Pontificia Universidad Católica de Valparaíso
Instituto de Física

# Tarea 2: Solución de ecuaciones lineales y no lineales

Nicolás Sepúlveda Quiroz [1]
Colaboradores: Waleska Osses, Nicolás Pinochet, Paulo Vásquez.
*Pontificia Universidad Católica de Valparaíso*

**Resumen**

La solución de ecuaciones lineales y no lineales es la herramienta que tenemos para entender la naturaleza u otras áreas que impliquen la evolución de un sistema descriptible matemáticamente. El uso de las computadoras para resolver estos sistemas implica la aplicación de algoritmos cuya efectividad depende del problema. En este informe se plantean dos problemas físicos para los cuales hay que hacer uso de los métodos numéricos, y se analizan las soluciones y funcionamiento de los algoritmos.

10 de enero, 2020

[1] nicosepulveda4673@gmail.com

## Índice



## 1. Introducción

Nuestra comprensión de las leyes de los fenómenos naturales nos plantea problemas matemáticos en función de las relaciones entre las distintas variables que definen un sistema. La solución de estos problemas que están definidos por las ecuaciones que rigen el comportamiento de las variables, nos determinan, al menos en la mecánica clásica, la evolución temporal del objeto de estudio. Es por esto, que las técnicas de resolución de ecuaciones han tenido que ser mejoradas constantemente, y la introducción de los computadores ha permitido una forma de aproximarse a las soluciones utilizando métodos numéricos.

Para resolver numéricamente las ecuaciones, se deben hacer aproximaciones sobre los problemas a través de la discretización de las funciones y luego aplicar algoritmos que nos permitan acercarnos dentro de un margen de error al cálculo teórico. En este trabajo, se analizan dos problemas físicos que involucran la solución de una ecuación lineal, la ecuación de Poisson; y una no lineal, buscando el estado de equilibrio en un enlace iónico entre iones de $Na^+$ y $Cl^-$. Para el primer problema, se utilizaron los métodos directos de eliminación de Gauss, de sustitución directa (matriz tridiagonal), y factorización LU (algoritmo Doolittle). Para el segundo, se utilizaron métodos de aproximación a la raíz de una función: método de bisección, de Newton-Raphson y de la secante.

## 2. Problema 1

En la teoría electromagnética, la Ecuación de Poisson para el potencial eléctrico es uno de los pilares fundamentales, que se encuentra como consecuencia de la ley de Coulomb y el cálculo vectorial. Resolviéndola en las condiciones de borde adecuadas, es decir, que tengan sentido físico para el potencial eléctrico, podemos encontrar el potencial en todo el espacio dentro de la región a evaluar, y con este, el campo eléctrico que es definido como el negativo de su derivada según la mecánica clásica. La simplicidad de trabajar con un campo escalar

y no con el campo eléctrico, que es vectorial, hace mucho más efectivo resolver las ecuaciones para esta función. Se caracteriza además, como problema matemático, por la unicidad de la solución que es necesaria para obtener una realidad física con puntos espaciales no multivaluados de potencial o campo eléctrico; y por la continuidad en la región en estudio que asegura la no existencia intensidades infinitas de campo eléctrico. En este problema se resolverá la ecuación de Poisson para un diodo de vacío en una dimensión:

$$\frac{d^2\tilde{\phi}}{d\tilde{x}^2} = -\frac{\rho(\tilde{x})}{\epsilon_o}, \tag{2.1}$$

y en el problema a resolver las condiciones de borde son $\tilde{\phi}_0 = 0$ y $\tilde{\phi}(d) = 1$. El diodo de vació acelera electrones desde el cátodo ($\tilde{x} = 0$) hacia el ánodo ($\tilde{x} = d$) manteniendo una corriente constante

$$I = -\rho(\tilde{x})v(\tilde{x})A. \tag{2.2}$$

Usando la ley de conservación de energía, se tiene que en cualquier lugar $\tilde{x}$:

$$K(0) + U(0) = K(x) + U(x), \tag{2.3}$$

donde se tiene la energía cinética $K = \frac{1}{2}m_e v^2$ y la energía potencial $U = e\tilde{\phi}$. Las condiciones iniciles $v(0) = 0$ y $\tilde{\phi}(0) = 0$ implican

$$\tilde{\phi}(\tilde{x}) = -\frac{1}{2e}mv(\tilde{x})^2, \tag{2.4}$$

de donde se encuentra la velocidad en función del potencial:

$$v(\tilde{x}) = \sqrt{\frac{2e\tilde{\phi}(\tilde{x})}{m_e}}, \tag{2.5}$$

que es equivalente a la velocidad en la ecuación (2.2). Utilizando ambas se encuentra la expresión para la densidad de carga:

$$\rho(\tilde{x}) = -\frac{I}{A}\sqrt{\frac{m_e}{2e\phi(\tilde{x})}}, \tag{2.6}$$

y al reemplazar en (2.1) se tiene la ecuación diferencial en función de $\phi(\tilde{x})$

$$\frac{d^2\tilde{\phi}(\tilde{x})}{d\tilde{x}^2} = \frac{I}{\epsilon_o A}\sqrt{\frac{m_e}{2e\tilde{\phi}(\tilde{x})}}. \tag{2.7}$$

Según el problema lo plantea, la solución a esta ecuación diferencial satisfaciendo las condiciones de borde es

$$\tilde{\phi}(\tilde{x}) = V_o\left(\frac{\tilde{x}}{d}\right)^{4/3}, \text{ con } I = \frac{4}{9}\frac{\epsilon_o A}{d^2}\sqrt{\frac{2e}{m_e}}V_o^{3/2} \tag{2.8}$$

y sustituyendo en la ecuación (2.7):

$$\frac{d\tilde{\phi}(\tilde{x})}{d\tilde{x}} = \frac{V_0}{d}\frac{4}{3}\left(\frac{\tilde{x}}{d}\right)^{1/3}$$

$$\frac{d^2\tilde{\phi}(\tilde{x})}{d\tilde{x}^2} = \frac{4}{9}\frac{V_o}{d}\left(\frac{\tilde{x}}{d^2}\right)^{-2/3}. \tag{2.9}$$

Por otro lado, sustituyendo la ecuación de $I$ en (2.8) en (2.7):

$$\frac{d^2\tilde{\phi}(\tilde{x})}{d\tilde{x}^2} = \frac{4}{9}\sqrt{\frac{V_0^3}{\tilde{\phi}(\tilde{x})}} = \frac{4}{9d^2}\sqrt{\frac{V_0^2 d^{4/3}}{x^{4/3}}}, \tag{2.10}$$

lo que equivale a la ecuación (2.9).

La ecuación diferencial debe ser adimensional para poder ser resuelta numéricamente. Para ello, se introducen las variables

$$\phi := \frac{\tilde{\phi}}{V_0} \text{ y } := \frac{\tilde{x}}{d}. \tag{2.11}$$

La relación de la segunda derivada entre las variables con y sin dimensión se encuentra a través de la regla de la cadena:

$$\frac{d^2\tilde{\phi}}{d\tilde{x}^2} = \frac{d}{d\tilde{x}}\left(\frac{d\tilde{\phi}}{d\tilde{x}}\right) = \frac{d}{d\tilde{x}}\left(\frac{d\tilde{\phi}}{dx}\frac{dx}{d\tilde{x}}\right) = \frac{V_0}{d}\frac{d}{dx}\frac{dx}{d\tilde{x}}\left(\frac{d\phi}{dx}\right) = \frac{V_0}{d^2}\frac{d^2\phi}{dx^2}, \tag{2.12}$$

y usando que la solución $\tilde{\phi}(\tilde{x})$ en las nuevas variables es

$$\phi(x) = x^{4/3}, \tag{2.13}$$

se encuentra la ecuación diferencial adimensional:

$$-\frac{d^2\phi}{dx^2} = -\frac{4}{9}x^{-2/3} := f(x). \tag{2.14}$$

Ahora que está en su forma dimensional, se puede resolver la segunda derivada espacial de $\phi$ en una aproximación dada por

$$-\frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2} = f_i, \quad i = 1, \ldots, n \tag{2.15}$$

en donde $f_i$ es la discretización de la función $f(x)$ en los puntos $x_i = i(\Delta x) = i/(n+1)$ y $u_i$ son las soluciones asociadas a aquellos puntos.

Este sistema de ecuaciones se puede escribir en su representación matricial como una matriz tridiagonal. Para mostrar esto, se encuentran los primeros elementos:

$$
\begin{aligned}
f_0 &= 0 \\
f_1 &= -\left(\frac{u_2 - 2u_1}{(\Delta x)^2}\right) \\
f_2 &= -\left(\frac{u_3 - 2u_2 + u_1}{(\Delta x)^2}\right) \\
f_3 &= -\left(\frac{u_4 - 2u_3 + u_2}{(\Delta x)^2}\right) \\
&\vdots \\
f_{n+1} &= 1,
\end{aligned}
\tag{2.16}
$$

en donde el primer y último elemento corresponden a las condiciones de borde. Reconociendo el patrón de las constantes, se encuentra que la matriz representante del sistema es:

$$
\mathbf{A} = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & \dots & 0 \\
1 & -2 & 1 & 0 & 0 & \dots & 0 & 0 \\
0 & 1 & -2 & 1 & 0 & \ddots & 0 & 0 \\
0 & 0 & 1 & -2 & 1 & \ddots & 0 & 0 \\
\vdots & \vdots & \ddots & \ddots & \ddots & \ddots & 0 & 0 \\
0 & 0 & \ddots & \ddots & 1 & -2 & 1 & 0 \\
0 & 0 & \dots & \dots & 0 & 1 & -2 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}_{n\times n}
\quad , \quad
\vec{\ell} = \begin{pmatrix}
\ell_0 = 0 \\
\Delta x^2 f_1 \\
\Delta x^2 f_2 \\
\Delta x^2 f_3 \\
\vdots \\
\vdots \\
\Delta x^2 f_n \\
\ell_{n+1} = 1
\end{pmatrix}
\tag{2.17}
$$

Como es mostrado en (1), los elementos correspondientes a "$l_0$" y "$l_{n-1}$" son las condiciones de borde cuyos valores son conocidos y forman la primera y última fila de la matriz A. Eliminando estas y restándolas a la segunda y penúltima fila respectivamente, se encuentra la matriz

$$
\mathbf{A} = \begin{pmatrix}
-2 & 1 & 0 & 0 & \dots & 0 \\
1 & -2 & 1 & 0 & \ddots & 0 \\
0 & 1 & -2 & 1 & \ddots & 0 \\
\vdots & \ddots & \ddots & \ddots & \ddots & 0 \\
0 & \ddots & \ddots & 1 & -2 & 1 \\
0 & \dots & \dots & 0 & 1 & -2
\end{pmatrix}_{n\times n}
\quad
\vec{\ell} = \begin{pmatrix}
\Delta x^2 f_1 + 0 \\
\Delta x^2 f_2 \\
\Delta x^2 f_3 \\
\vdots \\
\vdots \\
\Delta x^2 f_n + 1
\end{pmatrix},
\tag{2.18}
$$

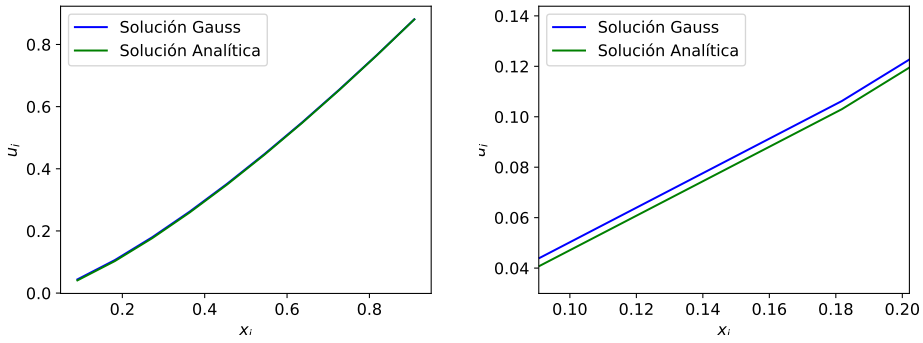



FIGURA 1. Eliminación Gaussiana vs Solución Analítica; n=10

en donde se observa que las condiciones iniciles quedan consecuentemente restadas del primer y último elemento del vector $\ell$. Dado que $f_0 = 0$, el primer elemento $\Delta x^2 f_1$ no se modifica, pero el elemento $\ell_n$ se convierte en $\ell_n = \Delta x^2 f_n - 1$.

En el vector $\vec{\ell}$ se reconoce qué función de $f_i$ se asocia a cada $\ell_i$. Y los coeficientes $a$, $b$, y $c$ equivalen a 1, $-2$ y 1 en el sistema (**??**). Este puede entonces ser escrito como

$$a_i u_{i-1} + b_i u_i + c_i u_{i+1} = \ell_i, \quad i = 1, 2, \ldots, n, \tag{2.19}$$

con las soluciones $u_i$ equivalentes al potencial en el punto $x = x_i$.

Para resolver el sistema de ecuaciones, se han utilizado los métodos de eliminación de Gauss y de sustitución directa en matrices de $nxn$ con $n = 10$, $n = 100$ y $n = 1000$; ambos códigos anexados en la última sección.

Para realizar la sustitución directa, se encontró, a partir de las soluciones de $i$ menor, una expresión para $\ell_{n+1}$ en función de $\ell_{i<n+1}$ y $u_1$:

$$1 = u_{n+1} = \sum_{0=1}^{n} (n-i)\ell(i+1) + (n+1)u_1, \tag{2.20}$$

y luego se sustituyó la solución de $u_1$ con la condición de borde $u_{n+1} = 1$ en cada una de las iteraciones de $u_i$.

En las figuras (1-6) se muestran los resultados para cada método en comparación con la solución analítica del problema en el intervalo $(0, 1)$.

Para los casos $n = 10$ y $n = 100$ se observa incluso en los gráficos no aumentados, como el método gaussiano logra una mayor precisión,

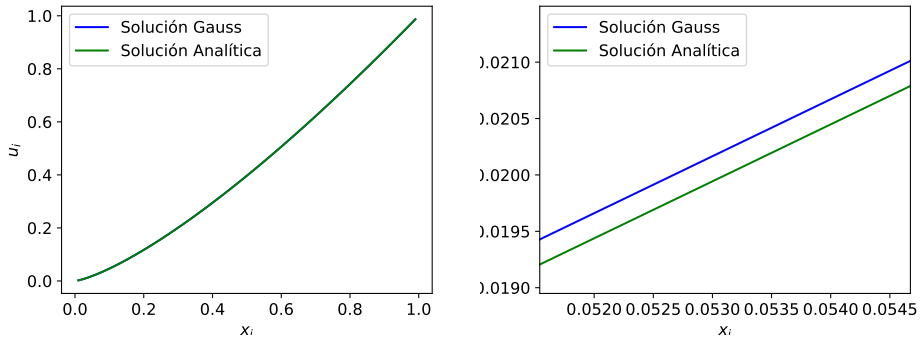



FIGURA 2. Eliminación Gaussiana vs Solución analítica; n=100

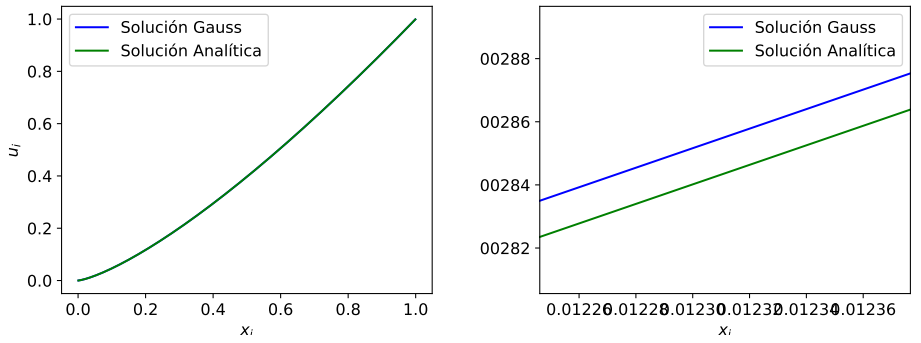



FIGURA 3. Eliminación Gaussiana vs Solución analítica; n=1000

acercándose más a la solución analítica. Los gráficos aumentados muestran la zona de menor $x_i$, en donde se observan las mayores diferencias en las curvas. Como referencia, se puede notar cómo en las figuras 2 y 4, para intervalos iguales de $x_i$ las separaciones son, gráficamente, aproximadamente iguales; sin embargo, la diferencia en cifras significativas del método gaussiano con respecto al otro es menor.

Comparando el caso $n = 1000$, se puede notar cómo la desviación con respecto a la solución analítica en intervalos de $x_i$ parecidos, es similar en ambos métodos. No obstante, el tiempo de cómputo del método de sustitución directa, al tener solo una operación de sustitución del último valor $f_n$ conocido y una suma por iteración, resulta mucho menos que en el método de Gauss; las operaciones fila incrementan sustancialmente

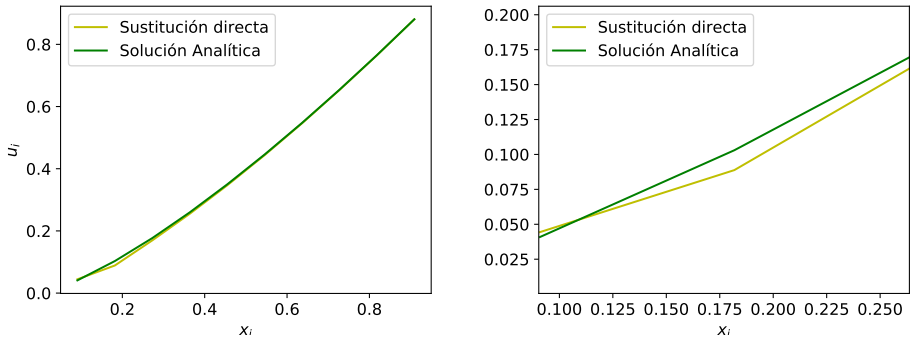



FIGURA 4. Sustitución directa vs Solución analítica; n=10

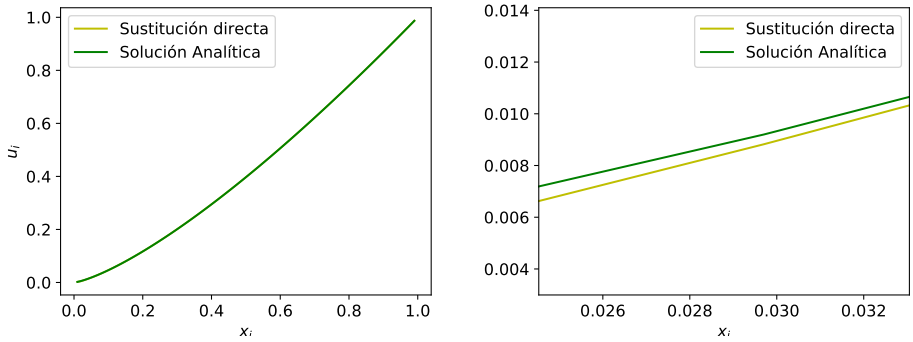



FIGURA 5. Sustitución directa vs Solución analítica; n=100

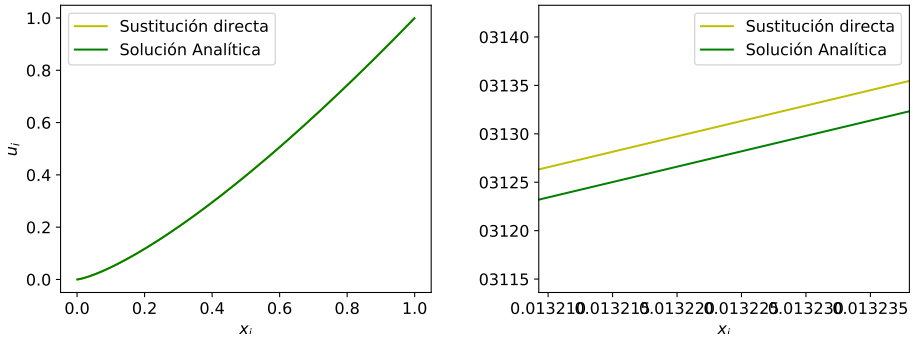



FIGURA 6. Sustitución directa vs Solución analítica; n=1000

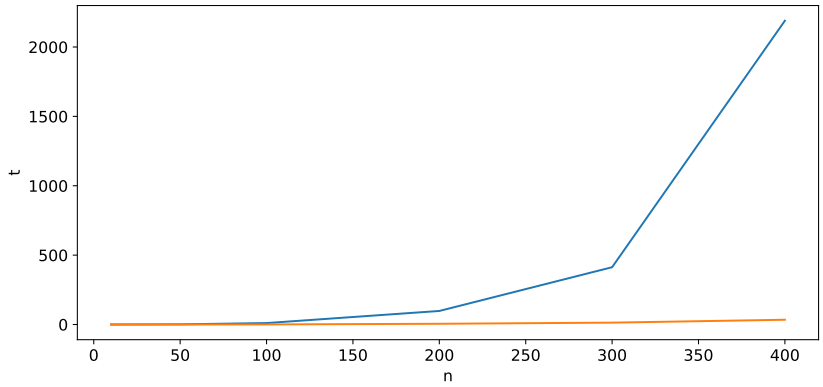



FIGURA 7. Comparación de tiempos de cómputo en segundos. Amarillo: Factorización LU; Azul: Eliminación Gaussiana.

la cantidad de operaciones. Por esto, la sustitución directa se vuelve una forma más eficiente de resolver el sistema cuando se trata de $n$ grande. Lo mismo ocurre con la factorización LU, que resulta tener una precisión idéntica al método de eliminación gaussiana, pero el tiempo de cálculo es considerablemente menor, como se observa en la figura 7. Debido al tiempo que le tardó al computador resolver el sistema, se muestran los resultados hasta $n = 400$, que son suficientes para mostrar el comportamiento.

En cuanto a los errores relativos asociados a ambos métodos, en las figuras 8 y 9 se observa como a medida que se aumenta el número de divisiones $(n)$, ambas soluciones aproximadas se acercan cada vez más a la analítica. El código para el método de Gauss fue ejecutado hasta $n = 1000$, en intervalos de 100 en 100, y el del otro método fue en números mas bajos, hasta un máximo de $n = 150$. El resultado es el esperado, mientras más particiones hay mayor precisión; sin embargo, tomando en cuenta las relaciones entre estos, los métodos pueden ser mas efectivos en distintas situaciones dependiendo de $n$, con consecuencias en el tiempo de iteración.

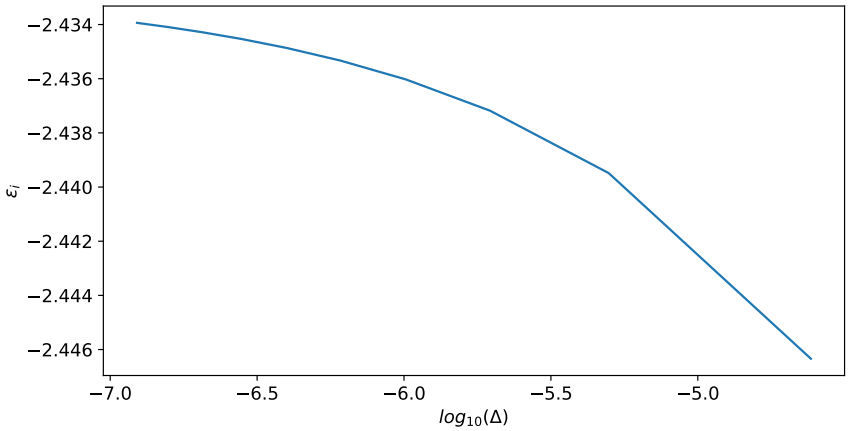



FIGURA 8. Error relativo máximo del método de Gauss, en escala logarítmica en función de $n$.

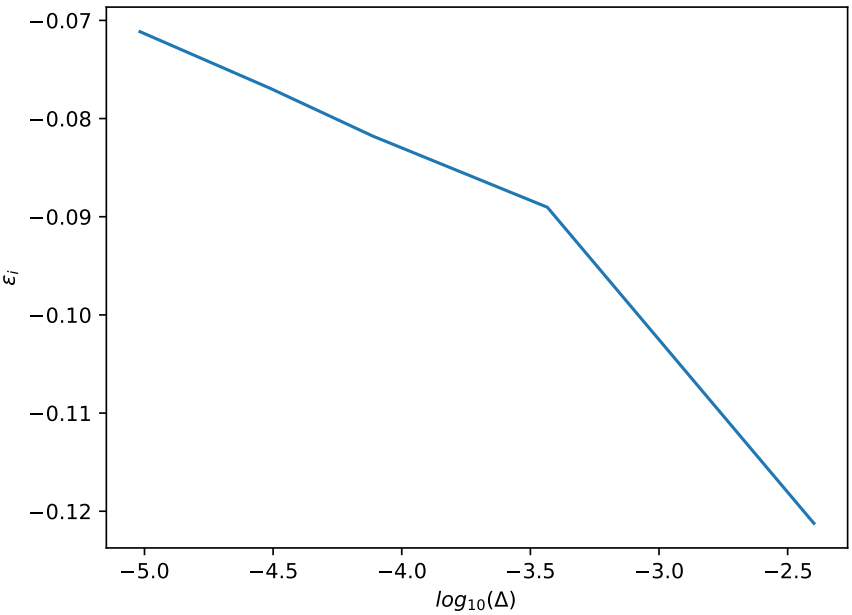
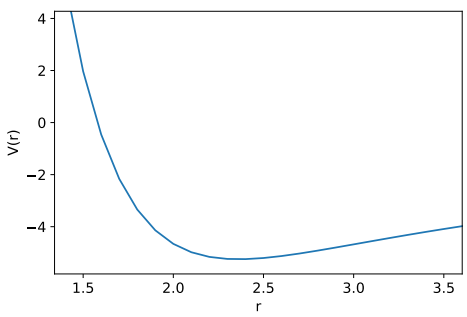
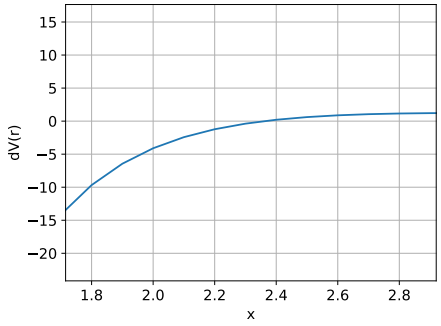



FIGURA 9. Error relativo máximo del método de Sustitución directas, en escala logarítmica en función de $n$.




(A)




(B)

FIGURA 10. (A) Pozo de potencial; (B) Raíz de la derivada del potencial.

## 3. PROBLEMA 2

En este problema se encontrará el radio de enlace para el enlace iónico de $Na^+$ y $Cl^-$, cuya función potencial define la energía necesaria para la transmisión de un electrón desde el átomo de sodio al de cloro. El potencial efectivo es descrito por:

$$V(r) = -\frac{e^2}{4\pi\epsilon_o r} + V_o e^{-r/r_o}, \tag{3.1}$$

en donde $V_0 = 1{,}09x10^3 eV$ y $r_0 = 0{,}330$Å.

Esta función, al ser una suma del potencial Coulombiano que disminuye como $\frac{1}{r}$ (negativo), y el término asociado a la distribución electrónica del sistema que disminuye exponencialmente (positivo); es caracterizada por un comportamiento de asintótico cercano a $r = 0$, y un crecimiento que se mantiene cercano al cero para los $r$ mayores. De esta forma, existe un pozo de potencial en $r_{en}$ en donde se mantiene una situación de equilibrio estable para el posicionamiento del ion contiguo (ver Figura 10($A$)). La determinación de esta posición de enlace se efectúa encontrando el mínimo de potencial, es decir, resolviendo

$$\frac{dV(r)}{dx} = 0. \tag{3.2}$$

La solución a la ecuación (3.2) es calculada numéricamente utilizando los métodos de la bisección, Newton-Raphson y de la secante; cuyos respectivos códigos son anexados en la última sección. La función se muestra en la figura 10($B$).

| Método | V'(r)=0 | V'($r_{en}$) |
|---|---|---|
| Bisección | 2.3616891958350035 | -4.440892098500626e-16 |
| Secante | 2.361689185753289 | -5.6689666738662936e-08 |
| Newton-Raphson | 1.5782485365668966 | — |

CUADRO 1. Solución $r_{en}$ para los diferentes métodos.

En el cuadro 1 se muestran las soluciones entregadas por los distintos métodos y el valor de $dV(r_{en})/dr$. El número de iteraciones necesarias para el método de la bisección fueron 10, mientras que para el de la secante 4. Si bien hay una diferencia significativa de iteraciones necesarias, no lo hay en el valor de $r_{en}$; pese a que si lo hay en la evaluación de $dV(r_{en})/dr$. Esto muestra cómo para este caso el método de la secante se vuelve mucho mas eficiente que el de la bisección, usando menos del 50 % de iteraciones. Esta eficiencia se logra, no obstante, eligiendo valores apropiados para $x_0$ y $x_1$ en el método de la secante, los cuales deben ser lo suficientemente cercanos a la raíz para asegurar la convergencia. En este caso se utilizó $x_0 = 0{,}1$ y $x_1 = 2{,}3$, sabiendo ya el punto en donde la derivada de $V(r)$ corta el eje $x$. Incluso, tomando valores mayores de $x_0$ la convergencia se logra en menos iteraciones.

Dado que el método de Newton-Raphson no tiene una condición global de convergencia garantizada, es posible que el valor inicial elegido haya causado que la raíz encontrada no sea la correcta. Por esto, se iteró el algoritmo para diferentes $x_0$ en el intervalo (0,3), y se observó cómo aproximadamente antes del $r_{en}$ conocido por los otros métodos la solución converge hacia 1,5782; y después de este, el algoritmo se dispara. Probablemente esto sea resultado de la derivada de $V(r)$ muy cercana a cero en la vecindad de la raíz, sobre todo para $r > r_{en}$; lo que causa que el algoritmo se atasque en un bucle. Esto se toma como una ventaja del método de la secante por sobre el de Newton-Raphson; no hay errores por la evaluación de la derivada de la función.

Para analizar si el error es debido a los valores muy pequeños de la derivada, se multiplicó su función por $10^8$ y se disminuyó la tolerancia $10^3$ veces; se escala la función pero la raíz queda en el mismo punto. En la figura 11 se muestra una iteración del método para 100 valores de $x_0$ entre 1,5 y 4. Las soluciones entregadas por el algoritmo son exactamente el valor elegido para $x_0$, excepto en un rango muy pequeño entre $r = 2{,}28$ y $r = 2{,}46$, lo que muestra el acotado radio de convergencia del algoritmo para este problema[2], comparándolo con los otros métodos.

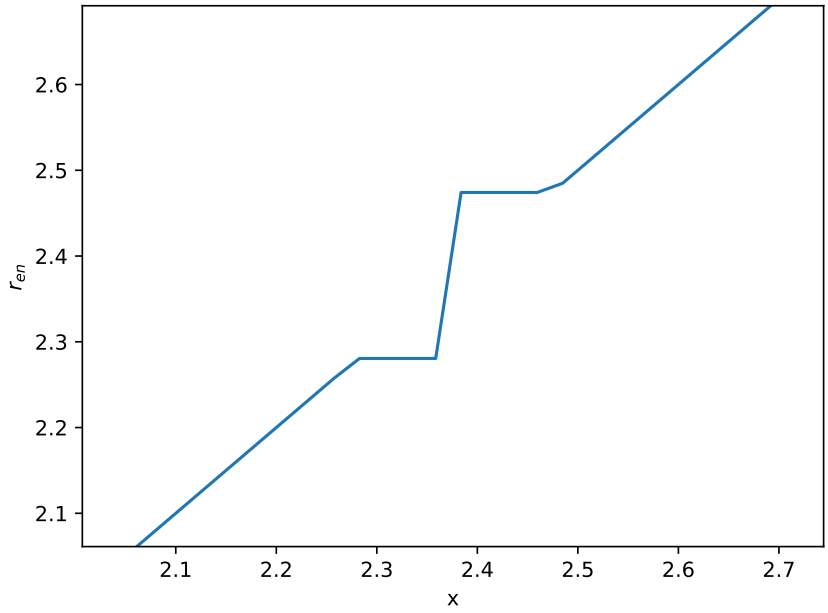



FIGURA 11. Radio de convergencia del método de Newton-Raphson.

## 4. ANEXO

1. Definición de la matriz del problema 1.

```
def S(n):
#discretizaci n de f(x) (no est n en la ecucion x0 y xn)
        dl=1/(n+1) #en total son n puntos en los que se resuelve la funci n

        l=[None]*n
        f=[]
        for i in range(0,n):

                l[i]=(i+1)*dl
                f.append(((l[i]**(-2/3))*(-(4/9)*(-((dl)**2))))) #

        f[n-1]=f[n-1]-1

        #Matriz del potencial electrost tico:
        V=np.zeros((n,n))
        for i in range(0,n):
            V[i,i]=-2
            if i!=0:
```

```
            V[i,i-1]=1
        if i!=n-1:
            V[i,i+1]=1

    S=np.column_stack((V,f))

    return(S)
```

2. Script Eliminación Gaussiana

```
    def triangular(a):
        for k in range(0,n):
            if a[k,k] != 0:
                for i in range(k+1,n): #la fila que voy a cambiar
                    factor=a[i,k]/a[k,k]
                    for j in range(k,n+1): #el elemento columna de la
                        a[i,j]=a[i,j]-factor*a[k,j]
        return(a)


def sust_inv(a):
        n=np.shape(a)[1]-1

        x=[None]*n
        x[n-1]=a[n-1,n]/a[n-1,n-1]
        #resolver la raz x_i, que parte desde (n-1):
        for i in reversed(range(0,n-1)):
            suma=a[i,n] #cte. en el i
            for j in range(i+1,n): #sumo todo desde el coef i+1
                                   #que estoy tomando hasta el ltimo , que
                suma=suma-a[i,j]*x[j]

            x[i]=suma/a[i,i]

        return(x)
```

3. Solución analítica y sustitución directa

```
    #---------- solucin analtica
def phi(r):
        dl=1/(n+1)
        phi=r**(4/3)
        #phi=r**2*(dl**2)
        return phi

        def sust(a):
            x=[None]*n
            x[0]=(1-(sum(a[i,n]*(n-i) for i in range(0,n-1))))/(n+1)
            a[n-1,n]=a[n-1,n]+1
```

```
                for j in range(1,n):
                        x[j]=sum(a[i,n]*(j-i) for i in range(0,j-1))+(j+1)*x[0]
                return(x)

        ss=sust(S(n))
```

4. Cálculo de errores

```
    t=None
# error de dx
error=[]
tiempoGE=[]
tiempoLU=[]
#dl en funci n de n:
dl_n=[]
for m in n_vector:
        print(m)
        n=m

        def p(n):
                p=[]
                for i in l(n):
                        p.append(phi(i))
                return(p)
        dl_n.append(1/(n+1))

        errores_n=[]
        for i in range(1,n+1):
                errores_n.append(np.log(abs((-ss[i-1]+p(n)[i-1])/p(n)[i-1])))
#               errores_n.append(np.log(abs((x[i-1]-p(m)[i-1])/p(m)[i-1])))


        error.append(max(errores_n))


plt.plot(np.log(dl_n),error)
plt.xlabel('$log_{10}(\Delta)$')
plt.ylabel('$\epsilon_i$')
plt.show()
```

5. Factorización LU

```
    t2_start = perf_counter()


        a=np.zeros((n,n))
        b=[None]*n
        for i in range(0,n):
                for j in range (0,n):
                        a[i,j]=S(n)[i,j]
```

```
                b[i]=S(n)[i,n]


            l=np.identity(n)
            u=np.zeros((n,n))
            for j in range(n):

                for i in range(j+1):
                    suma=sum(u[s,j]*l[i,s] for s in range(i))
                    u[i,j]=a[i,j]-suma
                for i in range(j,n):
                    suma2=sum(u[s,j]*l[i,s] for s in range(j))
                    l[i,j]=(a[i,j]-suma2)/u[j,j]
            #definir z
            z=[None]*n
            y=[None]*n
            z[0]=b[0]
            for i in range(1,n):
                suma=sum(l[i,j]*z[j] for j in range(0,i))
                z[i]=b[i]-suma
            y[n-1]=z[n-1]/u[n-1,n-1]
            for i in reversed(range(0,n)):
                y[i]=(z[i]-sum(u[i,j]*y[j] for j in range(i+1,n)))/u[i,i]
            return(y)
        LU_=LU(a)

        t2_stop = perf_counter()
        tiempoLU.append(t2_stop-t2_start)
```

## 6. Método de la bisección

```
    import numpy as np
import math as m
import matplotlib.pyplot as plt
from funci n import V,dV
v_0=1.09*(10**3) #electronVolt
r_0=0.330 #amstrongs
e_0=8.85*(10**(-32)) #C^2/(N*m^2)
q=1.602176565*10**(-19) #C

plt.rcParams['figure.figsize'] = (5, 4.0) # set default size of plots
font = {'family' : 'sans',
        'weight' : 'normal',
        'size'   : 12}
plt.rc('font', **font)

#se define el potencial
```

```
nmax=100


a=0.1
b=10

fa=dV(a)
fb=dV(b)
if np.sign(fa)==np.sign(fb):
        print("La funci n tiene el mismo signo en "+str(a)+" y "+str(b))
        print(str(np.sign(fa)))
        print(str(np.sign(fb)))


intervalo=b-a

for n in range(0,nmax):
        intervalo=intervalo/2
        p=a+intervalo
        fp=dV(p)
        print("El nuevo punto es "+str(p))


        e=0.01
        if abs((p-a)/p)<e:
                print("Convergencia obtenida")


        if np.sign(fa)==np.sign(fp):
                a=p
                fa=fp
        else:
                b=p
                fb=fp

print(p)
print(dV(p))

x=np.linspace(0.1,5,50)
print((q**2)*(10**(-10))*6.241509*(10**18)/(4*m.pi*e_0*x**2)-(v_0*m.e**(-x/r_0))/


plt.plot(x,V(x))
plt.xlabel('r')
plt.ylabel('V(r)')
plt.show()
```

```
plt.plot(x,dV(x))
plt.grid()
plt.xlabel('x')
plt.ylabel('dV(r)')
plt.show()
```

## 7. Método de la secante

```
    #secante
import math as m

v_0=1.09*(10**3) #electronVolt
r_0=0.330 #amstrongs
e_0=8.85*(10**(-32)) #C^2/(N*m^2)
q=1.60*10**(-19) #C


def Secante(x0,x1,tol,N,f):

    i=1

    while i<=N:
        x= x1 - (x1-x0)*f(x1)/(f(x1)-f(x0))
        print(i,x)

        if abs(x-x1)<tol:
            return x
        i=i + 1
        x0=x1     # redefine x0
        x1=x      #redefine x1

    print('El metodo fracaso despues de %d iteraciones' %N)

f=lambda r: (q**2)*(10**(-10))*6.24*(10**18)/(4*m.pi*e_0*r**2)-(v_0*m.e**(-r/r_0)
x0=0.1
x1=2.3
tol=0.0001
N=100

x=Secante(x0,x1,tol,N,f)
print()
print('La solucion es: ',x)


print(f(x))
```

## 8. Método de Newton-Raphson

```
    import numpy as np
from funci n import V,dV
```

```
import matplotlib.pyplot as plt
import math as m
v_0=1.09*(10**3) #electronVolt
r_0=0.330 #amstrongs
e_0=8.85*(10**(-32)) #C^2/(N*m^2)
q=1.602176565*10**(-19) #C
def newton(r):
        h=V(r)/(((q**2)*(10**(-10))*6.241509*(10**18)/(4*m.pi*e_0*r**2)-(v_0*m.e*
        while abs(h)>0.0000001:
                h=(V(r)/((q**2)*(10**(-10))*6.241509*(10**18)/(4*m.pi*e_0*r**2)-(

                x1=r-h
                r=x1
        return(r)

x0=2.35

print(newton(x0))

print(dV(newton(x0)))


l=np.linspace(1.5,4,100)
j=[]
for i in l:
        j.append(newton(i))
plt.plot(l,j)
plt.xlabel('x')
plt.ylabel('$r_{en}$')
plt.show()
```

## 5. Bibliografía


1.http://www.math.pitt.edu/ sussmanm/1080/Supplemental/Chap3-4up.pdf

2.Ehiwario, J.C. and Aghamie, S.O.: Comparative study of bisection, Newton-Raphson and secant methods of root-finding problems.