Mathematical Modelling with Python

Thien V. Hoang

Ngày 4 tháng 7 năm 2017

Tóm tắt nội dung

Python là ngôn ngữ lập trình bậc cao mã nguồn mở có tính phổ biến cực lớn trong giới công nghệ. Trong trại hè Toán học và Ứng dụng PiMA 2017, chúng ta sẽ tìm hiểu về tính ứng dụng của Python trong nghiên cứu các mô hình toán học từ thực tế. Bài viết giới thiệu Python, các thuật ngữ lập trình, một số pacakge hữu ích, và cách sử dụng Jupyter để trình bày một bản báo cáo và phân tích. Các bạn chú ý các thuật ngữ tiếng Anh trong ngoặc đơn có thể được sử dụng nhiều lần trên bài viết. Ngoài ra bài viết cung cấp các thông tin chuyên sâu dành cho những bạn có thích thú tìm hiểu. Tác giả bài viết không sử dụng hệ điều hành Windows, tuy nhiên cách thực hiện sẽ không quá khác biệt. Các mục được đánh dấu sao (*) trong bài là những mục nâng cao, chuyên sâu, phù hợp hơn với những bạn có background về tin học do đó không yêu cầu các bạn nhập môn lập trình tìm hiểu.

1 Giới thiệu về lập trình

Lập trình có thể được hiểu là "hướng dẫn" cho máy tính của bạn làm theo ý bạn thông qua những câu lệnh. Lập trình có thể được dùng cho nhiều mục đích khác nhau, từ việc tạo ra những ứng dụng và trò chơi trên điện thoại cho đến hệ thống xe tự lái, xa hơn nữa là công cụ tính toán và phân tích trong các trung tâm nghiên cứu về vũ trụ như NASA. Tuy vậy, ở một mức độ cơ bản có thể hiểu lập trình là giải quyết những bài toán (problem solving). Bài toán lớn hơn lại có thể được thiết kế thành những bài toán nhỏ và sau đó ghép các kết quả lại với nhau, giống như một **kỹ sư**. Bài toán cũng có thể được mô hình theo các công thức giống như các **nhà toán học** thường làm khi nghiên cứu. Và công việc lập trình cũng đòi hỏi tư duy lập ra những giả thuyết, dự đoán, khả năng kiểm tra lời giải, và quan sát tính chất phức tạp của hệ thống máy tính. Điều này khiến bạn trở thành một **nhà khoa học máy tính**.

Máy tính chỉ hiểu thông tin ở dạng mã nhị phân. Con người không thể hiểu được. Người ta bắt đầu cho ra những ngôn ngữ bậc thấp đầu tiên như Assembly ở dạng chữ, dễ đọc hơn, rồi từ đó biên dịch ra ngôn ngữ máy. Nhưng sau này nhận thấy ngôn ngữ bậc thấp còn quá phức tạp, khó tiếp cận, khiến cho công nghiệp máy tính phát triển chậm chạp, người ta nghĩ ra ngôn ngữ lập trình bậc cao với cú pháp và từ khóa thân thuộc với con người. Python là một trong số đó.

Lập trình là một công việc rất đơn giản, càng nghiên cứu thì càng thấy sự khó khăn phức tạp. Trong quá trình code các bạn rất dễ gặp lỗi nếu không kiểm soát tốt. Có ba loại lỗi là lỗi cú pháp (syntax), lỗi phát sinh trong thời gian chạy (runtime error), và lỗi trong thuật toán khiến bài toán cho kết quả không mong muốn (semantic error). Nếu so sánh với toán học:

- Syntax error: Là những lỗi về cú pháp và cách trình bày như viết sai ký hiệu toán học khiến lời giải trở nên vô nghĩa, không có giá trị. Ví dụ như ký hiệu

tính tổng sigma ($\sum$) mà bạn xoay 90 độ viết thành chữ M thì chẳng ai hiểu đó là gì.

- Runtime error: Là những lỗi phát sinh trong quá trình bạn giải toán mà bạn quên kiểm soát như vi phạm vào điều kiện xác định làm cho lời giải không còn giá trị. Ví dụ khi giải phương trình, trong một bước nào đó bạn chia hai vế cho một biểu thức mà không nhận ra rằng biểu thức đó sẽ có giá trị bằng 0 tại một số trường hợp. Lúc này lời giải của bạn sẽ có lúc cho ra đáp án, có lúc *không thể* đưa ra đáp án vì sự vi phạm này.
- Semantic error: Lỗi này là nguy hiểm nhất và khó sửa nhất. Bạn đã ràng buộc rất tốt ở hai lỗi trên, tức là bạn không sáng tạo ra ký hiệu toán học mới mà cũng đã xét đầy đủ điều kiện xác định. Nhưng cuối cùng lời giải của bạn cho ra kết quả sai. Và bạn phát hiện ra bạn đã vô tình viết ký hiệu ∞ thành số 8. Mọi thứ trong lời giải đều hoạt động rất trơn tru, không có lỗi buộc phải dừng, nhưng kết quả lại không như mong muốn.

Việc sửa lỗi (debug) là một công việc không mấy dễ chịu đối với lập trình viên. Tuy nhiên giống như các nhà toán học, cảm giác giải được bài toán là cảm giác hưng phấn nhất và khiến cuộc đời trở nên đáng sống hơn bao giờ hết.

2 Tại sao lại dùng Python?

- Đơn giản, ngắn gọn, dễ đọc, dễ hiểu.
- Code linh động: không cần khai báo kiểu dữ liệu của biến hay hàm.
- Bộ nhớ đệm được tự động quản lý.
- Có hệ thống thư viện khổng lồ (bao gồm cả thư viện từ bên thứ ba), có cộng đồng năng động và rộng lớn.
- Python được thông dịch: từng dòng lệnh được dịch và chạy (execute). Tốn ít thời gian phân tích code và dừng ngay ở lệnh bị lỗi giúp dễ debug hơn.
- Python là một ngôn ngữ rất mạnh mẽ với rất nhiều hàm và cấu trúc dữ liệu được "làm sẵn" khiến cho công việc rất dễ dàng.

Tuy nhiên, do từng dòng lệnh được dịch trong lúc chương trình được thực thi (runtime) nên code Python chạy lâu hơn C/C++.

* Các khái niệm khác

- **CPython:** là một hệ thống xử lý (implementation[1]) code Python chính thức của Python Software Foundation. CPython có nền tảng là ngôn ngữ C, thường được đề cập để phân biệt với các implementations khác như Jython (nền tảng là Java). Trong phạm vi tài liệu này, mọi đề cập đến 'Python' xin được ngầm hiểu là 'CPython', trừ khi được nhắc cụ thể.
- **Cython:** là một phần mềm giúp chuyển code Python sang code C để chạy code nhanh hơn (transpiler).
- **Just-in-time (JIT) compiler:** là trình biên dịch chạy trong runtime giúp tìm ra các đoạn *bytecode* được interpreted quá nhiều lần để compile hẳn chúng ra ngôn ngữ máy để tiết kiệm thời gian chạy chương trình. JIT can thiệp được vào các thông tin trong runtime mà một compiler bình thường không làm được

[1]Implementation hàm chỉ tập hợp các cách thức, phương pháp để chuyển đổi code thành ngôn ngữ máy. Trong phần lớn trường hợp implementation là trình biên dịch (compiler) hoặc trình thông dịch (interpreter) hoặc có sự phối hợp của cả hai.

để tối ưu chương trình tốt hơn, chẳng hạn như tự động *inline* một hàm. **PyPy** là một dự án JIT vừa phát triển gần đây của ngôn ngữ Python giúp code có thể chạy nhanh hơn Python thông thường một cách đáng kể.

3 * Các khái niệm cơ bản trong Python

- **Package:** Một gói thư viện mở rộng cho Python có thêm nhiều tính năng để hoạt động tốt hơn và để người lập trình code nhanh hơn. Package có thể được phân phối (distribute) bởi Python Software Foundation (có sẵn khi cài Python) hoặc bên thứ ba (third party) như scipy, numpy, etc. (phải cài thêm thông qua *pip*).
- **Module:** Là thành phần con của Package. Một module có thể là một folder bao gồm nhiều module khác hoặc là một file, nhưng không nhỏ hơn một file. Thông thường người ta không quá khắt khe về việc phân biệt package và module.
- **Class:** Bên trong một module được định nghĩa nhiều class khác nhau. Ý nghĩa cơ bản của class là tạo ra một đối tượng, một kiểu dữ liệu mới bao gồm những *thuộc tính* và *hàm* tự định nghĩa hoặc tái định nghĩa để dễ kiểm soát, dễ theo dõi khi lập trình.
- **Function:** Một đoạn mã được copy & paste nhiều lần có thể được nhóm lại và định nghĩa thành một hàm để code nhanh hơn và linh động hơn qua các *tham số* (parameters).
- **pip:** Là một package manager giúp bạn cài đặt thêm những package mới theo nhu cầu, thông qua lệnh `pip install <package>` trên Terminal. Chữ 'pip' là viết tắt đệ quy của câu "Pip install packages" (Source: Wikipedia).

4 Làm sao để sử dụng Python?

Trên macOS có cài đặt sẵn Python 2. Trên Ubuntu có cài đặt sẵn Python 2 và Python 3. Để kiểm tra mình có Python trên máy chưa, vào Terminal (hoặc Command Prompt trên Windows) gõ lệnh `python -version` để xem có báo lỗi hay không.

Sau đây là các cách khác nhau để chạy code Python:

- Code trên file: Mở một file bằng editor (e.g. Notepad, Sublime, Atom, etc.) tùy ý và gõ code ở đó. Lưu lại thành một file có đuôi (extension) là `.py`. Vào Terminal chạy lệnh:
  `$ python file-name.py`
- Chạy từng câu lệnh một trên Terminal: Bằng cách gõ lệnh `python` trong Terminal, nó sẽ mở ra một process Python. Gõ từng lệnh một và enter sẽ cho ra kết quả.
- **IPython Notebook:** (recommended) Một môi trường lập trình Python có nền tảng HTML. Trong một notebook, bạn có thể viết code vào các 'khối' (cell) code và chạy cell đó một cách trực quan. Các khối khi nhận lệnh chạy sẽ nối đuôi nhau thực thi trong một process Python. Ngoài ra các phép tính toán và tài liệu có thể được trình bày tường minh trong các loại cell khác để bổ sung thông tin cho code. Có hỗ trợ viết bằng LaTeX và export notebook thành file LaTeX để các bạn đưa vào bài nghiên cứu của mình.

Trong tài liệu này cung cấp cho các bạn các ví dụ minh họa. Lưu ý rằng các ví dụ này là liên tiếp nhau, được đánh dấu bằng `In [x]:` và toàn bộ phần code được lùi

một đoạn vào trong. Cell 2 sẽ được chạy nối tiếp sau khi cell 1 kết thúc. Với mỗi cell có thể có kết quả in ra, kết quả này được lùi ra ngoài so với phần code.

Xin lưu ý rằng khi các bạn code Python trên Terminal / Command Prompt (tức là cứ gõ một lệnh rồi enter) thì các bạn không cần dùng hàm `print(var)` mà chỉ đơn thuần gõ `var` cũng đủ để hiện ra kết quả. Tuy nhiên, phương pháp chuẩn để hiện kết quả của biến vẫn là cách đầu tiên. Các bạn có thể tùy ý lựa chọn theo trường hợp.

5 Kiến thức căn bản về ngôn ngữ Python

Khác với sự gò bó của C++ và Java, chỉ với một cú pháp nhỏ trong Python lại sinh ra rất nhiều phép tính và thao tác đến mức không thể kể hết trong tài liệu này. Tuy nhiên, để giới thiệu các bạn đến với Python, sau đây là những kiến thức không thể thiếu:

- Các kiểu dữ liệu: `int, float, complex, bool, str, list, dict, tuple`. Cách ép kiểu dữ liệu.
- Các phép toán cơ bản (cộng, trừ, ...). Thao tác thêm, bớt, truy xuất phần tử, hàm sắp xếp. Thao tác cắt mảng.
- Vòng lặp và rẽ nhánh.
- Cách viết một hàm, một class.

6 Giới thiệu các package

6.1 `numpy`

`numpy` thường được sử dụng để tạo các object mảng đa chiều (multi-dimensional array). Mặc dù về cơ bản, Python đã có hỗ trợ mảng đa chiều (list & nested-list) nhưng:

- Trong Python, mảng không có sự thống nhất. Các phần tử có thể chứa bất kỳ dữ liệu kiểu gì cũng được. Mặc dù nghe có vẻ là một "tính năng" linh động nhưng trong nhiều trường hợp việc có một cấu trúc dữ liệu (data structure) ổn định và thống nhất lại được ưa chuộng hơn.
- Do tính chất nói trên của dữ liệu nên các thao tác với mảng được tối ưu rất nhiều qua việc tích hợp với C/C++ và Fortran.
- `numpy` hỗ trợ nhiều phép tính với ma trận như tìm ma trận nghịch đảo, tìm tích vô hướng giữa hai ma trận, etc.

Chúng ta bắt đầu với `numpy` bằng cách `import` thư viện này vào file Python (hoặc kernel) đang code.

Các chủ đề được thảo luận:

- Giới thiệu về array và matrix.
- Các phép tính với array và matrix.
- Các thông số quan trọng khi thống kê dữ liệu trong array.
- Pass by reference.
- Chọn lọc dữ liệu bằng cách dùng mặt nạ.
- Vector hóa một hàm

6.2 `scipy`