Pythonを用いた科学技術計算

麻生 洋一

Pythonとは?

- 汎用のインタープリタ言語
  - オランダ人Guido van Rossumによって1989年開発開始
  - Perl, Ruby等と似たような立ち位置
  - ちょっとしたスクリプトから、Webアプリ、本格的なGUIアプリケーション まで、幅広く使われている
  - 名前はMonty Pythonに由来

- モジュールによる機能拡張を基本とする
  - コア言語仕様は非常にシンプルで学びやすい
  - 強力な標準ライブラリ+膨大なThird Partyライブラリ
  - CやC++による拡張機能も書きやすい

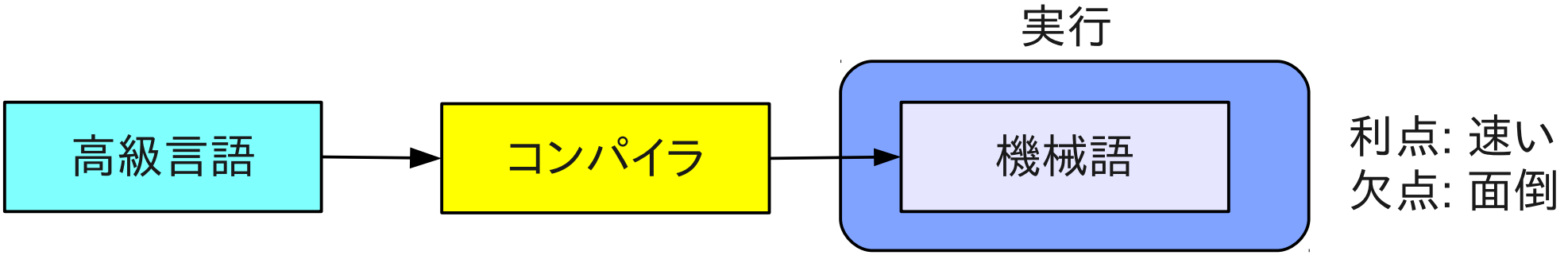
- オブジェクト指向
  - Multi-Paradigm: 手続き型、関数型、オブジェクト指向 etc
  - オブジェクト指向がMain Paradigm
  - Pythonでは全てがオブジェクト

- オープンソース
  - 幅広いプラットフォームで利用可能
    Linux, Windows, Mac, FreeBSD, Solaris, iPhone etc ...
  - 無料で利用可能
    - 自分が書いたスクリプトを他人に渡す際、相手に高い金を払わせる必要が無い
  - Matlabとは比較にならないユーザーベース

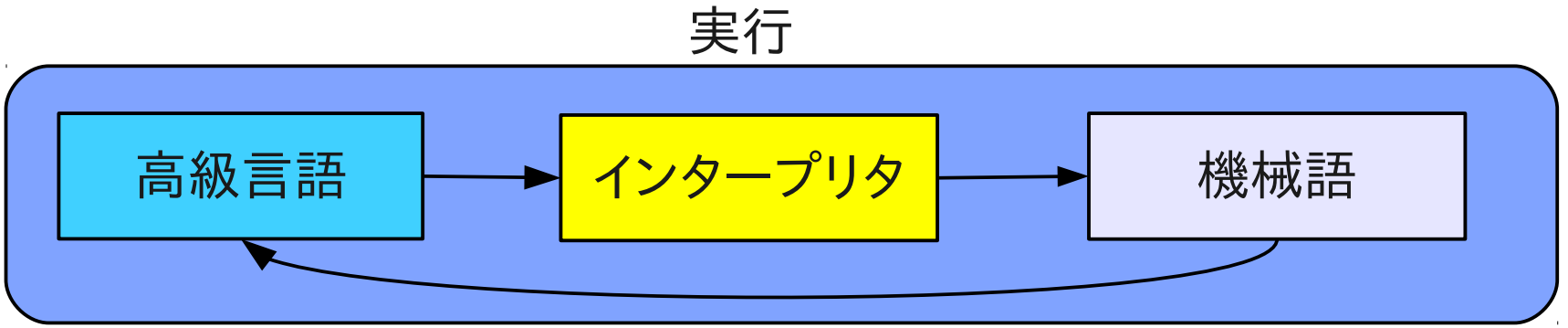
インタープリタ言語

コンパイラー言語: C/C++, Fortran, Pascal, etc

まとめて翻訳



インタープリタ言語: Python, Perl, Lisp, etc



利点:

- 簡単、使い易い
- 動的型付
- 自動メモリ管理(Garbage Collection)

欠点:

- 遅い
- 非効率的

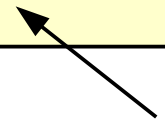
MATLAB型言語

科学計算用にインタープリタ言語の欠点(遅さを)補う

インタープリタ言語 + 高速な多次元配列型

配列の演算

```
a = np.array([1,2,3,4])
b = 2*a
```



ここは内部的にCで書かれた関数が計算している

愚直な(遅い)実装

```
a = np.array([1,2,3,4])
b = np.empty(4)
for ii = range(4):
   b[ii] = 2*a[ii]
```

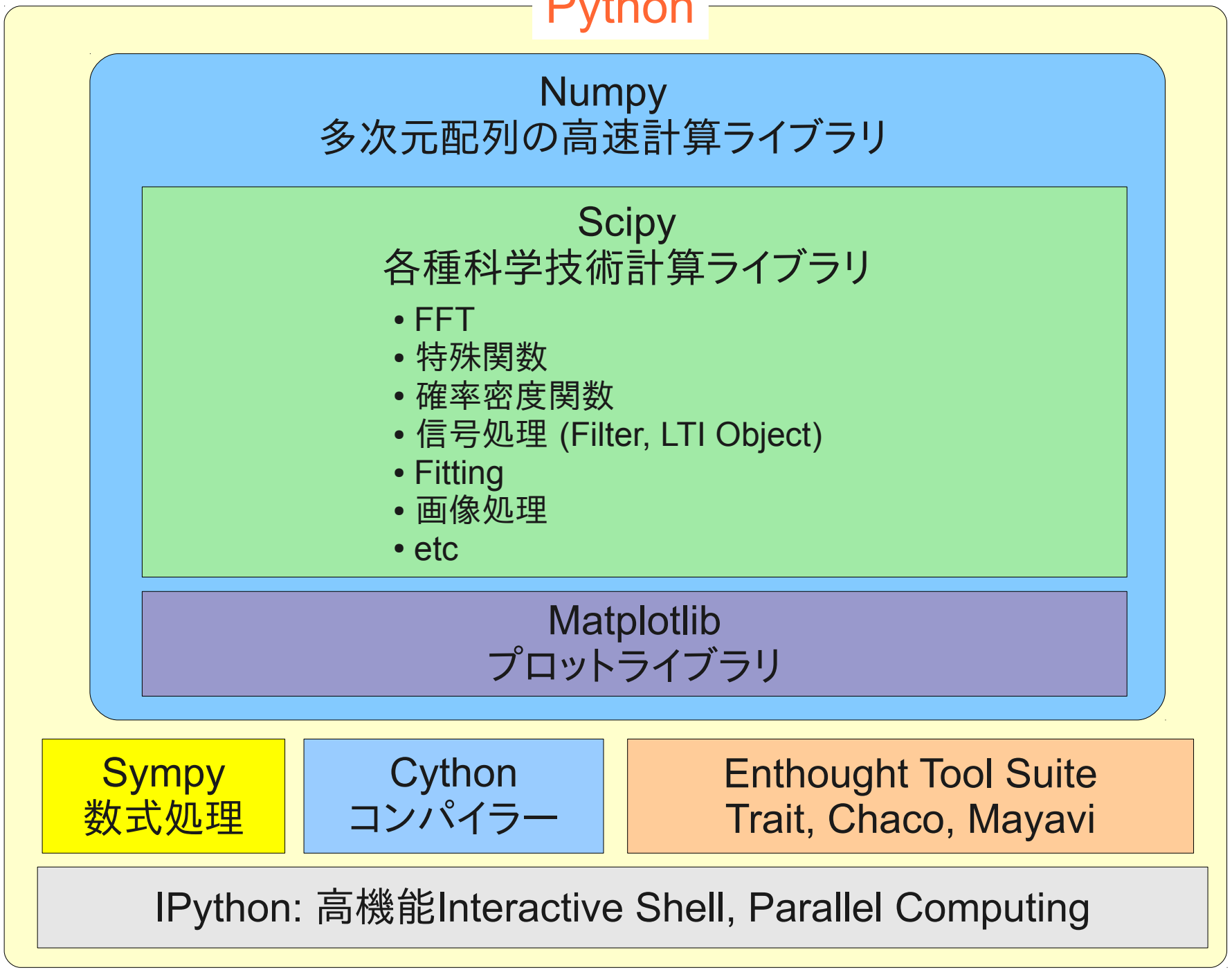
多次元配列型に対する高速な: 算術演算、線形代数演算、数学関数、etc

Pythonにおける多次元配列型

Numpy

- Python用の多次元配列モジュール
- Pythonで数値計算する際のDe facto Standard
- 計算コードはCで実装(速い)
- Matlabの基本機能に相当する部分を提供
  - 多次元配列生成、操作
  - 線形代数演算
  - 数学関数
  - 統計関数
  - 多項式
  - etc ...
- Numpyをベースにして各種科学計算モジュールが開発されている

Pythonの科学技術計算ライブラリ概要



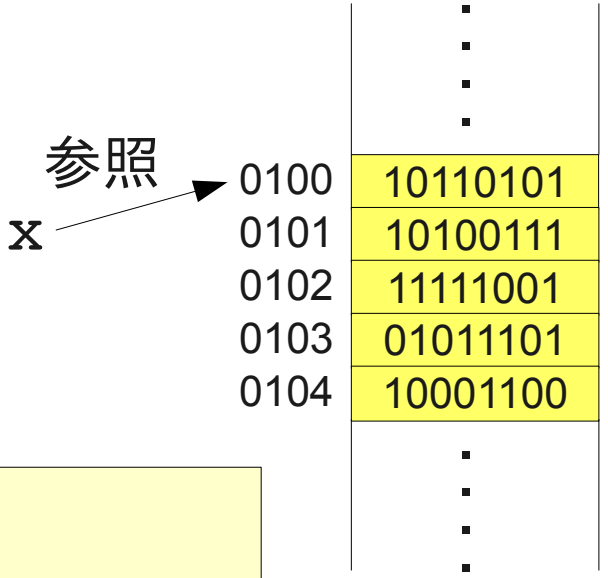
何故MatlabはUncoolか?

Matlabは使い易く、優れた科学技術計算プラットフォームである。
しかし、、

- 高い
  - 毎年お布施を払わなければならない。
  - Matlabを使っていない人とのコラボレーションが困難
- 言語仕様がダサい
  - 歴史のあるソフトなので、過去からのしがらみを背負っている
    - 関数を一々別ファイルにしなければならない
      → 逐次実行型的なプログラムを書いてしまう
    - オブジェクト指向のサポートが不完全かついびつ
    - 引数の参照渡しができない(実は不可能では無いが極めて面倒)
      → 大きなデータを扱うアプリケーションでは問題となる
    - 名前付き引数が使えない
- 拡張性が低い
  - コア部分は非公開
  - C/FortranとのインターフェースAPIはあるが、使いにくい
  - 科学計算以外のライブラリは極めて貧弱
- その他
  - Matlabコンパイラーはコンパイラーでは無い

引数の参照渡し

- オブジェクトはメモリーに格納されている
- Pythonに於いて変数はオブジェクトのあるメモリを参照しているだけ



例

```
a=1  #メモリー上に存在する1という値を持ったintオブジェクト
     #をaという名前で参照する
some_function(a)   #some_function()にはaの値(1)が渡される
                   #のか、それともaが参照するオブジェクトへの参照
                   #が渡されるのか?
```

Pythonでは常に参照が渡される
Matlabは基本的に値渡し

参照渡し: メモリー番地(ポインター)が渡されるだけなので、処理が軽い

値渡し: some_function()用に、aのコピーが作られる
もしaが重力波データのような巨大な配列だった場合、コピーに時間がかかり、メモリーも無駄に使う → Matlabの重大な欠点

(正確には、Matlabはpass-by-value-with-lazy-copy)

referenceSample.py

参照渡しの例

配列の一部を変更する関数

```
a=np.array([0,1,2,3])

def test(x):
    x[0] = 10

test(a)   # a=[10,1,2,3]
```

関数呼び出し前後でaの中身が変化している

メモリーのコピーが発生しない

Matlabの場合

```
a = [0,1,2,3];

function x=test(x)
    x(1) = 10;

b = test(a)
```

ここでメモリーコピーが発生

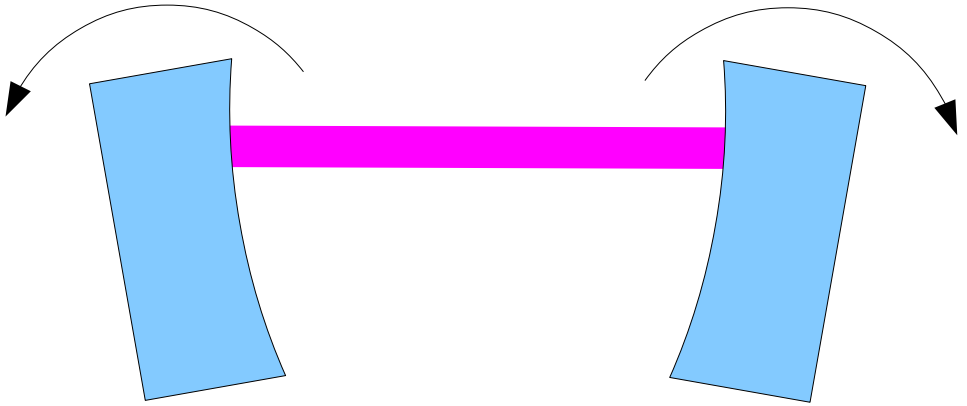
配列の一部を操作したいだけなのに配列全体をコピー

→ 大きな配列では遅い

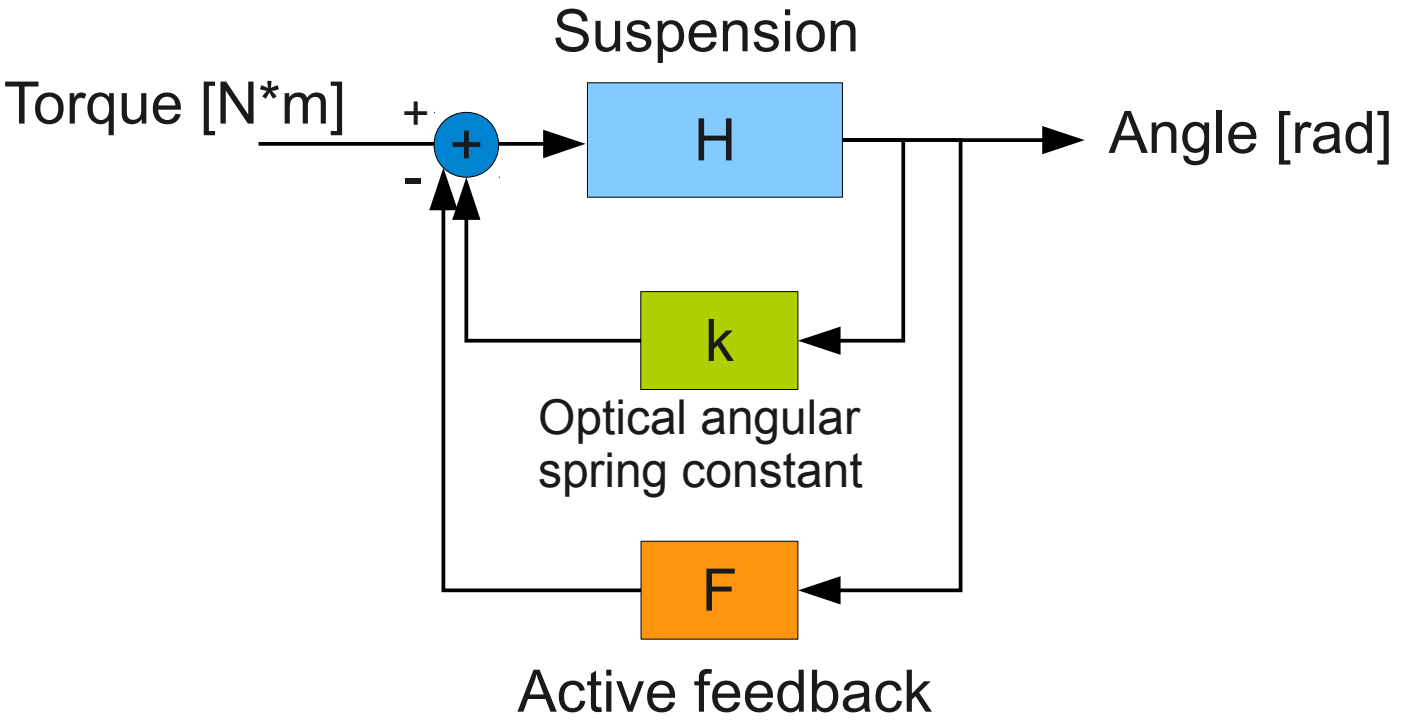
実例

- 基本
- プロット
- 信号処理
- Sidles-Sigg
- OFDM
- 機器制御
  - スペアナ、DAQ、Picomotor,ctypes
- gtrace
- 速度

Sidles-Sigg Instability



輻射圧による
回転方向不安定性



LTI(Linear Time Invariant) Systemの有理式表現

$$H(s)=\frac{b_0 s^n+b_1 s^{n-1}+\ldots+b_{n-1}s+b_n}{a_0 s^n+a_1 s^{n-1}+\ldots+a_{n-1}s+a_n}$$

サスペンション伝達関数

$$H_{\mathrm{susp}}(s)=\frac{1}{m s^2+(m\omega_0/Q)s+m\omega_0^2}$$

⟶ b=[1], a=[m, m*w0/Q, m*w0**2]

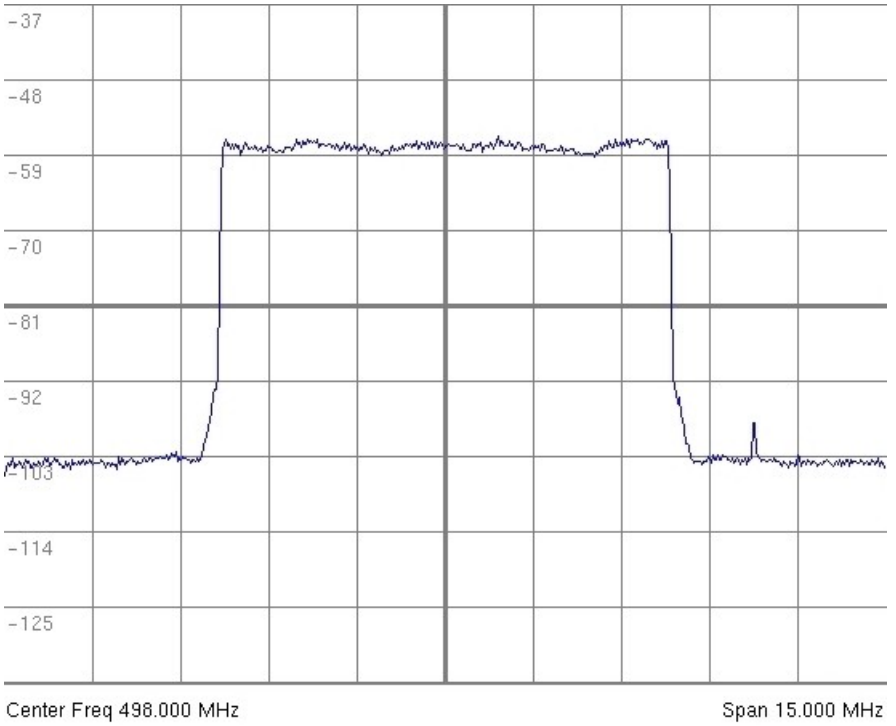
被制御対象に不安定ポールがある場合の安定条件

閉ループが安定であるためには$N=-n_q$、すなわちナイキスト線図の軌跡は点$(-1, j0)$の回りを、右半平面内にある開ループ伝達関数のポールの個数と同じ回数反時計回りに回転しなければならない。

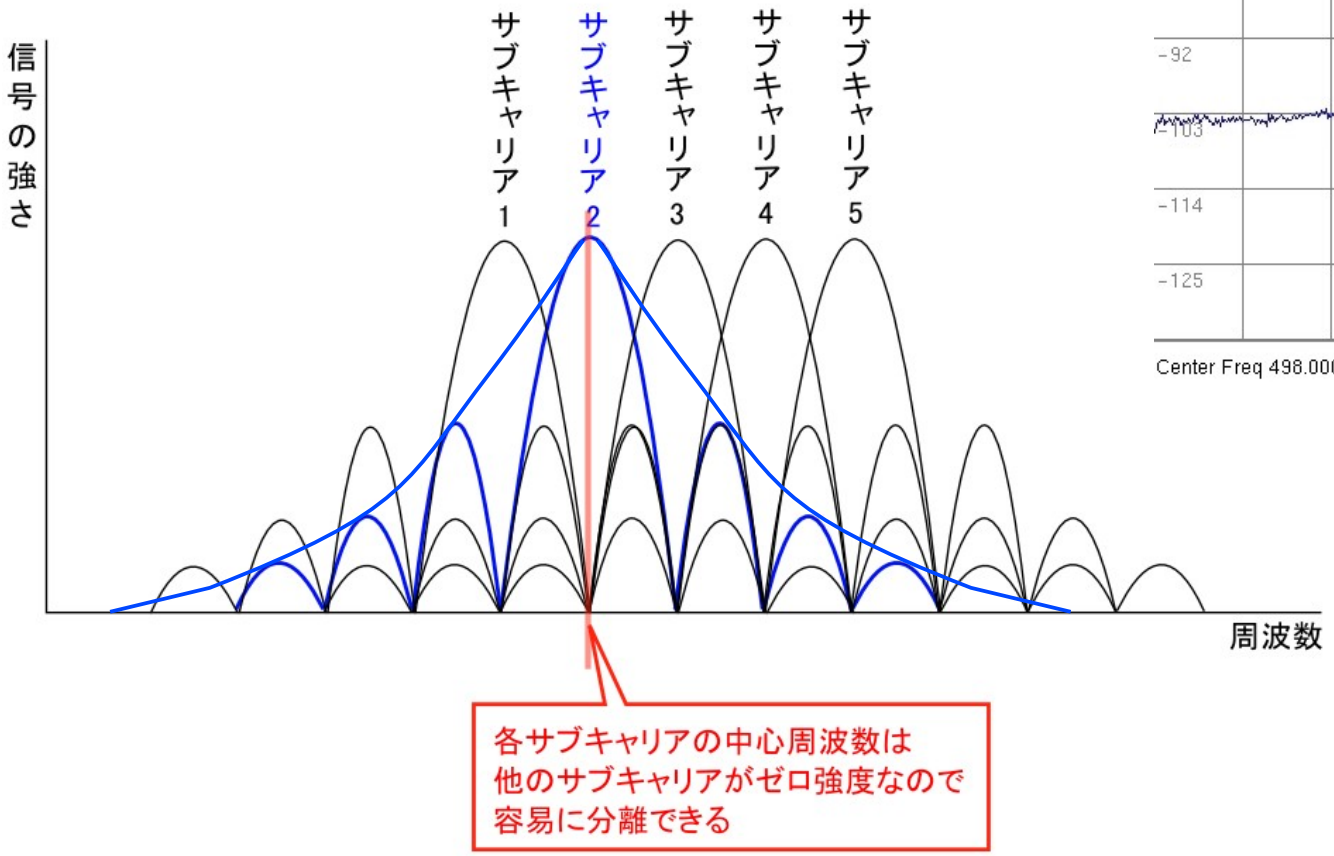
OFDM(Orthogonal Frequency Division Multiplex)
直交周波数分割多重

- 地デジ、無線LAN等の変調方式
- 周波数帯域の有効活用

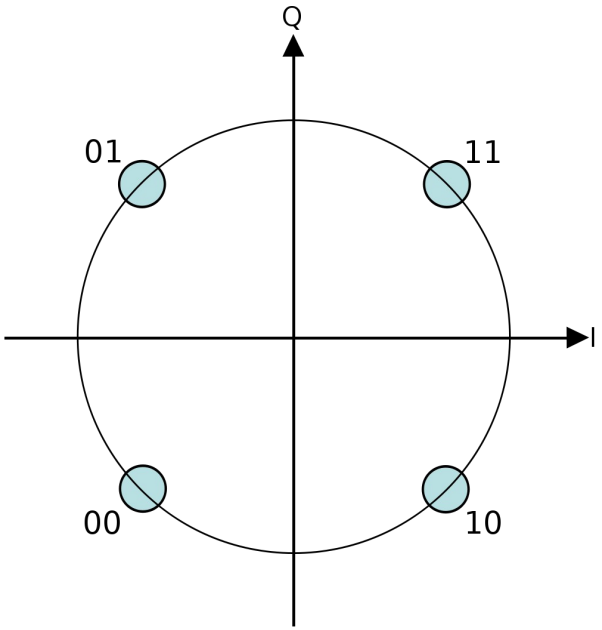
実際のOFDM変調スペクトル



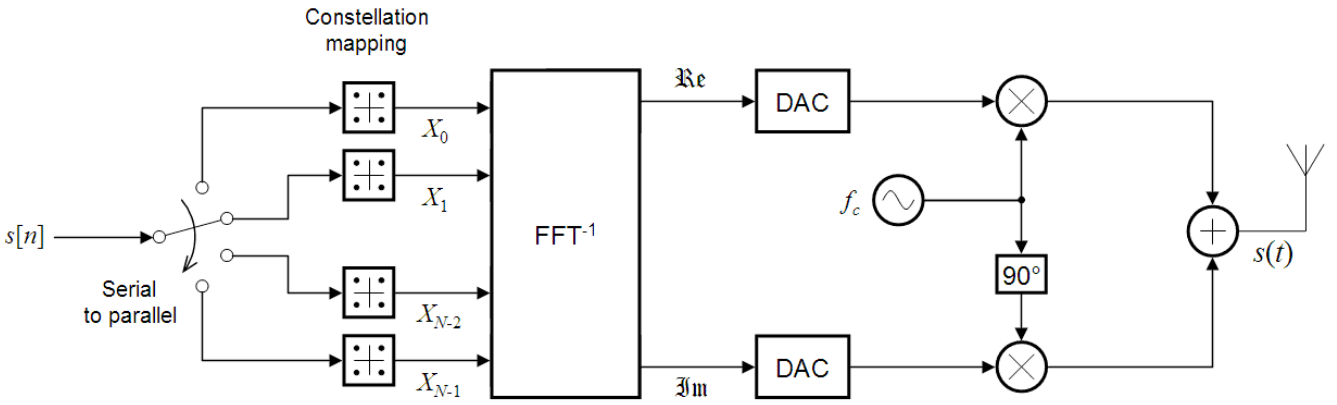
OFDMの周波数



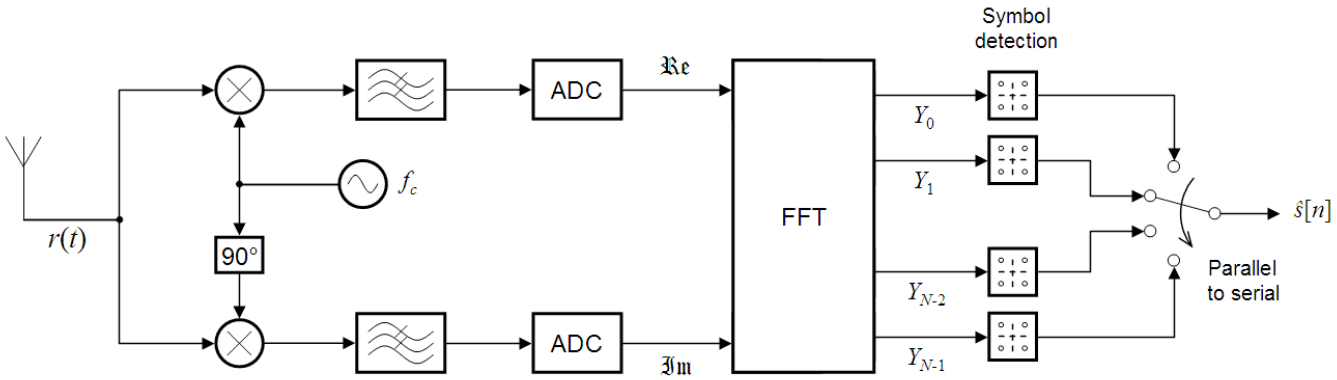
Phase Shift Keying (PSK)



Transmitter



Receiver



Pythonによる機器制御

TCP/IPは標準装備

- TCP/IP <-> GPIB変換器: GPIB機器を制御可能
- 直接GPIBコントローラを駆動するライブラリーも存在する

- PicoMotor
  - Model 8750 Intelligent Picomotor Network Controller
  - Network経由で制御可能(神岡で稼働中)

ドライバーとCのライブラリが提供される機器 (ADC/DAC etc)

- Cのライブラリーがあれば、Ctypes経由で利用可能
  - NI-DAQmx
  - MCC DAQ hardwares

Ctypes: PythonからCの関数を呼び出すモジュール
- 全てをCで書くと大変
- 機器とのやり取りだけ、Cの関数を呼び出す

USBやSerialポートを直接制御することも可能(pyusb, pySerial)

gtrace

gtrace is a 2D optical ray tracing library, which can track the evolution of Gaussian beam propagation.

Features

- Calculate the reflection and deflection of a beam by an optics.
- Track the Gaussian beam evolution (beam size, ROC, and Gouy phase) as a beam propagates.
- Optical path length is correctly calculated when transmitting optical components.
- Astigmatism is included on non-normal incidences to optics.
- DXF files can be generated.
- Fully object-oriented.

Sympy: 数式処理

- Pythonで実装された数式処理ライブラリ
- 代数、微分、積分、極値、多項式展開、微分方程式、etc ...
- ちょっとした記号計算に便利

Pythonの数値計算パフォーマンス

Matlabと比べて速いか,遅いか?

→ 同等、又は速い

Matlab型言語のパフォーマンスは何が決めているか?

- 中で呼び出しているBLAS/LAPACK等の数値計算ライブラリー
- 呼び出しに関わるオーバーヘッド

BLAS/LAPCKにはいろんな種類がある

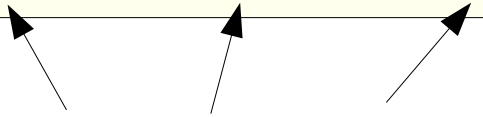
- REFBLAS: リファレンス実装，遅い
- ATLAS: コンパイル時に最適化, かなり速い
- Intel MKL: Intel CPU向け最適化, 速い
- AMD ACML: AMD CPU向け最適化, 速い
- GotoBLAS: 後藤さんが職人芸で最適化, 超速い

最近のMatlabはMKL or ACML
numpyは上記から選べる

numexpr

numpyの構文をさらに高速に実行

```
d = a**2 + 3*b + sin(c)
```



各演算を行なってから和を取る
内部的にforループが4回 or 5回

本来ならば以下のようにして欲しい

```
for (i=0; i < n; i++)
{
    d[i] = a[i]*a[i] + 3*b[i] + sin(c[i]);
}
```

numexprは上記をやってくれる

さらに: 並列実行, キャッシュ最適化, etc ...

計算速度向上

Cコードのインライン化

- weave
- 計算時間のかかる部分をCで記述
- 自動でコンパイル、リンク

Pythonコードのコンパイル

- Cython
- PythonコードをCに変換してコンパイル
- 静的型宣言を追加することで、最適化された高速コードを生成可能
- PythonからCのライブラリを利用可能

Python環境のインストール

Windows

科学技術ライブラリが一括インストールできるパッケージ

- EPD(Enthought Python Distribution): 商用だがアカデミック利用は無料
- Python(x,y): 完全無料

MacOSX

- Pythonはデフォルトでインストールされている
- Numpyも最新版では含まれている?
- EPDはMacOSX版がある

Linux

パッケージマネージャからインストール可能

IDE(統合開発環境)

- Emacs
- Spyder(Matlab like)

Web Resources

Pythonの入門ドキュメントはWeb上に大量にころがってる
日本語のドキュメントもある

科学技術関連ライブラリの情報はほとんど英語

http://www.scipy.org/

https://granite.phys.s.u-tokyo.ac.jp/wiki/Lab/index.php?PythonForScience#cd5a378a

結論

- Pythonを科学技術計算に利用する流れが急速に広まっている
- Matlabの置き換えとして充分利用可能
- プロット品質はMatlabより上
- 拡張性もMatlabとは比較にならないほど高い
- C/C++との親和性が高く、高速化が可能
- 広いユーザーベースで、様々なプロジェクトが進行中
- 企業によるサポートもある
- Matlabに高い金を払う理由は薄い

それでもMatlabを使わなければならない場面はある

- Simulink
  Open Source代替 Scicos (Scilabの一部)
- AdvLIGO Digital SystemのRCG
- Optickle

Numpyの実例

モジュールのインポート

(参照: numpyIntro.py)

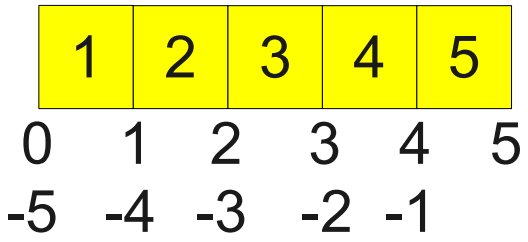
```
import numpy as np
```

arrayオブジェクトの生成

```
a=np.array([1,2,3,4])
```

- PythonにおけるSlicing (Indexing)

```
a=[1,2,3,4,5]
```




```
>>>a[0:2]
[1,2]
>>>a[3:]
[4,5]
>>>a[-4:-1]
[2,3,4]
```

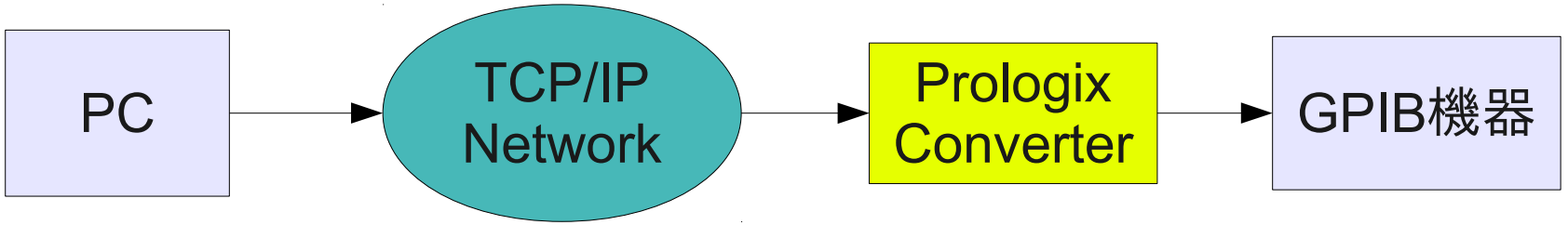
Pythonに於けるIndexは、要素の境目を指していると考える。

```
>>>a[2]
3
>>>a[-1]
5
```

単独Indexは、境目の右側にある要素を指す

GPIB機器の制御

Prologix GPIB-Ethernet Converter



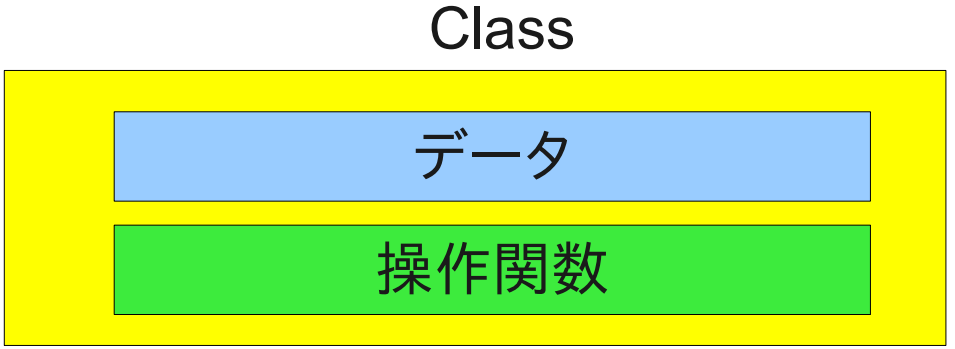
- TCP/IPでGPIBコマンドを送信
- PythonのTCP/IP Socketライブラリを用いて通信
- Prologixのコンバーターを表すクラス(netGPIB)を実装
- GPIB機器を用いた自動測定、データダウンロード

オブジェクト指向

コンピュータで行うこと

Object: データ構造とその操作関数が一体化したもの

Class: Objectの定義



例: 銀行口座クラス

データ: 口座番号(Int), 残高(Int)
操作関数: 初期化(口座開設), 入金, 出金
演算子: 二つの口座の合算(+)

Instance: Classの定義に従って実体化したObjectの事

クラスの例: Fds

Fds: 2次元データを保持するクラス
　　時系列・スペクトル etc

データメンバ: x　(サイズ = 1 × n)
　　y　(サイズ = m × n)

長さの異なるFdsオブジェクト間の演算が定義されている

例) 周波数スペクトルに伝達関数を掛ける
- 普通周波数スペクトルの方がサンプル点数が多い
- 掛け算をするには、伝達関数を補間する必要がある
- Fdsではこの補間を自動的に行う

実験データの処理に便利

(参照: fdsExamples.py)