This code is written to evaluate Equivalent Clean Air (ECA) flow rate, also called the clean air delivery rate (CADR), using two difefrent approaches.

The first approach is called log-linear approach that is conventionally used to evaluate ECA based on first-order decay assumption. The nex approach is called Integral approach which uses area under decay curve to estimate ECA.

First, we import all the needed python libraries.

In [1]:
```
import numpy as np
from scipy import optimize
import matplotlib.pyplot as plt
from scipy import stats
```

Next, we have to import concentration dataset alongside its measurment errors, either manually or though reading from a file.

In the following setion, 8 different cases are provided as examples of concentration dataset. The examples used are as follows:

- Case #1: Ideal first order
- Case #2: Non-biological particle by Zeng et al. (link).
- Case #3: Hypothetical E. Coli by Stephens et al. link.
- Case #4-6: Bacteriophage MS2 by Ratliff et al. doi
- Case #7: High res. VOC (formaldehyde) by Schumacher doi
- Case #8: Low res. VOC (formaldehyde) by Law et al. doi

Case #1: Ideal first order

In [ ]:
```
t_c = np.array([0, 15, 30, 60, 90, 120])                                    # Time poi
t_t = np.array([0, 15, 30, 60, 90, 120])                                    # Time poi

num_simulations = 10000                                                     # Number o

c_test = np.array([ 8.00E+10, 2.29E+10, 6.57E+09,
                    5.39E+08, 4.42E+07, 3.63E+06])                          # Test con

c_control = np.array([8.00E+10, 4.28E+10, 2.29E+10,
                      6.57E+09, 1.88E+09, 5.39E+08])                        # Control

c_test = np.log10(c_test)
c_control = np.log10(c_control)

c_bg_test=1                                                                 # If there
c_bg_control=1
c_bg_test = np.log10(c_bg_test)
c_bg_control = np.log10(c_bg_control)



sigma_test = np.array([0,0,0,0,0,0])                                        # Test sta
sigma_control = np.array([0,0,0,0,0,0])                                     # Control
```

```
V = 3000                                                                    # Chamber
#V = 85                                                                     # Chamber
```

Case #2: Real non-biological particles: Smoke (Truncated dataset)

In [ ]:

```
num_simulations = 10000                                                     # Number o



t_t = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
                11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21])                # Time poi

t_c = np.array([0, 1,   2, 3,   4, 5,   6, 7, 8, 9, 10,
                11,     12,     13,     14,     15, 16, 17,     18,     19,     20,     21



c_control = np.array([46629.5822,       45262.5924,     38944.8016,     41968.5662,     39
                    35389.2249,  37112.7987,     34422.0024,     33845.9813,     31813.524,
                    30782.9434,  29226.4405,     27259.9242,     27281.7278,     26790.9496
                    25085.1379,  22133.3491,     22427.7312,     21338.2308,     18838.1459
                    19229.6865,  19123.3036])                               # Test con

c_control=np.log10(c_control)

c_test = np.array([20733.1887, 17199.744, 15496.7706, 13660.3969, 11310.3118,
                   10883.8787, 9122.8056, 8170.2825, 6648.0319, 6299.9989,
                   4945.4016, 4635.3165, 3815.3086,3194.0609, 2984.6603,
                   2455.5343, 2276.8268, 1918.0086, 1619.6262, 1389.2609,
                   1244.1919, 1231.9916])                                   # Control
c_test=np.log10(c_test)


sigma_control = np.array([0., 0., 0., 0., 0., 0., 0., 0., 0.,
                          0., 0., 0., 0., 0., 0., 0., 0.,0., 0., 0., 0., 0.])

sigma_test = np.array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
                       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])



c_bg_test=113.7572667                                                       # If there
c_bg_control=607.22006143
#c_bg_test = np.log10(c_bg_test)
#c_bg_control = np.log10(c_bg_control)

V = 1296.048                                                                # Chamber
#V = 36.7                                                                   # Chamber
```

Case #2: Real non-biological particles: Smoke (full dataset)

In [ ]:

```
num_simulations = 10000                                                     # Number o

t_c = np.array([0, 1,   2,      3,      4,      5,      6,      7,      8,      9,      10
                16,     17,     18,     19,     20,     21,     22,     23,     24,     25
                32,     33,     34,     35,     36,     37,     38,     39,     40,     41
```

```
                48,     49,     50,     51,     52,     53,     54,     55,     56,     57
```

```
t_t = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
                11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21])                    # Time poi

c_control = np.array([46629.5822,       45262.5924,     38944.8016,     41968.5662,     39
                    35389.2249,  37112.7987,     34422.0024,     33845.9813,     31813.524,
                    30782.9434,  29226.4405,     27259.9242,     27281.7278,     26790.9496
                    25085.1379,  22133.3491,     22427.7312,     21338.2308,     18838.1459
                    19229.6865,  19123.3036,     18965.1773,     17917.88,       16800.8839
                    17120.304,   15590.7547,     15133.0791,     14335.7846,     14481.8507
                    13082.9319,  13615.1729,     12671.266,      12284.8735,     12212.7403
                    10803.6982,  10626.2898,     9913.9165,      9592.189,       9748.4621,
                    9832.5099,   8845.3218,      8723.0745,      8549.6231,      7938.6471,
                    7994.5961,   7957.8449,      7420.9003,      7394.3616,      6939.229,
                    6480.9155,   6413.2256,      6032.8, 6506.7329,      5738.9208,
                    5651.0126,   5511.8419,      5562.7665,      5061.1998,      5221.2187]


c_control=np.log10(c_control)

c_test = np.array([20733.1887, 17199.744, 15496.7706, 13660.3969, 11310.3118,
                   10883.8787, 9122.8056, 8170.2825, 6648.0319, 6299.9989,
                   4945.4016, 4635.3165, 3815.3086,3194.0609, 2984.6603,
                   2455.5343, 2276.8268, 1918.0086, 1619.6262, 1389.2609,
                   1244.1919, 1231.9916])                                       # Control
c_test=np.log10(c_test)

sigma_control = np.array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
                          0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
                          0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
                          0., 0., 0., 0., 0., 0., 0., 0., 0.])                 # Test sta


sigma_test = np.array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
                       0., 0., 0., 0., 0.])



c_bg_test=113.7572667                                                           # If there
c_bg_control=607.22006143

#c_bg_test = np.log10(c_bg_test)
#c_bg_control = np.log10(c_bg_control)                                          # Control
V = 1296.048                                                                    # Chamber
#V = NA                                                                         # Chamber
```

Case #3: Hypothetical E.Coli

In [ ]:

```
t_c = np.array([0, 15, 30, 45, 60 ])                                            # Time poi
t_t = np.array([0, 15, 30, 45, 60 ])                                            # Time poi


c_test = np.array([5000, 2700, 1200, 400, 100])                                 # Test con
c_control = np.array([5000, 4200, 3000,2700, 2000])                             # Control
c_test = np.log10(c_test)
c_control = np.log10(c_control)

c_bg_test=1                                                                     # If there
```

```
c_bg_control=1
c_bg_test = np.log10(c_bg_test)
c_bg_control = np.log10(c_bg_control)


sigma_test = np.array([0,0,0,0,0])                                  # Test sta
sigma_control = np.array([0,0,0,0,0])                               # Control
V = 500                                                             # Chamber
#V = 14.2                                                           # Chamber
```

Case #4: Ratliff et al BPI

In [ ]:

```
t_c = np.array([0, 15, 30, 60, 90 ,120, 1200])                      # Time poi
t_t = np.array([0, 15, 30, 60, 90 ,120, 1200])                      # Time poi

c_test = np.array([8.46, 7.63, 7.22, 6.21, 5.89, 5.64, 10**0.1 ])   # Test cor
c_control = np.array([8.47, 7.68, 7.16, 7.09, 6.21, 6.10, 10**0.1 ])# Control

sigma_test = np.array([0.000001, 0.24 ,0.51 ,0.52 ,0.53 ,0.5, 0])       # Test
sigma_control = np.array([0.000001 ,0.36, 0.39, 0.3, 0.36, 0.49, 0])     # Cor



c_bg_test=0                                                         # If there
c_bg_control=0


V = 3000                                                            # Chamber
#V = 85                                                             # Chamber
```

Case #5: Ratliff et al PCO1

In [ ]:

```
t_c = np.array([0, 15, 30, 60, 90])                                 # Time poi
t_t = np.array([0, 15, 30, 60, 90])                                 # Time poi



c_test = np.array([ 8.11212724060371, 7.589987734, 6.745802232,
                    6.013936402, 5.394571206 ])                     # Test cor

c_control = np.array([8.46745953764963, 7.684595602, 7.163725729,
                      7.086631702, 6.211568868])                    # Control


sigma_test = np.array([ 0.108808569398572, 0.254108644, 0.169662969,
                        0.262771866,     0.047624259])                  # Test

sigma_control = np.array([      0.237966029977382, 0.358965447, 0.393189888,
                           0.301479043,  0.361953917])                  # Control



c_bg_test=0                                                         # If there
c_bg_control=0

V = 3000                                                            # Chamber
#V = 85                                                             # Chamber
```

Case #6: Ratliff et al PCO2

In [ ]:

```
t_c = np.array([0, 15, 30, 60, 90])                                          # Time poi
t_t = np.array([0, 15, 30, 60, 90])                                          # Time poi
c_test = np.array([8.032891171, 7.087704857, 6.424694684,
                   5.320557829, 4.709637767])                                # Test co

c_control = np.array([8.467459538, 7.684595602, 7.163725729,
                      7.086631702, 6.211568868])                             # Control

sigma_test = np.array([0.033101768,     0.029661622,    0.219839957,
                       0.117283171,     0.129724927])                          # Tes

sigma_control = np.array([0.23796603,   0.358965447,    0.393189888,
                          0.301479043,  0.361953917])                        # Control



c_bg_test=0                                                                  # If there
c_bg_control=0

V = 3000                                                                     # Chamber
#V = 85                                                                      # Chamber
```

Case #7: Real VOC (High Resolution) (Schumacher et. al. 2024)

In [46]:

```
# full dataset

t_c = np.array([0, 1,   2,      3,      4,      5,      6,      7,      8,      9,      10
                11,     12,     13,     14,     15, 16, 17,     18,     19,     20,
                21,     22,     23,     24,     25,     26,     27,     28,     29,     30
                31, 32, 33,     34,     35,     36,     37,     38,     39,     40,
                41,     42,     43,     44,     45,     46,     47, 48, 49,     50,
                51,     52,     53,     54,     55,     56,     57,     58,     59, 60])

t_t = np.array([0, 1,   2,      3,      4,      5,      6,      7,      8,      9,      10
                11,     12,     13,     14,     15, 16, 17,     18,     19,     20,
                21,     22,     23,     24,     25,     26,     27,     28,     29,     30
                31, 32, 33,     34,     35,     36,     37,     38,     39,     40,
                41,     42,     43,     44,     45,     46,     47, 48, 49,     50,
                51,     52,     53,     54,     55,     56,     57,     58,     59, 60])



c_test = np.array([833.75, 761.28, 658.93, 567.20, 499.72, 451.38, 422.54,
377.29, 321.06, 295.83, 223.47, 192.89, 168.51, 180.00, 154.03, 117.60, 114.59,
94.81, 86.12, 74.05, 65.78, 61.08, 55.48, 51.09, 47.80, 43.67, 38.95, 35.24,
32.66, 30.53, 28.45, 26.76, 27.44, 29.12, 23.70, 24.67, 21.88, 19.60, 18.40,
17.51, 19.38, 19.07, 16.17, 14.67, 13.91, 13.55, 12.91, 12.21, 11.86, 12.43,
12.45, 12.32, 11.57, 11.21, 11.55, 10.61, 10.93, 11.39, 11.73, 11.32, 11.30 ])

c_test = np.log10(c_test)
c_control = np.array([929.45, 951.57, 923.26, 906.75, 912.08, 933.73, 915.11,
904.99, 891.02, 898.66, 899.28, 881.89, 880.40, 889.34, 888.76, 896.80, 891.25,
880.13, 870.49, 862.30, 862.03, 866.09, 874.08, 864.19, 880.57, 872.92, 861.04,
856.26, 862.26, 863.67, 843.84, 844.73, 842.50, 838.72, 844.15, 843.70, 855.47,
834.44, 830.98, 823.93, 822.37, 825.25, 848.16, 812.26, 885.45, 863.72, 826.80,
833.43, 822.13, 809.45, 806.02, 800.97, 801.26, 795.55, 836.46, 827.70, 785.76,
786.31, 786.06, 851.69, 837.34])                                                  # Co
```

```
c_control = np.log10(c_control)
sigma_test = np.array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                       0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                       0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                       0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                       0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                       0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                       0])                                                  # Test sta


sigma_control = np.array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                          0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                          0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                          0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                          0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                          0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                          0])                                               # Test sta

c_bg_test=9.66                                                                 # If th
c_bg_control=250
#c_bg_test = np.log10(c_bg_test)
#c_bg_control = np.log10(c_bg_control)

V = 1034.72                                                                 # Chamber
#V = 29.3                                                                   # Chamber
```

Case #8: Law et al 2024 (truncated dataset)

In [9]:

```
#0-40 minutes for control and 0-25 minutes for test

t_t = np.array([0, 5, 10, 15, 20, 25])                                      # Time poi
t_c = np.array([0, 5, 10, 15, 20, 25, 30, 35, 40 ])                         # Time poi


c_test = np.array([0.958783047, 0.502557891,    0.40034575,
                   0.332838565, 0.301776099,    0.272451193])                 # Test c

c_control = np.array([0.972665911, 0.969369171, 0.959128797, 0.959302774,
                      0.95947675, 0.950973937, 0.949412555, 0.940911944,
                      0.941083718])                                         # Control

sigma_test = np.array([0, 0, 0, 0, 0, 0 ])                                  # Test sta
sigma_control = np.array([0, 0, 0, 0, 0, 0, 0, 0, 0])                       # Control

c_test = np.log10(c_test)
c_control=np.log10(c_control)


c_bg_test=0.24                                                              # If there
c_bg_control=0.65


V = 1059.44174
#V = 30
```

Case #8: Law et al 2024 (Full datset)

In [27]:

```
t_c = np.array([0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60])          # Time poi
```

```
t_t = np.array([0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60])              # Time poi


#0-60 minutes dataset (full dataset)

c_test = np.array([0.958783047, 0.502557891, 0.40034575, 0.332838565,
                   0.301776099, 0.272451193, 0.272625169, 0.251974851,
                   0.262558772, 0.262730546, 0.252494577, 0.252666351,
                   0.24242818 ])                                                 # Test cor

c_test=np.log10(c_test)

c_control = np.array([0.972665911, 0.969369171, 0.959128797, 0.959302774,
                      0.95947675, 0.950973937, 0.949412555, 0.940911944,
                      0.941083718, 0.910021252, 0.910193026, 0.899954854,
                      0.901861986])                                             # Control

c_control=np.log10(c_control)

sigma_control = np.array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])              # Control
sigma_test    = np.array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ])             # Test sta


c_bg_test=0.24                                                                  # If there
c_bg_control=0.65

V = 1059.44174                                                                  # Chamber
#V = 30                                                                         # Chamber
```

Monte Carlo Analysis

1- First-order decay method:

The loss rate has been calculated based on the following formula

<!DOCTYPE html>

$$K = \frac{\sum_{i=0}^{np} t_i \ln\left(\frac{C_i - C_bg}{C_0 - C_bg}\right)}{\sum_{i=0}^{np} t_i^2}$$

Then, loss rates will be pplugged into the CADR formula to obtain value for CADR

<!DOCTYPE html>

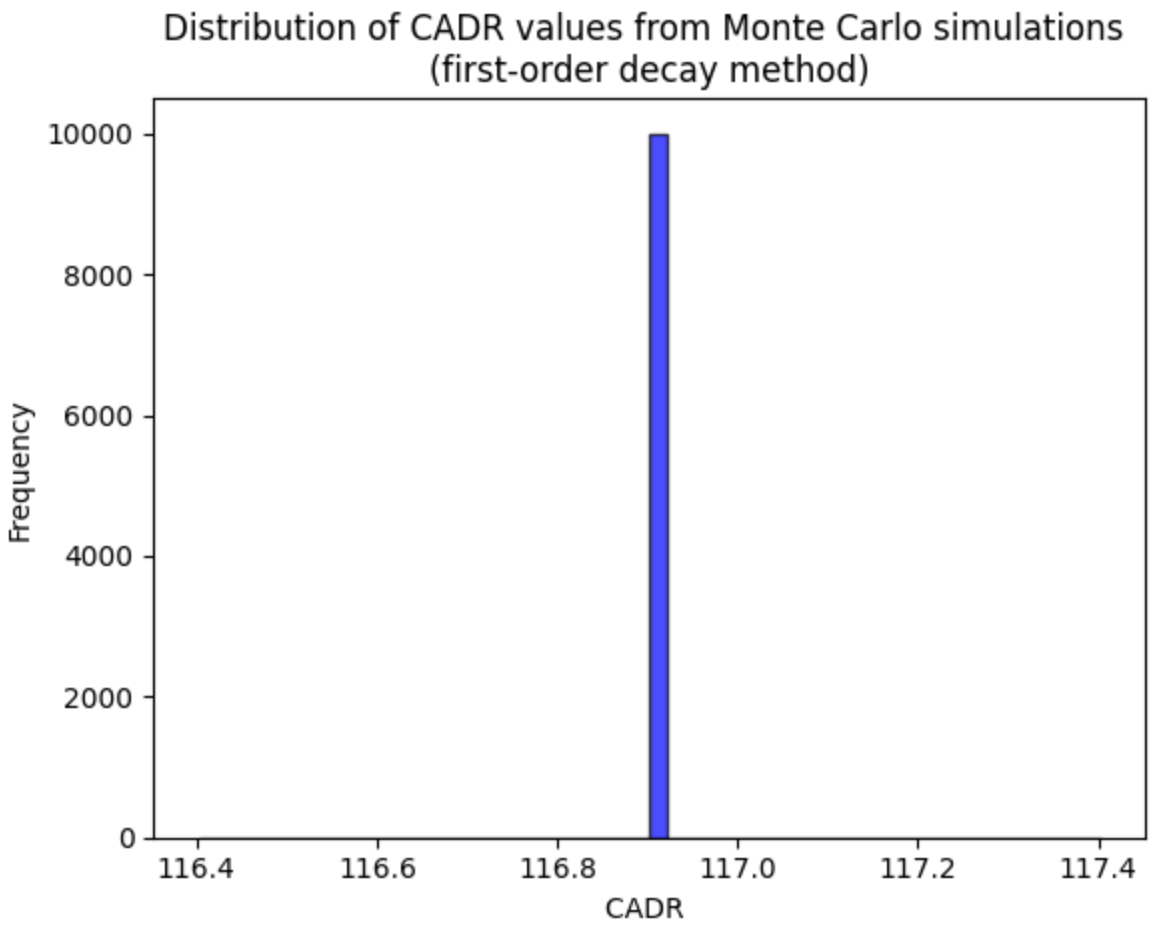
$$CADR = V(K_{Test} - K_{Control})$$

In [47]:

```
num_simulations = 10000
# Function to compute the CADR component for either test or control
def compute_component(t, c, c_bg):
    #print((c / c[0]))
    ln_c_ratio = -np.log((10**c-c_bg) / (10**c[0]-c_bg))

    numerator = np.sum((t * ln_c_ratio) )
    denominator = np.sum((t ** 2) )

    return (numerator / denominator)

# Original CADR value
CADR_original = V * (compute_component(t_t, c_test, c_bg_test) - compute_component(t_c, c_
```

```
print(f"Original CADR value = {CADR_original}","\n")


# Monte Carlo simulations
CADR_values = np.zeros(num_simulations)

for i in range(num_simulations):
    c_test_simulated = np.random.normal((c_test), (sigma_test/3))
    c_control_simulated = np.random.normal((c_control),(sigma_control/3))
    CADR_values[i] = V * (compute_component(t_t, c_test_simulated, c_bg_test) - compute_co

# Calculate mean and standard deviation of CADR
CADR_mean = np.mean(CADR_values)
CADR_std = np.std(CADR_values)

print(f"Mean CADR value from Monte Carlo simulations = {CADR_mean}","\n")
print(f"Standard deviation of CADR value from Monte Carlo simulations = {CADR_std}","\n")
print(f"Error = {100*CADR_std/CADR_mean}%","\n")

# Plotting the distribution of CADR values
plt.hist(CADR_values, bins=50, alpha=0.7, color='b', edgecolor='black')
plt.title('Distribution of CADR values from Monte Carlo simulations \n(first-order decay r
plt.xlabel('CADR ')
plt.ylabel('Frequency')

# Disable scientific notation for x-axis and y-axis
plt.ticklabel_format(style='plain', axis='x')
plt.ticklabel_format(style='plain', axis='y')

# Optionally, adjust the format of the ticks
from matplotlib.ticker import ScalarFormatter
plt.gca().xaxis.set_major_formatter(ScalarFormatter(useOffset=False))
plt.gca().yaxis.set_major_formatter(ScalarFormatter(useOffset=False))

plt.show()
```

```
Original CADR value = 116.90209908956523

Mean CADR value from Monte Carlo simulations = 116.90209908956517

Standard deviation of CADR value from Monte Carlo simulations = 5.684341886080802e-14

Error = 4.862480597312211e-14%
```



The CADR base on the integral method can be calculated as follows:

<!DOCTYPE html>

$$\mathrm{CADR}_{\mathrm{Integral}} = \frac{\mathrm{VC}_{\mathrm{test}}(0)}{\int_0^{\infty}(C_{\mathrm{test}}(t) - C_{bg,control})\,dt} - \frac{\mathrm{VC}_{\mathrm{control}}(0)}{\int_0^{\infty}(C_{\mathrm{control}}(t) - C_{bg,control})\,dt}$$

Where the integral in the denominator can be calculated using either trapezoidal or logarithmic trapezoidal method. here, we use logarithmic trapezoidal method which looks like the following:

<!DOCTYPE html>

$$\int_0^{\infty} C(t)\,dt = \sum_{n=1}^{\infty}\left(\frac{t_i - t_{i-1}}{\ln\left(\frac{C(t_{i-1})-C_{bg}}{C(t_i)-C_{bg}}\right)}(C(t_{i-1}) - C(t_i))\right)$$

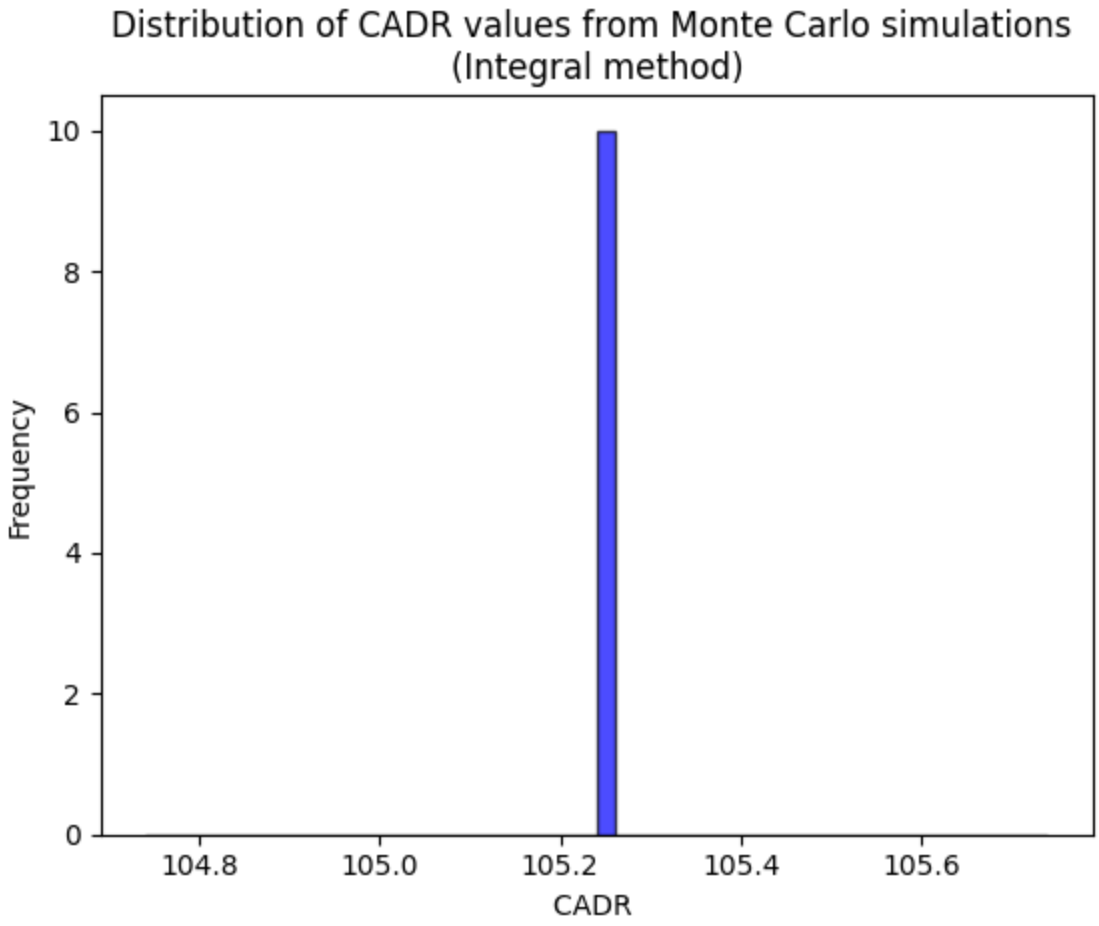
In [48]:

```
import numpy as np
import matplotlib.pyplot as plt

num_simulations = 10
# Function to compute the integral
def compute_integral(t, c, c_bg):
    N = len(t)
    integral_sum = 0.0

    for i in range(1, N):

        t_diff = t[i] - t[i-1]
        ln_c_ratio = np.log((10**c[i-1]-c_bg) / (10**c[i]-c_bg))
        #print(ln_c_ratio)
```

```
        c_diff = 10**c[i-1] - 10**c[i]
        integral_sum += (t_diff / ln_c_ratio) * c_diff
    #print(integral_sum)
    return integral_sum

# Function to compute the CADR component for either test or control
def compute_component(t, c, c_bg):

    integral_c = compute_integral(t, c, c_bg)
    #print(integral_c)
    return (10**c[0]) / integral_c


# Original CADR value
CADR_original = V* (compute_component(t_t, c_test, c_bg_test) - compute_component(t_c, c_c

print("test",compute_component(t_t, c_test, c_bg_test))
print("control",compute_component(t_c, c_control, c_bg_control))
print(f"Original CADR value = {CADR_original}\n")

# Monte Carlo simulations
CADR_values = np.zeros(num_simulations)
'''
for i in range(num_simulations):
    c_test_simulated = np.random.normal(c_test, sigma_test)
    plt.plot(t_t, c_test_simulated)
    c_control_simulated = np.random.normal(c_control, sigma_control)
    plt.plot(t_c, c_control_simulated)

    CADR_values[i] = V* (compute_component(t_t, c_test_simulated) - compute_component(t_c,
#print(CADR_values)
plt.show()
'''

#fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 6))

for i in range(num_simulations):
    c_test_simulated = np.random.normal(c_test, sigma_test/3)
    c_control_simulated = np.random.normal(c_control, sigma_control/3)

    #ax1.plot(t_t, c_test_simulated, alpha=0.1)
    #ax2.plot(t_c, c_control_simulated, alpha=0.1)

    CADR_values[i] = V * (compute_component(t_t, c_test_simulated, c_bg_test) - compute_cc
# Customize plots
'''
ax1.set_title('Test Simulations')
ax1.set_xlabel('Time')
ax1.set_ylabel('Concentration')

ax2.set_title('Control Simulations')
ax2.set_xlabel('Time')
ax2.set_ylabel('Concentration')

plt.tight_layout()
plt.show()
'''
# Print CADR values if needed
# print(CADR_values)


# Calculate mean and standard deviation of CADR
CADR_mean = np.mean(CADR_values)
CADR_std = np.std(CADR_values)

print(f"Mean CADR value from Monte Carlo simulations = {CADR_mean}\n")
```

```
print(f"Standard deviation of CADR value from Monte Carlo simulations = {CADR_std}\n")
print(f"Error = {CADR_std / CADR_mean * 100}%\n")

# Plotting the distribution of CADR values
plt.hist(CADR_values, bins=50, alpha=0.7, color='b', edgecolor='black')
plt.title('Distribution of CADR values from Monte Carlo simulations \n(Integral method)')
plt.xlabel('CADR ')
plt.ylabel('Frequency')

# Disable scientific notation for x-axis and y-axis
plt.ticklabel_format(style='plain', axis='x')
plt.ticklabel_format(style='plain', axis='y')

# Optionally, adjust the format of the ticks
from matplotlib.ticker import ScalarFormatter
plt.gca().xaxis.set_major_formatter(ScalarFormatter(useOffset=False))
plt.gca().yaxis.set_major_formatter(ScalarFormatter(useOffset=False))

plt.show()
```

```
test 0.1272083774282268
control 0.025499043930522245
Original CADR value = 105.24068155674486

Mean CADR value from Monte Carlo simulations = 105.24068155674486

Standard deviation of CADR value from Monte Carlo simulations = 0.0

Error = 0.0%
```



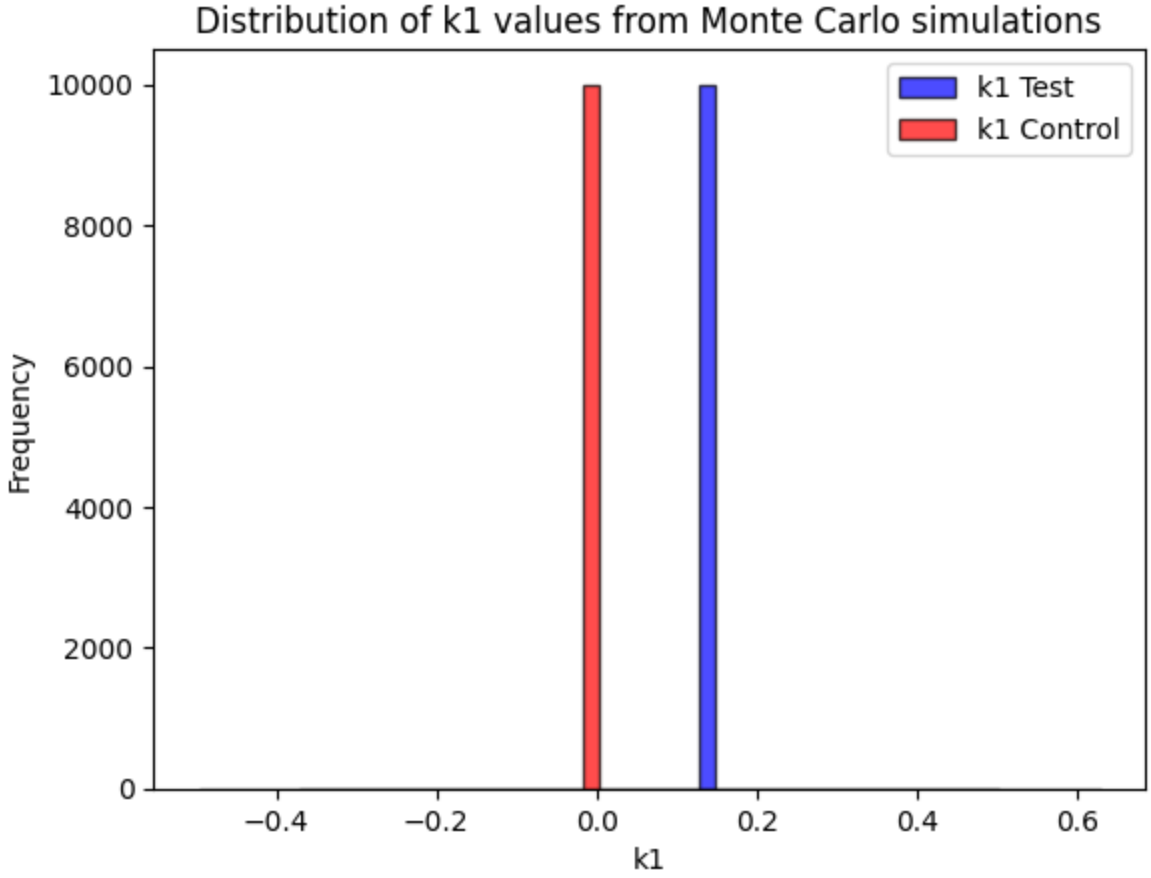
In [11]:

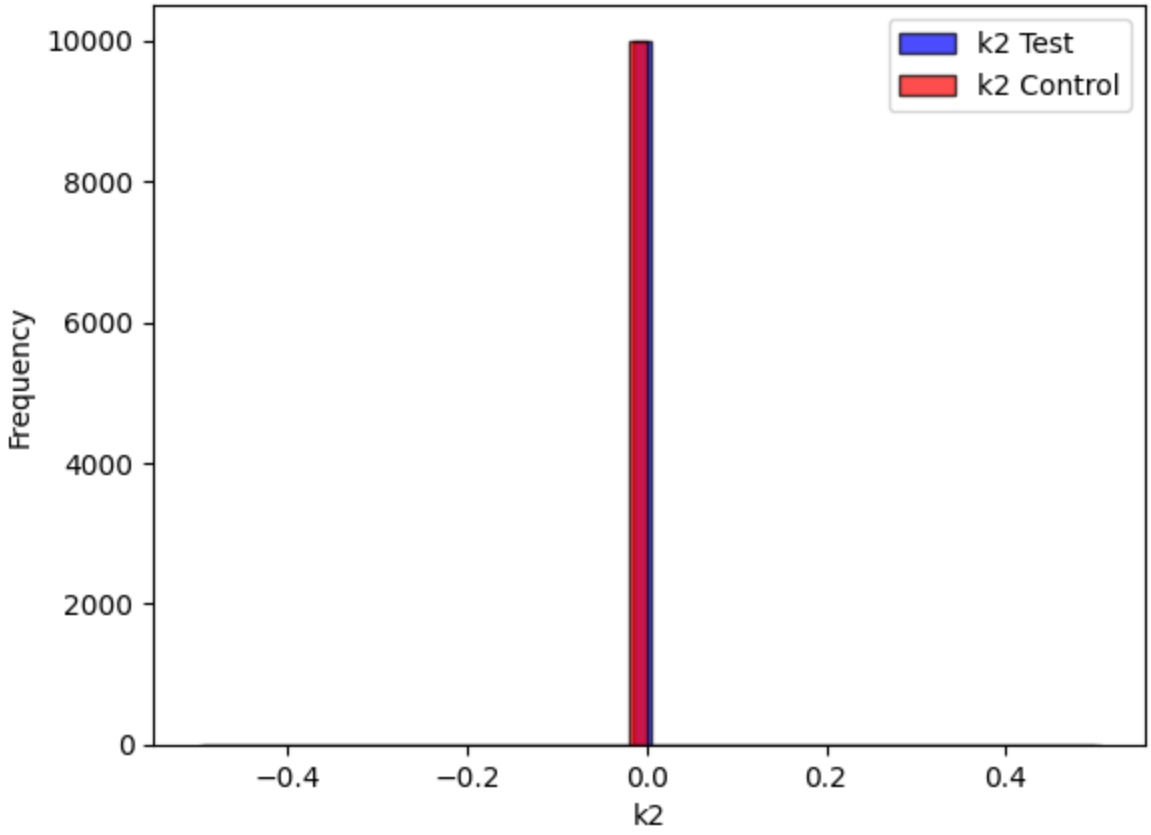
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import t
from scipy import stats
```

```
num_simulations = 10000

# Define the compute_k1 function using the first part of the new formula
def compute_k1(t, c,c_bg):
    ln_c_ratio = -np.log((10**c-c_bg) / (10**c[0]-c_bg))
    numerator = np.sum(t * ln_c_ratio)
    denominator = np.sum(t ** 2)
    return numerator / denominator

# Define the compute_k2 function using the second part of the new formula
def compute_k2(t, c, k,c_bg, alpha=0.05):
    ln_c_ratio = -np.log((10**c-c_bg) / (10**c[0]-c_bg))

    # Calculate the mean of the time points
    t_mean = np.mean(t)

    # Calculate the second term
    residual_sum = np.sum(((ln_c_ratio - k * t)) ** 2)
    #print( k)
    critical_value = stats.t.ppf(1- alpha / 2, len(t) - 1)
    #second_term = critical_value * np.sqrt(residual_sum / ((len(t) - 1) * np.sum((t) ** 2
    second_term = np.sqrt(residual_sum / ((len(t) - 1) * np.sum((t) ** 2)))

    return second_term


# Monte Carlo simulations for the first formula


k1_values_test = np.zeros(num_simulations)
k1_values_control = np.zeros(num_simulations)

for i in range(num_simulations):
    c_test_simulated = np.random.normal(c_test, sigma_test / 3)
    c_control_simulated = np.random.normal(c_control, sigma_control / 3)
    k1_values_test[i] = compute_k1(t_t, c_test_simulated,c_bg_test)
    k1_values_control[i] = compute_k1(t_c, c_control_simulated, c_bg_control)

# Monte Carlo simulations for the second formula
k2_values_test = np.zeros(num_simulations)
k2_values_control = np.zeros(num_simulations)

for i in range(num_simulations):
    c_test_simulated = np.random.normal(c_test, sigma_test / 3)
    c_control_simulated = np.random.normal(c_control, sigma_control / 3)
    k_test = compute_k1(t_t, c_test_simulated, c_bg_test)
    k_control = compute_k1(t_c, c_control_simulated, c_bg_control)
    k2_values_test[i] = compute_k2(t_t, c_test_simulated, k_test, c_bg_test)
    k2_values_control[i] = compute_k2(t_c, c_control_simulated, k_control, c_bg_control)

# Calculate mean and standard deviation for both formulas
k1_mean_test = np.mean(k1_values_test)
k1_std_test = np.std(k1_values_test)
k1_mean_control = np.mean(k1_values_control)
k1_std_control = np.std(k1_values_control)

k2_mean_test = np.mean(k2_values_test)
k2_std_test = np.std(k2_values_test)
k2_mean_control = np.mean(k2_values_control)
k2_std_control = np.std(k2_values_control)

print(f"Mean k1 value for test from Monte Carlo simulations = {k1_mean_test}\n")
print(f"Standard deviation of k1 value for test from Monte Carlo simulations = {k1_std_tes
print(f"Mean k1 value for control from Monte Carlo simulations = {k1_mean_control}\n")
print(f"Standard deviation of k1 value for control from Monte Carlo simulations = {k1_std_
```

```
print(f"Mean k2 value for test from Monte Carlo simulations = {k2_mean_test}\n")
print(f"Standard deviation of k2 value for test from Monte Carlo simulations = {k2_std_tes
print(f"Mean k2 value for control from Monte Carlo simulations = {k2_mean_control}\n")
print(f"Standard deviation of k2 value for control from Monte Carlo simulations = {k2_std_

# Plotting the distribution of k1 values
plt.hist(k1_values_test, bins=50, alpha=0.7, color='b', edgecolor='black', label='k1 Test'
plt.hist(k1_values_control, bins=50, alpha=0.7, color='r', edgecolor='black', label='k1 Co
plt.title('Distribution of k1 values from Monte Carlo simulations')
plt.xlabel('k1')
plt.ylabel('Frequency')
plt.legend()
plt.show()

# Plotting the distribution of k2 values
plt.hist(k2_values_test, bins=50, alpha=0.7, color='b', edgecolor='black', label='k2 Test'
plt.hist(k2_values_control, bins=50, alpha=0.7, color='r', edgecolor='black', label='k2 Co
plt.title('Distribution of k2 values from Monte Carlo simulations')
plt.xlabel('k2')
plt.ylabel('Frequency')
plt.legend()
plt.show()
```

```
Mean k1 value for test from Monte Carlo simulations = 0.12891970374259107

Standard deviation of k1 value for test from Monte Carlo simulations = 2.7755575615628914e
-17

Mean k1 value for control from Monte Carlo simulations = 0.0026821435337412926

Standard deviation of k1 value for control from Monte Carlo simulations = 4.33680868994201
8e-19

Mean k2 value for test from Monte Carlo simulations = 0.005654409953692421

Standard deviation of k2 value for test from Monte Carlo simulations = 8.673617379884035e-
19

Mean k2 value for control from Monte Carlo simulations = 0.0001176333295346026

Standard deviation of k2 value for control from Monte Carlo simulations = 4.06575814682064
16e-20
```


Distribution of k1 values from Monte Carlo simulations
k1 Test
k1 Control
Frequency
k1
10000
8000
6000
4000
2000
0
−0.4
−0.2
0.0
0.2
0.4
0.6


Distribution of k2 values from Monte Carlo simulations
k2 Test
k2 Control
Frequency
k2
10000
8000
6000
4000
2000
0
−0.4
−0.2
0.0
0.2
0.4

In [ ]: