```python
from Model import Model
from scipy.integrate import solve_ivp
import numpy as np
import pandas as pd
from inspect import isfunction

class GeneralModel(Model):
    def __init__(self, times, B, A, K, iv_list, input_fluxes):
        self.times = times
        self.B = B
        self.A = A
        self.K = K
        self.iv_list = iv_list
        self.input_fluxes = input_fluxes

        self.Y = self.ode_solver()
        self.create_headers()
        self.df = self.get_x_df()
        self.get_diagnostic_variables()

    def ode_solver(self):
        t = [self.times[0], self.times[-1]]
        y = solve_ivp(self.right_hand_equation, t, self.iv_list, t_eval =
        self.times, vectorized = True)
        return y

    def get_input(self, t, y):
        if isfunction(self.input_fluxes):
            self.tmp_input_fluxes = self.input_fluxes(t, y)
        else:
            self.tmp_input_fluxes = self.input_fluxes

        if isfunction(self.B):
            self.tmp_B = self.B(t, y)
        else:
            self.tmp_B = self.B

        if isfunction(self.A):
            self.tmp_A = self.A(t, y)
        else:
            self.tmp_A = self.A

        if isfunction(self.K):
            self.tmp_K = self.K(t, y)
        else:
            self.tmp_K = self.K

    def right_hand_equation(self, t, y):
        self.get_input(t, y)
        dydt = np.multiply(self.tmp_B, self.tmp_input_fluxes) +
        np.matmul(np.matmul(self.tmp_A, self.tmp_K), y)
        return dydt

    def get_x(self):
        return self.Y.y

    def get_x_df(self):
        res = self.Y.y.T
        df = pd.DataFrame(res)
        df.columns = self.create_headers()
        return df

    def write_output(self, filename):
        self.df.to_csv(filename, index=False)

    def get_diagnostic_variables(self):

        carbon_storage_capacity = []
        carbon_storage_potential = []
        residence_time = []
        carbon_storage = []

```

```
        for i in range(0, len(self.times)):

            t = self.times[i]
            Y = self.Y.y
            y = Y[:, i]
            y = y.reshape([y.shape[0], 1])

            dydt = self.right_hand_equation(t, y)
            matrix_AK = np.matmul(self.tmp_A, self.tmp_K)
            inverse_AK = np.linalg.inv(matrix_AK)
            matrix_Residence = np.matmul(-inverse_AK, self.tmp_B)
            Residence_time = np.sum(matrix_Residence)

            carbon_storage_capacity.append(np.sum(np.multiply(matrix_Residence,
            self.tmp_input_fluxes)))
            carbon_storage_potential.append(-np.sum(np.matmul(inverse_AK, dydt)))
            residence_time.append(Residence_time)
            carbon_storage.append(np.sum(y))

        self.df["residence_time"] = residence_time
        self.df["carbon_storage"] = carbon_storage
        self.df["carbon_storage_capacity"] = carbon_storage_capacity
        self.df["carbon_storage_potential"] = carbon_storage_potential

    def sasu_spinup(self):
        #Xss = -(AK)^(-1)BU
        dimensions = self.Y.y.T.shape
        X = np.zeros(dimensions)

        for i in range(0, len(self.times)):
            t = self.times[i]
            Y = self.Y.y
            y = Y[:, i]
            y = y.reshape([y.shape[0], 1])
            self.get_input(t, y)
            dydt = 0
            matrix_AK = np.matmul(self.tmp_A, self.tmp_K)
            inverse_AK = np.linalg.inv(matrix_AK)
            matrix_BU = np.multiply(self.tmp_B, self.tmp_input_fluxes)
            X[i] = np.matmul(-inverse_AK, matrix_BU).reshape(7)

        col_headers = self.create_headers()
        for i in range(0, dimensions[1]):
            self.df[col_headers[i]] = X[:, i]

    def create_headers(self):
        col_headers = []
        rows = self.Y.y.shape[0]
        for i in range(0, rows):
            col_headers.append("X" + str(i+1))
        return col_headers
```