## Liquid-Line Restriction

### Description

def description

return "A liquid-line restriction fault occurs when particles accumulate within the refrigerant filter located between the condenser and the expansion valve in the refrigerant circuit of a vapor compression cycle. The accumulation increases the flow resistance of the refrigerant circuit and the pressure difference across the compressor. It also reduces the evaporating temperature and leads to lower cooling capacity, efficiency, and SHR. The lower SHR leads to increased latent load to meet a particular sensible load. This measure simulates a liquid-line restriction by modifying the Coil:Cooling:DX:SingleSpeed object in EnergyPlus assigned to the heating and cooling system. The fault intensity (F) for this fault is defined as the ratio of increase in the pressure difference between the condenser outlet and evaporator inlet due to the restriction."

end

### Modeler Description

def modeler\_description

return "Thirty two user inputs (DX coil where the fault occurs / Ratio of liquid line restriction fault / rated cooling capacity / rated sensible heat ratio / rated volumetric flow rate / minimum-maximum evaporator air inlet wet-bulb temperature / minimum-maximum condenser air inlet dry-bulb temperature / minimum/maximum rated COP / percentage change of UA with increase of fault level / time required for fault to reach full level / fault starting month / fault starting date / fault starting time / fault ending month / fault ending date / fault ending time) can be defined or remained with default values. Based on user inputs, the cooling capacity (Q ̇\_cool) and EIR in the DX cooling coil model is recalculated to reflect the faulted operation. The time required for the fault to reach the full level is only required when user wants to model dynamic fault evolution. If dynamic fault evolution is not necessary for the user, it can be defined as zero and the fault intensity will be imposed as a step function with user defined value. However, by defining the time required for the fault to reach the full level, fault starting month/date/time and fault ending month/date/time, the adjustment factor AF is calculated at each time step starting from the starting month/date/time to gradually impose fault intensity based on the user specified time frame. AF is calculated as follows, AF\_current = AF\_previous + dt/tau where AF\_current is the adjustment factor calculated based on the previously calculated adjustment factor (AF\_previous), simulation timestep (dt) and the time required for the fault to reach the full level (tau)."

end

### Measure Type

EnergyPlus Measure

**Taxonomy**

HVAC.Cooling

### Arguments

def arguments(workspace)

args = OpenStudio::Ruleset::OSArgumentVector.new

list = OpenStudio::StringVector.new

list << $all\_coil\_selection

singlespds = workspace.getObjectsByType("Coil:Cooling:DX:SingleSpeed".to\_IddObjectType)

singlespds.each do |singlespd|

list << singlespd.name.to\_s

end

twostages = workspace.getObjectsByType("Coil:Cooling:DX:TwoStageWithHumidityControlMode".to\_IddObjectType)

twostages.each do |twostage|

list << twostage.name.to\_s

end

#make choice arguments for Coil:Cooling:DX:SingleSpeed

coil\_choice = OpenStudio::Ruleset::OSArgument::makeChoiceArgument("coil\_choice", list, true)

coil\_choice.setDisplayName("Enter the name of the faulted Coil:Cooling:DX:SingleSpeed object. If you want to impose the fault on all coils, select #{$all\_coil\_selection}")

coil\_choice.setDefaultValue($all\_coil\_selection)

args << coil\_choice

# make a double argument for the fault level

fault\_lvl = OpenStudio::Ruleset::OSArgument.makeDoubleArgument('fault\_lvl', false)

fault\_lvl.setDisplayName('Ratio of non-condensable in the system [-]')

fault\_lvl.setDefaultValue(0.1) # defaulted at 10%

args << fault\_lvl

# rated cooling capacity

q\_rat = OpenStudio::Ruleset::OSArgument.makeDoubleArgument('q\_rat', true)

q\_rat.setDisplayName('Rated cooling capacity of the cooling coil for bypass factor model adjustment. If your system is autosized or you do not know what this is, please run the OS Measure Auto Size to Hard Size before this Measure. If your system is hard sized, leave this value at -1.0. (W)')

q\_rat.setDefaultValue(-1.0)

args << q\_rat

# rated sensible heat ratio

shr\_rat = OpenStudio::Ruleset::OSArgument.makeDoubleArgument('shr\_rat', true)

shr\_rat.setDisplayName('Rated sensible heat ratio of the cooling coil for bypass factor model adjustment. If your system is autosized or you do not know what this is, please run the OS Measure Auto Size to Hard Size before this Measure. If your system is hard sized, leave this value at -1.0.')

shr\_rat.setDefaultValue(-1.0)

args << shr\_rat

# rated volumetric flow rate

vol\_rat = OpenStudio::Ruleset::OSArgument.makeDoubleArgument('vol\_rat', true)

vol\_rat.setDisplayName('Rated air flow rate of the cooling coil for bypass factor model adjustment. If your system is autosized or you do not know what this is, please run the OS Measure Auto Size to Hard Size before this Measure. If your system is hard sized, leave this value at -1.0. (m3/s)')

vol\_rat.setDefaultValue(-1.0)

args << vol\_rat

# fault level limits

min\_fl = OpenStudio::Ruleset::OSArgument.makeDoubleArgument('min\_fl', true)

min\_fl.setDisplayName('Maximum value of fault level [-]')

min\_fl.setDefaultValue(0.8)

args << min\_fl

# coefficients of models should be inputs.

# the model simulates the ratio of the cooling capacity and EIR of the faulted system to the ones of the non-faulted case

# the form of the model is

# RATIO = 1 + FaultLevel\*(a1+a2\*Tdb,i+a3\*Twb,i+a4\*Tc,i+a5\*FaultLevel+a6\*FaultLevel\*FaultLevel+a7\*(Rated COP))

# undercharging model

args = enter\_coefficients(args, $q\_para\_num, "Q\_#{$faultnow}", [-1.952000, 3.894600, -1.668500, -0.044405, -0.015189, -0.062296], '')

args = enter\_coefficients(args, $eir\_para\_num, "EIR\_#{$faultnow}", [2.464800, -3.189800, 1.058200, 0.040316, -0.000329, -0.038164], '')

min\_wb\_tmp\_uc = OpenStudio::Ruleset::OSArgument.makeDoubleArgument('min\_wb\_tmp\_uc', true)

min\_wb\_tmp\_uc.setDisplayName('Minimum value of evaporator air inlet wet-bulb temperature [C]')

min\_wb\_tmp\_uc.setDefaultValue(12.8) # the first number is observed from the training data, and the second number is an adjustment for range

args << min\_wb\_tmp\_uc

max\_wb\_tmp\_uc = OpenStudio::Ruleset::OSArgument.makeDoubleArgument('max\_wb\_tmp\_uc', true)

max\_wb\_tmp\_uc.setDisplayName('Maximum value of evaporator air inlet wet-bulb temperature [C]')

max\_wb\_tmp\_uc.setDefaultValue(23.9)

args << max\_wb\_tmp\_uc

min\_cond\_tmp\_uc = OpenStudio::Ruleset::OSArgument.makeDoubleArgument('min\_cond\_tmp\_uc', true)

min\_cond\_tmp\_uc.setDisplayName('Minimum value of condenser air inlet temperature [C]')

min\_cond\_tmp\_uc.setDefaultValue(18.3)

args << min\_cond\_tmp\_uc

max\_cond\_tmp\_uc = OpenStudio::Ruleset::OSArgument.makeDoubleArgument('max\_cond\_tmp\_uc', true)

max\_cond\_tmp\_uc.setDisplayName('Maximum value of condenser air inlet temperature [C]')

max\_cond\_tmp\_uc.setDefaultValue(46.1)

args << max\_cond\_tmp\_uc

min\_cop\_uc = OpenStudio::Ruleset::OSArgument.makeDoubleArgument('min\_cop\_uc', true)

min\_cop\_uc.setDisplayName('Minimum value of rated COP')

min\_cop\_uc.setDefaultValue(3.74)

args << min\_cop\_uc

max\_cop\_uc = OpenStudio::Ruleset::OSArgument.makeDoubleArgument('max\_cop\_uc', true)

max\_cop\_uc.setDisplayName('Maximum value of rated COP')

max\_cop\_uc.setDefaultValue(4.69)

args << max\_cop\_uc

# model for BF offset

bf\_para = OpenStudio::Ruleset::OSArgument.makeDoubleArgument('bf\_para', false)

bf\_para.setDisplayName('Percentage change of UA with increase of fault level level (% of UA/% of fault level)')

bf\_para.setDefaultValue(0.373) # default change of bypass factor level with fault level in %

args << bf\_para

#Parameters for transient fault modeling

#make a double argument for the time required for fault to reach full level

time\_constant = OpenStudio::Ruleset::OSArgument::makeDoubleArgument('time\_constant', false)

time\_constant.setDisplayName('Enter the time required for fault to reach full level [hr]')

time\_constant.setDefaultValue(0) #default is zero

args << time\_constant

#make a double argument for the start month

start\_month = OpenStudio::Ruleset::OSArgument::makeDoubleArgument('start\_month', false)

start\_month.setDisplayName('Enter the month (1-12) when the fault starts to occur')

start\_month.setDefaultValue(6) #default is June

args << start\_month

#make a double argument for the start date

start\_date = OpenStudio::Ruleset::OSArgument::makeDoubleArgument('start\_date', false)

start\_date.setDisplayName('Enter the date (1-28/30/31) when the fault starts to occur')

start\_date.setDefaultValue(1) #default is 1st day of the month

args << start\_date

#make a double argument for the start time

start\_time = OpenStudio::Ruleset::OSArgument::makeDoubleArgument('start\_time', false)

start\_time.setDisplayName('Enter the time of day (0-24) when the fault starts to occur')

start\_time.setDefaultValue(9) #default is 9am

args << start\_time

#make a double argument for the end month

end\_month = OpenStudio::Ruleset::OSArgument::makeDoubleArgument('end\_month', false)

end\_month.setDisplayName('Enter the month (1-12) when the fault ends')

end\_month.setDefaultValue(12) #default is Decebmer

args << end\_month

#make a double argument for the end date

end\_date = OpenStudio::Ruleset::OSArgument::makeDoubleArgument('end\_date', false)

end\_date.setDisplayName('Enter the date (1-28/30/31) when the fault ends')

end\_date.setDefaultValue(31) #default is last day of the month

args << end\_date

#make a double argument for the end time

end\_time = OpenStudio::Ruleset::OSArgument::makeDoubleArgument('end\_time', false)

end\_time.setDisplayName('Enter the time of day (0-24) when the fault ends')

end\_time.setDefaultValue(23) #default is 11pm

args << end\_time

return args

end

### Initial Condition

#Select DX unit object that is being faulted.

runner.registerInitialCondition("Imposing performance degradation on #{coil\_choice}.")

### Final Condition

#Impose performance degradation due to fault on the DX unit object.

runner.registerFinalCondition("Imposed performance degradation on #{coil\_choice}.")

### Not Applicable

#When the fault level is defined as less than 0.001,

runner.registerAsNotApplicable("LiquidLineRestriction is not running for #{coil\_choice}. Skipping......")

### Warning

n/a

### Error

#When fault intensity constant value is defined but the range is outside the limit (0-1),

runner.registerError("Fault level #{fault\_lvl} for #{coil\_choice} is outside the range from 0 to 1. Exiting......")

#When Condenser Type option of DX unit is not defined as AirCooled

runner.registerError("#{coil\_choice} is not air cooled. Impossible to continue in LiquidLineRestriction. Exiting......")

#When DX unit performance curve is not defined,

runner.registerError("No Temperature Adjustment Curve for #{coil\_choice} #{curve\_name} model. Exiting......")

#When the DX unit cannot be found,

runner.registerError("Measure LiquidLineRestriction cannot find #{coil\_choice}. Exiting......")

### Information

* Measures below share the same resource codes.
  + Condenser Fouling
  + Liquid-line Restriction
  + Nonstandard Charging
  + Presence of Noncondensable
* Works with,
  + Coil:Cooling:DX:SingleSpeed
  + Coil:Cooling:DX:TwoStageWithHumidityControlMode.
* Future refinement items are,
  + Capability to work with other DX models.
  + Capability of generic autosizing to hardsizing.

### Code Outline

* Define arguments.
  + DX unit where the fault occurs.
  + Fault intensity in constant value.
  + DX unit rated cooling capcity.
  + DX unit rated sensible heat ratio.
  + DX unit rated volumetric flow rate.
  + Minimum value of fault level
  + Minimum value of evaporator air inlet wet-bulb temperature
  + Maximum value of evaporator air inlet wet-bulb temperature
  + Minimum value of condenser air inlet temperature
  + Maximum value of condenser air inlet temperature
  + Minimum value of rated COP
  + Maximum value of rated COP
  + Percentage change of UA with increase of fault level level
* Find the DX unit where fault occurs.
* Check whether fault intensity value is valid between 0-1.
* Check rated SHR in the selected DX unit and replace with degraded value... shr\_modification
* Create string object in idf (with EMS) for fault implementation.
  + Create fractional schedule object for fault level implementation... \_create\_schedules\_and\_typelimits
    - Create schedule object according to fault level... \_create\_schedule\_objects\_create\_schedule\_objects
      * Returns workspace object in certain category... get\_workspace\_objects
      * Trim name without space and symbols... name\_cut
    - Create schedule object with zero and one... no\_fault\_schedules
  + Append EMS code for altering cooling capacity and EIR due to fault... \_write\_ems\_curves
    - Write EMS code to generate capacity and EIR performance curve... \_write\_q\_and\_eir\_curves
      * Write EMS code to alter performance curve... \_write\_curves
        + Get parameters from biquadratic function... para\_biquadratic\_limit
        + Write EMS main program to alter temperature curve... main\_program\_entry
    - Write EMS code to alter capacity and EIR performance... \_write\_q\_and\_eir\_adj\_routine
      * Returns parameters for capacity and EIR calculation... \_get\_parameters
        + Returns list of parameters min & max temperature & COP... \_get\_ext\_from\_argumets
        + Returns an array of coefficients... runner\_pass\_coefficients
      * Write EMS code to calculate fault impact ratio... general\_adjust\_function
      * Write dummy EMS code in case of fault is not modeled... dummy\_fault\_sub\_add
  + Append EMS code including fault level schedule... fault\_level\_sensor\_sch\_insert
  + Append EMS code for defining EMS sensor object... \_write\_ems\_sensors
    - Create EMS sensor object... ems\_sensor\_str
    - Check whether the same object already exists... check\_exist\_workspace\_objects
  + Append EMS code for defining EMS output object.

### Tests

* Test invalid user argument values to make sure measure fails gracefully
* Test different levels of fault intensity
* Test different sets of rated conditions
* Test different sets of coefficients for regression model