TD1: Đa thức một biến

# Mô tả

Đa thức một biến được biểu diễn dưới dạng

trong đó .

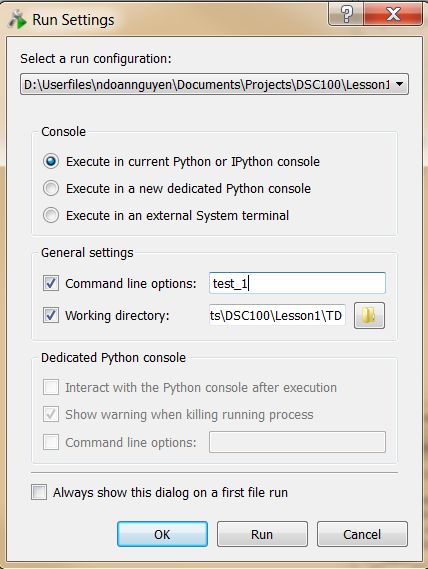
Trong TD này ta làm việc với các đa thức với hệ số thực. Ta muốn xây dựng một class Polynomial trong Python thực hiện những phép toán cơ bản trên đa thức như cộng, trừ, nhân, chia, luỹ thừa, tìm ước chung lớn nhất và bội chung nhỏ nhất, tìm nghiệm của đa thức…

Bạn cần viết code trong file Polynomial.py và chạy các test trong TestPolynomial.py. Trong file Polynomial.py, bạn cần hoàn thành các class Polynomial, LinearPolynomial, QuadraticPolynomial và CubicPolynomial. Ngoài ra đã có một class PolynomialError được viết sẵn mà bạn có thể sử dụng. Bạn có thể viết các hàm phụ khác ngoài các hàm đã yêu cầu trong đề.

Bạn có thể sử dụng các hàm từ thư viện khác miễn là hàm đó không trực tiếp trả lời câu hỏi trong bài (mục đích của TD là để bạn làm quen với việc thực hiện các thao tác trên class và instance).

Để test, bạn có thể sử dụng file TestPolynomial.py. Bạn không cần thay đổi nội dung file này trừ mục đích sửa syntax để với Python3.

Để chạy từng test, trong Spyder, bạn chọn Run -> Configure và gõ tên test tương ứng với bài tập (test\_1, test\_2, test\_3, …, test\_13)



Nếu code viết đúng, khi chạy code bạn sẽ nhìn thấy dòng “Test … OK”. Nếu sai, bạn sẽ nhìn thấy ví dụ giải thích vì sao sai.

# Bài 1. Tạo đa thức

Ta biết rằng một đa thức hoàn toàn được xác định nếu các hệ số của nó được xác định. Giả sử ta biết các hệ số của nó và biểu diễn nó trong một list của Python.

Ví dụ với đa thức , list hệ số tương ứng là [1, 2, 3, 0, 0, 4].

1. Hãy viết hàm **\_\_init\_\_** trong class Polynomial để khởi tạo đa thức, khi biết list các hệ số của nó. Chú ý rằng đôi khi list này có thể kết thúc bằng các số 0, như [1, 2, 3, 0, 0, 0].

*Ví dụ, khi gọi*

*p = Polynomial([1, 2, 3, 0, 0, 4])*

*thì p được khởi tạo như một instance của class Polynomial.*

1. Hãy viết hàm **getCoefficients** trong class Polynomial để trả lại list các hệ số của một đa thức.

*Ví dụ,*

*p = Polynomial([1, 2, 3, 0, 0, 4])*

*p.getCoefficients()*

*thì nhận được kết quả [1, 2, 3, 0, 0, 4].*

Chú ý list này phải ở dạng sao cho hệ số cuối cùng (cao nhất) khác 0. Ví dụ, list hệ số của đa thức không là [].

# Bài 2. Bậc của đa thức

Viết hàm **getDegree** trong class Polynomial để trả lại một số nguyên là bậc của một đa thức.

Ta quy ước bậc của đa thức không là -1.

*Ví dụ, trong ví dụ trên, khi gọi*

*p.getDegree()*

*thì nhận được kết quả 5.*

# Bài 3. Cộng đa thức

Viết hàm **add** trong class Polynomial nhận tham số là một đa thức khác và trả lại kết quả là một Polynomial bằng tổng của hai đa thức.

*Ví dụ,*

*p = Polynomial([1, 2, 3, 0, 0, 4])*

*q = Polynomial([1, 2, 3, 0, 0, -4])*

*r = p.add(q)*

*r.getCoefficients()*

*thì nhận được kết quả [2, 4, 6].*

# Bài 4. Trừ đa thức

Viết hàm **substract** trong class Polynomial nhận tham số là một đa thức khác và trả lại kết quả là một Polynomial bằng hiệu của đa thức đang làm việc với đa thức mới.

Bạn có thể làm trực tiếp phép trừ trên các list hoặc viết một hàm tính “đa thức đối” rồi áp dụng bài 3.

*Ví dụ,*

*p = Polynomial([1, 2, 3, 0, 0, 4])*

*q = Polynomial([1, 2, 3, 0, 0, -4])*

*r = p.substract(q)*

*r.getCoefficients()*

*thì nhận được kết quả [0, 0, 0, 0, 0, 8].*

# Bài 5. Nhân đa thức

Viết hàm **multiply** trong class Polynomial nhận tham số là một đa thức khác và trả lại kết quả là một Polynomial bằng tích của hai đa thức.

# Bài 6. Luỹ thừa một đa thức

Viết hàm **power** trong class Polynomial nhận tham số là một số nguyên không âm và trả lại kết quả là một Polynomial bằng luỹ thừa của đa thức đó với số mũ tương ứng.

# Bài 7. Chia đa thức

Viết hàm **divide** trong class Polynomial nhận tham số là một đa thức khác và trả lại kết quả là một tuple gồm hai Polynomial, một là thương và một là đa thức dư của phép chia. Nếu đa thức chia bằng 0, raise một Exception thuộc type PolynomialError.

*Ví dụ,*

*p = Polynomial([1, 0, 1])*

*q = Polynomial([0, 1])*

*r, s = p.divide(q)*

*r.getCoefficients()*

*s.getCoefficients()*

*thì nhận được kết quả [0, 1] và [1].*

# Bài 8. Tính chia hết của đa thức

Viết hàm **isDivisor** trong class Polynomial nhận tham số là một đa thức mới và trả lại kết quả là True nếu đa thức đang làm việc chia hết đa thức mới, False nếu không. Nếu đa thức đang làm việc bằng 0, kết quả cũng là False.

*Ví dụ,*

*p = Polynomial([1, 1])*

*q = Polynomial([1, 0, -1])*

*p.isDivisor(q)*

*thì nhận được kết quả True.*

# Bài 9. Ước chung lớn nhất

Viết hàm **getGcd** trong class Polynomial nhận tham số là một đa thức mới và trả lại kết quả là ước chung lớn nhất của chúng. (Nhắc lại rằng ước chung lớn nhất có hệ số cao nhất là 1).

*Ví dụ,*

*p = Polynomial([1, 0, -1])*

*q = Polynomial([1, 0, 0, -1])*

*g = p.getGcd(q)*

*g.getCoefficients()*

*thì nhận được kết quả [-1, 1].*

# Bài 10. Đa thức bậc nhất, bậc hai và bậc ba

Hãy viết các class LinearPolynomial, QuadraticPolynomial, CubicPolynomial thừa kế class Polynomial, sao cho nếu list hệ số không ứng với một đa thức bậc nhất, hai, ba thì chương trình báo lỗi.

*Ví dụ,*

*p = LinearPolynomial([1, 2])*

*q = QuadraticPolynomial([1, 2, -1])*

*r = CubicPolynomial([1, 2, 3, -1, 2])*

*thì hai dòng đầu OK và dòng thứ ba báo lỗi.*

# Bài 11. Nghiệm của đa thức bậc nhất và bậc hai

Viết hàm **getRoots** trong class Polynomial trả lại list [] cho mọi đa thức thuộc class này.

Override hàm **getRoots** trong các class con LinearPolynomial, QuadraticPolynomial để trả lại list các nghiệm phức của các đa thức bậc nhất và bậc hai thay vì list rỗng [].

Lưu ý, list trả về cần được sắp xếp phần thực tăng dần. Nếu phần thực bằng nhau thì xếp phần ảo tăng dần.

*Ví dụ,*

*p = LinearPolynomial([1, 2])*

*q = QuadraticPolynomial([1, 0, 1])*

*r = CubicPolynomial([1, 2, 3, -1])*

*p.getRoots()*

*q.getRoots()*

*r.getRoots()*

*thì nhận được kết quả [-0.5], [-1j, 1j], [].*

(Lưu ý, ta chỉ làm việc với các đa thức có hệ số thực, nhưng đa thức vẫn có thể có nghiệm phức). Nếu có nghiệm kép, nghiệm kép được liệt kê 2 lần trong list.

# Bài 12. Nghiệm (gần đúng) của đa thức

Implement thuật toán Muller trong trang 42, 43 của bài báo sau để viết bổ sung vào hàm **getRoots** cho class Polynomial, giúp tìm tất cả các nghiệm phức của mọi đa thức với hệ số thực.

<http://www.math.niu.edu/~dattab/MATH435.2013/ROOT_FINDING.pdf>

*Ví dụ,*

*p = Polynomial([-16, 0, 0, 0, 0, 0, 0, 0, 1]) #x^8 – 16 = 0*

*p.getRoots()*

*thì nhận được kết quả*

*[-1.4142139079116434,*

*(-1.0000000371326812-1.0000000304500047j),*

*(-1.0000000371326812+1.0000000304500047j),*

*(-6.582452068848053e-09-1.4142136149947437j),*

*(-6.582452068848053e-09+1.4142136149947437j),*

*(1.000000187078139-1.0000000807868332j),*

*(1.000000187078139+1.0000000807868332j),*

*(1.4142136211856318+0j)]*

* Yêu cầu: Không sử dụng thư viện trực tiếp. (Ví dụ trang sau cho ta một hàm thực hiện yêu cầu của bài tập <https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.roots.html> )
* Có thể sử dụng thư viện cmath của Python để tìm các hàm cơ bản về số phức như phần thực, phần ảo, tìm liên hợp, tính căn bậc hai của số phức.
* Nếu muốn tự test, bạn chỉ cần test các đa thức bậc 7, 8 trở xuống với các hệ số vừa phải (có giá trị tuyệt đối lớn hơn 0.1 và nhỏ hơn 100), các đa thức bậc cao hơn liên quan đến vấn đề về sai số, không cần giải quyết trong TD này.
* Bạn có thể làm tròn để kết quả dễ nhìn hơn.

# Bài 13. Cực đại, cực tiểu địa phương của đa thức

Từ kết quả của bài 12, hãy viết hàm **getLocalMin** và **getLocalMax** trong class Polynomial để trả về list tất cả các số thực là điểm cực tiểu (tương ứng, cực đại) của một đa thức, xếp theo thứ tự tăng dần.

Ví dụ

*p = Polynomial([-16, 0, 0, 0, 0, 0, 0, 0, 1]) #x^8 – 16 = 0*

*p.getLocalMin()*

thì nhận được kết quả [0.0].

còn *p.getLocalMax()*

thì nhận được kết quả [].

Bạn có thể viết một hàm phụ **getDerivative** để tính đạo hàm (và đạo hàm cấp cao, nếu cần) của đa thức.

------------Kết thúc------------