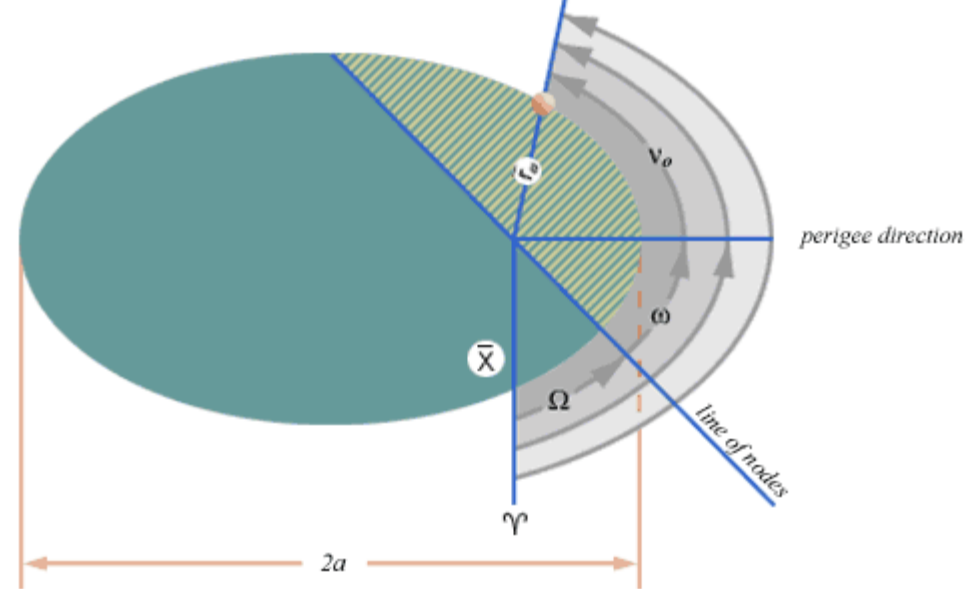
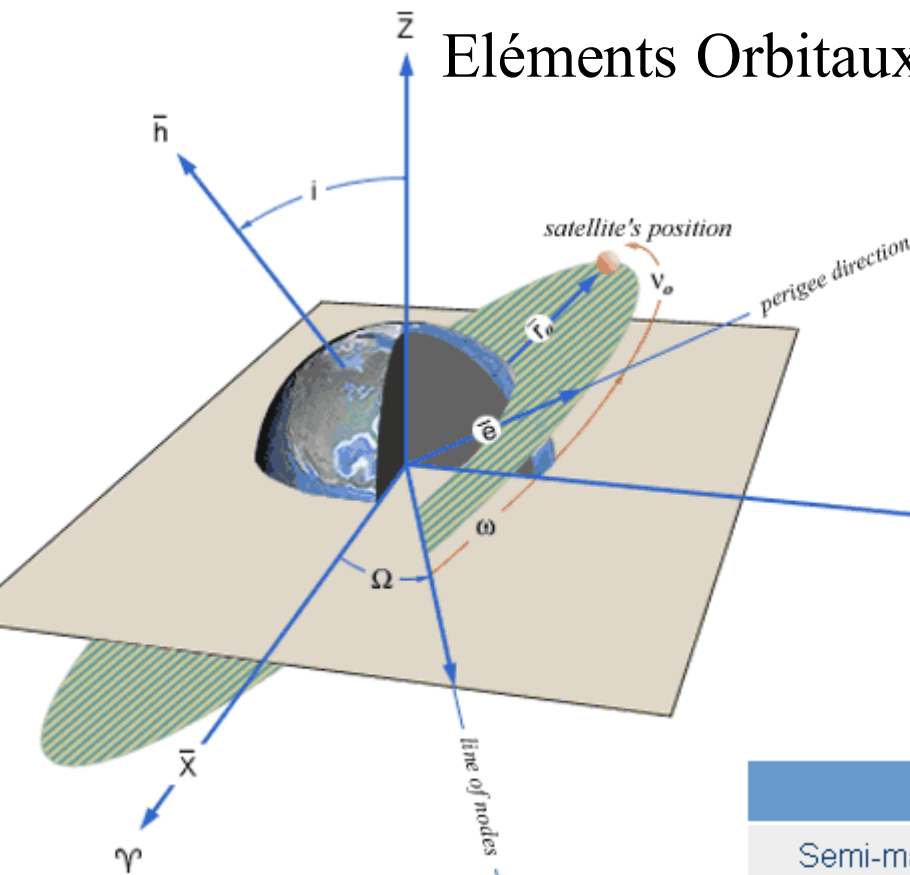


# Notions d'Astronomie

Détermination de position de planètes à l'aide des paramètres orbitaux

Travail à réaliser : représentation graphique des positions des planètes à l'aide de la librairie g2

# Eléments Orbitaux d'un satellite autour de la Terre

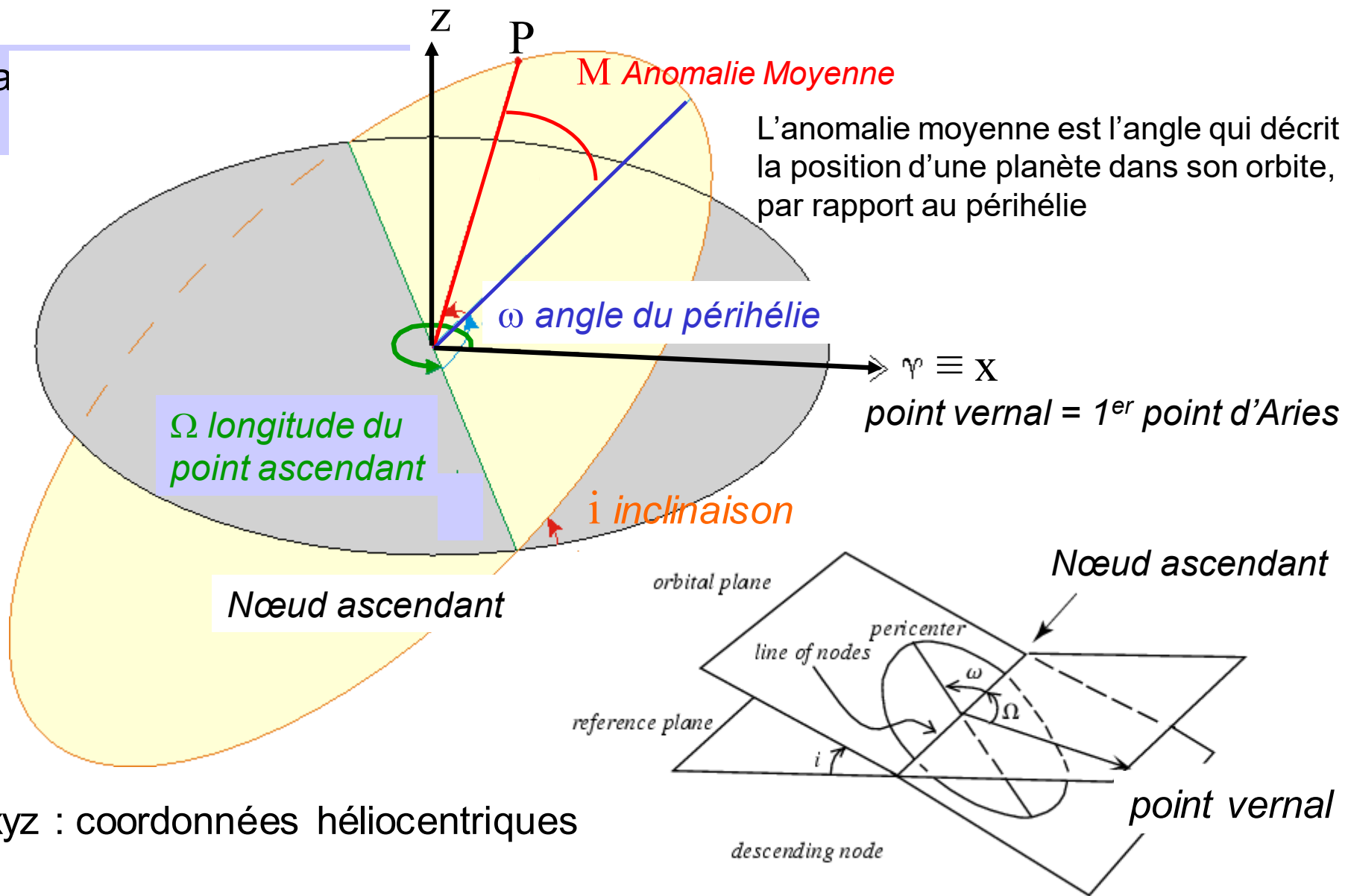


*point vernal :  
position du Soleil sur la sphère  
céleste au moment de  
l'équinoxe du printemps.*

Orbital Elements :		
Semi-major axis	$a$	Defines the size of the orbit.
Eccentricity	$e$	Defines the shape of the orbit. $\sqrt{1 - \frac{b^2}{a^2}}$
Inclination	$i$	Defines the orientation of the orbit with respect to the Earth's equator.
Argument of Perigee	$\omega$	Defines where the low point, perigee, of the orbit is with respect to the Earth's surface.
Right Ascension of the Ascending Node	$\Omega$	Defines the location of the ascending and descending orbit locations with respect to the Earth's equatorial plane.
True/Mean Anomaly	$v$	Defines where the satellite is within the orbit with respect to perigee.

# Eléments Orbitaux d'une planète P autour du Soleil

Pla  
de



# Calcul des positions des planètes à partir de leurs éléments orbitaux

<http://home.att.net/~srschmitt/planetorbits.html>

Référence à lire !

<http://scienceworld.wolfram.com/physics/OrbitalElements.html>

**Planetary Mean Orbits (J2000)**  
(epoch = J2000 = 2000 January 1.5)  
(see the table of rates below)

$\omega$  angle du périhélie  $M$  Anomalie Moyenne

Planet (mean)	a AU	e	i deg	$\Omega$ longitude du point ascendant en degrés	$\bar{\omega} = \Omega + \omega$ deg	$L = \bar{\omega} + M$ deg
Mercury	0.38709893	0.20563069	7.00487	48.33167	77.45645	252.25084
Venus	0.72333199	0.00677323	3.39471	76.68069	131.53298	181.97973
Earth	1.00000011	0.01671022	0.00005	-11.26064	102.94719	100.46435
Mars	1.52366231	0.09341233	1.85061	49.57854	336.04084	355.45332
Jupiter	5.20336301	0.04839266	1.30530	100.55615	14.75385	34.40438
Saturn	9.53707032	0.05415060	2.48446	113.71504	92.43194	49.94432
Uranus	19.19126393	0.04716771	0.76986	74.22988	170.96424	313.23218
Neptune	30.06896348	0.00858587	1.76917	131.72169	44.97135	304.88003
Pluto	39.48168677	0.24880766	17.14175	110.30347	224.06676	238.92881

$\bar{\omega} = \Omega + \omega$  est un angle « cassé »

# Avant-Projet : représentation graphique des positions des planètes avec la librairie g2 + une librairie maison

```
typedef                                                                                               astro.h
struct OrbitalElement {    // orbital parameters (J2000 epoch)
    double a;               // semi-major axis [AU]
    double e;               // eccentricity of orbit
    double i;               // inclination of orbit [deg]
    double O;               // longitude of the ascending node [deg]
    double w;               // longitude of perihelion [deg]
    double L;               // mean longitude [deg]
} OrbitalElement;

typedef
struct HelioCoord {
    double x, y, z;
} HelioCoord;

typedef
struct Astre
{
    char name[256];
    double mass;
    OrbitalElement orbitJ2000;    // orbital parameters (J2000 epoch)
    OrbitalElement rateJ2000;    // deviation rates
    OrbitalElement orbit;
    HelioCoord pos;
} Astre;
```

```
#include <cstdlib>
#include <iostream>
#include <cmath>
#include <cassert>
#include <iomanip>
#include "astro.h"
```

Constantes prédéfinies et prototypes des fonctions de la librairie maison

```
double DayNumber( int y, int m, int d, int h, int m );
void InitAstreDB(Astre *astre);
void GetHelioCoord(double d, Astre *astre);
void GetOrbitalElements(double d, Astre *astre);
double TrueAnomaly( double M, double e);
```

```
using namespace std;
```

```
main()
```

```
{
```

```
// declaration d'un tableau d'astres
```

```
Astre astre[10];
```

```
// remplissage de la base de donnees
```

```
InitAstreDB(astre);
```

```
int year=2005, month=5, day=1, hour=0, mins=0;
```

1/05/2005  
à 0h00

```
// calcul du nombre de jours depuis J2000
```

```
double d = DayNumber( year, month, day, hour, mins );
```

```
// calcul de la position de chacune des planetes
```

```
for (int p = 0; p<10; p++) {
```

```
    Astre *obj=&astre[p];
```

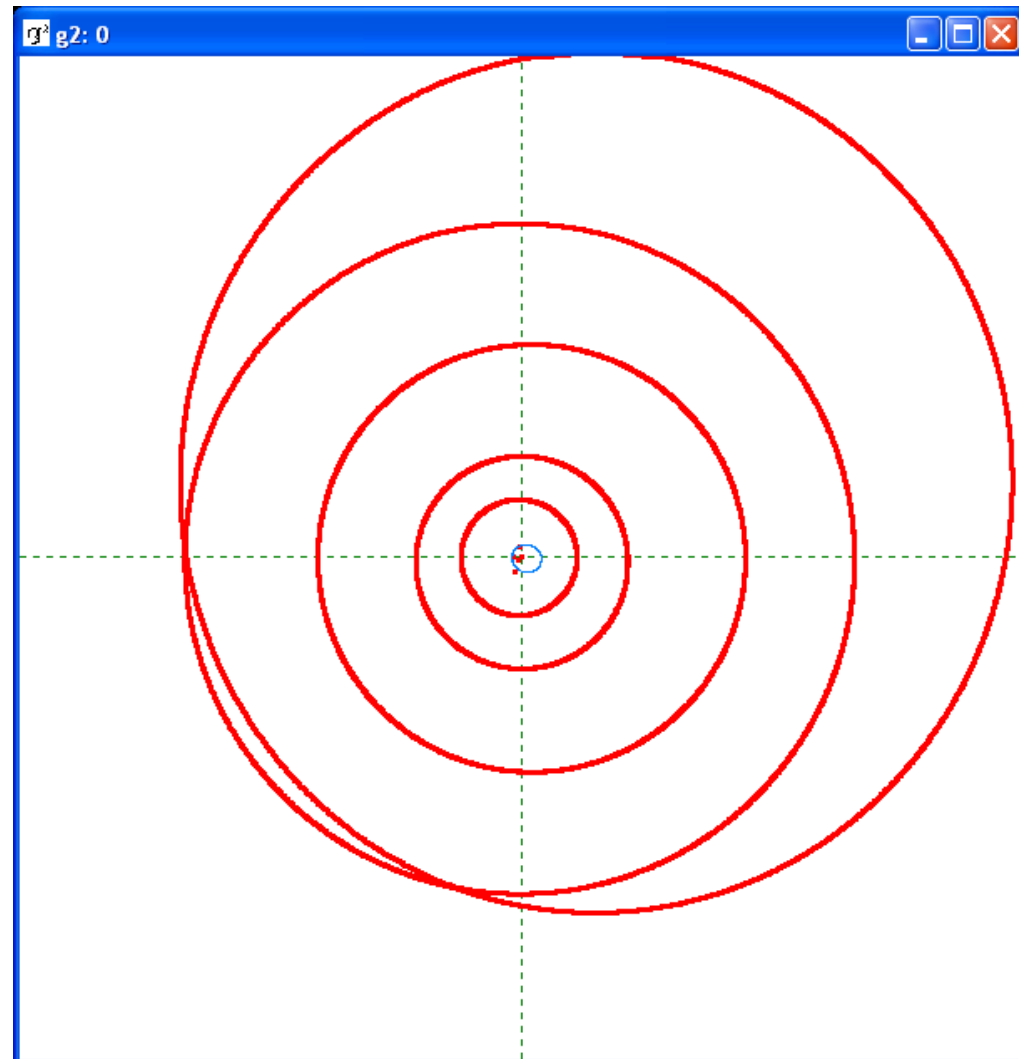
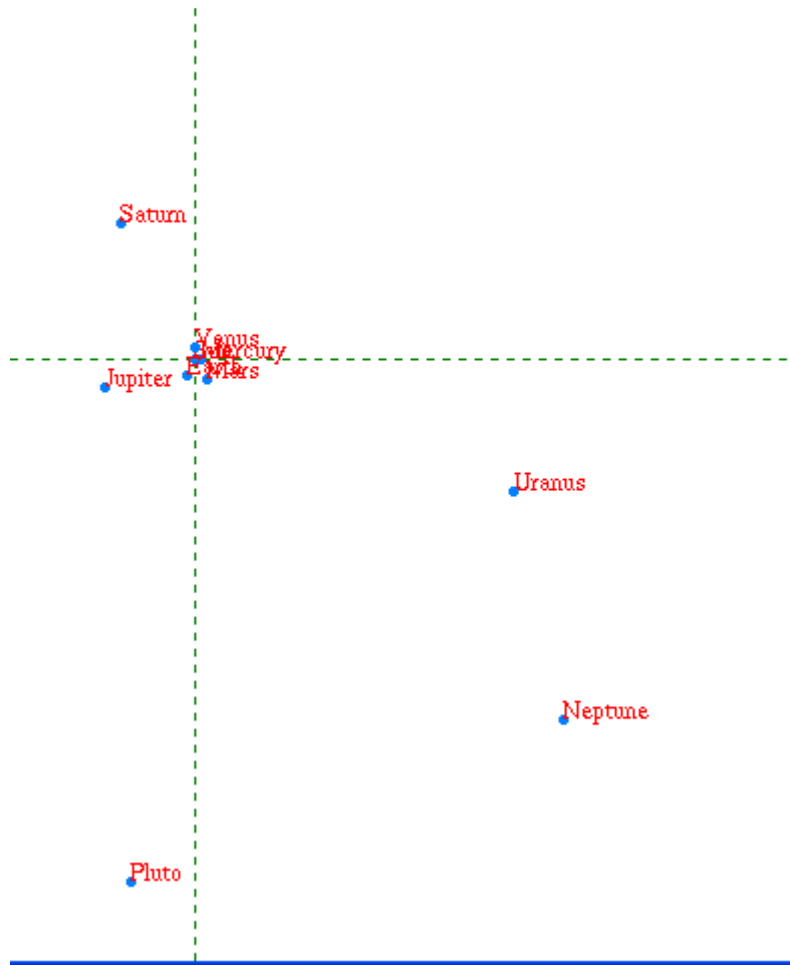
```
    GetHelioCoord(d, obj);
```

```
}
```

```
}
```

Expliquer l'intérêt de passer le pointeur obj en paramètre

Date 21/05/2005



Question : déterminer la période récente pendant laquelle Pluton est plus proche du Soleil que Neptune ?

# Simulation de la trajectoire d'une sonde dans le système solaire

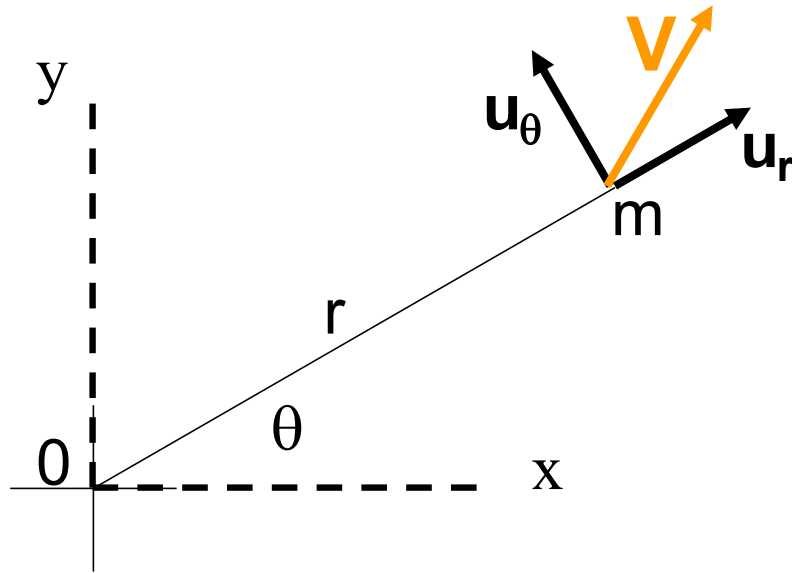
Introduction de la librairie gsl et application à la  
dynamique d'un ressort

Rappel sur des notions sur la gravitation universelle

Travail à réaliser : dynamique d'une sonde soumise  
aux interactions gravitationnelles des planètes



# Dynamique d'une masse attachée à un ressort



$$\frac{d^2 \mathbf{OM}}{dt^2} = -\omega_0^2 \mathbf{OM}$$

Pour transformer cette E.D.O. du 2<sup>nd</sup> ordre en un système du 1<sup>er</sup> ordre. On introduit la variable vitesse  $\mathbf{v}$

$$\begin{cases} \frac{d\mathbf{OM}}{dt} = \mathbf{v} \\ \frac{d\mathbf{v}}{dt} = -\omega_0^2 \mathbf{OM} \end{cases} \Rightarrow \frac{d}{dt} \begin{pmatrix} x \\ y \\ v_x \\ v_y \end{pmatrix} = \begin{pmatrix} v_x \\ v_y \\ -\omega_0^2 x \\ -\omega_0^2 y \end{pmatrix} \quad \text{de la forme} \quad \frac{d}{dt} \mathbf{y}_{vec} = \mathbf{f}(\mathbf{y}_{vec}, t)$$

Intégrable si conditions  
Initiales connues

$$\mathbf{y}_{vec}^0 = \mathbf{y}_{vec}(t=0) = (x_0 \quad y_0 \quad v_x^0 \quad v_y^0)^t$$

## Méthode la plus simple pour intégrer une E.D.O. : méthode d'Euler

- On discrétise le temps à pas constants tels que  $t = n.\Delta t$

- Pour intégrer  $\frac{d}{dt} \mathbf{y}_{vec} = \mathbf{f}(\mathbf{y}_{vec})$

on approxime la dérivation temporelle par  $\frac{d}{dt} \mathbf{y}_{vec} \approx \frac{\mathbf{y}_{n+1}^{vec} - \mathbf{y}_n^{vec}}{\Delta t}$

Le membre de droite est estimé à l'instant  $t = n.\Delta t$ .

On obtient finalement la suite récurrente :  $\mathbf{y}_{n+1}^{vec} = \mathbf{y}_n^{vec} + \mathbf{f}(\mathbf{y}_n^{vec}) \Delta t$

Avantage :

- facile à implanter.

Inconvénients :

- pas de contrôle d'erreurs lors de l'intégration.
- Précision faible.

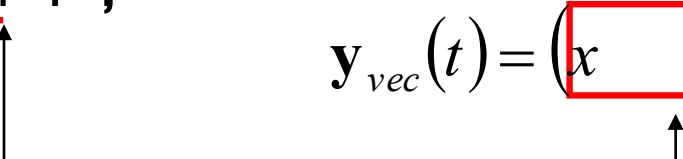
En pratique, utilisation de techniques d'intégration à pas liés avec estimation de l'erreur.

Sous \$HOME/projet2005/ressort/src, éditer ressort.cc

```
#include <stdio.h>
#include <gsl/gsl_errno.h>
#include <gsl/gsl_matrix.h>
#include <gsl/gsl_odeiv.h>
#include <gsl/gsl_ieee_utils.h>
```

Fichiers header (include) contenant les prototypes des fonctions de GSL et des constantes prédéfinies

```
/* DIM dimension du systeme d'equations */
const int DIM=4 ;
```

$$\mathbf{y}_{vec}(t) = \begin{pmatrix} x & y & v_x & v_y \end{pmatrix}^t$$


```
/* limite supérieure du pas d'intégration */
const double hmax=0.01 ;
```

On fixe ici une limite supérieure au pas d'intégration  $\Delta t$

```

/* ecriture du systeme
   d'equations differentielles du 1er ordre */

/* eq du ressort en 2D sous la forme dyvec/dt = fvec */

int fonction (double t, const double *yvec, double *fvec,
              void *params)
{
    double x =yvec[0];
    double y =yvec[1];
    double vx=yvec[2];
    double vy=yvec[3];

    fvec[0] = vx;
    fvec[1] = vy;
    fvec[2] = -x;
    fvec[3] = -y;

    return GSL_SUCCESS;
}

```

$$\frac{d}{dt} \begin{pmatrix} x \\ y \\ v_x \\ v_y \end{pmatrix} = \begin{pmatrix} v_x \\ v_y \\ -\omega_0^2 x \\ -\omega_0^2 y \end{pmatrix}$$

En prenant

$$\omega_0 = 1$$

**= 1**

Pour contrôler la précision de calcul, la méthode utilisée ici nécessite la définition de la matrice Jacobienne

$$\text{à partir de } \frac{d}{dt} \begin{pmatrix} x \\ y \\ v_x \\ v_y \end{pmatrix} = \begin{pmatrix} v_x \\ v_y \\ -\omega_0^2 x \\ -\omega_0^2 y \end{pmatrix} = \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \end{pmatrix} \quad [J] = \begin{pmatrix} \frac{\partial f_0}{\partial x} & \frac{\partial f_0}{\partial y} & \frac{\partial f_0}{\partial v_x} & \frac{\partial f_0}{\partial v_y} \\ \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} & \frac{\partial f_1}{\partial v_x} & \frac{\partial f_1}{\partial v_y} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} & \frac{\partial f_2}{\partial v_x} & \frac{\partial f_2}{\partial v_y} \\ \frac{\partial f_3}{\partial x} & \frac{\partial f_3}{\partial y} & \frac{\partial f_3}{\partial v_x} & \frac{\partial f_3}{\partial v_y} \end{pmatrix}$$

Dans le cas du modèle de ressort traité ici, elle s'écrit :

$$[J] = \begin{pmatrix} J_{00} & J_{01} & J_{02} & J_{03} \\ J_{10} & J_{11} & J_{12} & J_{13} \\ J_{20} & J_{21} & J_{22} & J_{23} \\ J_{30} & J_{31} & J_{32} & J_{33} \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -\omega_0^2 & 0 & 0 & 0 \\ 0 & -\omega_0^2 & 0 & 0 \end{pmatrix}$$

```
/* définition de la matrice Jacobienne */
```

```
int jacobien (double t, const double *yvec, double *dfdy,  
             double *dfdt, void *params)  
{
```

Astuce préconisée par les concepteurs de la librairie gsl

```
    gsl_matrix_view dfdy_mat =  
        gsl_matrix_view_array (dfdy, DIM, DIM);
```

```
    gsl_matrix *J = &dfdy_mat.matrix;
```

```
    gsl_matrix_set_zero(J);
```

```
    gsl_matrix_set (J, 0, 2, +1.0);
```

```
    gsl_matrix_set (J, 1, 3, +1.0);
```

```
    gsl_matrix_set (J, 2, 0, -1.0);
```

```
    gsl_matrix_set (J, 3, 1, -1.0);
```

avec  $\omega_0 = 1$

```
    ...  
    return GSL_SUCCESS;
```

```
}
```

$$[J] = \begin{pmatrix} J_{00} & J_{01} & J_{02} & J_{03} \\ J_{10} & J_{11} & J_{12} & J_{13} \\ J_{20} & J_{21} & J_{22} & J_{23} \\ J_{30} & J_{31} & J_{32} & J_{33} \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -\omega_0^2 & 0 & 0 & 0 \\ 0 & -\omega_0^2 & 0 & 0 \end{pmatrix}$$

```
main()
```

```
{
```

```
/* choix de la méthode d'intégration ici Runge-Kutta ordre 8 */
```

```
const gsl_odeiv_step_type *T = gsl_odeiv_step_rk8pd;
```

```
/* INTERNE à gsl */
```

```
gsl_odeiv_step *s = gsl_odeiv_step_alloc (T, DIM);
```

```
/* Controle de l'erreur relative à chaque pas ici  $10^{-6}$  */
```

```
gsl_odeiv_control *c = gsl_odeiv_control_y_new (0., 1.e-6);
```

```
/* INTERNE à gsl */
```

```
gsl_odeiv_evolve *e = gsl_odeiv_evolve_alloc (DIM);
```

```
/* Remplissage de la variable structure sys avec  
les fonctions servant au calcul de f et de [J] */
```

```
gsl_odeiv_system sys = {fonction, jacobien, DIM, NULL};
```

```
/* INTERNE à gsl */
```

```
gsl_ieee_env_setup();
```

```
/* temps initial et temps final */
```

```
double t = 0.0, tfin = 10.0;
```

```
/* pas initial d'integration des equations diff */
```

```
double h = hmax;
```

```
/* conditions initiales {x0 y0 x0' y0'} */
```

```
double y[DIM] = { 1.0, 1.0, 0., 0. };
```

```
for (;;) {
```

```
    int status = gsl_odeiv_evolve_apply (e, c, s, &sys,  
                                         &t, tfin,  
                                         &h, y);
```

Intégration  
temporelle

```
    if (status != GSL_SUCCESS) break;
```

```
/* limite sup du pas d'integration fixee par l'utilisateur */
```

```
if (h>hmax) h=hmax;
```

```
printf("%lf \t %lf \t %lf \t %lf\n", t, h, y[0], y[1]);
```

```
if (t>=tfin) break;
```

```
}
```

```
gsl_odeiv_evolve_free(e);  
gsl_odeiv_control_free(c);  
gsl_odeiv_step_free(s);
```

Libération de la mémoire

```
}
```



# gsl: GNU scientific library : librairie mathématique

- But : l'utiliser pour intégrer un système d'équations différentielles

```
cd $HOME/projet2005/ressort/src  
make clean; make all
```

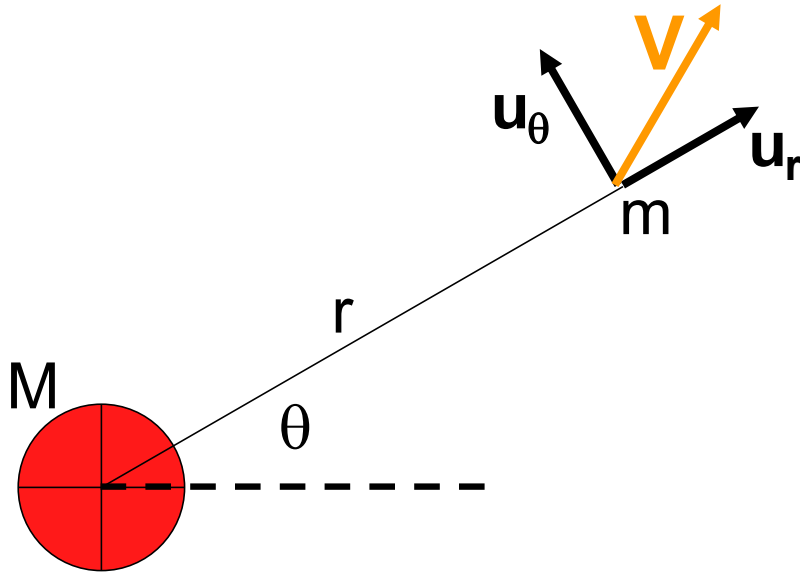
```
# Makefile  
CC  = gcc -g  
OBJ  = main.o  
  
LIBS = -lgsl -lgslcblas -lm  
INCS =  
BIN  = ressort  
  
all: $(BIN)  
  
clean:  
    rm -f $(OBJ) $(BIN)  
  
$(BIN): $(OBJ)  
    $(CC) $(INCS) $(OBJ) -o $(BIN) $(LIBS)  
  
.c.o:  
    $(CC) $(INCS) -c $<
```

Librairies de GSL

**\$(BIN) = valeur de la  
variable BIN**

make all ⇒ lance la création du  
fichier exécutable ressort

# Notions sur la Gravitation Universelle



Equation de la dynamique :

$$m \frac{d\mathbf{v}}{dt} = -m \frac{G M}{r^2} \mathbf{u}_r \quad \text{où} \quad \mathbf{v} = \frac{d\mathbf{r}}{dt}$$

$$\frac{d\mathbf{v}}{dt} = -\frac{G M}{r^2} \mathbf{u}_r$$

Rem : la cinématique du point  $m$  ne dépend pas de sa masse

$$G = 6.67 \cdot 10^{-11} \text{ N.m}^2/\text{kg}^2 \quad M(\text{soleil}) = 1.989 \cdot 10^{30} \text{ kg}$$

$$1 \text{ Unité astronomique} = 1 \text{ distance Terre-Soleil} = 149.6 \cdot 10^9 \text{ m}$$

$$\mathbf{v} = \frac{d\mathbf{r}}{dt} = \frac{d}{dt}(r \mathbf{u}_r) \quad \text{où} \quad \mathbf{r}(t) = r(\cos \theta(t) \mathbf{u}_x + \sin \theta(t) \mathbf{u}_y)$$

$$\mathbf{v} = \frac{d\mathbf{r}}{dt} = \dot{r} \mathbf{u}_r + r \frac{d\mathbf{u}_r}{d\theta} \dot{\theta} = \dot{r} \mathbf{u}_r + r \dot{\theta} \mathbf{u}_\theta$$

$$\boldsymbol{\gamma} = \frac{d\mathbf{v}}{dt} = (\ddot{r} - r \dot{\theta}^2) \mathbf{u}_r + (2 \dot{r} \dot{\theta} - r \ddot{\theta}) \mathbf{u}_\theta$$

Après projection

$$\textcircled{1} \left\{ \begin{array}{l} \ddot{r} - r \dot{\theta}^2 = -\frac{GM}{r^2} \end{array} \right.$$

$$\textcircled{2} \left\{ \begin{array}{l} 2 \dot{r} \dot{\theta} - r \ddot{\theta} = 0 \end{array} \right.$$

$$\textcircled{2} \Rightarrow \frac{d}{dt}(r^2 \dot{\theta}) = 0 \Rightarrow$$

Loi des aires  $r^2 \dot{\theta} = C^{te} = C_0$  3

La loi des aires exprime en fait la conservation du moment cinétique

$\boldsymbol{\sigma} = m \mathbf{r} \times \mathbf{v} = \mathbf{C}^{te}$

Transformons l'équation ① en introduisant la variable  $u=1/r$  et en Remarquant que  $r$  est une fonction de  $\theta(t)$

D 'après la loi des aires  $\dot{\theta} = C_0 / r^2 = C_0 u^2$

On a 
$$\dot{r} = \frac{dr}{dt} = \frac{dr}{d\theta} \dot{\theta} = \frac{dr}{d\theta} \frac{C_0}{r^2} = -C_0 \frac{du}{d\theta}$$

De même 
$$\ddot{r} = \frac{d\dot{r}}{dt} = \frac{d\dot{r}}{d\theta} \dot{\theta} = \frac{d}{d\theta} \left( -C_0 \frac{du}{d\theta} \right) C_0 u^2 = -C_0^2 u^2 \frac{d^2 u}{d\theta^2}$$

L'équation ① devient : 
$$-C_0^2 u^2 \frac{d^2 u}{d\theta^2} - C_0^2 u^3 = -G M u^2$$

$$\frac{d^2 u}{d\theta^2} + u = \frac{G M}{C_0}$$

Formule de Binet

Solution de l'équation de Binet 
$$\frac{d^2 u}{d\theta^2} + u = \frac{GM}{C_0}$$

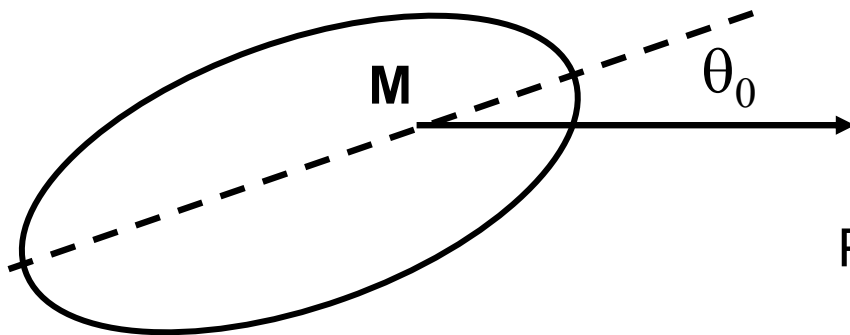
Solution générale sans 2<sup>nd</sup> membre :  $u = A \cos(\theta - \theta_0)$

Solution particulière :  $u_0 = \frac{GM}{C_0}$

Solution complète : 
$$u = A \cos(\theta - \theta_0) + u_0 = \frac{1}{p} (1 + e \cos(\theta - \theta_0))$$

C'est une ellipse dont un des foyers est occupé par M

Pour  $\theta = \theta_0$ , m se trouve au périhélie



$$u(\theta_0) = \frac{1+e}{p}$$

Pour  $\theta = \pi + \theta_0$ , m se trouve à l'aphélie

$$u(\pi + \theta_0) = \frac{1-e}{p}$$

Conservation de l'énergie totale

$$E = T + V = \frac{1}{2} m v^2 - \frac{GMm}{r} = C^{te}$$

Seulement à périhélie et à l'aphélie, on a  $\mathbf{v} = \cancel{\dot{r}} \mathbf{u}_r + r \dot{\theta} \mathbf{u}_\theta$

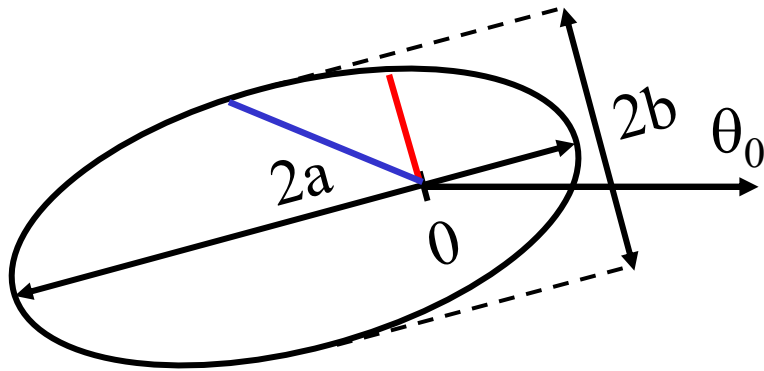
De plus, d'après la loi des aires  $r \dot{\theta} = C_0 u$ , on obtient :

$$\frac{1}{2} C_0^2 u^2(\theta_0) - GM u(\theta_0) = \frac{1}{2} C_0^2 u^2(\pi + \theta_0) - GM u(\pi + \theta_0)$$

Après factorisation  $u(\pi + \theta_0) + u(\theta_0) = \frac{2GM}{C_0^2} = \frac{2}{p}$

étant donnés  $u(\pi + \theta_0) = \frac{1-e}{p}$  et  $u(\theta_0) = \frac{1+e}{p}$

$$\Rightarrow \boxed{p = \frac{C_0^2}{GM}}$$



$$u = \frac{1}{r} = \frac{1}{p} (1 + e \cos(\theta - \theta_0))$$

$$a = \frac{p}{1 - e^2} \quad \text{et} \quad b = \frac{p}{\sqrt{1 - e^2}}$$



L'aire balayée par seconde  
est donnée par

$$\frac{dS}{dt} = \frac{1}{2} r^2 \dot{\theta} = \frac{C_0}{2}$$

Intégrée sur 1 période de révolution

$$S = \frac{C_0}{2} T = \pi a b = \pi a^2 \sqrt{1 - e^2} \quad \Rightarrow \quad T = \frac{2\pi}{C_0} a^2 \sqrt{1 - e^2}$$

Elevons l'expression de T au carré et remplaçons  $C_0^2 = GMp$

$$T^2 = \frac{4\pi^2}{C_0^2} a^4 (1 - e^2) = \frac{4\pi^2}{GM} a^3 \quad \Rightarrow \quad T^2 = \frac{4\pi^2}{GM} a^3$$