0.1 Ejercicios de Scilab

1. Escriba una función `[A, info] = Heron(a, b, c)` que calcula, usando la fórmula de Herón de Alejandría, el área de un triángulo cuyos lados miden `a`, `b`, `c`. El parámetro de salida `info` valdrá 1 si los datos están bien.

   En esta y en todas las funciones que usted escriba:

   - Si los datos pueden ser inadecuados o si el proceso puede no acabar satisfactoriamente debe haber un parámetro de salida que lo indique, por ejemplo `info`.
   - En los comentarios iniciales debe quedar claro lo que hace la función, el método utilizado, el significado de los parámetros de entrada (datos) y de salida (resultados).
   - Cuando se presente algún error no previsto, además de influir en el parámetro `info`, la función debe escribir un aviso indicando el tipo de error, `printf(... )`.

2. Escriba una función `d = dist(x, y)` que calcula la distancia entre dos puntos.

3. Escriba una función `a = area(x, y, z)` que calcula el área de un triángulo cuyos vértices son `x`, `y`, y `z`.

4. ¿Cual es el área del triángulo cuyos vértices están dados por los 6 últimos dígitos de su documento de identidad?

5. Escriba una función `[xmin, fmin] = minimo(f, a, b, h)` que halla el minimizador de una función en un intervalo, simplemente comparando todos los valores $f(a)$, $f(a+h)$, $f(a+2h)$, ... Muestre el resultado de su utilización en un ejemplo específico.

6. Escriba una función `[r, info] = raiz(f, a, b, h, eps)` que halla una raíz de la función en el intervalo, simplemente averiguando si entre los los valores $f(a)$, $f(a+h)$, $f(a+2h)$, ... hay alguno tal que $|f(x)| \leq \varepsilon$. Muestre el resultado de su utilización en un ejemplo específico.

7. Escriba una función `[a, info] = ordenaSegun(a, j)` que ordena de manera ascendente las filas de una matriz según los elementos de la columna j. Muestre el resultado de su utilización en un ejemplo específico.

0.2 Procesos inestables

1. En un archivo `.sci` escriba una función `function y = redDec(x, n)` que redondea a x con n cifras decimales.

2. Verifique si la siguiente función redondea un número con n cifras significativas.

```
function y = fun_xyz(x, n)
  y = 0.0
  if x == 0, return, end

  if x >= 0
    signo = 1
  else
    signo = -1
    x = -x
  end

  m = ceil( log10(x) )
  k = 10^m
  t = x/k
  c = 10^n
  y = round(t*c)/c
  y = y*k*signo
endfunction
```

   Pruébela varias veces, con número positivos, negativos, con cero, con números cercanos a cero, con número grandes, con números negativos de valor absoluto grande, ...

3. Escriba una función `function z = sumaRed(x, y, n)` que redondea a x y a y, efectúa la suma $x+y$ y redondea el resultado con n cifras significativas.

4. Escriba funciones
   ```
   function z = prodRed(x, y, n)
   function z = restaRed(x, y, n)
   function z = divRed(x, y, n)
   function z = raiz2Red(x n)
   ```
   que efectúan el producto, la resta, la división y la raiz cuadrada con n cifras significativas.

5. Considere la sucesión definida por

$$\begin{aligned} x_0 &= 1 \\ x_1 &= 1/3 \\ x_n &= \frac{13}{3}x_{n-1} - \frac{4}{3}x_{n-2}, \quad n \geq 2. \end{aligned} \tag{1}$$

   Demuestre por inducción que

$$x_n = \frac{1}{3^n}, \quad n = 0, 1, 2, ... \tag{2}$$

6. Haga un progama, archivo `.sce`, que produzca una tabla con cinco columnas:

i) n, de 0 a 20.

ii) x_n según (2).

iii) x_n según (1) trabajando con todas las cifras que Scilab usa normalmente.

iv) x_n según (1) trabajando con 10 cifras significativas.

v) x_n según (1) trabajando con 5 cifras significativas.

Los números decimales deben aparecer con 15 cifras decimales (puede usar formato `%20.15f`.

7. Comente los resultados de la tabla, concluya.

0.3 Método de Gauss sin y con pivoteo

El objetivo es escribir una función que implemente el método de Gauss sin pivoteo, una función que implemente el método de Gauss con pivoteo parcial y comparar los resultados. En ambos casos, en todas las operaciones se redondea con un número fijo de cifras. Puede ser con un número fijo de cifras significativas o con un número fijo de cifras decimales.

1. Escriba funciones, `redond(x, n)`, `sumaRed(x, y, n)`, `multRed(x, y, n)`, `divRed(x, y, n)`, `restaRed(x, y, n)`, `redMatr(a, n)`, para redondear un número con n cifras, sumar (redondea los números, suma y redondea el resultado), multiplicar, dividir, restar y redondear una matriz.

2. Escriba funciones

   ```
   [x, info]    = solTriSupR(a, b, nc)
   [a, b, info] = triangR(a0, b0, nc)
   [x, info]    = GaussR(a, b, nc)
   ```

   que, respectivamente, resuelve un sistema triangular superior, triangulariza un sistema de ecuaciones y resuelve un sistema de ecuaciones por el método de Gauss. En las tres funciones las operaciones se hacen utilizando redondeo con `nc` cifras. Siempre, por convención, el parámetro de salida o resultado `info` valdrá 1 si y solamente si el proceso se desarrolló satisfactoriamente. No olvide incluir en la primera función (en el sentido de su utilización en el tiempo) el redondeo de los datos.

   Puede ensayar la función con `solTriSupR` con ejemplos construidos "a mano" o con datos aleatorios. Por ejemplo, puede usar órdenes semejantes a

   ```
   n = 4;
   a  = (rand(n,n) - 0.5)*20; a = triu(a);
   x0 = (rand(n,1) - 0.5)*20; b = a*x0;
   xTS = solTriSup0(a, b); errTS = norm(xTS-x0)
   ```

   El valor de `errTS` debe ser cero o casi cero. Para GaussR puede utilizar órdenes semejantes a las anteriores, excluyendo `a = triu(a)`.

   Para escribir las 3 funciones, pueden ser útiles pero no indispensables, los siguientes pasos:

   - escribir funciones `x = solTriSup0(a, b)`, `[a, b] = triang0(a0, b0)`, `x = Gauss0(a, b)` que no controlan la existencia de pivotes nulos ni la posiblidad de que la matriz no sea invertible.
   - escribir funciones `[x, info] = solTriSup(a, b)`, `[a, b, info] = triang(a0, b0)`, `[x, info] = Gauss(a, b)` que controlan la existencia de pivotes nulos, que hacen intercambio de filas cuando es necesario y que consideran la posibilidad de que la matriz sea no invertible. En estas funciones se pueden utilizar las operaciones matriciales de Scilab.
   - escribir funciones `[x, info] = solTriSup1(a, b)`, `[a, b, info] = triang1(a0, b0)`, `[x, info] = Gauss1(a, b)`, semejantes a las 3 anteriores pero que no utilizan las operaciones matriciales de Scilab. Por ejemplo, la orden `a(i,i+1:n)*x(i+1:n)` se puede remplazar por

     ```
     t = 0
     for j = i+1:n
       t = t + a(i,j)*x(j)
     end
     ```

3. Escriba funciones

   ```
   [a, b, info] = triangPP_R(a0, b0, nc)
   [x, info]    = GaussPP_R(a, b, nc)
   ```

   que, respectivamente, triangulariza un sistema de ecuaciones con pivoteo parcial y resuelve un sistema

de ecuaciones por el método de Gauss con pivoteo parcial. En estas funciones las operaciones se hacen utilizando redondeo con `nc` cifras.

Puede ser útil escribir funciones previas análogas a las indicadas para las funciones `triangR`, etc.

4. Construya un ejemplo 2×2 o 3×3 o 4×4, donde:

   - no haya diferencia (o muy pequeña) al utilizar `GaussR` y `GaussPP_R` con 15 cifras significativas.
   - haya diferencia clara al utilizar `GaussR` y `GaussPP_R` con un número de cifras entre 4 y 12.

0.4 Método de Cholesky

1. Con un valor grande y adecuado de n (este valor de n será común para este paso y los siguientes) construya aleatoriamente dos matrices cuadadradas. Por medio de las funciones `tic()` y `toc()` obtenga una estimación del tiempo requerido para efectuar el producto.

   Es posible que obtenga un aviso semejante a

   ```
                !--error 17
   rand: stack size exceeded (Use stacksize function to increase it)
   ```

   Entonces mire la ayuda de `stacksize` y aumente el tamaño de la pila por una orden semejante a `stacksize(8000000)` o `stacksize(16000000)` o ...

2. ¿Cuál es el número de operaciones de punto flotante del producto? ¿Qué pasa con el tiempo cuando se duplica el valor de n? Explique.

3. Construya aleatoriamente un sistema de ecuaciones. Obtenga una estimación del tiempo requerido para efectuar la solución del sistema por medio de la orden `a\b`.

4. Construya aleatoriamente una matriz simétrica definida positiva. Puede ser por ordenes semejantes a

   ```
   a = rand(n,n); a = a*a'
   ```

   o, preferiblemente,

   ```
   a = rand(n,n); a = a+a' + 2*n*eye(n,n)
   ```

   ¿Porqué?

5. Obtenga una estimación del tiempo requerido para efectuar la factorización de Cholesky por medio de la función de Scilab `chol`.

6. Una de las maneras de obtener la factorización de Cholesky $U^{\mathrm{T}}U = A$ es mediante las fórmulas

$$
\begin{aligned}
t &= a_{kk} - \sum_{i=1}^{k-1} u_{ik}^2 \\
&\textbf{si } \ t \leq \varepsilon, \text{ entonces } A \text{ no es definida positiva} \\
u_{kk} &= \sqrt{t} \\
u_{kj} &= \frac{a_{kj} - \sum_{i=1}^{k-1} u_{ik}u_{ij}}{u_{kk}}, \ j = k+1, ..., n
\end{aligned}
$$

   Escriba una función `u = chol0(a)` que efectúe la factorización de Cholesky. Es conveniente usar las operaciones matriciales de Scilab. Por ejemplo, el cálculo de

$$\sum_{i=1}^{k-1} u_{ik}^2$$

   se puede hacer por `u(1:k-1,k)'*u(1:k-1,k)` o también por
   `norm( u(1:k-1,k) )^2`

7. Obtenga una estimación del tiempo requerido por la función `chol0`.

8. Para resolver una sistema de ecuaciones $Ax = b$, con matriz definida positiva, es preferible, en lugar del método de Gauss, utilizar el método de Cholesky, es decir

   - Obtener U tal que $U^{\mathrm{T}}U = A$.
   - Resolver $U^{\mathrm{T}}y = b$.
   - Resolver $Ux = y$.

   ¿Cual es, aproximadamente, el número de operaciones de punto flotante requeridas para el proceso anterior?

9. Obtenga una estimación del tiempo requerido para la solución de un sistema con matriz definida positiva mediante las tres órdenes

```
u = chol(a)
y = (u')\b
x = u\y
```

10. Compare el tiempo anterior con el tiempo de `x = a\b`. Concluya.

11. Escriba una función `x = solTriSup(a, b)` que resuelve un sistema de ecuaciones con matriz triangular superior. Suponga provisionalmente que los tamaños son compatibles, que la matriz sí es triangular superior y que no hay elementos diagonales nulos.

12. Escriba una función `x = solTriInf(A, b)` que resuelve un sistema de ecuaciones con matriz triangular inferior.

13. Escriba una función `x = sol_Ut_b(U, b)` que resuelve el sistema triangular inferior $U^{\mathrm{T}}x = b$ donde U^{T} es triangular inferior, es decir, U es triangular superior. Esta función es muy parecida a la anterior, pero la información sobre la matriz está en diferente sitio.

14. Escriba la función `x = solDP(a, b)` que resuelve un sistema de ecuaciones con matriz definida positiva mediante las órdenes

```
u = chol(a)
y = sol_Ut_b(U, b)
x = solTris(u, y)
```

15. Obtenga una estimación del tiempo de la función anterior. Compare con el tiempo de `x = a\b`. Concluya.

0.5 Método de Gauss-Seidel

Se desea resolver un sistema de ecuaciones lineales $Ax = b$. El algortimo para el método de GS se puede escribir así:

datos: $A,\ b,\ x^0,\ \varepsilon,$ maxit
$x = x^0$
para $k = 1, ...,$maxit
    nrmD$\leftarrow 0$
    para $i = 1, ..., n$
        $\delta_i = (b_i - A_{i.}\, x)/a_{ii}$
        $x_i \leftarrow x_i + \delta_i$
        nrmd$\leftarrow$nrmD$+|\delta_i|$
    fin-para i
    si nrmD $\leq \varepsilon$ **ent** $x^* \approx x$, **salir**
fin-para k

1. Escriba una función `[x, k, info] = GS(a, b, x0, eps, maxit)` que implementa el método de GS. Los resultados serán: `x` aproximación de la solución (posiblemente), `k` el número de iteraciones completas realizadas e `info` que valdrá 1 si se obtuvo una buena aproximación de la solución.

2. Considere una matriz A de la forma

$$A = \begin{bmatrix} d_1 & 0 & v_1 & 0 & 0 & 0 & \cdots & & & \\ 0 & d_2 & 0 & v_2 & 0 & 0 & & & & \\ u_1 & 0 & d_3 & 0 & v_3 & 0 & & & & \\ 0 & u_2 & 0 & d_4 & 0 & v_4 & & & & \\ & & & & & & & & & \\ & & & & & & & & & \\ & & & & & & & d_{n-2} & 0 & v_{n-2} \\ & & & & & & & 0 & d_{n-1} & 0 \\ & & & & & & & u_{n-2} & 0 & d_n \end{bmatrix}$$

   En palabras, los elementos diagonales de A son d_1, d_2, ..., d_n; en la segunda superdiagonal están los valores v_1, v_2, ..., v_{n-2}. en la segunda subdiagonal están los valores u_1, u_2, ..., u_{n-2}. Los otros elementos (entradas) de A son nulos.

   Escriba una funcion `A = matriz3(d, u, v)` que, dados los tres arreglos `d`, `u` y `v`, construye explícitamente la matriz A.

3. Escriba una función `[x, k, info] = GS3(d, u, v, b, x0, eps, maxit)` que implementa el método de GS para resolver $Ax = b$, donde A es una matriz con la forma descrita anteriormente. Esta función **no** construye explícitamente la matriz A. Los resultados parciales (puede agregar temporalmente `disp` o `printf`) y finales deben ser los mismos por cualquiera de los dos formas siguientes:

   - utilizar `GS3` .
   - con `matriz3` construir explícitamente A y utilizar `GS` .

   Es necesario adaptar el algortimo para tener en cuenta las especificidades de cada fila de A:

para $i = 1, ..., n$
  si $i \leq 2$
    $\delta_i \leftarrow \cdots$
  sino
    si $i \leq n-2$
      $\delta_i \leftarrow \cdots$
    sino
      $\delta_i \leftarrow \cdots$
    fin-si
  fin-si
  $x_i \leftarrow \cdots$
  $\vdots$
fin-para i

4. Utilice `GS3` con

```
d = [ 10 11 12 13 14]'
u = [ 1 2 3]'
v = [ -3 -2 -1]'
b = [11 -16 34 -43 65]'
```

   ¿Cual es la solución?

5. Calcule, en función de n, el número de operaciones de punto flotante en cada iteración.

6. Escoja un valor grande de n y utilice `GS3` con

```
d = 100*ones(n,1);
u = -2*ones(n-2,1);
v =  3*ones(n-2,1);
b = 101*ones(n,1);
b(1:2) = 103;
b(n-1:n) = 98;
eps = 1.0e-6
```

   Construya explícitamente A. Evalúe el tiempo de `A\b` y de `GS3`. Justifique. ¿Qué pasa cuando n es más grande.

7. Dé una razón (teórica) que justifica la convergencia del método GS para los datos anteriores.

8. ¿Qué pasa con el número de iteraciones y con el tiempo si `eps = 1.0e-12` ?

9. ¿Cuál es el mayor valor de n para poder resolver $Ax = b$, construyendo explícitamente A y usando `A\b`?

   ¿Cuál es el mayor valor de n para poder resolver $Ax = b$, con la información de d, u y v, usando `GS3`?

0.6 Método de Muller

Este método sirve para hallar raíces reales o complejas de polinomios reales $p(x) = a_0+a_1x+a_2x^2+...+a_nx^n$. El polinomio p se puede expresar en función de sus raíces:

$$p(x) = a_n(x - r_1)(x - r_2)\cdots(x - r_n).$$

Las raíces complejas, no reales, siempre vienen por parejas, es decir si $r = a+ib$, $b \neq 0$, es una raíz, entonces $\bar{r} = a - ib$, el conjugado de r, también es raíz. Para las raíces complejas $q(x) = (x - r)(x - \bar{r})$ divide a $p(x)$.

$$q(x) = (x - r)(x - \bar{r}) = (x - a - ib)(x - a + ib) = (x - a)^2 + b^2 = x^2 - 2ax + (a^2 + b^2).$$

Si se halla una raíz real r entonces $q(x) = (x - r)$ divide a $p(x)$.

En general, si $q(x)$ divide a $p(x)$, entonces existe un polinomio $s(x)$ tal que

$$\begin{aligned} p(x) &= q(x)s(x), \\ \text{grado}(p) &= \text{grado}(q) + \text{grado}(s). \end{aligned}$$

Entonces para sequir obteniendo las raíces de $p(x)$ basta con obtener las raíces de $s(x)$, polinomio más sencillo.

En el método de la secante, dados dos valores x_0 y x_1 se busca la recta que pasa por los puntos $(x_0, f(x_0))$, $(x_1, f(x_1))$; el siguiente valor x_2 está dado por el punto donde la recta corta el eje x. En el método de Muller, en lugar de una recta, se utiliza una parábola. Dados tres valores x_0, x_1 y x_2, se construye la parábola $P(x)$ que pasa por los puntos $(x_0, f(x_0))$, $(x_1, f(x_1))$ y $(x_2, f(x_2))$; el siguiente valor x_3 está dado por el (un) punto tal que $P(x_3) = 0$.

La parábola se puede escribir de la forma $P(x) = a(x - x_2)^2 + b(x - x_2) + c$. Las fórmulas que permiten calcular a, b y c son:

$$\begin{aligned} d &= (x_0 - x_1)(x_0 - x_2)(x_1 - x_2), \\ a &= \frac{-(x_0 - x_2)(f(x_1) - f(x_2)) + (x_1 - x_2)(f(x_0) - f(x_2)}{d}, \\ b &= \frac{(x_0 - x_2)^2(f(x_1) - f(x_2)) - (x_1 - x_2)^2(f(x_0) - f(x_2)}{d}, \\ c &= f(x_2). \end{aligned} \tag{3}$$

Entonces

$$x_3 - x_2 = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Para reducir los errores de redondeo se "racionaliza" el numerador y buscando que el denominador resultante sea grande (en valor absoluto)

$$\begin{aligned} D &= b^2 - 4ac \\ R &= \sqrt{D} \\ \delta &= \begin{cases} b + R & \text{si} \quad |b + R| \geq |b - R| \\ b - R & \text{si} \quad |b + R| < |b - R| \end{cases} \\ x_3 &= x_2 - \frac{2c}{\delta} \ . \end{aligned} \tag{4}$$

Si en una iteración $D = b^2 - 4ac < 0$ es necesario utilizar, a partir de ahí, aritmética compleja (Scilab lo hace automáticamente). Eso hace que los siguientes valores a, b y c no sean necesariamente reales. Muy posiblemente $b^2 - 4ac$ tampoco es real.

MÉTODO DE MULLER PARA UNA RAÍZ

datos: p, x_0, x_1, x_2, ε_f, ε_0, maxit
$f_0 = p(x_0)$, $f_1 = p(x_1)$, $f_2 = p(x_2)$
`info`$= 0$
para $k = 1, ..., \text{maxit}$
  si $|f_2| \leq \varepsilon_f$ **ent** $r = x_2$, `info`$= 1$, **parar**
  $d = (x_0 - x_1)(x_0 - x_2)(x_1 - x_2)$
  si $|d| \leq \varepsilon_0$ **ent parar**
  calcular a, b y c según (3)
  $D = b^2 - 4ac$
  $R = \sqrt{D}$
  $\delta_1 = b + R, \quad \delta_2 = b - R$
  si $|\delta_1| \geq |\delta_2|$ **ent** $\delta = \delta_1$, **sino** $\delta = \delta_2$
  si $|\delta| \leq \varepsilon_0$ **ent parar**
  $x_3 = x_2 - 2c/\delta$
  $x_0 = x_1$, $x_1 = x_2$, $x_2 = x_3$, $f_0 = f_1$, $f_1 = f_2$
  $f_2 = p(x_2)$
fin-para k

Si el algoritmo anterior acaba normalmente, `info` valdrá 1 y r será una raíz, real o compleja. En la anterior descripción del algoritmo, $|t|$ indica valor absoluto si t es real, e indica módulo si t es complejo.

1. Averigüe cómo efectuar división de polinomios en Scilab.

2. Escriba una función `[r, info] = Muller1(coef, x0, x1, x2, epsf, eps0, maxit)` que obtiene (si al final `info` vale 1) una raíz de un polinomio cuyos coeficientes reales están, en orden creciente de la potencia, en el vector `coef`. En el algoritmo presentado, esta función va desde tomar o redefinir x_0, x_1, x_2 hasta **sino** $q(x) = (x - r)(x - \bar{r})$.

3. Escriba una función `[rr, info] = Muller(coef, x0, epsf, eps0, maxit)` que obtiene (si al final `info` vale 1) todas las raíces de un polinomio cuyos coeficientes reales están, en orden creciente de la potencia, en el vector `coef`. Las raíces quedarán en el vector `rr`. Esta función utiliza varias veces `Muller1`.

MÉTODO DE MULLER

datos: p, x_0, ε_f, ε_0, maxit
$r = x_0, \quad h = 0.5$
mientras grado$(p) \geq 3$
  $x_0 = r, \;\; x_1 = x_0 + h, \;\; x_2 = x_1 + h$
  $(r, \; info) = Muller1(p, x_o, x_1, x_2, \varepsilon_f, \varepsilon_0, maxit)$
  si $info = 0$, **ent parar**
  si $|imag(r)| \leq \varepsilon_0$ **ent** $q(x) = (x - r)$
  sino $q(x) = (x - r)(x - \bar{r})$
  $p(x) = p(x)/q(x)$
fin-mientras
calcular raíces de p (de grado no superior a 2)

4. Sean k_1, k_2, ..., k_7 los primeros siete dígitos del número de su documento de identidad. Halle las seis raíces del polinomio cuyos coeficientes, en orden creciente de la potencia, son k_7, k_6, ..., k_1. Muestre los resultados intermedios para la primera utilización de `Muller1`.

0.7 Interpolación polinomial

Dados n puntos (x_1, y_1), (x_2, y_2), ..., (x_n, y_n), con $x_i \neq x_j$ si $i \neq j$, entonces existe un único polinomio p de grado menor o igual a $n-1$ tal que

$$p(x_i) = y_i, \quad i = 1, ..., n.$$

El polinomio $p(x) = a_0 + a_1 x + a_2 x^2 + \cdots a_{n-1} x^{n-1}$ se puede obtener de dos maneras: resolviendo un sistema de ecuaciones y utilizando polinomios de Lagrange.

1. El sistema de ecuaciones $n \times n$ que hay que resolver es

$$\begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & x_2^3 & \cdots & x_2^{n-1} \\ 1 & x_3 & x_3^2 & x_3^3 & \cdots & x_3^{n-1} \\ \vdots & & & \ddots & & \\ 1 & x_n & x_n^2 & x_n^3 & \cdots & x_n^{n-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix}$$

   Escriba una función `p = polInterpSE(x, y)` que obtiene el polinomio de interpolación resolviendo el sistema de ecuaciones. Las coordenadas de los puntos están en los vectores columna `x` y `y`. Como `x` es un vector columna, entonces las columnas de la matriz de coeficientes del sistema se pueden construir en Scilab por órdenes análogas a

```
x.^k
```

2. Hay n polinomios de Lagrange, $L_1(x)$, $L_2(x)$, ..., $L_n(x)$,

$$L_k(x) = \frac{\prod_{i=1, i\neq k}^{n} (x - x_i)}{\prod_{i=1, i\neq k}^{n} (x_k - x_i)}$$

   Estos polinomios se pueden construir de manera iterativa

$L_k(x) \leftarrow 1$
$d \leftarrow 1$
para $i = 1, ..., n$
  si $i \neq k$
    $d \leftarrow d\,(x_k - x_i)$
    $L_k(x) \leftarrow L_k(x)\,(x - x_i)$
  fin-si
fin-para
$L_k(x) \leftarrow L_k(x)/d$

   Escriba una función `Lk = poliLagr(x, k)` que obtiene el polinomio de Lagrange $L_k(x)$. Los valores x_i están en el arreglo `x`. La siguiente tabla muestra algunos ejemplos de construcción de polinomios en Scilab.

polinomio	Scilab
0	`poly([0], 'x', 'c')`
1	`poly([1], 'x', 'c')`
$x - b$	`poly([-b  1], 'x', 'c')`
$x - b$	`poly([b], 'x', 'r')`

3. La fórmula para obtener p utilizando polinomios de Lagrange es

$$p(x) = \sum_{k=1}^{n} y_k L_k(x).$$

En seudolenguaje,

$p(x) \leftarrow 0$
para $k = 1, ..., n$
construir $L_k(x)$
$p(x) \leftarrow p(x) + y_k\, L_k(x)$
fin-para

Escriba una función `p = polInterpPL(x, y)` que obtiene el polinomio de interpolación utilizando polinomios de Lagrange. Las coordenadas de los puntos están en los vectores columna `x` y `y`.

4. Escriba una función que calcula el factorial de un entero no negativo.

5. Sean `x`, `y` dos vectores columna con las coordenadas de n puntos: $(0, (-1)^{n-1}(n-1)!)$, $(1, 0)$, $(2, 0)$, $(3, 0)$, ..., $(n-1, 0)$. Considere $n = 5$ y construya explícitamente los vectores columna `x`, `y`. Obtenga `p`, el polinomio de interpolación, usando `polInterpSE`. Construya un vector columna `yse` con los valores del polinomio `p` evaluado en los valores del vector columna `x`. Teóricamente los dos vectores, `y` y `yse` son iguales. Obtenga la norma de `y-yse`.

6. Obtenga `p`, el polinomio de interpolación, usando `polInterpPL`. Construya un vector columna `ypl` con los valores del polinomio evaluado en los valores del vector columna `x`. Obtenga la norma de `y-ypl`.

7. Repita los dos pasos anteriores para $n = 3, 4, ..., 20$. Construya una tabla de tres columnas, en la primera los valores de n, en la segunda columna los valores `norm(y-yse)` y en la tercera columna los valores `norm(y-ypl)`. Concluya.

0.8 Error local y global del método del trapecio y de Simpson

1. Escriba una función `[a, b] = rectaMinC(X, Y)` que obtiene los dos coeficientes de la recta de aproximación por mínimos cuadrados $y = ax + b$ para los puntos cuyas coordenadas están en los dos vectores columna X y Y.

```
m = size(X,1)
A = [ones(m,1) X]
aa = (A'*A)\(A'*Y)
b = aa(1)
a = aa(2)
```

2. Escriba una función `s = trapecio(f, a, b, n)` que obtiene una aproximación de la integral definida por medio de la fórmula del trapecio

$$\int_a^b f(x)\,dx \approx h\left(\frac{f(a)+f(b)}{2} + \sum_{i=1}^{n-1} f(x_i)\right)$$
$$h = \frac{b-a}{n}$$
$$x_i = a + ih$$

Las órdenes importantes son semejantes a:

```
h = (b-a)/n
s = ( f(a) + f(b) )/2
for i=1:n-1
  s = s + f( a+i*h)
end
s = s*h
```

3. Escriba una función `s = Simpson(f, a, b, n)` que obtiene una aproximación de la integral definida por medio de la fórmula de Simpson (con n par)

$$\int_a^b f(x)\,dx \approx \frac{h}{3}\left(f(a)+f(b)+4\sum_{i=1,3,\ldots}^{n-1} f(x_i) + 2\sum_{i=2,4,\ldots}^{n-2} f(x_i)\right)$$
$$h = \frac{b-a}{n}$$
$$x_i = a + ih$$

4. Escoja una función (diferente de un polinomio de grado inferior a 7) de la cual conoce la (una) antiderivada, es decir, para la cual se puede obtener fácilmente el valor exacto de una integral definida. Por ejemplo $f(x) = e^x$. Con $a = 0$, $h = 0.2$, $b = a + h$, $n = 1$, obtenga el valor aproximado por la fórmula del trapecio, el valor exacto de $\int_a^b f(x)\,dx$ y e el valor absoluto del error cometido. Estos valores de h y e deben ser almacenados en arreglos H y E.

5. Repita el paso anterior para $h = 0.21,\ 0.22, ...,\ 0.25$ (de nuevo $b = a + h$, $n = 1$).

6. Se puede suponer que el error local es aproximadamente

$$e_{\text{loc}} \approx ch^k$$

entonces

$$\log(e_{\text{loc}}) \approx \log(c) + k\log(h)\,.$$

Es decir, se tendría la ecuación de una recta. Graficar $\log(E)$ contra $\log(H)$. ¿Corresponde aproximadamente a una recta?

7. Por medio de la orden `[k, logc] = rectaMinC(log(H), log(E))` obtenga el valor de k. ¿Corresponde aproximadamente a lo previsto teóricamente para el error local del método del trapecio?

8. Ahora se va a "mirar" el orden del error global. Sea $a = 0$, $b = 0.6$, $n = 2, 3, 4, 5, 6$, $h = (b-a)/n$. Para esos valores de h, obtenga por la fórmula del trapecio, el valor exacto de $\int_a^b f(x)\,dx$ y e el valor absoluto del error cometido. Estos valores de h y e deben ser almacenados en arreglos H y E.

9. Se puede suponer que el error global es aproximadamente

$$e_{\text{glob}} \approx dh^m$$

entonces

$$\log(e_{\text{glob}}) \approx \log(d) + m\log(h)\,.$$

Es decir, se tendría la ecuación de una recta. Graficar $\log(E)$ contra $\log(H)$. ¿Corresponde aproximadamente a una recta?

10. Por medio de la orden `[m, logd] = rectaMinC(log(H), log(E))` obtenga el valor de m. ¿Corresponde aproximadamente a lo previsto teóricamente para el error global para el método del trapecio?

11. Se va a repetir el proceso anterior para la fórmula de Simpson. Para el error local de la fórmula de Simpson, efectuar pasos análogos a 4, 5, 6 y 7, con $a = 0$, $h = 0.2,\ 0.21,\ 0.22,\ 0.23,\ 0.24,\ 0.25$, $b = a + h$, $n = 2$.

12. Para el error global de la fórmula de Simpson, efectuar pasos análogos a 8, 9 y 10, con $a = 0$, $b = 0.6$, $n = 2, 4, 6, 8, 10,\ \ h = (b-a)/n$.

0.9 Comparación de métodos de Simpson y Gauss-Legendre

Se desea comparar para una función diferente de un polinomio, con antiderivada conocida (se puede obtener el valor exacto de una integral definida), el método de Simpson y el método de cuadratura de Gauss-Legendre, utilizando el mismo número de evaluaciones de la función.

1. Escoja una función, diferente de un polinomio, de la cual conoce la (una) antiderivada. Por ejemplo $f(x) = e^x$. Escoja un intervalo "razonable", por ejemplo, $a = 1$, $b = 2$. Escriba una función `y = f08(x)` donde se calcula la función f.

2. Escoja dos valores $a < b$ y obtenga el valor exacto de $\displaystyle\int_a^b f(x)\,dx$

3. Obtenga una aproximación de $\displaystyle\int_a^b f(x)\,dx$ utilizando `intg`.

4. Obtenga una aproximación de $\displaystyle\int_a^b f(x)\,dx$ utilizando la fórmula de Simpson con 6 subintervalos (hay 7 evaluaciones de f).

5. Escriba una función `s = Simpson38(f, a, b, n)` que obtiene una aproximación de la integral definida por medio de la fórmula de Simpson 3/8 con n múltiplo de 3.

$$\int_{x_0}^{x_3} f(x)\,dx \approx \frac{3}{8}h(y_0 + 3y_1 + 3y_2 + y_3).$$

Para el intervalo $[a, b]$, con n (múltiplo de 3) subintervalos

$$\int_a^b f(x)\,dx \approx \frac{3}{8}h(y_0 + 3y_1 + 3y_2 + 2y_3 + 3y_4 + 3y_5 + 2y_6 + 3y_7 + \cdots + 3y_{n-2} + 3y_{n-1} + y_n).$$

Los pasos principales pueden ser:

$$\begin{aligned}
&h = (b-a)/n\\
&s \leftarrow f(a) + f(b)\\
&s \leftarrow s + 3\sum_{i=1,4,\ldots}^{n-2} f(x_i)\\
&s \leftarrow s + 3\sum_{i=2,5,\ldots}^{n-1} f(x_i)\\
&s \leftarrow s + 2\sum_{i=3,6,\ldots}^{n-3} f(x_i)\\
&s \leftarrow 3hs/8
\end{aligned}$$

donde $x_i = a + ih$. O también,

$h = (b-a)/n$
$s \leftarrow f(a) + f(b)$
para $i = 1, ..., n-1$
    $y \leftarrow f(a + ih)$
    si $3|i$ **ent** $s \leftarrow s + 2y$
    sino $s \leftarrow s + 3y$
fin-para
$s \leftarrow 3hs/8$

La notación $p|q$ significa "p divide a q". En Scilab se puede escribir `if modulo(q, p) == 0`

6. Obtenga una aproximación de $\int_a^b f(x)\,dx$ mediante

   `h = (b-a)/5,`

   `Simpson(f08, a, a+2*h, 2) + Simpson38(f08, a+2*h, b, 3)`

   (se utilizan 6 evaluaciones de f).

7. En el método de cuadratura de Gauss-Legendre de orden m el cálculo numérico de la integral se hace mediante la siguiente aproximación:

$$\int_{-1}^{1} f(x)\,dx \approx w_{m1}f(x_{m1}) + w_{m2}f(x_{m2}) + \cdots w_{mm}f(x_{mm}).$$

   Los valores w_{mj}, los pesos o ponderaciones, y los valores x_{mj}, las abcisas, son fijos y conocidos.

Cuadratura de Gauss-Legendre

m	$\pm x_{mj}$	w_{mj}
1	0.000000000000000	2.000000000000000
2	0.577350269189626	1.000000000000000
3	0.000000000000000	0.888888888888889
	0.774596669241483	0.555555555555556
4	0.339981043584856	0.652145154862546
	0.861136311594053	0.347854845137454
5	0.000000000000000	0.568888888888889
	0.538469310105683	0.478628670499366
	0.906179845938664	0.236926885056189
6	0.238619186083197	0.467913934572691
	0.661209386466265	0.360761573048139
	0.932469514203152	0.171324492379170

En Abramowitz y Stegun hay una tabla muy completa. Para integrar en otro intervalo es necesario hacer un cambio de variable,

$$\int_a^b f(x)\,dx \approx \frac{b-a}{2}\sum_{j=1}^{m} w_{mj}\; f\left(\frac{b-a}{2}x_{mj} + \frac{a+b}{2}\right).$$

Escriba una función `s = GaussLeg1(f, a, b, m)` que calcula de manera aproximada $\int_a^b f(x)\,dx$ por el método de cuadratura de Gauss-Legendre de orden m. En esta función se definen y se utilizan dos matrices `X` y `W`. La matriz `X` es triangular inferior de tamaño 6×6: en las primeras m posiciones de la fila m están las m abcisas y, de manera análoga, la matriz `W` tiene los pesos correspondientes. La parte principal de esta función puede ser semejante a

```
X = [ 0 0 0 0 0 0;
       //
      -0.577350269189626 0.577350269189626 0 0 0 0
      //
       0 -0.774596669241483 0.774596669241483 0 0 0
       //
      -0.339981043584856 0.339981043584856 ..
      -0.861136311594053 0.861136311594053  0 0
      //
       0 -0.538469310105683 0.538469310105683 ..
```

```
      -0.906179845938664 0.906179845938664 0
      //
      -0.238619186083197 0.238619186083197 ..
      -0.661209386466265 0.661209386466265 ..
      -0.932469514203152 0.932469514203152
     ]
W = [2 0 0 0 0 0
    1 1 0 0 0 0
    0.88888888888889 0.555555555555556 0.555555555555556 0 0 0
    //
    0.652145154862546 0.652145154862546 ..
    0.347854845137454 0.347854845137454 0 0
    //
    0.568888888888889 0.478628670499366 0.478628670499366 ..
    0.236926885056189 0.236926885056189 0
    //
    0.467913934572691 0.467913934572691 ..
    0.360761573048139 0.360761573048139 ..
    0.171324492379170 0.171324492379170
    ]

  u = (b-a)/2
  v = (a+b)/2
  s = 0
  for j=1:m
    tj = u*X(m,j) + v
    s = s + W(m,j)*f(tj)
  end
  s = u*s
```

8. Si el intervalo $[a, b]$ es muy grande, éste se puede dividir en n subintervalos y en cada uno de ellos utilizar cuadratura de Gauss-Legendre de orden m. Escriba una función `s = GaussLeg(f, a, b, m, n)` que calcula de manera aproximada $\int_a^b f(x)\,dx$ por el método de cuadratura de Gauss-Legendre de orden m en cada uno de los n subintervalos. La parte principal de esta función puede ser semejante a

```
s = 0
h = (b-a)/n
for i=1:n
  s = s + GaussLeg1(f, a+(i-1)*h, a+i*h, m)
end
```

9. Compare ahora los errores obtenidos, utilizando 6 evaluaciones de la función, en los sguientes casos
   a) `h = (b-a)/5, Simpson(f08, a, a+2*h, 2) + Simpson38(f08, a+2*h, b, 3)`
   b) promedio del error con Simpson $n = 4$ y del error con $n = 6$.
   c) GL, $m = 1$, $n = 6$
   d) GL, $m = 2$, $n = 3$
   e) GL, $m = 3$, $n = 2$
   f) GL, $m = 6$, $n = 1$

   Cambie de intervalo. Cambie de función. Observe y concluya.

0.10 Método de Newton en $\mathbb{R}^n$ y cálculo numérico de la matriz jacobiana

1. Considere una función $f : \mathbb{R}^n \to \mathbb{R}^n$, por ejemplo,

$$f(x_1, x_2) = (\, 2x_1^2 + 3x_1x_2 + 4x_2 - 38,\ -5x_1x_2 + 6x_1 - 6x_2 + 36\,)$$

   Escriba una función `fx = f09(x)` que, para un vector columna `x`, construye el vector columna `fx` con las dos componentes de $f(x)$.

2. Encuentre, usando `fsolve`, un vector x tal que $f(x) = 0$ para la función definida en `f09`.

3. Dada una función $f : \mathbb{R}^n \to \mathbb{R}^n$, se denotará por $f'(x)$ la matriz jacobiana de f evaluada en x. En esta matriz $n \times n$, la fila i está formada por las n derivadas parciales de f_i (la i-ésima componente de f).

   Escriba una función `J = jac09(x)` que para un vector columna `x`, construye la matriz jacobiana `J` de la función definida en `f09`.

4. En el método de Newton en $\mathbb{R}^n$ (para hallar un x tal que $f(x) = 0$), en cada iteración hay dos pasos fundamentales

$$\begin{aligned} \text{resolver} \quad f'(x^k)\, d^k &= -f(x^k) \\ x^{k+1} &= x^k + d^k\,. \end{aligned}$$

   Estos pasos presuponen que para caulquier x^k se puede obtener $f(x^k)$ y $f'(x^k)$. El método termina satisfactoriamente si

$$||f(x^k)|| \le \varepsilon\,.$$

   Las terminaciones no deseadas, pero posibles, son

   - demasiadas iteraciones sin encontrar un x adecuado.
   - $f'(x^k)$ no es invertible (su determinante es nulo o casi nulo).

   Escriba una función `[x, info, k] = Newton1(f, jac, x0, eps, maxit)` que implementa el método de Newton. El resultado `info` valdrá 1 si se obtuvo un x adecuado, 2 si en `maxit` iteraciones no se pudo obtener una solución adecuada y valdrá 0 si en alguna iteración la matriz jacobiana es singular o casi singular. En la función `f` se calcula $f(x)$, en la función `jac` se calcula $f'(x)$, el vector columna `x0` es la aproximación inicial. El número de iteraciones realizadas estará en `k`.

5. Verifique la función anterior mediante `[x, info, k] = Newton1(f09, jac09, [1; 1], 1.0e-4, 20)` y con otros ejemplos.

6. Averigüe mediante qué función de Scilab puede obtener numéricamente la matriz jacobiana de una función.

   Escriba una función `[x, info, k] = Newton2(f, x0, eps, maxit)` que implementa el método de Newton. Aquí no es necesario tener una función externa que evalúa la matriz jacobiana, basta con tener f.

7. Para una función $\varphi : \mathbb{R} \to \mathbb{R}$ la derivada se puede aproximar por

$$\varphi'(x) \approx \frac{\varphi(x+h) - \varphi(x)}{h} \quad \text{y también} \quad \varphi'(x) \approx \frac{\varphi(x+h) - \varphi(x-h)}{2h}$$

La derivada parcial se puede aproximar por

$$\frac{\partial f_i}{\partial x_j}(x) \approx \frac{f_i(x_1, x_2, ..., x_j + h, ..., x_{n-1}, x_n) - f_i(x_1, ..., x_n)}{h}$$

con h cercano a 0. Una mejor aproximación es

$$\frac{\partial f_i}{\partial x_j}(x) \approx \frac{f_i(x_1, x_2, ..., x_j + h, ..., x_{n-1}, x_n) - f_i(x_1, x_2, ..., x_j - h, ..., x_{n-1}, x_n)}{2h}.$$

Escriba una función `Dij = der_fi_xj(f, i, j, x)` que calcula la anterior aproximación.

8. Escriba una función `J = jacobiana3(f, x)` que calcula aproximadamente la matriz jacobiana.

9. Escriba una función `[x, info, k] = Newton3(f, x0, eps, maxit)` que implementa el método de Newton aproximando la matriz jacobiana mediante la función anterior.

10. Escriba una función `Dj = der_f_xj(f, j, x)` que construye un vector columna con als n derivadas parciales con respecto a x_j.

11. Escriba una función `J = jacobiana4(f, x)` que calcula aproximadamente la matriz jacobiana usando al función anterior.

12. Escriba una función `[x, info, k] = Newton4(f, x0, eps, maxit)` que implementa el método de Newton aproximando la matriz jacobiana mediante la función anterior.

13. Con los mismos parámetros de entrada, observe en número de iteraciones para cada una de las 3 implementaciones.

14. Una manera más precisa para tratar de medir la eficiencia, es mediante el número de llamados a la función f. Obtenga este número para cada una de las implementaciones (puede utilizar `global`). Concluya.

0.11 Solución de ecuaciones e integración numérica

Encontrar x tal que $\int_a^x f(t)\,dt = A$

0.12 Derivación numérica, integración numérica y solución de ecuaciones

Sea $f : \mathbb{R} \to \mathbb{R}$ una función derivable, a un real, L un real positivo. Se desea encontrar b tal que la longitud de la curva $y = f(x)$ entre los puntos $(a, f(a))$ y $(b, f(b))$ sea igual a L, es decir, se desea encontrar b solución de la ecuación

$$\psi(b) = \int_a^b \sqrt{1 + (f'(x))^2}\, dx - L = 0. \tag{5}$$

1. Escriba las funciones

   - `y = f0506(x)` que calcula $f(x) = e^x$.
   - `y = f0507(x)` que calcula $f(x) = 3 + x$.
   - `y = f0508(x)` que calcula $f(x) = 12.3456$.
   - `y = f0509(x)` que calcula $f(x) = \sqrt{9 - x^2}$.

2. Escriba la función `Df = derivada(f, x)` que calcula numéricamente una aproximación de $f'(x)$ por medio de la fórmula

   $$f'(x) \approx \frac{f(x + h) - f(x - h)}{2h}$$

   usando $h = 0.01$. Verifique que produce buenas aproximaciones con las funciones `f0506`, ... para diferentes valores de x.

3. Escriba la función `y = g(f, x)` que calcula $\sqrt{1 + (f'(x))^2}$. Para el cálculo de $f'(x)$ use `derivada`.

4. Escriba la función `longi = longiSimpson(f, a, b, n)` que calcula numéricamente una aproximación de

   $$\int_a^b \sqrt{1 + (f'(x))^2}\, dx$$

   por el método de Simpson utilizando n (par) subintervalos. La fórmula de Simpson para aproximar numéricamente $\int_a^b \varphi(x)\, dx$ con n par es

   $$\int_a^b \varphi(x)\, dx \approx \frac{h}{3}\left(\varphi(a) + \varphi(b) + 4 \sum_{i=1,3,\ldots,n-1} \varphi(x_i) + 2 \sum_{i=2,4,\ldots,n-2} \varphi(x_i)\right) \tag{6}$$

   Esta función llama varias veces la función `g`. Si en el caso (6) se calcula $\varphi(a)$, $\varphi(b)$, ..., aquí es necesario `g(f, a)`, `g(f, b)`, ...

5. Obtenga, utilizando razones analíticas o geométricas, la longitud exacta de la curva $y = f(x)$ entre los puntos $(a, f(a))$ y $(b, f(b))$, en los siguientes casos

   - $f(x) = 3 + x$, $a = 1$, $b = 2$.
   - $f(x) = 12.3456$, $a = 1$, $b = 11$.
   - $f(x) = \sqrt{9 - x^2}$, $a = -3\sqrt{2}/2$, $b = 3\sqrt{2}/2$.

   Compare con los resultados numéricos obtenidos con `longiSimpson`.

6. Utilice `longiSimpson` para $f(x) = \sqrt{9 - x^2}$ con $a = -3$ y $b = 3$, con $a = -2.99$ y $b = 2.99$ y con $a = -2.95$ y $b = 2.95$. ¿Qué resultados arroja `longiSimpson`? ¿porqué?

7. Escriba la función `[b, info] = secanteLongi(f, a, L, b0, eps, maxit, nSimpson)` que resuelve (5), la ecuación inicial $\psi(b) = 0$, por el método de la secante. Esta función llama varias veces la función `longiSimpson`. La fórmula general del método de la secante para resolver la ecuación $\varphi(x) = 0$, partiendo de x_0 y x_1 es

$$x_{k+1} = x_k - \frac{\varphi(x_k)(x_k - x_{k-1})}{\varphi(x_k) - \varphi(x_{k-1})}, \quad k = 1, 2, ...$$

   En un programa no es necesario, en cada momento, tener todos los x_k. Basta con tener x_0, x_1 y x_2.

$$x_2 = x_1 - \frac{\varphi(x_1)(x_1 - x_0)}{\varphi(x_1) - \varphi(x_0)}\,. \tag{7}$$

   Obviamente es necesario actualizarlos en cada iteración. Es decir, el x_0 de una iteración era el x_1 de la iteración anterior, $\varphi(x_0)$ corresponde al $\varphi(x_1)$ de la iteración anterior. Si para la aplicación de la fórmula (7) es necesario calcular $\varphi(x_1)$, en la función `secanteLongi` la variable es b y se requiere

   ```
   longiSimpson(f, a, b1, n) - L
   ```

   b_0, la aproximación inicial de b, es `b0`. En la función b_1 se puede construir simplemente por $b_0 + h$. El parámetro `eps` sirve para detener el proceso iterativo cuando $|\psi(b)| \leq$`eps`. El valor `maxit` indica el número máximo de iteraciones. El último parámetro es el número de subintervalos que se usará en el métodode Simpson.

8. Sean $k_1k_2k_3k_4$ los últimos 4 dígitos de su documento de identidad. Escriba la función `y = fmia(x)` que calcula la función $f(x) = (k_1 + 1)e^{(k_2+1)x}$.

9. Obtenga b para la función anterior con $a = k_3$ y $L = k_4 + 1$.

0.13 Solución de ecuaciones, derivación numérica y optimización

Sea $f : \mathbb{R} \to \mathbb{R}$ derivable. Bajo condiciones adecuadas, una manera de hallar un minimizador consiste en obtener una solución de $f'(x) = 0$. Para esto se puede utilizar el método de la secante o el método de Newton para f'. Para evaluar la derivada, esta se puede aproximar numéricamente.

1. Busque una función que tenga por lo menos una raíz real y escríbala en Scilab en una función llamada por ejemplo `y = f_39( x )`.

2. La fórmula del método de la secante para resolver la ecuación $g(x) = 0$, conocidos x_{k-1} y x_k, es

$$x_{k+1} = x_k - \frac{(x_k - x_{k-1})g(x_k)}{g(x_k) - f(x_{k-1})}$$

   Para hacer un programa, en cada iteración, solo se requieren los dos últimos x, entonces se utiliza simplemente

$$x_2 = x_1 - \frac{(x_1 - x_0)g(x_1)}{g(x_1) - f(x_0)}$$

   Obviamente, el x_0 de una iteración es el x_1 de la anterior y el x_1 de una iteración es el x_2 de la anterior.

   El proceso acaba satisfactoriamente si $|g(x_k)| \leq \varepsilon$. Las terminaciones no deseadas, pero posibles son: demasiadas iteraciones y un denominador nulo o casi nulo.

   Escriba una función `[x, info] = secante(g, x0, eps, maxit)` que realiza el método de la secante. El resultado `info` valdrá

   0: si hay un denominador nulo o casi nulo.
   1: se obtuvo un solución con la precisión deseada.
   2: muchas iteraciones sin obtener un resultado aceptable.

   Para hacer únicamente una evaluación de f por iteración el esquema de la función puede ser semejante a la siguiente porción de código:

```
x1 = x0 + 0.1
g0 = g(x0)
g1 = g(x1)
for ...

  denominador = g1 - g0
  if abs(denominador) ..

  x2 = ... /denominador
  g2 = g(x2)
  if abs(g2) <= eps

  x0 = x1
  x1 = x2
  g0 = g1
  g1 = g2
```

3. Verifique su funcionamiento por lo menos con dos funciones.

4. De un ejemplo de una función que cumpla cada una de las siguientes condiciones y escríbala en Scilab

   - Tiene un minimizador.
   - Tiene un maximixador.
   - Tiene por lo menos un minimizador y un maximizador.
   - No tiene minimizador ni maximizador.

5. Considere las siguientes aproximaciones:

$$\varphi'(x) \approx \frac{\varphi(x+h) - \varphi(x-h)}{2h}, \qquad \varphi''(x) \approx \frac{\varphi(x+h) - 2\varphi(x) + \varphi(x-h)}{h^2}.$$

   Sea $f : \mathbb{R} \to \mathbb{R}$ derivable. Si x^* es un minimizador (un punto de mínimo), entonces $f'(x^*) = 0$.

   Escriba una función `[xmin, info] = minSecante(f, x0, eps, maxit)`, adaptación de la función `secante`, para hallar un minimizador de f mediante la obtención de una raíz de $f'(x)$. La derivada de f se obtendrá numéricamente, es decir, se hará algo semejante a

```
h  = 0.01
h2 = 2*h
g0 = ( f(x0+h) - f(x0-h) )/h2
```

   El resultado `info` valdrá

   0: hay un denominador nulo o casi nulo.
   1: se obtuvo un minimizador
   2: se obtuvo un maximizador
   3: se obuvio un punto crítico
   4: muchas iteraciones sin obtener un resultado aceptable.

   ¿Dentro de `minSecante`, cómo saber si el punto obtenido tal que $f'(x) \approx 0$ corresponde a un minimizador?

6. Utilice la función `minSecante` con las funciones de la parte 4.

7. Escriba una función `[x, info] = Newton(g, der_g, x0, eps, maxit)` que implementa el método de Newton para resolver $f(x) = 0$. En la función `g` está definida g y en `der_g` está definida g'. Recuérdese que al fórmula del método de Newton para resolver $g(x) = 0$ es

$$x_{k+1} = x_k - \frac{g(x_k)}{g'(x_k)}$$

8. Escriba una función `[xmin, info] = minNewton(f, x0, eps, maxit)` que busca un minimizador de f buscando un raíz de $f'(x) = 0$, mediante el método de Newton. Las derivadas f' y f'' se aproximarán numéricamente.

9. Utilice la función `minNewton` con las funciones de la parte 4.

0.14 Control de paso: método de Fehlberg

Un método muy popular es el de **Runge-Kutta-Fehlberg**, construido a partir de un método de RK de orden 5 (el método A) y de un método de RK de orden 6 (el método B). Una de sus ventajas está dada por el siguiente hecho: los valores K_1, K_2, K_3, K_4 y K_5 son los mismos para los dos métodos.

En la descripción del algoritmo usaremos la siguiente notación de Matlab y de Scilab: `u = [u; t]`

MÉTODO RUNGE-KUTTA-FEHLBERG

datos: x_0, y_0, b, h_0, ε, h_{min}
$x = x_0$, $y = y_0$, $h = h_0$
$X = [x_0]$, $Y = [y_0]$
mientras $x < b$
  $h = \min\{h, b - x\}$
  `hbien` $= 0$
  mientras `hbien` $= 0$
    calcular y_A, y_B
    $e = |y_A - y_B|/h$
    si $e \leq \varepsilon$
      $x = x + h$, $y = y_B$
      `bienh` $= 1$
      $X = [X;\ x]$, $Y = [Y;\ y]$
    fin-si
    $C_0 = 0.84(\varepsilon/e)^{1/4}$
    $C = \max\{C_0, 0.1\}$, $C = \min\{C, 4\}$
    $h = Ch$
    si $h < h_{min}$ **ent parar**
  fin-mientras
fin-mientras

Recuérdese que siempre $K_1 = hf(x_i, y_i)$. La siguiente tabla tiene los coeficientes para el cálculo de los K_j, donde

$$K_j = hf(x_i + \alpha_j h,\ y_i + \beta_{j1}K_1 + \beta_{j2}K_2 + \cdots + \beta_{j,j-1}K_{j-1})$$

j	α_j	β_{j1}	β_{j2}	...		
2	$\frac{1}{4}$	$\frac{1}{4}$				
3	$\frac{3}{8}$	$\frac{3}{32}$	$\frac{9}{32}$			
4	$\frac{12}{13}$	$\frac{1932}{2197}$	$-\frac{7200}{2197}$	$\frac{7296}{2197}$		
5	1	$\frac{439}{216}$	-8	$\frac{3680}{513}$	$-\frac{845}{4104}$	
6	$\frac{1}{2}$	$-\frac{8}{27}$	2	$-\frac{3544}{2565}$	$\frac{1859}{4104}$	$-\frac{11}{40}$

$$y_A = y_i + \frac{25}{216}K_1 + 0K_2 + \frac{1408}{2565}K_3 + \frac{2197}{4104}K_4 - \frac{1}{5}K_5$$
$$y_B = y_i + \frac{16}{135}K_1 + 0K_2 + \frac{6656}{12825}K_3 + \frac{28561}{56430}K_4 - \frac{9}{50}K_5 + \frac{2}{55}K_6$$

Los errores locales son respectivamente $O(h^5)$ y $O(h^6)$. La aproximación del error está dada por

$$|\text{error}| \approx e = \frac{|y_A - y_B|}{h}. \tag{8}$$

El coeficiente para la modificación del valor de h está dado por:

$$C_0 = 0.84\left(\frac{\varepsilon}{e}\right)^{1/4}, \quad C = \min\{C_0, 4\}, \quad C = \max\{C, 0.1\}. \tag{9}$$

Las fórmulas anteriores buscan que C no sea muy grande ni muy pequeño. Más específicamente, C debe estar en el intervalo $[0.1,\ 4]$.

La salida no deseada del algoritmo anterior se produce cuando h se vuelve demasiado pequeño. Esto se produce en problemas muy difíciles cuando, para mantener el posible error dentro de lo establecido, ha sido necesario disminuir mucho el valor de h, por debajo del límite deseado.

Ejemplo: Resolver, por el método RKF con control de paso, la ecuación diferencial

$$\begin{aligned} y' &= 2x^2 - 4x + y \\ y(1) &= 0.7182818 \end{aligned}$$

en el intervalo $[1, 3]$, con $h_0 = 0.5$ y $\varepsilon = 10^{-6}$.

$$\begin{aligned} y_A &= -0.01834063 \\ y_B &= -0.01830704 \\ e &= 0.00006717 \end{aligned}$$

$h = 0.5$ no sirve.

$$\begin{aligned} C_0 &= 0.29341805 \\ C &= 0.29341805 \\ h &= 0.14670902 \\ y_A &= 0.51793321 \\ y_B &= 0.51793329 \\ e &= 0.00000057 \end{aligned}$$

$h = 0.14670902$ sirve.

$$\begin{aligned} x &= 1.14670902 \\ y &= 0.51793329 \\ C_0 &= 0.96535578 \\ C &= 0.96535578 \\ h &= 0.14162640 \\ y_A &= 0.30712817 \\ y_B &= 0.30712821 \\ e &= 0.00000029 \end{aligned}$$

$h = 0.14162640$ sirve.

$$\begin{aligned} x &= 1.28833543 \\ y &= 0.30712821 \end{aligned}$$

x	h	$\tilde{y}(x)$
1.0000000	0.1467090	0.7182818
1.1467090	0.1416264	0.5179333
1.2883354	0.1622270	0.3071282
1.4505624	0.1686867	0.0572501
1.6192491	0.1333497	-0.1946380
1.7525988	0.1329359	-0.3736279
1.8855347	0.1191306	-0.5206051
2.0046653	0.1092950	-0.6137572
2.1139603	0.1024064	-0.6566848
2.2163666	0.0971218	-0.6506243
2.3134884	0.0928111	-0.5948276
2.4062996	0.0891591	-0.4877186
2.4954587	0.0859853	-0.3273334
2.5814440	0.0831757	-0.1114979
2.6646196	0.0806534	0.1620898
2.7452730	0.0783639	0.4958158
2.8236369	0.0762674	0.8921268
2.8999043	0.0743333	1.3535162
2.9742376	0.0257624	1.8825153
3.0000000		2.0855366

1. Considere la ecuación diferencial $y' = 2x\cos(3x) + y$ con la condición inicial $y(a) = d$, donde a es el penúltimo dígito de su número de cédula y d el último dígito. Sea $b = a + 2$, $h_0 = 1$, $\varepsilon = 10^{-4}$, $h_{\min} = 0.001$. Encuentre una solución aproximada por el método anterior en el intervalo $[a, b]$. Muestre en los dos primeros pasos, como en el ejemplo, los valores intermedios: y_A, y_B, e, C_0, C, nuevo h, ...

2. Sea `X = a:0.1:b`. Obtenga la solución aproximada por Scilab en los valores de `X`.

3. Grafique la solución de Scilab.

4. En la misma gráfica, muestre la solución obtenida en 1.

0.15 Método del disparo

Se utiliza para ecuaciones diferenciales de segundo orden con condiciones de frontera.

Considere la ecuación diferencial

$$\begin{aligned} y'' - 3y' + 2y &= 4x + e^{3x} \\ y(a) &= y_a \\ y(b) &= y_b \end{aligned}$$

donde $k_1k_2k_3$ son los 3 últimos dígitos de su documento de identidad, $a = k_1/10$, $b = a + 1$, $y_a = k_2$ y $y_b = k_3$.

Suponga temporalmente, para facilitar la exposición, que las condiciones iniciales son

$$\begin{aligned} y(1) &= 3 \\ y(2) &= 4\,. \end{aligned}$$

1. Obtenga la solución exacta (analítica). Puede hacerlo "a mano" o utilizar un software para matemáticas como Maxima, Derive, Mathematica, Maple, ... o acudir a un amigo(a) que sepa ecuaciones diferenciales lineales de segundo orden con coeficientes constantes no homogenea.
2. Escriba esta ecuación en la forma $y'' = f(x, y, y')$.
3. Mediante un cambio de variable convierta la ecuación en un sistema de dos ecuaciones diferenciales

$$\begin{aligned} u_1' &= \varphi_1(x, u_1, u_2) \\ u_2' &= \varphi_2(x, u_1, u_2) \\ u_1(a) &= y_a \\ u_1(b) &= y_b, \end{aligned}$$

   con $u_1 = y$ y $u_2 = y'$.
4. Convierta este sistema en uno con condiciones iniciales suponiendo algún valor para $u_2(a)$. Por ejemplo suponga que $u_2(1) = 0.5$, es decir se tendría

$$\begin{aligned} u_1' &= \varphi_1(x, u_1, u_2) \\ u_2' &= \varphi_2(x, u_1, u_2) \\ u_1(1) &= 3 \\ u_2(1) &= 0.5\,. \end{aligned}$$

   El anterior sistema se puede escribir

$$\begin{aligned} u' &= \varphi(x, u) \\ u(1) &= \begin{bmatrix} 3 \\ 0.5 \end{bmatrix} \end{aligned} \tag{10}$$

   donde

$$\begin{aligned} u = u(x) &= \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \\ u' &= \begin{bmatrix} u_1' \\ u_2' \end{bmatrix} \\ \varphi(x, u) &= \begin{bmatrix} \varphi_1(x, u_1, u_2) \\ \varphi_2(x, u_1, u_2) \end{bmatrix} \end{aligned}$$

5. Escriba una función `der_u = f14(x, u)` tal que para `x`, un real, y `u`, un vector columna 2×1, implementa la función φ cuyo resultado es también un vector columna 2×1. Puede ser análoga a

```
function der_u = f14(x, u)
  der_u = zeros(2,1)
  der_u(1) = x + u(1) + u(2)
  der_u(2) = x*u(1)*u(2)
endfunction
```

6. Aplique RK4 con $h = 0.1$ para resolver (10) en el intervalo $[a, b]$. El resultado será una matriz U de dos filas en la que las columnas corresponden a las aproximaciones $\tilde{u}(1)$, $\tilde{u}(1.1)$, $\tilde{u}(1.2)$, ..., $\tilde{u}(2)$.

7. Si la suposición $u_2(1) = 0.5$ hubiera sido correcta, entonces se tendría $\tilde{u}_1(2) = 4$. Haga otras suposiciones para $u_2(1)$ (ensayo y error) hasta que $\tilde{u}_1(2) \approx 4$.

8. Sea $t = u_2(a)$ y $g = \tilde{u}_1(b)$ el valor obtenido después de haber aplicado RK4. Es claro que g depende de t, es decir, $g = g(t)$. Se desea que $g(t) = u_1(b) = y_b$. O sea, se desea resolver

$$G(t) = 0$$

   donde

$$G(t) = g(t) - y_b.$$

   Adapte el método de la secante para resolver $G(t) = 0$.

9. Ya resuelta la anterior ecuación, la última matriz U es (aproximadamente) correcta, o sea, en la primera fila estarán los valores aproximados de $u_1 = y$.

   Haga una tabla con los valores x, $y(x)$ exacto (obtenido según la solución en el paso 1.) y $\tilde{u}_1$.

10. Grafique la solución exacta y la solución obtenida numéricamente.

0.16 Trazadores cúbicos (splines)

Dados n puntos (x_1, y_1), (x_2, y_2), ... (x_n, y_n), con

$$x_1 < x_2 < \cdots < x_n,$$

el trazador cúbico, spline, o función polinomial cúbica por trozos, se define así:

$$S(x) = \begin{cases} S_1(x) & \text{si } \ x \in [x_1, x_2] \\ S_2(x) & \text{si } \ x \in [x_2, x_3] \\ \vdots & \\ S_{n-1}(x) & \text{si } \ x \in [x_{n-1}, x_n] \end{cases} \tag{11}$$

En cada uno de los $n-1$ intervalos, $S_i(x)$ es un polinomio cúbico.

$$S_i(x) = a_i(x - x_i)^3 + b_i(x - x_i)^2 + c_i(x - x_i) + d_i, \quad i = 1, ..., n-1. \tag{12}$$

Conocer $S(x)$ quiere decir conocer $4(n-1)$ coeficientes: a_i, b_i, c_i, d_i, para $i = 1, ..., n-1$.
Se requiere que $S(x)$ pase por los puntos, y que sea doblemente derivable.

$$\begin{aligned} S(x_i) &= y_i, \quad i = 1, ..., n \\ S_i(x_{i+1}) &= S_{i+1}(x_{i+1}), \quad i = 1, ..., n-2 \\ S_i'(x_{i+1}) &= S_{i+1}'(x_{i+1}), \quad i = 1, ..., n-2 \\ S_i''(x_{i+1}) &= S_{i+1}''(x_{i+1}), \quad i = 1, ..., n-2 \end{aligned}$$

Supongamos además que S tiene condiciones de "frontera libre":

$$\begin{aligned} S_1''(x_1) &= 0\,, \\ S_{n-1}''(x_n) &= 0\,. \end{aligned}$$

Sea $h_j = x_{j+1} - x_j$, el tamaño del intervalo $[x_j, x_{j+1}]$. Las condiciones anteriores se convierten en:

$$\begin{aligned} S_i(x_i) = d_i = y_i & \qquad i = 1, ..., n-1, \\ S_{n-1}(x_n) = a_{n-1}h_{n-1}^3 + b_{n-1}h_{n-1}^2 + c_{n-1}h_{n-1} + d_{n-1} = y_n & \\ a_ih_i^3 + b_ih_i^2 + c_ih_i + d_i = d_{i+1} & \qquad i = 1, ..., n-2, \\ 3a_ih_i^2 + 2b_ih_i + c_i = c_{i+1} & \qquad i = 1, ..., n-2, \\ 6a_ih_i + 2b_i = 2b_{i+1} & \qquad i = 1, ..., n-2. \end{aligned}$$

Para resolver estas ecuaciones:

- $d_i = y_i, \quad i = 1, ..., n-1.$
- $h_i = x_{i+1} - x_i, \quad i = 1, ..., n-1.$

- resolver $Ab = \zeta$, sistema tridiagonal $n \times n$. Sea $\alpha \in \mathbb{R}^n$ la diagonal, $\beta \in \mathbb{R}^{n-1}$ la superdiagonal, $\gamma \in \mathbb{R}^{n-1}$ la subdiagonal,

$$\begin{aligned}
\alpha &= (1,\, 2(h_1 + h_2),\, 2(h_2 + h_3),\, ...,\, 2(h_{n-2} + h_{n-1}),\, 1)\,, \\
\beta &= (0,\, h_2,\, h_3,\, ...,\, h_{n-1}) \\
\gamma &= (h_1,\, h_2,\, ...,\, h_{n-2},\, 0) \\
\zeta_1 &= 0 \\
\zeta_i &= 3(y_{i-1} - y_i)/h_{i-1} \,+\, 3(-y_i + y_{i+1})/h_i, \quad i = 2, ..., n-1, \\
\zeta_n &= 0
\end{aligned}$$

  Temporalmente, $b \in \mathbb{R}^n$.

- $c_i = \dfrac{1}{h_i}(y_{i+1} - y_i) - \dfrac{h_i}{3}(2b_i + b_{i+1}), \quad i = 1, ..., n-1.$

- $a_i = \dfrac{b_{i+1} - b_i}{3h_i}, \;\; i = 1, ..., n-1.$

- suprimir b_n .

1. Escriba una función `A = matrTrid(d, supd, subd)` que construye una matriz tridiagonal. Los parámetros son vectores columna de n, $n-1$ y $n-1$ elementos respectivamente. Corresponden a la diagonal, superdiagonal y subdiagonal.

2. Escriba una función `x = solTrid(d, supd, subd, b)` que resuelve el sistema tridiagonal $Ax = b$. Puede usar un método eficiente para matrices tridiagonales o simplemente construir explícitamente la matriz tridiagonal.

3. Sean k_1k_2 los dos últimos dígitos del número de su cédula. Considere los puntos
   $(1,1)$, $(3,1)$, $(4,(5+k_1+k_2)/10)$, $(5,1)$, $(7,1)$.
   Con ellos construya dos vectores columna `x` , `y` .

4. Escriba un función `[a, b, c, d] = splineCub(x, y)` que calcula los coeficientes del trazador cúbico $S(x)$. Utilícela para los vectores `x` , `y` .

5. Escriba una función `st = evalSpline(x, a, b, c, d, t)` tal que dado los valores x_i en `x`, los vectores `a, b, c` y `d` con los coeficientes de $S(x)$ y un número t, calcula $S(t)$.

$$\begin{aligned}
j &= \begin{cases} 1 & \text{si} \quad t \le x_2 \\ n-1 & \text{si} \quad t \ge x_{n-1} \\ i & \text{si} \quad x_i \le t \le x_{i+1} \end{cases} \\
S(t) &= S_j(t)
\end{aligned}$$

6. Grafique los puntos (x_i, y_i) .

7. Construya el vector columna `T = ( x(1):0.1:x(n) )'` . Construya un vector columna `S` con los valores de S en los puntos `T(i)` .

8. Grafique los puntos cuyas coordenadas están en `S` y `T` .