Mechtools User's Manual
V.0.1

By Murat Tahtali (2019)

INTRODUCTION

Mechtools is a simple mechanism visualization and position simulation/solution tool.

It requires a correct set of error-functions and the associated Jacobian matrix implemented in the case specific Matlab function.

This case specific function is used to call the solver function SOLVEMECH.

The solver works independent from the visualization side where the various components need to be defined and the solution variables linked to them.

THE CASE SPECIFIC FUNCTION

The solver side is a generalized Newton-Raphson that requires a function and Jacobian matrix evaluator for the mechanism that is being simulated with the following function declaration format:

```
function [F,Fx]=specific_function_name(X,Input_var,[const_var1,const_var2,…])
```

Inputs:
X is a column vector containing N-rows, which are the current estimates of all the unknown parameters to be solved. These can be displacements, angles, or any other parameters that need solving.

Input_var is the changing input, such as the crank angle.

Outputs:
F is a column vector of N-rows, where each row is one of the functions evaluated at X, to be made equal to zero with the correct X. These functions are obtained from the Re and Im parts of the loop equations. F is also called the vector of error functions, since any value other than zero is the evaluation error for that function.

Fx is the N by N Jacobian matrix evaluated at X.

A typical implementations of the case specific function is given below:

```
function [F,Fx]=slider4bar(X,th12,a1,a2,a4,a5,b1,c1)
%SLIDER4BAR Loop closure equations and Jacobian for the example in
%In this example, the changing input is the crank angle th12. The rest of the
%inputs are constants, such as link lengths or any other constants.

%The following variables where extracted from X foe convenience. Any other
```

```
%calculations can be done here, as long as the function returns F and Fx.
th14=X(1);
th15=X(2);
S4=X(3);
S6=X(4);
F=[b1-a2*cos(th12)+a5*cos(th15)+S4*cos(th14)
    -c1-a2*sin(th12)+a5*sin(th15)+S4*sin(th14)
    b1-S6+a5*cos(th15)+a4*cos(th14)
    -a1-c1+a5*sin(th15)+a4*sin(th14)];

Fx=[-S4*sin(th14) -a5*sin(th15) cos(th14) 0
    S4*cos(th14)   a5*cos(th15)  sin(th14) 0
    -a4*sin(th14) -a5*sin(th15)      0    -1
    a4*cos(th14)   a5*cos(th15)      0     0];
```

The Solver

The solver function is SOLVEMECH. This function is generic and does not need any editing. Its declaration is as follows:

```
function [X,k,F,Fx]=solvemech(mechfun,X0, ...
                    epsilonE,epsilonS,maxiter,options,varargin)
```

mechfun is the specific function called with a "@" in front of it.

X0 is the column vector containing the initial guess of all the unknowns.

epsilonE is the vector of equation tolerances, the size of **X0**.

epsilonS is the vector of solution tolerances, the size of **X0**.

The solution will be deemed converged when both **F<epsilonE** and **abs(X^k-X^{k-1})<epsilonE**, element-wise.

maxiter is the maximum number of iterations before the solver gives up if the tolerances haven't been reached.

options for future use, currently enter [].

varargin the list of input arguments *`[const_var1,const_var2,…]`* declared in the specific function.

X is the converged or returned solution.

k is the number of iteration after which convergence was achieved. If **maxiter** is reached without convergence, it will return **–maxiter** (minus maxiter).

F is the function vector as described earlier, returned for debugging purposes if needed.

Fx is the Jacobian matrix as described earlier, returned for debugging purposes if needed.

A typical call to SOLVEMECH would be, referring to the specific function example:

```
[X,k]=solvemech(@slider4bar,X,epsilonE,epsilonS,maxiter,[],theta12,a1,a2,a4,a
5,b1,c1);
```
Noting that **F** and **Fx** were not required.
An external loop can then be programmed to increment **theta12** to solve the position variables and simulated full rotation of the input crank.

VISUALIZATION

To visualize the mechanism's motion, the components have to be created and the solution variables declared and initialized.

The components are declared using the following syntax:

```
[mech,lastindex]=createmechobject(mech,'typename', type_num, varargin);
```

Variables shown in square brackets are optional settings, such as
[,COLOR,SILENT,ORDER,LABEL,LABELOFFSET]
Refer to the Components Reference section for details.

varargin has the following syntax for each different type of component, where **typename** is the component name, and **type_num** is an incremental index for each component of that type.

BLOCK:

```
varargin={X0,Y0,WIDTH,HEIGHT,ANGLE[,COLOR,SILENT,ORDER,LABEL,LABELOFFSET]}
```

HINGE:

```
varargin={X0,Y0[,SCALE,COLOR,SILENT,ORDER,LABEL,LABELOFFSET[dx dy]]}
```

LINK:

```
varargin={ COORDTYPE,X0,Y0,X1,Y1[,tickness,COLOR,SILENT,ORDER,LABEL,LABELOFFSE
T]}
```

```
COORDTYPE =

            'c' for Cartesian coordinates

            'r' for polar coordinates
```

SLIDERGRND:

```
varargin={ X_center,Y_center[,Y_offset,Length,Angle_deg,scale,COLOR,SILENT,ORD
ER]}
```

PISTON:

```
varargin={COORDTYPE,X0,Y0,X1,Y1,THROW[,tickness,COLOR,SILENT,ORDER,LABEL,LABE
LOFFSET]}
```

SYMBOLIC AND NUMERIC VARIABLES

The variables used in the solution should be declared and initialized so that they can be used in the components' refresh.

An example of variables initialization is:

```
[symb,X]=initvariables('theta14',theta14,'theta15',theta15,'S4',S4,'S6',S6,'t
heta12',theta12_start);
```

The text entries are the symbolic references, and the variables are their initialized values. At this point, **X** will contain the initial guess of N (=4 in this example) unknowns, plus the initial value of the input variable.

symb contains the list of the unknown variables' names that can be used instead of any of the numerical variables when declaring the mechanical components. This will bind those inputs to the updated values of these variables. When using these variables, they should be in text form, as **'theta14'** in single quotes.

All numerical values up to and excluding the scale of the components can be referred symbolically containing mathematical expressions, such as **'link(1).x1'** referring to link 1's x1 coordinate (it's end's x-coordinate, noting that x0 is it starting x-coordinate).

REGENERATING THE MECHANISM

After each crank increment, the REGENMECH function should be called with the following syntax to regenerate the mechanism at it converged position:

```
mech=regenmech(mech,symb,X);
```

SIMULATION LOOP

A typical simulation loop will look like the following, provided that all relevant variables have been initialised:

```
axis([-150 120 -120 100]);
%axis square
axis equal
%Store AXIS settings to be used later in each refresh
ax=axis;
drawnow %force the graphics to draw now
%%
for i=0:Npoints,
    theta12=theta12_start+i*dtheta12;
    th12(i+1)=theta12;

[X,k]=solvemech(@slider4bar,X,epsilonE,epsilonS,maxiter,[],theta12,a1,a2,a4,a
5,b1,c1);
    %check if the solver converged (if k is non zero)
    if k>0, %regenerate the mechanism
        theta14=X(1);
        th14(i+1)=theta14;
        theta15=X(2);
        S4=X(3);
        s4(i+1)=S4;
        S6=X(4);
        mech=regenmech(mech,symb,X);
        axis(ax) %set the original axis so that the figure is not jumpy due
to limits changing
        drawnow %force the graphics to draw now
    end
end
```

Refer to **Mechanism1_V2.m** for a working example.

COMPONENTS REFERENCE

```
BLOCK Generates a block at a point with given angle

Usage: BLOCK(X0,Y0,WIDTH,HEIGHT,ANGLE[,COLOR,SILENT,ORDER,LABEL,LABELOFFSET])

      BLOCK(CC) regenerate an existing BLOCK object

SILENT =

           0 generate only, don't display

           1 generate and display

COLOR = 'r','b','k','g',...
```

```
HINGE Generates a fixed revolute joint

Usage:

HINGE(X0,Y0[,SCALE,COLOR,SILENT,ORDER,LABEL,LABELOFFSET[dx dy]])

HINGE(CC) regenerate an existing HINGE object

SILENT =

           0 generate only, don't display

           1 generate and display

COLOR = 'r','b','k','g',...
```

```
LINK Generates a straight link

Usage:

LINK(COORDTYPE,X0,Y0,X1,Y1[,tickness,COLOR,SILENT,ORDER,LABEL,LABELOFFSET])

LINK(CC) regenerate an existing LINK object
```

```
COORDTYPE =

           'c' for Cartesian coordinates

           'r' for polar coordinates

               X1 and Y1 become Length and Angle, respectively

SILENT =

           0 generate only, don't display

           1 generate and display

COLOR = 'r','b','k','g',...
```

```
SLIDERGRND Generates a ground surface

Usage:

SLIDERGRND(X_center,Y_center[,Y_offset,Length,Angle_deg,scale,COLOR,SILENT,OR
DER])

SLIDERGRND(CC) regenerate an existing SLIDERGRND object

SILENT =

           0 generate only, don't display

           1 generate and display

COLOR = 'r','b','k','g',...
```

```
PISTON Generates a piston

Usage:

PISTON(COORDTYPE,X0,Y0,X1,Y1,THROW[,tickness,COLOR,SILENT,ORDER,LABEL,LABELOF
FSET])

PISTON(CC) regenerate an existing PISTON object

COORDTYPE =

           'c' for Cartesian coordinates

           'r' for polar coordinates

               X1 and Y1 become Length and Angle, respectively

SILENT =

           0 generate only, don't display

           1 generate and display

COLOR = 'r','b','k','g',...
```