Python-Kurs, Blatt 09, WiSe 2020/21

Für dieses Blatt haben Sie die Wahl: Sie können *entweder* Aufgabe 1 bearbeiten, *oder* die Aufgaben 2-4. (Oder, wenn Sie Lust haben auch gerne alle Aufgaben – als Studienleistung ist aber nur einer der beiden Blöcke gefragt. Es werden insgesamt bis zu 3 Punkte angerechnet.)

Aufgabe 1 ist eine Aufgabe, die Ihnen im Forschungs-Alltag tatsächlich begegnen könnte, und hat daher einen weiteren Fokus. Dafür könnte Ihnen Ihre Arbeit auch weit über das Kursende hinaus nützlich sein (unabhängig von Ihrem Studienfach). Die übrigen Aufgaben stehen nicht in Kontext zueinander; hier liegt der Fokus auf den verschiedenen besprochenen Features der MatPlotLib.

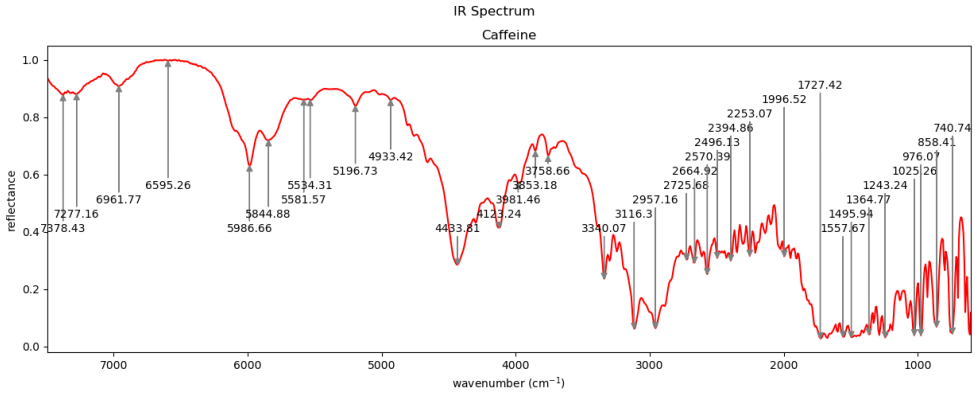
1 Mini-Projekt: Spektroskopie-Klasse

Wir wollen hier eine Klasse entwickeln, die echte Messdaten verarbeitet und ansprechend auf dem Bildschirm darstellt. Am Ende sollen die Zeilen:

```
class SpectrumPlot :
    # ... Ihr Code hier

caffeine = SpectrumPlot("caffeine.jdx")
caffeine.show()
```

die folgende Ausgabe auf dem Bildschirm erzeugen:



(oder oder eine Ausgabe, die dem gezeigten Bild möglichst nahe kommt.)

1.1 Informationen zur Quelldatei

Auf GRIPS finden Sie die Datei `caffeine.jdx`. (Tatsächlich wurde die Datei über `https://webbook.nist.gov/cgi/cbook.cgi?ID=C58082&Type=IR-SPEC&Index=1` aus der öffentlich zugänglichen Datenbank NIST-Datenbank entnommen.) Die Datei enthält ein IR-Reflexions-Spektrum von Koffein im *JCAMP Chemical Spectroscopic Data Exchange Format*. Dabei handelt es sich um ein CSV-basiertes Dateiformat, das von verschiedenen in der Chemie gängigen Gerätesoftware-Paketen benutzt wird. Sie können die Datei mit einem normalen Texteditor öffen und lesen.

Die Datei besteht aus zwei Blöcken, die von verschiedenen Messdurchgängen stammen. Jeder Block beginnt mit Kommentaren (die durch Raute-Symbole (`#`) eingeleitet werden, und endet mit der Zeile

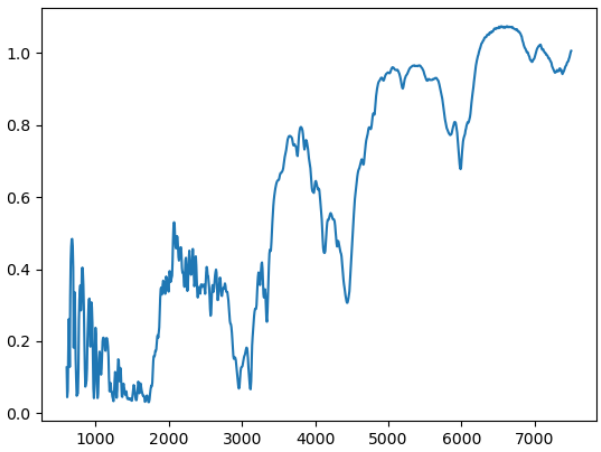
`##END=`. Die Kommentare enthalten Metainformationen wie z. B. Datum der Messung, die uns hier nicht näher interessieren müssen. Einzige Ausnahme soll die Zeile `##TITLE=...` sein.

Die eigentlichen Mess-Daten liegen als Tabelle vor, wobei die einzelnen Spalten durch ein Leerzeichen voneinander abgetrennt sind. Die erste Spalte enthält die sogenannte *Wellenzahl* in cm^{-1}, die auf der X-Achse aufgetragen wird. Die anderen Spalten enthalten die Intensität des Mess-Signals. Wir können uns hier auf die zweite Spalte beschränken.

Je nach Messgerät kann es sich dabei bereits um *normierte* Werte zwischen 0 und 1 handeln, oder um *counts*, d. h. eine Geräte-Interne Messeinheit. Für unsere Plots wollen wir alle Werte normieren, d. h. jeden einzelnen Wert einer Messreihe durch den größten Wert teilen.

1.2 Daten einlesen

Bevor Sie mit der Entwicklung der Klasse beginnen, schreiben Sie zuerst Code, der „nur“ die Messdaten aus der Datei einliest und in zwei Listen `wavenumbers` und `intensities` speichert. Benutzen Sie dabei das Modul `csv`. Testen Sie auch, ob Sie damit einen ersten Plot erstellen können. Dieser sollte dann so aussehen:



Erweitern Sie Ihren Code anschließend so, dass auch der Titel der Messreihe in einer Variable gespeichert wird, d. h. machen Sie die Zeile ausfindig, die mit `##TITLE=` beginnt, und speichern Sie das Ende dieser Zeile.

1.3 Grundgerüst der Klasse

Beginnen Sie nun mit dem Grundgerüst der Klasse `SpectrumPlot`. Machen Sie sich klar, ob, und wenn ja welche Klassen-Attribute Sie benötigen, und welche Instanz-Attribute Sie schon jetzt identifizieren können. Ein Instanzattribut soll ein `bool` `plotReady` sein, der den Default-Wert `False` hat. Dieses Instanzattribut soll angeben, ob bereits alle notwendigen Aufrufe der MatPlotLib schon geschehen sind.

Tipp: Legen Sie `fig = plt.figure()` als Instanzattribut an.

Legen Sie die Methode `loadFile(self, filename)` an. Die Magic Method `__init__(self, filename)` soll u. a. die Methode `loadFile` aufrufen. In `loadFile` soll das Flag `plotsReady` auf `False` zurückgesetzt

werden. Es soll auch möglich sein, eine Instanz von `SpectrumPlot` anzulegen, ohne eine Datei zu benennen, die geladen werden soll.

Hintergrund: Sie könnten Ihr Programm auch so konzipieren, dass die Methode `__init__` bereits das Laden übernimmt. Um ein Programm übersichtlich zu halten, sollten eigenständige Gedankenschritte aber in eigenständigen Methoden implementiert werden. Vorbereiten der Instanz-Attribute *ist ein Gedankenschritt;* Laden der Datei *ist ein weiterer.*

Schreiben Sie außerdem eine Methode `show` und eine Methode `preparePlots`. Wenn `plotsReady == False` soll `show` zunächst `preparePlots` aufrufen werden. Nachdem sichergestellt ist, dass der Plot vorbereitet wurde, soll `self.fig.show()` aufgerufen werden. Wann muss `plotsReady = True` gesetzt werden? (Ignorieren Sie für diesen Arbeitsschritt zunächst, dass wir später noch die Minima des Graphen markieren wollen. Achten Sie aber darauf, dass die X-Achse *absteigend* nummeriert ist, d. h. dass die größten Werte links stehen.)

Hintergrund: Auch hier wäre es leichter, lediglich eine Funktion `show` zu schreiben, die den Plot erstellt im selben Schritt auf dem Bildschirm ausgibt. Als Anwender wollen Sie aber vielleicht vor dem Anzeigen noch andere Elemente auf den Plot zeichnen, die Sie beim Entwickeln noch nicht vorgesehen hatten. Vielleicht wollen Sie dem geladenen Plot andere Daten zum Vergleich überlagern. Für diese Übung nutzen wir diese Möglichkeit zwar nicht; guter Code hält sich aber auch immer Möglichkeiten zur Erweiterung offen.

1.4 Minima finden

Schreiben Sie nun eine Funktion `findMinima`, die die Tiefpunkte des Plots ausfindig macht. Achtung: Messwerte sind in der Praxis immer „verrauscht“. Was wir als Mensch als Plateau wahrnehmen, enthält daher viele kleine Sprünge nach oben und unten, d. h. eine vielzahl von „falschen Minima“. Fällt Ihnen eine Methode ein, die diese falschen Minima erkennt/unterdrückt?

Tipp: Sie werden hierzu vermutlich neue Instanzvariablen einführen wollen.

Ihre Methode muss hierzu nicht perfekt sein. Wenn Sie nicht jedes kleine Rauschen als Minimum detektieren, haben Sie schon „gewonnen“.

1.5 Minima anzeigen

Ergänzen Sie Ihren Code in der Methode `preparePlot` so, dass die gefundenen Minima im Plot mit einem Pfeil und der zugehörigen Wellenzahl markiert werden.

Tipp: Die MatPlotLib hat manchmal Probleme mit senkrechten Pfeilen. Ein Pfeil von `(x, y1)` nach `(x + 1, y2)` sieht – für genügend große Werte von `x` – immer noch senkrecht aus und kann von der MatPlotLib ohne Probleme erzeugt werden.

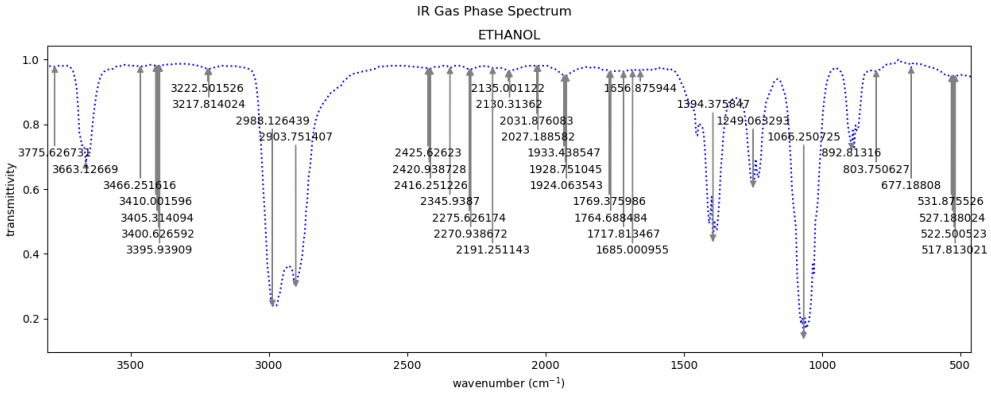
1.6 Robustheit prüfen

Benutzen Sie nun Ihre Klasse `SpectrumPlot`, um auch die bereitgestellte Datei `ethanol.jdx`[1] zu laden und anzuzeigen. Im bereitgestellten Lösungsvorschlag erzeugen die Zeilen:

[1] ebenfalls von der NIST Datenbank; siehe `https://webbook.nist.gov/cgi/cbook.cgi?ID=C64175&Mask=80`

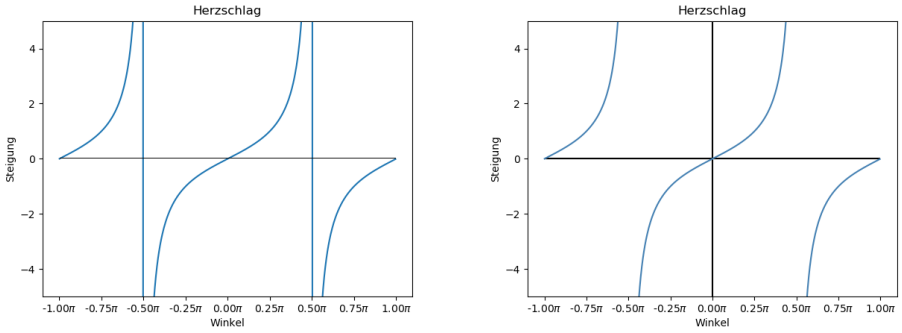
```
ethanol = SpectrumPlot("ethanol.jdx")
ethanol.supTitle = "IR Gas Phase Spectrum"
ethanol.linetype = "b:"
ethanol.ylabel = "transmittivity"
ethanol.show()
```

die folgende Ausgabe auf dem Bildschirm :

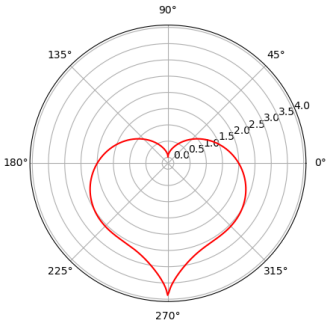


2 Tangens und Herz

Erzeugen Sie den *linken* der beiden folgenden Plots. *Optional*: Erzeugen Sie den *rechten* Plot:



Erzeugen Sie dann dieses Herz:



Benutzen Sie für das Herz:

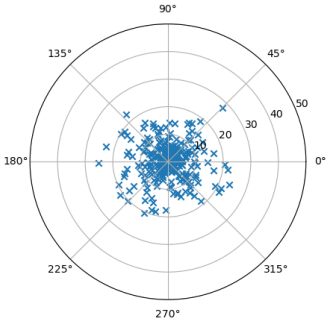
```
heart = lambda t : (math.sin(t) * math.sqrt( abs(math.cos(t)) )) / \
                   (math.sin(t) + 7/5) - 2 * math.sin(t) + 2
```

(Original-Gleichung von `https://pavpanchekha.com/blog/heart-polar-coordinates.html`

Den Herzschlag erhalten Sie einfach über `math.tan`.

3 Dartscheibe

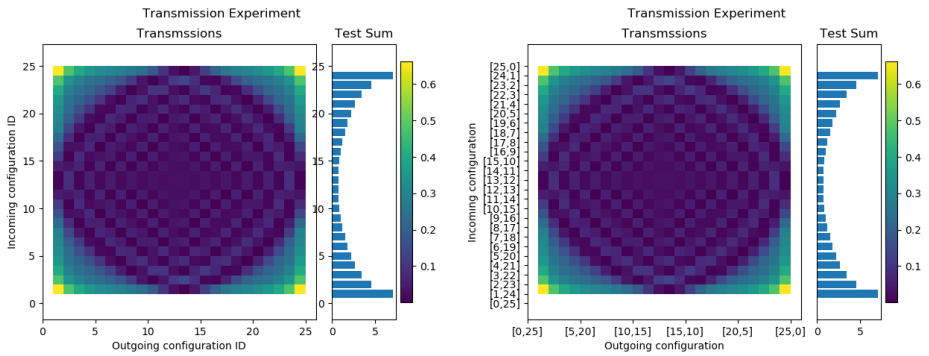
Simulieren Sie den Ausgang von 300 Dart-Würfen, und zeigen Sie die Treffer wie unten an:



Tipp: Benutzen Sie das Modul `random`.

4 Transmissionswahrscheinlichkeiten

Auf GRIPS finden Sie die Datei `transition-probabilties.txt`. Diese ist Teil meiner Masterarbeit, und enthält den Ausgang einer Simulation. Erzeugen Sie aus den Daten darin den linken der beiden Plots unten. *Optional* können Sie die Formatierung auch so erweitern, dass der rechte Plot angezeigt wird.



Anleitung:
Die Datei `transition-probabilties.txt` basiert auf dem CSV-Format. Als Trennzeichen wurde der Tabulator (`"\t "`) verwendet. Sie besteht aus zwei Daten-Blöcken, wovon der erste Block die „bunte Fläche“ beschreibt und der zweite Block die Daten für das Balkendiagramm enthält. Vor jedem Datenblock stehen Header-Zeilen, die Sie ignorieren können. Sie erkennen einen Datenblock daran, dass die Zeile mit `incoming` beginnt. In der ersten Daten-Zeile stehen auch die Spaltenüberschriften. Danach finden Sie in Spalte 1 die Zeilenüberschrift (z. B. `[12,13]`) und in allen weiteren Spalten die Ergebnisse der Simulation.

Sie werden feststellen, dass einige Ergebnisse als `nan` beschrieben sind. `nan` steht für *not a number*, und repräsentiert in der EDV Berechnungen, die durch mathematische Fehler abgebrochen wurden. $\sqrt{-1}$ liefert in vielen Implementierungen das Ergebnis `nan`. Python und die MatPlotLib können mit `nan`s „sinnvoll“ umgeheh: `float("nan")` liefert den „Zahlenwert“ `nan`; beim Plotten werden Felder mit Zahlenwert `nan` einfach ausgelassen.

Lesen Sie also die Datei zeilenweise ein, und überprüfen Sie, ob die aktuell geladene Zeile mit `incoming` beginnt. Wenn dies so ist, haben Sie den ersten Datenbereich erreicht. Benutzen Sie hierfür das Modul `csv`. Lesen Sie von hier weg die Datei weiter ein, bis Sie auf eine Leerzeile stoßen. Die hier mit `csv` eingelesenen Zeilen können Sie Spaltenweise mit `float(Spaltenwert)` in Zahlen verwandeln. Laden Sie so die ganze Wertetabelle als *zwei*dimensionale `list` in den Arbeitsspeicher.

Nach derselben Manier können Sie auch den zweiten Datenblock laden. Dieser sollte dann als *ein*dimensionale Liste vorliegen.