```
    )
    plt.figure()


# In[ ]:


# Function to calculate the autocorrelation of a variable
def calculate_autocovariation(data, N_range, T_range, no_of_sweeps,
no_of_equilibrating_sweeps):
    no_of_sampling_sweeps = no_of_sweeps-no_of_equilibrating_sweeps
    autocovariation = np.zeros((len(N_range),len(T_range),no_of_sampling_sweeps))

    for N_index in range(len(N_range)):
        for T_index in range(len(T_range)):
            ave = np.sum(data[N_index,T_index,:])/no_of_sampling_sweeps
            for tau in range(no_of_sampling_sweeps):
                autocorr = 0.0
                for i in range(no_of_equilibrating_sweeps,no_of_sweeps-tau):
                    autocorr +=
                    (data[N_index,T_index,i]-ave)*(data[N_index,T_index,i+tau]-ave)
                autocovariation[N_index,T_index,tau] = autocorr
            if autocovariation[N_index,T_index,0] == 0:
                autocovariation[N_index,T_index,:] = np.ones(no_of_sampling_sweeps)
            else:
                autocovariation[N_index,T_index,:] =
                autocovariation[N_index,T_index,:]/autocovariation[N_index,T_index,0]

    return autocovariation


# In[ ]:


# Function of magnetisation near Tc
def shape_function(x, a, Tc, b):
        return a*(((Tc-x)/Tc)**b)


# In[ ]:


# Function to fit data
def data_fitting(data, N_range, N_index_range, T_range, T_index_range, guess):
    x_data = np.zeros((len(N_index_range),len(T_index_range)))
    y_data = np.zeros((len(N_index_range),len(T_index_range))) # data sliced by
    N_index_range and T_index_range
    for N_index in range(len(N_index_range)):
        for T_index in range(len(T_index_range)):
            x_data[N_index,T_index] = T_range[T_index_range[T_index]]
            y_data[N_index,T_index] = data[N_index_range[N_index],T_index_range[T_index]]

    params = np.zeros((len(N_index_range),3))
    errs = np.zeros((len(N_index_range),3))

    for N_index in range(len(N_index_range)):
        popt, pcov =
        optimize.curve_fit(shape_function,x_data[N_index,:],y_data[N_index,:],guess)
        params[N_index,:] = popt
        errs[N_index,:] = pcov.diagonal()

    return params, errs


# In[ ]:


```