Numerical Python

CS*101* `lec15`

Solving Equations

2019-11-06

Announcements

quiz: `quiz15` due on Thurs 07/11

lab: `lab` on Fri 08/11

hw: `hw07` due today

`exam02` on 13/11

Roadmap

- Week 1 (9/9): L0 Intro; L1 PY - Intro
- Week 2 (16/9): L2 PY – Data& Bool; L3 PY – if & Loops
- Week 3 (23/9): L4 PY - Lists; L5 PY - Funcs
- 30/9: L6 PY - Files; ✗
- 9/10: L7 PY - Muta
- 12/10: Exam 01
- Week 6 (14/10): L8 PY - Dict; L9 PY - IO
- Week 7 (21/10): L10 PY - Numpy; L11 PY - Plot
- Week 8 (28/10): L12 PY - State; L13 PY - Rand
- Week 9 (4/11): L14 PY – De-bug; L15 PY - Eqn
- Week 10 (11/11): L16 PYA I; Exam 02
- Week 11 (18/11): L17 PY - Opt II; L18 PY - Mod
- Week 12 (25/11): L19 PY – Sym I; L20 PY – Sym II
- Week 13 (2/12): L21 PY – Sym III; L22 ML - Intro
- Week 14 (9/12): L23 ML - IO; L24 ML - Poly
- Week 15 (16/12): L25 ML - Stats; L26 ML - Fit
- Week 16 (23/12): Review
- Week 17 (30/12): Finals

Objectives

A. Represent and solve equations in an efficient manner. => Similar to **lec on Random and Numpy**
B. Locate a function’s zeroes using a graphical method or Newton’s method. => **lec on plotting**
C. Locate a function’s minima using a graphical method or ‘scipy.optimize.minimize‘.

Question

```
x = np.ones( 10 )
for i in range( 10 ):
    try:
        ???
    except:
        print( 'Error on step %d.'%i )
        continue
```

Which of the following candidates for `???` would *not* produce an error message?

A `x += x[ i+1 ]`

B `x[ i ] /= 0`

C `x[ -i-1 ] = sum( x[ :i ] )`

D `x[ 10-i ] = sum( x[ :i ] )`

Question

```
x = np.ones( 10 )
for i in range( 10 ):
    try:
        ???
    except:
        print( 'Error on step %d.'%i )
        continue
```

Which of the following candidates for `???` would *not* produce any error message?

A `x += x[ i+1 ]` index error

B `x[ i ] /= 0` ⋆(surprise! `numpy` can handle)

C `x[ -i-1 ] = sum( x[ :i ] )` ⋆

D `x[ 10-i ] = sum( x[ :i ] )` index error

Equations

Equations

How do we represent equations on computers?

Equations

How do we represent equations on computers?

A. As a function

B. Write some expressions

C. Write as a series

D. Write as symbolic terms (in later lectures)

... (more)

Equations

How do we represent equations on computers?

A. As a function

B. Write some expressions

C. Write as a series

D. Write as symbolic terms (in later lectures)

... (more)

1. In other words, we convert the equation into something that can be calculated. We want *numbers* out of them.

Equations

How do we represent equations on computers?

A. As a function

B. Write some expressions

C. Write as a series

D. Write as symbolic terms (in later lectures)

... (more)

1. In other words, we convert the equation into something that can be calculated. We want *numbers* out of them.
2. Many times we represent the function as a pair of arrays, `x` and `y` (like for plotting).
3. We can also represent equations using symbols from the library `sympy`, (later lectures).

Equations

Suppose you wish to evaluate the function:

$$y = a\sin^3 x + b\sin^2 x + c\sin x + d$$

Equations

Suppose you wish to evaluate the function:

$$y = a\sin^3 x + b\sin^2 x + c\sin x + d$$

On a computer, which way is better?

```
A.  y = a*sin(x)**3 + b*sin(x)**2 + c*sin(x) + d

B.  t = sin(x)
y = a*t**3 + b*t**2 + c*t + d
```

Equations

Suppose you wish to evaluate the function:

$$y = a\sin^3 x + b\sin^2 x + c\sin x + d$$

On a computer, which way is better?

```
A.  y = a*sin(x)**3 + b*sin(x)**2 + c*sin(x) + d
```

```
B.  t = sin(x)
y = a*t**3 + b*t**2 + c*t + d
```

The first way takes three times longer!

$\sin$ is calculated every single time it is used.

Equations

What about calculating π? Which is faster?

A. The Monte Carlo method?

B. Series solution?

$$\frac{\pi}{4} = +1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \frac{1}{13} - \frac{1}{15}, \ldots$$

π using Monte Carlo

```
import numpy.random as npr
def mc_pi( n ):
  xy = npr.rand( n,2 ) * 2 - 1
  n_circle = 0
  for pair in xy:
    ........
  return estimate
```

π using Series summation

```
def series_pi( n ):
  result = 0
  for k in range( 1,n ):
    term = ( ( -1 ) ** ( k+1 ) ) / ( 2 * k - 1 )
    result += term
  return result*4
```

Equations

Which way is more efficient computationally?

Equations

Which way is more efficient computationally?

The series solution is much better, and other better ways may exist.

We can quantify this if we can compare algorithm.

How to quantify?

Code performance

In order to compare algorithms, we need a way to measure a code's runtime (called "wallclock time").

Code performance

In order to compare algorithms, we need a way to measure a code's runtime (called "wallclock time").

`timeit` module provides three ways to time your code:

Code performance

In order to compare algorithms, we need a way to measure a code's runtime (called "wallclock time").

`timeit` module provides three ways to time your code:

```
» import timeit
```

A. Interpreter: `timeit.timeit(code, number=10000)`

Code performance

In order to compare algorithms, we need a way to measure a code's runtime (called "wallclock time").

`timeit` module provides three ways to time your code:

» `import timeit`

A. Interpreter: `timeit.timeit(code, number=10000)`

i. » `timeit.timeit('some code as string here', number=10000)` or

ii. » `code` = ́some code but as a string ́

» `timeit.timeit(code, number=10000)`

or

iii. » `timeit.timeit(code, setup = optional, number=10000)`

Code performance

In order to compare algorithms, we need a way to measure a code's runtime (called "wallclock time").

`timeit` module provides three ways to time your code:

- `» import timeit`
- A. Interpreter: `timeit.timeit(code, number=10000)`
  - i. `» timeit.timeit('some code as string here', number=10000)` or
  - ii. `» code` = śome code but as a string ´ `» timeit.timeit(code, number=10000)` or
  - iii. `» timeit.timeit(code, setup = optional, number=10000)`
- `number` = the number of times to run to get an average time
- `setup` = setup python before running `code`. e.g., setup = import math

Code performance

```
import timeit
```

B. Jupyter notebook: `%timeit codeJupyter` (this is easiest)

`codeJupyter` is just your `def` function

These commands run your code many times and return an average time to completion.

```
%timeit  mc_pi( 1e5 )
%timeit  series_pi( 1e5 )
```

Code performance example

Jupyter:

```
def fib_a( n ):
    sqrt_5 = 5**0.5;
    p = ( 1 + sqrt_5 ) / 2;
    q = 1 / p;
    return int( (p**n + q**n) / sqrt_5 + 0.5 )

%timeit -n 10 fib_a(50)
```

`-n 10` means run 10 times

Fibonacci sequence example

$$F_n = F_{n-1} + F_{n-2} \qquad F_1 = F_2 = 1$$

$$1\,,1\,,2\,,3\,,5\,,8\,,13\,,21\,,34\,,55\,,...$$

Fibonacci sequence example

$$F_n = F_{n-1} + F_{n-2} \qquad F_1 = F_2 = 1$$

$$1\,,1\,,2\,,3\,,5\,,8\,,13\,,21\,,34\,,55\,,...$$

The closed-form formula for the nth Fibonacci term is:

$$F_n = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^n + \left(\frac{2}{1+\sqrt{5}}\right)^n}{\sqrt{5}+\frac{1}{2}}$$

Analytical Fibonacci

```
def fib_a( n ):
    sqrt_5 = 5**0.5;
    p = ( 1 + sqrt_5 ) / 2;
    q = 1 / p;
    return int( (p**n + q**n) / sqrt_5 + 0.5 )
```

Recursive Fibonacci

```
def fib_r( n ):
    if n == 1 or n == 2:
        return 1
    else:
        return fib_r( n-1 ) + fib_r( n-2 )
```

Comparison

```
%timeit fib_a( 12 )
%timeit fib_r( 12 )
```

Comparison

```
%timeit fib_a( 12 )
%timeit fib_r( 12 )
```

On my machine, `fib_a` is 55 $\times$ faster than `fib_r` for `n = 12`.

Will this performance get better or worse for larger `n`?

Equations - series

How do you calculate the value of $\sin x$ or $\exp x$? or $\exp(-x)$?

Equations - series

How do you calculate the value of $\sin x$ or $\exp x$? or $\exp(-x)$?

$$\begin{aligned} \exp(-x) &= 1 - x + \frac{x^2}{2} - \frac{x^3}{6} + ... \\ &= \frac{x^0}{0!} - \frac{x^1}{1!} + \frac{x^2}{2!} - \frac{x^3}{3!} + \frac{x^4}{4!} + ... \end{aligned}$$

Equations - series

How do you calculate the value of $\sin x$ or $\exp x$? or $\exp(-x)$?

$$\begin{aligned}\exp(-x) &= 1 - x + \frac{x^2}{2} - \frac{x^3}{6} + \ldots \\ &= \frac{x^0}{0!} - \frac{x^1}{1!} + \frac{x^2}{2!} - \frac{x^3}{3!} + \frac{x^4}{4!} + \ldots\end{aligned}$$

This series is well-behaved, *but*...

Equations - series

Intermediate terms can behave like:

if x = 10,

$$\frac{10^5}{5!} = \frac{100,000}{120} = 833.333$$

Equations - series

Intermediate terms can behave like:
if x = 10,

$$\frac{10^5}{5!} = \frac{100,000}{120} = 833.333$$

or

$$\frac{10^{12}}{12!} = \frac{1,000,000,000,000}{479,001,600} = 2,087.675$$

Very large numbers result, leading to inefficient calculation and possible numerical error.

Alternating negative terms will also lead to numerical errors.

So what can we do?

Equations - series

To break a big number to a combination of smaller number.
In this case, use

$$e^x = \frac{1}{e^{-x}}$$

$$e^x = \left(e^{\frac{x}{n}}\right)^n$$

Original:

$$e^{12} = \frac{12^0}{0!} - \frac{12^1}{1!} + \frac{12^2}{2!} - \frac{12^3}{3!} ... + \frac{12^{10}}{12!} + ..$$

Improved:

$$e^{12} = \left(e^{\frac{12}{4}}\right)^4 = e^3 * e^3 * e^3 * e^3$$

$$e^3 = \frac{3^0}{0!} - \frac{3^1}{1!} + \frac{3^2}{2!} - \frac{3^3}{3!} ... + \frac{3^{10}}{12!} + ...$$

Second one has smaller numbers to divide

Question

Suppose that you wish to evaluate the function:

$$t(x) = a\exp(3x) + b\exp(2x) + c\exp(x).$$

A
```
t = a*exp(3*x) + b*exp(2*x) + c*exp(x)
```

B
```
z = exp(x)
t = a*z**3 + b*z**2 + c*z + d
```

Question

Suppose that you wish to evaluate the function:

$$t(x) = a\exp(3x) + b\exp(2x) + c\exp(x).$$

On a computer, which is better?

A
```
t = a*exp(3*x) + b*exp(2*x) + c*exp(x)
```

B
```
z = exp(x)
t = a*z**3 + b*z**2 + c*z + d
```

⋆⋆⋆

Solving Equations in x

Solving eqns

Let's consider how to find a specific solution to an equation, a value of x for which $f(x)$ has a desired property.

Methods:

Solving eqns

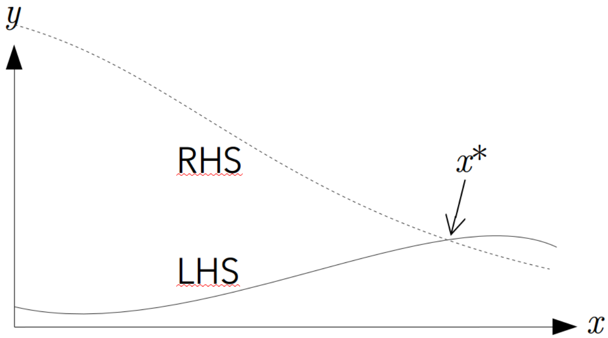
Let's consider how to find a specific solution to an equation, a value of x for which $f(x)$ has a desired property.

Methods:

A. Plot LHS == RHS
B. Newton's method or variant
C. Use scipy.optimize
... (more)

Solving eqns - Plot

The easiest way is to plot LHS v. RHS and find the crossover point:



Solving eqns - Plot

$$x^2 + 5x - (2x^2 - 3) = -2x^2 - x$$

```
x**2 + 5*x - (2*x**2 - 3) == -2*x**2 - x
```

Solving eqns - Plot

$$x^2 + 5x - (2x^2 - 3) = -2x^2 - x$$

```
x**2 + 5*x - (2*x**2 - 3) == -2*x**2 - x
```

```
x = np.linspace( -10,10,1001 )
lhs = x**2 + 5*x - (2*x**2 - 3)
rhs = -2*x**2 - x
plt.plot( x,lhs,'r', x,rhs,'b' )
plt.plot( x,lhs-rhs,'g' )
```

Solving eqns - Plot

$$x^2 + 5x - (2x^2 - 3) = -2x^2 - x$$

```
x**2 + 5*x - (2*x**2 - 3) == -2*x**2 - x
```

```
x = np.linspace( -10,10,1001 )
lhs = x**2 + 5*x - (2*x**2 - 3)
rhs = -2*x**2 - x
plt.plot( x,lhs,'r', x,rhs,'b' )
plt.plot( x,lhs-rhs,'g' )
```

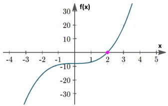
This works, but we need something better than eyeballing it.

Solving eqns - Newton's method

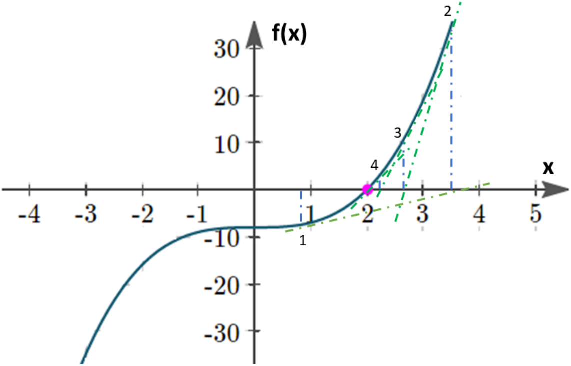
Newton's method uses the function and its derivative to locate the x-value of the zero, x^*.

The trick, of course, is that you need $f'(x) = \frac{d[f(x)]}{dx}$

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$



$\rightarrow$



Solving eqns - Newton’s method

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

```
def dfdx( f,x,h=1e-3 ):
    return ( f( x+h ) - f( x ) ) / h

def newton( f,x0,tol=1e-3 ):
    d = abs( 0 - f( x0 ) )
    while d > tol:
        x0 = x0 - f( x0 ) / dfdx( f,x0 )
        d = abs( 0 - f( x0 ) )
    return ( x0,f( x0 ) )
```

Questions

For

$$\cos x + 2 = x^3 - x^2$$

What are the parameters needed for `newton( f,x0,tol=1e-3 )` to work?

Questions

For

$$\cos x + 2 = x^3 - x^2$$

What are the parameters needed for `newton( f,x0,tol=1e-3 )` to work?

```
def f(x):
    import numpy as np
    return (( np.cos( x ) + 2 ) - ( x**3 - x**2 ))

x0 = any number

newton( f, x0, tol=1e-3 )
```

Solving eqns - scipy.optimize

`import scipy.optimize`

There is a ready-made Newton's method in scipy.optimize

`> scipy.optimize.newton( f,x0 )`

We can also find minima using

`> scipy.optimize.fmin( f,x0 )`.

Solving eqns - scipy.optimize

```
import scipy.optimize
```

There is a ready-made Newton's method in scipy.optimize

```
> scipy.optimize.newton( f,x0 )
```

We can also find minima using

```
> scipy.optimize.fmin( f,x0 ).
```

This requires you to be clever in preparing `f`, you may have to manipulate your function.

Solving eqns - scipy.optimize

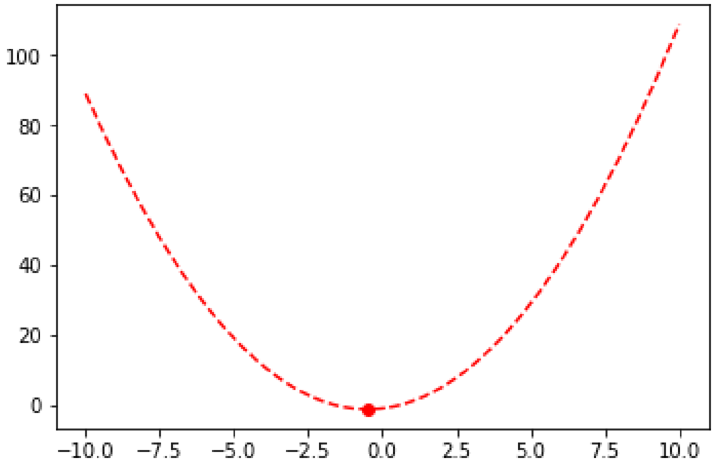
```
import matplotlib.pyplot as plt
import numpy as np
import scipy.optimize

def f( x ):
    return x**2 + x - 1

x = np.linspace( -10,10,1000 )
xstar = scipy.optimize.fmin( f,x0=3 )

plt.plot( x,f( x ),'r--', xstar,f( xstar ),'ro' )
plt.show()
```

Solving eqns - scipy.optimize



Optimization (Preview)

Optimization

On vacation, you purchased a range of n souvenirs of varying weight and value. When it comes time to pack, you find that your bag has a weight limit of 22 kg. What is the best set of items to take on the flight?

Summary

A. Choose the correct way to represent equations

    More function calls $\rightarrow$ slower

    Simple codes are generally faster

B. `import timeit` to time commands

C. Solution methods

    Plotting graphs to find solutions to equations $\rightarrow$ intersections

    Newton's method

    `import scipy.optimize as sco`

    `sco.newton(...)`

    `sco.fmin(...)`

    `sco.minimize(...)` more powerful but complicated than `sco.fmin(...)`