lecture17

March 30, 2021

1 Series Solutions With Python

We are going to see how we can use Python to graph solutions (or approximations to solutions) of ODEs. The example we are using is:

$$(1-t^2)x''(t) - 2tx'(t) + \alpha(\alpha+1) = 0.$$

Solutions can be expanded in a power series centered at zero on the interval $(-1,1)$. We can write this as:

$$x(t) = a_0 \sum_{n=0}^{\infty} a_{2n} t^{2n} + a_1 \sum_{n=0}^{\infty} a_{2n+1} t^{2n+1} =: a_0 x_E(t) + a_1 x_O(t).$$

Recall that the relation that the coefficients satisfies is:

$$a_{k+2} = \frac{k(k+1) - \alpha(\alpha+1)}{(k+2)(k+1)} a_k.$$

Writting this interms of k instead of $k+2$ gives:

$$a_k = \frac{(k-2)(k-1) - \alpha(\alpha+1)}{k(k-1)} a_{k-2}$$

This is not a relation that is easy to solve for and we will use python to compute the coefficients. The first thing we will do is to write a function that computes the coefficients in terms of $\alpha, x(0), x'(0)$. As you can see, this is kind of diffucult to reat, but it's doable.

[1]:
```
import sympy as sp

def A(k):
    a = sp.symbols('a')
    x0=sp.symbols('x0')
    x0p=sp.symbols('x0p')
    if k==0:
        return x0
    if k==1:
        return x0p
    return (((k-2)*(k-1) - a*(a+1))/((k-2)*(k)))*A(k-2)

print(A(20))
```

```
zoo*a*x0*(a + 1)*(-a*(a + 1)/8 + 3/4)*(-a*(a + 1)/24 + 5/6)*(-a*(a + 1)/48 +
7/8)*(-a*(a + 1)/80 + 9/10)*(-a*(a + 1)/120 + 11/12)*(-a*(a + 1)/168 +
13/14)*(-a*(a + 1)/224 + 15/16)*(-a*(a + 1)/288 + 17/18)*(-a*(a + 1)/360 +
19/20)
```

So, maybe the symbolic function has some uses, but it seems to be complicated here. Actually, it's really just the fact that α (`a` in the code) is symbolic that is causing problems. In the code below, we keep `x0` and `x0p` as symbolic variables, but define `a` as a literal number.

In this case, we can also get python to write out a polynomial. Of course, the code below doesn't print the polynomial in the nicest format – and you can always do "easy" things to make this polynomial look better.

[2]:
```
def A(k):
    a = 3
    x0=sp.symbols('x0')
    x0p=sp.symbols('x0p')
    if k==0:
        return x0
    if k==1:
        return x0p

    return (((k-2)*(k-1) - a*(a+1))/((k-1)*(k)))*A(k-2)

poly = ""
for k in range(0,10):
    poly = poly + f"({A(2*k+1)})t^{2*k+1}+"
print(poly)
```

```
(x0p)t^1+(-1.66666666666667*x0p)t^3+(0)t^5+(0)t^7+(0)t^9+(0)t^11+(0)t^13+(0)t^15
+(0)t^17+(0)t^19+
```

This is good, but if we want to have Python plot stuff, we need to have literal values for `x0` and `x0p`. We do that below:

[3]:
```
def A(k):
    a = 3
    x0=1
    x0p=1
    if k==0:
        return x0
    if k==1:
        return x0p

    return (((k-2)*(k-1) - a*(a+1))/((k-1)*(k)))*A(k-2)

#And odd polynomial
poly = ""
for k in range(0,10):
```

```
    poly = poly + f"({A(2*k+1)})t^{2*k+1}+"
print(poly)
```

```
(1)t^1+(-1.6666666666666667)t^3+(-0.0)t^5+(-0.0)t^7+(-0.0)t^9+(-0.0)t^11+(-0.0)t
^13+(-0.0)t^15+(-0.0)t^17+(-0.0)t^19+
```

As we saw in class, if $\alpha = 2n$ is even, $x_E(t)$ will be an $2n$ degree polynomial and if $\alpha = 2n+1$ is odd, then $x_O(t)$ will be a polynomial of degree $2n+1$.

We need to fix the function a little and make it so that we can pass in the arguments for `a`, `x0`, `x0p`. This is just basic good programming practice.

[4]:
```
def A(k, a, x0,x0p):
    if k==0:
        return x0
    if k==1:
        return x0p

    return (((k-2)*(k-1) - a*(a+1))/((k-1)*(k)))*A(k-2, a, x0, x0p)

poly = ""
for k in range(0,10):
    poly = poly + f"({A(2*k+1, 3, 1, 1)})t^{2*k+1}+"
print(poly)
```

```
(1)t^1+(-1.6666666666666667)t^3+(-0.0)t^5+(-0.0)t^7+(-0.0)t^9+(-0.0)t^11+(-0.0)t
^13+(-0.0)t^15+(-0.0)t^17+(-0.0)t^19+
```

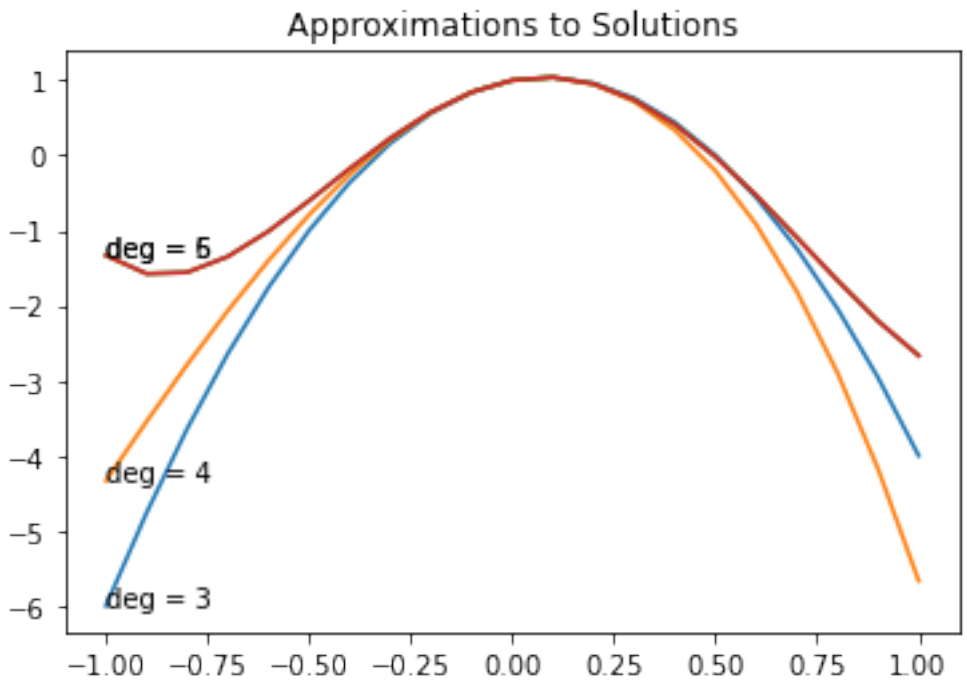
Below, we plot the solution for the parameters $a = 3$, $x(0) = 1$ and $x'(0) = 1$. We plot degree $3, 4, 5, 6$ polynomials. Each of these is an approximation to the actual solution.

[5]:
```
import numpy as np
import matplotlib.pyplot as plt

a = 3
x0 = 1
x0p = 1

h=.1
T = np.arange(-1,1+h,h)
plt.figure()
plt.title('Approximations to Solutions')

for deg in range(3,7):
    X = np.zeros(T.size)
    for k in range(0,deg):
        X = X + A(k,a,x0,x0p)*(T**k)
    plt.plot(T,X)
    plt.annotate(f"deg = {deg}", xy=(-1,X[0]))
```

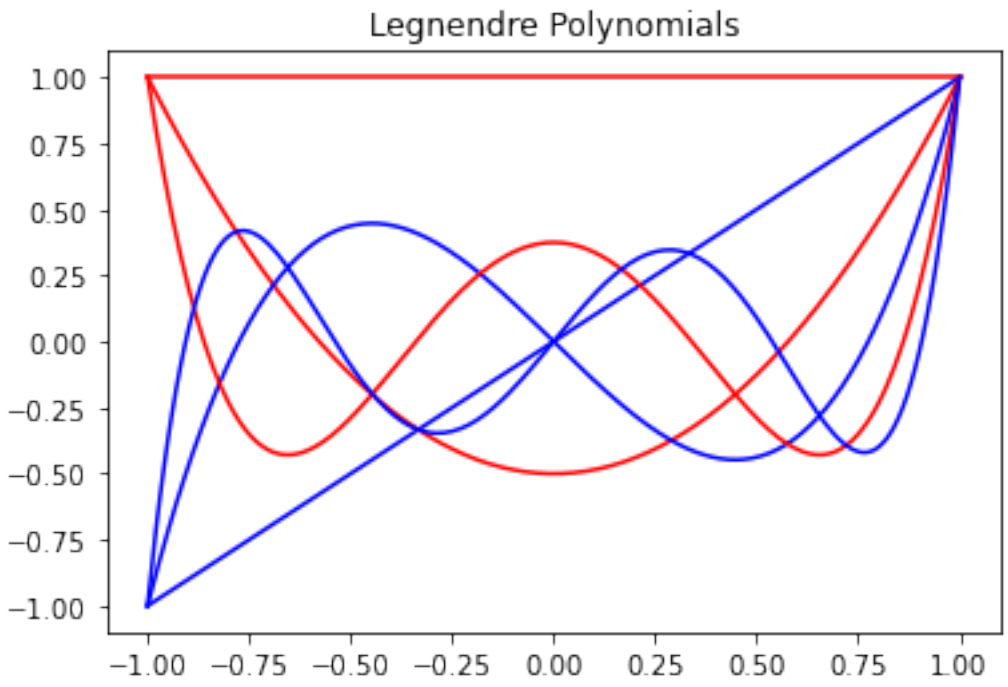


The *Legendre Polynomial of degree* n is the polynomial solution to the equation with $\alpha = n$ and with $P(1) = 1$. We will denote these as $P_n(t)$ for now. Graph $P_0, \ldots, P_5$ below. To normalize, we will divide `X` by `X[-1]` since `x[-1]` is the last entry of `X` and this corresponds to $x(1)$.

[6]:
```
h=.01
T = np.arange(-1,1+h,h)
plt.figure()
plt.title('Legnendre Polynomials')




for n in range(0,3):
    Xe = np.zeros(T.size)
    Xo = np.zeros(T.size)
    ae = 2*n
    ao = 2*n + 1
    for k in range(0,ao+1):
        Xe = Xe + A(k, ae, 1, 0)*(T**k)
        Xo = Xo + A(k, ao, 0, 1)*(T**k)
    Xe = Xe/Xe[-1]
    Xo = Xo/Xo[-1]
    plt.plot(T,Xe, 'r-')
    plt.plot(T,Xo, 'b-')
```


Legnendre Polynomials
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

[ ]: