemente ejecuta comandos, como imprimir algo un cierto número de veces. Una función produce información al recibir datos del programa principal y devolver un valor al programa principal.

En Fortran, las funciones y subrutinas son distintas. Una función es similar a una función matemática, que toma uno o varios parámetros como entradas y devuelve un solo valor de salida. Una subrutina en Fortran es un bloque de código que realiza alguna operación en las variables de entrada y, como resultado de llamar a la subrutina, las variables de entrada se modifican.

9. Código escrito

9.1. Esquemas numéricos

Se desarrolla un módulo propio en el que se almacenarán los integradores que se van a emplear a lo largo de los códigos. Se crea una subrutina *Integradores* la cual almacena la estructura de un Runge-Kutta de segundo orden y la de un Euler.

```python
import numpy as np
import math
import matplotlib.pyplot as mlp

def Runge_Kutta_2_orden(paso_temp,F,U):

    U2 = U + paso_temp*0.5*F(U)
    U = U + paso_temp*0.5*(F(U) + F(U2))

    return U

def Euler(paso_temp,F,U):

    U = U + paso_temp*F(U)
    return U

```

Figura 1: Integradores de Euler y Runge-Kutta de orden 2

Como se puede observar en el código, la variable U corresponde al vector de estado, mientras que F es una función cuyo resultado es el producto de la matriz de estado por el vector de estado correspondiente. Gracias a estos integradores podremos resolver diversos problemas numéricos. A continuación se presentan algunos casos prácticos:

9.1.1. Oscilador armónico

El problema planteado consiste en la resolución de la siguiente ecuación:

$$\ddot{x} + x = 0 \tag{2}$$

En primer lugar se va a resolver utilizando un integrador de Euler inverso. El código utilizado es el siguiente:

```
#Paso temporal
numPasos=1000
step=20*math.pi/numPasos

# Matriz inversa B
B=(1/(1+step**2))*np.matrix([[1,step],[-step,1]])

# Definición de vectores y condiciones iniciales
U=np.zeros((2*numPasos,1))
X=np.zeros((numPasos,1))
V=np.zeros((numPasos,1))
U[0,0]=1
U[1,0]=0
X[0,0]=1
V[0,0]=0

# Integrador de Euler inverso
for i in range(numPasos-1):
    Un=B*np.matrix([[U[2*i,0]],[U[2*i+1,0]]])
    U[2*i+2,0]=Un[0,0]
    X[i+1]=Un[0,0]
    U[2*i+3,0]=Un[1,0]
    V[i+1]=Un[1,0]
```

Figura 2: Integración oscilador armónico por Euler inverso

Como se ve, se define la integración para un tiempo de 10 vueltas alrededor del centro. Se define la matriz inversa y se procede a la integración mediante un bucle "forz diferentes arrays. El resultado se muestra a continuación:

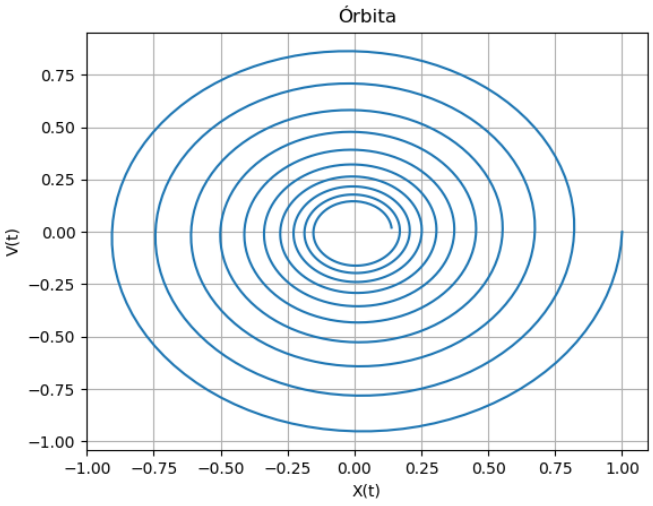

Figura 3: Integración oscilador armónico por Euler inverso: Resultados

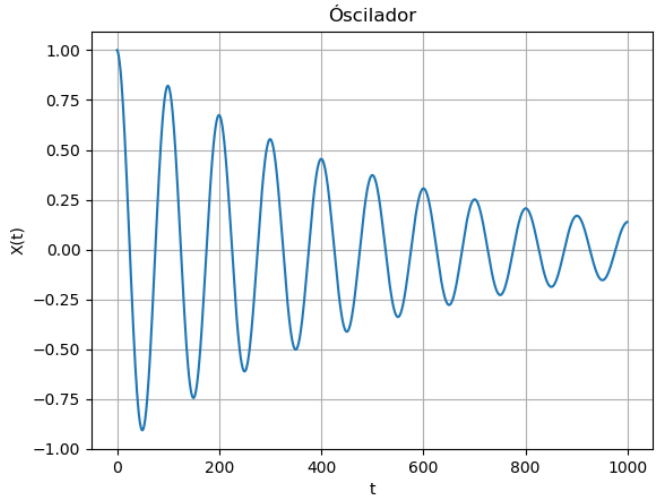

Figura 4: Integración oscilador armónico por Euler inverso: Resultados

La órbita decae hacia el centro conforme avanza el tiempo. Esto es debido a que el esquema de Euler no se encuentra dentro de la región de estabilidad absoluta para el problema del oscilador armónico. Por ello serían necesarios esquemas de mayor orden, como el caso del Runge-Kutta de orden 2, ya comentado anteriormente.

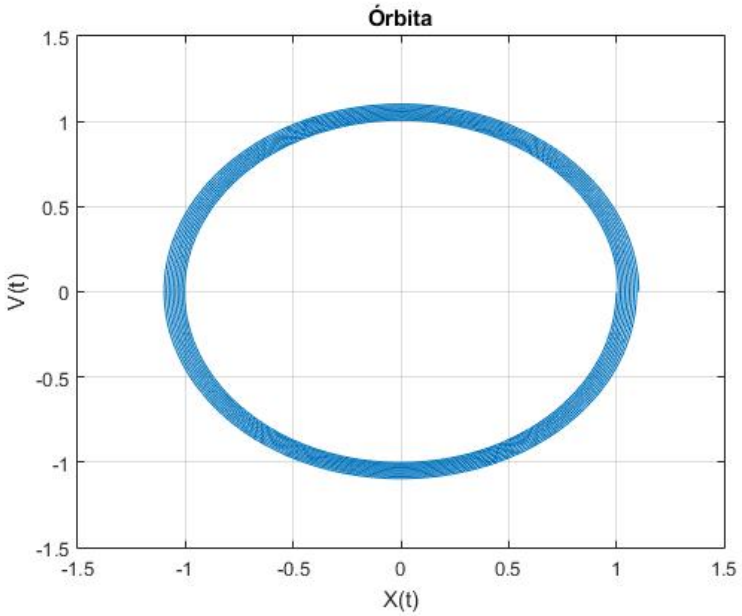

Figura 5: Integración oscilador armónico con Runge-Kutta orden 2: Resultados

Como puede apreciarse, nos vamos acercando a la región de estabilidad absoluta, pero para

ser más precisos sería necesario el uso de otro sistema de integración de mayor orden.

9.1.2. Órbita kepleriana

A continuación se muestra a modo de ejemplo el uso del integrador .ºdeint"de python para el cálculo de la trayectoria de dos cuerpos de masa unidad según las fuerzas de gravitación existentes entre ambos.
La ecuación que describe el comportamiento de una órbita kepleriana es:

$$\ddot{r} = -\frac{1}{r^3} u_r \tag{3}$$

El código de python utilizado para la resolución de la ecuación anterior es el siguiente:

```
#Definición de constantes
G=1.0
m1=1
m2=1
mu=G*(m1+m2)
N=10000
DeltaT=1000/N
t=np.linspace(0.0,1000.0,N)

#Condiciones iniciales
y0=[1.0, 0.0, 0.0, 1.2]

#devuelve la derivada del vector de estado
def func(y,t):
    r=y[:2]
    v=y[2:]
    return [v[0], v[1], -mu*r[0]/norm(r)**3, -mu*r[1]/norm(r)**3]

# Integracion con 'odeint'
solucion=odeint(func,y0,t)
r=solucion[:, :2]     # vector de posición
v=solucion[:, 2:]     # vector de velocidad
```

Figura 6: Resolución órbita kepleriana mediante odeint:código

El resultado obtenido se muestra a continuación:

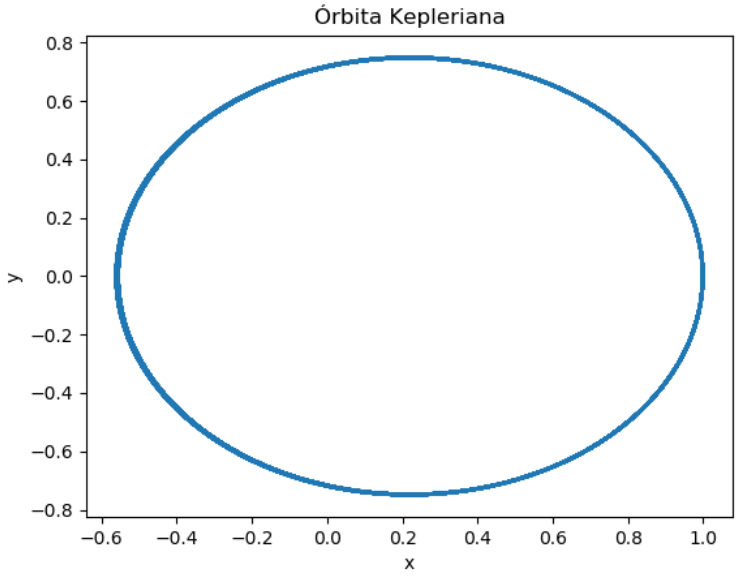

Figura 7: Resolución órbita kepleriana mediante odeint:resultados

Como puede apreciarse el error que comete el integrador odeint es prácticamente nulo, aunque será calculado en siguientes apartados.

9.1.3. Problema N cuerpos

Por último se presenta el código utilizado mediante subrutinas, explicado en clase, para la simulación del problema de los N cuerpos. Este código incluye además un ejemplo de programación orientada a objetos, método muy utilizado en lenguajes como python o C++.

```
class Body:
    def __init__(self,Mass, Position, Velocity):
        self.m=Mass
        self.r=Position
        self.v=Velocity

Earth=Body(10**20,[0,0,0],[10,10,10])
Moon=Body(10**20,[1000, 1000, 1000],[10,10,10])
Satellite=Body(10**20,[5000, 5000, 5000],[10,10,10])
m=[Earth.m,Moon.m,Satellite.m]

r0=np.matrix([Earth.r,Moon.r,Satellite.r])
V0=np.matrix([Earth.v,Moon.v,Satellite.v])
```

Figura 8: Problema N cuerpos: Programación orientada a objetos

Se ha modelizado en nuestro código la presencia de tres cuerpos, como son la tierra, la luna y un satélite en órbita. La función que calcula el comportamiento de cada uno de estos cuerpos tiene la siguiente forma:

```
def Nbodies (N,r,V,drdt,dvdt):
    a=np.array([0,0,0])
    for i in range(N):
        for j in range(N):
            if j!=i:
                d=r[j,:]-r[i,:]
                a[:]=a[:]+G*m[j]*d/(abs(d))**3
        drdt[i,:]=V[i,:]
        dvdt[i,:]=a[:]

U=np.zeros((2*N,3))
U[0:3,0:3]=r0
U[3:6,0:3]=V0
F=np.zeros((2*N,3))
Nbodies (N,U[0:3,0:3],U[3:6,0:3],F[0:3,0:3],F[3:6,0:3])
```

Figura 9: Problema N cuerpos: Función de cálculo

Una vez implementado el modelo, sería necesario la inclusión de un integrador en el código, que actualizase la posición y velocidad de cada cuerpo para cada iteración. Sin embargo, este hecho se implementará en posteriores versiones.

9.2. Error del esquema numérico

Para determinar la precisión de un esquema numérico se emplea la extrapolación de Richardson:

$$E_1^N = \frac{U_1^N - U_2^{2N}}{1 - 0.5^q} \tag{4}$$

Donde U_2^{2N} es la solución evaluada con un paso temporal que es la mitad que el de la solución U_1^N y q es el orden del esquema numérico.

El problema a integrar es un problema de 2 cuerpos. A continuación se muestran los resultados obtenidos a partir de los códigos *errores* y *errores_rg*. En el primero se emplea el esquema numérico propio de Python *odeint* mientras que el otro importa los esquemas numéricos de los almacenados por nosotros mismos, de los cuales sabemos toda la información.

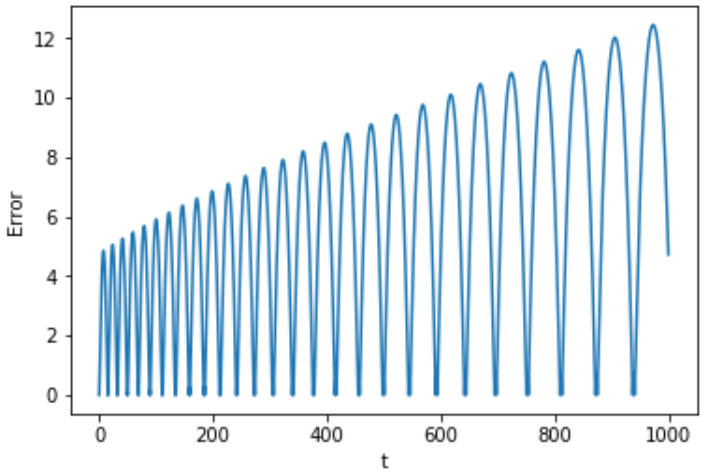

Figura 10: Error esquema Runge-Kutta 2 orden

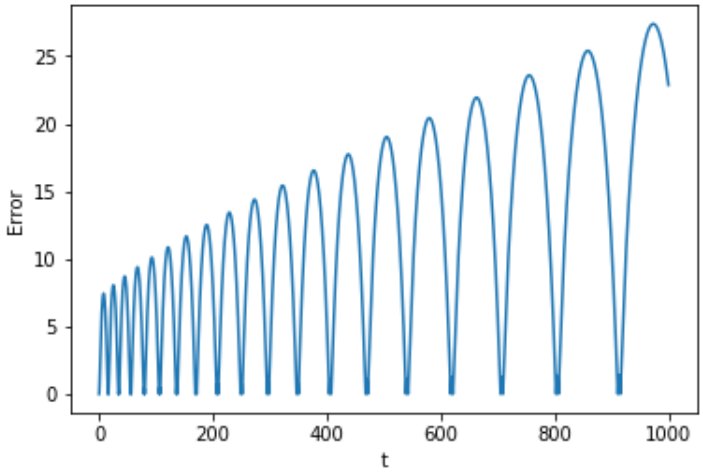

Figura 11: Error esquema Euler

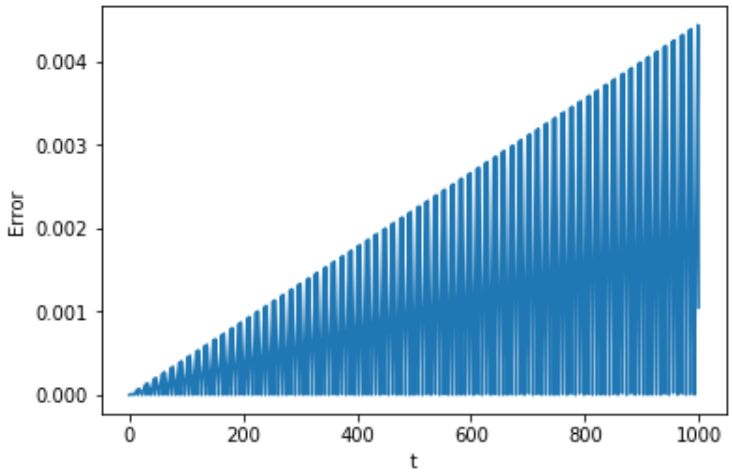

Figura 12: Error esquema odeint, orden 4

Se muestra para un número de pasos de 100000. El integrador propio de Python muestra errores claramente inferiores debido a que es de alto orden y es más apropiado para integrar órbitas. Los otros 2 esquemas al responder a un orden menor que 4 presentan elevadas desviaciones.