FIZ1431 ⋆ Física Computacional 31 de Marzo de 2021

Tarea 1

Brian Tapia Contreras †

Pregunta 1

En primer lugar, elaboré un programa en fortran (1.1.1) para escribir un archivo de extensión *.dat* con los datos de la tabla. Este código, al igual que todos los demás códigos que se referenciarán en este documento, se encontraran anexos al final.

- Para ajustar un polinomio de segundo orden, se definió una función auxiliar $l(x)$ de la forma descrita en la ecuación (1), y se utilizó *fit* en la interfaz de **GNU Plot**.

$$l(x) = a(x - x_0)^2 + b(x - x_0) + c \tag{1}$$

donde x_0 corresponde a un parámetro de desplazamiento en el eje x. La necesidad de definir la función de esta forma recae en que *fit*, en una primera instancia, pareciera asumir que queremos ajustar datos cerca del origen. Al hacer un fit de una función cuadrática de una forma más genérica, esta curva se parece a una función lineal en el intervalo de datos en el que estamos trabajando.

En este caso, definimos $x_0 = 1990$. De esta manera, el dato de menor valor de nuestra tabla quedará en cero para los efectos del ajuste.

El ajuste resultante se muestra en la Figura 1.

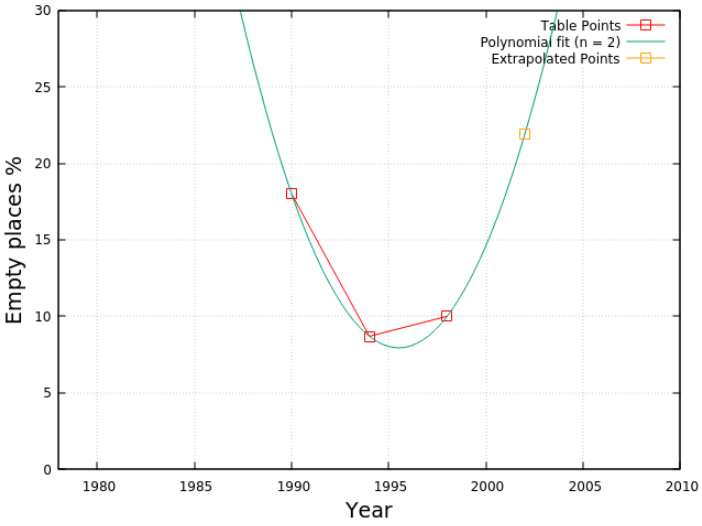


Figura 1: Gráfico de los datos y ajuste de función cuadrática. Se agrega un punto extrapolado. Código anexo en 1.1.2.

†brian.tapia@uc.cl

- En la Figura 1 se grafica el valor extrapolado para el año 2002. Se obtuvo evaluando la función ajustada en el año requerido. Para esto, también se creó un programa en fortran (1.1.3), los parámetros que se utilizaron para definir la función y que se muestran en el programa se extrajeron directamente desde **GNU Plot**.

- El valor exacto obtenido para el porcentaje de sillas vacías en el año 2002 fue de un 21,9 %. Considerando el 22 % informado por la revista, podríamos decir que el porcentaje estimado es bastante bueno. El error porcentual corresponde al 0,0045 %.

- Repitiendo el procedimiento realizado en el ajuste cuadrático, pero esta vez con $l(x)$ correspondiente a un polinomio de grado 3 de la siguiente forma:

$$l(x) = a(x - x_0)^3 + b(x - x_0)^2 + c(x - x_0) + d \tag{2}$$

Se llegó a los siguientes valores para los parámetros del ajuste:

$a = 2{,}13337 \times 10^{-15}$
$b = 0{,}33125$
$c = -3{,}65$
$d = 18$

Viéndose el ajuste de la siguiente manera:

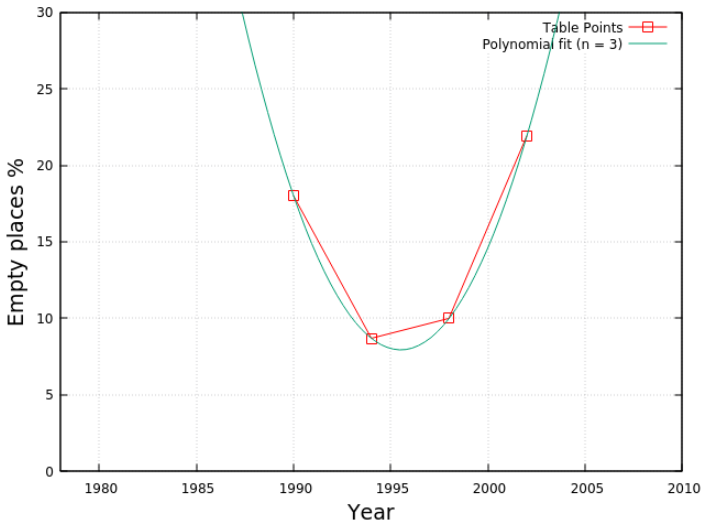


Figura 2: Gráfico de los datos y ajuste de un polinomio interpolante de grado 3. Código anexo en 1.1.4.

Notamos que se parece mucho al ajuste cuadrático, y en efecto, al observar los parámetros nos damos cuenta que b, c, d en el ajuste de tercer grado, corresponden exactamente a a, b, c en el ajuste cuadrático. A su vez, a, en el último ajuste, toma un valor muy cercano a cero.

Pregunta 2

- Debido a que en esta pregunta el código es la parte fundamental, lo adjuntaré directamente con los comentarios correspondientes. No estará en el anexo del coumento.

```
PROGRAM deg2eqsol
        implicit none

        real(8)::pr1,pr2,pi1,pi2

        real(8)::a,b,c
        real(8)::discriminant

        ! Consulta al usuario por los 3 coeficientes de la ecuación.
        write(6,*)'Escoja 3 numeros reales: a,b,c (uno por uno).'
        do while(1.eq.1)
                write(6,*)'a: '
                read(5,*)a
                if(a.ne.0.0)then
                        exit
                elseif(a.eq.0.0)then
                        write(6,*)'Escoja "a" distinto de 0'
                endif
        enddo
        write(6,*)'b: '
        read(5,*)b
        write(6,*)'c: '
        read(5,*)c
        ! FIN de la consulta.

        ! Defino el discriminante para ocuparlo posteriormente en las
        ! soluciones.
        discriminant = b**2 - 4*a*c

        ! Condicionales para saber si las soluciones son reales o
        ! imaginarias

        ! En estricto rigor, esto es ver si el discriminante es positivo
        ! o negativo, pero lo hacemos así para evitar la resta y el posible
        ! error en la resta de cifras significativas.

        ! Raices reales:
        if(b**2 .ge. 4*a*c)then
                pr1 = (-b + (discriminant)**(0.5))/(2*a)
                pr2 = (-b - (discriminant)**(0.5))/(2*a)
                write(6,*)pr1,'Raiz Real'
                write(6,*)pr2,'Raiz Real'
```

```
        ENDIF

        !Raices complejas:
        if(b**2 .lt. 4*a*c)then
                pr1 = (-b)/(2*a)
                pr2 = (-b)/(2*a)

                pi1 = ((-1*discriminant)**(0.5))/(2*a)
                pi2 = ((-1*discriminant)**(0.5))/(2*a)
                write(6,*)pr1,'+',pi1,'i Raiz Compleja'
                write(6,*)pr2,'-',pi2,'i Raiz Compleja'
        ENDIF

END PROGRAM
```

En rasgos generales, lo que busca el código es solicitar al usuario 3 números reales. En base a estos se construye un polinomio de grado 2 y se buscan las dos soluciones, utlizando la siguiente ecuación:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{3}$$

El término $\Delta = b^2 - 4ac$ se denomina el discriminante, y será nuestro criterio para determinar si las soluciones son reales ($\Delta \geq 0$), o complejas ($\Delta < 0$). En la práctica, para evitar errores de cancelación de cifras significativas, realizaremos la comparación entre ambos términos que conforman el discriminante, y no los restaremos hasta entregar la solución.

Pregunta 3

- Para abordar este problema serán fundamentales las ecuaciones de movimiento en una o dos dimesiones. En el primer caso, donde el movimiento es sólo vertical, podemos modelar la altura de la partícula (y su velocidad) utilizando la siguiente expresión genérica, que resulta de asumir una aceleración constante (aceleración de gravedad) e integrar dos veces respecto al tiempo:

$$y(t) = -\frac{1}{2}gt^2 + v_0 t + y_0 \tag{4}$$

$$v_y(t) = -gt + v_0 \tag{5}$$

Donde y_0 y v_0 son condiciones iniciales. Notamos que estas condiciones iniciales serán, en un inicio, h y 0 respectivamente, por lo que tenemos las ecuaciones listas para describir el movimiento de la partícula antes del primer choque. Sin embargo, después del primer choque, debido a que tenemos un coeficiente de restitución que puede ser distinto de uno, las condiciones iniciales pueden cambiar, y por lo tanto las ecuaciones que dominan el movimiento de la partícula.

El planteamiento numérico para llevar a cabo el problema, consistirá en modelar el movimiento de la partícla en estos intervalos regidos por la misma ecuación de movimiento, esto es, los intervalos entre dos botes distintos. Para ello, nos interesa caracterizar estos botes, y en este caso lo haremos utilizando el tiempo.

Notamos que los botes ocurren cuando $y = 0$, por lo que podemos despejar el tiempo de la ecuación.

$$0 = -\frac{1}{2}gt^2 + v_0 t + y_0 \tag{6}$$

$$\Rightarrow \frac{1}{2}gt^2 - v_0 t = t\left(\frac{1}{2}gt - v_0\right) = y_0 \tag{7}$$

Para simplificar los cálculos, diferenciamos las condiciones iniciales que pueden existir en los botes:

- En el caso del primer bote, teníamos como condiciones iniciales $y_0 = h$, $v_0 = 0$, lo que resulta directamente en:

$$t_{bote} = \sqrt{\frac{2h}{g}} \tag{8}$$

- En cualquier otro caso, consideraremos la altura inicial del bote como 0, mientras que su velocidad la dejaremos expresada como v_0, resultando:

$$t_{bote} = \frac{2v_0}{g} \tag{9}$$

Notamos que la solución $t = 0$ es trivial y no corresponde al momento de un bote posterior, por lo que fue descartada.

Teniendo estos tiempos, podemos fácilmente tener un contador de tiempo y obtener las alturas de la pelota como función del tiempo rigiéndonos bajo una única ecuación de movimiento válida en este intervalo de tiempo. Estos datos serán guardados en un archivo.

Sin embargo, aún después de esto nos falta modelar el caso del bote en sí. Esto ocurrirá cuando nuestro contador de tiempo supere este tiempo máximo que llamamos t_{bote}. En este caso, reseteamos el tiempo para que calce con el tiempo del bote, y agregaremos estos datos al archivo. Además, será momento de resetear nuestras condiciones iniciales y nuestro contador de tiempo. Aquí es donde entra el juego el coeficiente de restitución, qué, siguiendo la relación explicitada en la Ecuación 10, dará un nuevo valor a la velocidad inicial. Este reseteo de parámetros dará lugar a unas nuevas ecuaciones de movimiento, y a un nuevo loop sobre lo ya mencionado.

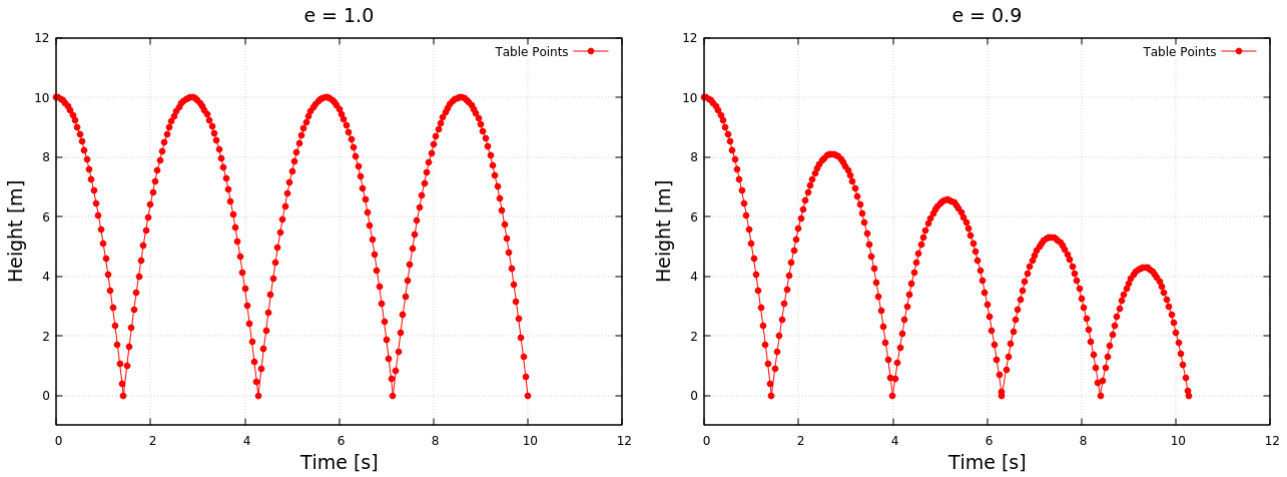
$$v_0 = -ev_y \tag{10}$$

; donde v_y es la velocidad con la que la partícula llega al choque.

En el programa existen dos condiciones de término. La primera la pone el usuario, consiste en un tiempo total de simulación, que se interpreta como que después de dicho tiempo no pueden existir más botes. La segunda es más automática y la utilizo por las limitaciones numéricas de la simulación, consiste en que t_{bote} no puede ser nunca mayor que el diferencial dt que utilizamos para iterar sobre el tiempo, de lo contrario saldremos de el **do while** principal. Esto puede ocurrir antes del tiempo total de la simulación, o bien puede no ocurrir nunca, depende muy directamente del coeficiente de restitución.

El código que realiza esto se encuentra con una explicación más detallada en el anexo 1.3.1.

A continuación, gráficos de cómo varía la altura de la partícula como función del tiempo, a distintos coeficientes de restitución. En el archivo de entrega se incluirán gifs de simulaciones de la partícula en el espacio (código anexo de las animaciones en 1.3.4).



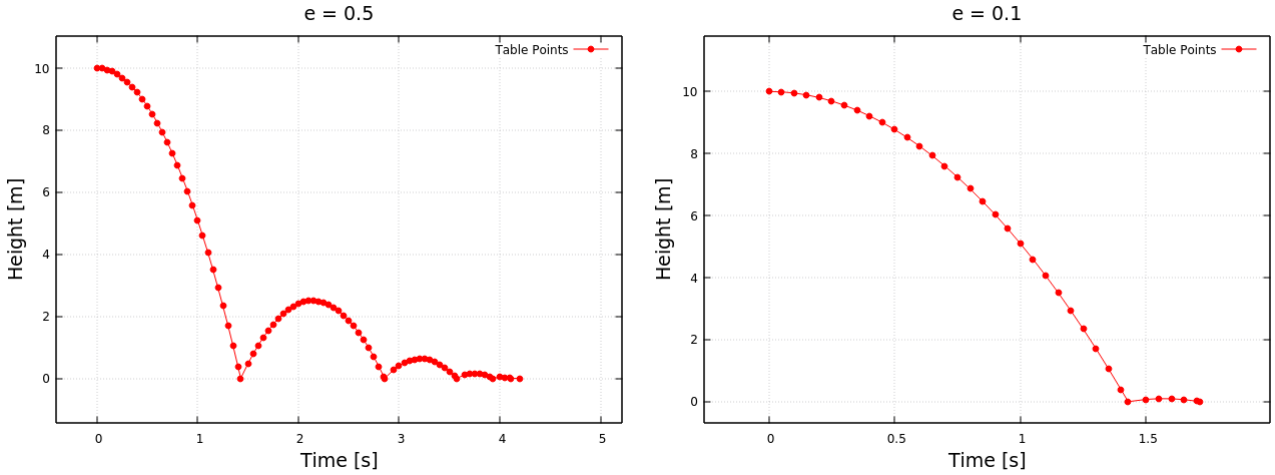


Figura 3: Altura de la partícula como función del tiempo a distintos coeficientes de restitución. Código anexo en 1.3.3.

Notar la diferencia en los límites del eje temporal, el primer bote siempre se da en el mismo tiempo.

Lo observado en los gráficos es consistente con lo que debería pasar en cada caso. A mayor coeficiente de restitución, menos energía se disipa en el choque.

- Para estudiar el caso en que la partícula tiene una velocidad inicial en el eje x, escribimos las ecuaciones de movimiento para dicha dirección:

$$x(t) = v_{x0}t + x_0 \tag{11}$$

$$v_x(t) = v_{x0} \tag{12}$$

Notamos que estas condiciones no alteran lo que ya habíamos hecho en el código, por lo que son completamente compatibles y las podemos añadir en forma de una variable más. En este caso, por simplicidad y simetría traslacional, estableceremos $x_0 = 0$, y v_{x0} será un parámetro que deberá ingresar el usuario. El código final está en el anexo 1.3.2.

Ahora que generalizamos este código, recuerdo también que el código para realizar todas las animaciones pedidas (anexo 1.3.4) también se extiende a este cambio en el código. Lo único necesario es cambiar el nombre del *file*.

El formato de nombre de un file es el siguiente:

freep{}___e{}_{}.dat

Entre las primeras llaves se puede llenar con un 1 o 2, dependiendo del inciso del ejercicio que queramos correr (2 incluye desplazamiento en x). El segundo espacio corresponde al primer digito del coeficiente de restitución, y el tercer espacio al primer decimal del coeficiente de restitución.

Pregunta 4

- Para definir la función raíz, nos aprovechamos de la función *sqrt*, ya incorporada en fortran. Se escribió un programa (1.4.1) que consulta al usuario un orden de magnitud para δ, y luego hace los cálculos para obtener la derivada de forma numérica, evaluada en $x = 1$.

  De forma analítica obtenemos lo siguiente:

$$\left(\frac{d}{dx}\sqrt{x}\right)\Big|_{x=1} = \frac{1}{2\sqrt{x}}\Big|_{x=1} = \frac{1}{2} \tag{13}$$

  Numéricamente, se obtuvo un valor aproximado de 0,49875, dando un error relativo de $2{,}48757 \times 10^{-3}$.

- Se escribió un programa muy similar (1.4.2), pero esta vez no se consulta por el orden de magnitud de δ, sino que estos vienen guardados en un vector de antemano, acorde a lo pedido en el enunciado.

  Se realizó el cálculo del error relativo en cada caso, y los resultados se pueden observar en la siguiente figura:

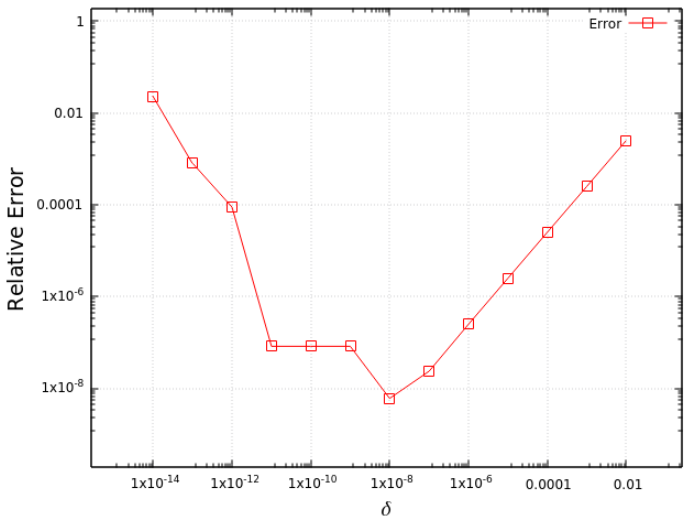


Figura 4: Error relativo como función del valor de δ. Código anexo en 1.4.3.

Este gráfico es esperado y perfectamente consistente con lo visto en clases. Notamos que el error relativo no es monótono, si no que tiene un mínimo y desde ese mínimo crece hacia ambos lados de δ. A valores más pequeños de δ, el crecimiento es culpa de los errores de redondeo, mientras que a mayores valores de δ nos encontramos con errores de truncamiento.

Pregunta 5

- Comenzamos, por supuesto, de la ecuación dada:

$$\sqrt{x^2+1}-1 \tag{14}$$

Si la calculamos directamente, estaremos arrastrando errores debido a la pérdida de cifras significativas, ocasionadas por la resta presente en la expresión.

Para observar la presencia de estos errores, buscaremos una forma analítica de expresar esta expresión, valga la redundancia, sin incluir ninguna resta en ella. Para ello, racionalizamos:

$$\sqrt{x^2+1}-1 \cdot \frac{\sqrt{x^2+1}+1}{\sqrt{x^2+1}+1} = \frac{x^2+1-1}{\sqrt{x^2+1}+1} = \frac{x^2}{\sqrt{x^2+1}+1} \tag{15}$$

Escribimos en fortran un programa que calcule ambas expresiones. Este programa se encuentra en el anexo 1.5.1.

Utilizando **GNU Plot** se graficó el valor obtenido en cada una, como función de un $x \in [0, 10^{-4}]$, intervalo donde ambas partes de la resta deberían ser similares en magnitud. Esto se realizó definiendo las variables del programa en precisión simple y doble precisión. Los resultados se muestran a continuación.

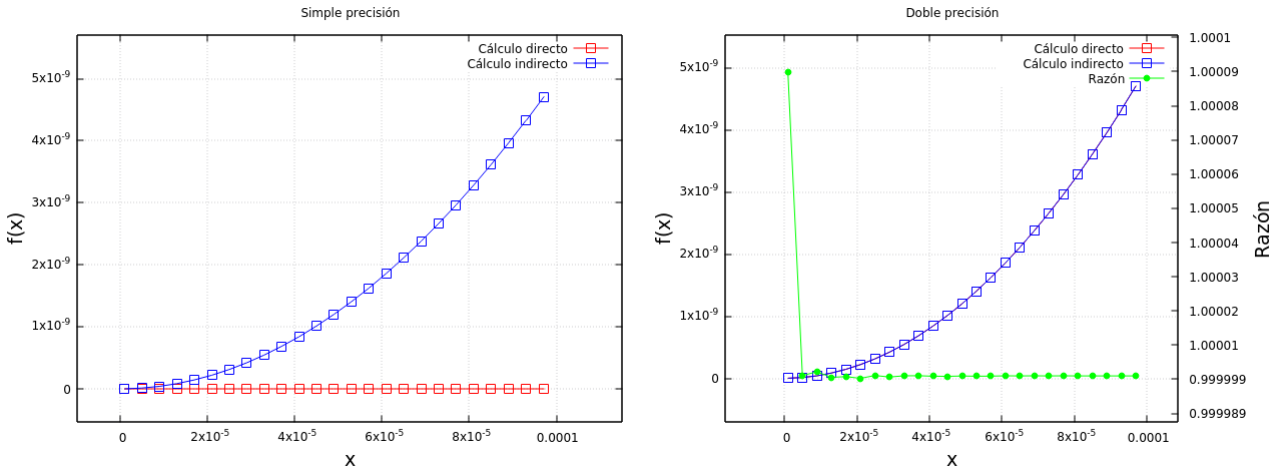


Figura 5: Cálculos directo e indirecto de la expresión pedida. Código anexo en 1.5.2.

Se puede observar qué, definiendo las variables en simple precisión, el cancelamiento de cifras significativas es notorio, llevando todos los cálculos directos a cero. Por supuesto notamos que con el cálculo indirecto el resultado es bastante fidedigno.

En doble precisión la diferencia es minúscula, pero existe. Para verla recurrimos a la razón entre ambas. Si fueran completamente iguales esta razón se mantendría constante en uno, sin embargo, la razón fluctúa de manera muy débil al rededor de 1, sobretodo a valores de x más cercanos a cero.

Pregunta 6

- Se construyó un código que permite calcular la energía media para la molécula H_2, asumiendo que se comporta como un oscilador armónico simple cuántico.

  Para ello, se utilizó la relación dada por Boltzmann y Gibbs mostrada en el enunciado. El código se escribió siguiendo los rasgos requeridos en la parte a) de la pregunta. Este se muestra anexo en 1.6.1.

- Junto con calcular la energía media, se midió el tiempo de ejecución en cada cálculo (para cada N dado). Los resultados se muestran resumidos en el siguiente gráfico.

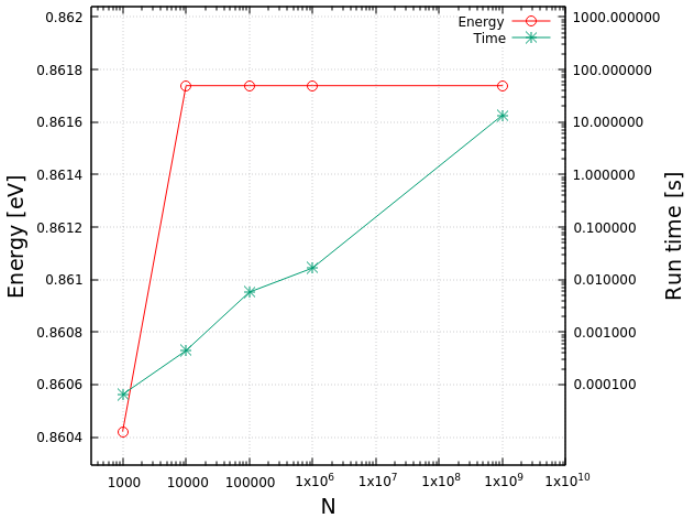


Figura 6: Energía media y tiempo de ejecución como funcion del parámetro N. Código anexo en 1.6.2.

Se puede concluir que, respecto al tiempo de ejecución del programa, este aumenta de manera exponencial, por lo que es mucho más eficiente, en órdenes de magnitud, correr un problema con menos iteraciones. Respecto a los resultados de energía, observamos que la suma converge rápidamente, siento un $N = 10^4$ suficiente para llegar al valor de convergencia.

Combinando ambas conclusiones, podríamos decir que existe un punto que maximiza la eficiencia del programa, sin perder precisión en el resultado y minimizando el tiempo de ejecución.

- Por alguna razón cada vez que corro el código para la parte c me salen absolutamente todas las medias de energía iguales, y cada vez que corro el mismo código me salen valores de medias distintas... todas iguales pero cada vez que corro el código cambia ese valor. De verdad no entiendo por qué, adjunto mi código en 1.6.3, si me pudiesen hacer un feedback de esto en específico lo agradecería mucho.

1. Códigos Anexos

1.1. Pregunta 1

1.1.1. writedoc

Escribe la tabla inicial en el archivo *table.dat*.

```
!1990, 1994, 1998
!18  , 8.7 , 10

PROGRAM writedoc
        implicit none

        integer::i
        integer,allocatable::yrs(:)
        real(8),allocatable::pct(:)

        allocate(yrs(3))
        yrs(1)=1990
        yrs(2)=1994
        yrs(3)=1998

        allocate(pct(3))
        pct(1)=18.d0
        pct(2)=8.7d0
        pct(3)=10.d0

        open(1,file='table.dat',status='unknown')

        do i=1,3
                write(1,"(i4,1x,F4.1)")yrs(i),pct(i)
        enddo

        close(1)
        deallocate(yrs,pct)

END PROGRAM writedoc
```

1.1.2. Figure 1

Grafica la Figura 1.

```
file = 'table.dat'
file2 = 'extrapolations.dat'

set xrange [1978:2010]
set yrange [   0:  30]

set xlabel "{/:=14 Year }"
```

```
set ylabel "{/:=14 Empty places %}"

l(x) = a*(x-1990)**2 + b*(x-1990) + c

fit l(x) file via a,b,c

plot file using 1:2 with linespoints pointtype 4 pointsize 1.5 linecolor "red" title 'Table
     l(x) title 'Polynomial fit (n = 2)',\
     file2 using 1:2 with linespoints pointtype 4 pointsize 1.5 linecolor "orange" title 'Ex
```

1.1.3. extrapolate

Consulta y calcula N puntos a extrapolar. Guarda en un archivo *.dat*.

```
PROGRAM extrapolate
        implicit none

        integer::i,N

        !ext servirá para almacenar los valores
        !que queremos obtener de la extrapolación.
        integer,allocatable::ext(:)

        !definimos las variables obtenidas en
        !el ajuste del polinomio de grado 2.
        real(8),parameter::a=0.33125d0,b=-3.65d0,c=18d0

        !definimos el desplazamiento de la funcion.
        real(8),parameter::x0=1990d0

        !definimos una variable para el valor de la extrapolacion.
        real(8)::extval

        !Preguntamos cuantos valores quiere extrapolar
        !el usuario, para definir ext.
        write(6,*)'Numero de puntos para extrapolar?'
        read(5,*)N

        allocate(ext(N))
        open(1,file='extrapolations.dat',status='unknown')

        do i=1,N
                write(6,*)'Ingrese año'
                read(5,*)ext(i)
                extval = a*(ext(i)-x0)**2+b*(ext(i)-x0)+c
                !Escribimos el valor extrapolado en un archivo.
                write(1,"(i4,1x,F4.1)")ext(i),extval
        enddo
```

```
        close(1)
        deallocate(ext)

END PROGRAM extrapolate
```

1.1.4. Figure 2

Grafica la Figura 2.

```
file = 'table2.dat'

set xrange [1978:2010]
set yrange [   0:  30]

set xlabel "{/:=14 Year }"
set ylabel "{/:=14 Empty places %}"

l(x) = a*(x-1990)**3 + b*(x-1990)**2 + c*(x-1990) + d

fit l(x) file via a,b,c,d

plot file using 1:2 with linespoints pointtype 4 pointsize 1.5 linecolor "red" title 'Table
     l(x) title 'Polynomial fit (n = 3)'
```

1.2. Pregunta 2

El código está expuesto en el desarrollo del problema.

1.3. Pregunta 3

1.3.1. freeparticle

Realiza todos los cálculos y guardado de datos para la primera parte del problema 3 (partícula libre).

```
PROGRAM freeparticle
        implicit none

        ! Definimos la constante de aceleración gravitatoria:
        real(8),parameter::g=9.80665 !m/s**2

        ! h: altura inicial (en m).
        ! e: coeficiente de inelasticidad.
        ! endt: tiempo de la simulación. Condición secundaria de término.
        real(8)::h,e,endt
```

```
! v: velocidad en todo t.
! v0: Condiciones iniciales (al inicio de la simulación o después
!     de un bote.

! Análogo para y.
real(8)::v,v0
real(8)::y,y0

! totalt: tiempo total transcurrido.
! t     : tiempo dentro de un loop a realizar por cada rebote
! dt    : step del tiempo total transcurrido.
! tmax  : tiempo entre rebotes.
! deltat: se explicará más adelante.
real(8)::totalt,t,dt,tmax,deltat

! Pedimos al usuario una alura inicial.
do while(1.eq.1)
        write(6,*)'Ingrese una altura inicial (positiva) en metros:'
        read(5,*)h
        if(h.gt.0)then
                exit
        ! No permite alturas menores o iguales a 0.
        else
                write(6,*)'Altura inválida'
        endif
enddo

! Pedimos al usuario un coeficiente de restitución inicial.
do while(1.eq.1)
        write(6,*)'Ingrese un coeficiente de restitución inicial:'
        read(5,*)e
        if(e.ge.0 .and. e.le.1)then
                exit
        else
                write(6,*)'Coeficiente inválido'
        endif
enddo

! Pedimos al usuario un tiempo final (en segundos).
! Este tiempo se interpreta como: No hay más botes en t.ge.endt
! Será una condicion secundaria de término.
write(6,*)'Se definirá el tiempo inicial en 0 segundos.'
write(6,*)'Ingrese el tiempo final para la simulación (segundos):'
read(5,*)endt

totalt = 0.0d0
```

```
t  = 0.0d0
dt = 0.05d0

y  = h

! Condiciones iniciales.
y0 = h
v0 = 0.0d0

open(1,file='freep1.dat',status='unknown')

! Comenzamos el loop principal.
do while(totalt.le.endt)

        ! Condición de entrada al subloop por bote:
        ! Calculamos analíticamente el tiempo teórico en el que la
        ! partícula debería tocar el suelo (y = 0).
        ! (Tiempo entre rebotes).
        if(y0.ne.0 .and. v0.eq.0)then
                tmax = sqrt((2*y0)/g)
        endif

        if(y0.eq.0)then
                tmax = (2*v0)/g
        endif

        ! Condición principal de salida.
        ! Para evitar errores en el do while (evitar que no salga del
        ! loop), definimos esta condición.
        ! Si el tiempo entre rebotes es menor o igual que el diferencial
        ! de tiempo que estamos ocupando, termina la simulación.
        if(tmax.le.dt)then
                exit
        endif

        ! Subloop por bote: Calcula y, v para cada t menor a tmax.
        !                                          Guarda la info en 1.
        do while(t.le.tmax)
                write(6,*)'Subloop', totalt
                y = y0 + v0*t - 0.5*(g*t**2)
                v = v0 - g*t

                write(1,*)0.0d0,totalt,y

                t = t+dt
                totalt = totalt+dt
        enddo
```

```
                ! En la última iteración del loop anterior debemos obtener un t
                ! igual o mayor a tmax. En este caso, nos interesa redefinir las
                ! condiciones iniciales del problema, adaptándolas segun el
                ! modelo de choque que nos entregue el parámetro "e".
                ! Esto se entiende como un "nuevo rebote", y a parte de
                ! resetear las condiciones iniciales, resetea el tiempo "t" a 0.
                if(t.gt.tmax)then
                        write(6,*)'Bote. Reseteando parámetros.'

                        ! Nos interesa graficar el momento exacto del rebote. Por lo
                        ! que redefino totalt y guardo ese punto exacto,
                        ! evaluando en tmax.

                        !deltat: Diferencia entre el totalt y el punto de rebote.
                        deltat = tmax-t
                        y = y0 + v0*tmax - 0.5*(g*tmax**2)
                        v = v0 - g*tmax
                        write(1,*)0.0d0,totalt+deltat,y

                        y0 = 0

                        ! Define el choque.
                        v0 = -1*e*v

                        ! Acá es muy conflictivo que dt sea mayor que tmax, por ello
                        ! definimos la condicion principal de salida.
                        t = dt - tmax + t
                        totalt = totalt+dt

                endif

                ! A partir de acá el loop continúa, ahora con t = 0 y las
                ! nuevas condiciones iniciales, se vuelve a entrar en el
                ! subloop por rebote.

                ! Notamos que totalt, a excepción del momento de reseteo de
                ! parámetros, siempre aumenta.



        enddo

        close(1)

END PROGRAM
```

1.3.2. freeparticle2

Realiza todos los cálculos y guardado de datos para la segunda parte del problema 3 (partícula libre).

```
PROGRAM freeparticle2
        implicit none

        ! Definimos la constante de aceleración gravitatoria:
        real(8),parameter::g=9.80665 !m/s**2

        ! h: altura inicial (en m).
        ! e: coeficiente de inelasticidad.
        ! endt: tiempo de la simulación. Condición secundaria de término.
        real(8)::h,e,endt

        ! v: velocidad en todo t.
        ! v0: Condiciones iniciales (al inicio de la simulación o después
        !     de un bote.

        ! Análogo para y.
        real(8)::v,v0
        real(8)::y,y0
        real(8)::x,x0
        real(8)::vx,vx0

        ! totalt: tiempo total transcurrido.
        ! t     : tiempo dentro de un loop a realizar por cada rebote
        ! dt    : step del tiempo total transcurrido.
        ! tmax  : tiempo entre rebotes.
        ! deltat: se explicará más adelante.
        real(8)::totalt,t,dt,tmax,deltat

        ! Pedimos al usuario una alura inicial.
        do while(1.eq.1)
                write(6,*)'Ingrese una altura inicial (positiva) en metros:'
                read(5,*)h
                if(h.gt.0)then
                        exit
                ! No permite alturas menores o iguales a 0.
                else
                        write(6,*)'Altura inválida'
                endif
        enddo
```

```
! Pedimos al usuario un coeficiente de restitución inicial.
do while(1.eq.1)
        write(6,*)'Ingrese un coeficiente de restitución inicial:'
        read(5,*)e
        if(e.ge.0 .and. e.le.1)then
                exit
        else
                write(6,*)'Coeficiente inválido'
        endif
enddo

! Pedimos al usuario un tiempo final (en segundos).
! Este tiempo se interpreta como: No hay más botes en t.ge.endt
! Será una condicion secundaria de término.
write(6,*)'Se definirá el tiempo inicial en 0 segundos.'
write(6,*)'Ingrese el tiempo final para la simulación (segundos):'
read(5,*)endt

write(6,*)'Ingrese una velocidad inicial en el eje x:'
read(5,*)vx0

totalt = 0.0d0

t  = 0.0d0
dt = 0.05d0

y  = h

! Condiciones iniciales.
y0 = h
v0 = 0.0d0

x0 = 0

open(1,file='freep2.dat',status='unknown')

! Comenzamos el loop principal.
do while(totalt.le.endt)

        ! Condición de entrada al subloop por bote:
        ! Calculamos analíticamente el tiempo teórico en el que la
        ! partícula debería tocar el suelo (y = 0).
        ! (Tiempo entre rebotes).
        if(y0.ne.0 .and. v0.eq.0)then
                tmax = sqrt((2*y0)/g)
        endif
```

```
if(y0.eq.0)then
        tmax = (2*v0)/g
endif

! Condición principal de salida.
! Para evitar errores en el do while (evitar que no salga del
! loop), definimos esta condición.
! Si el tiempo entre rebotes es menor o igual que el diferencial
! de tiempo que estamos ocupando, termina la simulación.
if(tmax.le.dt)then
        exit
endif

! Subloop por bote: Calcula y, v para cada t menor a tmax.
!                                          Guarda la info en 1.
do while(t.le.tmax)
        write(6,*)'Subloop', totalt
        y = y0 + v0*t - 0.5*(g*t**2)
        v = v0 - g*t

        ! Agregamos la pos en x.
        x = vx0*totalt

        write(1,*)x,totalt,y

        t = t+dt
        totalt = totalt+dt
enddo

! En la última iteración del loop anterior debemos obtener un t
! igual o mayor a tmax. En este caso, nos interesa redefinir las
! condiciones iniciales del problema, adaptándolas segun el
! modelo de choque que nos entregue el parámetro "e".
! Esto se entiende como un "nuevo rebote", y a parte de
! resetear las condiciones iniciales, resetea el tiempo "t" a 0.
if(t.gt.tmax)then
        write(6,*)'Bote. Reseteando parámetros.'

        ! Nos interesa graficar el momento exacto del rebote. Por lo
        ! que redefino totalt y guardo ese punto exacto,
        ! evaluando en tmax.

        !deltat: Diferencia entre el totalt y el punto de rebote.
        deltat = tmax-t
        y = y0 + v0*tmax - 0.5*(g*tmax**2)
        v = v0 - g*tmax
```

```
                    ! Tenemos que recalcular la posición en x para hacer calzar
                    ! los datos.
                    x = vx0*totalt+deltat

                    write(1,*)x,totalt+deltat,y

                    y0 = 0

                    ! Define el choque.
                    v0 = -1*e*v

                    ! Acá es muy conflictivo que dt sea mayor que tmax, por ello
                    ! definimos la condicion principal de salida.
                    t = dt - tmax + t
                    totalt = totalt+dt

                endif

                ! A partir de acá el loop continúa, ahora con t = 0 y las
                ! nuevas condiciones iniciales, se vuelve a entrar en el
                ! subloop por rebote.

                ! Notamos que totalt, a excepción del momento de reseteo de
                ! parámetros, siempre aumenta.



        enddo

        close(1)

END PROGRAM
```

1.3.3. Figure 3

Grafica los plots en Figura 3. Para graficar los distintos gráficos, cambiar nombre de *file* a aquel que termine con el coeficiente de restitución deseado.

```
file = 'freep1___e0_1.dat'

set yrange [-1:12]

set title "{/:=14 e = 0.1}"

set xlabel "{/:=14 Time [s] }"
```

```
set ylabel "{/:=14 Height [m]}"

plot file using 2:3 with linespoints pointtype 7 pointsize 1 linecolor "red" title 'Table Po
```

1.3.4. Free Particle Animaciones

Realiza las animaciones de las partículas libres a distintos coeficientes de restitución (se cambia al *file* deseado de la misma manera que el archivo anterior).

```
file = 'freep1___e1_0.dat'
t0 = 0
tf = 10
dt = 0.1
set xrange [-1:1]
set yrange [-1:15]
set tics font ",15"
set xlabel "{/:=14 x (m)}"
set ylabel "{/:=14 y (m)}"

n = system(sprintf('cat %s | wc -l', file))

do for [j=1:n] {
    set title 'Iteración '.j
    pause 0.05
    plot file u 1:3 every ::1::j w l lc 'red' lw 2 title 'Trayectoria', \
         file u 1:3 every ::j::j w p lc 'black' pt 7 ps 3 title 'Partícula'
}
```

1.4. Pregunta 4

1.4.1. derivadaraiz1

Calcula numéricamente la derivada de la raíz de x evaluada en $x = 1$, para el parámetro δ fijo. Calcula el error relativo respecto al resultado analítico.

```
program derivadaraiz
        implicit none

        real(8)::valorreal,rerr
        real(8)::n, delta, x, df

        ! Pedimos al usuario el orden de magnitud de delta.
        ! Esperados: (-1, -2, -3, ...)
        write(6,*)'Ingrese el orden de magnitud de delta.'
        read(5,*)n

        x = 1
        delta = 10**n
```

```
        df = (sqrt(x+delta)-sqrt(x))/delta

        write(6,*)df

        valorreal = 1/(2*sqrt(x))

        ! Relative error.
        rerr = (valorreal - df)/valorreal
        rerr = abs(rerr)

        write(6,*)'Error relativo',rerr
end program
```

1.4.2. derivadaraiz2

Calcula numéricamente la derivada de la raíz de x evaluada en $x = 1$, para distintos valores de δ. Calcula el error relativo respecto al resultado analítico.

```
program derivadaraiz
        implicit none

        real(8)::valorreal,rerr
        real(8)::delta, x, df
        real(8)::n

        x = 1
        n = 2
        open(1,file='relativeerr.dat',status='unknown')
        do while(n.le.14.0)
                delta = 10**(-n)

                write(6,*)10**(-2)
                df = (sqrt(x+delta)-sqrt(x))/delta
                write(6,*)df

                valorreal = 1/(2*sqrt(x))

                ! Relative error.
                rerr = (valorreal - df)/valorreal
                rerr = abs(rerr)

                write(6,*)'n',-n,'Error relativo',rerr
                n = n+1

                write(1,*)delta,rerr
        enddo

        close(1)
```

```
end program
```

1.4.3. Figure 4

Grafica la Figura 4.

```
file = 'relativeerr.dat'

set xlabel "{/:=14 [U+1D6FF] }"
set ylabel "{/:=14 Relative Error}"

set logscale x 10
set logscale y 10

plot file using 1:2 with linespoints pointtype 4 pointsize 1.5 linecolor "red" title 'Error
```

1.5. Pregunta 5

1.5.1. sustrac1

Calcula directa e indirectamente la expresión pedida.

```
program sustrac
        implicit none

        real(8)::x
        real(8)::directo,alt

        x = 0.000001
        open(1,file='p5_1.dat',status='unknown')
        do while(x.le.1e-4)
                directo = sqrt(x**2 + 1) - 1

                alt = (x**(2))/(sqrt(x**2 + 1) + 1)


                write(1,*)x, directo, alt, directo/alt

                x = x + (4e-6)
        enddo
        close(1)
end program
```

1.5.2. Figure 5

Grafica la Figura 5. Ambos plots son afines, hay que cambiar el *file*.

```
file = 'p5_2.dat'

set y2tics 0.09,0.01
```

```
set ytics nomirror

set xlabel "{/:=14 x }"
set ylabel "{/:=14 f(x)}"

set y2label "{/:=14 Razón}"

set y2range [0.9999999:1.0000001]

set title 'Doble precisión'

!set logscale x 10
unset logscale x
set logscale y 10
unset logscale y
unset logscale y2


plot file using 1:2 with linespoints pointtype 4 pointsize 1.5 linecolor "red" title 'Cálcul
     file using 1:3 with linespoints pointtype 4 pointsize 1.5 linecolor "blue" title 'Cálcu
     file using 1:4 with linespoints pointtype 7 pointsize 1   linecolor "green" title 'Razo
```

1.6. Pregunta 6

1.6.1. energiamedia

Calcula numéricamente la energía media en la molécula H_2, y mide los tiempos de ejecución.

```
program energiamedia
        implicit none

        real(8),parameter::kb=8.617333262145e-5 !eV/K
        real(8),parameter::w=1.13e13 !rad/s
        real(8),parameter::hbar=6.582119569e-16 !eV*s

        real(8)::beta,hw
        real(8)::T,En,n,totN
        real(8)::expbEn
        real(8)::Z, sE, mE

        real(8)::totNs(5),time1,time2

        integer(8)::i


        T = 1e4 !K

        beta = 1.0/(kb*T) !eV
```

```
        hw    = hbar*w !eV**2*rad


        totNs = (/ 3.0, 4.0, 5.0, 6.0, 9.0 /)

        n = 0
        En = 0

        open(1,file='p6_1.dat',status='unknown')

        ! Primer do, para correr sobre los distintos N.
        do i = 1,5
                totN = 10.0**totNs(i)
                CALL CPU_TIME(time1)

                ! Segundo ciclo. Realiza todas las sumas respetando
                ! los rasgos pedidos en la P6.a)
                do while(n.le.totN)

                        En = hw*(n+(1.0/2.0))

                        ! Calcula la exponencial.
                        expbEn = exp(-1.0*beta*En)

                        ! Sumas parciales.
                        Z  = Z + expbEn
                        sE = sE + En*expbEn

                        n = n+1.0
                enddo

                CALL CPU_TIME(time2)

                ! Energia media: Relación para obtener valor final.
                mE = sE/Z

                write(6,*)mE, time2-time1
                write(1,*)totN, mE, time2-time1
        enddo

        close(1)

end program
```

1.6.2. Figure 6

Grafica la Figura 6.

```
file = 'p6_1.dat'

set y2tics 1e-4,1e1
set ytics nomirror

set xlabel "{/:=14 N}"
set ylabel "{/:=14 Energy [eV]}"
set y2label "{/:=14 Run time [s]}"

set logscale x 10
set logscale y2 10

plot file using 1:2 with linespoints pointtype 6 pointsize 1.5 linecolor "red" title 'Energy
         file using 1:3 with linespoints pointtype 3 pointsize 1.5 title 'Time' axis x1y2
```

1.6.3. Código que no funciona.

```
program energiamedia2
        implicit none

        real(8),parameter::kb=8.617333262145e-5 !eV/K
        real(8),parameter::w=1.13e13 !rad/s
        real(8),parameter::hbar=6.582119569e-16 !eV*s

        real(8)::beta,hw
        real(8)::T,En,n,totN
        real(8)::expbEn
        real(8)::Z, sE, mE

        real(8)::totNs(4), mEs(4),time1,time2

        integer(8)::i




        hw   = hbar*w !eV**2*rad


        totNs = (/ 3.0, 4.0, 5.0, 6.0 /)

        n = 0
        En = 0

        open(1,file='p6_2.dat',status='unknown')
```

```
        T = 500 !K

        ! Primer do, para correr sobre los distintos N.
        do while(T.le.1e4)

                beta = 1.0/(kb*T) !eV
                write(6,*)beta

                do i = 1,4

                        totN = 10.0**totNs(i)
                        !write(6,*)totN

                        ! Segundo ciclo. Realiza todas las sumas respetando
                        ! los rasgos pedidos en la P6.a)
                        do while(n.le.totN)


                                En = hw*(n+(1.0/2.0))

                                ! Calcula la exponencial.
                                expbEn = exp(-1.0*beta*En)

                                ! Sumas parciales.
                                Z  = Z + expbEn
                                sE = sE + En*expbEn

                                n = n+1.0
                        enddo

                        ! Energia media: Relación para obtener valor final.
                        mE = sE/Z
                        mEs(i) = mE

                        !write(6,*)i,mE

                enddo

                !write(6,*)T, mEs(1), mEs(2), mEs(3), mEs(4)
                write(1,*)T, mEs(1), mEs(2), mEs(3), mEs(4)

                T = T + 500
        enddo

        close(1)

end program
```