



Das deutsche Python-Forum

[Python-Forum.de](#) [Foren-Übersicht](#) [Python Programmierforen](#) [Allgemeine Fragen](#)



Einlesen von IMC Famos (.dat, .raw) daten

[Antworten](#)

28 Beiträge

1

2



Squippy
User



Sonntag 30. Juni 2019, 17:08

Hallo Zusammen,
ich bin recht neu im Bereich Python unterwegs und habe großes Interesse, es für einige auswertungen und plotten von Messdaten zu verwenden. Allerdings gibt es offensichtlich noch keine Funktion, die das Einlesen von Famos dateien ermöglicht.

Ich bin bei der Suche danach jedoch auf MathLab Funktionen gestoßen (importieren oder Lesen), die genau das machen soll.

Meine Frage an euch ist nun, ob jemand mal drüber gucken kann und mir sagen kann, ob man es auch umschreiben kann für Python. Mir fehlt dafür leider noch etwas die Erfahrung und wenn es nicht funktioniert, kann ich mir erstmal die Zeit sparen um mich dort einzuarbeiten und muss evtl. doch auf MathLab umsteigen :/

Besten Dank und Gruß aus dem Rheinland!

Den Code zum Importieren von Famos daten mittel MathLab wäre dieser hier:

CODE: ALLES AUSWÄHLEN

```
function [dataOut]=FAMOSimport(filename)
% Usage: data=FAMOSimport(filename);
%
```

```

% FAMOSimport() opens (MARC generated) FAMOS files and imports all signals.
%
% *****
%
%

% Preset output to empty;
dataOut=[];

%% Check for valid input file
if exist('filename','var')~=1 ...
    || isempty(filename)
        [filename, pathname] = uigetfile( ...
            {'*.dat','FAMOS measurement files'; ...
            '*.*', 'All files'}, 'Select FAMOS measurement file ...');
        if isequal(filename,0)
            disp('FAMOS-measurement file import cancelled. ');
            return;
        end
        filename=fullfile(pathname, filename);
        clear pathname;
    end
    if exist(filename,'file')~=2
        disp('Given file could not be found. Aborting import. ');
        return;
    end

%% Load input file
fid=fopen(filename,'r','l');
data=fread(fid,inf,'uint8=>char','l');
fclose(fid);
clear fid

%% Parse header information
dataOut.FileName=filename;
dataOut.TYPE='AFT MARC (FAMOS)';
header=strfind(data(1:200),[char(13) char(10) 'INO,']);
subIdx=strfind(data(header(1):header(1)+50),',');
dataOut.Device=strtrim(data(header(1)+subIdx(5):header(1)+subIdx(6)-2));

%% Parse measurement entries information
units=strfind(data,[char(13) char(10) 'ICR.']):

```



__deets__
User



🕒 Sonntag 30. Juni 2019, 18:55

Das kann man sicher auch nach Python konvertieren.





ThomasL
User



🕒 Sonntag 30. Juni 2019, 20:17

Die Frage ist, hast du nur Zugriff auf die Dateien in dem Format oder Zugriff auf die IMC Software. Wenn letzteres, kann diese Dateien auch in ASCII oder Excel Format speichern.

https://en.wikipedia.org/wiki/Imc_FAMOS

Ich bin Pazifist und greife niemanden an, auch nicht mit Worten.

Für alle meine Code Beispiele gilt: "There is always a better way."

<https://projecteuler.net/profile/Brotherluii.png>



Squipy
User



🕒 Sonntag 30. Juni 2019, 20:40

Danke für eure Antworten,
ich habe zwar Zugriff auf die Software, allerdings steht es außer Frage, dass ich mit was anderem als .dat Dateien arbeite. Andere Software ist darauf ausgelegt und leider sind ASCII Dateien wesentlich größer.



__deets__
User



🕒 Sonntag 30. Juni 2019, 20:57

Wie gesagt - gehen tuts. Machen musst du es selbst, und dir dafür genug Python raufschaffen.



Sirius3
User



🕒 Sonntag 30. Juni 2019, 21:21

@Squipy: das was sich da DAT nennt, ist ja auch ASCII. Und mit ein bißchen Pythonkenntnissen auch nicht schwer zu entschlüsseln.



Squipy
User



🕒 Montag 22. Juli 2019, 07:33

Hi Zusammen,
wenn ich mir eine kleinen Beispieldatei ansehe, sind die ersten Zeilen nicht schwer zu identifizieren

CODE: ALLES AUSWÄHLEN

```
ICF,2,1,1;ICK,1,3,1,1;  
INv,1,13,7,1,14,32,0,0;  
INO,1,7,1,0,,0,;  
INL,1,10,1252,0x409;  
ICG,1,5,2,3,2;  
ICD,1,13,1,1,1,s,0,0,0;  
...
```

Das kann ich mir auch aus dem MatLab code herauslesen, was, welche Angabe ist. Allerdings sehen die tatsächlichen Messdaten (Zeitstempel und Messwert) so aus:



Kann mir da evtl. jemand auf die Sprünge helfen?
Danke!



Sirius3
User



🕒 Montag 22. Juli 2019, 07:54

Das sind binär codierte Doubles. Dafür gibts entweder das `struct`-Modul, oder Du benutzt `numpy.frombuffer` oder ähnliches.



Squipy
User



🕒 Sonntag 11. August 2019, 20:09

Danke für den Hinweis.

Ist es möglich die binär codierten Daten zu entpacken, wenn man die gröÙe nicht kennt? bspw. die unpack

funktion erwartet einen definierten buffer, wenn ich es richtig verstanden habe. Das heißt ich erhalte immer den error:

unpack requires a buffer of 32 bytes

Ist es möglich diese Daten auch ohne Information der Größe zu decodieren?

Die Datei ist etwa so aufgebaut, dass zu anfang strings Informationen zu den Messdaten enthalten sind und dann die binär codierten Zeitdaten kommen. Die Beschreibung dafür, habe ich gefunden und ich kann mir dementsprechend die für mich nötigen Informationen herausziehen. Das würde ich gerne alles in einem dict speichern, was auch bis zu den codierten doubles recht gut klappt. Die Dateistruktur sieht wie folgt aus:

CODE: ALLES AUSWÄHLEN

```
ICF,2,1,1;ICK,1,3,1,1;
INv,1,13,7,1,14,32,0,0;
INO,1,7,1,0,,0;
INL,1,10,1252,0x409;
ICG,1,5,2,3,2;
ICD,1,13,1,1,1,s,0,0,0;
INT,1,27,26, 8,2018,23,38,16.60999999;
ICC,1,3,1,1;
ICP,1,16,1,8,8,64,0,0,1,0;
ICb,1,28,1,0,1,1,0,4808,0,4808,1,0,0,;
ICR,1,11,0,0,0,1,1,A;
ICN,1,29,0,0,0,17,AI_CuRotorCurrent,0,;
ICC,1,3,2,1;
ICP,1,16,2,8,8,64,0,0,1,0;
ICb,1,31,1,0,2,1,4808,4808,0,4808,1,0,0,;
ICR,1,11,0,0,0,1,1,s;
ICS,1,9618,1, `Íý...@ àtÄ...@ À'3†@ `Û6†@ àY†@ À~†@ •~†@ òÂ†@ ÷†@
†@ 7†@ @†@ à-†@ ... usw.
```

Besten Dank!



Sirius3
User



🕒 Sonntag 11. August 2019, 21:19

Wenn Du eine Beschreibung hast, dann sollte darin ja auch irgendwo vorkommen, wie viele Bytes bzw. Doubles dort binär gespeichert sind. Diese Anzahl mußt Du einfach nur in ein bytes-Objekt laden und decodieren. Es könnten ja vielleicht die 9618 sein?

Woher hast Du denn die Fehlermeldung mit den 32 Bytes?

Was hast Du bisher schon geschrieben? Wie sieht die Beschreibung aus? Beispieldaten?

Es ist halt schwierig zu helfen, wenn man nur Fragmente hat.



Sirius3
User



🕒 Sonntag 11. August 2019, 21:40

Die zweite Zahl scheint immer die Anzahl Bytes danach bis zum ; zu sein, damit läßt sich doch schonmal ein Block-Leser bauen:

CODE: ALLES AUSWÄHLEN

```
def iter_blocks(data):
    while data:
        entry = re.search(rb'^\s*\I(\S\S),(\d+),(\d+),' , data)
        if entry is None:
            raise ValueError("Block expected")
        typ, num, length = entry.groups()
        length = int(length)
        end = entry.end()
        block = data[end:end+length]
        if data[end+length] != 59: # ;
            raise ValueError(f"' ;' expected, found {data[end+length]}")
        data = data[end+length+1:]
        yield typ, num, block
```

Für den Block mit der Kennung CS muß man dann aber noch das 1, wegsplitten, was auch immer die 1 bedeuten mag:

CODE: ALLES AUSWÄHLEN

```
for typ, num, block in iter_blocks(data):
    if typ == b'CS':
        _, numbers = block.split(b',', 1)
        numbers = numpy.frombuffer(numbers)
```

So oder so ähnlich.



sparrow
User



🕒 Montag 12. August 2019, 04:31

Wenn ich den Code aus dem ersten Post anschau, sollte die 1 für dem Datentyp uint8 stehen. Das deutet auch darauf hin, dass der sich ändern kann.



Squipy
User



🕒 Sonntag 1. September 2019, 17:55

Moin Zusammen,
besten Dank Sirius! Durch dein Beispielcode habe ich immerhin schonmal etwas lauffähiges
hinbekommen.

Funktioniert auch soweit ganz gut. Das Problem ist leider nur die Performance...

Bei etwas größeren Dateien ist es sogar so, dass mein Rechner völlig stecken bleibt, ich verstehe aber
nicht wieso...

Der Code sieht wie folgt aus:

CODE: ALLES AUSWÄHLEN

```
import re
import numpy

path = r"C:\Users\Sascha\Desktop\python_test\testData.DAT"

def load_channels(path):

    channelunit=dict()
    channeltriggertime=dict()
    channelsamplingrate = dict()
    channelvalue = dict()

    with open(path, "rb") as binary_file:

        # Read the whole file at once
        data = binary_file.read()
        dataBlock = data.split(b";")

        #####Looking for channel names and units#####
        cnInfo = [s for s in dataBlock if b"ICN," in s]
        crInfo = [s for s in dataBlock if b"ICR," in s]
        crInfo = [s for s in crInfo if not b",s" in s]
        ntInfo = [s for s in dataBlock if b"INT," in s]
        cdInfo = [s for s in dataBlock if b"ICD," in s]

        if len(cnInfo) == len(crInfo) == len(ntInfo) == len(cdInfo):

            for i in range(0,len(cnInfo)):
```


Du verwendest meinen Code ja auch gar nicht, sondern hackst wie wild auf den Daten herum. Wenn das funktioniert, dann ist das Zufall.

Warum verwendest Du ``iter_blocks`` nicht, um die Blöcke zu lesen?

Wenn Du eine Schleife über einen Index benutzt, dann machst Du in Python etwas falsch.

Um Geschwindigkeit zu gewinnen, hilft es, die selben Daten nicht immer wieder zu verarbeiten, sondern nur einmal.

Statt vier Wörterbücher zu verwalten, die alle die selben Schlüssel haben, solltest Du ein Wörterbuch benutzen, dessen Werte komplexer sind.



Squipy
User



🕒 Montag 2. September 2019, 07:03

Naja das Problem ist, das mehrere Blöcke ausgelesen werden müssen, die unterschiedliche Länge haben und jenachdem wieviele Kanäle in der Datei vorhanden sind, müssen Blöcke mit dem selben Namen (bspw. |CN) ausgelesen werden. `iterblocks` geht davon aus nur einen bestimmten Block zu decodieren, deswegen habe ich das gemacht. Ansonsten verwende ich ihn eigentlich ziemlich genau so wie du ihn mir bereitgestellt hast, nur habe ich das Gefühl, dass dieser sehr lange rumrödelt sobald die Datei etwas größer wird :/



Antworten ↩



28 Beiträge 1 2 >

< Zurück zu „Allgemeine Fragen“

Gehe zu ▼

🏠 Python-Forum.de < Foren-Übersicht

🗑 Alle Cookies löschen Alle Zeiten sind UTC+01:00

✓ Nutzungsbedingungen

🔒 Datenschutz

📄 Datenschutzerklärung

📄 Impressum

Powered by phpBB™

Deutsche Übersetzung durch phpBB.de