Cómputo científico para probabilidad y estadística. Tarea 2. Descomposición QR y mínimos cuadrados.

Juan Esaul González Rangel

Septiembre 2023

1. Implementar el algoritmo de Gram-Schmidt modificado 8.1 del Trefethen (p. 58) para generar la descomposición QR.

   En el archivo `QR.py`, el algoritmo está implementado en la función `QR`.

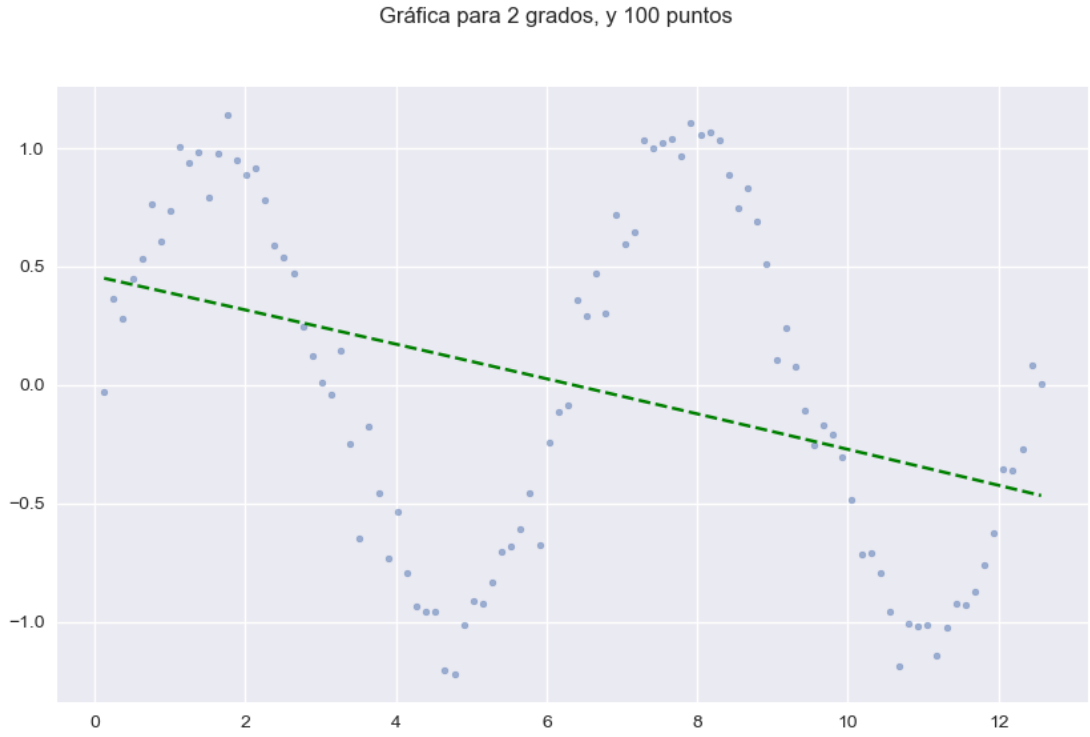
2. Implementar el algoritmo que calcula el estimador de mínimos cuadrados en una regresión usando la descomposición QR.

   En el archivo `QR.py`, el algoritmo está implementado en la función `LSQR`.

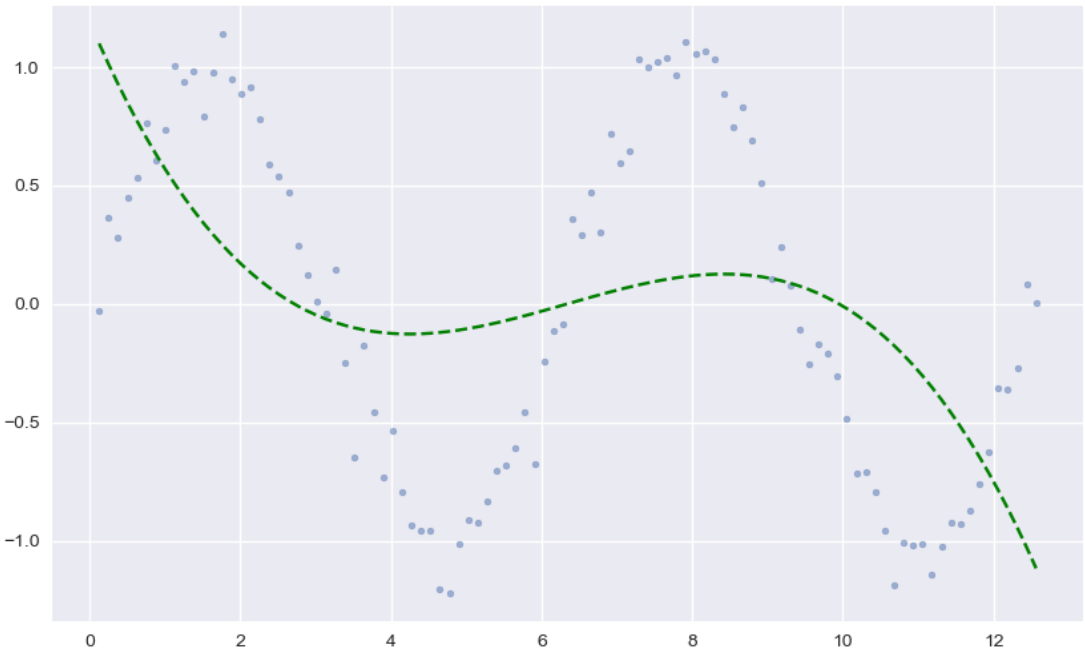
3. Generar $\mathbf{Y}$ compuesto de $y_i = \text{sen}(x_i) + \varepsilon_i$ donde $\varepsilon_i \sim N(0, \sigma)$ con $\sigma = 0.11$ para $x_i = \frac{4\pi i}{n}$ para $i = 1, \ldots, n$.

   Hacer un ajuste de mínimos cuadrados a $\mathbf{Y}$, con descomposición QR, ajustando un polinomio de grado $p - 1$.

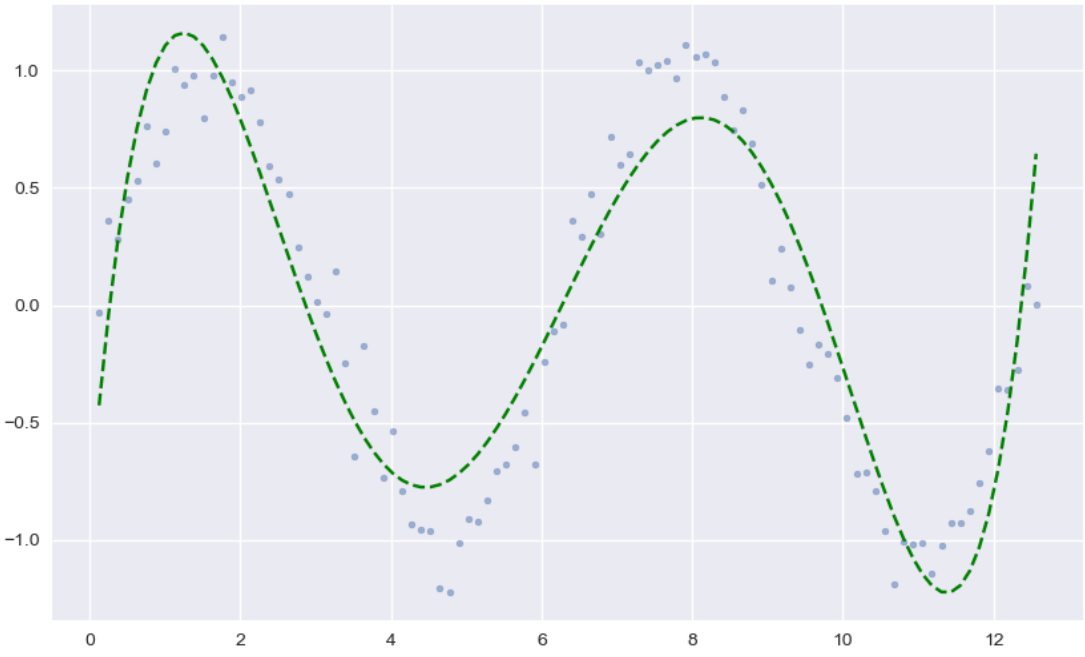
   - Considerar los 12 casos: $p = 3, 4, 6, 100$ y $n = 100, 1000, 10000$.
     En el archivo `.py` está implementada la regresión polinómica para los valores dados.
   - Graficar el ajuste en cada caso.
     Las siguientes gráficass muestran el ajuste obtenido en cada caso.



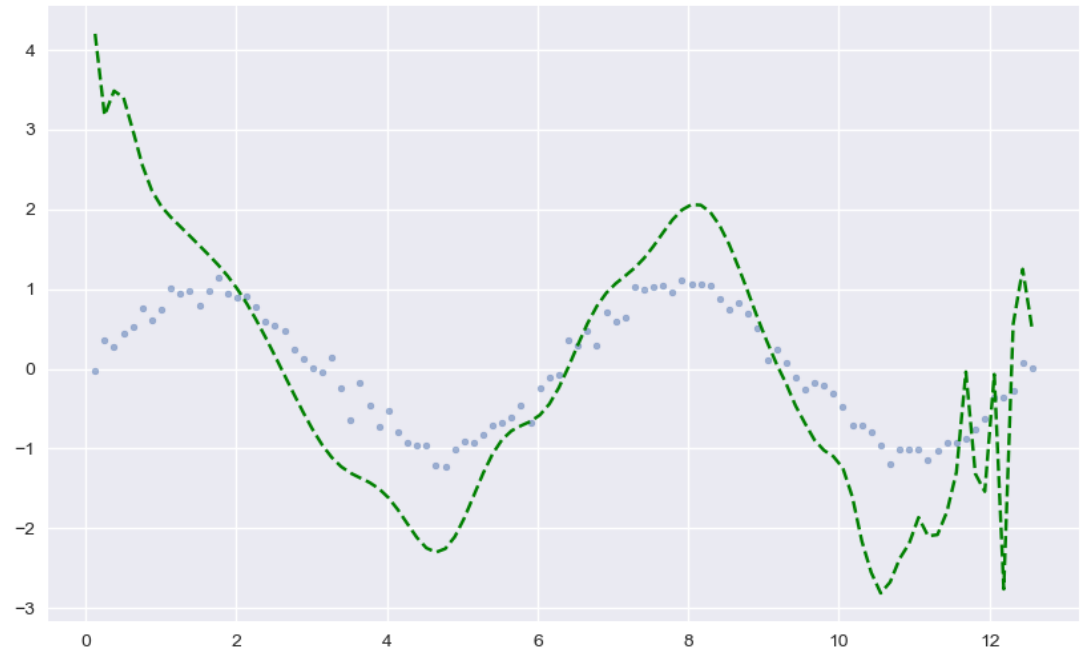
Gráfica para 3 grados, y 100 puntos



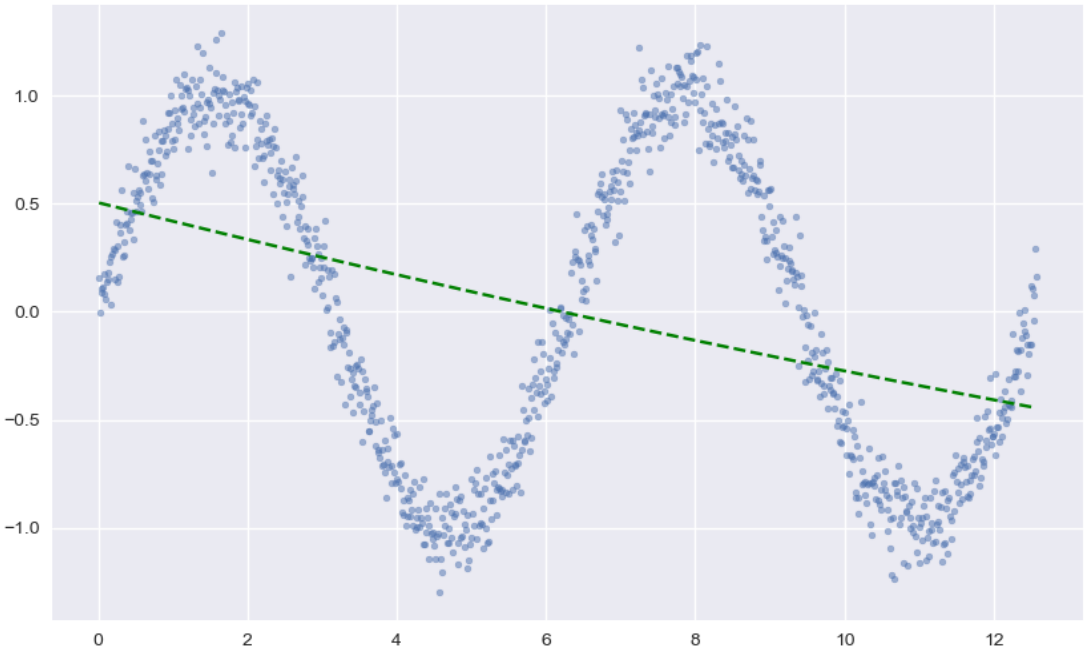
Gráfica para 5 grados, y 100 puntos



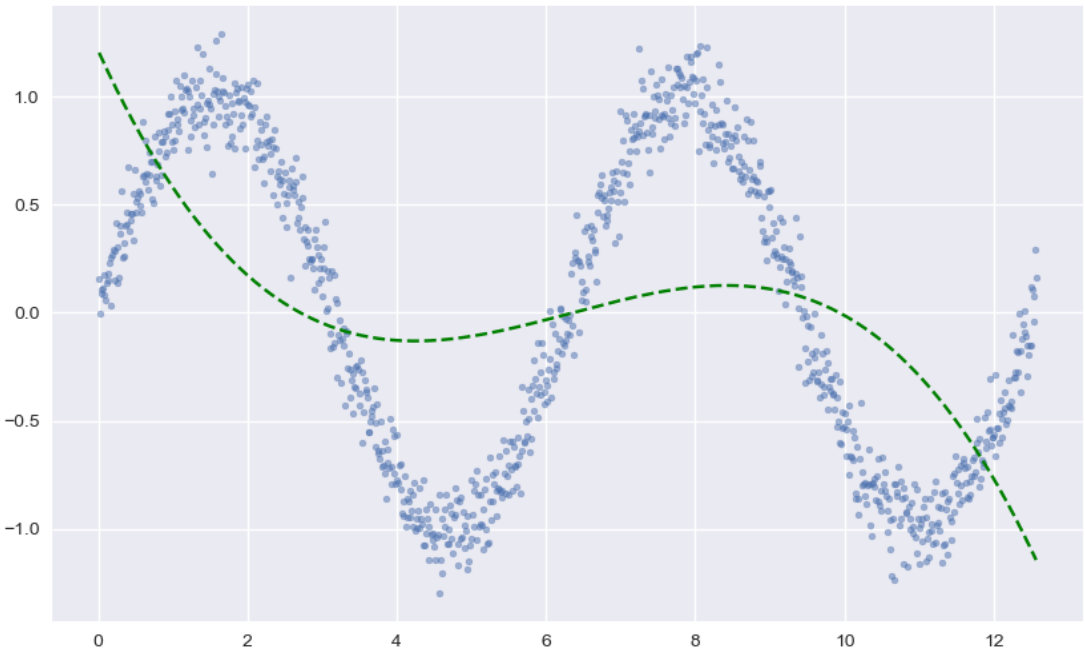
Gráfica para 99 grados, y 100 puntos



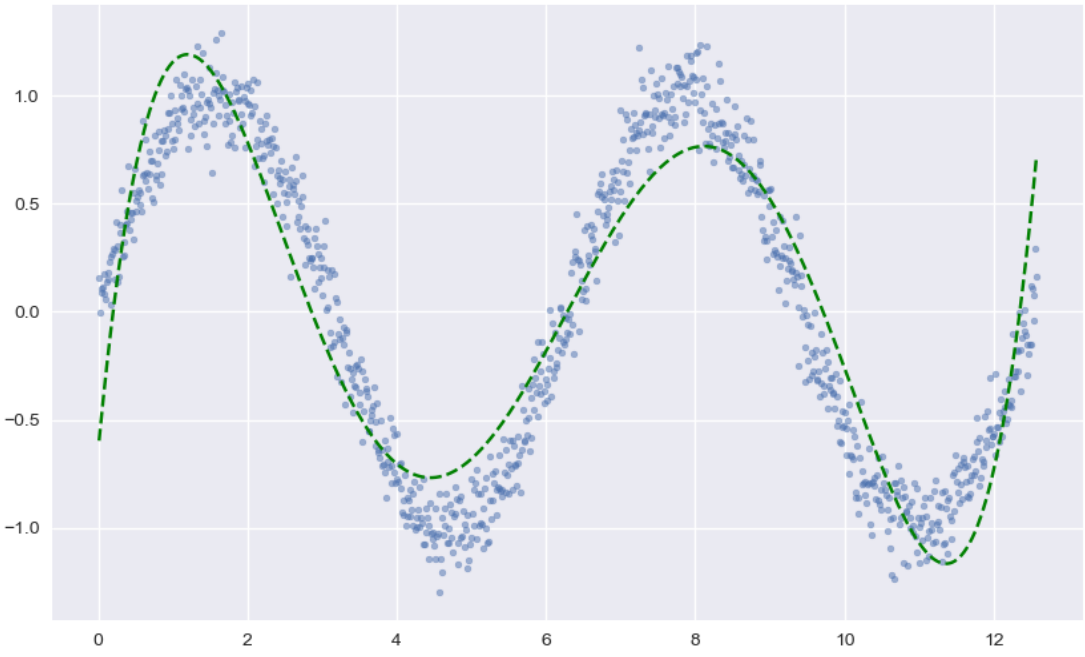
Gráfica para 2 grados, y 1000 puntos



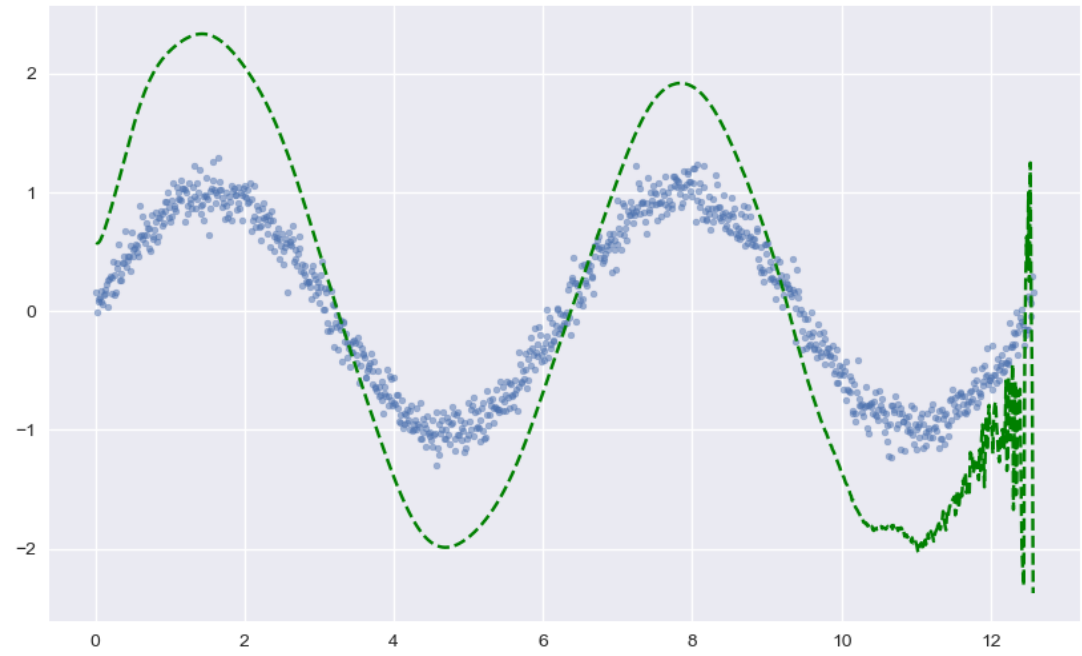
Gráfica para 3 grados, y 1000 puntos



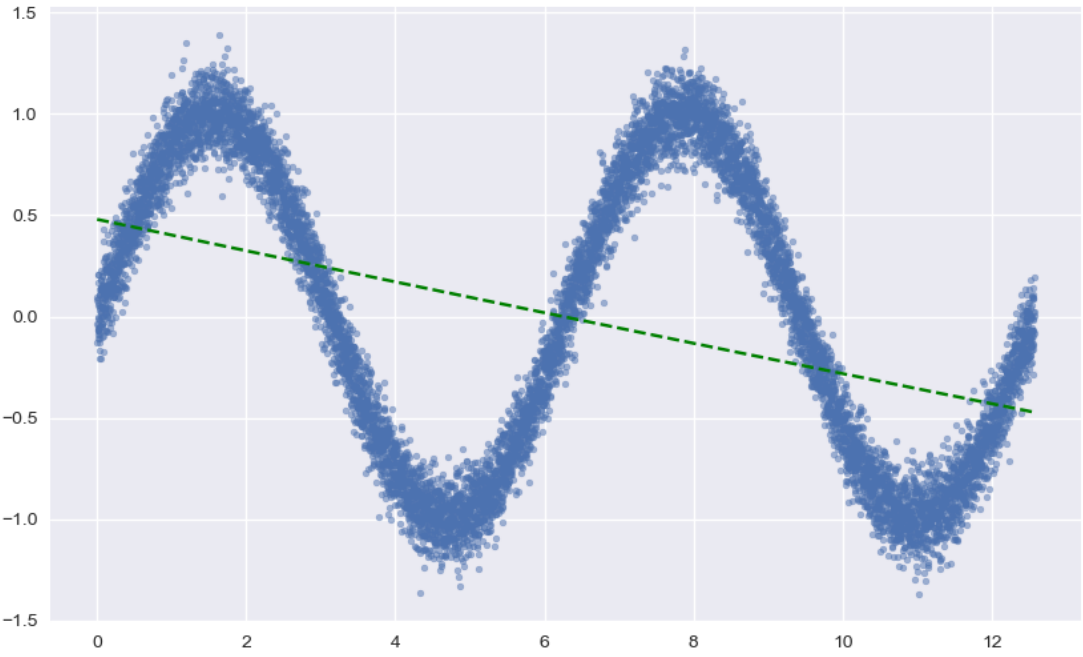
Gráfica para 5 grados, y 1000 puntos



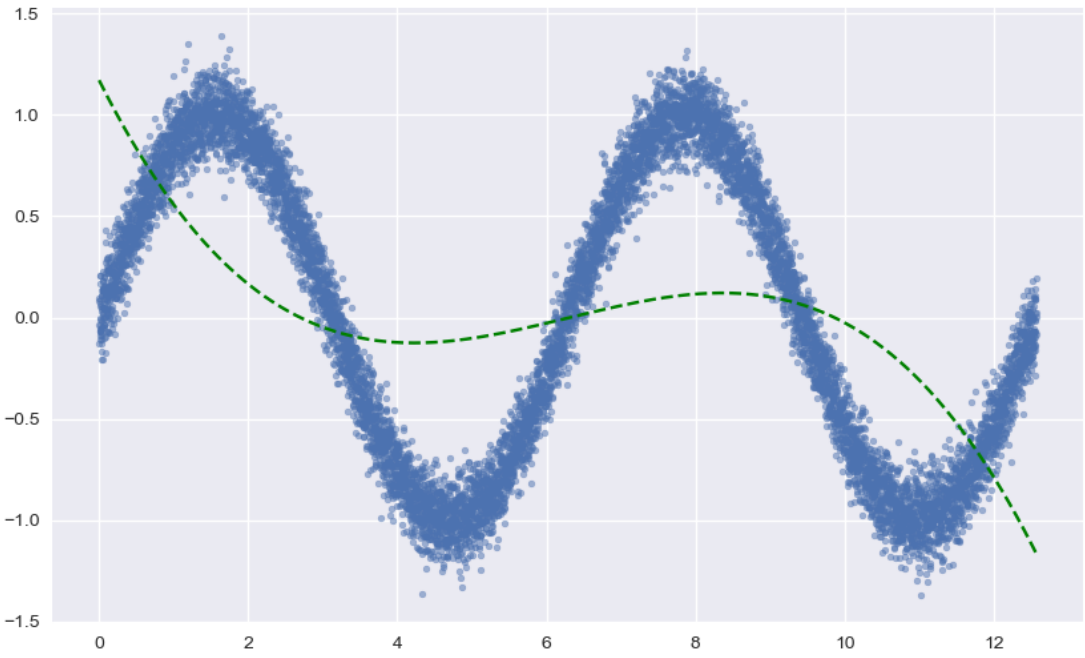
Gráfica para 99 grados, y 1000 puntos



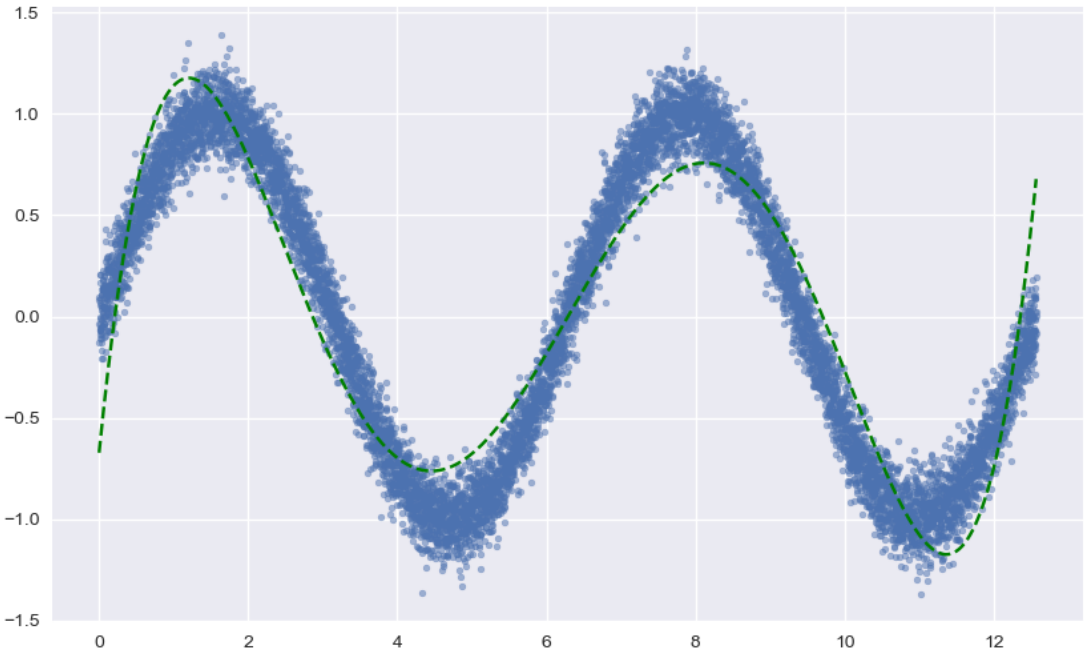
Gráfica para 2 grados, y 10000 puntos

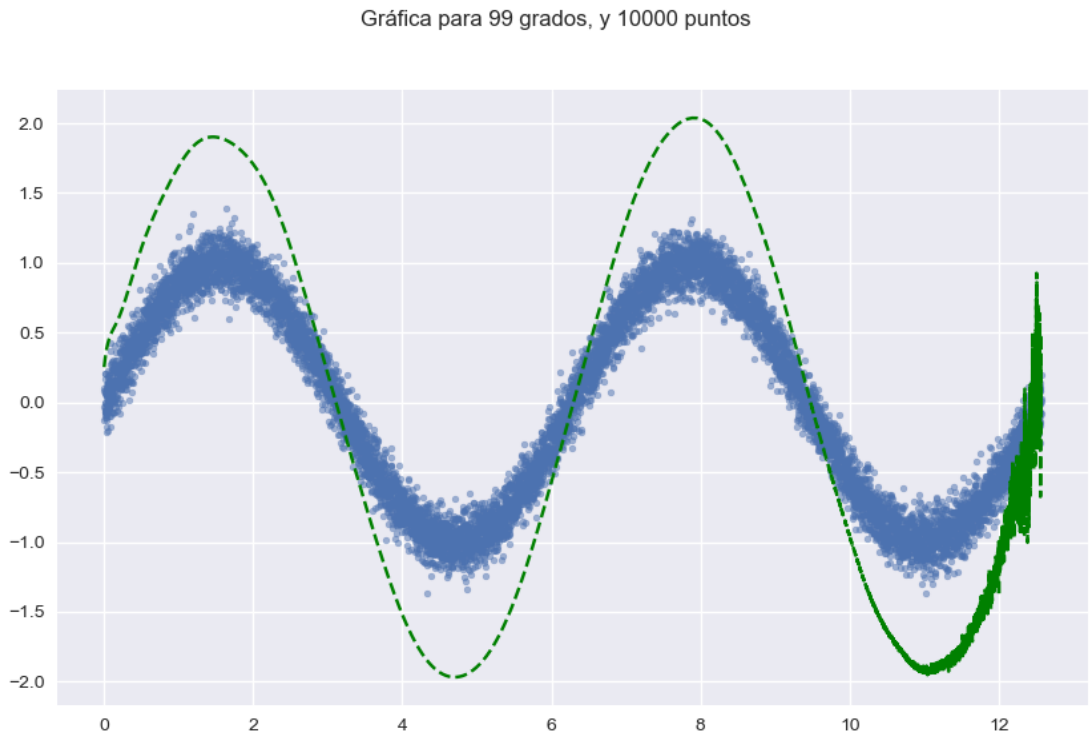


Gráfica para 3 grados, y 10000 puntos



Gráfica para 5 grados, y 10000 puntos



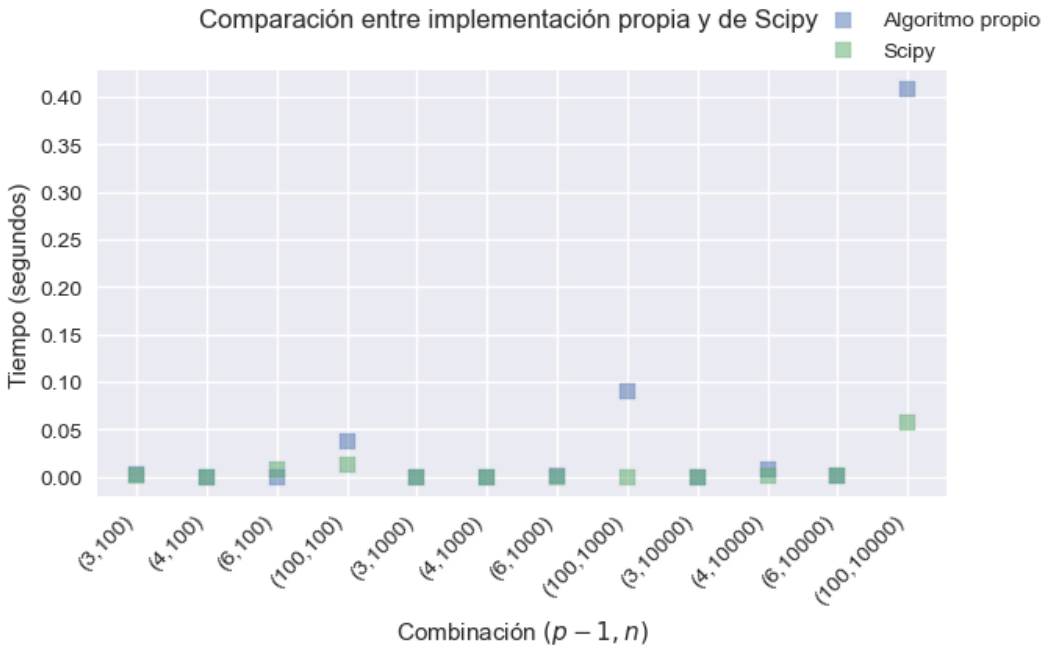


Como podemos observar, el mejor ajuste para cualquier cantidad de puntos se logra cuando se utiliza un polinomio de grado 5, para grados menores, la curva ajustada resulta muy insuficiiente, y para grados mayores, tiende a presentar demasiadas oscilaciones, lo que indica un comportamiento irregular que está más influido por el ruido.

Cabe resaltar también que cuando aumenta el grado del polinomio de manera considerable, la precisión del ajuste decrece al punto de que los límites de la función se mueven de $(-1, 1)$ a $(-2, 2)$. Esto puede tener su causa en que con una cantidad de columnas tan grande, el número de cndición de la matriz crece considerablemente, llegando a ser en uno de los casos del orden de 10^{111}, sin embargo, cuando aumenta el número de filas manteniendo el número de columnas relativamente pequeño, el número de condición parece crecer más lentamente, teniendo para una matriz (10000×6) un número de condición del orden de 100^6.

- Medir tiempo de ejecución de su algoritmo, comparar con descomposición QR de `scipy` y graficar los resultados.

  En la siguiente figura tenemos una comparación entre los tiempos de ejecución del algoritmo propio y el algoritmo de `scipy` para la descomposición QR. Podemos notar que para vaalores pequeños de $p - 1$, ambos algoritmos tienen tiempos de ejecución bastante similares, incluso cuando n es muy grande, sin embargo, a medida que crece p, se nota una diferencia sustancial, incluso manteniendo el valor n pequeño.



4. Hacer $p = 0.1n$, o sea, diez veces más observaciones que coeficientes en la regresión, ¿Cual es la n máxima que puede manejar su computadora?

   Con un bloque `try-except` de `Python` corrimos el algoritmo para los tamaños $50, 75, 100, 125, 150, 155, 281$ y 300 para p (definiendo $n = 10 * p$) después de varias pruebas. Aunque en varios casos el algoritmo logra terminar su ejecución, esto no significa que todos los resultados que se obtienen sean útiles. En particular, se encontraron las siguientes observaciones:

   - Para valores de n menores a 1550, el algoritmo se ejecuta con normalidad y todas las entradas del vector de coeficientes son no nulas, lo que parece indicar que aún es información útil.
   - Cuando pasamos a $n = 1560$, las últimas entradas del vector comienzan a ser idénticas a 0, independientemente de los puntos ajustados, lo que indica que posiblemente comienza a haber valores fuera del rango de precisión de punto flotante que maneja la máquina, este patrón continúa, agregando cada vez más ceros en la parte final del vector de coeficientes, hasta el punto en que prácticamente todos son iguales a cero.
   - A partir de $n = 2800$ encontraos varios avisos con `Runtime error`, lo que indica que existen problemas con la aritmética del punto flotante que estamos usando, ya sea valores que escapan del rango del sistema de punto flotante, u operaciones inválidas. Esto confirma que coeficientes obtenidos con este tamaño de matriz no son realmente útiles en la práctica (al menos con esta implementación).
   - Cuando usamos matrices con n aproximadamente igual a 3000 obtenemos que varios de los valores se vuelven un `nan`, es decir, un objeto que resulta de una operación en la que no fue posible obtener un valor numérico como resultado, la falla de nuevo es que hay operaciones que escapan de la precisión del sistema de punto flotante. Evidentemente, estos valores no son útiles para su implementación en ningún contexto.

   En resumen, aunque la computadora puede correr el programa para tamaños de matriz muy grande, hay que tener en cuenta que la precisión de las operaciones se va perdiendo cada vez que incrementamos el tamaño de la matriz (y por lo tanto su número de condición), lo que causa, que aunque tengamos valores como salida de la ejecucion, no es recomendable utilizarlos. Según lo observado en los experimentos anteriores, el tamaño más grande que es seguro para su uso en el caso en que $n = 10p$ es 1550×155.