Digitalization of Food Properties using Python with Applications

Prepared by:

Dr. Gokhan Bingol (gbingol@pebytes.com)

October 30, 2023

(Initial publication: December 05, 2022)

Document version: 2.0

Updates will be available at: https://www.pebytes.com/pubs

Follow on YouTube: https://www.youtube.com/@sciencesuit

Table of Contents

1. INTRODUCTION....3
2. FUNDAMENTALS....5
2.1. Construction of a Food Object....5
2.2. Comparison of Food Objects....8
2.3. Mathematical Operations....10
2.3.1. Addition....10
2.3.2. Subtraction....12
2.3.3. Multiplication....14
2.3.4. Pitfalls....14
2.4. Thermo-physical Properties....16
2.4.1. Specific heat capacity (Cp)....16
2.4.2. Thermal conductivity (k)....16
2.4.3. Density (ρ)....16
2.5. Dielectric Properties....18
2.5.1. Meat and meat products....18
2.5.2. Fruits and vegetables....18
2.5.3. Cereal grains....19
2.6. Water Activity....20
2.6.1. Raoult's law....20
2.6.2. Money-Born equation....21
2.6.3. Norrish equation....21
2.6.4. Ferro Fontan-Chirife-Boquet equation....21
2.6.5. Change of a_w with temperature....22
2.6.6. Method selection for a_w prediction....23
3. APPLICATIONS....24
3.1. Material Balance....24
3.2. Energy Balance....27
3.3. Freezing....29
3.3.1. Unfrozen food....29
3.3.2. Frozen food....29
3.3.3. Estimation of initial freezing temperature....30
3.3.4. Estimation of ice fraction....30
3.4. Water Activity / Drying....33
3.5. Heat Transfer....38
4. DISCUSSION....42
5. ACKNOWLEDGMENT....44
6. REFERENCES....45

1. INTRODUCTION

In broad terms digitalization refers to the process of utilizing digital technologies to change the core of business conduct. If correctly implemented, digitalization can lead to digital transformation and thereby improve productivity, reduce costs and pave the way for the future of manufacturing[1]. A literature survey by Demartini *et al*. (2018) found that the topic of the digitalization in the food industry has been studied since 2016 with the common keywords associated "Factory of the future" and "Food". The authors stated that the food companies are slower to adopt digital technologies. Nonetheless like all processing industries, the food industry is also seeking ways to enhance efficiency, reduce costs and become more environmentally friendly. Adoption of digitalization can play a significant role in achieving these goals.

Digitalization of food is challenging and Britannica[2] defines food as "*substance consisting essentially of protein, carbohydrate, fat, and other nutrients …*". Not only food has a complex composition, but also comes in various shapes, colors, odors, etc. Therefore, to tackle the high level of complexity, some level of abstraction was required. Therefore, in this work food was considered to consist of macronutrients (carbohydrate, lipid and protein) and also water, ash and salt. This is consistent with USDA NAL Database[3] which has compositional data of approximately 9000 food items. This level of abstraction facilitates various tasks. However, it does not exempt us from the complexity of carbohydrates, lipids and proteins that are divided into sub-groups with different physical and thermal properties.

In programming languages, two major trends can be seen: i) procedural programming, ii) object-oriented programming. Several object-oriented programming languages support operator overloading[4] (e.g. C++, Python etc.). These languages allow definitions such as $Food = Food + Food$, therefore enabling construction of new food items from existing ones, such as after a mixing operation.

Foods are also subject to various operations that may require the knowledge of different physical properties. For example, to calculate the heat required to raise the food's temperature requires specific heat capacity(C_p) whereas heat transfer modeling requires thermal conductivity and C_p. For microwave processing, dielectric properties should also be known.

[1] https://social-innovation.hitachi/en-us/think-ahead/manufacturing/industrial-digitalization-for-smart-manufacturing/
[2] Britannnica, https://www.britannica.com/topic/food
[3] FoodData Central Data, https://fdc.nal.usda.gov/download-datasets.html
[4] Wikipedia, https://en.wikipedia.org/wiki/Operator_overloading

In this document, the complexities of calculation/automation of various food properties, as well as their use in food process calculations will be reduced by using the **scisuit's**[5] open-source food class that can be found at GitHub[6]. Reduction in computational complexities not only will shorten the amount of work but will also enable to model a wider range of processes.

The target audience of this work is food process engineers. This document assumes that the reader already has some knowledge in food/chemical engineering concepts and basic to intermediate level of understanding of Python. The code used in this document was generated in a Windows 11 operating system using Visual Studio Code (1.83.1) environment, running Python 3.10.6. Detailed examples of applications in food process engineering will be presented.

[5] At least v1.1.2 (https://pypi.org/project/scisuit/)
[6] GitHub, gbingol (Gokhan Bingol) · GitHub

2. FUNDAMENTALS

2.1. Construction of a Food Object

In order to perform any digital operations on food, it must first be defined and constructed. Food is considered as an object comprised of a combination of carbohydrate (CHO), fat, protein, water, ash and salt. It should be noted that salt has been considered as a separate entity than ash as it has considerable effect on electric and dielectric properties.

To define a food object, first **scisuit**'s *food process engineering* (**fpe**) library, which is part of engineering library (**eng**), should be imported, `import scisuit.eng.fpe`. For example, milk may have approximately 88.13% water, 3.15% protein, 4.80% CHO, 3.25% lipid and 0.67% ash. Therefore, to define *milk* as a *food* object:

```
import scisuit.eng.fpe as fpe

milk = fpe.Food(water=88.13, protein=3.15, cho=4.8, lipid=3.25, ash=0.67)
print(milk)
```

```
Type = Food
Weight (unit weight) = 1.0
Temperature (C) = 20.0
water (%) = 88.13
cho (%) = 4.8
protein (%) = 3.15
lipid (%) = 3.25
ash (%) = 0.67
aw = 0.98
```

Note that, although it is generally easier to work with percentages it is possible to specify fractions as well, e.g. *water=0.8813*. It is seen from the output that the milk has a weight of 1 unit where the unit of the weight can be anything from gram, kg to lbs. Although the temperature was not explicitly set, a default temperature was assigned as 20°C. Here it should be noted that behind the scenes it is possible to express weight as a fraction and work with the fraction; however, this is not possible with the temperature itself. Therefore, Celsius was chosen for the unit of temperature. In the output only the percentages of *available* ingredients are listed. If the result of the computation of water activity (a_w) is different from None then a_w is included in the output as well.

When defining a Food object, if the sum of the percentages was not equal to 100%, then it would *have been adjusted* to be 100%.

```
f = fpe.Food (water=40, protein=20)
print(f)
```

```
Type = Food
Weight (unit weight)=1.0
Temperature (C)=20.0
water (%)= 66.67
protein (%)= 33.33
aw=0.908
```

In the above example, adjustment is simply made by dividing with the sum, such as *water=40/(40+20), protein=20/60*.

In the example on page 5, although the variable name was *milk*, there is no accurate mechanism to have a prior knowledge that it can be a dairy product let alone milk itself. However, in the literature, there are specific computations, such as freezing temperature, water activity or dielectric properties, available for different food groups. To take advantage of these specific computations and therefore increase accuracy, the variable, namely *milk*, can also be constructed in the following way:

```
milk = fpe.Dairy(water=88.13, protein=3.15, cho=4.8, lipid=3.25, ash=0.67)
print(milk)
```

```
Type = Dairy
Weight (unit weight) = 1.0
Temperature (C) = 20.0
water (%) = 88.13
cho (%) = 4.8
protein (%) = 3.15
lipid (%) = 3.25
ash (%) = 0.67
aw = 0.98
```

Now that the variable *milk* is defined to be a dairy product, any equations specialized for dairy products will be used for *milk*. Although USDA's classification[7] for foods is rather comprehensive and detailed, for the sake of simplicity 10 sub-groups for foods were assumed:

[7]FoodData Central, https://fdc.nal.usda.gov/fdc-app.html#/

1. Beverage	5. Juice	9. Sweet
2. Cereal	6. Legume	10. Vegetable
3. Dairy	7. Meat	
4. Fruits	8. Nut	

Although, in the above examples, *milk* was defined using 5 ingredients, it is not necessarily the case. For example, *grape* can be defined as a food item as well:

```
grape= fpe.Fruit (water =80, cho=20)
print(grape)
```

```
Type = Fruit
Weight (unit weight)=1.0
Temperature (C)=20.0
water (%)= 80.0
cho (%)= 20.0
aw=0.976
```

2.2. Comparison of Food Objects

Let f_1, f_2 be two different food objects. $f_1 = f_2$ if and only if f_1 and f_2 have the same ingredients in exact quantities and belong to the same base- or sub-group type. The following example should clarify the concept.

```
#Example 1 : different ingredients (protein vs cho)
f1= fpe.Food (water =80, protein=20)
f2= fpe.Food (water =80, cho=20)
print(f"Different ingredients: {f1 == f2}") #    == is the comparison operator

#Example 2: different % of same ingredients (cho’s different)
f1= fpe.Food (water =80, cho=20)
f2= fpe.Food (water =60, cho=40)
print(f"Same ingredients different %: {f1 == f2}")

#Example 3 : same % and same ingredients
f1= fpe.Food (water =40, cho=10)
f2= fpe.Food (water =80, cho=20)
print(f"Same ingredients and %: {f1 == f2}")

#Example 4: same % and ingredients but different sub-groups (Dairy vs Fruit)
f1= fpe.Dairy (water =80, cho=20)
f2= fpe.Fruit (water =80, cho=20)
print(f"Different sub-groups: {f1 == f2}")

#Example 5: same % and ingredients but different groups (Food vs Fruit)
f1= fpe.Food (water =80, cho=20) #base
f2= fpe.Fruit (water =80, cho=20) #sub-group
print(f"Different groups: {f1 == f2}")
```

```
Different ingredients: False
Same ingredients different %: False
Same ingredients and %: True
Different sub-groups: False
Different groups: False
```

Examples 1-3 are fairly straightforward to understand; however, examples 4 & 5 require further clarification which is presented in the following figure:

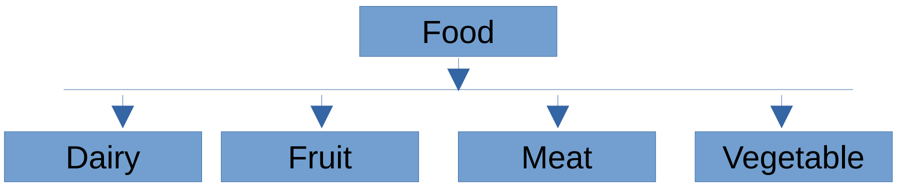


***Fig. 2.1:** Simple food hierarchy*

In Fig. (2.1) for simplicity only 5 nodes are presented and on the top node, the *Food* class is found. Below the *Food* class 4 specialized classes, which are sub-classes of *Food* class, are shown. At this point it is helpful to understand that in object oriented programming languages a sub-class is a super class but a super class is not a sub-class. Therefore, for example, a *Dairy* is a *Food* but a *Food* is not (necessarily) a *Dairy*. Thus when equality operator is called, initially, the node type (class type) is checked and if the class types are different than f_1 and f_2 are considered as different. If the class types are the same, then the composition is inspected and if the compositions are same then f_1 and f_2 are considered as equal.

Although at this stage, the above assumptions are satisfactory for many applications it is still not 100% accurate. Consider apples and oranges, which belong to class *Fruit*. If the low protein contents (<1%) are omitted both apples and oranges have very similar composition in terms of CHO and water and thus would be considered as equal. However, needless to say, we should not mix apples and oranges since as shown below they belong to different sub-classes:

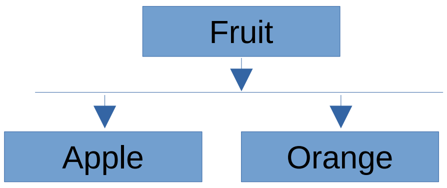


***Fig. 2.2:** Simple fruit hierarchy*

It is seen that without any further information (color, texture, etc.) composition alone cannot be the sole decision maker and the class of the food item must taken into account as well.

2.3. Mathematical Operations

The food object supports some essential arithmetic operations: Let f_1, f_2, f_3 and f_4 denote different food items and *a* and *b* an arbitrary numbers.

2.3.1. Addition

In terms of unit operations, addition can be considered as mixing of different food items.

$$a \cdot f_1 + b \cdot f_2 = f_3 \tag{2.1}$$

Addition of two food items could yield a food item that is different than the operands (f_1, f_2). If f_1 and f_2 belong to the same sub-group then f_3 will belong to the same sub-group. Otherwise it will belong to the base of f_1 and f_2.

```
f1= fpe.Food (water =80, cho=20)
f2= fpe.Food (water =60, cho=40)

f3 = f1 + f2
f4 = 2*f1 + 3*f2

print(f3)
print(f4)
```

```
Type = Food
Weight (unit weight) = 2.0
Temperature (C) = 20.0
water (%) = 70.0
cho (%) = 30.0
aw = 0.959

Type = Food
Weight (unit weight) = 5.0
Temperature (C) = 20.0
water (%) = 68.0
cho (%) = 32.0
aw = 0.955
```

Here, it should be noticed that behind the scenes mass balance was performed automatically. It has already been mentioned that if the temperature was not explicitly set, the temperature would be set to 20°C by default.

If f_1 and f_2 were at different temperatures, then energy balance would be performed as well, as demonstrated in the following script:

```
#Continuing from the previous example
f1.T = 30
f2.T = 50

f5 = 2*f1+f2
print(f5)
```

```
Type = Food
Weight (unit weight) = 3.0
Temperature (C) = 36.04
water (%) = 73.33
cho (%) = 26.67
```

Assuming the reference temperature as 0°C, energy balance is computed in the following way:

Mass balance:

$$m_{\mathrm{in}}=m_1+m_2=m_{out} \tag{2.2}$$

Energy balance:

$$E_{\mathrm{in}}=m_1 \cdot cp_1 \cdot T_1+m_2 \cdot cp_2 \cdot T_2=E_{out} \tag{2.3}$$

The energy exiting the system can be expressed:

$$E_{out}=m_{out} \cdot cp_{average} \cdot T_{mix} \tag{2.4}$$

where average C_p is,

$$Cp_{average}=\frac{m_1 \cdot Cp_1+m_2 \cdot Cp_2}{m_1+m_2} \tag{2.5}$$

Note that in Eq. (2.5), specific heat capacity was assumed to be linear in terms of temperature. This assumption is reasonable since the leading coefficient of the polynomials in Eq. (2.8) are in the order of 10^{-6}.

Finally, let's observe the effect of adding *i)* Same sub-group, *ii)* a sub-group and a base, *iii)* different sub-groups:

```
#both are Dairy (same sub-group)
f1= fpe.Dairy (water =80, cho=20)
f2= fpe.Dairy (water =60, cho=40)

print(f1 + f2)
```

```
#sub-group + base (Meat + Food)
f3= fpe.Meat (water =80, cho=20)
f4= fpe.Food (water =60, cho=40)

print(f3 + f4)


#sub-group + sub-group (Fruit + Vegetable)
f5= fpe.Fruit (water =80, cho=20)
f6= fpe.Vegetable (water =60, cho=40)

print(f5 + f6)
```

```
Type = Dairy
water (%) = 70.0
cho (%) = 30.0
Type = Food
water (%) = 70.0
cho (%) = 30.0

Type = Food
water (%) = 70.0
cho (%) = 30.0
```

The idea here is simple when visualized in terms of Fig. (2.1). Addition of classes belonging to same node yields that particular node type, whereas, addition of different nodes yields the base class (*Food*). Therefore, addition of f_1 and f_2 yielded *Dairy*, whereas f_3+f_4 and f_5+f_6 yielded the base class type, *Food*.

2.3.2. Subtraction

Subtraction can be likened to extraction, drying, etc. and is defined as follows:

$$f_1 - f_2 = f_3 \tag{2.6}$$

It should be noted that, similar to addition, subtraction operation could give a food item that is different than the operands (f_1, f_2). Unlike addition where temperatures and the compositions of f_1 and f_2 could be different, the following conditions must be met for Eq. (2.6) to succeed:

1. f_1 and f_2 must belong to the same group/sub-group.
2. f_1 must have all of the ingredients f_2 has and moreover the amount in f_2 cannot be greater than f_1,
3. The temperatures of f_1 and f_2 must be equal.

Let’s turn milk into milk powder by removing water, in other words we are performing a drying operation:

```
milk = fpe.Dairy(water=88.13, protein=3.15, cho=4.80, lipid=3.25, ash=0.67)
water = fpe.Food(water=100)

#remove water from milk
powder = milk - 0.87*water
print(powder)
```

```
AssertionError: Foods must have same type
```

Now that we have violated *rule #1*, an error has been raised. But logically we know that we can remove water from milk and get milk powder, which is a dairy product. Therefore, at this point we need to change our approach:

```
#generalize milk and water as Food
milk = fpe.Food(water=88.13, protein=3.15, cho=4.80, lipid=3.25, ash=0.67)
water = fpe.Food(water=100)

#remove water from milk (Food - Food = Food)
powder = milk - 0.87*water
print(powder)

#convert “general” powder into a dairy powder
dairy_powder = fpe.Dairy(**powder.ingredients()) #ingredients returns a Python dictionary
print(dairy_powder)
```

```
Type = Food
Weight (unit weight) = 0.13
water (%) = 8.69
cho (%) = 36.92
protein (%) = 24.23
lipid (%) = 25.0
ash (%) = 5.15

Type = Dairy
Weight (unit weight) = 1.0
water (%) = 8.69
cho (%) = 36.92
protein (%) = 24.23
lipid (%) = 25.0
ash (%) = 5.15
```

2.3.3. Multiplication

$$a \cdot f_1 = b \cdot f_1 = f_1 \tag{2.7}$$

There is no logical definition as $food = food \times food$, therefore unlike addition or subtraction, which yielded new food materials, multiplication does not yield a new one. It only changes the unit weight.

```
f1 = fpe.Food (water =60, cho=40)
f2 = 2*f1

print(f2)
```

```
Type = Food
Weight (unit weight) = 2.0
Temperature (C) = 20.0
water (%) = 60.0
cho (%) = 40.0
aw = 0.937
```

Overloading (redefinition) of multiplication operator is necessary to be able to perform addition or subtraction operations. Consider the following code snippet: `f4 = 0.2*f1 + 0.8*f2`

First of all, here we are implicitly saying that f_4 is comprised of 20% f_1 and 80% f_2. To perform the addition, behind the scenes temporary food materials from 0.2*f1 and 0.8*f2 are created, say $temp_1$ and $temp_2$, respectively. Now f_4 can be computed as $f_4 = temp_1 + temp_2$. Therefore, when addition is performed the input weights can be taken into account to correctly compute the composition and temperature of f_4.

2.3.4. Pitfalls

It has been mentioned in section 2.3.1 and that addition and subtraction operations could give a new food item. Conceptually this is correct, however, mathematically using the new food object <u>*as is*</u> will most likely *result in error*.

Let’s demonstrate it with a simple example:

```
f1 = fpe.Food (water =80, cho=20)
f2 = fpe.Food (water =60, cho=40)

#create a food mix (weight=1+1=2)
f_mix = f1 + f2
```

```
milk = fpe.Food(water=88.13, protein=3.15, cho=4.80, lipid=3.25, ash=0.67)
water = fpe.Food(water=100)

#create a food powder (weight=1-0.87=0.13)
powder = milk - 0.87*water

#create another food item from the mix and the powder
newmix = 2*powder + 3*f_mix

print(newmix)
```

```
Type = Food
Weight (unit weight) = 6.26 (=2*0.13 + 3*2) #ERROR
Temperature (C) = 20.0
water (%) = 67.45
 ...
```

There is error in weight and percentages of ingredients of *newmix* since *f_mix* and *powder* were just result of mathematical operations. Before we could use *f_mix and powder* as independent food items, they must to be *normalized*. This is simply done by calling the *normalize* member function.

```
f_mix.normalize()
powder.normalize()

#mixing f_mix and powder
newmix = 2*powder + 3*f_mix
print(newmix)
```

```
Type = Food
Weight (unit weight) = 5.0 #(=2*1.0 + 3*1.0), 1.0 is due to normalization
Temperature (C) = 20.0
water (%) = 45.48
cho (%) = 32.77
protein (%) = 9.69
lipid (%) = 10.0
ash (%) = 2.06
aw = 0.863
```

The *normalize* member function simply resets the weight to 1.0. After normalizing *f_mix and powder*, the weight of *newmix* is the expected 5 units and also note the differences in percentages of ingredients.

2.4. Thermo-physical Properties

Eqs. 2.8 to 2.10 are taken from Choi & Okos (1986). Unless otherwise stated, *T* are in °C.

2.4.1. Specific heat capacity (Cp)

$$
\begin{aligned}
&Cp_{Water}=4.1289-9.0864\cdot10^{-5}\cdot T+5.4731\cdot10^{-6}\cdot T^2\\
&Cp_{Protein}=2.0082+1.2089\cdot10^{-3}\cdot T-1.3129\cdot10^{-6}\cdot T^2\\
&Cp_{Lipid}=1.9842+1.4733\cdot10^{-3}\cdot T-4.8008\cdot10^{-6}\cdot T^2\\
&Cp_{CHO}=1.5488+1.9625\cdot10^{-3}\cdot T-5.9399\cdot10^{-6}\cdot T^2\\
&Cp_{Ash}=1.0926+1.8896\cdot10^{-3}\cdot T-3.6817\cdot10^{-6}\cdot T^2\\
&Cp_{Salt}=0.88
\end{aligned}
\qquad \textbf{(2.8)}
$$

where the unit of C_p is *kJ/kg°C* and C_p of salt is from Engineering Toolbox[8].

2.4.2. Thermal conductivity (k)

$$
\begin{aligned}
&k_{Water}=4.57109\cdot10^{-1}+1.7625\cdot10^{-3}\cdot T-6.7036\cdot10^{-6}\cdot T^2\\
&k_{Protein}=1.7881\cdot10^{-1}+1.1958\cdot10^{-3}\cdot T-2.7178\cdot10^{-6}\cdot T^2\\
&k_{Lipid}=1.8071\cdot10^{-1}-2.7604\cdot10^{-4}\cdot T-1.7749\cdot10^{-7}\cdot T^2\\
&k_{CHO}=2.0141\cdot10^{-1}+1.3874\cdot10^{-3}\cdot T-4.3312\cdot10^{-6}\cdot T^2\\
&k_{Ash}=3.2962\cdot10^{-1}+1.4011\cdot10^{-3}\cdot T-2.9069\cdot10^{-6}\cdot T^2\\
&k_{Salt}=5.704 \quad \text{at } 20°\text{C}
\end{aligned}
\qquad \textbf{(2.9)}
$$

where the unit of *k* is *W/m°C* and *k* value of salt is taken from (Riedel 1962).

2.4.3. Density (ρ)

$$
\begin{aligned}
&\rho_{Water}=997.18+3.1439\cdot10^{-3}\cdot T-3.7574\cdot10^{-3}\cdot T^2\\
&\rho_{Protein}=1329.9-5.1840\cdot10^{-1}\cdot T\\
&\rho_{Lipid}=925.59-4.1757\cdot10^{-1}\cdot T\\
&\rho_{CHO}=1599.1-3.1046\cdot10^{-1}\cdot T\\
&\rho_{Ash}=2423.8-2.8063\cdot10^{-1}\cdot T\\
&\rho_{Salt}=2165
\end{aligned}
\qquad \textbf{(2.10)}
$$

where the unit of ρ is *kg/m*^{3} and density of salt is found from Wikipedia[9].

Let's see the thermo-physical properties in action:

[8]Engineering Toolbox, https://www.engineeringtoolbox.com/specific-heat-capacity-d_391.html

[9] Wikipedia, https://en.wikipedia.org/wiki/Sodium_chloride

```
milk = fpe.Food(water=88.13, protein=3.15, cho=4.80, lipid=3.25, ash=0.67)

print(f"cp={milk.cp()}")
print(f"rho={milk.rho()}")
print(f"k={milk.k()}")
```

```
cp=3.852 [10]
rho=1041.6
k=0.4569
```

Now let’s change the temperature to 50°C:

```
milk.T = 50
print(f"cp={milk.cp()}")
print(f"rho={milk.rho()}")
print(f"k={milk.k()}")
```

```
cp=3.865
rho=1033.3
k=0.494
```

It is seen that ρ decreased from 1041.6 to 1033.3 kg/m^3 as temperature increased from 20 to 50°C. To see how ρ_{milk} changes with temperature in the range of say 10 to 75°C, the following code is needed:

```
import scisuit.plot as plt

T = range(10, 80, 5)
rho = [milk.rho(v) for v in T]

plt.scatter(x=T, y=rho)
plt.xlabel("Temperature(°C)")
plt.ylabel("Density (kg/m3)")
plt.show()
```

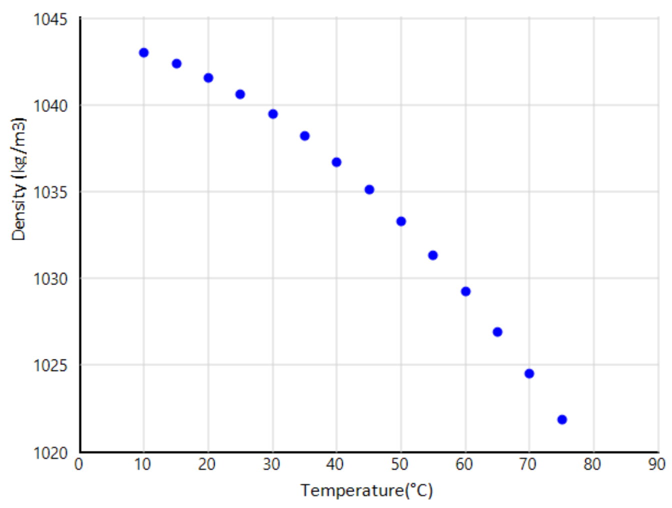


Fig. 2.3: *Change of milk’s density with temperature*

[10] Actual Python output is “cp=3.8521885907719993” which has been manually rounded for presentation purposes. Similar reasoning applied elsewhere as well.

2.5. Dielectric Properties

Dielectric properties of foods are affected by many factors, such as temperature and moisture content of food and frequency of the microwaves. Therefore, there are several equations for estimation of dielectric constant (ε′) and loss factor (ε″) in literature for particular food items. Gulati & Datta (2013) and Calay *et al*. (2007) present predictive equations for certain food categories, such as cereal grains, meat and meat products and fruits and vegetables.

2.5.1. Meat and meat products

If the meat contains salt (0-6%) and T (°C)=[0, 70]:

$$\begin{aligned} \epsilon' &= -52 - 0.03T + 1.2X_w + (4.5 + 0.07T)X_{salt} \\ \epsilon'' &= -22 - 0.013T + 0.48X_{water} + (4 + 0.05T)X_{salt} \end{aligned} \quad \textbf{(2.11)}$$

If the fat content (0-20%) is known and T (°C)=[-30, 0]:

$$\epsilon' = 29.3 + 0.076T - 0.3X_w - 0.11X_{fat} \quad \textbf{(2.12)}$$

If T (°C)=[-30, 0]:

$$\begin{aligned} \epsilon' &= 23.6 + 0.0767T - 0.231X_w \\ \epsilon'' &= 9.8 + 0.028T - 0.0117X_w \end{aligned} \quad \textbf{(2.13)}$$

If the mass fraction of ash is known:

$$\begin{aligned} \epsilon' &= X_w(1.0707 - 0.0018485T) + M_{ash}4.7947 + 8.5452 \\ \epsilon'' &= X_w(3.4472 - 0.01868T + 0.000025T^2) + M_{ash}(-57.093 + 0.23109T) - 3.5985 \end{aligned} \quad \textbf{(2.14)}$$

where X_w is the moisture content (wet basis), X_{fat} and X_{salt} are fat and salt contents in percentages and M_{ash} is the mass fraction of ash.

2.5.2. Fruits and vegetables

If *f=2.45 GHz* and *T(°C)=[0, 70]* and *X*

$$\begin{aligned} \epsilon' &= 2.14 - 0.104T + 0.808X_w \\ \epsilon'' &= 3.09 - 0.0638T + 0.213X_w \end{aligned} \quad \textbf{(2.15)}$$

If *f=[0.9, 3] GHz* and T (°C)=*[0, 70]* and X_w *(%)=[50, 90]*:

$$\begin{aligned}\epsilon' &= -12.8 - 0.103\,T + 0.788\,X_w + 5.49\,f \\ \epsilon'' &= 10.1 + 0.008\,T + 0.221\,X_w - 3.53\,f\end{aligned} \quad \textbf{(2.16)}$$

If neither of the above conditions are matched, then the following equations are used:

$$\begin{aligned}\epsilon' &= 38.57 + 0.1255 + 0.456\,X_w - 14.54\,X_{ash} - 0.0037\,T\,X_w + 0.07327\,X_{ash}\,T \\ \epsilon'' &= 17.72 - 0.4519\,T + 0.001382\,T^2 - 0.07448\,X_w + 22.93\,X_{ash} - 13.44\,X_{ash}^2 \\ &\quad + 0.002206\,X_w\,T + 0.1505\,X_{ash}\,T\end{aligned} \quad \textbf{(2.17)}$$

where X_w, X_{ash} are moisture and ash contents (%), respectively.

2.5.3. Cereal grains

If *f=[2, 3] GHz* and T (°C)=*[10, 30]* and X_w *(%)=[3, 30]*:

$$\begin{aligned}\epsilon' &= 1.71 + 0.0701\,X_w \\ \epsilon'' &= 0.12 + 0.00519\,X_w\end{aligned} \quad \textbf{(2.18)}$$

If *f=[2, 3] GHz* and T (°C)=*[10, 30]* and X_w *(%)=[3, 30]*:

$$\begin{aligned}\epsilon' &= 1.82 + 0.0621\,X_w - 0.0253\,f \\ \epsilon'' &= 1.72 + 0.066\,X_w - 0.0254\,f + 0.0003\,\rho_d\end{aligned} \quad \textbf{(2.19)}$$

If none of the above conditions are matched, then the following equations are used:

$$\begin{aligned}\epsilon' &= \left(1 + \frac{0.504\,X_w\,\rho_b}{\sqrt{X_w} + logf}\right)^2 \\ \epsilon'' &= 0.146\,\rho_b^2 + 0.004615\,X_w^2\,\rho_b^2\left(0.32\log(f) + 1.743/\log(f) - 1\right)\end{aligned} \quad \textbf{(2.20)}$$

where *f* is the frequency and ρ_b is the bulk density (kg/m^3).

2.6. Water Activity

The definition of water activity (a_w) is (Scott 1953):

$$Water\ activity = \frac{Water\ vapor\ pressure\ of\ food}{Water\ vapor\ pressure\ of\ pure\ water} \qquad \textbf{(2.21)}$$

It is seen from Eq. (2.21) that, a_w is not directly related to the amount of ingredients but more to how ingredients bind the water which can be through various mechanisms. Therefore, to accurately estimate a_w a truly finer detail of knowledge on ingredients is required than the definition of Food object presented in section 2.1. However, it is still possible to estimate a *rough* value of a_w using the predictive equations. Several equations for prediction of water activity are listed in the book by Barbosa-Cánovas *et al*. (2007). Since we will be using the term *solute* actively in this section, let's first define it:

$$Solute = CHO + lipid + protein + ash \qquad \textbf{(2.22)}$$

2.6.1. Raoult's law

Raoult's law is the basic equation for computing a_w of ideal solutions (Şahin and Sumnu 2006):

$$a_w = \frac{X_w}{X_w + \left(\frac{M_w}{M_s}\right) \cdot X_s} \qquad \textbf{(2.23)}$$

where X is the mass fraction and subscripts w and s denotes water and solute, respectively, and M stands for molecular weight. **scisuit** uses a slightly modified version of Eq. (2.23):

$$a_w = \frac{X_w}{X_w + \left(\frac{M_w}{M_s}\right) \cdot X_s + 2 \cdot \left(\frac{M_w}{M_{salt}}\right) \cdot X_{salt}} \qquad \textbf{(2.24)}$$

2.6.2. Money-Born equation

The equation is proposed by Money and Born (1951) and is used for calculating a_w of sugar confections, such as jams, fondant creams and boiled sweets (Barbosa-Cánovas *et al*. 2007).

$$a_w = \frac{1}{1+0.27n} \qquad \textbf{(2.25)}$$

where *n* is defined as:

$$n = \frac{m_{CHO}}{180.16} \qquad \textbf{(2.26)}$$

Note from Eq. (2.26) that **scisuit** assumes that CHO is made up of fructose (molecular weight of fructose is 180.16 g/mol).

2.6.3. Norrish equation

Proposed by Norrish (1966) and is useful for large concentrations of solute and used for non-electrolyte solutions containing both single and multiple solutes (Barbosa-Cánovas *et al*. 2007). In generalized form the equation can be expressed as:

$$\ln a_w = \ln X_{H_2O} + \frac{\sum K_i (X_i)^2}{\sum (X_i)^2} \cdot (1 - X_w)^2 \qquad \textbf{(2.27)}$$

where *X* is the mole fraction and *K* is the empirical constant for the solute.

2.6.4. Ferro Fontan-Chirife-Boquet equation

Developed by Fontan *et al*. 1981 (Barbosa-Cánovas *et al*. 2007).

$$a_w = X_w \left[e^{K_m \cdot X_s^2} \right] \qquad \textbf{(2.28)}$$

where X_w and X_s is mole fraction of water and solute, respectively, and *K* is the correlating constant for the solute. To compute the mole fractions, it was assumed that the amount of fructose, glycerol and alanine were equal to the amount of CHO, lipid and protein, respectively. Thus, the mole number (*n*) for CHO (n_{CHO}) was computed using Eq. (2.26), whereas n_{Lipid} and $n_{Protein}$ were computed using the following equation:

$$n_{Lipid} = \frac{m_{Lipid}}{92.0944} \quad \text{and} \quad n_{Protein} = \frac{m_{Protein}}{89.09} \tag{2.29}$$

K_m in Eq. (2.28) was computed using Ferro Fontan-Chirife-Boquet equation as follows:

$$K_m = \sum_{s=1}^{n} K_s C_s \left[\frac{\dot{M}}{M_s} \right] \tag{2.30}$$

where K_{CHO}, K_{Lipid} and $K_{Protein}$ are -2.15, -1.16 and -2.52 respectively. C_s is the weight ratio of solute s to total solids. $\dot{M}$ (average molecular weight) in Eq. (2.30) is expressed as:

$$\dot{M} = \sqrt{\sum_{s=1}^{n} \left(\frac{C_s}{M_s} \right)} \tag{2.31}$$

2.6.5. Change of a_w with temperature

To take into account the change of a_w with temperature, a modified version of Clausius-Clapeyron equation was used:

$$\frac{d \ln(a_w)}{d\, 1/T} = \frac{-Q_s}{R} \tag{2.32}$$

where R is the universal gas constant (kPa·m^3/kg·K) and Q_s is known as the moisture binding energy and several of them are tabulated by Iglesias and Chirife (1982). It should be noted that the values of Q_s greatly vary for different food items. Discretization of Eq. (2.32) gives,

$$\frac{\ln \frac{(a_{w2})}{(a_{w1})}}{\frac{1}{T_2} - \frac{1}{T_1}} = \frac{-Q_s}{R} \tag{2.33}$$

Since values of Q_s varies greatly an attempt was made to *guess* the desired energy:

$$Q_s = m_{average} \cdot Cp_{average} \cdot \Delta T \tag{2.34}$$

where $m_{average}$ was computed based on average molecular weight and ΔT is the temperature difference, the difference between food’s current temperature and its default temperature (20°C).

2.6.6. Method selection for a_w prediction

Before attempting to use any of the above equations, the following checks are made:

1. *%water* < 1 or *%water* > 99.99 $\rightarrow$ a_w=0.01 or a_w=0.999

2. %CHO>98 $\rightarrow$ a_w=0.70

3. 0<Solute<1% $\rightarrow$ a_w=0.99

If none of the above matches, then the following are checked:

1. If the food belongs to the group, namely *Sweet*, *Money-Born* equation is used.

2. If *%salt* ≥*1* or *%water*≥*90* $\rightarrow$ Raoult's law is used.

If the last two conditions are not satisfied, then the amount of solute is computed: if *%solute* ≥*70* $\rightarrow$ Norrish equation is used.

If neither of the above conditions are met then *Ferro Fontan-Chirife-Boquet* equation is used.

3. APPLICATIONS

3.1. Material Balance

Background

Material balance calculations are commonly used for formulating products from available raw materials, evaluating final compositions, processing yields, etc. (Toledo 2007).

In this section 3 examples will be presented. In the first example, we are interested in finding the final composition (output) of the food item from the given inputs, whereas in the 2nd and 3rd ones, the interest is to find the amount of input to obtain the requested output.

First example is straightforward and fairly intuitive whereas second example requires some counter-intuitive approach. For both examples there are two food items, namely *A* and *B* and compositions are as follows:

1. *Food A* (15% protein, 20% fat, 63% water),
2. *Food B* (3% protein, 80% fat, 15% water)

Let's form two food objects from the above-given compositions:

```
import scisuit.eng.fpe as fpe

A = fpe.Food(protein=15, water=63, lipid=20)
B = fpe.Food(protein=3, water=15, lipid=80)
```

Note that the percentages do not exactly add up to 100%.

Example 3.1.1

What would be the final composition if 90 kg of *A* and 10 kg of *B* are mixed?

Solution: Knowing that the final weight will be 100 kg, we can use the following straightforward approach:

```
C = 90*A + 10*B
print(C)
```

```
water (%) = 59.39
protein (%) = 14.08
lipid (%) = 26.53
```

Example 3.1.2

What is the required amounts of *A* and *B* to make a 100 kg of final product which contains 26.5% of fat?

Solution:

Note that the question is the opposite of Example (3.1.1) such that the final composition is given but the amounts of inputs are queried.

The code to solve this is very short but requires some care, first let’s look at the code:

```
C = 0.265*fpe.Food(lipid=1)
amounts = C.makefrom([A, B])

print(amounts)
```

```
[0.9005, 0.0995]
```

which means ~90 kg A and ~10 kg B should be used.

Things to note:

1. *C* contains other ingredients (Ex. 3.1.1), but it was defined as only containing lipid and fat content was to be 26.5%, however, was defined as 100% fat.
2. Although the final weight of mixture is 100 kg, it was not explicitly used in the code.

Let’s take a look under the hood and see how makefrom member function works:

1. Solves the equation $A \cdot x = b$ where the first row of ***A*** is all 1 and first entry of ***b*** is 1.
2. The 2nd, 3rd and so forth rows of ***A*** are based on the ingredients the food object has. So if it has only lipid, then 2nd row of ***A*** will be amount of lipid from each input food item and the second entry of ***b*** will be amount of lipid times its weight. Therefore, the linear system formed for this example is:

$$\begin{bmatrix} 1 & 1 \\ 0.2 & 0.8 \end{bmatrix} \cdot \begin{bmatrix} A \\ B \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \times 0.265 \end{bmatrix}$$

   Notice that *n* ingredients in the target food requires *n+1* input foods.

3. Each entry of the vector, namely ***x***, is between 0.0 and 1.0. However, it should be rather straightforward to scale it up to actual amounts.

Example 3.1.3

Determine the amounts of meat, fat, and water that must be used to make 100 kg of a sausage formulation (*adapted from Toledo 2007*).

1. *Meat:* 14% fat, 67% water, 19% protein,
2. *Fat:* 89% fat, 8% water, 3% protein,
3. *Sausage:* 20% fat, 65% water, 15% protein.

Solution:

This is a slightly more complex example than Example 3.1.2 and solution by hand can be error-prone (see Toledo 2007 page 81).

```
import scisuit.eng.fpe as fpe

#defining inputs
meat = fpe.Food(lipid=14, water=67, protein=19)
fat =fpe.Food(lipid=89, water=8, protein=3)
water = fpe.Food(water=100)

#target food
sausage = 85/100*fpe.Food(lipid=20, water=65) # 85/100 is to circumvent auto-adjustment

amounts = sausage.makefrom([meat, fat, water])
print(amounts)
```

```
[0.7732, 0.1031, 0.1237]
```

Therefore, approximately 77.3 kg meat, 10.3 kg fat and 12.4 kg water are needed to make 100 kg of sausage.

Notice that although *sausage* had 3 ingredients (fat, protein and water), only 2 of them, *lipid* and *water*, were used. Had we used *protein* instead of *water* in the definition of *sausage*, then there would have been a Runtime Error("*List contains food items that has no common ingredient with the target*").

Furthermore, in section 2.1, it was mentioned that when the sum of ingredients do not add up to 100%, it is automatically adjusted to 100%. In order to circumvent this auto-adjustment, in the assignment phase of *sausage*, the constructed food object was multiplied by *85/100*.

3.2. Energy Balance

Background

Energy balance on a system is based on the first law of thermodynamics. Energy balance calculations are used in almost all operations from evaporators, dryers to microwaves.

Example

Calculate the heat required to raise the temperature of a 4.535 kg roast containing 15% protein, 20% fat, and 65% water from 4.44 to 65.55°C.

Solution:

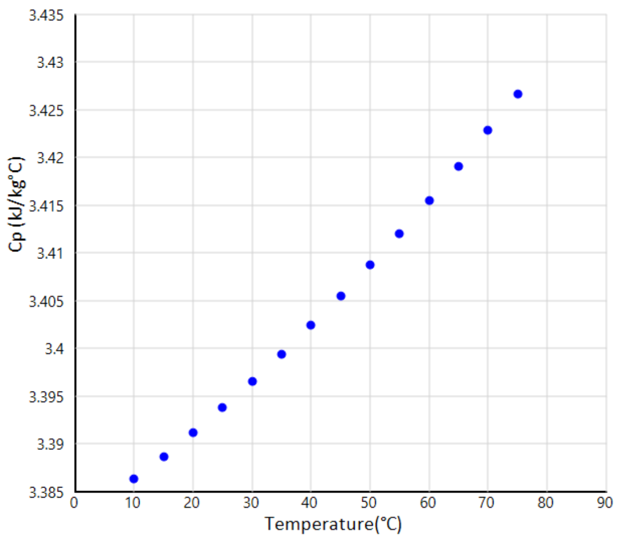
Before we proceed with the details, let's construct the food object:

```
import scisuit.eng.fpe as fpe

roast = fpe.Food(protein=15, lipid=20, water=65)
m = 4.535 #kg
t1, t2 = 4.44, 65.55 #°C
dT = t2-t1
```

The solution is fairly simple and all needed is to apply the following equation:

$$Q=m\cdot C_p\cdot\Delta T \quad \textbf{(3.1)}$$

However, before applying Eq. (3.1), let's see how Cp changes with temperature.



```
import scisuit.plot as plt
T = range(10, 80, 5)
cp = [roast.cp(v) for v in T]

plt.scatter(x=T, y=cp)
plt.xlabel("Temperature(°C)")
plt.ylabel("Cp (kJ/kg°C)")
plt.show()
```

Fig 3.1: Change of roast's Cp with temperatures

It is seen from Fig 3.1 that the Cp of roast increases with temperature, although in a very narrow range. Therefore, either Eq. (3.1) can be used as is or can be modified to take into account the change of Cp with temperature:

$$Q = m \cdot \frac{Cp(T_1) + Cp(T_2)}{2} \cdot (T_2 - T_1) \quad \textbf{(3.2)}$$

Application of Eq. (3.1):

```
Q= m*roast.cp()*dT
print(f"Heat required: {Q} kJ")
```

```
Heat required: 939.8 kJ
```

Application of Eq. (3.2):

```
Cp_avg = (roast.cp(t1) + roast.cp(t2))/2.0

Q= m*Cp_avg*dT
print(f"Heat required: {Q} kJ")
```

```
Heat required: 942.7 kJ
```

Since C_p was nearly linear and changed in a very narrow range (see Fig. 3.1), as expected there was almost no difference in the required energy computed using Eqs. (3.1) or (3.2).

Using Siebel's equation (Eq. 3.3), Toledo (2007) estimated C_p as 3.182 kJ/kg·K and the required heat as ~882 kJ.

$$C_{average} = 1674.72\,F + 837.36\,SNF + 4186.8\,M \quad \textbf{(3.3)}$$

where unit of $C_{average}$ is J/kg·K and F, SNF and M are mass fractions of fat, solids non-fat and moisture, respectively.

```
from scisuit.eng.fpe import Cp

cp = Cp(roast)

Q= m*cp.Siebel()*dT
print(f"Heat required: {Q} kJ")
```

```
Heat required: 881.8 kJ
```

3.3. Freezing

Background

Cooling is a fundamental operation in food processing and preservation (Toledo 2007). Above the freezing point, enthalpy (*h*) consists of sensible energy; however, below freezing point, *h* consists of both sensible and latent energy (ASHRAE 2006).

3.3.1. Unfrozen food

If the temperature is above freezing point then the enthalpy can be computed using the following equation (Chen 1985):

$$H = H_f + (T - T_f)\cdot(4.19 - 2.30\,x_s - 0.628\,x_s^3) \tag{3.4}$$

where H is the enthalpy (kJ/kg), H_f is the enthalpy at initial freezing temperature (kJ/kg), T is the temperature and T_f is the initial freezing temperature and x_s is the mass fraction of food solids.

H_f can be computed using Chang and Tao (1981) equation:

$$H_f = 9.79246 + 405.096\cdot x_{w0} \tag{3.5}$$

where x_{w0} is the mass fraction of water above initial freezing point.

3.3.2. Frozen food

The temperature is below freezing point and the enthalpy can be computed using the following equation (Chen 1985; ASHRAE 2006).

$$H = (T - T_r)\cdot\left[1.55 + 1.26\,x_s - \frac{(x_{w0} - x_b)L_0 T_f}{T_r\cdot T}\right] \tag{3.6}$$

where L_o is the latent heat of fusion of water = 333.6 kJ/kg, T_r is the reference temperature (typically -40°C) and x_b is mass fraction of bound water and can be computed as follows (Schwartzberg 1976; ASHRAE 2006).

$$x_b = 0.4\,x_p \tag{3.7}$$

where x_p is the mass fraction of protein.

3.3.3. Estimation of initial freezing temperature

Chang and Tao (1981) developed the following equations to estimate the initial freezing temperature (in K) of food items (ASHRAE 2006):

Meat group:

$$T_f = 271.18 + 1.47\,x_{w0} \tag{3.8}$$

Fruit/Vegetable Group:

$$T_f = 287.56 - 49.19\,x_{w0} + 37.07\,x_{w0}^2 \tag{3.9}$$

Juice Group:

$$T_f = 120.47 + 327.35\,x_{w0} - 176.49\,x_{w0}^2 \tag{3.10}$$

3.3.4. Estimation of ice fraction

Predicts the mass fraction of water that has crystallized below the initial freezing point, which is a function of temperature (ASHRAE 2006). Tchigeov (1979) developed the following equation:

$$x_{ice} = \frac{1.105\,x_{w0}}{1 + \dfrac{0.7138}{\ln\left(T_f - T + 1\right)}} \tag{3.11}$$

Example 3.3.1

A 150 kg beef carcass (57.26% water, 17.32% protein, 24.05% fat) is to be frozen to a temperature of –20°C. The initial temperature of the carcass is 10°C. How much heat must be removed? (*adapted from ASHRAE 2006*).

Solution:

The initial freezing point of T_f is -1.7°C (ASHRAE 2006). Let's first define the givens in Python language.

```
import scisuit.eng.fpe as fpe

carcass = fpe.Food(water=57.26, protein=17.32, lipid=24.05)
m = 150 #kg
Tf = -1.7
```

In order to compute the amount of heat removal enthalpy at both frozen (-20°C<T_f) and unfrozen (10°C>T_f) state must be known.

```
carcass.T = 10
h10 = carcass.enthalpy(Tf)

carcass.T = -20
h_20 = carcass.enthalpy(Tf)

Q = m*(h10 - h_20) #kJ
print(Q)
```

```
35004.1
```

Note that behind the scenes enthalpy of unfrozen food (*h10*) using Eqs. (3.4 & 3.5) and enthalpy of frozen food (*h_20*) was computed using Eqs. (3.6 & 3.7).

Note that the initial freezing point of T_f was found from ASHRAE (2006) as -1.7°C. When this information is not available Eqs. (3.8), (3.9) and (3.10) can be used. Let's predict the initial freezing point of the variable, namely *carcass*:

```
carcass = fpe.Food(water=57.26, protein=17.32, lipid=24.05)

Tf = carcass.freezing_T()
```

```
NotImplementedError: Only implemented for Juice, Fruit/Veggies and Meat
```

The error is clear, i.e. there is no *general equation* to compute the freezing point of foods; however, there are specialized equations to compute the freezing point of certain food groups. Therefore, the definition of *carcass* has to be modified:

```
carcass = fpe.Meat(water=57.26, protein=17.32, lipid=24.05)

Tf = carcass.freezing_T()
print(Tf) #°C
```

```
-1.12
```

Example 3.3.2

A 150 kg beef carcass is to be frozen to -20°C. What are the masses of the frozen and unfrozen water at -20°C? (*ASHRAE 2006*)

Solution:

First the fraction of water that is frozen needs to be found, in other words the ice fraction.

```
carcass = fpe.Meat(water=57.26, protein=17.32, lipid=24.05)
carcass.T = -20
print(f"Frozen fraction: {carcass.x_ice(-1.12)}")
```

```
Frozen fraction: 0.5178
```

Total amount of water: $150 \times 0.5726 = 85.89\, kg\, water$.

The amount of frozen water: $150 \times 0.52 = 78\, kg$

Unfrozen water: $85.89 - 78 = 7.89\, kg$.

3.4. Water Activity / Drying

Background

Water activity of foods is an important physical property to predict food stability and shelf life. For example, in mathematical modeling during storage of a food material, the storage conditions and packaging material can be simulated for extended shelf-life (Sahin & Sumnu 2006). There are several equations to predict the water activity of foods and an extensive discussion is presented by Barbosa-Cánovas *et al*. (2007).

Example 3.4.1

Predict the a_w of extruded pasta at moisture content of 9% (w.b.) at 35 and 50°C (*adapted from Heldman and Lund 2007*).

Solution:

The actual values of a_w at 35 and 50°C are 0.35 and 0.51, respectively.

```
f = fpe.Food(water=9, cho=91)

for t in [35, 50]:
    f.T = t
    print(f"At {t}°C aw={f.aw()}")
```

```
At 35°C aw=0.32
At 50°C aw=0.41
```

It is seen that the computation of a_w=*0.32* at 35°C is reasonably close to 0.35. whereas at 50°C, there is a deviation of ~0.1. For most purposes the computational results of a_w might be accurate enough to make some informed decisions.

Example 3.4.2

NaCl, sucrose or the NaCl-sucrose solutions are commonly used for osmotic dehydration of potatoes. Estimate aw of 20% sucrose solution, 20% NaCl solution and a solution containing 10% NaCl and 10% sucrose (Sahin & Sumnu 2006).

Solution:

```
nacl = fpe.Food(water = 80, salt=20)
sucrose = fpe.Food(water=80, cho=20)
```

```
solution = fpe.Food(water=80, cho=10, salt=10)

print(f"nacl = {nacl.aw()}")
print(f"sucrose = {sucrose.aw()}")
print(f"solution = {solution.aw()}")
```

```
nacl = 0.867
sucrose = 0.976
solution = 0.918
```

The values reported by Sahin & Sumnu (2006) are 0.867, 0.987 and 0.923 for *nacl*, *sucrose* and *solution*, respectively. The minor difference between the a_w's predicted and reported by Sahin & Sumnu (2006) is from the fact that CHO is assumed to be made up of fructose which has a molecular weight of 180.16 g/mol whereas molecular weight of sucrose is 342 g/mol. Therefore, the difference in molecular weights will affect result computed by Eq. (2.24). Please see section 4 for a discussion on possible remedies to overcome such differences.

Example 3.4.3

Outside air with a relative humidity of 60 % and dew point of 1 °C will be used for drying of 200 kg of sliced apples with an initial moisture content of 80%. The air is first heated to 50° C and then enters to the adiabatic dryer. The capacity of the blower is 1.5 m^3/s. The exit air has a dew point 20 °C. Plot moisture content – aw graph and compute the required drying time to store it safely.

Solution:

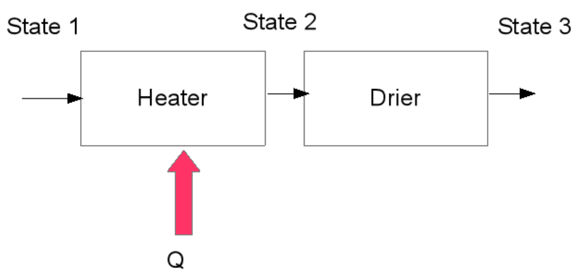


Fig. 3.2: *A sketch of the drying system*

State 1: *Outside air:* Pressure, RH and $T_{dew\text{-}point}$ known are known for the entering air.

State 2: *Heated air*: Simple heating does not change absolute humidity, therefore $W_2 = W_1$. Pressure and Tdry-bulb are known.

State 3: *Exiting the dryer:* In an adiabatic dryer, the enthalpy remains constant, therefore $H_3 = H_2$. Pressure and $T_{dew\text{-}point}$ are known.

The givens for humid air are:

```
P = 101.325 #kPa
Tdp1, Tdp3 = 1, 20 #dew-point temperatures
Tdb2 = 50 #dry-bulb temperature (heating)
V_flow = 1.5 #m3/s volumetric flow rate
```

Let’s define the moist-air’s 3 states and find the necessary psychrometric properties:

```
import scisuit.eng as eng

state1= eng.psychrometry(RH=60, Tdp=Tdp1, P=P)
w1, v1 = state1.W, state1.V #absolute humidity and specific volume

ma = V_flow/v1 #kg da/s (mass flow rate)
w2 = w1 #absolute humidity does not change during simple heating
state2 = eng.psychrometry(W=w2, Tdb=Tdb2, P=P)
h2 = state2.H #enthalpy

h3 = h2 #enthalpy does not change in adiabatic conditions
state3 = eng.psychrometry(H=h3, P=P, Tdp=Tdp3)
w3=state3.W #absolute humidity of exiting air
```

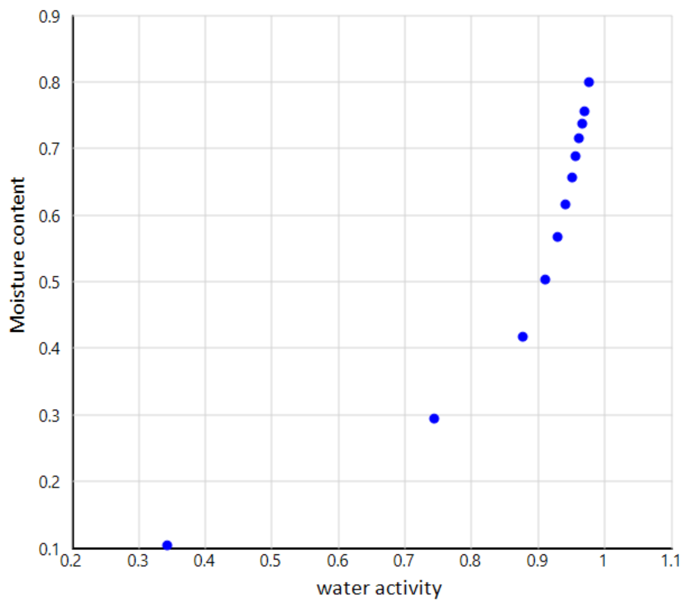
In the code below, we calculate the amount of water removed from the apple at given intervals and then find the moisture content and water activity at each point in the interval.

```
apple = eng.Food(water = 80, cho = 20) #define moist apple
water = eng.Food(water=100) #define water as food (as it is removed from apple)

mc, Aw, time = [apple.water], [apple.aw()], [0]
Duration = range(30, 140, 10) #intervals in minutes to be inspected
for t in Duration:
    mwater = ma*(w3-w1)*t*60 #amount of water removed
    driedapple = 200*apple - mwater*water

    mc.append(driedapple.water)
    Aw.append(driedapple.aw())
    time.append(t)
```

All that is left is to plot a scatter chart of MC vs aw and time vs aw.

```
plt.scatter(y=mc, x=Aw)
plt.xlabel("water activity")
plt.ylabel("Moisture content")

plt.figure()

plt.scatter(x=time, y=Aw)
plt.xlabel("time")
plt.ylabel("water activity")

plt.show()
```

The above code will produce the following plots:



As expected as the moisture content decreased the water activity decreased.

Note that until 40% moisture content, aw decreased very slowly but after then rather quickly.

***Fig. 3.3**: a_w vs MC (w.b.) for apple*

The reason for the aw to decrease slowly and then rather quickly could be the fact that different equations are used for different compositions. During drying the moisture content changes which corresponds to composition changes. The following table summarizes the equations used.

Table 3.1: Predictive equations used during drying of apple slices

% MC Range	Equation
80.0 - 71.48	Raoult
68.83 - 41.67	Ferro-Fontan, Chirife &Boquet
29.37 - 10.49	Norrish

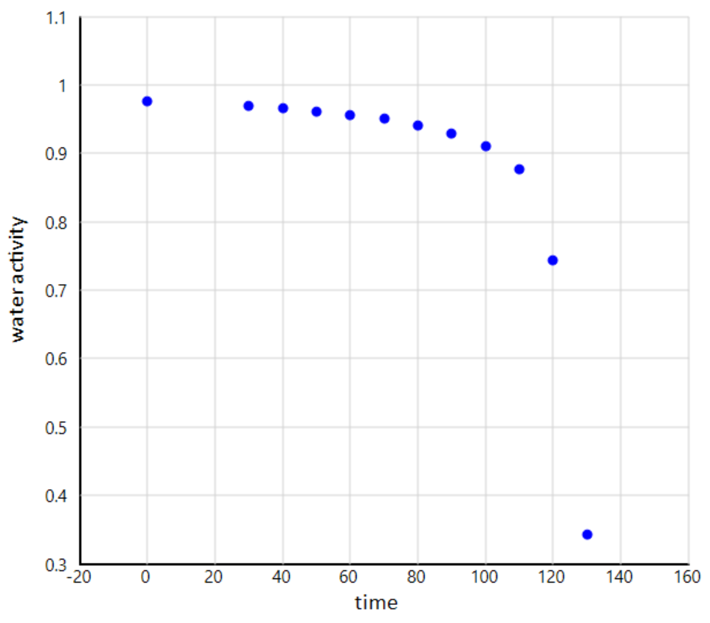


Fig. 3.4: Change of water activity with time

It is seen that after 120 minutes the apple slices' water activity decreased below 0.7 and therefore microbiological and most chemical reactions would be at its minimum (as seen from Fig 3.5) and therefore would be safe to store.

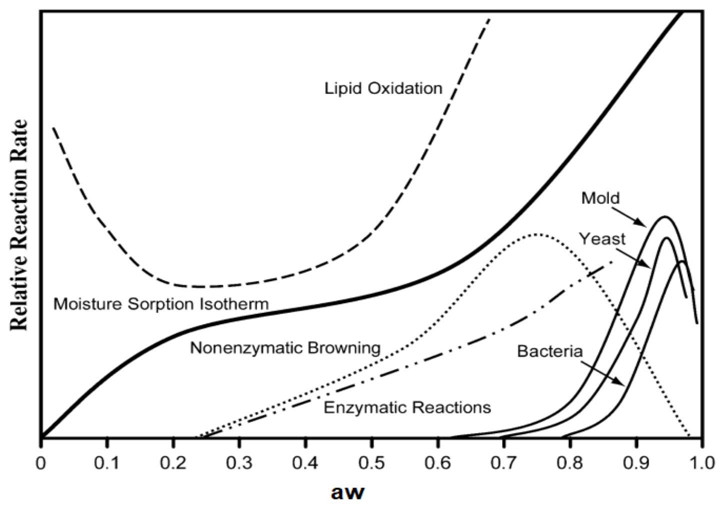


Fig. 3.5: Relative chemical& microbiological reaction rate as a function of water activity

Adapted from Barbosa-Cánovas *et al*. (2007).

<u>*Original from*</u>: **Labuza TP, Tannenbaum SR & Karel M** (1970). Water content and stability of low moisture and intermediate moisture foods. J*ournal of Food Technology*, **24**, 543–550

3.5. Heat Transfer

Background

In heat transfer analysis, it so happens that some bodies' interior temperature remains uniform throughout the heat transfer process and therefore the temperature of such bodies can be taken to be a function of time only, *T(t)* (Cengel 2002). The equation *T(t)* is:

$$\ln \frac{T(t)-T_\infty}{T_i-T_\infty}=\frac{-hA_s}{\rho V C_p}dt \tag{3.12}$$

where h is the convective heat transfer coefficient, *T(t)* is the temperature after time t, T_i is the initial temperature of the body and T_∞ is the temperature of the fluid. A_s is the surface area, V is the volume and ρ and C_p are density and specific heat capacity of the body respectively.

To use Eq. (3.12), *Bi* number, which is computed as shown below, should be less than 0.1.

$$Bi=\frac{h\cdot(V/A_s)}{k} \tag{3.13}$$

where k is the thermal conductivity of the body.

The computation of convective heat transfer depends on several factors; however, for the flow of gas or liquid over cylinder with circular cross-section the following equations can be used (Cengel 2002).

$$Nu=C\,\mathrm{Re}^m\,Pr^{1/3} \tag{3.14}$$

where the values for constants C and m depends on the Reynolds number. A Python function to compute Nusselt number in the Reynolds number range of 0.4-4000 is given below:

```
def Nu(Re, Pr):
    C, m = 0,0
    if 0.4 <= Re < 4:
        C , m= 0.989, 0.330
    elif 4 <= Re < 40:
        C, m = 0.911, 0.385
    elif 40 <= Re < 4000:
        C, m = 0.683, 0.466

    return C*Re**m*Pr**(1/3)
```

h can be computed using the following equation:

$$h = \frac{Nu \cdot k}{L_c} \tag{3.15}$$

where *k* is the thermal conductivity of the fluid and L_c is the characteristic length.

Example 3.5.1

Thompson seedless grape with 80% moisture content is put in a convective dehydrator operated at 58°C and with air flow at a velocity of 0.6 m/s. The initial temperature of grape is 26°C and its length and diameter are 29.7 and 1.9 cm, respectively. Obtain the center and surface temperatures of grape for the first 22 minutes (Bingol 2008).

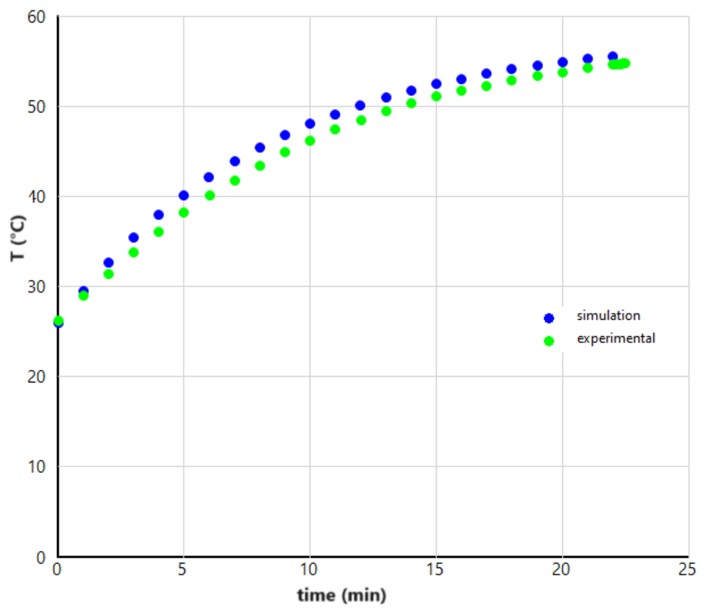
<u>*Solution*</u>:

Assumption: Grape is a cylinder with circular cross-section.

```
import scisuit.eng.fpe as fpe
import scisuit.eng as eng
import scisuit.plot as plt
from math import pi, exp

grape = fpe.Food(water=80, cho=20)

Tair, T0_grape = 58, 26 #C
V = 0.6 #m/s
D, L = 1.9 /100,  29.7/100 #m

#critical length
Lc = D / 2

#film temperature
Tf = (Tair + T0_grape)/2

air = eng.Air(T=Tf+273.15)

Re = (air.rho()*V*D) / air.mu()

Nu_ = Nu(Re, air.Pr())
h = Nu_*air.k() / Lc

""" Lumped Heat Capacity Analysis """
Volume = pi*(D**2/4)*L
SurfaceArea = 2*pi*D**2/4 + pi*D*L

Lc_lumped = Volume/SurfaceArea #critical length in lumped analysis
```

```
Bi = h*Lc_lumped/grape.k()
b = h /(grape.rho()* grape.cp()*1000*Lc_lumped)

# 22 minutes, sample every minute
t_sim = range(0, 23)
T_sim = [exp(-b*t*60)*(T0_grape - Tair) + Tair for t in t_sim]

# Experimental values
t_exp = [0, 1.007, 2.006, 3.01, 4.001, 5.009, 6.005, 7.007, 8.005, 9.002, 10.004, 11.004,
    12.005, 13.008, 14.005, 15.002, 16.001, 17.008, 18.005, 19.005, 20.007, 21.005,
    22.002, 22.104, 22.206, 22.309, 22.401, 22.506]

T_exp = [26.268, 28.933, 31.393, 33.828, 36.028, 38.173, 40.093, 41.778, 43.443, 44.953,
    46.208, 47.438, 48.498, 49.403, 50.308, 51.113, 51.673, 52.248, 52.873, 53.343, 53.748,
    54.218, 54.593, 54.573, 54.588, 54.653, 54.713, 54.733]

plt.scatter(x=t_sim, y=T_sim, label="simulation")
plt.scatter(x=t_exp, y=T_exp, label="experimental")
plt.legend()
plt.show()
```

If *h* and *Bi* were printed out, the values are 36.3 W/m^2°C and 0.38. Although *Bi*>0.1, the lumped heat transfer analysis can still be applied with minor errors. The accuracy of simulation is shown in the following figure:



It is seen that the accuracy of the prediction of experimental data by simulation is very well.

The average error was 1.49°C and the maximum error was 2.14°C at t=7 min.

Fig 3.6: Experimental vs simulated temperature profile of grapes

It should also be noted that the simulated temperatures were always greater than the experimental ones. This could be due to the fact that experimental values presented here are the average of center and surface temperatures of grape. Since *Bi* number is 0.38, there was still minor thermal resistance in the body that slowed down the heat transfer to the center and therefore caused minor thermal gradient, $T=T(r, t)$, instead of a uniform temperature distribution, $T=T(t)$. As a result the center temperature remained slightly lower than surface temperature. If, instead, simulated values were compared with the surface temperatures of grapes, the average and maximum errors were 1.15 and 1.99°C, respectively, both of which were lower than the comparison with the average temperatures.

4. DISCUSSION

As stated in the introduction section, food is a complex material and is subjected to various processes. Therefore, it would be naive to assume that the current work covered all aspects of food processing. However, this study clearly illustrated that digitalization of food properties has several benefits such as shortening the amount of work and decreasing the complexity to model a process and thereby paving the way for more complex studies.

The construction of a food object (section 2.1), requires parameters such as lipid, protein, CHO, salt only specified by percentages/fractions. However, it is known that there are many different types of lipids, proteins, CHO's and salts. In order to take into account these finer details, each of these macro-molecules can be defined as a Python class as shown in the following conceptual example:

```
# detailed definition of CHO
carbs = fpe.CHO(glucose = 40, fructose = 60)

#food made of water and CHO (glucose + fructose)
food = fpe.Food(CHO = 40* carbs, water = 60)
```

Having the above-level of of details will increase accuracy of predictions, however, it should also be noted that such an attempt will also drastically increase the complexity of the overall work.

It was also seen that when constructing a food object, if the sum of the percentages was not equal to 100%, then it would have been automatically *adjusted* to be 100% (see section 2.1). Although this approach ensures that a food object is compositionally well defined, if not paid attention it might lead to confusions and thereby errors especially when working with mass balances where a well-defined food object is not necessarily required. Therefore, in the following versions of **scisuit** auto-adjustment is planned to be optional.

It is possible to increase the accuracy of water activity predictions by including other available equations (such as Pitzer, Bromley, Ross, etc.). Furthermore, as shown above, including finer details of composition should further increase the accuracy of predictions. Thus, combined with thermo-physical properties such estimations would serve the basis for more complex mathematical models and would be valuable when making informed decisions.

Given the fact that programming languages such as Python, C++ offer many options for overloading operators (binary, arithmetic, etc.), not only the work presented here can be taken further but also the idea of digitalization of food properties can be extended to different types of materials.

5. ACKNOWLEDGMENT

I sincerely thank Prof Murat Ö. Balaban for his feedback in writing this document. His expertise, enthusiasm and attention to the detail not only have been an inspiration but also have been an invaluable asset in improving and refining this work.

6. REFERENCES

ASHRAE (2006). ASHRAE Handbook: Refrigeration, American Society of Heating, Refrigerating and Air-Conditioning Engineers, Amer Society of Heating, Atlanta, GA.

Barbosa-Cánovas GV, Fontana AJ, Schmidt SJ & Labuza TP (2007). Water activity in foods: fundamentals and applications, Blackwell Publishing and the Institute of Food Technologists.

Bingol G (2008). The Effects of Different Pretreatments on Drying Rate and Color Kinetics of Convective and Microwave-assisted Convective Drying of Thompson Seedless Grapes. Istanbul Technical University, PhD Thesis.

Calay RK, Newborough M, Probert D, Calay PS (1995). Predictive equations for the dielectric properties of foods. International Journal of Food Science and Technology, 29, 699-713.

Cengel YA (2002) Heat Transfer: A Practical Approach. 2nd Edition, McGraw-Hill, New York.

Chang HD & Tao LC (1981). Correlations of enthalpies of food systems. *Journal of Food Science*, 46:1493.

Chen CS (1985). Thermodynamic analysis of the freezing and thawing of foods: Enthalpy and apparent specific heat. *Journal of Food Science*, 50:1158.

Choi Y & Okos MR (1986). Effects of temperature and composition on the thermal properties of foods. Food Engineering and Process Applications, Vol. 1: Transport Phenomena (pp. 93–101). New York: Elsevier.

Demartini M, Pinna C, Tonelli F, Terzi S, Sansone C, Testa C (2018). Food industry digitalization: from challenges and trends to opportunities and solutions. *IFAC Papers Online*, 51-11, 1371-1378.

Fontan CF, Chirife J & Boquet R (1981). Water activity in multicomponent non-electrolyte solutions. *Journal of Food Technology*, **16**, 553–559.

Gulati T & Datta AK (2013). Enabling computer-aided food process engineering: Property estimation equations for transport phenomena-based models. *Journal of Food Engineering*, **116**, 483–504.

Heldman DR & Lund DB (2007). Handbook of Food Engineering, 2nd Edition, CRC Press, Boca Raton FL.

Iglesias HA & Chirife J (1982). Handbook of Food Isotherms: Water Sorption Parameters for Food and Food Components. Academic Press, New York.

Money RW & Born R (1951). Equilibrium humidity of sugar solutions. *Journal of Science Food Agric*. 2-180.

Norrish RS (1966). An equation for the activity coefficients and equilibrium relative humidities of water in confectionary syrups. *Journal of Food Technology*, 1-25.

Riedel L (1962). Thermal Conductivities of Aqueous Solutions of Strong Electrolyte, Chem.-1ng.-Technik., **23 (3)**, 59-64.

Sahin S & Sumnu SG (2006). Physical Properties of Foods. Springer New York, NY.

Schwartzberg HG (1976). Effective heat capacities for the freezing and thawing of food. *Journal of Food Science*, **41(1)**, 152-156.

Scott WJ (1953). Water relations of *Staphylococcus aureus* at 30°C. *Australian Journal of Biological Sciences,* **6**, 549–564.

Tchigeov G (1979). Thermophysical processes in food refrigeration technology. Food Industry, Moscow.

Toledo RT (2007). Fundamentals of Food Process Engineering, 3rd Edition, Springer New York, NY.