In [5]:
```python
from numpy import mean, array, cos, sin, arctan, log, sqrt, linspace
import numpy as np
from math import pi
from scipy.optimize import curve_fit, OptimizeWarning, fsolve
import warnings

porosity = 0.60
heat_capacity = 4190000
heat_capacity_solid = 2000000
diffusivity = 'TODO'
conductivity = 0.84
beta = 0.001
shallow_mid_dist = 0.1
str_temp_time_to = """7/12/2014 0:00,7/12/2014 1:00,7/12/2014 2:00,7/12/2014 3:00,7/12/2014 4:00,
                      7/12/2014 5:00,7/12/2014 6:00,7/12/2014 7:00,7/12/2014 8:00,7/12/2014 9:00,
                      7/12/2014 10:00,7/12/2014 11:00,7/12/2014 12:00,7/12/2014 13:00,7/12/2014 1
                      7/12/2014 15:00,7/12/2014 16:00,7/12/2014 17:00,7/12/2014 18:00,7/12/2014 1
                      7/12/2014 20:00,7/12/2014 21:00,7/12/2014 22:00,7/12/2014 23:00,7/13/2014 0
str_temp_to = """24.51,24.38,24.20,24.07,23.88,23.70,23.51,23.45,23.45,23.63,23.95,24.26,24.51,
                 24.70,24.82,24.88,24.88,24.88,24.76,24.63,24.51,24.45,24.32,24.20,24.13"""
str_temp_time_tz = """7/12/2014 0:00,7/12/2014 1:00,7/12/2014 2:00,7/12/2014 3:00,7/12/2014 4:00,
                      7/12/2014 5:00,7/12/2014 6:00,7/12/2014 7:00,7/12/2014 8:00,7/12/2014 9:00,
                      7/12/2014 10:00,7/12/2014 11:00,7/12/2014 12:00,7/12/2014 13:00,7/12/2014 1
                      7/12/2014 15:00,7/12/2014 16:00,7/12/2014 17:00,7/12/2014 18:00,7/12/2014 1
                      7/12/2014 20:00,7/12/2014 21:00,7/12/2014 22:00,7/12/2014 23:00,7/13/2014 0
str_temp_tz = """21.07,21.07,21.07,21.07,21.01,21.01,20.94,20.94,20.88,20.82,20.82,20.82,20.82,
                 20.82,20.88,20.88,20.94,20.94,21.01,21.01,21.07,21.07,21.07,21.07,21.01"""
expected = 10.3008
```

In [6]:
```python
def get_frequency(data):
    """Derive the frequency to be used in the transient functions."""
    # Frequency is derived from the frequency of the data. Hourly is 1/24, every two hours might 
    # One of the examples given works well with frequency = 1/24
    # All the tests seem to utilize data from a 24 hour period, so the frequency might always be 
    return 2 / (len(data) - 1)


def parse_csv_str(csv_str, data_type):
    """
    Receiving a CSV string.

    This function ignores all newlines and tabs before parsing
    the data into arrays of type 'datetime' or 'float'.
    """
    # Strip all special (newline, tab, etc.) characters.
    csv_str = csv_str.replace('\n', '')
    csv_str = csv_str.replace('\r', '')
    csv_str = csv_str.replace('\t', '')
    csv_str = csv_str.rstrip(',')

    # Split on commas
    if data_type == 'datetime':
        return csv_str.split(',')

    elif data_type == 'float':
        # Data type is numerical (float), so we need to parse each value
        csv_array = [float(x) for x in csv_str.split(',')]
        return csv_array


def transient_ydata_func(parms, a, b, c, d, e, f, g, h):
    """Solved to discover the optimal seepage parameters."""
    return (a * cos((parms[2] * 1) * pi * parms[0]) + b * sin((parms[2] * 1) * pi * parms[0]) +
            c * cos((parms[2] * 2) * pi * parms[0]) + d * sin((parms[2] * 2) * pi * parms[0]) +
            e * cos((parms[2] * 3) * pi * parms[0]) + f * sin((parms[2] * 3) * pi * parms[0]) +
            g * cos((parms[2] * 4) * pi * parms[0]) + h * sin((parms[2] * 4) * pi * parms[0]) + pa
```

In [7]:
```python
# Parse the time series strings into arrays of strings (dates) and floats (temperatures)
temp_time_to = parse_csv_str(str_temp_time_to, 'datetime')
temp_time_tz = parse_csv_str(str_temp_time_tz, 'datetime')

try:
    temp_to = parse_csv_str(str_temp_to, 'float')
    temp_tz = parse_csv_str(str_temp_tz, 'float')
except ValueError:
    print("Could not parse provided time series data. Please check your input.")
```

```
    if len(temp_time_to) + len(temp_time_tz) + len(temp_to) + len(temp_tz) < 12 * 4:
        print("Not enough time series data provided. Please check your input. There must be at least 1
    elif len(temp_time_to) != len(temp_time_tz) or len(temp_time_tz) != len(temp_to) or len(temp_to)
        print("Please check your input. The four time series inputs should have the same number of poi
```

In [8]:
```
RN1 = len(temp_to)
RN2 = len(temp_tz)
RN3 = 15
RN4 = 15

for i in range(16, RN1, 1):
    if temp_to[i] - temp_to[0] <= 1:
        RN3 = RN3 + 1

for i in range(16, RN2, 1):
    if temp_tz[i] - temp_tz[0] <= 1:
        RN4 = RN4 + 1

RN5 = len(temp_to)
RN6 = len(temp_tz)

# Ao for To and Tz:
L15 = mean(temp_to)
L20 = mean(temp_tz)

frequency = get_frequency(temp_to)

# xdata = linspace(0, 24, 25)
xdata = linspace(0, len(temp_to) - 1, len(temp_to))
```

In [9]:
```
# https://stackoverflow.com/questions/31301017/catch-optimizewarning-as-an-exception
with warnings.catch_warnings():
    warnings.simplefilter("error", OptimizeWarning)
    try:
        A1B1_calculated, pcov = curve_fit(transient_ydata_func, [xdata, L15, frequency], temp_to)
        A2B2_calculated, pcov = curve_fit(transient_ydata_func, [xdata, L20, frequency], temp_tz)
    except OptimizeWarning:
        print("Covariance of the parameters can not be estimated.")
    except RuntimeError:
        print("Least-Squares minimization has failed.")
```

In [10]:
```
# Run McCallum method:
P = 1  # Period (days)

# calculate amplitude & phase angle of the shallow depth
A1 = A1B1_calculated[0]
B1 = A1B1_calculated[1]
Po = 0
Pz = 0

if A1 == 0:
    A1 = 1E-99
if B1 == 0:
    B1 = 1E-99

Ao = (A1 ** 2 + B1 ** 2) ** 0.5
if A1 < 0:
    Po = arctan(B1 / A1) + pi
else:
    Po = arctan(B1 / A1)

# calculate amplitude & phase angle of the deeper depth
A2 = A2B2_calculated[0]
B2 = A2B2_calculated[1]

if A2 == 0:
    A2 = 1E-99
if B2 == 0:
    B2 = 1E-99

Az = (A2 ** 2 + B2 ** 2) ** 0.5
if A2 < 0:
    Pz = arctan(B2 / A2) + pi
else:
    Pz = arctan(B2 / A2)
```

```python
    # Calculate Amplitude ratio & phase shift
    AR = Az / Ao
    PS = (Pz - Po) / (2 * pi)  # phase shift unit is "day"
    # Needed to change from < to <= in the case of 0, which resulted in NaN.
    if PS <= 0:
        PS = (2 * pi + Pz - Po) / (2 * pi)

    # ***start McCallum (2012) calculation***
    dz = shallow_mid_dist  # depth
    n = porosity  # porosity
    PfCf = heat_capacity  # volumetric heat capacity of fluid
    PsCs = heat_capacity_solid  # volumetric heat capacity of solid
    PC = n * PfCf + (1 - n) * PsCs  # heat capacity of saturated media
    r = PC / PfCf
    LnAR = log(AR)

    # thermal front velocity
    v = dz * (P ** 2 * LnAR ** 2 - 4 * pi ** 2 * PS ** 2) / (
            PS * (16 * pi ** 4 * PS ** 4 + 8 * P ** 2 * pi ** 2 * PS ** 2 * LnAR ** 2 + P ** 4 * LnAR

    # Darcy velocity (seepage flux)
    qz = v * r
    X12 = qz * 100  # in cm/day

    # calculate De (effective thermal diffusivity)
    De = (dz ** 2 * P ** 2 * LnAR * (4 * pi ** 2 * PS ** 2 - P ** 2 * LnAR ** 2)) / (
            PS * (P ** 2 * LnAR ** 2 + 4 * pi ** 2 * PS ** 2) * (P ** 2 * LnAR ** 2 - 4 * pi ** 2 * P

    # 'calculate thermal conductivity k
    Beta = beta
    ke = (De - beta * abs(v)) * PC / 86400

    print(f'Seepage:      {round(X12, 4)}')
    print(f'Diffusivity:  {round(De, 4)}')
    print(f'Conductivity: {round(ke, 4)}')
```

```
Seepage:      -0.1658
Diffusivity:  0.0137
Conductivity: 0.5243
```

In [ ]: