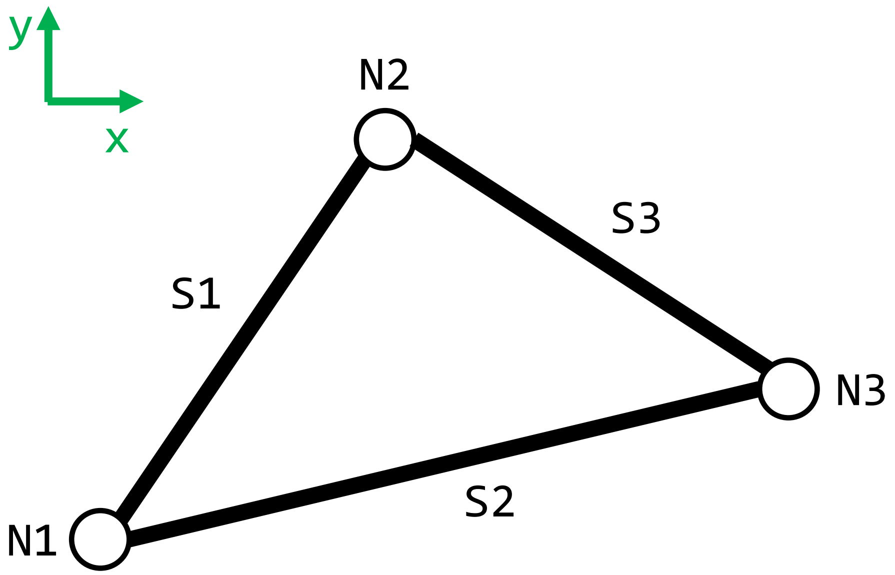
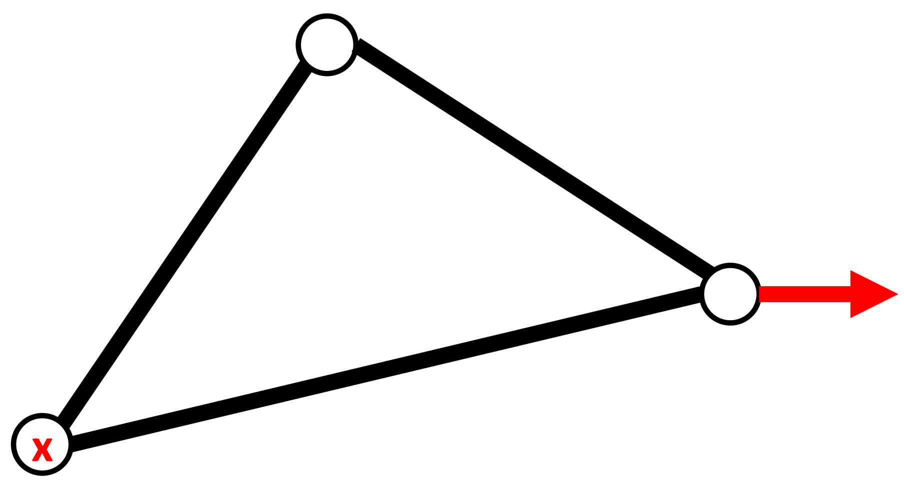
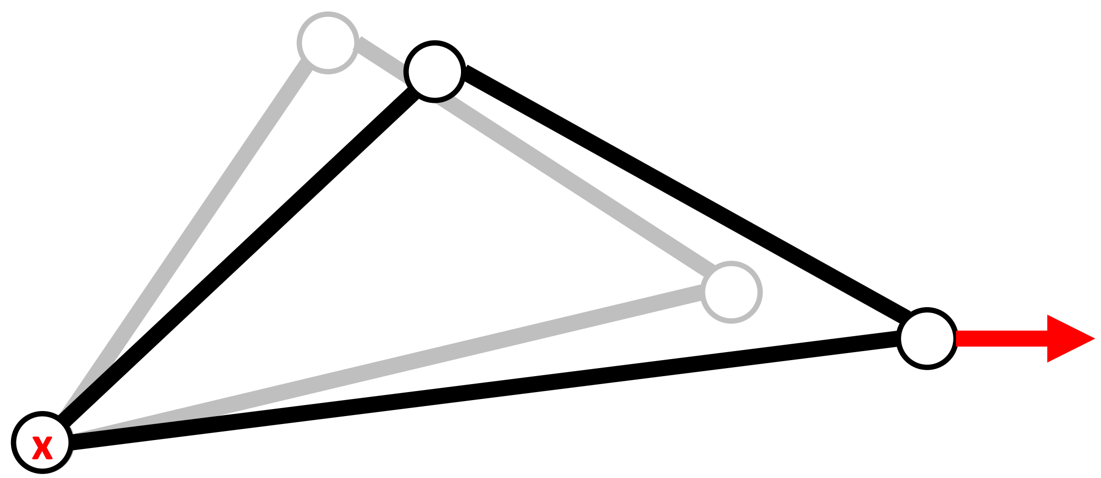
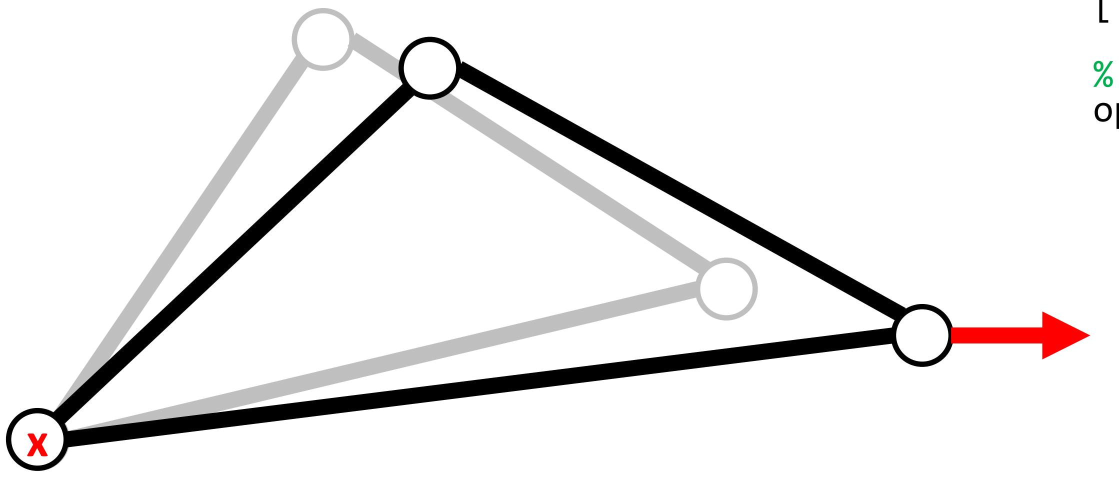






```
% nodal coordinates (x,y)
% one row for each node
nodes.position = [ 0.0 , 0.0     %N1
                   0.7 , 1.0     %N2
                   1.7 , 0.4 ] ; %N3

% each spring connects 2 nodes
% one row for each spring
% node indexes (start,end)
springs.nodes = [ 1 , 2          %S1
                  1 , 3          %S2
                  2 , 3 ] ;      %S3

% spring properties
% one row for each spring
% rest_length is length when spring force is zero
springs.rest_length = [ 1.0
                        1.0
                        1.0 ] ;
% is compression allowed for each spring?
springs.compression = [ false
                        false
                        false ] ;

% stiffness is defined by polynomial coefficients
% each n'th column is the coefficient for (length-rest_length)^n
% leave empty for zero force during tension/compression
springs.stiffness_tension = [ 2.0 ,  0.0     % F = 2.0*dx
                              1.0 , +0.1     % F = 1.0*dx + 0.1*dx*dx
                              1.0 , -0.1 ] ; % F = 1.0*dx + 0.1*dx*dx
springs.stiffness_compression = [] ;

% nodal boundary conditions (x,y)
% is node position locked in each direction?
nodes.fixed = [ true  , true
                false , false
                false , false ] ;
% external applied force in each direction
nodes.force = [  0.0 , 0.0
                 0.0 , 0.0
                +1.0 , 0.0 ] ;

% solve equilibrium node positions
% note: springs_eq should not be different from springs
[ nodes_eq , springs_eq ] = springs_solve( nodes , springs ) ;
```



Two choices for objective function (i.e. the function to be minimized numerically):

Sum of elastic energy in each spring:

$$\min \sum_s \frac{1}{2} F_s \left(L_s - L_{0,s} \right)$$

F_s = elastic force in spring s
L_s = length of spring s
$L_{0,s}$ = rest length of spring s

Sum of net force magnitude at each node:

$$\min \sum_n \left\| \mathbf{F}_{\text{ext},n} + \sum_s \mathbf{F}_{s,n} \right\|$$

$F_{\text{ext},n}$ = external force applied to node n
$F_{s,n}$ = elastic force applied to node n by spring s

```
% solver parameters not specified are treated as default
options = springs_default_options() ;
[ nodes_eq , ~ ] = springs_solve( nodes , springs , options ) ;

% these are the default options
options = struct( ...
          'precision' , 'double' , ...
          'algorithm' , 'newton' , ...
          'objective' , 'energy' , ...
          'num_iter_save' , 0 , ...
          'num_iter_print' , 0 , ...
          'num_iter_max' , 0 , ...
          'include_force_fixed_nodes' , false , ...
          'use_numerical_hessian' , false , ...
          'tolerance_change_energy' , 1.0e-12 , ...
          'tolerance_sum_net_force' , 1.0e-12 ) ;

% choices for floating point precision
options.precision = 'single' ; % 32 bit
options.precision = 'double' ; % 64 bit (default)

% choices for algorithm
options.algorithm = 'newton' ;   % newton method, 2nd order (default)
options.algorithm = 'steepest' ; % gradient descent, 1st order
options.algorithm = 'anneal' ;   % simulated annealing

% choices for objective
options.algorithm = 'energy' ;   % sum of elastic spring energy across all springs
options.algorithm = 'sumforce' ; % sum of net force magnitude across all nodes
options.algorithm = 'maxforce' ; % maximum net force magnitude at a node

% choices for hessian calculation
% note: only applies to newton method algorithm
options.use_numerical_hessian = false ; % analytical spring energy hessian (default)
options.use_numerical_hessian = true ;  % compute numerically by finite differences

% choices for force-based objective functions
options.include_force_fixed_nodes = false ; % exclude net forces at fixed nodes in objective
options.include_force_fixed_nodes = true ;  % include net forces at fixed nodes in objective

% convergence criteria
% note: a value <= 0 will use the default tolerance for the specific algorithm
% note: if use_sum_net_force is true, then these two tolerances should be the same
options.tolerance_change_energy = 1e-24 ; % change in objective function
options.tolerance_sum_net_force = 1e-24 ; % change in sum of nodal net force magnitudes

% status updates
% note: a value <= 0 will use the default tolerance for the specific algorithm
options.num_iter_save = 0 ;  % save intermediate network state every # iterations
options.num_iter_print = 0 ; % print convergence updates every # iterations
options.num_iter_max = 0 ;   % maximum number of iterations allowed
```

```
% solve equilibrium node positions
% note: springs_eq should not be different from springs
[ nodes_eq , springs_eq ] = springs_solve( nodes , springs ) ;

% there are several ways to display spring network results.
% see help documentation for display_2D() for more details.

% this is one example with spring networks colored by stiffness
h_before = display_2D( nodes_eq       , springs , 'stiffness_tension' , [0,2] , parula(10) , true ) ;
h_after  = display_2D( nodes_eq_force , springs , 'stiffness_tension' , [0,2] , parula(10) , true ) ;
linkaxes( [h_before.ax,h_after.ax] , 'xy' ) ;

% this is one example with spring networks colored by calculated spring forces
h_before = display_2D( nodes_eq       , springs , springs_tension(nodes_eq       ,springs) , [0,0.2] , parula(10) , true ) ;
h_after  = display_2D( nodes_eq_force , springs , springs_tension(nodes_eq_force,springs) , [0,0.2] , parula(10) , true ) ;
linkaxes( [h_before.ax,h_after.ax] , 'xy' ) ;
```