Cubic with Two Factorable Derivatives

Generic testing link.

The Problem

Often we want to have a polynomial whose first and second derivative are suitably "nice" for students in order to allow them to factor the polynomial itself, as well as its first and second derivatives, in order to find zeros for things like graphing. Unfortunately, it turns out, that creating a polynomial that is factorable, and whose first and second derivative are also factorable, is a *highly* nontrivial problem. Here we provide code that works (and generates *relatively* nice polynomials as well) to generate such a polynomial that is a cubic function (this also helps avoid requiring something like the rational root theorem for factoring the original polynomial - in this case the cubic is always factorable by grouping as well).

The Result

The original functions is $f(x) = 8\,x^3 - 7\,x^2 - 5\,x + \frac{35}{8}$ which should factor into:

$$\frac{1}{4}\left(2\,x + \sqrt{\frac{5}{2}}\right)\left(2\,x - \sqrt{\frac{5}{2}}\right)(8\,x - 7)$$

The first derivative is $f'(x) = 24\,x^2 - 14\,x - 5$ which should factor into

$$(4\,x + 1)\cdot(6\,x - 5)$$

The second derivative is $f''(x) = 48\,x - 14$ which is linear so it factors trivially (Yes, we kind of cheat here).

The Magic

Here is the sagecode that generates the above problem:

```
\begin{sagesilent}
def RandInt(a,b):
    """ Returns a random integer in [‘a‘,‘b‘]. Note that ‘a‘ and ‘b‘ should be integers themselves to avoid u
    """
    return QQ(randint(int(a),int(b)))
    # return choice(range(a,b+1))

def NonZeroInt(b,c, avoid = [0]):
    """ Returns a random integer in [‘b‘,‘c‘] which is not in ‘av‘.
        If ‘av‘ is not specified, defaults to a non-zero integer.
    """
    while True:
        a = RandInt(b,c)
        if a not in avoid:
            return a



## Start with a while loop to make sure the result has reasonable coefficients, at least in general size.
p1f2 = 9999*x^3 + 9999*x^2 + 9999*x + 9999
```

```
while ((abs(p1f2.coefficient(x^3))>100) or (abs(p1f2.coefficient(x^2))>100) or (abs(p1f2.coefficient(x))>100)
    ### We start by taking a product of factors to get a factorable first derivative.

    # Make sure the leading coefficient is divisible by 3, which will help ensure the numbers stay nice(ish)
    p1c1 = 1
    p1c3 = 1
    while mod(p1c1*p1c3,3)>0:
        p1c1 = NonZeroInt(-5,5)
        p1c2 = NonZeroInt(-5,5)
        p1c3 = RandInt(1,6)
        p1c4 = -sign(p1c1)*sign(p1c2)*RandInt(1,5)# Rigged sign to make sure we get a difference of squares i


    p1fact1 = p1c1*x-p1c2
    p1fact2 = p1c3*x-p1c4

    p1f1 = expand(p1fact1*p1fact2)



    ### Now we make the original function by integrating, then finding an appropriate ''C'' to add to make it

    p1f2temp = integral(p1f1,x)

    # Now, we assume we are going to factor by grouping, to be kind, so we extract the necessary constant we
    p1c5 = p1f2temp.coefficient(x^2)*p1f2temp.coefficient(x)/p1f2temp.coefficient(x^3)

    p1f2 = p1f2temp+p1c5

    if p1f2.coefficient(x^3)*p1f2.coefficient(x)<0:
        p1sqrtval = p1f2.coefficient(x)/p1f2.coefficient(x^3)
        p1f2fact1a = x - sqrt(abs(p1sqrtval))
        p1f2fact1b = x + sqrt(abs(p1sqrtval))
        p1f2fact1 = -sign(p1sqrtval)*(p1f2fact1a)*(p1f2fact1b)#(-sqrt(-p1f2.coefficient(x^3))*x + sqrt(p1f2.c
        p1f2fact2 = p1f2.coefficient(x^3)*x + p1f2.coefficient(x^2)
        p1f1zero1 = -p1f2fact1a(x=0)/p1f2fact1a.coefficient(x)
        p1f1zero2 = -p1f2fact1b(x=0)/p1f2fact1b.coefficient(x)
        p1f1zero3 = -p1f2fact2(x=0)/p1f2fact2.coefficient(x)
    else:
        p1f2fact1 = p1f2.coefficient(x^3)*x^2 + p1f2.coefficient(x)
        p1f2fact2 = x + p1f2.coefficient(x^2)/p1f2.coefficient(x^3)
        p1f1zero1 = 0
        p1f1zero2 = 0
        p1f1zero3 = -p1f2fact2(x=0)/p1f2fact2.coefficient(x)

    p1f2check = expand(p1f2fact1*p1f2fact2)



    ### To get the second derivative function we can take the derivative of the original - which must be line

    p1f3 = derivative(p1f1,x)

\end{sagesilent}
```