lec9

August 12, 2016

1 Solving linear equations

When we want to solve a hard problem, it is good to start with the simplest possible problem.

So we will start with solving one equation with one variable of the form $ax = b$

In fact, we'll make our life even easier, and so we don't worry about reading the input for now So, our goal is to find a function solve1(a,b) that will solve equations of the form $ax + b = 0$. For example $solve1(2, -4) = 2$

```
In [10]: from __future__ import division

In [20]: def solve1(a,b): #solve xa + b =0
             return -b/a

In [145]: solve1(2,-3)

Out[145]: 1.5
```

Let's now try to solve *two* equations, of the form $ax + by + c = 0, dx + ey + f = 0$ So, $solve2(a, b, c, d, e, f)$ should return a list $[x, y]$ of the solution for x and the solution for y.

The idea is the following: if $a \neq 0$, then we can divide the first equation by a to get an equation of the form $1x + b'y + c' = 0$

Then we can subtract the first equation times d from the second equation, to make the second equation have the form $e'y + f' = 0$.

But then the second equation is only over *one* variable, which we already know how to solve. So, we can get a solution for y, and then plug it into the first equation to get a solution for x.

```
In [18]: def solve2(a,b,c,d,e,f): # solve xa+by+c = 0 , xd+ey+f = 0
             if a: # True if a is not zero
                 # x = (-by-c)/a
                 # d(-by-c)/a+ey+f=0
                 y = solve1(-d*b/a+e,-d*c/a+f)
                 x = (-b*y-c)/a
                 return x,y
             # x = (-ey-f)/d
             # a(-ey-f)/d+by+c=0
             y = solve1(-a*e/d+b,-f*a/d+c)
             x = (-e*y-f)/d
             return x,y
```

```
In [59]: solve2(1,1,-10,1,-1,-4)

Out[59]: (7.0, 3.0)
```

Now we want to solve three equations of the form:
$ax + by + cz + dw + e = 0$
$fx + gy + hz + iw + j = 0$
$kx + ly + mz + nw + o = 0$
We are starting to run out of letters, so we will write this as:
$a_{0,0}x_0 + a_{0,1}x_1 + a_{0,2}x_2 + a_{0,3} = 0$
$a_{1,0}x_0 + a_{1,1}x_1 + a_{1,2}x_2 + a_{1,3} = 0$
$a_{2,0}x_0 + a_{2,1}x_1 + a_{2,2}x_2 + a_{2,3} = 0$
We will represent the input as a *list of lists*:
[
$[a_{0,0}, a_{0,1}, a_{0,2}, a_{0,3}]$,
$[a_{1,0}, a_{1,1}, a_{1,2}, a_{1,3}]$,
$[a_{2,0}, a_{2,1}, a_{2,2}, a_{2,3}]$,
]
Our solution will have the following form:

We will write a function $solve3(eqs)$ that given a list eqs that contains three lists $eqs[0], eqs[1], eqs[2]$ where each of those corresponds to an equation, returns a list of three numbers that is the solution to the equations.

The approach would be as follows:

1. Make sure that the first equation has the first coefficient equal to one. That is, it should be of the form $1x_0 + a'_{0,1}x_1 + a'_{0,2}x_2 + a'_{0,3} = 0$ for some numbers $a'_{0,1}, a'_{0,2}, a'_{0,3}$.
2. Subtract a multiple of this first equation from all the rest of the equations, so that all the rest of the equations have the first coefficient equalling zero
3. Run $solve2$ on the second and third equations with 2 variables to get solution (y, z)
4. Compute $x = -a'_{0,3} - a'_{0,2}z - a'_{0,2}y$
5. Return $[x, y, z]$

```
In [80]: # first solve using "wishful thinking"
         def solve3(eqs):
             make_first_coeff_nonzero(eqs) # make 1st coef of 1st equation nonzero

             eqs[0] = multiply_equation(eqs[0],1/eqs[0][0])
             # make 1st coef of 1st equation equal to 1

             for i in [1,2]:
                 eqs[i] = add_equations(eqs[i],multiply_equation(eqs[0],-eqs[i][0]))
                 # zero out first coefficient in 2nd and 3rd equations

             (y,z) = solve2(eqs[1][1],eqs[1][2],eqs[1][3],
                            eqs[2][1],eqs[2][2],eqs[2][3])
             # solve 2nd and 3rd equations on 2nd and 3rd variables
```

```
        x = -eqs[0][1]*y - eqs[0][2]*z - eqs[0][3]
        # solve 1st variable given solutions for 2nd and 3rd variables

        return (x,y,z)
```

In [110]:
```
def make_first_coeff_nonzero(eqs):
    """Switch order of equations so 1st coef of 1st equation is nonzero"
    if eqs[0][0]:
        return
    if eqs[1][0]:
        eqs[0], eqs[1] = eqs[1] , eqs[0]
        return
    if eqs[2][0]:
        eqs[0] , eqs[2] = eqs[2], eqs[0]
        return
    sys.exit("Oh oh! All first coefficients are zero - can't solve!")
```

In [100]:
```
def multiply_equation(eq,num):
    """Multiply all coefficients of equation eq by number num.
       Return result"""
    res = []
    for x in eq:
        res.append(x*num)
    return res
```

In [101]:
```
def add_equations(eq1,eq2):
    """Add eq1 and eq2. Return result"""
    res = []
    for i in range(len(eq1)):
        res.append(eq1[i]+eq2[i])
    return res
```

In [102]:
```
# recalling the definition of solve3:

def solve3(eqs):
    make_first_coeff_nonzero(eqs)  # make 1st coef of 1st equation nonze
    eqs[0] = multiply_equation(eqs[0],1/eqs[0][0])
    # make 1st coef of 1st equation equal 1

    for i in [1,2]:
        eqs[i] = add_equations(eqs[i],multiply_equation(eqs[0],-eqs[i][0]
    # make 1st coef of 2nd and 3rd equation equal zero

    (y,z) = solve2(eqs[1][1],eqs[1][2],eqs[1][3],eqs[2][1],eqs[2][2],eqs
    # solve 2nd and 3rd equations for 2nd and 3rd variables

    x = -eqs[0][1]*y - eqs[0][2]*z - eqs[0][3]
```

```
            # solve 1st variable using solution for 2nd and 3rd variable

            return (x,y,z)
```

```
In [112]: solve3([ [1,1,1,-6] , [1,1,-1,0], [1,-1,1,-2]])
```

```
Out[112]: (1.0, 2.0, 3.0)
```

2 Labwork

2.0.1 Exercise 1

We wrote a function $solve1(a, b)$ that gets input coefficients for an equation $ax + b = 0$ and outputs a solution for x.

But we can also write an equation in the form $cx = d$. Write a function $other_solve1(c, d)$ that outputs the solution x such that $cx = d$.

Below are some examples for the output of other_solve1

```
In [2]: def other_solve1(c,d):
            return float(d)/float(c)
```

```
In [12]: other_solve1(1.0,1.0)
```

```
Out[12]: 1.0
```

```
In [13]: other_solve1(1.0,2.0)
```

```
Out[13]: 2.0
```

```
In [14]: other_solve1(2.0,1.0)
```

```
Out[14]: 0.5
```

```
In [15]: other_solve1(3.0,-2.0)
```

```
Out[15]: -0.6666666666666666
```

2.0.2 Exercise 2

Write a similar function for solving 2 equations in 2 variables. $other_solve2(a, b, c, d, e, f)$ should output two numbers x, y such that $ax + by = c$ and $dx + ey = f$.

Below are some examples for the output of $other_solve2$

```
In [16]: def other_solve2(a,b,c,d,e,f):
             return  solve2(a,b,-c,d,e,-f)
```

```
In [21]: other_solve2(1.0,1.0,3.0,1.0,-1.0,1.0)
```

```
Out[21]: (2.0, 1.0)
```

```
In [22]: other_solve2(0.0,1.0,2.0,2.0,3.0,10.0)
```

```
Out[22]: (2.0, 2.0)
```

2.0.3 Exercise 3

There are some inputs that "breal" the $solve2$ function we saw in class. That is, there are some examples of equations it will not be able to solve and will output an error instead. Can you find such an example?

That is, find 6 numbers a, b, c, d, e, f such that $solve2(a, b, c, d, e, f)$ will result in an error

2.0.4 Exercise 4

Write a function $solve4(eqs)$ that solves *four* equations in *four* variables. The function will get a list of 4 equations, each of them a list of 5 numbers which correspond to the coefficients of an equation in variables x_0, x_1, x_2, x_3 of the form $a_0x_0 + a_1x_2 + a_2x_2 + a_3x_3 + a_4 = 0$.

That is, $solve4$ gets as input a list `eqs` such that $\texttt{eqs} = [\texttt{eq0}, \texttt{eq1}, \texttt{eq2}, \texttt{eq3}]$.
Each one of the $\texttt{eq}i$'s is itself a list of 5 numbers corresponding the coefficients $a_{i,0}, a_{1,1}, \ldots, a_{i,4}$ in the equation $a_{i,0}x_0 + a_{i,1}x_1 + a_{i,2}x_2 = a_{i,3}x_3 + a_{i,4} = 0$.

The function should return (x_0, x_1, x_2, x_3): the solution for the four variables.

For example:

```
In [114]: solve4([ [1,1,1,1,-10] , [1,1,1,-1,-2], [1,1,-1,1,-4], [1,-1,1,1,-6] ])

Out[114]: [1.0, 2.0, 3.0, 4.0]
```