Übungsserie 1

Schreiben Sie für jede der folgenden Aufgaben die Python-Befehle oder Programme in eine separate Python-Textdatei *Gruppe_S1_AufgX.m* (Gruppe ersetzen Sie durch Ihre Gruppenbezeichnung, z.B. IT19a_WIN01, S1 steht für die Serie 1, X ist die Aufgabennummer), fassen Sie diese in eine ZIP-Datei *Gruppe_S1.zip* zusammen und laden Sie dieses File vor der nächsten Übungsstunde nächste Woche auf Moodle hoch. Die einzelnen Dateien müssen ausführbar sein und in den Kommentarzeilen (beginnen mit #) soll bei Funktionen ein Beispiel eines funktionierenden Aufrufs angegeben werden. Verspätete Abgaben können nicht mehr berücksichtigt werden.

Aufgabe 1 (ca. 30 Min.):

Schreiben Sie ein Skript oder eine Funktion `Gruppe_S1_Aufg1.py`, welches Ihnen die folgenden mathematischen Funktionen auf dem angegebenen Intervall plottet .

- $f(x) = x^5 - 5x^4 - 30x^3 + 110x^2 + 29x - 105$ für $x \in [-10, 10]$. Schränken Sie die $x-$Achse und $y-$Achse mit den Befehlen `plt.xlim()` und `plt.ylim()` so ein, dass Sie alle 5 Nullstellen des Polynoms vom Grafikfenster von Auge ablesen können. Blenden Sie dazu ein Gitternetz ein mit dem Befehl `plt.grid()`. Lesen Sie im User Guide für Matplotlib (https://matplotlib.org/users/index.html) nach, wie sie diese Befehle korrekt benutzen können.
- Geben Sie jetzt zusätzlich noch die Ableitungsfunktion $f'(x)$ und die Stammfunktion $F(x)$ von $f(x)$ an und plotten Sie diese in die gleiche Grafik. Die Integrationskonstante soll 0 sein. Beschriften Sie die Achsen (`plt.xlabel()`, `plt.ylabel()`) und fügen Sie einen Titel (`plt.titel()`) sowohl eine Legende (`plt.legend()`) für die drei Funktionen zum Bild hinzu.

Aufgabe 2 (ca. 60 Min.):

Verallgemeinern Sie nun Aufgabe 1 für ein beliebiges Polynom und schreiben Sie eine Funktion `Gruppe_S1_Aufg2.py`, welche Ihnen

- die Funktionswerte für das beliebige Polynom $p(x) = a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0$ vom Grad $n \geq 0$ für ein vorgegebenes x- Intervall berechnet sowie
- die Ableitungsfunktion $p'(x)$ als auch die Stammfunktion $P(x)$ gemäss den bekannten Ableitungs- bzw. Integralregeln für Polynome berechnet. Die Integrationskonstante soll dabei 0 sein.

Input ist ein Vektor $\boldsymbol{a}$ mit den Koeffizienten $a_0, a_1, ..., a_n$ sowie die Intervallsgrenzen $[x_{min}, x_{max}]$. Output ist der Vektor $\boldsymbol{x} = [x_0, x_1, ..., x_m]$ mit $x_{min} \leq x_i \leq x_{max}$ (wählen Sie für die x_i eine vernünftig kleine Schrittweite), der Vektor $\boldsymbol{p}$ mit den Funktionswerten, der Vektor $\boldsymbol{dp}$ mit den Werten der Ableitung sowie der Vektor $\boldsymbol{pint}$ mit den Werten der Stammfunktion.

Überprüfen Sie die Korrektheit Ihrer Implementierung, indem Sie ein zusätzliches Skript `Gruppe_S1_Aufg2_skript.py` schreiben, in dem Sie Ihre Funktion aufrufen und die in Aufgabe 1 erzeugte Abbildung reproduzieren.

Bemerkungen:

- Existierende Python Programme, die Polynome auswerten (wie z.B. `np.polyval()`) oder ableiten/integrieren dürfen nicht verwendet werden.

- Mit dem Befehl `np.shape(a)` können Sie die Dimension eines Arrays a bestimmen. Sie müssen sicherstellen, dass $\boldsymbol{a}$ nicht leer und entweder ein Zeilen- oder Spaltenvektor ist, ansonsten muss Ihre Funktion abbrechen und eine Fehlermeldung ausgeben (z.B. mit `raise Exception('Fehler')`).
- Die erste Zeile Ihrer Funktion wird von der Art sein

  `def IT19aZH01_S1_Aufg2(a,xmin,xmax)`

  Die letzte Zeile Ihrer Funktion wird von der Art sein

  `return(x,p,dp,pint)`
- Die Funktion kann z.B. als `IT19aZH01_S1_Aufg2.py` gespeichert werden (irgendeine andere Benennung mit der Endung *.py wäre aber auch möglich, z.B. `my_file.py` ... innerhalb der gleichen *.py Datei können Sie eben mehrere Funktionen mit natürlich verschiedenen Funktionsnamen definieren). Wenn Sie die Funktion in Ihrem Skript `IT19aZH01_S1_Aufg2_skript.py` aufrufen wollen, müssen Sie sie zuerst importieren, mit der Befehlszeile

  `from IT19aZH01_S1_Aufg2 import IT19aZH01_S1_Aufg2`

  Der Aufruf im Skript erfolgt dann z.B. mit `[x,p,dp,pint] = IT19aZH01_S1_Aufg2([2,1,3],-5,5)`

Aufgabe 3 (ca. 30 Min.):

Die Fakultät einer natürlichen Zahl n kann mit einer rekursiven Funktion berechnet werden der Art:

```
def fact_rec(n)
# y = fact_rec(n) berechnet die Fakultät von n als fact_rec(n) = n * fact_rec(n -1) mit fact_rec(0) = 1
# Fehler, falls n < 0 oder nicht ganzzahlig
  import numpy as np
  if n < 0 or np.trunc(n) != n:
    raise Exception('The factorial is defined only for positive integers')
  if n <=1:
    return 1
  else:
    return n*fact_rec(n-1)
```

Schreiben Sie ein Skript `Gruppe_S1_Aufg3.py`, welches die obige Funktion erhält sowie eine weitere Funktion (`fact_for()`), die die Fakultät nicht rekursiv berechnet sondern mit einer for-Schleife.

Vergleichen Sie beide Funktionen indem Sie die Ausführungszeiten der beiden Funktionen jeweils 100 mal berechnen und den Durchschnitt miteinander vergleichen, z.B. mit dem Befehl

- `t1=timeit.repeat("fact_rec(500)", "from __main__ import fact_rec", number=100)`

Beantworten Sie die folgenden Fragen als Kommentare im Skript:

- Welche der beiden Funktionen ist schneller und um was für einen Faktor? Weshalb?
- Gibt es in Python eine obere Grenze für die Fakultät von n
  - als ganze Zahl (vom Typ 'integer')? Versuchen Sie hierzu, das Resultat für $n \in [190, 200]$ als float auszugeben.
  - als reelle Zahl (vom Typ 'float')? Versuchen Sie hierzu, das Resultat für $n \in [170, 171]$ als float auszugeben.