Introduction to programming with Matlab/Python — Lecture 3

Brian Thorsbro

Department of Astronomy
and Theoretical Physics,
Lund University, Sweden.

Monday, November 6, 2017

Courses

These lectures are a mini-series companion to:

ASTM13 Dynamical Astronomy

ASTM21 Statistical tools in Astrophysics

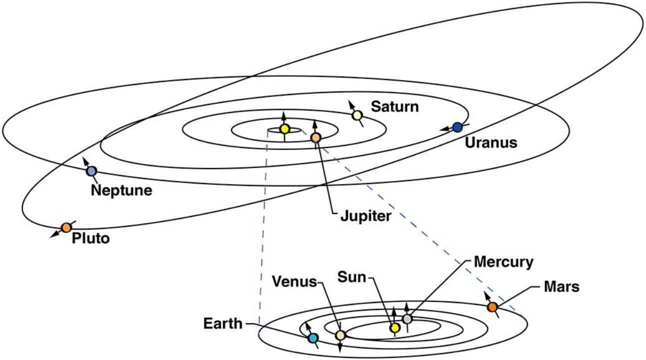
Matlab installed in the lab (Lyra). Personal laptops are OK!
Install Matlab from: http://program.ddg.lth.se/
Install Python3 from: https://www.anaconda.com/download/

Outline

Content in this lecture:

- Solar system orbits
  (example computational task)
- Integration algorithm
- State Machine

The Solar System

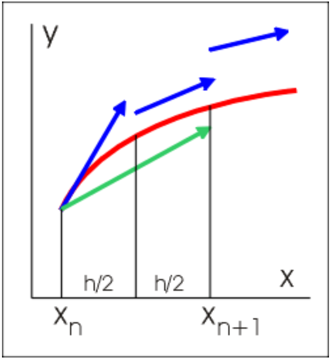


$$\text{Law of Universal Gravitation: } \hat{F}_{12} = -\hat{F}_{21} = -\frac{GmM(\hat{r}_2 - \hat{r}_1)}{|\hat{r}_2 - \hat{r}_1|^3},$$

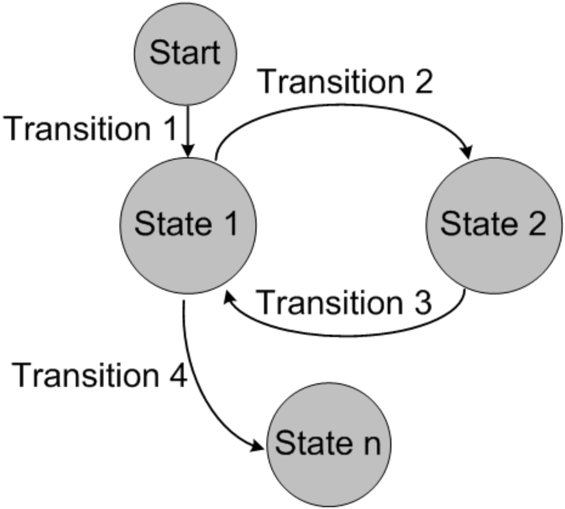
$$\text{Equation of motion: } \hat{F} = m\hat{a}.$$

Integration

Multi-body systems to complicated to integrate analytically. Numerical solvers needed with discrete time steps. An example algorithm is Runge-Kutta of the 4^{th} order.



The state machine



Designing the state

The state at one time step, 5 data points for each planet (Sun included):

- x - positions on x axis
- vx - velocity in x direction
- y - positions on y axis
- vy - velocity in y direction
- m - mass of object

Matlab Initial State

```
% planet configurations
% RS: radius of orbit in AU
% Ks: mass as ratio of sun (here actually earth mass, but will normalize just below)
% Vs: initial speed modifier to get ellipeses
% Ps: plot style for the planet
%          sun ; mercu ; venus ; earth ;  mars ; jupi ; satu ;
Rs = [       0 ;   0.4 ;   0.7 ;     1 ;   1.5 ;  5.2 ;  9.5  ];
Ks = [ 332946 ; 0.055 ; 0.815 ;     1 ; 0.107 ;  318 ;   95  ];
Vs = [       0 ;  1.05 ;  1.02 ;  1.02 ;  1.02 ; 1.02 ; 1.02  ];
Ps = {    'xr' ;  '-g' ;  '-g' ;  '-b' ;  '-r' ; '-k' ; '-k'  };

% normalize mass to solar mass
Ks = Ks / Ks(1);

% initial value settings
InitState = zeros(5*length(Rs),1);
for i=0:length(Rs)-1
    if Rs(i+1)==0 % special "sun" support
        initial_speed = 0;
    else
        initial_speed = (2*pi*sqrt((Ks(1)+Ks(i+1))/Rs(i+1)))*Vs(i+1);
    end
    InitState(i*5+1) = Rs(i+1);         % x
    InitState(i*5+2) = 0.0;             % vx
    InitState(i*5+3) = 0.0;             % y
    InitState(i*5+4) = initial_speed;   % vy
    InitState(i*5+5) = Ks(i+1);         % mass
end
```

Python Initial State

```
from numpy import *
from matplotlib.pyplot import *

# planet configurations
# RS: radius of orbit in AU
# Ks: mass as ratio of sun (here actually earth mass, but will normalize just below)
# Vs: initial speed modifier to get ellipeses
# Ps: plot style for the planet
#         sun ; mercu ; venus ; earth ;  mars ; jupi ; satu ;
Rs = array([      0.0 ,   0.4 ,   0.7 ,   1.0 ,   1.5 ,  5.2 ,  9.5  ])
Ks = array([ 332946.0 , 0.055 , 0.815 ,   1.0 , 0.107 ,  318 , 95.0  ])
Vs = array([      0.0 ,  1.05 ,  1.02 ,  1.02 ,  1.02 , 1.02 , 1.02  ])
Ps = array([     'xr' ,  '-g' ,  '-g' ,  '-b' ,  '-r' , '-k' , '-k'  ])

# normalize mass to solar mass
Ks = Ks / Ks[0]

# initial value settings
#InitState = zeros((5*size(Rs),1))
InitState = zeros(5*size(Rs))
for i in range(0,size(Rs)):
    if Rs[i]==0:  # special "sun" support
        initial_speed = 0
    else:
        initial_speed = (2*pi*sqrt((Ks[0]+Ks[i])/Rs[i]))*Vs[i]
    InitState[i*5+0] = Rs[i]           # x
    InitState[i*5+1] = 0.0             # vx
    InitState[i*5+2] = 0.0             # y
    InitState[i*5+3] = initial_speed   # vy
    InitState[i*5+4] = Ks[i]           # mass
```

Matlab Runge-Kutta

```
function [ Times, States ] = RK4(generate_ds, tspan, InitState, timestepsize)
    % set step size
    h = timestepsize; % renaming to short name

    % allocate memory
    estimatedcols = floor((tspan(2)-tspan(1)) / h) + 1;
    States = zeros(length(InitState),estimatedcols);
    Times = zeros(1,estimatedcols);
    idx = 1;

    % set the initial state
    States(:,1) = InitState;
    Times(1) = tspan(1);

    State = InitState;
    for t=tspan(1):h:tspan(2)
        % calculate new a value based on fourth order Runge-Kutta
        k1 = h * generate_ds(t, State);
        k2 = h * generate_ds(t+h/2, State+k1/2);
        k3 = h * generate_ds(t+h/2, State+k2/2);
        k4 = h * generate_ds(t+h, State+k3);
        State = State + (k1+2*k2+2*k3+k4)/6;

        % save the state
        idx = idx+1;
        States(:,idx) = State;
        Times(idx) = t+h;
    end
end
```

Python Runge-Kutta

```
# Runge-Kutta integrator of order 4
def RK4(generate_ds, tspan, InitState, timestepsize):

    # set step size
    h = timestepsize  # renaming to short name

    # allocate memory
    estimatedcols = int(floor((tspan[1]-tspan[0]) / h) + 1)
    States = zeros((size(InitState),estimatedcols))
    Times = zeros(estimatedcols)
    idx = 0

    # set the initial state
    States[:,0] = InitState
    Times[0] = tspan[0]

    State = InitState;
    for t in arange(tspan[0],tspan[1],h):
        # calculate new a value based on fourth order Runge-Kutta
        k1 = h * generate_ds(t, State)
        k2 = h * generate_ds(t+h/2, State+k1/2)
        k3 = h * generate_ds(t+h/2, State+k2/2)
        k4 = h * generate_ds(t+h, State+k3)
        State = State + (k1+2*k2+2*k3+k4)/6
        # save the state
        idx = idx+1
        States[:,idx] = State
        Times[idx] = t+h
    return Times,States
```

ODE to linear system

Runge-Kutta only do first order integration, but higher order ODEs can be converted to linear systems.

$$\hat{F} = m\hat{a} = m\frac{\mathrm{d}^2\hat{x}}{\mathrm{d}t^2}$$

$$\Downarrow$$

$$\begin{cases} \hat{a} = \frac{\mathrm{d}\hat{v}}{\mathrm{d}t} \\ \hat{v} = \frac{\mathrm{d}\hat{x}}{\mathrm{d}t} \end{cases}$$

Force is additive so all the forces affecting one planet are just summed.

Matlab linear system

```
function [ dS ] = dState( t, S )
% function for usage in Runge-Kutta

% allocating memory
dS = zeros(length(S),1);

% loop through planets to be updated
for i=0:length(S)/5-1
    % speed
    dS(i*5+1) = S(i*5+2);
    dS(i*5+3) = S(i*5+4);
    % prep for acceleration calc
    x = S(i*5+1);
    y = S(i*5+3);
    % loop through planets affecting the current planet
    for j=0:length(S)/5-1
        if j~=i
            k = S(j*5+5);
            px = S(j*5+1);
            py = S(j*5+3);
            d = sqrt((px-x).^2 + (py-y).^2);
            % acceleration
            dS(i*5+2) = dS(i*5+2) - 4*pi^2*k .* (x - px)./(d.^3);
            dS(i*5+4) = dS(i*5+4) - 4*pi^2*k .* (y - py)./(d.^3);
        end
    end
end
end
```

Python linear system

```
# set up the differential, function for usage in Runge-Kutta
def dState(t, S):
    # allocating memory
    dS = zeros(size(S))

    # loop through planets to be updated
    for i in range(0,int(size(S)/5)):
        # speed
        dS[i*5+0] = S[i*5+1]
        dS[i*5+2] = S[i*5+3]
        # prep for acceleration calc
        x = S[i*5+0]
        y = S[i*5+2]
        # loop through planets affecting the current planet
        for j in range(0,int(size(S)/5)):
            if j!=i:
                pk = S[j*5+4]
                px = S[j*5+0]
                py = S[j*5+2]
                d = sqrt((px-x)**2 + (py-y)**2)
                # acceleration
                dS[i*5+1] = dS[i*5+1] - 4*(pi**2)*pk * (x - px)/(d**3)
                dS[i*5+3] = dS[i*5+3] - 4*(pi**2)*pk * (y - py)/(d**3)
    return dS
```

Matlab putting it together

```
% the time span to run the simulation in years
tspan = [ 0 50 ];
timestepsize = 0.001;

% call the integrator
[ Times, States ] = RK4(@dState, tspan, InitState, timestepsize);

% plot planetary orbit
clf reset
figure(1);
hold on
for i=0:size(Rs,1)-1
    xidx = i*5+1;
    yidx = i*5+3;
    plot(States(xidx,:), States(yidx,:), Ps{i+1});
end
title('Planet orbits');
xlabel('x');
ylabel('y');
axis([-12 12 -12 12]);
axis square;
```

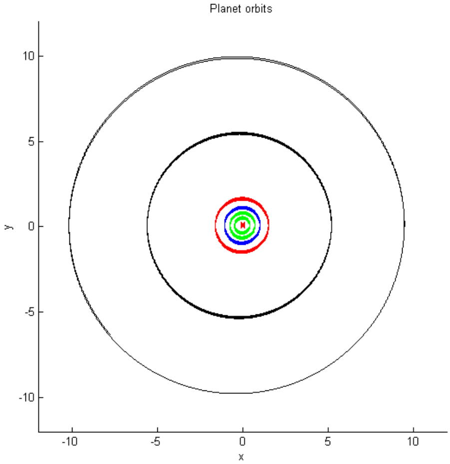
Python putting it together

```
# the time span to run the simulation in years
tspan = [ 0 , 50 ]
timestepsize = 0.001

# call the integrator
Times,States = RK4(dState, tspan, InitState, timestepsize)

# plot planetary orbit
figure(1)
for i in range(0,size(Rs)):
    xidx = i*5+0
    yidx = i*5+2
    plot(States[xidx,:], States[yidx,:], Ps[i], ms=1)
title('Planet orbits')
xlabel('x')
ylabel('y')
xlim(-12,12)
ylim(-12,12)
```

Result



The End

Questions?