1. Rigid body formulation using carden angle (mentioned in the paper), show pro/con versus using quaternion

2. Show the formulation of body-joint and how it used to model the "Youbot robot"

3. Study structure of Modia 3D based on ModiaLang and ModiaBase

4. Study the solution sequence outlined in the paper (previous) about the index reduction of DAE

5. Study the solution sequence of Modeling Toolkit

1.Rigid body formulation using carden angle

- Euler angles are useful to represent an aribitrary rotation in R^3

$$R(\alpha,\beta,\gamma\,) = Rot(\hat{z}, \alpha)Rot(\hat{y}, \beta)Rot(\hat{x}, \gamma)$$

$$Rot(\hat{z}, \alpha) = \begin{bmatrix} cos\alpha & -sin\alpha & 0 \\ sin\alpha & cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$Rot(\hat{y}, \beta) = \begin{bmatrix} cos\beta & 0 & sin\beta \\ 0 & 1 & 0 \\ -sin\beta & 0 & cos\beta \end{bmatrix}$$

$$Rot(\hat{x}, \gamma) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\gamma & -sin\gamma \\ 0 & sin\gamma & cos\gamma \end{bmatrix}$$

$$Rot(\alpha, \beta, \gamma) = \begin{bmatrix} cos\alpha \cdot cos\beta & cos\alpha \cdot sin\beta \cdot sin\gamma - sin\alpha \cdot cos\gamma & cos\alpha \cdot sin\beta \cdot cos\gamma + sin\alpha \cdot sin\gamma \\ sin\alpha \cdot cos\beta & sin\alpha \cdot sin\beta \cdot sin\gamma + cos\alpha \cdot cos\gamma & sin\alpha \cdot sin\beta \cdot cos\gamma - cos\alpha \cdot sin\gamma \\ -sin\beta & cos\beta \cdot sin\gamma & cos\beta \cdot cos\gamma \end{bmatrix}$$

- Quaternion
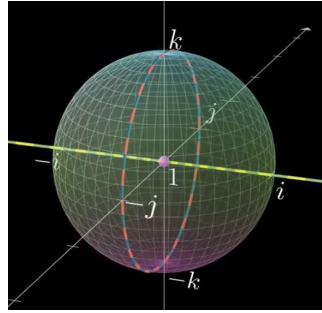$$q = a + bi + cj + dk = [s, v](s - real, v - imaginary)$$

$$i^2 = j^2 = k^2 = ijk = -1$$

$\times$	1	i	j	k
1	1	i	j	k
i	i	-1	k	$-j$
j	j	$-k$	-1	i
k	k	j	$-i$	-1

$$q = a + bi + cj + dk = q_0 + q_1 i + q_2 j + q_3 k$$

$$v' = R \cdot v$$

$$R = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1 q_2 - q_0 q_3) & 2(q_0 q_2 + q_1 q_3) \\ 2(q_0 q_3 + q_1 q_2) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2 q_3 - q_0 q_1) \\ 2(q_1 q_3 + q_0 q_2) & 2(q_0 q_1 + q_2 q_3) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix}$$

quaternion is the projection from 4 dimensional

If s is positive, then the coordinate is inside the sphere while if s is negative, the coordinate is outside the sphere

For unit vector in cartension system we can set s = 0, and v as the coordinate. eg: v for x-axis would be 1,0,0 so q for x-axis = [0,1,0,0]

Rotating a vector v along the axis u, rotating angle θ, we can get
$v' = q \cdot v \cdot q^* = [0, cos(\theta)v + (1 - cos(\theta)(u \cdot v)u + sin(\theta)(u \times v))]$ that will give the same result as the matrix shown above.

- **Pro and con for Tait-Byran angles(Euler angle):**

  Pro:
  1. State variables are illustrative for the user (intuitive)
  2. Tait-Byran angles do not introduce nonlinear algebraic constraints and are directly defined in ODE form. (If the main rotation is approximately around one axis Tait-Byran angles behave nearly linear, so that integrators with adaptive step size selection can use larger steps compared to a description with quaternions.)

  Con: Gimbal lock (When the second rotation about the local y-axis is $\alpha_2 = 90° + n \cdot 180°$ where $n \in Z$, x- and z-axes are parallel.) The default hierachy order is x-y-z, which means when x rotates, y and z axis are rotating with x; when y rotats, z rotating with y; when z rotates, only it self rotating
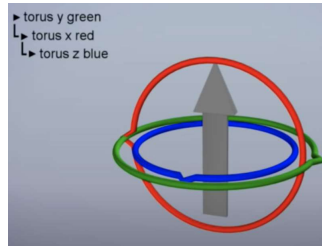
Solution: In Modia3D this situation is avoided by adaptive rotation sequence handling (change hierarchy of axis when we are close to gimbal lock situation, eg, x-y-z to y-x-z)

# 2. Formulation of body-joint and how it used to model the "Youbot robot"

- Idea from Modia3D "YoubotsGripping" https://github.com/ModiaSim/Modia3D.jl/blob/main/test/Robot/YouBotsGripping.jl (https://github.com/ModiaSim/Modia3D.jl/blob/main/test/Robot/YouBotsGripping.jl)

For $\omega = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix}$, Bracket ω: $[\omega] = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix}$.

Screw axes $S_i = (\omega_i, v_i)$. Furthermore, bracket of the screw axes $[S_i] = \begin{bmatrix} [\omega_i] & v_i \\ 0 & 0 \end{bmatrix}$

Trasformation matrix of end-effector: $T = e^{[s_1]\theta_1} e^{[s_2]\theta_2} e^{[s_3]\theta_3} \ldots \cdot M$, where M is the stationary position.

Example: Youbot(dummy parameters from "module YouBotsGripping")
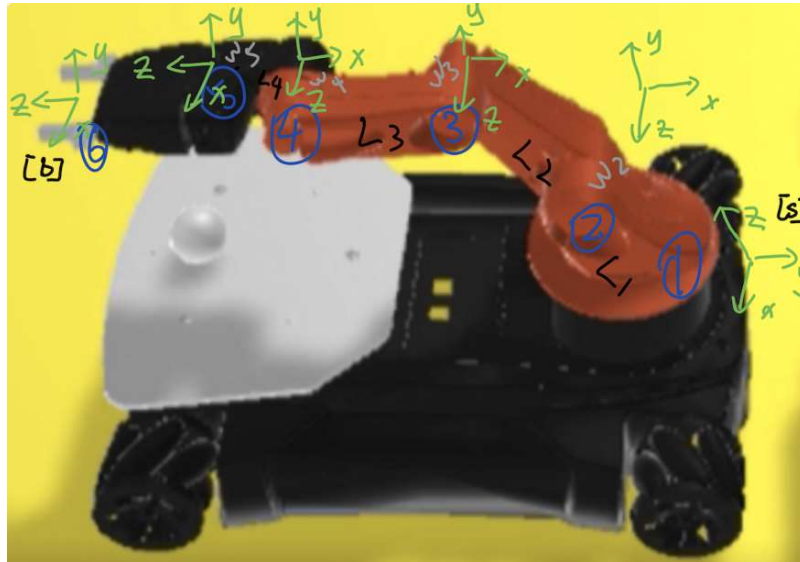
$$M = \begin{bmatrix} 1 & 0 & 0 & -W_2 + W_3 - W_4 + W_5 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & L_1 + L_2 + L_3 + L_4 + L_5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The screw axes for Youbot $S_i = (\omega_i, v_i), i = 1, \ldots .6$ are

i	w_i	v_i
1	(0, 0, 1)	(0, 0, 0)
2	(1, 0, 0)	(0, L_1, 0)
3	(1, 0, 0)	(0, L_1+L2, 0)
4	(1, 0, 0)	(0, L_1+L2+L3, 0)
5	(0, 0, 1)	(0, W2-W3+W4-W5, 0)
6	(0, 0, 0)	(1, 0, 0)

EndEffector Position: Multiply the homogeneous matrices of each frame together (current frame transformations) to get the transformation T_{sb} base to end effector.

Position of the end-effector wrb to [S] is then given by the 4th column [x , y , z , 1].

```julia
# A = [1 2;3 4]
# exp(A)
# e^A #\euler + Tab


# function to compute [S]
function bracket(omega, v)
    return [0 -omega[3] omega[2] v[1]; omega[3] 0 -omega[1] v[2];-omega[2] omega[1] 0 v[3]; 0 0 0 0]
#     return [0 -omega[3] omega(2) v(1); omega(3) 0 -omega(1) v(2);-omega(2) omega(1) 0 v(3); 0 0 0 0]
end
# a = [1 2 3];b= [5 0 0]
# bracket(a,b)

L1=1;L2=2;L3=2;L4=1;L5=1.5
W2=1;W3=1;W4=1;W5=0.5
omega1=[0 0 1];omega2=[1 0 0];omega3=[1 0 0];omega4=[1 0 0];omega5=[0 0 1];omega6=[0 0 0]
v1=[0 0 0];v2=[0 L1 0];v3=[0 L1+L2 0];v4=[0 L1+L2+L3 0];v5=[0 W2-W3+W4-W5 0];v6=[1 0 0]
BrS1=bracket(omega1,v1)
BrS2=bracket(omega2,v2)
BrS3=bracket(omega3,v3)
BrS4=bracket(omega4,v4)
BrS5=bracket(omega5,v5)
BrS6=bracket(omega6,v6)
M=[1 0 0 -W2+W3-W4+W5; 0 0 -1 0; 0 1 0 L1+L2+L3+L4+L5; 0 0 0 1]
Theta=[pi -pi/4 -pi/4 -1.057 0 0]
T = e^(BrS1*Theta[1])*e^(BrS2*Theta[2])*e^(BrS3*Theta[3])*e^(BrS4*Theta[4])*e^(BrS5*Theta[5])*e^(BrS6*Theta[6]
Position = T[1:3,4]
Position #End-effectpr position with respect to frame [s]
```

3-element Vector{Float64}:
  0.4999999999999987
 -4.64293092352121
  0.23700120561659155

3.Study structure of Modia 3d based on ModiaLang and ModiaBase

ModiaLang

- **ModiaLang**: ModiaLang is a minimalistic environment in form of a Julia package to model and simulate physical systems (electrical, mechanical, thermo-dynamical, etc.) described by differential and algebraic equations. A model component is constructed by "expression = expression" equations.https://juliahub.com/docs/ModiaLang/R9zzo/0.7.2/autodocs/

ModiaLang.simulate!—Method

```julia
import ModiaLang
ModiaLang.simulate!(model [, algorithm]; merge = nothing,
        tolerance = 1e-6, startTime = 0.0, stopTime = 1.0, interval = NaN,
        adaptive = true, log = true, logParameters = true, logStates = true,
        requiredFinalStates = nothing)
```

Instantiate model:Needs to be done before simulate the model, basically just opening one part of memory space for that model
firstOrder = @instantiateModel(FirstOrder2, logCode=true)

Map: Setting and modifying parameters of components and attributes of variables (Modify parameters and initial values of model), keep the model but change some parameters inside the model(change or modify the component already in the system)
FirstOrder2 = FirstOrder | Map(T = 0.4, x = Var(init=0.6))

Merge: Define parameters and init/start values that shall be merged with the previous values stored in model, before simulation is started. The used **merge operator |** is an overloaded binary operator of bitwise or with recursive merge semantics. eg: dict1 = {'a': 10, 'b': 5, 'c': 3}; dict2 = {'d': 6, 'c': 4, 'b': 8}; dict3 = dict1 | dict2; dict3 = {'a': 10, 'd': 6, 'c': 4, 'b': 8} (replacing all same elements to dict2)

Der: derivative

In [58]:

```
#example of ModiaLang.simulate! and map
using Modia
@usingModiaPlot #ENV["MODIA_PLOT"] = "PyPlot"  # Use plot package defined with
Modia.usePlotPackage("PyPlot")              # ENV["MODIA_PLOT"] or Modia.usePlotPackage(..)

Pendulum = Model(
    L = 0.8u"m",
    m = 1.0u"kg",
    d = 0.5u"N*m*s/rad",
    g = 9.81u"m/s^2",
    phi = Var(init = 1.57*u"rad"),
    w   = Var(init = 0u"rad/s"),
    equations = :[
            w = der(phi)
            0.0 = m*L^2*der(w) + d*w + m*g*L*sin(phi)
            r = [L*cos(phi), -L*sin(phi)]
    ]
)
Pendulum = Pendulum | Map(L = 0.5u"m")
pendulum1 = @instantiateModel(Pendulum)
simulate!(pendulum1, Tsit5(), stopTime = 10.0u"s", log=true)
# ENV["MODIA_PLOT"] = "PyPlot"
# ModiaResult.activate("PyPlot")
ModiaPlot_PyPlot.plot(pendulum1, [("phi", "w"); "r"], figure = 1)
```

```
using ModiaPlot_PyPlot

Instantiating model Main.Pendulum
... Simulate model Pendulum
      Initialization at time = 0.0 s
      Termination of Pendulum at time = 10.0 s
        cpuTime        = 0.212 s
        allocated      = 4.14 MiB
        algorithm      = Tsit5
        FloatType      = Float64
        interval       = 0.02 s
        tolerance      = 1.0e-6 (relative tolerance)
        nStates        = 2
        nResults       = 501
        nGetDerivatives = 1416 (total number of getDerivatives! calls)
        nf             = 913 (number of getDerivatives! calls from integrator)
        nZeroCrossings = 0 (number of getDerivatives! calls for zero crossing detection)
        nJac           = 0 (number of Jacobian computations)
        nAcceptedSteps = 152
        nRejectedSteps = 0
        nErrTestFails  = 0
        nTimeEvents    = 0
        nStateEvents   = 0
        nRestartEvents = 0
```

- **Reference :( ).** If the value has references to other declared variables in the model, the expressions needs to be quoted that is enclosed in :( ).
- **Parent** refers to the reference system of the object
- A **RevoluteWithFlange** joint, that is a revolute joint with a flange, constrains the motion of axle with respect to world so that axle can only rotate around its z-axis. That rotates **obj1::Object3D** into **obj2::Object3D** along the axis axis of **obj1** (From my understanding, obj2 can only rotating along the axis of obj1(default axis is z))
- **Prismatic joint** joint = Prismatic(; obj1, obj2, path="", axis=1, s=0, v=0, canCollide=true) :translates obj2::Object3D with respect to obj1::Object3D along coordinate axis
- **Connections:** connect(connect-reference-1,connect-reference-2, ...)(in this case the damper side a is connecting to the flange of revlute joint-rev)

In [ ]: ▶
```julia
module Pendulum3

using Modia

# Modia equation-based models
include("$(Modia.modelsPath)/AllModels.jl")

Pendulum = Model(
    world = Object3D(feature=Scene(animationFile="Pendulum3.json")),
#    obj1  = Object3D(feature=Solid(shape=Beam(axis=1, length=1.0, width=0.2, thickness=0.2),
#                solidMaterial="Steel", visualMaterial=VisualMaterial(color="Blue"))), #Beam doesnt work for som
    obj1  = Object3D(feature=Solid(shape=Box(lengthX=1.0, lengthY=0.2, lengthZ=0.2),
                solidMaterial="Steel", visualMaterial=VisualMaterial(color="Blue"))),

    obj2  = Object3D(parent=:obj1, feature=Visual(shape=Cylinder(diameter=0.1, length=0.21),
                visualMaterial=VisualMaterial(color="Red")), translation=[-0.5, 0.0, 0.0]),
                #parent:obj1 means that obj2 is wrb to obj1 tralates -0.5 x
    rev   = RevoluteWithFlange(obj1=:world, obj2=:obj2),

    damper  = Damper | Map(d=100.0),
    fixed   = Fixed,
    connect = :[(damper.flange_b, rev.flange),
                (damper.flange_a, fixed.flange)]
)

pendulum = @instantiateModel(buildModia3D(Pendulum), unitless=true)
simulate!(pendulum, stopTime=3.0)

@usingModiaPlot
plot(pendulum, "rev.phi")

end
```

In [ ]: ▶
```julia
module Pendulum1
# import Modia3D
using Modia
@usingModiaPlot
Modia.usePlotPackage("PyPlot")
Pendulum = Model(
    world     = Object3D(feature=Scene()),
    body      = Object3D(feature=Solid(massProperties=MassProperties(mass=1.0))),
    bodyFrame = Object3D(parent=:body, translation=[-0.5, 0.0, 0.0]),
    rev       = Revolute(obj1=:world, obj2=:bodyFrame)
)

# Pendulum = Pendulum | Map( bodyFrame = Object3D(parent=:body, translation=[-0.3, 0.0, 0.0]))

pendulum = @instantiateModel(buildModia3D(Pendulum), unitless=true)
simulate!(pendulum, stopTime=3.0u"s", merge = Map(bodyFrame = Object3D(parent=:body, translation=[-0.3, 0.0, 0.0
time=get_result(pendulum,"time")
angle=get_result(pendulum,"rev.phi")
# simulate!(pendulum, stopTime=3.0)

# @usingModiaPlot    ENV["MODIA_PLOT"] = "PyPlot" # use the plot package defined by ENV["MODIA_PLOT"]
# ModiaPlot_PyPlot.plot(pendulum, "rev.phi")
# plot(pendulum, "rev.phi")
plot(ustrip(time),ustrip(angle))
end
```
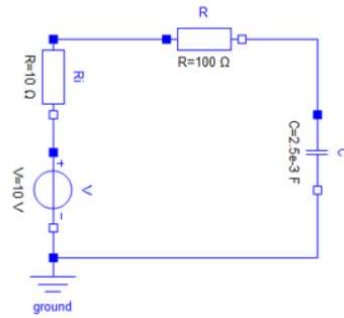
ModiaBase

ModiaBase provides functions to support the transformation of a Differential Algebraic Equation system (DAE) to an explicit Ordinary Differential Equation system (ODE) https://modiasim.github.io/ModiaBase.jl/stable/index.html (https://modiasim.github.io/ModiaBase.jl/stable/index.html)

- 1. Regular DAEs (Index Zero DAEs) https://modiasim.github.io/ModiaBase.jl/stable/Tutorial.html#.-Regular-DAEs-(Index-Zero-DAEs) (https://modiasim.github.io/ModiaBase.jl/stable/Tutorial.html#.-Regular-DAEs-(Index-Zero-DAEs))

R

R=100 Ω

R=10 Ω

Ri

V=10 V

V

C=2.5e-3 F

C

ground

Bi-Partite Graph

	Ri.v	Ri.n.v	R.v	R.p.v	C.v	der(C.v)	V.p.i	
1								Ri.v = R.p.v - Ri.n.v
2								Ri.R * V.p.i = Ri.v
3								R.v = R.p.v - C.v
4								R.R * -(V.p.i) = R.v
5								C.C * der(C.v) = -(V.p.i)
6								Ri.n.v = 10
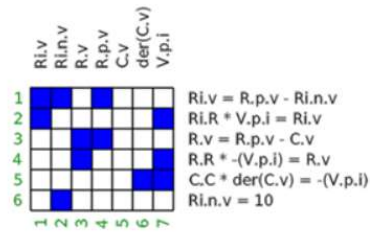
Assignment : Red marks show the assigned variables.Blue marks show if a variable is part of the respective equation
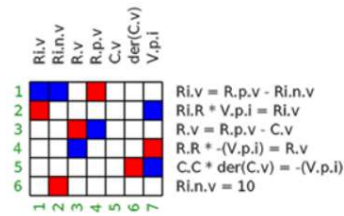
	Ri.v	Ri.n.v	R.v	R.p.v	C.v	der(C.v)	V.p.i	
1								Ri.v = R.p.v - Ri.n.v
2								Ri.R * V.p.i = Ri.v
3								R.v = R.p.v - C.v
4								R.R * -(V.p.i) = R.v
5								C.C * der(C.v) = -(V.p.i)
6								Ri.n.v = 10

Block Lower Triangular transformation : Red marks show the assigned variables. Blue marks show if a variable is part of the respective equation. A grey area marks an algebraic loop.

	Ri.n.v	R.v	V.p.i	Ri.v	R.p.v	der(C.v)	C.v	
6								Ri.n.v = 10
3								R.v = R.p.v - C.v
4								R.R * -(V.p.i) = R.v
2								Ri.R * V.p.i = Ri.v
1								Ri.v = R.p.v - Ri.n.v
5								C.C * der(C.v) = -(V.p.i)
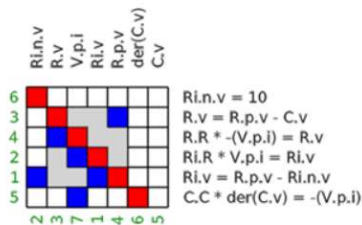
```
In [11]:  ▶  #1. Bi-partite graph of low pass filter
             G = Vector{Int}[ [1,2,4],
                              [1,7],
                              [3,4],
                              [3,7],
                              [6,7],
                              [2] ]
             #2.assignment
             # using Modia
             using ModiaBase
             M           = 7  # Number of variables
             vActive     = fill(true,M)
             vActive[5] = false     # state C.v is known
             assign      = ModiaBase.matching(G, M, vActive)

             # assign = [2,6,3,1,0,5,4]
             #3.sortting
             blt = ModiaBase.BLT(G, assign)

             #     blt = [ [6],
             #   ·         [3,4,2,1],
             #   ·         [5] ]
```

Out[11]: 3-element Vector{Any}:
           Any[6]
           Any[2, 4, 3, 1]
           Any[5]

Modia

- Combining a multibody system with equation based models (missing)
- Equation based pendulum example including uncertainty

The following equations describe a damped pendulum:

$$\frac{d\varphi}{dt} = \omega$$

$$0 = m \cdot L^2 \cdot \frac{d\omega}{dt} + d \cdot \omega + m \cdot g \cdot L \cdot sin(\varphi)$$

$$r = \begin{pmatrix} L * cos(\varphi) \\ -L * sin(\varphi) \end{pmatrix}$$

```
In [ ]:    ▶    using Modia
                # @usingModiaPlot  # Use plot package defined with
                                 # ENV["MODIA_PLOT"] or Modia.usePlotPackage(..)
                @usingModiaPlot
                Modia.usePlotPackage("PyPlot")

                Pendulum = Model(
                    L = 0.8u"m",
                    m = 1.0u"kg",
                    d = 0.5u"N*m*s/rad",
                    g = 9.81u"m/s^2",
                    phi = Var(init = 1.57*u"rad"),
                    w   = Var(init = 0u"rad/s"),
                    equations = :[
                            w = der(phi)
                          0.0 = m*L^2*der(w) + d*w + m*g*L*sin(phi)
                            r = [L*cos(phi), -L*sin(phi)]
                    ]
                )

                pendulum1 = @instantiateModel(Pendulum)
                simulate!(pendulum1, Tsit5(), stopTime = 10.0u"s", log=true)
                ModiaPlot_PyPlot.plot(pendulum1, [("phi", "w"); "r"], figure = 1)

                #add uncertainty

                using Measurements

                PendulumWithUncertainties = Pendulum | Map(L = (0.8 ± 0.2)u"m",
                                                           m = (1.0 ± 0.2)u"kg",
                                                           d = (0.5 ± 0.2)u"N*m*s/rad")

                pendulum2 =  @instantiateModel(PendulumWithUncertainties,
                                               FloatType = Measurement{Float64})

                simulate!(pendulum2, Tsit5(), stopTime = 10.0u"s")
                ModiaPlot_PyPlot.plot(pendulum2, [("phi", "w"); "r"], figure = 2)
```

4.Study the solution sequence outlined in the paper (previous) about the index reduction of DAE

Terminology:

- **A differential-algebraic equation (DAE)** is an equation involving an unknown function and its derivatives.

- **Index** is a notion used in the theory of DAEs for measuring the distance from a DAE to its related ODE. Specifically, it is the number of times needed to differentiate the DAEs to get a system of ODEs with respect to t.

Steps:

- **Matching**: Bipartite graph
  Find a unique mapping between variables and equations. Used both in BLT sorting and Pantelides algorithm.
  If we can find the matching from the incidence matrix, that means the DAE is an index-1 or index-0 such that we can proceed to BLT sorting.
  If the matching cannot be found, then we need to apply an index reduction stretegy of the DAE.

- **BLT sorting**: (Block Lower Triangular form with an algebraic loop.)
  An DAE in BLT form with algebraic loops is Index 1

Firstly, construct equation dependency graph.
Secondly, sort into blocks of equations using Tarjan's strongly connected mponent algotthm

- **Solving**: Numerical method such as BDF(Backward Differentiation Formula) and DAE integrator also doable

- **Index Reduction**: If the above Matching/Sorting strategies are not applicable to the current DAE, we need to apply an Index Reduction to the DAE and reduce toe DAE to first order. (For example, Pantelides algorithm)

Pantelides algorithm

- **MSS:** A structurally singular subset is called minimally structurally singular (MSS) if none of its proper subsets is structurally singular.
  A system of equations is called structurally singular with respect to a certain set of variables if it contains a structurally singular subset with respect to the same set of variables.

- An **assignment** is a set of edges $(i - j)$ such that no node or j appears in more than one edge in the set. An assignment is complete if it leaves no E-node exposed; otherwise it is called a partial assignment.
  Edges in the assignment are called matching edges.
  A **node** is exposed if it does not appear in any matching edge.
  An **augmenting path** is a path with exposed nodes at both ends and alternating nonmatching and matching edges between them; in the trivial case, the path may just consist of only one (nonmatching) edge between two exposed nodes.

- **Variable Association List A:** Relationships between variables and their derivatives with respect to time are expressed through a Variable Association List A defined as: $A(j) = k$ if $X_k = \frac{dX_j}{dt}, 1 \leq j, k \leq 2n + m; 0, otherwise$
  Example: $X_3 = \dot{x}, X_1 = x, so X_3 = \frac{dX_1}{dt}, so A(1) = 3$

- **Equation Association List B:** $B(i) = l$ if the l th equation has been created by differentiating equation i; 0, otherwise. List B is initialized to all zeros.

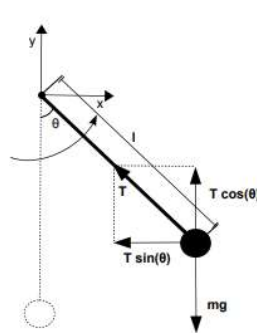Pantelides algorithm: Detailed code solution procedure can be found in this slide
https://ptolemy.berkeley.edu/projects/embedded/eecsx44/lectures/Spring2013/modelica-dae-part-2.pdf
(https://ptolemy.berkeley.edu/projects/embedded/eecsx44/lectures/Spring2013/modelica-dae-part-2.pdf)
and the principles of Pantelides algorithm are there https://epubs.siam.org/doi/pdf/10.1137/0909014
(https://epubs.siam.org/doi/pdf/10.1137/0909014)

- Pendulum example

Pendulum in Cartesian coordinate system

$$-T \cdot \frac{x}{l} = m\ddot{x}$$
$$-T \cdot \frac{y}{l} - mg = m\ddot{y}$$
$$x^2 + y^2 = l^2$$

Simplified using
$$-T/l = \lambda$$
$$m = 1$$
$$l^2 = L$$

Simplified

$$\ddot{x} = \lambda \cdot x$$
$$\ddot{y} = \lambda \cdot y - g$$
$$x^2 + y^2 = L$$

Is this an DAE?
Can we solve it?
Can we create BLT?
IDEA: Symbolically differentiate

Rewritten in first order

$$\dot{x} = u$$
$$\dot{y} = v$$
$$\dot{u} = \lambda \cdot x$$
$$\dot{v} = \lambda \cdot y - g$$
$$x^2 + y^2 = L$$

Incidence Matrix

	$\dot{x}$	$\dot{y}$	$\dot{u}$	$\dot{v}$	λ
f_1	1	0	0	0	0
f_2	0	1	0	0	0
f_3	0	0	1	0	1
f_4	0	0	0	1	1
f_5	0	0	0	0	0

$$vmap = \{x \mapsto \dot{x}, y \mapsto \dot{y}, u \mapsto \dot{u}, v \mapsto \dot{v}, \dot{x} \mapsto \ddot{x}, \dot{y} \mapsto \ddot{y}\}$$
$$eqmap = \{f_5 \mapsto f_6, f_6 \mapsto f_7, f_1 \mapsto f_8, f_2 \mapsto f_9\}$$
$$assign = \{\dot{x} \mapsto f_1, \dot{y} \mapsto f_2, \dot{u} \mapsto f_8, \dot{v} \mapsto f_4,$$
$$\lambda \mapsto f_3, \ddot{x} \mapsto f_7, \ddot{y} \mapsto f_9\}$$

(1) $\dot{x} = u$ $f_1(\dot{x}, u) = 0$

(2) $\dot{y} = v$ $f_2(\dot{y}, v) = 0$

(3) $\dot{u} = \lambda \cdot x$ $f_3(\dot{u}, \lambda, x) = 0$

(4) $\dot{v} = \lambda \cdot y - g$ $f_4(\dot{v}, \lambda, y) = 0$

(5) $x^2 + y^2 = L$ $f_5(x, y) = 0$

(6) $2x\dot{x} + 2y\dot{y} = 0$ $f_6(x, \dot{x}, y, \dot{y}) = 0$

(7) $2x\ddot{x} + 2\dot{x}^2 + 2y\ddot{y} + 2\dot{y}^2 = 0$ $f_7(x, \dot{x}, \ddot{x}, y, \dot{y}, \ddot{y}) = 0$

(8) $\ddot{x} = \dot{u}$ $f_8(\ddot{x}, \dot{u}) = 0$

(9) $\ddot{y} = \dot{v}$ $f_9(\ddot{y}, \dot{v}) = 0$

$|G.F| = 9$ $|G.V| = 11$

Two variables out of the set G.V can be given arbitrary initialization values, as long as all constraints above are satisfied.

5.Study the solution sequence of Modeling Toolkit

General example: https://mtk.sciml.ai/stable/#Feature-Summary (https://mtk.sciml.ai/stable/#Feature-Summary)
High level examples: https://juliapackages.com/p/modelingtoolkit (https://juliapackages.com/p/modelingtoolkit)
ModelingToolkit.jl is a modeling language for high-performance symbolic-numeric computation in scientific computing and scientific machine learning.

In [ ]:

```julia
using ModelingToolkit
using DifferentialEquations: solve
# using Plots: plot
import Plots
# import Pyplot
function unitstep_fol_factory(;name)
    @parameters τ
    @variables t x(t)
    D = Differential(t)
    ODESystem( D(x) ~ (1 - x)/τ ; name, defaults=Dict(x=>0.0,  τ=>1.0))
#     ODESystem( D(x) ~ (1 - x)/τ ; name)
end

ODEProblem(unitstep_fol_factory(name=:fol), [], (0.0,5.0),[]) |> solve
prob = ODEProblem(unitstep_fol_factory(name=:fol), [:x => 2.0], (0.0,10.0), [:τ => 5.0])

Plots.plot(solve(prob))
```

- Second Order ODE

$$x'' = \sigma \cdot (y - x)$$
$$y' = x \cdot (\rho - z) - y$$
$$z' = x \cdot y - \beta \cdot z$$

```julia
using ModelingToolkit, OrdinaryDiffEq

@parameters t σ ρ β
@variables x(t) y(t) z(t)
D = Differential(t)

eqs = [D(D(x)) ~ σ*(y-x),
       D(y) ~ x*(ρ-z)-y,
       D(z) ~ x*y - β*z]

@named sys = ODESystem(eqs)
sys = ode_order_lowering(sys)

u0 = [D(x) => 0.0,
      x => 1.0,
      y => 0.0,
      z => 0.0]

p  = [ σ => 10.0,
       ρ => 10.0,
       β => 8/3]

tspan = (0.0, 100.0)
prob = ODEProblem(sys, u0, tspan, p, jac=true)
sol = solve(prob, Tsit5())
import Plots;
Plots.plot(sol, vars=(x, y))
```
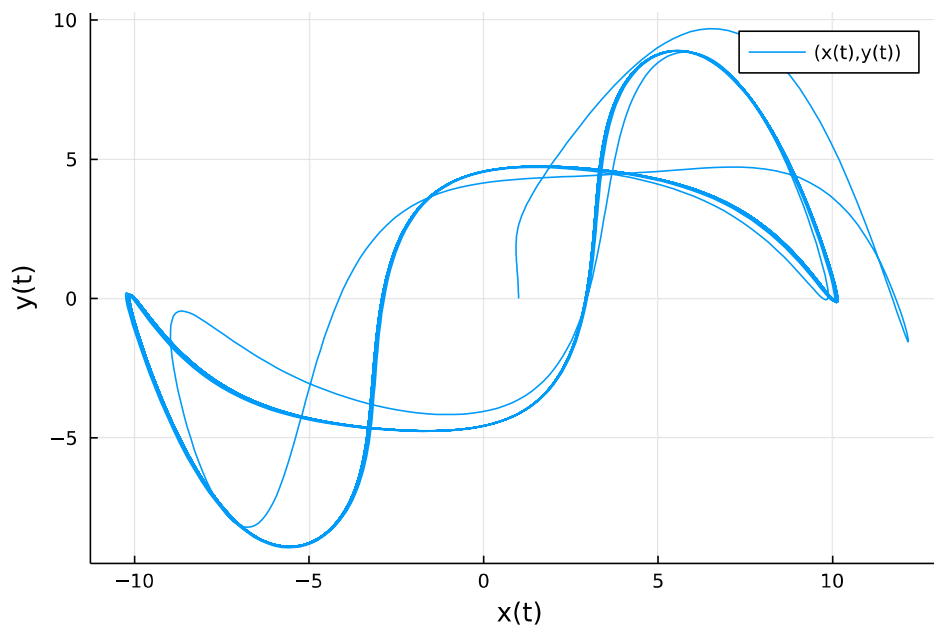
Out[57]:

- Pendulum Example: DAE reduction

$$x' = v_x$$
$$v'_x = Tx$$
$$y' = v_y$$
$$v'_y = Ty - g$$
$$0 = x^2 + y^2 - L^2$$

```julia
using ModelingToolkit
using LinearAlgebra
using OrdinaryDiffEq
import Plots

function pendulum!(du, u, p, t)
    x, dx, y, dy, T = u
    g, L = p
    du[1] = dx
    du[2] = T*x
    du[3] = dy
    du[4] = T*y - g
    du[5] = x^2 + y^2 - L^2
    return nothing
end
pendulum_fun! = ODEFunction(pendulum!, mass_matrix=Diagonal([1,1,1,1,0]))
u0 = [1.0, 0, 0, 0, 0]
p = [9.8, 1]
tspan = (0, 10.0)
pendulum_prob = ODEProblem(pendulum_fun!, u0, tspan, p)
traced_sys = modelingtoolkitize(pendulum_prob)  # use modelingtoolkitize to transform the numerical code into sy
pendulum_sys = structural_simplify(dae_index_lowering(traced_sys))#run dae_index_lowering lowering
prob = ODAEProblem(pendulum_sys, Pair[], tspan) #transform back to numerical code with ODEProblem
sol = solve(prob, Tsit5(),abstol=1e-8,reltol=1e-8)
Plots.plot(sol, vars=states(traced_sys))
```