TU Delft Delft University of Technology

Advanced Dynamics (wb2630-T1)

Q1 2014

Homework No. 8

Main Instructor: Dr.-Ing. Heike Vallery
Homework Coordinator: Dr. ir. Arend L. Schwab

Date: 20-Oct-2014 **Deadline:** Before lecture on Oct 27.

Instructions:

- Show all work, clearly and in order, if you want to get full credit.
- Follow the homework rules as published on Blackboard.
- Justify your answers algebraically whenever possible to ensure full credit. Sketch all relevant graphs.
- Circle or otherwise indicate your final answers.
- Clearly distinguish between scalars and vectors, for example by underlining vectors ($\underline{x}$) and not underlining scalars (x). **If you use another notation, define it like this**: A vector will be written as ....., a scalar will be written as .....
- Success!

1. (*35 points*) PENDULUM

Consider a point mass of mass m that is connected to a fixed point 0 via a massless rigid rod of length l (Fig.1), and which moves under the influence of gravity g. Assume the location of the point mass is described by the angle ϕ of the rigid rod with respect to the vertical, where ϕ is zero when the pendulum is at the bottom position and increases when the mass moves to the right from there. At time $t_0 = 0$, the pendulum has initial conditions $\phi(t_0) = 0$, $\dot{\phi}(t_0) = \frac{\pi}{2}\,\mathrm{s}^{-1}$ (Make sure to handle units correctly throughout the problem).

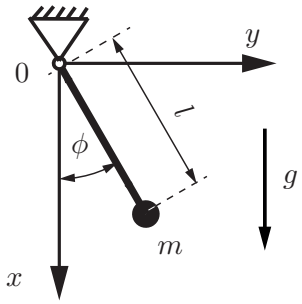

Figure 1: A pendulum consisting of a point mass and a massless rigid rod.

a. (*5 pts*) What is the maximum height the point mass will reach, measured from its bottom position, given the values $m = 2\,\mathrm{kg}$, $l = 9.81\,\mathrm{m}$, $g = 9.81\,\mathrm{m/s^2}$?

b. (*5 pts*) Derive the equations of motion for this system and solve for angular acceleration $\ddot{\phi}$ of the rod as a function of ϕ, $\dot{\phi}$, m, g,, and l using a method of your choice.

c. (*5 pts*) Make a small-angle assumption and solve for $\phi(t)$ analytically. Given the values $m = 2\,\mathrm{kg}$, $l = 9.81\,\mathrm{m}$, $g = 9.81\,\mathrm{m/s^2}$, find $\phi(t_e)$ for $t_e = 0.2\,\mathrm{s}$, with the above initial conditions.

d. (*3 pts*) Using the state vector $\boldsymbol{x} = \begin{pmatrix}\phi & \dot{\phi}\end{pmatrix}^T$, re-write your (original, nonlinear) equations of motion in the form of a system of first-order ordinary differential equations (ODE).

e. (*15 pts*) Numerically integrate this ODE system by hand with the same initial conditions, using a single step of Euler's method with step size $h = t_e - t_0 = 0.2\,\mathrm{s}$. Compare your result for $\phi(t_e)$ to the solution you found in the previous part of this problem. Which of the two results do you trust more?

f. (*2 pts*) Name two measures that could be taken (individually or in combination) to obtain a more accurate result by numerical integration.

2. (*30 points*) NUMERICAL INTEGRATION

First, read and understand the below example of the file `Integrate_ModifiedEuler.m` containing the modified Euler method:

```
function [tout,yout] = Integrate_ModifiedEuler(odefun,tspan,Ts,y0,par)
%[tout,yout] = Integrate_ModifiedEuler(odefun,tspan,Ts,y0,par)
%with tspan = [T0 TFINAL] integrates the system of differential equations
%dy/dt = f(t,y) from time T0 to TFINAL, with initial conditions
%specified by the row vector y0 and step size Ts,
%using the modified Euler method
%(D. Greenwood, "Advanced Dynamics", page 341).
%
%odefun must be a function handle to the function that calculates state
%derivatives dy/dt (as a column vector), when provided time t and a row or
%column vector y.
%Parameters can be passed on to odefun using the additional input par.
%
%Outputs:
%Each row in the solution array yout corresponds to a time
%returned in the column vector tout.
%
%usage resembles the Matlab function ode45
%H. Vallery, Oct 2014

%time vector (t_1 to t_numsteps):
tout=(tspan(1):Ts:tspan(end))';

numsteps=length(tout);%number of steps
numstates=length(y0);%number of states

%initialize outputs to allocate memory:
yout=zeros(numsteps,numstates);

%set initial values:
y_n=y0;%initial condition

for n=1:numsteps
%first generate outputs:
t_n=tout(n);
t_nplus1=t_n+Ts;
yout(n,:)=y_n;

%(1) calculate function derivative:
dy_n=feval(odefun,t_n,y_n',par)';%transpose output to produce row vector

%(2) calculate predictor:
ytilde_nplus1=y_n+Ts*dy_n;%integrate to obtain ytilde_{n+1}

%(3) calculate derivative of predicted states:
dytilde_nplus1=feval(odefun,t_nplus1,ytilde_nplus1',par)';

%(4) apply correction:
y_nplus1=y_n+.5*Ts*(dytilde_nplus1+dy_n);

%re-define n+1 back to n:
y_n=y_nplus1;
```

```
end
```

Now, implement also Euler's method and a 4th-order Runge-Kutta, in new functions with names `Integrate_Euler.m` and `Integrate_RungeKutta4.m`.
Integrate the sample ODE system of the "van der Pol" oscillator (with $\mu = 1$) for 15 seconds, with a step size of 0.01 s and initial conditions $\boldsymbol{y}_0 = \begin{pmatrix} 2 & 0 \end{pmatrix}^T$, using this command:

```
[t,y]=Integrate_Euler(@vanderpoldemo,[0 15],0.01,[2 0],1);%par=mu=1.
```

Report the final value of the state vector $\boldsymbol{y}$ for the three different solvers.

3. (*35 points*) MODEL PLANE

A model plane of mass $m = 1\,\text{kg}$ is suspended from the ceiling of a room (Fig. 2) by a spring with linear characteristics. The spring has a resting length of $l_0 = 1\,\text{m}$ and a spring constant $k = 20\,\text{N/m}$. The attachment point A at the ceiling is the origin of the space-fixed $\mathcal{N}$ (XYZ) coordinate system. The coordinates of the attachment point P on the plane are given as (.1 m, 0, -.1 m) in the local $\mathcal{B}$ (xyz) coordinate system that is fixed to the plane. The origin of the plane's local frame $\mathcal{B}$ is at the plane's center of mass location S. Type I Euler angles ψ, θ, ϕ (convention following Greenwood) describe the plane's orientation with respect to the $\mathcal{N}$-frame. Gravity $g = 9.81\,\text{m/s}^2$ points in positive Z-direction.

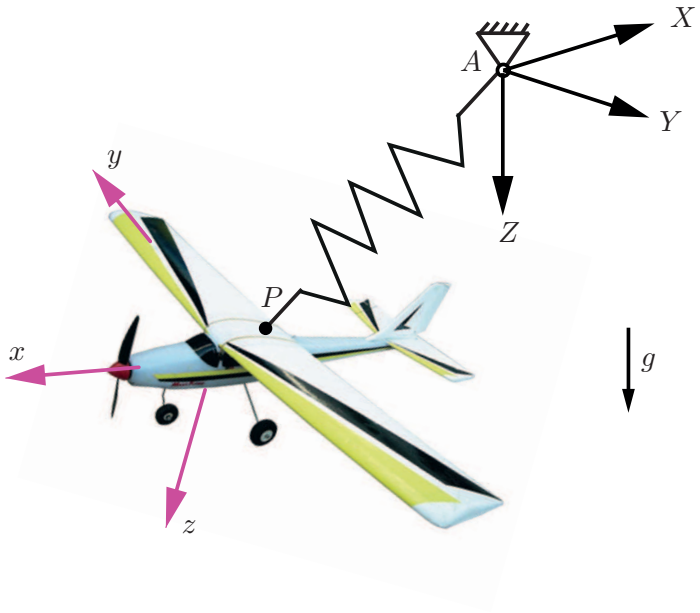

Figure 2: A model plane suspended from the ceiling.

Assume the plane's mass distribution is symmetric to the body-fixed xy and xz planes, and the mass moments of inertia about the body-fixed axes are $I_{xx} = I_{yy} = \frac{1}{2}I_p$, $I_{zz} = I_p$, with $I_p = .01\,\text{kgm}^2$. Use the Newton-Euler equations to find the linear accelerations of the plane's center of mass, expressed in the spaced-fixed frame, as well as angular acceleration, expressed in the body-fixed frame, as functions of the states and external forces acting on the plane.
Then, complete the code of the function `plane_equationsofmotion.m` (stated below), which calculates state derivatives. The function is called by the script `simulate_plane.m` (see below). Apply the solvers you implemented in the previous question to integrate the equations of motion for the given initial conditions, step size, and duration (as specified in the provided code).
By adding some lines of code at the end of `simulate_plane.m`, determine the distance between the final center of mass positions, as obtained with Euler's method and 4th-order Runge-Kutta. Also, calculate the principal angle of rotation between the two calculated final orientations (so between the final orientation obtained with Euler's method and the one obtained with Runge-Kutta).

This is the content of `plane_equationsofmotion.m` (missing code indicated by xxx):

```
function dx=plane_equationsofmotion(t,x,par)
%function dx=plane_equationsofmotion(t,x,par)
%This function contains the equations of motion of a model airplane
%connected to the ceiling by a linear spring, as used in homework
%assignments of the course "Advanced Dynamics" at TUD.
%
%Input: t (time) and
%x: Cartesian positions, Euler angles, linear speeds, omega
%Output: dx: derivatives thereof
%
%H. Vallery, October 2014

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%extract parameters:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

g=par.g;%[m/s^2], acceleration of gravity (points in positive z direction)

%spring parameters:
l0=par.l0;%m, resting length of spring
k=par.k;%N/m, stiffness of spring
rp_B=par.rp_B;%position vector of point P (spring attachment point) in B frame

%mass parameters of the plane:
m=par.m;% mass of plane
Ixx=par.Ixx;% moment of inertia about x axis
Iyy=par.Iyy;% moment of inertia about y axis
Izz=par.Izz;% moment of inertia about z axis

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%read current states:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

sX=x(1);
sY=x(2);
sZ=x(3);
psi=x(4);
theta=x(5);
phi=x(6);

dsX=x(7);
dsY=x(8);
dsZ=x(9);
omegax=x(10);
omegay=x(11);
omegaz=x(12);

rs_N=[sX;sY;sZ];%position of point S (center of mass) in N coordinates

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%calculate forces and moments in body frame:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%construction of rotation matrix that maps vectors that are
%expressed in the N frame to their representation in the B frame:
R_psi=xxx
R_theta=xxx
R_phi=xxx
R_total=xxx
```

```
%position vector of point P in the N frame:
rp_N=xxx

%external forces acting on the plane:
Fspring_N=xxx;%spring force in the N frame
Fg_N=[0;0;m*g];%gravitational force in the N frame
Ftot_N=(Fspring_N+Fg_N);%total force in the N frame

%map the spring force to the body frame:
Fspring_B=xxx;%spring force in the body frame

%calculate the external moments acting about the plane's center of mass:
M=cross(xxx);%moment of the spring in the body frame
%components about the body-fixed axes:
Mx=M(1);My=M(2);Mz=M(3);

%--------------------------------
%apply Newton-Euler:
%--------------------------------

%Newton equations (in space-fixed frame):
ddsX=xxx;
ddsY=xxx;
ddsZ=xxx;

%Euler equations (in body-fixed frame):
domegax= xxx;
domegay= xxx;
domegaz= xxx;

%--------------------------------
%calculate state derivatives:
%--------------------------------

%Euler angle derivatives, Greenwood (3.16)
dpsi=sec(theta)* (omegay*sin(phi)+omegaz*cos(phi));
dtheta = omegay*cos(phi)-omegaz*sin(phi);
dphi = omegax + dpsi *sin(theta);

dx=[dsX;dsY;dsZ;dpsi;dtheta;dphi;ddsX;ddsY;ddsZ;domegax;domegay;domegaz];
```

The function is called by this script `simulate_plane.m`:

```
%This script simulates a model plane connected to the ceiling by a linear
%spring, as treated in homework assignments of the course "Advanced
%Dynamics" at TUD.
%Author: H. Vallery, October 2014

%--------------------------------
%define constant parameters:
%--------------------------------
Ts=.01;%[s], sampling time
endtime=5;%[s] %end time of integration
par.Ts_anim=.01;%[s] pause between frames during animation
%(can be used for slow motion if larger than Ts)
```

```
%gravity:
par.g=9.81;%[m/s^2] acceleration of gravity (points in positive z direction)

%mass properties:
Ip=1*.1^2;%[kgm^2] moment of inertia of the plane about the z axis
par.Ixx=.5*Ip;
par.Iyy=.5*Ip;
par.Izz=Ip;

par.m=1;%[kg], mass of the plane

%geometry:
px=.1;%[m] local x coordinate of point P (spring attachment point) in B frame
pz=-.1;%[m] local z coordinate of P
par.rp_B=[px;0;pz];%coordinates of point P in B frame

par.l0=1;%[m], resting length of spring
par.k=20;%[N/m], stiffness of spring

%additional geometric data for aninmation of the plane:
par.length_plane=2;%[m], length of the plane's body
par.wingspan=4;%[m] wingspan
par.height_fin=.5;%[m] height of the fin
par.r_wingcenter=[.1;0;-.01];%[m] vector from CoM to wing center in body frame

%-----------------------------
%set initial conditions:
%-----------------------------

%Cartesian positions of the plane's center of mass:
sX=-px;
sY=.2;%.2;
sZ=par.l0+par.m*par.g/par.k;
%corresponding velocities:
dsX=2;
dsY=0;
dsZ=0;
%Euler angles (type I) to describe the orientation of the plane with
%respect to the inertial N frame (XYZ):
psi=0*pi/3;%rotation about Z axis
theta=0*pi/20;%rotation about intermediate Y' axis
phi=0*pi/10;%rotation about new x axis
%angular velocities (expressed in the body-fixed B frame (xyz) of the plane):
omegax=0;
omegay=0;
omegaz=pi/4/10;

%-----------------------------
%integrate:
%-----------------------------
%vector of initial conditions:
x0=[sX,sY,sZ,psi,theta,phi,dsX,dsY,dsZ,omegax,omegay,omegaz];
options = odeset('AbsTol',1e-10,'RelTol',1e-8);
%[t,y]=ode45(@plane_equationsofmotion,[0:Ts:endtime],x0,options,par);
%[t,y] = Integrate_Euler(@plane_equationsofmotion,[0,endtime],Ts, x0,par);
%[t,y] = Integrate_ModifiedEuler(@plane_equationsofmotion,[0,endtime],Ts, x0,par);
%[t,y] = Integrate_RungeKutta2(@plane_equationsofmotion,[0,endtime],Ts, x0,par);
[t,y] = Integrate_RungeKutta4(@plane_equationsofmotion,[0,endtime],Ts, x0,par);
```

```
%extract the generalized coordinates as time series:
qmatrix=y(:,1:6);

%-----------------------------------
%animate the result:
%-----------------------------------
planeanimation(qmatrix,par);
```

The animation function `planeanimation.p` is also provided (closed source).

4. (*50 points*) BONUS PROBLEM: THREEWHEELER

We consider a three-wheeled cart with massless wheels, as treated in the lecture of week 7 (Fig.3), which moves in the horizontal (XY) plane. The front wheel at point C is an actuated omni-wheel, while the unactuated rear wheels at the corners A and B roll in direction of the body-fixed x-axis, without slip in any of the horizontal directions. The cart's instantaneous configuration is described by a vector of generalized coordinates $\boldsymbol{q} = \begin{pmatrix} s_X & s_Y & \theta \end{pmatrix}^T$. The components s_X and s_Y describe the position of the cart's center of mass S, expressed in the space-fixed $\mathcal{N}$-frame, and θ is the heading angle. The omni-wheel is actuated in such a way that the ground exerts friction forces F_x and F_y on the the cart at point C. Assume the cart has a mass m, and it is a block of homogeneous material, with length l and width d. You may re-use and refer to the constraint equation as derived in the lecture.

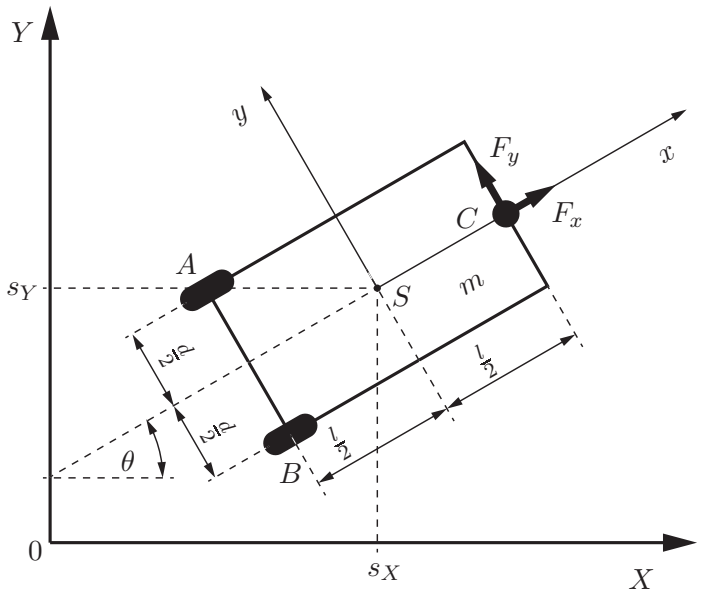

Figure 3: A three-wheeled cart moving in the horizontal plane.

The function `cart_equationsofmotion.m` that calculates state derivatives, the animation function `cartanimation.m`, and the script `simulate_cart.m` that calls the two functions are provided. The files currently generate movement of the cart for a constant force at the omni-wheel in negative y-direction, as used to generate one of the videos shown in the lecture. Within this exercise, you will modify the code to investigate a different movement.

a. (*5 pts*) Assume the cart now moves backwards in such a way that $s_X = -v_0 t$ and $\theta = \pi/8 \sin(2\pi f t)$, with the constant parameters $v_0 = 20\,\mathrm{m/s}$ and $f = 2\,\mathrm{Hz}$. The initial Y-position $s_Y(t_0)$ of the center of mass S at $t_0 = 0$ is 0. Find the initial velocity $\dot{s}_Y(t_0)$ of the cart at $t_0 = 0$ such that the constraint equation (as derived in the lecture) is fulfilled. You should do this by replacing the code in the definition of initial conditions in `simulate_cart.m`.

b. (*15 pts*) Write a new function `inversedynamics_cart.m` that is called by `simulate_cart.m` and use it to calculate the applied forces F_x and F_y in the body-fixed frame that are needed to generate the above movement, for $N = 500$ samples of time (with a sampling rate of $Ts = 0.01\,\mathrm{s}$). What are the final values for F_x and F_y (so at $t_e = (N-1) \cdot Ts$)?

Hint: Your function should have a header like this, and it should return scalar values for forces

when provided scalar values of kinematic data:

```
function [Fx,Fy]=inversedynamics_cart(theta,ddsX,ddsY,domega,par)
%This function calculates forces Fx and Fy as functions of given motion
%(not checking for violation of constraints!)
%par: parameter struct containing length and mass properties of the cart
```

Call it within a loop in `simulate_cart.m`.

c. (*15 pts*) Add the calculated forces as a matrix with N rows and 2 columns (containing the components F_x and F_y) as another parameter in the Matlab structure `par`.
Then, with the same initial conditions (also including the velocity component $\dot{s}_Y(t_0)$ you found in the first part of this problem), apply these forces again in the forward simulation (by reading out the matrix entries from `par` in the equations of motion), using the Modified Euler Method for the N samples of time. Finally, calculate the absolute error between the final center of mass X-component $\hat{s}_X(t_e)$ of the cart, as calculated in the forward simulation, and the true value $s_X(t_e) = -v_0 t_e$, with $t_e = (N-1) \cdot Ts$.

d. (*15 pts*) Calculate the constraint force acting on the rear axle of the cart for the N samples of your forward simulation. What is the maximum absolute value of this constraint force?
Assuming a constant vertical contact force at each contact point due to weight of the cart (so neglecting the dynamic shifting of weight between the three wheels), and assuming a friction coefficient of $\mu = 0.7$ and acceleration of gravity $g = 9.81\,\mathrm{m/s^2}$ pointing in negative Z direction, do you think the assumption of no slip at the rear wheels was valid?

Content of the file `simulate_cart.m`:

```
%This script simulates a three-wheeled cart,
%as treated in week 7 of the course "Advanced
%Dynamics" at TUD.
%Author: H. Vallery, October 2014


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%define constant parameters:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Ts=.04;%[s], sampling time
endtime=10;%[s] %end time of integration
par.Ts_anim=.04;%[s] pause between frames during animation
%(can be used for slow motion if larger than Ts)


%geometry:

par.length_cart=2;%[m], length of the cart
par.width_cart=1;%[m] width of the cart

%mass properties:
par.m=20;%[kg], mass of the cart
par.Is=par.m*1/12*(par.length_cart^2+par.width_cart^2);%[kgm^2] moment of inertia
%of the cart about the z axis

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%set initial conditions:
```

```
%--------------------------------

%Cartesian positions of the cart's center of mass:
sX=0;%[m]
sY=.2;%[m]
%corresponding velocities:
dsX=2;%[m/s]
dsY=2;%[m/s]

%orientation of the cart with
%respect to the inertial N frame (XYZ):
theta=0;%[rad], rotation about z axis
%angular velocity:
omega=2;%[rad/s]
%Remark: These initial conditions fulfill the given constraint

%--------------------------------
%integrate:
%--------------------------------
x0=[sX,sY,theta,dsX,dsY,omega];%vector of initial conditions:
options = odeset('AbsTol',1e-10,'RelTol',1e-8);
%[t,y]=ode45(@cart_equationsofmotion,[0:Ts:endtime],x0,options,par);
%[t,y] = Integrate_Euler(@cart_equationsofmotion,[0,endtime],Ts, x0,par);
[t,y] = Integrate_ModifiedEuler(@cart_equationsofmotion,[0,endtime],Ts, x0,par);
%[t,y] = Integrate_RungeKutta2(@cart_equationsofmotion,[0,endtime],Ts, x0,par);
%[t,y] = Integrate_RungeKutta4(@cart_equationsofmotion,[0,endtime],Ts, x0,par);


%extract the generalized coordinates as time series:
qmatrix=y(:,1:3);

%--------------------------------
%animate the result:
%--------------------------------

cartanimation(qmatrix,par);
```

Content of the file `cart_equationsofmotion.m`:

```
function dx=cart_equationsofmotion(t,x,par)
%function dx=cart_equationsofmotion(t,x,par)
%This function contains the 2D equations of motion of a three-wheeled cart
%driving around on the ground, as used in homework
%assignments of the course "Advanced Dynamics" at TUD.
%
%Inputs:
%t (time)
%state vector x: Cartesian x/y positions (in m) of cart center of mass,
% heading angle theta (in rad), & corresponding velocities (in m/s, rad/s).
%par: parameter struct containing length and mass properties of the cart
%
%Output: dx: derivatives of the states
%
%Author: H. Vallery, October 2014

%--------------------------------
%extract parameters:
%--------------------------------
```

```
%mass parameters of the cart:
m=par.m;% mass of cart
Is=par.Is;% moment of inertia about center of mass

%geometry of the cart:
l=par.length_cart;%[m], length of the cart
%d=par.width_cart;%[m] width of the cart, not needed


%--------------------------------
%read current states:
%--------------------------------

sX=x(1);
sY=x(2);
theta=x(3);

dsX=x(4);
dsY=x(5);
omega=x(6);

%--------------------------------
%Lagrange:
%--------------------------------

Fx=-50;%[N], forward force
Fy=0;%[N], steering force

%for steering movie:
%Fx=0;%forward force
%Fy=-50;%steering force

%for backward movie:
%Fx=-50;%forward force
%Fy=0;%steering force


A=[m 0 -2*Is/l*sin(theta);
    0 m 2*Is/l*cos(theta);
    sin(theta) -cos(theta) l/2];

b=[ Fx*cos(theta) - 2*Fy*sin(theta);
    Fx*sin(theta)+2*Fy*cos(theta);
    -dsX*omega*cos(theta)-dsY*omega*sin(theta)];

ddq=A\b;
ddsX=ddq(1);
ddsY=ddq(2);
domega=ddq(3);


%--------------------------------
%calculate state derivatives:
%--------------------------------

dx=[dsX;dsY;omega;ddsX;ddsY;domega];
```

Content of the file `cartanimation.m`:

```
function []=cartanimation(qmatrix,par);
%function []=cartanimation(qmatrix,par);
%
%This function animates a cart given generalized coordinates (GCs) as
%functions of time, as specified in the qmatrix.
%These generalized coordinates of the cart are:
%2 for translation: sX, sY (coordinates of the center of mass)
%1 for rotation: theta
%The three columns of the qmatrix must contain time series of these GCs.
%parameters contain geometric descriptions of the cart.
%
%You can test this function with some fake data:
% Tstest=.01;
% t=(0:Tstest:20)';
% fakedata=[3*cos(2*pi*1*t),3*sin(2*pi*1*t),2*pi*1*t+pi/2];
% cartanimation(fakedata,par)
%
%Author: H. Vallery, October 2014

Ts_anim=par.Ts_anim;%sampling time of animation (used to pause)
numsamp=size(qmatrix,1);%number of samples

%--------------------------------
%initialize the figure:
%--------------------------------

%initial values:
sX=qmatrix(1,1);
sY=qmatrix(1,2);
theta=qmatrix(1,3);

rs=[sX;sY];%position of point S (center of mass) in N coordinates


%geometric data for aninmation of the cart:
length_cart=par.length_cart;%[m], length of the cart
width_cart=par.width_cart;%[m] width of the cart

%points describing the cart's geometry in body coordinates (B frame):
cart_body_B=[-length_cart/2 -length_cart/2 length_cart/2 -length_cart/2;%x co-
%ordinates of body endpoints in body coordinates
    -width_cart/2 width_cart/2 0 -width_cart/2];%y coordinates
numpbody=size(cart_body_B,2);%number of points


%rotation matrix construction:
R_theta=[cos(theta) sin(theta);
    -sin(theta) cos(theta)];%rotation about z axis

%geometric data expressed in the N frame:
cart_body_N= R_theta'*cart_body_B+repmat(rs,1,numpbody);
track_N= cart_body_N(:,1:2);

%visualize:
figure(88)
clf
handle_cart_body=plot(cart_body_N(1,:),cart_body_N(2,:),'b','linewidth',4);
hold all
```

```
handle_cart_wheels=plot(cart_body_N(1,:),cart_body_N(2,:),'r*','linewidth',4);

plot(0,0,'bx')


xlabel('x')
ylabel('y')



axis equal
maxdim=5;
xlim([-maxdim,maxdim])
ylim([-maxdim,maxdim])

%--------------------------------
%animate by iterating through all following samples:
%--------------------------------

for indexsamples=1:numsamp
    sX=qmatrix(indexsamples,1);
    sY=qmatrix(indexsamples,2);
    theta=qmatrix(indexsamples,3);

    rs=[sX;sY];%position of point S in N coordinates

    R_theta=[cos(theta) sin(theta);
        -sin(theta) cos(theta)];%rotation about z axis


    %geometric data expressed in the N frame:
    cart_body_N= R_theta'*cart_body_B+repmat(rs,1,numpbody);
    track_N= cart_body_N(:,1:2);


    %update the data in the plot (faster than new plot each time):
    set(handle_cart_body,'XData',cart_body_N(1,:),'YData',cart_body_N(2,:));
    set(handle_cart_wheels,'XData',cart_body_N(1,:),'YData',cart_body_N(2,:));
    plot(track_N(1,:),track_N(2,:),'k.');

    pause(Ts_anim)%generate soft realtime

end
```