Understanding the extended Euclidean algorithm

The extended Euclidean algorithm is essentially the same as the Euclidean algorithm to find the `gcd`, the greatest common divisors of two integers `a` and `b`.

However, it takes advantage of information that is generated in running the standard Euclidean algorithm, but which is normally discarded, in order to compute multiplicative inverses in modular arithmetic.

I found a nice page about this topic here.

multiplicative inverses

Consider the multiplication table for two fields, one a prime number and one not prime.

```
    0   1   2   3   4
0   0   0   0   0   0
1   0   1   2   3   4
2   0   2   4   1   3
3   0   3   1   4   2
4   0   4   3   2   1
```

Above is the table for `Z5`. Notice that in each case we generate all of the members of `Z5` by one multiplication. `1` and `4` are each their own multiplicative inverses, while `2` and `3` are also inverses.

On the other hand, for `Zn` where `n` is not prime, we might see:

```
    0   1   2   3   4   5   6   7
0   0   0   0   0   0   0   0   0
1   0   1   2   3   4   5   6   7
2   0   2   4   6   0   2   4   6
3   0   3   6   1   4   7   2   5
4   0   4   0   4   0   4   0   4
5   0   5   2   7   4   1   6   3
6   0   6   4   2   0   6   4   2
7   0   7   6   5   4   3   2   1
```

Notice that when `z = 8`, the numbers which are prime: `3, 5, 7`, generate all of the elements of `Z`. The others do not, and furthermore, they never generate `1`. Thus `4` mod `8` has no multiplicative inverse, while `7 x 7` equals `1` mod `8`.

Extended Euclidean algorithm

I want to demonstrate the EEA but first I'd like to find a pair of numbers for which the `gcd` is equal to `1` and yet takes a few steps, starting from smallish numbers.

Let's try 231 (3 x 7 x 11), and 130 (2 x 5 x 13) :

```
  a    b    r   q
231  130  101   1
130  101   29   1
101   29   14   3
 29   14    1   2
 14    1    0  14
```

That looks reasonable. The `gcd` is the value of `b` when `r == 0`, that is, `1`.

Now, rearrange the data to be equations of the form `r = a - qb`.

```
101 = 231 - (1)130
 29 = 130 - (1)101
 14 = 101 - (3)29
  1 =  29 - (2)14
  0 =  14 - (1)14
```

backward

There are two related methods: forward and backward. The backward method starts with the next to last equation:

```
1 = 29 + (-2)14
```

Substitute for `14` from the previous equation

```
1 = 29 + (-2)[101 - 3(29)]
1 = (-2)101 + (7)29
```

Next, substitute for `29`:

```
1 = -(2)101 + (7)[130 - (1)101]
1 = (7)130 + (-9)101
```

At each stage we generate a true equality, always the larger number has two terms to be combined. And the signs stay with the terms: all terms with `(101)` will be negative, for example.

Then finally, substitute for `101`

```
1 = (7)130 + (-9)[231 - (1)130]
1 = (-9)231 + 16(130)
```

We can confirm that the arithmetic is correct at every stage.

We have expressed the `gcd` as a *linear combination* of `a` and `b`. The form is

```
gcd = xa + yb
```

Now, realize that if we do `mod 231` on both sides we have:

```
16(130) mod 231 = 1
```

`16` and `130` are modular multiplicative inverses mod `231`. We can check this easily:

```
>>> 16 * 130 % 231
1
```

Here we used four equations and `b` ended up with a positive factor. An odd number of equations would give a negative factor `y` in `yb`. In that case we take take the modulus of `y` by adding `a` to it.

forward

Here are the equations again for reference.

```
101 = 231 - (1)130
 29 = 130 - (1)101
 14 = 101 - (3)29
  1 =  29 - (2)14
  0 =  14 - (1)14
```

In thinking about this, forget about the fact that the algorithm is constantly switching `a` for `b` and `b` for `r`. Here, `a` and `b` retain their initial values.

Start with the first equation:

```
101 = 231 - (1)130
    = a - b
```

The next line is

```
29 = 130 - (1)101
   = b + (-1)(a - b)
   = (-1)a + 2b
```

And the next:

```
14 = 101 - (3)29
   = (a - b) + (-3) [(-1)a + (2)b]
   = 4a - 7b
```

Finally,

```
1 = 29 + (-2)14
  = (-1)a + 2b + (-2)[4a - 7b]
  = -9a + 16b
```

These are the same values for `x` and `y` that we had from the backwards method.

The challenge now is to convert these to Python code.