F210 Programacion Aplicada a Finanzas
Polinomios de Taylor

1. Vamos ahora a *extender* la clase de polinomios poly con una *subclase* Taylor, que computará para una función cualquiera $f : R \to R$ el polinomio de Taylor de grado N asociado, $P(x, x_0)$, alrededor un punto x_0 dado

$$P(x, x_0) = f(x_0) + f'(x_0)(x - x_0) + f"(x_0)\frac{(x - x_0)^2}{2!} + ... + f^N(x_0)\frac{(x - x_0)^N}{N!} \quad (1)$$

Los atributos de instancia de clase deberían ser

- `fT` – con la función f a aproximar –,
- `N` `--`grado del polinomio `--`,
- `x0` `--` punto de evaluacion –,
- `feval` `--`lista con todos los valores de f a utilizar en el cómputo de las derivadas, $f(x_0 + (N - 2i)h)$, con $0 \leq i \leq N$ –,
- `fprime` `--` lista con los valores de las derivadas numericas desde orden 0 hasta orden N – ,
- `h` con el incremento a usar en las derivadas – por defecto use 0.01 –,
- `prtTaylor` – variable boolean que indica con `True` si al invocar `print(a)` imprimirá el polinomio en la forma en 1 , y en caso contrario imprimirá el polinomio usando la función `__str__` de la superclase `poly`.
- `digits` entero conteniendo lo digitos a usar para los coeficientes al momento de invocar `print(a)`

Los métodos de instancia de clase deben contener **exclusivamente**

- El constructor `__init__(self,f,N,x_0)`.h es el incremento default que se utilizará para computar las derivadas numéricas.
- Una versión `__str__(self)` donde si el atributo de instancia de clase `prtTaylor` es `True` escriba el polinomio en la forma descripta en 1, y en caso contrario lo imprima usando el `__str__` de la superclase poly
- Un derivador numérico `derivada_n(self,n)` de funciones que devuelva el valor de la derivada numérica centrada de orden n que dedujimos en clase, y se valga de los resultados aportados por el evaluador del punto anterior

$$f^n(x_0) = \frac{\sum_{i=0}^{n} (-1)^i \binom{n}{i} f(x_0 + (n - 2i)h)}{(2h)^n}$$

Para calcular los numeros combinatorios $\binom{n}{i}$, construya una función auxiliar recursiva que compute el problema a partir del triangulo de Pascal utilizando la propiedad $\binom{n}{i} = \binom{n-1}{i-1} + \binom{n-1}{i}$ y el hecho que $\binom{n}{n} = \binom{n}{0} = 1$ – casos base –

- Una funcion auxiliar `get_parms(self)` que a partir de los atributos de instancia de subclase devuelva la lista de coeficientes que correspondería al polinomio si éste fuera escrito para inicializarse como una instancia de `poly` , lista que debe ser usada durante el `__init__` method de la superclase `poly` via el superobjeto `super`:

  ```
  super(Taylor,self).__init__(self.get_parms())
  ```

  (ver uso de superobjeto en el capitulo 5 de Python in a Nutshell)

2. Encuentre matematicamente y computacionalmente la serie de Taylor de orden 5 para las siguientes funciones:

   - $f(x) = e^x$ alrededor de $x_0 = 0$. Compare los resultados obtenidos para los coeficientes. Pruebe usando valores de $h = 0.1, 0.01, 0.001$.Compare la gráfica del polinomio con la gráfica de la funcion. Si usted quiere tener una buena aproximación de e^x entre -10 y 10, busque haciendo prueba y error de que orden deberia ser el polinomio
   - $f(x) = \ln(1+x)$ alrededor de $x_0 = 0$. Verifique que para valores de x cercanos a cero obtiene $\ln(1+x) \simeq x$. Compare los resultados obtenidos para los coeficientes. Pruebe usando valores de $h = 0.1, 0.01, 0.001$.Compare la gráfica del polinomio con la gráfica de la funcion. Si usted quiere tener una buena aproximación de $\ln(1+x)$ entre -0.5 y 0.5, busque haciendo prueba y error de que orden deberia ser el polinomio.
   - $f(x) = \sin(x)$ alrededor de $x_0 = 0$. Trate de buscar una explicacion de porque solo sobreviven los terminos impares de la serie de Tayor. Compare la gráfica del polinomio con la gráfica de la funcion. Si usted quiere tener una buena aproximación de $\sin(x)$ entre -π y π, busque haciendo prueba y error de que orden deberia ser el polinomio.

     Con ese polinomio, muestre computacionalmente usando las operaciones de polinomios que

     $$\lim_{x\to 0} \frac{\sin(x)}{x} = 1$$