Process System Engineering #4

#03150796 Amane Suzuki

October 27, 2015

1 Abstract

The purpose of this problem set is to estimate the costs of electricity, steam, reactor, and coolant.

2 Theory

Electiricity Cost Electricity cost is represented by,

$$\text{Cost}_e = \text{Price}_e \text{Power}_{\text{sum}}$$

Steam Cost In this problem set, we use steam to separate BR and toluene. The latent heat of water convert to heat of vaporization of toluene and evaporate it.

The weight of toluene is represented by,

$$w_{\text{toluene}} = \frac{G_p}{\gamma_0}$$

Given β as efficiency, heat balance is represented by,

$$w_{\text{toluene}} C_{p,\text{toluene}} = \beta w_{\text{steam}} C_{p,\text{steam}}$$

Steam cost is represented by,

$$\text{Cost}_{\text{steam}} = w_{\text{steam}} \text{Price}_{\text{steam}}$$

Reactor Cost Reactor cost is represented by,

$$\text{Cost}_{\text{reactor}} = \frac{\text{Price}_{\text{reactor}}}{\text{Span}}$$

Coolant Cost Coolant cost is represented by,

$$\text{Cost}_{\text{coolant}} = \text{Price}_{\text{coolant}} \frac{Q_{\text{sum}}}{C_{p,\text{steam}} \Delta T}$$

3 Algorythm

Coolant price is decided by coolant temperature. Figure.1 shows to connect each point linearly because the prices are descrete.

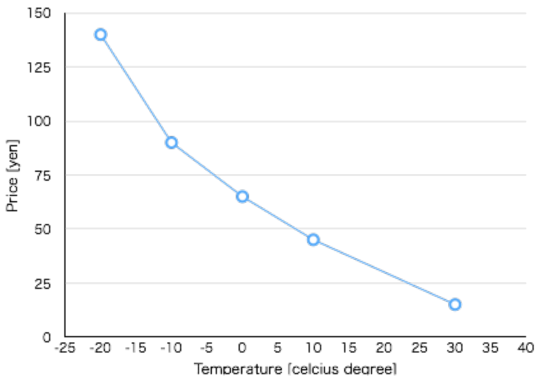

Figure 1: Price of Coolant

In `calc_cost()` function, calculate each cost and sum up these.

```
def calc_cost()
  calc electricity_cost by power consumption
  show electricity_cost

  calc toluene_weight
  calc steam_weight by heat balance
  calc steam_cost
  show steam_cost

  calc reactor_price by reactor size
  calc reactor_cost
  show reactor_cost

  calc coolant_weight by total heat of reaction
  calc coolant_cost
  show coolant_cost

  calc total_cost
  show total_cost
end
```

4 Result

(1) $N = 1, \gamma_0 = 0.05, T_2 = 258$

Use listing 1,

```
$ ruby reactor.rb
input n[-], gamma_0[wt%], T2[K]
1
0.05
258
elec: 0.106 [yen/s]
steam: 23.455 [yen/s]
reactor: 1.092 [yen/s]
coolant: 2.807 [yen/s]
total: 27.46 [yen/s]
```

Electricity Cost	0.106 ¥/s
Steam Cost	23.455 ¥/s
Reactor Cost	1.092 ¥/s
Coolant Cost	2.807 ¥/s
Total	27.46 ¥/s

Table 1: Result in $(N, \gamma_0, T_2) = (1, 0.05, 258)$

(2) $N = 3, \gamma_0 = 0.04, T_2 = 258$

Use listing 1,

```
$ ruby reactor.rb
input n[-], gamma_0[wt%], T2[K]
3
0.04
258
elec: 0.004 [yen/s]
steam: 29.319 [yen/s]
reactor: 2.17 [yen/s]
coolant: 2.706 [yen/s]
total: 34.198 [yen/s]
```

Electricity Cost	0.004 ¥/s
Steam Cost	29.319 ¥/s
Reactor Cost	2.17 ¥/s
Coolant Cost	2.706 ¥/s
Total	34.198 ¥/s

Table 2: Result in $(N, \gamma_0, T_2) = (3, 0.04, 258)$

(3) $N = 5, \gamma_0 = 0.03, T_2 = 300$

Use listing 1,

```
$ ruby reactor.rb
input n[-], gamma_0[wt%], T2[K]
5
0.03
300
elec: 0.048 [yen/s]
steam: 39.091 [yen/s]
reactor: 3.321 [yen/s]
coolant: 0.466 [yen/s]
total: 42.927 [yen/s]
```

Electricity Cost	0.048 ¥/s
Steam Cost	39.091 ¥/s
Reactor Cost	3.321 ¥/s
Coolant Cost	0.466 ¥/s
Total	34.198 ¥/s

Table 3: Result in $(N, \gamma_0, T_2) = (5, 0.03, 300)$

5 Discussion

The most major factor deciding cost of product is the cost of steam. So when we optimize cost of product, we should intend to minimalize the cost of steam.

6 Source Program

Listing 1: reactor.rb

```
include Math
require "./init"

class Plant
  def initialize(n, feed, coolant)
    @n = n # number of reactors [-]
    @feed = feed # feed [wt%]
    @rate = PRODUCT_FLOW_RATE/(DENSITY*CONVERSION*feed) # feed speed [m3 h-1]
    @k = (1.0/(1-CONVERSION)-1)/RESIDENCE_TIME # reaction constant [h-1]
    @tau = (1.0/@k)*((1-CONVERSION)**(-1.0/@n)-1) # residence time when n!=1
    @volume = @rate*@tau # [m3]
    @diameter = (@volume/(ALPHA*2*PI))**(1/3.0)*2 # [m]
    @height = @diameter*ALPHA # [m]
    @surface = @diameter*PI*@height # [m2]
    @coolant = coolant # T2 [K]
    @data = Array.new(n) # data of each reactor
  end

  def show()
    # conditions
    puts "Conditions:"
    puts "(N, gamma_0, T2) = (#{@n}, #{@feed}, #{@coolant})"

    # reactor size
    puts "Reactor Size:"
    puts "V = #{@volume.round(3)} [m3]"
    puts "D = #{@diameter.round(3)} [m]"
    puts "H = #{@height.round(3)} [m]"

    # result of each reactor
    puts "Results:"
    for n in 0...@n
      puts "##{n+1}"
      puts "Re = #{@data[n][:re].round(3)}"
      puts "n = #{@data[n][:revolution].round(3)} [rps]"
      puts "P = #{@data[n][:power].round(3)} [W]"
    end

    puts "Total:"
    puts "Ptot = #{@total_power.round(3)} [W]"
  end
```

```ruby

  def calc()
    prop = (1-CONVERSION)**(1.0/@n) # proportional constant [-]
    for n in 0...@n
      @data[n] = reactor(@feed*(prop**(n)),@feed*(prop**(n+1)), 0)
    end

    @total_power = 0
    @total_heat = 0
    @data.each do |data|
      @total_power += data[:power]
      @total_heat += data[:heat]
    end
  rescue
    @total_power = nil
    @total_heat = nil
  end

  def calc_cost()
    electricity_cost = ELECTRICITY_PRICE*@total_power/(1000*3600) # [yen s-1]
    puts "elec: #{electricity_cost.round(3)} [yen/s]"

    toluene_wt = (PRODUCT_FLOW_RATE/@feed)/3600/1000 # [ton s-1]
    toluene_heat = toluene_wt*TOLUENE_LATENT_HEAT # [kJ s-1]
    steam_heat = toluene_heat/THERMAL_EFFICIENCY # [kJ s-1]
    steam_wt = steam_heat/WATER_LATENT_HEAT # [ton s-1]
    steam_cost = steam_wt*STEAM_PRICE # [yen s-1]
    puts "steam: #{steam_cost.round(3)} [yen/s]"

    reactor_price = (40000000.0+5.1e6*sqrt(@volume))*@n
    reactor_cost = reactor_price/(5*330*24*3600) # [yen s-1]
    puts "reactor: #{reactor_cost.round(3)} [yen/s]"

    coolant_wt = @total_heat/(WATER_SPECIFIC_HEAT*WATER_TEMP_RISE)/1000 # [ton s-1]
    coolant_cost = coolant_price(@coolant)*coolant_wt # [yen s-1]
    puts "coolant: #{coolant_cost.round(3)} [yen/s]"

    total_cost = electricity_cost+steam_cost+reactor_cost+coolant_cost
    puts "total: #{total_cost.round(3)} [yen/s]"
  end

  def coolant_price(t)
    #TODO: hard coding. it should be rewrittend
    t = t-273
    if t>30
      return nil
    elsif t>10
      return 15+(45-15)*((30-t)*1.0/(30-10))
    elsif t>0
      return 45+(65-45)*((10-t)*1.0/(10-0))
    elsif t>-10
```

```
      return 65+(90-65)*((0-t)*1.0/(0+10))
    elsif t>=-20
      return 90+(140-90)*((-10-t)*1.0/(-10+20))
    else
      return nil
    end
  end

  def reactor(gamma_in, gamma_out, power)
    # heat transfer rate
    heat_of_reaction = HEAT_OF_POLY*(@rate*DENSITY*(gamma_in-gamma_out)/3.6)/BUTADIENE_M
    heat = heat_of_reaction + power
    h = heat/@surface/(REACTION_TEMP-@coolant)

    # viscosity [Pa s]
    viscosity = ((POLYMER_LENGTH)**1.7)*((1-(gamma_out/@feed))**2.5)*exp(21.0*@feed)*1e-3

    # dimensionless numbers
    pr = viscosity*TOLUENE_SPECIFIC_HEAT/THERMAL_CONDUCTIVITY
    nu = h*@diameter/THERMAL_CONDUCTIVITY
    re = (2*nu/pr**(1/3.0))**1.5

    # power consumption
    revolution = re*viscosity/DENSITY/(@diameter/2)**2
    np = 14.6*re**(-0.28)
    power_new = np*DENSITY*(revolution**3)*(@diameter/2)**5

    if (power_new - power).abs < ACCURACY
      return {
        re: re,
        revolution: revolution,
        power: power_new,
        heat: heat
      }
    else
      return reactor(gamma_in, gamma_out, power_new)
    end
  rescue SystemStackError
    # when power consumption go infinity, return nil
    return nil
  end
end

puts "input n[-], gamma_0[wt%], T2[K]"
n = gets.to_i
feed = gets.to_f
coolant = gets.to_i

plant = Plant.new(n, feed, coolant)
plant.calc()
# plant.show()
plant.calc_cost()
```