Guías para Métodos Numéricos

Héctor F. Villafuerte

Julio, 2010

Contenido, parte I

Guía	Breve Descripción
Guía 1	Introducción, representación y manipulación de números, tips, gráficas
Guía 2	Manipulación de texto, operadores relacionales y lógicos, ejecución de un programa, control de flujo, iteración
Guía 3	Declaración de funciones, Método de Newton
Guía 4	Funciones con lambda, Raíces de ecuaciones de una variable (comandos `bisect` y `newton`), aplicaciones
Guía 5	Conceptos de Álgebra Lineal, manejo de estructuras de datos (`arrays`)
Guía 6	Métodos de Jacobi y de Gauss-Seidel; temas relacionados.
Guía 7	Visualización, Descenso del Gradiente y Gradiente Conjugado.
Guía 8	Sage (CAS), Programación Lineal.

Métodos Numéricos **Guía 1**
UVG-MM2010, MM2022

Instrucciones: En sus cursos de matemática se le ha recomendado que ataque los problemas desde varios enfoques: **analítico**, **gráfico**, y **numérico**. En este curso nos interesan particularmente los últimos dos enfoques. Responda las siguientes preguntas dejando constancia clara de su procedimiento; recuerde incluir **gráficas**, **código en Python**, y explicaciones que considere pertinentes.

1. Motivación

1.1. Ventajas de usar Python

La mayoría de nosotros usamos calculadoras científicas en nuestra formación profesional. En mi época teníamos la HP-48 y la TI-82. Ahora los estudiantes usan principalmente la TI-Voyage. No mucho ha cambiado. Estas herramientas funcionan para ejercicios relativamente sencillos. En otras palabras, no *escalan* bien. Las computadoras nos permiten resolver problemas de mucha mayor escala. Si además utilizamos un lenguaje como *Python* que posee una sintaxis clara y mucha funcionalidad ya incluida en distintos módulos, estaremos entonces más cerca de plantear soluciones a problemas.

Otra ventaja de Python es que tiene una licencia de *software libre*. Ésto posiblemente no sea del todo apreciado en una sociedad acostumbrada a la piratería de software, pero creo que las siguientes son descripciones reveladoras:

> "I think, fundamentally, open source does tend to be more stable software. It's the right way to do things. I compare it to *science vs. witchcraft*. In science, the whole system builds on people looking at other people's results and building on top of them. In witchcraft, somebody had a small secret and guarded it; but never allowed others to really understand it and build on it. Traditional software is like witchcraft. In history, witchcraft just died out. The same will happen in software. When problems get serious enough, you can't have one person or one company guarding their secrets. You have to have everybody share in knowledge."*Linus Torvalds*, el padre de Linux
>
> "You can read a theorem and its proof in a book in the library, then you can use that theorem for the rest of your life free of charge, but for many computer algebra systems license fees have to be paid regularly. With this situation two of *the most basic rules of conduct in mathematics are violated*: In mathematics information is passed on free of charge and everything is laid open for checking. Not applying these rules to computer algebra systems that are made for mathematical research means moving in a most undesirable direction. Most important: Can we expect somebody to believe a result of a program that he is not allowed to see? Moreover: Do we really want to charge colleagues in Moldava several years of their salary for a computer algebra system?"*Joachim Neubueser*, profesor RWTH Aachen

Todavía otra ventaja de Python es que se ha convertido en un lenguage bastante *popular*; desde websites que consultamos diariamente hasta producción de películas!

"Python is fast enough for our site and allows us to produce maintainable features in record times, with a minimum of developers."*Cuong Do*, ingeniero de **youtube.com**

"Google has made no secret of the fact they use Python a lot for a number of internal projects. Even knowing that, once I was an employee, I was amazed at how much Python code there actually is in the **Google** source code system."*Guido van Rossum*, creador de Python

"Python plays a key role in our production pipeline. Without it a project the size of **Star Wars**: Episode II would have been very difficult to pull off. From crowd rendering to batch processing to compositing, Python binds all things together."*Tommy Burnette*, Industrial Light and Magic

2. Representación de números

En matemática es usual trabajar con distintos tipos de números; por ejemplo, los naturales $\mathbb{N}$, los enteros $\mathbb{Z}$, los reales $\mathbb{R}$, los complejos $\mathbb{C}$, etc. Empecemos entonces con algunos ejemplos de números en Python.

2.1. Enteros

```
In [4]: 37 + 45
Out[4]: 82
In [5]: 12345678 * 234895764
Out[5]: 2899947465907992L
In [6]: 9876 ** 31
Out[6]: 679224775725704412629230506720745926286574564496096986645708538730872413923464283617375765169404210
```

Python maneja dos tipos de enteros: `int`, `long`.

1. ¿Cuáles cree que son algunas de las ventajas de esta distinción para $\mathbb{Z}$? ¿Desventajas?

   Solución: Piense en la memoria necesaria para almacenar `int` versus `long`. Piense en lo conveniente que resulta no tener que preocuparse del problema de desbordamiento ("overflow") al trabajar con enteros.

2. En Python los enteros tipo `long` poseen *precisión ilimitada*. Experimente: genere números enteros más y más grandes para entender este concepto. En una computadora, ¿cuál cree que es la limitante al trabajar con enteros de precisión ilimitada?

   Solución: Nuevamente piense en la memoria de almacenamiento.

2.2. Reales

Desde un punto de vista numérico es usual representar a $\mathbb{R}$ (en realidad, a un subconjunto de $\mathbb{Q}$) con números de *punto flotante*; `float` en Python. Algunos ejemplos:

```
In [15]: -25.3 + 12.1
Out[15]: -13.200000000000001
In [16]: 1e3
Out[16]: 1000.0
In [17]: 79.8**2
Out[17]: 6368.04
In [18]: 79.8**20
Out[18]: 1.096624203713586e+38
In [19]: 79.8**200
```

3. ¿Cuál es el resultado de la simple operación aritmética $1.0 - 0.9 - 0.1$ en *teoría*? ¿Y en *computadora*? ¿Por qué cree que existe esta diferencia?

> **Solución:** En teoría esperamos que $1 - 0.9 - 0.1$ sea cero. Sin embargo en una computadora con aritmética de *punto flotante* el resultado no es cero (cierto, es muy pequeño pero no es cero). Piense en aritmética de punto flotante como el equivalente computacional de *notación científica* para representar a x:
>
> $$x = (-1)^s b 10^e \qquad \text{Notación científica}$$
> $$x = (-1)^s m 2^e \qquad \text{Punto flotante}$$
>
> donde s determina el signo, b es la base, e el exponente, m es la mantisa. Debería ser claro que hay números que no pueden representarse exactamente usando potencias de dos.

2.3. Complejos

Python incluye soporte para $z \in \mathbb{C}$, i.e. $z = a + bi$ donde $a, b \in \mathbb{R}, i = \sqrt{-1}$. Sin embargo, la sintaxis sigue la convención ingenieril de denotar i por j.

```
In [6]: 3 + 4j
Out[6]: (3+4j)
In [7]: abs(3 + 4j)
Out[7]: 5.0
In [8]: real(3 + 4j)
Out[8]: array(3.0)
In [9]: imag(3 + 4j)
Out[9]: array(4.0)
In [11]: conj(3 + 4j)
Out[11]: (3-4j)
In [12]: (3 + 4j)*(3 - 4j)
Out[12]: (25+0j)
```

4. Empleando la fórmula de Euler $e^{i\theta} = \cos(\theta) + i\sin(\theta)$ experimente en Python con distintos valores de θ.

Solución:

```
In [30]: k = array([4, 3, 2, 1, 0.5])
In [31]: 1/k
Out[31]: array([ 0.25      ,  0.33333333,  0.5       ,  1.        ,  2.        ])
In [32]: 1/k*pi
Out[32]: array([ 0.78539816,  1.04719755,  1.57079633,  3.14159265,  6.28318531])
In [33]: theta = 1/k*pi
In [36]: 1j
Out[36]: 1j
In [37]: 1j*1j
Out[37]: (-1+0j)
In [38]: exp(1j*theta)
Out[38]:
array([  7.07106781e-01 +7.07106781e-01j,
         5.00000000e-01 +8.66025404e-01j,
         6.12303177e-17 +1.00000000e+00j,
        -1.00000000e+00 +1.22460635e-16j,   1.00000000e+00 -2.44921271e-16j])
In [39]: cos(theta) + 1j*sin(theta)
Out[39]:
array([  7.07106781e-01 +7.07106781e-01j,
         5.00000000e-01 +8.66025404e-01j,
         6.12303177e-17 +1.00000000e+00j,
        -1.00000000e+00 +1.22460635e-16j,   1.00000000e+00 -2.44921271e-16j])
In [40]: exp(1j*theta) == cos(theta) + 1j*sin(theta)
Out[40]: array([ True,  True,  True,  True,  True], dtype=bool)
In [46]: allclose( exp(1j*theta) , cos(theta) + 1j*sin(theta) )
Out[46]: True
```

3. Manipulación de números

Existen varias funciones para manipular números, e.g. `int, float, round, ceil, floor, abs`.

5. Elija un número decimal $x \in [0, 10]$ y utilícelo como argumento para cada una de las funciones anteriores.

Solución: Ésto puede resolverse fácilmente usando construcciones de Python como *comprehensión de listas* ("list comprehension"), *cadenas con formatos* ("string formatting"), con el comando `eval`:

```
In [49]: F = ['int', 'float', 'round', 'ceil', 'floor', 'abs']
In [50]: [f for f in F]
Out[50]: ['int', 'float', 'round', 'ceil', 'floor', 'abs']
```

```
In [51]: ['%s(%g)' % (f,7.5) for f in F]
Out[51]: ['int(7.5)', 'float(7.5)', 'round(7.5)', 'ceil(7.5)', 'floor(7.5)', 'abs(7.5)']
In [52]: [eval('%s(%g)' % (f,7.5)) for f in F]
Out[52]: [7, 7.5, 8.0, 8.0, 7.0, 7.5]
In [53]: [eval('%s(%g)' % (f,9.1)) for f in F]
Out[53]: [9, 9.0999999999999996, 9.0, 10.0, 9.0, 9.0999999999999996]
In [54]: [eval('%s(%g)' % (f,2)) for f in F]
Out[54]: [2, 2.0, 2.0, 2.0, 2.0, 2]
```

6. ¿Cuánto es $1/2$ en teoría? ¿Y en la computadora? Éste es un ejemplo del fenómeno de *división entera*. ¿Cómo cree que puede solucionarse este problema[1]?

Solución:

```
In [56]: 1/2
Out[56]: 0
In [57]: 1.0/2
Out[57]: 0.5
In [58]: 1/2.0
Out[58]: 0.5
```

7. Elija un número $x \in [0, 10]$ con, al menos, tres decimales. ¿Cuál es el resultado de `round( x *10)/10.0`? Explique.

Solución: Podemos explorar fácilmente el comportamiento de `round( x *10)/10.0` con una gráfica (gráficas se trata más adelante en esta guía). Noten que `round` no funciona con *arrays*, por lo que hay que *vectorizar* esta función (en el futuro trabajaremos más con la función `vectorize`).

```
In [115]: x = linspace(0,1)
In [116]: round( x*10 )/10.0
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/home/hector/<ipython console> in <module>()
TypeError: only length-1 arrays can be converted to Python scalars
In [117]: Round = vectorize(lambda x: round(x))
In [118]: Round( x*10 )/10.0
Out[118]:
array([ 0. ,  0. ,  0. ,  0.1,  0.1,  0.1,  0.1,  0.1,  0.2,  0.2,  0.2,
        0.2,  0.2,  0.3,  0.3,  0.3,  0.3,  0.3,  0.4,  0.4,  0.4,  0.4,
        0.4,  0.5,  0.5,  0.5,  0.5,  0.6,  0.6,  0.6,  0.6,  0.6,  0.7,
```

[1] Se puede argumentar que ésto no es un problema sino una característica ("feature") de diseño

```
        0.7,  0.7,  0.7,  0.7,  0.8,  0.8,  0.8,  0.8,  0.8,  0.9,  0.9,
        0.9,  0.9,  0.9,  1. ,  1. ,  1. ])
In [120]: plot(x, Round( x*10 )/10.0, 'o')
Out[120]: [<matplotlib.lines.Line2D object at 0xacde7ac>]
In [121]: grid()
In [122]: title('Round( x*10 )/10.0')
Out[122]: <matplotlib.text.Text object at 0xacdeb0c>
```

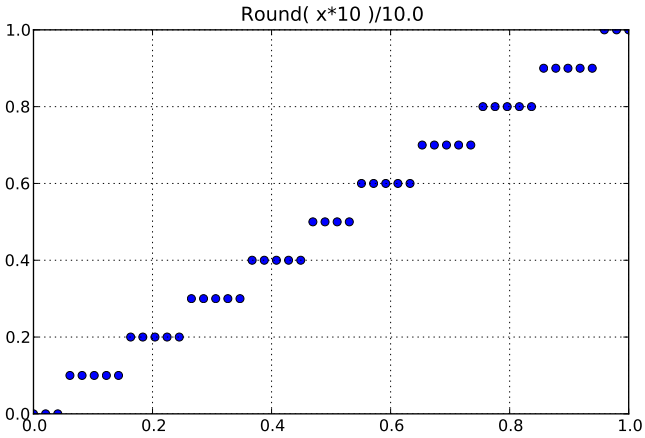

Figura 1: Round

Luego de ver la gráfica, ¿puede ahora explicar el comportamiento de `round( x *10)/10.0`?

4. Tips

4.1. Variables

Para declarar variables algunos lenguajes utilizan la *disciplina estática*[2]. Por ejemplo, en Java, tenemos que **explícitamente** definir el **tipo** de la variable: `int a,b,c`, `String myName`, etc. Otros lenguajes, como Python, utilizan la *disciplina dinámica* para declarar variables; en otras palabras, las variables asumen automáticamente el tipo del valor que se les asigna:

```
In [44]: a = 8
In [45]: a
Out[45]: 8
In [46]: type(a)
Out[46]: <type 'int'>
In [47]: a = 3.14
In [48]: a
```

[2] "Static typing"

```
Out[48]: 3.1400000000000001
In [49]: type(a)
Out[49]: <type 'float'>
```

```
In [37]: a = -9.1
In [38]: b = 2
In [39]: c = b - a
In [40]: c
Out[40]: 11.1
In [41]: z = a + b*1j
In [42]: z
Out[42]: (-9.0999999999999996+2j)
```

8. Elabore una tabla de ventajas y desventajas de las disciplinas estática y dinámica para declaración de variables.

4.2. Tab-completion

Completación usando `[TAB]`:

```
In [1]: m[TAB]
map    max    min    mkdir  mv
```

4.3. Introspección

Introspección –derivado de *introspicĕre*, mirar adentro– es una característica muy útil para **explorar** objetos en Python. Es parte de la magia de tab-completion.

```
In [1]: a = -1 + 1j
In [2]: a.[TAB]
  a.conjugate
  a.imag
  a.real
In [3]: a.conjugate()
Out[3]: (-1-1j)
In [4]: a.imag
Out[4]: 1.0
In [5]: a.real
Out[5]: -1.0
```

4.4. Ayuda

Saber pedir ayuda es una virtud. En Python es muy fácil. Basta con usar `?` al final del nombre de una función (u objeto). Veamos la ayuda para `real` (que usamos anteriormente):

```
In [65]: real?
Type:           function
Base Class:     <type 'function'>
String Form:    <function real at 0x8ebe0d4>
Namespace:      Interactive
File:           /usr/lib/python2.6/dist-packages/numpy/lib/type_check.py
Definition:     real(val)
Docstring:
    Return the real part of the elements of the array.

    Parameters
    ----------
    val : array_like
        Input array.

    Returns
    -------
    out : ndarray
        If `val` is real, the type of `val` is used for the output.  If `val`
        has complex elements, the returned type is float.

    See Also
    --------
    real_if_close, imag, angle

    Examples
    --------
    >>> a = np.array([1+2j,3+4j,5+6j])
    >>> a.real
    array([ 1.,  3.,  5.])
    >>> a.real = 9
    >>> a
    array([ 9.+2.j,  9.+4.j,  9.+6.j])
    >>> a.real = np.array([9,8,7])
    >>> a
    array([ 9.+2.j,  8.+4.j,  7.+6.j])
```

5. Arrays

Trabajar sólo con números (`int, long, float`) es algo limitado. Principalmente estaremos trabajando con **arreglos de números** (`array`), de alguna manera análogos a vectores en $\mathbb{R}^n$. Existen varias formas de definir arreglos, e.g. `array, arange, linspace`, entre otras.

```
In [54]: x = array([1, 2, 3])
```

```
In [55]: y = array([-1, 0, 7])
In [56]: x + y
Out[56]: array([ 0,  2, 10])
In [57]: x * y
Out[57]: array([-1,  0, 21])
In [61]: arange(7)
Out[61]: array([0, 1, 2, 3, 4, 5, 6])
In [62]: arange(7.0)
Out[62]: array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.])
In [63]: linspace(0,1)
Out[63]:
array([ 0.        ,  0.02040816,  0.04081633,  0.06122449,  0.08163265,
        0.10204082,  0.12244898,  0.14285714,  0.16326531,  0.18367347,
        0.20408163,  0.2244898 ,  0.24489796,  0.26530612,  0.28571429,
        0.30612245,  0.32653061,  0.34693878,  0.36734694,  0.3877551 ,
        0.40816327,  0.42857143,  0.44897959,  0.46938776,  0.48979592,
        0.51020408,  0.53061224,  0.55102041,  0.57142857,  0.59183673,
        0.6122449 ,  0.63265306,  0.65306122,  0.67346939,  0.69387755,
        0.71428571,  0.73469388,  0.75510204,  0.7755102 ,  0.79591837,
        0.81632653,  0.83673469,  0.85714286,  0.87755102,  0.89795918,
        0.91836735,  0.93877551,  0.95918367,  0.97959184,  1.        ])
```

6. Gráficas

Ya que tenemos arreglos de números (`arrays`) podemos entonces elaborar gráficas fácilmente.

```
In [1]: x = linspace(-2,2)
In [2]: plot(x, x**2 - 1)
Out[2]: [<matplotlib.lines.Line2D object at 0xa1467ac>]
In [3]: grid()
In [4]: plot(x, sin(x))
Out[4]: [<matplotlib.lines.Line2D object at 0xa10eb6c>]
In [5]: plot(x, cos(x))
Out[5]: [<matplotlib.lines.Line2D object at 0xa102dcc>]
```

Con lo que obtenemos la siguiente gráfica:

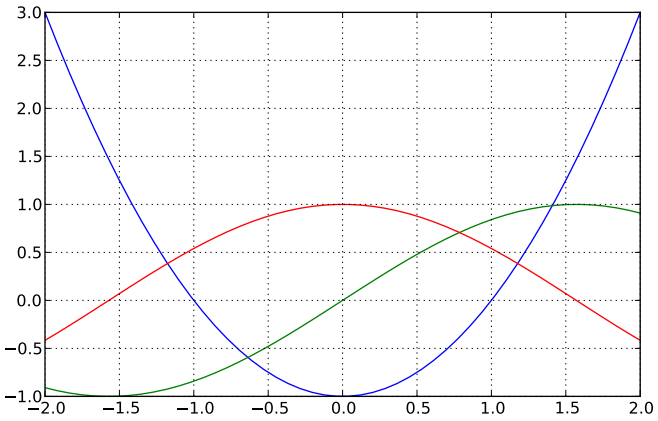

Figura 2: Gráficas básicas

En Python, todo lo que podamos dibujar en una gráfica tiene un comando equivalente. Veamos cómo agregar títulos, etiquetas, leyendas, cambiar ejes, etc.

```
x = linspace(-pi, pi)
plot(x, sin(x), label='seno')
plot(x, cos(x), label='coseno')
grid()
print axis()
axis([-3,3,-1,1])
print axis()

xlabel('Tiempo [s]')
ylabel('Amplitud [V]')
title('Funciones trigonometricas')
legend(loc=2)
```

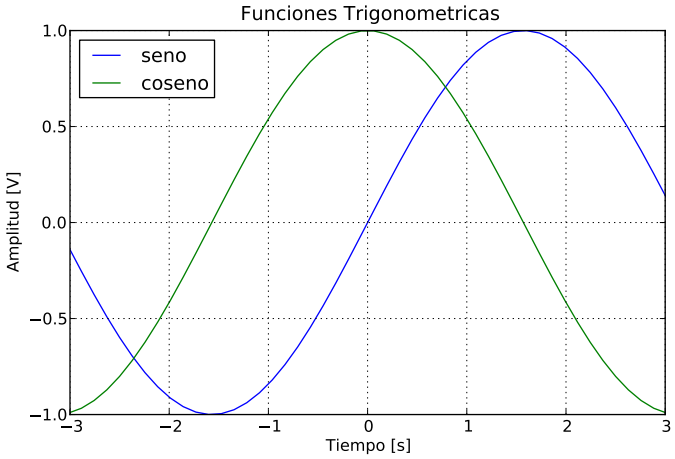

Figura 3: Gráficas básicas

9. Sea p_n el polinomio de Taylor de grado n para $\sin(x)$ centrado en $x = 0$ en el intervalo $x \in [-\pi, \pi]$. Elabore una gráfica que incluya a $\sin(x), p_1, p_3, p_5, p_7$. ¿Qué puede decir de la convergencia de p_n cuando $n \to \infty$? *Ayuda:* la función factorial está dentro del módulo `math`, e.g. `math.factorial(5)` = 120.

Solución:

```
In [137]: x = linspace(-pi, pi, 1000)
In [138]: fact = math.factorial
In [139]: p1 = x
In [140]: p3 = p1 - 1.0/fact(3)*x**3
In [141]: p5 = p3 + 1.0/fact(5)*x**5
In [142]: p7 = p5 - 1.0/fact(7)*x**7
In [143]: plot(x, sin(x), x, p1, x, p3, x, p5, x, p7)
In [145]: grid()
In [146]: axis([-pi,pi, -2,2])
```

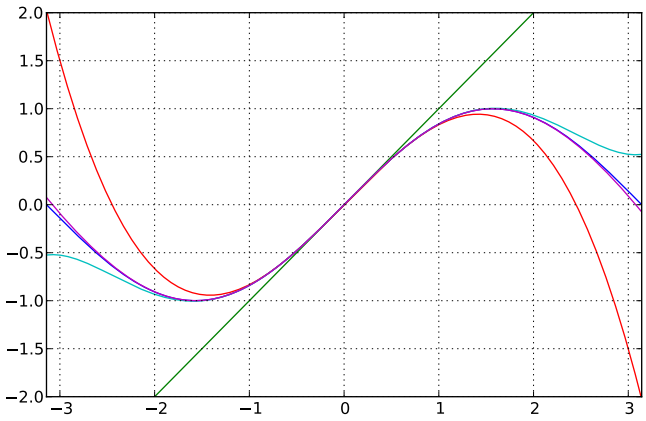

Figura 4: Polinomios de Taylor para $\sin(x)$ alrededor de $x = 0$

10. Estudie el comportamiento de $f(x) = \ln\sqrt{1+x} - \ln\sqrt{x}$, cuando $x \to \infty$. Note que analíticamente obtendrá una forma indeterminada (regla de L'Hôpital). Considere $x \in [10^{10} - 50, 10^{10} + 50]$ y grafique $f(x)$. Usando propiedades de logaritmos encuentre una función simplificada, tal que $g(x) = f(x)$. Grafique $g(x)$. ¿A qué cree que se deben las diferencias entre las gráficas? *Ayuda:* En Python (y muchos otros lenguajes) el logaritmo natural se denota por `log`, e.g. $\ln(2)$ es `log(2)`.

Solución:

```
In [154]: x = linspace(10**10 - 50, 10**10 + 50)
In [155]: f = log(sqrt(1 + x)) - log(sqrt(x))
In [156]: g = 0.5*log(1 + 1/x)
In [158]: plot(x,f, label='f(x)')
```

```
Out[158]: [<matplotlib.lines.Line2D object at 0xb112bec>]
In [159]: plot(x,g, label='g(x)')
Out[159]: [<matplotlib.lines.Line2D object at 0xb222c4c>]
In [160]: grid()
In [161]: legend()
```

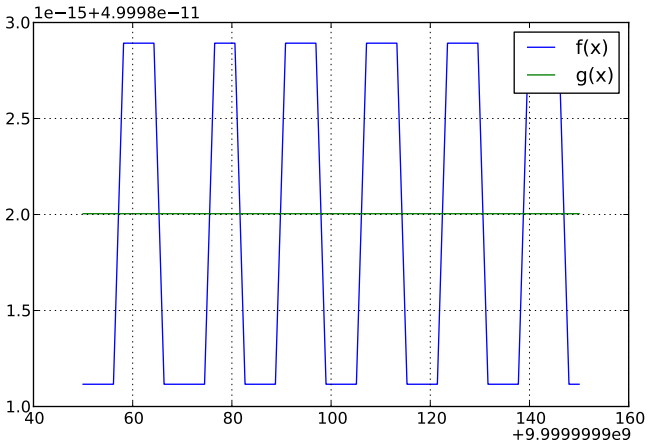

Figura 5: Gráficas de $f(x)$ y $g(x)$. Cancelación sustractiva.

El fenómeno observado en la gráfica anterior se denomina *cancelación sustractiva* y se debe a la pérdida de cifras significativas.

Métodos Numéricos
UVG-MM2010, MM2022

Guía 2

Instrucciones: En sus cursos de matemática se le ha recomendado que ataque los problemas desde varios enfoques: **analítico**, **gráfico**, y **numérico**. En este curso nos interesan particularmente los últimos dos enfoques. Responda las siguientes preguntas dejando constancia clara de su procedimiento; recuerde incluir **gráficas**, **código en Python**, y explicaciones que considere pertinentes.

1. Representación de texto

1. Sea `x` una cadena (`str`) o una lista (`list`). ¿Qué obtenemos al evaluar `x[-1]`? *Ayuda:* en la notación `x[k]`, `k` es el índice.

Solución:

```
In [199]: A = 'abcdefg'
In [200]: list(A)
Out[200]: ['a', 'b', 'c', 'd', 'e', 'f', 'g']
In [201]: L = list(A)
In [202]: L[0]
Out[202]: 'a'
In [203]: L[1]
Out[203]: 'b'
In [204]: L[-1]
Out[204]: 'g'
In [205]: L[-2]
Out[205]: 'f'
```

2. ¿Qué hace el comando `len()`?

Solución:

```
In [206]: len?
Type:           builtin_function_or_method
Base Class:     <type 'builtin_function_or_method'>
String Form:    <built-in function len>
Namespace:      Python builtin
Docstring:
    len(object) -> integer

    Return the number of items of a sequence or mapping.
```

3. ¿Qué hace el comando `join()`?

Solución:

```
In [207]: A.join?
Type:           builtin_function_or_method
Base Class:     <type 'builtin_function_or_method'>
String Form:    <built-in method join of str object at 0xb2f2ae0>
Namespace:      Interactive
Docstring:
    S.join(sequence) -> string

    Return a string which is the concatenation of the strings in the
    sequence.  The separator between elements is S.

In [211]: A.join('--')
Out[211]: '-abcdefg-'
In [212]: '--'.join(A)
Out[212]: 'a--b--c--d--e--f--g'
```

2. Booleanos

4. **Tablas de verdad.**

Solución:

```
In [215]: [P for P in [False, True]]
Out[215]: [False, True]
In [217]: [(P,Q) for P in [False, True] for Q in [False, True]]
Out[217]: [(False, False), (False, True), (True, False), (True, True)]
In [218]: [(P, Q, P and Q) for P in [False, True] for Q in [False, True]]
Out[218]:
[(False, False, False),
 (False, True, False),
 (True, False, False),
 (True, True, True)]
In [219]: [(P, Q, P or Q) for P in [False,True] for Q in [False,True]]
Out[219]:
[(False, False, False),
 (False, True, True),
 (True, False, True),
 (True, True, True)]
```

3. Control de flujo

5. La función Heaviside,

$$H(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$$

Solución: Puede implementarse en un programa que incluya el siguiente código:

```
x = float(raw_input('Heaviside, H(x). Ingrese x: '))
if x < 0:
    print 0.0
else:
    print 1.0
```

al ejecutarlo obtenemos resultados como los siguientes:

```
In [232]: run heaviside.py
Heaviside, H(x). Ingrese x: 8.7
1.0
In [233]: run heaviside.py
Heaviside, H(x). Ingrese x: -4.5
0.0
```

Sin embargo ésto es poco versátil, pues hay que llamar al programa para cada valor que se desee evaluar. Una mejor implementación es usar funciones (ésto es algo que se trabajará con más detalle en el futuro):

```
from pylab import *

def heaviside(x):
    if x<0:
        return 0.0
    else:
        return 1.0

Heaviside = vectorize(heaviside)

x = linspace(-5,5)
plot(x,Heaviside(x), 'o')
grid()
axis([-5,5,-1,2])
show()
```

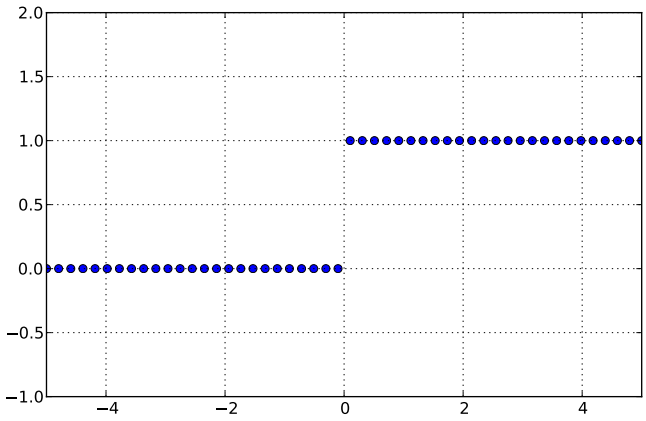

Figura 1: Función Heaviside

6. Dado un N calcule la sumatoria

$$\sum_{k=1}^{N} \frac{1}{k}$$

Solución: Una forma de resolver este problema es empleando *comprehensión de listas*:

```
In [267]: N = 10
In [268]: range(1,N)
Out[268]: [1, 2, 3, 4, 5, 6, 7, 8, 9]
In [269]: [k for k in range(1,N)]
Out[269]: [1, 2, 3, 4, 5, 6, 7, 8, 9]
In [270]: [1.0/k for k in range(1,N)]
Out[270]:
[1.0,
 0.5,
 0.33333333333333331,
 0.25,
 0.20000000000000001,
 0.16666666666666666,
 0.14285714285714285,
 0.125,
 0.1111111111111111]
In [271]: sum([1.0/k for k in range(1,N)])
Out[271]: 2.8289682539682537
In [272]: N = 100
In [273]: sum([1.0/k for k in range(1,N)])
```

```
Out[273]: 5.1773775176396208
```

Otra forma, más corta, es empleando *arrays*:

```
In [275]: N = 10
In [277]: arange(1.0, N)
Out[277]: array([ 1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9.])
In [278]: 1/arange(1.0, N)
Out[278]:
array([ 1.        ,  0.5       ,  0.33333333,  0.25      ,  0.2       ,
        0.16666667,  0.14285714,  0.125     ,  0.11111111])
In [279]: sum(1/arange(1.0, N))
Out[279]: 2.8289682539682537
In [280]: N = 100
In [281]: sum(1/arange(1.0, N))
Out[281]: 5.1773775176396208
```

7. Explique el funcionamiento del siguiente código:

```
from pylab import *

eps = 1.0
while 1.0 != 1.0 + eps:
  print '...', eps
  eps = eps/2.0
print 'final eps:', eps
```

Solución: El resultado de la corrida de este programa es:

```
In [284]: run eps.py
... 1.0
... 0.5
... 0.25
... 0.125
... 0.0625
... 0.03125
... 0.015625
... 0.0078125
... 0.00390625
... 0.001953125
[...]
... 3.5527136788e-15
... 1.7763568394e-15
... 8.881784197e-16
```

```
... 4.4408920985e-16
... 2.22044604925e-16
final eps: 1.11022302463e-16

In [286]: eps
Out[286]: 1.1102230246251565e-16
In [287]: 1.0 + eps
Out[287]: 1.0
In [288]: (1.0 + eps) == 1.0
Out[288]: True
```

En este problema exploramos el *cero* de la computadora (posiblemente el "cero" en su computadora sea distinto a éste). Note como la comparación del final, `(1.0 + eps) == 1.0`, devuelve `True`; ésto significa que, para la computadora, `eps` es cero.

Métodos Numéricos
UVG-MM2010, MM2022

Guía 3

Instrucciones: En sus cursos de matemática se le ha recomendado que ataque los problemas desde varios enfoques: **analítico**, **gráfico**, y **numérico**. En este curso nos interesan particularmente los últimos dos enfoques. Responda las siguientes preguntas dejando constancia clara de su procedimiento; recuerde incluir **gráficas**, **código en Python**, y explicaciones que considere pertinentes.

1. Escriba un programa que despliegue en pantalla la siguiente pirámide:

```
1
2 2
3 3 3
4 4 4 4
```

Solución:

```
In [74]: for k in range(1,4+1):
   ....:     print k*str(k)
   ....:
   ....:
1
22
333
4444
In [75]: for k in range(1,4+1):
   ....:     print ' '.join( k*str(k) )
   ....:
   ....:
1
2 2
3 3 3
4 4 4 4
```

En general, para una pirámide con N filas:

```
N = int(raw_input('Ingrese N, fila: '))

for k in range(1,N+1):
    print ' '.join( k*str(k) )
```

2. Experimente con la siguiente función y describa su funcionamiento:

```
def what(n):
    i = 1
    while i * i < n:
        i = i + 1
    return (i * i == n, i)
```

Solución: Podemos experimentar evaluando `what` con valores individuales, e.g. `what(4), what(5)`, etc. Sin embargo, es mejor evaluar esta función en varios valores de n (piense en un *array*):

```
In [3]: what(1)
Out[3]: (True, 1)
In [4]: what(2)
Out[4]: (False, 2)
In [5]: arange(26.0)
Out[5]:
array([  0.,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  10.,
        11.,  12.,  13.,  14.,  15.,  16.,  17.,  18.,  19.,  20.,  21.,
        22.,  23.,  24.,  25.])
In [7]: [(k, what(k)) for k in arange(26.0)]
Out[7]:
[(0.0, (False, 1)),
 (1.0, (True, 1)),
 (2.0, (False, 2)),
 (3.0, (False, 2)),
 (4.0, (True, 2)),
 (5.0, (False, 3)),
 (6.0, (False, 3)),
 (7.0, (False, 3)),
 (8.0, (False, 3)),
 (9.0, (True, 3)),
 (10.0, (False, 4)),
 (11.0, (False, 4)),
 (12.0, (False, 4)),
 (13.0, (False, 4)),
 (14.0, (False, 4)),
 (15.0, (False, 4)),
 (16.0, (True, 4)),
 (17.0, (False, 5)),
 (18.0, (False, 5)),
 (19.0, (False, 5)),
 (20.0, (False, 5)),
 (21.0, (False, 5)),
 (22.0, (False, 5)),
 (23.0, (False, 5)),
```

```
 (24.0, (False, 5)),
 (25.0, (True, 5))]
In [9]: [(k**2, what(k**2)) for k in arange(1,10.0)]
Out[9]:
[(1.0, (True, 1)),
 (4.0, (True, 2)),
 (9.0, (True, 3)),
 (16.0, (True, 4)),
 (25.0, (True, 5)),
 (36.0, (True, 6)),
 (49.0, (True, 7)),
 (64.0, (True, 8)),
 (81.0, (True, 9))]
```

Concluimos entonces que `what(n)` devuelve las raíces cuadradas de números naturales ($n \in \mathbb{N}$).

3. Sea *aliquot* de x la suma de todos los factores de x, excepto x mismo. Por ejemplo, $\text{aliquot}(12) = 1 + 2 + 3 + 4 + 6 = 16$. Escriba una función que calcule $\text{aliquot}(x)$, para $x \in \mathbb{N}$.

Solución: Utilizaremos *comprehensión de listas* y la notación `lambda` para definir una función en Python:

```
In [91]: range(1,12)
Out[91]: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
In [92]: [k for k in range(1,12)]
Out[92]: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
In [93]: [k for k in range(1,12) if k>7]
Out[93]: [8, 9, 10, 11]
In [94]: [k for k in range(1,12) if 3<k<9]
Out[94]: [4, 5, 6, 7, 8]
In [95]: [k for k in range(1,12) if (12%k == 0)]
Out[95]: [1, 2, 3, 4, 6]
In [96]: sum([k for k in range(1,12) if (12%k == 0)])
Out[96]: 16
In [97]: aliquot = lambda N: sum([k for k in range(1,N) if (N%k == 0)])
In [103]: aliquot(12)   # 1 + 2 + 3 + 4 + 6
Out[103]: 16
In [104]: aliquot(6)    # 1 + 2 + 3
Out[104]: 6
In [105]: aliquot(10)   # 1 + 2 + 5
Out[105]: 8
In [107]: aliquot(7)    # 1
Out[107]: 1
```

4. **Método de Newton.**

(a) Resuelva $x^2 = 612$ empleando iteraciones que *no* utilicen raíces cuadradas.

Solución: En la terminal podemos hacer fácilmente las iteraciones necesarias:

```
In [65]: x = 10.0
In [66]: x = x - (x**2 - 612)/(2*x); x
Out[66]: 35.600000000000001
In [67]: x = x - (x**2 - 612)/(2*x); x
Out[67]: 26.395505617977527
In [68]: x = x - (x**2 - 612)/(2*x); x
Out[68]: 24.790635492455475
In [69]: x = x - (x**2 - 612)/(2*x); x
Out[69]: 24.738688294075324
In [70]: x = x - (x**2 - 612)/(2*x); x
Out[70]: 24.738633753766084
```

(b) Encuentre los ceros de $f(x) = 1 - x^2$ empleando el método de Newton, con $x_0 = 0$.

Solución:

```
In [109]: x = 0.0
In [110]: x = x - (1 - x**2)/(2*x); x
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
ZeroDivisionError: float division
```

Obtenemos el error `ZeroDivisionError` porque $f'(x)$ es cero en el valor inicial $x_0 = 0$.

(c) Utilice el método de Newton para encontrar los ceros de $f(x) = x^3 - 2x + 2$, con $x_0 = 0$.

Solución: Note los resultados de las siguientes iteraciones:

```
In [149]: x = 0.0
In [150]: x = x - (x**3 - 2*x + 2)/(3*x**2 - 2); x
Out[150]: 1.0
In [151]: x = x - (x**3 - 2*x + 2)/(3*x**2 - 2); x
Out[151]: 0.0
In [152]: x = x - (x**3 - 2*x + 2)/(3*x**2 - 2); x
Out[152]: 1.0
In [153]: x = x - (x**3 - 2*x + 2)/(3*x**2 - 2); x
Out[153]: 0.0
```

Claramente estamos en una situación que no nos llevará a los ceros de $f(x)$. Para orientarnos un poco podemos realizar una gráfica:

```
In [144]: x = linspace(-1,1)
In [145]: plot(x, x**3 - 2*x + 2)
```

```
In [146]: grid()
```

Resulta claro ahora que $f(x)$ no tiene ceros en $x \in [-1, 1]$, ¿pero qué pasa fuera de dicho intervalo? Para responder esta pregunta consideremos el siguiente polinomio en donde asumimos que todas las raíces son reales (recuerde que podemos tener raíces complejas):

$$(x - c_1)(x - c_2) \cdots (x - c_n) = 0$$

sustituimos ahora x por $1/x$:

$$\begin{aligned}(1/x - c_1)(1/x - c_2) \cdots (1/x - c_n) &= 0 \\ \left(\frac{1 - c_1 x}{x}\right)\left(\frac{1 - c_2 x}{x}\right) \cdots \left(\frac{1 - c_n x}{x}\right) &= 0 \\ \frac{(1 - c_1 x)(1 - c_2 x)(1 - c_n x)}{x^n} &= 0 \\ (1 - c_1 x)(1 - c_2 x)(1 - c_n x) &= 0\end{aligned}$$

este último polinomio tiene ceros en $1/c_1, 1/c_2, \ldots, 1/c_n$. La importancia de este ejercicio es notar que basta con explorar el intervalo $x \in [-1, 1]$ en el polinomio original y en el *polinomio recíproco*. Por ejemplo, si el polinomio original tiene un cero en $x = 12$, el polinomio recíproco tendrá un cero en $x = 1/12 \approx 0.083\bar{3} \in [-1, 1]$.

Para terminar, notamos en la gráfica de $f(1/x)$ que el polinomio original sólo tiene un cero real:

```
In [147]: plot(x, 1 - 2*x**2 + 2*x**3)
In [148]: 1/(-0.57)
Out[148]: -1.7543859649122808
In [154]: x = -2.0
In [155]: x = x - (x**3 - 2*x + 2)/(3*x**2 - 2); x
Out[155]: -1.8
In [156]: x = x - (x**3 - 2*x + 2)/(3*x**2 - 2); x
Out[156]: -1.7699481865284974
In [157]: x = x - (x**3 - 2*x + 2)/(3*x**2 - 2); x
Out[157]: -1.7692926629059409
In [158]: x = x - (x**3 - 2*x + 2)/(3*x**2 - 2); x
Out[158]: -1.7692923542386998
```

Métodos Numéricos **Guía 4**
UVG-MM2010, MM2022

Instrucciones: En sus cursos de matemática se le ha recomendado que ataque los problemas desde varios enfoques: **analítico**, **gráfico**, y **numérico**. En este curso nos interesan particularmente los últimos dos enfoques. Responda las siguientes preguntas dejando constancia clara de su procedimiento; recuerde incluir **gráficas**, **código en Python**, y explicaciones que considere pertinentes.

1. Ejemplo de solución

El problema de bungee-jumping. Estudios han demostrado que la probabilidad de lesionar una vértebra incrementa significativamente si la velocidad en caída libre excede 36 m/s después de los primeros 4 segundos de caída. Queremos determinar la *masa crítica* dado este escenario.

Empleando desarrollos analíticos tenemos que la velocidad de caída como una función del tiempo $v(t)$ es:

$$v(t) = \sqrt{\frac{gm}{c_d}} \tanh\left(\sqrt{\frac{gc_d}{m}} t\right)$$

donde el c_d [kg/m] es el coeficiente de resistencia aerodinámica, g [m/s^2] es la gravedad, m [kg] es la masa de la persona en caída libre, t [s] es el tiempo.

Considere las siguientes constantes en el problema:

$$\begin{aligned} v &= 36 \text{ [m/s]} \\ g &= 9.81 \text{ [m/s}^2\text{]} \\ t &= 4 \text{ [s]} \\ c_d &= 0.25 \text{ [kg/m]} \end{aligned}$$

Como nuestro interés está en la masa m, empezamos planteando una función $f(m)$ así:

$$f(m) = \sqrt{\frac{gm}{c_d}} \tanh\left(\sqrt{\frac{gc_d}{m}} t\right) - v$$

1.1. Enfoque analítico

Convénzase de la difícil, acaso imposible, que es encontrar los ceros de $f(m)$ desde una perspectiva puramente analítica (algebraica). Pasamos entonces a atacar el problema desde otros enfoques.

1.2. Enfoque gráfico

Para obtener una estimación inicial es recomendable realizar gráficas del problema en cuestión. En este caso podemos graficar $f(m)$ versus m para un intervalo apropiado de la masa m. Acá está el script:

```
from pylab import *

v = 36    # m/s
g = 9.81  # m/s**2
t = 4     # s
cd = 0.25 # kg/m

# f(m) = sqrt(g*m/cd)*tanh(sqrt(g*cd/m)*t) - v
m = linspace(100,200)
f = sqrt(g*m/cd)*tanh(sqrt(g*cd/m)*t) - v

plot(m,f)
grid()
show()
```

Al realizar la gráfica notará que intersecta el eje m cerca de 140 [kg]. Tenemos ahora información suficiente para proceder al análisis numérico.

1.3. Enfoque numérico

Por intuitiva que resulte una gráfica en la mayoría de los casos necesitamos respuestas más acertadas provistas por *algoritmos numéricos*. En el siguiente script póngale especial atención a las funciones `bisect` y `newton` del módulo `scipy.optimize`.

```
from pylab import *
from scipy.optimize import bisect, newton

v = 36    # m/s
g = 9.81  # m/s**2
t = 4     # s
cd = 0.25 # kg/m

# f(m) = sqrt(g*m/cd)*tanh(sqrt(g*cd/m)*t) - v
m = linspace(100,200)
f = sqrt(g*m/cd)*tanh(sqrt(g*cd/m)*t) - v

def F(m):
    return sqrt(g*m/cd)*tanh(sqrt(g*cd/m)*t) - v

print 'Scipy Biseccion:     ', bisect(F, 140, 150)
print 'Scipy Newton-Raphson:', newton(F, 130)

plot(m,f)
grid()
show()
```

Al ejecutar este script obtenemos:

```
Scipy Biseccion:      142.737633108
Scipy Newton-Raphson: 142.737633108
```

Podemos decir entonces que para m mayores a 142 [kg] (314 [lb]) existe un riesgo significativo de lesionar una vértebra al saltar en bungee. Están advertidos.

1. ¿Por qué cree que podemos llamar a la función `newton`, que implementa el método de Newton-Raphson, sin necesidad de pasarle la información de la derivada? *Ayuda:* Vea la documentación de `newton`.

> **Solución:** Porque la función `scipy.optimize.newton` implementa ambos métodos: el Método de Newton-Raphson y el Método de la Secanta. Al no obtener la información de $f'(x)$ entonces automáticamente emplea el método de la Secante.

2. Funciones

Ya en el laboratorio anterior aprendieron a definir funciones en Python empleando la estructura `def`. Ahora veremos otra forma, más breve, para definir funciones. Consideren $f(x) = x^2$, usando `def`:

```
In [53]: def f1(x):
   ....:     return x*x
   ....:
In [54]: f1(3)
Out[54]: 9
In [55]: f1(-4)
Out[55]: 16
In [56]: type(f1)
Out[56]: <type 'function'>
```

Ahora definimos $f(x) = x^2$ con la notación `lambda`:

```
In [57]: f2 = lambda x: x*x
In [58]: f2(3)
Out[58]: 9
In [59]: f2(-4)
Out[59]: 16
In [60]: type(f2)
Out[60]: <type 'function'>
```

Consideremos ahora la sumatoria $\sum k^2$ como una función del límite superior N:

$$f(N) = \sum_{k=1}^{N} k^2$$

Experimentando con el siguiente código llegamos a la definición de $f(N)$ empleando `lambda`:

```
In [62]: range(1,10)
Out[62]: [1, 2, 3, 4, 5, 6, 7, 8, 9]
In [63]: [k**2 for k in range(1,10)]
Out[63]: [1, 4, 9, 16, 25, 36, 49, 64, 81]
In [64]: sum([k**2 for k in range(1,10)])
Out[64]: 285
In [65]: f = lambda N: sum([k**2 for k in range(1,N+1)])
In [66]: f(9)
Out[66]: 285
```

Note además que empleando la notación `lambda` podemos definir funciones que toman más de un parámetro, e.g. $f(x,y) = x^2 + y^2$:

```
In [52]: f = lambda x,y: x**2 + y**2
In [53]: f(3,4)
Out[53]: 25
In [54]: f(-1,3)
Out[54]: 10
```

2. Experimente con varios valores de N en la sumatoria anterior, $f(N) = \sum k^2$, y compare esos resultados con la función cúbica:

$$g(N) = \frac{1}{6}N(N+1)(2N+1)$$

Grafique $f(N)$ y $g(N)$ en el mismo plano. ¿Qué observa? *Ayuda:* para graficar considere el uso de `vectorize`.

Solución: Las gráficas de $f(N)$ y $g(N)$ coinciden porque, en efecto, tenemos que:

$$\sum_{k=1}^{N} k^2 = \frac{1}{6}N(N+1)(2N+1)$$

```
In [106]: suma_cuadrados = lambda N: sum(arange(1,N+1)**2)
In [108]: Suma_Cuadrados = vectorize(suma_cuadrados)
In [109]: N = arange(1,25)
In [112]: plot(N, Suma_Cuadrados(N), 'o')
In [113]: plot(N, N*(N+1)*(2*N+1)/6)
```

3. Raíces de ecuaciones de una variable

3. Encuentre las constantes a, b para las cuales la catenaria

$$y = b \cosh\left(\frac{x-a}{b}\right)$$

pasa por los puntos $(1,1)$ y $(2,3)$.

Solución: Con un poco de álgebra, despejamos para a:

$$\begin{aligned} y &= b\cosh\left(\frac{x-a}{b}\right) \\ \operatorname{arccosh}\left(\frac{y}{b}\right) &= \frac{x-a}{b} \\ b\cdot\operatorname{arccosh}\left(\frac{y}{b}\right) &= x-a \\ a &= x - b\cdot\operatorname{arccosh}\left(\frac{y}{b}\right) \end{aligned}$$

Considerando ahora los puntos $(1,1)$ y $(2,3)$ obtenemos:

$$\begin{aligned} a &= 1 - b\cdot\operatorname{arccosh}\left(\frac{1}{b}\right) \\ a &= 2 - b\cdot\operatorname{arccosh}\left(\frac{3}{b}\right) \end{aligned}$$

es decir

$$\begin{aligned} 1 - b\cdot\operatorname{arccosh}\left(\frac{1}{b}\right) &= 2 - b\cdot\operatorname{arccosh}\left(\frac{3}{b}\right) \\ 0 &= 1 + b\left(\cdot\operatorname{arccosh}\left(\frac{1}{b}\right) - \operatorname{arccosh}\left(\frac{3}{b}\right)\right) \end{aligned}$$

El problema se reduce ahora a encontrar los ceros de

$$f(b) = 1 + b\left(\cdot\operatorname{arccosh}\left(\frac{1}{b}\right) - \operatorname{arccosh}\left(\frac{3}{b}\right)\right)$$

Luego de un poco de experimentación notará que el intervalo de interés está en $b \in (0,1)$:

```
In [133]: f = lambda b: 1 + b*(arccosh(1/b) - arccosh(3/b))
In [147]: b = linspace(0, 1.1)
In [148]: plot(b, f(b))
In [149]: grid()
In [150]: from scipy.optimize import bisect, newton
In [151]: bisect(f, 0.6, 0.8)
Out[151]: 0.7773057038029948
```

```
In [152]: newton(f, 0.7)
Out[152]: 0.77730570380246444
In [156]: b = newton(f, 0.7)
In [160]: a = 1 - b*arccosh(1/b); a
Out[160]: 0.42482219457985526
In [161]: a = 2 - b*arccosh(3/b); a
Out[161]: 0.42482219457985781
```

Por lo que $a \approx 0.4248$ y $b \approx 0.7773$; con estos valores obtenemos una catenaria que pasa por los puntos $(1,1)$ y $(2,3)$:

```
In [166]: x = linspace(0, 2.5)
In [167]: plot(x, b*cosh((x - a)/b))
In [168]: grid()
```

4. En la determinación del pH de un ácido monoprótico se llega a la siguiente ecuación:

$$\frac{V_b}{V_a + V_b} C_b + [H^+] = \frac{K_W}{[H^+]} + \frac{V_a}{V_a + V_b} C_a \frac{1}{1 + \frac{[H^+]}{K_a}}$$

con volúmenes V_a=10mL, V_b=3mL, concentraciones C_a=C_b=0.1M, y constantes de equilibrio $K_a = 10^{-3}, K_W = 10^{-14}$. Se quiere determinar el pH de este ácido. *Ayuda para no-químicos:* $\text{pH} = -\log([H^+])$. *Ayuda con la notación:* $[H^+]$ es la incógnita (puede sustituir por x para emplear una notación mas familiar).

(a) Resuelva para $[H^+]$ empleando el *método de bisección*. Inicie con el intervalo $0.001 \leq [H^+] \leq 0.003$. ¿Cuál es entonces el pH del ácido?

(b) Compruebe su respuesta anterior empleando el *método de Newton* con $[H^+]_0 = 0.001$.

(c) Si no se le hubiera dado la ayuda del intervalo inicial en el método de bisección y el valor inicial en el método de Newton, hubiera podido determinar estos valores con una gráfica de $f([H^+])$ versus $[H^+]$. Realice la gráfica que muestre el cero que encontró.

Solución: Este problema fue tomado de *"Numerical calculus and analytical chemistry"*, Martinez y Guineo.

$$\frac{V_b}{V_a+V_b}C_b + [H^+] = \frac{K_W}{[H^+]} + \frac{V_a}{V_a+V_b}C_a\frac{1}{1+\frac{[H^+]}{K_a}}$$

$$\frac{0.3}{13} + [H^+] = \frac{10^{-14}}{[H^+]} + \frac{1}{13}\frac{1}{1+\frac{[H^+]}{10^{-3}}}$$

$$\frac{0.3}{13} + x = \frac{10^{-14}}{x} + \frac{1}{13}\left(\frac{1}{1+\frac{x}{10^{-3}}}\right)$$

$$\frac{0.3}{13} + x - \frac{10^{-14}}{x} - \frac{1}{13}\left(\frac{1}{1+\frac{x}{10^{-3}}}\right) = 0$$

$$\frac{3}{130} + x - \frac{1}{10^{14}x} - \frac{1}{13}\left(\frac{1}{1+10^3x}\right) = 0$$

Note entonces que:

$$f(x) = \frac{3}{130} + x - \frac{1}{10^{14}x} - \frac{1}{13}\left(\frac{1}{1+10^3x}\right)$$

Empleando el *método de bisección* con a_0 =0.001, b_0 =0.003:

Iteración, k	a_k	b_k	c_k	$f(c_k)$
1	0.001000	0.003000	0.002000	-0.000564
2	0.002000	0.003000	0.002500	0.003599
3	0.002000	0.002500	0.002250	0.001658
4	0.002000	0.002250	0.002125	0.000587
5	0.002000	0.002125	0.002063	0.000022

Con lo cual obtenemos pH$= -\log(0.002063) = 2.69$.

Empleando ahora el *método de Newton* con $[H^+]_0 = x_0$ =0.001, para lo cual es necesario calcular $f'(x)$:

$$f'(x) = 1 + \frac{1}{10^{14}x^2} + \frac{1}{13}\left(\frac{10^3}{(1+10^3x)^2}\right)$$

Iteración, k	x_k
0	0.001000
1	0.001711
2	0.002024
3	0.002059
4	0.002060
5	0.002060

Con lo que nuevamente obtenemos pH$= -\log(0.002060) = 2.69$.

5. Determinar el flujo de fluidos a través de tuberías es un problema frecuente en ciencia e ingeniería. Ejemplos de aplicaciones típicas son el estudio de flujos de líquidos o gases en sistemas de enfriamiento; la circulación

de sangre en las venas; transmisión de nutrientes en sistemas vasculares de plantas, etc. La resistencia al flujo en dichas tuberías está parametrizada por un coeficiente adimensional f llamado *coeficiente de fricción de Darcy* (o coeficiente de rozamiento). Empleando la *ecuación de Colebrook* se puede determinar dicho f:

$$0 = \frac{1}{\sqrt{f}} + 2\log_{10}\left(\frac{\varepsilon}{3.7D} + \frac{2.51}{Re\sqrt{f}}\right)$$

donde ε [m] es la rugosidad, D [m] es el diámetro, Re (adimensional) es el número de Reynolds dado por:

$$Re = \frac{\rho v D}{\mu}$$

donde ρ [kg/m^3] es la densidad del fluido, v [m/s] es la velocidad, μ [N $\cdot$ s/m^2] es la viscosidad dinámica. En el caso del flujo de aire en un tubo delgado tenemos los siguientes parámetros:

$$\begin{aligned}
\varepsilon &= 0.0015\text{mm} \\
D &= 0.005\text{m} \\
v &= 40\text{m/s} \\
\rho &= 1.23\text{kg/m}^3 \\
\mu &= 1.79 \cdot 10^{-5}\text{N} \cdot \text{s/m}^2
\end{aligned}$$

Encuentre el coeficiente de fricción de Darcy f.

Solución: Empezamos definiendo las constantes del problema:

```
eps = 0.0015e-3
D = 0.005
v = 40.0
rho = 1.23
mu = 1.79e-5
Re = rho*v*D/mu

colebrook = lambda f: 1/sqrt(f) + 2*log10(eps/(3.7*D) + 2.51/(Re*sqrt(f)))
f = linspace(0.01, 0.1)

plot(f, colebrook(f))
grid()
show()
```

Con la gráfica vemos que tenemos un cero cerca de $f = 0.02$ (recuerde que acá f representa el coeficiente de fricción de Darcy). Empleamos ahora un método numérico para obtener una mejor aproximación:

```
In [185]: bisect(colebrook, 0.02, 0.04)
Out[185]: 0.028967810171307066
In [186]: newton(colebrook, 0.02)
```

```
Out[186]: 0.028967810171438697
```

Por lo que el coeficiente en este problema es $f \approx 0.0289$.

Métodos Numéricos **Guía 5**
UVG-MM2010, MM2022

Instrucciones: En sus cursos de matemática se le ha recomendado que ataque los problemas desde varios enfoques: **analítico**, **gráfico**, y **numérico**. En este curso nos interesan particularmente los últimos dos enfoques. Responda las siguientes preguntas dejando constancia clara de su procedimiento; recuerde incluir **gráficas**, **código en Python**, y explicaciones que considere pertinentes.

1. Introducción

Seguiremos empleando Python con sus extensiones científicas, e.g. SciPy, NumPy, Matplotlib. Acá presentamos un breve ejemplo en donde se ingresa la matrix

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

se encuentra la matriz inversa A^{-1}, y se multiplica $A \cdot A^{-1}$ para obtener la matriz identidad.

```
In [53]: A = array([[1., 2.], [3., 4.]])

In [54]: A
Out[54]:
array([[ 1.,  2.],
       [ 3.,  4.]])

In [55]: inv(A)
Out[55]:
array([[-2. ,  1. ],
       [ 1.5, -0.5]])

In [56]: dot(A, inv(A))
Out[56]:
array([[ 1.,  0.],
       [ 0.,  1.]])
```

En los siguientes links pueden encontrar más información:

- `http://docs.scipy.org/doc/numpy/reference/routines.linalg.html`
- `http://docs.scipy.org/doc/scipy/reference/linalg.html`
- `http://docs.scipy.org/doc/scipy/reference/sparse.linalg.html`
- `http://wiki.aims.ac.za/mediawiki/index.php/Python:scipy_module:linalg`

2. Álgebra Lineal

1. Dada la matriz A denotamos la *norma* de A como $\|A\|$. Investigue cómo determinar $\|A\|$. *Ayuda:* existen varias definiciones para la norma de una matriz, e.g. la *norma de Frobenius*, la *norma máxima*, etc. Encuentre los comandos para obtener la norma en Python.

2. Investigue el concepto *número de condicionamiento*[1] $\kappa(A)$ de una matriz A. ¿Qué es $\kappa(A)$ si A es una matriz singular?

3. Investigue el concepto *matriz de diagonal estrictamente dominante*[2]. ¿Qué relación tiene este concepto con los métodos (iterativos) de Jacobi y de Gauss-Seidel?

3. Estructuras de datos

4. Investigue cómo crear arreglos (`array`) en Python dadas las siguientes condiciones:
   (a) Un arreglo de longitud (`len`) n para un n dado, lleno de ceros.
   (b) Un arreglo con una *progresión aritmética*. Investigue qué es una progresión aritmética.
   (c) Un arreglo a partir de una lista (`list`).

5. Investigue el concepto de *"array slicing"*. *Ayuda:* `http://en.wikipedia.org/wiki/Array_slicing#1991:_Python`

6. Dada la matriz A de $n \times n$, explique el resultado de la siguiente operación de "slicing": `A[k, arange(n) <> k]` para un entero `k`. Además determine los valores válidos para dicho `k`. *Ayuda:* Considere el siguiente ejemplo.

```
In [28]: A = array([[1,2,3],[4,5,6],[7,8,9]])
In [29]: A[0, arange(3)<>0]
Out[29]: array([2, 3])
In [30]: A[1, arange(3)<>1]
Out[30]: array([4, 6])
In [31]: A[2, arange(3)<>2]
Out[31]: array([7, 8])
In [32]: A
Out[32]:
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

[1] Condition number
[2] Diagonally dominant matrix

Métodos Numéricos
UVG-MM2010, MM2022

Guía 6

Instrucciones: En sus cursos de matemática se le ha recomendado que ataque los problemas desde varios enfoques: **analítico**, **gráfico**, y **numérico**. En este curso nos interesan particularmente los últimos dos enfoques. Responda las siguientes preguntas dejando constancia clara de su procedimiento; recuerde incluir **gráficas**, **código en Python**, y explicaciones que considere pertinentes.

1. Introducción

Para ejemplificar el proceso iterativo, considere el siguiente sistema lineal:

$$\begin{aligned} 12x_0 + 3x_1 - 5x_2 &= 1 \\ x_0 + 5x_1 + 3x_2 &= 28 \\ 3x_0 + 7x_1 + 13x_2 &= 76 \end{aligned}$$

con aproximación inicial

$$\vec{x}^{(0)} = [x_0^{(0)}, x_1^{(0)}, x_2^{(0)}] = [0, 0, 0]$$

En Python procedemos así:

```
In [387]: A = array([ [12.0, 3, -5], [1, 5, 3], [3, 7, 13] ])
In [388]: A
Out[388]:
array([[ 12.,   3.,  -5.],
       [  1.,   5.,   3.],
       [  3.,   7.,  13.]])
In [389]: b = array([1.0, 28, 76])
In [390]: x = zeros(3)
In [391]: n = A.shape[0]
In [392]: x
Out[392]: array([ 0.,  0.,  0.])
In [393]: x = array([ (b[k] - dot( A[k, arange(n)<>k], x[arange(n)<>k] ))/A[k,k]
                      for k in arange(n) ]); x
Out[393]: array([ 0.08333333,  5.6       ,  5.84615385])
In [394]: x = array([ (b[k] - dot( A[k, arange(n)<>k], x[arange(n)<>k] ))/A[k,k]
                      for k in arange(n) ]); x
Out[394]: array([ 1.11923077,  2.07564103,  2.81153846])
...
In [402]: x = array([ (b[k] - dot( A[k, arange(n)<>k], x[arange(n)<>k] ))/A[k,k]
                      for k in arange(n) ]); x
Out[402]: array([ 0.99988778,  2.99667865,  3.99442064])
In [403]: x = array([ (b[k] - dot( A[k, arange(n)<>k], x[arange(n)<>k] ))/A[k,k]
```

```
                    for k in arange(n) ]); x
Out[403]: array([ 0.99850561,  3.00337006,  4.00181432])
```

Observamos entonces que $\vec{x}^{(k)}$ converge a $\vec{x} = [1, 3, 4]$.

Nota: una sola línea de código puede ser bastante densa. Asegúrese de entender todos los conceptos que se están empleando en la *ecuación de recurrencia*; e.g. "array slicing", comprehensión de listas, producto punto (`dot`), conversión de lista a `array`:

```
array([ (b[k] - dot( A[k, arange(n)<>k], x[arange(n)<>k]))/A[k,k] for k in arange(n) ])
```

2. Problemas

1. A continuación se presenta una versión simplificada, pero funcional, del método de Gauss-Seidel. Agregue *comentarios descriptivos* que ayuden a explicar este algoritmo; por ejemplo:

   - $i \leftarrow i + 1$ incrementa el contador de la iteración.

   GAUSS-SEIDEL(A, b, x, i_{max})

   $i \leftarrow 0$
   $n \leftarrow \text{rank}(A)$
   while $i < i_{max}$
   do for $j \leftarrow 1$ **to** n
   do $x_j \leftarrow \frac{1}{A_{jj}} \left(b_j - \sum_{\substack{i=1 \\ i \neq j}}^{n} A_{ji} x_i \right)$
   $i \leftarrow i + 1$
   return x

2. Se tiene el siguiente sistema de ecuaciones

$$\begin{aligned} 4x + y + 2z &= 4 \\ 3x + 5y + z &= 7 \\ x + y + 3z &= 3 \end{aligned}$$

   (a) Empleando la aproximación inicial $\vec{x}^{(0)} = \vec{0}$, resuelva con el método de Jacobi.

   Solución: Siguiendo el ejemplo de la Introducción, podemos ejecutar la ecuación de recurrencia del método de Jacobi, paso a paso, en la terminal:

```
In [46]: A = array([ [4.0, 1, 2], [3,5,1], [1,1,3] ])
b = array([4.0, 7, 3])

n = A.shape[0]
x = zeros(n)
In [52]: x
```

```
Out[52]: array([ 0.,  0.,  0.])
In [53]: x = array([ (b[k] - dot( A[k, arange(n)<>k], x[arange(n)<>k] ))/A[k,k]
                     for k in arange(n) ]); x
Out[53]: array([ 1. ,  1.4,  1. ])
In [54]: x = array([ (b[k] - dot( A[k, arange(n)<>k], x[arange(n)<>k] ))/A[k,k]
                     for k in arange(n) ]); x
Out[54]: array([ 0.15,  0.6 ,  0.2 ])
...
In [70]: x = array([ (b[k] - dot( A[k, arange(n)<>k], x[arange(n)<>k] ))/A[k,k]
                     for k in arange(n) ]); x
Out[70]: array([ 0.49741655,  0.99723626,  0.49757858])
In [71]: x = array([ (b[k] - dot( A[k, arange(n)<>k], x[arange(n)<>k] ))/A[k,k]
                     for k in arange(n) ]); x
Out[71]: array([ 0.50190165,  1.00203435,  0.5017824 ])
In [74]: dot(A,x)
Out[74]: array([ 4.01320572,  7.01765909,  3.00928318])
In [75]: dot(A,x) - b
Out[75]: array([ 0.01320572,  0.01765909,  0.00928318])
```

Notamos que $\vec{x}^{(k)}$ converge a $[0.5, 1.0, 0.5]$. Noten también el último comando ejecutado: `dot(A,x) - b`, equivalente a $A\vec{x} - \vec{b}$. Algunos libros definen el vector *residuo* como $\vec{r} = \vec{b} - A\vec{x}$ y es una forma de determinar qué tan lejos estamos del valor de convergencia, pues esperamos que $\vec{r} \to \vec{0}$.

Ahora bien, en lugar de iterar en la terminal (lo cual puede resultar bastante tedioso), es mejor encapsular esas ideas en una función como la siguiente:

```
def jacobi(A,b,error):
    n = A.shape[0]
    x = zeros(n)
    iter = 0
    res = b - dot(A,x)
    while norm(res) > error:
        x = array([ (b[k] - dot( A[k, arange(n)<>k], x[arange(n)<>k] ))/A[k,k] \
                    for k in arange(n) ])
        res = b - dot(A,x)
        iter = iter + 1
    return x, iter
```

Por supuesto, para que la función anterior funcione asumimos que A es estrictamente dominante por la diagonal y tomamos la simplificación que $\vec{x}^{(0)} = \vec{0}$.

(b) Empleando la aproximación inicial $\vec{x}^{(0)} = \vec{0}$, resuelva con el método de Gauss-Seidel.

Solución: Similar al inciso anterior, podemos ejecutar la ecuación de recurrencia en la terminal. Noten las similitudes y diferencias entre las ecuaciones de recurrencia para los métodos de Jacobi y

Gauss-Seidel:

```
# Jacobi
x = array([ (b[k] - dot( A[k, arange(n)<>k], x[arange(n)<>k] ))/A[k,k]
             for k in arange(n) ])

# Gauss-Seidel
for k in arange(n):
    x[k] = (b[k] - dot( A[k, arange(n)<>k], x[arange(n)<>k] ))/A[k,k]
```

Mejor aún, también podemos crear un función:

```
def gauss_seidel(A,b,error):
    n = A.shape[0]
    x = zeros(n)
    iter = 0
    res = b - dot(A,x)
    while norm(res) > error:
        for k in arange(n):
            x[k] = (b[k] - dot( A[k, arange(n)<>k], x[arange(n)<>k] ))/A[k,k]
        res = b - dot(A,x)
        iter = iter + 1
    return x, iter
```

Al iterar en la terminal, o al emplear la función anterior, nuevamente notamos que $\vec{x}^{(k)}$ converge a $[0.5, 1.0, 0.5]$.

(c) ¿Cuántas iteraciones necesitó en los incisos anteriores para obtener convergencia en 2 cifras significativas?

Solución: Así como está redactada, esta pregunta es ambigua (perdón por éso). Lo más apropiado es utilizar la anterior definición para el residuo $\vec{r} = \vec{b} - A\vec{x}$ y considerar la norma $\|\vec{r}\|$. Claramente, esperamos que $\|\vec{r}\| \to 0$. En Python:

```
In [27]: run ~/docs/uvg/mm2010-met-num/2010/labs/lab06/lab06.py

In [28]: A = array([ [4.0, 1, 2], [3,5,1], [1,1,3] ])
In [29]: b = array([4.0, 7, 3])
In [30]: jacobi(A,b,0.001)
Out[30]: (array([ 0.49993463,  0.99993007,  0.49993873]), 30)
In [32]: gauss_seidel(A,b,0.001)
Out[32]: (array([ 0.50016 ,  0.999936,  0.499968]), 6)
```

Gauss-Seidel es el claro ganador con sólo 6 iteraciones versus 30 de Jacobi.

(d) Encuentre el *radio espectral* ρ de la *matriz de iteración* en el método de Gauss-Seidel. *Ayuda:* Recuerde que la matriz de iteración *no* es la matriz A en $A\vec{x} = \vec{b}$.

Solución: Lo primero que hay que tener claro es que la *matriz de iteración* en el método de Gauss-Seidel **no** es la matriz A. Al considerar la derivación del Método de Gauss-Seidel (Jacobi es similar) tomamos $A = L + D + U$ donde L es la matriz triangular inferior, D es la matriz diagonal y U es la matriz triangular superior. Tenemos entonces:

$$\begin{aligned} A\vec{x} &= \vec{b} \\ (L + D + U)\vec{x} &= \vec{b} \\ (L + D)\vec{x} + U\vec{x} &= \vec{b} \\ (L + D)\vec{x} &= \vec{b} - U\vec{x} \\ \vec{x} &= (L + D)^{-1}\left(\vec{b} - U\vec{x}\right) \end{aligned}$$

Esta última ecuación sugiere un esquema iterativo:

$$\vec{x}^{(k+1)} = (L + D)^{-1}\left(\vec{b} - U\vec{x}^{(k)}\right)$$

que puede interpretarse como la *ecuación de recurrencia*:

$$\vec{x}^{(k+1)} = \vec{c} - M\vec{x}^{(k)}$$

donde

$$\begin{aligned} \vec{c} &= (L + D)^{-1}\vec{b} \\ M &= (L + D)^{-1}U \end{aligned}$$

Noten entonces que el vector $\vec{c}$ es constante, y la matriz M es la matriz de iteración del método de Gauss-Seidel. Ahora en Python, noten los comandos `diag`, `diagflat`, `tril`, `triu`:

```
In [65]: A
Out[65]:
array([[ 4.,  1.,  2.],
       [ 3.,  5.,  1.],
       [ 1.,  1.,  3.]])
In [66]: diag(A)
Out[66]: array([ 4.,  5.,  3.])
In [67]: diagflat(diag(A))
Out[67]:
array([[ 4.,  0.,  0.],
       [ 0.,  5.,  0.],
       [ 0.,  0.,  3.]])
In [68]: tril(A)
Out[68]:
array([[ 4.,  0.,  0.],
```

```
       [ 3.,  5.,  0.],
       [ 1.,  1.,  3.]])
In [69]: triu(A)
Out[69]:
array([[ 4.,  1.,  2.],
       [ 0.,  5.,  1.],
       [ 0.,  0.,  3.]])
```

Tenemos entonces que D es `diagflat(diag(A))`, L+D es `tril(A)` y U es `triu(A) - diagflat(diag(A))`. Con lo que calculamos M y sus eigenvalores:

```
In [73]: A
Out[73]:
array([[ 4.,  1.,  2.],
       [ 3.,  5.,  1.],
       [ 1.,  1.,  3.]])
In [74]: D = diagflat(diag(A)); D
Out[74]:
array([[ 4.,  0.,  0.],
       [ 0.,  5.,  0.],
       [ 0.,  0.,  3.]])
In [75]: U = triu(A) - D; U
Out[75]:
array([[ 0.,  1.,  2.],
       [ 0.,  0.,  1.],
       [ 0.,  0.,  0.]])
In [76]: M = dot( inv(tril(A)), U); M
Out[76]:
array([[ 0.        ,  0.25      ,  0.5       ],
       [ 0.        , -0.15      , -0.1       ],
       [ 0.        , -0.03333333, -0.13333333]])
In [78]: eigvals(M)
Out[78]: array([ 0.        , -0.2       , -0.08333333])
In [80]: max(abs(eigvals(M)))
Out[80]: 0.20000000000000001
In [81]: max(abs(eigvals(M))) < 1
Out[81]: True
```

Finalmente tenemos que el *radio espectral* ρ de la matriz de iteración para Gauss-Seidel es 0.2, que es menor que 1, por lo que esperamos obtener convergencia sin importar el vector inicial $\vec{x}^{(0)}$.

3. La *interpolación polinomial* consiste en determinar el polinomio (único) de grado $n-1$ que pasa por n puntos. Estos polinomios tienen la forma:

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$$

donde los a_k son coeficientes constantes. Una forma directa (trivial) de encontrar dichos coeficientes es generar un sistema con n ecuaciones lineales a partir de los n puntos dados. Queremos entonces determinar el polinomio de cuarto grado

$$f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4$$

que pasa por los siguientes cinco puntos:

x	y
200	0.746
250	0.675
300	0.616
400	0.525
500	0.457

Adicionalmente, determine e interprete el *número de acondicionamiento* κ de la matriz A,

$$\kappa(A) = \|A\| \cdot \|A^{-1}\|$$

Note que $\kappa(A) = \infty$ si A es una matriz singular.

Solución: Empecemos planteando el sistema lineal con el cual obtendríamos los coeficientes del polinomio interpolante. La idea es sustituir los puntos dados:

$$f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4$$

entonces,

$$\begin{aligned} 0.746 &= a_0 + a_1 200 + a_2 200^2 + a_3 200^3 + a_4 200^4 \\ &\vdots \\ 0.457 &= a_0 + a_1 500 + a_2 500^2 + a_3 500^3 + a_4 500^4 \end{aligned}$$

en álgebra matricial:

$$X\vec{a} = \vec{y}$$

$$\begin{bmatrix} 1 & 200 & 200^2 & \cdots & 200^4 \\ \vdots & & & & \\ 1 & 500 & 500^2 & \cdots & 500^4 \end{bmatrix} \begin{bmatrix} a_0 \\ \vdots \\ a_4 \end{bmatrix} = \begin{bmatrix} 0.746 \\ \vdots \\ 0.457 \end{bmatrix}$$

Esta es una *matriz de Vandermonde*, en Python podemos usar el comando `vander`:

```
In [261]: x = array([200, 250, 300, 400, 500], dtype=long)
```

```
In [262]: V = vander(x)
In [263]: A = V.T[::-1].T
In [264]: A
Out[264]:
array([[          1,         200,       40000,     8000000,  1600000000],
       [          1,         250,       62500,    15625000, 3906250000],
       [          1,         300,       90000,    27000000, 8100000000],
       [          1,         400,      160000,    64000000, 25600000000],
       [          1,         500,      250000,   125000000, 62500000000]])
```

Antes de tratar de encontrar el polinomio interpolante, podemos empezar a estudiar las características de la matriz, e.g. su *número de acondicionamiento* κ, el *radio espectral* ρ de esta matriz en el método de Gauss-Seidel. Para encontrar el radio espectral de la matriz de iteración en Python:

```
x = array([200, 250, 300, 400, 500], dtype=long)
V = vander(x)
A = V.T[::-1].T
D = diagflat(diag(A))
U = triu(A) - D
M = dot( inv(tril(A)), U)
ev = eigvals(M)
rho = max(abs(ev))
```

Donde $\rho \approx 0.9985$ está peligrosamente cercano a 1 (recuerde que preferimos que $\rho \ll 1$). Ya ésto no nos da muchas esperanzas para aplicar métodos iterativos en esta matriz. Otra característica que podríamos analizar es el número de acondicionamiento κ. En la computadora vemos que la elección que tomemos para representar números (`int, long, float`) nos lleva a distintas respuestas:

```
def kappa_interpol(typ):
    x = array([200, 250, 300, 400, 500], dtype=typ)
    V = vander(x)
    A = V.T[::-1].T
    kappa = norm(A)*norm(inv(A))
    return kappa

In [318]: kappa_interpol(int)
Out[318]: nan
In [319]: kappa_interpol(long)
Out[319]: nan
In [320]: kappa_interpol(float)
Out[320]: 11711900877602.697
In [321]: kappa_interpol(float64)
Out[321]: 11711900877602.697
```

Vemos entonces que en esta matriz $\kappa \to \infty$, es decir, esta matriz es casi singular. Cualquier intento de resolver este problema numéricamente es un ejercicio en la futilidad.

4. Dado el siguiente sistema:

$$\begin{aligned} 2x - 6y - z &= -38 \\ -3x - y + 7z &= -34 \\ -8x + y - 2z &= -20 \end{aligned}$$

Resuelva empleando el método de Gauss-Seidel. *Ayuda:* De ser necesario, reordene las ecuaciones para obtener convergencia (ésto está relacionado con la idea de *matriz de diagonal estrictamente dominante*).

Solución: En Python:

```
In [326]: A = array([ [2.0, -6, -1], [-3,-1,7], [-8,1,-2] ]); A
Out[326]:
array([[ 2., -6., -1.],
       [-3., -1.,  7.],
       [-8.,  1., -2.]])
In [327]: b = array([-38.0, -34, -20]); b
Out[327]: array([-38., -34., -20.])
In [329]: gauss_seidel(A,b,0.01)
Out[329]: (array([ Inf, -Inf, -Inf]), 320)
```

Lo anterior se debe a que la matriz no es estrictamente dominante por la diagonal. Al permutar las filas tenemos:

```
In [332]: Ap = array([ [-8,1,-2], [2.0, -6, -1], [-3,-1,7] ]); Ap
Out[332]:
array([[-8.,  1., -2.],
       [ 2., -6., -1.],
       [-3., -1.,  7.]])
In [333]: bp = array([-20.0, -38, -34]); bp
Out[333]: array([-20., -38., -34.])
In [334]: gauss_seidel(Ap,bp,0.01)
Out[334]: (array([ 4.,  8., -2.]), 5)
```

Donde $\vec{x}$ converge a $[4, 8, -2]$ en sólo 5 iteraciones.

5. En algunos casos se puede resolver un sistema de ecuaciones *no* lineales empleando los métodos iterativos vistos en clase. Considere el sistema:

$$\begin{aligned} x^2 + 10x + 2y^2 - 13 &= 0 \\ 2x^3 - y^2 + 5y - 6 &= 0 \end{aligned}$$

Para obtener un esquema iterativo reescribimos las ecuaciones anteriores:

$$x = \frac{13 - x^2 - 2y^2}{10}$$
$$y = \frac{6 - 2x^3 + y^2}{5}$$

Resuelva para x, y empleando el método de Gauss-Seidel.

Solución: Podemos empezar ganando intuición geométrica si graficamos estas funciones. En Cálculo 2 se trabajaron funciones de varias variables de la forma $f(x, y) : \mathbb{R}^2 \to \mathbb{R}$. Para graficarlas podemos generar *superficies en el espacio* o tomar *curvas (contornos) de nivel*. Es precisamente con ésta última idea que las graficaremos en Python usando `contour`:

```
z1 = lambda x,y: x**2 + 10*x + 2*y**2 - 13
z2 = lambda x,y: 2*x**3 - y**2 + 5*y - 6

x,y = mgrid[-12:3:100j, -6:6:100j]
cs1 = contour(x,y,z1(x,y), 1,colors='k')
x,y = mgrid[-2:5:100j, -5:5:100j]
cs2 = contour(x,y,z2(x,y), 1,colors='k')
clabel(cs1,fontsize=9)
clabel(cs2,fontsize=9)
axis('equal')
grid()
```

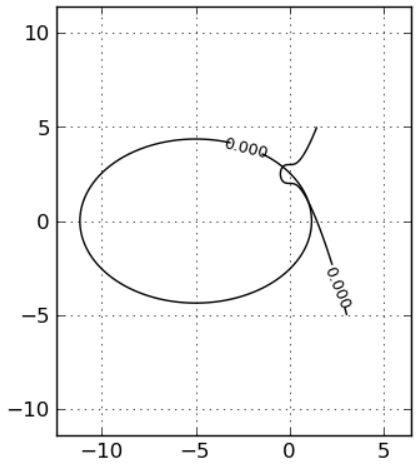

Figura 1: Contornos para $z = 0$

Tenemos entonces dos puntos de intersección. Veamos ahora la contraparte iterativa:

```
def itera_no_lineal(x, y, iter_max=50):
```

```
    iter = 0
    while iter < iter_max:
        x = (13 - x**2 - 2*y**2)/10.0
        y = (6 - 2*x**3 + y**2)/5.0
        iter = iter + 1
    return x, y

In [19]: itera_no_lineal(0,0)
Out[19]: (1.0000006066400808, 0.99999851029323739)
In [20]: itera_no_lineal(-5,0)
Out[20]: (0.99996359464080398, 1.0000893972793692)
In [21]: itera_no_lineal(-5,-2)
OverflowError: (34, 'Numerical result out of range')
```

Encontramos ya uno de los dos puntos de intersección $(1,1)$. Noten también que al tomar la aproximación inicial $(-5,-2)$ tenemos divergencia de la ecuación de recurrencia. Es interesante (y útil) encontrar la región en $\mathbb{R}^2$ en donde tenemos convergencia. En Python:

```
def itera_decide(x, y, iter_max=50):
    iter = 0
    try:
        while iter < iter_max:
            x = (13 - x**2 - 2*y**2)/10.0
            y = (6 - 2*x**3 + y**2)/5.0
            iter = iter + 1
        conv = 1
    except:
        conv = 0
    return conv

x,y = mgrid[-5:5:100j, -5:5:100j]
z = array([itera_decide(float(x[i,j]), float(y[i,j])) for i in range(100) for j in range(100)])
z.shape = (100,100)
matshow(z.T, cmap=cm.hot)
xticks( arange(0,100,10), arange(-5,5))
yticks( arange(0,100,10), arange(5,-5,-1))
grid()
```

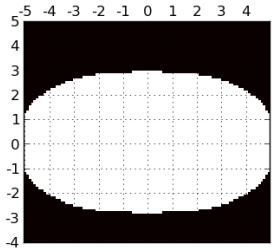

Figura 2: Región de convergencia

Al comparar estas dos gráficas vemos que el otro punto de intersección está peligrosamente cerca de la región de divergencia. Nuevamente los intentos numéricos no nos llevarán muy lejos. Sin embargo, gráficamente podemos estimar que es $(-0.374, 2.88)$.

6. Muestre que A es una *matriz definida positiva*[1]. Es decir, $\vec{x}^T A\vec{x} > 0, \forall \vec{x} \neq 0, \vec{x} \in \mathbb{R}^n$.

$$A = \begin{bmatrix} 12 & 4 & -1 \\ 4 & 7 & 1 \\ -1 & 1 & 6 \end{bmatrix}$$

Solución: Posiblemente la forma más sencilla de demostrar que A es *definida positiva* es a través de la caracterización empleando eigenvalores; si asumimos que $\vec{x}$ es un eigenvector de A, tenemos que,

$$A\vec{x} = \lambda\vec{x}$$

luego, al premultiplicar por $\vec{x}^T$,

$$\begin{aligned} \vec{x}^T A\vec{x} &= \lambda\vec{x}^T\vec{x} \\ &= \lambda\|x\|^2 \end{aligned}$$

por definición, si A es *definida positiva* satisface

$$\vec{x}^T A\vec{x} > 0$$

entonces debe cumplirse que todos los eigenvalores son positivos, i.e. $\lambda > 0$. Obtengamos ahora los eigenvalores en Python:

[1] En inglés, *positive definite matrix*.

```
In [291]: A = array([ [12.0, 4, -1], [4, 7, 1], [-1, 1, 6]])
In [292]: eigvals(A)
Out[292]: array([ 14.23589531,   3.89774951,   6.86635518])
```

Dado que todos los eigenvalores son positivos entonces podemos concluir que A es definida positiva.

7. La matriz A de $n \times n$ satisface $A^4 = -1.6A^2 - 0.64I$, donde I es la matriz identidad. Determine

$$\lim_{n\to\infty} A^n$$

Ayuda: éste es un problema analítico, i.e. no necesita emplear la computadora.

Solución: Empleando un poco de álgebra matricial, tenemos que:

$$\begin{aligned}
A^4 &= -1.6A^2 - 0.64I \\
A^4 + 1.6A^2 + 0.64I &= 0 \\
A^4 + 2(0.8)A^2 + (0.8)^2 I &= 0 \\
(A^2 + 0.8I)^2 &= 0 \\
A^2 + 0.8I &= 0 \\
A^2 &= -0.8I \\
A^3 &= -0.8A \\
A^4 &= -0.8A^2 \\
&= (-0.8)^2 I \\
A^5 &= (-0.8)^2 A
\end{aligned}$$

en general, para $k \in \mathbb{N}$,

$$\begin{aligned}
A^{2k} &= (-0.8)^k I \\
A^{2k+1} &= (-0.8)^k A
\end{aligned}$$

por lo tanto,

$$\lim_{n\to\infty} A^n = 0$$

Métodos Numéricos **Guía 7**
UVG-MM2010, MM2022

Instrucciones: En sus cursos de matemática se le ha recomendado que ataque los problemas desde varios enfoques: **analítico**, **gráfico**, y **numérico**. En este curso nos interesan particularmente los últimos dos enfoques. Responda las siguientes preguntas dejando constancia clara de su procedimiento; recuerde incluir **gráficas**, **código en Python**, y explicaciones que considere pertinentes.

1. Introducción

Varios de los métodos numéricos que hemos tratado en clase tienen justificaciones geométricas muy claras. Por ejemplo, el método del Descenso del Gradiente busca el punto mínimo del paraboloide $f(\vec{x}) = \frac{1}{2}\vec{x}^T A\vec{x} - \vec{b}^T\vec{x}$.

¿Cómo podemos *ver funciones vectoriales* del tipo $\mathbb{R} \to \mathbb{R}^n$? ¿Cómo podemos *ver funciones de varias variables* del tipo $\mathbb{R}^n \to \mathbb{R}$? La siguiente es una breve guía para utilizar herramientas de visualización científica en Python.

Utilizaremos el módulo `mlab` del paquete Mayavi2. Les recomiendo que le den un vistazo a la documentación pues es muy buena:

`http://code.enthought.com/projects/mayavi/docs/development/html/mayavi/mlab.html`

Para corroborar que tienen Mayavi2 instalado, podemos correr uno de los muchos tests de visualización:

```
In [1]: from enthought.mayavi import mlab
In [2]: mlab.test_plot3d()
```

```
In [25]: mlab.test_surf()
In [26]: mlab.axes()
In [27]: mlab.outline()
```

Las funciones `plot3d` y `surf` permiten graficar funciones vectoriales y funciones de varias variables, respectivamente.

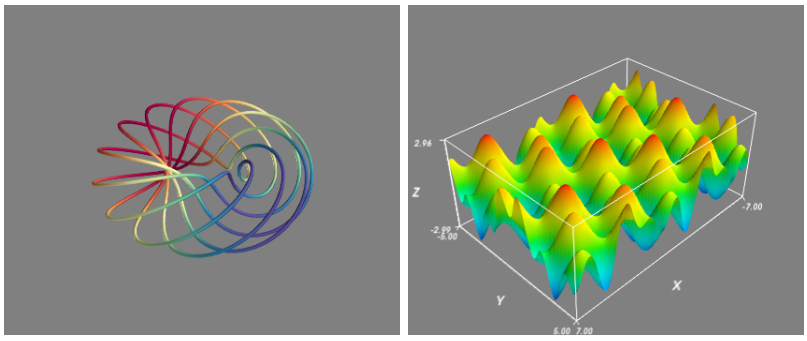

Figura 1: Tests de Mayavi2

Consideremos el ejemplo de la función vectorial $\vec{f}: \mathbb{R} \to \mathbb{R}^3$, para $t \in [0, 10]$;

$$\vec{f}(t) = \begin{bmatrix} \cos(2\pi t) \\ \sin(2\pi t) \\ t \end{bmatrix}$$

En Python:

```
In [15]: t = linspace(0,10)
In [16]: x = cos(2*pi*t)
In [17]: y = sin(2*pi*t)
In [18]: z = t
In [19]: mlab.plot3d(x,y,z)
```

La gráfica anterior puede mejorarse de la siguiente manera:

```
In [20]: t = linspace(0,10,1000)
In [21]: x = cos(2*pi*t)
In [22]: y = sin(2*pi*t)
In [23]: z = t
In [24]: mlab.plot3d(x,y,z, tube_radius=0.1)
```

Figura 2: Espiral en $\mathbb{R}^3$

Veamos ahora un gaussiano en 3D. Consideremos la función de varias variables $f : \mathbb{R}^2 \to \mathbb{R}$ en $[-2, 2] \times [-2, 2]$;

$$f(x, y) = e^{-(x^2+y^2)}$$

En Python, note el uso del comando `mgrid`:

```
In [31]: x,y = mgrid[-2:2:100j, -2:2:100j]
In [32]: z = exp(-(x**2 + y**2))
In [33]: mlab.surf(x,y,z)
In [35]: mlab.contour_surf(x,y,z, contours=15)
```

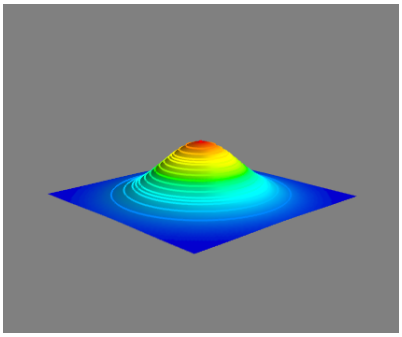

Figura 3: Gaussiano en 3D

Si por ejemplo quisiéramos estudiar el escenario de un *punto silla* podemos tomar la superficie en el espacio (función de varias variables):

$$f(x,y) = x^2 - y^2$$

En Python, note el uso del parámetro `warp_scale`:

```
In [38]: x,y = mgrid[-2:2:100j, -2:2:100j]
In [39]: z = x**2 - y**2

In [40]: mlab.surf(x,y,z)
In [43]: mlab.surf(x,y,z, warp_scale=0.5)
```

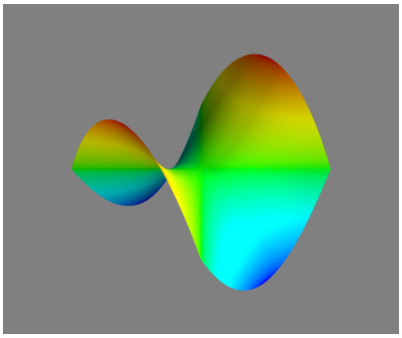

Figura 4: Punto silla en una superficie en el espacio

2. Problemas

1. Luego de estudiar la guía de visualización, grafique la función vectorial

$$\vec{f}(t) = \begin{bmatrix} t\cos(2\pi t) \\ t\sin(2\pi t) \\ t \end{bmatrix}$$

para $t \in [0,5]$.

Solución:

```
In [52]: t = linspace(0,5,1000)
In [53]: x = t*cos(2*pi*t)
In [54]: y = t*sin(2*pi*t)
In [55]: z = t
In [56]: mlab.plot3d(x,y,z, tube_radius=0.1)
```

Figura 5: Espiral expansiva

2. Grafique la función vectorial

$$\vec{f}(t) = \begin{bmatrix} \cos(t) \\ \sin(t) \\ 2 - \sin(t) \end{bmatrix}$$

para $t \in [0, 2\pi]$.

Solución:

```
In [58]: t = linspace(0,2*pi,1000)
In [59]: x = cos(t)
In [60]: y = sin(t)
In [61]: z = 2 - sin(t)
In [62]: mlab.plot3d(x,y,z, tube_radius=0.1)
```

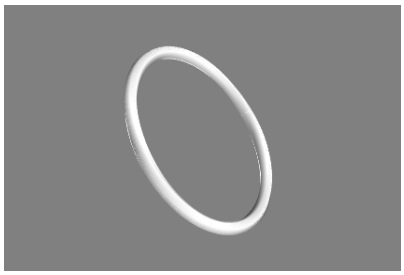

Figura 6: Elipse en $\mathbb{R}^3$

3. Grafique la función de varias variables

$$f(x,y) = 4x^2 + y^2$$

en $[-1,1] \times [-2,2]$.

Solución:

```
In [63]: x,y = mgrid[-1:1:100j, -2:2:100j]
In [64]: z = 4*x**2 + y**2
In [66]: mlab.surf(x,y,z, warp_scale=0.5)
```

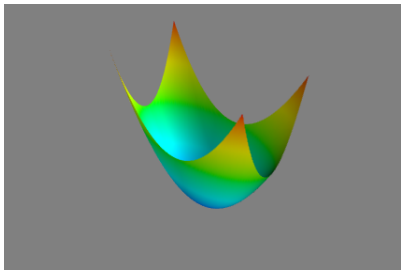

Figura 7: Paraboloide

4. Grafique la función de varias variables

$$f(x,y) = \sin(x-y)$$

en $[-2\pi, 2\pi] \times [-2\pi, 2\pi]$.

Solución:

```
In [67]: x,y = mgrid[-2*pi:2*pi:100j, -2*pi:2*pi:100j]
In [68]: z = sin(x - y)
In [69]: mlab.surf(x,y,z)
```

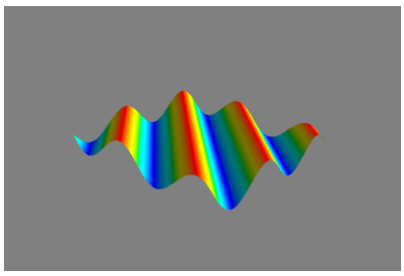

Figura 8: Superficie en el espacio

5. Este es el pseudocódigo del Método del Descenso del Gradiente:

DESCENSO-GRADIENTE$(A, b, x^{(0)}, \text{error})$

$i \leftarrow 0$
$x \leftarrow x^{(0)}$
$r \leftarrow b - Ax$
while $\|r\| > \text{error}$
    do
        $\alpha \leftarrow r^T r / r^T A r$
        $x \leftarrow x + \alpha r$
        $r \leftarrow b - Ax$
        $i \leftarrow i + 1$
return x, i

Esta es una implementación sencilla en Python del Método del Descenso del Gradiente:

```
def grad_desc(A,b,error):
    n = A.shape[0]
    iter = 0
    x = zeros(n)
    res = b - dot(A,x)
    while norm(res) > error:
        alpha = dot(res, res)/dot(res, dot(A,res))
        x = x + alpha*res
        res = b - dot(A,x)
        iter = iter + 1
    return x, iter
```

Por ejemplo,

```
In [115]: A = array([ [12.0, 3, -5], [1,5,3], [3,7,13] ])
In [116]: b = array([1.0, 28, 76])
In [117]: grad_desc(A, b, 0.001)
Out[117]: (array([ 1.00005046,  2.99987191,  4.00007285]), 22)
```

(a) Compare el pseudocódigo y la implementación en Python. Agréguele comentarios descriptivos a la función `grad_desc(A,b,error)`.

6. El Método del Gradiente Conjugado implementa una mejora al Método del Descenso del Gradiente[1]; pues utiliza el vector de búsqueda:

$$\vec{s}^{(k+1)} = \vec{r}^{(k+1)} + \beta^{(k)} \vec{s}^{(k)}$$

[1] Recuerde que en el Método del Descenso del Gradiente tenemos que el vector de búsqueda es simplemente el residuo, $\vec{s} = \vec{r}$.

donde $\beta^{(k)}$ se elige de forma que dos vectores de búsqueda sucesivos sean conjugados, es decir:

$$\begin{aligned}
\vec{s}^{(k+1)^T} A\vec{s}^{(k)} &= 0 \\
\left(\vec{r}^{(k+1)} + \beta^{(k)}\vec{s}^{(k)}\right)^T A\vec{s}^{(k)} &= 0 \\
\vec{r}^{(k+1)^T} A\vec{s}^{(k)} + \beta^{(k)}\vec{s}^{(k)^T} A\vec{s}^{(k)} &= 0 \\
\beta^{(k)} &= -\frac{\vec{r}^{(k+1)^T} A\vec{s}^{(k)}}{\vec{s}^{(k)^T} A\vec{s}^{(k)}}
\end{aligned}$$

Tenemos entonces el siguiente pseudocódigo:

GRADIENTE-CONJUGADO$(A, b, x^{(0)}, error)$

```
1  i ← 0
2  x ← x^(0)
3  r ← b − Ax
4  s ← r
5  while ||r|| > error
6        do
7           α ← s^T r / s^T A s
8           x ← x + αs
9           r ← b − Ax
10          β ← −r^T A s / s^T A s
11          s ← r + βs
12          i ← i + 1
13 return x, i
```

(a) Implemente el anterior pseudocódigo en Python.

Solución: Definimos la función `conj_grad(A,b,error)`,

```
def conj_grad(A,b,error):
    n = A.shape[0]
    iter, x = 0, zeros(n)
    r = b - dot(A,x)
    s = r
    while norm(r) > error:
        q = dot(A,s)
        alpha = dot(s, r)/dot(s, q)
        x = x + alpha*s
        r = b - dot(A,x)
        beta = -dot(r, q)/dot(s, q)
        s = r + beta*s
        iter = iter + 1
    return x, iter
```

(b) Utilice el Método del Gradiente Conjugado para resolver el ejemplo dado en el *Problema 5*. Compare el número de iteraciones necesarias para ambos métodos.

Solución: Por completud, comparemos los cuatro métodos que hemos estudiado para resolver sistemas de ecuaciones lineales: Jacobi, Gauss-Seidel, Descenso del Gradiente y Gradiente Conjugado.

```
In [1]: run /home/hector/docs/uvg/mm2010-met-num/2010/labs/lab07/lab07.py
In [2]: A
Out[2]:
array([[ 12.,   3.,  -5.],
       [  1.,   5.,   3.],
       [  3.,   7.,  13.]])
In [3]: b
Out[3]: array([  1.,  28.,  76.])
In [5]: jacobi(A,b,0.001)
Out[5]: (array([ 0.99996753,  3.00006482,  4.00002598]), 17)
In [7]: gauss_seidel(A,b,0.001)
Out[7]: (array([ 1.00004448,  2.99988947,  4.00004925]), 7)
In [11]: grad_desc(A,b,0.001)
Out[11]: (array([ 1.00005046,  2.99987191,  4.00007285]), 22)
In [16]: conj_grad(A,b,0.001)
Out[16]: (array([ 1.00007491,  2.9998829 ,  4.00012051]), 21)
```

La promesa de que el Método del Gradiente Conjugado es más eficiente no se ve cumplida acá (requirió 21 iteraciones versus 7 iteraciones para el Método de Gauss-Seidel). Lo importante es recordar cuáles son las condiciones que debe cumplir la matrix A, ¿las recuerda? Sin embargo, note que en el siguiente problema el Gradiente Conjugado será un claro ganador.

7. Considere la función de varias variables

$$f(\vec{x}) = \frac{1}{2}\vec{x}^T A\vec{x} - \vec{b}^T\vec{x}$$

$$A = \begin{bmatrix} 3 & 2 \\ 2 & 6 \end{bmatrix} \qquad \vec{b} = \begin{bmatrix} 2 \\ -8 \end{bmatrix}$$

(a) Considere $\vec{x} = [x, y]$, escriba $f(\vec{x})$ ahora como $f(x, y)$.

Solución:

$$f(\vec{x}) = \frac{1}{2}\vec{x}^T A\vec{x} - \vec{b}^T\vec{x}$$
$$f(x, y) = \frac{3}{2}x^2 + 2xy + 3y^2 - 2x + 8y$$

(b) Grafique el paraboloide $f(\vec{x})$.

Solución: En Python, note el uso de la función `points3d` para graficar el punto $(2, -2, -10)$ que es el mínimo del paraboloide (y está relacionado al $\vec{x}$ que obtendremos en los siguientes incisos):

```
In [122]: x,y = mgrid[-10:10:100j, -10:10:100j]
In [123]: z = 3.0/2*x**2 + 2*x*y + 3*y**2 - 2*x + 8*y
In [127]: mlab.surf(x,y,z, warp_scale='auto')
In [128]: mlab.points3d(2,-2,-10)
```

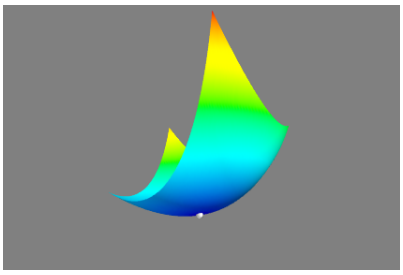

Figura 9: Paraboloide

(c) Encuentre el vector $\vec{x}$ que minimiza el paraboloide empleando el Método del Descenso del Gradiente.

Solución: Ver último inciso.

(d) Encuentre el vector $\vec{x}$ que minimiza el paraboloide empleando el Método del Gradiente Conjugado.

Solución: Ver último inciso.

(e) Compare el número de iteraciones necesarias para ambos métodos.

Solución:

```
In [18]: A = array([[3,2],[2,6]])
In [19]: b = array([2,-8])
In [20]: jacobi(A,b,0.001)
Out[20]: (array([ 1.99975915, -1.99975915]), 12)
In [21]: grad_desc(A,b,0.001)
Out[21]: (array([ 1.99962177, -1.99991404]), 15)
In [22]: grad_desc(A,b,0.001)
Out[22]: (array([ 1.99962177, -1.99991404]), 15)
In [23]: conj_grad(A,b,0.001)
Out[23]: (array([ 2., -2.]), 2)
```

Note la clara ventaja que el Método del Gradiente Conjugado tiene sobre los otros métodos en este caso.

Métodos Numéricos **Guía 8**
UVG-MM2010, MM2022

Instrucciones: En sus cursos de matemática se le ha recomendado que ataque los problemas desde varios enfoques: **analítico**, **gráfico**, y **numérico**. En este curso nos interesan particularmente los últimos dos enfoques. Responda las siguientes preguntas dejando constancia clara de su procedimiento; recuerde incluir **gráficas**, **código en Python**, y explicaciones que considere pertinentes.

1. Introducción

En este laboratorio cubriremos brevemente un aspecto técnico (Sage) y un aspecto teórico (programación lineal).

1.1. Sage (técnico)

Sage es una distribución de software matemático basada en Python. A continuación se detallan los pasos necesarios para crear un usuario en el servidor en línea de Sage (en University of Washington, Seattle). Elijan **uno** de los siguientes servidores (dependiendo de la carga presente, uno puede ser más rápido que el otro):

- `http://www.sagenb.org/`
- `http://alpha.sagenb.org/`

Tendrá una página como la siguiente:

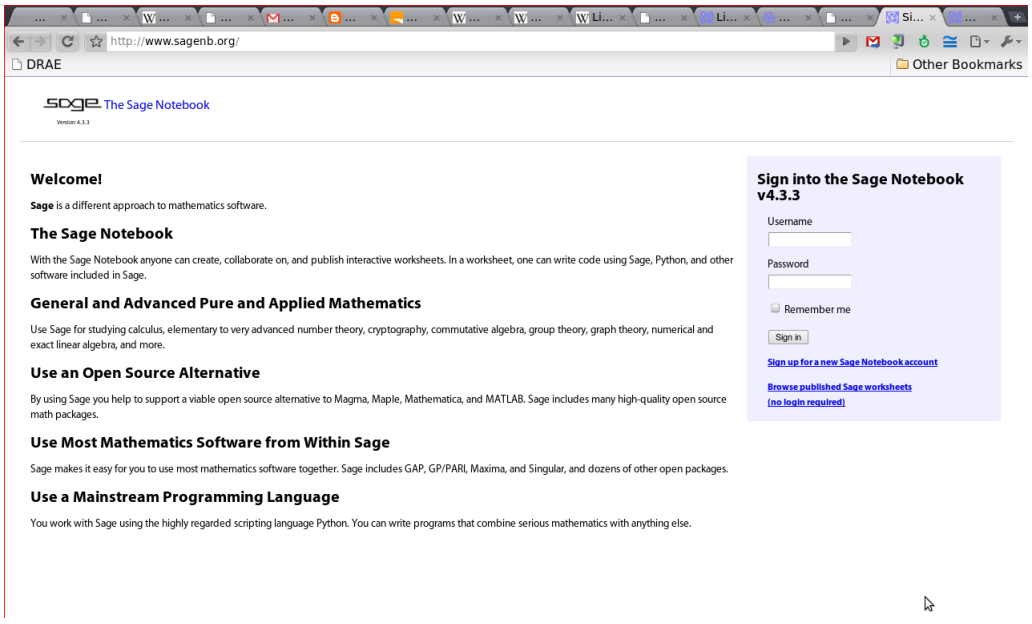

Figura 1: Sage, página de inicio

Siga el link para crear una nueva cuenta: `Sign up for a new Sage Notebook account`. Obtendrá la siguiente página:

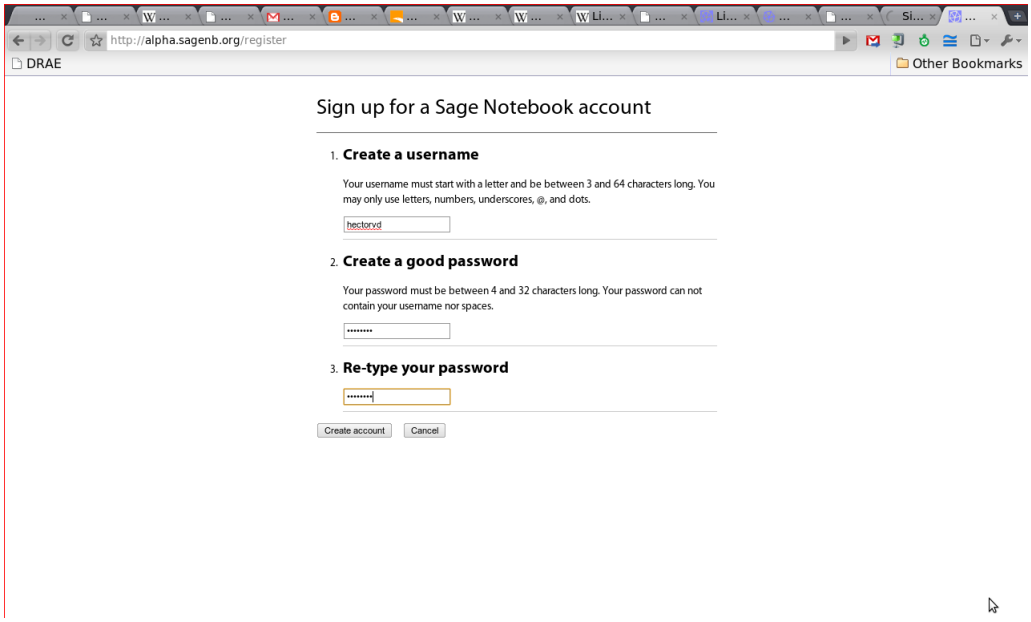

Figura 2: Sage, nueva cuenta

Ingrese la información necesaria. Luego entrará al *Notebook* de Sage, en donde podrá organizar su trabajo en distintas *Worksheets*. Cree una nueva worksheet haciendo click en `New Worksheet`, como se muestra en la figura:

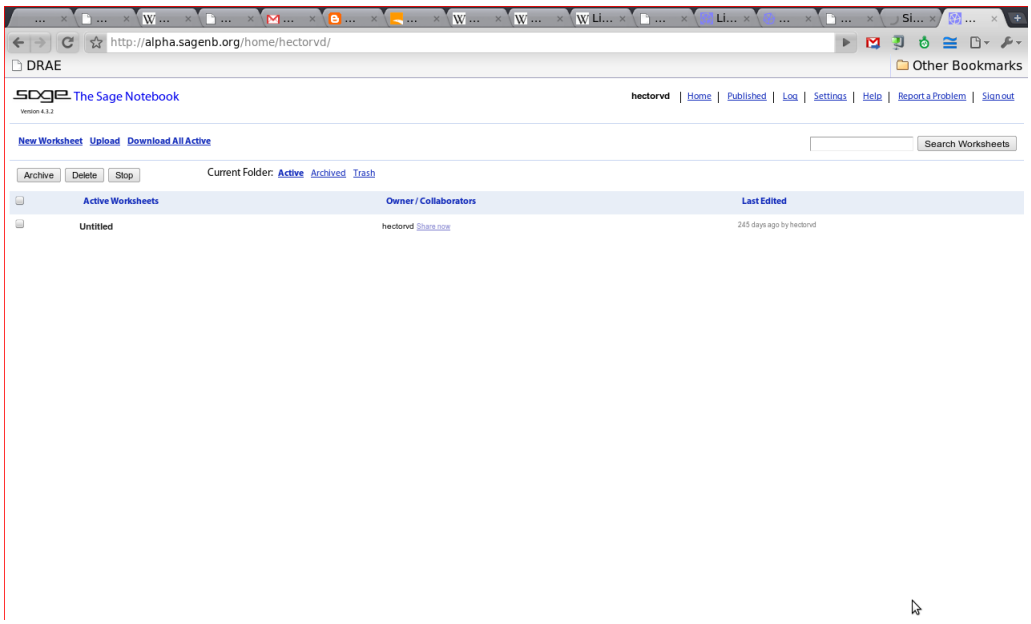

Figura 3: Sage, Notebook

En cada worksheet puede empezar a ingresar comandos en las celdas (*cells*). Por ejemplo en la siguiente figura se muestra una worksheet cuya primera celda tiene el comando:

```
1 integrate(sin(x)*exp(x))
```

en notación matemática, ésto equivale a la integral indefinida:

$$\int \sin(x)e^x \, dx$$

Nota importante: a diferencia de los métodos numéricos que hemos estado trabajando, Sage maneja álgebra simbólica; en otras palabras, Sage es un CAS[1].

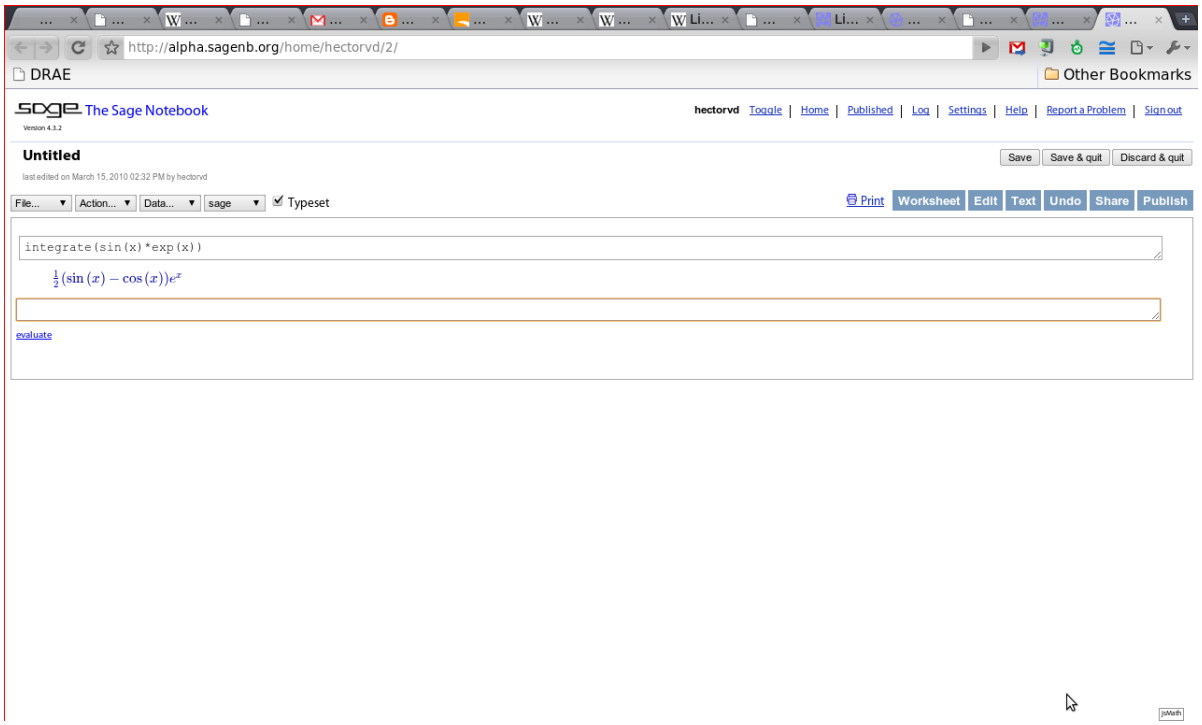

Figura 4: Sage, worksheet

Nota: para evaluar cada celda puede hacer click en `evaluate`, o bien, puede presionar Shift + Enter.

Con ésto basta para empezar a trabajar la siguiente sección; sin embargo, si desea más información acerca de Sage puede tomar las siguientes referencias:

- `www.sagemath.org`
- `williamstein.org/home/wdj/teaching/calc1-sage/an-invitation-to-sage.pdf`

1.2. Programación Lineal (teórico)

Imagine que trabaja en una línea de producción de chocolates. Tienen dos tipos: Chocolate Fino y Chocolate Especial. El Fino cuesta Q6, el Especial cuesta Q1 (al público no le queda claro por qué este chocolate es *especial*, pero ése es otro problema). Debido a cuestiones de proveedores, no puede producir más de 300 chocolates Finos al día; ni más de 200 al día del Especial. Por limitaciones en la máquina empacadora, no puede procesar más de 400 chocolates al día. Ahora la pregunta, ¿cuántos chocolates de cada clase tenemos que producir para

[1] Computer Algebra System

maximizar nuestros ingresos? En notación matemática, tenemos entonces la siguiente *función objetivo* (donde c_f, c_e denotan la cantidad diaria de cada tipo de chocolate):

$$\max(6\,c_f + c_e)$$

sujeto a las siguientes *restricciones*,

$$\begin{aligned} c_f &\leq 300 \\ c_e &\leq 200 \\ c_f + c_e &\leq 400 \\ c_f, c_e &\geq 0 \end{aligned}$$

Que puede reescribirse en notación matricial. La función objetivo es ahora:

$$\max\left(\begin{bmatrix}6\\1\end{bmatrix}^T \begin{bmatrix}c_f\\c_e\end{bmatrix}\right)$$

sujeto a las restricciones:

$$\begin{bmatrix}1 & 0\\0 & 1\\1 & 1\\-1 & 0\\0 & -1\end{bmatrix} \begin{bmatrix}c_f\\c_e\end{bmatrix} \leq \begin{bmatrix}300\\200\\400\\0\\0\end{bmatrix}$$

En breve, tenemos la función objetivo:

$$\vec{c}^T\,\vec{x}$$

sujeto a las restricciones,

$$A\,\vec{x} \leq \vec{b}$$

Nota importante: Existen varios algoritmos para resolver estos problemas de programación lineal; sin embargo, no entraremos en detalles. Estos problemas de optimización pueden involucrar maximizar o minimizar la función objetivo. Una forma sencilla de utilizar el siguiente software es considerar que lo único que hace es minimizar $\vec{c}^T\,\vec{x}$; así que si necesita maximizar puede considerar la función objetivo $-\vec{c}^T\,\vec{x}$. Note también que ingresa la matriz transpuesta, A^T en lugar de A. En una nueva worksheet de Sage ingrese las siguientes celdas:

```
RealNumber=float
Integer=int
from cvxopt.base import matrix
from cvxopt import solvers
```

```
c = matrix([6.0, 1])
A = matrix([[1.0, 0, 1, -1, 0], [0, 1, 1, 0, -1]])
b = matrix([300.0, 200, 400, 0, 0])
```

```
sol = solvers.lp(-c,A,b)
```

```
print sol['x']
```

Con lo que obtenemos que el máximo de los ingresos los alcanzamos al producir 300 de chocolate fino y 100 de chocolate especial: Q1900.

2. Preguntas

1. Debido a un reciente estudio de mercado se ha decidido empezar a producir un nuevo chocolate: Maya. Su precio es de Q13 (se dice que es orgánico, ecológico, etc.). Tiene ahora el siguiente escenario:

$$\max(6c_f + c_e + 13c_m)$$

sujeto a las siguientes *restricciones*,

$$\begin{aligned} c_f &\leq 300 \\ c_e &\leq 200 \\ c_f + c_e + c_m &\leq 400 \\ c_f + 3c_m &\leq 600 \\ c_f, c_e, c_m &\geq 0 \end{aligned}$$

(a) Escriba este problema en notación matricial.

Solución: La nueva función objetivo:

$$\max \left(\begin{bmatrix} 6 \\ 1 \\ 13 \end{bmatrix}^T \begin{bmatrix} c_f \\ c_e \\ c_m \end{bmatrix} \right)$$

sujeto a las restricciones:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 0 & 3 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} c_f \\ c_e \\ c_m \end{bmatrix} \leq \begin{bmatrix} 300 \\ 200 \\ 400 \\ 600 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

(b) Resuelva este problema en Sage.

Solución:

```
from cvxopt.base import matrix
from cvxopt import solvers

c=matrix([6.0, 1, 13])
A=matrix([[1.0, 0, 1, 1, -1,  0, 0],
          [  0, 1, 1, 0,  0, -1, 0],
          [  0, 0, 1, 3,  0,  0,-1]])
b=matrix([300.0, 200, 400, 600, 0, 0, 0])
sol = solvers.lp(-c,A,b)
print sol['x']
```

Con lo que obtenemos:

```
[ 3.00e+02]
[ 6.16e-07]
[ 1.00e+02]
```

Ésto se interpreta entonces como $c_f = 300, c_e = 0, c_m = 100$; con lo que nuestros ingresos (la función objetivo) son de Q3100.

2. Usted conoce al chef del restaurante La Fritura Feliz. Es un buen lugar para disfrutar de yuca frita, aritos de cebolla, carnitas, chicharrones, pollo frito, arroz frito, helado tempura, y así por el estilo. Lamentablemente el restaurante ha recibido críticas que la dieta no es del todo saludable, así que el chef decidió incluir algunos vegetales: zanahoria, repollo y pepino. La Oficina de Inspección Nutricional (OIN) le proporcionó algunos de los siguientes datos:

	Zanahoria	Repollo	Pepino	Requisito por plato
Vitamina A [mg/kg]	35	0.5	0.5	0.5 mg
Vitamina C [mg/kg]	60	300	10	15 mg
Fibra dietética [g/kg]	30	20	10	4 g
Precio [Q/kg]	0.75	0.5	0.15	-

Denote los kilogramos de zanahoria, repollo y pepino con x_1, x_2, x_3, respectivamente. El objetivo es minimizar el precio adicional por plato y cumplir con los requisitos de la OIN.

(a) Plantee la *función objetivo* en términos de las cantidades de vegetales x_1, x_2, x_3.

Solución: $\min(0.75x_1 + 0.5x_2 + 0.15x_3)$

(b) Plantee las *restricciones* a las cuales está sujeta la función objetivo.

Solución:

$$\begin{aligned} 35x_1 + 0.5x_2 + 0.5x_3 &\geq 0.5 \\ 60x_1 + 300x_2 + 10x_3 &\geq 15 \\ 30x_1 + 20x_2 + 10x_3 &\geq 4 \\ x_1, x_2, x_3 &\geq 0 \end{aligned}$$

(c) Escriba este problema en notación matricial.

Solución: La función objetivo:

$$\max \left(\begin{bmatrix} 0.75 \\ 0.5 \\ 0.15 \end{bmatrix}^T \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \right)$$

sujeto a las restricciones:

$$\begin{bmatrix} 35 & 0.5 & 0.5 \\ 60 & 300 & 10 \\ 30 & 20 & 10 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \geq \begin{bmatrix} 0.5 \\ 15 \\ 4 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Note la desigualdad ($\geq$), ésto requirirá que multipliquemos todo por -1.

(d) Resuelva este problema en Sage.

Solución:

```
from cvxopt.base import matrix
from cvxopt import solvers

c=matrix([0.75, 0.5, 0.15])
A=matrix([[ 35,  60, 30, 1, 0, 0],
          [0.5, 300, 20, 0, 1, 0],
          [0.5,  10, 10, 0, 0, 1]])
b=matrix([0.5, 15, 4, 0, 0, 0])
sol = solvers.lp(c,-A,-b)
print sol['x']
```

Con lo que obtenemos:

```
[ 9.53e-03]
[ 3.83e-02]
[ 2.95e-01]
```

Ésto se interpreta entonces como 9.5 g de zanahoria, 38 g de repollo, 295 g de pepino; con lo que el precio adicional por plato (la función objetivo) es de Q0.07.