Mục lục

[I.](#_heading=h.lnxbz9) Solver 2

[1.](#_heading=h.2xcytpi) Hàm solve() 2

[2.](#_heading=h.1ci93xb) Hàm solve\_linear() 4

[3.](#_heading=h.3whwml4) Hàm solve\_linear\_system() 5

[4.](#_heading=h.2bn6wsx) Hàm solve\_undetermined\_coeffs() 5

[5.](#_heading=h.qsh70q) Hàm nsolve() 6

[6.](#_heading=h.3as4poj) Hàm checksol() 6

[7.](#_heading=h.1pxezwc) Hàm unrad() 7

[II.](#_heading=h.49x2ik5) Scipy 8

[1.](#_heading=h.2p2csry) Sử dụng linprog 8

[2.](#_heading=h.147n2zr) Tổng kết 11

[III.](#_heading=h.3o7alnk) Pulp 12

[1.](#_heading=h.23ckvvd) Sử dụng pulp 12

[IV. Ortools 16](#_heading=h.s2cg9ievyf9o)

[1.](#_heading=h.dgjvndsen18j) Sử dụng liner\_solver 16

# Solver

## Hàm solve()

### Sử dụng hàm solve()

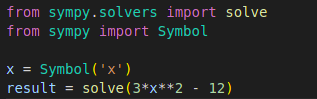
Ứng dụng: Dùng để giải phương trình và hệ phương trình.

+ Mọi phương trình ta đều cho vế phải bằng 0

+ Ta có thể trả về các dạng dữ liệu khác nhau với flag (dict, set, …) bằng cách thêm tham số vào sau hàm (VD: solve(3\*x + 2, dict=True)

+ Ta có thể trả về kết quả theo nghiệm có trong phương trình, ví dụ đối với phương trình 5x + y = 0, nếu ta dùng solve(5\*x + y, y) thì kết quả sẽ ra là [-5\*x]

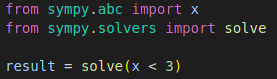
#### VD 1.1.1: Phương trình 3x2 = 12



- Đầu tiên chúng ta import các thư viện, các hàm cần thiết  
- x = Symbol(‘x’) có nghĩa là chúng ta khai báo x là một Symbol (biểu tượng toán học), ở đây x là ẩn số

- Ta sẽ chuyển phương trình sang dạng 3x2 - 12 = 0 để   
- Phương trình 3x2 - 12 được chuyển sang code thành dạng 3\*x\*\*2 - 12, tham số đầu tiên của hàm solve() sẽ là phương trình   
- Kết quả khi ra sẽ ra là:    
- Để lấy kết quảlà một list các giải pháp, ta gán cờ dict=True trong hàm solve()  
  
- Kết quả khi giải quyết là:   
- Để lấy kết quảlà một set các ký hiệu và giải pháp, ta gán cờ set=True trong hàm solve()  
  
- Kết quả khi giải quyết là: 

#### VD 1.1.2: Biểu thức x < 3



- Thư viện sympy.abc giúp chúng ta xuất các chữ cái sang dạng biểu tượng tương tự với câu lệnh x = Symbol(‘x’)  
- Kết quả là: 

#### VD 1.1.3: Phương trình 3x + y = 0

Text

Description automatically generated

- Ta thêm x là một tham số của hàm solve  
- Kết quả là :

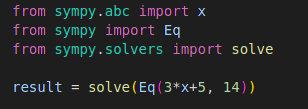
- Tương tự ta có thể lấy kết quả theo nghiệm y  
   
- Kết quả là: 

### Sử dụng Eq()

Ứng dụng: giúp chúng ta biểu diễn 2 đôi tượng bằng nhau

VD: để biểu diễn phương trình 15x + 3 = y, ta dùng Eq(15\*x + 3, y) với tham số đầu là vế trái, tham số sau là vế phải

#### VD 1.2.1: Phương trình 3x + 5 = 14



- Kết quả là: 

### Sử dụng diff()

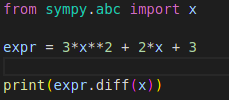
Ứng dụng: giúp chúng ta tìm đạo hàm của một đa thức

VD: 3x2 + 2x + 5, để tìm đạo hàm ta dùng hàm diff() như sau:

diff(3\*x\*\*3 + 5\*x + 5)

Kết quả sẽ ra là 9\*x\*\*2 + 5

#### VD 1.3.1: Đạo hàm của phương trình 3x2 + 2x + 3



- diff(x) chỉ ra đạo hàm theo x

- Kết quả là: 

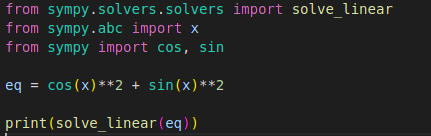
## Hàm solve\_linear()

### Sử dụng hàm solve\_linear()

Ứng dụng: dùng để giải các phương trình tuyến tính, hàm sẽ trả về tuple như sau:

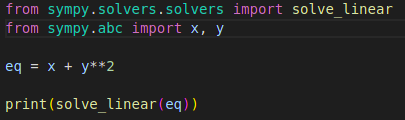
* (0,1): hàm f không phụ thuộc vào các kí hiệu đã cho
* (0,0): hàm không có giải pháp
* (symbol, solution): giải pháp của ký hiệu
* (n,d): n là tử số và d là mẫu số khi n không tuyến tính với bất kì ký hiệu nào

#### VD 2.1.1: Phương trình cos2x + sin2x

****

Vì cos2x + sin2x = 1, kết quả không phục thuộc vào ký hiệu x nên kết quả sẽ ra là: 

#### VD 2.1.2: Phương trình x + y2



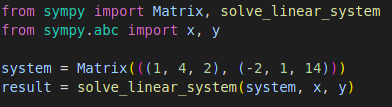
Kết quả sẽ ra là: 

## Hàm solve\_linear\_system()

### Sử dụng hàm solve\_linear\_system()

Ứng dụng: Giải hệ gồm N phương trình tuyến tính với M biến

#### VD 3.1.1: Hệ phương trình x + 4y = 2 và -2x + y = 14

****

Đầu tiên khai báo system là một ma trận phương trình

Dùng system là một tham số và các ký hiệu trong hàm solve\_linear\_system

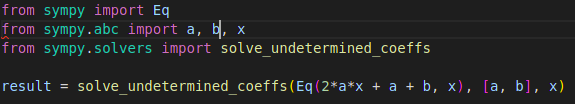
Kết quả giải quyết là: 

## Hàm solve\_undetermined\_coeffs()

### Sử dụng hàm solve\_undetermined\_coeffs()

Ứng dụng: Tìm các hệ số chưa xác định thỏa p(x;a1,...,ak) = q(x) với p và q là đa thức đơn biến phụ thuộc vào k tham số

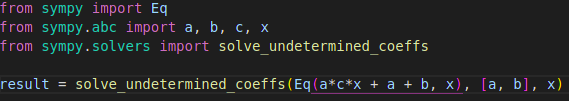
#### VD 4.1.1: Phương trình 2ax + a + b = x

****

Tham số đầu tiền là phương trình 2ax + a + b = x, tham số thứ hai là các hệ số, tham số thứ ba là các ký hiệu

Kết quả giải quyết là: 

#### VD 4.1.1: Phương trình acx + a + b = x



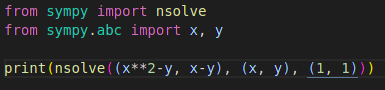
kết quả giải quyết là:

## Hàm nsolve()

### Sử dụng hàm nsolve()

Ứng dụng: dùng để giải cái phương trình phi tuyến tính (phương trình đại số bậc 2 trở lên)

#### VD 5.1.1: Phương trình x2-y và x-y



Tham số đầu vào là hệ phương trình

Tham số thứ hai là các ký hiệu

Tham số thứ ba là khoảng đoán của kết quả

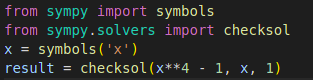
Kết quả giải quyết là: 

## Hàm checksol()

### Sử dụng hàm checksol()

Ứng dụng: dùng để kiểm tra xem giải pháp có phải bằng 0 hay không

#### VD 6.1.1: Phương trình x4-1, phép thử 1



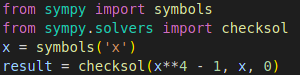
Tham số đầu vào là phương trình

Tham số thứ hai là các ký hiệu

Tham số thứ ba là kết quả thử

Kết quả sẽ ra là: 

#### VD 6.1.2: Phương trình x4-1, phép thử 0



Các tham số tương tự VD 6.1

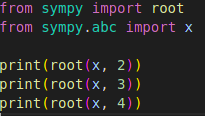
Kết quả sẽ ra là: 

## Hàm unrad()

### Sử dụng hàm root()

Ứng dụng: dùng để lấy căn của một biểu thức

#### VD 7.1.1: Lấy căn của x



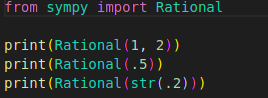
Với tham số đầu là x và tham số thứ hai là cấp độ lấy căn, ví dụ root(x, 3) là căn bậc 3 của x

Kết quả sẽ ra là: 

### Sử dụng Rational()

Ứng dụng: dùng để trả về dạng số hữu tỉ

#### VD 7.2.1: Lấy số hữu tỉ của một số



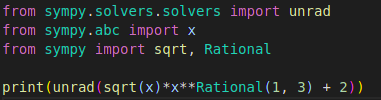
Dùng str() để biểu diễn float dưới dạng đơn giản hơn

Kết quả sẽ ra là:

### Sử dụng unrad()

Ứng dụng: dùng để khai căn

#### VD 7.3.1: Khai căn cho biểu thức



Với phương trình là = 0 thì khi khai căn sẽ ra là

Kết quả là 

# Scipy

## Sử dụng linprog

* Ứng dụng: linprog dùng để giải quyết tối thiểu hóa vấn đề và không chấp nhận các điều kiện có dấu lớn hơn hay dấu lớn hơn hoặc bằng. Vì thế chúng ta phải đảo dấu mội vài phương trình

#### VD 1.1.1: Ta có phương trình như sau:

Text

Description automatically generated

- Thay vì chúng ta maximize z = x + 2y, chúng ta có thể minimize dạng âm của nó

(-z = -x -2y)

- Ở phương trình màu vàng chúng ta có dấu lớn hơn, hãy biến đổi nó thành dấu nhỏ hơn

(x - 2y <= 2)

- Sau những thay đổi đó, ta có hệ phương trình mới :

Text

Description automatically generated

- Đầu tiên ta import thư viện linprog



- Bước tiếp theo chúng ta định nghĩa các tham số đầu vào

A screenshot of a computer screen

Description automatically generated with medium confidence

**obj:** giữ các hệ số của x y trong z

**lhs\_ineq:** giữ các hệ số vế **trái** của bất bình đẳng thức (đỏ, xanh, vàng)

**rhs\_ineq:** giữ các hệ số vế **phải** của bất bình đẳng thức (đỏ, xanh, vàng)

**lhs\_eq:** giữ các hệ số vế **trái** của bình đẳng thức (xanh lá)

**rhs\_eq:** giữ các hệ số vế **phải** của bình đẳng thức (xanh lá)

- Bước tiếp theo là xác định giới hạn của x và y, ở đây là x >= 0, y >= 0 là nằm giữa từ 0 tới dương vô cùng



- Cuối cùng tối ưu hóa và giải quyết vấn đề dùng hàm linprog:

Text

Description automatically generated

**c:** hệ số từ hàm mục tiêu (hàm z)

**A\_ub, b\_ub:** hệ số vế trái, vế phải của bất bình đẳng thức

**A\_eq, b\_eq:** hệ số vế trái, vế phải của bình đẳng thức

**bounds:** giới hạn khoảng của x, y

**method:** bạn có thể dùng method để định nghĩa phương pháp lập trình tuyến tính muốn sử dụng:

* interior-point: phương pháp điểm bên trong
* revised simplex: phương pháp đơn giản 2 pha đã sửa đổi
* simplex: phương pháp đơn giản 2 pha kế thừa

- Hàm linprog sẽ trả về một cấu trúc dữ liệu với các thuộc tính:

Text

Description automatically generated

Ta quan tâm các kết quả sau:

**fun:** gái trị tối ưu của hàm z (nếu tìm thấy)

**nit:** số lần lặp cần thiết để kết thúc phép tính

**slack:** giá trị còn dư của các phương trình

* số đầu là của phương trình 1 (2x + y <= 20), khi x = 7.72.. và y = 4.5454
* các số sau là phần dừ của các phương trình còn lại

**success:** hiển thị kết quả có được tối ưu hay không

**x:** giữ giá trị tối ưu (ở đây là x và y)

#### VD 1.1.2: ta có phương trình sau:

Text

Description automatically generated

Như ví dụ 1.1.1, ta có chương trình như sau:

Text

Description automatically generated

Kết quả là:

Text, chat or text message

Description automatically generated

## Tổng kết

* Scipy chỉ phù hợp với các vấn đề nhỏ, chúng ta có thể tìm những thư viện khác với các lí do như sau:
* Scipy không thể chạy các bộ giải bên ngoài
* Không thể hoạt động với các biến quyết định số nguyên
* Không hỗ trợ chức năng xây dựng mô hình, bạn phải xác định mảng và ma trận
* Không cho phép xác định trực tiếp các vấn đề về tối đa hóa, phải chuyển đổi chúng thành các vấn đều tối thiểu
* Sử dụng các dấu nhỏ hơn thay vì các dấu lơn hơn

# Pulp

## Sử dụng pulp

* Ứng dụng: Pulp có một API lập trình tuyến tính thuận tiện hơn Scipy. Bạn không cần phải chuyển đổi các vấn đề toán học, mọi thứ sạch và ít lỗi hơn
* Đầu tiên ta import thư viện



#### VD 1.1.1: Ta có phương trình sau

Text

Description automatically generated

* Đầu tiên ta khai báo một LpProblem đại diện cho model:



**sense:** Ta có thể chọn LpMinimize hoặc LpMaximize (tối thiểu hoặc tối đa)

* Tiếp theo ta khai báo các giá trị x, y



Do x >= 0 và y >= 0 nên ta sẽ đặt lowBound = 0 (thấp nhất là 0)

* Sau đó chúng ta dùng các biến x, y vừa đặt để tạo các biểu thức và ràng buộc khác

A screenshot of a video game

Description automatically generated with medium confidence

* Chúng ta không cần phải thêm dữ liệu dạng ma trận hay danh sách, chỉ cần viết các biểu thức và toán tử và nối chúng vào mô hình

Text

Description automatically generated

* LpProblem cho phép bạn thêm các bộ giá trị ràng buộc và tên của chúng, phần tử đầu tiên là LpConstraint, phần tử thứ hai là tên của chúng
* Khai báo hàm mục tiêu



* dùng solve() để gọi bộ giải cơ bản, sửa đổi đối tượng model và sẽ trả về giá trị tối ưu, đối với phần còn lại của mã trạng thái, xem LpStatus[].



* dùng value() để xem các kết quả được trả về dưới dạng thuộc tính của model

Text

Description automatically generated

Kết quả là:

Text

Description automatically generated

* Lưu ý với hàm solve(), nó sẽ thay đổi trạng thái của đối tượng x và y
* Chúng ta có thể xem mình đang dùng bộ giải nào bằng cách:



* Ta có thể thay đổi bộ giải, ví dụ như bộ giải GLPK, ta làm như sau



và thay đổi ở chỗ status



* Ta gán tham số solver=GLPK, msg dùng để thông báo thông tin của bộ giải, nếu bạn muốn thông tin bộ giải thì ta bỏ qua tham số msg hoặc đặt msg=True
* Pulp có thể giải các bài toán lập trình tuyến tính số nguyên hỗn hợp, để khai báo biến số nguyên hay số nhị phân, chỉ cần thêm cat=”Integer” hoặc cat=”Binary” vào LpVariable



#### VD 1.1.2 Ta có phương trình sau:

Text

Description automatically generated

* Dựa vào VD 2.1, ta có chương trình như sau:

Text

Description automatically generated

* Kết quả sẽ ra là

Text

Description automatically generated

* Giả sử x1 là dương và x3 phải là 0 hoặc ngược lại, ta sẽ sử dụng 2 biến quyết định nhị phân y1 và y3:

A screenshot of a computer

Description automatically generated with medium confidence

**Dòng 8:** khai báo biến quyết định nhị phân y

**Dòng 15:** khai báo một số lớn tùy ý, 100 là đủ lớn trong trường hợp này

**Dòng 16:** nếu y1 = 0 thì x1 bằng 0, nếu không thì nó có thể là số bất kỳ không âm

**Dòng 17:** nếu y3 = 0 thì x3 bằng 0, nếu không thì nó có thể là số bất kỳ không âm

**Dòng 18:** nếu y1 hoặc y3 (hoặc cả hai) bằng 0 thì x1 hoặc x3 (hoặc cả hai) cũng phải bằng 0

Đây là kết quả:

Text

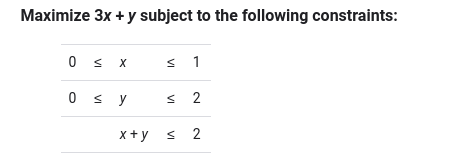
Description automatically generated

# IV. Ortools

## Sử dụng liner\_solver

Ứng dụng: là một bộ giải phương trình tuyến tính của Ortools

#### VD 1.1.1: Maximize 3x+y với các ràng buộc sau:



- Đầu tiên ta import các thư viện cần thiết



- Tiếp theo ta khai báo bộ giải, ở đây là GLQP

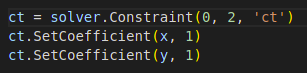


- Tạo các biến x và y với ràng buộc, sử dụng NumVar



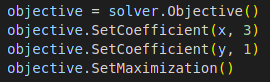
+ 0 <= x <= 1 và 0 <= y <= 2

- Tiếp theo khai báo ràng buộc tuyến tính (x + y <= 2)



+ **SetCoefficient** dùng để cài đặt hệ số x y của ràng buộc

- Tiếp theo, tạo hàm mục tiêu



+ **SetMaximization** cho biết đây là vấn đề cần được tối đa hóa

- Bước cuối cùng là kích hoạt solver và hiển thị kết quả

Text

Description automatically generated