Numpy saved my PhD and Pandas saved my job

...

or how to manipulate and analyse data using Python

What? *Numpy* saved your PhD?

Yeah.

But before that,

I'm Ivan Marin, data scientist at Vivo Data Labs - Telefônica.

ivan.smarin@telefonica.com

@ispmarin

http://mfactor.sdf.org

The problem

Calculate

$$\Omega_{uf} = -Q_0 z e^{-i\beta_{uf}} + C$$

For each point on this

$$a_n = \frac{1}{\pi}\int_0^{\pi} \lambda(\theta)\cos(n\theta)d\theta$$

In an iterative way, several times.

The *real* problem

That one was just one element. There were 10 different elements:

$$\Omega_n(\chi) = a_n \frac{1}{2\pi i} F_n(\chi) = a_n \frac{1}{4\pi i} \left[\chi^n + \left(\frac{1}{\chi}\right)^n \right] \ln \frac{Z-1}{Z+1} + a_n \frac{p_n(\chi)}{2\pi i}$$

$$\Omega_n(\chi) = a_n \frac{1}{2\pi i} F_n(\chi) = a_n \frac{1}{4\pi i} \left[\chi^n + \left(\frac{1}{\chi}\right)^n \right] \ln \frac{Z-1}{Z+1} + a_n \frac{p_n(\chi)}{2\pi i}$$

$$\left[\chi^n + \left(\frac{1}{\chi}\right)^n \right] \ln \frac{\chi-1}{\chi+1} \cdot 2\beta_n + 2\sum_{j=1}^{n-1} \beta_{n-j} \left[\chi^j + \left(\frac{1}{\chi}\right)^j \right]$$

One huge matrix:

$$\begin{matrix}
F_{1,0}(\theta_1)\cos(0) - \frac{1}{P_{1,0}} & \int_0^\pi F_{1,1}(\theta_1)\cos(0) & \int_0^\pi F_{1,2}(\theta_1)\cos(0) & \int_0^\pi F_{2,0}(\theta_1)\cos(0) & \int_0^\pi F_{2,1}(\theta_1)\cos(0) & \int_0^\pi F_{2,2}(\theta_1)\cos(0) & \int_0^\pi C\cos(0) \\
\int_0^\pi F_{1,0}(\theta_1)\cos(\theta_1) & \int_0^\pi F_{1,1}(\theta_1)\cos(\theta_1) - \frac{1}{P_{1,1}} & \int_0^\pi F_{1,2}(\theta_1)\cos(\theta_1) & \int_0^\pi F_{2,0}(\theta_1)\cos(\theta_1) & \int_0^\pi F_{2,1}(\theta_1)\cos(\theta_1) & \int_0^\pi F_{2,2}(\theta_1)\cos(\theta_1) & \int_0^\pi C\cos(\theta_1) \\
\int_0^\pi F_{1,0}(\theta_1)\cos(2\theta_1) & \int_0^\pi F_{1,1}(\theta_1)\cos(2\theta_1) & \int_0^\pi F_{1,2}(\theta_1)\cos(2\theta_1) - \frac{1}{P_{1,2}} & \int_0^\pi F_{2,0}(\theta_1)\cos(2\theta_1) & \int_0^\pi F_{2,1}(\theta_1)\cos(2\theta_1) & \int_0^\pi F_{2,2}(\theta_1)\cos(2\theta_1) & \int_0^\pi C\cos(2\theta_1) \\
\int_0^\pi F_{1,0}(\theta_2)\cos(0) & \int_0^\pi F_{1,1}(\theta_2)\cos(0) & \int_0^\pi F_{1,2}(\theta_2)\cos(0) & \int_0^\pi F_{2,0}(\theta_2)\cos(0) - \frac{1}{P_{2,0}} & \int_0^\pi F_{2,1}(\theta_2)\cos(0) & \int_0^\pi F_{2,2}(\theta_2)\cos(0) & \int_0^\pi C\cos(0) \\
\int_0^\pi F_{1,0}(\theta_2)\cos(\theta_2) & \int_0^\pi F_{1,1}(\theta_2)\cos(\theta_2) & \int_0^\pi F_{1,2}(\theta_2)\cos(\theta_2) & \int_0^\pi F_{2,0}(\theta_2)\cos(\theta_2) & \int_0^\pi F_{2,1}(\theta_2)\cos(\theta_2) - \frac{1}{P_{2,1}} & \int_0^\pi F_{2,2}(\theta_2)\cos(\theta_2) & \int_0^\pi C\cos(\theta_2) \\
\int_0^\pi F_{1,0}(\theta_2)\cos(2\theta_2) & \int_0^\pi F_{1,1}(\theta_2)\cos(2\theta_2) & \int_0^\pi F_{1,2}(\theta_2)\cos(2\theta_2) & \int_0^\pi F_{2,0}(\theta_2)\cos(2\theta_2) & \int_0^\pi F_{2,1}(\theta_2)\cos(2\theta_2) & \int_0^\pi F_{2,2}(\theta_2)\cos(2\theta_2) - \frac{1}{P_{2,1}} & \int_0^\pi C\cos(2\theta_2) \\
F_{1,0}(\chi_{ref}) & F_{1,1}(\chi_{ref}) & F_{1,2}(\chi_{ref}) & F_{2,0}(\chi_{ref}) & F_{2,1}(\chi_{ref}) & F_{2,2}(\chi_{ref}) & 1
\end{matrix}
\begin{pmatrix} a_{1,0} \\ a_{1,1} \\ a_{1,2} \\ a_{2,0} \\ a_{2,1} \\ a_{2,2} \\ C \end{pmatrix}
=
\begin{pmatrix}
\int_0^\pi \Re\left(Q_0 z e^{-i\gamma}\right)\cos(0)d\theta_1 \\
\int_0^\pi \Re\left(Q_0 z e^{-i\gamma}\right)\cos(\theta_1)d\theta_1 \\
\int_0^\pi \Re\left(Q_0 z e^{-i\gamma}\right)\cos(2\theta_1)d\theta_1 \\
\int_0^\pi \Re\left(Q_0 z e^{-i\gamma}\right)\cos(0)d\theta_2 \\
\int_0^\pi \Re\left(Q_0 z e^{-i\gamma}\right)\cos(\theta_2)d\theta_2 \\
\int_0^\pi \Re\left(Q_0 z e^{-i\gamma}\right)\cos(2\theta_2)d\theta_2 \\
\Phi_0
\end{pmatrix}$$

iterated and over.

A language issue

The code was initially written in C++:

- Performance
- Linear algebra libraries
- Numerical solvers
- Integral solvers

And it was a pain in the behind to work with:

- Data structures
- Compiling code with tons of libraries
- Bugs

Python to the rescue

With Python it was way easier to

- implement all elements with lots of code reuse
- Implement the numerical engine with very few classes
- Code readability was way better
- It was also easier to communicate with other researchers

But you all know about that. What about *performance*?

Numpy to Python's (and mine) rescue

```
import numpy as np

a = np.array(range(0,10))

b = np.array(range(10,20))
```

Numpy provides a multidimensional array object and an assortment of routines for fast operations on arrays, including linear algebra and tons of other stuff.

Numpy to Python's (and mine) rescue

Numpy supports

- Large (multi-dimensional) arrays
- Large matrices
- High-level mathematical functions
- Linear algebra operations
- Interface with C/C++/Fortran code
- Fourier transforms

Pretty much the building blocks for numerical computing.

But *fast.*

Example time!

Numpy is best used in a *vectorized* way.

Scalar product in naive Python:

```
c = []
for i in range(len(a)):
    c.append(a[i]*b[i])
```

Example time! C version

Vector multiplication C version:

```
for (i = 0; i < rows; i++): {
  c[i] = a[i]*b[i];
}
```

Fast.

Example time! 2D C version

2D vector multiplication C version:

```
for (i = 0; i < rows; i++): {
  for (j = 0; j < columns; j++): {
    c[i][j] = a[i][j]*b[i][j];
  }
}
```

Fast and ugly.

Example time! Numpy version

```
c = a * b
```

No, really. That's it. For both 1D and 2D.

C speed with Python simplicity!

But how?

Two magic words: *vectorization* and *broadcasting*.

Vectorization describes code without explicit loops and indexing. The magic is done behind the stage, with pre-compiled C code.

Vectorized code has several benefits:

- Way less code
- Less code == less bugs
- More similar to mathematical notation

But how?

Broadcasting describes the implicit element by element operations that Numpy does.

So even if a and b are multidimensional arrays with different shapes, if they follow the broadcast rules (similar to matrix multiplication rules), things will just work.

Pandas all the way

Numpy is awesome for numerical computation.

But what about data besides numerical arrays?

The problem

I had to parse around 20 Excel spreadsheets with information.

For each one of them, I had to get different fields, from different sheets, with different formats:

- dates
- numbers
- names

... you name it.

The real problem

The company was already managing the spreadsheets... with more spreadsheets.

Excel external references:

```
=SUM([Budget.xlsx]Annual!C10:C25)
```

Spreadsheets all the way down.

Pandas!

```
import pandas as pd

df = pd.read_excel(horrible_excel_file, sheetname=’shte’, skiprows=120, parse_cols=’C:F1’)

df.describe()
```

And done.

Pandas

Pandas is a library for data analysis.

The idea is to provide data structures for fast and flexible work on labeled or relational data. It can work with

- Tabular data (SQL, Excel)
- Ordered or unordered Time Series data
- Matrix data with row and columns information
- Observational or statistical datasets

Pandas

The main data structures are

- Series: 1D
- DataFrame: 2D (or higher dimensions)

It handles

- Missing data (NaN)
- Group by
- Joins, merges, slicing, reshaping
- Time series indexes with datetime
- Aggregation and statistical operations (mean, standard deviation, count, unique)

Example time!