In [1]:
```
#import library
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

In [2]:
```
def initial_hydrocarbon_in_place(Nfoi, Gfgi, Rv, Rs):
  """
  Calculate OOIP and OGIP from Nfoi and Gfgi
  And output the result to labels in the plot
  """
  import matplotlib.patches as mpl_patches

  Rvi, Rsi = Rv[0], Rs[0]
  OOIP = Nfoi + Gfgi * Rvi
  OGIP = Gfgi + Nfoi * Rsi

  labels = []
  labels.append("Nfoi = {0:.4g} STB".format(Nfoi))
  labels.append("Gfgi = {0:.4g} SCF".format(Gfgi))
  labels.append("OOIP = {0:.4g} STB".format(OOIP))
  labels.append("OGIP = {0:.4g} SCF".format(OGIP))

  handles = [mpl_patches.Rectangle((0, 0), 1, 1, fc="white", ec="white",
                                   lw=0, alpha=0)] * 4
  return labels, handles, OOIP, OGIP
```

In [3]:
```
def calculate_params(p, Bo, Bg, Rv, Rs, Np, Gp, Gi, cf, cw, swi,Rp):
      """
      Calculate Material Balance Paramaters for Oil Reservoir

      Output: F, Bto, Btg, Efw, Eo, Eg
      """
      pi = p[0]
      Rsi = Rs[0]
      Rvi = Rv[0]
      Boi = Bo[0]
      Bgi = Bg[0]

      # calculate Efw
      Efw = ((cf + cw * swi) / (1 - swi)) * (pi - p)

      # calculate F, Bto, and Btg
      F = (Np * ((Bo - (Rs * Bg)) / (1 - (Rv * Rs)))) + ((Gp - Gi) * ((Bg - (Rv * Bo))
      Btg = ((Bg * (1 - (Rs * Rvi))) + (Bo * (Rvi - Rv))) / (1 - (Rv * Rs))  # in RB/ST
      Bto = ((Bo * (1 - (Rv * Rsi))) + (Bg * (Rsi - Rs))) / (1 - (Rv * Rs))  # in RB/sc

      # calculate Eo and Eg
      Eo = Bto - Boi
      Eg = Btg - Bgi

      return F, Bto, Btg, Efw, Eo, Eg


def gascap(Gfgi, Nfoi, Bg, Bo):
    """
    Calculate Total Oil+Gas Expansion Factor from known Gas Cap ratio
    Gfgi and Nfoi known from volumetrics
    """
    Bgi, Boi = Bg[0], Bo[0]
```

```
        m = (Gfgi * Bgi) / (Nfoi * Boi)
        return m

def plot(oil_type, F, Bto, Btg, Efw, Eo, Eg, Np, Bo, Rs, Rv, start=0, end=-1, figsize=(10
        """
        Create Material Balance Plots for Oil Reservoir

        Input:
        oil_type: 'undersaturated' or 'saturated'
        """
        import numpy as np
        import matplotlib.pyplot as plt
        from scipy.optimize import curve_fit
        import matplotlib.patches as mpl_patches

        # plot attributes
        title_size = 15
        title_pad = 14

        # linear function for curve-fit
        def linear_zero_intercept(x, m):
            y = m * x
            return y

        def linear_with_intercept(x, m, c):
            y = m * x + c
            return y

        if oil_type == 'undersaturated':

          plt.figure(figsize=figsize)

          " Plot 1: F vs (Eg+Boi*Efw) "

          plt.subplot(1,2,1)
          Boi = Bo[0]
          x1, y1 = (Eg + Boi * Efw), F
          plt.plot(x1, y1, '.-')
          plt.title(r'Plot 1: $F$ vs $(E_o+B_{oi}*E_{fw})$', size=title_size, pad=title_pad
          plt.xlabel(r'$E_o+B_{oi}E_{fw}$ (RB/STB)', size=15)
          plt.ylabel(r'$F$ (res bbl)', size=15)

          ## curve-fitting to calculate the slope as OOIP
          x1_norm = x1 / max(x1) # normalize x
          y1_norm = y1 / max(y1) # normalize y

          x1_norm = x1_norm[start:end]
          y1_norm = y1_norm[start:end]

          popt, pcov = curve_fit(linear_zero_intercept, x1_norm, y1_norm)

          m = popt[0]
          Nfoi = m * max(y1) / max(x1) # denormalize the slope, hence the OGIP

          ## Calculate OOIP and OGIP from Nfoi
          Rsi = Rs[0]
          Gfgi = 0 # no free gas phase in undersaturated oil
          OOIP = Nfoi
          OGIP = Nfoi * Rsi
```

```
    ## Output results into text in plot
    labels, handles, OOIP, OGIP = initial_hydrocarbon_in_place(Nfoi, Gfgi, Rv, Rs)

    ## plot the regression line
    x1_fit = np.linspace(min(x1), max(x1), 5)
    y1_fit = linear_zero_intercept(x1_fit, Nfoi)
    plt.plot(x1_fit, y1_fit, label='{} MMSTB'.format(np.round(Nfoi * 1E-6, 3)))


    " Plot 2: F/(Eg+Boi*Efw) vs Np (Waterdrive Diagnostic Plot) "

    plt.subplot(1,2,2)
    x2, y2 = Np, F / (Eg + Boi * Efw)
    plt.plot(x2, y2, '.-')
    plt.title('Plot 2: Waterdrive Diagnostic Plot', size=title_size, pad=title_pad)
    plt.xlabel(r'$N_p$ (STB)', size=15)
    plt.ylabel(r'$\frac{F}{(E_o+B_{oi}E_{fw})}$ (STB)', size=15)

    ## curve-fitting to calculate the slope as OOIP, here [1:] because NaN is removed
    x2_norm = x2[1:] / max(x2[1:]) # normalize x
    y2_norm = y2[1:] / max(y2[1:]) # normalize y
    popt, pcov = curve_fit(linear_with_intercept, x2_norm, y2_norm)

    m, c = popt[0], popt[1]
    m = m * max(y2[1:]) / max(x2[1:]) # denormalize the slope
    Nfoi = c * max(y2[1:]) # denormalize the intercept, hence the OGIP

    ## Calculate OOIP and OGIP from Nfoi
    Rsi = Rs[0]
    Gfgi = 0 # no free gas phase in undersaturated oil
    OOIP = Nfoi
    OGIP = Nfoi * Rsi

    ## Output results into text in plot
    labels, handles, OOIP, OGIP = initial_hydrocarbon_in_place(Nfoi, Gfgi, Rv, Rs)

    ## plot the regression line
    x2_fit = np.linspace(min(x2[1:]), max(x2[1:]), 5)
    y2_fit = linear_with_intercept(x2_fit, m, Nfoi)
    plt.plot(x2_fit, y2_fit, label='{} MMSTB'.format(np.round(Nfoi * 1E-6, 3)))
    plt.tight_layout(1)
    plt.show()

  if oil_type == 'saturated':

    plt.figure(figsize=figsize)

    " Plot 1: F/Eo vs Eg/Eo "

    plt.subplot(1,3,1)
    x1, y1 = (Eg / Eo), (F / Eo)
    plt.plot(x1, y1, '.-')
    plt.title('Plot 1: F/Eo vs Eg/Eo', size=title_size, pad=title_pad)
    plt.xlabel(r'$\frac{Eg}{Eo}$ (STB/scf)', size=15)
    plt.ylabel(r'$\frac{F}{Eo}$ (STB)', size=15)

    ## curve-fitting to calculate the slope as Gfgi, intercept as Nfoi
    x1_norm = x1[1:] / max(x1[1:]) # normalize x
    y1_norm = y1[1:] / max(y1[1:]) # normalize y
    popt, pcov = curve_fit(linear_with_intercept, x1_norm, y1_norm)
```

```python
    m, c = popt[0], popt[1]
    Gfgi = m = m * max(y1[1:]) / max(x1[1:]) # denormalize the slope
    Nfoi = c = c * max(y1[1:]) # denormalize the intercept

    ## calculate OOIP and OGIP from Nfoi and Gfgi
    Rsi, Rvi = Rs[0], Rv[0]
    OOIP = Nfoi + Gfgi * Rvi
    OGIP = Gfgi + Nfoi * Rsi

    ## Output results into text in plot
    labels, handles, OOIP, OGIP = initial_hydrocarbon_in_place(Nfoi, Gfgi, Rv, Rs)

    ## plot the regression line
    x1_fit = np.linspace(min(x1[1:]), max(x1[1:]), 5)
    y1_fit = linear_with_intercept(x1_fit, m, c)
    plt.plot(x1_fit, y1_fit)

    plt.legend(handles, labels, loc='best', fontsize='small',
               fancybox=True, framealpha=0.7,
               handlelength=0, handletextpad=0)

    " Plot 2: p/z vs Gp "

    plt.subplot(1,3,2)
    x2, y2 =  (Eo / Eg), (F / Eg)
    plt.plot(x2, y2, '.-')
    plt.title('Plot 2: F/Eg vs Eo/Eg', size=title_size, pad=title_pad)
    plt.xlabel(r'$\frac{Eo}{Eg}$ (scf/STB)', size=15)
    plt.ylabel(r'$\frac{F}{Eg}$ (scf)', size=15)

    ## curve-fitting to calculate the slope as Nfoi, intercept as Gfgi
    x2_norm = x2[1:] / max(x2[1:]) # normalize x
    y2_norm = y2[1:] / max(y2[1:]) # normalize y
    popt, pcov = curve_fit(linear_with_intercept, x2_norm, y2_norm)

    m, c = popt[0], popt[1]
    Nfoi = m = m * max(y2[1:]) / max(x2[1:]) # denormalize the slope
    Gfgi = c = c * max(y2[1:]) # denormalize the intercept

    ## calculate OOIP and OGIP from Nfoi and Gfgi
    Rsi, Rvi = Rs[0], Rv[0]
    OOIP = Nfoi + Gfgi * Rvi
    OGIP = Gfgi + Nfoi * Rsi

    ## Output results into text in plot
   ## labels, handles, OOIP, OGIP = initial_hydrocarbon_in_place(Nfoi, Gfgi, Rv, Rs)

    ## plot the regression line
    x2_fit = np.linspace(min(x2[1:]), max(x2[1:]), 5)
    y2_fit = linear_with_intercept(x2_fit, m, c)
    plt.plot(x2_fit, y2_fit)

    plt.legend(handles, labels, loc='best', fontsize='small',
               fancybox=True, framealpha=0.7,
               handlelength=0, handletextpad=0)

    plt.tight_layout(1)

    plt.show()
```

In [4]:
```
data=pd.read_excel("Book1.xlsx")
data.columns
```

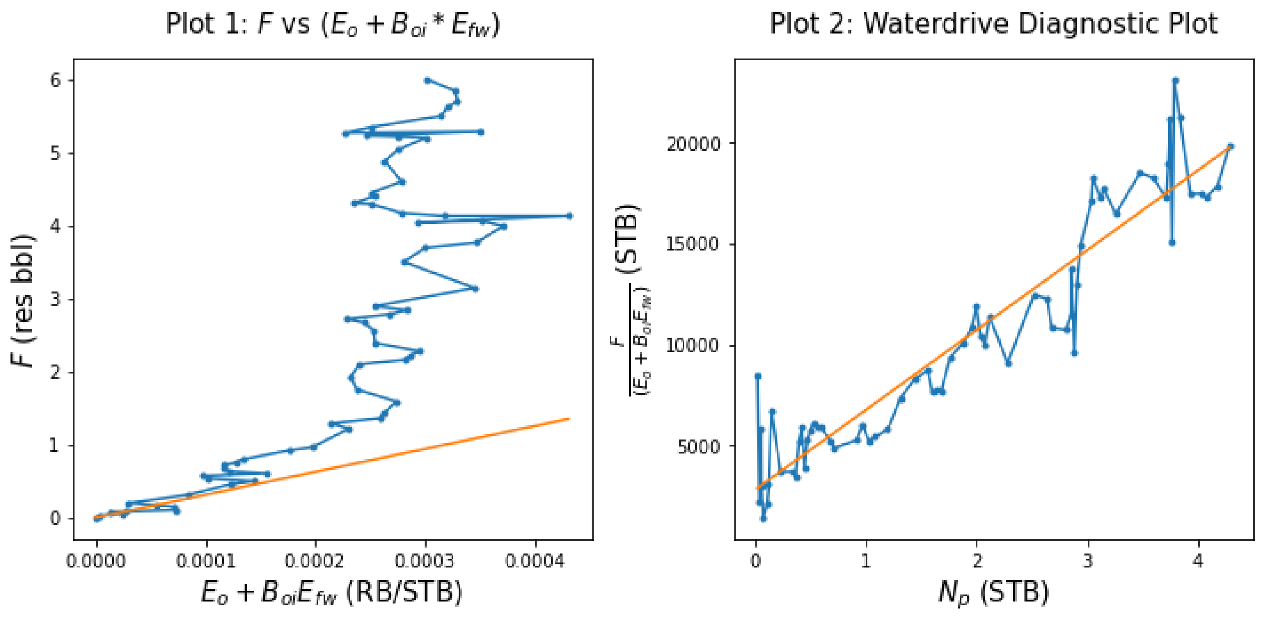
Out[4]:
```
Index(['Date', 'p(psia)', 'Np(MMSTB)', 'Np(STB)', 'Gp(MMSCF)', 'Gp(SCF)',
       'Wp(MMSTB)', 'Wp(STB)', 'Rp', 'Bo', 'Bg(ft3/scf)', 'Bg(bbl/scf)', 'Rs',
       'Rv'],
      dtype='object')
```

In [5]:
```
#input all the reqiured inputs
p=data['p(psia)'].values
Bo=data['Bo'].values
Bg=data['Bg(bbl/scf)'].values
Rv=data['Rv'].values
Rs=data['Rs'].values
Np=data['Np(MMSTB)'].values
Gp=data['Gp(MMSCF)'].values
Rp=data['Rp'].values
Gi=0
cf=3.5E-12
cw=3.4E-12
swi=0.20
```

In [6]:
```
#calculate parameters for the plotting the graph
F, Bto, Btg, Efw, Eo, Eg=calculate_params(p, Bo, Bg, Rv, Rs, Np, Gp, Gi, cf, cw, swi,Rp)
```

In [7]:
```
#creating MBAL plots and automatically give in placeresults
plot('undersaturated',F, Bto, Btg, Efw, Eo, Eg, Np, Bo, Rs, Rv,end=4)
```

```
<ipython-input-3-c6de9dd70c9c>:107: RuntimeWarning: invalid value encountered in true_div
ide
  x2, y2 = Np, F / (Eg + Boi * Efw)
<ipython-input-3-c6de9dd70c9c>:135: MatplotlibDeprecationWarning: Passing the pad paramet
er of tight_layout() positionally is deprecated since Matplotlib 3.3; the parameter will
become keyword-only two minor releases later.
  plt.tight_layout(1)
```



In [8]:
```
#schilthuis water influx
def calculate_aquifer(pressure, Bw, Wp, Np, Bo, Nfoi, cf, cw, swi, Boi):
        """Calculate Material Balance parameters of Undersaturated Oil Reservoir for Schi
        # in case of undersaturated (above bubblepoint), Rp = Rs = Rsi, Gfgi = Bgi = Eg =
        import numpy as np

        F = Np * Bo
        Eo = Bo - Boi

        delta_pressure = pressure - pressure[0]
        delta_pressure = np.abs(delta_pressure)

        Efw = ((cf + (cw * swi)) / (1 - swi)) * delta_pressure

        We_schilthuis = (Bw * Wp) + F - (Nfoi * Eo) - ((Nfoi * Boi) * Efw)

        return We_schilthuis
```

In [9]:

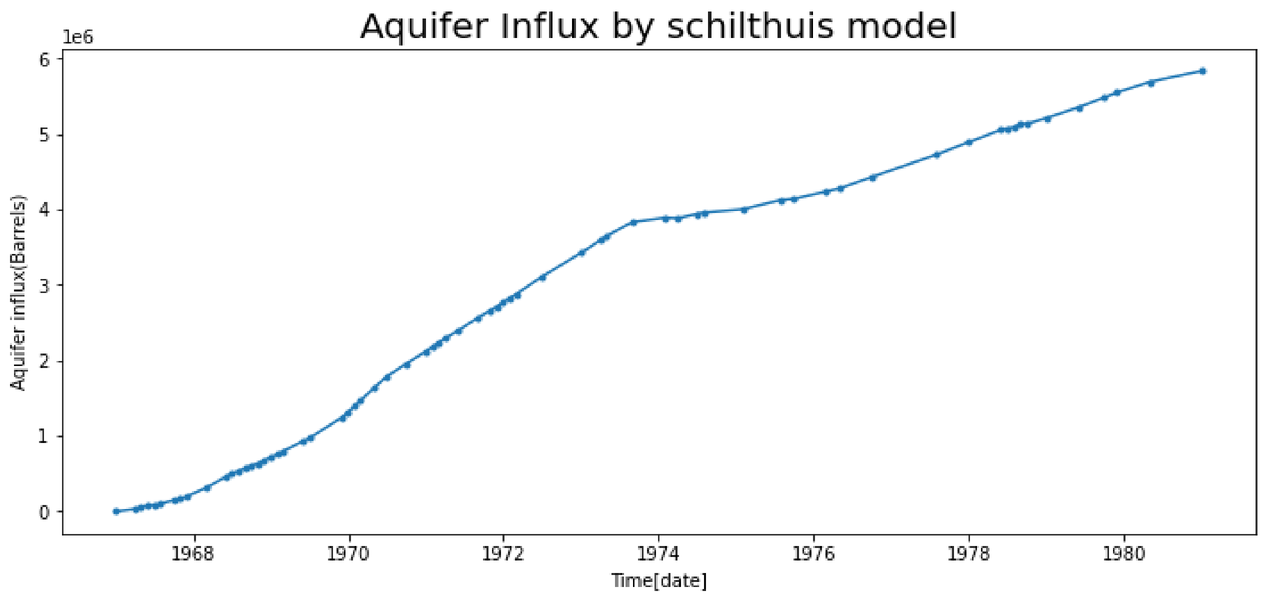
```
pressure=data['p(psia)'].values
Bw=1
Wp=data['Wp(STB)'].values
Np=data['Np(STB)'].values
Bo=data['Bo'].values
Nfoi=2810
cf=3.50E-06
cw=3.40E-06
swi=0.2
Boi=1.3307
We_schilthuis=calculate_aquifer(pressure, Bw, Wp, Np, Bo, Nfoi, cf, cw, swi, Boi)
We_schilthuis
```

Out[9]:

```
array([-4.23403647e-02,  3.08360560e+04,  5.39851479e+04,  7.84723870e+04,
        8.25994422e+04,  1.02021204e+05,  1.47497872e+05,  1.66152079e+05,
        1.97720910e+05,  3.10658827e+05,  4.53168892e+05,  4.97015987e+05,
        5.35133935e+05,  5.69722802e+05,  6.04806625e+05,  6.35387014e+05,
        6.73894451e+05,  7.16622878e+05,  7.56375112e+05,  7.98647144e+05,
        9.27554963e+05,  9.68528171e+05,  1.24100512e+06,  1.31804574e+06,
        1.39402343e+06,  1.47082038e+06,  1.63017017e+06,  1.78658739e+06,
        1.95553968e+06,  2.11849425e+06,  2.17773488e+06,  2.23328229e+06,
        2.28926419e+06,  2.39371256e+06,  2.55872047e+06,  2.66088438e+06,
        2.71130633e+06,  2.77043583e+06,  2.82513749e+06,  2.87719752e+06,
        3.10832664e+06,  3.42398132e+06,  3.58899873e+06,  3.64789947e+06,
        3.83305619e+06,  3.89103758e+06,  3.88286865e+06,  3.94050938e+06,
        3.95796564e+06,  4.00219868e+06,  4.12494089e+06,  4.14064635e+06,
        4.23764308e+06,  4.27591108e+06,  4.42914716e+06,  4.72479434e+06,
        4.89137991e+06,  5.05282962e+06,  5.07212428e+06,  5.08832458e+06,
        5.13369356e+06,  5.13309886e+06,  5.20941032e+06,  5.35151553e+06,
        5.48246445e+06,  5.55011452e+06,  5.68777008e+06,  5.83414203e+06])
```

In [10]:

```
plt.figure(figsize=(12,5))
plt.plot(data['Date'],We_schilthuis,'.-')
plt.title('Aquifer Influx by schilthuis model',size=20)
plt.xlabel('Time[date]')
plt.ylabel('Aquifer influx(Barrels)')
plt.show()
```



In [11]:
```python
#class veh():
def calculate_aquifer_constant(r_R, h, cf, cw, poro, theta=360):
        """
        Calculate theoretical aquifer constant for VEH (assuming cylindrical reservoir)
        Input:
        r_R = reservoir radius
        """
        import numpy as np

        ct = cf + cw  # total compressibility, in aquifer sw=1
#         theta = 360  # full circle cylindrical
        B_star = 1.119 * poro * ct * h * (r_R ** 2) * (theta / 360)

        return B_star

def calculate_aquifer(datetime, pressure, cf, cw, perm, poro, mu_w, r_R, B_star, rw=0.5):
        import numpy as np

        def qd(rd, td):
            """
            Dimensionless cumulative production (QD) using Klins et al. Polynomial
            Approach to Bessel Functions in Aquifer
            """
            from scipy.special import j1
            from scipy.special import j0
            import math
            import mpmath
            def csch(x):
                if x > 100:
                    return 0
                else:
                    return float(mpmath.csch(x))

            def beta(b,rd):
                return b[0]+b[1]*csch(rd)+b[2]*rd**b[3]+b[4]*rd**b[5]

            # Algorithm
            if td < 0.01:
                return 2*td**0.5/3.14159265359**0.5
            else:
                b = [1.129552, 1.160436, 0.2642821, 0.01131791, 0.5900113, 0.04589742, 1.
                qd_inf = (b[0]*td**b[7]+b[1]*td+b[2]*td**b[8]+b[3]*td**b[9])/(b[4]*td**b[
                if rd > 100:
                    return qd_inf

                b1 = [-0.00222107, -0.627638, 6.277915, -2.734405, 1.2708, -1.100417]
                b2 = [-0.00796608, -1.85408, 18.71169, -2.758326, 4.829162, -1.009021]

                alpha1 = beta(b1,rd)
                alpha2 = beta(b2,rd)
                J0Alpha1 = j0(alpha1)
                J0Alpha2 = j0(alpha2)
                J1Alpha1rd = j1(alpha1*rd)
                J1Alpha2rd = j1(alpha2*rd)

                qd_fin = (rd**2-1)/2 - (2*math.exp(-alpha1**2*td)*J1Alpha1rd**2)/(alpha1*
            return min(qd_inf, qd_fin)

        def time_pressure_difference(datetime):
            """Calculate time and pressure differences"""

            # Subtracting datetimes to get time differences from initial production date
            diff = datetime - datetime[0]

            # convert datetime format to integer
            time_array = []
            for k in range(len(diff)):
                diffr = diff[k] / np.timedelta64(1, 'D')
                time_array.append(float(diffr))

            # convert time difference from day to hour
            time_array = np.array(time_array) * 24

            # create j index for dataframe
            j_index = np.arange(0, (len(datetime)), 1)

            # calculate delta_pressure for each date
            # append an array consists of two initial pressures [pi, pi] (as dummy) to th
            pi = pressure[0]

            p_dummy = np.append(np.array([pi, pi]), pressure)
            delta_p_j = [b - a for a, b in zip(p_dummy[:-2], p_dummy[2:])]
            delta_p_j = 0.5 * np.array(np.abs(delta_p_j))

            # pre-processing
            j_array = np.arange(1, (len(time_array) + 1), 1)
            delta_p_j_array = delta_p_j[1:]

            array_j = []
            array_time = []
            delta_pressure = []
            array_time_repeat = []

            for i in range(len(time_array)):
                new_j = j_array[:i]
                new_time = time_array[:i]
                new_delta_p_j = delta_p_j_array[:i]

                array_j.append(new_j)
                array_time.append(new_time)
                delta_pressure.append(new_delta_p_j)
                # make arrays of repeated times
                new_time_repeat = np.repeat((time_array[i]), i)
                array_time_repeat.append(new_time_repeat)

            # To calculate delta_time, SUBTRACT arrr_time TO arrr_time_repeat
            delta_time = np.subtract(array_time_repeat, array_time)  # numpy subtract arr

            return delta_time, delta_pressure


        def calculate_parameter_VEH(index, delta_time, cf, cw, perm, poro, mu_w, r_R):
            """Calculate dimensionless time (t_DR) and dimensionless aquifer influx (W_eD

            # Calculate t_DR and W_eD
            ct = cf + cw
            t_DR_factor = (0.0002637 * perm) / (poro * mu_w * ct * (r_R ** 2))

            t_DR_arr = []
            W_eD_arr = []

            for i in range(len(delta_time[index])):
                t_DR = t_DR_factor * (delta_time[index])[i]

                # Dimensionless radius
                rd = r_R / rw

                # Use Bessel function to calculate dimensionless Qd (W_eD)
                W_eD = qd(rd, t_DR)

                # "calculate W_eD using Eq 6.36 and 6.37 for infinite reservoir (See: 6_e
                # if t_DR > 0.01 and t_DR <= 200:
                #     # use Eq 6.36
                #     W_eD = ((1.12838 * np.sqrt(t_DR)) + (1.19328 * t_DR) + (0.269872 *
                #                 0.00855294 * (t_DR ** 2))) / (1 + (0.616599 * np.sqrt(t
                # if t_DR > 200:
                #     # use Eq 6.37
                #     W_eD = ((2.02566 * t_DR) - 4.29881) / np.log(t_DR)

                W_eD_arr.append(float(W_eD))
                t_DR_arr.append(float(t_DR))
            return (t_DR_arr, W_eD_arr)

        # Calculate time differences
        delta_time, delta_pressure = time_pressure_difference(datetime)

        # Calculate aquifer influx
        We_veh = []

        for x in range(len(datetime)):  # range from j index 1 to 9

            t_DR_arr, W_eD_arr = calculate_parameter_VEH(x, delta_time, cf, cw, perm, por

            # calculate We, Equation 8.7

            W_eD_multipy_delta_p_j = delta_pressure[x] * W_eD_arr
            sigma_We = np.sum(W_eD_multipy_delta_p_j)
            We = B_star * sigma_We
            We_veh.append(float(We))

        return We_veh
```

In [12]:
```
#converting datetime format
data['Date']=pd.to_datetime(data['Date'],format='%Y-%m-%d')
data
```

Out[12]:

	Date	p(psia)	Np(MMSTB)	Np(STB)	Gp(MMSCF)	Gp(SCF)	Wp(MMSTB)	Wp(STB)
0	1966-12-31	4714.097000	0.000000	0.000000e+00	0.000000	0.000000e+00	0.000000	0.000000
1	1967-03-31	4667.921900	0.023152	2.315223e+04	12.902651	1.290265e+07	0.000013	12.902651
2	1967-04-30	4424.097000	0.040424	4.042406e+04	22.836518	2.283652e+07	0.000023	22.836518
3	1967-05-31	4551.921900	0.058839	5.883942e+04	31.229356	3.122936e+07	0.000031	31.229356
4	1967-06-30	4395.969350	0.061825	6.182518e+04	32.604690	3.260469e+07	0.000033	32.604690
...	...	...	...	...	...	...	...	...
63	1979-05-31	2779.420800	3.928389	3.928389e+06	2065.849901	2.065850e+09	0.002066	2065.849901
64	1979-09-30	2757.651930	4.023485	4.023485e+06	2111.035934	2.111036e+09	0.002111	2111.035934
65	1979-11-30	2732.966700	4.071946	4.071946e+06	2138.868420	2.138868e+09	0.002139	2138.868420
66	1980-04-30	2741.200000	4.173338	4.173338e+06	2197.191207	2.197191e+09	0.002197	2197.191207
67	1980-12-31	2821.298558	4.284769	4.284769e+06	2260.760870	2.260761e+09	0.002261	2260.760870

68 rows × 14 columns

In [13]:
```
#acquifer constant
r_R=1.5*3281
h=300
poro=0.2
#calculating aquifer constant(B*)in unit rB/d-psi
B_star=calculate_aquifer_constant(r_R, h, cf, cw, poro)
B_star
```

Out[13]:
```
11220.840950908501
```

In [14]:
```
datetime=data['Date']
pressure=data['p(psia)'].values
perm=100
mu_w=0.318
B_star=B_star
```

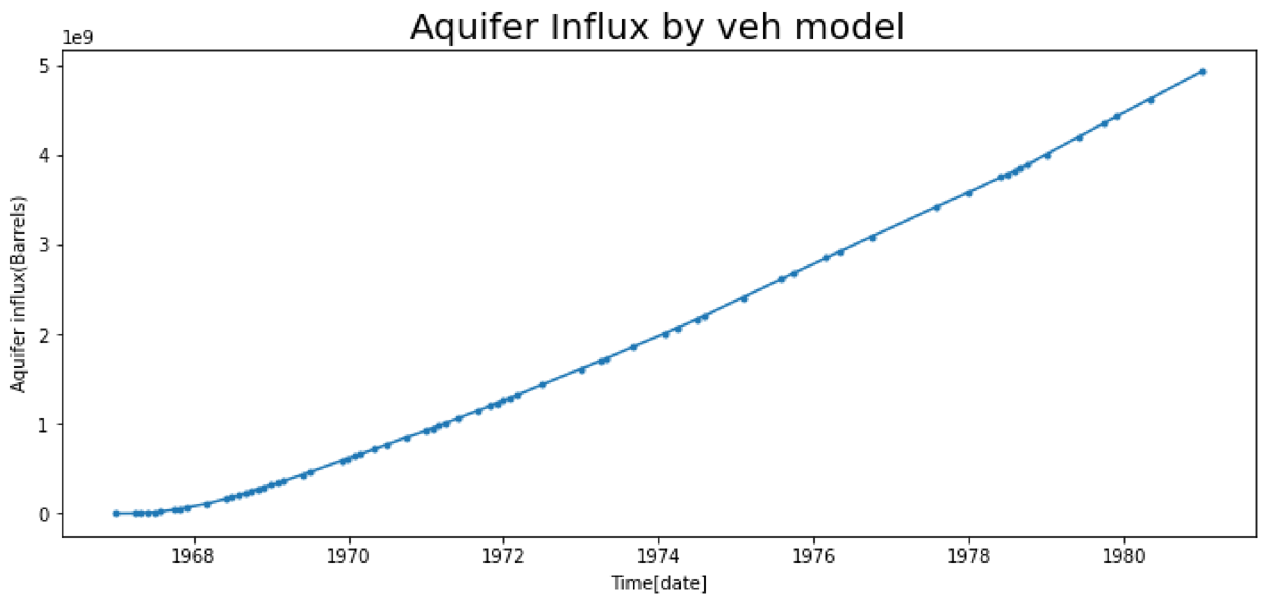
In [15]:
```
We=calculate_aquifer(datetime, pressure, cf, cw, perm, poro, mu_w, r_R, B_star)
We
```

```
<ipython-input-11-c8383cf502d6>:108: VisibleDeprecationWarning: Creating an ndarray from
ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with dif
ferent lengths or shapes) is deprecated. If you meant to do this, you must specify 'dtype
```

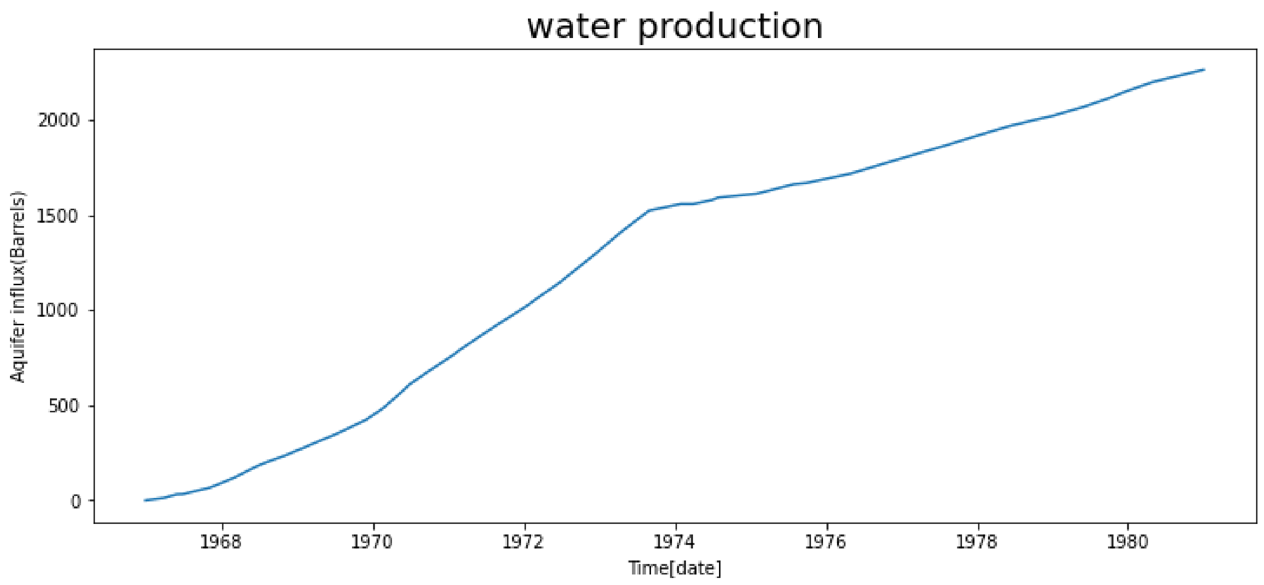
```
=object' when creating the ndarray
  delta_time = np.subtract(array_time_repeat, array_time)  # numpy subtract array to arra
y
```

Out[15]:
```
[0.0,
 1232355.6153628519,
 5203875.435055299,
 9210085.337648734,
 12551392.635741565,
 22997801.274979737,
 44503272.93759489,
 55264625.77478771,
 69176622.49540691,
 108567558.46560863,
 162732793.89495593,
 182670059.03402296,
 202384793.82480344,
 224498900.77329347,
 247831618.0879962,
 271488274.2131465,
 295543661.91371644,
 318754864.7842112,
 341873030.5195956,
 363242820.82226545,
 436074118.624019,
 462473042.9474366,
 590402744.9457686,
 615674217.3698503,
 641647916.5407826,
 666628224.1016694,
 718738809.1877946,
 770860718.9998268,
 850467248.7893631,
 926705455.7301481,
 954332006.4432113,
 980289906.6674954,
 1008282005.4395105,
 1063987972.4572848,
 1148566431.3628254,
 1203353010.6639369,
 1231065832.62827,
 1260083038.4305449,
 1291036616.961913,
 1319296369.6490667,
 1439897484.876425,
 1616293735.364746,
 1703480980.144796,
 1734531997.0181236,
 1861719440.3890333,
 2013225670.0452566,
 2075137373.9189756,
 2172163887.7918215,
 2205233175.4054737,
 2413653338.8631573,
 2617638818.7611566,
 2687556705.773564,
 2856448177.4449267,
 2924320148.330869,
 3094572996.474119,
 3426029078.289542,
 3589668161.9466233,
 3754329493.7643266,
 3786617827.9376044,
 3822392389.2553606,
 3860151464.1980505,
 3896160954.1404243,
 4012057720.304333,
 4205025621.8985896,
 4362766143.312319,
 4440366932.422426,
 4630621473.165852,
 4935585653.05066]
```

In [16]:
```
plt.figure(figsize=(12,5))
plt.plot(datetime,We,'.-')
plt.title('Aquifer Influx by veh model',size=20)
plt.xlabel('Time[date]')
plt.ylabel('Aquifer influx(Barrels)')
plt.show()
```



In [17]:
```
plt.figure(figsize=(12,5))
plt.plot(data['Date'],data['Wp(STB)'])
plt.title('water production',size=20)
plt.xlabel('Time[date]')
plt.ylabel('Aquifer influx(Barrels)')
plt.show()
```



In [ ]: