確認テストの解答

｛其の一｝

【問題】Fijiではjarsフォルダ内のij-1.48v.jar（1.48vの部分はバージョンアップなどで変わる），そして，素のImageJではImageJフォルダ内のij.jarが，それぞれImageJの中核部分である．このファイルを演習②と同様に別のフォルダにコピーし，zip形式として扱えるようリネームのうえ，展開して中身を確認しよう．内蔵プラグインはどこにどのように収められているだろうか？「ImageJの大半の機能がプラグインによって実現している」という主張は定量的に示すことができるだろうか？（ヒント；ij.jar内にてImageJのメニュー構成の基盤を記述しているIJ_Props.txtファイルが参考になる）

【答え】

素のImageJに最初から収められ，ImageJの機能の大半を実装している内蔵プラグイン群はij.jarの中のij/pluginフォルダ内と，さらにそのサブフォルダであるij/plugin/filterフォルダ，ij/plugin/frameフォルダ，ij/plugin/toolフォルダに配置されている．その数はFijiでも素のImageJでも同じバージョンをもとにしている場合には変わらず，筆者の環境では合計242ファイル，1.65MBであった．ImageJ全体は400ファイル，3.37MBのclassファイルであるから，ファイル数換算で61%，classファイル容量換算で49%がプラグインの占める割合である．なおソースファイルの行数から見積もると全体の46%（57671行/125077行）がプラグインである．

｛ 其の二 ｝

【問題】演習③では複数の画像処理を組み合わせて，最終的な結果を得た．1つの画像スタックを対象に解析するのであれば，1ステップずつマウスとキーボードを使って実行してもよい．しかし現実には薬剤処理・変異体・ストレス条件など多くの実験区画があり，これらに対して同一の画像解析手法をミスなく適用せねばならない．そのようなシチュエーションでは労力的にも信頼性の点でも，ユーザーが1つ1つ手動で画像処理のステップを踏むよりも，一連の操作をマクロによって言語化し自動化することが必要となる．さらにこれは，解析終了後，ノイズ抑制の度合いを変えるなど画像処理工程のパラメータを見直して，再解析することも容易にする．そこで，演習③で用いた画像ファイルについて，最初に各画素で時間軸方向へのガウスぼかし（σ=2）を施し，その後にKbi Flowを用いた流動解析を行うマクロを作成しよう．さらにガウスぼかしのσを5に上げると，速度測定の結果はどうなるだろうか（ヒント：手動での操作をもとにマクロを作る場合にはユーザーの操作を記録する[Plugins→Macros→Record]が役立つ）．

【答え】

```
1  run("Kbi Filter1d", "filter=gauss gausssigma=2 axis=z overwrite");
2  run("Kbi Flow", "mode=measure widthxy=16 stepxy=8 widtht=-1 stept=1
   subtavgt sensitivity=0.10 trackfactor=-1 mintrackdt=-1");
3  run("Kbi Flow", "datatitle=ER-flow.tif mode=redisplay dispvect vectscale=-5
   vectmowing=0 vecttype=arrow_speed dispspeed mapzoom=2 dispplots");
```

時間軸方向のガウスぼかしのσを5にするには1行目のgausssigma=2をgausssigma=5とする．流動速度は最大値や平均値などいずれも「時間軸方向のガウスぼかしをかけない場合」＞「σが2の場合」＞「σが5の場合」の関係になる．つまり時間軸方向にスムーシングを施すと（ノイズ抑制で実施される場合がある），速度が低く見積られるバイアスが働くということである．