CE 5364 Groundwater Transport Phenemona
Fall 2025 Exercise Set 6

LAST NAME, FIRST NAME

R00000000

Purpose :

Apply selected analytical models for reactive transport

Assessment Criteria :

Completion, results plausible, format correct, example calculations shown.

Problem 1 (Problem 6-7, pg. 588)

An instantaneous release of biodegradable constituients occurs in a 1-D aquifer. Assume the mass released is $1.0\ kg$ over a $10\ m^2$ area normal to the flow direction, $\alpha_l = 1.0\ m$, the seepage velocity is 1.0 $\frac{m}{day}$, and the half-life of the decaying constituient is 33 years.

Determine:

1. The maximum concentration at 100 meters from the source.
2. Plot a concentration history (annual intervals) for a 40 year period from release date for a location 100 meters from the source.

A. Known quantities

initial mass = 1 kg.
cross section area = 10 m^2
$\alpha_L = 1.0\ m$
$v_x = 1\ \frac{m}{d}$
$\lambda = \frac{t_{1/2}}{0.693}$

B. List unknown quantities

$C(100,t)_{max}$
Plot $C(100,t)$ for $t = 1, 2, \ldots, 40\ yr$

C. Governing principles

- Finite mass implies instant release. Use equation 6.18 as modified on g. 175 in book

Using Eq. (6.18) and incorporating radioactive decay,

$$C(x,t) = \left\{\frac{\mu}{\sqrt{4\pi D_x t}} \exp\left[-\frac{(x - v_x t)^2}{4 D_x t}\right]\right\} \exp(-\lambda t)$$

D. Solution details (e.g. step-by-step computations)

1. Build a prototype function

In [1]:
```
def c1addinst(distance,time,mass,dispersion,velocity,decay):
    import math
    term0 = math.exp(-1.0*decay*time)
    term1 = math.sqrt(4.0*math.pi*dispersion*time)
    term2 = math.exp(-((distance-velocity*time)**2)/(4.0*dispersion*time))
    c1addinst = term0*(mass/term1)*term2
    return(c1addinst)
```

2. Build input data manager, report intermediate computations

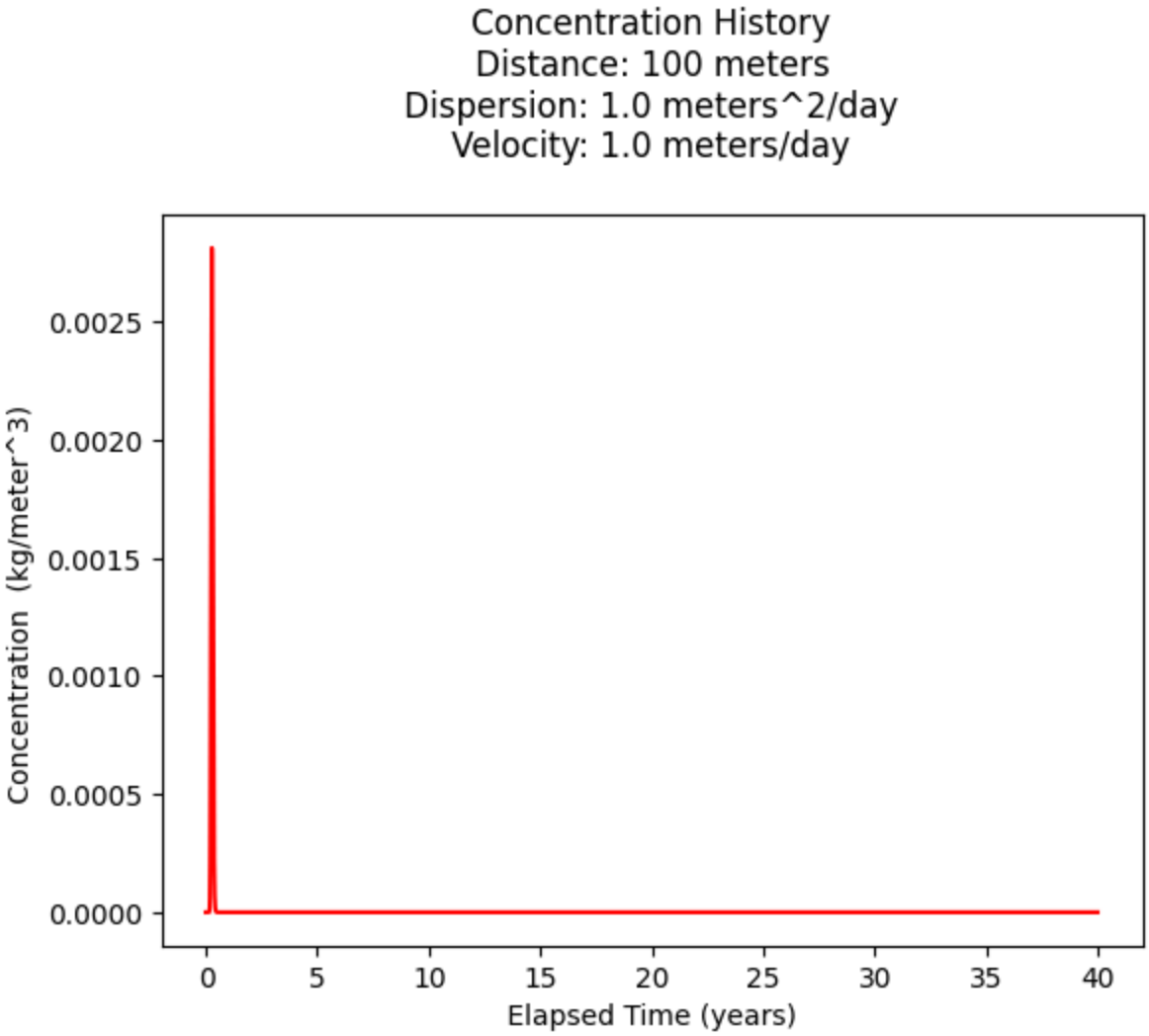
In [2]:
```
import math
total_mass  = 1.0
area        = 10
mass        = total_mass/area
velocity    = 1.0 #m/day
dispersivity = 1.0 #m
dispersion  = velocity*dispersivity #m^2/day
half_life   = 33 #years
decay = math.log(2)/(half_life)/365 #1/days
print("Mass : ",round(mass,3)," kg/m^3")
print("Decay constant : ",round(decay,6)," day^-1 ")
print("Dispersion : ",round(dispersion,3)," m^2/day")
#print(math.log(2))
```

```
Mass :  0.1  kg/m^3
Decay constant :  5.8e-05  day^-1
Dispersion :  1.0  m^2/day
```

3. Plot a concentration history over 40 years

In [3]:
```
deltat      = (1.0) #days
howmany =     365*40/deltat
howmany = int(howmany)
```

```
t = [] #days
for i in range(howmany):
    t.append(float(i)*deltat)
    if t[i] == 0: # trap zero time to prevent divide by zero
        t[i]= 0.00000001

distance        = 100  #years as days

c = [0 for i in range(howmany)] #concentration

for i in range(howmany):
    c[i]=c1addinst(distance,t[i],mass,dispersion,velocity,decay)

# rescale time into years
for i in range(howmany):
    t[i]=t[i]/365.
#
# Import graphics routines for picture making
#
from matplotlib import pyplot as plt
#
# Build and Render the Plot
#
plt.plot(t,c, color='red', linestyle = 'solid')  # make the plot object
plt.title(" Concentration History \n Distance: " + repr(distance) + " meters \n" +
plt.xlabel(" Elapsed Time (years) ") # label x-axis
plt.ylabel(" Concentration  (kg/meter^3)      ") # label y-axis
#plt.plot([365,365],[0,c0])
#plt.plot([365*2,365*2],[0,c0])
#plt.text(365,100,"  year 1")
#plt.text(365*2,100,"  year 2")
#plt.savefig("ogatabanksplot.png") # optional generates just a plot for embedding i
plt.show() # plot to stdio -- has to be last call as it kills prior objects
plt.close('all') # needed when plt.show call not invoked, optional here
#sys.exit() # used to elegant exit for CGI-BIN use
#print("Center of Distribution Position : ",round(time*velocity,2)," length units")
```

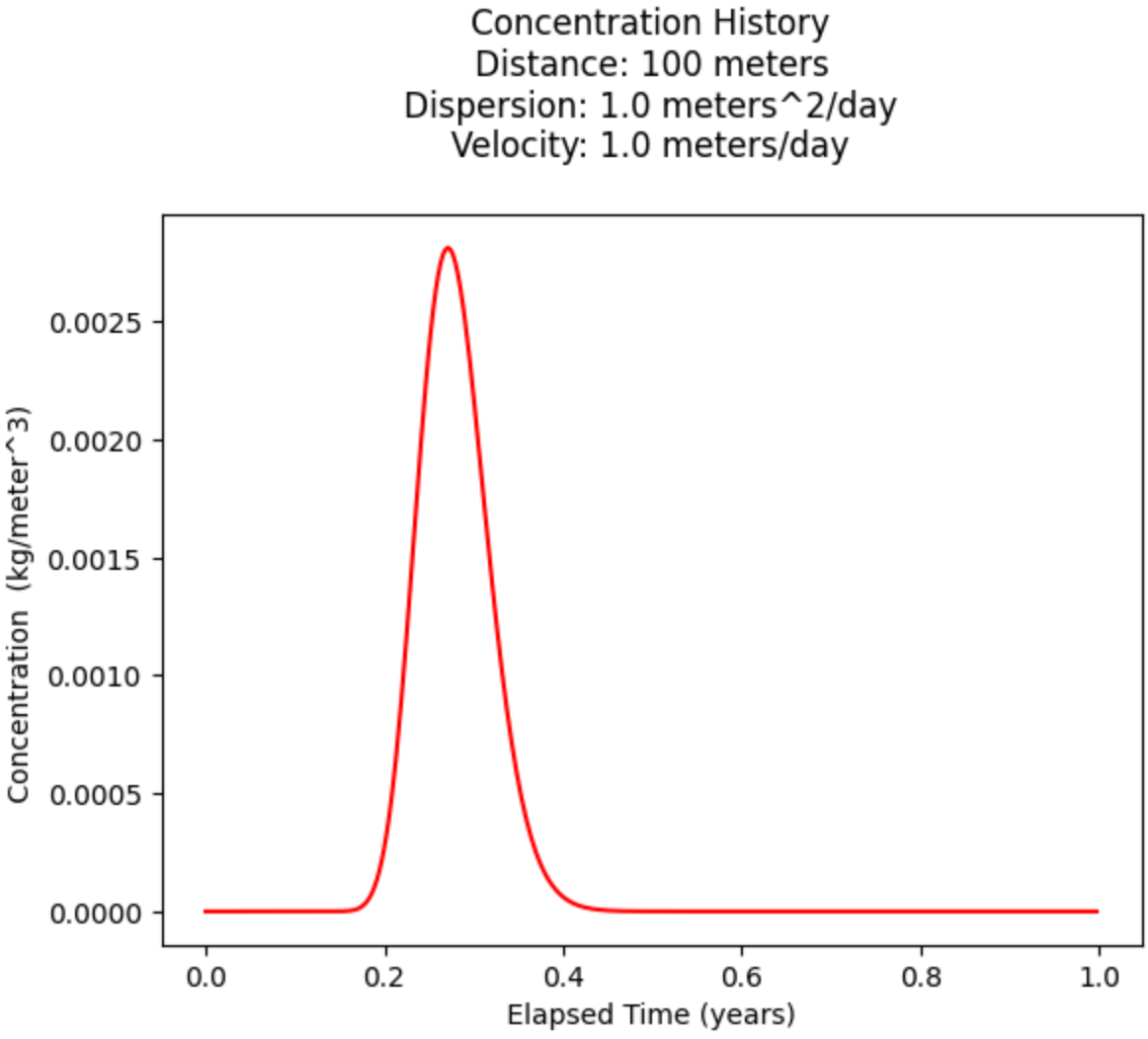


4. Plot a concentration history over one year (easier to see)

In [4]:

```
deltat      = (1.0) #days
howmany =     365*1/deltat
howmany = int(howmany)

t = [] #days
for i in range(howmany):
    t.append(float(i)*deltat)
    if t[i] == 0: # trap zero time to prevent divide by zero
        t[i]= 0.00000001

distance        = 100  #years as days

c = [0 for i in range(howmany)] #concentration

for i in range(howmany):
    c[i]=c1addinst(distance,t[i],mass,dispersion,velocity,decay)

# rescale time into years
for i in range(howmany):
    t[i]=t[i]/365.
```

```
#
# Import graphics routines for picture making
#
from matplotlib import pyplot as plt
#
# Build and Render the Plot
#
plt.plot(t,c, color='red', linestyle = 'solid')  # make the plot object
plt.title(" Concentration History \n Distance: " + repr(distance) + " meters \n" +
plt.xlabel(" Elapsed Time (years) ") # label x-axis
plt.ylabel(" Concentration  (kg/meter^3)      ") # label y-axis
#plt.plot([365,365],[0,c0])
#plt.plot([365*2,365*2],[0,c0])
#plt.text(365,100,"  year 1")
#plt.text(365*2,100,"  year 2")
#plt.savefig("ogatabanksplot.png") # optional generates just a plot for embedding i
plt.show() # plot to stdio -- has to be last call as it kills prior objects
plt.close('all') # needed when plt.show call not invoked, optional here
#sys.exit() # used to elegant exit for CGI-BIN use
#print("Center of Distribution Position : ",round(time*velocity,2)," length units")
```



5. Looks like maximum occurs at about 100 days, but lets just be stupid about it and find the value from the plot, and report the time of occurance (in days)

In [5]:
```
print("Maximum concentration : ",max(c)," kg/m^3")
print("          Observed at : ",t[c.index(max(c))]*365," days")
print(f"           C_max ~ {c1addinst(distance,t[c.index(max(c))]*365,mass,dispersio
```

```
Maximum concentration :  0.0028119432286784715  kg/m^3
          Observed at :  99.0  days
          C_max ~ 0.0028 kg/m^3
```

Problem 2

A batch isotherm test was performed with several 1-L solutions of the chemical of interest and one soil type, 20 g in each solution container. The initial and final solution concentrations are shown in the table. Fit the linear, Freundlich, and Langmuir isotherm equations to this data.

Initial Concentration (mg/L)	Equilibrium Concentration (mg/L)
7.10	6.71
4.53	4.18
1.89	1.63
1.31	1.10
1.03	0.85

Determine:

1. The Linear isotherm equation for these data (i.e. fit the isotherm model to the data), plot the isotherm and data
2. The Freundlich isotherm equation for these data, plot the isotherm and data
3. The Langmuir isotherm equation for these data, plot the isotherm and data
4. Which isotherm model produces the best fit for these data?

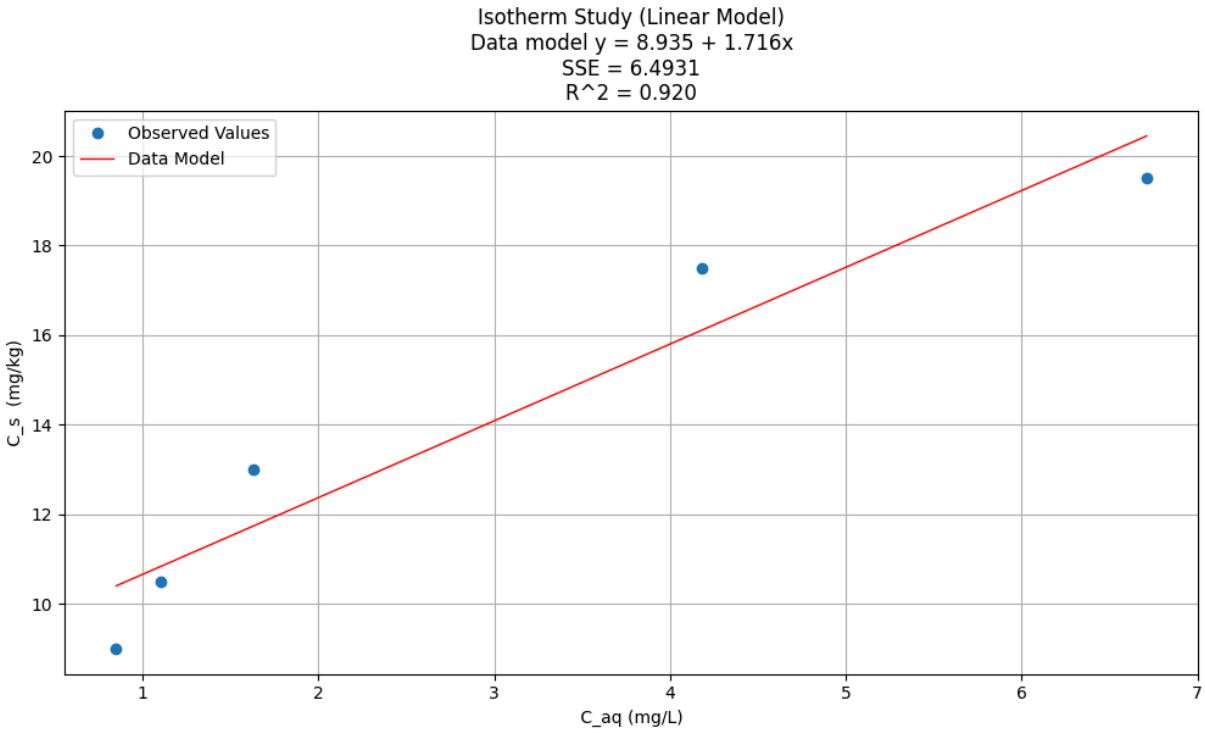
Show calculations and identify all fitted parameter values.

Using curve fitting methods from Theodore G. Cleveland, Farhang Forghanparast (2021), Computational Thinking and Data Science: Instructor's Notes for ENGR 1330 at TTU, with contributions by: Dinesh Sundaravadivelu Devarajan, Turgut Batuhan Baturalp (Batu), Tanja Karp, Long Nguyen, and Mona Rizvi. Whitacre College of Engineering, DOI (pending)

1. The Linear isotherm equation for these data (i.e. fit the isotherm model to the data), plot the isotherm and data

In [6]:
```
p5df = [[7.10 ,6.71],
[4.53 ,4.18],
[1.89 ,1.63],
[1.31 ,1.10],
[1.03 ,0.85]]
howmanyrows = len(p5df)# allocate lists
c0=[0 for i in range(howmanyrows)]
cEq=[0 for i in range(howmanyrows)]
cS=[0 for i in range(howmanyrows)]
cSoS=[0 for i in range(howmanyrows)]
#input values
massS = 0.020 #kg
volL = 1 #L
# build lists
for i in range(howmanyrows):
    c0[i]=p5df[i][0]
    cEq[i]=p5df[i][1]
    cS[i]=(c0[i]-cEq[i])*volL/massS
    cSoS[i]=cEq[i]/cS[i]

# Now we need to do some analysis to earn our keep
# heres how to do the fits using python
#Load the necessary packages
import numpy as np
import pandas as pd
import statistics
import statsmodels.formula.api as smf # here is the regression package to fit lines
data = pd.DataFrame({'X':cEq, 'Y':cS}) # we use X,Y as column names for simplicity
#data.head()

# Initialise and fit linear regression model using `statsmodels`
model = smf.ols('Y ~ X', data=data) # model object constructor syntax
model = model.fit()

# Predict values
y_pred = model.predict(data)

# Extract fitted coefficients
beta0 = model.params['Intercept']   # intercept
beta1 = model.params['X']           # slope

sse = model.ssr
rsq = model.rsquared



from matplotlib import pyplot as plt

titleline = (
    f"Isotherm Study (Linear Model)\n"
    f"Data model y = {beta0:.3f} + {beta1:.3f}x\n"
```

```
    f"SSE = {sse:.4f}\n"
    f"R^2 = {rsq:.3f}"
)

# Plot regression against actual data
plt.figure(figsize=(12, 6))
plt.plot(data['X'], data['Y'], 'o')           # scatter plot showing actual data
plt.plot(data['X'], y_pred, 'r', linewidth=1)   # regression line
plt.xlabel(" C_aq (mg/L) ") # label x-axis
plt.ylabel(" C_s  (mg/kg) ") # label y-axis
plt.legend(['Observed Values','Data Model'])
plt.title(titleline)
plt.grid(which="both")
plt.show() # plot to stdio -- has to be last call as it kills prior objects
plt.close('all') # needed when plt.show call not invoked, optional here
#sys.exit() # used to elegant exit for CGI-BIN use
```



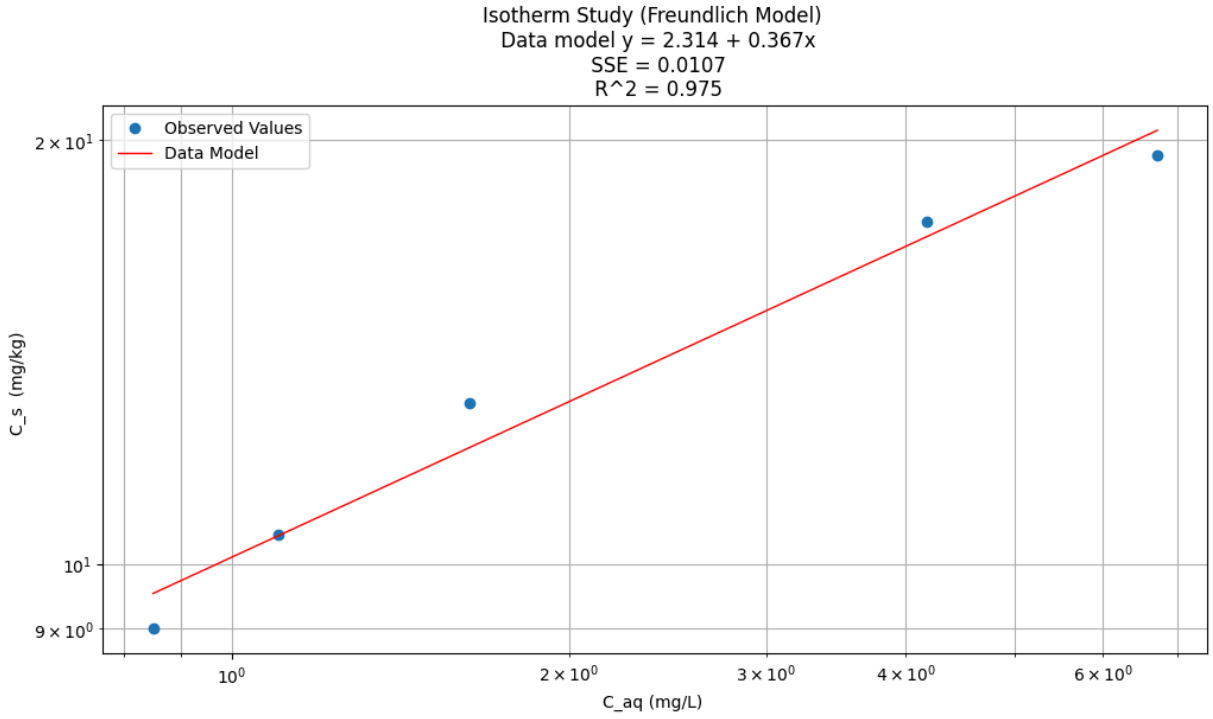
2. The Freundlich isotherm equation for these data, plot the isotherm and data

In [7]:
```
data = pd.DataFrame({'X':cEq, 'Y':cS}) # we use X,Y as column names for simplicity
import math
data['lnX']=data['X'].apply(math.log)
data['lnY']=data['Y'].apply(math.log)
#data.head()
```

In [8]:
```
# Initialise and fit linear regression model using `statsmodels`
model = smf.ols('lnY ~ lnX', data=data) # model object constructor syntax
model = model.fit()

# Predict values
y_pred = model.predict(data)
```

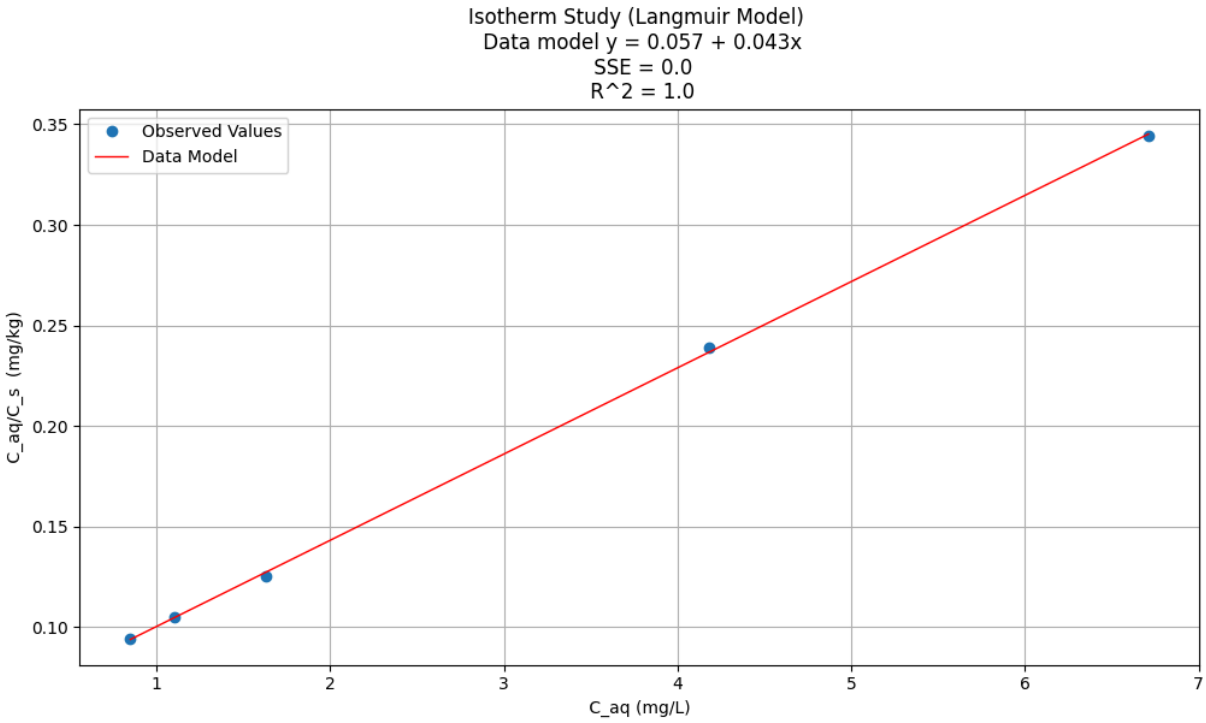
```
beta0 = model.params.iloc[0] # the fitted intercept
beta1 = model.params.iloc[1]
sse = model.ssr
rsq = model.rsquared

data['Ymod']=math.exp(beta0)*(data['X']**beta1)
print(data.head())

titleline = "Isotherm Study (Freundlich Model) \n Data model y = " + str(round(beta
titleline = titleline + '\n SSE = ' + str(round(sse,4)) + '\n R^2 = ' + str(round(r

# Plot regression against actual data
plt.figure(figsize=(12, 6))
plt.plot(data['X'], data['Y'], 'o')           # scatter plot showing actual data
plt.plot(data['X'],data['Ymod'], 'r', linewidth=1)   # regression line
plt.xlabel("C_aq (mg/L) ") # label x-axis
plt.ylabel("C_s  (mg/kg) ") # label y-axis
plt.yscale('log') # set y-axis to display a logarithmic scale ##################
plt.xscale('log') # set x-axis to display a logarithmic scale ##################
plt.legend(['Observed Values','Data Model'])
plt.title(titleline)
plt.grid(which="both")
plt.show() # plot to stdio -- has to be last call as it kills prior objects
```

	X	Y	lnX	lnY	Ymod
0	6.71	19.5	1.903599	2.970414	20.324788
1	4.18	17.5	1.430311	2.862201	17.086669
2	1.63	13.0	0.488580	2.564949	12.097387
3	1.10	10.5	0.095310	2.351375	10.472862
4	0.85	9.0	-0.162519	2.197225	9.528125



3. The Langmuir isotherm equation for these data, plot the isotherm and data

In [9]:
```
data = pd.DataFrame({'X':cEq, 'Y':cSoS}) # we use X,Y as column names for simplicit
#data.head()

# Initialise and fit linear regression model using `statsmodels`
model = smf.ols('Y ~ X', data=data) # model object constructor syntax
model = model.fit()

# Predict values
y_pred = model.predict(data)

beta0 = model.params.iloc[0] # the fitted intercept
beta1 = model.params.iloc[1]
sse = model.ssr
rsq = model.rsquared

titleline = "Isotherm Study (Langmuir Model) \n Data model y = " + str(round(beta0,
titleline = titleline + '\n SSE = ' + str(round(sse,4)) + '\n R^2 = ' + str(round(r

# Plot regression against actual data
plt.figure(figsize=(12, 6))
plt.plot(data['X'], data['Y'], 'o')           # scatter plot showing actual data
plt.plot(data['X'], y_pred, 'r', linewidth=1)   # regression line
plt.xlabel(" C_aq (mg/L) ") # label x-axis
plt.ylabel(" C_aq/C_s  (mg/kg) ") # label y-axis
plt.legend(['Observed Values','Data Model'])
plt.title(titleline)
plt.grid(which="both")
plt.show() # plot to stdio -- has to be last call as it kills prior objects
plt.close('all') # needed when plt.show call not invoked, optional here
#sys.exit() # used to elegant exit for CGI-BIN use
```



4. Which isotherm model produces the best fit for these data?

Using R^2 as selection criterion, Langmuir is bestest.

Problem 3

The following table has data from a column test with bromide (conservative) and chromium (sorbed). The porosity of the soil was 0.485, the bulk density was 1.85 g/cc, velocity was 0.244 cm/min, and the column was 25.4 cm long with a diameter of 2.54 cm.

Time (min)	Bromide $\frac{C}{Co}$	Chromium $\frac{C}{Co}$
0	0.000	0.000
15	0.000	0.000
30	0.005	0.000
45	0.003	0.000
60	0.013	0.000
75	0.075	0.000
90	0.137	0.000
105	0.530	0.000
120	0.841	0.000
135	1.000	0.000
150	1.000	0.000
165	1.000	0.009
180	1.000	0.186
195	1.000	0.595
210	1.000	0.791
225	1.000	0.875
240	1.000	0.913
255	1.000	0.946
270	1.000	0.946
285	1.000	1.000
300	1.000	1.000
315	1.000	1.000
330	1.000	1.000

Time (min)	Bromide $\frac{C}{Co}$	Chromium $\frac{C}{Co}$
345	1.000	1.000
360	1.000	1.000

Determine:

1. The dispersivity in cm
2. The retardation coefficient for Cr.

solution details (e.g. step-by-step computations)

1. Prototype function from class notes

In [10]:
```
#
# prototype decaying species function
#
def c1dadrd(c_source,space,time,dispersion,velocity,retardation,decay):
    from math import sqrt,erf,erfc,exp  # get special math functions
    dee = dispersion/retardation
    vee = velocity/retardation
    uuu = (vee**2 + 4.0*decay*dee)
    uuu = sqrt(uuu)
    arg1 = (space*(vee-uuu))/(2.0*dee)
    arg2 = (space - uuu*time)/(2.0*sqrt(dee*time))
    arg3 = (space*(vee+uuu))/(2.0*dee)
    arg4 = (space + uuu*time)/(2.0*sqrt(dee*time))
    temp1 = c_source/2.0
    temp2 = exp(arg1)
    temp3 = erfc(arg2)
    temp4 = exp(arg3)
    temp5 = erfc(arg4)

    c1dadrd = temp1*(temp2*temp3+temp4*temp5)

    return c1dadrd
```

1. copy the data from the table above

- cut-n-paste, then
- insert delimiters
- Parse into useable lists

In [11]:
```
# copy the data from the table above (cut-n-paste, then insert delimiters)
p4df = [[0 ,0.000 ,0.000],
[15 ,0.000 ,0.000],
[30 ,0.005 ,0.000],
[45 ,0.003 ,0.000],
[60 ,0.013 ,0.000],
[75 ,0.075 ,0.000],
[90 ,0.137 ,0.000],
```

```
[105 ,0.530 ,0.000],
[120 ,0.841 ,0.000],
[135 ,1.000 ,0.000],
[150 ,1.000 ,0.000],
[165 ,1.000 ,0.009],
[180 ,1.000 ,0.186],
[195 ,1.000 ,0.595],
[210 ,1.000 ,0.791],
[225 ,1.000 ,0.875],
[240 ,1.000 ,0.913],
[255 ,1.000 ,0.946],
[270 ,1.000 ,0.946],
[285 ,1.000 ,1.000],
[300 ,1.000 ,1.000],
[315 ,1.000 ,1.000],
[330 ,1.000 ,1.000],
[345 ,1.000 ,1.000],
[360 ,1.000 ,1.000]]
# count rows
howmanyrows = len(p4df)
# allocate lists
t=[0 for i in range(howmanyrows)]
bromide=[0 for i in range(howmanyrows)]
chromium=[0 for i in range(howmanyrows)]
# parse into useful lists
for irow in range(howmanyrows):
    t[irow]=p4df[irow][0]
    bromide[irow]=p4df[irow][1]
    chromium[irow]=p4df[irow][2]

if t[0]==0:t[0]=0.00001
```

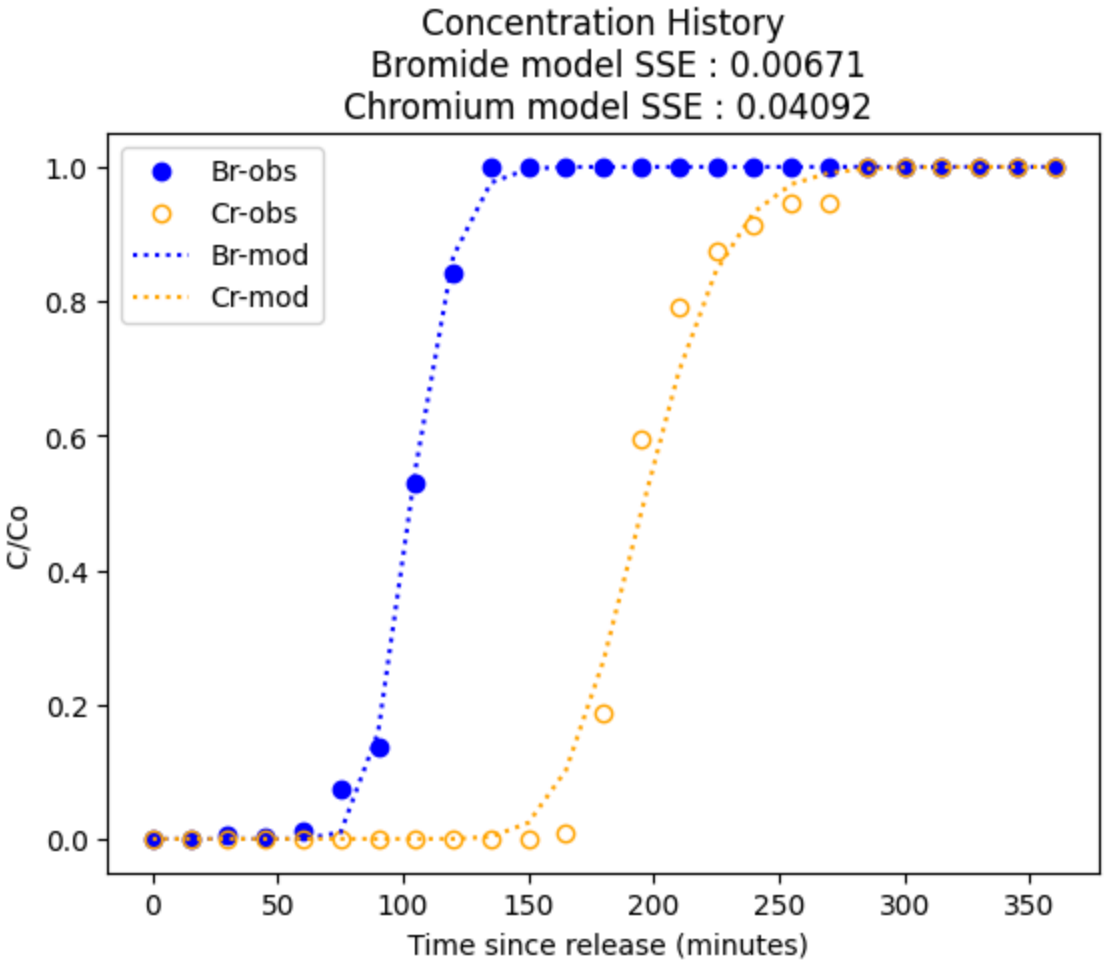
2. build an input data manager

- echo inputs
- plot observations and model results
- trial-and-error (or some optimization) to minimize prediction error by changing dispersivity and retardation to recover values
- report results

In [12]:
```
# input data manager
porosity = 0.485
velocity = 0.244 #cm/min
dispersivity = 0.232 #initial guess - change to fit bromide curve
retardationBr = 1.0
retardationCr = 1.9
dCr = (1.2e-09)*60*100*100 #molecular diffusivity - in cm^2/min
dBr = (1.6e-04)*100*100/1440 #molecular diffusivity in cm^2/min
spdischarge = velocity*porosity
dispersion = dispersivity*velocity
length = 25.4 #cm
diameter = 2.54 #cm
rhob = 1.85 #g/ml
#echo inputs
```

```
print("        ---- Supplied Values ----  ")
print("                  Porosity : ",porosity)
print("             Pore Velocity : ",round(velocity,3)," cm/min" )
print("              Dispersivity : ",dispersivity," cm ")
print(" Molecular Diffusivity Br : ",round(dBr,6)," cm^2/min ")
print(" Molecular Diffusivity Cr : ",round(dCr,6)," cm^2/min ")
print("    Retardation Factor Br : ",retardationBr)
print("    Retardation Factor Cr : ",retardationCr)
print("        ---- Computed Values ----  ")
print("       Specific Discharge : ",spdischarge," cm/min ")
print("                Dispersion : ",round(dispersion,3)," cm^2/min ")
print("                Kd-bromide : ",round((retardationBr-1)*(porosity/rhob),3))
print("               Kd-chromium : ",round((retardationCr-1)*(porosity/rhob),3))

# build simulation results
brmodel = [0 for i in range(howmanyrows)]
crmodel = [0 for i in range(howmanyrows)]
for irow in range(howmanyrows):
    brmodel[irow]= c1dadrd(1.0,length,t[irow],dispersion+dBr,velocity,retardationBr
    crmodel[irow]= c1dadrd(1.0,length,t[irow],dispersion+dCr,velocity,retardationCr
# compute some error measure
sseBr = 0.0
sseCr = 0.0
for irow in range(howmanyrows):
    sseBr = sseBr + ((bromide[irow]-brmodel[irow])**2)
    sseCr = sseCr + ((chromium[irow]-crmodel[irow])**2)
#plot simulation and observation results
from matplotlib import pyplot as plt
plt.plot(t,bromide, color='blue', linestyle = "none", marker = 'o')  # make the plo
plt.plot(t,chromium, color='orange', linestyle = "none", marker = 'o', fillstyle =
plt.plot(t,brmodel, color='blue', linestyle = "dotted" )  # make the plot object
plt.plot(t,crmodel, color='orange', linestyle = "dotted")  # make the plot object
plt.title(" Concentration History \n  " + " Bromide model SSE : " + repr(round(sseB
plt.xlabel(" Time since release (minutes)") # label x-axis
plt.ylabel(" C/Co        ") # label y-axis
#plt.xscale('log')
#plt.yscale('log')
plt.legend(["Br-obs","Cr-obs","Br-mod","Cr-mod"])
plt.show() # plot to stdio -- has to be last call as it kills prior objects
plt.close('all') # needed when plt.show call not invoked, optional here
#sys.exit() # used to elegant exit for CGI-BIN use
```

```
       ---- Supplied Values ----
                 Porosity :  0.485
            Pore Velocity :  0.244  cm/min
             Dispersivity :  0.232  cm
Molecular Diffusivity Br :  0.001111  cm^2/min
Molecular Diffusivity Cr :  0.00072  cm^2/min
   Retardation Factor Br :  1.0
   Retardation Factor Cr :  1.9
       ---- Computed Values ----
      Specific Discharge :  0.11834  cm/min
               Dispersion :  0.057  cm^2/min
               Kd-bromide :  0.0
              Kd-chromium :  0.236
```



3. Discussion of Results

Trial-and-error fit could be replaced by an optimizer (Solver in excel, or some python equilivant), but hardly worth the effort - once you have the advection part (C/Co = 0.5) located, then dispersivity controls curvature. If one could argue for different dispersivities for each species could get a better fit, but dispersivtiy is considered a material property of the porous media. The solution above uses literature supplied diffusivities from:

For **Bromide**

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2898110/

by van Dam et al. (1997) and Beltra et al. (2001). In the period from 19 September to 16 October, the foliage died off linearly until no active canopy remained. The soil was kept bare during the next year (2001).

Input Data for the Substances

Bromide ion was selected as a tracer for water flow because it barely interacts with soil. The ionic mass of bromide ion is 79.9 g mol^{-1}. The vapor pressure of bromide ion was set to zero and its solubility in water at 10^5 mg l^{-1} (an arbitrary high value). The coefficient for diffusion of bromide ion in water (Lide 1999), translated to 20°C, was taken to be 1.6 10^{-4} m^2 day^{-1}. Adsorption of bromide ion to soil was set to zero. The half-life of transformation in soil was taken to be 10^5 days (an arbitrary high value) at 20°C. The concept of the transpiration stream concentration factor (TSCF) for uptake by plant roots (Briggs et al. 1982) does not hold for ionic species. As there is no alternative in the model and as no quantitative information was available, the TSCF was set at 0.5.

and for **Chromium**

s, initially
and 1.25c,
concentra-
glass tank
at bath at
lly, and a
nt solution
perimental
ach of the
ue, c, that
cides with
l boundary
lve Fick's
re, the so-
ortions. In
ollowed by
and R_b of
ig current
atus, w is

Table 1. Diffusion Coefficients, D,[a] of $CrCl_3$ in Aqueous Solutions at Various Concentrations, c

c/mol dm^{-3}	$D/10^{-9}$ $m^2 \cdot s^{-1}$ [a]	$S_D/10^{-9}$ $m^2 \cdot s^{-1}$ [b]
	$T = 298.15$ K	
0.005	1.326	0.010
0.008	1.200	0.010
0.01	1.170	0.016
0.02	1.096	0.017
0.03	1.060	0.008
0.05	1.026	0.005
	$T = 303.15$ K	
0.005	1.847	0.010
0.008	1.328	0.011
0.01	1.310	0.010
0.02	1.260	0.010
0.03	1.200	0.011
0.05	1.170	0.010

[a] D is the mean diffusion coefficient for three experiments. [b] S_D is the standard deviation of that mean.

In [ ]: