Physikalisch-chemisches Praktikum
ETH Zürich, HS23

Measuring Specific Heat Capacity of Ethanol Mixtures, Solution Enthalpy of Ammonium Nitrate, Calorimetric Titration of Vinegar and Melting Enthalpy of Ice

Janosch Jörg
DCHAB
jjoerg@ethz.ch

Maria Zimmermann
DCHAB
mazimmerm@ethz.ch

Assistant:
Dhiman Ghosh
dhiman.ghosh@phys.chem.ethz.ch

Abstract

The specific heat capacity of ethanol solutions at different concentrations was examined, yielding $2920 \pm 100\,\mathrm{J/(K\,kg)}$ for 80.09 %, $2590 \pm 100\,\mathrm{J/(K\,kg)}$ for 88.86 % and $2430 \pm 100\,\mathrm{J/(K\,kg)}$ for 93.73 %. Combining these values with results from other teams, the entire dilution range of ethanol was surveyed. With the same apparatus, the enthalpy of solution of ammonium nitrate was determined at $29.0 \pm 1.6\,\mathrm{kJ/mol}$. The acetate concentration of a vinegar sample was determined by calorimetric titration, i. e. measuring heat produced by neutralisation of acetic acid. The continuous addition of sodium hydroxide yielded $49.74 \pm 0.11\,\mathrm{g/L}$, while adding fixed doses of NaOH indicated a concentration of $50.6 \pm 1.2\,\mathrm{g/L}$ of acetic acid. Finally, the melting enthalpy of ice was examined, returning a value of $330 \pm 50\,\mathrm{kJ/kg}$.

Zurich, February 4, 2024

Janosch Jörg Maria Zimmermann

1 Introduction

Certain heat effects can be observed during chemical transformations, phase transitions, dissolution or mixing of substances. The general rule is that in a closed system such as the calorimeter any change in the internal system energy U only occurs through the exchange of heat δQ or work δW (here: work necessary to change the volume of the system against external pressure).

If the volume remains constant, this results in the relationship

$$dU = \delta Q \tag{1}$$

At constant pressure the enthalpy change dH is also equal to the amount of heat exchanged:

$$dH = \delta Q \tag{2}$$

If adiabatic conditions apply as well, the system's enthalpy remains the same and $dH = 0$ because no heat is exchanged. Within the system, thermal and chemical processes contribute to enthalpy changes:

$$dH = dH_{\text{thermal}} + dH_{\text{chemical}} \tag{3}$$

If a change in temperature dT occurs within a system at constant pressure where no alteration to the substances' chemical compositions can be observed, the following relation can be established:

$$dH = dH_{\text{thermal}} = C_p dT \tag{4}$$

where the proportionality constant C_p is called heat capacity of the system. If the materials' chemical compositions change at constant temperature and constant pressure, the following applies:

$$dH = dH_{\text{chemical}} = \Delta_r H dn \tag{5}$$

where $\Delta_r H$ is called enthalpy of reaction and dn refers to the amount n of chemically converted substance.

Overall, under isobaric conditions the following applies:

$$Q = C_p \Delta T + \Delta_r H \Delta n \tag{6}$$

assuming the change in temperature and chemical composition are fairly minor and therefore considering C_p and $\Delta_r H$ to be constant.

C_p of a given system is the sum of all its components heat capacities and can be determined via calibration by supplying a known amount of heat $Q = UIt$, where U is the operating voltage and I the operating current of the heating system used and t is the duration of the heatflow to the system.

$\Delta_r H$ can be found by thermically isolating the system and measuring ΔT during the respective reaction or process.

2 Experimental

2.1 Chemicals

The following experiments were conducted using ethanol (absolute $\geq$ 99.8 %) and ammonium nitrate (80.04 g/mol) ($\geq$ 99 %) from `Sigma Aldrich` as well as commercially available apple cider vinegar purchased form the `Coop` supermarket chain.

Additionally, a preprepared 1 M sodium hydroxide solution was used. Its origin and purity are unknown to the authors.

2.2 Procedure

A total of 5 experiments were prepared and carried out using the same setup.

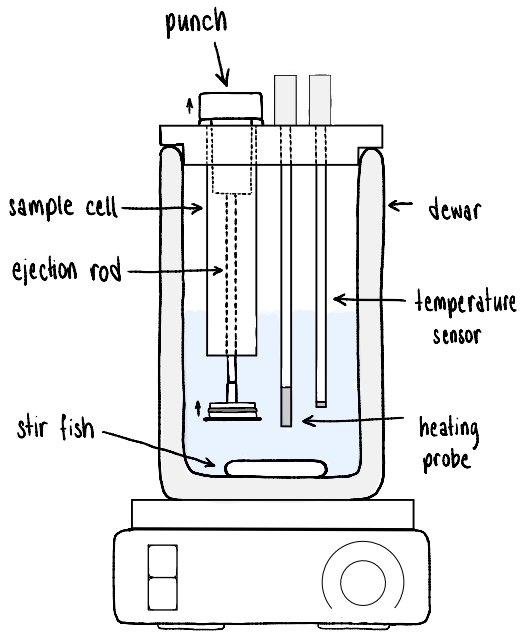


Figure 1: The entire setup for calorimetric measurements, including the `LAUDA MT` circulation thermostat, the `KNICK SE 204` measuring cell and the `KNICK Conductometer 703`.

2.2.1 Calibration of the calorimeter

100 mL of deionised water were measured in a volumetric flask and transferred into a `KGW ISOTHERM FB3 450mL` dewar vessel. (Fig. 1) The temperature of the sample was measured under constant light stirring, using an insertable `NATIONAL LM35` temperature sensor with a resolution better than 0.001 K. After waiting for the temperature to even out and recording T_{initial}, a constant heat supply to the system was induced by activating the heating element (`ELEKTRO-AUTOMATIK EA-3002L` for exactly 120 s. The power delivered was constant at $10.1\,\text{V} * 1.61\,\text{A} = 16.26\,\text{W}$ for this and all following measurements. The resulting temperature was noted down and the respective difference in temperature calculated.

2.2.2 Specific heat capacity of ethanol/water mixtures

The same procedure was repeated for various ethanol/water solutions, whereby each team was assigned three samples of different concentration. For the 80 % ethanol solution 72.5 g of ethanol were weighed in a previously tared beaker. 17.98 g of deionised water were added and 100 mL of the resulting solution were measured in a volumetric flask and transferred to the dewar to be heated. Similarly, 90 g of ethanol and 11.28 g of deionised water were used to create the 90 % solution and 98.72 g of ethanol and 6.6 g of deionised water were used to create the 95 % solution. The dewar was rinsed thoroughly with deionised water before each new measurement.

2.2.3 Solution enthalpy of ammonium nitrate

For the next experiment 100 mL of deionised water were measured in a volumetric flask and transferred to the dewar. 600 mg of NH_4NO_3, crushed beforehand using a mortar, were weighed on a `METTLER TOLEDO AB 204` analytical balance and transferred to the bottom plate of the sample cell. The cell was

closed and inserted into the lid of the dewar. The temperature recording was started and once the temperature had sufficiently equilibrated, the ammonium nitrate was released into the dewar by activating the ejection rod via the punch. (Fig. 1) . Once the temperature had equilibrated once more, the system was calibrated similarly to the previous experiments by supplying heat via heating element and noting down the resulting temperature change. The experiment was repeated twice more, with 595 mg and 601 mg of ammonium nitrate respectively.

2.2.4 Calorimetric titration of vinegar

Measurements in this experiment were conducted with a slightly modified calorimeter setup forgoing the probe vessel for the teflon hose of a `KNF SIMDOS 10` membrane dosing pump. 25 mL of vinegar were measured via volumetric pipette and transferred to a volumetric flask to be filled up to 100 mL and transferred to the dewar. The dosing pump containing a 1 M NaOH solution was set to continuous flow of 40 mL within 3 min and the temperature was recorded during this time. The experiment was repeated with a non-continuous-flow pump setting, instead opting to dose out 2 mL within 5 s every 20 s.

2.2.5 Melting enthalpy of ice and ice water

The calorimeter setup in this last experiment consisted only of the dewar, its lid and the temperature sensor. 100 mL of deionised water were measured in a volumetric flask and transferred to the dewar and T_{initial} measured. A beaker with ice water was prepared and its weight together with that of an empty 20 mL volumetric pipette measured via analytical balance. The pipette was then used to rapidly transfer 20 mL of ice water to the dewar. The weight of the same beaker and pipette were measured again and the difference to its initial weight, equivalent to the mass of the ice water added, recorded at 18.4 g. T_{final} was measured once the temperature had dropped significantly. The experiment was repeated twice more, using 17.5 g and 18.1 g of water respectively.

Following the same procedure, but with ice cubes instead of ice water, the measurement was repeated three times. A petri dish was lined with paper towels, placed on an analytical balance, several ice cubes transferred to it and the weight recorded. Using plastic forceps dry ice cubes were transferred to the dewar, until the resulting weight difference was approximately equal to the mass of the ice water added previously. By this method, the mass of the added ice cubes was measured at 14.34 g, 17.71 g and 17.55 g respectively.

3 Results and Discussion

The analysis and calculations were conducted using the R programming language [4] and Python Jupyter Notebooks [3]. The scripts are included in the appendix of this report. A.3 All uncertainties stated are provided for a 95 % confidence interval. The uncertainties were modelled with the aid of `METAS UncLib`, a Python uncertainty modelling library by the Swiss institute of metrology METAS.[5]

3.1 Calibration

Measuring the temperature response from the system while heating well-defined 100 mL of water and using (4) yields a net heat capacity of the empty dewar of 163 ± 8 J/K.

3.2 Specific heat capacity of ethanol/water mixture

Running the same procedure for a liquid with unknown heat capacity allows deriving the heat capacity of the solution itself by deducting the dewars heat capacity from the heat capacity of the entire system. The latter was determined thorugh calibration.The entire laboratory collaborated to measure the specific heat capacity of

ethanol water mixtures. The authors' mixtures contained 80.09 %, 88.86 % and 93.73 % of ethanol respectively (mass percent). Specific heat capacities of $2920 \pm 100\,\mathrm{J/(K\,kg)}$, $2590 \pm 100\,\mathrm{J/(K\,kg)}$ and $2430 \pm 100\,\mathrm{J/(K\,kg)}$ were found. This corresponds well with literature values of $2900\,\mathrm{J/(K\,kg)}$, $2700\,\mathrm{J/(K\,kg)}$ and $2560\,\mathrm{J/(K\,kg)}$.

The uncertainties for these measurements were derived from the standard deviation of the system calibration, since the scattering in the raw data can be assumed the same for ethanol and for water.

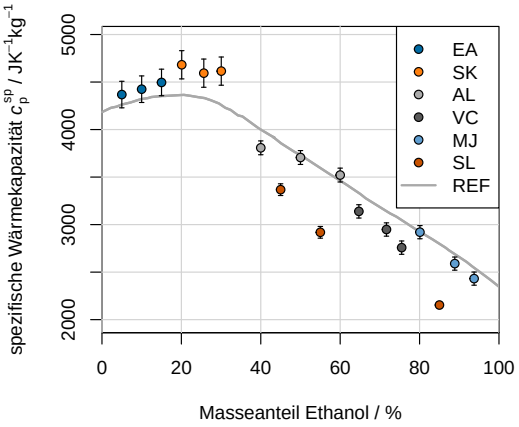


Figure 2: The specific heat capacity c_p^{sp} as a function of the ethanol share in water. The grey line represents literature values[1], the coloured dots show the values obtained by the teams, indicated by their initials. Our values are the blue ones on the right-hand side labelled "MJ".

3.3 Solution enthalpy of ammonium nitrate

By comparing the temperature drop caused by dissolving 600 mg of NH_4NO_3 with the rise in temperature upon heating and using (6) as well as the knowledge of the systems heat capacity garnered through calibration, the solution enthalpy of ammonium nitrate was calculated at $\Delta_{sol}H_{\mathrm{NH_4NO_3}} = 29.0 \pm 1.6\,\mathrm{kJ/mol}$.

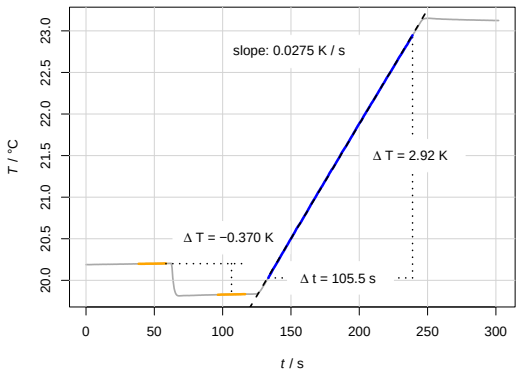


Figure 3: An example of the data recorded for the determination of the solution enthalpy of ammonium nitrate. The orange lines were read out for the temperature drop of dissolution, the blue slope was used as calibration reference. This procedure was identically repeated three times.

3.4 Calorimetric titration of vinegar

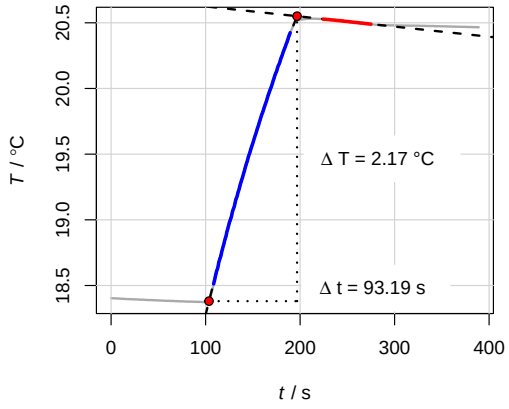


Figure 4: Continuous titration of vinegar with sodium hydroxide.

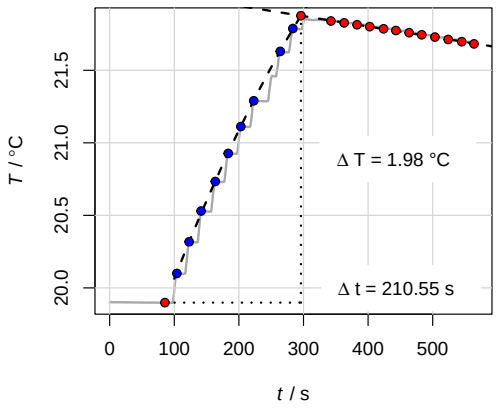


Figure 5: Discrete titration of vinegar with sodium hydroxide. A high resolution version of this plot can be found in the appendix.6

By calorimetric titration with NaOH, the acetate concentration of a vinegar sample was found to be $0.8283 \pm 0.0018\,\mathrm{M}$ with continuous titration and $0.842 \pm 0.018\,\mathrm{M}$ with stepwise titration. This converts to $49.74 \pm 0.11\,\mathrm{g/L}$ and $50.6 \pm 1.2\,\mathrm{g/L}$ of acetic acid respectively.

3.5 Melting enthalpy of ice

From measuring the temperature drop of well-defined amounts of water and ice three times each, a melting enthalpy of $330 \pm 50\,\mathrm{kJ/kg}$ was determined. This corresponds well with the literature value of $333.55\,\mathrm{kJ/kg}$.[2]

References

[1] G.C. BensonP.J. D'Arcy. "Spezifische Wärmekapazität von Wasser/Ethanol-Mischungen bei 15 °C". In: *J. Chem. Eng. Data* 27 (1982).

[2] W.M. Haynes. *CRC Handbook of Chemistry and Physics*. CRC Press, 1981.

[3] Fernando Pérez and Brian E. Granger. "IPython: a System for Interactive Scientific Computing". In: *Computing in Science and Engineering* 9.3 (May 2007), pp. 21–29. ISSN: 1521-9615. DOI: 10.1109/MCSE.2007.53. URL: https://ipython.org.

[4] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2021. URL: https://www.R-project.org/.

[5] Michael Wollensack. *METAS UncLib*. URL: https://www.metas.ch/metas/en/home/fabe/hochfrequenz/unclib.html. (accessed: 12.01.2024).

A Appendix

A.1 Course book exercises

exercises :

① H_2O : $c_p^{sp} = 4182\ JK^{-1}kg^{-1}$ $|:1000 = 4{,}182\ JK^{-1}g^{-1}$

$\rightarrow c_p = M\, c_p^{sp} \approx 75{,}36\ JK^{-1}mol^{-1}$

② Ice : $\Delta_m H^{sp} = 3{,}338 \cdot 10^5\ Jkg^{-1}$ $|:1000 = 3{,}338 \cdot 10^2\ Jg^{-1}$

$\rightarrow \Delta_m H = M\, \Delta_m H^{sp} = 3{,}338 \cdot 10^2 \cdot 18{,}02 \approx 6{,}015 \cdot 10^3\ Jmol^{-1}$

③ $Q = C_p \cdot \Delta T = \left(C_{p,w} + C_{p,K}\right)\Delta T = \left(c_p^{spec}{}_w \cdot m_w + C_{pK}\right)\Delta T =$

$= \left(4182\ JK^{-1}kg^{-1} \cdot 0{,}1\,kg + 120\ JK^{-1}\right)\Delta T = 538{,}2\ JK^{-1}\Delta T = 500\ J \rightarrow \Delta T = 0{,}929\,K$

if assumption: $C_{p,K} \approx 0 \rightarrow Q = C_{pw} \cdot \Delta T = c_p^{spec}{}_w \cdot m_w \cdot \Delta T = 418{,}2\ JK^{-1} \cdot \Delta T = 500\ J \rightarrow \Delta T = 1{,}196\,K$

④ $Q = \underbrace{m_{ice}\,\Delta_m H}_{\text{diff amount for melting of 10g ice at } T_E} + \underbrace{m_{ice}\, c_p^{spec}{}_w\,(T_f - T_E)}_{\text{diff amount for heating of ice water}} + \underbrace{m_w\, c_p^{spec}{}_w\,(T_f - T_i)}_{\text{diff amount for cooling of (test) water}} + \underbrace{C_{pK}\,(T_f - T_i)}_{\text{diff amount for cooling of calorimeter}} = 0$

$\rightarrow T_f = \dfrac{(m_w c_p^{spec}{}_w + C_{pK})T_i + m_E c_p^{spec}{}_w T_E - m_E \Delta_m H}{(m_w + m_E)\, c_p^{spec}{}_w + C_{pK}}$

for $m_w = 100 = 0{,}1\ kg$, $C_{pK} = 120\ JK^{-1}$, $T_i = 293{,}15\,K\ (20°C)$, $T_E = 273{,}15\,K\ (0°C)$ & $m_{ice} = 10\,g = 0{,}01\ kg$ $\left(c_p^{spec}{}_w = 2100\ JK^{-1}kg^{-1}\right)$

$\Rightarrow T_f = 285{,}95\,K\ (12{,}8°C)$

if ice water (0°C) is add instead of ice :

$Q = m_{icewater}\, c_p^{spec}{}_w\,(T_f - T_E) + m_w\, c_p^{spec}{}_w\,(T_f - T_i) + C_{pK}\,(T_f - T_i) = 0$

$\rightarrow T_f \cdot \left(c_p^{spec}{}_w \cdot m_{icew.} + c_p^{spec}{}_w \cdot m_w + C_{pK}\right) = m_{icew.} \cdot T_E \cdot c_p^{spec}{}_w + T_i \cdot \left(m_w \cdot c_p^{spec}{}_w + C_{pK}\right)$

$\rightarrow T_f = \dfrac{T_E \cdot m_{icew.}\, c_p^{spec}{}_w + T_i\left(m_w\, c_p^{spec}{}_w + C_{pK}\right)}{c_p^{spec}{}_w \cdot (m_{icew.} + m_w) + C_{pK}}$

for $m_w = 0{,}1\,kg$, $C_{pK} = 120\ JK^{-1}$, $T_i = 293{,}15\,K\ (20°C)$, $T_E = 273{,}15K\ (0°C)$ & $m_{icew.} = 0{,}01kg$ $\left(c_p^{spec}{}_w = 2100\ JK^{-1}kg^{-1}\right)$

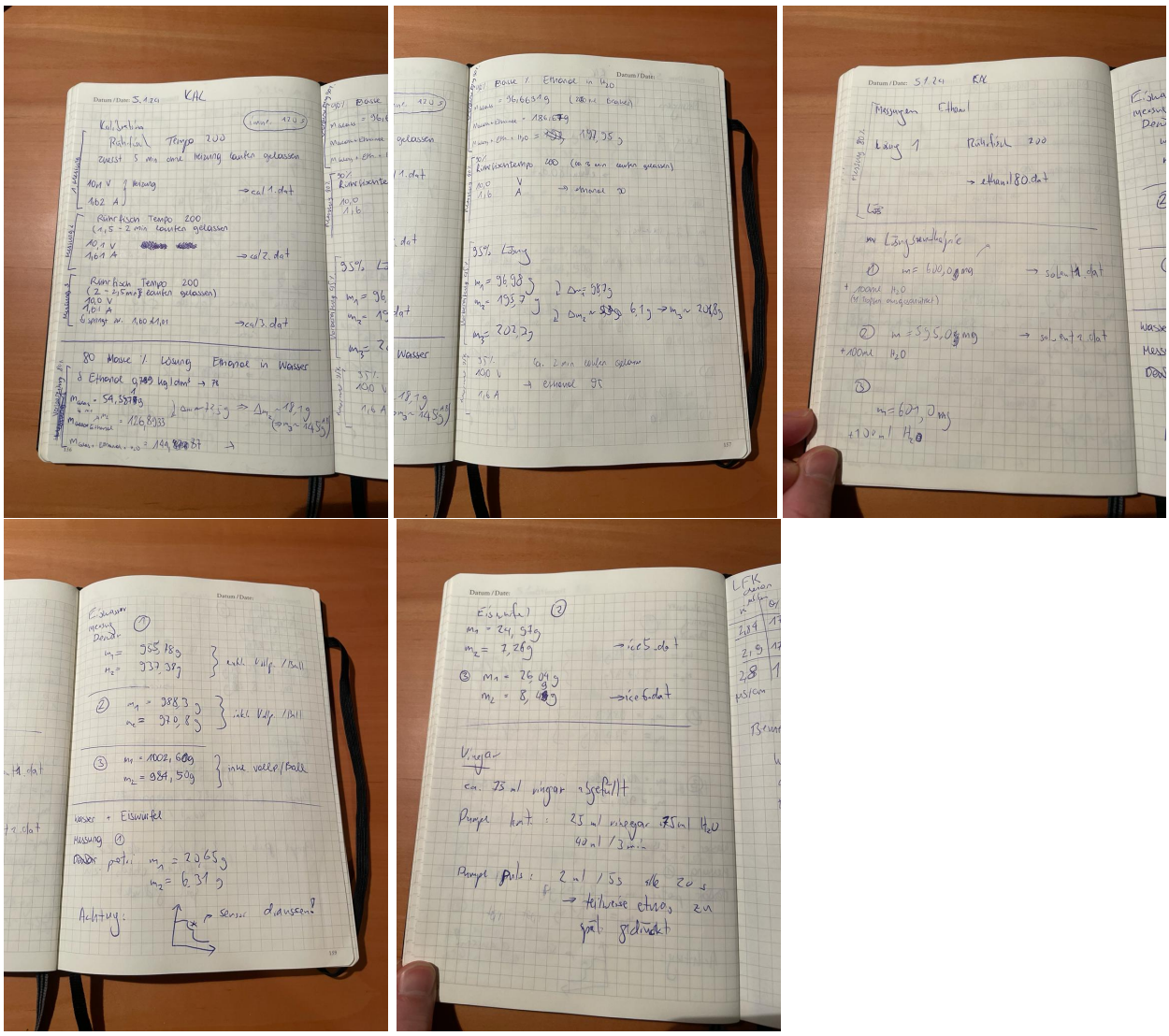
$\Rightarrow T_f = 291{,}95\,K\ (18{,}8°C)$

⑤ $0 = c_p^{spec}{}_{\substack{liq \\ sol.}} \cdot m_{\substack{liq \\ sol}} \cdot \Delta T + n_{liq} \cdot \Delta_{sol} H_{liq}$

$c_p^{spec}{}_{\substack{liq \\ sol.}} = 4{,}053\ JK^{-1}g^{-1}$ $\quad m_{\substack{liq \\ sol}} = 1\,mol \cdot 42{,}9\,\frac{g}{mol} + 100\,mol \cdot 18{,}02\,\frac{g}{mol} = 222{,}6\,g$ $\quad \Delta_{sol} H_{\substack{liq \\ sol}} = -35{,}941$

$\Delta T = -\dfrac{n_{liq} \cdot \Delta_{sol} H_{liq}}{c_p^{spec}{}_{\substack{liq \\ sol}} \cdot m_{\substack{liq \\ sol}}} = \dfrac{1 \cdot 35{,}941}{4{,}053 \cdot 222{,}6} \approx 0{,}04\,K = 0{,}04°C$

A.2 Labjournal



A.3 R and Python scripts

A.3.1 Calibration and specific heat capacity of ethanol/water mixture

```
rm(list=ls())

# CONSTANTS
# switch to False to reduce processing time :)
DRAW.PLOTS = TRUE

THRES <<- 0.2

HEIGHT <<- 4
WIDTH <<- 5
```

```
IMPORT_PATH <<- "raw_data/"
EXPORT_PATH_CAL <<- "exports/calib/"
EXPORT_PATH_ETH <<- "exports/ethanol/"

ref.V <<- 0.1  # L

water.c.sp <<- 4182  # J / K / kg
water.rho <<- 0.9982  # kg / L
water.m <<- water.rho * ref.V  # kg

eth.rho <<- 0.7893  # kg / L

heater.U <<- 10.0  # V
heater.I <<- 1.61  # A
heater.P <<- heater.U * heater.I  # W

# for clean display:
# quartz(height=HEIGHT, width=WIDTH)


# HELPER FUNCTIONS
source("helpers.R")
source("kal_routines.R")


# LINEAR REGRESSION MAIN FUNCTION
calibration.linreg <- function(
  import.datfile,
  export.path,
  export.plot,
  export.stats,
  plot.init=TRUE
) {

  calib = read.table(join(IMPORT_PATH, import.datfile))
  time = calib$V1
  temp = calib$V2

  statistics = process.slope(
    time, temp,
    export.path=export.path,
    export.stats=export.stats,
    plot.init=plot.init
  )

  if(DRAW.PLOTS) {
    plot.save(export.path, export.plot)
  }
```

```
  statistics
}



# PROCESS ETHANOL MIXTURE DATA
ethanol_mixture <- function(import.csv) {
  mix = read.csv(join(IMPORT_PATH, import.csv), dec=".")

  mix$eth.m = mix$m2 - mix$m1
  mix$water.m = mix$m3 - mix$m2
  mix$total.m = mix$eth.m + mix$water.m
  mix$share.m = mix$eth.m / mix$total.m
  mix$eth.V = mix$eth.m / eth.rho
  mix$water.V = mix$water.m / water.rho
  mix$total.V = mix$eth.V + mix$water.V
  mix$share.V = mix$eth.V / mix$total.V
  mix$eth.m.perV = mix$eth.m * (ref.V / mix$total.V)
  mix$water.m.perV = mix$water.m * (ref.V / mix$total.V)
  mix$total.m.perV = mix$eth.m.perV + mix$water.m.perV

  mix
}



slopes.water = c()

for (i in 1:3) {
  st = calibration.linreg(join("cal",i,".dat"), EXPORT_PATH_CAL,
  ↪  join("cal",i,".pdf"), join("cal",i,"_stats.csv"))
  slopes.water = cbind(slopes.water, st$reg.b)
}

# calculate heat capacity of setup
slope.water = mean(slopes.water)
slope.water.se = sd(slopes.water) / sqrt(2)

system.C = heater.P / slope.water
water.C = water.m * water.c.sp

dewar.C = system.C - water.C

mix = ethanol_mixture("ethanol_composition.csv")
mix$C = c(0,0,0)
mix$c.sp = c(0,0,0)
```

```
for (i in 1:3) {
  target = mix$mass_percent_aim[i]
  st = calibration.linreg(join("ethanol",target,".dat"), EXPORT_PATH_ETH,
  ↪  join("ethanol",target,".pdf"), join("ethanol",target,"_stats.csv"))
  mix$C[i] = heater.P / st$reg.b - dewar.C  # J
  mix$c.sp[i] = mix$C[i]/ mix$total.m.perV[i]
}

write.csv(mix, join(EXPORT_PATH_ETH, "ethanol_results.csv"))
```

A.3.2 Uncertainty estimate of calibration and specific heat capacity of ethanol/water mixture

In [1]:
```
import math

from metas_unclib import *

from Metas.UncLib.LinProp import UncBudget

import pandas as pd

import itables

itables.init_notebook_mode(all_interactive=True)
```

In [2]:
```
import metas_unclib as mu
```

In [3]:
```
def unc_budget(unc_item, show_table=True):
    tree = UncBudget.ComputeTreeUncBudget(unc_item.net_object)

    table = pd.DataFrame(
        columns=("description", "uncertainty component", "uncertain
        index=range(len(tree)+1)
    )

    for i, elem in enumerate(tree):
        table.loc[i] = (
            elem.get_Description(),
            elem.get_UncComponent(),
            elem.get_UncPercentage(),
        )
    table.loc[len(tree)] = (
        "SUMMARY",
        unc_item.stdunc,
        100.,
    )

    return table.sort_values("uncertainty percentage", ascending=Fa
```

In [4]:
```
def tolerance(value, a):
    """
    producer tolerance of value +/- a

    returns UniformDistribution(value - a, value + a)
    """
    return UniformDistribution(value - a, value + a)
```

C_{Dewar}

$$C_{Dewar} = C_{Sys,kal} - C_W = \frac{UI}{b} - m_W c_W^{sp}$$

c_L^{sp}: spezifische Wärmekapaziät

C: Wärmekapazität Wasser

m: Masse Lösungsmittel

U, I: Spannung, Strom Heizung

b: Steigung aus T-t-Diagramm

In [5]:
```
ref_V = ufloatfromdistribution(
  tolerance(0.1, 0.0001),  # L
  desc='reference volume / L'
)
ref_V
```

Out[5]: 0.1 ± 5.773502691896423e-05

In [6]:
```
# rho water
water_rho = ufloat(0.9982, desc="water rho / kg/L")
water_m = water_rho * ref_V
water_m
```

Out[6]: 0.09982 ± 5.76311038705101e-05

In [7]:
```
water_c_sp = ufloat(4182, desc="water specific heat capacity / J/K/
```

In [8]:
```
U = ufloatfromdistribution(
  tolerance(10.0, 0.05),  # V
  desc='voltage / V'
)
I = ufloatfromdistribution(
  tolerance(1.61, 0.005),  # V
  desc='current / A'
)
P = U * I
P
```

Out[8]: 16.1 ± 0.0547121254080546

In [72]:
```
# Janosch

water_slope_values = np.array((0.027875, 0.02771894, 0.02757587))
water_slope = ufloatfromsamples(
    water_slope_values,
    desc='slope of water calibration / K/s'
)
water_slope
```

Out[72]: 0.027723269999999998 ± 0.0001896243080064033 4

```
# check for Samuel

water_slope_values = np.array((0.02343122, 0.02377467,
0.02358517))
water_slope = ufloatfromsamples(
    water_slope_values,
    desc='slope of water calibration / K/s'
)
water_slope
```

In [73]:
```
dewar_C_array = U.value * I.value / water_slope_values - water_m.va
dewar_C_array
```

Out[73]:
```
array([160.13123534, 163.38305149, 166.39653356])
```

In [78]:
```
dewar_C = U * I / water_slope - water_m * water_c_sp
#dewar_C.mean()
dewar_C
```

Out[78]:
```
163.2923336505831 ± 4.441980282611897
```

In [77]:
```
ufloatfromsamples(dewar_C_array)
```

Out[77]:
```
163.30360679551532 ± 3.971403791973842
```

c_L^{sp}

$$c_L^{sp} = \frac{C_L}{m} = \frac{C_{Sys,L} - C_{Dewar}}{m} = \frac{UI}{bm} - \frac{C_{Dewar}}{m}$$

```
# even more brute force for other estimates
# dewar_C = ufloat(dewar_C_array.mean(), 1.4, desc="dewar_C")
```

In [68]:
```
# Janosch
sol_slope = ufloat(
    0.0398696981455289,
    2.96817283935158E-05,
    desc="slope of solution / K/s"
)
sol_slope
```

Out[68]:
```
0.0398696981455289 ± 2.96817283935158e-05
```

```
# check for Samuel
sol_slope = ufloat(
    0.025,  # estimated (fitted to reach target result)
    2.96817283935158E-05,
    desc="slope of solution / K/s"
)
sol_slope
```

In [79]:
```
# since the scale is very accurate in comparison to the other uncer
sol_m_perV = ufloat(
  0.08236242,  # kg
  desc='solution mass / kg'
)
```

In [80]:
```
# absolute heat capacity of ethanol
U * I / sol_slope - dewar_C
```

Out[80]:
```
240.5231138913868 ± 4.035878115463858
```

In [81]:
```
# specific heat capacity
sol_c_sp = (U * I / sol_slope - dewar_C) / sol_m_perV
sol_c_sp
```

Out[81]:
```
2920.301684814346 ± 49.00145133501247
```

In [66]:
```
unc_budget(sol_c_sp)
```

Out[66]:

	description	uncertainty component	uncertainty percentage
5	SUMMARY	77.177717	100.0
3	slope of water calibration / K/s	76.545151	98.367472
2	slope of solution / K/s	9.283376	1.446865
1	reference volume / L	2.926253	0.143761
4	voltage / V	1.342025	0.030237
0	current / A	0.833556	0.011665

In [23]:
```
print(unc_budget(sol_c_sp).to_latex())
```

```
\begin{tabular}{llll}
\toprule
{} &                        description & uncertainty component & u
ncertainty percentage \\
\midrule
5 &                            SUMMARY &             49.001451 &
100.0 \\
3 &  slope of water calibration / K/s &             48.228291 &
96.869234 \\
4 &                        voltage / V &              6.20108 &
1.601463 \\
0 &                        current / A &             3.851603 &
0.617824 \\
2 &           slope of solution / K/s &             3.650061 &
0.554859 \\
1 &              reference volume / L &             2.926253 &
0.35662 \\
\bottomrule
\end{tabular}
```

A.3.3 Specific heat capacity of ethanol - overview plot

```
# author: github.com/janjoch, 2024

rm(list = ls()) # tabula rasa

library(readxl)

source("helpers.R")

WIDTH <<- 5
HEIGHT <<- 4

start.quartz <- function(height=HEIGHT, width=WIDTH) {
  quartz(
    height=height,
    width=width
  )
}

# par(
#   mfrow=c(1,2),
#   mar=c(0.7, 5,1,0.7)
# )


# import data
data.import = read.csv("raw_data/waermekapazitaet_ethanol.csv")
data.lit = read.csv("raw_data/Literatur_Ethanol.csv", sep=";")

mass = (data.import$mass_percentage_measured * 100)
heat = data.import$specific_heat_capacity_mean
heat.ci = data.import$standard_error_result_95
group <- data.import$group

mass.lit = data.lit$mass_percentage
heat.lit = data.lit$specific_heat_capacity * 1000

legend.import = read.csv("raw_data/waermekapazitaet_ethanol_groups.csv")
legend.id = legend.import$id
legend.name = legend.import$name


min.id = which.min(heat)
max.id = which.max(heat)
max.val = heat[max.id] + heat.ci[max.id]
min.val = heat[min.id] - heat.ci[min.id]
span = max.val - min.val
```

```
margin = 0.05
max.val = max.val + margin * span
min.val = min.val - margin * span

plot.colorcycle = c('#006BA4', '#FF800E', '#ABABAB', '#595959', '#5F9ED1',
↪  '#C85200', '#898989', '#A2C8EC', '#FFBC79', '#CFCFCF')

par(mai = c(1,1.2,0.3,0.3))
plot.init.grey(
  mass.lit, heat.lit,
  xlab="Masseanteil Ethanol / %",

  ↪  ylab=expression("spezifische Wärmekapazität "*italic(c)[p]^sp*" / "*J*K^-┘
  ↪  1*kg^-1),
  #xaxt='n'
  xaxs="i",
  ylim=c(min.val, max.val),
  #ylim=c(acet.rho.data[1] - 1.2 * acet.rho.se[1], acet.rho.data[1] + 1.2
  ↪  * acet.rho.se[1]),
)
lines(
  mass.lit, heat.lit,
  col="darkgrey",
  lwd=2
)
# plot.grid(nx=NA)

FBy(
  mass,
  heat,
  heat.ci,
  bg=plot.colorcycle[group]
)

# break

legend(
  "topright",
  legend=append(legend.name, "REF"),
  pt.bg=plot.colorcycle[legend.id],
  pch=append(
    rep(21, length(legend.name)),
    -1
  ),
  lty=append(
      rep(0, length(legend.name)),
      1
  ),
```

```
  col=append(
      rep("black", length(legend.name)),
      "darkgrey"
  ),
  lwd=append(
      rep(1, length(legend.name)),
      2
  ),
  bg="white"
)

# text(35,3000, "PRELIMINARY!", cex=2)



plot.save("exports/", "ethanol_alldata_5_4_in.pdf")
```

A.3.4 Solution enthalpy of ammonium nitrate

```
rm(list=ls())

# raw = read.table(join(IMPORT_PATH, "sol_ent1.dat"))
# > time = raw$V1
# > temp = raw$V2


# CONSTANTS
DRAW.PLOTS = TRUE

THRES <<- 0.2
THRES.DROP <<- 0.1
THRES.DIFF <<- 0.007

EQUIB.WIDTH = 40

HEIGHT <<- 7
WIDTH <<- 10
HEIGHT <<- 5
WIDTH <<- 7

IMPORT_PATH <<- "raw_data/"
EXPORT_PATH_SOL <<- "exports/solution_enthalpy/"


# HELPER FUNCTIONS
source("helpers.R")
source("kal_routines.R")


analyse.solv.enth <- function(
  import.datfile,
  export.path,
  export.plot,
  export.stats.drop,
  export.stats.slope
) {

  sol = read.table(import.datfile)
  time = sol$V1
  temp = sol$V2

  # init plot
  plot.init.grey(time, temp)

  stats.drop = process.drop(
```

```
    temp,
    time,
    export.path=export.path,
    export.stats=export.stats.drop,
    detect.mode = "slope.next"
  )
  print(stats.drop)
  index.last = length(time)
  time.forslope = time[(stats.drop$index.detected):index.last]
  temp.forslope = temp[(stats.drop$index.detected):index.last]

  stats.slope = process.slope(
    time.forslope,
    temp.forslope,
    export.path=export.path,
    export.stats=export.stats.slope,
    plot.init=FALSE
  )

  plot.save(export.path, export.plot=export.plot)

  return(list(stats.drop, stats.slope))
}


# Massen eingewogen
masses = c(0.600, 0.595, 0.601)  # g

st.d.a = c() # stats.drop.all
st.s.a = c() # stats.slope.all

for(i in 1:3) {
  c(st.d, st.s) := analyse.solv.enth(
    join("raw_data/sol_ent", i, ".dat"),
    EXPORT_PATH_SOL,
    export.plot=join("sol", i, ".pdf"),
    export.stats.drop=join("sol", i, "_drop.csv"),
    export.stats.slope=join("sol", i, "_slope.csv")
  )
  st.d.a = rbind(st.d.a, st.d)
  st.s.a = rbind(st.s.a, st.s)
}


write.csv(st.d.a, join(EXPORT_PATH_SOL, "stats_drop.csv"))
write.csv(st.s.a, join(EXPORT_PATH_SOL, "stats_slope.csv"))
```

A.3.5 Uncertainty estimate of solution enthalpy of ammonium nitrate

In [40]:
```
import math
```

In [41]:
```
import numpy as np
```

In [42]:
```
from metas_unclib import *
```

In [44]:
```
from Metas.UncLib.LinProp import UncBudget
```

In [ ]:
```
import pandas as pd

import itables

itables.init_notebook_mode(all_interactive=True)
```

In [47]:
```
def unc_budget(unc_item, show_table=True):
    tree = UncBudget.ComputeTreeUncBudget(unc_item.net_object)

    table = pd.DataFrame(
        columns=("description", "uncertainty component", "uncertain
        index=range(len(tree)+1)
    )

    for i, elem in enumerate(tree):
        #print(element)
        table.loc[i] = (
            elem.get_Description(),
            elem.get_UncComponent(),
            elem.get_UncPercentage(),
        )
    table.loc[len(tree)] = (
        "SUMMARY",
        unc_item.stdunc,
        100.,
    )

    return table.sort_values("uncertainty percentage", ascending=Fa
```

In [4]:
```
def tolerance(value, a):
    """
    producer tolerance of value +/- a

    returns UniformDistribution(value - a, value + a)
    """
    return UniformDistribution(value - a, value + a)
```

$$\Delta_{sol} H_B = -UIt\frac{M_B}{m_B}\frac{\Delta T_{sol}}{\Delta T_C} = -UI\frac{M_B}{m_B}\frac{\Delta T_{sol}}{b}$$

B: Amonium Nitrate NH_4NO_3

In [5]:
```
U = ufloatfromdistribution(
  tolerance(10.0, 0.05),  # V
  desc='voltage / V'
)
I = ufloatfromdistribution(
  tolerance(1.61, 0.005),  # V
  desc='current / A'
)
P = U * I
P
```

Out[5]: 16.1 ± 0.0547121254080546

In [10]:
```
amm_M = ufloat(
    80.04,
    desc="molar mass ammonium nitrate NH4NO3 / g/mol",
)
```

In [20]:
```
amm_m = ufloatfromsamples(
    (0.600, 0.595, 0.601),
    desc="absolute mass of ammonium nitrate NH4NO3 / g",
)
amm_m
```

Out[20]: 0.5986666666666666 ± 0.0040742511466699575

In [32]:
```
amm_T_delta = ufloatfromsamples(
    (-0.3815051, -0.3662502, -0.3717871),
    desc="delta T of solution / K",
)
amm_T_delta
```

Out[32]: −0.37318080000000003 ± 0.009787627577074718

In [33]:
```
# Standardfehler der Regression wird ignoriert, da um über 1 Grösse
amm_b = ufloatfromsamples(
    (0.02780959, 0.02764169, 0.02755107),
    desc="heating rate / K/s",
)
amm_b
```

Out[33]: 0.02766745 ± 0.0001662514343030656 2

In [37]:
```
amm_H_sol = - U * I * amm_M / amm_m * amm_T_delta / amm_b  # J / mo
amm_H_sol / 1000  # kJ / mol
```

Out[37]: 29.03340925691424 ± 0.8118233607977087

In [48]:
```
unc_budget(amm_H_sol)
```

Out[48]:

	description	uncertainty component
5	SUMMARY	811.823361
2	delta T of solution / K	761.475931
0	absolute mass of ammonium nitrate NH4NO3 / g	197.588086
3	heating rate / K/s	174.459371
4	voltage / V	83.812233
1	current / A	52.057288

```
with old values: amm_T_delta = -0.3840371, -0.3723012, -0.3814346

amm_H_sol = 29506.18596234154 ± 671.6938464947295
```

In [ ]:
```
# reference value @25°C: 25.69 kJ/mol
# https://en.wikipedia.org/wiki/Enthalpy_change_of_solution#Depende

# Medvedev et al.
# @ 25°C
# https://pdf.sciencedirectassets.com/271405/1-s2.0-S0040603100X028
# 25.53 ± 0.24 J/mol (k=2)
```

A.3.6 Calorimetric titration of vinegar

```
rm(list=ls())

WIDTH <<- 8
HEIGHT <<- 6

if(TRUE) {
  WIDTH <<- 5
  HEIGHT <<- 4
}

# quartz(height=HEIGHT, width=WIDTH)
quartz.start <- function(height=HEIGHT, width=WIDTH) {
  quartz(
    height=height,
    width=width
  )
}

library(readxl)


IMPORT_PATH <<- "raw_data/"
EXPORT_PATH <<- "exports/vinegar/"


# HELPER FUNCTIONS
source("helpers.R")
#source("kal_routines.R")


data.table = read.table("raw_data/vinegar_cont.dat")
time = data.table$V1
temp = data.table$V2

i.start = which.min(temp) + 15
i.end = 380

j.start = 450
j.end = 551


jreg = lm(temp ~ time, subset=c(j.start:j.end))
#jreg.pred = predict(jreg)
#lines(jreg, col="black", lwd=2)
```

```
plot.init.grey(
  time,
  temp,
  xlab=expression(italic(t)*" / "*s),
  ylab=expression(italic(T)*" / "*degree*C)
  #xaxs="i",
  #yaxs="i"
)



ireg <- lm(temp ~  I(time) + I(time^2), subset=c(i.start:i.end))
print(summary(ireg))

# Berechnung der Modellfunktion inkl. Vertrauenskurven
x.c <- seq(100, 200, 1)
y.c <- predict(ireg, list(time=x.c)) # , interval="confidence")
matlines(x.c, y.c, lty=c(2,2,2), lwd=c(2,1,1), lw=1)


jreg = plot.regression(
  time,
  temp,
  subset=c(j.start:j.end),
  draw.annotation = FALSE,
)
jreg = lm(temp ~ time, subset=c(j.start:j.end))


# Berechnung der Fitkurven
time.space <- seq(from=90, to=220, by=0.1)
th1 <- predict(ireg, list(time=time.space))
th2 <- predict(jreg, list(time=time.space))


# Interpolation des Schnittpunkts (= Aequivalenzpunkt)
inter.time = approx(th1-th2, time.space, 0)$y
inter.temp = predict.linear(summary(jreg)$coef[1,1],
↪  summary(jreg)$coef[2,1], inter.time)

# find starting point of slope
base.temp = mean(temp[120:160])
base.time = approx(th1, time.space, base.temp)$y

time.rise = inter.time - base.time
temp.rise = inter.temp - base.temp
```

```
# uncertainty estimation
i.unc.pred = predict(ireg, list(time=inter.time), interval="confidence")
j.unc.pred = predict(jreg, list(time=inter.time), interval="confidence")

# assuming y = ax + b
i.a = temp.rise / time.rise
j.a = summary(jreg)$coef[2,1]

i.unc = i.unc.pred[3] - i.unc.pred[1]
j.unc = j.unc.pred[3] - j.unc.pred[1]

time.unc = (i.unc + j.unc) / (i.a - j.a)


print("Schnittpunkt:")
print(c(inter.time, inter.temp))
print("Zeitdauer:")
print(time.rise)
print("Unsicherheit:")
print(time.unc)


plot.line.highlight(time[i.start:i.end], temp[i.start:i.end])

plot.line.highlight(time[j.start:j.end], temp[j.start:j.end], col="red")

# delta t
plot.line.annot(
  c(base.time, inter.time),
  c(base.temp, base.temp)
)
plot.annot(
  mean(c(base.time, inter.time)) + 130, # + 8,
  base.temp + 0.1,
  TeX(paste(r"(\Delta t =)", sprintf("%0.2f", time.rise), "s"))
)

# delta T
plot.line.annot(
  c(inter.time, inter.time),
  c(base.temp, inter.temp)
)
plot.annot(
  inter.time + 90,
  mean(c(base.temp, inter.temp)),
  TeX(paste(r"(\Delta T =)", sprintf("%0.2f", inter.temp - base.temp),
  ↪  "°C"))
)
```

```
points(inter.time, inter.temp, pch=21, bg="red")
points(base.time, base.temp, pch=21, bg="red")

plot.save(EXPORT_PATH, "vinegar_cont.pdf")

if(FALSE) {
  plot((time[2:length(time)] + time[1:length(time) - 1]) / 2 ,
  ↪  temp[2:length(temp)] - temp[1:(length(temp)-1)], type="l",
  ↪  col="red")
}
```

```
rm(list=ls())

WIDTH <<- 12
HEIGHT <<- 8

if(TRUE) {
  WIDTH <<- 5
  HEIGHT <<- 4
}

# quartz(height=HEIGHT, width=WIDTH)
quartz.start <- function(height=HEIGHT, width=WIDTH) {
  quartz(
    height=height,
    width=width
  )
}

library(readxl)


IMPORT_PATH <<- "raw_data/"
EXPORT_PATH <<- "exports/vinegar/"


# HELPER FUNCTIONS
source("helpers.R")
#source("kal_routines.R")


data.table = read.table("raw_data/vinegar_noncont.dat")
time = data.table$V1
temp = data.table$V2
```

```
i.select = c(209, 247, 284, 328, 368, 407, 447, 529, 568)
j.select = c(686,  727,  767,  806,  849,  887,  928,  967, 1008, 1049,
↪  1091, 1129)


jreg = lm(temp ~ time, subset=j.select)
#jreg.pred = predict(jreg)
#lines(jreg, col="black", lwd=2)



plot.init.grey(
  time,
  temp,
  xlab=expression(italic(t)*" / "*s),
  ylab=expression(italic(T)*" / "*degree*C)
  #xaxs="i",
  #yaxs="i"
)



ireg <- lm(temp ~  I(time) + I(time^2), subset=i.select)
print(summary(ireg))

# Berechnung der Modellfunktion inkl. Vertrauenskurven
x.c <- seq(100, 32000, 1)
y.c <- predict(ireg, list(time=x.c)) # , interval="confidence")
matlines(x.c, y.c, lty=c(2,2,2), lwd=c(2,1,1), lw=1)


jreg = plot.regression(
  time,
  temp,
  subset=j.select,
  draw.annotation = FALSE,
)
jreg = lm(temp ~ time, subset=j.select)


# Berechnung der Fitkurven
time.space <- seq(from=70, to=320, by=0.1)
th1 <- predict(ireg, list(time=time.space))
th2 <- predict(jreg, list(time=time.space))


# Interpolation des Schnittpunkts (= Aequivalenzpunkt)
inter.time = approx(th1-th2, time.space, 0)$y
```

```
inter.temp = predict.linear(summary(jreg)$coef[1,1],
↪  summary(jreg)$coef[2,1], inter.time)

# find starting point of slope
base.temp = mean(temp[120:160])
base.time = approx(th1, time.space, base.temp)$y


time.rise = inter.time - base.time
temp.rise = inter.temp - base.temp

# uncertainty estimation
i.unc.pred = predict(ireg, list(time=inter.time), interval="confidence")
j.unc.pred = predict(jreg, list(time=inter.time), interval="confidence")

# assuming y = ax + b
i.a = temp.rise / time.rise
j.a = summary(jreg)$coef[2,1]

i.unc = i.unc.pred[3] - i.unc.pred[1]
j.unc = j.unc.pred[3] - j.unc.pred[1]

time.unc = (i.unc + j.unc) / (i.a - j.a)


print("Schnittpunkt:")
print(c(inter.time, inter.temp))
print("Zeitdauer:")
print(time.rise)
print("Unsicherheit:")
print(time.unc)

points(time[i.select], temp[i.select], pch=21, bg="blue")

points(time[j.select], temp[j.select], pch=21, bg="red")

# delta t
plot.line.annot(
  c(base.time, inter.time),
  c(base.temp, base.temp)
)
plot.annot(
  mean(c(base.time, inter.time)) + 260,
  base.temp + 0.1,
  TeX(paste(r"(\Delta t =)", sprintf("%0.2f", time.rise), "s"))
)

# delta T
```

```
plot.line.annot(
  c(inter.time, inter.time),
  c(base.temp, inter.temp)
)
plot.annot(
  inter.time + 150,
  mean(c(base.temp, inter.temp)),
  TeX(paste(r"(\Delta T =)", sprintf("%0.2f", inter.temp - base.temp),
  ↪  "°C"))
)

points(inter.time, inter.temp, pch=21, bg="red")
points(base.time, base.temp, pch=21, bg="red")

plot.save(EXPORT_PATH, "vinegar_uncont.pdf")

if(FALSE) {
  plot((time[2:length(time)] + time[1:length(time) - 1]) / 2 ,
  ↪  temp[2:length(temp)] - temp[1:(length(temp)-1)], type="l",
  ↪  col="red")
}
```

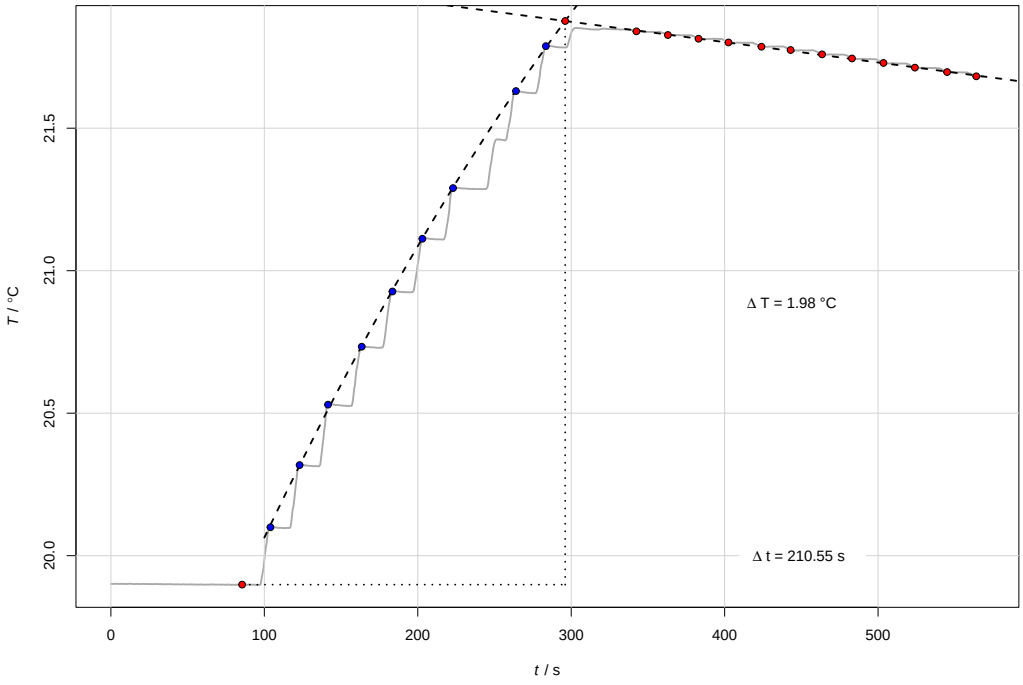


Figure 6: Discrete titration of vinegar with sodium hydroxide.

A.3.7 Melting enthalpy of ice

```
rm(list=ls())

# raw = read.table(join(IMPORT_PATH, "sol_ent1.dat"))
# > time = raw$V1
# > temp = raw$V2


# CONSTANTS
DRAW.PLOTS = TRUE

THRES <<- 0.2
THRES.DROP <<- 0.1
THRES.DIFF <<- 0.007

EQUIB.WIDTH = 40

HEIGHT <<- 7
WIDTH <<- 10

IMPORT_PATH <<- "raw_data/"
EXPORT_PATH_SOL <<- "exports/melting_enthalpy/"


# HELPER FUNCTIONS
source("helpers.R")
source("kal_routines.R")


analyse.ice <- function(
  import.datfile,
  export.path,
  export.stats,
  export.plot,
  detect.offset.high=10,
  detect.width.high=40,
  detect.offset.low=70,
  detect.width.low=40
) {
  raw = read.table(import.datfile)
  time = raw$V1
  temp = raw$V2

  #plot(time, temp, type="l")


  # init plot
```

```
  plot.init.grey(time, temp)

  stats.drop = process.drop(
    temp,
    time,
    export.path=export.path,
    export.stats=export.stats,
    detect.mode="min.after",
    detect.offset.high=detect.offset.high,
    detect.width.high=detect.width.high,
    detect.offset.low=detect.offset.low,
    detect.width.low=detect.width.low,
    label.position="center"
  )

  plot.save(export.path, export.plot=export.plot)

  stats.drop

}



import.datfile = join(IMPORT_PATH, "ice1.dat")
export.path = EXPORT_PATH_SOL
export.stats = "icewater1_stats.csv"
export.plot = "icewater1.pdf"

#analyse.ice(import.datfile, export.path, export.stats, export.plot)

st.w = c()  # water
st.i = c()  # ice (solid)

for(i in 1:3) {
  stats = analyse.ice(
    join(IMPORT_PATH, "ice", i, ".dat"),
    EXPORT_PATH_SOL,
    export.plot=join("icewater", i, ".pdf"),
    export.stats=join("icewater", i, "_stats.csv")
  )
  st.w = rbind(st.w, stats)
}

detect.widths.high = c(40,40,20)

for(i in 1:3) {
  stats = analyse.ice(
    join(IMPORT_PATH, "ice", i+3, ".dat"),
```

```
    EXPORT_PATH_SOL,
    export.plot=join("icecubes", i, ".pdf"),
    export.stats=join("icecubes", i, "_stats.csv"),
    detect.offset.high=10,
    detect.width.high=detect.widths.high[i],
    detect.offset.low=40
  )
  st.i = rbind(st.i, stats)
}
```

A.3.8 Uncertainty estimate of melting enthalpy of ice

In [1]:
```
import math

from metas_unclib import *

from Metas.UncLib.LinProp import UncBudget

import pandas as pd

import pandas as pd

import itables

itables.init_notebook_mode(all_interactive=True)
```

In [2]:
```
def unc_budget(unc_item, show_table=True):
    tree = UncBudget.ComputeTreeUncBudget(unc_item.net_object)

    table = pd.DataFrame(
        columns=("description", "uncertainty component", "uncertain
        index=range(len(tree)+1)
    )

    for i, elem in enumerate(tree):
        table.loc[i] = (
            elem.get_Description(),
            elem.get_UncComponent(),
            elem.get_UncPercentage(),
        )
    table.loc[len(tree)] = (
        "SUMMARY",
        unc_item.stdunc,
        100.,
    )

    return table.sort_values("uncertainty percentage", ascending=Fa
```

In [3]:
```
def tolerance(value, a):
    """
    producer tolerance of value +/- a

    returns UniformDistribution(value - a, value + a)
    """
    return UniformDistribution(value - a, value + a)
```

In [ ]:
```

```

$$\Delta_m H_E^{sp} = c_W^{sp} \left(\theta_{1f} \frac{M_{EW}}{M_E} \frac{\Delta\theta_2}{\Delta\theta_1} - \theta_{2f} \right)$$

In [4]:
```
#water_c_sp = ufloat(4182, desc="water specific heat capacity / J/K
water_c_sp = 4182
```

In [5]:
```
water_m1 = np.array((955.78, 988.3, 1002.60))  # g
water_m2 = np.array((937.38, 970.8,  984.50))  # g
water_m = water_m1 - water_m2  # g

ice_m1 = np.array((20.65, 24.97, 26.04))  # g
ice_m2 = np.array(( 6.31,  7.26,  8.49))  # g
ice_m = ice_m1 - ice_m2

(water_m, ice_m)
```

Out[5]:
```
(array([18.4, 17.5, 18.1]), array([14.34, 17.71, 17.55]))
```

In [ ]:
```

```

In [6]:
```
ice_temp_final = np.array((8.898531, 6.263302, 6.412726))  # °C
ice_temp_delta = np.array((-10.48647, -12.13266, -11.59748))  # K

water_temp_final = np.array((17.54753, 17.22439, 16.99892))  # °C
water_temp_delta = np.array((-2.524441, -2.409830, -2.492528))  # K
```

In [ ]:
```

```

In [7]:
```
ice_H_sp = (water_temp_final * water_m / ice_m * ice_temp_delta / wa
ice_H_sp
```

Out[7]:
```
array([353926.84412912, 332164.77441224, 314320.21564004])
```

In [11]:
```
ufloatfromsamples(ice_H_sp) / 1000
```

Out[11]:
```
333.4706113937995 ± 25.140414947719453
```

In [9]:
```
"330 ± 50 kJ/kg"
```

Out[9]:
```
'330 ± 50 kJ/kg'
```

```
Quelle Wikipedia: 333,5 kJ/kg
```

In [10]:
```
ice_H_sp.std() / np.sqrt(2) * 4.3
```

Out[10]:
```
49243.92859011987
```

A.3.9 Plotting helper scripts

```
# by github.com/janjoch, 2024

library(latex2exp)

# HELPER FUNCTIONS
join <- function(...) {
  paste(..., sep="")
}

# copied from https://stackoverflow.com/questions/1826519/how-to-assign-
↪  from-a-function-which-returns-more-than-one-value
':=' <- function(lhs, rhs) {
  frame <- parent.frame()
  lhs <- as.list(substitute(lhs))
  if (length(lhs) > 1)
    lhs <- lhs[-1]
  if (length(lhs) == 1) {
    do.call(`=`, list(lhs[[1]], rhs), envir=frame)
    return(invisible(NULL))
  }
  if (is.function(rhs) || is(rhs, 'formula'))
    rhs <- list(rhs)
  if (length(lhs) > length(rhs))
    rhs <- c(rhs, rep(list(NULL), length(lhs) - length(rhs)))
  for (i in 1:length(lhs))
    do.call(`=`, list(lhs[[i]], rhs[[i]]), envir=frame)
  return(invisible(NULL))
}

# usage
# func_that_returns_three_values <- function(a,b,c) {
#   return(list(a,b,c))
# }
# c(a,b,c) := func_that_returns_three_values(1,2,3)


predict.linear <- function(reg.a, reg.b, x) {
  y = reg.a + reg.b * x
  y
}


FBy <- function(x, y, sy, ...) {
  # copied from Meister
  arrows(x, y - sy, x, y + sy, code=3, angle=90, length=0.02)
  points(x, y, pch=21, ...)
```

```
}



# PLOT MODULAR FUNCTIONS
plot.init.grey <- function(
  x,
  y,
  init.par=TRUE,
  xlim=NULL,
  ylim=NULL,
  xlab=expression(italic(t)*" / "*"s"),
  ylab=expression(italic(T)*" / "*"°C"),
  type="l",
  lwd=2,
  col="darkgrey",
  ...
) {
  if(init.par) {
    par(mai = c(1,1.2,0.3,0.3))
  }
  plot(
    x,
    y,
    type=type,
    lwd=lwd,
    col=col,
    xlim=xlim,
    ylim=ylim,
    xlab=xlab,
    ylab=ylab,
    ...
  )
  plot.grid()
}


plot.grid <- function(nx=NULL, ny=NULL, lty=1, col="lightgray", lwd=1,
↪  ...) {
  grid(
    nx = nx,
    ny = ny,
    lty = lty,      # Grid line type
    col = col, # Grid line color
    lwd = lwd,      # Grid line width
    ...
  )
}
```

```
plot.axis.log <- function(begin=1e-10, end=1e10) {
  d = 10^c(-99:99)
  d = d[d>=begin & d<=end]
  dd = outer(c(1:9), 10^c(-99:99))
  dd = dd[dd>=begin & dd<=end]
  dlab = do.call(
    "expression",
    lapply(seq(along=log10(d)), function(i) substitute(10^E,
    ↪  list(E=log10(d)[i])))
  )
  list("big"=d, "small"=dd, "big.lab"=dlab)
}

plot.line.highlight <- function(x, y, col="blue") {
  lines(
    x,
    y,
    col=col,
    lw=3
  )

}

plot.line.annot <- function(x, y) {
  lines(
    x,
    y,
    lty=3,
    lw=2,
    col="black"
  )

}

plot.annot <- function(x, y, text, xjust=0.5, yjust=0.5, adj=0.15, ...) {
  legend(
    x,
    y,
    text,
    bg="white",
    box.col="white",
    adj=adj,
    xjust=xjust,
    yjust=yjust,
    ...
  )
}
```

```
plot.regression <- function(
  x,
  y,
  subset=NULL,
  draw.annotation=TRUE,
  slope.annot=(function(slope)(TeX(paste(r"(slope:)", sprintf("%0.4f",
  ↪  slope), "K/s")))),
  slope.annot.x.offset=0,
  slope.annot.y.offset=0,
  abline.lty=2,
  delta.annot.x=(function(delta)(TeX(paste(r"(\Delta t =)",
  ↪  sprintf("%0.2f", delta), "s")))),
  delta.annot.y=(function(delta)(TeX(paste(r"(\Delta T =)",
  ↪  sprintf("%0.2f", delta), "K")))),
  ...
) {
    if(!is.null(subset)) {
      x = x[subset]
      y = y[subset]
    }
    reg <- lm(y ~ x)
    y.pred <- predict(reg)

    x.min = min(x)
    x.max = max(x)
    y.pred.min = min(y.pred)
    y.pred.max = max(y.pred)

    x.delta = x.max - x.min
    y.pred.delta = y.pred.max - y.pred.min


    if(draw.annotation) {
      # delta t
      plot.line.annot(
        c(min(x), max(x)),
        c(min(y.pred), min(y.pred))
      )
      plot.annot(
        mean(c(min(x), max(x))),
        min(y.pred),
        delta.annot.x(x.delta)
      )

      # delta T
      plot.line.annot(
        c(max(x), max(x)),
```

```
        c(min(y.pred), max(y.pred))
      )
      plot.annot(
        max(x),
        mean(c(min(y.pred), max(y.pred))),
        delta.annot.y(y.pred.delta)
      )

      # slope
      plot.annot(
        min(x) + slope.annot.x.offset * x.delta,
        max(y.pred) - slope.annot.y.offset * y.pred.delta,
        slope.annot(summary(reg)$coef[2,1]),
        xjust=0.3,
        yjust=1
      )
    }

    # Regressionsgerade
    abline(reg, lty=abline.lty, lw=2, ...)

    reg
}

plot.save <- function(export.path, export.plot) {
  dev.copy2pdf(file=join(export.path, export.plot), width=WIDTH,
  ↪  height=HEIGHT)
}
```