Tarea 1.2 | Cálculo de altura de relleno, unidades de transferencia en destilación

Esteban Richmond-Salazar

25/3/2019

Contents

1 Problema N° 1

Valor: 18 puntos

Referencia: Adaptado de Problema 15.D4 Wankat (2008)

Una columna de destilación separa una alimentación formada por 40 % metanol y 60 % agua (fracciones molares). La alimentación es de dos fases con 60 % líquido. El producto destilado debe ser 92 % metanol y los fondos 4 % metanol. Se emplea un rehervidor PARCIAL y un condensador total. El reflujo es líquido saturado. La operación es a 101.3 kPa. Suponga flujo equimolar y emplee $L/D = 1.1$. Bajo estas condiciones $H_{ty} = 0.396$ m y $H_{tx} = 0.244$ m para ambas secciones de enriquecimiento y agotamiento. Determine la altura requerida de ambas secciones empleando el método de las unidades de transferencia para la fase líquida.

```
# Datos de EVL para el sistema metanol-agua a 1 atm, fracciones \
    molares (Fuente: Tabla 2-7 Wankat 2a ed.)
datos_x   = [0, 0.02, 0.04, 0.06, 0.08, 0.10, 0.15, 0.20, \
    0.30, 0.40, 0.50, 0.60, 0.70, 0.80, 0.90, 0.95, 1]
datos_y   = [0, 0.134, 0.23, 0.304, 0.365, 0.418, 0.517, 0.579, \
    0.665, 0.729, 0.779, 0.825, 0.870, 0.915, 0.958, 0.979, 1]

# Datos del enunciado
z_F       =   0.40          # mol/mol
q         =   0.60          # mol/mol
x_D       =   0.92          # mol/mol
x_B       =   0.04          # mol/mol
R_D       =   1.1
```

```
H_ty_a     =    0.396          # m
H_tx_a     =    0.244          # m
H_ty_e     =    0.396          # m
H_tx_e     =    0.244          # m

# ***** Número de puntos para resolver las integrales, mínimo 5 \
    *****
n_puntos_agot = 5
n_puntos_enri = 5

# ================== Solución ==================
html('<h2>Solución</h2>')

datos_eq  =  zip(datos_x,datos_y)     # Crea una lista de pares de \
    datos x,y

# Se definen algunas variables por utilizar (cuando se requiere \
    resolver ecuaciones por ejemplo)
%var x, y, B, D, xI, yI, a0, a1, a2, a3, a4

# Ajuste de datos de equilibrio a un modelo
modelo(x) =  a1 * x + a2*x^2 + a3 *x^a4         # Modelo arbitrario \
    que ajusta bien
ajuste    =  find_fit(datos_eq, modelo, solution_dict=True)
y_eq(x)   =  modelo(a1 = ajuste[a1], a2 = ajuste[a2], a3 = ajuste[a3\
    ], a4 = ajuste[a4])

# Balances de masa
F         = 1000 # kmol/h, Base de cálculo

BM        =  solve([    F ==     D +     B,  \
                  z_F*F == x_D*D + x_B*B], \
                  B, D, solution_dict = True)

B         =  numerical_approx(BM[0][B], digits=4)
D         =  numerical_approx(BM[0][D], digits=4)
R_B       =  (q*F + R_D*D - B) / B

print "F   =", F, "kmol/h, Base de cálculo"
print "B   =", B, "kmol/h"
print "D   =", D, "kmol/h"
print "R_B =", R_B


# Ecuaciones de las líneas de operación
y_q(x)    =  q/(q-1)*x - z_F/(q-1)
y_e(x)    =  R_D/(R_D+1) * x + x_D/(R_D+1)
```

```
y_a(x)     =  (R_B+1)/R_B * x - x_B/R_B

x_ali      =  find_root(y_q == y_e,  0, 1)  # Cruce línea q con \
   enriquecimiento
x_aeq      =  find_root(y_q == y_eq, 0, 1)  # Cruce con equilibrio

# Composiciones de entrada y salida
y_a_entra =  y_eq(x_B)
y_a_sale  =  y_q(x_ali)
x_a_entra =  x_ali
x_a_sale  =  numerical_approx(solve(y_a_entra == y_a, x)[0].rhs(), \
   digits = 4)

y_e_entra =  y_a_sale
y_e_sale  =  y_e(x_D)
x_e_entra =  x_D
x_e_sale  =  x_ali

print "\nEn zona de agotamiento"
print "----------------------"
print "x_entra = %.3g" %x_a_entra
print "x_sale  = %.3g" %x_a_sale
print "y_entra = %.3g" %y_a_entra
print "y_sale  = %.3g" %y_a_sale

print "\nEn zona de enriquecimiento"
print "--------------------------"
print "x_entra = %.3g" %x_e_entra
print "x_sale  = %.3g" %x_e_sale
print "y_entra = %.3g" %y_e_entra
print "y_sale  = %.3g" %y_e_sale


# Gráficas de puntos de datos y las diferentes líneas de operación
graf_data =  scatter_plot(datos_eq, xmin=0, xmax = 1, ymin = 0, ymax\
    = 1, axes_labels = ["$x_{metanol}$", "$y_{metanol}$"])
graf_li45 =  plot(x   ,  x,  0    ,  1    , color ="black")
graf_equi =  plot(y_eq,  x,  0    ,  1    , color ="orange"      , \
   legend_label = "Equilibrio")
graf_alim =  plot(y_q ,  x,  x_aeq,  z_F  , color = "red"        , \
   legend_label = "Alim.")
graf_enri =  plot(y_e ,  x,  x_ali,  x_D  , color = "greenyellow", \
   legend_label = "Enriq.")
graf_agot =  plot(y_a ,  x,  x_B  ,  x_ali, color = "green"      , \
   legend_label = "Agot." )

# Gráfica de las composiciones alrededor del rehervidor
```

```
graf_rehe =  line([(x_B, x_B), (x_B, y_a_entra), (x_a_sale, \
    y_a_entra)])

graf_prob =  graf_data + graf_li45 + graf_equi + graf_alim + \
    graf_enri + graf_agot + graf_rehe

# Cálculo de condiciones de interfase
kx_div_ky_enri = numerical_approx(H_ty_e/H_tx_e * R_D/(R_D+1), \
    digits = 4)
kx_div_ky_agot = numerical_approx(H_ty_a/H_tx_a * (R_B+1)/R_B, \
    digits = 4)


show(r"$\left( \frac{k_x}{k_y} \right)_{enri} = \frac{H_{ty_e}}{H_{\
    tx_e}} \frac{R_D}{R_D+1} =$",kx_div_ky_enri)
show(r"$\left( \frac{k_x}{k_y} \right)_{agot} = \frac{H_{ty_a}}{H_{\
    tx_a}} \frac{R_B+1}{R_B} =$",kx_div_ky_agot)

# ***** Genera valores de datos a lo largo de las composiciones en \
    entrada y salida de la zona de agotamiento
Delta_xa    = (x_a_entra - x_a_sale) / (n_puntos_agot - 1)
lista_xa    = [x_a_sale + Delta_xa * i for i in range(n_puntos_agot)]
lista_ya    = [y_a(xop) for xop in lista_xa]

lista_xIa = []    # Lista vacía
lista_yIa = []

# Para cada par (x, y) calcula (xI, yI)
for i in range(n_puntos_agot):
    xop = lista_xa[i]    # Lee un valor de x de la línea de operació\
    n
    yop = lista_ya[i]    # Lee el correspondiente valor de y

    # Encuentra el valor de xI que iguala la ecuación de la curva de\
     equilibrio con la recta de pendiente -kx/ky
    f_resolver = y_eq(xI) == -kx_div_ky_agot * xI + kx_div_ky_agot *\
    xop + yop
    raiz_xI    = find_root(f_resolver, 0, 1)

    # Calcula el valor de yI = y*(xI) y guarda los datos en las \
    listas
    raiz_yI  = y_eq(raiz_xI)
    lista_xIa.append(raiz_xI)
    lista_yIa.append(raiz_yI)

# Genera la gráfica de líneas con pendiente -kx/ky
graf_lin = line([(lista_xa[0], lista_ya[0]), (lista_xIa[0], \
```

```
    lista_yIa[0])], color = 'gray')
for i in range(n_puntos_agot-1):
    graf_lin = graf_lin + line([(lista_xa[i+1], lista_ya[i+1]), (\
    lista_xIa[i+1], lista_yIa[i+1])], color = 'gray')

# ========== Cálculo del número de unidades de transferencia fase lí\
    quida (como pide el enunciado) ==========
# Genera lista con valores 1/(x-xI) y luego forma pares con los \
    valores de x
f          = [1/(lista_xa[i]-lista_xIa[i]) for i in range(\
    n_puntos_agot)]
datos_f    = zip(lista_xa, f)

# Ajuste de los datos a un modelo para integrar fácilmente
datos_x_xI = zip(lista_xa, lista_xIa)
modelo(x)  = a0 + a1*x + a2*x^2 + a3*x^a4
ajuste     = find_fit(datos_x_xI, modelo, solution_dict=True)
xI_f(x)    = modelo(a0=ajuste[a0], a1=ajuste[a1], a2=ajuste[a2], a3=\
    ajuste[a3], a4=ajuste[a4])

# Calcula NTU
f_int(x)   = 1/(x-xI_f(x))
integral_f = numerical_integral(f_int, x_a_sale, x_a_entra)
N_tx_a     = integral_f[0]          # La función numerical_integral \
    devuelve dos valores: la integral y el error estimado

# Cálculo de la altura del relleno de la sección de agotamiento
H_c_a      = numerical_approx(H_tx_a*N_tx_a, digits = 3)

# Gráfica del NTU
# P = Polyhedron(vertices=[(x_a_sale,0)] + datos_f + [(x_a_entra,0)\
    ], base_ring=RDF).center()  # Centro del polihedro
graf_NTU   = scatter_plot(datos_f, axes_labels = ['$x$', r'$\frac\
    {1}{x-x_I}$'], facecolor = 'none') \
    + plot(f_int, xmin = x_a_sale, xmax = x_a_entra, color = 'green'\
    , fill = 'axis', fillcolor='green') \
    + text('$ABC = %s$' %n(N_tx_a, digits=3), ((x_a_entra+x_a_sale)\
    /2, median(datos_f)[1]/2), color = 'black')


# ***** Igual que todo lo anterior pero para la zona de \
    enriquecimiento
Delta_xe   = (x_e_entra - x_e_sale) / (n_puntos_enri - 1)
lista_xe   = [x_e_sale + Delta_xe * i for i in range(n_puntos_enri)]
lista_ye   = [y_e(xop) for xop in lista_xe]

lista_xIe  = []
```

```
lista_yIe  = []

for i in range(n_puntos_enri):
    xop    = lista_xe[i]
    yop    = lista_ye[i]

    f_resolver = y_eq(xI) == -kx_div_ky_enri * xI + kx_div_ky_enri *\
    xop + yop
    raiz_xI    = find_root(f_resolver, 0, 1)
    raiz_yI    = y_eq(raiz_xI)
    lista_xIe.append(raiz_xI)
    lista_yIe.append(raiz_yI)

for i in range(n_puntos_enri):
    graf_lin = graf_lin + line([(lista_xe[i], lista_ye[i]), (\
   lista_xIe[i], lista_yIe[i])], color = 'darkgray')

g          = [1/(lista_xe[i]-lista_xIe[i]) for i in range(\
   n_puntos_enri)]
datos_g    = zip(lista_xe, g)

datos_x_xI = zip(lista_xe, lista_xIe)
modelo(x)  = a0 + a1*x + a2*x^2 + a3*x^a4
ajuste     = find_fit(datos_x_xI, modelo, solution_dict=True)
xI_g(x)    = modelo(a0=ajuste[a0], a1=ajuste[a1], a2=ajuste[a2], a3=\
   ajuste[a3], a4=ajuste[a4])

g_int(x)   = 1/(x-xI_g)
integral_g = numerical_integral(g_int, x_e_sale, x_e_entra)
N_tx_e     = integral_g[0]

H_c_e      = numerical_approx(H_tx_e*N_tx_e, digits = 3)

graf_NTU = graf_NTU + scatter_plot(datos_g, axes_labels = ['$x$', r'\
   $\frac{1}{x-x_I}$'], facecolor = 'none') \
    + plot(g_int, xmin = x_e_sale, xmax = x_e_entra, color = '\
   greenyellow', fill = 'axis', fillcolor='greenyellow') \
    + text('$ABC = %s$' %n(N_tx_e, digits=3), ((x_e_entra+x_e_sale)\
   /2, median(datos_g)[1]/2), color = 'black')

# Muestra los resultados
show(graf_prob + graf_lin)
show(graf_NTU)

show(r"$H_{c,\text{agot}} = (H_{tL} N_{tL})_\text{agot} = %s \,\text\
   {m} * %s = %s \,\text{m}$" %(numerical_approx(H_tx_a, digits = 3)\
   , numerical_approx(N_tx_a, digits = 3), H_c_a))
```

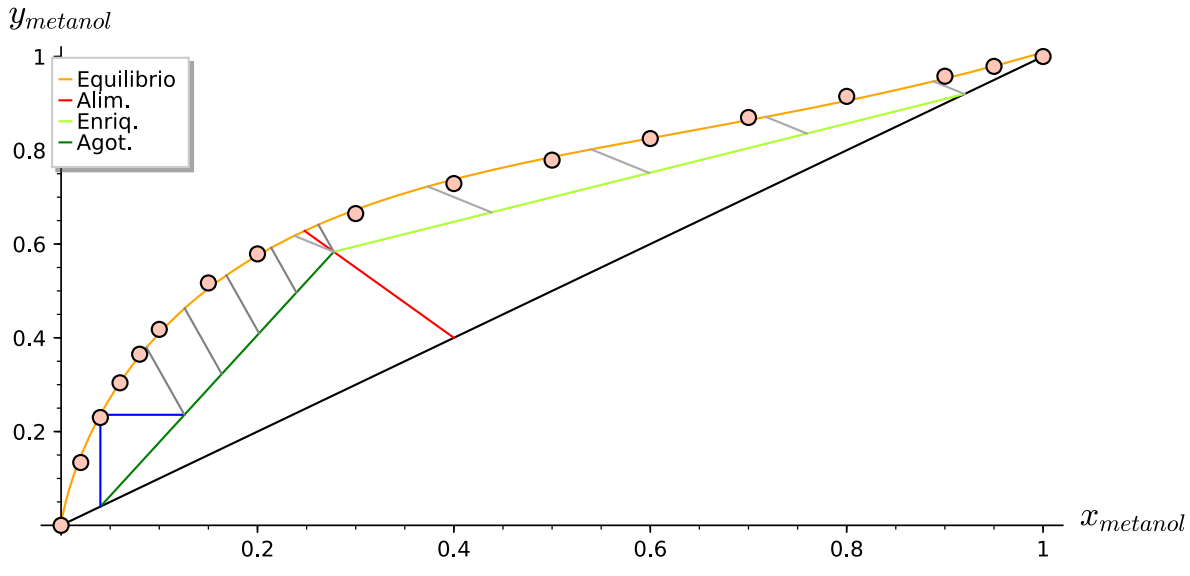
```
show(r"$H_{c,\text{enri}} = (H_{tL} N_{tL})_\text{enri} = %s \,\text\
    {m} * %s = %s \,\text{m}$" %(numerical_approx(H_tx_e, digits = 3)\
    , numerical_approx(N_tx_e, digits = 3), H_c_e))

# ***** Final del código *****
```

1.1 Solución

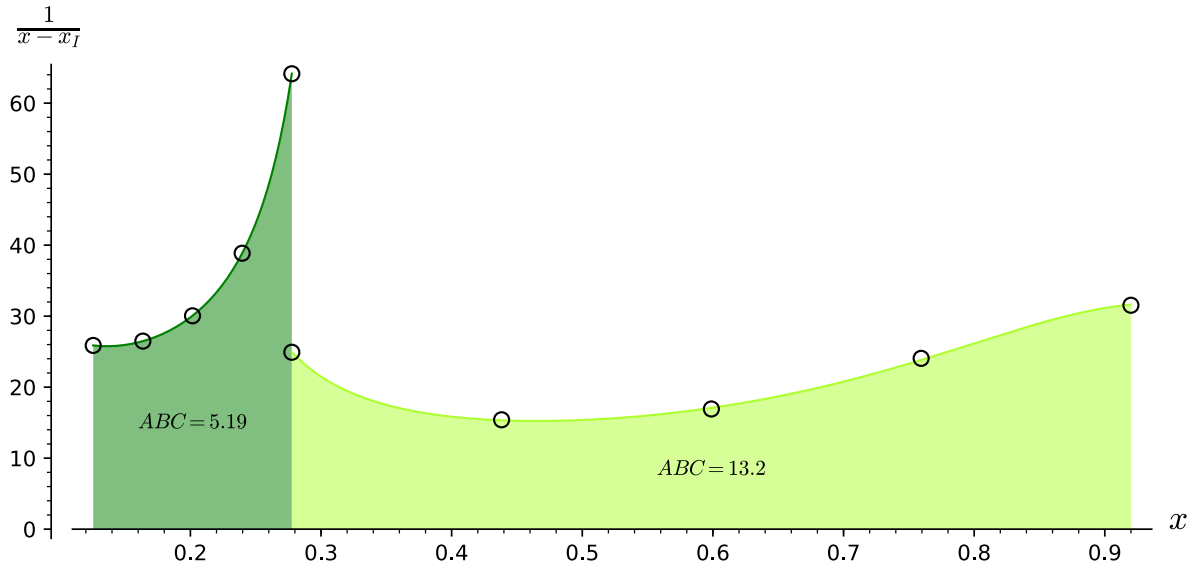
F = 1000 kmol/h, Base de cálculo
B = 590.9 kmol/h
D = 409.1 kmol/h
R_B = 0.7769
En zona de agotamiento
---
x_entra = 0.278
x_sale = 0.126
y_entra = 0.236
y_sale = 0.584
En zona de enriquecimiento
---
x_entra = 0.92
x_sale = 0.278
y_entra = 0.584
y_sale = 0.92

$\left(\frac{k_x}{k_y}\right)_{enri} = \frac{H_{tye}}{H_{txe}} \frac{R_D}{R_D+1} = 0.8501$

$\left(\frac{k_x}{k_y}\right)_{agot} = \frac{H_{tya}}{H_{txa}} \frac{R_B+1}{R_B} =$



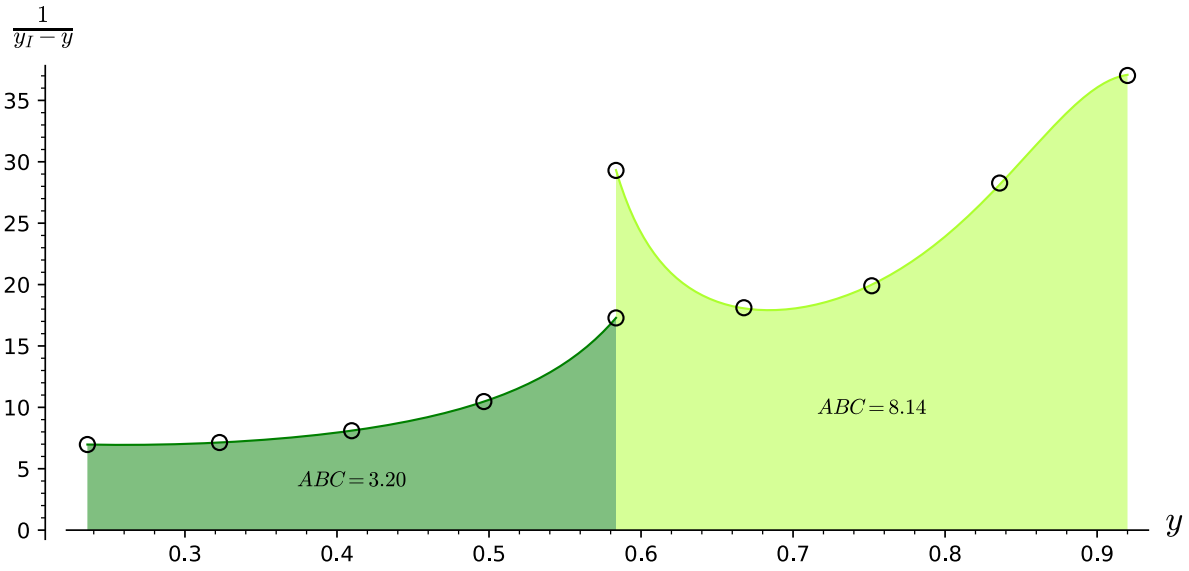
3.712



$H_{c,\text{agot}} = (H_{tL} N_{tL})_{\text{agot}} = 0.244\,\text{m} * 5.19 = 1.27\,\text{m}$
$H_{c,\text{enri}} = (H_{tL} N_{tL})_{\text{enri}} = 0.244\,\text{m} * 13.2 = 3.22\,\text{m}$

```
# ========== Cálculo del número de unidades de transferencia para la\
    fase de vapor (SOLO POR COMPARAR) ==========

# Se toman los mismos datos de composiciones en la interfase ya \
    calculados anteriormente
# Genera lista con valores 1/(yI-y) y luego forma pares con los \
    valores de y
f           = [1/(lista_yIa[i]-lista_ya[i]) for i in range(\
    n_puntos_agot)]
datos_f     = zip(lista_ya, f)

# Ajuste de los datos a un modelo para integrar fácilmente
datos_y_yI = zip(lista_ya, lista_yIa)
modelo(y)   = a0 + a1*y + a2*y^2 + a3*y^a4
ajuste      = find_fit(datos_y_yI, modelo, solution_dict=True)
yI_f(y)     = modelo(a0=ajuste[a0], a1=ajuste[a1], a2=ajuste[a2], a3=\
    ajuste[a3], a4=ajuste[a4])

# Calcula NTU
f_int(y)    = 1/(yI_f(y)-y)
integral_f = numerical_integral(f_int, y_a_entra, y_a_sale)
N_ty_a      = integral_f[0]          # La función numerical_integral \
    devuelve dos valores: la integral y el error estimado

# Cálculo de la altura del relleno
H_c_a       = numerical_approx(H_ty_a*N_ty_a, digits = 3)

# Gráfica
```

```
graf_NTU   = scatter_plot(datos_f, axes_labels = ['$y$', r'$\frac\
    {1}{y_I-y}$'], facecolor = 'none') \
     + plot(f_int, xmin = y_a_entra, xmax = y_a_sale, color = 'green'\
    , fill = 'axis', fillcolor='green') \
     + text('$ABC = %s$' %n(N_ty_a, digits=3), ((y_a_entra+y_a_sale)\
    /2, median(datos_f)[1]/2), color = 'black')

# ***** Igual que todo lo anterior pero para la zona de \
    enriquecimiento
f          = [1/(lista_yIe[i]-lista_ye[i]) for i in range(\
    n_puntos_enri)]
datos_f    = zip(lista_ye, f)

datos_y_yI = zip(lista_ye, lista_yIe)
modelo(y)  = a0 + a1*y + a2*y^2 + a3*y^a4
ajuste     = find_fit(datos_y_yI, modelo, solution_dict=True)
yI_f(y)    = modelo(a0=ajuste[a0], a1=ajuste[a1], a2=ajuste[a2], a3=\
    ajuste[a3], a4=ajuste[a4])

f_int(y)   = 1/(yI_f-y)
integral_f = numerical_integral(f_int, y_e_entra, y_e_sale)
N_ty_e     = integral_f[0]

H_c_e      = numerical_approx(H_ty_e*N_ty_e, digits = 3)

graf_NTU = graf_NTU + scatter_plot(datos_f, axes_labels = ['$y$', r'\
    $\frac{1}{y_I-y}$'], facecolor = 'none') \
     + plot(f_int, xmin = y_e_entra, xmax = y_e_sale, color = '\
    greenyellow', fill = 'axis', fillcolor='greenyellow') \
     + text('$ABC = %s$' %n(N_ty_e, digits=3), ((y_e_entra+y_e_sale)\
    /2, median(datos_f)[1]/2), color = 'black')

# Muestra los resultados
show(graf_NTU)

show(r"$H_{c,\text{agot}} = (H_{tG} N_{tG})_\text{agot} = %s \,\text\
    {m} * %s = %s \,\text{m}$" %(numerical_approx(H_ty_a, digits = 3)\
    , numerical_approx(N_ty_a, digits = 3), H_c_a))
show(r"$H_{c,\text{enri}} = (H_{tG} N_{tG})_\text{enri} = %s \,\text\
    {m} * %s = %s \,\text{m}$" %(numerical_approx(H_ty_e, digits = 3)\
    , numerical_approx(N_ty_e, digits = 3), H_c_e))
```



$H_{c,\text{agot}} = (H_{tG}N_{tG})_{\text{agot}} = 0.396\,\text{m} * 3.20 = 1.27\,\text{m}$
$H_{c,\text{enri}} = (H_{tG}N_{tG})_{\text{enri}} = 0.396\,\text{m} * 8.14 = 3.22\,\text{m}$

2 Problema N° 2

Valor: 7 puntos

Referencia: Adaptado de Problema 15.D6 Wankat (2008)

Una columna de destilación trabaja a reflujo total, separando una mezcla binaria cuya volatilidad relativa promedio es 2,3. Se obtienen fracciones molares del componente más liviano y_{sale} = 0,95 y y_{entra} = 0,05.

1. Si hay 7,5 m de relleno, determine H_{tOG} promedio, para ello considere la siguiente expresión del N_{tOG}.
   - $N_{tOG} = \frac{1}{1-\alpha} \ln\left(\frac{y_{entra}(1-y_{sale})}{y_{sale}(1-y_{entra})}\right) + \ln\left(\frac{1-y_{entra}}{1-y_{sale}}\right)$
2. Compruebe los resultados de la parte **a** mediante el cálculo gráfico usando un diagrama de McCabe-Thiele.
   - $N_{tOG} = \int_{y_{entra}}^{y_{sale}} \frac{1}{y^* - y}\, dy$
   - Para generar la curva de equilibrio se emplea la ecuación: $y^* = \frac{\alpha x}{1+(\alpha-1)x}$

```
html('<h2>Solución</h2>')

%var x

# Datos del enunciado
alfa     =     2.3
y_entra  =     0.05     # mol/mol
y_sale   =     0.95     # mol/mol
```

```
h_c         =     7.5         # m

y_eq(x)     =  alfa*x/(1+(alfa-1)*x)
y(x)        =  x

print "Parte a)"

N_tOG       =  1/(1-alfa) * ln((y_entra*(1-y_sale))/(y_sale*(1-y_entra\
    )))+ln((1-y_entra)/(1-y_sale))
H_tOG       =  h_c / N_tOG

# Muestra el resultado formateado, lo que está entre símbolos de dó\
    lar se mostrará en formato matemático, para ello se escribe en \
    LaTeX
show (r'$N_{tOG} = \frac{1}{1-\alpha } \ln \left ( \frac{y_{entra} (\
     1-y_{sale} )} {y_{sale} (1-y_{entra} )} \right ) + \ln \left (\\
    frac{1-y_{entra}}{1-y_{sale}} \right ) = %s$' %n(N_tOG, digits=4)\
    )
show (r'$H_{tOG} = h_c / N_{tOG} = %s$ m' %n(H_tOG, digits=3))

print "Parte b)"

plot(y_eq, x, 0, 1, thickness=2, axes_labels=['$x$','$y$'], title = \
    'Diagrama de McCabe-Thiele', fontsize=14) \
     + plot(y, x, 0, 1, color = 'black') \
     + line([(0, y_entra), (y_entra, y_entra)]) \
     + text('$y_{entra}$', (-0.04, y_entra), fontsize=16) \
     + line([(0, y_sale),(y_sale, y_sale)]) \
     + text('$y_{sale}$', (-0.04, y_sale), fontsize=16)

f           =  1/(y_eq-y)
N_tOG       =  integral_numerical(f,y_entra, y_sale)[0]
H_tOG       =  h_c / N_tOG  # m

plot(f, y_entra, y_sale, axes_labels=['$y$','$\\frac{1}{(y^*-y)}$'],\
     fill = 'axis', fontsize=14) \
     + text('$y_{entra}$', (y_entra, -2), fontsize=16) \
     + text('$y_{sale}$', (y_sale, -2), fontsize=16) \
     + text('$N_{tOG} = %s$' %n(N_tOG, digits=4), (0.5,25), fontsize\
    =16) \
     + text('$H_{tOG} = %s$ m' %n(H_tOG, digits=3), (0.5,20), \
    fontsize=16)

# ***** Final del código *****
```

2.1 Solución

Parte a)

$N_{tOG} = \frac{1}{1-\alpha} \ln\left(\frac{y_{entra}(1-y_{sale})}{y_{sale}(1-y_{entra})}\right) + \ln\left(\frac{1-y_{entra}}{1-y_{sale}}\right) = 7.474$

$H_{tOG} = h_c / N_{tOG} = 1.00$ m

Parte b)