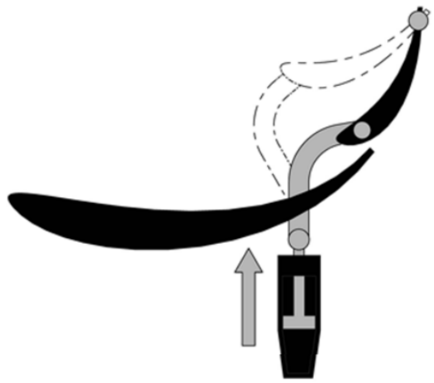
MSC Course Modelling and Simulation of Mechatronics Systems

# DRS project: kinematics push up mechanism

Team 13, 2022



## Initial setup

```
> restart: with(LinearAlgebra): with(MBSymba_r6): with(plots): with
  (Optimization):
```

## Utility functions

```
> getCoM:=proc(matrix_point)
      local i,n,xSum,ySum:
      xSum:=0: ySum:=0:
      n:=ColumnDimension(matrix_point):
      for i from 1 to n do
          xSum:=xSum+matrix_point[1,i]:
          ySum:=ySum+matrix_point[2,i]:
      end do:
      xSum/n,ySum/n:
  end proc:
```

## Data and shapes

### *Shapes*

```
> SMS:=[400,250]: # small plot size
> main_wing_matrix_point := <
      <0.29000000,     0.02281140,      0.,       1.>|
      <0.27550000,     0.01988240,      0.,       1.>|
      <0.26100000,     0.01696500,      0.,       1.>|
      <0.23200000,     0.01209300,      0.,       1.>|
      <0.20300000,     0.00815480,      0.,       1.>|
      <0.17400000,     0.00474730,      0.,       1.>|
      <0.15950000,     0.00329150,      0.,       1.>|
      <0.14500000,     0.00208510,      0.,       1.>|
      <0.13050000,     0.00099470,      0.,       1.>|
      <0.11600000,     0.00038860,      0.,       1.>|
      <0.10150000,     0.00010440,      0.,       1.>|
      <0.08700000,     0.,              0.,       1.>|
      <0.07250000,     0.00011310,      0.,       1.>|
```

```
      <0.05800000,    0.00083810,    0.,    1.>|
      <0.04350000,    0.00237220,    0.,    1.>|
      <0.02900000,    0.00499670,    0.,    1.>|
      <0.02175000,    0.00695710,    0.,    1.>|
      <0.01450000,    0.00953520,    0.,    1.>|
      <0.00725000,    0.01301230,    0.,    1.>|
      <0.00435000,    0.01483350,    0.,    1.>|
      <0.00290000,    0.01602540,    0.,    1.>|
      <0.,            0.01987950,    0.,    1.>|
      <0.00290000,    0.02240540,    0.,    1.>|
      <0.00435000,    0.02277080,    0.,    1.>|
      <0.00725000,    0.02311300,    0.,    1.>|
      <0.01450000,    0.02355670,    0.,    1.>|
      <0.02175000,    0.02356540,    0.,    1.>|
      <0.02900000,    0.02341170,    0.,    1.>|
      <0.04350000,    0.02271280,    0.,    1.>|
      <0.05800000,    0.02190950,    0.,    1.>|
      <0.07250000,    0.02121060,    0.,    1.>|
      <0.08700000,    0.02052620,    0.,    1.>|
      <0.10150000,    0.01976640,    0.,    1.>|
      <0.11600000,    0.01887610,    0.,    1.>|
      <0.13050000,    0.01820910,    0.,    1.>|
      <0.14500000,    0.01770450,    0.,    1.>|
      <0.15950000,    0.01737100,    0.,    1.>|
      <0.17400000,    0.01739130,    0.,    1.>|
      <0.20300000,    0.01795100,    0.,    1.>|
      <0.23200000,    0.01923860,    0.,    1.>|
      <0.26100000,    0.02108300,    0.,    1.>|
      <0.27550000,    0.02219950,    0.,    1.>|
      <0.29000000,    0.02356540,    0.,    1.>
   >:
> flap_wing_matrix_point := <
      <0.12001260,    0.00018864,    0.,    1.>|
      <0.11954484,    0.00030264,    0.,    1.>|
      <0.11814840,    0.00063936,    0.,    1.>|
      <0.11584368,    0.00118320,    0.,    1.>|
      <0.11266452,    0.00190956,    0.,    1.>|
      <0.10865784,    0.00278700,    0.,    1.>|
      <0.10388352,    0.00377940,    0.,    1.>|
      <0.09841368,    0.00484752,    0.,    1.>|
      <0.09233172,    0.00595140,    0.,    1.>|
      <0.08573112,    0.00705012,    0.,    1.>|
      <0.07871448,    0.00810336,    0.,    1.>|
      <0.07139184,    0.00907056,    0.,    1.>|
      <0.06387876,    0.00991224,    0.,    1.>|
      <0.05629524,    0.01058988,    0.,    1.>|
      <0.04876320,    0.01106808,    0.,    1.>|
      <0.04133724,    0.01129080,    0.,    1.>|
      <0.03419940,    0.01120224,    0.,    1.>|
      <0.02748000,    0.01079928,    0.,    1.>|
      <0.02129460,    0.01009392,    0.,    1.>|
      <0.01574868,    0.00911316,    0.,    1.>|
      <0.01093464,    0.00789648,    0.,    1.>|
      <0.00693060,    0.00649284,    0.,    1.>|
      <0.00379812,    0.00495492,    0.,    1.>|
      <0.00158256,    0.00333348,    0.,    1.>|
```

```
        <0.00031224,     0.00167172,      0.,       1.>|
        <0.,             0.,              0.,       1.>|
        <0.00063396,    -0.00157764,      0.,       1.>|
        <0.00218748,    -0.00296388,      0.,       1.>|
        <0.00462864,    -0.00414924,      0.,       1.>|
        <0.00791256,    -0.00512328,      0.,       1.>|
        <0.01198332,    -0.00587832,      0.,       1.>|
        <0.01677516,    -0.00641172,      0.,       1.>|
        <0.02221452,    -0.00672900,      0.,       1.>|
        <0.02822076,    -0.00684516,      0.,       1.>|
        <0.03470700,    -0.00678456,      0.,       1.>|
        <0.04158072,    -0.00657996,      0.,       1.>|
        <0.04875108,    -0.00626868,      0.,       1.>|
        <0.05616996,    -0.00585264,      0.,       1.>|
        <0.06365604,    -0.00534240,      0.,       1.>|
        <0.07109388,    -0.00477084,      0.,       1.>|
        <0.07836756,    -0.00416700,      0.,       1.>|
        <0.08536236,    -0.00355548,      0.,       1.>|
        <0.09196752,    -0.00295620,      0.,       1.>
    >:
> box_points_old := [[-0.020, 0.055], [0, 0.065], [0.100, 0.065],
  [0.100, 0.065], [0.080,0.065], [0.080, 0], [0.050, 0], [0.050,
  0.035], [0, 0.035], [-0.020, 0.045]]:
> pylon_points := [[0.260,0],[0.260,0.050],[0.280,0.050],[0.280,0]]
  :
```

***Data***

```
> pre_fixed_data := [
        # FIA regulation
        HEIGHT       = 0.220000,
        WIDTH        = 0.350000,
        min_dist     = 0.010000,
        max_dist     = 0.050000,

        # fixed points
        xA           = 0.280000,
        yA           = 0.000000,
        xD           = 0.335000,
        yD           = 0.154000,
        xR           = 0.280000,
        yR           = 0.022300,

        # fixed lengths
        L1           = 0.016000,
        L2           = 0.050000,
        L3           = 0.100000,
        L__wing      = 0.120000,
        W__wing      = 1010.000,
        d__wing      = 0.0060,         # main wing offset
        d__tip       = 0.0100,         # allowed by the FIA regulation

        # fixed angles
        gamma        = 5*Pi/180,       # main wing inclination

        # manouvre times
        T__opening   = 0.010,
```

```
      T__still    = 0.0300,
      T__closing  = 0.0100,

      # masses
      m__pist     = 0.0800,
      m__link     = 0.0375,         # L2*rho
      m__wing     = 2.0000,

      # physics constants
      rho__steel  = 0.75,           # linear density of a steel bar
  with radious 1cm
      g           = 9.81,

      # external forces (values from paper)
      F__drag_closed = 145.51319,
      F__drag_open   = 051.32626,
      F__down_closed = 819.11694,
      F__down_open   = 745.62411
  ]:
> data := pre_fixed_data union evalf(subs(pre_fixed_data,[
          Iz__pist    = 0,
          Iz__link    = m__link*(L2^2)/12,
          Iz__wing    = m__wing*(L__wing)/3
      ])):
```

# Kinematics

# Reference frames and points

**Ground Points**

```
> PA := make_POINT(ground,xA,yA,0):
> PD := make_POINT(ground,xD,yD,0):
```

**Reference frames**

```
> RF0  := translate(xA,yA,0):
> RF0a := RF0.translate(0,s(t)+L1,0):
> RF1  := RF0.translate(0,s(t)+L1,0).rotate('Z',psi1(t)):
> RF2  := RF1.translate(0,L2,0):
> RF3  := translate(xD,yD,0).rotate('Z', psi3(t)):
> RF_flap_wing := RF3.translate(-L__wing+d__tip,0,0):
> RF_main_wing := translate(d__wing, 0, 0).rotate('Z',gamma):
```

**Support points**

```
> PP := make_POINT(RF0,0,s(t),0):
> PB := origin(RF1):
> PC := origin(RF2):
> PC_3 := make_POINT(RF3,-L3,0,0):
```

The following points are used to calculate the distance
- PT: point on the tip of the flap wing
- PR: point of the main wing used as reference for the open/close distance

```
> PT := origin(RF_flap_wing):
> PR := make_POINT(RF_main_wing,xR,yR,0):
```

**Wings points (w.r.t. their reference frame)**

```
> flap_wing_points := [seq(convert((RF_flap_wing.
  flap_wing_matrix_point)[1..2,i],list),i=1..ColumnDimension
  (flap_wing_matrix_point))]:
> main_wing_points := [seq(convert((RF_main_wing.
  main_wing_matrix_point)[1..2,i],list),i=1..ColumnDimension
  (main_wing_matrix_point))]:
```

**CoM**

Body 1

```
> G1 := make_POINT(RF0a,0,-L1*4/5,0):
```

Body 2

```
> G2 := make_POINT(RF1,0,L2/2,0):
```

Body 3

```
> xG3,yG3 := evalf(getCoM(flap_wing_matrix_point)):
  G3 := make_POINT(RF_flap_wing,xG3,yG3,0):
```

## Constraints

```
> vCC := join_points(PC, PC_3):
  Phi := [comp_X(vCC,ground), comp_Y(vCC,ground)]: <%>;
```

$$\begin{bmatrix} \sin(\psi 1(t))\,L2 - \cos(\psi 3(t))\,L3 - xA + xD \\ -\cos(\psi 1(t))\,L2 - \sin(\psi 3(t))\,L3 - L1 - s(t) - yA + yD \end{bmatrix} \tag{2.2.1}$$

## Position analysis

### *Direct kinematic*

```
> qI := [s(t)]:
  qD := [psi1(t),psi3(t)]:
  qvars := qI union qD;
```

$$qvars := [s(t), \psi 1(t), \psi 3(t)] \tag{2.3.1.1}$$

```
> kin_sols := solve(Phi,qD,explicit=true):
  nops(kin_sols);
```

$$2 \tag{2.3.1.2}$$

```
> num_kin_sols := subs(data,kin_sols):
> "solution 1" = evalf(subs(s(t)=0,num_kin_sols[1]));
  "solution 2" = evalf(subs(s(t)=0,num_kin_sols[2]));
```

$$\text{"solution 1"} = [\psi 1(t) = -0.1824644031, \psi 3(t) = 1.093620968]$$
$$\text{"solution 2"} = [\psi 1(t) = -0.5760483505, \psi 3(t) = 1.289458931] \tag{2.3.1.3}$$

Our case is modeled by the third one

```
> kin_sol := kin_sols[1]:
```

### *Jacobian matrices*

With s(t) as independent variable

```
> JPhiD:=jacobianF(Phi,qD):
  JPhiI:=jacobianF(Phi,qI):
```

### *Singular configurations*

```
> SCs:=evalf(solve(subs(data,Phi union [Determinant(JPhiD)=0]),
  qvars,explicit=true)): <%>;
```

(2.3.3.1)

$$[[[s(t)=0.1380000000-0.02291287847\,I,\ \psi 1(t)=1.570796327+0.4435682544\,I,\ \psi 3(t)=0.4435682544\,I]],$$
$$[[s(t)=0.1380000000+0.02291287847\,I,\ \psi 1(t)=1.570796327-0.4435682544\,I,\ \psi 3(t)=-0.4435682544\,I]],$$
$$[[s(t)=0.2775528574,\ \psi 1(t)=-2.766169046,\ \psi 3(t)=-1.195372719]],$$
$$[[s(t)=-0.001552857370,\ \psi 1(t)=-0.3754236080,\ \psi 3(t)=1.195372719]]] \quad \textbf{(2.3.3.1)}$$

### *Inverse Kinematics*

Elongation "s(t)" of the piston to produce the opened and closed DRS configurations

```
> s_limit:=rhs(SCs[3][1])-0.001;
```

$$s\_limit := 0.2765528574 \quad \textbf{(2.3.4.1)}$$

```
> notime := map(x->x=op(0,x),qvars);
```

$$notime := [s(t)=s,\ \psi 1(t)=\psi 1,\ \psi 3(t)=\psi 3] \quad \textbf{(2.3.4.2)}$$

**Initial point** (10 mm distance from fixed wing)

```
> s_min := rhs(NLPSolve(subs(kin_sol,notime,data,comp_Y(PT,ground)-
  comp_Y(PR,ground)-min_dist)^2,s=0..s_limit)[2][1]);
```

$$s\_min := 0.000198939129915359 \quad \textbf{(2.3.4.3)}$$

**Final point** (50 mm distance from fixed wing)

```
> s_max := rhs(NLPSolve(subs(kin_sol,notime,data,comp_Y(PT,ground)-
  comp_Y(PR,ground)-max_dist)^2,s=0..s_limit)[2][1]);
```

$$s\_max := 0.0460748980054946 \quad \textbf{(2.3.4.4)}$$

Tests to check distances

```
> "actual minimum distance" = evalf(subs(kin_sol,s(t)=s_min,data,
  comp_Y(PT,ground)-comp_Y(PR,ground))),    # should be 10 mm
  "actual maximum distance" = evalf(subs(kin_sol,s(t)=s_max,data,
  comp_Y(PT,ground)-comp_Y(PR,ground)));    # should be 50 mm
```

"actual minimum distance" = 0.01000144013, "actual maximum distance" = 0.05000098592 **(2.3.4.5)**

```
> s_range := s_min..s_max;
```

$$s\_range := 0.000198939129915359\,..0.0460748980054946 \quad \textbf{(2.3.4.6)}$$

```
> s_stroke := s_max-s_min;
```

$$s\_stroke := 0.0458759588755792 \quad \textbf{(2.3.4.7)}$$

### *Working space*

```
> point_B := subs(kin_sol,data,s(t)=s,[comp_X(PB,ground),comp_Y(PB,
  ground)]):
  space_B := [seq(point_B,s=s_range,0.001)]:
> point_C := subs(kin_sol,data,s(t)=s,[comp_X(PC,ground),comp_Y(PC,
  ground)]):
  space_C := [seq(point_C,s=s_range,0.001)]:
```

### *Wing angle range (psi3(t))*

```
> psi3_closed := evalf(subs(kin_sol,data,s(t)=s_min,psi3(t)))
  :"absolute wing open angle"=%,"deg"=%*180/Pi;
  psi3_open := evalf(subs(kin_sol,data,s(t)=s_max,psi3(t)))
  :"absolute wing closed angle"=%,"deg"=%*180/Pi;
```

"absolute wing open angle" = 1.087075800, "deg" = 62.28485533

$$\text{"absolute wing closed angle"} = 0.5487704485, \text{"deg"} = 31.44223061 \quad \mathbf{(2.3.6.1)}$$

***Direct kinematic with independent psi3(t)***

```
> psi3_kin_sols := solve(subs(data,Phi),[op(convert(qvars,set)
  minus {psi3(t)})],explicit=true):
  nops(psi3_kin_sols);
```

$$1 \quad \mathbf{(2.3.7.1)}$$

```
> psi3_kin_sol := psi3_kin_sols[1];
```

$$psi3\_kin\_sol := \left[ \psi 1(t) = \arcsin(2.\cos(\psi 3(t)) - 1.100000000), s(t) = -0.005000000000 \sqrt{-21. - 400.\cos(\psi 3(t))^2 + 440.\cos(\psi 3(t))} - 0.1000000000 \sin(\psi 3(t)) + 0.1380000000 \right] \quad \mathbf{(2.3.7.2)}$$

Check the solution

```
> "closed configuration"=evalf(subs(psi3(t)=psi3_closed,data,
  psi3_kin_sol));
  "open configuration"=evalf(subs(psi3(t)=psi3_open,data,
  psi3_kin_sol));
```

$$\text{"closed configuration"} = [\psi 1(t) = -0.1706753036, s(t) = 0.0001994128]$$

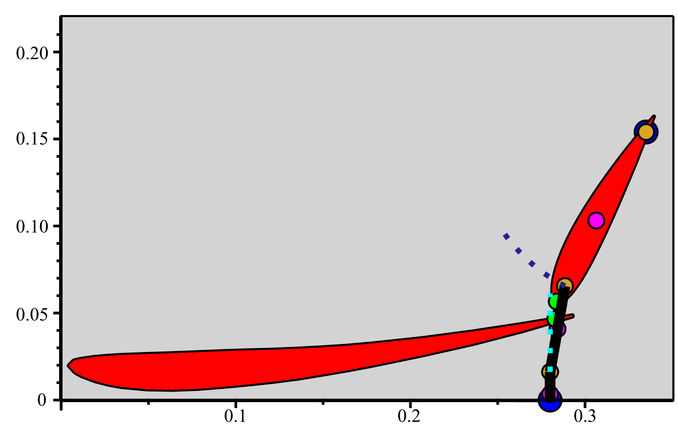
$$\text{"open configuration"} = [\psi 1(t) = 0.6514412053, s(t) = 0.04607560011] \quad \mathbf{(2.3.7.3)}$$

***Drawing...***

```
> draw_mech := proc(sol,data,dof)
      global PA,PB,PC,PD,G1,G2,G3:
      local pA,pB,pC,pD,pP,pT,pR,g1,g2,g3,r:
      r:=0.004:

      pA:=[comp_X(PA,ground),comp_Y(PA,ground)]:
      pB:=[comp_X(PB,ground),comp_Y(PB,ground)]:
      pC:=[comp_X(PC,ground),comp_Y(PC,ground)]:
      pD:=[comp_X(PD,ground),comp_Y(PD,ground)]:
      g1:=[comp_X(G1,ground),comp_Y(G1,ground)]:
      g2:=[comp_X(G2,ground),comp_Y(G2,ground)]:
      g3:=[comp_X(G3,ground),comp_Y(G3,ground)]:
      pT:=[comp_X(PT,ground),comp_Y(PT,ground)]:
      pR:=[comp_X(PR,ground),comp_Y(PR,ground)]:

      display([
              plottools:-line(subs(sol,data,dof,pA),subs(sol,data,
  dof,pB),color=black,thickness=4),
              plottools:-line(subs(sol,data,dof,pB),subs(sol,data,
  dof,pC),color=black,thickness=4),

              plottools:-disk(subs(sol,data,dof,pB),r,color=
  "Goldenrod"),
              plottools:-disk(subs(sol,data,dof,pC),r,color=
  "Goldenrod"),
              plottools:-disk(subs(sol,data,dof,pD),r,color=
  "Goldenrod"),

              plottools:-disk(subs(sol,data,dof,pT),r,color=green),
              plottools:-disk(subs(sol,data,dof,pR),r,color=green),
```

```
                plottools:-disk(subs(sol,data,dof,g1),r,color=
magenta),
                plottools:-disk(subs(sol,data,dof,g2),r,color=
magenta),
                plottools:-disk(subs(sol,data,dof,g3),r,color=
magenta),

                plottools:-polygon(subs(sol,data,dof,
flap_wing_points),color=red),
                plottools:-polygon(subs(sol,data,dof,
main_wing_points),color=red),

                plottools:-disk(subs(sol,data,dof,pA),r*1.5,color=
blue),
                plottools:-disk(subs(sol,data,dof,pD),r*1.5,color=
blue),

                plottools:-rectangle([0, 0],[0.350, 0.220], color =
"LightGrey"),
                plottools:-polygon(pylon_points, color = red),
                plottools:-rectangle([0, 0],subs(data,[WIDTH, HEIGHT]
),color="LightGrey"),

                plottools:-curve(space_B,color=cyan,linestyle=dot,
thickness=2),
                plottools:-curve(space_C,color=navy,linestyle=dot,
thickness=2)
          ],
          scaling=constrained
     ):
end proc:
> animate(draw_mech,[kin_sol,data,s(t)=S],S=s_range);
```

$S = 0.00019894$




## Velocity analysis

### *Velocity ratio*

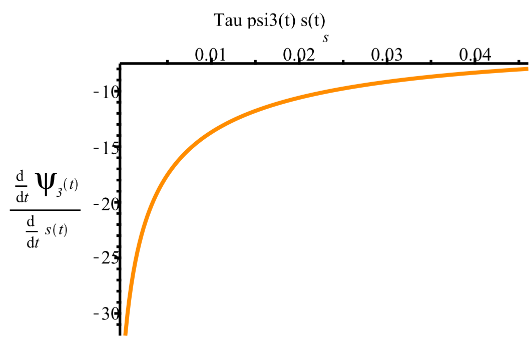
```
> tau := combine(simplify(-MatrixInverse(JPhiD).JPhiI): <%>;
  "dependent variables"=qD;
```

$$\left[\begin{array}{c} \frac{\sin(\psi 3(t))}{L2\cos(-\psi 3(t)+\psi 1(t))} \\ -\frac{\cos(\psi 1(t))}{L3\cos(-\psi 3(t)+\psi 1(t))} \end{array}\right]$$

$$\text{"dependent variables"} = [\psi 1(t), \psi 3(t)] \qquad (2.4.1.1)$$

Ratio between the opening velocity of the wing and the velocity of s(t).

```
> plot(
      subs(kin_sol,data,s(t)=S,tau[2]), S=s_range,
      title="Tau psi3(t) s(t)", labels=[s,typeset(diff(psi__3(t),t)
  /diff(s(t),t))],
      color="DarkOrange",
      size=SMS
  );
```




### *Dependent variables velocities*

```
> vel_kin_sol:=op(solve(diff(Phi,t),diff(qD,t))):
```

### *Velocity of points*

Velocity of the point where  drag force atcs

```
> vel_PP  := collect(subs(vel_kin_sol,velocity(PP)),{diff}):
```

Velocity of the centre of mass of the different bodies

```
> vel_P1  := collect(subs(vel_kin_sol,velocity(G1)),{diff}):
> vel_P2  := collect(subs(vel_kin_sol,velocity(G2)),{diff}):
> vel_P3  := collect(subs(vel_kin_sol,velocity(G3)),{diff}):
```

### *Angular velocities of points*

Angular velocity of the reference frames of the different bodies

```
> avel_P1  := collect(subs(vel_kin_sol,angular_velocity(RF1)),{diff}
  ):
> avel_P2  := collect(subs(vel_kin_sol,angular_velocity(RF2)),{diff}
  ):
> avel_P3  := collect(subs(vel_kin_sol,angular_velocity(RF3)),{diff}
  ):
```

## Acceleration analysis

### *Dependent variables accelerations*

```
> acc_kin_sol:=op(solve(diff(Phi,t,t),diff(qD,t,t))):
```

***Acceleration of points***

Acceleration of the centre of mass of the different bodies

```
> acc_P1 := collect(subs(acc_kin_sol,vel_kin_sol,acceleration(G1)),
  {diff}):
> acc_P2 := collect(subs(acc_kin_sol,vel_kin_sol,acceleration(G2)),
  {diff}):
> acc_P3 := collect(subs(acc_kin_sol,vel_kin_sol,acceleration(G3)),
  {diff}):
```

***Angular acceleration of points***

Acceleration of the centre of mass of the different bodies

```
> aacc_P1 := collect(subs(acc_kin_sol,vel_kin_sol,
  angular_acceleration(RF1)),{diff}):
> aacc_P2 := collect(subs(acc_kin_sol,vel_kin_sol,
  angular_acceleration(RF2)),{diff}):
> aacc_P3 := collect(subs(acc_kin_sol,vel_kin_sol,
  angular_acceleration(RF3)),{diff}):
```

## Opening profile

```
> T__tot := subs(data,T__opening+T__still+T__closing);
```

$$T_{tot} := 0.0500 \tag{2.6.1}$$

```
> base_profile := a0+a1*t+a2*t^2+a3*t^3+a4*t^4+a5*t^5;
```

$$base\_profile := t^5\, a5 + t^4\, a4 + t^3\, a3 + t^2\, a2 + t\, a1 + a0 \tag{2.6.2}$$

**Opening part**

To find the constants, we can plug in what we know about the profile (s_min, s_max etc).

```
> opening_known_conditions:=[
      # position
      subs(t=0,data,base_profile=s_min),
      subs(t=T__opening,data,base_profile=s_max),
      # velocity
      subs(t=0,data,diff(base_profile,t)=0),
      subs(t=T__opening,data,diff(base_profile,t)=0),
      # acceleration
      subs(t=0,data,diff(base_profile,t,t)=0),
      subs(t=T__opening,data,diff(base_profile,t,t)=0)
  ]:
> opening_coefficients:=op(solve(opening_known_conditions,[seq
  (a||i,i=0..5)]));
```

$$opening\_coefficients := \left[a0 = 0.0001989391299,\ a1 = 0.,\ a2 = 0.,\ a3 = 458759.5888,\ a4 = -6.881393831 \times 10^{7},\ a5 = 2.752557533 \times 10^{9}\right] \tag{2.6.3}$$

```
> opening_profile:=subs(opening_coefficients,base_profile);
```

$$opening\_profile := 2.752557533 \times 10^{9}\, t^5 - 6.881393831 \times 10^{7}\, t^4 + 458759.5888\, t^3 + 0.0001989391299 \tag{2.6.4}$$

**Still part**

```
> still_profile:=s_max;
```

$$still\_profile := 0.0460748980054946 \tag{2.6.5}$$

**Closing part**

```
> closing_known_condition_equations:=subs(data,[
      # position
      subs(t=T__opening+T__still,data,base_profile=s_max),
      subs(t=T__tot,data,base_profile=s_min),
      # velocity
      subs(t=T__opening+T__still,data,diff(base_profile,t)=0),
      subs(t=T__tot,data,diff(base_profile,t)=0),
      # acceleration
      subs(t=T__opening+T__still,data,diff(base_profile,t,t)=0),
      subs(t=T__tot,data,diff(base_profile,t,t)=0)
  ]);
```

$$closing\_known\_condition\_equations := [1.024000000 \times 10^{-7}\, a5 + 2.560000000 \times 10^{-6}\, a4 + 0.000064000000\, a3 + 0.00160000\, a2 + 0.0400\, a1 + a0 = 0.0460748980054946,\ a0 + 0.0500\, a1 + 0.00250000\, a2 + 0.000125000000\, a3 + 6.250000000 \times 10^{-6}\, a4 + 3.125000000 \times 10^{-7}\, a5 = 0.000198939129915359,\ 0.00001280000000\, a5 + 0.000256000000\, a4 + 0.00480000\, a3 + 0.0800\, a2 + a1 = 0,\ a1 + 0.1000\, a2 + 0.00750000\, a3 + 0.000500000000\, a4 + 0.00003125000000\, a5 = 0,\ 0.001280000000\, a5 + 0.01920000\, a4 + 0.2400\, a3 + 2\, a2 = 0,\ 2\, a2 + 0.3000\, a3 + 0.03000000\, a4 + 0.002500000000\, a5 = 0] \tag{2.6.6}$$

```
> closing_coefficients:=op(solve(closing_known_condition_equations,
  [seq(a||i,i=0..5)]));
```

$$closing\_coefficients := [a0 = 487.4322620,\ a1 = -55051.15065,\ a2 = 2.477301779 \times 10^{6},\ a3 = -5.550991024 \times 10^{7},\ a4 = 6.193254448 \times 10^{8},\ a5 = -2.752557533 \times 10^{9}] \tag{2.6.7}$$

```
> closing_profile:=subs(closing_coefficients,base_profile);
```

$$closing\_profile := -2.752557533 \times 10^{9}\, t^{5} + 6.193254448 \times 10^{8}\, t^{4} - 5.550991024 \times 10^{7}\, t^{3} + 2.477301779 \times 10^{6}\, t^{2} - 55051.15065\, t + 487.4322620 \tag{2.6.8}$$

**Complete profile**

```
> s_profile:=piecewise(
      t>=0                  and t<=T__opening,
  opening_profile,
      t>T__opening          and t<=T__opening+T__still,
  still_profile,
      t>T__opening+T__still and t<=T__tot,
  closing_profile
  );
```

$$s\_profile := \begin{cases} 2.752557533 \times 10^{9}\, t^{5} - 6.881393831 \times 10^{7}\, t^{4} + 458759.5888\, t^{3} + 0.000198 \\ 0.0460748980054946 \\ -2.752557533 \times 10^{9}\, t^{5} + 6.193254448 \times 10^{8}\, t^{4} - 5.550991024 \times 10^{7}\, t^{3} + 2.477301779 \times 10^{6}\, t^{2} - \end{cases}$$

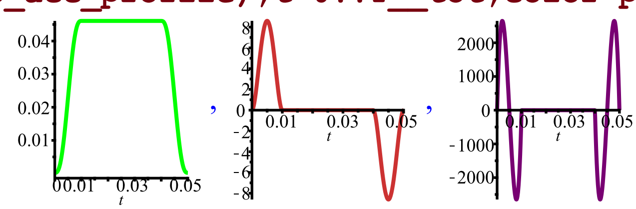
```
> s_vel_profile:=subs(diff(s_profile,t));
```

$$s\_vel\_profile := \begin{cases} 1.376278766 \times 10^{10}\, t^4 - 2.752557532 \times 10^{8}\, t^3 + 1.376278766 \times 10^{6}\, t^2 \\ 0 \\ -1.376278766 \times 10^{10}\, t^4 + 2.477301779 \times 10^{9}\, t^3 - 1.665297307 \times 10^{8}\, t^2 + 4.954603558 \times 10 \end{cases}$$

```
> s_acc_profile:=subs(diff(s_vel_profile,t));
```

$$s\_acc\_profile := \begin{cases} 5.505115064 \times 10^{10}\, t^3 - 8.257672596 \times 10^{8}\, t^2 + 2.752557532 \times 10^{6}\, t \\ 0 \\ -5.505115064 \times 10^{10}\, t^3 + 7.431905337 \times 10^{9}\, t^2 - 3.330594614 \times 10^{8}\, t + 4.954603558 \times 10 \end{cases}$$

```
> profiles:=subs(data,[
      diff(s(t),t,t)=s_acc_profile,
      diff(s(t),t)=s_vel_profile,
      s(t)=s_profile
  ]):
> plot(subs(data,s_profile),t=0..T__tot,color=green),
  plot(subs(data,s_vel_profile),t=0..T__tot,color=orange),
  plot(subs(data,s_acc_profile),t=0..T__tot,color=purple)
```




## Known conditions

Initial conditions

### *Position*

```
> ics_qI := [s(t)=eval(subs(t=0,data,s_profile))]; # it corresponds
  to s_min
```

$$ics\_qI := [s(t) = 0.0001989391299] \quad \textbf{(2.7.1.1)}$$

```
> ics_qD := evalf(subs(ics_qI,data,kin_sol));
```

$$ics\_qD := [\psi 1(t) = -0.1706738276, \psi 3(t) = 1.087076363] \quad \textbf{(2.7.1.2)}$$

```
> ics_pos := ics_qI union ics_qD;
```

**(2.7.1.3)**

$$ics\_pos := [s(t) = 0.0001989391299, \psi 1(t) = -0.1706738276, \psi 3(t) = 1.087076363] \quad \textbf{(2.7.1.3)}$$

***Velocity***

```
> ics_qI_vel := [diff(s(t),t)=eval(subs(t=0,data,s_vel_profile))]:
> ics_qD_vel := evalf(subs(ics_qI_vel,data,vel_kin_sol)):
> ics_vel := ics_qI_vel union ics_qD_vel;
```

$$ics\_vel := \left[\frac{d}{dt} s(t) = 0., \frac{d}{dt} \psi 1(t) = 0., \frac{d}{dt} \psi 3(t) = -0.\right] \quad \textbf{(2.7.2.1)}$$

***Acceleration***

```
> ics_qI_acc := [diff(s(t),t,t)=eval(subs(t=0,data,s_acc_profile))]
  :
> ics_qD_acc := [seq(diff(qD[i],t,t)=evalf(rhs(subs(data,
  ics_qI_acc,ics_vel,ics_pos,acc_kin_sol[i]))),i=1..nops(qD))]:
> ics_acc := ics_qI_acc union ics_qD_acc;
```

$$ics\_acc := \left[\frac{d^2}{dt^2} s(t) = 0., \frac{d^2}{dt^2} \psi 1(t) = 0., \frac{d^2}{dt^2} \psi 3(t) = 0.\right] \quad \textbf{(2.7.3.1)}$$

# Dynamics

```
> _gravity:=make_VECTOR(ground,0,-g,0):
```

**Bodies**

```
> PIST:=make_BODY(G1,m__pist,0,0,Iz__pist):
> LINK:=make_BODY(G2,m__link,0,0,Iz__link):
> FLAP_WING:=make_BODY(G3,m__wing,0,0,Iz__wing):
```

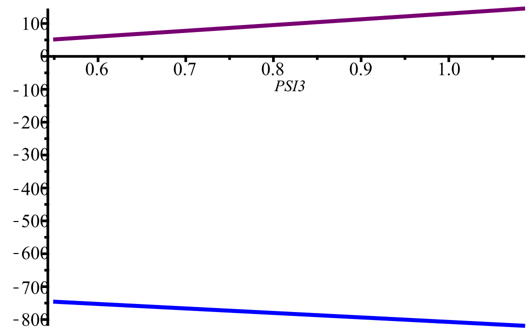
**Acting forces**

Piston force

```
> piston_force:=make_VECTOR(RF0,0,F__piston(t),0):          #-
> FP:=make_FORCE(piston_force,PP,PIST):
```

Air contact forces (dragforce plus downforce)

```
> air_forces:=make_VECTOR(ground,F__drag(t),-F__down(t),0):
> FA:=make_FORCE(air_forces,G3,FLAP_WING):
> air_forces_law:=[
      F__drag(t)=F__drag_open+(F__drag_closed-F__drag_open)*(psi3
  (t)-psi3_open)/(psi3_closed-psi3_open),
      F__down(t)=F__down_open+(F__down_closed-F__down_open)*(psi3
  (t)-psi3_open)/(psi3_closed-psi3_open)
  ]:
> display([
          plot(subs(air_forces_law,psi3(t)=PSI3,data,F__drag(t)),
  PSI3=psi3_closed..psi3_open),
          plot(subs(air_forces_law,psi3(t)=PSI3,data,-F__down(t)),
  PSI3=psi3_closed..psi3_open)
      ],
      color=[purple,blue],
      size=SMS
  );
```




## Newton Euler

### *Equation of motion*

**Internal forces**

```
> PJ_force := make_FORCE(make_VECTOR(ground,Np(t),0,0),PP,PIST):
> PJ_torque := make_TORQUE(make_VECTOR(ground,0,0,Tp(t)),PIST):
> RJ1_force := make_FORCE(make_VECTOR(ground,rj1x(t),rj1y(t),0),PB,
  PIST,LINK):
> RJ2_force := make_FORCE(make_VECTOR(ground,rj2x(t),rj2y(t),0),PC,
  LINK,FLAP_WING):
> RJ3_force := make_FORCE(make_VECTOR(ground,rj3x(t),rj3y(t),0),PD,
  FLAP_WING):
> rvars := [Np(t),Tp(t),rj1x(t),rj1y(t),rj2x(t),rj2y(t),rj3x(t),
  rj3y(t)]:
```

**Set of forces**

```
> forces := {PJ_force,PJ_torque,RJ1_force,RJ2_force,RJ3_force,FP,
  FA}:
```

**Newton-Euler Equations**

```
> newton_equations({PIST} union forces):
  euler_equations({PIST} union forces,G1):
  NE_eqns1 := [comp_X(%%,ground), comp_Y(%%,ground), comp_Z(%,
  ground)]:
<FA> FORCE is not valid: it must be applied to a BODY
<RJ3_force> FORCE is not valid: it must be applied to a BODY

> newton_equations({LINK} union forces):
  euler_equations({LINK} union forces,G2):
  NE_eqns2 := [comp_X(%%,ground), comp_Y(%%,ground), comp_Z(%,
  ground)]:
<FA> FORCE is not valid: it must be applied to a BODY
<FP> FORCE is not valid: it must be applied to a BODY
<PJ_force> FORCE is not valid: it must be applied to a BODY
<RJ3_force> FORCE is not valid: it must be applied to a BODY

> newton_equations({FLAP_WING} union forces):
  euler_equations({FLAP_WING} union forces,G3):
  NE_eqns3 := [comp_X(%%,ground), comp_Y(%%,ground), comp_Z(%,
  ground)]:
<FP> FORCE is not valid: it must be applied to a BODY
<PJ_force> FORCE is not valid: it must be applied to a BODY

> eqns_NE := NE_eqns1 union NE_eqns2 union NE_eqns3:
```

```
nops(eqns_NE) + nops(Phi) = nops(rvars) + nops(qvars);
```

$$11 = 11 \tag{3.1.1.1}$$

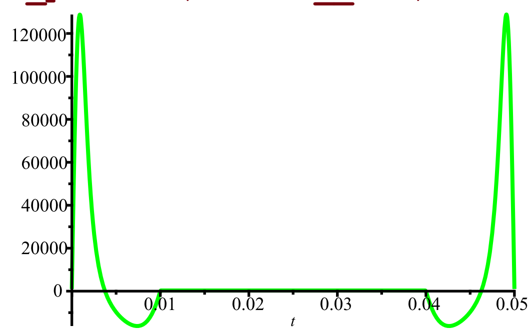
### *Inverse dynamic*

Piston force profile given trajectory

```
> sol_rvars := op(solve(eqns_NE,rvars union [F__piston(t)])):
  nops(sol_rvars);
```

$$9 \tag{3.1.2.1}$$

```
> piston_force_NE := simplify(combine(rhs(sol_rvars[9]))):
> piston_force_NE_profile := subs(air_forces_law,acc_kin_sol,
  vel_kin_sol,kin_sol,profiles,data,piston_force_NE):
> ## plot ##
  plot(piston_force_NE_profile,t=0..T__tot,color=green,size=SMS);
```




### *Direct dynamic*

Variable profile given the force.

In practice this just a check of the inverse dynamic.

**Set up the system**

```
> eqns_NE_static := evalf(subs(air_forces_law,ics_acc,ics_vel,
  ics_pos,data,eqns_NE));
  ics_rvars := op(solve(eqns_NE_static[1..-2],rvars)): <%>;
```

$$eqns\_NE\_static := \left[ -1.\, Np(t) - 1.\, rj1x(t),\, 0.784800 - 1.\, F_{piston}(t) - 1.\, rj1y(t), \right.$$
$$-0.003200000000\, Np(t) + 0.01280000000\, rj1x(t) - 1.\, Tp(t),\, rj1x(t) - 1.\, rj2x(t),$$
$$0.367875 + rj1y(t) - 1.\, rj2y(t),\, 0.02463676358\, rj1x(t) + 0.02463676358\, rj2x(t)$$
$$- 0.004246160620\, rj1y(t) - 0.004246160620\, rj2y(t),\, -145.5132885 + rj2x(t)$$
$$- 1.\, rj3x(t),\, 838.7370169 + rj2y(t) - 1.\, rj3y(t),\, 0.03767539849\, rj2x(t)$$
$$\left. - 0.01799901303\, rj2y(t) + 0.05085213524\, rj3x(t) - 0.02850866573\, rj3y(t) \right]$$

$$\tag{3.1.3.1}$$

$$\left[\begin{array}{c} Np(t) = -0.1669624750 + 0.1723505852\, F_{piston}(t) \\ Tp(t) = 0.002671399600 - 0.002757609363\, F_{piston}(t) \\ rj1x(t) = 0.1669624750 - 0.1723505852\, F_{piston}(t) \\ rj1y(t) = 0.7848000000 - 1.\, F_{piston}(t) \\ rj2x(t) = 0.1669624750 - 0.1723505852\, F_{piston}(t) \\ rj2y(t) = 1.152675000 - 1.\, F_{piston}(t) \\ rj3x(t) = -145.3463260 - 0.1723505852\, F_{piston}(t) \\ rj3y(t) = 839.8896919 - 1.\, F_{piston}(t) \end{array}\right] \quad (3.1.3.1)$$

```
> ics_piston_NE := [F__piston(t)=evalf(subs(t=0,
  piston_force_NE_profile))]:
> ics_dae_NE := subs(ics_piston_NE,t=0,convert(ics_vel union
  ics_pos union ics_rvars,D));
```

$$ics\_dae\_NE := [\mathrm{D}(s)(0) = 0., \mathrm{D}(\psi 1)(0) = 0., \mathrm{D}(\psi 3)(0) = -0., s(0) = 0.0001989391299, \psi 1(0) = -0.1706738276, \psi 3(0) = 1.087076363, Np(0) = 172.7335602, Tp(0) = -2.763736962, rj1x(0) = -172.7335602, rj1y(0) = -1002.406007, rj2x(0) = -172.7335602, rj2y(0) = -1002.038132, rj3x(0) = -318.2468487, rj3y(0) = -163.3011151] \quad (3.1.3.2)$$

```
> eqns_NE union Phi union ics_dae_NE:
> sys_NE := subs(
      air_forces_law,
      F__piston(t)=piston_force_NE_profile,
      data,
      eqns_NE union Phi union ics_dae_NE
  ):
```

**System solution**

Sometimes the dsolve face a singularity and throws an exception; the following approach avoids interruptions.

```
> st := time():
  solved:=false:
  do
      try
          dsol_NE := dsolve(sys_NE,numeric,implicit=true,stiff=
  true,output=listprocedure,relerr=1e-5):
          solved:=true:
      catch:
          print("Exception generated, automatic retrial"):
      end try;
  until solved:
  "computation time"=time() - st;
```

$$\text{"computation time"} = 2.094 \quad (3.1.3.3)$$

```
> dsol_NE;
```

$$[t = \mathbf{proc}(t)\ \ldots\ \mathbf{end\ proc}, Np(t) = \mathbf{proc}(t)\ \ldots\ \mathbf{end\ proc}, Tp(t) = \mathbf{proc}(t)\ \ldots\ \mathbf{end\ proc}, \psi 1(t) \quad (3.1.3.4)$$

$= \mathbf{proc}(t)$ ... **end proc**, $\frac{d}{dt}\ \psi 1(t) = \mathbf{proc}(t)$ ... **end proc**, $\psi 3(t) = \mathbf{proc}(t)$

...

**end proc**, $\frac{d}{dt}\ \psi 3(t) = \mathbf{proc}(t)$ ... **end proc**, $rj1x(t) = \mathbf{proc}(t)$ ... **end proc**, $rj1y(t) =$

$\mathbf{proc}(t)$

...

**end proc**, $rj2x(t) = \mathbf{proc}(t)$ ... **end proc**, $rj2y(t) = \mathbf{proc}(t)$ ... **end proc**, $rj3x(t) = \mathbf{proc}(t)$

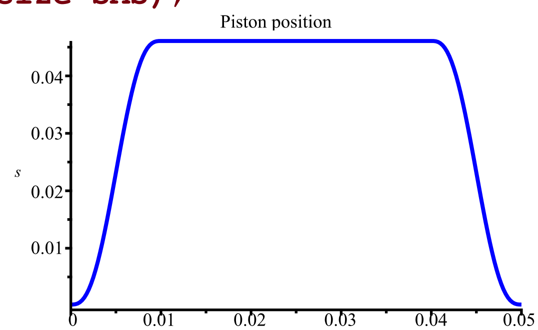
...

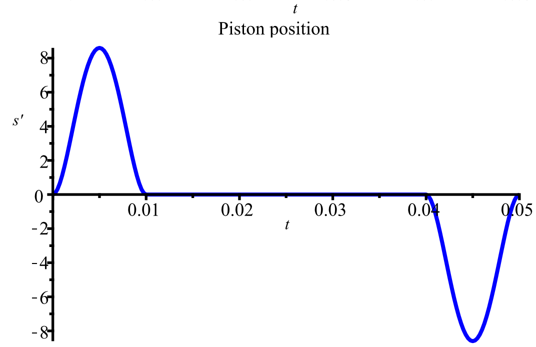
**end proc**, $rj3y(t) = \mathbf{proc}(t)$ ... **end proc**, $s(t) = \mathbf{proc}(t)$ ... **end proc**, $\frac{d}{dt}\ s(t) = \mathbf{proc}(t)$

...

**end proc**]

```
> ## plot ##
  odeplot(dsol_NE,[t,s(t)],t=0..T__tot,color=blue,title="Piston
  position",size=SMS);
  ## plot ##
  odeplot(dsol_NE,[t,diff(s(t),t)],t=0..T__tot,color=blue,title=
  "Piston position",size=SMS);
```




```
> ## plot ## odeplot(dsol_NE,[t,psi3(t)],t=0..T__tot,color=purple,
  title="Flap wing angle",size=SMS,numpoints=100);
  ## plot ## odeplot(dsol_NE,[t,diff(psi3(t),t)],t=0..T__tot,color=
  pink, title="Flap wing velocity",size=SMS,numpoints=100);
```

```
> ## plot ## odeplot(dsol_NE,[t,Np(t)],t=0..T__tot,color=brown,
  title="Reaction force on the piston",size=SMS,numpoints=100);
> ## plot ## odeplot(dsol_NE,[t,rj3x(t)],t=0..T__tot,color=khaki,
  title="Reaction force along x of the revolute joint in point C",
  size=SMS,numpoints=100);
  ## plot ## odeplot(dsol_NE,[t,rj3y(t)],t=0..T__tot,color=gold,
  title="Reaction force along y of the revolute joint in point C",
  size=SMS,numpoints=100);
```

## Lagrange

### *Equation of motion*

**Constraints defintion**

```
> lvars := [seq(lambda||i(t),i=1..nops(Phi))]:
> constraints := make_CONSTRAINT(Phi,lvars);
```

$$constraints := \mathrm{table}\big(\big[expr = \big[\sin(\psi 1(t))\, L2 - \cos(\psi 3(t))\, L3 - xA + xD, -\cos(\psi 1(t))\, L2 - \sin(\psi 3(t))\, L3 - L1 - s(t) - yA + yD\big], obj = CONSTRAINT, vars = \big[\lambda 1(t), \lambda 2(t)\big]\big]\big) \quad (3.2.1.1)$$

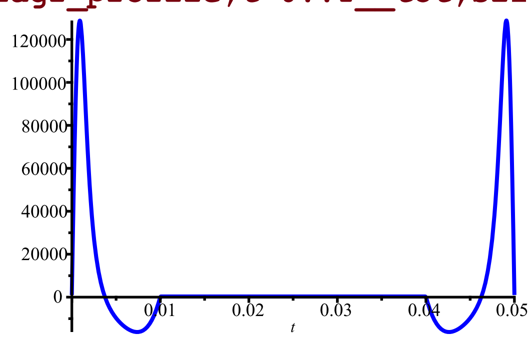
**Lagrange equations**

```
> eqns_lagr := lagrange_equations({PIST,LINK,FLAP_WING,FP,FA,
  constraints},qvars union lvars,t):
```

### *Inverse dynamic*

As before, piston force profile given trajectory

```
> sol_vars_lagr:=op(solve(eqns_lagr[1..3],lvars union [F__piston(t)
  ])):
> piston_force_lagr:=simplify(combine(rhs(sol_vars_lagr[3]))):
> piston_force_lagr_profile:=subs(air_forces_law,acc_kin_sol,
  vel_kin_sol,kin_sol,profiles,data,piston_force_lagr):
> plot(piston_force_lagr_profile,t=0..T__tot,size=SMS,color=blue);
```




### *Direct dynamic*

As before, check the result of the inverse dynamic

**Set up the system**

```
> lvars := [seq(lambda||i(t),i=1..nops(Phi))]:
> constraints := make_CONSTRAINT(Phi,lvars):
> AA,BB:=GenerateMatrix(eqns_lagr[1..nops(qvars)] union diff(Phi,t,
  t),diff(qvars,t,t) union lvars):
> AAN:=evalf(simplify(subs(data,ics_pos,AA))):
> BBN:=evalf(subs(air_forces_law,ics_vel,ics_pos,data,BB)):
> lin_sol:=LinearSolve(AAN,BBN):
```

```
> ics_piston_lagr:=[F__piston(t)=solve(lin_sol[1],F__piston(t))];
  # the following ways give the same results
      solve(lin_sol[2],F__piston(t)),
      solve(lin_sol[3],F__piston(t)):
```

$$ics\_piston\_lagr := \left[F_{piston}(t) = 1003.196915\right] \quad (3.2.3.1)$$

```
> ics_lvars:=[seq(subs(ics_piston_lagr,lvars[i]=convert(lin_sol,
  list)[i]),i=-2..-1)]:
> ics_dae_lagr:=subs(ics_piston_lagr,t=0,convert(ics_vel union
  ics_pos union ics_lvars,D));
```

$$ics\_dae\_lagr := \left[D(s)(0) = 0., D(\psi 1)(0) = 0., D(\psi 3)(0) = -0., s(0) = 0.0001989391299, \psi 1(0) = -0.1706738276, \psi 3(0) = 1.087076363, \lambda 1(0) = -172.7346128, \lambda 2(0) = -1002.044240\right] \quad (3.2.3.2)$$

```
> sys_lagr:=subs(
      air_forces_law,
      F__piston(t)=piston_force_lagr_profile,
      data,
      eqns_lagr union Phi union ics_dae_lagr
  ):
```

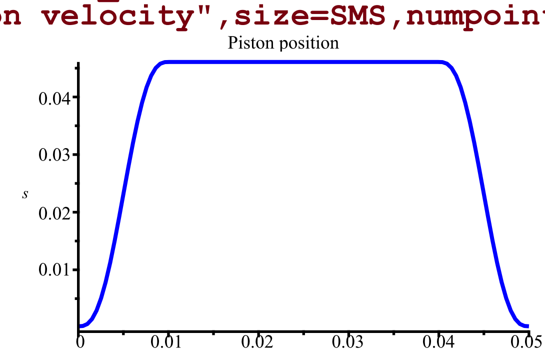
**System solution**

Sometimes the dsolve face a singularity and throws an exception; the following approach avoids interruptions.

```
> st := time():
  solved:=false:
  do
      try
          dsol_lagr:=dsolve(sys_lagr,numeric,implicit=true,stiff=
  true,relerr=1e-5):
          solved:=true:
      catch:
          print("Exception generated, automatic retrial"):
      end try;
  until solved:
  "computation time"=time() - st;
```

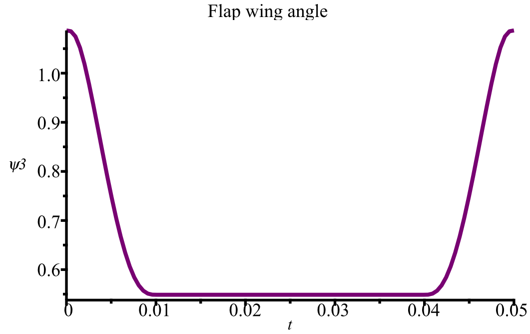
$$\text{"computation time"} = 1.531 \quad (3.2.3.3)$$

```
> ## plot ##
  odeplot(dsol_lagr,[t,s(t)],t=0..T__tot,color=blue,title="Piston
  position",size=SMS,numpoints=100);
  ## plot ## odeplot(dsol_lagr,[t,diff(s(t),t)],t=0..T__tot,color=
  cyan, title="Piston velocity",size=SMS,numpoints=100);
```




```
> ## plot ##
  odeplot(dsol_lagr,[t,psi3(t)],t=0..T__tot,color=purple, title=
```

```
"Flap wing angle",size=SMS,numpoints=100);
## plot ## odeplot(dsol_lagr,[t,diff(psi3(t),t)],t=0..T__tot,
color=pink, title="Flap wing velcoity",size=SMS,numpoints=100);
```




# Virtual work

Principle of Virtual Work: VW = 0. Helpful to find input/output relationship between forces.

## *Static case*

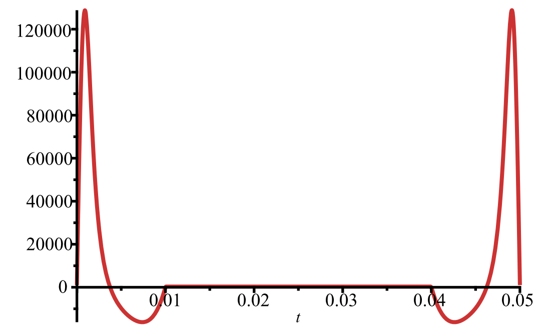
```
> PVW_static:=
      dot_prod(piston_force,vel_PP)
      +dot_prod(air_forces,vel_P3)
      +dot_prod(make_VECTOR(ground,0,-m__pist*g,0),vel_P1)
      +dot_prod(make_VECTOR(ground,0,-m__link*g,0),vel_P2)
      +dot_prod(make_VECTOR(ground,0,-m__wing*g,0),vel_P3):
> piston_force_pvw_static := rhs(op(solve(PVW_static,{F__piston(t)}
  ))):
> piston_force_pvw_static_profile := subs(air_forces_law,kin_sol,s
  (t)=s_profile,data,piston_force_pvw_static):
> ## plot ## plot(piston_force_pvw_static_profile,t=0..T__tot,size=
  SMS,view=[0..T__tot,0..800]);
```

## *Dynamic case*

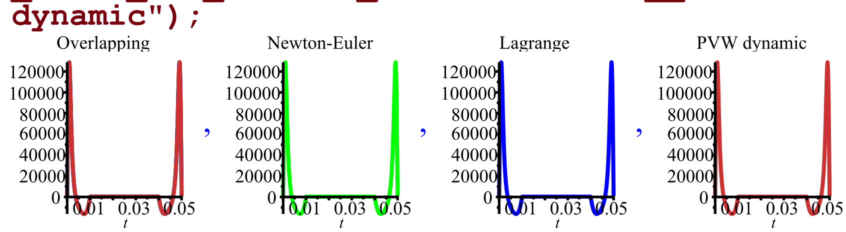
Principle of d'Alembert

```
> PVW_dynamic:=
      # linear forces
      dot_prod(piston_force,vel_PP)
      +dot_prod(air_forces,vel_P3)
      +dot_prod(make_VECTOR(ground,0,-m__pist*g,0),vel_P1)
      +dot_prod(make_VECTOR(ground,0,-m__link*g,0),vel_P2)
      +dot_prod(make_VECTOR(ground,0,-m__wing*g,0),vel_P3)

      # angluar momentum
      # there are not external moments to consider

      # linear inertia forces
      -dot_prod(m__pist*acc_P1,vel_P1)
      -dot_prod(m__link*acc_P2,vel_P2)
      -dot_prod(m__wing*acc_P3,vel_P3)

      # angular inertia forces
      -dot_prod(Iz__pist*aacc_P1,avel_P1)
      -dot_prod(Iz__link*aacc_P2,avel_P2)
      -dot_prod(Iz__wing*aacc_P3,avel_P3)
```

```
:
> piston_force_pvw_dynamic := rhs(op(solve(PVW_dynamic,{F__piston
  (t)}))):
> piston_force_pvw_dynamic_profile:=subs(air_forces_law,
  acc_kin_sol,vel_kin_sol,kin_sol,profiles,data,
  piston_force_pvw_dynamic):
> ## plot ##
  plot(piston_force_pvw_dynamic_profile,t=0..T__tot,size=SMS,color=
  orange);
```




```
> display([
          plot(piston_force_NE_profile,t=0..T__tot),
          plot(piston_force_lagr_profile,t=0..T__tot),
          plot(piston_force_pvw_dynamic_profile,t=0..T__tot)
      ],
      color=[green,blue,orange],
      title="Overlapping"
  ),
  plot(piston_force_NE_profile,t=0..T__tot,color=green,title=
  "Newton-Euler"),
  plot(piston_force_lagr_profile,t=0..T__tot,color=blue,title=
  "Lagrange"),
  plot(piston_force_pvw_dynamic_profile,t=0..T__tot,color=orange,
  title="PVW dynamic");
```




**Conclusion**: the piston force is the same for all the three methods adopted.