Supporting Information

J.Chem. Metrol. **X:X (202X) XX-XX**

Uncertainty of small enthalpy effects measured by isothermal calorimetric titration

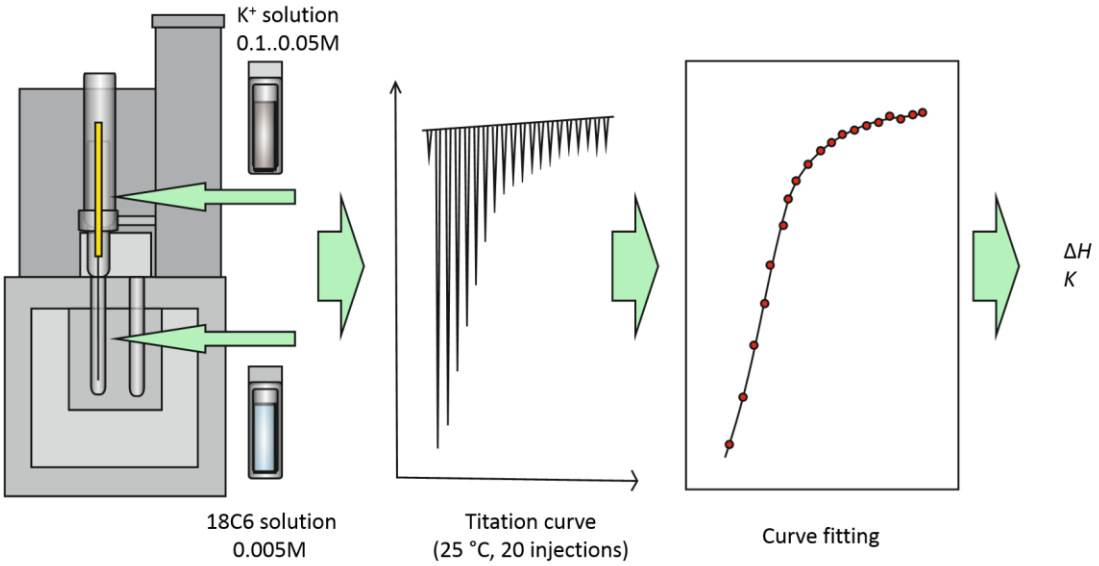
Astrid Darnell[1*], Lauri Sikk[1], Ly Porosk[2] and Ivo Leito[1]

[1]University of Tartu, Institute of Chemistry, Ravila 14A, Tartu, Estonia

[2]University of Tartu, Institute of Technology, Nooruse 1, Tartu, Estonia

Table Contents **Page**

1. **Experimental scheme for isothermal titration calorimetry**



2. **Component-by-component approach to measurement uncertainty estimation [1]**

1. Specification of the measured quantity and choosing a suitable mathematical model best describes the measured quantity and experimental conditions.

2. Identifying and quantifying the uncertainty contributions of the input quantities of the mathematical model.

3. Using the mathematical model and the uncertainty estimates of the input quantities, estimating the combined standard uncertainty.

4. Presenting the final result together with the uncertainty estimate.

3. **The equations used for computational estimation of heat effects of the calorimetric experiments [2]**

Equation (1) is used to calculate the heat effect of binding:

$$Q = \frac{nM_t\Delta HV_0}{2}\left[1 + \frac{X_t}{nM_t} + \frac{1}{nKM_t} - \sqrt{\left(1 + \frac{X_t}{nM_t} + \frac{1}{nKM_t}\right)^2 - \frac{4X_t}{nM_t}}\right] \quad (.)$$

In equation (1), Here, Q describes the total (cumulative) measurable heat effect of the process in the solution in the active volume of the calorimetric cell, n is the stoichiometry coefficient of the reaction, M_t is the bulk concentration of the receptor, ΔH the molar enthalpy effect of the binding reaction, V_0 is the active volume of calorimeter, X_t is the bulk concentration of the cation, and K is the equilibrium constant of the binding reaction.

Equation (2) is used to estimate the heat effect of the i-th experimental injection $\Delta Q(\mathrm{i})$:

$$\Delta Q(\mathrm{i}) = Q(\mathrm{i}) + \frac{v_\mathrm{i}}{V_0}\left[\frac{Q(\mathrm{i})+Q(\mathrm{i}-1)}{2}\right] - Q(\mathrm{i}-1) \qquad (.)$$

In equation (2), Q(i) and Q(i-1) are the cumulative heat effects after the titrant injections i and i-1, respectively, both found from eq (.). The parameter v_i is the injection volume of the i-th injection. The second term of the equation is used to account for the effect of displacing some of the solution from the active volume into the overflow compartment.

Equation (3) is used to estimate the active concentration of cell compound, M_t:

$$M_\mathrm{t} = M_\mathrm{t0}\left[\frac{1-\frac{V\mathrm{tot_i}}{2V_0}}{1+\frac{V\mathrm{tot_i}}{2V_0}}\right] \qquad (.)$$

Equations (4) and (5) are used to estimate the active concentration of the syringe compound, X_t:

$$X_\mathrm{t} = X_\mathrm{t0}\left[\frac{1}{1+\frac{V\mathrm{tot_i}}{2V_0}}\right] \qquad (.)$$

$$X_\mathrm{t0} = c_\mathrm{syr}\left[\frac{V\mathrm{tot_i}}{V_0}\right] \qquad (.)$$

In equations (3)-(5), , X_t0 and M_t0 are respective bulk concentrations before accounting for the volume effect. For the determination of X_t0, c_syr is the concentration of titrant solution in the calorimeter`s syringe, $V_\mathrm{tot,i}$ is the total injection volume after the i-th injection and V_0 the active volume of the calorimetric cell.

4. Uncertainty estimation for the experimentally determined heat effects

In order to carry out the estimation of measurement uncertainty of the heat effects measured for individual injections, the following main groups of uncertainty contributions were identified and evaluated:

1) Uncertainty of solution concentrations from making and diluting the solutions (weighing, volumetry, reagent purity);
2) Uncertainty of titrant volume delivery during the experiments;
3) Uncertainty from instrumental limits of heat effect measurement;
4) Uncertainty of calorimeter cell volume V_0 and its effect on calculations of the active concentrations and thus the heat effects of the binding reactions;
5) Uncertainty of accounting for the effects of heat of dilution;
6) Uncertainty of peak area integration.

The computational estimates of experimental heat effects of individual injections and their uncertainties are presented in Table 1. The component-by-component uncertainty evaluation enables obtaining uncertainty budgets for every individual injection, which are displayed in Table 2.

5. Experimental and computational peak areas, their uncertainty estimates and values for the subtracted heat of dilution effects, experiment K2.

Table S1. Example of experiment K2- the experimental and computational values of heat effects and their uncertainty estimates.

Computational heat effects and uncertainty estimates			Experimental heat effect and heat of dilution	
Inj. no.	ΔQ_i (μcal)	$u_{\Delta Qi}$ (μcal)	ΔQ_{i_exp}-$\Delta Q_{i,h.o.d,exp}$ (μcal)	$\Delta Q_{i\_h.o.d.,exp}$ (μcal)
1	-16.61	0.75	-14.25	-0.48
2	-161.25	5.35	-167.31	-5.45
3	-152.00	6.57	-155.07	-4.51
4	-142.52	7.34	-143.62	-3.90
5	-133.54	7.82	-134.11	-3.74
6	-125.05	8.12	-126.36	-3.51
7	-117.04	8.29	-117.32	-3.16
8	-109.47	8.34	-110.25	-3.02
9	-102.33	8.32	-102.83	-2.81
10	-95.60	8.24	-95.96	-2.87
11	-89.25	8.10	-90.12	-2.53
12	-83.27	7.93	-84.38	-2.50
13	-77.63	7.72	-79.06	-2.28
14	-72.32	7.49	-73.71	-2.36
15	-67.31	7.24	-69.59	-2.06
16	-62.60	6.98	-65.17	-1.98
17	-58.16	6.71	-61.30	-1.95
18	-53.97	6.43	-57.87	-1.93
19	-50.03	6.14	-54.03	-1.97
20	-46.32	5.86	-50.94	-1.78

h.o.d.- heat of dilution

6. Experimentally determined heat effects, estimates of measurement uncertainty for the heat effects and the heat of dilution correction values

Table S2. Uncertainty budgets for injections 1-20 for experiment K2.[a]

Inj. No	V_0	c_K	c_{18C6}	$V_{tot,}$(i)	$V_{tot,}$(i-1)
1	0%	**39%**	21%	**39%**	-
2	0%	**38%**	**23%**	**38%**	0%
3	0%	**20%**	14%	**42%**	**24%**
4	0%	**13%**	10%	**44%**	**33%**
5	1%	**9%**	8%	**45%**	**38%**
6	1%	6%	**7%**	**45%**	**41%**
7	1%	5%	**6%**	**45%**	**43%**
8	1%	4%	**5%**	**45%**	**44%**
9	2%	3%	**5%**	**45%**	**45%**
10	2%	2%	**5%**	**45%**	**46%**
11	2%	2%	**4%**	**45%**	**46%**
12	2%	1%	**4%**	**45%**	**47%**
13	3%	1%	**4%**	**45%**	**47%**
14	3%	1%	**4%**	**45%**	**47%**
15	3%	1%	**3%**	**45%**	**47%**
16	**4%**	1%	3%	**45%**	**47%**
17	**4%**	0%	3%	**45%**	**48%**
18	**4%**	0%	3%	**44%**	**48%**
19	**5%**	0%	3%	**44%**	**48%**
20	**5%**	0%	3%	**43%**	**48%**

V_0- active volume of the calorimetric cell; c_K- syringe concentration of ligand; c_{18C6}-cell concentration of receptor; ; V_{tot},i- total cumulative injection volume after injection i; V_{tot},i-1-total cumulative injection volume after injection i-1;

The three main contributors to the uncertainty of a given experimental heat effect are marked in bold.

[a]The uncertainty contributions of v_i, volume of injection i; V_{tot}(i-2), total injection volume at injection i-2 were close to 0% for all injections and have been omitted from the table.

7. Description of the custom Python script and the procedure for data fitting

The script works in the following steps:

- The script requires the experimental concentrations, experimentally determined peak areas, their estimated measurement uncertainties and experimental injection volumes as the input data ;
- The script also requires an initial estimate of K and ΔH values;
- The script will start with the initial estimate of K and ΔH value and calculate the computationally estimated heats for each injection of the titration
- The script will estimate the sum of squares of the differences between the experimental and computational heat effect values
- The script will minimize the sum of squares to find ΔH and $\log K$ estimates
- The script will estimate the uncertainty by the following steps
    - Start with the obtained K and ΔH values from previous fitting routine
    - Calculate the theoretical curve using these parameter values
    - Calculate the Jacobian, its weighted form (weighted by the standard deviation of each experimental data point) and transpose the matrix.
    - Calculate the covariance matrix and its inverse
    - Calculate the weighted sum of squares and estimate the goodness of fit through the chi-square: chi^2/degrees of freedom
    - Estimate the expanded uncertainty of fit for K and ΔH values by finding the standard uncertainty from using the covariance matrix and multiplying that by a suitable coverage factor to estimate approx. 95% confidence interval.
- The fitting routine is carried out once more with the experimentally determined peak area uncertainties estimated with the K and ΔH values estimated by the initial fit. This is to check that the estimated parameter values do not cause a notable change in the peak uncertainties and results of data fitting. When difference of less than 1% is observed between the results obtained from two consecutive iterations of the script, the result is deemed conclusive.

8. Script used for data fitting

```
#! /usr/bin/python3
# -*- coding: utf-8 -*-

from scipy.optimize import fsolve
from scipy.optimize import minimize
import matplotlib.pyplot as plt
import numpy as np
import sys
import math

# reaction
# M + L --> ML; K1, dH1

################################################
#initial guess of the K and dH values (K1,dH1)
initial_guess=[135,-26.5]
diff_K_delta=0.01
diff_dH_delta=0.01
cellVolume=0.2006*10**(-3) #L; working volume of calorimeter

#number of first injections to remove from fitting
n_remove=1
################################################

#file name and location for script input data
infile_name="C:/Users/Astrid/OwnCloud/ITC_NMR_2020/ITC/Scripts/exp_data_real.txt"

#experimental parameters: volM; concM; concL
#(active volume; initial cell cocentration; initial syringe concentration)
exp_parameters=[]
exp_parameters.append(0.2006*10**(-3))
exp_parameters.append(0.005029392)
exp_parameters.append(0.050411873)
#print(exp_parameters)

with open(infile_name,"r") as infile:
        lines=infile.readlines()

#extract experimental Q list and injection volumes from input data
expQ_list=[]
injections=[]
stdDev=[]
for i in range(1,len(lines)):
```

```
        expQ_list.append(float(lines[i].rstrip().split()[1]))
        injections.append(float(lines[i].rstrip().split()[2]))
        stdDev.append(float(lines[i].rstrip().split()[3]))

#calculate total volume injected at each peak
injVol=[]
for i in range(len(injections)):
        tempVol=0
        for j in range(i+1):
                tempVol+=injections[j]
        injVol.append(tempVol)

#create new experimental Qlist with n_remove points removed for calculating SS
expQ_pruned_list=expQ_list.copy()
z=0
while z<n_remove:
        expQ_pruned_list.pop(0)
        z+=1

#function for calculating sum of squares
def calc_SS(list1,list2):
        SS=0
        if len(list1)!=len(list2):
                print("unequal size of list in sum of squares calculation: "+str(len(list1))+"
"+str(len(list2)))
                return False
        else:
                for i in range(len(list1)):
                        SS+=(list1[i]-list2[i])**2
                return SS

#function for calculating sum of squares weighted by 1/stdev of experimental data points
def calc_SS_stddev_weighted(list1,list2):
        SS=0
        if len(list1)!=len(list2):
                print("unequal size of list in sum of squares calculation: "+str(len(list1))+"
"+str(len(list2)))
                return False
        else:
                for i in range(len(list1)):
                        SS+=((list1[i]-list2[i])**2)/stdDev[i]
                return SS

#function for calculating analytical solution for equilibrium constant equation

#function for calculating heats
#This function returns results identical to Excel caculations
```

```
def calc_heats(K1,dH1,volM,concM,concL):
	list_Q_working=[]
	for i in range(len(injections)):
		#calculate Q at injection i
		#calculate L0 at injection i (active concentration of titrant)
		L0=injVol[i]*concL/(cellVolume)*(1/(1+(injVol[i]/(2*cellVolume))))
		#calculate M0 at injection i (active concentration of cell compound)
		M0=((cellVolume*concM-0.5*concM*injVol[i])/(cellVolume+0.5*(injVol[i])))
		a=1+(L0/(M0))+(1/(K1*M0))
		Qj=(M0*dH1*cellVolume/2)*(a-(a*a-4*L0/(M0))**0.5)
		if i==0:
			Qj_1=0
		else:
			#calculate L0 at injection i-1
			L1=injVol[i-1]*concL/(cellVolume)*(1/(1+(injVol[i-1]/(2*cellVolume))))
			#calculate M0 at injection i-1
			M1=((cellVolume*concM-0.5*concM*injVol[i-1])/(cellVolume+0.5*(injVol[i-
1])))
			a=1+(L1/(M1))+(1/(K1*M1))
			Qj_1=((M1*dH1*cellVolume/2)*(a-(a*a-4*L1/(M1))**0.5))

		#calculate dQ
		dQ=(Qj-Qj_1+(0.5*(Qj-Qj_1)*(injections[i]/(cellVolume))))
		#6 calculate heat effect of injection
		list_Q_working.append(dQ)
	#convert to ucal
		list_Q_working_ucal=[]
	for i in list_Q_working:
	list_Q_working_ucal.append((i*1*10**9)/4.184)

	#print(list_Q_working_ucal)
	return list_Q_working_ucal

#function for minimizing Sum of Squares
def minFunction(variables):
	global iter_count
	input_vars=variables

	#calculate theoretical heat effect based on K and dH
	vars_with_exp=np.append(input_vars,np.asarray(exp_parameters))
	Qlist_ucal=calc_heats(*vars_with_exp)
	print(Qlist_ucal)
	#remove n_remove first peaks from fitting (not included in SS calculation)
	z=0
	while z<n_remove:
		Qlist_ucal.pop(0)
		z+=1
```

```
        #calculate sum of squares (SS)
        SS=calc_SS_stddev_weighted(expQ_pruned_list,Qlist_ucal)

        #return calculated SS
        print("iteration no: ",iter_count,"multi spectra SS ",SS)
        iter_count+=1
        return SS

#function for calculating partial derivatives of residuals in respect to K and dH
def calcDerivs(modelParameters,diff_K_delta,diff_dH_delta):
        #using 2 point derivation (K+d-(K-d))/2d
        print("calculating numerical derivative of residuals with respect to K and dH with delta values of
+-{} and +- {}".format(diff_K_delta,diff_dH_delta))
        #calculate derivatives in respect to K
        KderivUpperVarlist=np.copy(modelParameters)
        KderivUpperVarlist[0]+=diff_K_delta
        KderivLowerVarlist=np.copy(modelParameters)
        KderivLowerVarlist[0]-=diff_K_delta
        KderivUpperQlist=calc_heats(*KderivUpperVarlist)
        KderivLowerQlist=calc_heats(*KderivLowerVarlist)
        KderivList=[]
        for i in range(len(KderivUpperQlist)):
                KderivList.append((KderivLowerQlist[i]-KderivUpperQlist[i])/(2*diff_K_delta))
#theoretically should include expQ but these cancel out and also change sign so it is lower-upper instead
of upper-lower
        #calculate derivatives in respect to dH
        dHderivUpperVarlist=np.copy(modelParameters)
        dHderivUpperVarlist[1]+=diff_dH_delta
        dHderivLowerVarlist=np.copy(modelParameters)
        dHderivLowerVarlist[1]-=diff_dH_delta
        dHderivUpperQlist=calc_heats(*dHderivUpperVarlist)
        dHderivLowerQlist=calc_heats(*dHderivLowerVarlist)
        dHderivList=[]
        for i in range(len(dHderivUpperQlist)):
                dHderivList.append((dHderivLowerQlist[i]-dHderivUpperQlist[i])/(2*diff_dH_delta))

        return KderivList,dHderivList

#function for calculating statistics of fit
def calc_fitting_statistics(last_params):
        vars_with_exp=np.append(last_params,np.asarray(exp_parameters))
        print("calculating statistics of fit")

        #1 calculate jacobian matrix, its weighted form and transpose -> this is to find the standard error
of parameters K and dH

        #calculate theoretical curve using last parameters from fitting routine
```

```
        Qlist_ucal=calc_heats(*vars_with_exp)
        #calculate derivatives
        Kderivs,dHderivs=calcDerivs(vars_with_exp,diff_K_delta,diff_dH_delta)
        #i=0 --> K
        #i=1 --> dH
        Jacobian_T=np.array([Kderivs,dHderivs])
        Jacobian=np.copy(Jacobian_T)
        Jacobian=Jacobian.T
        #calculate Jacobian, weighted by stdev of each experimental data point
        for i in range(len(stdDev)):
                for j in range(len(Jacobian[i])):
                        Jacobian[i][j]=Jacobian[i][j]/stdDev[j]

        #calculate covariance matrix and its inverse
        Cov=np.linalg.inv(np.matmul(Jacobian_T,Jacobian))
        #calculate standard sum of squares
        SS=calc_SS(expQ_list,Qlist_ucal)
        #calculate weighted sum of squares
        SSw=calc_SS_stddev_weighted(expQ_list,Qlist_ucal)
        dof=len(expQ_list)-n_remove-2
        print("chi^2/DoF=", SS/dof)
        SE_K=math.sqrt(Cov[0][0])
        SE_dH=math.sqrt(Cov[1][1])
        print("Minimization results:\n")
        print("K= ",last_params[0], " +- ", SE_K*1.96, " (95%)")
        print("dH= ",last_params[1], " +- ", SE_dH*1.96, " (95%)")

#function for plotting results
def plot_graphs(last_params):
        #calculate theoretical curve using last parameters from fitting routine
        vars_with_exp=np.append(last_params,np.asarray(exp_parameters))
        Qlist_ucal=calc_heats(*vars_with_exp)
        inj=list(range(len(Qlist_ucal)))
        fig = plt.figure()
        ax = fig.add_subplot(111)
        ax.plot(inj,expQ_list,marker='s',label='exp Q')
        ax.plot(inj,Qlist_ucal,marker='s',label='simulated Q')
        plt.legend(loc='best')
        plt.show()

#minimize sum of squares and print statistics
iter_count=0
result=minimize(minFunction,initial_guess,method='Nelder-Mead',options={'maxiter':10000})
# minimization result
print(result)
#calculate statistics for K and dH values and some additional data
calc_fitting_statistics(result.x)
```

```
#plot graphs
plot_graphs(result.x)
```

9. References

[1] S.L.R. Ellison, A. Williams, and Eurachem Working Group on Uncertainty in Chemical Measurement (1995). Quantifying uncertainty in analytical measurement, Eurachem, London.

[2] (2014). MicrocalTM iTC200 System User Manual MAN0560,.