Departamento de Física
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
UNIVERSIDAD DE CHILE FI3104
Métodos Numéricos para la Ciencia
Álvaro Núñez, Otoño 2022

Tarea #4

Benjamín Mancilla

Introducción

Nuevamente Newton Raphson es estudiado. Este es el método numérico para resolver sistemas lineales y no lineales a partir de la tangente sobre la curva de la función y la intersección con el eje de coordenadas. Sin embargo, en esta ocasión el método se expande para 2 dimensiones. Mantiene la misma estructura que el anterior con la diferencia que los términos son matrices de N*1 para F(x) y x, además de N*N para el Jacobiano, el cual representa la derivada en este caso.

$$\begin{matrix} x_1 \\ \vdots \\ x_{n_{n+1}} \end{matrix} = \begin{matrix} x_1 \\ \vdots \\ x_{n_n} \end{matrix} - \left(J(x_n)^{-1}\right) \cdot \begin{matrix} f_1(x_{1,\dots,x_n}) \\ \vdots \\ f_2(x_{1,\dots,x_n}) \end{matrix}$$

Pregunta 1

El problema de las áreas de la ecuación de Van der Waals (*Construcción de Maxwell)* se reduce a resolver un sistema no lineal de 2 variables. Para esto aplicamos Newton Raphson para dos dimensiones. La metodología consiste en calcular el Jacobiano de las funciones del enunciado y luego aplicar el método de forma matricial en Python, es decir con *Arrays* de la librería *numpy* y funciones para el algebra lineal de la librería *autograd*. Esta ultima es para el Jacobiano específicamente.

La dificultad resultante es la de los puntos iniciales, ya que este método puede diverger si estos están en una mala posición. Es difícil definir "*mala posición*" debido a que la convergencia del método de Newton únicamente de la forma de la función, por lo que en algunos casos el método puede funcionar perfectamente, aunque los puntos iniciales sean distantes de la raíz, como en otros que por más que estén cerca el método termina divergiendo.
Esto se soluciona usando el hint, es decir por cada iteración de temperatura, los resultados son almacenados en memoria no solo para la conclusión, sino que también para reutilizarlos para la siguiente iteración. Esto en Python se resuelve con una lista fuera del *for* que corre por el arreglo de temperaturas.
Además, se implementa un test con un *print* de ambas funciones sobre las raíces encontradas para cada temperatura, como no son muchos datos no es necesario un gráfico o un condicional para alertar cualquier raíz errada. El test entrega resultados entre exponente -16 y -17.

Conclusión, el método sigue siendo eficaz y sobre todo rápido en dos dimensiones a tal punto que se acerca al error de la máquina. Como se conocen los puntos v_g y v_l para cada temperatura, se puede calcular el área I y II, o mas bien integrar la función de presión con limites los volúmenes anteriores, esto último nos debería entregar un valor igual a 0, ya que las áreas son iguales y opuestas.

Nota: esto ultimo si se hace en el código para confirmar los resultados.

Pregunta 2

Para calcular el parámetro de magnetización *m* se deben despejar las raíces de la ecuación:

$$m = tanh(\beta m + h)$$

En este caso se estudiará como este parámetro se ve afectado por la intensidad de un campo magnético externo *h*, con una temperatura fija β. Nuevamente se usa Newton para el cálculo numérico.

Iterando en cadena, es decir un *for* dentro de otro *for,* se obtiene un gráfico para cada temperatura.

Ciertas temperaturas tienden de forma "logarítmica" al 1, mientras que otras parecen rectas. Como m representa la magnetización, y por ende la fase del material, se puede decir que cada comportamiento corresponde a una de estas.

Como se puede ver en la documentación de *Curie-Weiss Model* para altas temperaturas se tiene una fase paramagnética (β tendiendo a 0) y para bajas temperaturas se tiene una fase ferromagnética (β tendiendo a 10).

En conclusión, si una curva tiende rápidamente a 1 (ósea aparenta una recta) se trata de una fase ferromagnética, en cambio si tiende de forma logarítmica a esta, se trata de una fase paramagnética.

Gráficos