MSC Course Modelling and Simulation of Mechatronics Systems

DRS project: pod-pull 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:
> getWingHeight:=proc(l,matrix_point)
      local i,n,h:
      n:=ColumnDimension(matrix_point):
      for i from 1 to n do
          if matrix_point[1,i] < l then
              h:=matrix_point[2,i]:
              break:
          end if:
      end do:
      (h+matrix_point[2,i+1])/2:
  end proc:
```

Drawing procedures

```
> draw_mech := proc(sol,data,dof,range,pylon)
      global PA,PB,PC,PD,PP,PT,PR,G1,G2,G3:
      local pA,pB,pC,pD,pP,pT,pR,g1,g2,g3,space_B,space_C,r,i:
      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)]:
    pP:=[comp_X(PP,ground),comp_Y(PP,ground)];
    pT:=[comp_X(PT,ground),comp_Y(PT,ground)]:
    pR:=[comp_X(PR,ground),comp_Y(PR,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)];

    space_B := evalf([seq(subs(sol,data,s(t)=i,pB),i=range,0.001)
]):
    space_C := evalf([seq(subs(sol,data,s(t)=i,pC),i=range,0.001)
]):

    display([
            plottools:-line(subs(sol,data,dof,pP),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:-curve(space_B,color=cyan,linestyle=dot,
thickness=2),
            plottools:-curve(space_C,color=navy,linestyle=dot,
thickness=2),

            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:-rectangle(subs(sol,data,dof,[pP[1]-r,pP[2]
-r]),subs(sol,data,dof,[pP[1],pP[2]+r]),color="Goldenrod"),

            plottools:-rectangle(subs(sol,data,dof,[pP[1]-0.012,
pP[2]-0.010]),subs(sol,data,dof,[pP[1],pP[2]+0.010]),color=
"DarkGrey"),

            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:-polygon(pylon,color=red),
              plottools:-rectangle([0, 0],subs(data,[WIDTH, HEIGHT]
  ),color="LightGrey")
          ],
          scaling=constrained
      ):

  end proc:
> draw_mech_dyna := proc(sol,data,dof,range)
      local pA,pB,pC,pD,pP,pT,pR,g1,g2,g3,r,i,scale,borderBool,
  space_B,space_C,s:
      r:=0.004:
      scale:=9000:
      borderBool := true:

      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)]:
      pP:=[comp_X(PP,ground),comp_Y(PP,ground)];
      pT:=[comp_X(PT,ground),comp_Y(PT,ground)]:
      pR:=[comp_X(PR,ground),comp_Y(PR,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)];

      space_B := evalf([seq(subs(sol,data,s(t)=i,pB),i=range,0.001)
  ]):
      space_C := evalf([seq(subs(sol,data,s(t)=i,pC),i=range,0.001)
  ]):

      display([

              plots:-arrow(subs(sol,data,[[[pP[1]-0.050,pP[2]
  +0.007],[-F__piston(t)/(1*scale),0]]]),color="Yellow",border=
  borderBool),
              plots:-arrow(subs(diff(s(t),t)=vel,sol,data,[[[pP[1]
  -0.020,pP[2]-0.005],[subs(sol,-vel)/(0.001*scale),0]]]),color=
  "Blue",border= borderBool),
              plots:-arrow(subs(sol,data,[[[g3[1]+0,g3[2]+0],[subs
  (air_forces_law,sol,data,F__drag(t))/scale,0]]]),color="Yellow",
  border= borderBool),
              plots:-arrow(subs(sol,data,[[[g3[1]+0,g3[2]+0],[0,-
  subs(air_forces_law,sol,data,F__down(t))/(1*scale)]]]),color=
  "Yellow",border= borderBool),


              textplot([0.032,0.200,"Velocity"],'color'="Blue"),
              textplot([0.050,0.185,"External forces"],'color'=
  "Yellow"),

              plots:-arrow(subs(sol,data,[[[g1[1]+0,g1[2]+0],[0,-
  subs(data,m__pist*g)/scale]]]),color="Yellow",border=
  borderBool),
```

```
            plots:-arrow(subs(sol,data,[[[g2[1]+0,g2[2]+0],[0,-
subs(data,m__link*g)/scale]]]),color="Yellow",border=
borderBool),
            plots:-arrow(subs(sol,data,[[[g3[1]+0,g3[2]+0],[0,-
subs(data,m__wing*g)/scale]]]),color="Yellow",border=
borderBool),

            plottools:-line(subs(sol,data,dof,pP),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:-line(subs(sol,[pP[1]+0.020,pP[2]+0.040]),
subs(sol,[pP[1]-p3(t)/4,pP[2]+0.040]),color="Goldenrod",
thickness=50),

            plottools:-curve(space_B,color=cyan,linestyle=dot,
thickness=2),
            plottools:-curve(space_C,color=navy,linestyle=dot,
thickness=2),

            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:-rectangle(subs(sol,data,dof,[pP[1]-r,pP[2]
-r]),subs(sol,data,dof,[pP[1],pP[2]+r]),color="Goldenrod"),

            plottools:-rectangle(subs(sol,data,dof,[pP[1]-0.012,
pP[2]-0.010]),subs(sol,data,dof,[pP[1],pP[2]+0.010]),color=
"DarkGrey"),

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

```
        ],
        scaling=constrained
    )
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.>
  >:
> pylon_points := [[0.1275, 0.0197664], [0.1275, 0.115], [0.101,
  0.115], [0.086, 0.122], [0.086, 0.138], [0.101, 0.145], [0.173,
  0.145], [0.173, 0.115], [0.1565, 0.115], [0.1565, 0.070],
  [0.1565, 0.050], [0.1565, 0.0182091]]:
> opt_pylon_points := [[0.1275, 0.0197664], [0.1275, 0.110],
  [0.101, 0.110], [0.086, 0.117], [0.086, 0.133], [0.101, 0.140],
  [0.173, 0.140], [0.173, 0.110], [0.1565, 0.110], [0.1565, 0.065],
  [0.1565, 0.045], [0.1565, 0.0182091]]:
```

Data

```
> guess_data:=[
     yA           = 0.130000,
```

```
    L2           = 0.049000,
    L3           = 0.070000
    # psi0       = 0
]:
> pre_fixed_data := [
    # FIA regulation
    HEIGHT       = 0.220000,
    WIDTH        = 0.350000,
    min_dist     = 0.010000,
    max_dist     = 0.050000,

    # fixed points
    xA           = 0.172675,
    # yA         = 0.130000,       # optimized
    xD           = 0.335000,
    yD           = 0.154000,
    xR           = 0.280000,
    yR           = 0.022300,

    # fixed lengths
    L1           = 0.083446,
    # L2         = 0.049000,       # optimized
    # L3         = 0.070000,
    L__wing      = 0.120000,
    W__wing      = 1010.000,
    # H3         = <H3_value>,     # calculated below depending on
L3
    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   = 0.100,
    T__still     = 0.300,
    T__closing   = 0.100,

    # 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,

    # control
    kp           = 10000,            # position gain
```

```
      kpv          = 10000              # velocity gain
  ]:
> H3_eqn := H3 = getWingHeight(subs(guess_data,pre_fixed_data,
  L__wing-L3),flap_wing_matrix_point):
> fixed_data := pre_fixed_data union evalf(subs(guess_data,
  pre_fixed_data,[
          Iz__pist    = 0,
          Iz__link    = m__link*(L2^2)/12,
          Iz__wing    = m__wing*(L__wing^2)/3
      ])):
> data :=  [op(guess_data), H3_eqn, op(fixed_data)];
```

$$data := [yA = 0.130000, L2 = 0.049000, L3 = 0.070000, H3 = 0.01117944000, HEIGHT \quad \textbf{(1.2.2.1)}$$

$$= 0.220000, WIDTH = 0.350000, min\_dist = 0.010000, max\_dist = 0.050000, xA$$

$$= 0.172675, xD = 0.335000, yD = 0.154000, xR = 0.280000, yR = 0.022300, L1$$

$$= 0.083446, L_{wing} = 0.120000, W_{wing} = 1010.000, d_{wing} = 0.0060, d_{tip} = 0.0100, \gamma = \frac{\pi}{36},$$

$$T_{opening} = 0.100, T_{still} = 0.300, T_{closing} = 0.100, m_{pist} = 0.0800, m_{link} = 0.0375, m_{wing}$$

$$= 2.0000, \rho_{steel} = 0.75, g = 9.81, F_{drag\_closed} = 145.51319, F_{drag\_open} = 51.32626,$$

$$F_{down\_closed} = 819.11694, F_{down\_open} = 745.62411, kp = 10000, kpv = 10000, Iz_{pist} = 0., Iz_{link}$$

$$= 7.503124999 \times 10^{-6}, Iz_{wing} = 0.009600000000]$$

Kinematic

Recursive approach

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):
> RF1 := RF0.translate(-s(t),0,0): #rotate('Z',psi0).
> RF2 := RF1.translate(L1,0,0).rotate('Z', psi2(t)):
> 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 := origin(RF1):
> PB := origin(RF2):
  PC := make_POINT(RF2,L2,0,0):
> PC_3 := make_POINT(RF3,-L3,H3,0):
```

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(RF1,L1/5,0,0):
```

Body 2

```
> vCB := convert(project(join_points(PB,PC),RF2)[comps],list):
> G2 := make_POINT(RF2,vCB[1]/2,vCB[2]/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 := combine([comp_X(vCC,ground),comp_Y(vCC,ground)]): <%>;
```

$$\begin{bmatrix} -\sin(\psi 3(t))\,H3 - L2\cos(\psi 2(t)) - \cos(\psi 3(t))\,L3 - L1 + s(t) - xA + xD \\ \cos(\psi 3(t))\,H3 - L2\sin(\psi 2(t)) - \sin(\psi 3(t))\,L3 - yA + yD \end{bmatrix} \quad \textbf{(2.2.1)}$$

Position analysis

Direct kinematics

```
> qI := [s(t)]:
  qD := [psi2(t),psi3(t)]:
  qvars := qI union qD;
```

$$qvars := [s(t), \psi 2(t), \psi 3(t)] \quad \textbf{(2.3.1.1)}$$

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

$$2 \quad \textbf{(2.3.1.2)}$$

```
> num_kin_sols := subs(data, kin_sols):
```

Check to decide the solution

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

$$\text{"solution 1"} = [\psi 2(t) = 1.323317734,\ \psi 3(t) = -0.1796443249]$$

$$\text{"solution 2"} = [\psi 2(t) = -0.7325904295,\ \psi 3(t) = 1.087109357] \quad \textbf{(2.3.1.3)}$$

Our case is modeled by the second one

```
> kin_sol := kin_sols[2]:
```

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)): <%>;
  nops(SCs);
```

$$\begin{aligned}&[[[s(t) = -0.07887900000 + 0.009846582313\ \mathrm{I},\ \psi 2(t) = -1.570796327\\ &\quad - 0.4359409505\ \mathrm{I},\ \psi 3(t) = 1.729165193 - 0.4359409505\ \mathrm{I}]],\\ &\quad [[s(t) = -0.07887900000 - 0.009846582313\ \mathrm{I},\ \psi 2(t) = -1.570796327\\ &\quad + 0.4359409505\ \mathrm{I},\ \psi 3(t) = 1.729165193 + 0.4359409505\ \mathrm{I}]],\\ &\quad [[s(t) = -0.1963392696,\ \psi 2(t) = 2.940042489,\ \psi 3(t) = 3.098411355]],\\ &\quad [[s(t) = 0.03858126963,\ \psi 2(t) = 0.2015501651,\ \psi 3(t) = 0.3599190312]]]\end{aligned}$$

$$4 \qquad \textbf{(2.3.3.1)}$$

The last singular configuration is close to the working spacce of the mechanism, so we set a limit for the displacement of the piston in order to avoid it.

```
> s_limit := rhs(SCs[4][1])-0.001;
```

$$s\_limit := 0.03758126963 \qquad \textbf{(2.3.3.2)}$$

Inverse Kinematics

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

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

$$notime := [s(t) = s,\ \psi 2(t) = \psi 2,\ \psi 3(t) = \psi 3] \qquad \textbf{(2.3.4.1)}$$

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,assume=nonnegative)[2]
  [1]);
```

$$s\_min := 4.57178532896224 \times 10^{-7} \qquad \textbf{(2.3.4.2)}$$

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,assume=nonnegative)[2]
  [1]);
```

$$s\_max := 0.0355834351678901 \qquad \textbf{(2.3.4.3)}$$

Tests to check distances

```
> "actual minimum distance" = evalf(subs(kin_sol,s(t)=s_min,data,
  comp_Y(PT,ground)-comp_Y(PR,ground))),    # max 0.010m
  "actual maximum distance" = evalf(subs(kin_sol,s(t)=s_max,data,
  comp_Y(PT,ground)-comp_Y(PR,ground)));    # max 0.050m
```

"actual minimum distance" = 0.00999999438, "actual maximum distance" = 0.04999998206 **(2.3.4.4)**

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

$$s\_range := 4.57178532896224 \times 10^{-7}..0.0355834351678901 \qquad \textbf{(2.3.4.5)}$$

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

$$s\_stroke := 0.0355829779893572 \qquad \textbf{(2.3.4.6)}$$

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" = 1.087104543, "deg" = 62.28650217

"absolute wing closed angle" = 0.5487811208, "deg" = 31.44284209 **(2.3.5.1)**

Direct kinematic with psi3(t) as independent

Express the other variables depending on the wing angle psi3(t).

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

$$1 \qquad \textbf{(2.3.6.1)}$$

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

Check the solution:

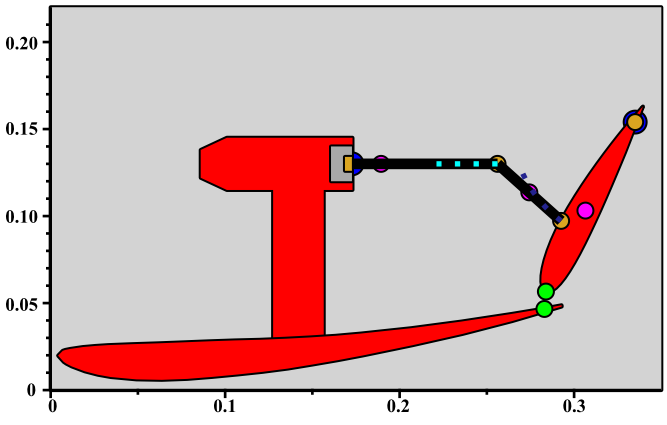
```
> "closed configuration"=evalf(subs(psi3(t)=psi3_closed,data,
  psi3_kin_sol));
  "open configuration"=evalf(subs(psi3(t)=psi3_open,data,
  psi3_kin_sol));
```

"closed configuration" = $[\psi 2(t) = -0.7325847458, s(t) = 4.8653 \times 10^{-7}]$

"open configuration" = $[\psi 2(t) = -0.06080231714, s(t) = 0.03558345130]$ **(2.3.6.2)**

Drawing...

```
> ## animate ##
  animate(draw_mech,[kin_sol,data,s(t)=S,s_range,pylon_points],S=
  s_range,frames=20, title="Kinematics of the mechanism", font=
  [TIMES,BOLD,12]);
```



Velocity analysis

Velocity ratio

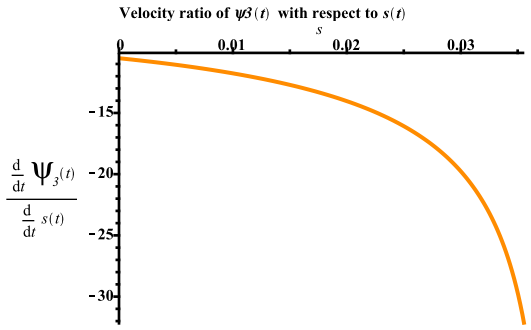
```
> tau := combine(simplify(-MatrixInverse(JPhiD)).JPhiI): <%>;
  "dependent variables"=qD;
```

$$\begin{bmatrix} -\dfrac{\sin(\psi 3(t))\,H3+\cos(\psi 3(t))\,L3}{L2\,(H3\cos(-\psi 3(t)+\psi 2(t))+L3\sin(-\psi 3(t)+\psi 2(t)))} \\ \dfrac{\cos(\psi 2(t))}{H3\cos(-\psi 3(t)+\psi 2(t))+L3\sin(-\psi 3(t)+\psi 2(t))} \end{bmatrix}$$

$$\text{"dependent variables"} = [\psi 2(t),\ \psi 3(t)] \qquad \textbf{(2.4.1.1)}$$

Ratio between the opening velocity of the wing angle psi3(t) and the velocity of the piston s(t)

```
> plot(
      subs(kin_sol,data,s(t)=S,tau[2][1]), S=s_range,
      title=typeset("Velocity ratio of ", psi3(t)," with respect to
  ", s(t)), labels=[s,typeset(diff(psi__3(t),t)/diff(s(t),t))],
      color="DarkOrange", font=[TIMES,BOLD,10],
      size=SMS
  );
```



Dependent variables velocities

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

$$\left[\left[\frac{d}{dt}\psi 2(t) = -\left((\sin(\psi 3(t))\,H3+\cos(\psi 3(t))\,L3)\left(\frac{d}{dt}s(t)\right)\right)\Big/\right.\right. \qquad \textbf{(2.4.2.1)}$$

$$(L2\,(H3\cos(\psi 2(t))\cos(\psi 3(t))+H3\sin(\psi 3(t))\sin(\psi 2(t))$$

$$-L3\cos(\psi 2(t))\sin(\psi 3(t))+L3\cos(\psi 3(t))\sin(\psi 2(t)))\Big)\Big],$$

$$\left[\frac{d}{dt}\psi 3(t) = \left(\cos(\psi 2(t))\left(\frac{d}{dt}s(t)\right)\right)\Big/(H3\cos(\psi 2(t))\cos(\psi 3(t))\right.$$

$$+H3\sin(\psi 3(t))\sin(\psi 2(t))-L3\cos(\psi 2(t))\sin(\psi 3(t))$$

$$\left.\left.+L3\cos(\psi 3(t))\sin(\psi 2(t))\right)\right]\Big]$$

Velocity of points

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

Duration of the complete repositioning profile of the piston.

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

$$T_{tot} := 0.500 \qquad \textbf{(2.6.1)}$$

Constant acceleration and deceleration

The following is the first version we used for the of the motion profile.
In this, when the mechanism get opened, the piston is accelerated with constant accleration for half of the period and then decelerated for the same amount of time.
We faced that with this profile the force was highly complex and impossible to manage (see later for the shape).
Even if the behaviour of this profile is far from what happens in reality, we decided to leave it (only as an example) as it represents the **fastest profile** possible given a certain acceleration.
In the next chapter there is the better approximated profile we used for the main studies.

Base profile given T

```
> raw_base_profile:=piecewise(
      t>=0 and t<=T__opening/2,                          s_min+2*
  (s_max-s_min)*(t/T__opening)^2,
```

```
    t>T__opening/2 and t<=T__opening,                    s_max-2*
(s_max-s_min)*(t-T__opening)^2/T__opening^2,
    t>T__opening and t<=T__opening+T__still,             s_max,
    t>T__opening+T__still and t<=T__tot-T__closing/2, s_max-2*
(s_max-s_min)*((t-(T__opening+T__still))/T__closing)^2,
    t>T__tot-T__closing/2 and t<=T__tot,                 s_min+2*
(s_max-s_min)*((t-T__tot)/T__closing)^2
):
```

```
> raw_s_profile:=subs(data,raw_base_profile);
```

$$raw\_s\_profile := \begin{cases} 4.57178532896224 \times 10^{-7} + 7.11659559787145\, t^2 & 0 \le t \le 0.0500000000 \\ 0.0355834351678901 - 7.11659559787145\, (-0.100 + t)^2 & 0.05000000000 < t \le 0. \\ 0.0355834351678901 & 0.100 < t \le 0.400 \\ 0.0355834351678901 - 7.11659559787145\, (-0.400 + t)^2 & 0.400 < t \le 0.4500000 \\ 4.57178532896224 \times 10^{-7} + 7.11659559787145\, (t - 0.500)^2 & 0.4500000000 < t \le 0. \end{cases}$$

```
> raw_s_vel_profile:=subs(data,diff(raw_base_profile,t));
```

$raw\_s\_vel\_profile :=$ **(2.6.1.2)**

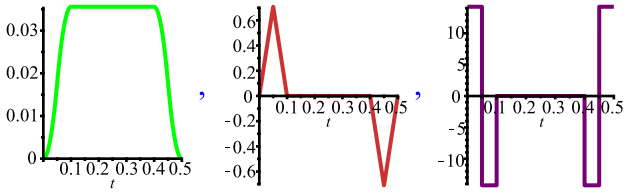
$$\begin{cases} 14.2331911957429\, t & 0 \le t \le 0.05000000000 \\ 1.42331911957429 - 14.2331911957429\, t & 0.05000000000 < t \le 0.100 \\ 0 & 0.100 < t \le 0.400 \\ 5.69327647829716 - 14.2331911957429\, t & 0.400 < t \le 0.4500000000 \\ 14.2331911957429\, t - 7.11659559787145 & 0.4500000000 < t \le 0.500 \end{cases}$$

```
> raw_s_acc_profile:=subs(data,diff(raw_base_profile,t,t));
```

$$raw\_s\_acc\_profile := \begin{cases} 14.2331911957429 & 0 \le t \le 0.05000000000 \\ -14.2331911957429 & 0.05000000000 < t \le 0.100 \\ 0 & 0.100 < t \le 0.400 \\ -14.2331911957429 & 0.400 < t \le 0.4500000000 \\ 14.2331911957429 & 0.4500000000 < t \le 0.500 \end{cases}$$ **(2.6.1.3)**

```
> raw_profiles:=[
    diff(s(t),t,t)=raw_s_acc_profile,
    diff(s(t),t)=raw_s_vel_profile,
    s(t)=raw_s_profile
]:
```

```
> plot(raw_s_profile,     subs(data,t=0..T__tot), color=green),
  plot(raw_s_vel_profile, subs(data,t=0..T__tot), color=orange),
  plot(raw_s_acc_profile, subs(data,t=0..T__tot), color=purple)
```



Minimum jerk trajectory

We know we want the piston to pass from the closed configuration to the open one.
We want it to follow the trajectory with the **smoothest path**.
We found the way to follow on paper reachable through the link:
http://courses.shadmehrlab.org/Shortcourse/minimumjerk.pdf

The profile that minimizes the jerk has to have its sixth derivative is equal to zero. The following is the generic profile with opening and closing phase.

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

$$base\_profile := t^5\, a5 + t^4\, a4 + t^3\, a3 + t^2\, a2 + t\, a1 + a0 \qquad \textbf{(2.6.2.1)}$$

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)]));
```

$$opening\_coefficients := [a0 = 4.571785329 \times 10^{-7},\ a1 = 0.,\ a2 = 0.,\ a3 = 355.8297799,\ a4 = -5337.446698,\ a5 = 21349.78679] \qquad \textbf{(2.6.2.2)}$$

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

$$opening\_profile := 21349.78679\, t^5 - 5337.446698\, t^4 + 355.8297799\, t^3 + 4.571785329 \times 10^{-7} \qquad \textbf{(2.6.2.3)}$$

Still part

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

$$still\_profile := 0.0355834351678901 \qquad \textbf{(2.6.2.4)}$$

Closing part

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

$$closing\_coefficients := [a0 = 378.0691416,\ a1 = -4269.957359,\ a2 = 19214.80811,\ a3 = -43055.40337,\ a4 = 48037.02029,\ a5 = -21349.78679] \qquad \textbf{(2.6.2.5)}$$

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

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
  );
```

$$s\_profile := \begin{cases} 21349.78679\,t^5 - 5337.446698\,t^4 + 355.8297799\,t^3 + 4.571785329 \times 10^{-7} & \\ 0.0355834351678901 & \\ -21349.78679\,t^5 + 48037.02029\,t^4 - 43055.40337\,t^3 + 19214.80811\,t^2 - 4269.957359\,t + 378.0 & \end{cases}$$

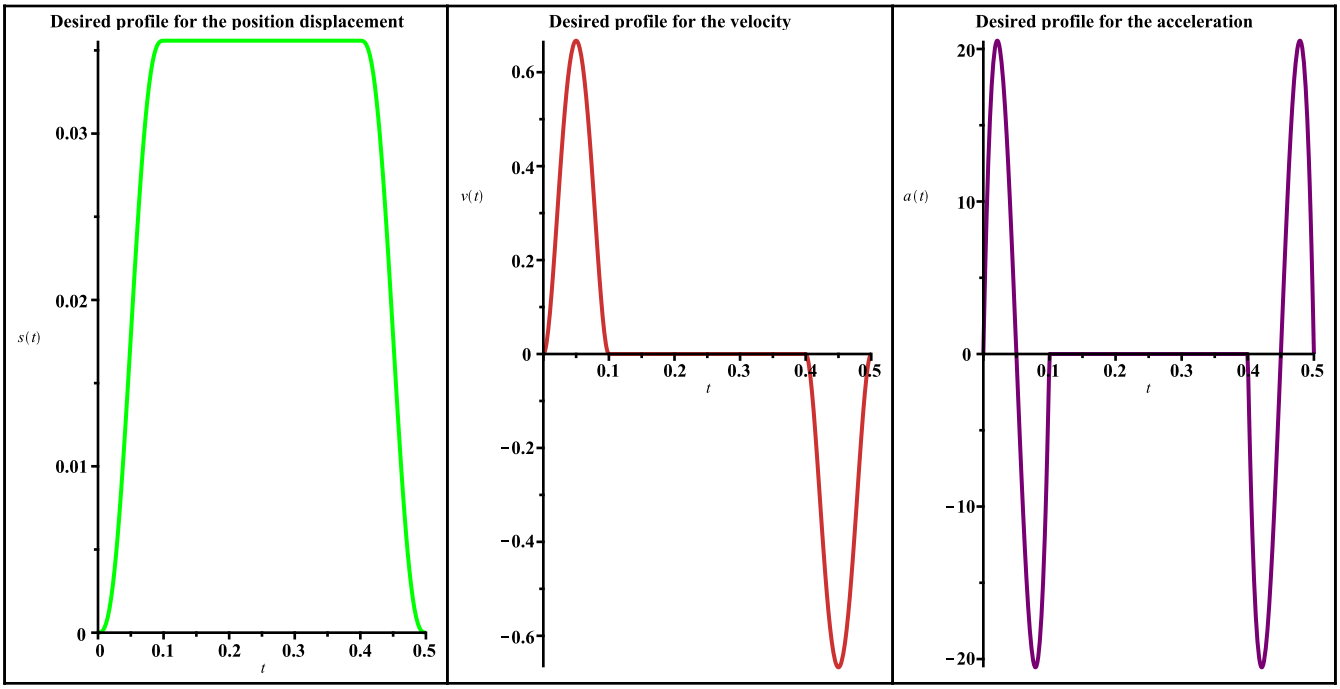
```
> s_vel_profile:=subs(diff(s_profile,t));
```

$$s\_vel\_profile := \begin{cases} 106748.9340\,t^4 - 21349.78679\,t^3 + 1067.489340\,t^2 & \\ 0 & T_{o} \\ -106748.9340\,t^4 + 192148.0812\,t^3 - 129166.2101\,t^2 + 38429.61622\,t - 4269.957359 & T \end{cases}$$

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

$$s\_acc\_profile := \begin{cases} 426995.7360\,t^3 - 64049.36037\,t^2 + 2134.978680\,t & 0 \le t \le T_{ope} \\ 0 & T_{opening} < t \le T_{oper} \\ -426995.7360\,t^3 + 576444.2436\,t^2 - 258332.4202\,t + 38429.61622 & T_{opening} + T_{still} < t \end{cases}$$

```
> profiles:=subs(data,[
      diff(s(t),t,t)=s_acc_profile,
      diff(s(t),t)=s_vel_profile,
      s(t)=s_profile
  ]):
> display(Array([plot(subs(data,s_profile),t=0..T__tot,color=green,
  title="Desired profile for the position displacement",font=
  [TIMES,BOLD,12],labels=[t,s(t)]),
  plot(subs(data,s_vel_profile),t=0..T__tot,color=orange,title=
  "Desired profile for the velocity",font=[TIMES,BOLD,12],labels=
  [t,v(t)]),
  plot(subs(data,s_acc_profile),t=0..T__tot,color=purple,title=
  "Desired profile for the acceleration",font=[TIMES,BOLD,12],
  labels=[t,a(t)])
  ]))
```



Known conditions

Initial conditions

Position

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

$$ics\_pos := [s(t) = 4.571785329 \times 10^{-7},\ \psi 2(t) = -0.7325850177,\ \psi 3(t) = 1.087104759] \quad \textbf{(2.7.1.1)}$$

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;
```

(2.7.2.1)

$$ics\_vel := \left[\frac{d}{dt} s(t) = 0., \frac{d}{dt} \psi 2(t) = -0., \frac{d}{dt} \psi 3(t) = 0. \right] \quad \textbf{(2.7.2.1)}$$

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;
```

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

Dynamic

```
> _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):
```

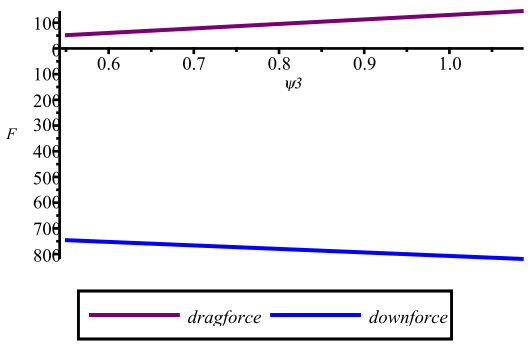
Acting forces

Piston force

```
> piston_force:=make_VECTOR(RF0,-F__piston(t),0,0):
> 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,
      legend = [dragforce, downforce],
      labels = [psi3, F]
  );
```



Newton Euler

Equation of motion

Internal forces

```
> PJ_force := make_FORCE(make_VECTOR(ground,0,Np(t),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: <%>;
```

$$\left[\left[\left[-m_{pist}\left(\frac{d^2}{dt^2}\ s(t)\right) + F_{piston}(t) - rj1x(t)\right],\right.\right. \tag{3.1.1.1}$$

$$\left[m_{pist}\, g - Np(t) - rj1y(t)\right],$$

$$\left[\frac{L1\ (Np(t) - 4\ rj1y(t))}{5} - Tp(t)\right],$$

$$\left[-m_{link}\left(\frac{d^2}{dt^2}\ s(t) + \frac{L2\left(\left(\frac{d}{dt}\ \psi 2(t)\right)^2 \cos(\psi 2(t)) + \left(\frac{d^2}{dt^2}\ \psi 2(t)\right)\sin(\psi 2(t))\right)}{2}\right)\right.$$

$$\left. + rj1x(t) - rj2x(t)\right],$$

$$\left[\frac{m_{link}\, L2\left(-\left(\frac{d}{dt}\ \psi 2(t)\right)^2 \sin(\psi 2(t)) + \left(\frac{d^2}{dt^2}\ \psi 2(t)\right)\cos(\psi 2(t))\right)}{2} + m_{link}\, g\right.$$

$$\left. + rj1y(t) - rj2y(t)\right],$$

$$\left[Iz_{link}\left(\frac{d^2}{dt^2}\ \psi 2(t)\right)\right.$$

$$\left. + \frac{((-rj1y(t) - rj2y(t))\cos(\psi 2(t)) + \sin(\psi 2(t))\ (rj1x(t) + rj2x(t)))\ L2}{2}\right],$$

$$\left[\left(-0.001587809302\cos(\psi 3(t))\ m_{wing} + \left(-0.05172339628 + L_{wing}\right.\right.\right.$$

$$\left.\left.\left. - 1.\, d_{tip}\right) m_{wing}\sin(\psi 3(t))\right)\left(\frac{d^2}{dt^2}\ \psi 3(t)\right) + \left(\left(-0.05172339628 + L_{wing}\right.\right.$$

$$\left.\left. - 1.\, d_{tip}\right) m_{wing}\cos(\psi 3(t)) + 0.001587809302\, m_{wing}\sin(\psi 3(t))\right)\left(\frac{d}{dt}\ \psi 3(t)\right)^2$$

$$\left. - 1.\, F_{drag}(t) + rj2x(t) - 1.\, rj3x(t)\right],$$

$$\left[\left(\left(0.05172339628 - 1.\, L_{wing} + d_{tip}\right) m_{wing}\cos(\psi 3(t))\right.\right.$$

$$-0.001587809302\, m_{wing} \sin(\psi 3(t))\left(\frac{d^2}{dt^2}\,\psi 3(t)\right) + ($$

$$-0.001587809302 \cos(\psi 3(t))\, m_{wing} + (-0.05172339628 + L_{wing}$$

$$-1.\, d_{tip})\, m_{wing} \sin(\psi 3(t))\left(\frac{d}{dt}\,\psi 3(t)\right)^2 + m_{wing}\, g + F_{down}(t) + rj2y(t)$$

$$-1.\, rj3y(t)\Big],$$

$$\Big[Iz_{wing}\left(\frac{d^2}{dt^2}\,\psi 3(t)\right) + ((-0.05172339628 + L_{wing} - 1.\, d_{tip})\, rj2y(t)$$

$$+ 0.001587809302\, rj2x(t) - 0.001587809302\, rj3x(t) + (0.05172339628 - 1.\, L_{wing}$$

$$+ d_{tip})\, rj3y(t))\cos(\psi 3(t)) + (0.001587809302\, rj2y(t) + (0.05172339628 - 1.\, L_{wing}$$

$$+ d_{tip})\, rj2x(t) + (-0.05172339628 + L_{wing} - 1.\, d_{tip})\, rj3x(t)$$

$$- 0.001587809302\, rj3y(t))\sin(\psi 3(t)) + rj2y(t)\, L2 \cos(\psi 2(t))$$

$$-1.\, rj2x(t)\, L2 \sin(\psi 2(t)) + (-1.\, xD - 1.\, s(t) + L1 + xA)\, rj2y(t) + (-1.\, yA$$

$$+ yD)\, rj2x(t)\,]]$$

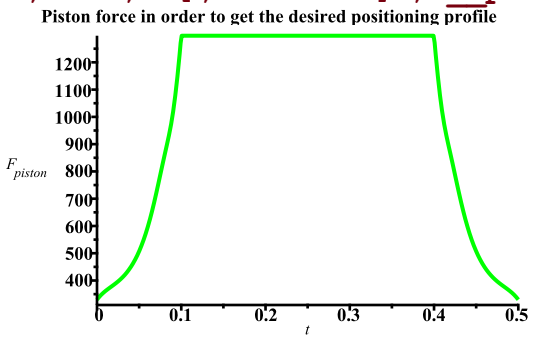
Inverse dynamic

Piston force profile given trajectory

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

$$9 \qquad \textbf{(3.1.2.1)}$$

```
> 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,
  title="Piston force in order to get the desired positioning
  profile",font=[TIMES,BOLD,11], labels=[t,F__piston]);
```



Direct dynamic

Variable profile given the force.

In practice this is 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)): <%>;
```

$$\left[\begin{array}{c} Np(t) = 0.9687375000 + 0.8995836339\, F_{piston}(t) \\ Tp(t) = 0.02844653278 + 0.07506665592\, F_{piston}(t) \\ rj1x(t) = F_{piston}(t) \\ rj1y(t) = -0.1839375000 - 0.8995836339\, F_{piston}(t) \\ rj2x(t) = F_{piston}(t) \\ rj2y(t) = 0.1839375000 - 0.8995836339\, F_{piston}(t) \\ rj3x(t) = -145.5132278 + F_{piston}(t) \\ rj3y(t) = 838.9209070 - 0.8995836339\, F_{piston}(t) \end{array}\right] \qquad \textbf{(3.1.3.1)}$$

```
> 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));
```

$$ics\_dae\_NE := [\mathrm{D}(s)(0) = 0., \mathrm{D}(\psi 2)(0) = -0., \mathrm{D}(\psi 3)(0) = 0., s(0) = 4.571785329 \times 10^{-7}, \psi 2(0) = -0.7325850177, \psi 3(0) = 1.087104759, Np(0) = 297.6541768, Tp(0) = 24.78565970, rj1x(0) = 329.8030646, rj1y(0) = -296.8693768, rj2x(0) = 329.8030646, rj2y(0) = -296.5015018, rj3x(0) = 184.2898368, rj3y(0) = 542.2354677] \qquad \textbf{(3.1.3.2)}$$

```
> 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):
          solved:=true:
      catch:
          print("Exception generated, automatic retrial"):
      end try;
  until solved:
  "computation time"=time() - st;
```

"Exception generated, automatic retrial"

"Exception generated, automatic retrial"

"Exception generated, automatic retrial"

"Exception generated, automatic retrial"

"computation time" = 21.437 **(3.1.3.3)**

```
> dsol_NE;
```

$$\left[t = \mathbf{proc}(t)\ \ldots\ \mathbf{end\ proc},\ Np(t) = \mathbf{proc}(t)\ \ldots\ \mathbf{end\ proc},\ Tp(t) = \mathbf{proc}(t)\ \ldots\ \mathbf{end\ proc},\ \psi 2(t)\right. \quad \textbf{(3.1.3.4)}$$

$$= \mathbf{proc}(t)\ \ldots\ \mathbf{end\ proc},\ \frac{d}{dt}\ \psi 2(t) = \mathbf{proc}(t)\ \ldots\ \mathbf{end\ proc},\ \psi 3(t) = \mathbf{proc}(t)$$

...

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

$$\mathbf{proc}(t)$$

...

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

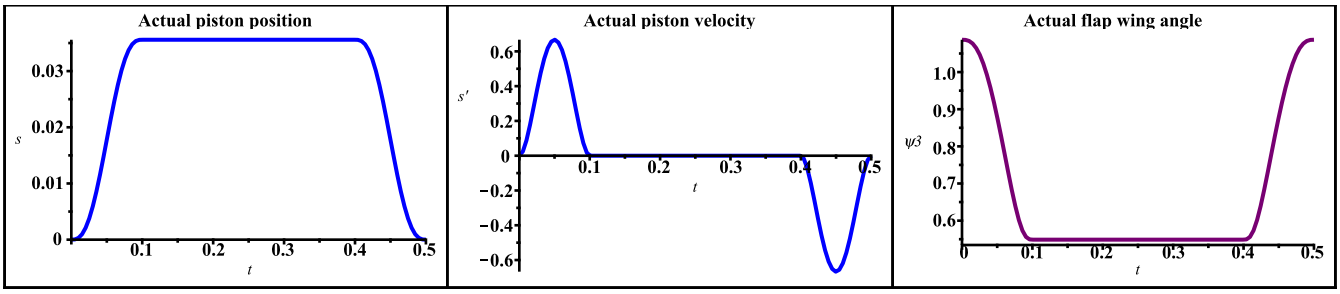
...

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

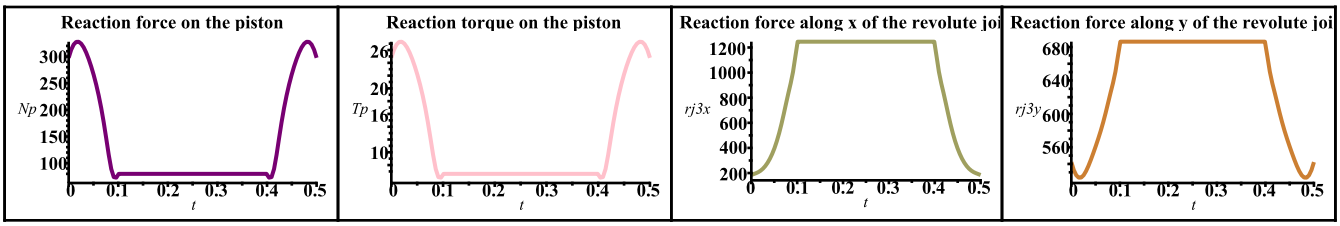
...

$$\mathbf{end\ proc}\,]$$

```
> ## plot ##
  display(Array([
  odeplot(dsol_NE,[t,s(t)],t=0..T__tot,color=blue,title="Actual
  piston position",size=SMS,numpoints=100,font=[TIMES,BOLD,12]),
  odeplot(dsol_NE,[t,diff(s(t),t)],t=0..T__tot,color=blue,title=
  "Actual piston velocity",size=SMS,numpoints=100,font=[TIMES,BOLD,
  12]),
  odeplot(dsol_NE,[t,psi3(t)],t=0..T__tot,color=purple, title=
  "Actual flap wing angle",size=SMS,numpoints=100,font=[TIMES,BOLD,
  12])
  ]));
```




```
> ## plot ##
  display(Array([
  odeplot(dsol_NE,[t,Np(t)],t=0..T__tot,color=purple, title=
  "Reaction force on the piston",size=SMS,numpoints=100,font=
  [TIMES,BOLD,12]),
  odeplot(dsol_NE,[t,Tp(t)],t=0..T__tot,color=pink, title="Reaction
  torque on the piston",size=SMS,numpoints=100,font=[TIMES,BOLD,12]
  ),
  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,font=[TIMES,BOLD,12]),
  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,font=[TIMES,BOLD,12])
  ]));
```



Effect of friction

Note: the correct expression implies using abs(Np(t)), but in the current setup Np(t) alone gives the same result, so we save a lot of time in the computation.

```
> piston_force_fric:=make_VECTOR(RF0,-F__piston(t) + tanh(diff(s
  (t),t))*eta__fric*(Np(t)),0,0):
> FP:=make_FORCE(piston_force_fric,PP,PIST):
> 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
```

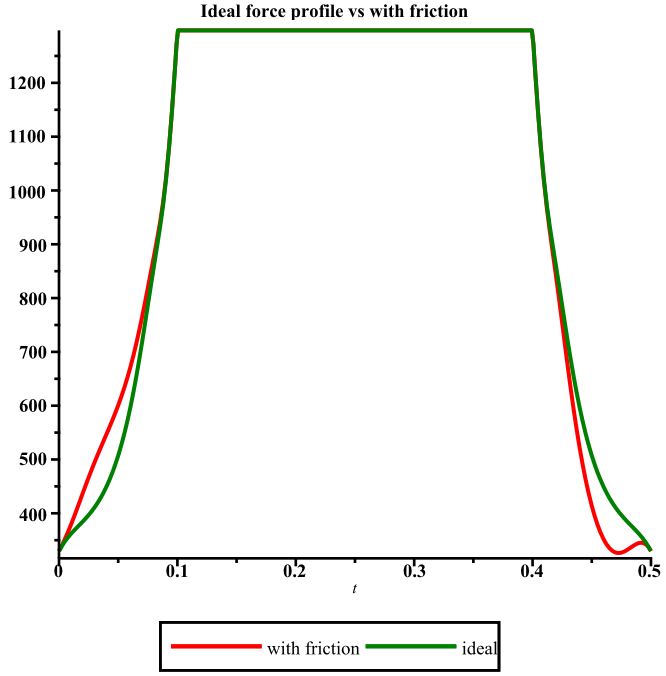
```
<FP> FORCE is not valid: it must be applied to a BODY
<PJ_force> FORCE is not valid: it must be applied to a BODY
<RJ3_force> FORCE is not valid: it must be applied to a BODY
```

```
> 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_fric := NE_eqns1 union NE_eqns2 union NE_eqns3:
> sol_rvars_fric:=op(solve(eqns_NE_fric,rvars union [F__piston(t)])
  ):
  nops(%);
```

9 **(3.1.4.1)**

```
> piston_force_NE_fric:=simplify(combine(rhs(sol_rvars_fric[9]))):
> piston_force_NE_profile_fric:=subs(air_forces_law,acc_kin_sol,
  vel_kin_sol,kin_sol,profiles,data,eta__fric=0.6,
  piston_force_NE_fric):
> display([
          plot(piston_force_NE_profile_fric,subs(data,t=0..T__tot),
  color=red),
          plot(piston_force_NE_profile,subs(data,t=0..T__tot),
  color="Green")
  ],
  title="Ideal force profile vs with friction",font=[TIMES,BOLD,
  12],legend=["with friction","ideal"],legendstyle=[font=[TIMES,
  BOLD,12]]);
```



Lagrange

Equation of motion

```
> FP:=make_FORCE(piston_force,PP,PIST):
```

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

$$forces := \{FA, FP, PJ\_force, PJ\_torque, RJ1\_force, RJ2\_force, RJ3\_force\} \quad (3.2.1.1)$$

Constraints defintion

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

$$constraints := \mathrm{table}\left(\left[expr = \left[-\sin(\psi 3(t))\, H3 - L2\cos(\psi 2(t)) - \cos(\psi 3(t))\, L3 - L1 + s(t) - xA + xD,\ \cos(\psi 3(t))\, H3 - L2\sin(\psi 2(t)) - \sin(\psi 3(t))\, L3 - yA + yD\right],\ obj = CONSTRAINT,\ vars = \left[\lambda 1(t), \lambda 2(t)\right]\right]\right) \quad (3.2.1.2)$$

Lagrange equations

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

$$\left[\left[\frac{\left(2\, m_{link} + 2\, m_{pist}\right)\left(\frac{d^2}{dt^2}\, s(t)\right)}{2} + \frac{L2\left(\frac{d^2}{dt^2}\, \psi 2(t)\right)\sin(\psi 2(t))\, m_{link}}{2} + \frac{L2\left(\frac{d}{dt}\, \psi 2(t)\right)^2\cos(\psi 2(t))\, m_{link}}{2} + \lambda 1(t) - F_{piston}(t)\right],\right. \quad (3.2.1.3)$$

$$\left[\frac{\left(L2^2\, m_{link} + 4\, Iz_{link}\right)\left(\frac{d^2}{dt^2}\, \psi 2(t)\right)}{4} + \frac{m_{link}\, L2\left(\frac{d}{dt}\, \psi 2(t)\right)\cos(\psi 2(t))\left(\frac{d}{dt}\, s(t)\right)}{2} + \frac{m_{link}\, L2\sin(\psi 2(t))\left(\frac{d^2}{dt^2}\, s(t)\right)}{2} + \frac{1}{2}\left(\left(-\left(\frac{d}{dt}\, \psi 2(t)\right)\cos(\psi 2(t))\left(\frac{d}{dt}\, s(t)\right) m_{link} + \left(m_{link}\, g - 2\, \lambda 2(t)\right)\cos(\psi 2(t)) + 2\sin(\psi 2(t))\, \lambda 1(t)\right) L2\right)\right],$$

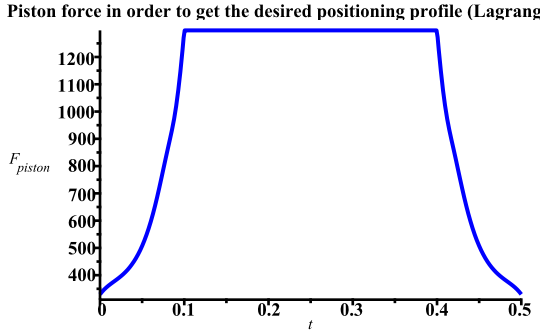
$$\left[\left(\left({L_{wing}}^2 + \left(-2.\, d_{tip} - 0.1034467926\right) L_{wing} + 0.002677830862 + 0.1034467926\, d_{tip} + {d_{tip}}^2\right) m_{wing} + Iz_{wing}\right)\left(\frac{d^2}{dt^2}\, \psi 3(t)\right) - \left(H3\, \lambda 1(t) + \lambda 2(t)\, L3 + \left(-0.05172339628 + L_{wing} - 1.\, d_{tip}\right) g\, m_{wing}\right)\cos(\psi 3(t)) - \left(0.001587809302\, m_{wing}\, g - 1.\, L3\, \lambda 1(t) + H3\, \lambda 2(t)\right)\sin(\psi 3(t)) - \left(\left(-0.05172339628 + L_{wing} - 1.\, d_{tip}\right) F_{down}(t) - 0.001587809302\, F_{drag}(t)\right)\cos(\psi 3(t)) - \left(0.001587809302\, F_{down}(t) + \left(-0.05172339628 + L_{wing} - 1.\, d_{tip}\right) F_{drag}(t)\right)\sin(\psi 3(t))\right],$$

$$\left[\left[-\sin(\psi 3(t))\,H3 - L2\cos(\psi 2(t)) - \cos(\psi 3(t))\,L3 - L1 + s(t) - xA + xD\right],\left[\cos(\psi 3(t))\,H3 - L2\sin(\psi 2(t)) - \sin(\psi 3(t))\,L3 - yA + yD\right]\right]$$

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 ##
  plot(piston_force_lagr_profile,t=0..T__tot,size=SMS,color=blue,
  font=[TIMES,BOLD,11], labels=[t,F__piston],title="Piston force in
  order to get the desired positioning profile (Lagrange)");
```



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_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)):
> sys_lagr:=subs(
      air_forces_law,
      F__piston(t)=piston_force_lagr_profile,
```

```
        data,
        eqns_lagr union ics_dae_lagr
    ):
```

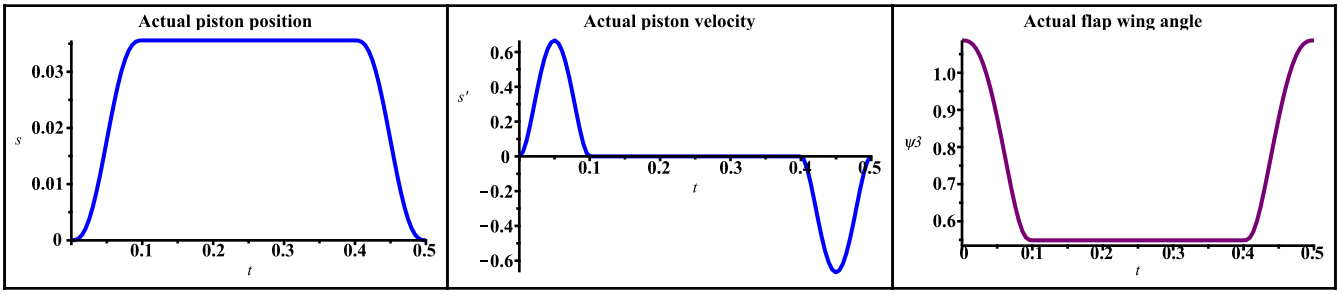
System solution

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):
          solved:=true:
      catch:
          print("Exception generated, automatic retrial"):
      end try;
  until solved:
  "computation time"=time() - st;
```

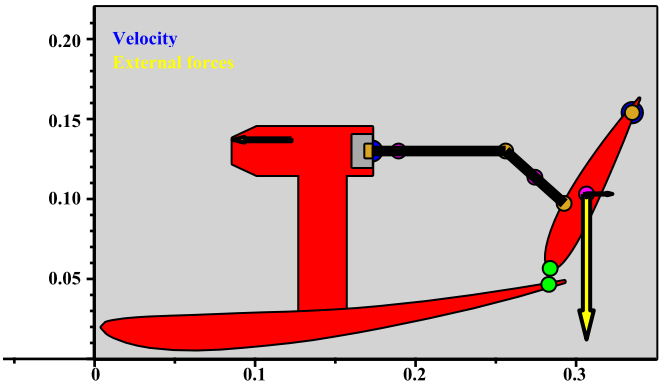
"computation time" = 16.938 **(3.2.3.1)**

```
> ## plot ##
  display(Array([
  odeplot(dsol_lagr,[t,s(t)],t=0..T__tot,color=blue,title="Actual
  piston position",size=SMS,numpoints=100,font=[TIMES,BOLD,12]),
  odeplot(dsol_lagr,[t,diff(s(t),t)],t=0..T__tot,color=blue,title=
  "Actual piston velocity",size=SMS,numpoints=100,font=[TIMES,BOLD,
  12]),
  odeplot(dsol_lagr,[t,psi3(t)],t=0..T__tot,color=purple, title=
  "Actual flap wing angle",size=SMS,numpoints=100,font=[TIMES,BOLD,
  12])
  ]));
```




```
> dsol_lagr(0.01):
  psi2__vec := [evalf(seq((dsol_lagr(i1)[4]),subs(data,i1=0..
  T__tot+0.001),0.01))]:
  psi3__vec := [evalf(seq((dsol_lagr(i1)[6]),subs(data,i1=0..
  T__tot),0.01))]:
  Fpist__vec := [evalf(seq(F__piston(t)=eval
  (piston_force_lagr_profile,t=i1),subs(data,i1=0..T__tot),0.01))]:
  s__vec := [evalf(seq(dsol_lagr(i1)[-2],subs(data,i1=0..T__tot),
  0.01))]:
  diff_s__vec := subs(diff(s(t),t)=vel,[evalf(seq(dsol_lagr(i1)
  [-1],subs(data,i1=0..T__tot),0.01))]):
> display([seq(draw_mech_dyna([psi2__vec[i1],psi3__vec[i1],
  diff_s__vec[i1], s__vec[i1],Fpist__vec[i1]],data,aa=aaa,s_range),
  i1=1..51)],insequence=true,title="Dynamics of the system",font=
```

```
[TIMES,BOLD,12]);
```



Virtual work

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

Static case

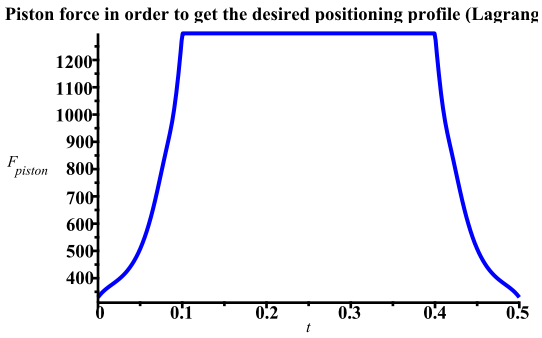
```
> PVW_static:=
      dot_prod(piston_force,vel_PP)
      +dot_prod(air_forces,vel_P3)
      +dot_prod(make_VECTOR(ground,0,-m__pist*g,0),vel_P1)
      +dot_prod(make_VECTOR(ground,0,-m__link*g,0),vel_P2)
      +dot_prod(make_VECTOR(ground,0,-m__wing*g,0),vel_P3):
> piston_force_pvw_static := rhs(op(solve(PVW_static,{F__piston(t)}
  ))):
> 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,color=blue,font=[TIMES,BOLD,11], labels=[t,F__piston],title=
  "Piston force without considering inertia forces(PVW)");
```

Principle of d'Alembert

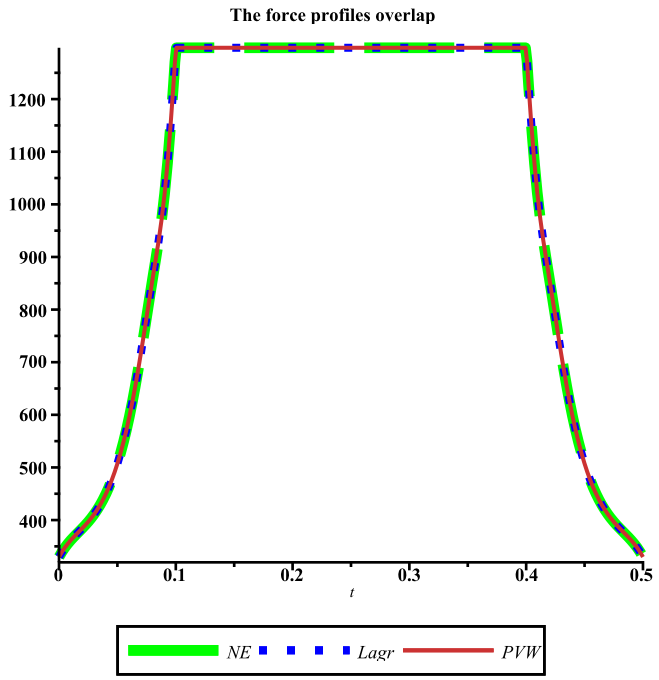
Dynamic case.

```
> 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=
  blue,font=[TIMES,BOLD,11], labels=[t,F__piston],title="Piston
  force in order to get the desired positioning profile (Lagrange)
  ");
```




```
> display([
          plot(piston_force_NE_profile,t=0..T__tot,thickness=4,
  linestyle=dash),
          plot(piston_force_lagr_profile,t=0..T__tot,thickness=3,
  linestyle=dot),
          plot(piston_force_pvw_dynamic_profile,t=0..T__tot)
      ],
      color=[green,blue,orange],
      legend=[NE,Lagr,PVW],
      title="The force profiles overlap",
      font=[TIMES,BOLD,12]
  );
  ## plot ## plot(piston_force_NE_profile,t=0..T__tot,color=green,
  title="Newton-Euler"),
  ## plot ## plot(piston_force_lagr_profile,t=0..T__tot,color=blue,
  title="Lagrange"),
  ## plot ## 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, just take one.

```
> piston_force_to_optimize := subs(air_forces_law,acc_kin_sol,
  vel_kin_sol,kin_sol,profiles,pre_fixed_data,piston_force_NE):
> piston_force_profile := piston_force_NE_profile:
```

Optimization

Setup

```
> guess_data;
```

$$[yA = 0.130000,\ L2 = 0.049000,\ L3 = 0.070000] \tag{4.1.1}$$

```
> opt_vars := map(x->lhs(x),guess_data);
```

$$opt\_vars := [yA, L2, L3] \tag{4.1.2}$$

```
> yA__min  := 0.130000;
  yA__max  := 0.160000;
  L2__min  := 0.030000;
  L2__max  := 0.070000;
  L3__min  := 0.050000;
  L3__max  := 0.110000;
```

$$yA_{min} := 0.130000$$
$$yA_{max} := 0.160000$$
$$L2_{min} := 0.030000$$
$$L2_{max} := 0.070000$$
$$L3_{min} := 0.050000$$
$$L3_{max} := 0.110000 \tag{4.1.3}$$

```
> yA_range:=yA__min..yA__max;
```

$$yA\_range := 0.130000\,..0.160000 \tag{4.1.4}$$

```
> L2_range:=L2__min..L2__max;
```

$$L2\_range := 0.030000 .. 0.070000 \quad \textbf{(4.1.5)}$$

```
> L3_range:=L3__min..L3__max;
```

$$L3\_range := 0.050000 .. 0.110000 \quad \textbf{(4.1.6)}$$

```
> number_of_frame := 10:
> number_of_step := 3:
> psi3_step := (psi3_closed-psi3_open)/number_of_step:
  "psi3_step_deg"=%*180/Pi
```

$$\text{"psi3\_step\_deg"} = 10.28122003 \quad \textbf{(4.1.7)}$$

```
> t_step := T__tot/number_of_step;
```

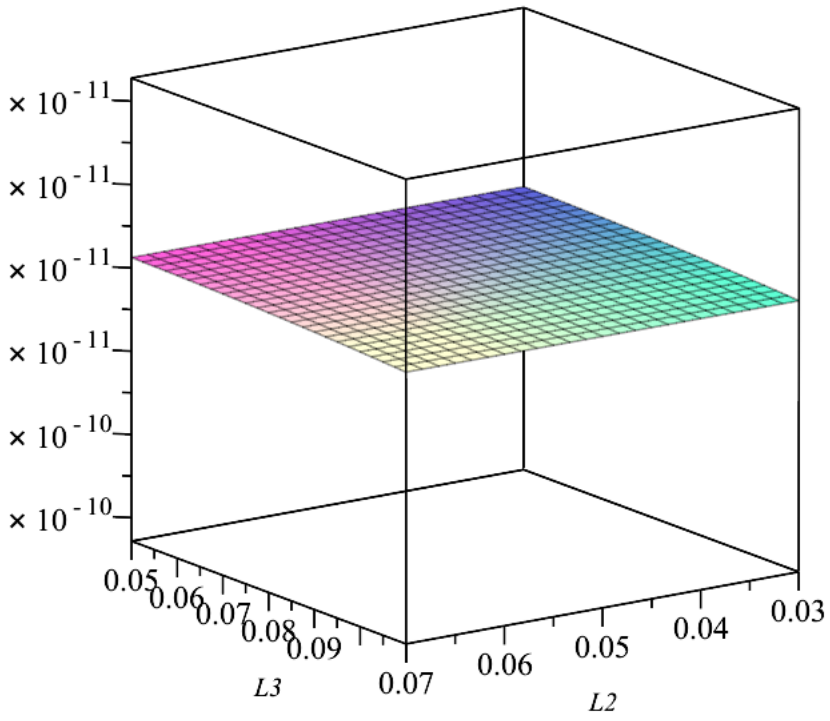
$$t\_step := 0.1666666667 \quad \textbf{(4.1.8)}$$

NB: all the plots are done with H3 fixed. For the minimisation, its value will be computed depending on the value of the guess.

Respect of the kinematic constraint

```
> psi3_Phi:= simplify(combine(subs(psi3_kin_sol,pre_fixed_data,1/2*
  DotProduct(Phi,Phi,conjugate=false))));
> C_phi_vector:=[seq(subs(psi3=i,psi3_Phi),i=psi3_open..
  psi3_closed,psi3_step)]:
> C_phi:=0:
  for x in C_phi_vector do
       C_phi:=C_phi+x:
  end do:
> ## animate ##
  animate(plot3d,[subs(yA=YA,H3_eqn,C_phi),L2=L2_range,L3=
  L3_range],YA=yA_range);
```

$YA = 0.13000$

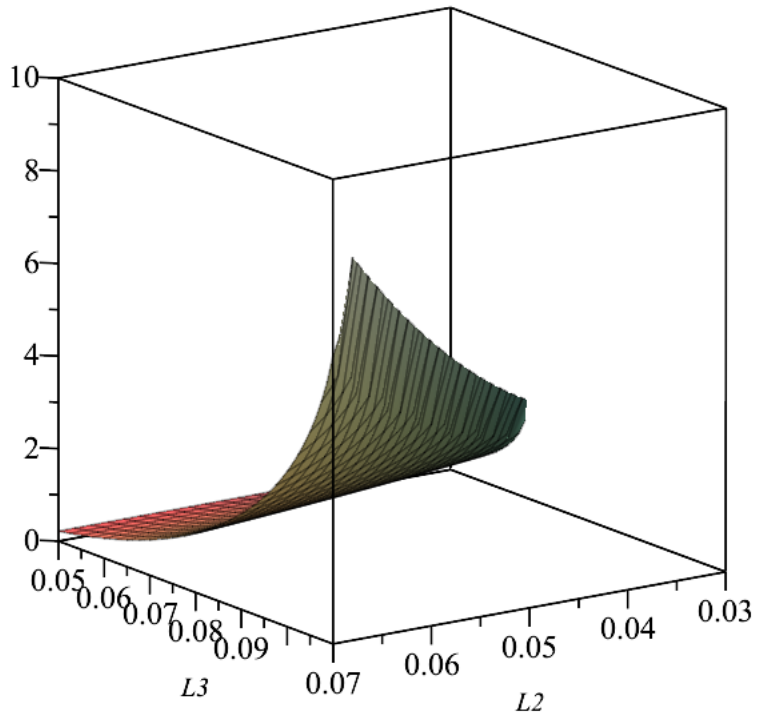


Minimisation of the piston stroke

```
> C_stroke:=(subs(psi3_kin_sol,psi3(t)=psi3_open,s(t))
            -subs(psi3_kin_sol,psi3(t)=psi3_closed,s(t)))^2/
            s_stroke^2:
  ## check ## subs(data,sqrt(C_stroke));
```

```
> ## animate ##
  animate(plot3d,[subs(yA=YA,H3_eqn,C_stroke),L2=L2_range,L3=
  L3_range,view=0..10],YA=yA_range);
```

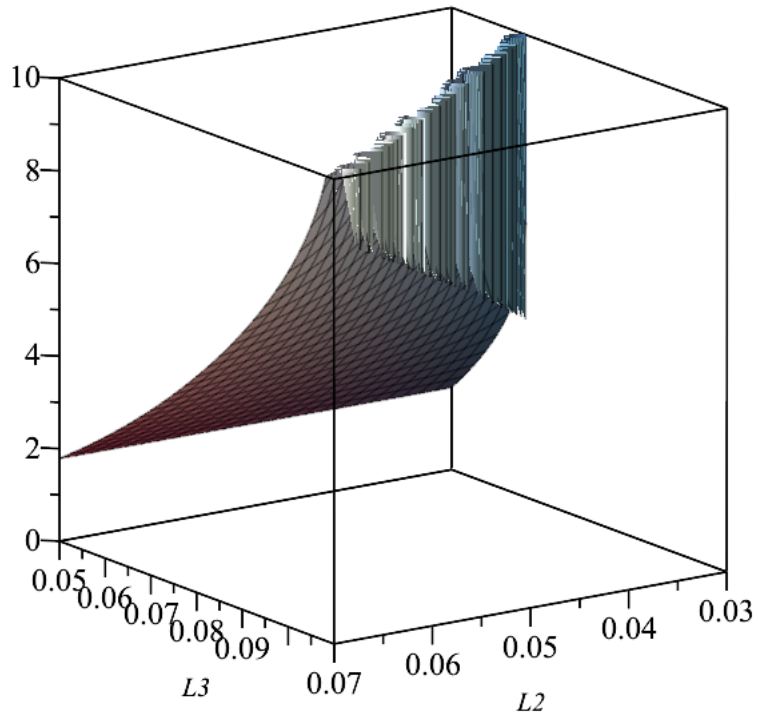
$YA = 0.13000$



Maximization of the velocity

```
> psi3_tau_inv := subs(psi3_kin_sol,pre_fixed_data,-1/tau[2][1]):
> C_vel_vector := [seq(subs(psi3(t)=psi3,psi3_tau_inv),psi3=
  psi3_open..psi3_closed,psi3_step)]:
> C_vel := 0:
  for x in C_vel_vector do
       C_vel := C_vel+x/evalf(subs(data,x)):
  end do:
> ## animate ##
  animate(plot3d,[subs(yA=YA,H3_eqn,C_vel),L2=L2_range,L3=L3_range,
  view=0..10],YA=yA_range);
```

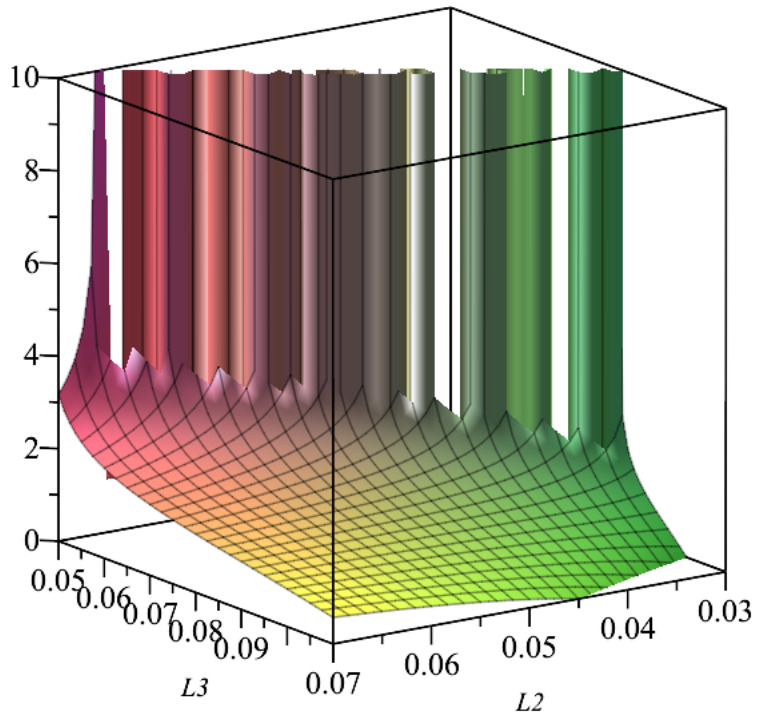
$YA = 0.13000$



Minimization of the force

```
> # it takes lot of time (order of minutes) but it finishes
> C_force_vector := [seq(
      evalf(eval(subs(t=TT,piston_force_to_optimize))),
      TT=0..T__tot,t_step
  )]:
> C_force := 0:
  for x in C_force_vector do
       C_force := C_force+x/evalf(subs(data,x)):
  end do:
> ## animate ##
  animate(plot3d,[subs(H3_eqn,yA=YA,C_force),L2=L2_range,L3=
  L3_range,view=0..10],YA=yA_range,frames=5);
```

$YA = 0.13000$

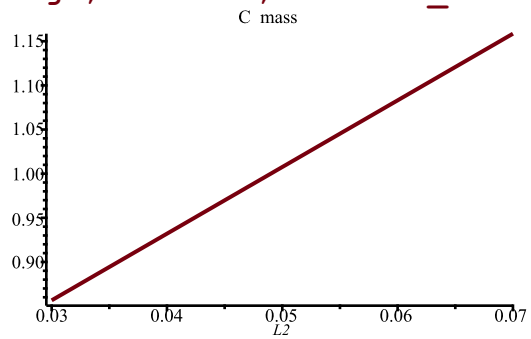


Minimisation of the mechanism mass

```
> parametric_mass := subs(pre_fixed_data,L1*rho__steel+L2*
  rho__steel):
  C_mass:=parametric_mass/evalf(subs(data,parametric_mass));
```

$$C\_mass := 7.550246895\ L2 + 0.6300379024 \qquad (4.6.1)$$

```
> ## plot ##
  plot(C_mass,L2=L2_range,size=SMS,title="C_mass");
```



Global cost function

```
> obj := C_phi+C_stroke+C_vel+C_force+C_mass:
> evalf(subs(data,obj));
```

$$8.99999925730586 \qquad (4.7.1)$$

```
> ## animate ## animate(plot3d,[subs(yA=YA,H3_eqn,obj),L2=L2_range,
```

```
  L3=L3_range,view=0...40],YA=yA_range);#,frames=10);
```

Cost computation procedure

```
> p := proc(params)    # variables definition
      local H3_eqn_tmp,i,cost:

      # params elaboration
      H3_eqn_tmp :=H3=getWingHeight(subs(params, pre_fixed_data,
  L__wing-L3),flap_wing_matrix_point):

      # verbose option
      if false then
          print(H3_eqn_tmp);
      end if:

      # cost computation and check
      cost := evalf(subs(params, H3_eqn_tmp, pre_fixed_data, obj));


      # verbose option
      if false then
          print(cost):
      end if:

      if Im(cost) <> 0 then
          cost := 10^5:
      end if:

      # verbose option
      if false then
          print(opt_vars_sub),print(cost):
      end if:

      cost:
  end proc:
```

Optimized solution, brute force

First run (raw)

Reduce the range of variables to the range in which the minimum is most likely to be encountered.
Each cell of the cost matrix differs from the others for at least one centimeter in the variables value.

```
> cm_factor := 100;
```

$$cm\_factor := 100 \qquad \textbf{(4.9.1.1)}$$

```
> cost_matrix := Array(
      yA__min*cm_factor..yA__max*cm_factor,
      L2__min*cm_factor..L2__max*cm_factor,
      L3__min*cm_factor..L3__max*cm_factor
  );
```

(4.9.1.2)

$$cost\_matrix := \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \textbf{(4.9.1.2)}$$

slice of 13 .. 16 × 3 .. 7 × 5 .. 11 Array

```
> for i from yA__min by 1/cm_factor to yA__max do
      for j from L2__min by 1/cm_factor to L2__max do
          for k from L3__min by 1/cm_factor to L3__max do
              opt_vars_sub := [
                  yA = i,
                  L2 = j,
                  L3 = k
              ]:
              cost_matrix[MTM:-int32(i*cm_factor),MTM:-int32(j*
  cm_factor),  MTM:-int32(k*cm_factor)] := p(opt_vars_sub):
          end do:
      end do:
  end do:
```

```
> cost_matrix[13];
```

100000	100000	100000	100000	100000	
100000	100000	100000	100000	100000	
100000	100000	8.68626814074627	10.0586285148808	100000	
100000	7.36401595841047	7.51176466320427	8.95461786515061	11.4775245008718	
6.54288095677357	6.20725949589498	7.01651308720431	8.37898617995464	10.3388306133403	13.3

Studying the cost matrix, we can reduce the range of the variable and improve the research to millimetric values:

Second run (more precise)

```
> new_yA__min  := 0.130000:
  new_yA__max  := 0.140000:
  new_L2__min  := 0.060000:
  new_L2__max  := 0.070000:
  new_L3__min  := 0.050000:
  new_L3__max  := 0.070000:
```

The computation takes some seconds, uncomment if you want to run it. Otherwise the results are those reported.

Results:

```
> min_cost := 10^5:
  opt_data_vars := [yA = 0.130000, L2 = 0.07000000000, L3 =
  0.05700000000]:
```

```
> (*precision_factor := 1000:
  for i from new_yA__min by 1/precision_factor to new_yA__max do
      for j from new_L2__min by 1/precision_factor to new_L2__max
  do
          for k from new_L3__min by 1/precision_factor to
  new_L3__max  do
```

```
                opt_vars_sub := [
                    yA = i,
                    L2 = j,
                    L3 = k
                ]:
                cost := p(opt_vars_sub):
                if cost < min_cost then
                    min_cost := cost:
                    opt_data_vars := opt_vars_sub:
                end if:
            end do:
        end do:
    end do:*)
> opt_data_vars
```

$$[yA = 0.130000,\ L2 = 0.07000000000,\ L3 = 0.05700000000] \qquad \textbf{(4.9.2.1)}$$

Optimized data

```
> opt_H3_eqn :=H3=getWingHeight(subs(opt_data_vars, pre_fixed_data,
  L__wing-L3),flap_wing_matrix_point):
> opt_fixed_data := pre_fixed_data union evalf(subs(opt_data_vars,
  pre_fixed_data,[
            Iz__pist   = 0,
            Iz__link   = m__link*(L2^2)/12,
            Iz__wing   = m__wing*(L__wing^2)/3
        ])):
> opt_tmp_data :=  [op(opt_data_vars), opt_H3_eqn, op
  (opt_fixed_data)]:
> opt_data := convert(
        convert(opt_tmp_data,set)
        minus {m__link=subs(opt_tmp_data,m__link)}
        union {m__link=subs(opt_tmp_data,L2*rho__steel)},
  list);
```

$$opt\_data := \Big[H3 = 0.01082898000,\ HEIGHT = 0.220000,\ L1 = 0.083446,\ L2 \qquad \textbf{(4.9.3.1)}$$

$$= 0.07000000000,\ L3 = 0.05700000000,\ L_{wing} = 0.120000,\ WIDTH = 0.350000,\ W_{wing}$$

$$= 1010.000,\ d_{tip} = 0.0100,\ d_{wing} = 0.0060,\ g = 9.81,\ \gamma = \frac{\pi}{36},\ kp = 10000,\ kpv = 10000,\ m_{link}$$

$$= 0.05250000000,\ m_{pist} = 0.0800,\ m_{wing} = 2.0000,\ xA = 0.172675,\ xD = 0.335000,\ xR$$

$$= 0.280000,\ yA = 0.130000,\ yD = 0.154000,\ yR = 0.022300,\ F_{down\_closed} = 819.11694,$$

$$F_{down\_open} = 745.62411,\ F_{drag\_closed} = 145.51319,\ F_{drag\_open} = 51.32626,\ Iz_{link}$$

$$= 0.00001531250000,\ Iz_{pist} = 0.,\ Iz_{wing} = 0.009600000000,\ T_{closing} = 0.100,\ T_{opening}$$

$$= 0.100,\ T_{still} = 0.300,\ max\_dist = 0.050000,\ min\_dist = 0.010000,\ \rho_{steel} = 0.75\Big]$$

Maple memory bug

We found out that at this point Maple forgot the values of the points we used initially. For this reason, we redefine them here.

```
> PA := make_POINT(ground,xA,yA,0):
  PD := make_POINT(ground,xD,yD,0):
  PP := origin(RF1):
  PB := origin(RF2):
  PC := make_POINT(RF2,L2,0,0):
  PC_3 := make_POINT(RF3,-L3,H3,0):
  PT := origin(RF_flap_wing):
  PR := make_POINT(RF_main_wing,xR,yR,0):
  G1 := make_POINT(RF1,L1/5,0,0):
  G2 := make_POINT(RF2,vCB[1]/2,vCB[2]/2,0):
  G3 := make_POINT(RF_flap_wing,xG3,yG3,0):
```

Optimized kinematic

Some changes are just to let the variable s(t) to start approximately from zero.

Initial and final point of the piston stroke for the optimized mechanism

```
> opt_SCs_v1 := evalf(solve(subs(opt_data,Phi union [Determinant
  (JPhiD)=0]),qvars,explicit=true)): <%>;
  nops(SCs);
```

$$\left[\left[\left[s(t) = -0.2046287600,\ \psi 2(t) = 2.953005400,\ \psi 3(t) = 3.140750075\right]\right],\right.$$
$$\left[\left[s(t) = 0.04687076004,\ \psi 2(t) = 0.1885872536,\ \psi 3(t) = 0.3763319288\right]\right],$$
$$\left[\left[s(t) = -0.07887900000 - 0.02079587781\ \mathrm{I},\ \psi 2(t) = 1.570796327 + 1.318838979\ \mathrm{I},\right.\right.$$
$$\left.\left.\psi 3(t) = -1.383051652 + 1.318838979\ \mathrm{I}\right]\right],$$
$$\left[\left[s(t) = -0.07887900000 + 0.02079587781\ \mathrm{I},\ \psi 2(t) = 1.570796327 - 1.318838979\ \mathrm{I},\right.\right.$$
$$\left.\left.\psi 3(t) = -1.383051652 - 1.318838979\ \mathrm{I}\right]\right]\left.\right]$$

$$4 \qquad \textbf{(4.9.5.1)}$$

```
> opt_s_limit := rhs(opt_SCs_v1[2][1])-0.001;
```

$$opt\_s\_limit := 0.04587076004 \qquad \textbf{(4.9.5.2)}$$

```
> opt_s_min := rhs(NLPSolve(subs(kin_sol,notime,opt_data,comp_Y(PT,
  ground)-comp_Y(PR,ground)-min_dist)^2,s=0..opt_s_limit,assume=
  nonnegative)[2][1]);
```

$$opt\_s\_min := 0.0238561534974296 \qquad \textbf{(4.9.5.3)}$$

```
> opt_s_max := rhs(NLPSolve(subs(kin_sol,notime,opt_data,comp_Y(PT,
  ground)-comp_Y(PR,ground)-max_dist)^2,s=0..opt_s_limit,assume=
  nonnegative)[2][1]);
```

$$opt\_s\_max := 0.0453122860709588 \qquad \textbf{(4.9.5.4)}$$

Tests to check distances

```
> "optimized minimum distance" = evalf(subs(kin_sol,s(t)=opt_s_min,
  opt_data,comp_Y(PT,ground)-comp_Y(PR,ground))),    # max 0.010m
  "optimized maximum distance" = evalf(subs(kin_sol,s(t)=opt_s_max,
  opt_data,comp_Y(PT,ground)-comp_Y(PR,ground)));    # max 0.050m
```

$$\text{"optimized minimum distance"} = 0.01000000808,\ \text{"optimized maximum distance"} = 0.05000003815 \qquad \textbf{(4.9.5.5)}$$

```
> opt_s_range := opt_s_min..opt_s_max;
```

$$opt\_s\_range := 0.0238561534974296\ ..0.0453122860709588 \qquad \textbf{(4.9.5.6)}$$

```
> opt_s_stroke := opt_s_max-opt_s_min;
```

$$opt\_s\_stroke := 0.0214561325735292 \qquad \textbf{(4.9.5.7)}$$

Check improvement:

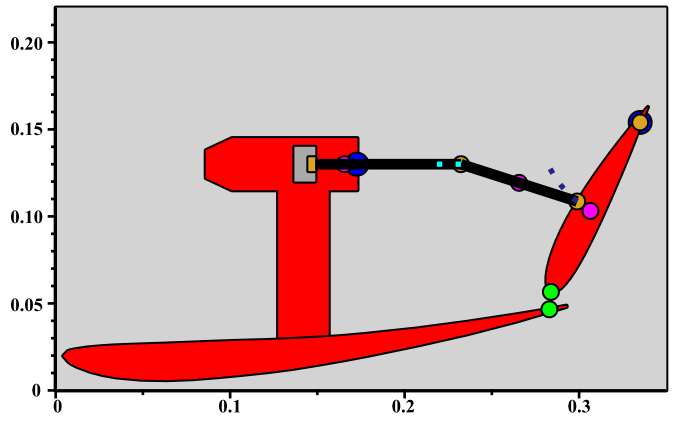
```
> "initial piston stroke"=s_stroke;
  "optimized piston stroke"=opt_s_stroke;
```

"initial piston stroke" = 0.0355829779893572

"optimized piston stroke" = 0.0214561325735292 **(4.9.5.8)**

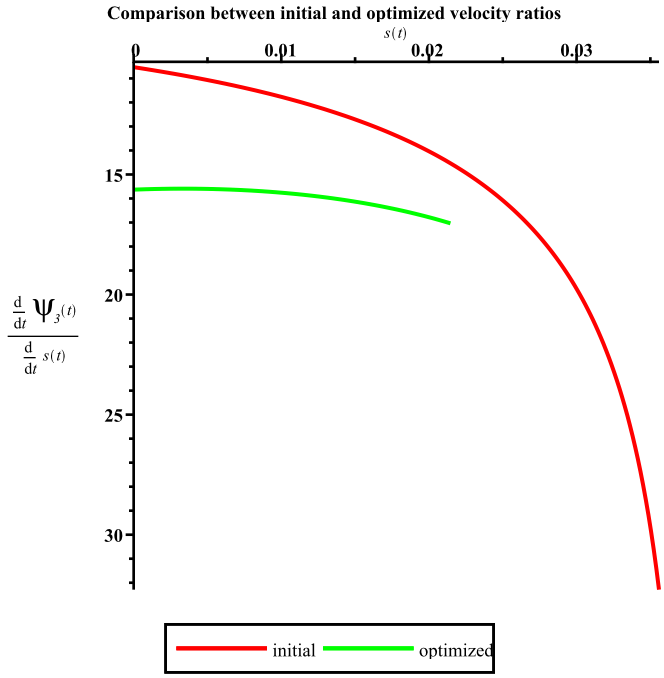
Drawing of the optimized mechanism

```
> ## animate ##
  animate(draw_mech,[kin_sol,opt_data,s(t)=S,opt_s_range,
  pylon_points],S=opt_s_range,title="Kinematics of the optimized
  mechanism", font=[TIMES,BOLD,12]);
```



Velocity ratio of the optimized mechanism

```
> display([
          plot(subs(s(t)=s(t)-s_min,kin_sol,data,s(t)=S,tau[2]), S=
  0..s_max-s_min),
          plot(subs(s(t)=s(t)-opt_s_min,kin_sol,opt_data,s(t)=S,tau
  [2]), S=0..opt_s_max-opt_s_min)
      ],
      title="Comparison between initial and optimized velocity
  ratios",
      color=[red,green],
      legend=["initial","optimized"],
      labels=[s(t),typeset(diff(psi__3(t),t)/diff(s(t),t))],
      font =[TIMES,BOLD,12]
  );
```



Minimum jerk profile with optimized data

```
> opt_opening_known_conditions:=[
      # position
      subs(t=0,opt_data,base_profile=opt_s_min),
      subs(t=T__opening,opt_data,base_profile=opt_s_max),
      # velocity
      subs(t=0,opt_data,diff(base_profile,t)=0),
      subs(t=T__opening,opt_data,diff(base_profile,t)=0),
      # acceleration
      subs(t=0,opt_data,diff(base_profile,t,t)=0),
      subs(t=T__opening,opt_data,diff(base_profile,t,t)=0)
  ]:
> opt_opening_coefficients:=op(solve(opt_opening_known_conditions,
  [seq(a||i,i=0..5)])):
> opt_opening_profile:=subs(opt_opening_coefficients,base_profile);
```

$$opt\_opening\_profile := 12873.67954\,t^5 - 3218.419886\,t^4 + 214.5613257\,t^3 + 0.02385615350 \quad \textbf{(4.9.5.9)}$$

```
> opt_still_profile:=opt_s_max:
> opt_closing_known_conditions:=subs(opt_data,[
      # position
      subs(t=T__opening+T__still,opt_data,base_profile=opt_s_max),
      subs(t=T__tot,opt_data,base_profile=opt_s_min),
      # velocity
      subs(t=T__opening+T__still,opt_data,diff(base_profile,t)=0),
      subs(t=T__tot,opt_data,diff(base_profile,t)=0),
      # acceleration
      subs(t=T__opening+T__still,opt_data,diff(base_profile,t,t)=
  0),
      subs(t=T__tot,opt_data,diff(base_profile,t,t)=0)
  ]):
> opt_closing_coefficients:=op(solve(opt_closing_known_conditions,
  [seq(a||i,i=0..5)]));
```

$$opt\_closing\_coefficients := [a0 = 227.9952647,\ a1 = -2574.735909,\ a2 = 11586.31159,\ a3 \quad \textbf{(4.9.5.10)}$$

$= -25961.92041,\ a4 = 28965.77897,\ a5 = -12873.67954\ ]$

```
> opt_closing_profile:=subs(opt_closing_coefficients,base_profile):
> opt_s_profile:=piecewise(
      t>=0                    and t<=T__opening,
  opt_opening_profile,
      t>T__opening            and t<=T__opening+T__still,
  opt_still_profile,
      t>T__opening+T__still and t<=T__tot,
  opt_closing_profile
  );
```

$$opt\_s\_profile := \begin{cases} 12873.67954\,t^5 - 3218.419886\,t^4 + 214.5613257\,t^3 + 0.02385615350 \\ 0.0453122860709588 \\ -12873.67954\,t^5 + 28965.77897\,t^4 - 25961.92041\,t^3 + 11586.31159\,t^2 - 2574.735909\,t + 2 \end{cases}$$

```
> opt_s_vel_profile:=subs(diff(opt_s_profile,t));
```

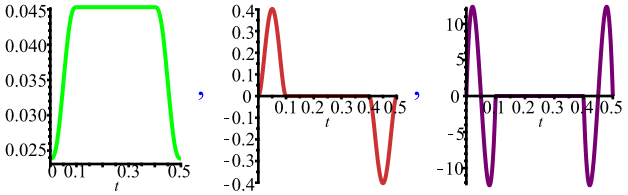
$$opt\_s\_vel\_profile := \begin{cases} 64368.39770\,t^4 - 12873.67954\,t^3 + 643.6839771\,t^2 \\ 0 \\ -64368.39770\,t^4 + 115863.1159\,t^3 - 77885.76123\,t^2 + 23172.62318\,t - 2574.735909 \end{cases}$$

```
> opt_s_acc_profile:=subs(diff(opt_s_vel_profile,t));
```

$$opt\_s\_acc\_profile := \begin{cases} 257473.5908\,t^3 - 38621.03862\,t^2 + 1287.367954\,t & 0 \le t \le \\ 0 & T_{opening} < t \le \\ -257473.5908\,t^3 + 347589.3477\,t^2 - 155771.5225\,t + 23172.62318 & T_{opening} + T_{still} \end{cases}$$

```
> opt_profiles:=subs(opt_data,[
      diff(s(t),t,t)=opt_s_acc_profile,
      diff(s(t),t)=opt_s_vel_profile,
      s(t)=opt_s_profile
  ]):
```

```
> plot(subs(opt_data,opt_s_profile),t=0..T__tot,color=green),
  plot(subs(opt_data,opt_s_vel_profile),t=0..T__tot,color=orange),
  plot(subs(opt_data,opt_s_acc_profile),t=0..T__tot,color=purple)
```



Initial condition of the optimized mechanism

```
> opt_ics_qI      := [s(t)=eval(subs(t=0,opt_data,opt_s_profile))]:
  # it corresponds to opt_s_min
> opt_ics_qD      := evalf(subs(opt_ics_qI,opt_data,kin_sol)):
> opt_ics_pos     := opt_ics_qI union opt_ics_qD;
```

$$opt\_ics\_pos := [s(t) = 0.02385615350,\ \psi 2(t) = -0.3110657143,\ \psi 3(t) = 1.087103992] \quad \textbf{(4.9.5.14)}$$

```
> opt_ics_qI_vel := [diff(s(t),t)=eval(subs(t=0,opt_data,
  opt_s_vel_profile))]:
> opt_ics_qD_vel := evalf(subs(opt_ics_qI_vel,opt_data,vel_kin_sol)
  ):
> opt_ics_vel     := ics_qI_vel union ics_qD_vel;
```

$$opt\_ics\_vel := \left[\frac{d}{dt}\ s(t) = 0.,\ \frac{d}{dt}\ \psi 2(t) = -0.,\ \frac{d}{dt}\ \psi 3(t) = 0.\right] \quad \textbf{(4.9.5.15)}$$

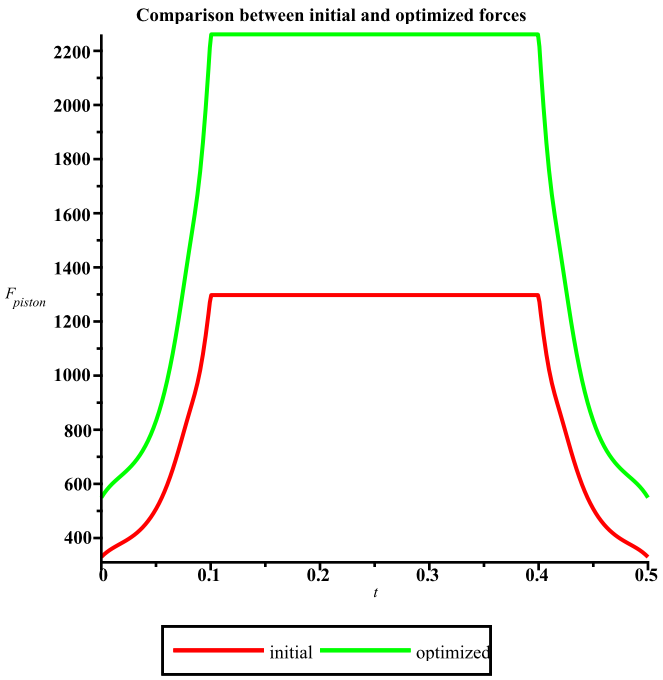
```
> opt_ics_qI_acc:=[diff(s(t),t,t)=eval(subs(t=0,opt_data,
  opt_s_acc_profile))]:
> opt_ics_qD_acc:=[seq(diff(qD[i],t,t)=evalf(rhs(subs(opt_data,
  opt_ics_qI_acc,opt_ics_vel,opt_ics_pos,acc_kin_sol[i]))),i=1..
  nops(qD))]:
> opt_ics_acc:=opt_ics_qI_acc union opt_ics_qD_acc;
```

$$opt\_ics\_acc := \left[\frac{d^2}{dt^2}\ s(t) = 0.,\ \frac{d^2}{dt^2}\ \psi 2(t) = -0.,\ \frac{d^2}{dt^2}\ \psi 3(t) = 0.\right] \quad \textbf{(4.9.5.16)}$$

Optimized dynamic

Inverse dynamics

```
> opt_piston_force_profile := subs(air_forces_law,acc_kin_sol,
  vel_kin_sol,kin_sol,opt_profiles,opt_data,
  piston_force_pvw_dynamic):
> ## plot ## plot(opt_piston_force_profile,t=0..T__tot,color=green,
  title="Optimized piston force");
> display([
          plot(piston_force_profile,t=0..T__tot),
          plot(opt_piston_force_profile,t=0..T__tot)
      ],
      title="Comparison between initial and optimized forces",
      color=[red,green],
      legend=["initial","optimized"],
      labels=[t,F__piston],
      font=[TIMES,BOLD,12]
  );
```



Direct

We need to recompute the dynamic with the new data.
We decided to use newton-euler, but lagrange works the same.

```
> opt_eqns_static := evalf(subs(air_forces_law,opt_ics_acc,
  opt_ics_vel,opt_ics_pos,opt_data,eqns_NE)):
  opt_ics_rvars := op(solve(opt_eqns_static[1..-2],rvars)):
> opt_ics_piston := [F__piston(t)=evalf(subs(t=0,
  opt_piston_force_profile))]:
> opt_ics_dae := subs(opt_ics_piston,t=0,convert(opt_ics_vel union
  opt_ics_pos union opt_ics_rvars,D));
```

$$opt\_ics\_dae := [D(s)(0) = 0., D(\psi 2)(0) = -0., D(\psi 3)(0) = 0., s(0) = 0.02385615350, \quad \textbf{(4.9.6.1)}$$
$$\psi 2(0) = -0.3110657143, \psi 3(0) = 1.087103992, Np(0) = 177.6028391, Tp(0)$$
$$= 14.76785578, rj1x(0) = 549.1722971, rj1y(0) = -176.8180391, rj2x(0)$$
$$= 549.1722971, rj2y(0) = -176.3030141, rj3x(0) = 403.6592035, rj3y(0)$$
$$= 662.4338507]$$

```
> opt_sys := subs(
      air_forces_law,
      F__piston(t)=opt_piston_force_profile,
      opt_data,
      eqns_NE union Phi union opt_ics_dae
  ):
```

System solution

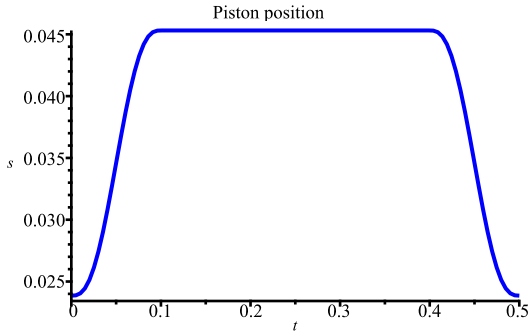
Sometimes the dsolve face a singularity and throws an exception; the following approach avoids interruptions.

```
> st := time():
  solved:=false:
  do
      try
          opt_dsol:=dsolve(opt_sys,numeric,implicit=true,stiff=
  true,output=listprocedure):
```

```
        solved:=true:
    catch:
        print("Exception generated, automatic retrial"):
    end try;
until solved:
"computation time"=time() - st;
```

"computation time" = 13.125 **(4.9.6.2)**

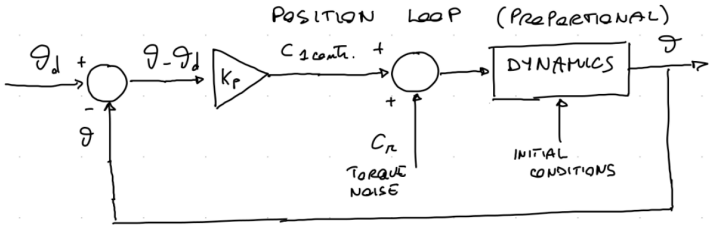
```
> ## plot ##
odeplot(opt_dsol,[t,s(t)],t=0..T__tot,color=blue,title="Piston
position",size=SMS,numpoints=100);
```



Control on the piston force

Calculate the actual profile of the variable s(t) when a controller is applied to the piston force

Position loop

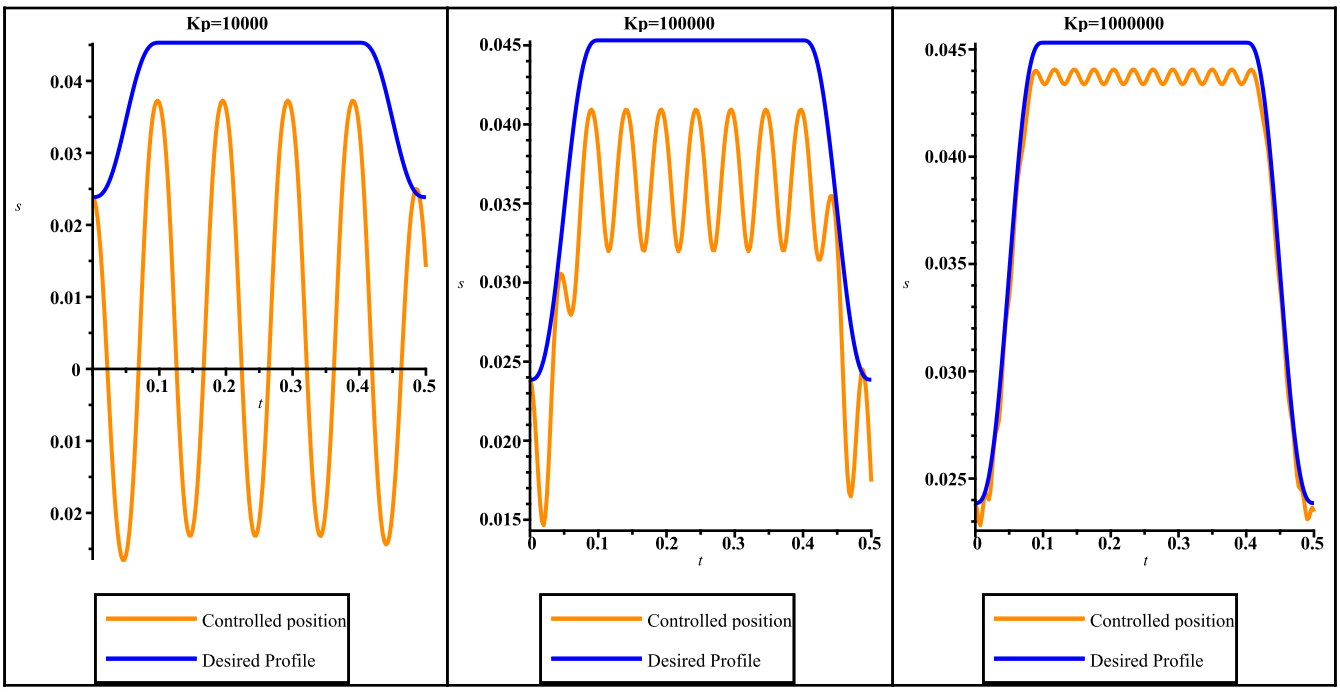



```
> pos_cntr_actions := [
    F__piston(t)=kp*(opt_s_profile-s(t))
  ]:
> pos_cntr_sys := subs(
    air_forces_law,
    pos_cntr_actions,
    opt_data,
    eqns_NE union Phi) union opt_ics_dae:
> st := time():
  solved:=false:
  do
    try
      pos_cntr_dsol := dsolve(pos_cntr_sys,numeric,implicit=
  true,stiff=true,output=listprocedure);
      solved:=true:
    catch:
      print("Exception generated, automatic retrial"):
    end try;
  until solved:
  "computation time"=time() - st;
```

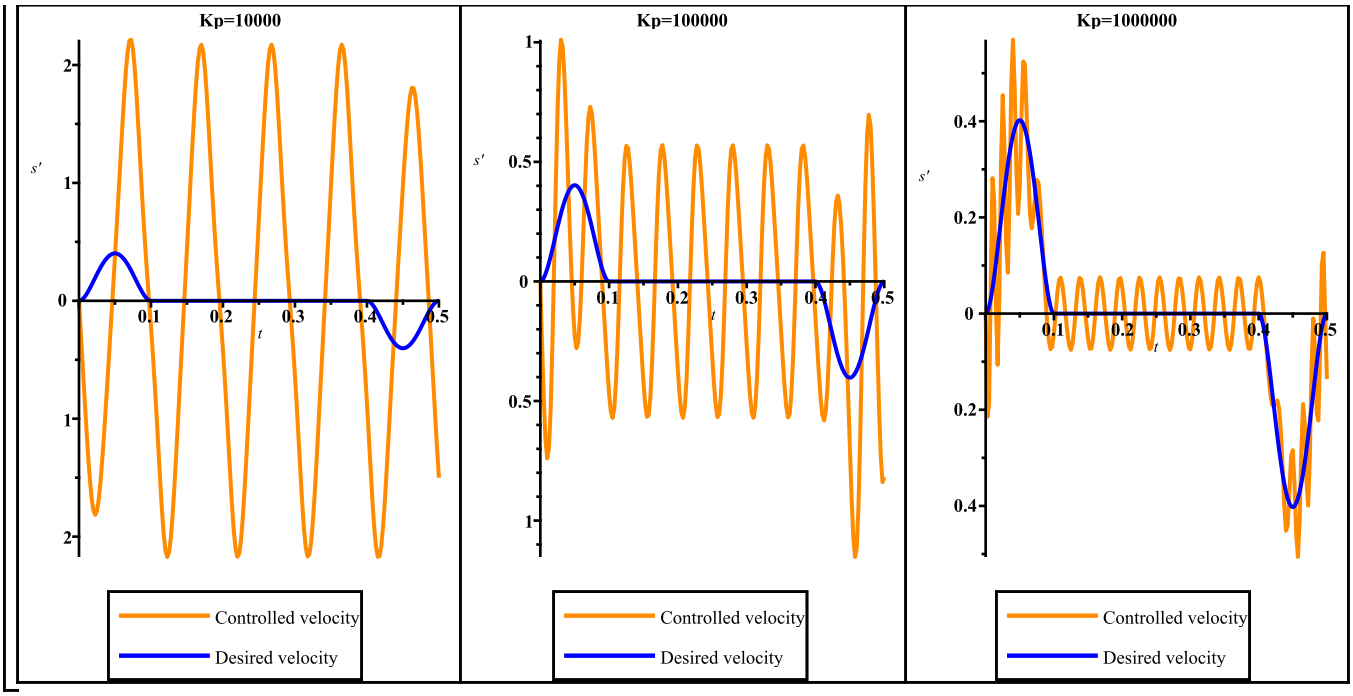
(5.1.1)

"computation time" = 0.766 **(5.1.1)**

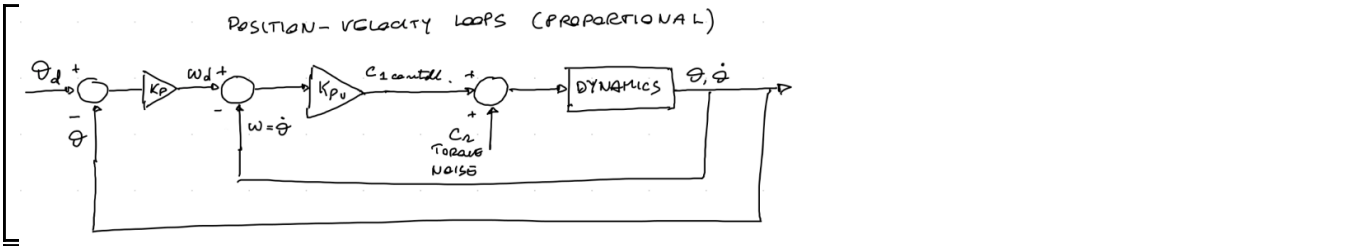
```
> unassign('k'):
> display(Array([
      seq(display([
          odeplot(dsolve(subs(air_forces_law,pos_cntr_actions,kp=k,
  opt_data,eqns_NE union Phi) union
  opt_ics_dae,numeric,implicit=true,stiff=true,output=
  listprocedure,maxfun=0),[t,s(t)],t=0..T__tot),
          plot(subs(opt_data,opt_s_profile),t=0..T__tot)
      ],
      color=["DarkOrange","Blue"],
      title=cat("Kp=",k),
      font=[TIMES,BOLD,12],
      legend=["Controlled position","Desired Profile"]
  ),k=[seq(10^e,e=4..6)])]));
```




```
> display(Array([
  seq(display([
          odeplot(dsolve(subs(air_forces_law,pos_cntr_actions,kp=k,
  opt_data,eqns_NE union Phi) union opt_ics_dae,numeric,implicit=
  true,stiff=true,output=listprocedure,maxfun=0),[t,diff(s(t),t)],
  t=0..T__tot),
          plot(subs(opt_data,opt_s_vel_profile),t=0..T__tot)
      ],
      color=["DarkOrange","Blue"],
      title=cat("Kp=",k),
      legend=["Controlled velocity","Desired velocity"],
      font=[TIMES,BOLD,12]
  ),k=[seq(10^e,e=4..6)])
  ]));
```



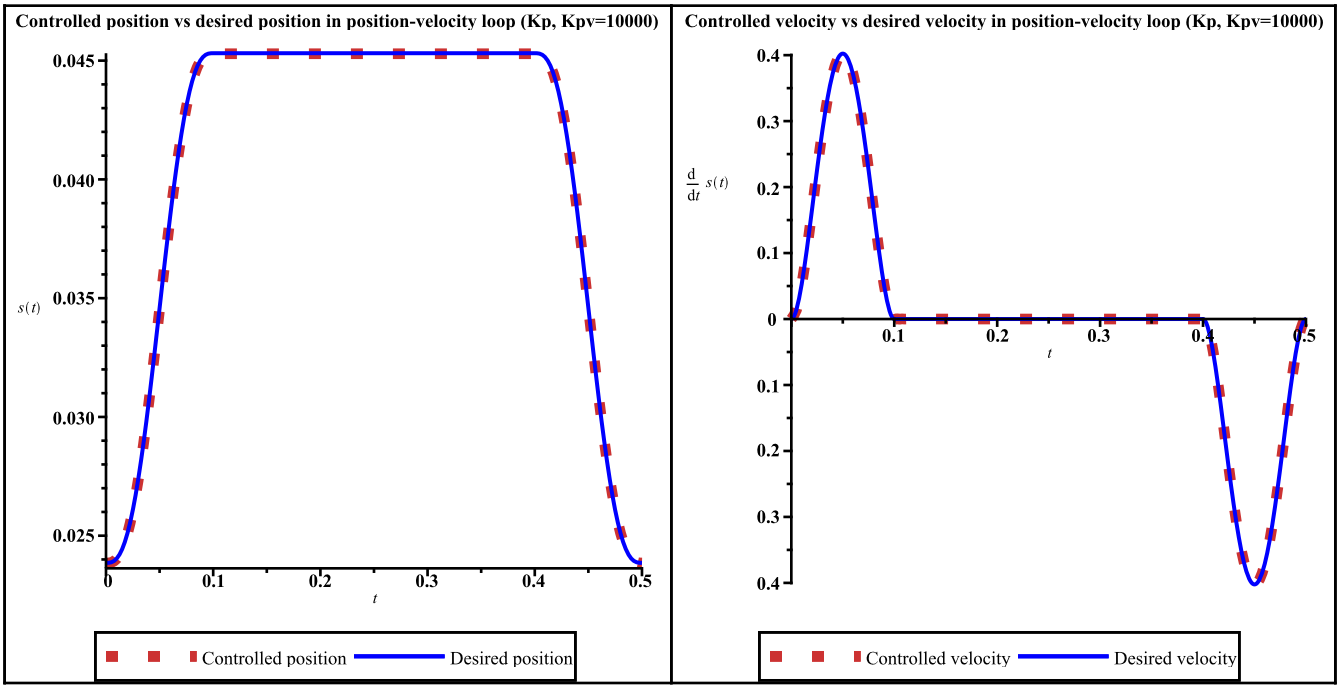
Position-velocity loop




```
> pos_vel_cntr_actions := {
      F__piston(t)=kpv*(kp*(opt_s_profile-s(t))-diff(s(t),t))
  }:
> pos_vel_cntr_sys := subs(
      air_forces_law,
      pos_vel_cntr_actions,
      opt_data,
      eqns_NE union Phi) union opt_ics_dae:
> st := time():
  solved:=false:
  do
      try
          pos_vel_cntr_dsol := dsolve(pos_vel_cntr_sys,numeric,
  implicit=true,stiff=true,output=listprocedure,maxfun=0);
          solved:=true:
      catch:
          print("Exception generated, automatic retrial"):
      end try;
  until solved:
  "computation time"=time() - st;
                    "Exception generated, automatic retrial"
                    "Exception generated, automatic retrial"
                    "Exception generated, automatic retrial"
```

"computation time" = 0.219 **(5.2.1)**

```
> display(Array([
      display([
         odeplot(pos_vel_cntr_dsol,[t,s(t)],t=0..T__tot, color=
  orange,linestyle=dot, thickness=4),
         plot(subs(opt_data,opt_s_profile),t=0..T__tot,color=blue)
  ],
           legend=["Controlled position","Desired position"],
           labels=[t,s(t)],
           title = "Controlled position vs desired position in
  position-velocity loop (Kp, Kpv=10000)",
           font =[TIMES,BOLD,12]
      ),
      display([
         odeplot(pos_vel_cntr_dsol,[t,diff(s(t),t)],t=0..T__tot,
  color=orange,linestyle=dot, thickness=4),
         plot(subs(opt_data,opt_s_vel_profile),t=0..T__tot,color=
  blue)],
           legend=["Controlled velocity","Desired velocity"],
           labels=[t,typeset(diff(s(t),t))],
           title = "Controlled velocity vs desired velocity in
  position-velocity loop (Kp, Kpv=10000)",
           font =[TIMES,BOLD,12]
      )
  ]));
```



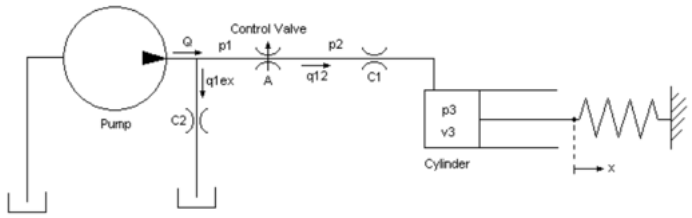
Hydraulic system

System description

Through a control valve, command the right flow to get the p3(t) pressure inside the piston needed to

generate the force calculated in the previous chapters.

The implemented model considers:
- Q(t) pump flow (**known at priori**)
- q1_lex(t) leakage flow
- q__12(t) control valve flow
- C2 leakage coefficient
- p1(t) pump pressure
- p2(t) pressure downstream control valve
- rho fluid density
- Cd orifice disharge coefficient
- A(t) orifice area of the control valve that depends on time (**unknown**)
- Ac cylinder cross section area
- p3(t) piston pressure (**target**, calculated propagating back the inverse dynamic)
- beta fluid bulk modulus
- V3(t) fluid volume at p3(t) (the fluid is compressible)
- q__23(t) flow exiting the control valve (same as the one entering q12(t))
- V30 fluid volume in pistonm as s=s_min



Hydraulic utilitites and data

```
> Sign__reg := x->tanh(x/epsilon):
  Abs__reg  := x->sqrt(x^2+epsilon^2):
> hydr_data:= [
      Cd  = 0.61,
      rho__fluid  = 800, # kg/m^3    - fluid density
      #el__fluid = 8e8,   # Pa        - bulk modulus
      # nu__fluid = 1.8e-5,# m^2/s   - kinematic viscosity
      C1   = 2e-8,        # m^3/s/Pa - hydraulic resistance
      C2   = 3e-9,        # m^3/s/Pa
      Ac   = 1e-3,        # m^2      - cross section of piston
      #A0   = 0.001,      # m^2      - orifice area
      #Ks   = 5e4,        # N/m
      V30  = 0,           # m^3      - initial cylinder volume
      epsilon = 1e-4
  ]:
```

The available flow-rate is determined by looking at the datasheet for the micro-hydraulic valves produced by Moog, the premier supplier for the Formula 1 teams.
https://www.moog.com/markets/motorsport/motorsport-products-for-formula-1.html

```
> flow_eqn:=Q(t)=0.007;
```

$$flow\_eqn := Q(t) = 0.007 \qquad \textbf{(6.2.1)}$$

System modeling

1. Flow rate conservation and pressure drop

```
> block1_eqns := [
      Q(t) = q__12(t)+q1__lex(t),
      q1__lex(t) = C2*p1(t)
  ]:
```

2. Turbolent flow through the control valve depending on the orifice area.
The sign and absolute value functions accomodate flow in either direction.

```
> block2_eqns:=[
      q__12(t)=Cd*A(t)
          *Sign__reg((p1(t)-p2(t)))
          *sqrt(2/rho__fluid*Abs__reg((p1(t)-p2(t))))
  ];
  block2_eqns_new:=[
      q__12(t)=Cd*A(t)
          *tanh(4*sqrt(A(t)*2*abs(p1(t)-p2(t))/(Pi*rho__fluid))
  /nu__fluid)
          *sqrt(2*abs(p1(t)-p2(t))/rho__fluid)*sign(p1(t)-p2(t))
  ];
```

$$block2\_eqns := \left[q_{12}(t) = \sqrt{2} \sqrt{\frac{\sqrt{(p1(t)-p2(t))^2+\epsilon^2}}{\rho_{fluid}}} \tanh\left(\frac{p1(t)-p2(t)}{\epsilon}\right) A(t)\, Cd \right]$$

$$block2\_eqns\_new := \left[q_{12}(t) = \sqrt{2} \sqrt{\frac{|p1(t)-p2(t)|}{\rho_{fluid}}} \tanh\left(\frac{4\sqrt{2} \sqrt{\frac{|p1(t)-p2(t)|\, A(t)}{\rho_{fluid}\, \pi}}}{\nu_{fluid}} \right) A(t)\, Cd \right] \quad \textbf{(6.3.1)}$$

3. Pressure drop before cylinder input

```
> block3_eqns:=[
      q__23(t)=q__12(t),
      q__23(t)=C1*(p2(t)-p3(t))
  ]:
```

4. Fluid flow and fluid volume relation
For this part, in a real case, it would be worth also to consider the fluid compressibility.
In our case we decided to skip it due to the complexity of the derivative of the pression p3(t), solved using the piston force calculated in the previous chapters (F__piston(t)).

```
> block4_eqns:=[
      #diff(p3(t),t)*V3(t) = beta*(q__12(t)-diff(V3(t),t)), # too
  complex to compute
      q__23(t)=diff(V3(t),t),
      V3(t) = V30+Ac*s(t)
  ]:
```

5. Forces balance, including the inertia force of the moving piston.

```
> block5_eqns:=[
      F__piston(t)=p3(t)*Ac
```

```
  ]:
```

Putting everything together

```
> hydr_eqns:=[
      op(block1_eqns),
      op(block2_eqns),
      op(block3_eqns),
      op(block4_eqns),
      op(block5_eqns)
  ]: <%>;
```

$$\left[\begin{array}{c} Q(t) = q_{12}(t) + q1_{lex}(t) \\ q1_{lex}(t) = p1(t)\ C2 \\ q_{12}(t) = \sqrt{2}\ \sqrt{\frac{\sqrt{(p1(t)-p2(t))^2+\epsilon^2}}{\rho_{fluid}}}\ \tanh\left(\frac{p1(t)-p2(t)}{\epsilon}\right) A(t)\ Cd \\ q_{23}(t) = q_{12}(t) \\ q_{23}(t) = (p2(t) - p3(t))\ C1 \\ q_{23}(t) = \frac{\mathrm{d}}{\mathrm{d}t}\ V3(t) \\ V3(t) = V30 + s(t)\ Ac \\ F_{piston}(t) = Ac\ p3(t) \end{array}\right] \quad \textbf{(6.3.2)}$$

Solving the system

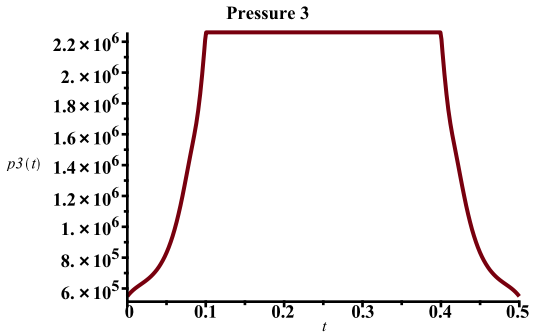
The objective is to solve the system for the control valve orifix area.
Our assumption is to know: the pump flow, the pump pressure, the time laws of s(t) and the requested piston force.
With the simplified system the problem becomes algebraic.

```
> hvars:=[q__12(t) ,q__23(t), q1__lex(t),V3(t),p1(t),p2(t),p3(t),s
  (t)]:
  nops(hydr_eqns)=nops(hvars);
```

$$8 = 8 \quad \textbf{(6.4.1)}$$

```
> pressure3_eqn:=op(solve(hydr_eqns[-1],[p3(t)]))[1];
  ## plot ##
  plot(subs(F__piston(t)=opt_piston_force_profile,profiles,
  opt_data,hydr_data,rhs(pressure3_eqn)),t=0..T__tot,title=
  "Pressure 3",size=SMS,font=[TIMES,BOLD,12], labels=[t, p3(t)]);
```

$$pressure3\_eqn := p3(t) = \frac{F_{piston}(t)}{Ac}$$




```
> flow12_eqn:=eval(subs(V3(t)=rhs(hydr_eqns[-2]),hydr_eqns[4],
  hydr_eqns[-3]));
  ## plot ## plot(subs(profiles,opt_data,hydr_data,rhs(flow12_eqn)
  ),t=0..T__tot,title="Flow q__12(t)",size=[400,300]);
```

$$flow12\_eqn := q_{12}(t) = \left(\frac{\mathrm{d}}{\mathrm{d}t}\, s(t) \right) Ac \tag{6.4.2}$$

```
> pressure2_eqn:=op(solve(subs(flow12_eqn, pressure3_eqn,hydr_eqns
  [4],hydr_eqns[-4]), [p2(t)]))[1];
  ## plot ## plot(subs(flow12_eqn,F__piston(t)=
  opt_piston_force_profile,profiles,hydr_data,rhs(pressure2_eqn)),
  t=0..T__tot,title="Pressure p2(t)",size=[400,300]);
```

$$pressure2\_eqn := p2(t) = \frac{q_{12}(t)\, Ac + F_{piston}(t)\, C1}{Ac\, C1} \tag{6.4.3}$$

```
> pressure1_eqn:=subs(
      op(solve(hydr_eqns[1],[q1__lex(t)])),
      op(solve(hydr_eqns[2],[p1(t)])[1])
  );
  ## plot ## plot(subs(Q(t)=0.02,flow12_eqn,F__piston(t)=
  opt_piston_force_profile,profiles,hydr_data,rhs(pressure1_eqn)),
  t=0..T__tot,size=[400,300],title="Pressure p1(t)");
```

$$pressure1\_eqn := p1(t) = \frac{-q_{12}(t) + Q(t)}{C2} \tag{6.4.4}$$

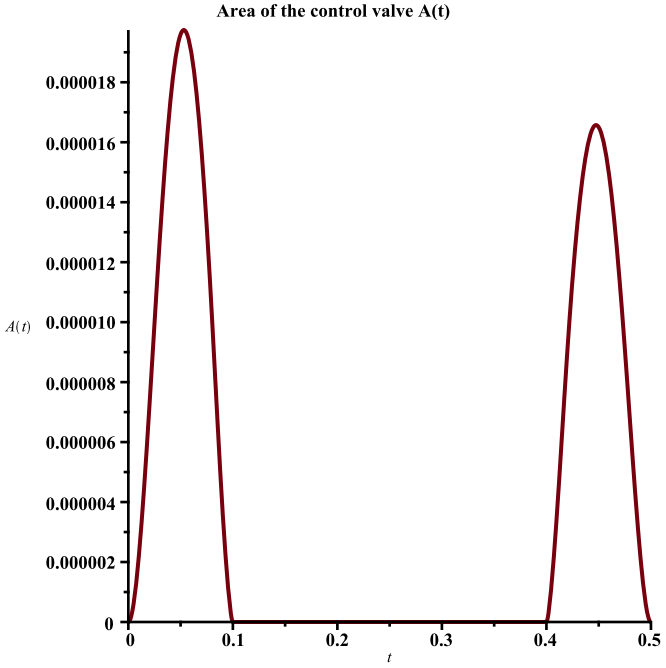
```
> hydr_eqns_red:=subs(pressure1_eqn,pressure2_eqn,pressure3_eqn,
  flow12_eqn,hydr_eqns[3]);
```

$$hydr\_eqns\_red := \left(\frac{\mathrm{d}}{\mathrm{d}t}\, s(t) \right) Ac \tag{6.4.5}$$

$$= \sqrt{2} \sqrt{\frac{\sqrt{\left(\frac{-\left(\frac{\mathrm{d}}{\mathrm{d}t}\, s(t) \right) Ac + Q(t)}{C2} - \frac{\left(\frac{\mathrm{d}}{\mathrm{d}t}\, s(t) \right) Ac^2 + F_{piston}(t)\, C1}{Ac\, C1} \right)^2 + \epsilon^2}}{\rho_{fluid}}} \tanh\left(\frac{\frac{-\left(\frac{\mathrm{d}}{\mathrm{d}t}\, s(t) \right) Ac + Q(t)}{C2} - \frac{\left(\frac{\mathrm{d}}{\mathrm{d}t}\, s(t) \right) Ac^2 + F_{piston}(t)\, C1}{Ac\, C1}}{\epsilon} \right) A(t)\, Cd$$

```
> orifix_area:=simplify(combine(rhs(op(solve(hydr_eqns_red,[A(t)]))
  [1]))):
> orifix_area_raw_profile:=evalf(subs(flow_eqn,F__piston(t)=
```

```
  opt_piston_force_profile,profiles,opt_data,hydr_data,orifix_area)
  ):
> # it works as the abs command because this one never ends
  computation
  orifix_area_profile := piecewise(
      t>=0 and t<=T__tot/2, orifix_area_raw_profile,
      t>T__tot/2 and t<T__tot, -orifix_area_raw_profile
  ):
> ## plot ##
  plot(orifix_area_profile,t=0..T__tot,title="Area of the control
  valve A(t)", font=[TIMES,BOLD,12], labels=[t,A(t)]);
```



DAE-toolbox

Documentation: https://ebertolazzi.github.io/course-ModellingAndSimulationOfMechatronicsSystem/

```
> read("DAE-toolbox.maplet"):
```

NE equations: semiexplicit form and index reduction

Used function: *first_order() - DAE_reduce_index-by-1*

```
> VARS_NE,DAE_NE := first_order(eqns_NE union Phi,qvars union
  rvars,t,flag=true):
> DVARS_NE := diff(VARS_NE,t):
> VARS_NE;
```

$$\left[s(t),\ \psi 2(t),\ \psi 3(t),\ Np(t),\ Tp(t),\ rj1x(t),\ rj1y(t),\ rj2x(t),\ rj2y(t),\ rj3x(t),\ rj3y(t),\ s_{dot}(t),\ \psi 2_{dot}(t),\ \psi 3_{dot}(t)\right] \quad (7.1.1)$$

```
> nops(DAE_NE) = nops(VARS_NE);
```

$$14 = 14 \quad (7.1.2)$$

Semiexplicit form

```
> E_NE,G_NE,A_NE,r_NE := DAE_separate_algebraic_bis(DAE_NE,
  DVARS_NE):
```

```
> r_NE:
```

Index reduction

First reduction

```
> E1_NE,G1_NE,A1_NE,r1_NE := DAE_reduce_index_by_1(E_NE,G_NE,A_NE,
  DVARS_NE):
  r1_NE;
```

$$12 \tag{7.1.3}$$

Second reduction

```
> E2_NE,G2_NE,A2_NE,r2_NE := DAE_reduce_index_by_1(E1_NE,G1_NE,
  A1_NE,DVARS_NE):
  r2_NE;
```

$$12 \tag{7.1.4}$$

Third reduction

```
> ## too complex ## E3_NE,G3_NE,A3_NE,r3_NE :=
  DAE_reduce_index_by_1(E2_NE,G2_NE,A2_NE,DVARS_NE):
  r3_NE;
```

$$r3\_NE \tag{7.1.5}$$

The algebraic part is empty, we can stop the reduction.

Since we performed three reduction before getting this result we can say that **the original DAE is index-3**

```
> INVARIANTS_NE := <A_NE,A1_NE,A2_NE>:
```

Lagrange equations

Used functions: *DAE3_get_ODE_and_invariants - DAE3_get_ORD_and_invariants_baumgarte*

To use the mentioned functions the DAE has to be the specific shape illustrated by the library.

```
> vvars := [u(t),v(t),w(t)];
```

$$vvars := [u(t), v(t), w(t)] \tag{7.2.1}$$

```
> VARS_lagr := qvars union vvars union lvars;
```

$$VARS\_lagr := [s(t), \psi 2(t), \psi 3(t), u(t), v(t), w(t), \lambda 1(t), \lambda 2(t)] \tag{7.2.2}$$

```
> REMOVE_T := map(x->x=op(0,x),VARS_lagr);
```

$$REMOVE\_T := [s(t) = s, \psi 2(t) = \psi 2, \psi 3(t) = \psi 3, u(t) = u, v(t) = v, w(t) = w, \lambda 1(t) = \lambda 1, \lambda 2(t) = \lambda 2] \tag{7.2.3}$$

```
> RENAME_QDOT := [seq(diff(qvars[i],t)=vvars[i],i=1..nops(qvars))];
```

$$RENAME\_QDOT := \left[\frac{d}{dt} s(t) = u(t), \frac{d}{dt} \psi 2(t) = v(t), \frac{d}{dt} \psi 3(t) = w(t)\right] \tag{7.2.4}$$

```
> DAE_lagr := RENAME_QDOT union subs(RENAME_QDOT,air_forces_law,
  F__piston(t)=kp*(desired_profile-s(t)),opt_data,eqns_lagr);
```

$$DAE\_lagr := \left[\frac{d}{dt} s(t) = u(t), \frac{d}{dt} \psi 2(t) = v(t), \frac{d}{dt} \psi 3(t) = w(t), 0.1325000000 \frac{d}{dt} u(t) + 0.001837500000 \left(\frac{d}{dt} v(t)\right) \sin(\psi 2(t)) + 0.001837500000\, v(t)^2 \cos(\psi 2(t)) + \lambda 1(t) - 10000\, desired\_profile + 10000\, s(t), 0.00007962500000 \frac{d}{dt} v(t)\right. \tag{7.2.5}$$

$$+ 0.001837500000 \sin(\psi 2(t)) \left(\frac{d}{dt} u(t)\right) + 0.03500000000 (0.5150250000$$

$$- 2 \lambda 2(t)) \cos(\psi 2(t)) + 0.07000000000 \sin(\psi 2(t)) \lambda 1(t), 0.01639736736 \frac{d}{dt} w(t)$$

$$- (0.01082898000 \lambda 1(t) + 0.05700000000 \lambda 2(t) + 1.143386965) \cos(\psi 3(t))$$

$$- (0.03115281851 - 0.05700000000 \lambda 1(t) + 0.01082898000 \lambda 2(t)) \sin(\psi 3(t))$$

$$- (39.15728639 + 7.678212536 \psi 3(t)) \cos(\psi 3(t)) - (-1.539454488$$

$$+ 10.41304681 \psi 3(t)) \sin(\psi 3(t)), -0.01082898000 \sin(\psi 3(t))$$

$$- 0.07000000000 \cos(\psi 2(t)) - 0.05700000000 \cos(\psi 3(t)) + 0.078879 + s(t),$$

$$0.01082898000 \cos(\psi 3(t)) - 0.07000000000 \sin(\psi 2(t)) - 0.05700000000 \sin(\psi 3(t))$$

$$+ 0.024000]$$

```
> DVARS_lagr := diff(VARS_lagr,t);
```

$$DVARS\_lagr := \left[\frac{d}{dt} s(t), \frac{d}{dt} \psi 2(t), \frac{d}{dt} \psi 3(t), \frac{d}{dt} u(t), \frac{d}{dt} v(t), \frac{d}{dt} w(t), \frac{d}{dt} \lambda 1(t), \frac{d}{dt} \lambda 2(t)\right] \quad (7.2.6)$$

```
> E_lagr,G_lagr,A_lagr,r_lagr := DAE_separate_algebraic_bis
  (DAE_lagr,DVARS_lagr):
  r_lagr;
```

$$6 \quad (7.2.7)$$

```
> E_lagr:
> Mass := E_lagr[4..6,4..6]:
> gfun := G_lagr[4..6]:
> res_lagr := DAE3_get_ODE_and_invariants_baumgarte( Mass, Phi,
  gfun, qvars, vvars, lvars )
```

$$res\_lagr := tbl \quad (7.2.8)$$

```
> eval(res_lagr):
> HIDDEN_STAB_lagr := res_lagr["h"];
```

$$HIDDEN\_STAB\_lagr := [[-w(t)^2 \sin(\psi 3(t)) H3 - L2 v(t)^2 \cos(\psi 2(t)) \quad (7.2.9)$$

$$- w(t)^2 \cos(\psi 3(t)) L3 - 2 \omega \eta (-w(t) \cos(\psi 3(t)) H3 + L2 v(t) \sin(\psi 2(t))$$

$$+ w(t) \sin(\psi 3(t)) L3 + u(t)) - \omega^2 (-\sin(\psi 3(t)) H3 - L2 \cos(\psi 2(t))$$

$$- \cos(\psi 3(t)) L3 - L1 + s(t) - xA + xD)],$$

$$[w(t)^2 \cos(\psi 3(t)) H3 - L2 v(t)^2 \sin(\psi 2(t)) - w(t)^2 \sin(\psi 3(t)) L3 - 2 \omega \eta ($$

$$-w(t) \sin(\psi 3(t)) H3 - L2 v(t) \cos(\psi 2(t)) - w(t) \cos(\psi 3(t)) L3)$$

$$- \omega^2 (\cos(\psi 3(t)) H3 - L2 \sin(\psi 2(t)) - \sin(\psi 3(t)) L3 - yA + yD)]]$$

```
> JHIDDEN_STAB_lagr := map(simplify,JACOBIAN(HIDDEN_STAB_lagr,
  VARS_lagr));
```

$$JHIDDEN\_STAB\_lagr := \left[\left[-\omega^2,\ \left(-L2\,\omega^2 + L2\,v(t)^2\right)\sin\left(\psi2(t)\right)\right.\right. \tag{7.2.10}$$
$$-2\,\omega\,\eta\,L2\,v(t)\cos\left(\psi2(t)\right),\ \left(-2\,H3\,\eta\,\omega\,w(t) - L3\,\omega^2 + L3\,w(t)^2\right)\sin\left(\psi3(t)\right) + \left($$
$$-2\,L3\,\eta\,\omega\,w(t) + H3\,\omega^2 - H3\,w(t)^2\right)\cos\left(\psi3(t)\right),\ -2\,\omega\,\eta,\ -2\,L2\,\left(\sin\left(\psi2(t)\right)\,\eta\,\omega$$
$$\left.+\,v(t)\cos\left(\psi2(t)\right)\right),\ \left(-2\,L3\,\eta\,\omega - 2\,H3\,w(t)\right)\sin\left(\psi3(t)\right) + 2\cos\left(\psi3(t)\right)\,\left(H3\,\eta\,\omega\right.$$
$$\left.-\,L3\,w(t)\right),\ 0,\ 0\left.\right],$$
$$\left[0,\ \left(L2\,\omega^2 - L2\,v(t)^2\right)\cos\left(\psi2(t)\right) - 2\,\omega\,\eta\,L2\,v(t)\sin\left(\psi2(t)\right),\ \left(2\,H3\,\eta\,\omega\,w(t)\right.\right.$$
$$+\,L3\,\omega^2 - L3\,w(t)^2\left.\right)\cos\left(\psi3(t)\right) + \sin\left(\psi3(t)\right)\,\left(-2\,L3\,\eta\,\omega\,w(t) + H3\,\omega^2\right.$$
$$\left.-\,H3\,w(t)^2\right),\ 0,\ 2\,L2\,\left(\cos\left(\psi2(t)\right)\,\eta\,\omega - v(t)\sin\left(\psi2(t)\right)\right),\ \left(2\,H3\,\eta\,\omega\right.$$
$$\left.\left.-\,2\,L3\,w(t)\right)\sin\left(\psi3(t)\right) + 2\cos\left(\psi3(t)\right)\,\left(L3\,\eta\,\omega + H3\,w(t)\right),\ 0,\ 0\right]\right]$$

```
> PARTIAL_ODE := [seq(DVARS_lagr[i]=LinearSolve(E_lagr,G_lagr)[i],
  i=1..nops(VARS_lagr))][1..6]:
> res_lagr["g"];
```

$$\begin{bmatrix} -w(t)^2\sin\left(\psi3(t)\right)\,H3 - L2\,v(t)^2\cos\left(\psi2(t)\right) - w(t)^2\cos\left(\psi3(t)\right)\,L3 \\ w(t)^2\cos\left(\psi3(t)\right)\,H3 - L2\,v(t)^2\sin\left(\psi2(t)\right) - w(t)^2\sin\left(\psi3(t)\right)\,L3 \end{bmatrix} \tag{7.2.11}$$

```
> LAMBDA1 := lambda1(t)=simplify(solve(subs(PARTIAL_ODE,opt_data,
  diff(res_lagr["g"][1],t)),lambda1(t))):
> LAMBDA2 := lambda2(t)=simplify(solve(subs(PARTIAL_ODE,opt_data,
  diff(res_lagr["g"][2],t)),lambda2(t))):
> DLAMBDA1 := diff(lambda1(t),t)=solve(subs(PARTIAL_ODE,diff(subs
  (LAMBDA2,rhs(LAMBDA1)),t)),diff(lambda1(t),t)):
> DLAMBDA2 := diff(lambda2(t),t)=solve(subs(PARTIAL_ODE,DLAMBDA1,
  diff(subs(LAMBDA1,rhs(LAMBDA2)),t)),diff(lambda2(t),t)):
> ODE := [op(PARTIAL_ODE),DLAMBDA1,DLAMBDA2]:
> RHS := map(x->rhs(x),ODE):
f
> f_file := FileTools[Text][Open]("f.txt", create, overwrite):
> FileTools[Text][WriteString](f_file, F_TO_MATLAB( <subs(REMOVE_T,
  RHS)>, subs(REMOVE_T,VARS_lagr), "f")): # f(t,x)
> FileTools[Text][Close](f_file):
jf
> jf_file := FileTools[Text][Open]("jf.txt", create, overwrite):
> FileTools[Text][WriteString](jf_file, JF_TO_MATLAB( subs
  (REMOVE_T,RHS), subs(REMOVE_T,VARS_lagr), "DfDx")): # Df(t,x)/Dx
> FileTools[Text][Close](jf_file):
jft
> jft_file := FileTools[Text][Open]("jft.txt", create, overwrite):
> FileTools[Text][WriteString](jft_file, JF_TO_MATLAB( JACOBIAN
  (subs(REMOVE_T, RHS), [t]), [t], "DfDt")): # # Df(t,x)/Dt
> FileTools[Text][Close](jft_file):
Others...
> F_TO_MATLAB( subs(REMOVE_T,HIDDEN_STAB_lagr), subs(REMOVE_T,
  VARS_lagr), "h"); # h(t,x)
```

```
> JF_TO_MATLAB( subs(REMOVE_T,JA), subs(REMOVE_T,[op(VARS),op
  (LAMBDA_VARS)]), "DhDx"); # Dh(t,x)/Dx
> JF_TO_MATLAB( JACOBIAN(subs(REMOVE_T, HIDDEN_STAB_lagr), [t]),
  [t], "DhDt"); # Dh(t,x)/Dt
```

Consistent initial condition

We can use the same we used here in Maple.
The only difference is that now the valocity variables have their own name.

```
> opt_ics_dae_lagr:=subs(ics_piston_lagr,t=0,convert( opt_ics_pos
  union opt_ics_vel union ics_lvars,D)):
> ics_dae_matlab := seq(op(0,VARS_lagr[i])=rhs(opt_ics_dae_lagr[i]
  ),i=1..nops(VARS_lagr));
```