TP N° 4: Dinámica Molecular Regida por el Paso Temporal

Grupo N° 5

Daniel Lobo | Agustín Golmar

Fundamentos

Sistema Físico

Oscilador armónico amortiguado:

$$f = ma = mr_2 =- kr - \gamma r_1$$

Las condiciones iniciales son r(0)=1 m y v(0)=-/(2m) m/s

La solución analítica viene dada por:

$$r = r(0)\ e^{-v(0)\ t}\ cos(t\ \sqrt{k/m - v(0)^2})$$

Campo gravitacional:

$$F_{ij} = \frac{Gm_i m_j e_{ij}}{r_{ij}^{\ 2}}$$

G = 6.67191(99)x10-11 m3/(kg * s^2) es la constante de gravitación universal

La interacción de las fuerzas debe descomponerse en componentes cartesianas:

$$F = (F_x, F_y) = (F_n \Delta x / |\Delta r|\ , F_n \Delta y / |\Delta r|)$$

Modelo Matemático

Velocity Verlet

$$r(t + \Delta t) = r(t) + \Delta t\, v(t) + \frac{\Delta t^2}{2m} f(t) + \Theta(\Delta t^3)$$

$$v(t + \tfrac{\Delta t}{2}) = v(t) + \frac{\Delta t}{2m} f(t)$$

$$v(t + \Delta t) = v(t + \tfrac{\Delta t}{2}) + \frac{\Delta t}{2m} f(t + \Delta t) + \Theta(\Delta t^2)$$

- Algoritmo que conserva la energía del sistema.
- Ejecuta la simulación en tiempo reverso (symplectics)
- Las posiciones y velocidades se mantienen en sincronía, pero se utilizan las velocidades intermedias durante la aproximación final.

Modelo Matemático

Beeman

$$r(t+\Delta t) = r(t) + \Delta t\, v(t) + \frac{\Delta t^2}{m}\left(\frac{2}{3} f(t) - \frac{1}{6} f(t-\Delta t)\right) + \Theta(\Delta t^4)$$

$$v_p(t+\Delta t) = v(t) + \frac{\Delta t}{m}\left(\frac{3}{2} f(t) - \frac{1}{2} f(t-\Delta t)\right) + \Theta(\Delta t^3)$$

$$v(t+\Delta t) = v(t) + \frac{\Delta t}{m}\left(\frac{1}{3} f(t+\Delta t) + \frac{5}{6} f(t) - \frac{1}{6} f(t-\Delta t)\right) + \Theta(\Delta t^3)$$

- Produce velocidades más exactas que el algoritmo de Verlet.
- Aplica fuerzas que dependen no solo de la posición, sino también de la velocidad.
- Se utiliza una predicción sobre las velocidades.
- Luego se corrige la velocidad final (ya que se supone que f(t+t) depende de v(t+t)).

Modelo Matemático

<u>Gear Predictor-Corrector:</u>

Se define la derivada k-ésima de la posición r: $r_k = \frac{\partial^k r}{\partial t^k}$

Luego se aplican las predicciones de las derivadas (en este caso hasta orden 5): $r_k^p(t+\Delta t) = \sum_{i=0}^{5-k} (i!)^{-1}\, r_{k+i}(t)\, \Delta t^i$

Luego se computa la fuerza mediante las variables predichas, es decir f(t+t). Ahora se puede definir un coeficiente: $\Delta a = \Delta r_2 = a(t+\Delta t) - a^p(t+\Delta t) = r_2(t+\Delta t) - r_2^p(t+\Delta t)$ $\quad \Delta R_2 = \frac{\Delta r_2 (\Delta t)^2}{2}$

Ahora se computan las variables corregidas: $r_k^c(t+\Delta t) = r_k^c(t+\Delta t) + \alpha_k\, \Delta R_2 \frac{k!}{(\Delta t)^k}$

Modelo Matemático

Gear Predictor-Corrector:

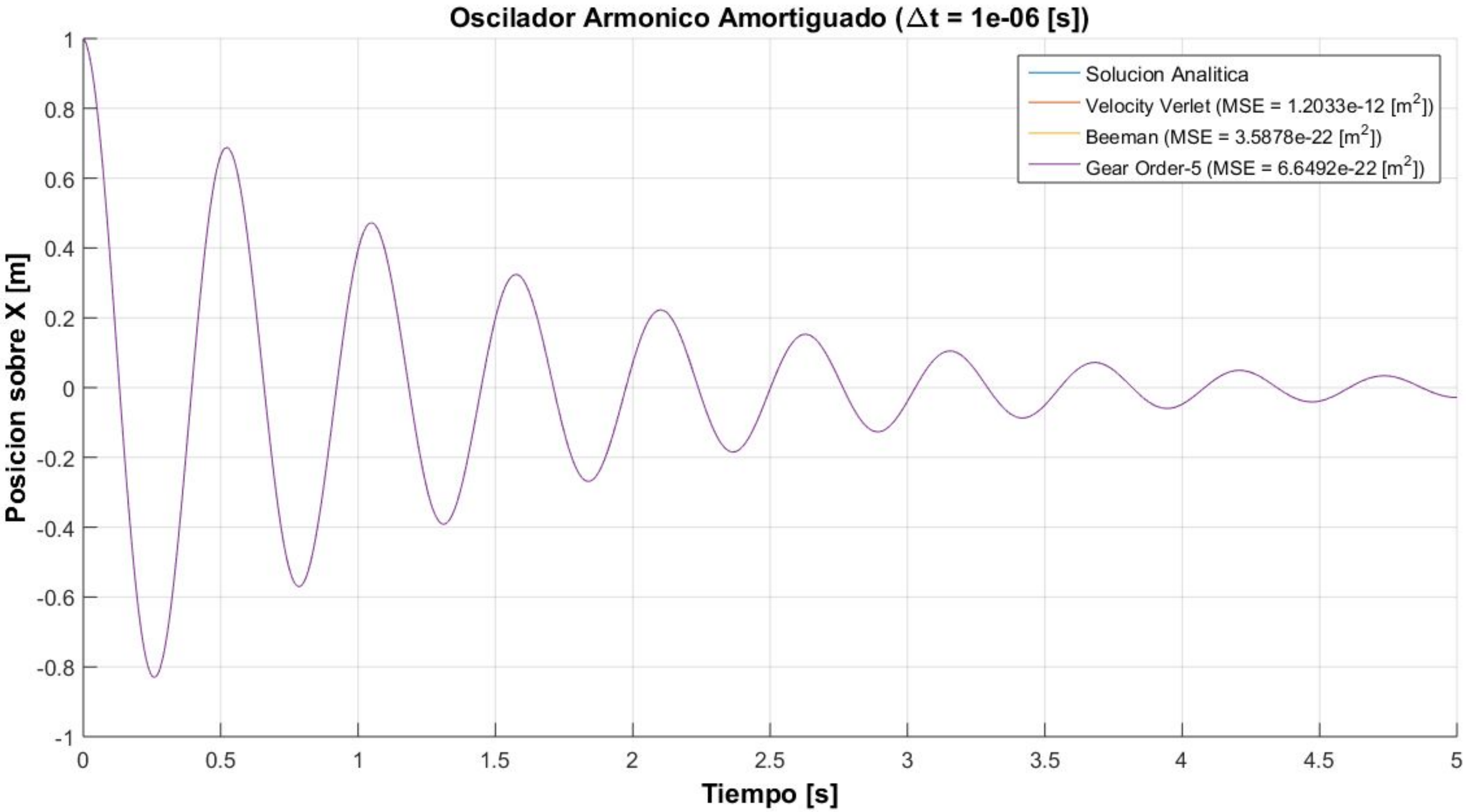
Los coeficientes del predictor α_k dependen del tipo de fuerza. Si esta depende únicamente de la posición, entonces:

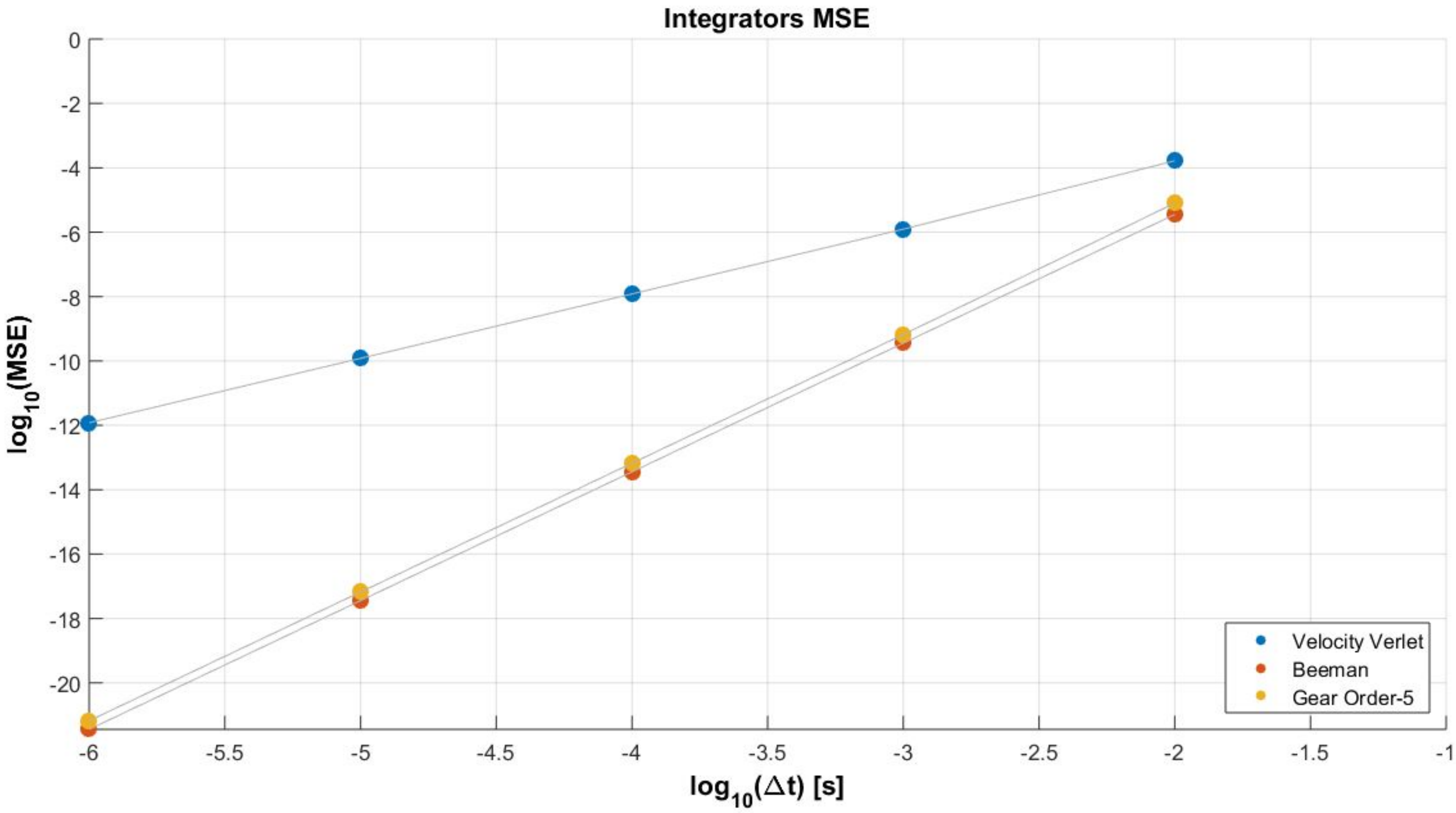
Orden	α_0	α_1	α_2	α_3	α_4	α_5
2	0	1	1	-	-	-
3	1/6	5/6	1	1/3	-	-
4	19/120	3/4	1	1/2	1/12	-
5	3/20	251/360	1	11/18	1/6	1/60

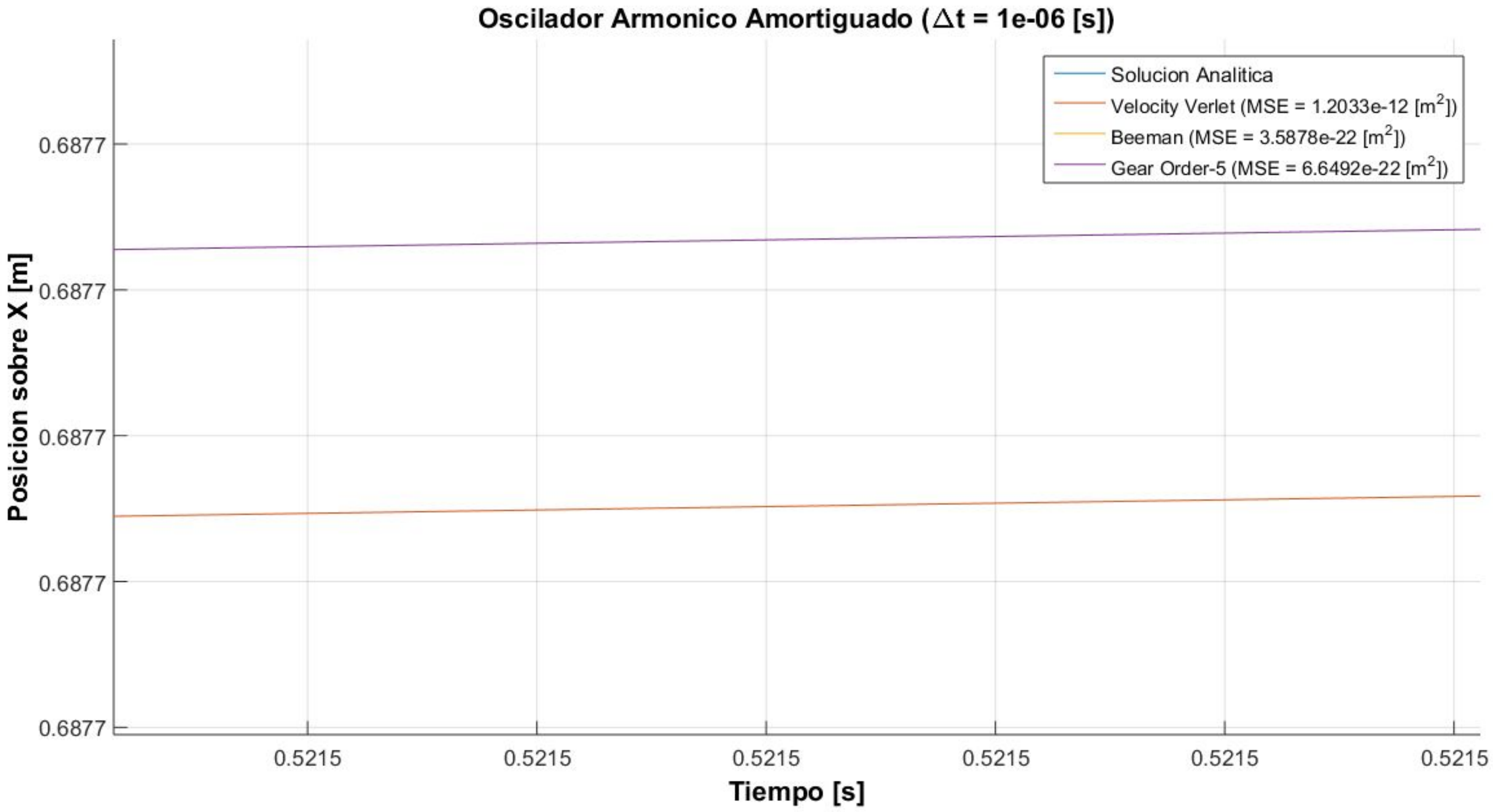
En el caso de que la fuerza dependa también de la velocidad, se modifica $\alpha_0 = 19/90$ para el orden 4, y $\alpha_0 = 3/16$ para el orden 5.

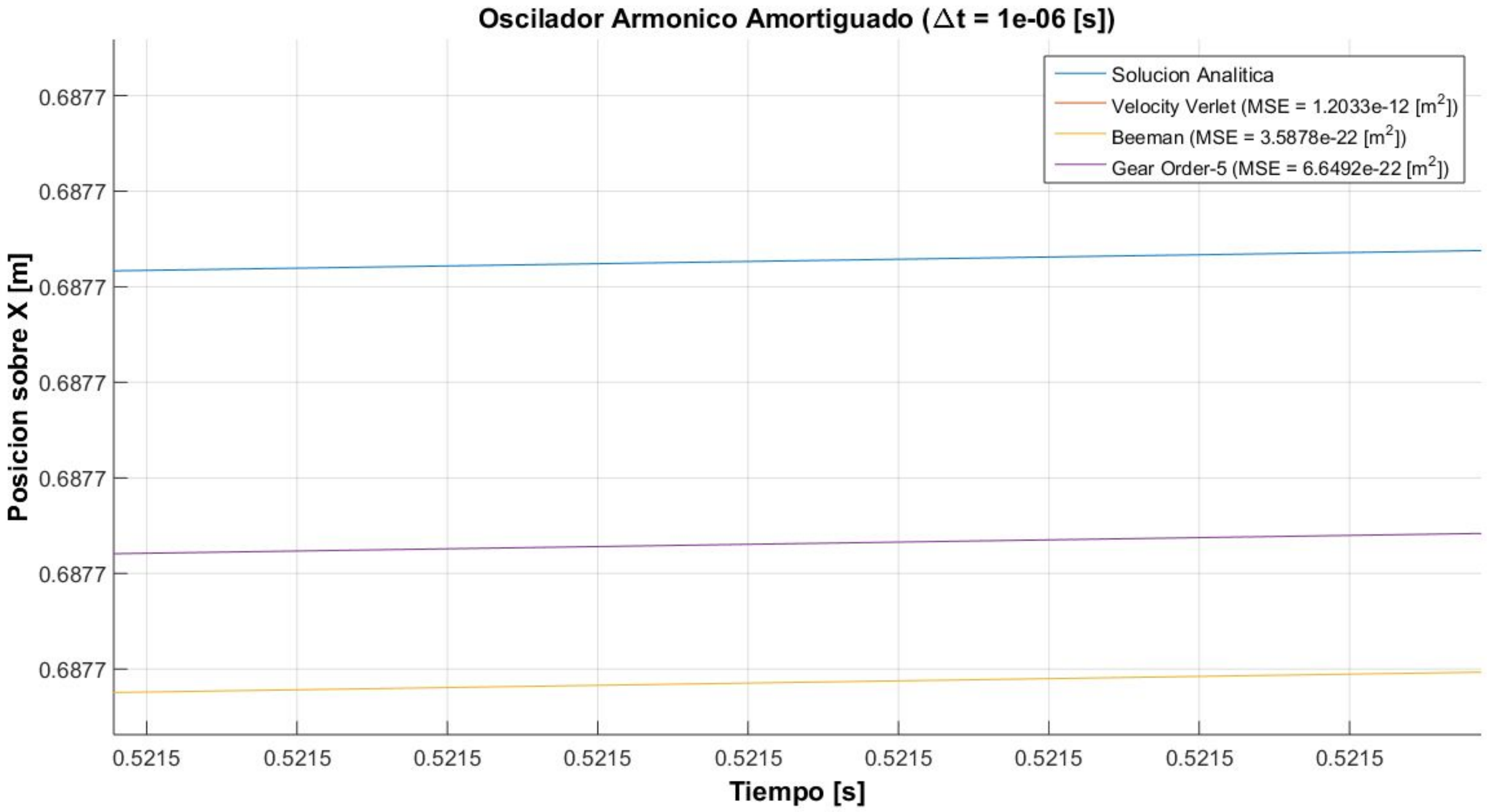
- Si el parámetro delta_t es demasiado pequeño, la simulación será demasiado extensa. Si es demasiado grande, se pueden provocar explosiones entre partículas o trayectorias erróneas. En general, para animar es deseable utilizar un múltiplo de t acorde.
- Para verificar el error es posible verificar la solución real de forma analítica (si es un sistema simple y dicha expresión existe), o se puede utilizar la conservación de energía (cinética y potencial en este caso). Se realizan varias simulaciones hasta que las variaciones de energía con respecto a un valor constante no varíen demasiado (cota de error).

Resultados del Oscilador Armónico Amortiguado


Oscilador Armonico Amortiguado (Δt = 1e-06 [s])
Solucion Analitica
Velocity Verlet (MSE = 1.2033e-12 [m^2])
Beeman (MSE = 3.5878e-22 [m^2])
Gear Order-5 (MSE = 6.6492e-22 [m^2])
Posicion sobre X [m]
Tiempo [s]
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5


Integrators MSE
0
-2
-4
-6
-8
-10
-12
-14
-16
-18
-20
log10(MSE)
-6
-5.5
-5
-4.5
-4
-3.5
-3
-2.5
-2
-1.5
-1
log10(Δt) [s]
Velocity Verlet
Beeman
Gear Order-5


Oscilador Armonico Amortiguado (∆t = 1e-06 [s])
Solucion Analitica
Velocity Verlet (MSE = 1.2033e-12 [m2])
Beeman (MSE = 3.5878e-22 [m2])
Gear Order-5 (MSE = 6.6492e-22 [m2])
Posicion sobre X [m]
0.6877
0.6877
0.6877
0.6877
0.6877
0.5215
0.5215
0.5215
0.5215
0.5215
0.5215
Tiempo [s]



Oscilador Armonico Amortiguado (∆t = 1e-06 [s])
Solucion Analitica
Velocity Verlet (MSE = 1.2033e-12 [m2])
Beeman (MSE = 3.5878e-22 [m2])
Gear Order-5 (MSE = 6.6492e-22 [m2])
0.6877
0.6877
0.6877
0.6877
0.6877
0.6877
0.6877
Posicion sobre X [m]
0.5215
0.5215
0.5215
0.5215
0.5215
0.5215
0.5215
0.5215
0.5215
Tiempo [s]

Implementación

Modelo Computacional

- TimeDrivenSimulation
- GravitationalField
- HarmonicOscillator
- Integrator
- IntegratorBuilder
- BeemanIntegrator
- GearIntegrator
- VelocityVerlet
- ForceField
- Vector

TimeDrivenSimulation class

```
public class TimeDrivenSimulation {

    protected final Integrator<MassiveParticle> integrator;
    protected final BiConsumer<Double, List<MassiveParticle>> spy;
    protected final double maxTime;
    protected final double Δt;

    public TimeDrivenSimulation(final Builder builder) {

    public static Builder of(final Integrator<MassiveParticle> integrator) {

    public TimeDrivenSimulation run() {

    public static class Builder {
}
```

GravitationalField class

```java
public class GravitationalField implements ForceField<MassiveParticle> {

    public static final double G = 6.67191E-20;

    public Vector apply(

    public boolean isVelocityDependent() {

    public boolean isConservative() {

    public Vector derivative1(

    public Vector derivative2(

    public Vector derivative3(

    public double energyLoss(final double time) {

    public double potentialEnergy(final MassiveParticle body) {

    public double potentialEnergy(final List<MassiveParticle> state) {

    protected double potential(

    protected Vector attraction(
}
```

HarmonicOscillator class

```java
public class HarmonicOscillator implements ForceField<MassiveParticle> {

    public static final double k = 10000.0;
    public static final double m = 70.0;
    public static final double γ = 100.0;
    public static final double A = 1.0;

    public final double E0;
    public final double ω;
    public final double ϕ;

    public HarmonicOscillator() {...

    protected static final double FACTORS [][] = {...

    public Vector apply(...

    public boolean isVelocityDependent() {...

    public boolean isConservative() {...

    public Vector derivative1(...

    public Vector derivative2(...

    public Vector derivative3(...

    public double energyLoss(final double time) {...

    public double potentialEnergy(final MassiveParticle body) {...
}
```

Integrator interface

```java
public interface Integrator<T extends MassiveParticle> {

    public List<T> getState();
    public ForceField<T> getForceField();
    public List<T> integrate(final double Δt);

    public default double getEnergy(final double time) {...
}
```

IntegratorBuilder class

```java
public abstract class IntegratorBuilder<T extends Integrator<MassiveParticle>> {

    protected final ForceField<MassiveParticle> force;
    protected List<MassiveParticle> state;

    public IntegratorBuilder(..

    public IntegratorBuilder<T> withInitial(..

    public abstract T build();
}
```

BeemanIntegrator class

```
public class BeemanIntegrator implements Integrator<MassiveParticle> {

    protected final ForceField<MassiveParticle> force;
    protected final List<MassiveParticle> state;
    protected final Vector [] fOld;

    public BeemanIntegrator(final Builder builder) {...

    public static Builder of(final ForceField<MassiveParticle> force) {...

    public List<MassiveParticle> getState() {...

    public ForceField<MassiveParticle> getForceField() {...

    public List<MassiveParticle> integrate(final double Δt) {...

    public static class Builder...

    protected Vector r(...

    protected Vector vp(...

    protected Vector v(...
}
```

GearIntegrator class

```
public class GearIntegrator implements Integrator<MassiveParticle> {

    protected static final double α [][] = {

    protected final ForceField<MassiveParticle> force;
    protected final List<MassiveParticle> state;
    protected final Vector [][] derivatives;
    protected final double [] Δ;

    public GearIntegrator(final Builder builder) {

    public static Builder of(final ForceField<MassiveParticle> force) {

    public List<MassiveParticle> getState() {

    public ForceField<MassiveParticle> getForceField() {

    public List<MassiveParticle> integrate(final double Δt) {

    protected double [] getΔ(final double Δt) {

    protected Vector r0p(

    protected Vector r1p(

    protected Vector r2p(

    protected Vector r3p(

    protected Vector r4p(final Vector r4, final Vector r5) {

    protected Vector r5p(final Vector r5) {

    public static class Builder
}
```

VelocityVerlet class

```
public class VelocityVerlet implements Integrator<MassiveParticle> {

    protected final ForceField<MassiveParticle> force;
    protected final List<MassiveParticle> state;

    public VelocityVerlet(final Builder builder) {

    public static Builder of(final ForceField<MassiveParticle> force) {

    public List<MassiveParticle> getState() {

    public ForceField<MassiveParticle> getForceField() {

    public List<MassiveParticle> integrate(final double Δt) {

    public static class Builder

    protected Vector r(

    protected Vector v(
}
```

ForceField interface

```
public interface ForceField<T extends MassiveParticle>
    extends BiFunction<List<T>, T, Vector> {

    public boolean isVelocityDependent();
    public boolean isConservative();

    public Vector derivative1(final List<T> state, final T body);
    public Vector derivative2(final List<T> state, final T body);
    public Vector derivative3(final List<T> state, final T body);

    public double energyLoss(final double time);
    public double potentialEnergy(final T body);

    public default double kineticEnergy(final T body) {..

    public default double mechanicalEnergy(final T body) {..

    public default double potentialEnergy(final List<T> state) {..

    public default double kineticEnergy(final List<T> state) {..

    public default double mechanicalEnergy(final List<T> state) {..

    public default double energy(..
}
```

Vector class

```
public class Vector {

    public static final Vector ZERO = Vector.of(0.0, 0.0);

    protected final double x;
    protected final double y;

    public static Vector of(final double x, final double y) {

    public Vector(final double x, final double y) {

    public double getX() {

    public double getY() {

    public Vector add(final Vector vector) {

    public Vector subtract(final Vector vector) {

    public double dot(final Vector vector) {

    public Vector over(final Vector vector) {

    public Vector multiplyBy(final double value) {

    public Vector dividedBy(final double value) {

    public Vector power(final double exponent) {

    public Vector versor() {

    public double magnitude() {

    public Vector tangent() {
}
```

Formato de Archivos

Input file: (formato JSON)

```
{
        "fps"                   : "40",
        "deltat"                : "0.000001",
        "integrator"    : "Beeman",
        "system"                : "HarmonicOscillator",
        "output"                : "resources/data/hm-beeman.data",
        "maxtime"               : "5.0"
}
```

Output files: (formato TXT)

- Simulation file:

```
<N>
<t0>
<x> <y> <r> <vx> <vy>
...
```

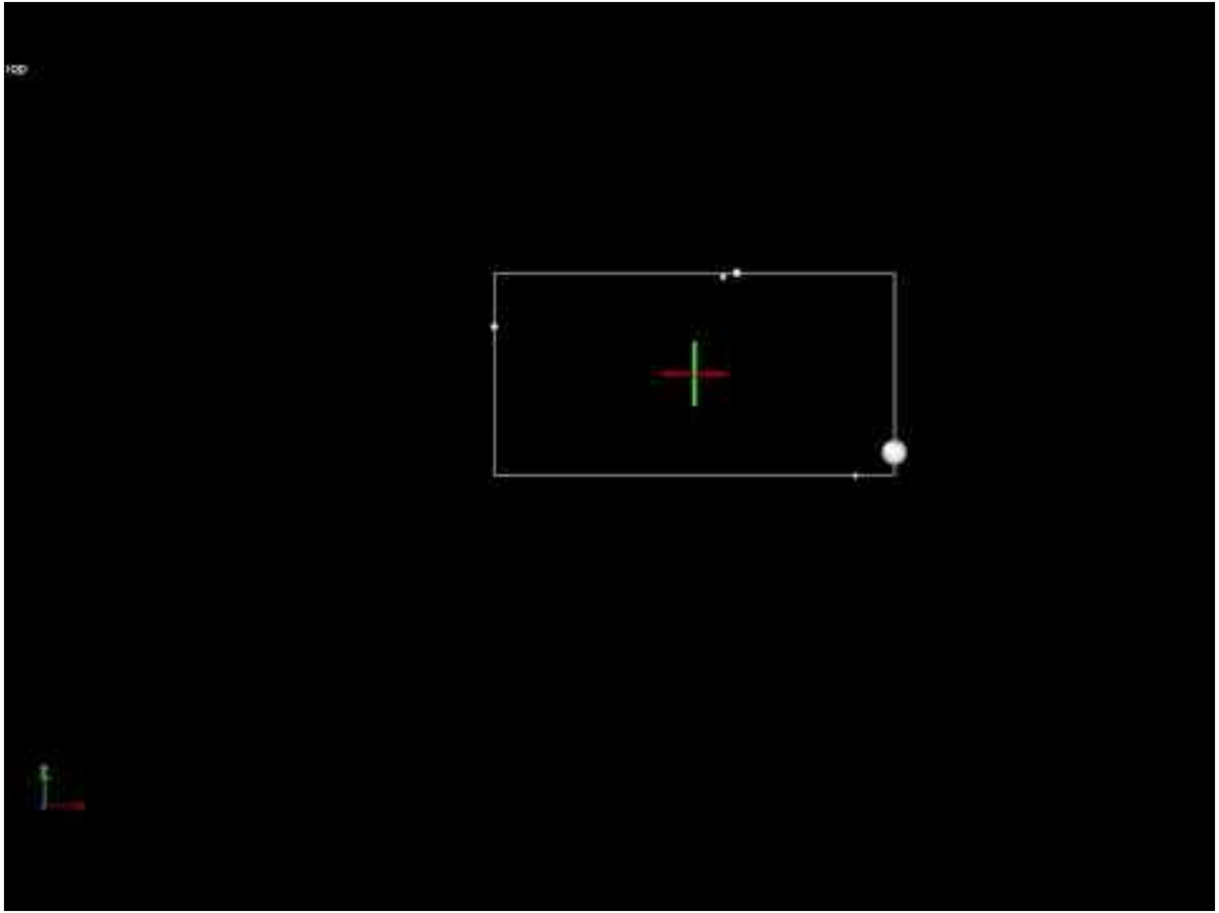
- Animated File:

```
<N>
<t0>
<x> <y> <r> <v>
...
```

Simulación

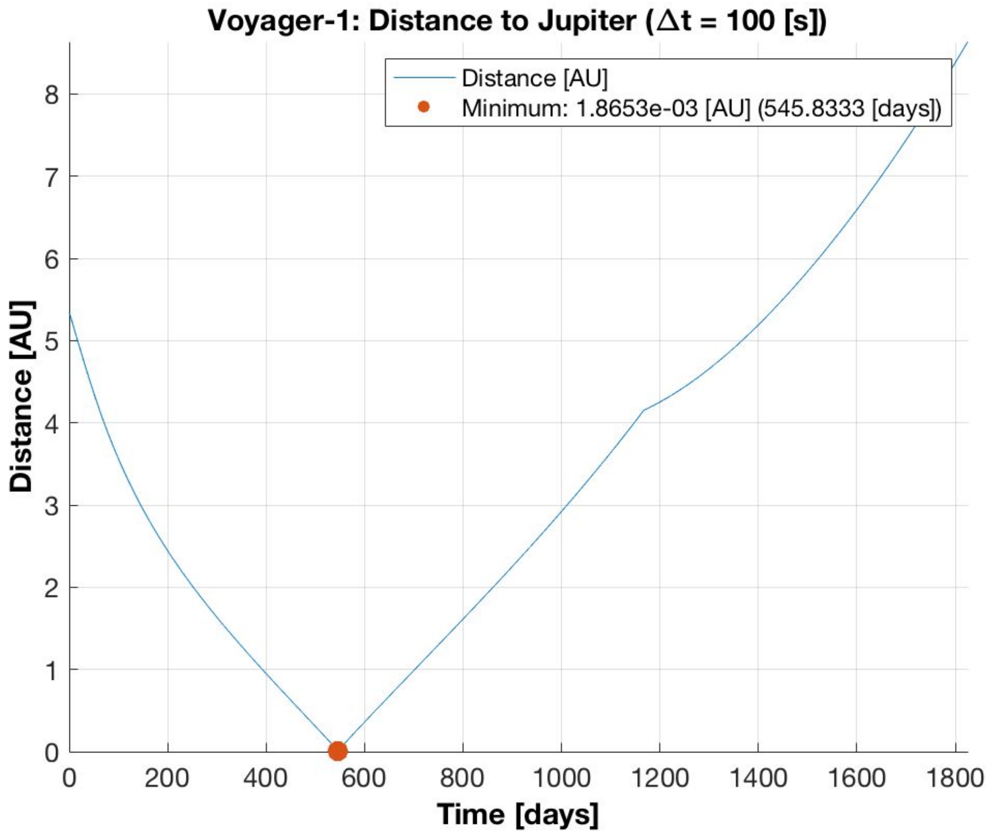
Sept 5th, 1977
14hs

Día óptimo en el que la trayectoria de la sonda alcanza la mínima distancia a Júpiter y Saturno.

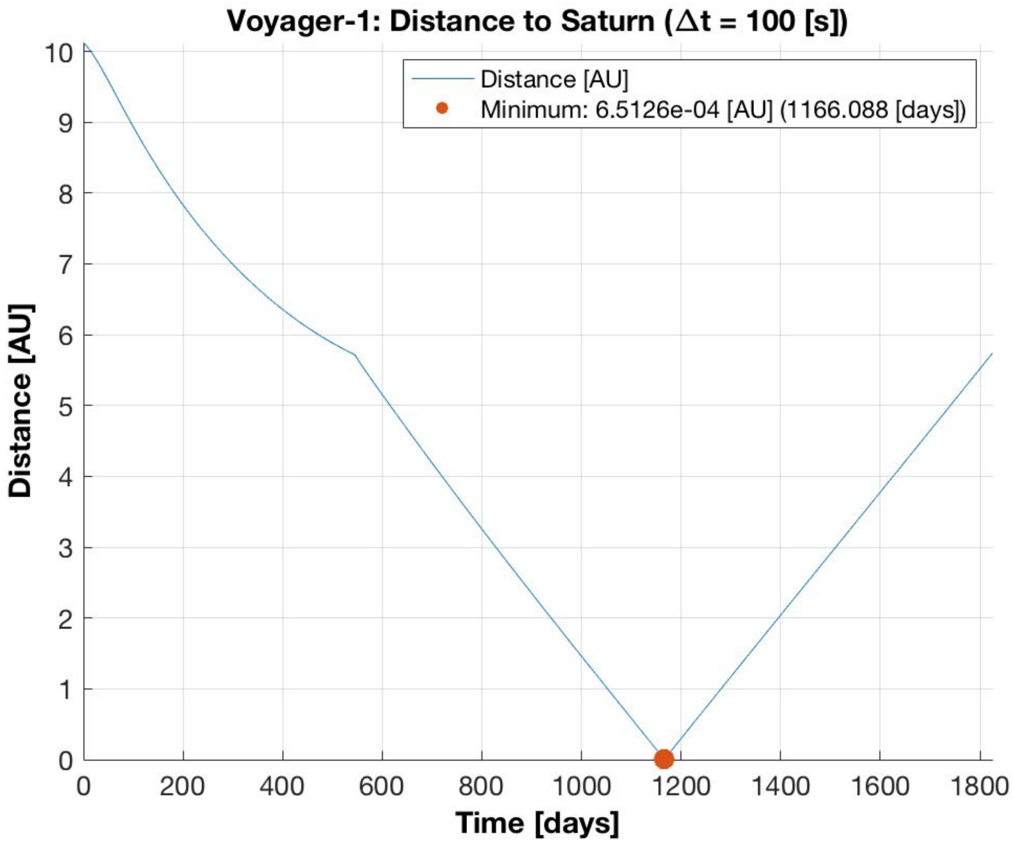


Resultados

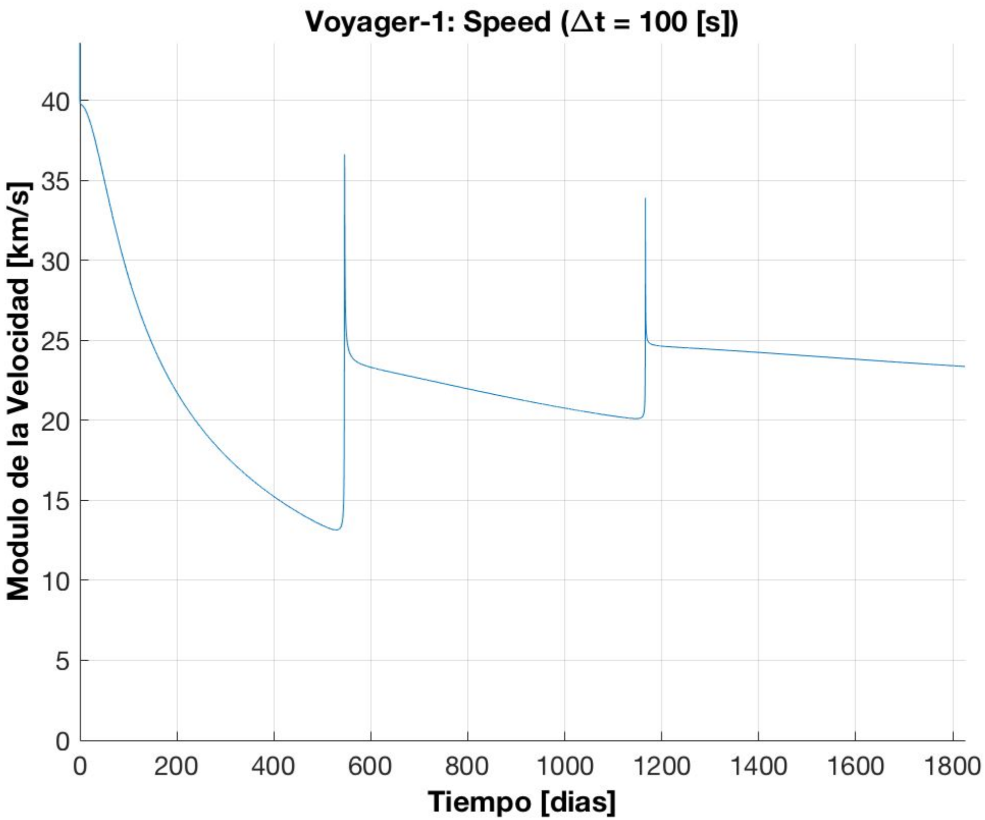
Distancia mínima y tiempo en alcanzar Júpiter



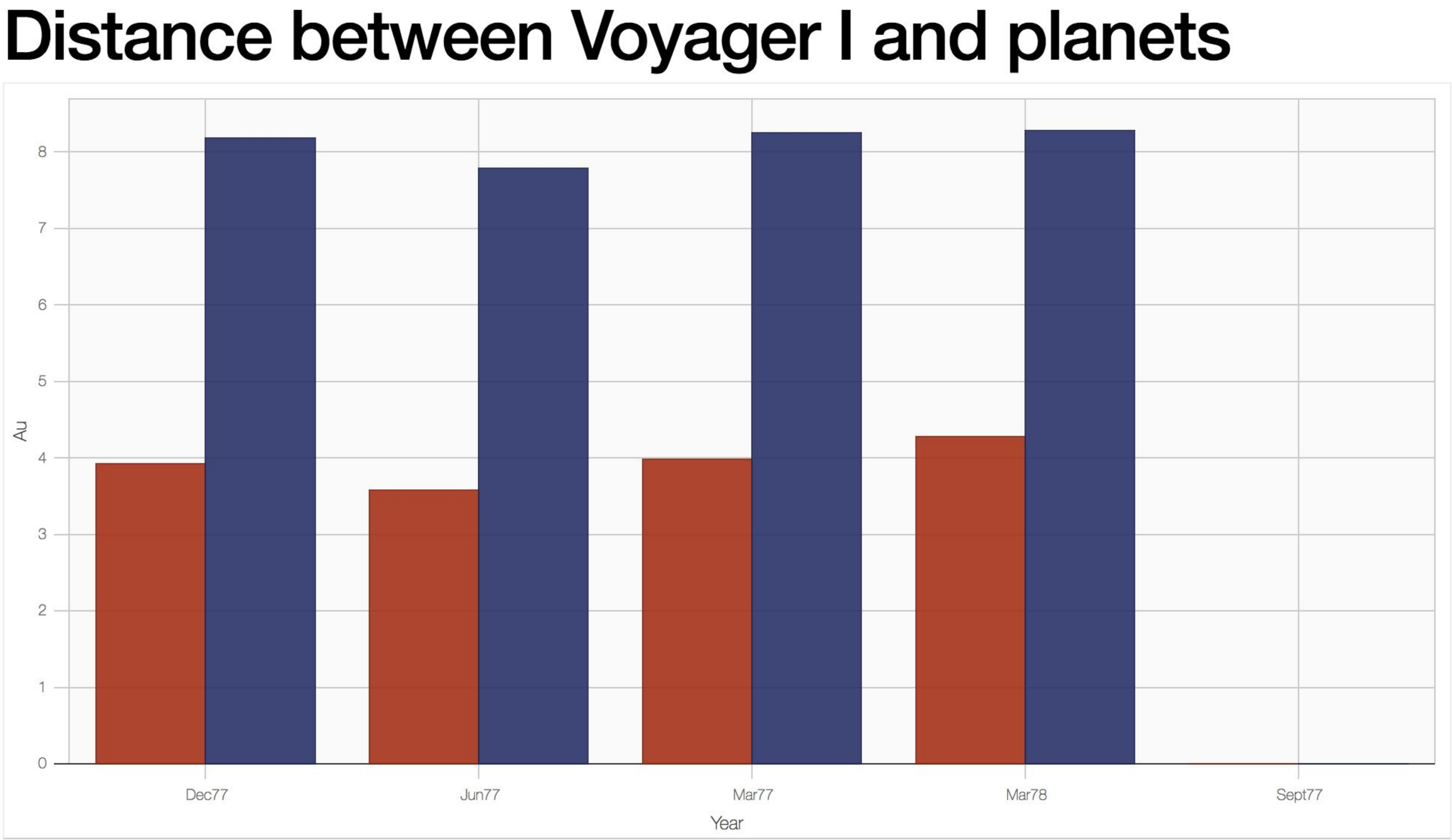
Distancia mínima y tiempo en alcanzar Saturno



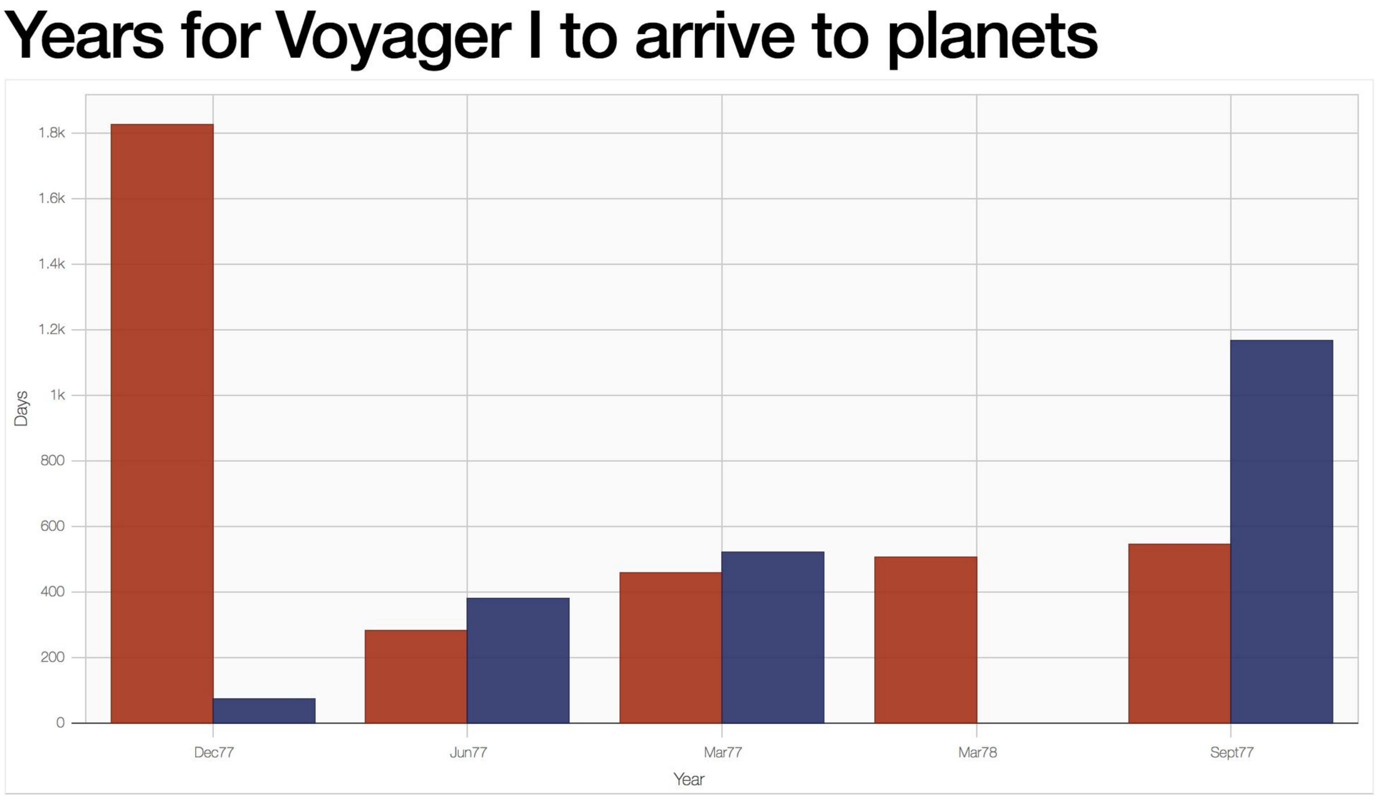
Módulo de la velocidad de la sonda en función del tiempo



Comparación de fechas para AU



Comparación de fechas para tiempo en alcanzar el planeta



Conclusiones

Conclusiones

- Es importante tomar bien las unidades y la precisión de todos los elementos en juego.
- Se priorizo para el Voyager I su cercanía a los planetas.
- Es importante tomar un delta_t adecuado
- Es prioritario realizar bien los métodos de integración.

Gracias!

Grupo 5: Golmar & Lobo