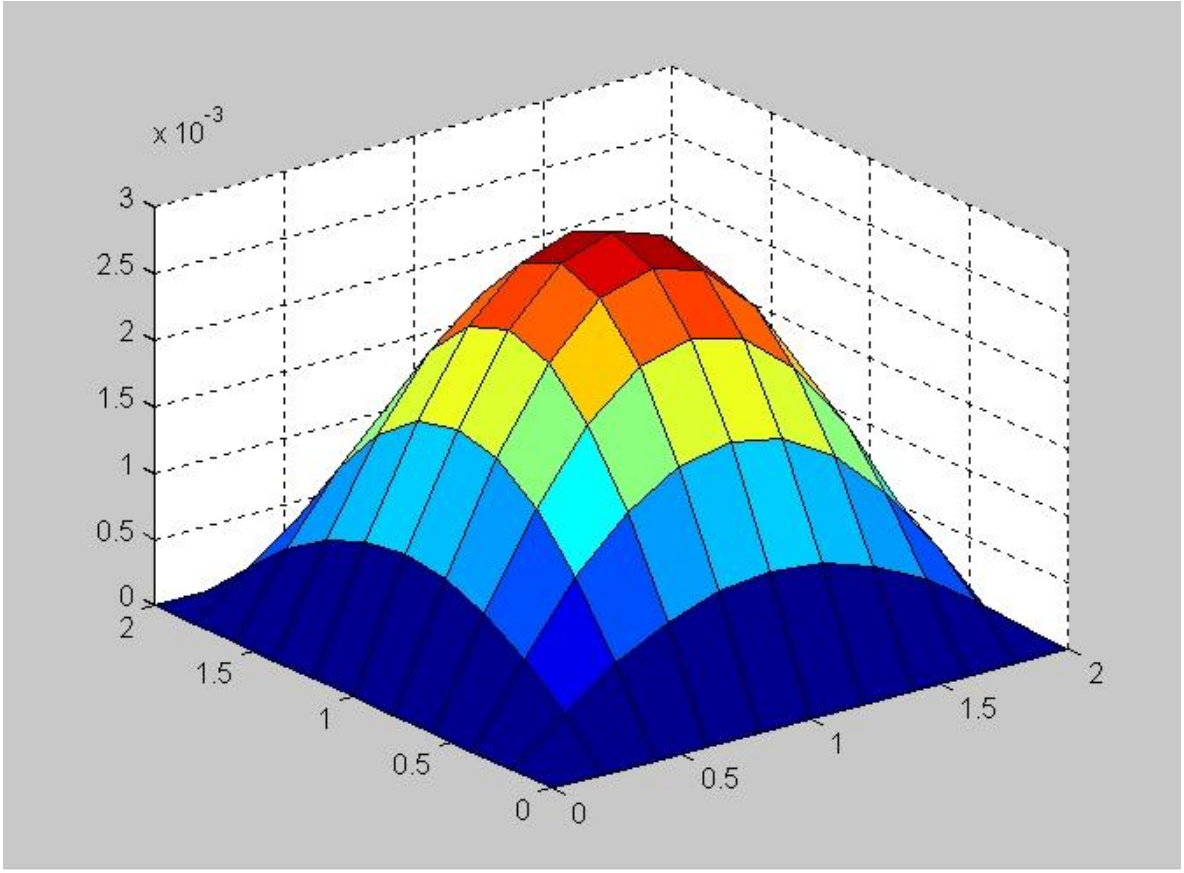
MÉTODOS NUMÉRICOS

EN CALC-JAVA



HERNAN PEÑARANDA V.

materias-hpv.comli.com

Sucre, 2011

1. INTRODUCCIÓN A CALC-JAVA

1.1. Introducción

En esta materia estudiaremos métodos que nos permitirán resolver problemas matemáticos complejos mediante la realización de una elevado número de cálculos aritméticos simples, sin embargo, debido al elevado número de cálculos que se deben realizar, estos métodos sólo son de utilidad práctica si se aplican con la ayuda de una computadora o una calculadora programable.

En el ámbito de las computadoras contamos con una gran variedad de lenguajes y software que permiten automatizar los procesos de cálculo, sin embargo, actualmente y a pesar de que los costos de las computadoras portátiles han rebajado considerablemente, sólo un pequeño porcentaje de los estudiantes cuentan con una de ellas, por lo que no es posible en la práctica emplear dicha herramienta.

Por ello es necesario pensar en otros dispositivos como las calculadoras programables. Pero aún en este ámbito sólo un porcentaje reducido de estudiantes cuentan con una calculadora programable (como la HP) con capacidades suficientes de procesamiento y memoria como para una adecuada enseñanza de la materia.

Por todo lo anterior nos vemos en la necesidad de recurrir a otros medios no convencionales para la enseñanza de la materia. En ese sentido y tomando en cuenta que la mayoría de los estudiantes cuentan con un teléfono móvil con la tecnología Java corriendo en ellos, se ha optado por emplear dicha herramienta para la enseñanza de la materia.

Si bien es cierto que los teléfonos móviles han sido creados con fines muy diferentes a los del cálculo numérico, no es menos cierto que poseen capacidades de procesamiento y memoria superiores a los de la mayoría de las calculadoras programables (incluida la HP), por lo que tienen el potencial suficiente como para ser empleados en el cálculo numérico y en la solución de problemas ingenieriles en general.

En este sentido existen ya algunos avances y actualmente se cuenta con algún software para este propósito. De dicho software podemos mencionar: *Forty8*, que es un emulador de la calculadora HP, pero que lamentablemente todavía no está del todo concluido; *MathPro*, que cuenta con un lenguaje de programación de alto nivel, un editor y la mayoría de las herramientas que posibilitan una adecuada programación en un celular, no obstante, es un software comercial y su compilador no es precisamente de los más eficientes; *Calc-Java*, que es una calculadora programable que cuenta con prácticamente todos los operadores y funciones matemáticas como para resolver la mayoría de los problemas numéricos, sus principales inconvenientes radican en el manejo de archivos (limitado a 16) y de registros (limitado también a 16).

Por sus características el software que emplearemos para la enseñanza de la materia será “Calc-Java”, sin embargo, cuando así sea conveniente recurriremos también a otras herramientas (como MathPro, Solve2Go, MobileMaths, etc.)

El propósito de este tema es que el estudiante se familiarice con el manejo de *Calc-Java* y que esté en condiciones de calcular el valor de expresiones matemáticas con esta herramienta.

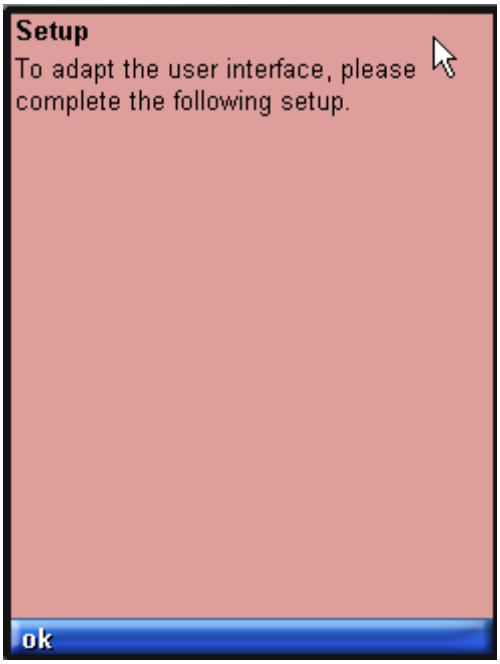
1.2. Instalación de “Calc-Java”

Lo primero que debemos hacer para trabajar con “Calc-Java” es bajar el software de Internet: http://midp-calc.sourceforge.net/Calc.html. *Calc-Java* se distribuye bajo la licencia GNU, por lo que no sólo es gratuito, sino que además el software puede ser mejorado o modificado sin necesidad de pagar permisos o licencias.

Actualmente existen 4 versiones del software las mismas que se ajustan a la mayoría de los celulares. Si se cuenta con un celular relativamente antiguo, entonces debe tratar la versión MIDP1 (Calc.jar), si se tiene un celular relativamente reciente es mejor probar la versión MIDP2 (CalcMIDP2.jar). Existen versiones específicas también para los modelos Nokia y Siemens. Debe bajar los instaladores a su computadora, pasarlos al celular y probar con las diferentes versiones existentes hasta que encuentre una que se adapte bien a su modelo.

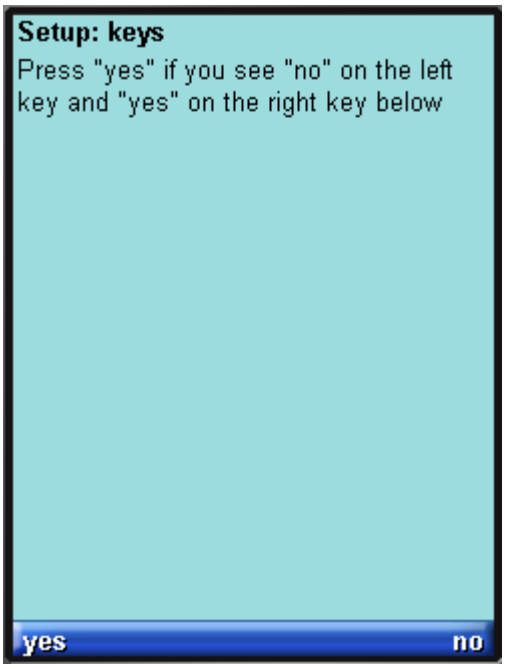
Alternativamente puede habilitar la navegación en Internet desde su celular, ingresar a la página (consulte con su proveedor del servicio telefónico para este fin), bajar los archivos (se recomienda probar primero los archivos .jad) e instalarlos en su celular.

Una vez que tenga el software de instalación en su celular, haga correr el mismo, entonces le aparecerá una pantalla similar a la que se muestra en la siguiente figura:

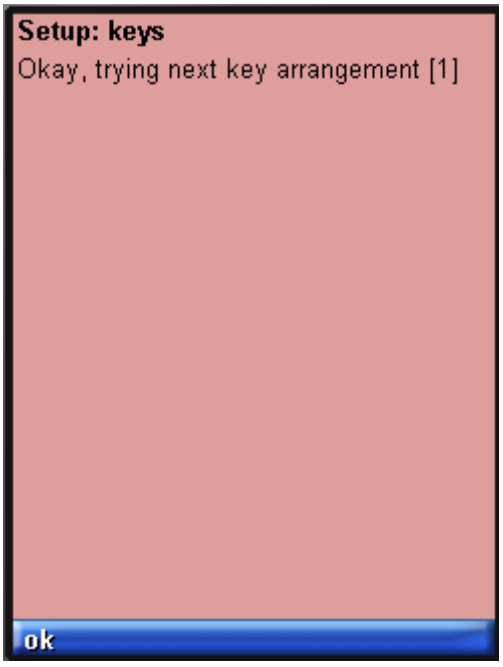


Donde se nos informa que debemos completar algunos pasos para configurar la interfaz del usuario, básicamente se trata de configurar tres teclas: las teclas izquierda, derecha y de borrar (clear o “C”).

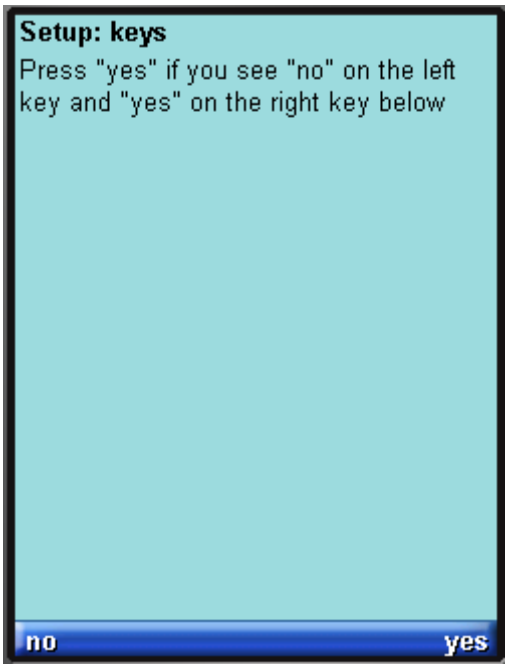
Para pasar a la siguiente pantalla simplemente pulsamos el botón correspondiente a “ok”, con lo que aparece la siguiente pantalla:



Si la palabra “yes” aparece a la derecha, entonces se pulsa el botón derecho y el proceso para esta etapa concluye, caso contrario, como ocurre en el ejemplo, se pulsa el botón correspondiente a la letra “no” (normalmente el derecho) y aparecerá una pantalla como la siguiente:

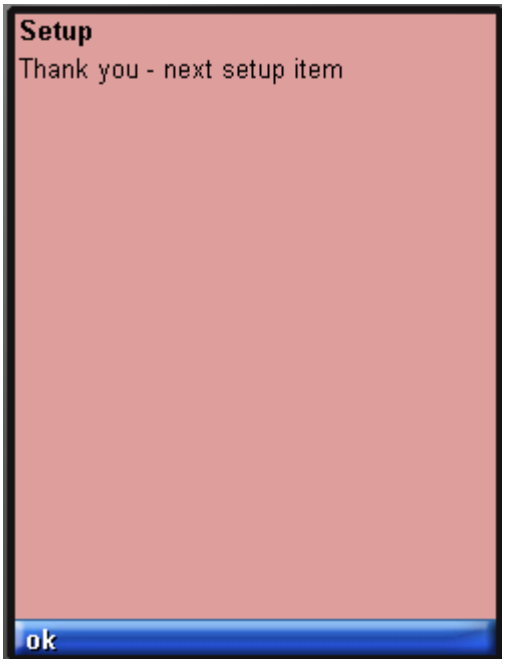


Donde se nos informa que se intentará otro arreglo. Para continuar se pulsa el botón correspondiente a la palabra “ok” y entonces vuelve a aparecer una pantalla similar a la del paso anterior, pero usualmente con la posición de las palabra “yes” y “no” modificadas, tal como se muestra en la siguiente figura:

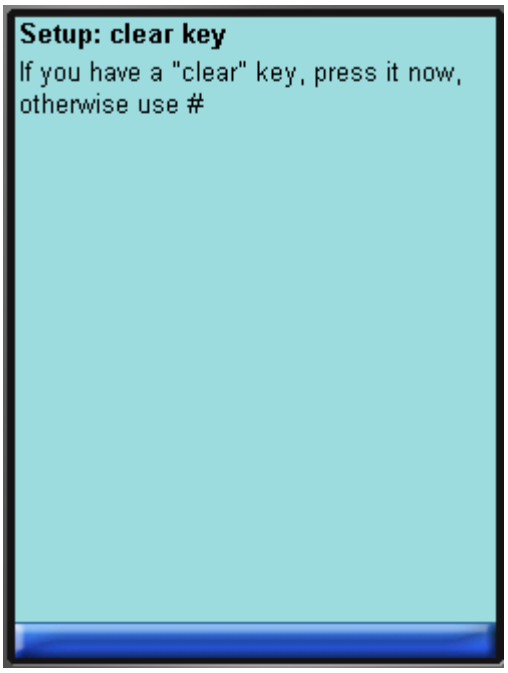


Si ahora las letras “yes” y “no” aparecen como en la figura, entonces la palabra “yes” ya está en la posición correcta, por lo que se pulsa el botón derecho y se concluye de esta manera con esta etapa de la configuración, caso contrario se vuelve a pulsar el botón “no” y se repite el proceso hasta que la palabra “yes” esté como se muestra en la figura (a la derecha).

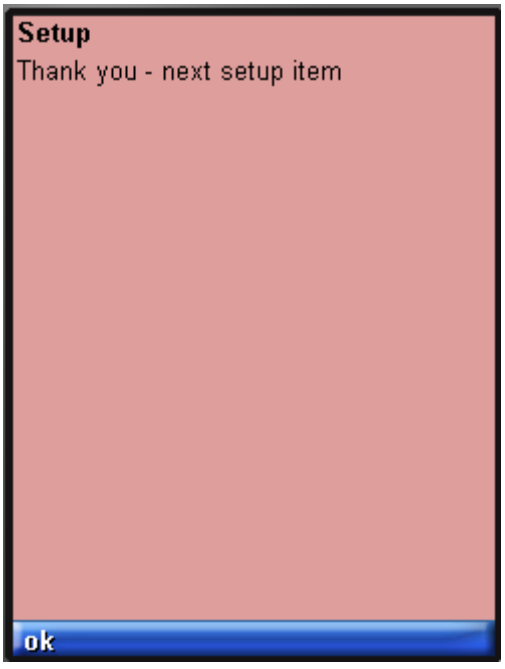
Una vez concluida esta etapa aparece la siguiente pantalla:



Que simplemente nos informa que ya hemos concluido la primera etapa. Entonces pasamos a la siguiente pulsando el botón “ok”, con ello aparece la siguiente pantalla:

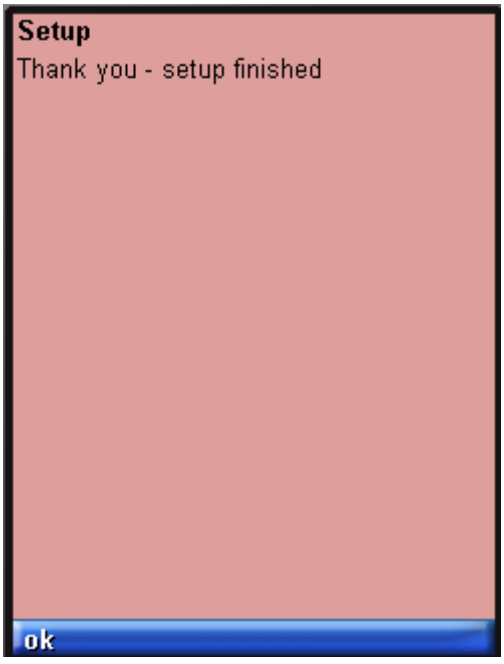


Ahora el programa nos pide que pulsemos el botón que en nuestro celular empleamos para borrar. Normalmente este botón tiene le letra “C” (de clear), pero puede variar según los modelos y las marcas. Para concluir con esta etapa simplemente pulsamos el botón que en nuestro celular empleamos para borrar. Si su celular no cuenta con dicha tecla, entonces deberá pulsar la tecla “#” (numeral). En cualquier de los dos casos aparece la siguiente pantalla:

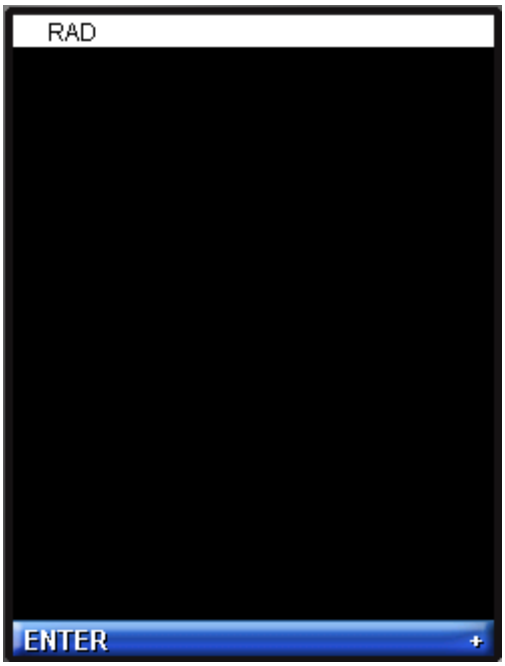


Que simplemente nos informa que hemos concluido la segunda etapa. Entonces pasamos a la tercera etapa pulsando el botón correspondiente a la tecla “ok”, con lo que aparece la siguiente pantalla:

Esta etapa es sólo cuestión de preferencia: si se prefiere ver los números a la izquierda se pulsa el botón correspondiente a la palabra “left”, caso contrario el botón correspondiente a la palabra “right”, con lo que aparece la siguiente pantalla:



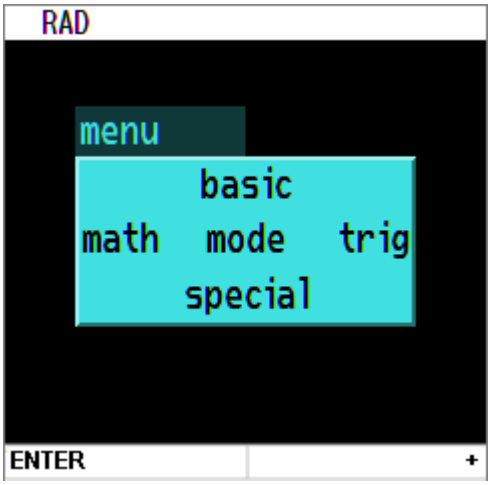
Que simplemente nos informa que el proceso de configuración ha concluido. Finalmente pulsamos el botón correspondiente a la palabra “ok” y ahora deberá aparecer una pantalla similar al que se muestra en la siguiente figura y que corresponde a la pantalla de la calculadora. Ahora ya estamos en condiciones de trabajar con la calculadora.



1.3. Sistema de menús en "Calc-Java"

La calculadora nos permite realizar prácticamente todas las operaciones matemáticas que son de utilidad tanto en ciencias exactas como en ingeniería. Todas las operaciones y funciones matemáticas (así como las funciones estadísticas, financieras y otras) están disponibles a través de una serie de menús que son accesibles mediante las teclas de navegación (los botones que empleamos en nuestro celular para movernos a través de las opciones del menú de nuestro celular). Dichas teclas normalmente están ubicadas en la parte superior del teclado numérico y en algunos casos se trata de una palanca de mando. Cuando no se cuenta con dichas teclas, las teclas de los números 4, 6, 8 y 2 hacen las veces de las teclas de navegación izquierda, derecha, arriba y abajo, mientras que la tecla número 5 hace las veces del botón central.

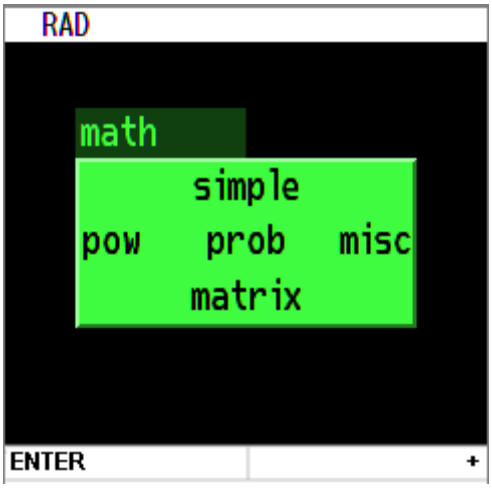
Para acceder al menú principal se pulsa el botón central del navegador, con lo que aparece en el centro de la pantalla el siguiente menú:



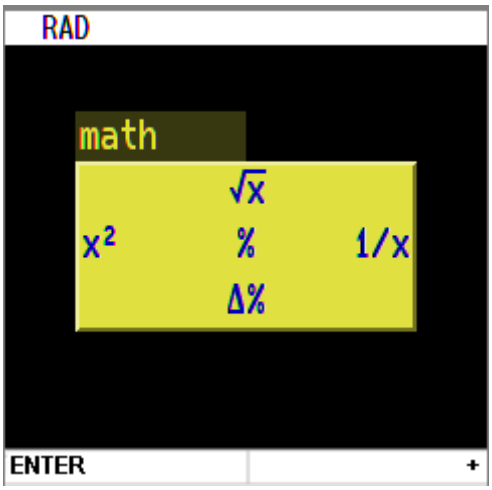
Nota: En caso de que el botón central del celular no funcione, se puede acceder primero a uno de los menús secundarios, pulsando cualquiera

de las teclas de navegación y desde allí volver al menú principal pulsando la tecla de borrado.

A cada uno de los submenús se accede con la tecla de navegación respectiva, así para acceder al menú “basic” (básico) se pulsa el botón de navegación hacia arriba, para “special” el de abajo, para “math” el de la izquierda y para “trig” el de la derecha. También es posible acceder directamente a los submenús, sin ingresar al menú principal, pulsando directamente una de las teclas de navegación, así para acceder directamente al menú “math” se pulsa directamente la tecla de navegación izquierda, con lo que aparecerá el siguiente menú:



Dentro de cada submenú, se accede a la opción respectiva de la misma forma: pulsando el botón de navegación respectivo, así para acceder al submenú simple (dentro del menú “math”) pulsamos la tecla de navegación hacia arriba, con lo que aparece el siguiente menú:

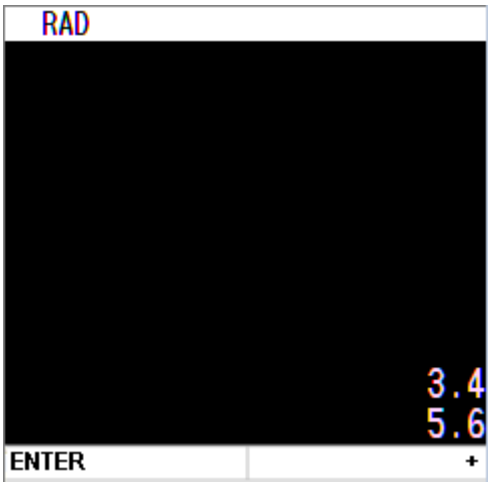


Para volver a un menú previo (o salir del menú principal) se pulsa la tecla de borrado.

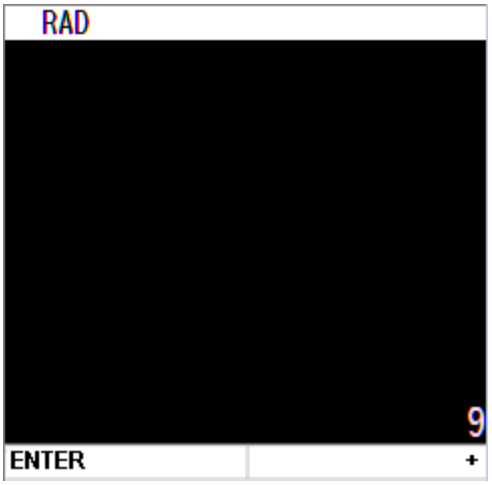
1.4. Cálculos de expresiones matemáticas en “Calc-Java”

Ahora que sabemos cómo navegar a través de los menús de *Calc-Java* estamos en condiciones de calcular el resultado de algunas expresiones matemáticas. Comenzaremos con algunas operaciones básicas. Para ello debemos tomar en cuenta que cuando se trabaja con *Calc-Java* se trabaja en el modo “RPN” (No-

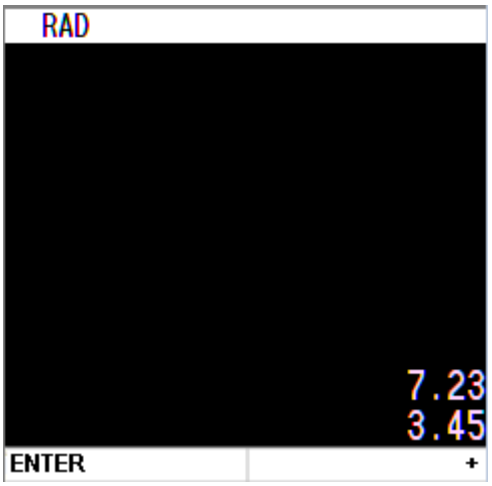
tación Polaca Inversa), lo que básicamente significa que primero se deben escribir los datos y luego elegir la operación o función que utiliza dichos datos. Así por ejemplo para sumar 3.4+5.6 escribimos primero el número 3.4 y pulsamos “ENTER”, luego escribimos el número 5.6 y volvemos a pulsar “ENTER” con lo que la pantalla de *Calc-Java* queda de la siguiente forma:



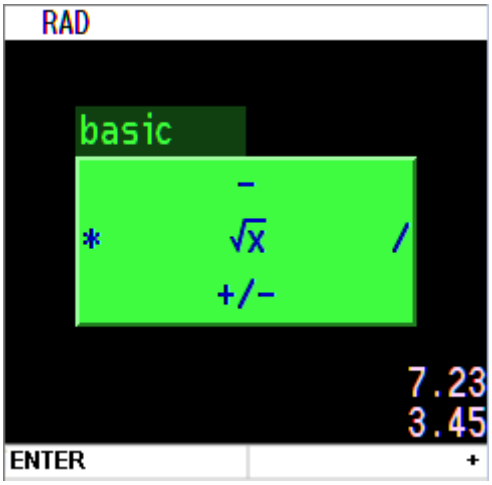
Ahora que tenemos los dos datos, elegimos la operación, es decir pulsamos la tecla correspondiente al botón “+”, con lo que obtenemos el resultado:



De manera similar para restar: 7.23-3.45, escribimos primero los números:



Luego accedemos al menú “basic” (navegador arriba):



Y elegimos la resta (navegador arriba), obteniendo así el resultado buscado:

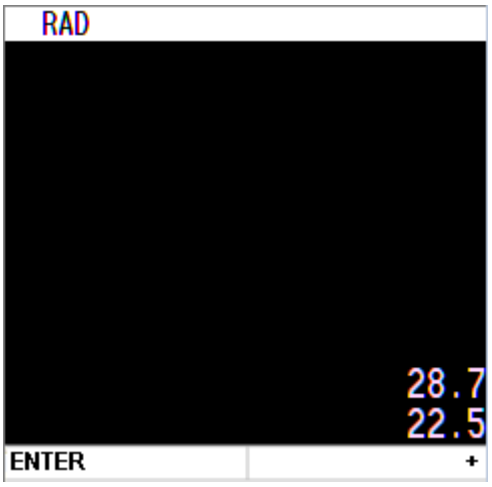
Otro aspecto que se debe tomar en cuenta, cuando se trabaja en RPN, es que no existen paréntesis, por lo que se deben realizar primero las operaciones que se encuentran en numeradores, denominadores o dentro de otras funciones u operadores y luego, con esos resultados realizar las operaciones restantes.

Para un mismo nivel, es decir para operaciones que se encuentran dentro del mismo numerador, denominador o función, se debe seguir el siguiente orden de prioridad: primero se deben realizar todas las operaciones de mayor nivel, tales como la potenciación, exponenciación, raíces cuadradas, funciones trigonométricas, etc., luego las operaciones de multiplicación y división y finalmente las operaciones de suma y resta.

Por ejemplo para calcular el resultado de la siguiente expresión:

$$\frac{4.1*7+7.5*3}{2.1*5-1.7*2}$$

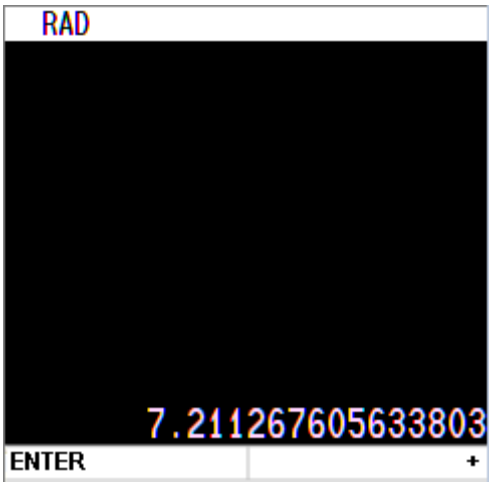
Realizamos por separado las operaciones del numerador y del denominador. En el numerador por su parte primero llevamos a cabo las multiplicaciones (que tienen un mayor nivel que las sumas): 4.1/ENTER; 7/*; 7.5/ENTER; 3 *, con lo que obtenemos:



Ahora sumamos esos dos valores para obtener el valor del numerador:

Procedemos de la misma forma con el denominador: 2.1/ENTER; 5/*; 1.7/ENTER; 2/*; -:

Finalmente dividimos estos dos valores (que son los valores del numerador y denominador) para obtener el valor final de la expresión: /:



En los siguientes ejemplos presentamos los pasos que se deben seguir en *Calc-Java* para calcular los resultados de algunas expresiones matemáticas.

1.4.1. Ejemplos

Calcule el valor de las siguientes expresiones:

1. $\sqrt[3]{\frac{\sin(9.2)-\cos(8.32)}{\tan(6.89)}}$

```
9.2/trig/normal/sin
8.32/trig/normal/sin
basic/-
6.89/trig/normal/tan
basic//
3/math/pow/x√y
```

Con lo que en *Calc-Java* se obtiene:

```
0.9893394872476779
```

2. $\frac{\left|\sinh^{-1}(2.2)-\cosh^{-1}(3.4)\right|}{\tanh^{-1}(0.9)}$

```
2.2/trig/archyp/asinh
3.4/trig/archyp/acosh
basic/-
0.9/trig/archyp/atanh
basic//
```

Siendo el resultado:

```
0.2478560569016753
```

3. $\left(\ln\left(\csc(7)+\sec(9.2)+\cot(3.4)\right)\right)^{9.3}$

```
7/trig/normal/sin; math/simple/"1/x"
9.2/trig/normal/cos; math/simple/"1/x"
+
3.4/trig/normal/tan; math/simple/"1/x"
+
math/pow/ln
9.3/math/pow/y^x
```

Siendo el resultado:

32.46983884042057

4. $\sqrt[5]{\frac{\sin(45°)-\cos(70°)}{3.4\tan(80°)}}$

trig/more/R->D->G
45/trig/normal/sin
70/trig/normal/cos
basic/-
3.4/ENTER
80/trig/normal/tan
basic/*
basic//
5/math/pow/x√y

Siendo el resultado:

0.4523214240258878

5. Parte entera de: $4.2^{7.6}-9.2^{8.2}$

4.2/ENTER
7.6/math/pow/y^x
9.2/ENTER
8.2/math/pow/y^x
basic/-
math/misc/int/trunc

Siendo el resultado:

-79939972

6. Entero más pequeño mayor o igual al resultado de: $(6.8*\sin(125°))^3$

trig/more/R->D->G (hasta que aparezca “DEG” en la pantalla)
6.8/ENTER
125/trig/normal/sin
basic/*
3/math/pow/y^x
math/misc/int/ceil

Siendo el resultado:

173

7. Resíduo de la división: $\frac{(6+5)!\ 5^3}{(3+4)^7}$

6/ENTER
5/+
math/prob/x!
5/ENTER
3/math/pow/y^x
basic/*
3/ENTER
4/+
7/math/pow/y^x
math/misc/mod

Siendo el resultado:

576506

8. $\frac{(4(6+3))!}{\sqrt{e^{85.4}}}$

4/ENTER
6/ENTER
3/+
basic/*
math/prob/x!
85.4/math/pow/e^x
math/simple/√
basic//

Siendo el resultado:

1.062088659949585e23

9. $\frac{e^{6!+5^6}}{3.45^{6.75^{4.3}}}$

6/math/prob/x!
5/ENTER
6/math/pow/y^x
+
math/pow/e^x
3.45/ENTER
6.75/ENTER
4.3/math/pow/y^x
math/pow/y^x
basic//

Siendo el resultado:

4.436305712367519e5118

10. $\sqrt[3]{\frac{(3+4!)-6^{3.2}}{e^{6.5}+\cos(6.7°)}}$

trig/more/R->D->G (hasta que aparezca “DEG” en pantalla)
3/ENTER
4/math/prob/x!
+
6/ENTER
3.2/math/pow/y^x
basic/-
6.5/math/pow/e^x
6.7/trig/normal/cos
+
basic//
3/math/pow/x√y

Siendo el resultado:

-0.759450757226137

11. $\frac{\sinh(6.5)\cosh(9.2)\tanh(9.3)}{45.2+6.3^{2.2}+6!}$

6.5/trig/hyp/sinh
9.2/trig/hyp/cosh

```
basic/*
9.3/trig/hyp/tanh
basic/*
45.2/ENTER
6.3/ENTER
2.2/math/pow/y^x
+
6/math/prob/x!
+
basic//
```

Siendo el resultado:

```
200.778365413566
```

1.4.2. Ejercicios

Calcule el valor de las siguientes expresiones:

1. $\ln(6.573)e^{7.8234}$
2. $\sqrt{\frac{\log(6.75)}{6.2+3^{6.7}}}\cos(30.2°)$
3. $sin^{-1}\left(\frac{\sqrt{7.2}+9.67}{8.43}\right)$
4. $e^{\frac{(\sinh(4.3))^{3.2}+(\cosh(2.3))^{5.4}}{\tan(9.8)}}$
5. $\sqrt[3]{(50!)\left(4^7\right)}$
6. $\sqrt{\frac{5^3-2^2+3!}{(7-3)^2}}$
7. Valor redondeado de: $\ln(6.7)+\log(4.3)$
8. Parte fraccionaria de: $\left(\sin(9.2)-\frac{\sec(5.6)}{\sinh(9.2)}\right)^{0.34}$
9. Entero más pequeño menor o igual al resultado: $\ln\left(\frac{4.5^{9.2}-9.8^{7.6}+e^{9.87}}{\sqrt{6.32+2^{0.98}}}\right)$
10. Cociente de la división: $\frac{5!+3^4}{4^3}$
11. $\frac{\sin^{-1}(0.35)+\cos^{-1}(0.89)+\tan^{-1}(5.45)}{3!e^{4/5}\pi}$
12. $Log\left(\frac{4.5^{3.2}+9.8^{1.6}+e^{4.5}}{\sqrt[5]{6.32+2^{0.98}}}\right)$

2. PROGRAMACIÓN DE FUNCIONES EN CALC-JAVA

En la solución de la mayoría de los problemas numéricos es necesario trabajar con alguna función (o ecuación), razón por la cual es importante aprender a programarlas.

En este tema haremos justamente eso, aprenderemos a programar funciones dependientes de una sola variable. Luego emplearemos el programa para calcular valores de la función en determinados puntos, graficar la función, encontrar las raíces o soluciones de la función, así como derivar e integrar la función.

En consecuencia, el objetivo del presente tema es que al concluir el mismo estén capacitados para programar cualquier función dependiente de una sola variable, así como emplear el programa elaborado para calcular valores puntuales, graficar, resolver, derivar e integrar la función programada.

2.1. El entorno de programación de Calc-Java

Para programar una función en Calc-Java se siguen los mismos pasos que para encontrar el valor de una expresión matemática, sólo que en una función se trabaja con valores cambiantes. Por esa razón, por el hecho de que cambian, dichos valores se conocen como *"variables".*

En Calc-Java, las variables pueden ser implementadas en dos formas: como posiciones de memoria o como posiciones en la pila.

Supongamos que queremos programar la siguiente función:

$$f(x) = x^3 \qquad (2.1)$$

Es decir queremos crear una función que calcule el cubo de un número. Así si "x" es 3, el valor de la función es:

$$f(3)=3^3=3*3*3=27$$

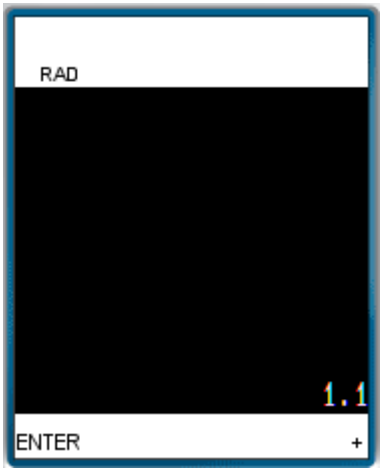
Si "x" es 2.1, el valor de la función es:

$$f(2.1)=2.1^3=2.1*2.1*2.1=9.261$$

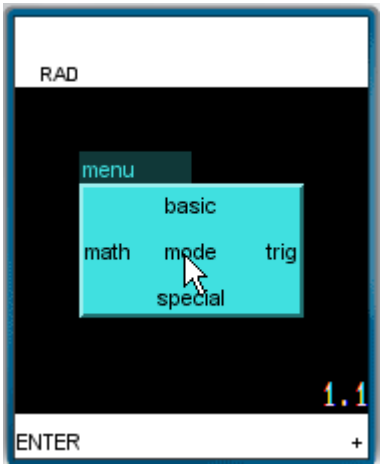
Y así para cualquier valor de "x". El propósito de elaborar un programa es justamente el evitarnos el trabajo de realizar una y otra vez las mismas operaciones. Cuando se elabora un programa, realizamos las operaciones una sola vez (cuando elaboramos el programa) y luego es el programa quien se encarga de ejecutarlas automáticamente las veces que se requiera.

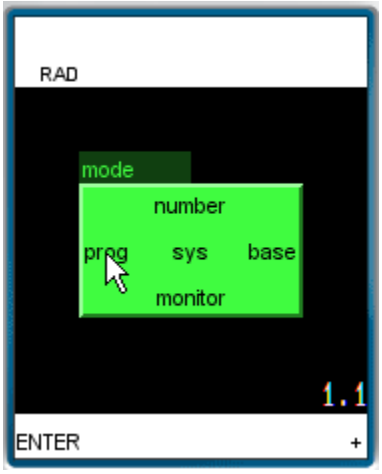
Vemos entonces como podemos programar esta simple función en Calc-Java. Haremos primero una versión del programa trabajando sólo con la pila. Cuando se trabaja en la pila, los datos (en este caso el valor de "x") deben encontrarse en la pila (en la primera o primeras posiciones de la pila). Aún en un programa tan sencillo como el presente, es necesario tener clara la lógica que seguirá el programa. En este caso por ejemplo para calcular el cubo de un número necesitamos multiplicar tres veces el mismo valor, dado que el valor a multiplicar se encuentra en la pila, lo que debemos hacer es duplicar dos veces ese valor (con ENTER) y luego multiplicar los tres valores que quedan en la pila.

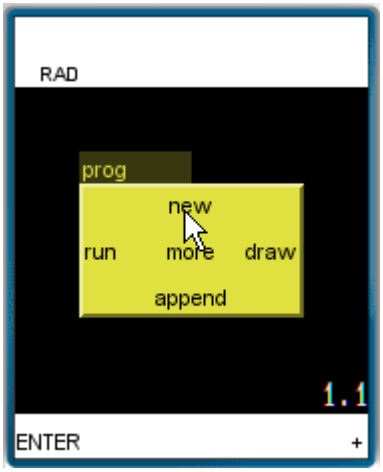
Ahora que tenemos clara la lógica, elaboremos el programa. Para ello escribimos en la pila un valor de prueba cualquiera, por ejemplo 1.1 (este es el valor de x):



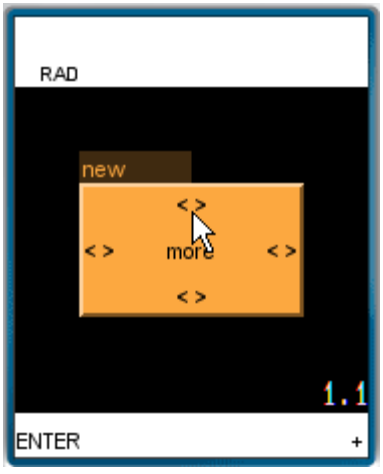
Ahora accedemos al menú “prg” y dentro elegimos la opción “new”:

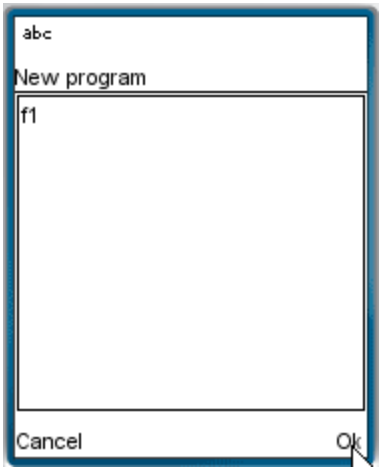




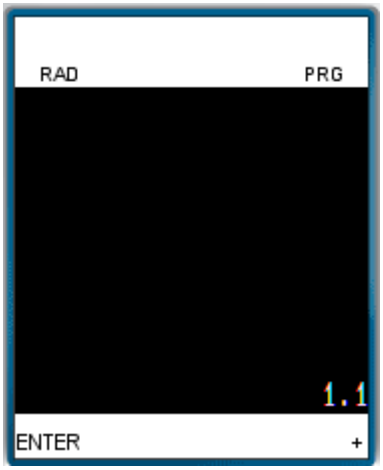


Elegimos una de las posiciones disponibles para guardar el programa, le damos un nombre, por ejemplo “f1” y pulsamos el botón correspondiente a la palabra “Ok”:

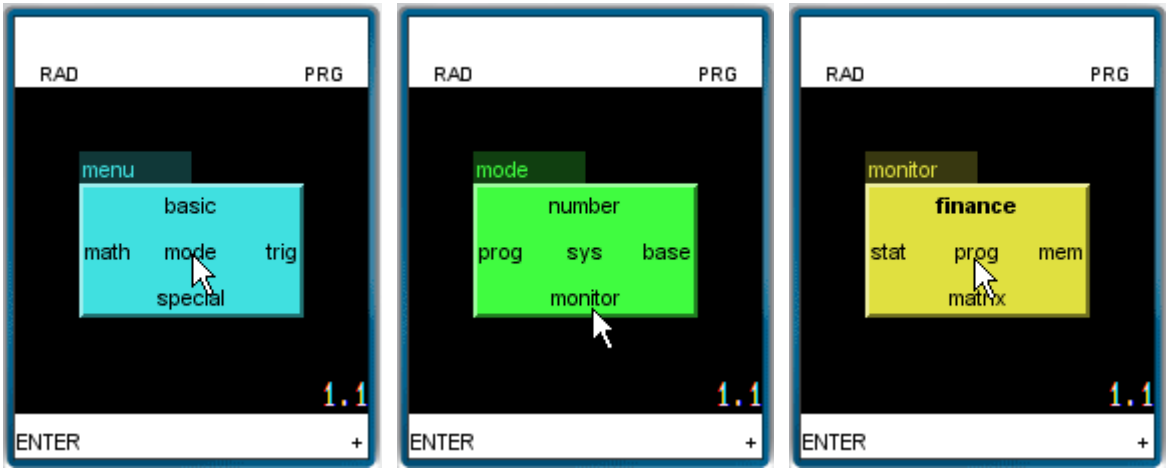




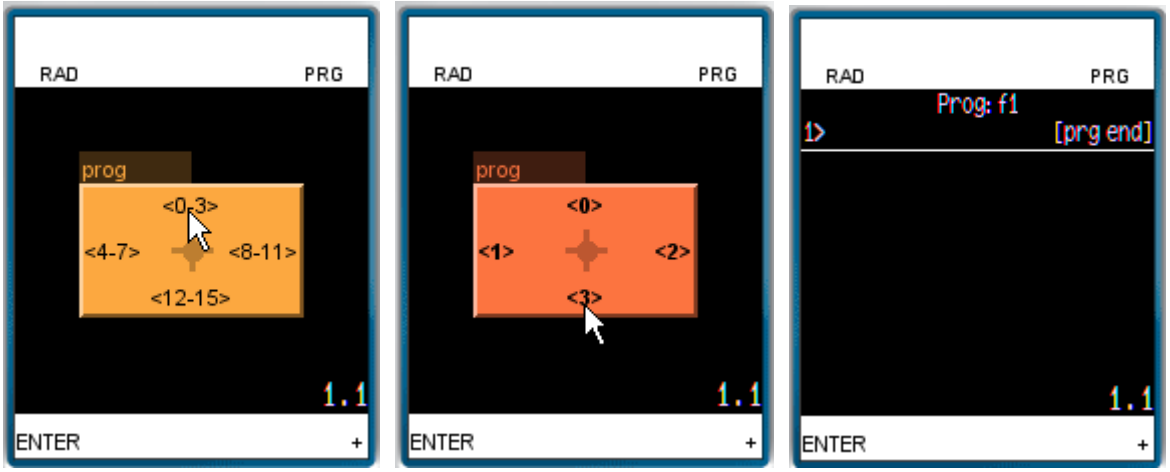
Ahora ya nos encontramos en el entorno de programación de Calc-Java, pero la única diferencia visible, con relación al entorno normal de trabajo, es que aparece la palabra “PRG” en la parte superior derecha de la pantalla:



Si bien no es imprescindible, es conveniente activar el “monitor” de programación, para ver tanto los pasos que programamos como para hacer las correcciones que fueran necesarias:



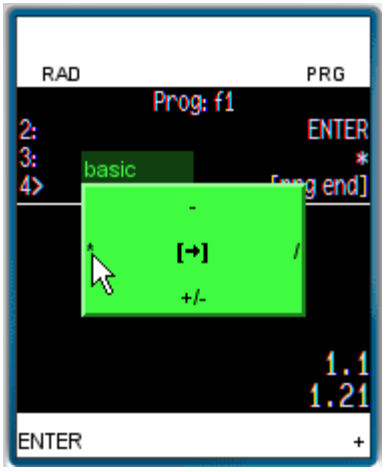
El número de líneas que tendrá el monitor se elije en función al tamaño de pantalla que tengamos disponible en nuestro celular:



Como se puede observar, el monitor aparece en la parte superior de la pantalla y por el momento sólo tiene una instrucción [prgend] (fin del programa).

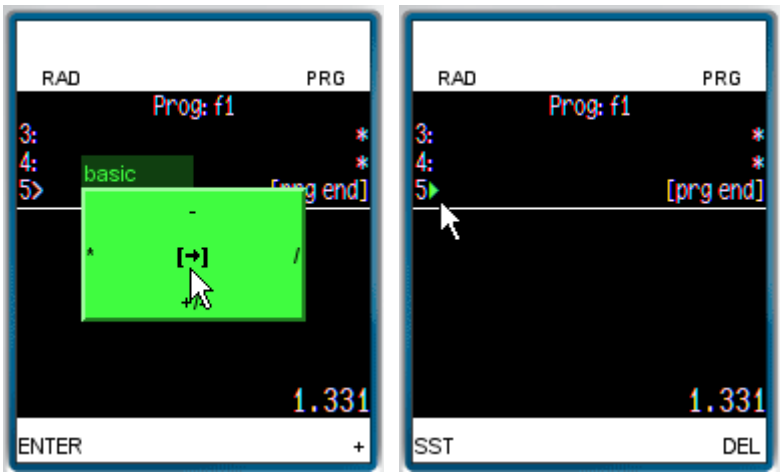
Ahora elaboramos el programa y como ya se describió en la lógica, duplicamos dos veces el valor de “x” (1.1), pulsando “ENTER”:

Como se puede observar, las dos instrucciones (ENTER) quedan registradas como pasos del programa en el monitor y en la pila tenemos ya los tres valores de “x” que debemos multiplicar. Ahora entonces multiplicamos dos veces:



Como se puede observar en el monitor, las dos multiplicaciones se registran también como pasos del programa (los pasos 3 y 4):

Con ello el programa estaría concluido. Podemos revisar los pasos que hemos programado pasando a la ventana del monitor (basic/[->]):

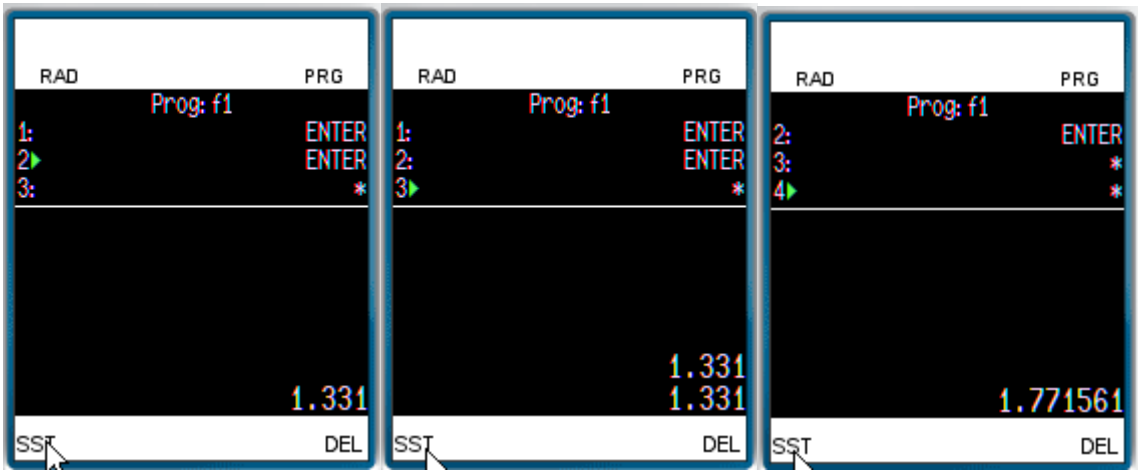


Como se puede observar, la marca visual que nos indica que estamos en la ventana del monitor es el triángulo verde que aparece en el mismo. Para recorrer el programa en el monitor, se emplean las teclas de navegación arriba, abajo, izquierda (para recorrer una página) y derecha (para avanzar una página). A medida que recorremos el programa en el monitor, el marcador verde nos va indicando en qué paso del programa nos encontramos, así en las tres siguientes pantallas nos encontramos en los pasos 1, 3 y 4 respectivamente.

Observe también que los botones que normalmente corresponden a las teclas “ENTER” y “+”, ahora han cambiado a “SST” y “DEL” respectivamente.

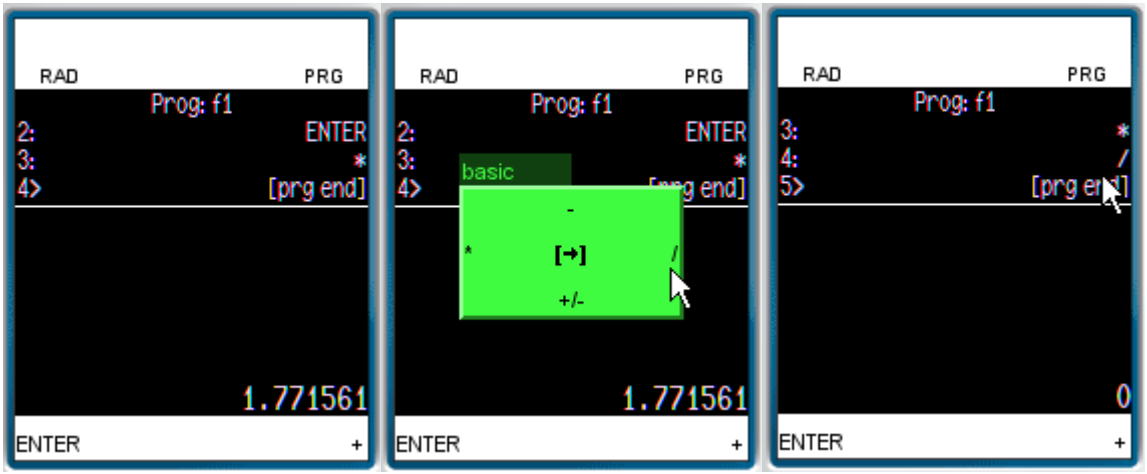
“SST” ejecuta el paso en el que nos encontramos y pasa al siguiente. “DEL” borra el paso en el que nos encontramos.

Por ejemplo, si vamos al segundo paso del programa y ejecutamos esa orden (ENTER), el valor de la pila (el resultado del programa) se duplica. Si luego ejecutamos la siguiente orden (la multiplicación), los dos valores de la pila (el resultado del programa y su copia) se multiplican:



Si hemos cometido un error al elaborar el programar. Supongamos por ejemplo que en lugar de la segunda multiplicación debería ir una división. Para corregir el error primero borramos la instrucción errónea (vamos al paso correspondiente, en este caso el número 4 y pulsamos la tecla “DEL”):

Luego salimos del monitor (pulsando el botón de navegación central) y ejecutamos la instrucción correcta (basic//):

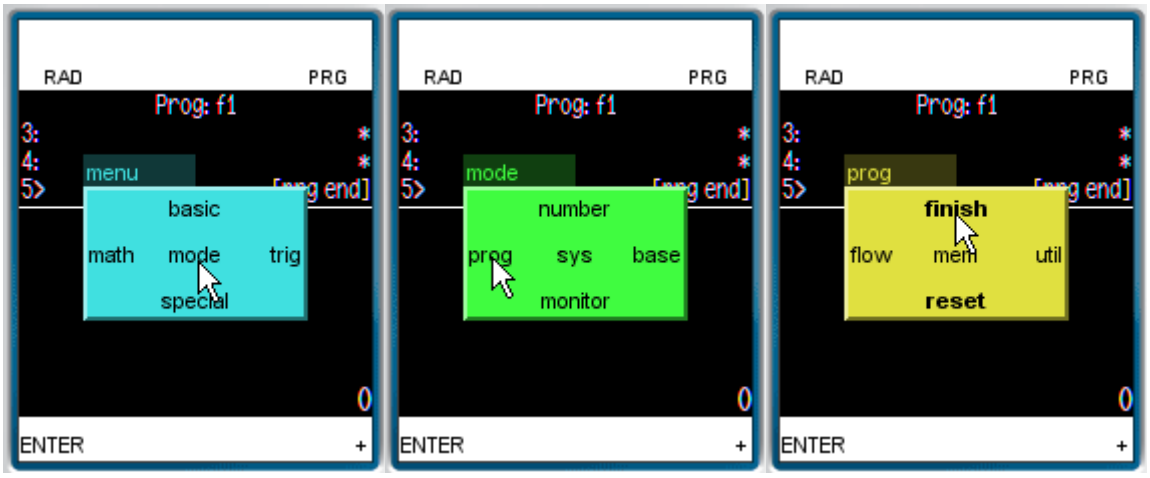


Y como se puede ver en el paso 4 se inserta la operación de división. Por supuesto que en este caso no existía el error, por lo que debemos corregir

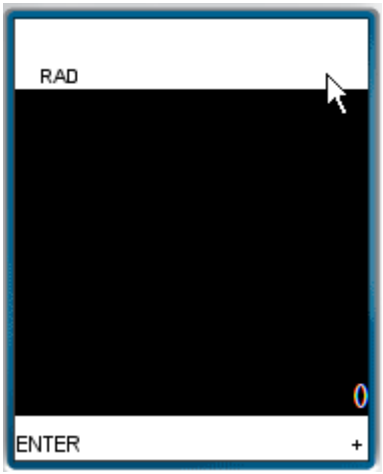
una vez más el programa borrando la operación de división reemplazándola por la de multiplicación:

Con lo que el programa vuelve a su estado anterior.

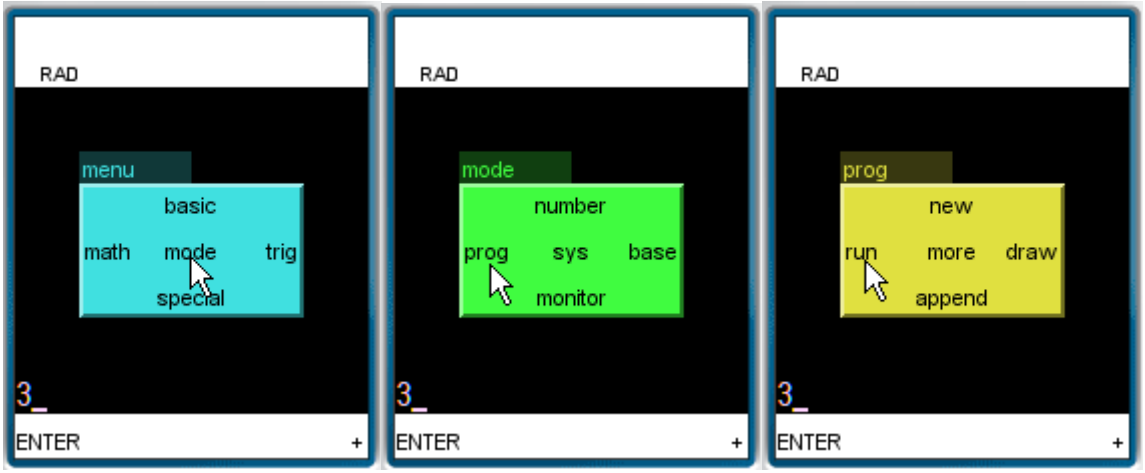
Finalmente salimos del entorno de programación:

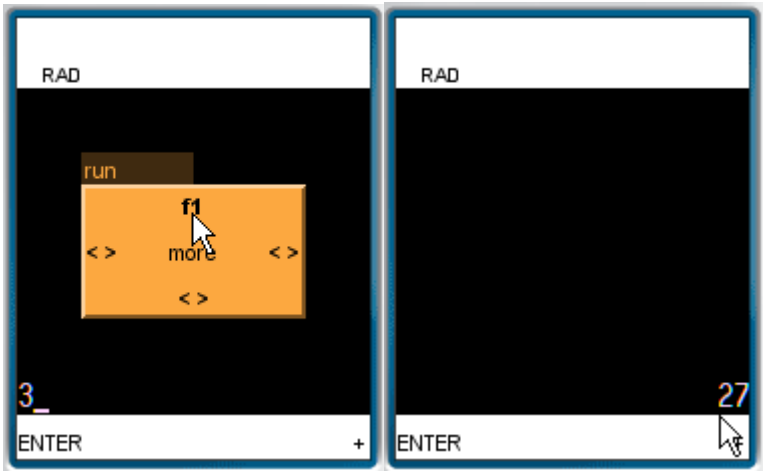


Con lo que volvemos al modo normal de trabajo, que básicamente difiere del entorno de programación porque desaparece “PRG” del menú y porque ya no son visibles las líneas del monitor.



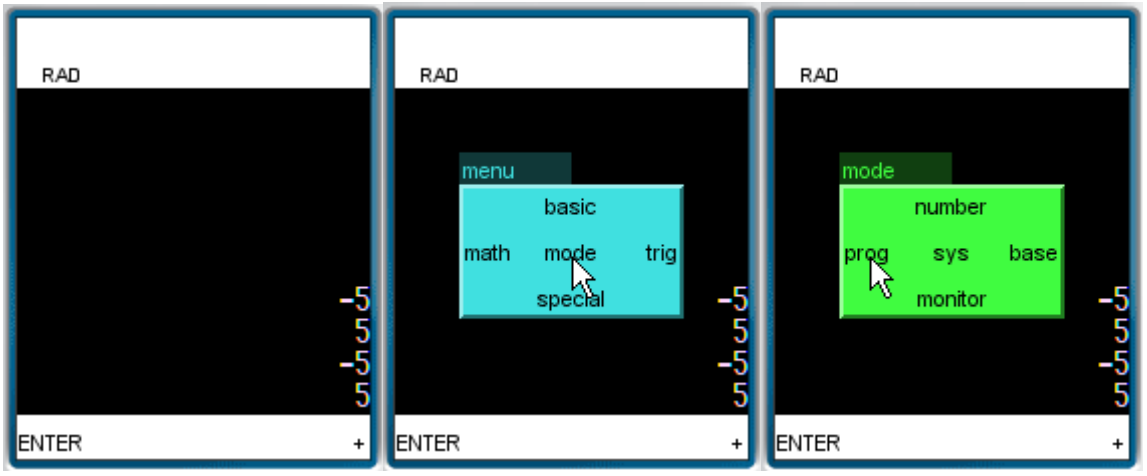
Ahora podemos comenzar a utilizar el programa elaborado. Comencemos por calcular el valor de la función para x=3 (f(3)):

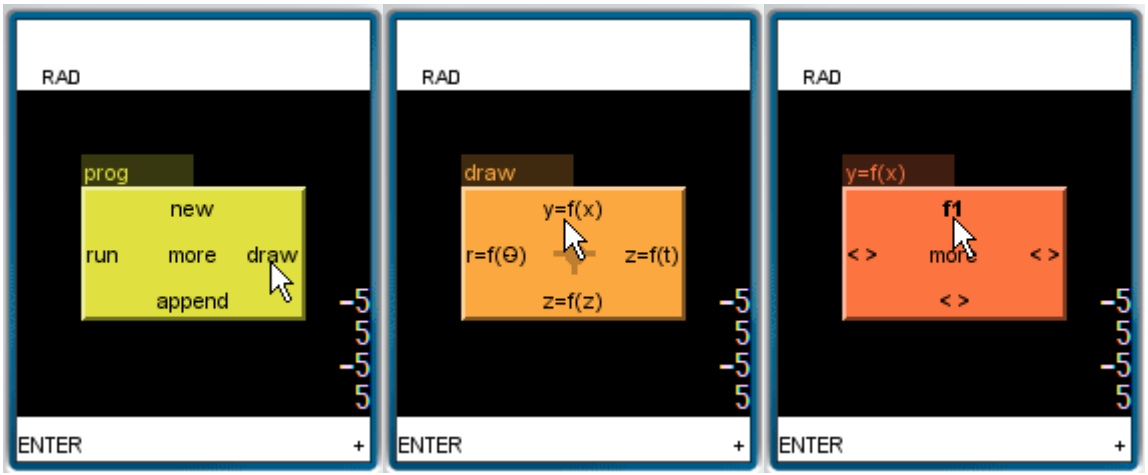


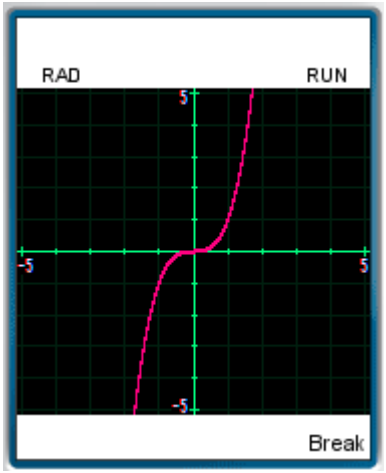


Y como se ve se obtiene 27, que es el resultado correcto. Igualmente haciendo correr el programa para x=2.1 (2.1/mode/prog/run/f1) se obtiene 9.261, que una vez más es el resultado correcto, por lo que podemos asumir que el programa está bien elaborado. El probar un programa con valores cuyos resultados son conocidos, es algo que se debe hacer siempre para verificar que las operaciones hayan sido correctamente programadas.

Ahora que la función está programada podemos también graficarla. Para ello primero escribimos los límites entre los cuales queremos que se grafique la función, en el orden: x mínimo, x máximo, y mínimo, y máximo. Así para graficar la función entre x=-5, x=5, y=-5 y y=5, hacemos lo siguiente:

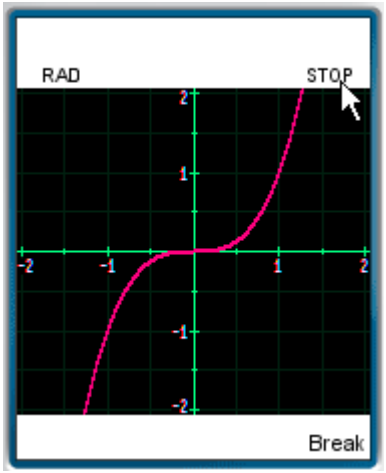




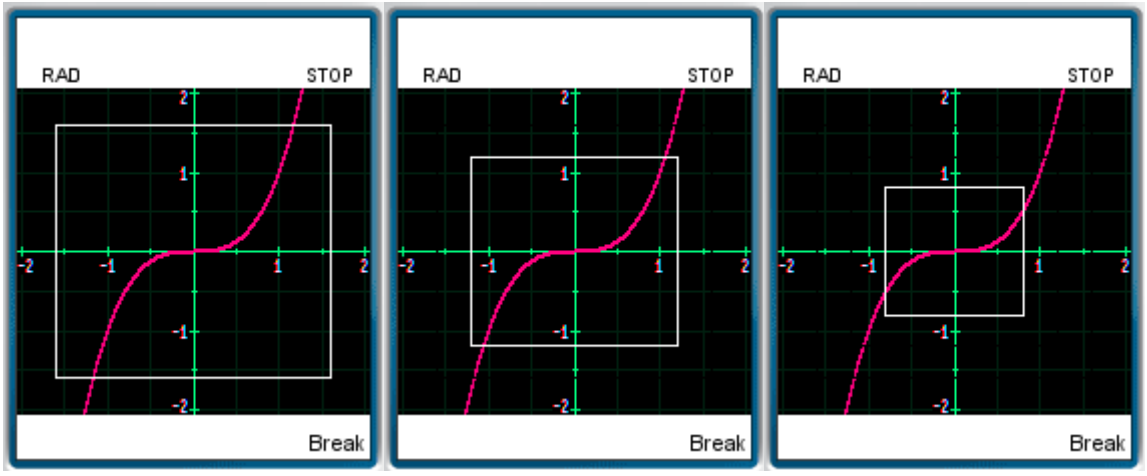


Como puede apreciar en el menú “draw”, Calc-Java permite realizar cuatro tipos de gráfica: rectangulares: y=f(x), polares: r=f(θ), de frecuencia f(t) y paramétricas z=f(z)). En nuestro caso, hemos elegido la gráfica rectangular, porque eso es lo que hemos programado (una función de x: y=f(x)).

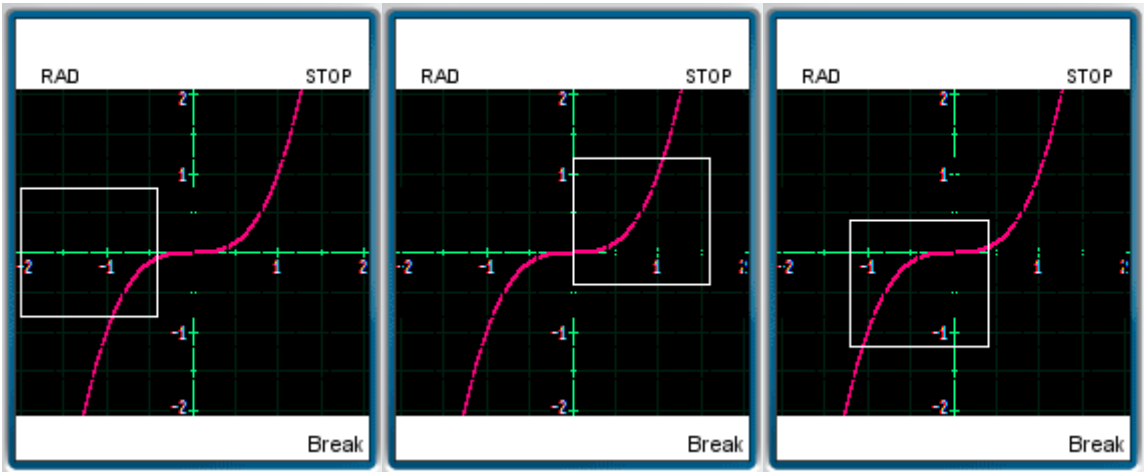
Observe también que en la gráfica resultante, mientras se muestra la gráfica, parpadea en el menú la palabra “RUN”, esa es la forma en la que Calc-Java nos avisa que sigue realizando cálculos. Lo que ocurre es que (aun cuando normalmente no se ve) Calc-Java sigue calculando puntos intermedios para mejorar la calidad de la gráfica y continúa con esa labor mientras la gráfica es visible. Puesto que ello consume la batería del celular, es conveniente detener los cálculos una vez que la gráfica tiene una calidad aceptable (lo que normalmente ocurre en menos de 2 segundos). Para ello se debe pulsar la tecla “*”, entonces en el menú aparece la palabra “STOP” en lugar de “RUN”:



Una vez que los cálculos han sido detenidos es posible ampliar o reducir la gráfica empleando el botón central del navegador para ampliar la gráfica y la tecla 1 para reducirla. Cuando se presiona el botón central del navegador aparece un recuadro blanco que puede ser reducido más aún pulsando otra vez pulsando repetidas veces el botón central del navegador:

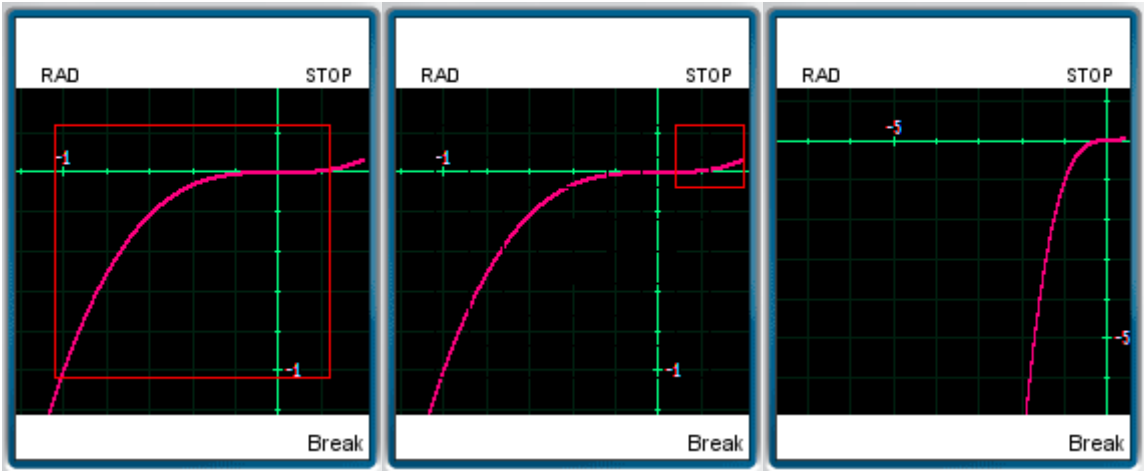


Este recuadro puede ser movido a través de la pantalla con las teclas de navegación (arriba, abajo, izquierda, derecha), hasta el lugar que se quiere ampliar. Mientras se mueve el recuadro la gráfica puede quedar con partes borrosas, tal como se muestra en las siguientes figuras, sin embargo, ello no tiene mayor importancia, pues como veremos, al ampliar la gráfica vuelve a ser dibujada.



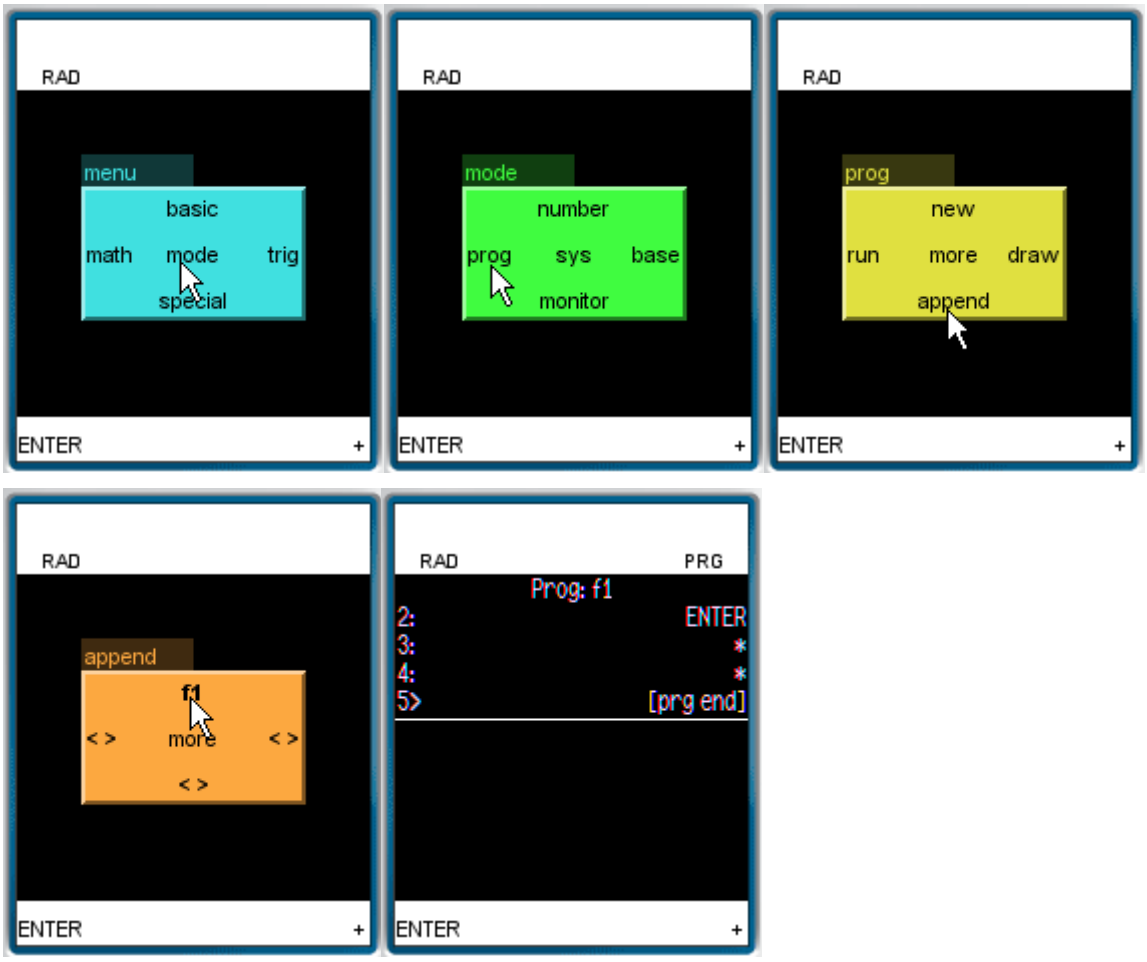
Para ampliar la selección, simplemente se vuelve a pulsar la tecla "*", con ello se vuelven a calcular puntos, por lo que vuelve a aparecer la palabra "RUN" en el menú. Al igual que el caso anterior, para evitar el desgaste de la batería, se debe detener los cálculos pulsando una vez más la tecla "*":

Para ampliar se sigue el mismo procedimiento, sólo que ahora la tecla a pulsar es la número 1 y el recuadro que es de color rojo en lugar de blanco:



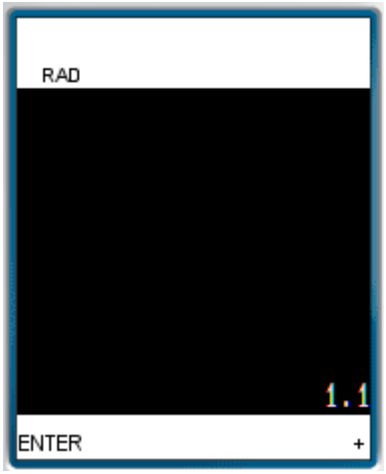
Una vez que se termina de trabajar con la gráfica se sale de la misma pulsando el botón correspondiente a la palabra "Break".

De esa manera hemos empleado la función programada tanto para calcular valores puntuales como para graficarla. Si en algún momento se quiere modificar la función (añadir código o corregir algún error), es puede editar el programa eligiendo la opción “append” en lugar de “run”:

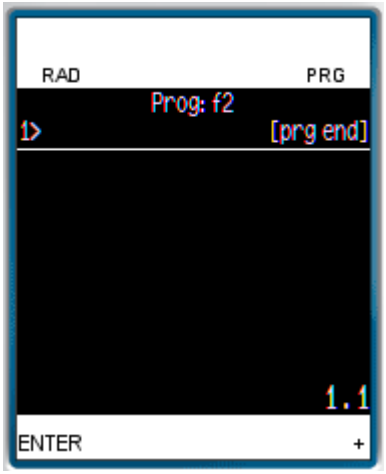


Una vez modificado (o corregido) el programa se sale del modo habitual (mode/prog/finish).

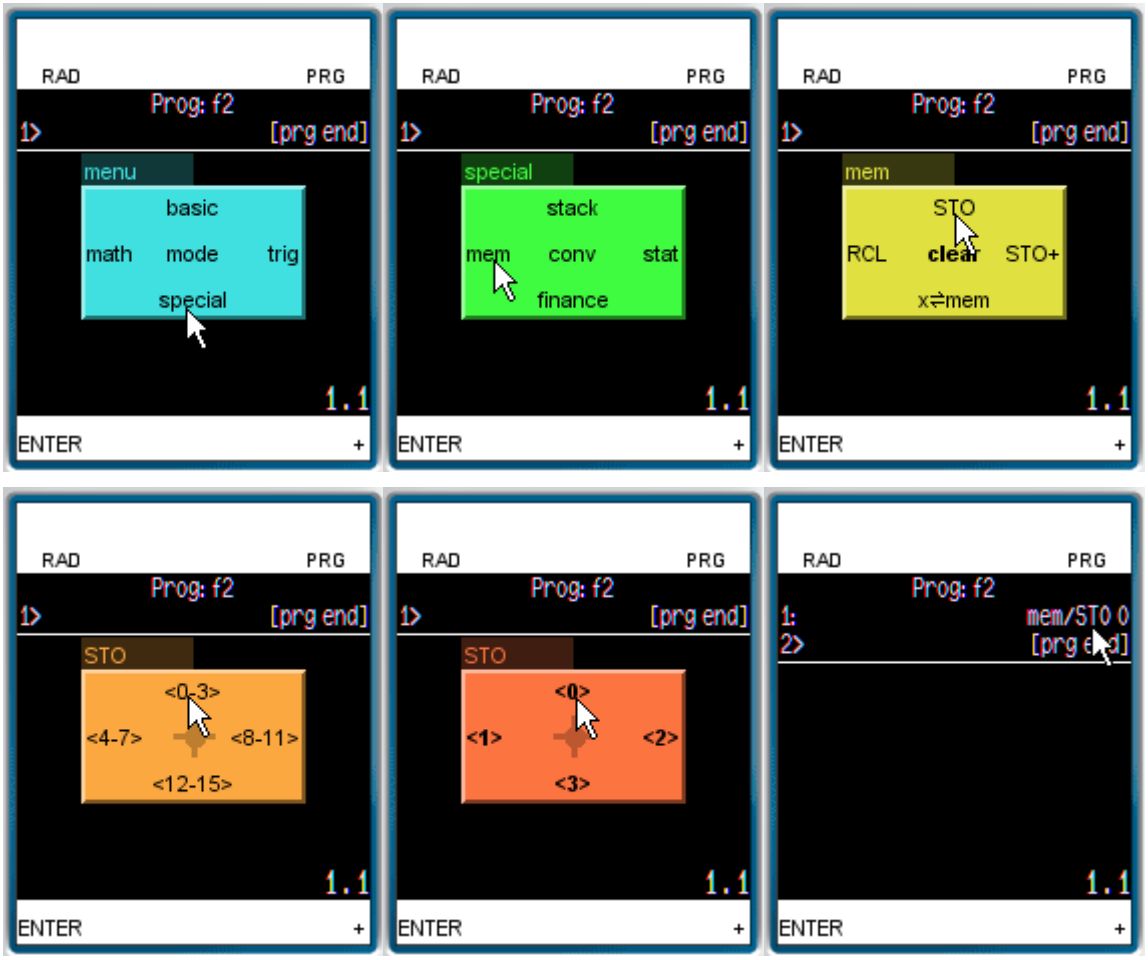
Ahora elaboremos el mismo programa pero empleando posiciones de memoria en lugar de la pila. Al igual que el caso anterior, primero escribimos un valor de prueba en la pila:



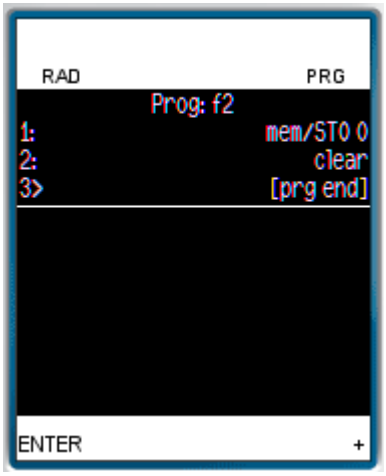
Luego creamos un nuevo programa llamado “f2” (mode/prog/new/<>/f2/OK):



Ahora guardamos ese valor (el valor de “x”) en una posición de memoria (en nuestro caso la memoria 0):



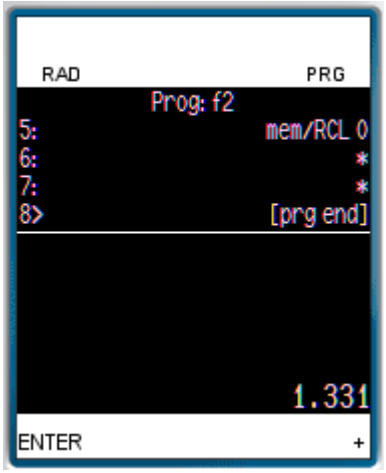
Con ello, como se puede observar aparece la instrucción “mem/STO 0” como el primer paso del programa. Ahora que ya tenemos el valor de “x” en memoria, podemos borrar el valor de la pila (con el botón de borrado), con lo que se añade la instrucción “clear” como un nuevo paso del programa:



Ahora simplemente recuperamos el valor de “x” (que acabamos de guardar en la posición 0):

Y se repite dos veces más el proceso (para recuperar los otros dos valores de “x”): special/mem/RCL/<0-3>/0; special/mem/RCL/<0-3>:

Y ahora simplemente se multiplican dos veces estos valores para obtener el resultado buscado (el cubo de x):



Con ello el programa estaría concluido y como se puede apreciar resulta un tanto más largo que la versión anterior. Esto sin embargo no es una regla, en ocasiones puede resultar más corto y sobre todo más claro trabajar con posiciones de memoria, en otros con la pila y en otros combinar ambos formas para.

Ahora salimos del programa (mode/prog/finish) y probamos el mismo con 3 (4/mode/prog/run/f2) y 2.1 (2.1/mode/prog/run/f2):

Y como era de esperar se obtienen los mismos resultados que con el programa anterior.

Ahora que ya sabemos crear un programa, elaboraremos un programa para una función un tanto más elaborada:

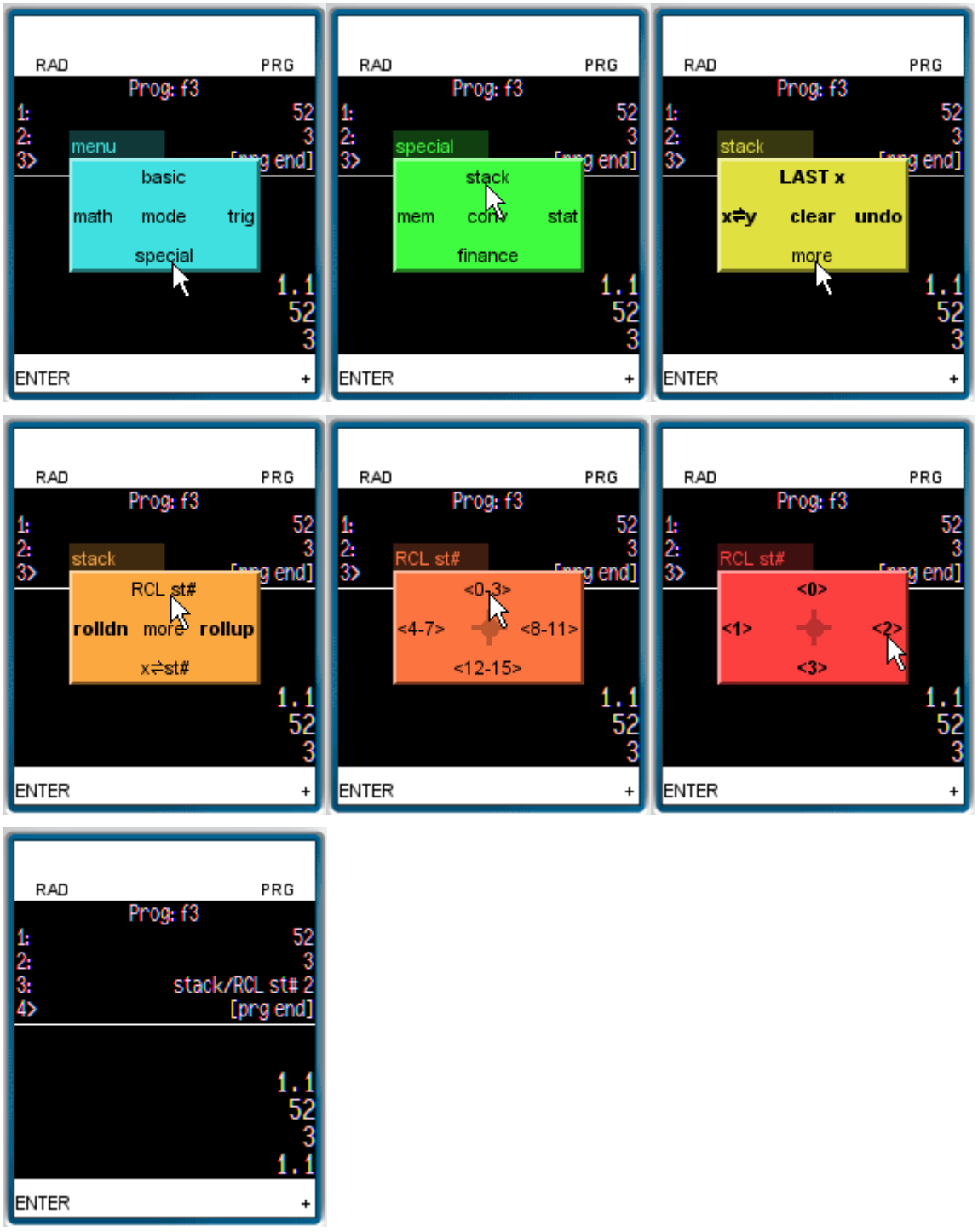
$$f(x) = \left(52 + 3\sqrt{x} - 8x^{0.8}\right)^{0.36} - x = 0$$

Esta es la forma en que generalmente se presentan las funciones en métodos numéricos. Lo que esta función nos informa es que la solución de la misma (el valor de "x" que resuelve esta ecuación) se encuentra cuando al reemplazar un valor de "x" en la función el resultado es cero (o casi cero).

Programemos la función empleando primero la pila. Como siempre es necesario colocar primero un valor de prueba en la pila (un valor cualquiera de "x"), para poder realizar las operaciones con el mismo. Una vez más el valor de prueba será 1.1 y el programa tendrá el nombre "f3" (mode\prog\new\<>\f3\OK):

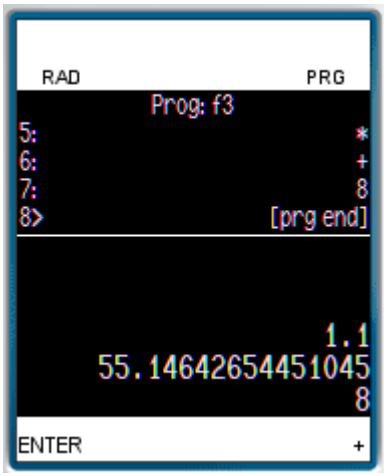
Comenzamos a programar la función escribiendo los números 52 y 3:

Ahora debemos calcular la raíz cuadrada de "x", pero no podemos emplear el valor de "x" que tenemos en la pila (que está en la posición 2), porque luego necesitamos el mismo dos veces más (para calcular $x^{0.8}$ y restar x), así es que debemos hacer una copia de dicho valor:

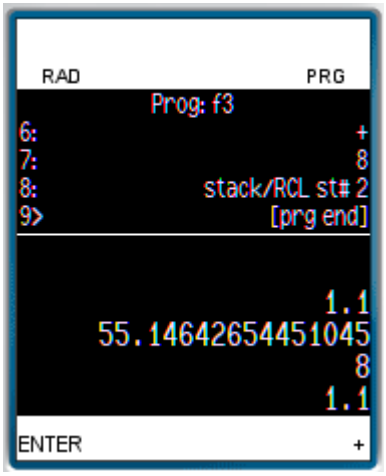


Ahora calculamos la raíz cuadrada de este valor (math/simple/√x), multiplicamos por 3 (que ya está en la pila: basic/*) y sumamos a 5 (que también ya está en la pila: +):

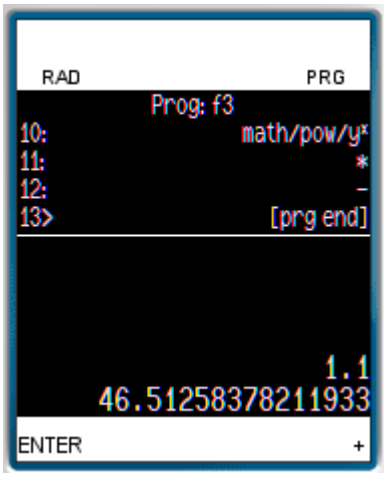
Ahora escribimos el número 8 (8/ENTER) (observe que todos los pasos se van registrando en el monitor):



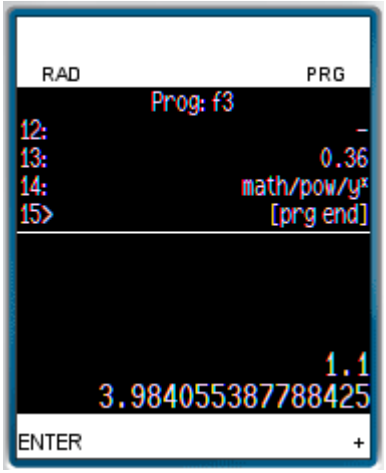
Ahora debemos calcular el valor de "$x^{0.8}$", por lo que una vez más necesitamos una copia del valor de "x", que casualmente se encuentra otra vez en la posición 2. Debe tomar en cuenta que la primera posición es la número 0 y que las posiciones relativa en la pila cambian constantemente, por lo que es sólo una casualidad que otra vez el valor de "x" (1.1) se encuentre en la misma posición. Hacemos una copia del valor de "x" (special/stack/more/RCL st#/<0-3>/2):



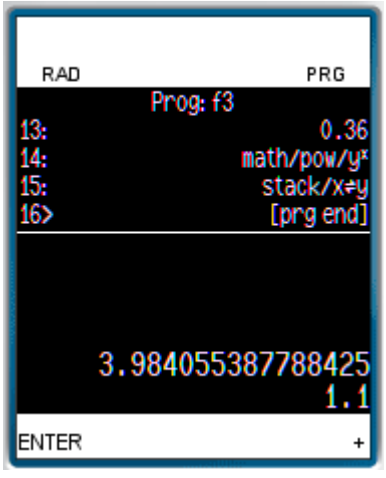
Elevamos ese valor a 0.8 (0.8/math/pow/y^x), multiplicamos por el número 8 (que ya está en la pila: basic/*) y restamos al resultado anterior (basic/-):



Ahora elevamos este resultado a 0.36 (0.36/math/pow/y^x):



Y finalmente restamos a este resultado el valor de “x”. Puesto que en este caso no necesitamos más el valor de “x”, utilizamos directamente el valor que tenemos en la pila. Para ello simplemente intercambiamos los dos valores que tenemos en pila (special/stack/x<->y):



Finalmente restamos estos dos valores:

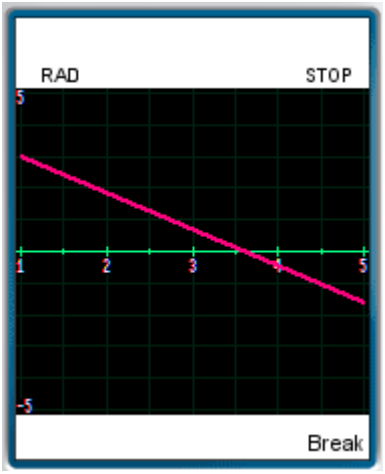
Ahora el programa ya está concluido, por lo que salimos del entorno de programación (mode/prog/finish) y probamos el programa calculando el valor de la función para x=2.3 (f(2.3)) (2.3/mode/prog/run/f3):

Si nuestro objetivo es encontrar la solución, es decir, encontrar el valor de “x” para el cual el resultado es cero, podemos emplear el programa probando con diferentes valores hasta que el resultado sea cero (o casi cero).

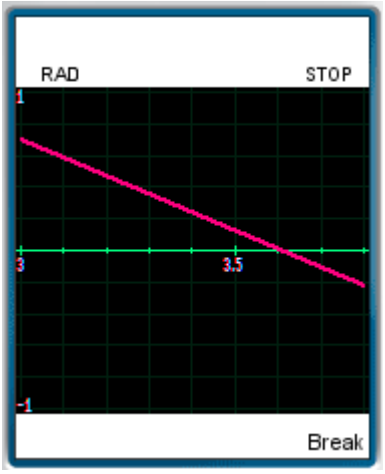
Así para “x=3.1” (3.1/mode/prog/run/f3) obtenemos: 0.5869622889544368 y para “x=4.2” (4.2/mode/prog/run/f3) -0.6816447963914534. Por lo tanto, dado que existe un cambio de signo entre estos dos valores, el resultado se debe encontrar entre 3.1 y 4.2. Entonces probamos con un valor intermedio, digamos 3.5 (3.5/mode/prog/run/f3) siendo el resultado: 0.1264105016382076, que ya está más cerca de cero, pero que todavía no es cero, probando con otro valor, digamos 3.6 (3.6/mode/prog/run/f3) obtenemos: 0.011145959138949926, que está aún más cerca de 0, con 3.7 obtenemos: -0.1041656362321059, que cambia de signo, por lo que la solución se debe encontrar entre 3.6 y 3.7, probando con 3.65 obtenemos: -0.04650107434619145, por lo que el valor debe ser un tanto menor.

Continuando de esta manera podemos seguir probando otros valores hasta que el resultado sea cero o cercano a cero. A este método se conoce con el nombre de “simple tanteo” y como se puede observar se trata de un método intuitivo que puede requerir muchas iteraciones (pruebas) antes de que se encuentre un resultado con una precisión aceptable.

Una mejor alternativa, consiste en graficar la función. Gráficamente la solución de una función se encuentra en la intersección entre la curva y el eje x. Por ejemplo graficando la función “f3” entre 1 y 5 (1/ENTER; 5/ENTER; -5/ENTER; 5/ENTER; mode/prog/draw/y=f(x)/f3) obtenemos:



Donde podemos ver que la solución se encuentra cerca de “x=3.5”, amplian-do este sector:

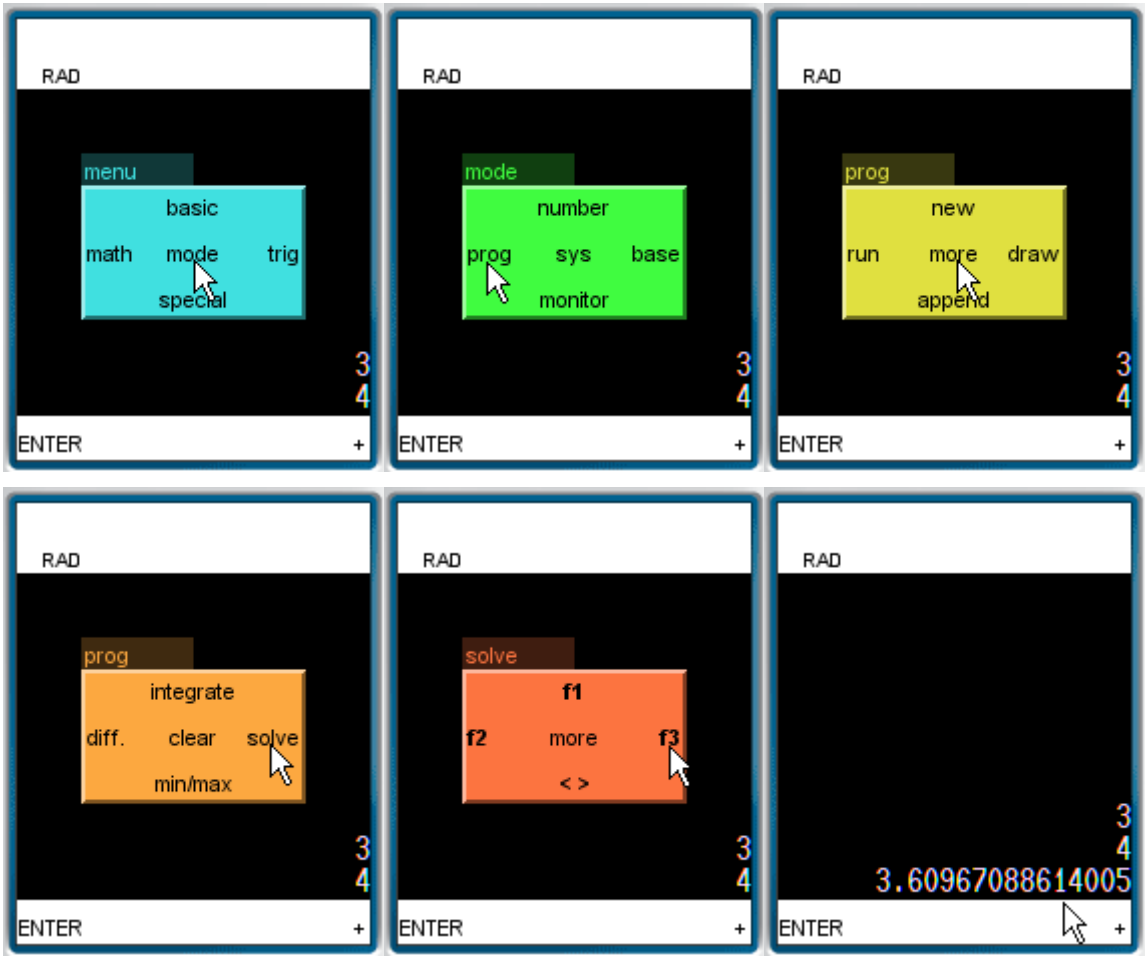


Vemos que la solución está muy cerca a 3.6, por lo que podríamos concluir que esta es la solución aproximada de la función.

Para encontrar una solución más exacta, podemos recurrir al solucionador (solve) de Calc-Java.

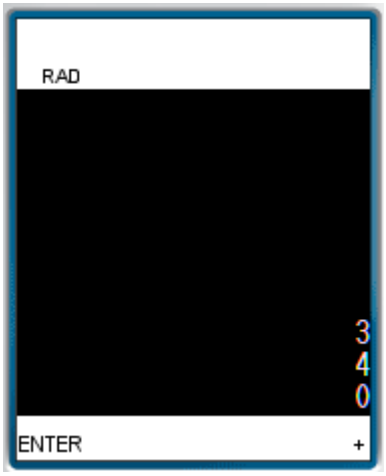
Para ello escribimos en la pila dos valores de “x” que sabemos se encuen-tran a ambos lados de la solución, por ejemplo para esta ecuación dos valo-res adecuados pueden ser 3 y 4 (3/ENTER; 4/ENTER), porque como vimos en la gráfica, para 3 el valor de la función es positivo y para 4 negativo.

Ahora simplemente vamos a la opción “solve” y Calc-Java se encarga de en-contrar el resultado más exacto:



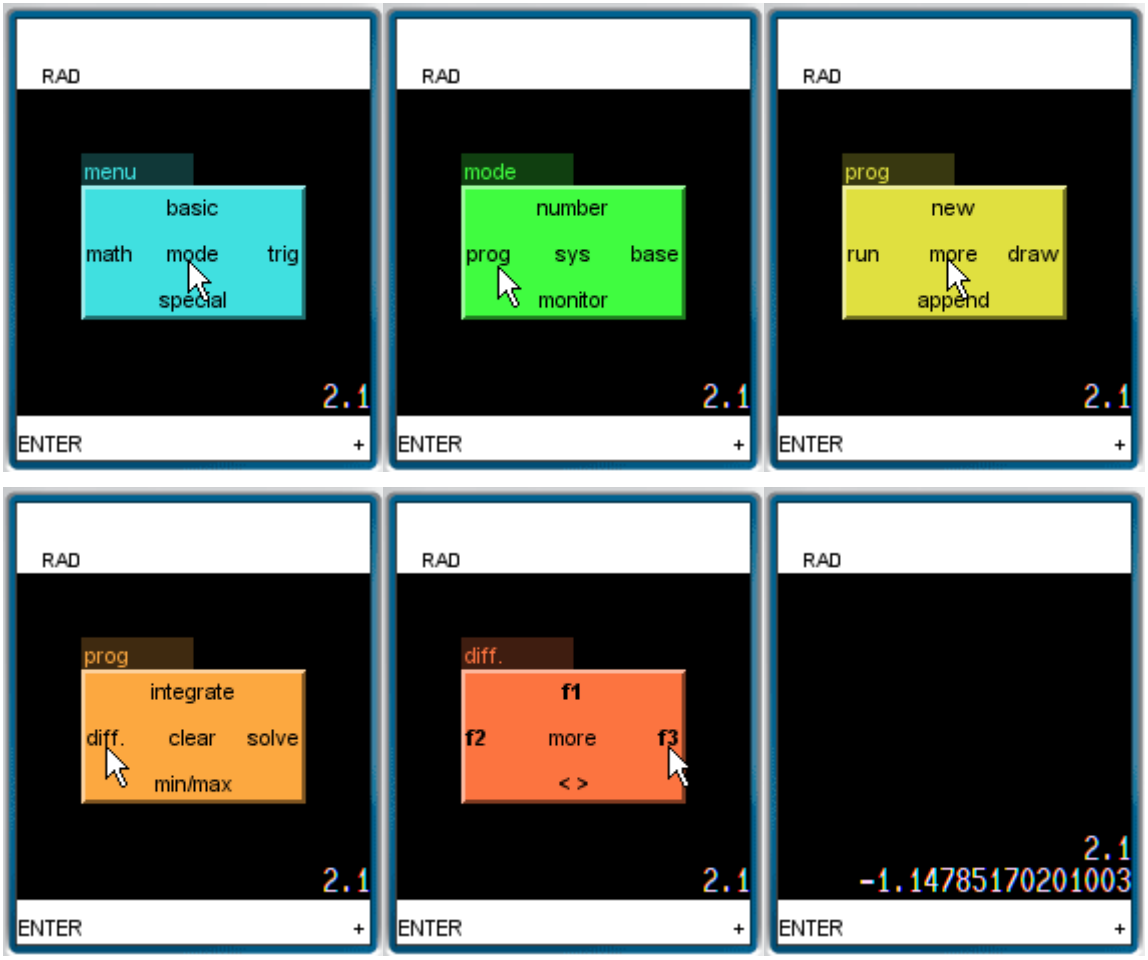
Vemos entonces que la solución más exacta es 3.60967088614005.

Podemos comprobar la exactitud de esta solución calculando el valor de la función con este resultado (mode/prog/run/f3):



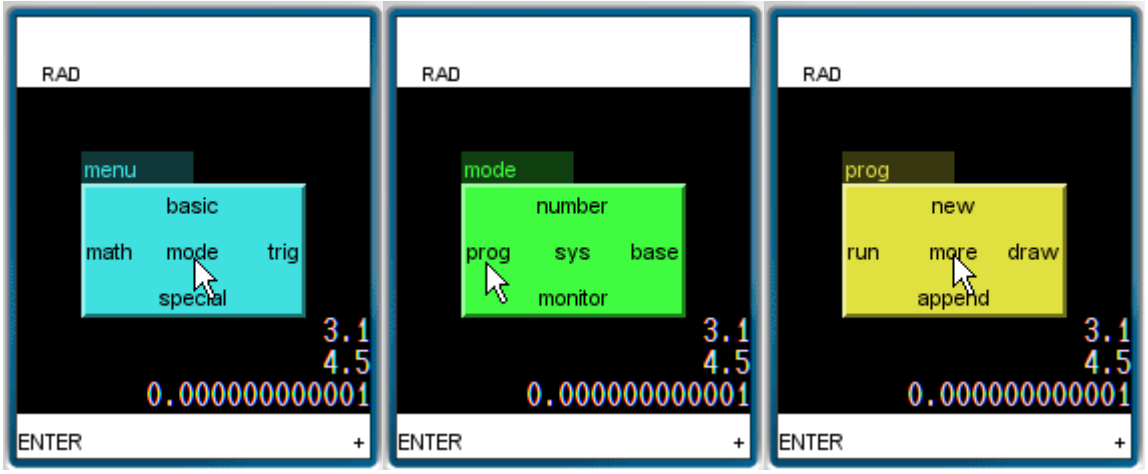
Como se puede observar, la solución encontrada es exacta, pues al reemplazarla en la función el resultado es cero.

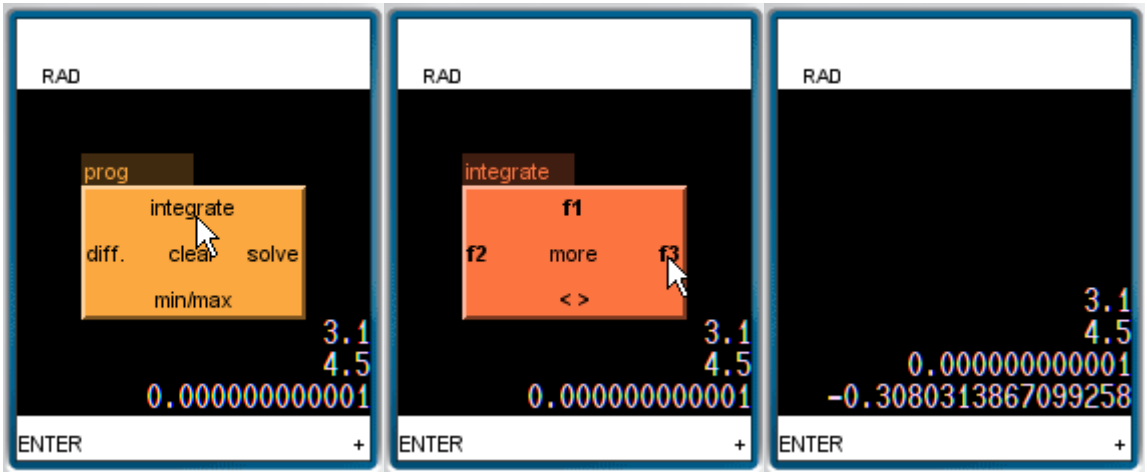
Pero con Calc-Java, no solo podemos encontrar la solución, sino también derivar la función en un punto dado, por ejemplo, para hallar la derivada de la función para “x=2.1” (f’(2.1)), escribimos el número (2.1/ENTER) y elegimos la opción “diff”:



Por consiguiente la derivada de la función en 2.1 es: f'(2.1)= -1.14785170201003.

Con Calc-Java también podemos integrar la función, así por ejemplo para integrar la función entre 3.1 y 4.5, calculando el valor de la integral con 12 dígitos de precisión escribimos: 3.1/ENTER; 4.5/ENTER; 1e-12/ENTER y luego elegimos la opción integrate:



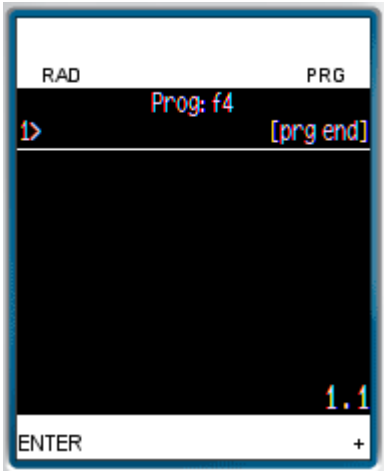


En consecuencia, la integral de la función entre 3.1 y 4.5 (con 12 dígitos de precisión) es: -0.308031867099258.

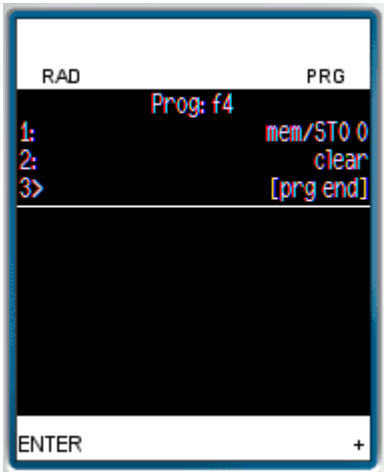
Los 17 pasos de programación de los que consta el programa elaborado son los siguientes:

```
1                    52
2                     3
3       stack/RCL st#2
4                    √x
5                     *
6                     +
7                     8
8       stack/RCL st# 2
9                   0.8
10          math/pow/y^x
11                    *
12                    -
13                 0.36
14          math/pow/yx
15          stack/x<=>y
16                    -
17            [prg end]
```

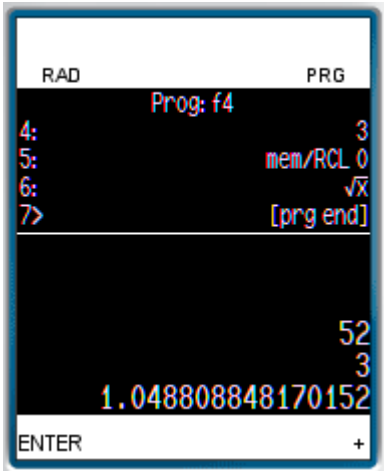
Ahora, para ver la otra alternativa de programación: con posiciones de memoria, volveremos a elaborar el programa, que ahora se llamará “f4” (1.1/ENTER; mode/prog/new/<>/f4/OK):



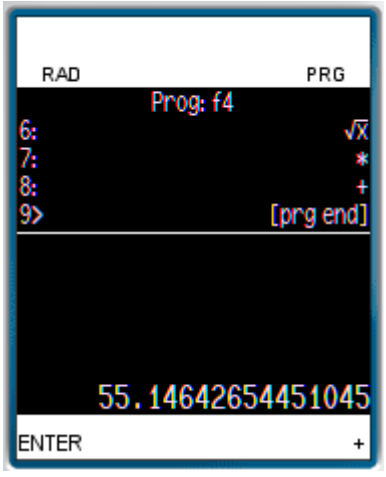
Comenzamos guardando el valor de “x” (1.1) en la posición 0 de la memoria (special/mem/STO/<0-3>/0) y borrando el dato de la pila (clear):



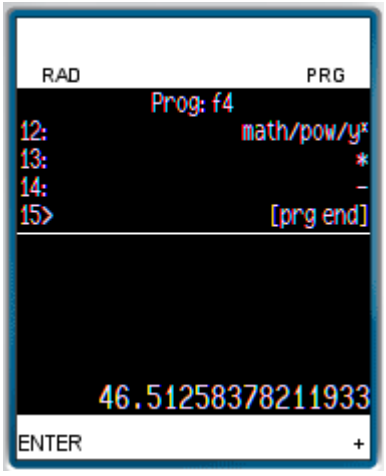
Ahora, y al igual que en el anterior programa colocamos en la pila los números 52 y 3 (52/ENTER; 3/ENTER), recuperamos el valor de “x” (special/mem/RCL/<0-3>/0) y calculamos la raíz cuadrada del mismo (math/simple/ √x):



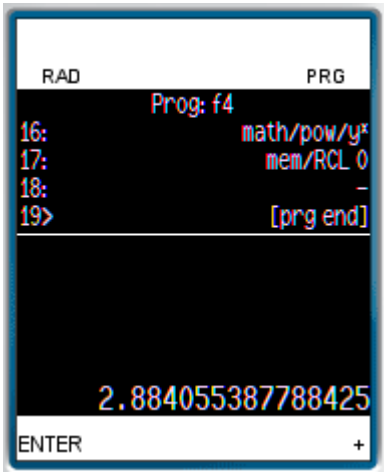
Entonces multiplicamos este valor por 3 (basic/*) y sumamos el resultado a 52 (+):



Ahora colocamos el número 8 en la pila (8/ENTER), recuperamos el valor de "x" (especial/mem/RCL/<0-3>/0), elevamos el mismo a 0.8 (0.8/math/pow/y^x), multiplicamos por el número 8 (basic/*) y restamos este resultado al resultado anterior (basic/-):



Finalmente, elevamos este resultado a 0.36 (0.36/math/pow/y^x) y le restamos el valor de "x" (special/mem/RCL/<0-3>/0; basic/-):



Con lo que el programa estaría concluido.

Ahora salimos del entorno de programación (mode/prog/finish) y calculamos el valor de la función para "x=2.3" (2.3/mode/prog/run/f4):

Y como era de esperar se obtiene el mismo resultado que con el programa anterior. Los 19 pasos de los que consta el programa son:

```
1              mem/STO 0
2                  clear
3                     52
4                      3
5              mem/RCL 0
6                     √x
7                      *
8                      +
9                      8
10             mem/RCL 0
11                   0.8
12           math/pow/y^x
13                     *
14                     –
15                  0.36
16           math/pow/yx
17             Mem/RCL/0
18                     -
19             [prg end]
```

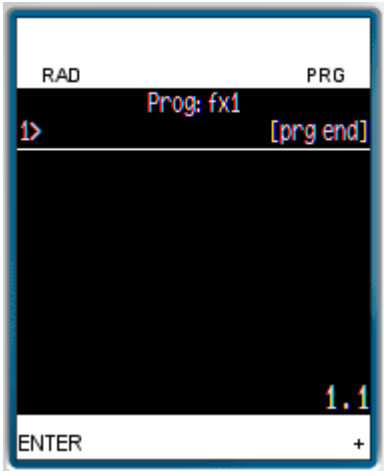
Como ya se dijo, la forma de programación que se elija (guardando las variables en la pila o en posiciones de memoria), depende sobre todo de cuán claro le resulte al programador emplear una u otra opción.

A continuación se presentan algunos otros ejemplos.

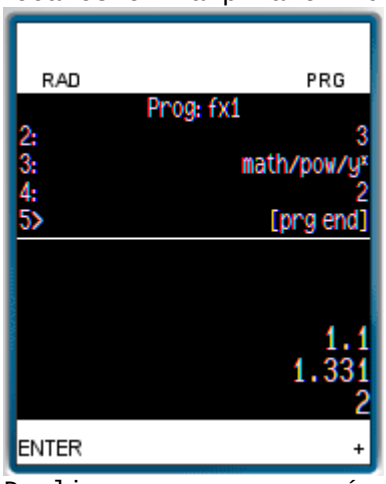
2.1.1. Ejemplos

1. Programe la función: $f(x)=x^3+2x^2+3x+4=0$. Luego encuentre los valores de la función para x= -3.1, -2.1, -1.1, 0.1, 1.1 y 2.1. Grafique la función entre “x=-5” y “x=0” (calculando los valores adecuados para “y”). En base a la gráfica estime la solución aproximada de la función. Calcule luego una solución más exacta con “solve”. Derive la función para “x=-1.2” e integre la función entre -1.7 y 0, con 10 dígitos de precisión.

Programamos entonces la función. Para ello colocamos un valor de prueba en la pila (1.1/ENTER), entramos al entorno de programación (mode/prog/new/<>/fx1/OK) y si no está visible, activamos el monitor (mode/monitor/prog/<4-7>/4):

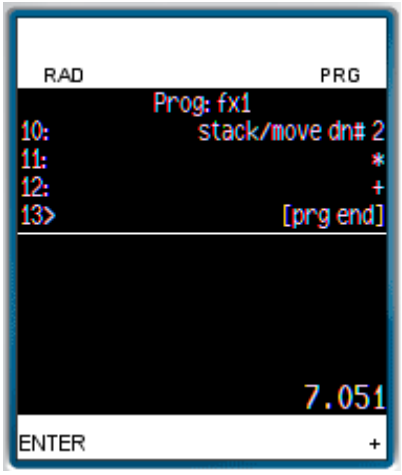


En esta versión emplearemos la pila, entonces para calcular el cubo de "x", hacemos una copia (ENTER) y elevamos la copia al cubo (3/math/pow/y^x) Y colocamos en la pila el número 2 (2/ENTER):

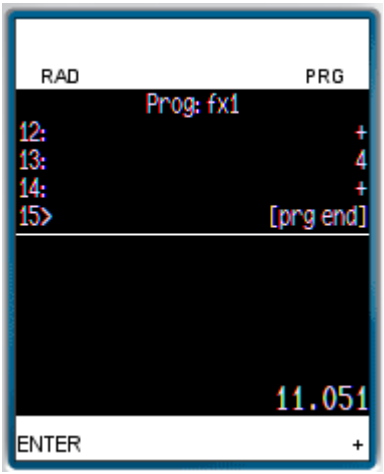


Duplicamos una vez más el valor de "x" (special/stack/more/RCL st#/<0-3>/2), elevemos ese valor al cuadrado (math/simple/x^2), multiplicamos por el número 2 que está en la pila (basic/*) y sumamos al resultado anterior (+):

Colocamos el número 3 en la pila (3/ENTER), movemos el valor de "x" a la primera posición de la pila (special/stack/more/more/move dn#/<0-3>/2), multiplicamos este valor por el número 3 de la pila (basic/*) y sumamos el resultado al resultado anterior (+):



Finalmente sumamos a este resultado el número 4:



Los pasos de este programa son:

1	ENTER
2	3
3	math/pow/y^x
4	2
5	Stack/RCL st#2
6	x^2
7	*
8	+
9	3
10	stack/move dn# 2
11	*
12	+
13	4
14	+
15	[prg end]

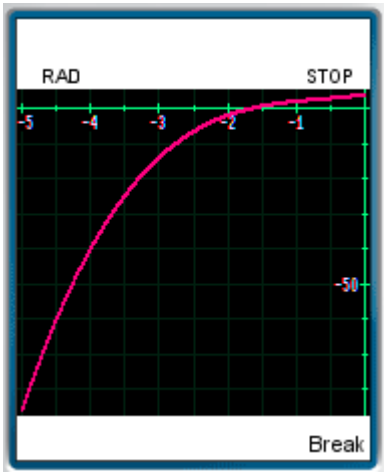
Cuando se trabaja con posiciones de memoria, los pasos del programa son:

1	mem/STO 0
2	Clear
3	mem/RCL 0
4	3
5	math/pow/y^x
6	2
7	mem/RCL 0
8	x^2
9	*
10	+
11	3
12	mem/RCL 0
13	*
14	+
15	4
16	+
17	[prg end]

Por supuesto, ambos programas devuelven los mismos resultados.

Para calcular los valores de la función en los puntos dados simplemente hacemos correr el programa con dichos valores, así para x= -3.1 (-3.1/mode/prog/run/fx1) obtenemos: -15.871. De la misma forma para los otros valores de “x” se obtiene: f(-2.1)=-2.741, f(-1.1)=1.789,f(0.1)=4.321, f(1.1)=11.051 y f(2.1)=28.381.

Para graficar la función entre x=-5 y x=0, escribimos primero esos dos límites (-5/ENTER; 0/ENTER), luego calculamos los límites para "y" con estos mismos valores de "x": -5/mode/prog/run/fx1; 0/mode/prog/run/fx1. Una vez que tenemos los cuatro límites elaboramos la gráfica (mode/prog/draw/y=f(x)/fx1):



Como se puede apreciar en la gráfica, la solución se encuentra entre -2 y -1, siendo un valor aproximado de la misma -1.5.

Empleando como límites: -2 y -1, calculamos una solución más exacta con solve: -2/ENTER, -1/ENTER; mode/prog/more/solve/fx1, siendo el resultado más exacto: -1.650629191439388.

Para calcular el valor de la derivada en "x=-1.2" seguimos los pasos: -1.2/mode/prog/more/diff/fx1, con lo que obtenemos: 2.519999999999527.

Por último, el valor de la integral se obtiene con los siguientes pasos: -1.7/ENTER; 0/ENTER; 1e-10/mode/prog/more/integrate/fx1, con lo que se obtiene el resultado: 3.652308333333333.

2. Programe la función: $f(x) = \cos(x) + (1+x^2)^{-1} = 0$. Luego encuentre los valores de la función para x= 1.5. Grafique la función entre "x=0->2π" y "y=-1->2". En base a la gráfica estime la solución aproximada de la función. Calcule luego una solución más exacta con "solve". Derive la función para "x=2.2" e integre la función entre 5 y 7.5, con 14 dígitos de precisión.

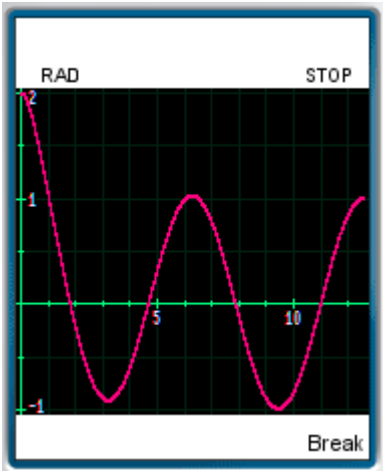
El programa para esta función, trabajando en la pila es el siguiente: 1.1/mode/prog/new/<>/fx5:

```
1                  ENTER
2               trig/cos
3                     1
4      stack/move dn# 2
5                    x²
6                     +
7                   1/x
8                     +
9             [prg end]
```

Haciendo correr el programa con 1.5, "f(1.5)", se obtiene:

1.5/mode/prog/run/fx5 => 0.3784295093600106

Graficamos la función entre 0, 2π, -1, 2: 0/ENTER; 2/ENTER; trig/more/π; basic/*; -1/ENTER; 2/ENTER; mode/prog/draw/y=f(x)/fx5:



Como ocurre normalmente con las ecuaciones trigonométricas que involucran senos y cosenos, la función tiene un comportamiento oscilatorio, por lo tanto tiene múltiples soluciones. Las soluciones aproximadas, visibles en el intervalo graficado, son: 1.8, 4.8, 7.9 y 11.

Las soluciones más exactas en este intervalo serían:

1/ENTER; 2/mode/prog/more/solve/fx5 => 1.807375379182475

4/ENTER; 5/mode/prog/more/solve/fx5 => 4.668505519149945

7/ENTER; 8/mode/prog/more/solve/fx5 => 7.869871742354907

10/ENTER; 12/mode/prog/more/solve/fx5 => 10.98735875694278

La derivada de la función en x=2.2 es:

2.2/mode/prog/more/diff./fx5 => -0.9375074753152148

Finalmente, la integral de la función entre 5 y 7.5 con 14 dígitos de precisión es:

5/ENTER; 7.5/ENTER; 1e-14/mode/prog/more/integrate/fx5 =>

1.961768278991084.

3. Programe la función: $f(x)=5x^{10}-38x^9+21x^8-5\pi x^6-3\pi x^5-5x^2+8x-3=0$. Luego encuentre los valores de la función para x= 1.5. Grafique la función entre "x=-1->1, y=-20->10" y entre "x=6->8, y=-6e7->6e8". En base a la gráfica estime la solución aproximada de la función. Calcule luego una solución más exacta con "solve". Derive la función para "x=-0.5" e integre la función entre 7.0 y 7.8, con 12 dígitos de precisión.

El programa para esta función, trabajando con "x" en la pila, es el siguiente: 1.1/mode/prog/new/<>/fx6:

```
1                    5
2          stack/RCL st# 1
3                   10
4             math/pow/y^x
5                    *
6                   38
7          stack/RCL st#2
8                    9
9             math/pow/y^x
10                   *
```

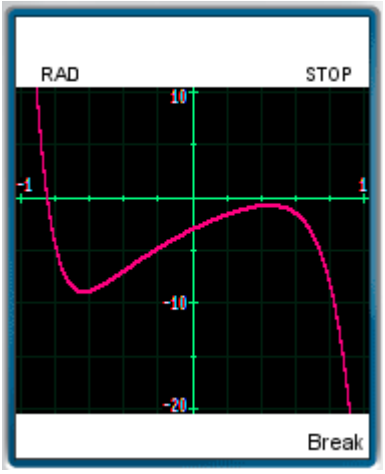
```
11                    -
12                   21
13       stack/RCL st# 2
14                    8
15            math/pow/y^x
16                    *
17                    +
18                    5
19                trig/π
20                    *
21       stack/RCL/st# 2
22                    6
23            math/pow/y^x
24                    *
25                    -
26                    3
27                trig/π
28                    *
29        stack/RCL st#2
30                    5
31            math/pow/y^x
32                    *
33                    -
34                    5
35       stack/RCL st# 2
36                   x^2
37                    *
38                    -
39                    8
40      stack/move dn# 2
41                    *
42                    +
43                    3
44                    -
45              [prg end]
```

Haciendo correr el programa con 0.6, “f(0.5)”, se obtiene:

0.5/mode/prog/run/fx6 => -0.7772659248357457

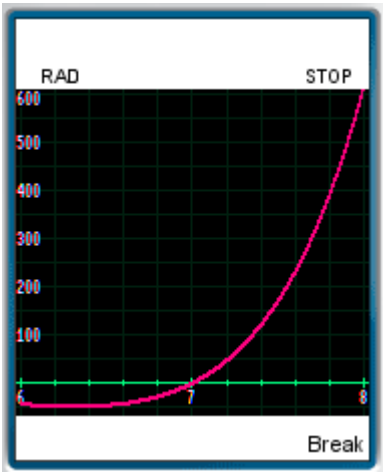
Graficamos la función entre -1->1, -20->10:

-1/ENTER; 1/ENTER; -20/ENTER; 20/ENTER; mode/prog/draw/y=f(x)/fx6:



Y entre 6->8, -3e8->6e8

6/ENTER; 8/ENTER; -6e7/ENTER; 6e8/ENTER; mode/prog/draw/y=f(x)/fx6:



Como se puede ver la primera solución es aproximadamente -0.85 y la segunda 7.

Las soluciones más exactas son:

-1/ENTER; -0.8/mode/prog/more/solve/fx6 => -0.8471298181318167

6/ENTER; 8/mode/prog/more/solve/fx6 => 7.010824348276166

La derivada de la función en x=-0.5 es:

-0.5/mode/prog/more/diff./fx5 => 10.25390624998181

Finalmente, la integral de la función entre 7.0 y 7.8 con 12 dígitos de precisión es:

7/ENTER; 7.8/ENTER; 1e-12/mode/prog/more/integrate/fx6 =>

115377284.7091005.

4. Programe el sistema de ecuaciones no lineales que se presenta al pie de la pregunta como una función de una sola variable: x. Luego grafique la función entre -5 y 5 para x y entre -5 y 5 para y. En base a la gráfica estime las soluciones aproximadas que se encuentran en ese intervalo. Luego, con “solve” encuentre las soluciones más exactas, encontrando también los respectivos valores de “y” y “z”.

$$3x^{2.1} - 5y = 7.0$$

$$y^{1.2} + 4z = 14.3$$
$$x + y^2 + z^2 = 14.0$$

Si todas las ecuaciones del sistema pueden ser colocadas en función de una sola variable, como ocurre en el presente caso, entonces es posible resolver el sistema de ecuaciones no lineales de la misma forma en que se resuelve una ecuación con una sola variable. En este sistema por ejemplo, si conocemos “x” podemos calcular “y” con la primera ecuación, luego con “y” podemos calcular “z” con la segunda, por lo tanto el sistema puede ser escrito como una función de una sola variable:

$$f(x) = x + y^2 + z^2 - 14.0 = 0$$
$$y = \frac{3x^{2.1} - 7.0}{5}$$
$$z = \frac{14.3 - y^{1.2}}{4}$$

En este caso, puesto que además queremos conocer los valores de “y” y “z” (las tres soluciones) deberemos trabajar con posiciones de memoria.

Como siempre antes de comenzar a realizar el programa se debe colocar un valor de prueba en la pila (digamos 1.1.). La lógica que se debe seguir es la siguiente: a) Con el valor de “x” calcular el valor de “y”; b) Con el valor de “y” calcular el valor de “z”; c) Con los valores de “x”, “y” y “z” calculados, calcular el valor de la función “f(x)”.

Además debemos decidir cuáles serán las posiciones de memoria para cada una de las variables. Para este ejemplo las posiciones de memoria serán: “0 para x”, “1 para y” y “2 para z”.

El programa elaborado siguiendo la lógica descrita es el siguiente (1.1/mode/prog/new/<>/fx3/OK):

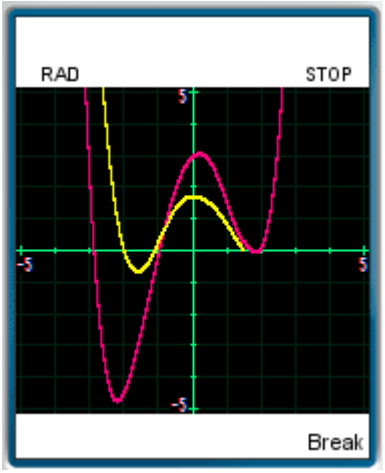
```
1              mem/STO 0
2                  Clear
3                      3
4              mem/RCL 0
5                    2.1
6            math/pow/y^x
7                      *
8                      7
9                      -
10                     5
11                     /
12             mem/STO 1
13                 clear
14                  14.3
15             mem/RCL 1
16                   1.2
17           math/pow/y^x
18                     -
19                     4
20                     /
21             mem/STO 2
22                 clear
23             mem/RCL 0
24             mem/RCL 1
25                    x^2
26                     +
27             mem/RCL 2
28                    x^2
29                     +
30                    14
31                     -
32             [prg end]
```

Calculamos algún valor de la función, por ejemplo para x=1.5: 1.5/mode/prog/run/fx3, con el cual obtenemos: 0.2769102679153484. Los valores respectivos de “y” y “z” para este valor de “x” son:

Valor de x: special/mem/RCL/<0-3>/1 => 0.005862654389754292

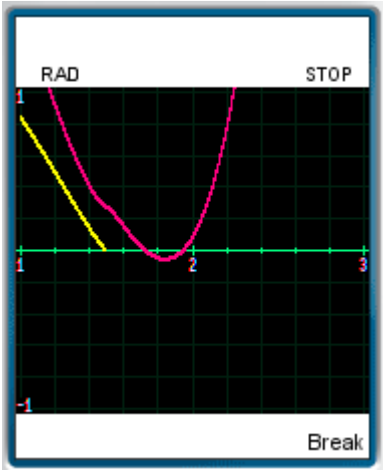
Valor de y: special/mem/RCL/<0-3>/2 => 3.574475611498679

Ahora graficamos la función: -5/ENTER; 5/ENTER; -5/ENTER; 5/ENTER; mode/prog/draw/y=f(x)/fx3:



Observe que en el sector izquierdo de la gráfica aparecen dos curvas, una en color púrpura y otra en amarillo. Esto se debe a que en esta parte la función tiene valores complejos (con parte imaginaria). Una de las dos curvas (la púrpura) corresponde a los valores reales del número complejo, mientras que la otra (la amarilla) corresponde a los valores imaginarios. Entonces, aun cuando aparentemente existen hasta cuatro soluciones, en realidad sólo existen 2 soluciones reales en el sector derecho de la gráfica.

Ampliando ese sector de la gráfica podemos apreciar con mayor claridad dichas soluciones:



Aproximadamente podemos decir que una de las soluciones es 1.65 y la otra 1.88.

Encontremos entonces soluciones más exactas con “solve”.

Para la primera solución:

1.5/ENTER; 1.8/ENTER; mode/prog/more/solve/fx3 => 1.72652614821958

Siendo los valores de “y” y “z”:

x: special/mem/RCL/<0-3>/1 => 0.488925748431201

y: special/mem/RCL/<0-3>/2 => 3.469066944331089

Para la segunda solución:

1.8/ENTER; 2/ENTER; mode/prog/more/solve/fx3 => 1.943797085063063

Siendo los valores de “y” y “z”:

x: special/mem/RCL/<0-3>/1 => 1.022803520608522

y: special/mem/RCL/<0-3>/2 => 3.318143437702437

5. Programe el sistema de cuatro ecuaciones no lineales que se presenta al pie de la pregunta como una función de una sola variable (x). Luego grafique la función entre 0 y 20 (para x), -20 y 20 (para y). En base a la gráfica estime las soluciones aproximadas que se encuentran en ese intervalo. Luego, con “solve” encuentre las soluciones más exactas, encontrando también los respectivos valores de “y”, “z” y “w”.

$$x^{1/2} + z^2 = 35$$
$$y + e^{z/2} - \frac{8}{z} = 30$$
$$w + z^2 - 2z = 31$$
$$x + w + y = 39$$

Primero colocamos una de las ecuaciones del sistema en la forma *f(x)=0* y las otras en función de *“x”*:

$$f(x) = x + w + y - 39 = 0$$
$$z = \sqrt{35 - \sqrt{x}}$$
$$y = 30 + \frac{8}{z} - e^{z/2}$$
$$w = 31 - z^2 + 2z$$

La lógica para programar esta función es: a) Con el valor conocido de “x” se calcula “z”; b) Con el valor de “z” se calcula “y”; c) Con el valor de “z” se calcula “w”; d) Con los valores calculados de “y” y “w” se calcula el valor de la función de x “f(x)”.

Las posiciones de memoria para este programa son: 0: x; 1: y; 2: z; 3: w.

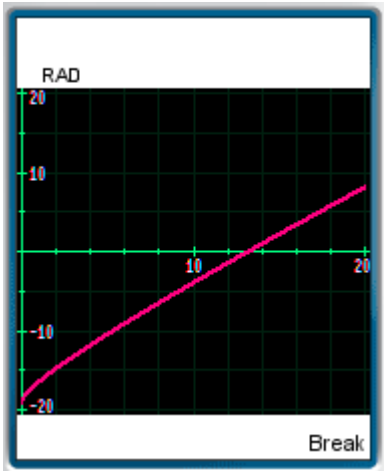
El programa elaborado siguiendo la lógica descrita y las posiciones de memoria indicadas es el siguiente (1.1/mode/prog/new/<>/fx4):

```
1              mem/STO 0
2                  Clear
3                     35
4              mem/RCL 0
5                     √x
6                      -
7                     √x
8              mem/STO 2
9                  clear
10                    30
11                     8
12             mem/RCL 2
13                     /
14                     +
15             mem/RCL 2
16                     2
17                     /
18           math/pow/e^x
19                     -
20             mem/STO 1
```

```
21                clear
22                   31
23            mem/RCL 2
24                   x²
25                    -
26                    2
27            mem/RCL 2
28                    *
29                    +
30            mem/STO 3
31                clear
32            mem/RCL 0
33            mem/RCL 3
34                    +
35            mem/RCL 1
36                    +
37                   39
38                    -
39            [prg end]
```

Probamos el programa con algún valor, por ejemplo 10.2: 10.2/mode/prog/run/fx4 => -3.682698837202406.

Ahora graficamos la función entre los límites dados: 0/ENTER; 20/ENTER; -20/ENTER; 20/ENTER; mode/prog/draw/y=f(x)/fx4:



Como se puede observar, una solución aproximada podría ser 13. Encontremos entonces una solución más exacta con "solve": 12/ENTER; 14/ENTER; mode/prog/more/solve/fx4 => 13.18918859759894. Siendo los valores de "y": special/RCL/<0-3>/1 => 14.97763572398031, "z"= special/RCL/<0-3>/2 => 5.600741714286865, "w" = special/RCL/<0-3>/3 => 10.83317567842075.

De esa manera habríamos encontrado las cuatro soluciones de este sistema.

2.1.2. Ejercicios

1. Programe una función: $f(x)=2x^2+1-e^x=0$. Calcule el valor de la función para x=1.6 y x=3.9. Grafique la función entre los límites que usted elija. En base a la gráfica estime la o las soluciones aproximadas. Encuentre las soluciones más exactas con "solve". Calcule la derivada en el punto que elija y la integral entre los límites que elija con la precisión que considere adecuada.

2. Programe una función: $f(z)=e^{z/2}+z^2-60=0$. Calcule el valor de la función para x=2.1 y x=3.2. Grafique la función entre los límites que usted elija. En base a la gráfica estime la o las soluciones aproximadas. Encuentre las soluciones más exactas con "solve". Calcule la derivada en el punto que elija y la integral entre los límites que elija con la precisión que considere adecuada.

3. Programe una función: $f(z)=ln(z)+e^{z+3}-z^{3.1}-42=0$. Calcule el valor de la función para x=1.7 y x=2.5. Grafique la función entre los límites que usted elija. En base a la gráfica estime la o las soluciones aproximadas. Encuentre las soluciones más exactas con "solve". Calcule la derivada en el punto que elija y la integral entre los límites que elija con la precisión que considere adecuada.

4. Programe una función: $f(x)=x^3-1.94702x^2-6.76988x+9.44203=0$. Calcule el valor de la función para x=3.1 y x=7.2. Grafique la función entre los límites que usted elija. En base a la gráfica estime la o las soluciones aproximadas. Encuentre las soluciones más exactas con "solve". Calcule la derivada en el punto que elija y la integral entre los límites que elija con la precisión que considere adecuada.

5. Programe el sistema de ecuaciones no lineales que se presenta al pie de la pregunta como una función de una sola variable: "x". Luego grafique la función entre los límites que considere adecuados. En base a la gráfica estime las soluciones aproximadas que se encuentran en ese intervalo. Luego, con "solve" encuentre las soluciones más exactas, encontrando también el respectivo valor de "y".

$$\begin{aligned} 2x^2+5xy-4x &= 115 \\ e^{\frac{x+y}{5}}+x^2y^2-70y &= 15 \end{aligned}$$

6. Programe el sistema de ecuaciones no lineales que se presenta al pie de la pregunta como una función de una sola variable: "x". Luego grafique la función entre los límites que considere adecuados. En base a la gráfica estime las soluciones aproximadas que se encuentran en ese intervalo. Luego, con "solve" encuentre las soluciones más exactas, encontrando también el respectivo valor de "y" y "z".

$$\begin{aligned} 15y+20x^2-x^3 &= 1500 \\ 9z-1.5x^2+e^{\frac{x}{2}} &= 300 \\ y^2-z^3-5x &= 166.81 \end{aligned}$$

3. ECUACIONES ALGEBRAICAS CON UNA INCÓGNITA - 1

3.1. Método de sustitución directa

Ahora que estamos en condiciones de resolver funciones con una incógnita en Calc-Java, estudiaremos algunos de los métodos numéricos que existen para este fin. Comenzaremos el estudio con el más sencillo de todos: el método de *Sustitución Directa*, el cual requiere que la función a resolver se encuentre en la forma:

$$x = g(x) \qquad (3.1)$$

Siendo esta su principal desventaja, pues en la mayoría de los casos es necesario despejar previamente la variable independiente, algo que no siempre es sencillo y en algunas ocasiones puede resultar incluso imposible.

En este método se comienza asumiendo un valor para "x" (la solución), se reemplaza en la ecuación despejada (3.1) y si el resultado es igual al valor asumido, el proceso concluye, caso contrario el valor calculado se convierte en el nuevo "x" y con el mismo se vuelve a repetir el proceso.

Por ejemplo dada la ecuación (resuelta ya en el anterior tema):

$$f(x) = \left(52 + 3\sqrt{x} - 8x^{0.8}\right)^{0.36} - x = 0$$

Lo primero que hacemos es despejar una de las "x", en este caso la más sencilla de despejar es la última:

$$x = \left(52 + 3\sqrt{x} - 8x^{0.8}\right)^{0.36} = g(x)$$

Esta función despejada (a la que denominamos "g(x)") es la función que debemos programar y resolver con el método. Comenzamos entonces programando la función: 1.1/mode/prog/new/<>/gx (donde 1.1 es el valor de prueba):

```
33                    52
34                     3
35        stack/RCL st#2
36                    √x
37                     *
38                     +
39                     8
40      stack/move dn# 2
41                   0.8
42            math/pow/y^x
43                     *
44                     –
45                  0.36
46            math/pow/yx
47              [prg end]
```

Ahora hacemos correr el programa con un programa con un valor asumido, por ejemplo 1.1:

1.1/mode/prog/run/gx => 3.984055387788425

Puesto que 1.1 y 3.984…, no son iguales, se emplea el valor calculado como valor asumido:

mode/prog/run/gx => 3.552012132021903

Y se repite el proceso hasta que el valor asumido y el calculado son iguales (o aproximadamente iguales), para conservar una copia del valor anterior, se puede pulsar ENTER, antes de realizar el nuevo cálculo. De esa manera se han obtenido los siguientes valores:

```
                1.1
3.984055387788425
3.552012132021903
3.618479599247315
3.608323565491999
3.609876925738287
3.609639376571533
3.609675704867659
3.609670149216255
 3.60967099883715
3.609670868905383
3.609670888775732
3.609670885736978
3.609670886201692
3.609670886130623
3.609670886141492
 3.60967088613983
3.609670886140084
3.609670886140045
3.609670886140051
 3.60967088614005
 3.60967088614005
```

Como se puede observar los dos últimos valores calculados son iguales, por lo tanto esa es la solución de la ecuación (que es el mismo resultado encontrado con “solve” en el tema anterior).

Gráficamente, cuando la función está en la forma “x=g(x)”, la solución se encuentra en la intersección entre la línea “x=y” (a 45 grados) y la función despejada “g(x)”, tal como se muestra en la siguiente gráfica, donde además se muestra cómo, a partir de un valor inicial asumido “x_1”, el método se va acercando, con cada iteración, a la solución:

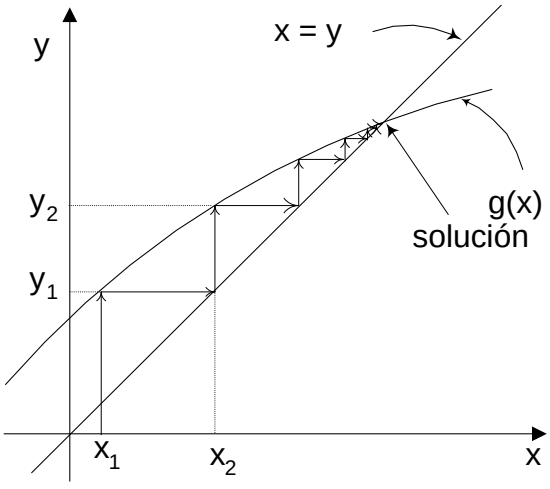


Figura 3.1. Método de Sustitución Directa

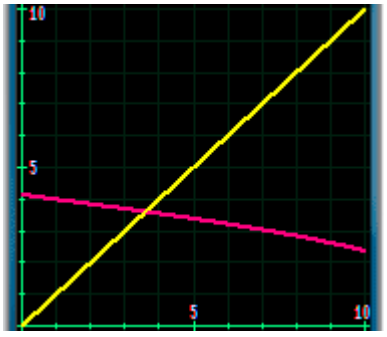
Para ver la solución gráficamente en Calc-Java debemos trabajar con valores complejos, uno de los valores a graficar (el valor de “y” o el de

"g(x)") será la parte imaginaria y el otro la compleja. Elegiremos "g(x)" para la parte real y "y" para la imaginaria (donde "y" es igual a "x"), entonces para ver gráficamente la solución es necesario modificar el programa "gx" de la siguiente forma: mode/prog/append/gx:

```
1                    52
2                     3
3        stack/RCL st#2
4                    √x
5                     *
6                     +
7                     8
8       stack/RCL st# 2   ;Copia del valor de x1
9                   0.8
10          math/pow/y^x
11                    *
12                    -
13                 0.36
14         math/pow/yx
15   trig/coord/r->cplx   ;Se crea el complejo con x1 y g(x)
16            [prg end]
```

Donde lo que se ha hecho es dejar el valor de "x" en la pila (por eso ahora se copia dicho valor con "RCL st#2" en lugar de utilizarlo como antes "mov dn# 2") y con el mismo se crea el número complejo (r->cplx). Ahora empleamos este programa para realizar la gráfica:

0/ENTER; 10/ENTER; 0/ENTER; 10/ENTER; mode/prog/draw/y=f(x)/gx:



Con lo que se corrobora que la solución se encuentra alrededor de x=3.6.

3.1.1. Programa

Si bien este método es lo suficientemente sencillo como para ser aplicado inclusive sin automatizar el proceso: haciendo correr repetidas veces la función programada, tal como se ha hecho en el ejemplo, es más eficiente tener el proceso automatizado mediante un programa.

El algoritmo básico del método es el siguiente:

a. Asumir un valor inicial para "x": x_1.

b. Calcular el valor de la función con x_1: $x_2=g(x_1)$.

c. Si x_1 y x_2 son iguales saltar al paso f.

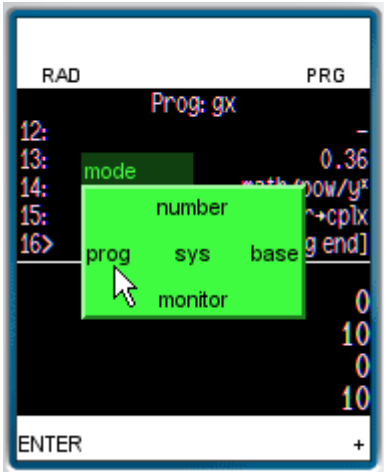
d. Hacer que x_1 tome el valor de x_2 ($x_1=x_2$).

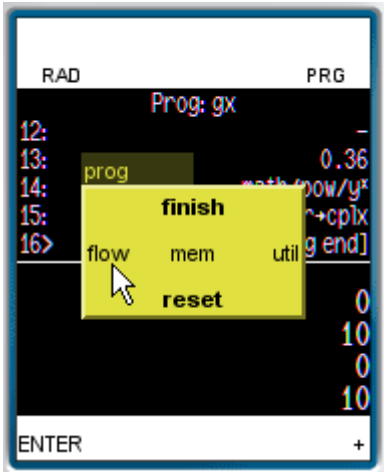
e. Repetir el proceso desde el paso b.

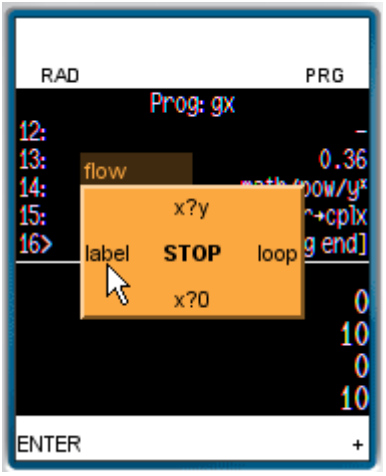
f. Devolver la respuesta: x_2.

Lamentablemente Calc-Java no permite llamar a una función desde otra función, por lo que tanto la función a resolver, así como el método que la resuelve, deben encontrarse en el mismo programa.

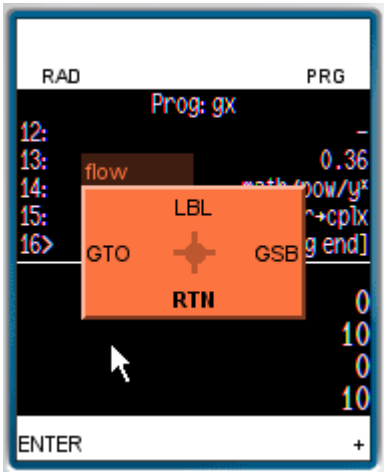
Para implementar este algoritmo (y los otros que veremos luego), es necesario conocer algunas instrucciones adicionales en Calc-Java, que se encuentran dentro de la opción “flow” (flujo) del menú “prog”:







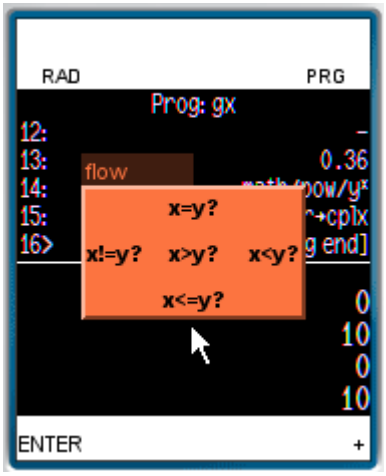
En el menú “label” dentro de “flow” tenemos las siguientes instrucciones:



LBL (label = etiqueta) nos permite fijar un punto dentro del programa al cual podemos saltar desde cualquier otra parte del mismo. Se emplea en conjunción con GTO (go to = ir a) que justamente permite hacer el salto mencionado. En un programa Calc-Java se puede incluir un máximo de 16 etiquetas (del 0 al 15). RTN (return = retorno) permite volver de un sector del programa, al cual se saltó con GSB (go subroutine = ir a subrutina) de manera que el programa continúe con la siguiente instrucción ubicada después del GSB que hizo la llamada.

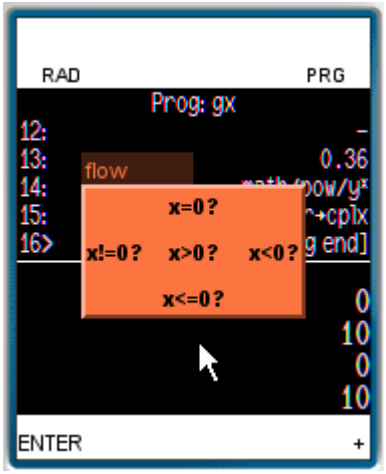
Tanto GTO, como GSB permiten saltar a un sector etiquetado del programa, pero GSB, continúa con la siguiente instrucción del programa una vez que en el sector etiquetado se ejecuta la instrucción RTN.

El menú “x?y” tiene las instrucciones que permiten comparar dos valores de la pila:



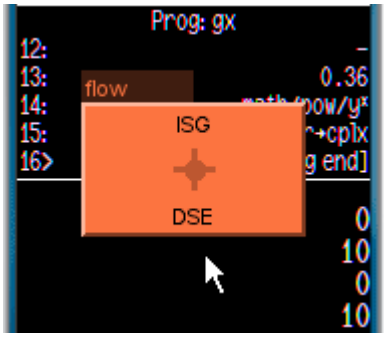
En estas instrucciones, al igual que en todas las que implican dos valores, “x” es el último valor de la pila y “y” el penúltimo. En todas estas instrucciones si el resultado de la comparación es verdadero, se ejecuta la siguiente instrucción del programa, caso contrario se salta la siguiente instrucción y se ejecuta la subsiguiente.

En el menú “x?0” se tienen las instrucciones que permiten comparar un valor que se encuentra en la pila con 0:



Al igual que en el menú anterior, si el resultado de la comparación es verdadero se ejecuta la siguiente instrucción del programa, caso contrario se ejecuta la subsiguiente.

El menú “loop” tiene dos instrucciones:



ISG (Increment, Skip if Greater = incrementar, saltar si es mayor), incrementa un contador ubicado en la posición de memoria que se especifica y

si el mismo es mayor al límite establecido salta la siguiente instrucción y ejecuta la subsiguiente, caso contrario ejecuta la siguiente instrucción. El contador debe ser guardado en una de las 16 posiciones de memoria de acuerdo al siguiente formato: “ccccc.fffii”, donde “ccccc”, es el valor del contador, “fff” es el valor con el cual se compara el contador e “ii” es el incremento, es decir el valor que se suma al contador en cada iteración.

DSE (Decrement, Skip if Equal = Disminuir, saltar si es igual). Esta instrucción es similar a ISG, sólo que en lugar de incrementar, disminuye el contador (“ccccc”) ubicado en la posición de memoria especificada (en el valor establecido por “ii”) y si es igual o menor al límite establecido (”fff”) salta la siguiente instrucción del programa (y ejecuta la subsiguiente), caso contrario ejecuta la siguiente instrucción del programa. El formato del contador almacenado en memoria es el mismo que ISG, es decir: “ccccc.fffii”.

Finalmente la instrucción STOP, sirve para finalizar la ejecución de un programa.

Para programar el método de sustitución directa, la función a resolver (la función resuelta en el ejemplo anterior) será programada en una subrutina, que tendrá la etiqueta 0. En consecuencia esa parte, con excepción de la etiqueta al principio de la misma y la instrucción RTN, al final, es la que se elaboró previamente. Por conveniencia, dicha subrutina será escrita al final del programa.

Entonces se codifica el algoritmo descrito previamente. El programa elaborado siguiendo dicho algoritmo es el siguiente (las letras a continuación del “;” son comentarios para aclara el programa): mode/prog/new/<>/sd:

```
17                  1.1  ;Valor asumido de x: x1
18                ENTER  ;Se duplica x1 para la primera iteración
19      prog/flow/LBL 1  ;Punto desde el cual se repite el proceso
20                Clear  ;Borrado del antiguo valor de x1
21                ENTER  ;Duplicado del nuevo valor de x1
22      prog/flow/GSB 0  ;Cálculo del nuevo valor de x: x2=g(x1)
23           stack/x⇔y   ;Intercambio de x1 con x2
24      prog/flow/x!=y?  ;Es x1 diferente de x2?
25      prog/flow/GTO 1  ;Sin son diferentes se salta a LBL 1
26                clear  ;Borrado del último valor de x1
27        prog/flow/RTN  ;Fin del método de sustitución directa
28      prog/flow/LBL 0  ;Inicio del bloque con la función a resolver
29                   52
30                    3
31       stack/RCL st#2
32                   √x
33                    *
34                    +
35                    8
36     stack/move dn# 2
37                  0.8
38           math/pow/y^x
39                    *
40                    -
41                 0.36
42           math/pow/yx
43         prog/flow/RTN  ;Fin del bloque con la función a resolver
44             [prg end]  ;Fin del programa
```

Haciendo correr el programa se obtiene:

mode/prog/run/sd => 3.60967088614005

Que es el mismo resultado obtenido anteriormente. Sin embargo para que el programa sea de utilidad práctica es necesario hacer algunas consideraciones adicionales:

En primer lugar debemos tomar en cuenta que no siempre se logra la *convergencia*, es decir no siempre el método se aproxima a la solución con cada iteración, y este método en particular es uno de los menos estables, por lo tanto es necesario establecer un límite de iteraciones para evitar ciclos infinitos.

En segundo lugar, aún cuando el proceso sea convergente, no siempre se consigue que los dos últimos valores calculados sean exactamente iguales (debido a errores de redondeo). Por eso es conveniente calcular el resultado con un determinado número de dígitos de precisión, es decir con un número de dígitos que deberán ser iguales en los dos últimos valores calculados. Para ello empleamos la expresión: $|x_2/x_1-1|<1x10^{-n}$, donde "x_1" y "x_2" son los dos últimos valores calculados y "n" es el número de dígitos de precisión. Cuando esta expresión es verdadera, los valores de "x_1" y "x_2" son iguales en el número de dígitos especificado y por lo tanto la solución es precisa en ese número de dígitos. En ingeniería usualmente sólo se requieren soluciones con unos 7 dígitos de precisión, por lo que trabajar con 12 dígitos suele ser más que suficiente.

Tomando en cuenta estos aspectos, el algoritmo para el método de sustitución directa es el siguiente:

a. Establecer la precisión "err", el límite de iteraciones "li" y asumir un valor inicial para "x": x_1.

b. Calcular el valor de la función con x_1: $x_2=g(x_1)$.

c. Si x_1 y x_2 son aproximadamente iguales, es decir si se cumple que $|x_1/x_2-1|<err$, saltar al paso h.

d. Disminuir el contador de repeticiones

e. Si el contador es igual a 0, generar un error y saltar al paso h.

f. Hacer que x_1 tome el valor de x_2 ($x_1=x_2$).

g. Repetir el proceso desde el paso b.

h. Devolver x_2.

La precisión debe ser escrita en la forma "1e-n", donde "n" es la precisión). El programa modificado, tomando en cuenta las anteriores observaciones es el siguiente:

```
1                 1e-12  ;Precisión (err)
2               50.00001  ;Límite de iteraciones (50)
3             mem/STO 15  ;Contador almacenado en la memoria 15
4                  clear  ;Borrado del valor del contador
5                    1.1  ;Valor asumido de x: x1
6                  ENTER  ;Se duplica x1 dos veces para la
7                  ENTER  ;primera iteración
8        prog/flow/LBL 1  ;Punto desde el cual se repite el proceso
9                  clear  ;Borrado de los dos últimos valores de
10                 clear  ;la pila
11                 ENTER  ;Se duplica el nuevo valor de x1
12       prog/flow/GSB 0  ;Cálculo del nuevo valor de x: x2=g(x1)
13                 ENTER  ;Se duplica x2
14      stack/move up# 2  ;Se sube x2 a la tercera posición de la pila
```

```
15                    /   ;Se calcula |x1/x2-1|
16                    1
17                    -
18                  abs
19       stack/RCL st# 2   ;Se copia la precisión
20        prog/flow/x>y?   ;Se verifica si se ha alcanzado la precisión
21       prog/flow/GTO 2   ;Si la precisión es correcta se salta a LBL 2
22      prog/flow/DSE 15   ;Se disminuye el contador en uno
23       prog/flow/GTO 1   ;Si no es 0 se repite el ciclo desde LBL 1
24                    0   ;Si es 0 se genera el error correspondiende
25                ENTER   ;a la división de 0 entre 0
26                    /
27      stack/move up# 4   ;Se sube el error a la quinta posición
28       prog/flow/LBL 2
29                Clear   ;Se borran los valores comparados
30                clear
31            Stack/x⇔y   ;Se intercambia "err" con "x2"
32                Clear   ;Se borra "err"
33          prog/flow/RTN ;Fin del método de sustitución directa
34       prog/flow/LBL 0   ;Inicio del bloque con la función a resolver
35                   52
36                    3
37         stack/RCL st#2
38                   √x
39                    *
40                    +
41                    8
42      stack/move dn# 2
43                  0.8
44            math/pow/y^x
45                    *
46                    –
47                 0.36
48            math/pow/yx
49          prog/flow/RTN   ;Fin la función a resolver
50              [prg end]   ;Fin del programa
```

Haciendo correr el programa se obtiene:

mode/prog/run/sd => 3.60967088613983

Que es igual al resultado anterior sólo en los primeros 12 dígitos (redondeados), porque tal como se puede ver en el programa, la precisión empleada en los cálculos es de 12 dígitos.

3.1.2. Ejemplos

1. Como primer ejemplo, emplearemos el programa elaborado para resolver la función: $f(x)=x^3+2x^2+3x+4=0$, con 9 dígitos de precisión y un límite de 100 iteraciones. Para estimar el valor inicial del método debe graficar previamente la ecuación igualada a "x" entre x=-5, x=5, y=-5, y=5.

Primero despejamos uno de los términos de la variable independiente para que la función se encuentre en la forma x=g(x):

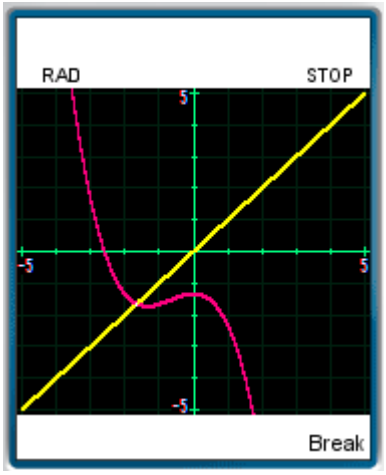
$$x = -(x^3+2x^2+4)/3$$

Primero programamos esta función para graficarla y ver así el lugar de la solución. Para ello, y como ya hicimos previamente, devolvemos un resultado

complejo donde el valor de “x” es la parte imaginaria y el valor de la función “g(x)” la parte real: mode/prog/new/gx:

```
1                   ENTER
2                       3
3              math/pow/y^x
4                       2
5          stack/RCL st#2
6                      x^2
7                       *
8                       +
9                       4
10                      3
11                      /
12                   +/-
13     trig/coord/r->cplx
14              [prg end]
```

Graficando esta función se obtiene: mode/prog/draw/y=f(x)/gx:



Donde se puede ver que la solución se encuentra entre -1 y -2, por lo que un valor inicial adecuado podría ser -1.5.

Ahora básicamente editamos el programa con el método de Sustitución Directa (mode/prog/append/sd), cambiamos la precisión, límite de iteraciones, valor inicial y reemplazamos el bloque etiquetado con LBL 0, con el programa empleado para obtener la gráfica, sólo que en lugar de un resultado imaginario devolvemos un resultado real. Las partes modificadas del programa “sd” son las siguientes:

```
1                   1e-9   ;Precisión (err)
2              100.00001   ;Límite de iteraciones (100)
3              mem/STO 15  ;Contador almacenado en la memoria 15
4                   clear  ;Borrado del valor del contador
5                    -1.5  ;Valor asumido de x: x1
   …
34       prog/flow/LBL 0   ;Inicio del bloque con la función a resolver
35                  ENTER
36                      3
37             math/pow/y^x
38                      2
39       stack/move dn# 2
40                     x^2
41                      *
```

```
42                    +
43                    4
44                    +
45                    3
46                    /
47                  +/-
48        prog/flow/RTN   ;Fin la función a resolver
49            [prg end]
```

Haciendo correr el programa se obtiene:

mode/prog/run/gx => -1.65062919001334

Que es la solución de la función con 9 dígitos de precisión.

2. Como segundo ejemplo, emplearemos el programa elaborado para resolver el sistema de tres ecuaciones que se presenta al pie de la pregunta, con 10 dígitos de precisión y un límite de 70 iteraciones. Para estimar el valor inicial del método debe graficar previamente la ecuación igualada a "x" entre x=1, x=2, y=-5, y=5.

$$3x^{2.1} - 5y = 7.0$$
$$y^{1.2} + 4z = 14.3$$
$$x + y^2 + z^2 = 14.0$$

Primero colocamos una de las ecuaciones del sistema en la forma *x=g(x)*, y las otras en función de la variable independiente:

$$x = g(x) = 14.0 - y^2 - z^2$$
$$y = \frac{3x^{2.1} - 7.0}{5}$$
$$z = \frac{14.3 - y^{1.2}}{4}$$

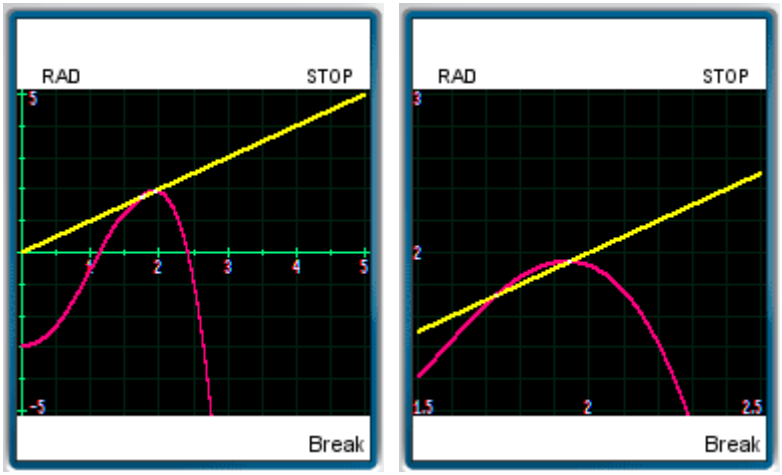
Programamos esta función, devolviendo el resultado (valor de "x" y "g(x)") como un número complejo: mode/prog/new/gx2

```
1                     3
2        stack/RCL st# 1
3                   2.1
4            math/pow/y^x
5                     *
6                     7
7                     -
8                     5
9                     /
10             mem/STO 1
11                 clear
12                  14.3
13             mem/RCL 1
14                  1.2
15           math/pow/y^x
16                    -
17                    4
18                    /
19             mem/STO 2
20                 clear
21                    14
22             mem/RCL 1
23                   x^2
```

```
24                    -
25             mem/RCL 2
26                   x²
27                    -
28     trig/coord/r*cplx
29             [prg end]
```

Graficamos esta función (y la ampliamos luego para ver con más claridad el lugar de las soluciones):

mode/prog/draw/y=f(x)/gx2:



Vemos entonces que las soluciones se encuentran entre 1.7 y 1.8 y entre 1.9 y 2.0, por lo que valores iniciales aproximados para el método de Sustitución Directa podrían ser 1.74 y 1.95.

Ahora modificamos el programa “sd” (Sustitución Directa) para cambiar la precisión, el límite de iteraciones, los valores iniciales (comenzando con 1.74) y la función a resolver. Las partes modificadas del programa son:

```
1                 1e-10  ;Precisión (err)
2               70.00001  ;Límite de iteraciones (70)
3             mem/STO 15  ;Contador almacenado en la memoria 15
4                  clear  ;Borrado del valor del contador
5                   1.74  ;Valor asumido de x: x1
   …
6         prog/flow/LBL 0  ;Inicio del bloque con la función a resolver
7                       3
8       stack/move dn# 1
9                     2.1
10           math/pow/y^x
11                      *
12                      7
13                      -
14                      5
15                      /
16             mem/STO 1
17                 clear
18                  14.3
19             mem/RCL 1
20                    1.2
21           math/pow/y^x
22                      -
23                      4
```

```
24                    /
25            mem/STO 2
26                clear
27                   14
28            mem/RCL 1
29                   x²
30                    -
31            mem/RCL 2
32                   x²
33                    -
34        prog/flow/RTN   ;Fin la función a resolver
35            [prg end]
```

Haciendo correr el programa se obtiene:

mode/prog/run/sd => 1.943797085076064

Es decir el método converge hacia la segunda solución, no la primera como se quería. Empleando un valor inicial igual a 1.95, es decir cambiado el valor del paso 5, se obtiene:

mode/prog/run/sd => 1.943797085067777

Es decir la misma solución que el caso anterior (por supuesto con los 10 dígitos de precisión especificados). Analizando la gráfica y tomando en cuenta la forma en que opera el método de Sustitución Directa, podemos ver que si se da un valor inicial por debajo de la primera raíz, en cada iteración el método se alejará más y más de la solución (es decir diverge de la solución) y si se da un valor por encima de la primera raíz igualmente se aleja de esta raíz y converge hacia la segunda, es por eso que en ambos casos se obtiene la segunda solución y no la primera.

En consecuencia, en este caso en particular, el método de Sustitución Directa no puede encontrar la primera raíz (o solución). Como ya se dijo, el método de sustitución directa es un método inestable y no es raro encontrar casos como este cuando se trabaja con el mismo.

3. Como tercer ejemplo, emplearemos “sd” para resolver la función que se presenta al pie de la pregunta, calculando el resultado con 12 dígitos de precisión y un límite de 50 iteraciones. Para estimar el valor inicial del método se graficará la función igualada a “x_1” entre los límites: 0, 1, 0 y 1. Se requiere conocer también los valores de “y_1”, “L_1”, y “V_1”. Los valores de “L_0”, “x_0”, “V_0” y “y_0” son: “L_0=300”, “x_0=0”, “V_0=100”, “y_0=0.2”.

$$f(x_1) = L_1 * x_1 + V_1 * y_1 - C_0 = 0$$

$$L_1 = \frac{L_c}{1 - x_1}$$

$$V_1 = M - L_1$$

$$y_1 = 1.420 x_1$$

$$L_c = L_0 (1 - x_0)$$

$$M = L_0 + V_0$$

$$C_0 = L_0 x_0 + V_0 y_0$$

Primero colocamos función (f(x1)) en la forma x=g(x), despejando “x_1” de la función:

$$x_1 = g(x_1) = \frac{C_0 - V_1 * y_1}{L_1}$$

Primero programamos la función devolviendo un resultado complejo para ver el lugar de las soluciones. Puesto que en este caso se quiere conocer también los valores de "y_1", "L_1" y "y_1", trabajaremos con posiciones de memoria, sin embargo debemos tomar muy en cuenta que no podemos emplear la posición de memoria 15, pues la misma es empleada como contador en el método de Sustitución Directa. Emplearemos las siguientes posiciones de memoria: 0=x_1; 1=L_0; 2=x_0; 3=V_0; 4=y_0; 5=C_0; 6=M; 7=Lc; 8=y_1; 9=L_1; y 10=V_1.

Debemos hacer notar también que es posible trabajar con la pila para todos los otros valores, excepto "y_1", "L_1" y "y_1", de manera que sólo se utilicen tres posiciones de memoria, aunque quizá el programa podría resultar menos entendible.

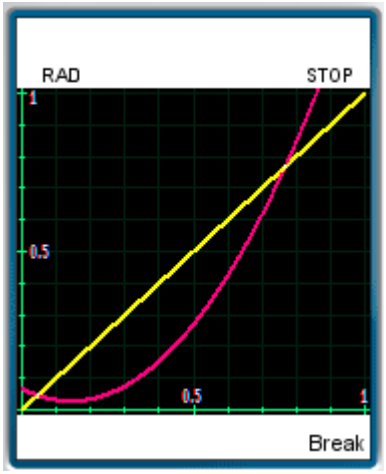
El programa es el siguiente: 0.1/mode/prog/new/gx3:

```
1             mem/STO 0
2                 clear
3                   300
4             mem/STO 1
5                 clear
6                     0
7             mem/STO 2
8                 clear
9                   100
10            mem/STO 3
11                clear
12                  0.2
13            mem/STO 4
14                clear
15            mem/RCL 1
16            mem/RCL 2
17                    *
18            mem/RCL 3
19            mem/RCL 4
20                    *
21                    +
22            mem/STO 5
23                clear
24            mem/RCL 1
25            mem/RCL 3
26                    +
27            mem/STO 6
28                clear
29            mem/RCL 1
30                    1
31            mem/RCL 2
32                    -
33                    *
34            mem/STO 7
35                clear
36                 1.42
37            mem/RCL 0
38                    *
39            mem/STO 8
40                clear
41            mem/RCL 7
42                    1
43            mem/RCL 0
44                    -
45                    /
```

```
46              mem/STO 9
47                  clear
48              mem/RCL 6
49              mem/RCL 9
50                      -
51             mem/STO 10
52                  clear
53              mem/RCL 5
54             mem/RCL 10
55              mem/RCL 8
56                      *
57                      -
58              mem/RCL 9
59                      /
60              mem/RCL 0
61             stack/x⇔y
62     trig/coord/r*cplx
63              [prg end]
```

Graficando esta función se obtiene:

0/ENTER; 1/ENTER; 0/ENTER; 1/ENTER; mode/prog/draw/y=f(x)/gx3



Como se puede ver en este caso también existen dos soluciones: una entre 0 y 0.1 y otra entre 0.7 y 0.8, sin embargo, por la forma de la curva podemos prever que el método sólo convergerá hacia la primera raíz (o solución) pero nunca a la segunda. De todas formas probaremos el método con dos valores iniciales aproximados, que pueden ser: 0.5 y 0.75, comenzando con 0.75 (que debería converger a la segunda solución).

Las modificaciones hechas al programa “sd” (es decir cambiar el error, el límite de iteraciones, el valor inicial asumido y la función a resolver) son las siguientes:

```
1                  1e-12  ;Precisión (err)
2              50.00001   ;Límite de iteraciones (50)
3             mem/STO 15  ;Contador almacenado en la memoria 15
4                  clear  ;Borrado del valor del contador
5                   0.75  ;Valor asumido de x: x1
  …
34       prog/flow/LBL 0  ;Inicio del bloque con la función a resolver
35             mem/STO 0
36                 clear
37                   300
```

```
38              mem/STO 1
39                  clear
40                      0
41              mem/STO 2
42                  clear
43                    100
44              mem/STO 3
45                  clear
46                    0.2
47              mem/STO 4
48                  clear
49              mem/RCL 1
50              mem/RCL 2
51                      *
52              mem/RCL 3
53              mem/RCL 4
54                      *
55                      +
56              mem/STO 5
57                  clear
58              mem/RCL 1
59              mem/RCL 3
60                      +
61              mem/STO 6
62                  clear
63              mem/RCL 1
64                      1
65              mem/RCL 2
66                      -
67                      *
68              mem/STO 7
69                  clear
70                   1.42
71              mem/RCL 0
72                      *
73              mem/STO 8
74                  clear
75              mem/RCL 7
76                      1
77              mem/RCL 0
78                      -
79                      /
80              mem/STO 9
81                  clear
82              mem/RCL 6
83              mem/RCL 9
84                      -
85             mem/STO 10
86                  clear
87              mem/RCL 5
88             mem/RCL 10
89              mem/RCL 8
90                      *
91                      -
92              mem/RCL 9
93                      /
94          prog/flow/RTN   ;Fin la función a resolver
95              [prg end]
```

Haciendo correr el programa se obtiene:

```
mode/prog/run/sd => 0.04587771997384095
```

Que es la solución correspondiente a la primera raíz y no a la segunda. Si se emplea un valor inicial igual a 0.5 se obtiene:

```
5                    0.5   ;Valor asumido de x: x1
mode/prog/run/sd => 0.04587771997385933
```

Que es igualmente la solución correspondiente a la primera raíz (con 12 dígitos de precisión), tal como se predijo que pasaría debido a la forma de la función y a la manera en que procede el método de Sustitución Directa.

3.1.3. Ejercicios

1. Resuelva el ejercicio 1, del tema 2, por el método de Sustitución Directa, con 14 dígitos de precisión, un límite de 40 iteraciones y estimando los valores iniciales en base a la gráfica de la función despejada.
2. Resuelva el ejercicio 2, del tema 2, por el método de Sustitución Directa con 10 dígitos de precisión, un límite de 60 iteraciones y estimando los valores iniciales en base a la gráfica de la función despejada.
3. Resuelva el ejercicio 3, del tema 2, por el método de Sustitución Directa con 9 dígitos de precisión, un límite de 50 iteraciones y estimando los valores iniciales en base a la gráfica de la función despejada.
4. Resuelva el ejercicio 4, del tema 2, por el método de Sustitución Directa con 8 dígitos de precisión, un límite de 100 iteraciones y estimando los valores iniciales en base a la gráfica de la función despejada.
5. Resuelva el ejercicio 5, del tema 2, por el método de Sustitución Directa con 11 dígitos de precisión, un límite de 70 iteraciones y estimando los valores iniciales en base a la gráfica de la función despejada.
6. Resuelva el ejercicio 6, del tema 2, por el método de Sustitución Directa con 9 dígitos de precisión, un límite de 80 iteraciones y estimando los valores iniciales en base a la gráfica de la función despejada.

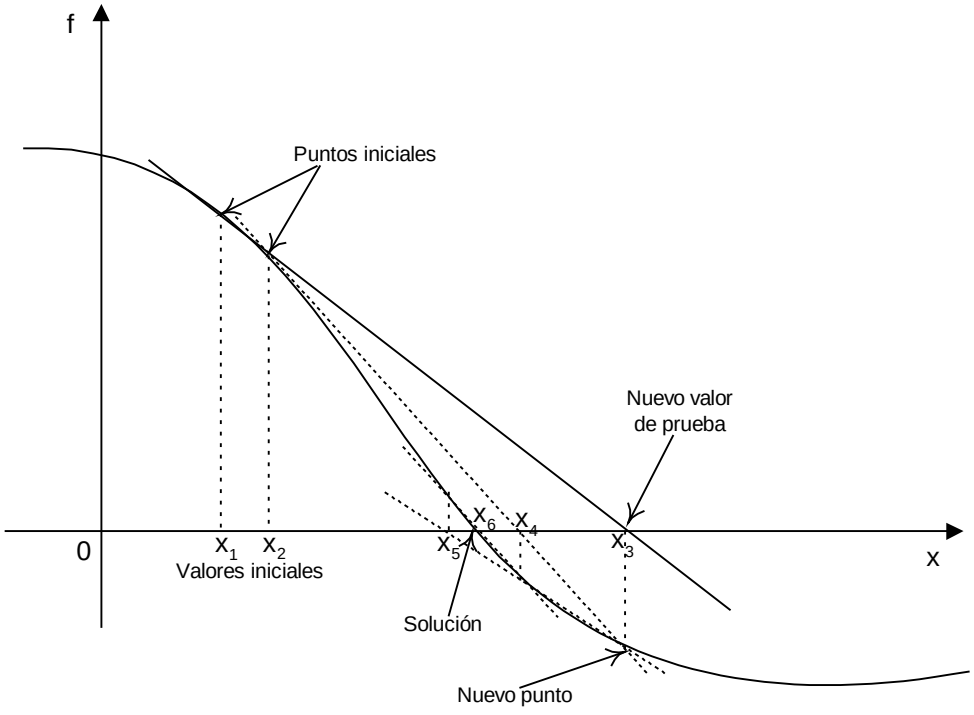
4. ECUACIONES ALGEBRAICAS CON UNA INCÓGNITA - 2

4.1. Método de la Secante

Este método, a diferencia del anterior (Sustitución Directa), requiere que la función se encuentre igualada a cero, es decir que se encuentre en la forma f(x)=0.

En general la mayoría de los métodos existentes para la resolución de ecuaciones algebraicas operan con la forma f(x)=0. Así ocurre por ejemplo con el método implementado en Calc-Java a través de la instrucción "solve" (y estudiada ya en el tema 2). Probablemente el método implementado en Calc-Java sea el de Regula-Falsi (conocido también como método de interpolación lineal), que es similar al que estudiaremos en este tema, sólo que en el método de la Secante no es necesario que los valores iniciales se encuentren a ambos lados de la solución.

Como se puede ver en la figura, el método comienza con dos valores iniciales asumidos, los cuales por lo general son dos valores consecutivos. Con estos valores se calcula el nuevo valor de prueba "x_3" en la intersección entre la línea recta que pasa a través de los dos puntos y la recta "f(x)=0". Entonces se comprueba si se ha encontrado la solución (es decir si el valor de "$f(x_3)$" es casi cero), de ser así el proceso concluye, caso contrario el proceso se repite empleando siempre los dos últimos puntos calculados, tal como se puede ver en la figura.



La ecuación que permite calcular el nuevo valor de prueba (resultante de la intersección de dos líneas rectas) puede ser deducida en base a la forma general de la ecuación de la línea recta *(y=f(x)=a+b*x)*, sustituyendo en la misma los dos puntos que limitan el segmento:

$$f(x_1) = a + bx_1 \tag{4.1}$$

$$f(x_2) = a + bx_2 \tag{4.2}$$

Restando ambas ecuaciones y efectuando algunas operaciones obtenemos la expresión para la constante *b*:

$$f(x_1) - f(x_2) = b(x_1 - x_2) \Rightarrow b = \frac{f(x_1) - f(x_2)}{x_1 - x_2} \tag{4.3}$$

Sustituyendo ahora esta expresión en le ecuación (4.1), y efectuando algunas operaciones obtenemos la expresión para la constante a:

$$\begin{aligned} & f(x_1) = a + \frac{f(x_1) - f(x_2)}{x_1 - x_2} x_1 \\ & \Rightarrow a = \frac{f(x_1)x_1 - f(x_1)x_2 - f(x_1)x_1 + f(x_2)x_1}{x_1 - x_2} = \frac{f(x_2)x_1 - f(x_1)x_2}{x_1 - x_2} \end{aligned} \tag{4.4}$$

Reemplazando estas expresiones en la ecuación general de la línea recta y tomando en cuenta que en la intersección la función es cero mientras que el nuevo valor de *"x" se* denomina x_3, se tiene:

$$f(x_3) = \frac{f(x_2)x_1 - f(x_1)x_2}{x_1 - x_2} + \frac{f(x_1) - f(x_2)}{x_1 - x_2} x_3 = 0 \tag{4.5}$$

Por consiguiente la ecuación que permite calcular el nuevo valor de prueba para el método de La Secante es:

$$x_3 = \frac{f(x_2)x_1 - f(x_1)x_2}{f(x_2) - f(x_1)} \tag{4.6}$$

4.2. Algoritmo del método de la Secante

La lógica del método de la secante es la siguiente:

a) Se fija el error permitido: err y el límite de iteraciones: li.

b) Se asumen dos valores iniciales para la solución (normalmente cercanos uno del otro): x_1 y x_2.

c) Con los valores asumidos se calculan los valores de la función: $f(x_1)$ y $f(x_2)$.

d) Con la ecuación 4.6 se calcula el nuevo valor de prueba: x_3.

e) Si x_2 y x_3 son aproximadamente iguales, es decir si: $|x_2/x_3-1|<err$, se salta al paso j.

f) Con x_3 se calcula el valor de la función: $f(x_3)$.

g) Si el valor absoluto de $f(x_3)$ es casi cero, es decir si: $|f(x3)|<err$, se salta al paso j.

h) Se disminuye el límite de iteraciones en 1 (li=li-1) y si su valor es cero (li=0), se genera un error y se salta al paso j.

i) Se intercambian valores: $x_1=x_2$; $x_2=x_3$; $f(x_1)=f(x_2)$; $f(x_2)=f(x_3)$ y se repite el proceso desde el paso c.

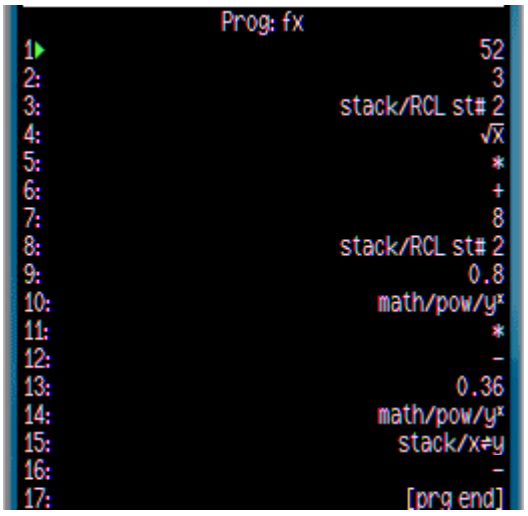
j) Se devuelve la solución (o raíz) de la función: x_3.

Para comprender mejor el algoritmo resolveremos la siguiente expresión (resuelta en el tema anterior), siguiendo los pasos descritos previamente.

$$f(x)=\left(52+3\sqrt{x}-8x^{0.8}\right)^{0.36}-x=0$$

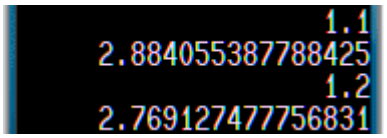
En este caso, no se requiere realizar ninguna operación adicional, pues la ecuación está ya en la forma f(x)=0.

Comenzamos entonces programando la función: 1.1/mode/prog/new/<>/fx (donde 1.1 es el valor de prueba):



Comenzamos entonces con dos valores iniciales asumidos, para demostrar que no necesariamente deben estar a ambos lados de la solución (como ocurre con "solve"), tomaremos como valore iniciales 1.1 y 1.2 (x_1=1.1; x_2=1.2) y con los mismos calculamos el valor de la función:

1.1/ENTER; ENTER; mode/prog/run/fx; 1.2/ENTER; ENTER; mode/prog/run/fx:



Con estos valores calculamos el nuevo valor de x: x_3, pero dado que repetiremos esta operación varias veces es conveniente programarlas:

mode/prog/new/<>/x3:

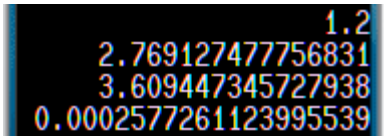
Haciendo correr el programa se obtiene: mode/prog/run/x3:

Que evidentemente es muy diferente al segundo valor asumido: 1.2, por lo que se continua con el siguiente paso calculando el valor de la función:

mode/prog/run/fx1:

Que a pesar de ser la primera iteración ya se está aproximando a cero, pero que todavía no está lo suficientemente cercano a cero, por lo que debemos repetir el procedimiento. Para ello se debe hacer un cambio de valores y dado que en la pila se encuentran: x_1, $f(x_1)$, x_2, $f(x_2)$, x_3 y $f(x_3)$, es suficiente borrar los dos valores superiores de la pila para que los cuatro valores restantes se conviertan en los nuevos valores de prueba para la siguiente iteración (asumiendo, por supuesto que no existe nada más en la pila):

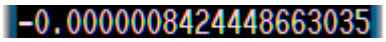
special/stack/more/rollup; special/stack/more/rollup; clear; clear:



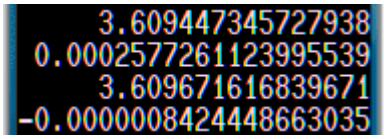
Que son los nuevos valores para la siguiente iteración.

Calculamos un nuevo valor de x: mode/prog/run/x3:

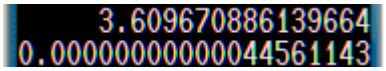
Que como se ve ya tiene cuatro dígitos iguales al valor anterior de x (x_2), es decir que ya tiene cuatro dígitos de precisión, sin embargo no son todavía suficientes, por lo que continuamos con el siguiente paso y calculamos el valor de la función: mode/prog/run/fx1:



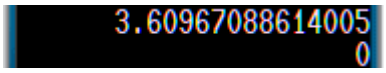
Que ya tiene 6 dígitos de exactitud, pero que aún no son suficientes, por lo que repetimos más el proceso borrando primero los dos valores superiores de la pila (antiguos valores de “x” y “f(x)”):



Con estos valores los nuevos valores de “x” y “f(x)” (x_3 y $f(x_3)$) son:



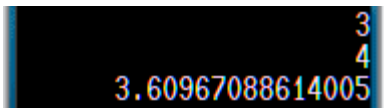
Que tienen 6 dígitos de precisión y 12 de exactitud. Repitiendo el proceso una vez más se obtiene:



Que tiene 11 dígitos de precisión y que es exacta (el valor de la función es 0), por lo que 3.60967088614005, es la solución de la ecuación. Resultado que concuerda con el obtenido con el método de Sustitución Directa, pero que en este caso requiere sólo 5 iteraciones.

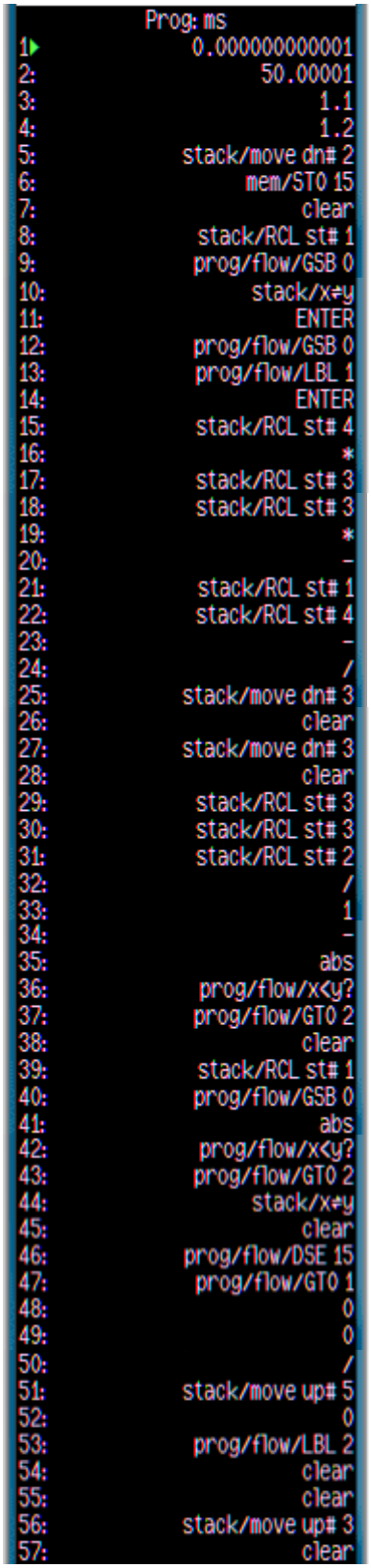
Podemos comparar este resultado con el obtenido con “solve” de Calc-Java:

3/ENTER; 4/ENTER/; mode/prog/more/solve/fx1:



4.2.1. Programa

El programa elaborado, siguiendo el algoritmo descrito y ejemplificado en el anterior acápite es el siguiente: mode/prog/new/<>/ms:



```
58:                clear
59:                clear
60:          prog/flow/RTN
61:       prog/flow/LBL 0
62:                   52
63:                    3
64:       stack/RCL st# 2
65:                   √x
66:                    *
67:                    +
68:                    8
69:       stack/RCL st# 2
70:                  0.8
71:          math/pow/y^x
72:                    *
73:                    -
74:                 0.36
75:          math/pow/y^x
76:            stack/x≠y
77:                    -
78:          prog/flow/RTN
79:            [prg end]
```

En este programa, los pasos 1 al 4 corresponden a los datos: error, límite de iteraciones y valores iniciales asumidos. En los pasos 5 al 6 se guarda en la memoria 15 el límite de iteraciones. En los pasos 8 al 12 se calculan los valores de las funciones y se reordena la pila de manera que quede en el orden: x_1, $f(x_1)$, x_2 y $f(x_2)$. En los pasos 14 al 24 se calcula el nuevo valor de x (x_3). En los pasos 35 a 37 se comprueba si se ha alcanzado la precisión, es decir si se cumple que $|x_1/x_2-1|<err$. En los pasos 37 al 43 se comprueba si se ha alcanzado la exactitud, es decir si se cumple que $|f(x_1)-f(x_2)|<err$. En los pasos 44 a 47 se verifica si se ha llegado al límite de iteraciones, es decir si la memoria 15 ha llegado a cero. En los pasos 48 a 52 se genera un error (en caso de haber llegado al límite de iteraciones) y se reordena la pila para mostrar el error y el último valor calculado. En los pasos 53 a 60, se borran los datos de la pila y se deja sólo el último valor de x_3 (y el error en caso de haberse alcanzado el límite de iteraciones). Finalmente a partir del paso 61 se escribe la función que se quiere resolver.

Haciendo correr el programa se obtiene: (mode/prog/run/md)

```
      3.609670886140053
```

Que es prácticamente el resultado exacto, aún cuando sólo se ha especificado una precisión de 12 dígitos.

4.2.2. Ejemplos

1. Como primer ejemplo, emplearemos el programa elaborado para resolver la función: $f(x)=x^3+2x^2+3x+4=0$, con 9 dígitos de precisión o exactitud y un límite de 30 iteraciones. Para estimar el valor inicial del método debe graficar previamente la ecuación entre x=-5, x=5, y=-5, y=5.

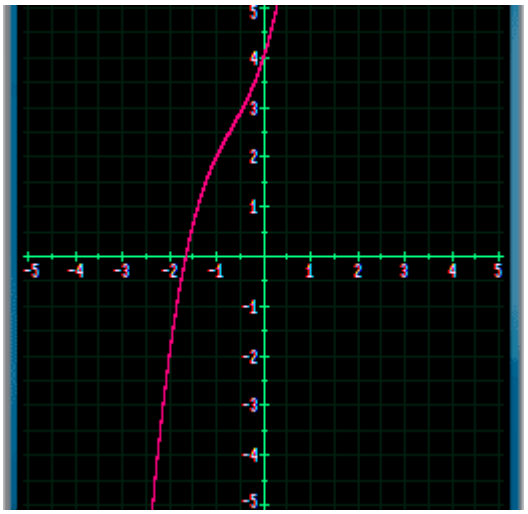
Primero programamos esta función para graficarla y ver así el lugar de la solución: mode/prog/new/fx:

```
Prog: fx
1:                    ENTER
2:                        3
3:             math/pow/y^x
4:                        2
5:          stack/RCL st# 2
```

```
6:                    x²
7:                     *
8:                     +
9:                     3
10:  stack/move dn# 2
11:                    *
12:                    +
13:                    4
14:                    +
15>          [prg end]
```

Graficando la función se obtiene:

-5/ENTER; 5/ENTER; -5/ENTER; 5/ENTER; mode/prog/draw/y=f(x)/fx:



De esta gráfica podemos deducir que la solución está entre -1 y -2, por lo que podemos emplear dichos valores como valores iniciales para encontrar una solución más exacta con el método de la secante.

Cambiamos primero los datos correspondientes al error, límite de iteraciones y valore iniciales asumidos:

```
Prog: ms
1:                0.000000001
2:                   30.00001
3:                          -1
4>                          -2
5:   stack/move dn# 2
```

Y programamos la función después de la etiqueta 0:

```
61▸  prog/flow/LBL 0
62:                  ENTER
63:                      3
64:          math/pow/yˣ
65:                      2
66:  stack/RCL st# 2
67:                     x²
68:                      *
69:                      +
70:                      3
71:  stack/move dn# 2
72:                      *
73:                      +
74:                      4
75:                      +
```

```
76:                    prog/flow/RTN
77:                        [prg end]
```

Haciendo correr el programa (mode/prog/run/fx) se obtiene:

```
                  -1.650629191440103
```

Solución que podemos comparar con la que se obtiene con "solve" de Calc-Java (-1/ENTER, -2/ENTER; mode/prog/more/solve/fx):

```
                                  -1
                                  -2
                  -1.650629191439388
```

Y que como se ve son iguales en los primeros 10 dígitos, que es el resultado correcto de acuerdo al error permitido: 1e-9. Volviendo a hacer correr el programa con un error igual a 1e-14 se obtiene:

```
                  -1.650629191439388
```

Que es el mismo resultado calculado con "solve".

2. Como segundo ejemplo, emplearemos el programa elaborado para resolver el sistema de tres ecuaciones que se presenta al pie de la pregunta, con 12 dígitos de precisión y un límite de 40 iteraciones. Para estimar el valor inicial del método debe graficar previamente la ecuación igualada a "x" entre x=1.6, x=2.1, y=-0.5, y=1.

$$3x^{2.1} - 5y = 7.0$$
$$y^{1.2} + 4z = 14.3$$
$$x + y^2 + z^2 = 14.0$$

Primero colocamos una de las ecuaciones del sistema en la forma *f(x)=0*, y las otras en función de la variable independiente:

$$f(x) = x + y^2 + z^2 - 14.0 = 0$$
$$y = \frac{3x^{2.1} - 7.0}{5}$$
$$z = \frac{14.3 - y^{1.2}}{4}$$

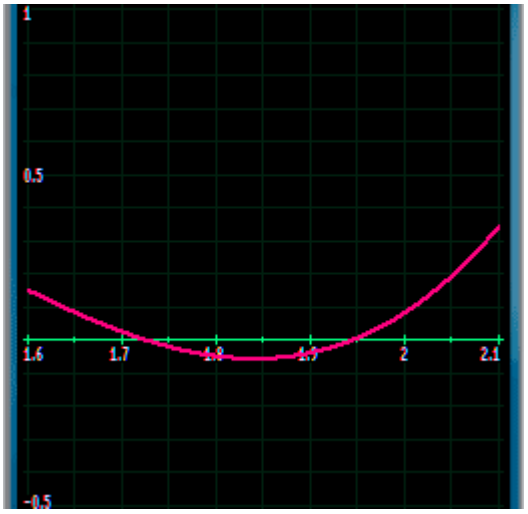
Programamos esta función, para graficarla y obtener así los valores iniciales para el método: mode/prog/new/fx

```
                    Prog: fx
1▶                  mem/STO 0
2:                      clear
3:                          3
4:                  mem/RCL 0
5:                        2.1
6:                  math/pow/y^x
7:                          *
8:                          7
9:                          -
10:                         5
11:                         /
12:                 mem/STO 1
13:                     clear
14:                      14.3
15:                 mem/RCL 1
16:                       1.2
17:                 math/pow/y^x
18:                         -
19:                         4
```

```
20:                    /
21:           mem/STO 2
22:                clear
23:           mem/RCL 0
24:           mem/RCL 1
25:                  x²
26:                   +
27:           mem/RCL 2
28:                  x²
29:                   +
30:                  14
31:                   -
32:           [prg end]
```

Graficamos esta función:

mode/prog/draw/y=f(x)/fx:



Vemos entonces que las soluciones se encuentran entre 1.7 y 1.75 (aproximadamente en 1.73) y entre 1.9 y 2.0 (aproximadamente 1.95), por lo que valores iniciales para el método de la Secante podrían ser 1.7 y 1.75.

Ahora modificamos el programa “ms” (Método de la Secante) para cambiar la precisión, el límite de iteraciones y los valores iniciales. Primero encontramos la solución comprendida entre 1.7 y 1.75.

El error, límite de iteraciones y valores iniciales son:

```
1:          0.000000000001
2:                40.00001
3:                     1.7
4:                    1.75
```

La función es:

```
61▸                prog/flow/LBL 0
62:           mem/STO 0
63:                clear
64:                    3
65:           mem/RCL 0
66:                  2.1
67:           math/pow/y^x
68:                    *
69:                    7
70:                    -
71:                    5
72:                    /
```

```
73:                    mem/STO 1
74:                        clear
75:                         14.3
76:                    mem/RCL 1
77:                          1.2
78:                   math/pow/y^x
79:                            -
80:                            4
81:                            /
82:                    mem/STO 2
83:                        clear
84:                    mem/RCL 0
85:                    mem/RCL 1
86:                          x²
87:                            +
88:                    mem/RCL 2
89:                          x²
90:                            +
91:                           14
92:                            -
93:                  prog/flow/RTN
94:                    [prg end]
```

Haciendo correr el programa se obtiene (mode/prog/run/ms):

```
                  1.72652614821958
```

Resultado que podemos comparar con el obtenido con “solve”:

1.7/ENTER; 1.75/ENTER; mode/prog/more/solve/fx:

```
                               1.7
                              1.75
                  1.72652614821958
```

Como se puede observar en esta ocasión ambos resultados concuerdan en todos los dígitos.

Ahora encontramos la solución comprendida entre 1.9 y 2. La única parte que cambia en el programa es la correspondiente a los datos:

```
3:                             1.9
4:                               2
```

Haciendo correr el programa se obtiene (mode/prog/run/ms):

```
                  1.943797085063069
```

Resultado que podemos comparar con el obtenido con “solve”:

1.9/ENTER; 2/ENTER; mode/prog/more/solve/fx:

```
                               1.9
                                 2
                  1.943797085063063
```

Que difiere del resultado calculado con el método de la Secante únicamente en el último dígito.

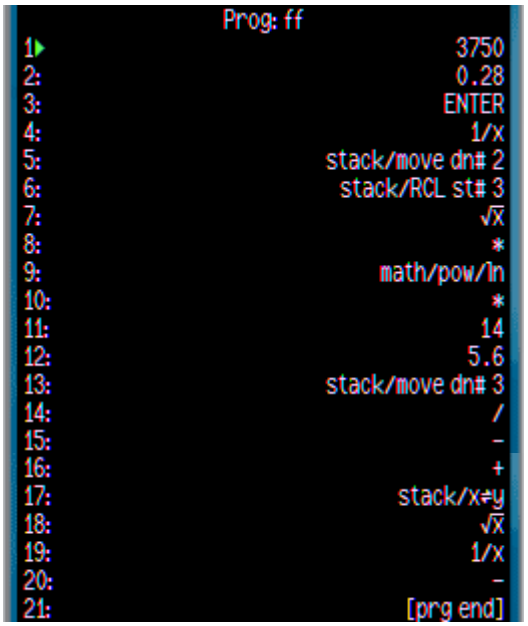
3. El factor de fricción *“f”* de tuberías que transportan suspensiones de partículas fibrosas, puede ser calculado con la ecuación de Lee – Duffy, que se presenta al pie de la pregunta. Donde “RE” es el número de Reynolds y “k” es una constante determinada por la concentración de la suspensión. Para una suspensión con una concentración del 0.08% k=0.28. Encuentre el valor de *“f”* por el método de la secante, con 11 dígitos de precisión y un límite de 40 iteraciones, para RE=3750. Estime los valores iniciales del método graficando la ecuación entre x=0, x=0.01, y=-5, y=5. Compare la solución encontrada con la obtenida con “solve”.

$$\frac{1}{\sqrt{f}}=\left(\frac{1}{k}\right)\ln\left(RE\sqrt{f}\right)+\left(14-\frac{5.6}{k}\right)$$

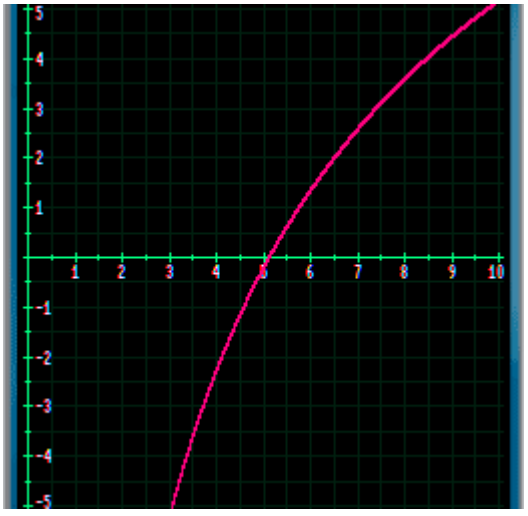
Primero escribimos la función en la forma f(*f*)=0:

$$f(f)=\left(\frac{1}{k}\right)\ln\left(RE\sqrt{f}\right)+\left(14-\frac{5.6}{k}\right)-\frac{1}{\sqrt{f}}=0$$

Y la programamos: (mode/prog/new/ff:)



Graficando la función entre los límites dados se obtiene:



Dado que los límites en x van desde 0 hasta 0.01, los valore de “x” de la gráfica están multiplicados por $1x10^{3}$, por lo tanto la solución aproximada es $5x10^{-3}$ (no, 5), o podemos decir que está comprendida entre $4x10^{-3}$ y $6x10^{-3}$. Emplearemos estos valores como valores iniciales del método de la Secante.

El error, número de iteraciones y valore iniciales son:

```
3:                    0.004
4:                    0.006
```

La función es:

```
61▶          prog/flow/LBL 0
62:                     3750
63:                     0.28
64:                    ENTER
65:                      1/x
66:        stack/move dn# 2
67:         stack/RCL st# 3
68:                      √x
69:                       *
70:          math/pow/ln
71:                       *
72:                       14
73:                      5.6
74:        stack/move dn# 3
75:                       /
76:                       -
77:                       +
78:          stack/x⇄y
79:                      √x
80:                      1/x
81:                       -
82:          prog/flow/RTN
83:              [prg end]
```

Haciendo correr el programa se obtiene:

```
0.005121893502614888
```

Con “solve” (y la función “ff”) se obtiene:

```
                    0.004
                    0.006
0.005121893502619243
```

Estos resultados son iguales en los primeros 12 dígitos. Como ya se dijo, en ingeniería por lo general es suficiente trabajar con unos 7 dígitos de precisión, por lo que el resultado obtenido tiene una precisión más que suficiente para la mayoría de las aplicaciones ingenieriles y en este caso en particular es por demás suficiente, pues normalmente este valor se lee de gráficas y en esos casos no es posible lograr una precisión superior a los 3 dígitos.

Observe que en la función se han escrito el número de Reynolds (RE) y la constante “k” al principio de la misma, para facilitar el hacer correr el programa con otros valores de “RE” y “k”. Así por ejemplo podemos calcular el factor de fricción cuando k=0.35 y RE=4700 y para demostrar, una vez más, que para el método de la secante no se requieren necesariamente valores a ambos lados de la solución (es decir no es imprescindible graficar previamente la función), emplearemos como valores iniciales 0.005 y 0.00051 (dos valores consecutivos):

```
3:                    0.005
4:                   0.0051
```

En la parte correspondiente a la función, sólo cambiamos los valores de “RE” y “k”:

```
61:          prog/flow/LBL 0
62:                     4700
63:                     0.35
```

Entonces haciendo correr el programa obtenemos:

```
0.004746596567615341
```

Por supuesto, para otros valores de “RE” y “k” no es necesario modificar los valores iniciales del método de la Secante, sino sólo cambiar los valores de “RE” y “k”.

Aún teniendo que modificar el programa (debido a que Calc-Java no permite llamar a funciones desde el interior de un programa), el proceso es mucho más rápido, confiable y preciso que leer los valores desde una gráfica, por lo que aún con estas limitaciones los métodos que hemos estudiado y que estudiaremos más adelante resultan de utilidad práctica.

Es posible también modificar la función de manera que los valores se almacenen directamente en posiciones de memoria y sean utilizados en el programa, evitando así la necesidad de editar y modificar el programa para calcular resultados con otros valores. Procediendo de esta manera, empleando las posiciones de memoria 0 y 1 para “RE” y “k” respectivamente, la función sería:

```
Prog: ms
61:                 prog/flow/LBL 0
62:                     mem/RCL 1
63:                           1/x
64:                     mem/RCL 0
65:                 stack/RCL st# 2
66:                            √x
67:                             *
68:                   math/pow/ln
69:                             *
70:                            14
71:                           5.6
72:                     mem/RCL 1
73:                             /
74:                             -
75:                             +
76:                   stack/x⇄y
77:                            √x
78:                           1/x
79:                             -
80:                   prog/flow/RTN
81:                     [prg end]
```

Entonces para calcular el factor de fricción para RE=4700 y k=0.35, guardamos previamente estos valores en memoria:

4700/special/mem/STO 0; 0.35/special/mem/STO 1

Y al hacer correr el programa “ms”, se obtiene, como era de esperar, el mismo resultado que con el programa previo:

```
0.004746596567615341
```

Claro que ahora resulta mucho más sencillo hacer correr el programa para otros valores de “RE” y “k”, sólo guardando nuevos valores en las posiciones 0 y 1 respectivamente.

4.2.3. Ejercicios

1. Resuelva el ejercicio 1, del tema 2, por el método de la Secante, con 14 dígitos de precisión, un límite de 40 iteraciones y estimando los valores iniciales en base a la gráfica de la función despejada.

2. Resuelva el ejercicio 2, del tema 2, por el método de la Secante con 10 dígitos de precisión, un límite de 60 iteraciones y estimando los valores iniciales en base a la gráfica de la función despejada.

3. Resuelva el ejercicio 3, del tema 2, por el método de la Secante con 9 dígitos de precisión, un límite de 50 iteraciones y estimando los valores iniciales en base a la gráfica de la función despejada.

4. Resuelva el ejercicio 4, del tema 2, por el método de la Secante con 8 dígitos de precisión, un límite de 100 iteraciones y estimando los valores iniciales en base a la gráfica de la función despejada.

5. Resuelva el ejercicio 5, del tema 2, por el método de la Secante con 11 dígitos de precisión, un límite de 70 iteraciones y estimando los valores iniciales en base a la gráfica de la función despejada.

6. Resuelva el ejercicio 6, del tema 2, por el método de la Secante con 9 dígitos de precisión, un límite de 80 iteraciones y estimando los valores iniciales en base a la gráfica de la función despejada.

5.ECUACIONES ALGEBRAICAS CON UNA INCÓGNITA - 3

5.1. Método de la Newton - Raphson

El método de Newton es probablemente el método más empleado en la práctica para resolver ecuaciones algebraicas con una incógnita. En este método se asume un valor inicial *(x_1)* y con el mismo se calcula la derivada de la función, que como se ve en la figura, es la tangente a la curva. Entonces en la intersección de la tangente con la recta *y=0* se determina el nuevo valor de prueba *(x_2)* y con el mismo el valor de la función (f(x_2)). Si el nuevo valor de la función es casi cero el proceso concluye, caso contrario se repite (empleando el nuevo valor de prueba) hasta que la función es casi cero (dentro de un margen de tolerancia especificado), o hasta que los dos últimos valores de prueba (x_1 y x_2) son iguales en un determinado número de dígitos.

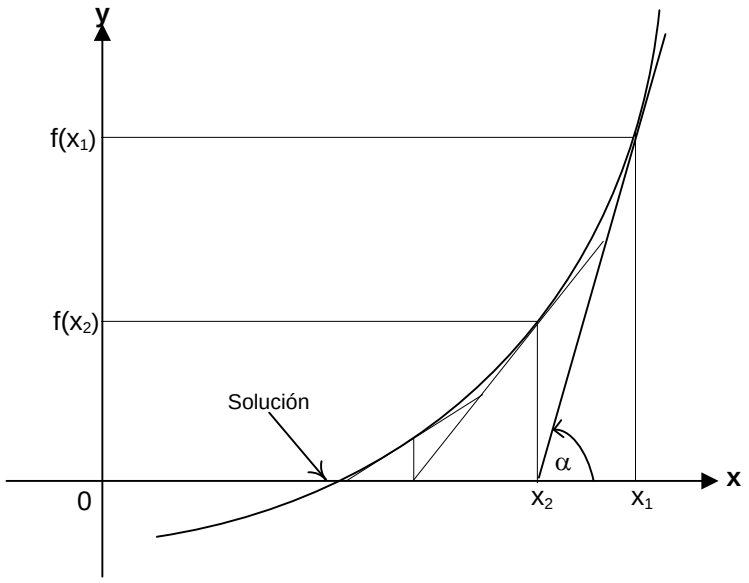


Figura 5.1. Método de Newton - Raphson

La ecuación que permite calcular el nuevo valor de prueba *(x_2)* puede ser deducida del triángulo que forman la tangente con los segmentos *(x_1 -> x_2)* y *(0->f_1)*:

$$\tan(\alpha) = f'(x_1) = \frac{f(x_1) - 0}{x_1 - x_2}$$

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

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} \qquad (5.1)$$

Como se puede observar, la ecuación es sencilla, pero involucra el cálculo de la derivada de la función. Ya vimos que Calc-Java permite calcular la derivada numérica de una función, sin embargo, como también vimos, Calc-Java no permite llamar a una función desde el interior de otra, lo que impide aprovechar en la práctica esta función de la calculadora.

El cálculo de la derivada analítica es un proceso que suele consumir mucho tiempo (en muchos casos más tiempo que el requerido para resolver la ecuación en sí), por ello en la práctica sólo se emplea la derivada analítica cuando la función se encuentra de forma explícita es resuelta frecuentemente. En esos casos se calcula la derivada analítica por separado (una sola vez) y se la reemplaza en la ecuación de Newton - Raphson, conjuntamente la función, obteniendo así la expresión analítica para la función en particular.

Por ejemplo, para calcular la raíz cuadrada de un número “n” por el método de Newton - Raphson, debemos resolver la siguiente función:

$$x = \sqrt{n}$$

Donde "x" es la raíz cuadrada del número "n". Elevamos ambos términos al cuadrado y llevamos esta función a la forma "f(x)=0":

$$f(x) = x^2 - n = 0$$

Cuya derivada es:

$$f'(x) = 2x$$

Aplicando la ecuación del método Newton (ecuación 5.1) a esta ecuación resulta la expresión:

$$x_2 = x_1 - \frac{x_1^2 - n}{2x_1} = \frac{2{x_1}^2 - {x_1}^2 + n}{2x_1} = \frac{{x_1}^2 + n}{2x_1}$$

$$x_2 = \frac{1}{2}\left(x_1 + \frac{n}{x_1}\right) \qquad (5.2)$$

Que es la ecuación de Newton-Raphson para el cálculo de la raíz cuadrada de un número real “n”. Para resolver esta ecuación (y cualquier otra deducida con el método de Newton-Raphson) el algoritmo es:

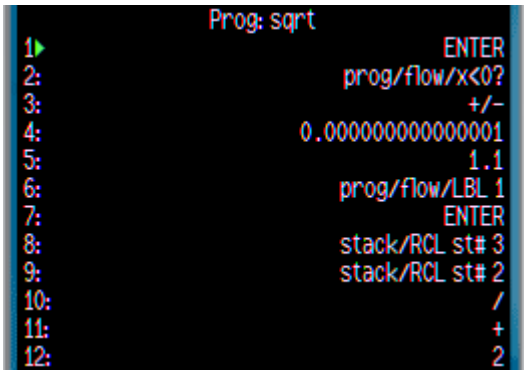
a) Se asume un valor inicial para la solución: x_1

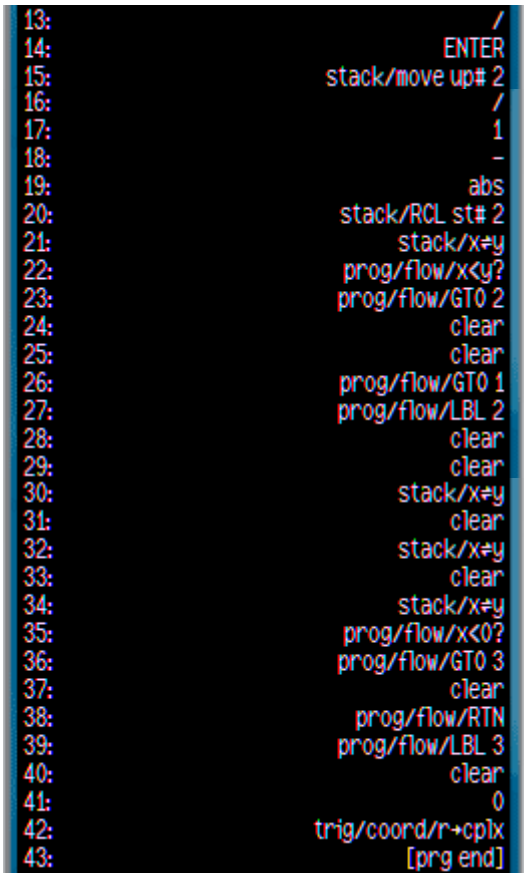
b) Con el valor asumido y la ecuación deducida (en el ejemplo 5.2) se calcula el nuevo valor de la solución: x_2

c) Si x_1 y x_2 son aproximadamente iguales el proceso concluye, siendo la solución x_2.

d) x_2 toma el valor de x_1 y se repite el proceso desde el paso (b).

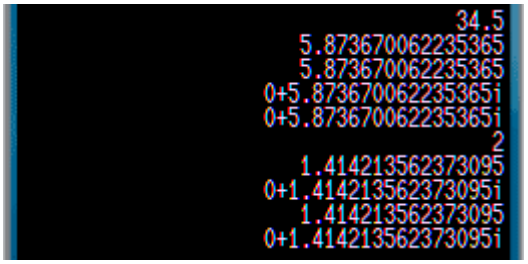
Siguiendo, este algoritmo, pero tomando en cuenta además que el resultado puede ser real o imaginario, el programa para calcular la raíz cuadrada de un número, con 15 dígitos de precisión (casi el valor exacto), siendo el valor inicial asumido “1.1” es el siguiente:





En la primera parte de este programa se duplica el número cuya raíz se quiera calcular (ENTER) y si la copia es negativa (x<0) se le cambia el signo (+/-), dejando el número original en la parte superior de la pila. Se procede así para trabajar siempre con números positivos, pues con números negativos (soluciones imaginarias) el método no logra convergencia. Luego se fija el error (15 dígitos) y desde el paso 5 se siguen los pasos descritos en el algoritmo hasta el paso 26. A partir del paso 27 se borran los datos adicionales que quedan en la pila (como los dos últimos valore comparados) hasta que queda el número original y el valor de la raíz. Entonces en el paso 35 se pregunta si el número original era negativo y de ser así se crea el resultado imaginario (a partir del paso 40), caso contrario simplemente se borra el número y se devuelve el resultado (pasos 37 y 38).

Haciendo correr el programa con ±34.5 y ±2 y corroborando los resultados con la función raíz cuadrada de Calc-Java se obtiene:



Y como se puede ver se obtienen en ambos casos los mismos resultados que con la función de la calculadora.

Sin embargo, y como ya se mencionó, el cálculo de la derivada analítica suele requerir mucho tiempo y en algunos casos es inclusive imposible de

derivar, por ello el anterior procedimiento sólo se justifica para funciones de uso muy frecuente (como es el caso de la raíz cuadrada).

Para los casos más frecuentes, la forma más práctica de obtener la derivada requerida por el método de Newton – Raphson es mediante la derivación numérica.

5.1.1. Cálculo numérico de la derivada

El cálculo numérico de las derivadas se efectúa empleando fórmulas que son deducidas en base al concepto de la derivada:

$$\frac{dy}{dx} = \lim_{\Delta x \to 0}\left(\frac{\Delta y}{\Delta x}\right) \qquad (3.19)$$

Donde numéricamente se consigue una aproximación al límite empleando valores de *"Δx"* pequeños. Básicamente se toman valores consecutivos de *"x"*, los que constituyen el incremento *"Δx"* y para estos valores se calculan los correspondientes valores de la función, los que constituyen los valores de *"Δy"*.

Así por ejemplo para calcular la derivada primera para un determinado valor de "x", podemos tomar dos valores cercanos a "x": uno ubicado al lado izquierdo: "x-h" y otro al lado derecho: "x+h", donde "h" es un valor pequeño (una fracción de x). Entonces al sustituir en la ecuación (3.19) (asumiendo que con este valor pequeño de "h" se alcanza el límite), obtenemos:

$$\frac{d}{dx}y \approx \left(\frac{f(x+h)-f(x-h)}{(x+h)-(x-h)}\right) = \frac{f(x+h)-f(x-h)}{2h}$$

Que es la fórmula de "diferencia central" para calcular la derivada primera.

Si en lugar de tomar dos valores a ambos lados de "x" se toma uno al lado izquierdo: "x-h" se tiene:

$$\frac{d}{dx}y \approx \left(\frac{f(x)-f(x-h)}{(x)-(x-h)}\right) = \frac{f(x)-f(x-h)}{h}$$

Que es la fórmula de "diferencia hacia atrás" para calcular la deriva primera.

Si en lugar de tomar un valor al lado izquierdo de "x" se toma uno al lado derecho: "x+h", se tiene:

$$\frac{d}{dx}y \approx \left(\frac{f(x+h)-f(x)}{(x+h)-(x)}\right) = \frac{f(x+h)-f(x)}{h}$$

Que es la fórmula de "diferencia hacia adelante" para calcular la derivada primera.

En general mientras más valores se tomen a ambos lados de "x" más exacto es el resultado calculado, es por ello que de estas tres fórmulas, la más exacta es la de diferencia central, porque toma dos valores.

La deducción de las fórmulas para las derivadas de segundo, tercer y otros órdenes superiores sigue el mismo principio, pero en lugar de la función original se emplean las fórmulas para las derivadas de orden inferior deducidas previamente. Así para obtener la derivada segunda se emplea las fórmulas de la derivada primera, para la tercer las fórmulas de la derivada segunda y así sucesivamente.

En función al número de valores que se toman alrededor de "x" se tienen diferentes tipos de errores: de primer orden si sólo se toma un valor a la derecha o a la izquierda (diferencia hacia adelante o hacia atrás); de segundo orden si se toman dos valores a ambos lados de "x" (diferencia central) y de cuarto orden si se toman 4 valores: 2 a la izquierda y 2 a la derecha de "x".

En el método de Newton – Raphson sólo necesitamos las fórmulas para las derivadas de primer orden, sin embargo, para uso posterior presentamos a continuación las fórmulas para calcular las derivadas hasta de cuarto orden.

5.1.1.1. Fórmulas de diferencia central. Error de orden h^2

$$
\begin{aligned}
f'(x) &= \frac{f(x+h) - f(x-h)}{2h} \\
f''(x) &= \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} \\
f'''(x) &= \frac{f(x+2h) - 2f(x+h) + 2f(x-h) - f(x-2h)}{2h^3} \\
f''''(x) &= \frac{f(x+2h) - 4f(x+h) + 6f(x) - 4f(x-h) + f(x-2h)}{h^4}
\end{aligned}
\qquad (5.3)
$$

5.1.1.2. Fórmulas de diferencia central. Error de orden h^4

$$
\begin{aligned}
f'(x) &= \frac{-f(x+2h) + 8f(x+h) + 8f(x-h) + f(x-2h)}{12h} \\
f''(x) &= \frac{-f(x+2h) + 16f(x+h) + 30f(x) + 16f(x-h) - f(x-2h)}{12h^2} \\
f'''(x) &= \frac{f(x+3h) + 8f(x+2h) - 13f(x+h) + 13f(x-h) - 8f(x-2h) + f(x-3h)}{8h^3} \\
f''''(x) &= \frac{-f(x+3h) + 12f(x+2h) - 39f(x+h) + 56f(x) - 39f(x-h) - 12f(x-2h) - f(x-3h)}{6h^4}
\end{aligned}
\qquad (5.4)
$$

5.1.1.3. Fórmulas de diferencia hacia adelante. Error de orden h

$$
\begin{aligned}
f'(x) &= \frac{f(x+h) - f(x)}{h} \\
f''(x) &= \frac{f(x+2h) - 2f(x+h) + f(x)}{h^2} \\
f'''(x) &= \frac{f(x+h) - 3f(x+2h) + 3f(x+h) - f(x)}{h^3} \\
f''''(x) &= \frac{f(x+4h) - 4f(x+3h) + 6f(x+2h) - 4f(x+h) + f(x)}{h^4}
\end{aligned}
\qquad (5.5)
$$

5.1.1.4. Fórmulas de diferencia hacia adelante. Error de orden h^2

$$
\begin{aligned}
f'(x) &= \frac{-f(x+2h) + 4f(x+h) - 3f(x)}{2h} \\
f''(x) &= \frac{-f(x+3h) + 4f(x+2h) - 5f(x+h) + 2f(x)}{h^2}
\end{aligned}
$$

$$f'''(x)=\frac{-3f(x+4h)+14f(x+3h)-24f(x+2h)+18f(x+h)-5f(x)}{2h^3}$$
$$f''''(x)=\frac{-2y_{i+5}+11y_{i+4}-24y_{i+3}+26y_{i+2}-14y_{i+1}+3y_i}{h^4} \qquad (5.6)$$

5.1.1.5. Fórmulas de diferencia hacia atrás. Error de orden h

$$f'(x)=\frac{f(x)-f(x-h)}{h}$$
$$f''(x)=\frac{f(x)-2f(x-h)+f(x-h)}{h^2}$$
$$f'''(x)=\frac{f(x)-3f(x-h)+4f(x-2h)-f(x-3h)}{h^3} \qquad (5.7)$$
$$f''''(x)=\frac{f(x)-5f(x-h)+6f(x-2h)-4f(x-3h)+f(x-4h)}{h^4}$$

5.1.1.6. Fórmulas de diferencia hacia atrás. Error de orden h^2

$$f'(x)=\frac{3f(x)-4f(x-h)+f(x-2h)}{2h}$$
$$f''(x)=\frac{2f(x)-5f(x-h)+4f(x-2h)-f(x-3h)}{h^2}$$
$$f'''(x)=\frac{5f(x)-18f(x-h)+24f(x-2h)-14f(x-3h)+3f(x-4h)}{2h^3} \qquad (5.8)$$
$$f''''(x)=\frac{3f(x)-14f(x-h)+26f(x-2h)-24f(x-3h)+11f(x-4h)-2f(x-5h)}{h^4}$$

En teoría, cuanto mayor es el orden de la fórmula más exacto es el resultado, no obstante, en la práctica las fórmulas de orden elevado implican un mayor número de operaciones, lo que a su vez implica una mayor acumulación de error debido al redondeo. Por esta razón en la práctica suelen emplearse las fórmulas de primer o segundo orden. Igualmente en teoría, mientras menor sea el valor de *"h"* más exacto es el resultado (porque más cerca está del límite 0), una vez más esto no es cierto debido a los errores de redondeo, pues mientras más pequeño el número mayor es el error de redondeo cometido.

Para el método de Newton-Raphson emplearemos la fórmula de diferencia central para el cálculo de la derivada numérica. Como incremento *"h"* emplearemos el valor de "x" multiplicado por $1x10^{-6}$, es decir: $h = x*10^{-6}$, valor que por lo general es lo suficientemente pequeño como para permitir un cálculo bastante aproximado de la derivada.

5.2. Algoritmo del método de Newton - Raphson

La lógica del método de la Newton - Raphson es la siguiente:

a) Se fija el error permitido: err y el límite de iteraciones: li.

b) Se asumen un valor inicial para la solución: x_1.

c) Con el valor asumido (x_1) se calcula el valor de la función: $f(x_1)$.

d) Si el valor de la función $f(x_1)$ es casi cero, es decir si $|f(x_1)|<err$, ya se ha encontrado la solución (x_1), por lo que el proceso concluye y se devuelve la solución: x_1.

e) Con la fórmula de la derivada central de segundo orden (5.3) se calcula el valor de la derivada numérica: $f'(x_1)$.

f) Con la ecuación 5.1 se calcula el nuevo valor de prueba: x_2.

g) Si x_2 y x_3 son aproximadamente iguales, es decir si: $|x_2/x_3-1|<err$, se ha encontrado la solución (x_2), por lo que el proceso concluye y se devuelve la solución: x_2.

h) Se disminuye el límite de iteraciones en 1 (li=li-1) y si su valor es cero (li=0), el proceso concluye generando un error y devolviendo el último valor de prueba calculado: x_2.

i) Se intercambian variables: $x_1=x_2$ y se repite el proceso desde el paso c.

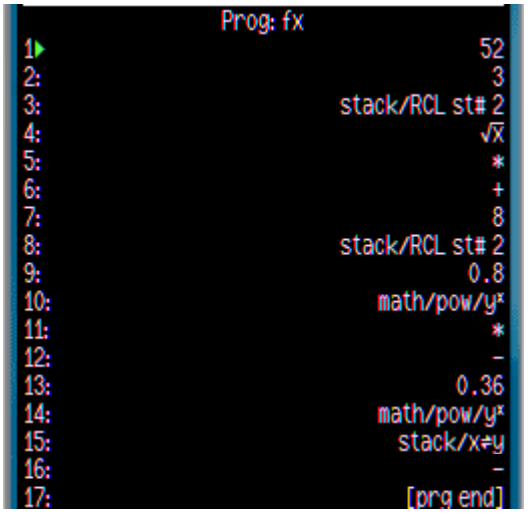
j) Se devuelve la solución (o raíz) de la función: x_3.

Para comprender mejor el algoritmo resolveremos la siguiente expresión (resuelta en el tema anterior), siguiendo los pasos descritos previamente.

$$f(x)=\left(52+3\sqrt{x}-8x^{0.8}\right)^{0.36}-x=0$$

En este caso, no se requiere realizar ninguna operación adicional, pues la ecuación está ya en la forma f(x)=0.

Comenzamos entonces programando la función: 1.1/mode/prog/new/<>/fx (donde 1.1 es el valor de prueba):

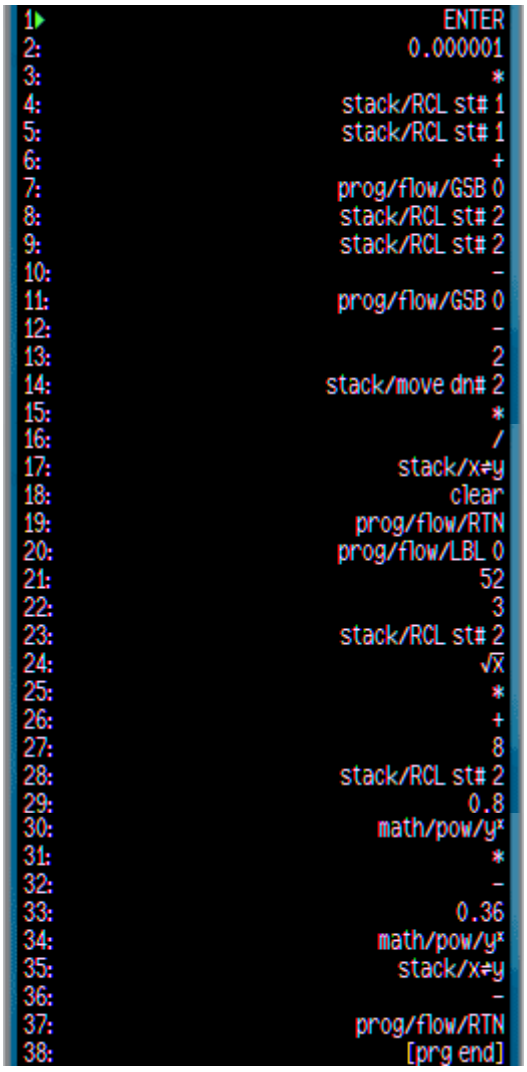


Entonces asumimos un valor inicial (x_1), en lo posible cercano a la solución, pero en este y sólo para demostrar que el método puede converger inclusive con valores no muy cercanos a la solución, emplearemos 1.1. Calculamos entonces el valor de la función:

1.1/ENTER; ENTER; mode/prog/run/fx;

Como este valor no es cercano a cero, calculamos el valor de la derivada numérica, pero como este cálculo debe ser hecho varias veces, es conveniente programar el mismo, lo que implica (debido a que Calc-Java no permite llamar a una función desde otra) volver a programar la función:

mode/prog/new/<>/df:



Podemos verificar qué tan preciso es el resultado calculado con la fórmula de diferencia central, calculando la derivada en el punto 1.1: 1.1/mode/prog/run/df:

En el mismo punto, el valor calculado con “diff” de Calc-Java es: 1.1/mode/prog/more/diff/fx:

El resultado más exacto para esta derivada, calculado con Mathematica, es: -1.1495224506855255 y como se puede observar, tanto el valor calculado con la fórmula de diferencia central como el calculado con “diff” son exactos en los primeros 13 dígitos, precisión que por lo general resulta ser más que suficiente en la mayoría de los cálculos ingenieriles, por lo que la fórmula de diferencia central puede ser empleada con relativa seguridad en la mayoría de los cálculos ingenieriles.

Prosiguiendo con el método, y una vez calculado el valor de la derivada, borramos los valores adicionales (los valore devueltos por “diff”), de manera que queden en la pila: x_1, $f(x_1)$ y $f'(x_1)$:

Con estos valores calculamos entonces el nuevo valor de x (x_2) aplicando la ecuación 5.1, es decir realizamos las operaciones: /; -; con lo que obtenemos el nuevo valor de "x" (x_2):

Como evidentemente este valor es muy diferente al inicialmente asumido (1.1) repetimos el proceso. Volvemos a calcular el valor de la función: ENTER/mode/prog/run/fx:

Dado que dicho valor todavía no es cero calculamos la derivada: special/stack/more/RCL st#/1; mode/prog/run/df:

Y con estos valores volvemos a calcular el nuevo valor de x: /; -:

Que ya es igual en tres dígitos al valor de la anterior iteración, pero que todavía no tiene una precisión aceptable. Por lo que repetimos el proceso una vez más, siendo el nuevo valor de la función:

Que ya tiene una exactitud de 8 dígitos. El valor de la derivada es:

Y el nuevo valor de x:

Que ya es igual al valor anterior en 9 dígitos, es decir que ya tiene una precisión de 9 dígitos, por lo que el proceso podría concluir en esta parte, sin embargo, repitiendo el proceso una vez más, el valor de la función es:

Lo que nos indica que el resultado es exacto. El nuevo valor de la derivada es:

Pero como la función es 0 (y 0 entre cualquier otro valor es cero), el nuevo valor de x es igual al anterior:

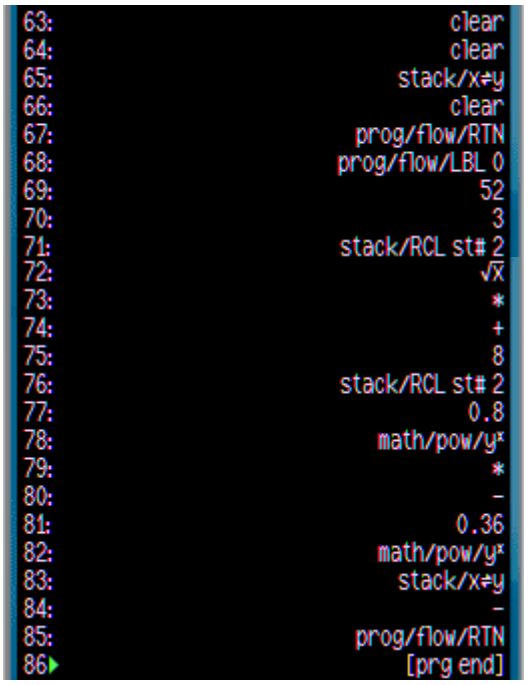
Como era de esperar, pues el resultado de la anterior iteración ya era el exacto (el valor de la función para el mismo es 0).

Como se puede ver el método de Newton-Raphson logra converger hacia el resultado en tan solo 3 iteraciones. Esta es una característica de este método y la razón por la cual es el método más empleado en la práctica (a pesar del inconveniente que representa el cálculo de la derivada).

5.2.1. Programa

El programa elaborado, siguiendo el algoritmo descrito y ejemplificado en el anterior acápite es el siguiente: mode/prog/new/<>/nr:

```
Prog: nr
1▶                0.000000000001
2:                         50.00001
3:                              1.1
4:                       stack/x≠y
5:                    mem/STO 15
6:                         clear
7:                  prog/flow/LBL 1
8:                         ENTER
9:                  prog/flow/GSB 0
10:                 stack/RCL st# 2
11:                 stack/RCL st# 1
12:                         abs
13:                 prog/flow/x<y?
14:                 prog/flow/GTO 2
15:                        clear
16:                        clear
17:                 stack/RCL st# 1
18:                      0.000001
19:                             *
20:                 stack/RCL st# 2
21:                 stack/RCL st# 1
22:                             +
23:                 prog/flow/GSB 0
24:                 stack/RCL st# 3
25:                 stack/RCL st# 2
26:                             -
27:                 prog/flow/GSB 0
28:                             -
29:                             2
30:               stack/move dn# 2
31:                             *
32:                             /
33:                             /
34:                 stack/RCL st# 1
35:                       stack/x≠y
36:                             -
37:                        ENTER
38:               stack/move up# 2
39:                 stack/RCL st# 1
40:                       stack/x≠y
41:                             /
42:                             1
43:                             -
44:                           abs
45:                 stack/RCL st# 3
46:                       stack/x≠y
47:                 prog/flow/x<y?
48:                 prog/flow/GTO 2
49:                        clear
50:                        clear
51:                        clear
52:                 prog/flow/DSE 15
53:                 prog/flow/GTO 1
54:                             0
55:                             0
56:                             /
57:               stack/move up# 2
58:                             0
59:                             0
60:                             0
61:                 prog/flow/LBL 2
62:                        clear
```



En este programa, los pasos 1 al 3 corresponden a los datos: error, límite de iteraciones y valor inicial asumido. En los pasos 4 al 6 se guarda en la memoria 15 el límite de iteraciones. A partir del paso 7 hasta el paso 67 se codifica el método propiamente. Finalmente a partir del paso 68 se programa la función que a resolver.

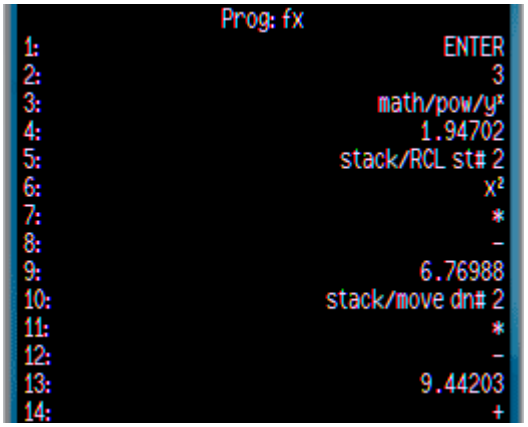
Haciendo correr el programa se obtiene: (mode/prog/run/nr), se obtiene:

Que como era de esperar es el mismo resultado obtenido paso a paso en el acápite anterior.

5.2.2. Ejemplos

1. Como primer ejemplo, emplearemos el programa elaborado para resolver la función: $f(x) = x^3 - 1.94702x^2 - 6.76988x + 9.44203 = 0$, con un error de 10 dígitos y un límite de 30 iteraciones. Para estimar los valores iniciales graficaremos la ecuación entre x=-5, x=5, y=-5, y=5.

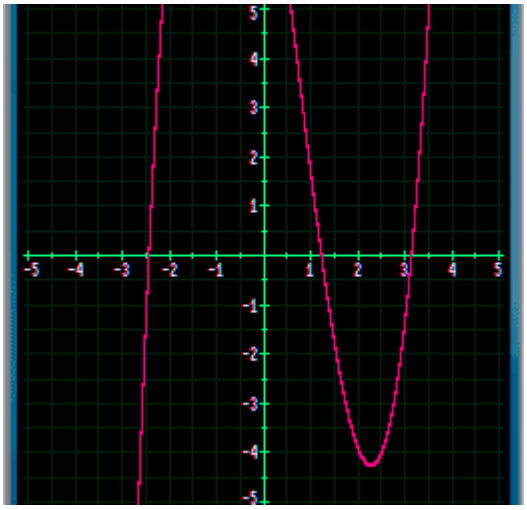
Primero programamos la función para graficarla y ver así el lugar de las soluciones: mode/prog/new/fx:



```
15▸                    [prg end]
```

Graficando la función entre los límites dados se obtiene:

-5/ENTER; 5/ENTER; -5/ENTER; 5/ENTER; mode/prog/draw/y=f(x)/fx:



Como se puede observar existen tres soluciones: una alrededor de -2.5, otra alrededor de 1.2 y otra alrededor de 3.1. Para encontrar estas tres soluciones hacemos correr el programa con cada uno de estos tres valores. Para la primera raíz cambiamos además el error y el límite de iteraciones:

```
                Prog: nr
1▸                0.0000000001
2:                     30.00001
3:                          -2.5
```

Programamos la función a partir de la etiqueta 0:

```
67▸                  prog/flow/RTN
68:              prog/flow/LBL 0
69:                         ENTER
70:                             3
71:                  math/pow/y^x
72:                       1.94702
73:              stack/RCL st# 2
74:                           x²
75:                             *
76:                             -
77:                       6.76988
78:              stack/move dn# 2
79:                             *
80:                             -
81:                       9.44203
82:                             +
83:                  prog/flow/RTN
84:                    [prg end]
```

Haciendo correr el programa (mode/prog/run/nr) se obtiene:

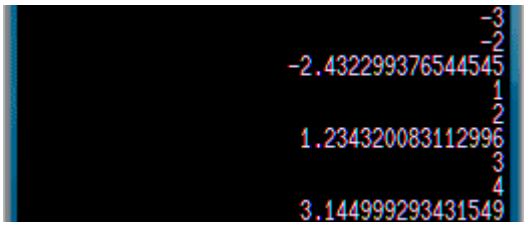
```
                -2.43229937654596
```

Que sería la primera solución. Cambiando el valor del paso 3 a 1.2 y haciendo correr el programa se obtiene:

```
                 1.234320083112996
```

Que es la segunda solución. Cambiando el valor del paso 3 a 3.5 y haciendo correr el programa se obtiene:

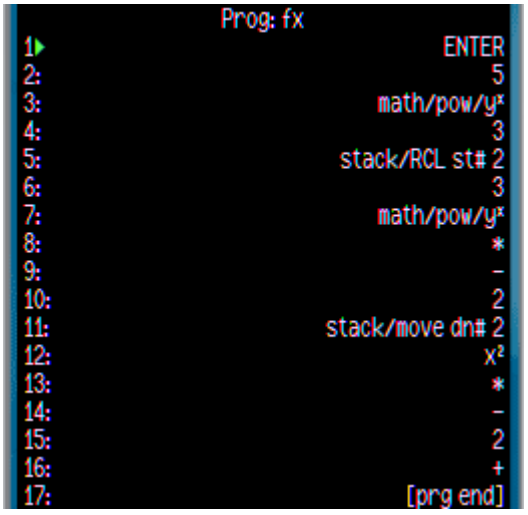
Que sería la tercera solución. Podemos comparar estos resultados con los obtenidos con "solve" de Calc-Java:



Tal como se puede ver, en todos los casos los resultados obtenidos con Newton Raphson tienen un error menor al especificado (10 dígitos).

2. Como segundo ejemplo encontraremos las soluciones de la función: $f(x)=x^5-3x^3-2x^2+2=0$, con un error de 12 dígitos y un límite de 40 iteraciones. Para estimar los valores iniciales graficaremos la ecuación entre: x=1.6, x=2.1, y=-0.5, y=1.

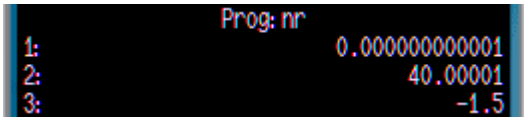
Programamos la función, para graficarla y obtener así los valores iniciales: mode/prog/new/fx



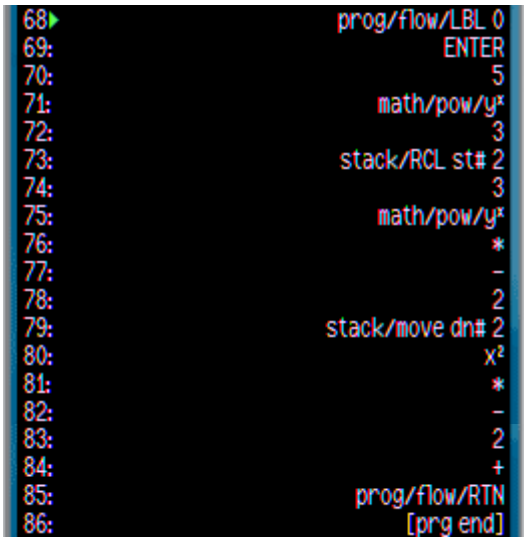
Graficando la función resulta: mode/prog/draw/y=f(x)/fx:

Y al igual que el caso anterior se tienen tres soluciones: la primera cerca a -1.5, la segunda cerca a 0.7 y la tercera cerca a 1.9.

Modificamos los datos del programa "nr" (con el valor inicial para la primera raíz):



Cambiamos la función a resolver:

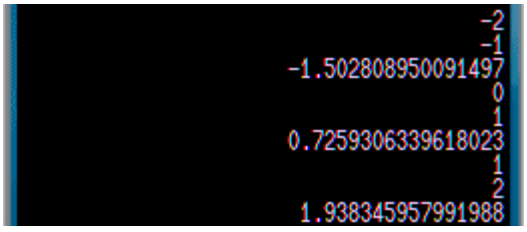


Haciendo correr el programa se obtiene:

Que sería la primera solución. Para la segunda solución hacemos correr el programa con 0.7 (modificando el valor del paso 3), de esa forma se obtiene:

Y para la tercera solución hacemos correr el programa con 1.9, modificando el valor del paso 3:

Con "solve" se obtienen los siguientes resultados:



Una vez más las soluciones obtenidas con Newton – Raphson tienen mayor exactitud que la especificada. Solo el segundo resultado difiere del obtenido con "solve".

3. La ecuación de Beattie – Bridgeman, que se presenta al pie de la pregunta, permite calcular con bastante confiabilidad, el volumen de fluidos para los cuales se conocen las constantes: A_0, B_0, a, b y c. En esta ecuación "V" es el volumen del fluido (en litros), "P" es la presión (en atm) y "T" la temperatura (en K). Para el isobutano, las constantes

de la ecuación de Beattie – Bridgeman son: "A_0=16.6037"; "B_0=0.2354"; "a=0.11171"; "b=0.07697" y "c= 300x10^4". Elabore un programa que reciba como datos (en la pila) la temperatura "T" y la presión "P" y que devuelva el correspondiente volumen "V" del isobutano, calculado con el método de Newton Raphson con 12 dígitos de precisión. El valor inicial para el método de Newton Raphson debe ser calculado con la ecuación del gas ideal, es decir: V=(R*T)/P.

$$V = \left(RT + \frac{\beta}{V} + \frac{\gamma}{V^2} + \frac{\delta}{V^3} \right) \frac{1}{P}$$

$$\beta = RTB_0 - A_0 - \frac{Rc}{T^2}$$

$$\gamma = -RTB_0 b + aA_0 - \frac{RB_0 c}{T^2}$$

$$\delta = \frac{RB_0 bc}{T^2}$$

$$R = 0.0827 \frac{atm.l}{K.gmol}$$

Primero escribimos la ecuación del método de Beattie – Bridgeman en la forma f(V)=0:

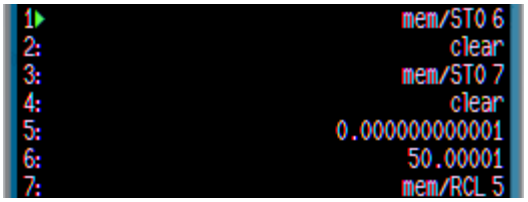
$$f(V) = \left(RT + \frac{\beta}{V} + \frac{\gamma}{V^2} + \frac{\delta}{V^3} \right) \frac{1}{P} - V$$

Para posibilitar que el programa sea empleado con otros compuestos diferentes al isobutano, las constantes de la ecuación de Beattie – Bridgeman se guardarán en las memorias 0 al 4, en el orden: A_0 = mem 0; B_0 = mem 1; a = mem 2; b = mem 3; c = mem 4. Igualmente, por conveniencia, se guardará el valor de la constante "R" en la posición de memoria 5. Dichos valores deben ser guardados en esas posiciones de memoria antes de llamar al método:

```
16.6037/special/mem/STO 0

0.2354/special/mem/STO 1

0.11171/special/mem/STO 2

0.07697/special/mem/STO 3

300e4/special/mem/STO 4

0.0827/special/mem/STO 5
```

Tal como se especifica en el enunciado, los valores de "T" y "P" deben estar en las posiciones 1 y 0 de la pila al momento de llamar al programa, pero para evitar el cometer errores al trabajar con la pila (pues el método de Newton – Raphson modifica la misma), se guardarán internamente en las posiciones de memoria 7 y 6 respectivamente.

Las partes del programa de Newton – Raphson que han sido modificadas para resolver este problema son las siguientes (los números de los pasos respectivos cambian porque lo primero que se hace es guardar los valores de "P" y "T" en las memorias 6 y 7):



```
8:                          mem/RCL 7
9:                                  *
10:                         mem/RCL 6
11:                                 /
12:                       stack/x⇄y
```

La parte correspondiente a la función de Beattie – Bridgeman es:

```
76:                     prog/flow/LBL 0
77:                         mem/RCL 5
78:                         mem/RCL 7
79:                                 *
80:                             ENTER
81:                         mem/RCL 1
82:                                 *
83:                         mem/RCL 0
84:                                 -
85:                         mem/RCL 5
86:                         mem/RCL 4
87:                                 *
88:                         mem/RCL 7
89:                                x²
90:                                 /
91:                                 -
92:                     stack/RCL st# 2
93:                                 /
94:                                 +
95:                         mem/RCL 5
96:                         mem/RCL 7
97:                                 *
98:                         mem/RCL 1
99:                                 *
100:                        mem/RCL 3
101:                                *
102:                              +/-
103:                        mem/RCL 2
104:                        mem/RCL 0
105:                                *
106:                                +
107:                        mem/RCL 5
108:                        mem/RCL 1
109:                                *
110:                        mem/RCL 4
111:                                *
112:                        mem/RCL 7
113:                               x²
114:                                /
115:                                -
116:                    stack/RCL st# 2
117:                               x²
118:                                /
119:                                +
120:                        mem/RCL 5
121:                        mem/RCL 1
122:                                *
123:                        mem/RCL 3
124:                                *
125:                        mem/RCL 4
126:                                *
127:                        mem/RCL 7
128:                               x²
129:                                /
130:                    stack/RCL st# 2
131:                                3
```

```
132:                    math/pow/y^x
133:                             /
134:                             +
135:                       mem/RCL 6
136:                             /
137:                       stack/x<>y
138:                             -
139:                    prog/flow/RTN
140:                      [prg end]
```

Para valores de T=408 (K) y P=36 (atm):

```
                                408
                                 36
```

Haciendo correr el programa se obtiene:

```
                  0.3886737246529363
```

Que es el volumen (en lit/mol) del isobutano a una temperatura de 408 K y una presión de 36 atm.

Para T=500 K y P=36 atm:

```
                                500
                                 40
```

Se obtiene:

```
                  0.8291979488105284
```

Para calcular el volumen molar de otros componentes diferentes al isobutano, lo único que se debe hacer es cambiar las constantes (memorias 0 a 4) por las del compuesto respectivo y hacer correr el programa.

5.2.3. Ejercicios

1. Aplicando el método de Newton-Raphson, deduzca la ecuación para el cálculo de la raíz cúbica de un número. Luego programa la ecuación resultante de manera que el resultado sea calculado con 14 dígitos de precisión. Compare los resultados del programa con los obtenidos con la función existente en Calc-Java.
2. Aplicando el método de Newton-Raphson, deduzca la ecuación para el cálculo de la raíz enésima (y) de un número real (x). Luego programe la ecuación resultante para calcular la raíz enésima de cualquier número real “x” con 14 dígitos de precisión. Compare los resultados del programa con los obtenidos con la función existente en Calc-Java.
3. Resuelva el ejercicio 1, del tema 2, por el método de Newton - Raphson, con 14 dígitos de precisión, un límite de 40 iteraciones y estimando los valores iniciales en base a la gráfica de la función despejada.
4. Resuelva el ejercicio 2, del tema 2, por el método de Newton - Raphson con 10 dígitos de precisión, un límite de 60 iteraciones y estimando los valores iniciales en base a la gráfica de la función despejada.
5. Resuelva el ejercicio 3, del tema 2, por el método de Newton - Raphson con 9 dígitos de precisión, un límite de 50 iteraciones y estimando los valores iniciales en base a la gráfica de la función despejada.
6. Resuelva el ejercicio 4, del tema 2, por el método de Newton – Raphson con 8 dígitos de precisión, un límite de 100 iteraciones y estimando los valores iniciales en base a la gráfica de la función despejada.

7. Resuelva el ejercicio 5, del tema 2, por el método de la Newton - Raphson con 11 dígitos de precisión, un límite de 70 iteraciones y estimando los valores iniciales en base a la gráfica de la función despejada.

8. Resuelva el ejercicio 6, del tema 2, por el método de la Newton - Raphson con 9 dígitos de precisión, un límite de 80 iteraciones y estimando los valores iniciales en base a la gráfica de la función despejada.

6. ECUACIONES POLINOMIALES – 1

Con los métodos estudiados en los temas anteriores se puede encontrar (una a la vez) las soluciones reales de la ecuación polinomial de la forma:

$$P_n(x) = a_1 x^n + a_2 x^{n-1} + a_3 x^{n-2} + a_4 x^{n-3} + ... + a_n x + a_{n+1} = 0 \qquad (6.1)$$

Para encontrar todas las soluciones: reales e imaginarias, existen otros métodos algunos de los cuales estudiaremos en este tema.

El propósito del presente tema es que al concluir el mismo estén capacitados para encontrar las soluciones reales e imaginarias ecuaciones algebraicas polinomiales.

6.1. División sintética

La división sintética es la forma más eficiente de calcular el valor de un polinomio y el de su derivada, siendo esa la razón por la cual se la emplea como parte de algunos métodos que resuelven ecuaciones polinomiales, como el método de Newton.

Cuando se divide un polinomio entre un monomio que está en la forma: "$x-x_1=0$" (o lo que es lo mismo "$x = x_1$"), donde "x_1" es un número real, el residuo de la división es el valor del polinomio para "x" igual a "x_1", es decir es el valor de la función cuando se reemplaza "x_1". Si se vuelve a dividir el cociente de esta división entre "$x-x_1$", el residuo de esta nueva división es la derivada primera de la función para "x" igual a "x_1".

Así por ejemplo para calcular el valor del siguiente polinomio, así como el de su derivada, para un valor de "x" igual a "2":

$$P_3(x) = x^3 + x^2 - 3x - 3 = 0$$

Llevamos a cabo dos divisiones sintéticas consecutivas, tal como se muestra a continuación:

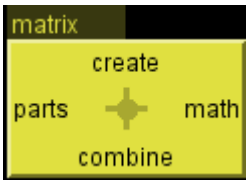
$x_1 = 2$	1	1	-3	-3
		2	6	6
	1	3	3	$\boxed{3}$ ← valor de la función: f(2)
		2	10	
	1	5	$\boxed{13}$	← derivada de la función: f'(2)

Como puede observar en este ejemplo, si "a" son los coeficientes del polinomio original, los coeficientes resultantes de la división sintética: "b" se calculan con:

$$b_i = a_i + b_{i-1} x_1 \quad \begin{cases} b_1 = a_1 \\ i = 2 \rightarrow n+1 \\ resíduo = b_{n+1} \end{cases} \qquad (6.2)$$

Donde "n" es el grado del polinomio. Entonces para llevar a cabo la división sintética lo que se debe hacer es programar la ecuación (6.2), pero ello implica trabajar con matrices (más propiamente con matrices unidimensionales o vectores) por lo que primero veremos algunas de las instrucciones que tienen que ver con matrices.

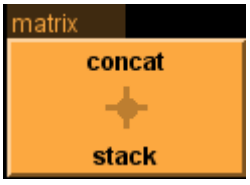
La mayoría de las instrucciones que tienen que ver con el manejo de matrices en Calc – Java se encuentran en el menú "math/matrix":



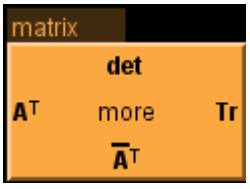
Donde, como es de suponer, el menú “create” tiene las instrucciones para crear matrices:

“parts”, las instrucciones para extraer partes de una matriz o dividirla:

“combine”, las instrucciones para combinar matrices:

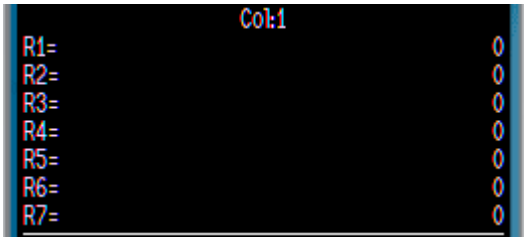


Y “math” algunas operaciones matemáticas con matrices:



Así por ejemplo, para crear una matriz vacía de 12 filas por 7 columnas, escribimos estos dos datos en la pila y luego ejecutamos la instrucción “new”: math/matrix/create/new, con lo que en la pila queda:

Que es la forma en la cual Calc – Java presenta las matrices. Para ver el contenido de una matriz (en este caso una matriz con ceros), se puede activar el monitor de matrices: mode/monitor/matrix/7:



Donde el número de filas a mostrar (en este caso 7) depende del tamaño de pantalla que se tenga disponible. Como se puede observar, el monitor de ma-

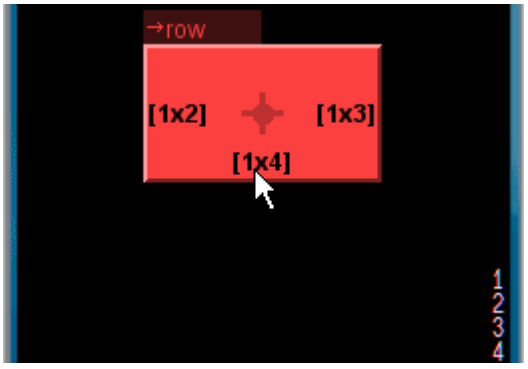
trices nos muestra en la parte superior el número de columna y a continuación el contenido de cada fila (R=Row=Fila). Para ver las otras filas y/o cambiar algún dato, se accede al monitor en la misma forma que cuando se edita un programa, es decir: basic/[->], y una vez en el monitor se pueden recorrer las filas y columnas con los botones de navegación, por ejemplo en la figura se muestra la columna 6 y el cursor está en la columna 5:

Cuando nos encontramos en el monitor, las funciones de los botones izquierda y derecha cambian a:

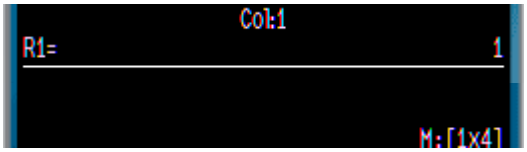
Donde “RCL” pone en la pila el valor de la celda en la que nos encontramos actualmente y “STO” permite guardar el valor de la pila en la celda. Por ejemplo, para almacenar en la celda {5,6} (fila 5, columna 6) el número 77.42, salimos del monitor (botón central del navegador), escribimos en la pila el número 77.42, volvemos a ingresar al monitor (basic/[->]) y pulsamos la tecla correspondiente a STO, con lo que resulta:

De esa manera podemos cambiar o recuperar los valores de una matriz, isn embargo, este procedimiento se usa normalmente sólo para corregir u obtener alguno de los valores de la matriz. Para crear una matriz con una serie de valores es más eficiente recurrir a las opciones ->row, para crear un vector fila (donde cada elemento se encuentra en una columna diferente), ->col, para crear un vector columna (donde cada elemento se encuentra en una fila diferente) y ->M para crear una matriz.

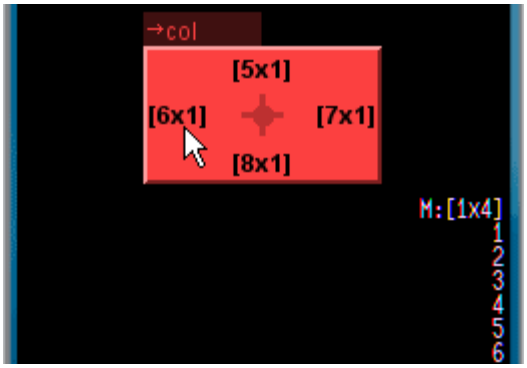
Por ejemplo para crear un vector fila con los elementos 1 2 3 y 4, escribimos dichos números en la pila, luego ejecutamos la instrucción: math/matrix/create/->row/[2-4]/[1x4]:



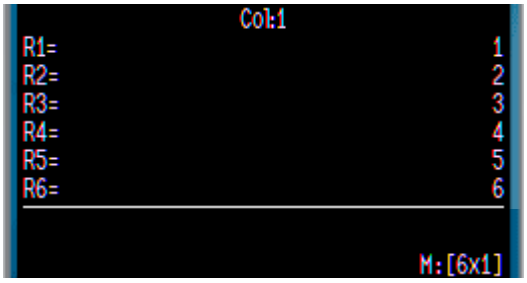
Con lo que se obtiene la matriz:



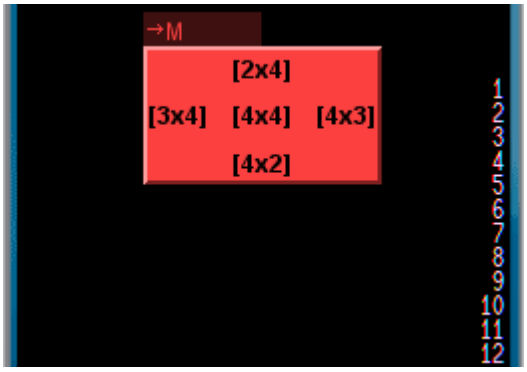
Igualmente, para crear un vector columna con los elementos 1, 2, 3, 4, 5 y 6, escribimos dichos números y luego ejecutamos la instrucción: math/matrix/create/->col/[5-8]/[6x1]



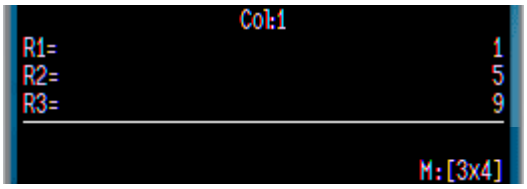
Con lo que queda la matriz:



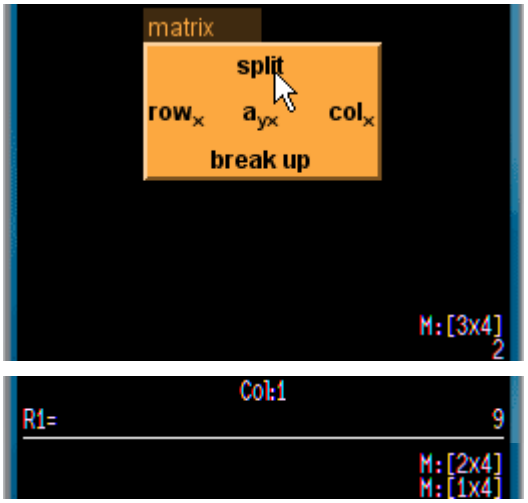
De manera similar podemos crear una matriz, por ejemplo una matriz de 3 filas por cuatro columnas con los números del 1 al 12 (math/matrix/create/->M/4/[3x4]:



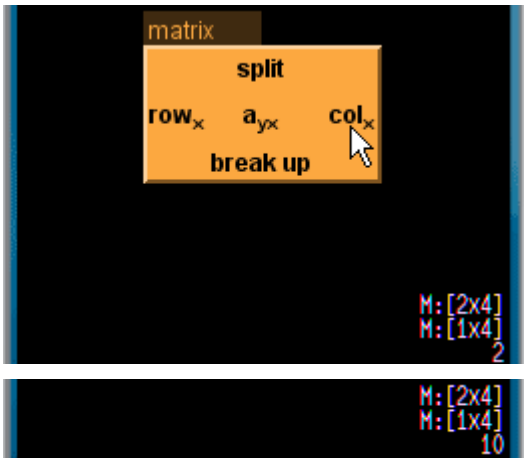
Con lo que se obtiene:



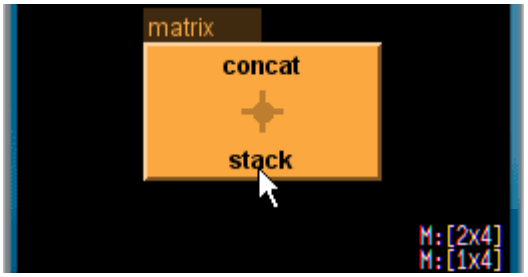
En cuanto a las partes, “split" divide la matriz a partir de la fila que se especifica en la pila, por ejemplo si a continuación de la anterior matriz escribimos 2 y luego ejecutamos la instrucción “split", (math/matrix/parts/split) obtenemos:



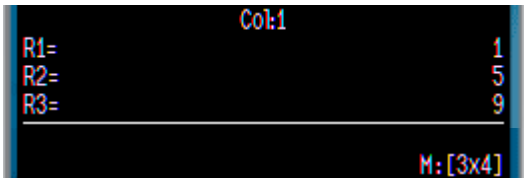
Donde como se puede observar quedan dos matrices una con las dos primeras filas y la segunda con la última fila. Para extraer el segundo elemento (la segunda columna) de la segunda matriz, escribimos el número 2 y luego ejecutamos la instrucción col$_x$ (math/matrix/parts/col$_x$):



De manera similar opera row$_x$ y a$_{yx}$, sólo que con row$_x$ se obtiene una fila (en lugar de una columna) y con a$_{yx}$, se obtiene el elemento que se encuentra en la fila y “y” en la columna “x”. Una vez dividida una matriz con “split”, podemos volver a unirla con “stack” (del menú “combine”), por ejemplo podemos unir la matriz que se encuentra en la pila (borrando previamente el número 10):

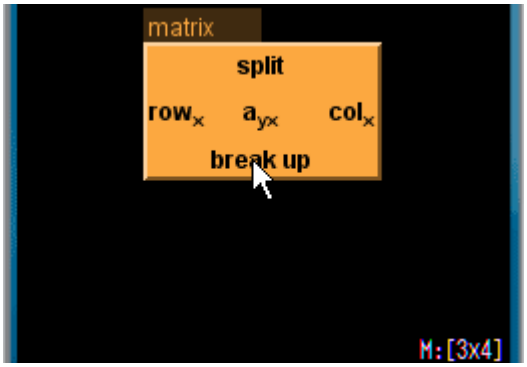


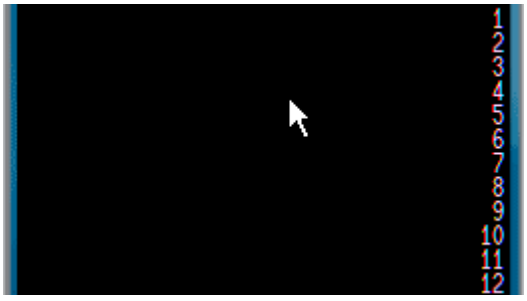
Con lo que se obtiene la matriz original:



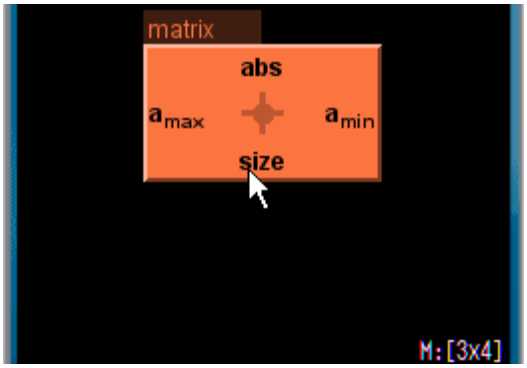
La instrucción “concat” hace lo mismo, pero en lugar de unir la matriz por sus filas (es decir una debajo de la otra) las une por las columnas (es decir una al lado de la otra).

Suele ser también de utilidad la instrucción “break up” que hace lo contrario de las instrucciones ->row, ->col o ->M, es decir rompe la matriz en sus elementos. Por ejemplo aplicando este comando a la matriz anterior se obtiene:



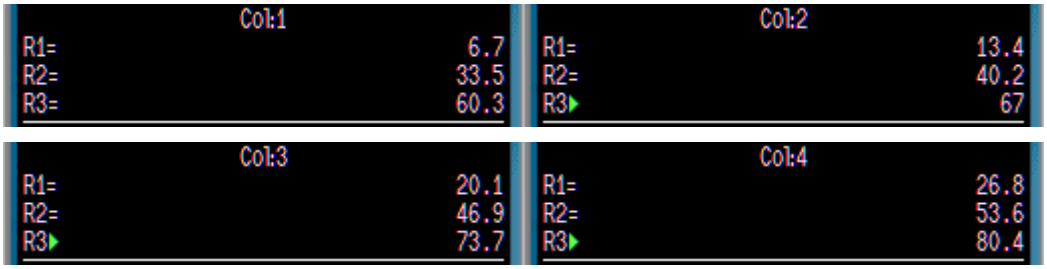


Para obtener las dimensiones de una matriz empleamos la instrucción “size” (math/matrix/math/more/size):

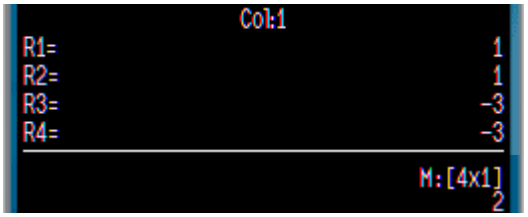


Por otro lado, los operadores básicos (suma, multiplicación, resta y división) funcionan también con matrices, por lo que es posible sumar, multiplicar y dividir matrices directamente con dichos operadores. Es posible

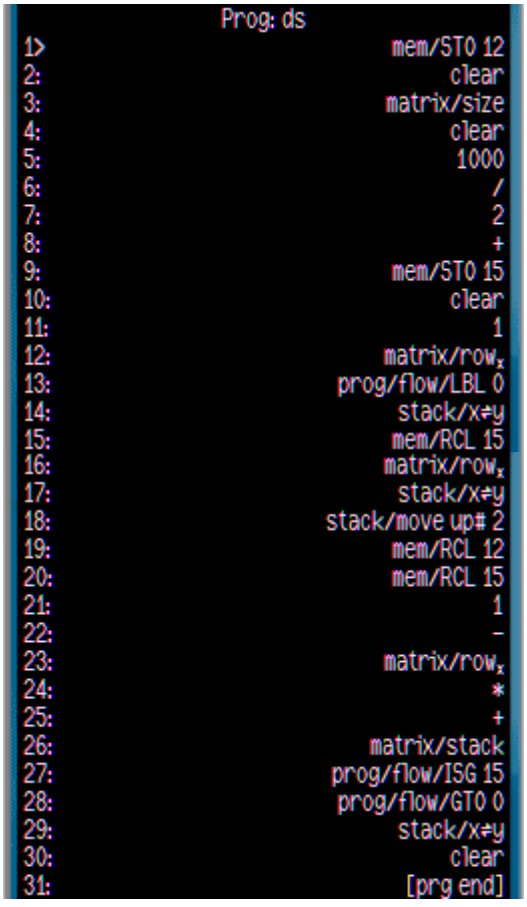
también multiplicar y dividir una matriz entre un escalar. Por ejemplo podemos multiplicar la matriz anterior por 6.7:



Ahora que ya conocemos las principales instrucciones para trabajar con matrices, podemos programar la ecuación (6.2). Para ello deberán estar en la pila la matriz de los coeficientes (a) y el dividendo (x_1). Podemos emplear como valores de prueba la ecuación cubica y el divisor de la página 103. Creamos primero el vector con los coeficientes (vector columna) y escribimos el divisor: 1/ENTER; 1/ENTER; -3/ENTER; -3/ENTER; math/matrix/create/->col/[3-4]/[4x1]; 2/ENTER:

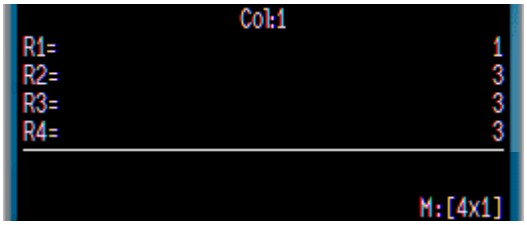


Ahora escribimos el programa para la ecuación (6.2):

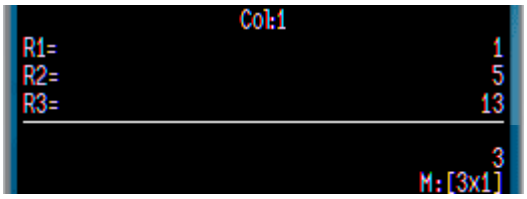


Que por supuesto no es la única forma, ni la más eficiente, de programar la ecuación (6.2) y que resulta inclusive un tanto largo para una ecuación tan sencilla. Ello se debe principalmente a que en Calc – Java no contamos con un comando que nos permita reemplazar directamente elementos dentro de una matriz, lo que en muchos nos obliga a dividirla y volver a combinarla.

Haciendo correr el programa (mode/prog/run/ds) con la matriz y el divisor antes escritos resulta:



Que concuerda con el resultado calculado manualmente, donde el valor del polinomio para x=2, es el último término (3). Extrayendo este término (3/matrix/parts/split; special/stack/x⇔y; 2/ENTER) y volviendo a hacer correr el programa (mode/prog/run/ds) se obtiene:

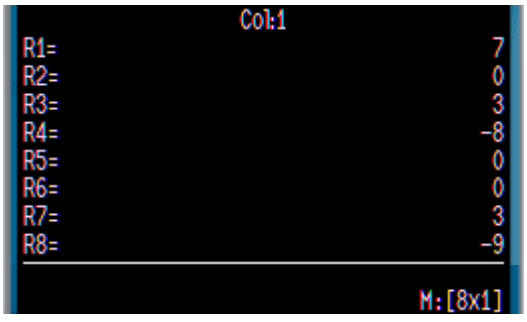


Que concuerda también con el resultado obtenido manualmente y donde, como se dijo, el último término (13) es la derivada primera del polinomio para "x=2".

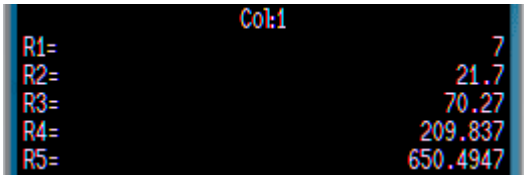
Como otro ejemplo, calculemos el valor de la función y el de la derivada primera para el polinomio que se presenta a continuación en x=3.1:

$$P_7(x) = 7x^7 + 3x^5 - 8x^4 + 3x - 9 = 0$$

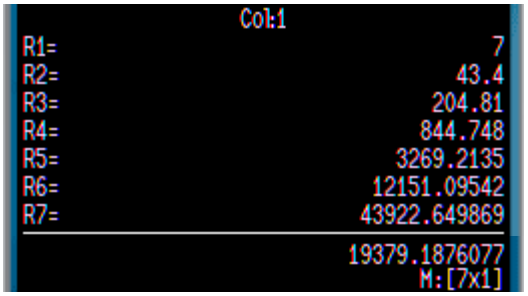
En este polinomio no existe coeficientes para todas las potencias (por ejemplo no existen coeficientes para x^3 y x^2). Cuando ello sucede, dichos coeficientes son cero, pero deben ser incorporados como tales en el vector, por lo tanto el vector para este polinomio es:



Introduciendo el divisor (3.1/ENTER) y haciendo correr el programa (mode/prog/run/ds) se obtiene:



Por lo tanto el valor de la función para x=3.1 es 19379.1876077. Para calcular la derivada, extraemos el valor de la función (7/matrix/parts/Split; special/stack/x⇔y), volvemos a introducir el divisor (3.1/ENTER) y volvemos a hacer correr el programa (mode/prog/run/ds), con lo que se obtiene:



Por lo tanto, la derivada primera del polinomio para x=3.1 es 43922.649869.

6.1.1. Ejercicios

1. Calcule el valor del polinomio: $693x^6-945x^4+315x^2-15=0$ y el de su derivada para “x=5.3”.
2. Calcule el valor del polinomio: $7x^7+3x^5-8x^4+3x-9=0$ y el de su derivada para “x=6.8”.
3. Calcule el valor del polinomio: $x^8-x^7-9x^6+13x^5+21x^4-125x^3+77x^2+111x-90=0$ y el de su derivada primera para “x=12.4”.
4. Calcule el valor del polinomio: $20x^{20}+3x^{17}-4x^{15}+12x^{12}-9x^8+6x^6-2x^4-4x^3+3x^2-x+9=0$ y el de su derivada para “x=3.9”.

6.2. Método de Newton – Raphson

Como ya vimos en el tema anterior, el método de *Newton – Raphson* (al igual que los otros métodos estudiados) permite encontrar una de las soluciones reales de una ecuación polinomial. Como también se vió, la raíz encontrada depende del valor inicial asumido. Si se dan valores iniciales cercanos a las soluciones, es posible encontrar todas las soluciones reales de un polinomio con el método de *Newton – Raphson*.

Por supuesto, podemos programar el polinomio en forma de función y resolver el mismo con el programa elaborado en el anterior tema, sin embargo, y como ya se dijo, cuando se trata de polinomios, resulta más eficiente recurrir a la división sintética. El método de *Newton – Raphson* en si no cambia, la lógica sigue siendo la misma, sólo que en el caso de los polinomios la función y la derivada se calculan mediante división sintética en lugar de emplear la función y la fórmula de diferencia central.

Una vez encontrada una raíz, es decir una vez encontrado el valor que hace cero el residuo, es posible emplear el polinomio residual (obtenido en la división sintética) para calcular otra de las raíces y continuar con este procedimiento hasta que el polinomio residual sea de segundo grado y resolver dicho polinomio con la fórmula de diferencia central.

No obstante, el procedimiento descrito puede dar lugar a resultados erróneos debido a la propagación del error: el error cometido al calcular la primera raíz se propaga a los coeficientes de la segunda y los de este a los de la tercera y así sucesivamente. Por esta razón, generalmente se prefiere calcular las soluciones con los coeficientes originales, empleando valores iniciales cercanos a las soluciones que se quieren calcular, si bien esto resulta en un procedimiento menos eficientes es por lo general más seguro.

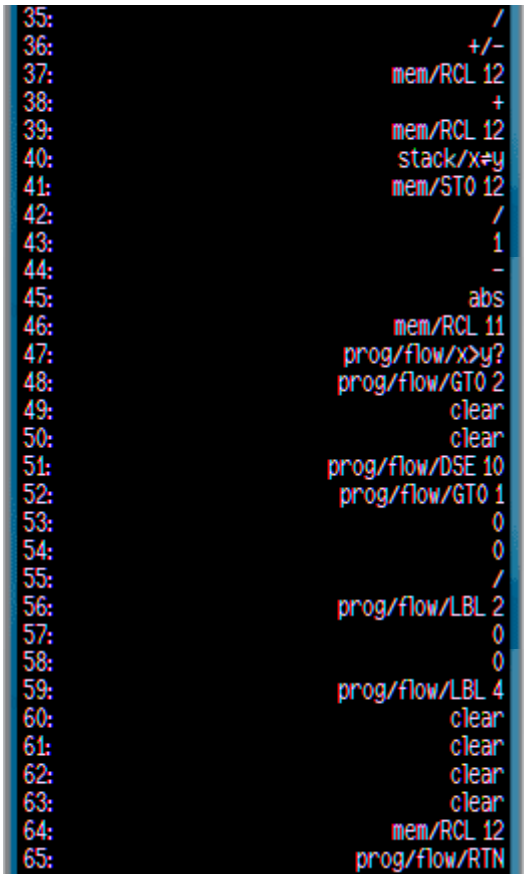
Para encontrar los valores iniciales es posible graficar la función y obtener de ella los valores iniciales. Existen también métodos numéricos (como QD) con los cuales es posible obtener valores iniciales adecuados.

6.2.1. Programa

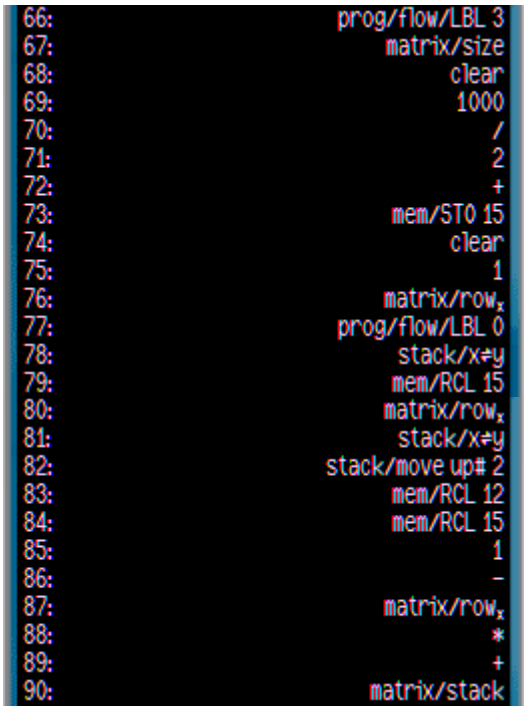
Como ya se dijo, el algoritmo del método no cambia, sólo cambia la forma en la que se calcula el valor de la función y el de su derivada. Los datos que requiere el programa son el vector con los coeficientes (vector columna) y el valor inicial asumido.

El error permitido (1e-14) y el límite de iteraciones (50) se mantienen en el interior del programa (memorias 11 y 10) pues los mismos no cambian con mucha frecuencia. La parte del programa correspondiente al método de Newton – Raphson propiamente, es la que se muestra a continuación y en la misma se emplea la memoria 12 para el valor de la variable independiente (asumido y calculado), 13 para el vector de coeficientes y 14 para el grado del polinomio:

```
Prog: nrp
1▶          0.00000000000001
2:                 mem/STO 11
3:                       clear
4:                          50
5:                 mem/STO 10
6:                       clear
7:                 mem/STO 12
8:                       clear
9:                 mem/STO 13
10:               matrix/size
11:                         -
12:                mem/STO 14
13:                      clear
14:                      clear
15:           prog/flow/LBL 1
16:                mem/RCL 13
17:           prog/flow/GSB 3
18:                mem/RCL 14
19:              matrix/split
20:                     ENTER
21:                       abs
22:                mem/RCL 11
23:            prog/flow/x>y?
24:           prog/flow/GTO 4
25:                      clear
26:                      clear
27:                  stack/x⇄y
28:           prog/flow/GSB 3
29:                mem/RCL 14
30:                         1
31:                         -
32:              matrix/split
33:                  stack/x⇄y
34:                      clear
```



Y la parte del programa que lleva a cabo la división sintética y que básicamente es una copia del programa elaborado en el acápite anterior (excepto que en este caso no es necesario volver a guardar el valor del divisor en la memoria 12) es la siguiente:



```
91:                    prog/flow/ISG 15
92:                    prog/flow/GTO 0
93:                    stack/x≠y
94:                    clear
95:                    prog/flow/RTN
96:                    [prg end]
```

Empleando este programa encontremos las soluciones de la ecuación cúbica empleada en la división sintética:

$$P_3(x) = x^3 + x^2 - 3x - 3 = 0$$

Para estimar los valores iniciales y al no contar por el momento con un método numérico adecuado, programamos el polinomio y graficamos el mismo:

Como se puede observar, el polinomio tiene tres soluciones reales, una cerca a -2, otra cerca a -1 y otra cerca a 2. Empleamos estos números como valores iniciales, primero con -2 obtenemos:

```
                    Col:1
R1=                      1
R2=                      1
R3=                     -3
R4=                     -3
                  M:[4x1]
-1.732050807568877
```

Y con -1 y 2 obtenemos:

```
                        -1
 1.732050807568877
```

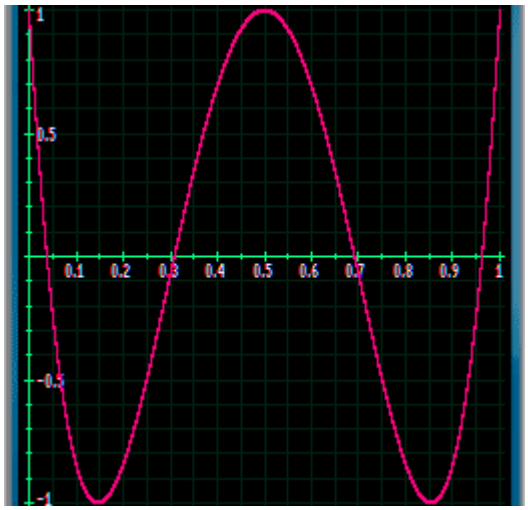
Que son las tres soluciones reales de la ecuación cúbica y que pueden ser corroborados con “solve” obteniéndose en los tres casos los mismos resultados. A continuación resolvemos unos cuantos ejemplos más con el método de Newton – Raphson para ecuaciones polinomiales.

6.2.2. Ejemplos

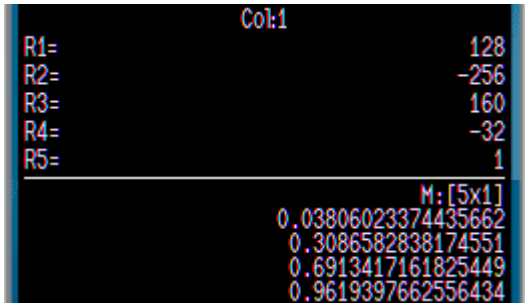
1. Encuentre las soluciones reales del polinomio que se presenta al final de la pregunta. Obtenga los valores iniciales graficando el polinomio.

$$P_4(x) = 128x^4 - 256x^3 + 160x^2 - 32x + 1 = 0$$

Para los valores iniciales programamos primero y graficamos el polinomio:



Entonces podemos tomar como valores iniciales: 0.05, 0.3, 0.7 y 1. Con estos valores obtenemos:

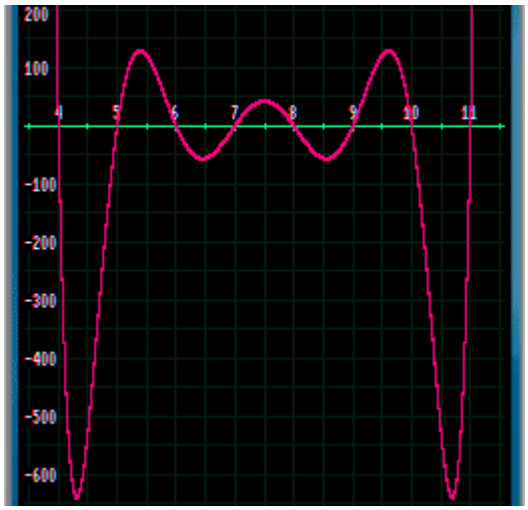


Que son las 4 soluciones reales del polinomio y que pueden ser verificadas con “solve”.

2. Encuentre las soluciones reales del polinomio que se presenta al final de la pregunta. Obtenga los valores iniciales graficando el polinomio.

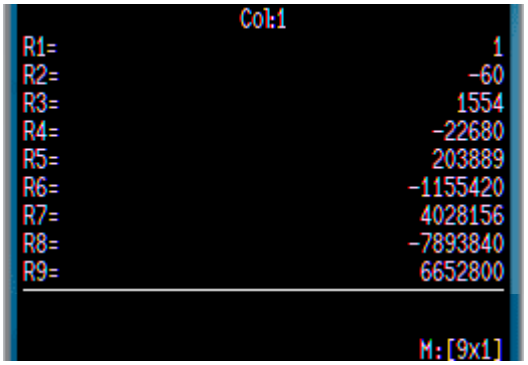
$$P_8(x) = x^8 - 60x^7 + 1554x^6 - 22680x^5 + 203889x^4 - 1155420x^3 + 4028156x^2 - 7893840x + 6652800 = 0$$

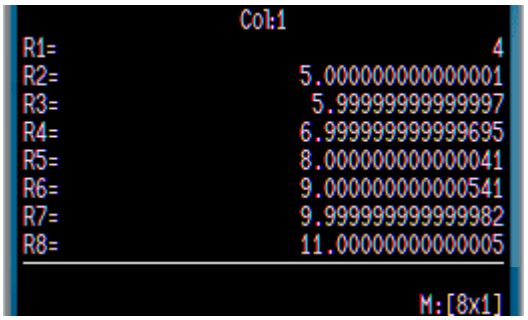
Al igual que el caso anterior, primero programamos y graficamos el polinomio para obtener los valores iniciales aproximados.



En este caso las soluciones parecen pasar exactamente por 4, 5, 6, 7, 8, 9, 10 y 11. Sin embargo, y para comprobar que el método converge con

valores cercanos, tomaremos como valores iniciales 3.7, 4.7, 5.7, 6.7, 7.7, 8.7, 9.8 y 10.7:



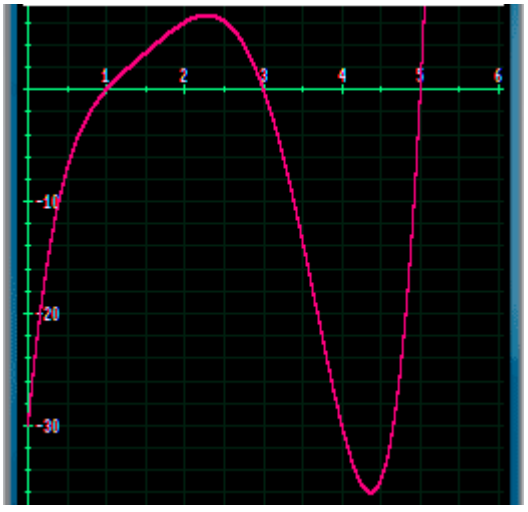


Observe que en la penúltima raíz se ha empleado como valor inicial 9.8 y no 9.7, esto debido a que en esa raíz en particular el método de Newton Raphson no converge hacia esa solución si se emplea como valor inicial 9.7. Los resultados han sido colocados en un vector (con "stack"), para agruparlos y disminuir la posibilidad de perder alguno de ellos en el proceso de cálculo.

3. Encuentre las soluciones reales del polinomio que se presenta al final de la pregunta. Obtenga los valores iniciales graficando el polinomio.

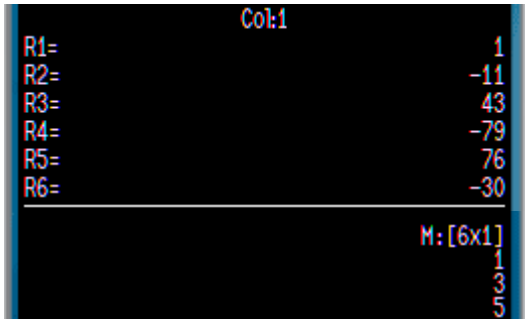
$$P_5(x) = x^5 - 11x^4 + 43x^3 - 79x^2 + 76x - 30 = 0$$

Primero programamos y graficamos el polinomio:



Como se puede observar, en este caso el polinomio sólo tiene 3 soluciones reales y dichas soluciones parecen ser 1, 3 y 5, por lo que podría-

mos emplear dichos valore como valores iniciales, sin embargo, y al igual que el ejemplo anterior emplearemos valores cercanos a estos (0.8, 2.8 y 4.8) simplemente para verificar que el método converge hacia la solución más cercana.



Que son las soluciones exactas y que pueden ser verificadas también con "solve".

6.2.3. Ejercicios

5. Encuentre las soluciones reales del polinomio que se presenta al pie de la pregunta. Obtenga los valores iniciales de la gráfica del polinomio.

$$P_3(x) = 5x^3 - 250x^2 + 4000x - 20000 = 0$$

6. Encuentre las soluciones reales del polinomio que se presenta al pie de la pregunta. Obtenga los valores iniciales de la gráfica del polinomio.

$$P_9(x) = 5x^9 - 8x^7 + 4x^6 - 2x^5 + x^4 - 6x^3 + 3x^2 - 2x - 80 = 0$$

7. Encuentre las soluciones reales del polinomio que se presenta al pie de la pregunta. Obtenga los valores iniciales de la gráfica del polinomio.

$$P_{10}(x) = 12x^{10} - 8x^9 + 6x^7 - 3x^3 + 5x^2 - 3x - 10 = 0$$

8. Encuentre las soluciones reales del polinomio que se presenta al pie de la pregunta. Obtenga los valores iniciales de la gráfica del polinomio.

$$P_8(x) = x^8 - 7x^7 + 11x^6 + 41x^5 - 183x^4 + 231x^3 + 21x^2 - 265x + 150 = 0$$

7. ECUACIONES POLINOMIALES - 2

En este tema continuamos el estudio de los métodos que nos permiten resolver ecuaciones polinomiales.

En el anterior tema los valores iniciales fueron estimados graficando la función, sin embargo, dicho procedimiento, sólo nos permite encontrar las soluciones reales. En este tema veremos un método que nos permite estimar valores iniciales para encontrar tanto soluciones reales como imaginarias.

Antes sin embargo revisaremos algunos de los métodos, estudiados en álgebra, para encontrar las soluciones reales e imaginarias de ecuaciones de segundo y tercer grado.

7.1. Ecuación Cuadrática

La ecuación cuadrática es la ecuación polinomial más simple y como se sabe sus soluciones, tanto reales como imaginarias, pueden ser calculadas aplicando la fórmula:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{7.1}$$

Donde "a", "b" y "c" son los coeficientes de la ecuación cuadrática: $ax^2+bx+c=0$.

Puesto que Calc – Java nos permite trabajar con valores imaginarios podemos aplicar directamente la anterior ecuación sin tener que hacer ninguna consideración especial (casos reales, iguales o imaginarios). Sin embargo, una solución más eficiente, propuesta por Roar Lauritzsen (el creador de Calc-Java: http://midp-calc.sourceforge.net/Calc.html), aprovecha el hecho de que en la solución de una ecuación cuadrática también se cumple que:

$$x_1 x_2 = \frac{c}{a} \tag{7.2}$$

Lo que propone es calcular primero el valor de la expresión:

$$q = \frac{b + signo(b)\sqrt{b^2 - 4ac}}{-2} \tag{7.3}$$

Que, debido a que multiplica el valor de la raíz por el signo de "b", siempre nos devuelve la suma de ambos valores, sin importar que "b" sea positivo o negativo.

Resulta claro entonces que si esta expresión se divide entre "a" obtenemos una de las soluciones (pues simplemente se reproduce una de las soluciones de la ecuación (7.1):

$$x_1 = \frac{q}{a} \tag{7.4}$$

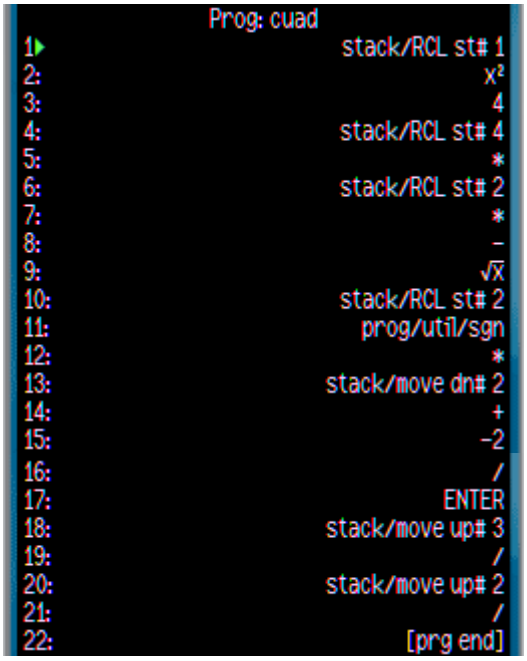
La segunda solución se obtiene entonces de las ecuaciones (7.2) y (7.4):

$$x_2 = \frac{c}{x_1 a} = \frac{c}{\frac{q}{a} a} = \frac{c}{q} \tag{7.5}$$

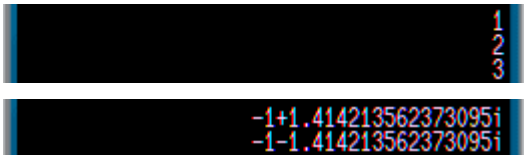
Estas son las expresiones que emplearemos para calcular las soluciones de la ecuación cuadrática.

7.1.1. Programa

El programa elaborado es esencialmente el propuesto por Roar Lauritzsen, sólo que en nuestro caso no dejamos los valores de los coeficientes en la pila. Para hacer correr el programa es necesario que los coeficientes “a”, “b” y “c” (y en ese orden) estén en la pila:



Primero probamos el programa con la ecuación: $x^2+2x+3=0$, que tiene dos soluciones imaginarias:



Y como se puede observar se obtienen los resultados correctos. Ahora probamos el programa con la ecuación: $x^2-12x+27=0$, que tiene dos soluciones diferentes:

Finalmente probamos el programa con la ecuación $x^2-24+144=0$, que tiene dos soluciones iguales:

Que también devuelve los resultados correctos.

Si bien el programa “cuad” puede ser empleado independientemente para calcular las soluciones de cualquier ecuación cuadrática, tal como se ha hecho en los anteriores ejemplos, su principal aplicación está en la solu-

ción de ecuaciones cuadráticas al interior de otros métodos como el de QD y Bairstow, tal como veremos posteriormente.

7.1.2. Ejercicios

1. Encuentre las soluciones de la ecuación: $2x^2-3.5x+4.2=0$.
2. Encuentre las soluciones de la ecuación: $3.2x^2+1.5x-5.8=0$.
3. Encuentre las soluciones de la ecuación: $x^2-7.3x+3.1=0$.
4. Encuentre las soluciones de la ecuación: $-2.3x^2-5.1x+7.76=0$.
5. Encuentre las soluciones de la ecuación: $3.21x^2-8.44x+11.23=0$.

7.2. Ecuación Cúbica

Es posible también encontrar directamente las raíces de la ecuación cúbica aplicando la solución propuesta por primera vez, en el año 1545, por Cardano, razón por la cual se conoce también como el método de Cardano. Para ello es necesario que la ecuación general:

$$b_1x^3+b_2x^2+b_3x+b_4=0 \tag{7.6}$$

Sea llevada a la forma:

$$x^3+a_1x^2+a_2x+a_3=0 \tag{7.7}$$

Lo que se consigue simplemente dividiendo todos los coeficientes entre el primero. Entonces se calculan los siguientes valores:

$$Q=\frac{3a_2-a_1^2}{9} \tag{7.8}$$

$$R=\frac{9a_1a_2-27a_3-2a_1^3}{54} \tag{7.9}$$

$$S=\sqrt[3]{R+\sqrt{Q^3+R^2}} \tag{7.10}$$

$$S=\sqrt[3]{R-\sqrt{Q^3+R^2}} \tag{7.11}$$

Y con los mismos se encuentran las tres soluciones:

$$x_1=S+T-\frac{1}{3}a_1 \tag{7.12}$$

$$x_2=-\frac{1}{2}(S+T)-\frac{1}{3}a_1+\frac{1}{2}\sqrt{-3}(S-T) \tag{7.13}$$

$$x_3=-\frac{1}{2}(S+T)-\frac{1}{3}a_1-\frac{1}{2}\sqrt{-3}(S-T) \tag{7.14}$$

Una vez más, dado que Calc – Java nos permite trabajar con valores imaginarios, no es necesario hacer ninguna consideración adicional, es suficiente evaluar las expresiones presentadas para obtener las tres soluciones.

7.2.1. Programa

El programa elaborado requiere que esté en la pila un vector con los coeficientes de la ecuación cúbica: [a_1, a_2, a_3, a_4]:

```
Prog: card
1▶                    1
2:           matrix/split
3:            stack/x⇄y
4:                    /
5:       matrix/break up
6:                    3
7:     stack/RCL st# 2
8:                    *
9:     stack/RCL st# 3
10:                  x²
11:                   -
12:                   9
13:                   /
14:                   9
15:    stack/RCL st# 4
16:                   *
17:   stack/move dn# 3
18:                   *
19:                  27
20:   stack/move dn# 3
21:                   *
22:                   -
23:                   2
24:    stack/RCL st# 3
25:                   3
26:          math/pow/yˣ
27:                   *
28:                   -
29:                  54
30:                   /
31:           stack/x⇄y
32:                   3
33:          math/pow/yˣ
34:    stack/RCL st# 1
35:                  x²
36:                   +
37:                  √x
38:    stack/RCL st# 1
39:    stack/RCL st# 1
40:                   +
41:                   3
42:         math/pow/ˣ√y
43:   stack/move up# 2
44:                   -
45:                   3
46:         math/pow/ˣ√y
47:    stack/RCL st# 1
48:    stack/RCL st# 1
49:                   +
50:               ENTER
51:   stack/move dn# 4
52:                  -3
53:                   /
54:               ENTER
55:   stack/move up# 3
56:                   +
57:   stack/move up# 4
58:                  -2
59:                   /
60:                   +
61:   stack/move up# 2
62:                   -
```

```
63:                              -3
64:                              √x
65:                               2
66:                               /
67:                               *
68:                  stack/RCL st# 1
69:                  stack/RCL st# 1
70:                               +
71:                stack/move up# 2
72:                               -
73:                stack/move dn# 2
74:                        [prg end]
```

Donde el penúltimo paso sólo lleva el primer resultado a la primera posición de la pila, porque casi siempre dicho resultado es el mayor de los tres.

Primero probamos el programa con la ecuación $x^3+2x^2+3x+4=0$, que tiene una solución real y dos imaginarias:

```
                Col:1
R1=                                1
R2=                                2
R3=                                3
R4=                                4
```

```
 -0.1746854042803059+1.546868887231396i
 -0.1746854042803059-1.546868887231396i
                    -1.650629191439388
```

Y como vemos el programa devuelve los resultados correctos (dos soluciones imaginarias y una real). Ahora probamos el programa con la ecuación: $x^3-6x^2+11x-6=0$, que tiene tres soluciones reales:

```
                Col:1
R1=                                1
R2=                               -6
R3=                               11
R4=                               -6
```

```
                                   1
                                   2
                                   3
```

Siendo las soluciones “1”, “2” y “3” que son las soluciones correctas. Probamos ahora el programa con la ecuación: $x^3-22x^2+145x-300=0$, que tiene dos soluciones iguales y una diferente:

```
                Col:1
R1=                                1
R2=                              -22
R3=                              145
R4=                             -300
```

```
                                   5
                                   5
                                  12
```

Finalmente probamos el programa con la ecuación: $x^3+21x^2+147x+343=0$, que tiene 3 soluciones iguales (y negativas):

```
                Col:1
R1=                                1
R2=                               21
R3=                              147
R4=                              343
```

Obteniéndose también los resultados correctos.

7.2.2. Ejercicios

6. Encuentre las soluciones de la ecuación: $x^3-3x^2-2.5x+3.1=0$.
7. Encuentre las soluciones de la ecuación: $x^3-8.2x^2-0.25x+49.786=0$.
8. Encuentre las soluciones de la ecuación: $x^3-16.6x^2+72.01x-64.236=0$.
9. Encuentre las soluciones de la ecuación: $3x^3+21x^2-15x-225=0$.
10. Encuentre las soluciones de la ecuación: $2.1x^3-20.328x^2+85.6918x-324.07=0$.

7.3. Método QD

Como ha quedado evidente en el método de Newton, la principal dificultad para encontrar las soluciones de un polinomio es el determinar los valores iniciales, más aún si dichas soluciones son imaginarias.

El método "QD", permite encontrar todas las soluciones de una ecuación polinomial (reales e imaginarias) sin requerir valores iniciales. Sin embargo este método tiene la desventaja de converger muy lentamente, requiriendo en consecuencia un número elevado de iteraciones. Por ello en la práctica, este método es usado normalmente sólo para obtener valores iniciales. Las soluciones más exactas se encuentran luego con otros métodos (como el de Newton).

La ecuación general del polinomio que se toma en cuenta en las ecuaciones de este método es:

$$P_n(x) = a_1x^n + a_2x^{n-1} + a_3x^{n-2} + a_4x^{n-3} + ... + a_nx + a_{n+1} = 0 \qquad (7.15)$$

Donde "n" es el grado del polinomio. Los valores iniciales se calculan con las ecuaciones:

$$p_1 = -\frac{a_2}{a_1};\quad p_i = 0 \quad \{i = 2 \rightarrow n \qquad (7.16)$$

$$d_i = \frac{a_{i+2}}{a_{i+1}} \quad \{i = 1 \rightarrow n-1 \qquad (7.17)$$

Una vez calculados los valores iniciales, los valores sucesivos se calculan con las ecuaciones:

$$\left.\begin{array}{l} q_1 = d_1 + p_1 \\ q_i = d_i - d_{i-1} + p_i \\ q_n = p_n - d_{n-1} \end{array}\right\} i = 2 \rightarrow n-1 \qquad (7.18)$$

$$e_i = \frac{q_{i+1}}{q_i}\mathrm{d_i} \quad \{i = 1 \rightarrow n-1 \qquad (7.19)$$

Si al efectuar estos cálculos los valores de e_i son cero o cercanos a cero el proceso concluye, caso contrario "p" se iguala a "q" ($p=q$) y "d" a "e"

(*d=e)* y se vuelven a evaluar las ecuaciones (7.18) y (7.19), repitiéndose el proceso hasta que los valores de e_i se hacen cero o casi cero.

Cuando el proceso concluye, las soluciones del polinomio son los valores de del vector *"q"*.

Si existen raíces imaginarias (complejas), algunos de los valores de e_i no se aproximan a cero, sino que fluctúan en signo, en esos casos las soluciones imaginarias se encuentran resolviendo la ecuación cuadrática $x^2-rx-s=0$, donde los valores de *"r"* y *"s"* se calculan con las siguientes expresiones:

$$\begin{aligned} r &= q_i + q_{i+1} \\ s &= -p_i q_{i+1} \end{aligned} \qquad (7.20)$$

Siendo *"i"* el índice del elemento e_i que fluctúa en signo.

7.3.1. Programa

Básicamente lo que se debe hacer para implementar el método QD es programar primero las ecuaciones (7.16) y (7.17), para obtener los vectores iniciales "p" y "d".

Con "p" y "d" calculados se programan las ecuaciones (7.18) y (7.19) para obtener los valores de "q" y "e". Si los valores de "e" son cercanos a cero, el proceso concluye, caso contrario se hace que "p=q" y "d=e" y se vuelven a calcular nuevos valores de "q" y "e", repitiendo este procedimiento hasta que los valores de "e" sean cercanos a cero.

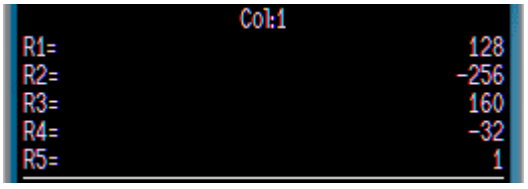
Cuando termina el proceso los valores de "q" son las soluciones del polinomio, o son los datos que se requiere para calcular las soluciones imaginarias aplicando la ecuación (7.20).

Todo el proceso será implementado en un programa, sin embargo, dado que el mismo es relativamente largo, se irá elaborando y probando por partes (simplemente facilitar su comprensión y corrección), añadiendo cada parte a la anterior, hasta completar el programa.

Comenzaremos entonces con el cálculo de los vectores iniciales "p" y "d". Como datos emplearemos los coeficientes correspondientes al siguiente polinomio:

$$P_4(x) = 128x^4 - 256x^3 + 160x^2 - 32x + 1 = 0 \qquad (7.21)$$

Creamos la matriz:

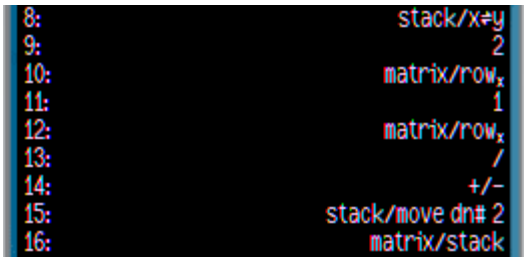


Y programamos la ecuación (7.16). Como contador de los ciclos emplearemos la memoria 15 y guardaremos el grado del polinomio en la memoria 14. Primero obtenemos el grado del polinomio, para ello empleamos "size", que nos devuelve el número de filas (5) y columnas (1), por lo que restando estos dos valores (5-1), obtenemos el grado del polinomio (4):

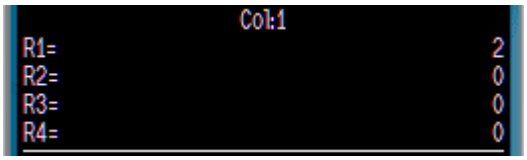
Con lo que queda almacenada en la memoria 14 el grado del polinomio.

Dado que todos los elementos de “p”, excepto el primero son cero, creamos una matriz con “n-1” elementos iguales a cero:

Entonces calculamos el primer término de “p” ($-a_2/a_1$) y añadimos el mismo en la primera posición de esta matriz:



Si en este punto salimos de programa y vemos (en el monitor) la matriz resultante, deberemos tener lo siguiente:



Que son los elementos del vector “d”. Ahora continuamos elaborando el programa y calculamos el vector “p” con la ecuación (7.17). Ello implica la ejecución de un ciclo iterativo que va desde 1 hasta “n-1”. Como contador de dicho ciclo emplearemos la memoria 15, sólo que por conveniencia, en lugar de repetir el ciclo desde 1 hasta “n-1”, repetiremos el mismo desde 2 hasta “n”.

En consecuencia debemos almacenar el número “2.nnn01” en la memoria 15, “2” es el valor inicial, “nnn” es el límite final (grado del polinomio) y “01” el incremento. Sin embargo, dado que “Calc-Java” emplea automáticamente como incremento “01”, aprovecharemos ese hecho y almacenaremos en la memoria 15 sólo el número “2.nnn” (sin el incremento), que en el caso del polinomio empleado como ejemplo (de grado 4), es “2.004”. Ello se consigue dividiendo el grado del polinomio entre 1000 y sumando 2 al resultado:

Y el ciclo donde se calcula los valores de la ecuación (7.17) es:

```
32:                    mem/RCL 15
33:                    matrix/row↓
34:                    /
35:                    stack/x⇄y
36:                    stack/move up# 2
37:                    matrix/stack
38:                    prog/flow/ISG 15
39:                    prog/flow/GTO 0
40:                    1
41:                    matrix/split
42:                    stack/x⇄st# 2
43:                    clear
44:                    clear
```

Si ahora salimos del programa y hacemos correr el mismo con los coeficientes de la matriz (7-21) (o hacemos correr el programa paso a paso con SST), obtendremos lo siguiente:

```
              Col:1
R1=                    -0.625
R2=                     -0.2
R3=                   -0.03125
                      M:[4x1]
                      M:[3x1]
```

Que son las matrices “p” y “d” respectivamente.

Con los valores de “p” y “d” podemos calcular ahora los elementos del vector “q” (ecuación 7.18) y como el proceso se repite desde esta parte colocamos una etiqueta en este paso, sin embargo, primero guardamos el valor del vector “p” en la memoria 13, pues el mismo se requiere en el caso de que las soluciones sean imaginarias (ecuación 7.20).

```
45:                    stack/x⇄y
46:                    mem/STO 13
47:                    clear
48:                    prog/flow/LBL 1
49:                    1
50:                    matrix/row↓
51:                    mem/RCL 13
52:                    1
53:                    matrix/row↓
54:                    stack/x⇄y
55:                    clear
56:                    +
57:                    mem/RCL 14
58:                    1
59:                    -
60:                    1000
61:                    /
62:                    2
63:                    +
64:                    mem/STO 15
65:                    clear
66:                    prog/flow/LBL 2
67:                    stack/x⇄y
68:                    mem/RCL 15
69:                    matrix/row↓
70:                    mem/RCL 15
71:                    1
72:                    -
73:                    matrix/row↓
74:                    -
75:                    mem/RCL 13
```

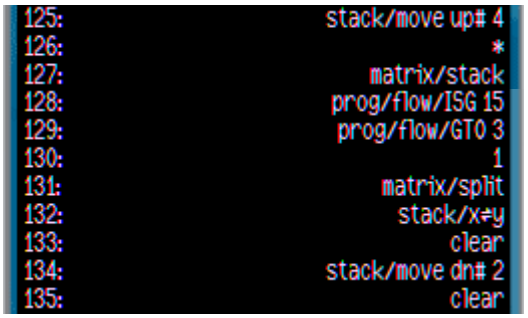
```
76:                    mem/RCL 15
77:                    matrix/row$_x$
78:                    stack/x<>y
79:                    clear
80:                    +
81:                    stack/x<>y
82:                    stack/move up# 2
83:                    matrix/stack
84:                    prog/flow/ISG 15
85:                    prog/flow/GTO 2
86:                    mem/RCL 13
87:                    mem/RCL 14
88:                    matrix/row$_x$
89:                    stack/x<>y
90:                    clear
91:                    stack/move dn# 2
92:                    mem/RCL 14
93:                    1
94:                    -
95:                    matrix/row$_x$
96:                    stack/x<>y
97:                    stack/move up# 3
98:                    -
99:                    matrix/stack
```

Si ahora hacemos correr el programa, con la matriz original, en la pila obtenemos la matriz “q” (y queda también la “d”):

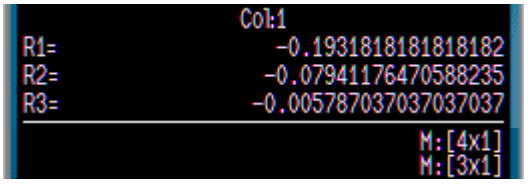
```
              Col:1
R1=             1.375
R2=             0.425
R3=           0.16875
R4=           0.03125
-------------------------
              M:[3x1]
              M:[4x1]
```

Finalmente con la matriz “q” podemos calcular la matriz “e” (ecuación 7.19):

```
100:                   mem/RCL 14
101:                   1
102:                   -
103:                   1000
104:                   /
105:                   1
106:                   +
107:                   mem/STO 15
108:                   clear
109:                   0
110:                   prog/flow/LBL 3
111:                   stack/x<>y
112:                   mem/RCL 15
113:                   1
114:                   +
115:                   matrix/row$_x$
116:                   mem/RCL 15
117:                   matrix/row$_x$
118:                   /
119:                   stack/x<>y
120:                   stack/move up# 2
121:                   stack/move dn# 3
122:                   mem/RCL 15
123:                   matrix/row$_x$
124:                   stack/x<>y
```

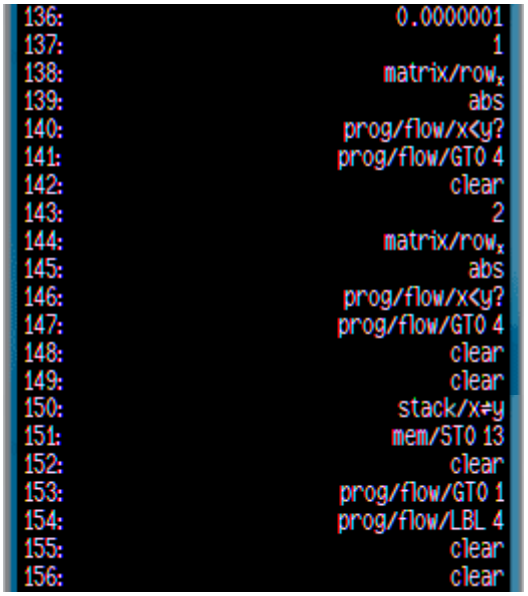


Si ahora salimos del programa y hacemos correr el mismo con la matriz original, en la pila quedan los vectores “q” y “e”:

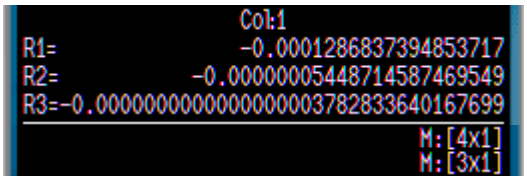


Ahora se debe verificar que estos valores sean cercanos a cero y de no ser así emplear estos vectores como nuevos valores de “p” y “d”. Como ya se puede observar en esta primera iteración, los últimos valores son los que más rápido convergen hacia cero, mientras que el primero es el que más lento converge. Dado que la convergencia del primer elemento es el factor limitante, se puede emplear dicho valor para determinar si los valores de “e” son casi cero. No obstante, cuando las soluciones son complejas (como veremos un poco más adelante), el valor correspondiente de “e_i” no converge hacia cero, sino que fluctúa en signo, por lo tanto si la primera raíz es compleja, el primer valor nunca convergerá hacia a cero y por lo tanto si sólo se controla el primer valor el método nunca dejará de iterar. Para evitar esta situación es conveniente controlar el valor no sólo del primer elemento, sino también del segundo.

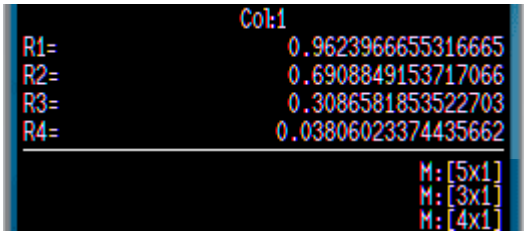
La parte del programa que realiza dicha verificación, con un error permitido igual a 1e-7, es:



Haciendo correr este programa, con la matriz original, los valores de “e” resultantes son:



Donde como se puede ver el último valor es muy cercano a cero, mientras que el primero sólo tiene 3 dígitos después del cero. Los valores de “q” son:

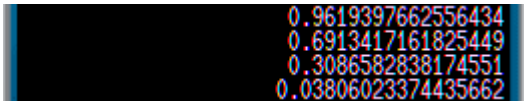


Que son las cuatro soluciones aproximadas del polinomio. Soluciones que pueden compararse con las obtenidas por ejemplo con Mathematica:

x_1 -> 0.03806023374435662
x_2 -> 0.3086582838174551
x_3 -> 0.6913417161825449
x_4 -> 0.9619397662556433

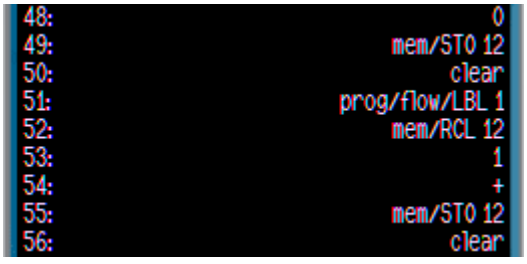
Como ya se dedujo en base a los valores de “e” el último resultado es el más exacto, el penúltimo tiene 6 dígitos de precisión y los dos primeros apenas 2. No obstante, y como ya se dijo, QD, normalmente se emplea sólo para obtener valores iniciales, siendo los resultados más exactos calculados con otro método como el de Newton.

Por ejemplo empleando estos valores, como valores iniciales del método de Newton obtenemos:



Que de hecho son las soluciones correctas.

Sólo con el fin de averiguar cuántas veces se repite el proceso antes de que se cumpla la condición del error, añadiremos un contador al programa (el mismo que deberá ser eliminado luego, pues no forma parte del método propiamente). Antes y después de la etiqueta “LBL 1” añadimos:



Y al final de la etiqueta “LBL 4” añadimos mem/RCL 12:

Con estas modificaciones, volviendo a hacer correr el programa se obtienen los mismos resultados y el número 19, el cual nos informa que el proceso se ha repetido 19 veces.

Probemos una vez más el método pero ahora con la ecuación: $x^8-7.3x^7-41.44x^6+373.474x^5+115.908x^4-4822.15x^3+6985.17x^2+5589.66x-10901.5=0$. En consecuencia, la matriz que se debe mandar al programa es:

```
                Col:1
R1=                                1
R2=                             -7.3
R3=                            -41.44
R4=                           373.474
R5=                           115.908
R6=                          -4822.15
R7=                           6985.17
R8=                           5589.66
R9=                          -10901.5
                              M:[9x1]
```

Haciendo correr el programa “qd” con esta matriz se obtiene:

```
                Col:1
R1=  -0.00000009216781556927675
R2=    0.0002011675433781508
R3=  -0.00000014100337827590 84
R4=  0.000000000000001433076803525 82
R5= -0.0000000000000000000000000007454747
R6=  -0.000000000000000219981329750529
R7=  -0.000000000000001407030585843091
                              M:[8x1]
                              M:[7x1]
                                125
```

Donde el número 125 nos informa el número de veces que se ha repetido el proceso: 125 veces, y a pesar de ello, como se puede observar, algunos de los valores de “e” (como el segundo y el tercero) no son todavía muy cercanos a cero.

Los valores de “q”, es decir las soluciones aproximadas son:

```
                Col:1
R1=   6.299992529136181
R2=  -5.500104046852091
R3=   5.100119028264328
R4=  -4.500000103618348
R5=   3.399993893724669
R6=   2.099989818857792
R7=   1.600009937932023
R8=  -1.200001057444554
```

Empleando estos valores como valores iniciales para el método de Newton se obtiene:

```
                Col:1
R1=   6.299992479923997
R2=  -5.499999617445176
R3=   5.100014573168351
R4=  -4.500000028717093
R5=   3.399993893724661
R6=   2.099989818857791
R7=   1.600009937932016
R8=  -1.200001057444546
```

Las mismas que pueden ser comparadas con soluciones obtenidas con Mathematica:

x_1 -> -5.499999617445176
x_2 -> -4.500000028717094
x_3 -> -1.2000010574445465
x_4 -> 1.6000099379320165
x_5 -> 2.0999898188577903
x_6 -> 3.399993893724658
x_7 -> 5.10001457316836
x_8 -> 6.299992479923989

Como una vez más se puede comprobar, los valores iniciales del método QD, son lo suficientemente aproximados como para permitir encontrar soluciones más precisas con el método de Newton. En realidad si bajamos más aún el error permitido, por ejemplo a 1e-3:

```
143:                              clear
144▸                              0.001
145:                                  1
```

Obtenemos los siguientes resultados.

```
                  Col:1
R1=    -0.0009456100829808699
R2=      0.0341520929635097 3
R3=    -0.0006997035110123481
R4=     0.00000271761325750948 4
R5=  -0.000000000000126525251955664 5
R6=     -0.00000002359461436793112
R7=    -0.000004408542626638347
                                M:[8x1]
                                M:[7x1]
                                     57
```

Como se puede ver, con esta precisión se requieren 57 iteraciones en lugar de 125. Siendo los valores de “q”:

```
                  Col:1
R1=      6.300504259772332
R2=     -5.51825820391114
R3=      5.118127289680532
R4=     -4.500367486363602
R5=      3.399995441479927
R6=      2.099989917955298
R7=      1.600012358005813
R8=     -1.200003576619159
```

Empleando estos valores como valores iniciales para el método de Newton, se obtiene:

```
                  Col:1
R1=      6.299992479923997
R2=     -5.499999617445176
R3=      5.100014573168351
R4=     -4.500000028717093
R5=      3.399993893724661
R6=      2.099989818857791
R7=      1.600009937932016
R8=     -1.200001057444546
```

Que son los mismos resultados obtenidos previamente con los valores iniciales más precisos. Por consiguiente si el objetivo es obtener valores iniciales para luego encontrar valores más precisos con otros métodos como el

de Newton, una precisión de 1e-3 para el método QD suele ser más que suficiente.

No obstante y sobre todo cuando se tienen raíces complejas, una precisión de esta magnitud puede dar lugar a resultados erróneos (raíces reales en lugar de imaginarias). Por esta razón mantendremos la precisión de 1e-7 en el método QD, a pesar de que en la mayoría de los casos una precisión menor como 1e-3 suele ser suficiente.

Ahora que ya sabemos cómo afecta la precisión en el número de iteraciones, borramos las líneas añadidas para ese fin y con ello habríamos programado las ecuaciones del método QD.

Veamos qué sucede cuando el polinomio tiene raíces (soluciones) complejas, como ocurre con el siguiente polinomio:

$$P_4(x) = x^4 - 6x^3 + 12x^2 - 19x + 12 = 0 \qquad (7.22)$$

```
                Col:1
R1=                 1
R2=                -6
R3=                12
R4=               -19
R5=                12
                M:[5x1]
```

Haciendo correr el programa se obtiene:

```
                Col:1
R1=  -0.00000003040911112100044
R2=        -102.114673492451
R3=  -0.0001186137267323944
                M:[4x1]
                M:[3x1]
```

Siendo los valores de “q”:

```
                Col:1
R1=     3.999999196516308
R2=   0.02910087094229025
R3=    0.9709801564583777
R4=    0.9999197760830236
```

Como en este caso el segundo valor de “e”, no se aproxima a cero, deducimos que existen dos soluciones complejas, las raíces 2 y 3, las mismas que se calculan con los elementos 2 y 3 de “q” y el elemento 2 de “p”, aplicando la ecuación (7.20) y resolviendo luego la ecuación cuadrática resultante:

```
                M:[4x1]
   0.02910087094229025
    0.9709801564583777
    0.9709801564583777
```

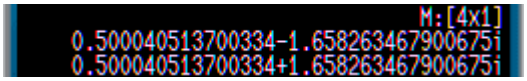
El valor de “p_2” se recupera de la memoria 13:

```
                M:[4x1]
   0.02910087094229025
    0.9709801564583777
    0.9709801564583777
                M:[4x1]
                     2
```

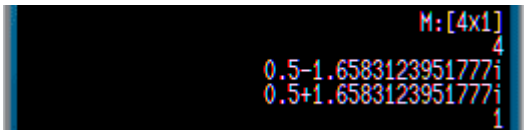
```
                M:[4x1]
   0.02910087094229025
    0.9709801564583777
    0.9709801564583777
     3.089536098510639
```

Y añadimos un 1 para tener todos los coeficientes de la ecuación cuadrática:

Finalmente, resolviendo la ecuación cuadrática (con “cuad”) obtenemos:



Que son las dos soluciones complejas (aproximadas) del polinomio. Empleando estas soluciones aproximadas (tanto reales como complejas), como valores iniciales del método de Newton, obtenemos:



Por supuesto, no es necesario calcular la segunda raíz compleja, pues sabemos que siempre es el par conjugado de la primera, por lo tanto puede ser obtenida directamente a partir de la primera con la instrucción “conj”: trig/more/cplx/conj.

Podemos comparar estas soluciones con las obtenidas con Mathematica:

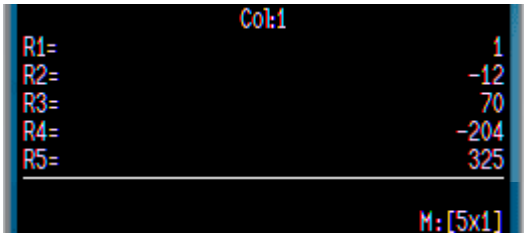
x_1 -> 0.5 - 1.6583123951777*I
x_2 -> 0.5 + 1.6583123951777*I
x_3 -> 1.
x_4 -> 4.

Y como se puede observar, son exactamente los mismos resultados.

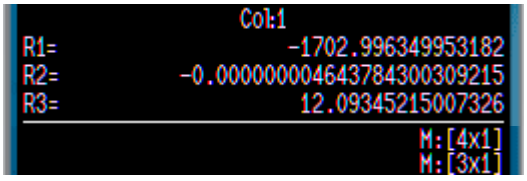
Resolvamos ahora un polinomio con dos soluciones complejas:

$$x^4 - 12x^3 + 70x^2 - 204x + 325 = 0 \qquad (7.23)$$

El vector que se manda a “qd” es:



Siendo los valores de e:



Como el primer y el tercer elemento de “e” no convergen hacia cero, sabemos que existen dos soluciones complejas: las soluciones 1 y 2 y las solu-

ciones 3 y 4, por lo tanto, en este caso todas las soluciones son complejas. Siendo los valores de “q”:

```
                Col:1
R1=          1710.98174175119
R2=         -1702.981738433106
R3=         -2.209330596159987
R4=          6.209327278075717
                      M:[3x1]
                      M:[4x1]
```

Con estos valores encontramos las soluciones (aproximadas) aplicando la ecuación (7.20), igual que en el caso anterior. Los coeficientes de la ecuación cuadrática, calculados a partir de los valores de “q” y “p” son:

```
                      M:[4x1]
                           1
                -8.000003318084269
                 25.00004406929781
```

Y las soluciones respectivas (obtenidas con “cuad”):

```
                      M:[4x1]
 4.000001659042135-3.000005132821938i
 4.000001659042135+3.000005132821938i
```

De manera similar, los coeficientes de la ecuación cuadrática para las otras dos soluciones son:

```
                           1
                -3.999996681915731
                 12.99995488509657
```

Y las soluciones respectivas (obtenidas con “cuad”):

```
 1.999998340957865-2.999993586870205i
 1.999998340957865+2.999993586870205i
```

Empleando estas soluciones como valores iniciales en el método de Newton calculamos soluciones más exactas (recordando que las soluciones complejas siempre están en pares conjugados):

```
                           4-3i
                           4+3i
                           2-3i
                           2+3i
```

Que son las soluciones exactas del polinomio.

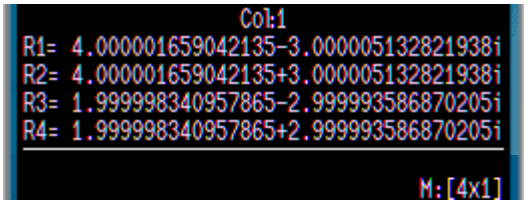
Como se ha hecho evidente en los anteriores ejemplos, el proceso de calcular las raíces imaginarias es moroso y se hace más moroso a medida que incrementa el grado del polinomio, por lo que es conveniente automatizar el proceso dentro del programa, pero como Calc-Java no permite llamar a un programa desde otro, no queda otra alternativa que copiar el programa “cuad” dentro del programa “qd”.

El código que se debe añadir al programa para este fin es el siguiente:

```
157:                       0
158:              matrix/stack
159:            mem/RCL 14
160:                    1000
161:                       /
162:                       1
163:                       +
164:            mem/STO 15
165:                   clear
166:                       0
167:          prog/flow/LBL 5
168:                 stack/x⇄y
```

```
169:                    0.01
170:              mem/RCL 15
171:          matrix/row$_x$
172:                     abs
173:          prog/flow/x<y?
174:          prog/flow/GTO 6
175:                   clear
176:                   clear
177:          stack/x⇄st# 2
178:                       1
179:              mem/RCL 15
180:          matrix/row$_x$
181:              mem/RCL 15
182:                       1
183:                       +
184:          matrix/row$_x$
185:                       +
186:                      +/-
187:              mem/RCL 15
188:                       1
189:                       +
190:          matrix/row$_x$
191:              mem/RCL 13
192:              mem/RCL 15
193:          matrix/row$_x$
194:           stack/x⇄y
195:                   clear
196:                       *
197:          stack/RCL st# 1
198:                      x²
199:                       4
200:          stack/RCL st# 4
201:                       *
202:          stack/RCL st# 2
203:                       *
204:                       -
205:                      √x
206:          stack/RCL st# 2
207:            prog/util/sgn
208:                       *
209:          stack/move dn# 2
210:                       +
211:                      -2
212:                       /
213:                   ENTER
214:          stack/move up# 3
215:                       /
216:          stack/move up# 2
217:                       /
218:             matrix/stack
219:              mem/RCL 15
220:                       1
221:                       +
222:              mem/STO 15
223:                   clear
224:          prog/flow/GTO 7
225:          prog/flow/LBL 6
226:                   clear
227:                   clear
228:          stack/x⇄st# 2
229:              mem/RCL 15
230:          matrix/row$_x$
231:          prog/flow/LBL 7
```

Haciendo correr el programa con los datos del último polinomio se obtiene:



Que son las soluciones complejas aproximadas del polinomio y que concuerdan (como era de esperar) con las soluciones calculadas previamente.

Sin embargo existen todavía algunas consideraciones adicionales que es necesario hacer en el método “QD”. Así, analizando las ecuaciones (7.16) y (7.17), se puede inferir fácilmente que si alguno de los coeficientes del polinomio es cero, se produciría una división entre cero, por lo que en esos casos es necesario realizar un cambio de variable. Otra situación en la que suele ser necesario un cambio de variables es cuando dos o más coeficientes están igualmente espaciados, pues también se puede producir una división entre cero.

El cambio de variable más sencillo y conveniente es aquel en el que se cambia la variable “x” por “x-1”. Para ello simplemente se divide (división sintética) el polinomio entre “x-1” y el polinomio resultante nuevamente entre “x-1” y así sucesivamente, siendo los residuos de cada división los coeficientes del nuevo polinomio. Este cambio de variable resulta conveniente, porque una vez encontradas las raíces con la nueva variable, las soluciones de la variable original se obtienen simplemente sumando 1 a estos resultados.

Por ejemplo para resolver el siguiente polinomio, donde el segundo coeficiente es cero:

$$P_4(x) = x^4 - 3x^2 + x - 5 = 0$$

Llevamos a cabo cuatro divisiones sintéticas:

```
R3=                                -2
R4=                                -1
R5=                                -6
-----------------------------------
                                   -6
                              M:[4x1]
```

```
R1=                                 1
R2=                                 2
R3=                                 0
-----------------------------------
                                   -6
                                   -1
                              M:[3x1]
```

```
R1=                                 1
R2=                                 3
-----------------------------------
                                   -6
                                   -1
                                    3
                              M:[2x1]
```

```
                                   -6
                                   -1
                                    3
                                    4
                                    1
```

Siendo estos (en orden invertido) los coeficientes del polinomio con el cambio de variable:

```
R1=                                 1
R2=                                 4
R3=                                 3
R4=                                -1
R5=                                -6
-----------------------------------
                              M:[5x1]
```

Haciendo correr “qd” con este vector resulta:

```
R1=                   -3.136281482431566
R2=-0.903226723479572+1.092824600320095i
R3=-0.903226723479572-1.092824600320095i
R4=                    0.9427349293907104
-----------------------------------------
```

Las soluciones del polinomio original se obtienen sumando a esta matriz la matriz unidad:

```
R1=                                 1
R2=                                 1
R3=                                 1
R4=                                 1
-----------------------------------
```

```
R1=                   -2.136281482431566
R2=0.0967732765204277+1.092824600320095i
R3=0.0967732765204277-1.092824600320095i
R4=                    1.94273492939071
----------------------------------------
```

Empleando estas soluciones como valores iniciales del método de Newton, resulta:

```
                      -2.136282774791117
 0.09318111352445112+1.091617990922614i
 0.09318111352445112-1.091617990922614i
                       1.949920547742215
```

Resultados que pueden ser comparados con los obtenidos por ejemplo con Mathematica:

x_1 -> -2.136282774791117
x_2 -> 0.09318111352445112 - 1.0916179909226145*I
x_3 -> 0.09318111352445112 + 1.0916179909226145*I
x_4 -> 1.9499205477422152

Siendo esencialmente los mismos.

Una vez más se hace evidente que el cálculo de los nuevos coeficientes (por división sintética) es un proceso moroso, por lo que es conveniente automatizar el mismo. Ello implica copiar el programa de división sintética dentro del nuevo programa (debido a que no se puede llamar a un programa desde otro).

La parte del programa que automatiza el proceso es la siguiente:

```
Prog: cvar
1▶                matrix/size
2:                          -
3:              mem/STO 14
4:                     clear
5:                         1
6:              mem/STO 12
7:                     clear
8:                         0
9:            prog/flow/LBL 3
10:                 stack/x≠y
11:           prog/flow/GSB 2
12:             mem/RCL 14
13:              matrix/split
14:       stack/move dn# 2
15:             matrix/stack
16:           prog/flow/DSE 14
17:           prog/flow/GTO 3
18:             matrix/stack
19:              matrix/size
20:                         -
21:              matrix/split
22:                     clear
23:           prog/flow/RTN
```

Mientras que la división sintética se copia a continuación, como una sub-rutina encerrada entre LBL 2 y RTN:

```
24:           prog/flow/LBL 2
25:              matrix/size
26:                     clear
27:                      1000
28:                         /
29:                         2
30:                         +
31:             mem/STO 15
32:                     clear
33:                         1
34:              matrix/row$_x$
35:           prog/flow/LBL 0
36:                 stack/x≠y
37:             mem/RCL 15
38:              matrix/row$_x$
39:                 stack/x≠y
40:       stack/move up# 2
41:             mem/RCL 12
42:             mem/RCL 15
```

```
43:                    1
44:                    -
45:          matrix/row_x
46:                    *
47:                    +
48:          matrix/stack
49:  prog/flow/ISG 15
50:  prog/flow/GTO 0
51:          stack/x<>y
52:                clear
53:  prog/flow/RTN
54:          [prg end]
```

Mandando a este programa la matriz:

```
          Col:1
R1=           1
R2=           0
R3=          -3
R4=           1
R5=          -5
──────────────────
          M:[5x1]
```

Se obtiene:

```
          Col:1
R1=           1
R2=           4
R3=           3
R4=          -1
R5=          -6
──────────────────
          M:[5x1]
```

Que, como era de esperar, son los mismos coeficientes calculados manualmente.

7.3.2. Ejemplos

1. Encuentre las soluciones reales e imaginarias del polinomio que se presenta al pie de la pregunta. Obtenga los valores iniciales con QD y las soluciones más exactas con Newton:

$$x^8 - 7x^7 + 11x^6 + 41x^5 - 183x^4 + 231x^3 - 21x^2 - 265x - 150 = 0$$

Mandando la matriz:

```
          Col:1
R1=           1
R2=          -7
R3=          11
R4=          41
R5=        -183
R6=         231
R7=         -21
R8=        -265
R9=        -150
──────────────────
          M:[9x1]
```

A “qd” se obtiene:

```
          Col:1
R1=  3.382352737341295
```

```
R2= 2.579653937492198-1.703085851092344i
R3= 2.579653937492198+1.703085851092344i
R4=                  -2.894155858029523
R5= 1.199121973240708-1.639261227675233i
R6= 1.199121973240708+1.639261227675233i
R7=-0.5228743503887914+0.30423654736033i
R8=-0.5228743503887914-0.30423654736033i
                                  M:[8x1]
```

Que son las 8 solucione (reales e imaginarias) aproximadas del polinomio. Empleando estas soluciones como valores iniciales del método de Newton se obtienen soluciones más precisas:

```
                 Col:1
R1=                   3.382353975978588
R2= 2.644710677521432-1.649034226910478i
R3= 2.644710677521432+1.649034226910478i
R4=                  -3.024270576725285
R5= 1.199121973240708-1.639261227675233i
R6= 1.199121973240708+1.639261227675233i
R7=-0.5228743503887914+0.30423654736033i
R8=-0.5228743503887914-0.30423654736033i
```

Que son prácticamente las mismas soluciones que se obtienen con Mathematica:

```
x -> -3.0242705767252844},
x -> -0.5228743503887914 - 0.3042365473603303*I
x -> -0.5228743503887914 + 0.3042365473603303*I
x -> 1.1991219732407075 - 1.639261227675233*I
x -> 1.1991219732407075 + 1.639261227675233*I
x -> 2.644710677521432 - 1.6490342269104785*I
x -> 2.644710677521432 + 1.6490342269104785*I
x -> 3.3823539759785883
```

2. Encuentre las soluciones reales e imaginarias del polinomio que se presenta al pie de la pregunta. Obtenga los valores iniciales con QD y las soluciones más exactas con Newton:

$$x^8 - 60x^7 + 1554x^6 - 22680x^5 + 203889x^4 - 1155420x^3 + 4028156x^2 - 7893840x + 6652800 = 0$$

Mandamos la siguiente matriz a “qd”:

```
                 Col:1
R1=                                1
R2=                              -60
R3=                             1554
R4=                           -22680
R5=                           203889
R6=                         -1155420
R7=                          4028156
R8=                         -7893840
R9=                          6652800
                                  M:[9x1]
```

Con lo que obtenemos las soluciones aproximadas:

```
                 Col:1
R1=                  11.00000244253071
R2=                  9.999998527465296
R3=                  8.999999228889507
R4=                  7.999999822560593
```

```
R5=                6.999999979554382
R6=                5.999999999012767
R7=                4.999999999986768
R8=                3.999999999999979
                                    M:[8x1]
```

Como se puede ver, en este caso todas la soluciones son reales. Con estos valores y el método de Newton encontramos soluciones más precisas:

```
                   Col:1
R1=               10.99999999999999
R2=               10.00000000000001
R3=               8.999999228889507
R4=               7.999999999999886
R5=               6.999999999999741
R6=               5.999999999999963
R7=               5.000000000000001
R8=                               4
                                    M:[8x1]
```

Resultados que pueden compararse con los obtenidos con Mathematica:

```
x -> 4.0000000000027995
x -> 4.999999999936522
x -> 6.000000000191773
x -> 6.999999999644557
x -> 8.000000000458584
x -> 8.999999999673697
x -> 10.000000000257753
x -> 10.99999999998726}
```

3. Encuentre las soluciones reales e imaginarias del polinomio que se presenta al pie de la pregunta. Obtenga los valores iniciales con QD y las soluciones más exactas con Newton:

$$2x^{20}+3x^{17}-4x^{15}+12x^{12}-9x^{8}+6x^{6}-2x^{4}-4x^{3}+3x^{2}-x+9=0$$

El vector de coeficientes del polinomio es:

```
                   Col:1
R1=                                2
R2=                                0
R3=                                0
R4=                                3
R5=                                0
R6=                               -4
R7=                                0
R8=                                0
R9=                               12
R10=                               0
R11=                               0
R12=                               0
R13=                              -9
R14=                               0
R15=                               6
R16=                               0
R17=                              -2
R18=                              -4
R19=                               3
R20=                              -1
R21=                               9
```

```
M:[21x1]
```

Como este vector tiene ceros, primero efectuamos el cambio de variable con el programa “cvar”:

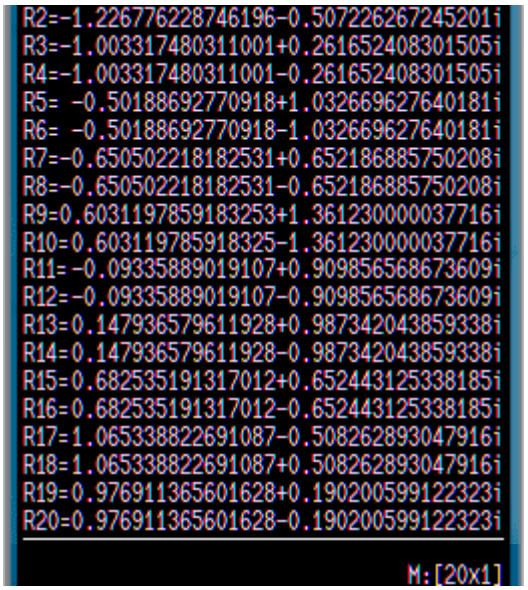
```
          Col:1
R1=             2
R2=            40
R3=           380
R4=          2283
R5=          9741
R6=         31412
R7=         79500
R8=        161760
R9=        268696
R10=       367732
R11=       416636
R12=       391470
R13=       305061
R14=       197076
R15=       105470
R16=        46596
R17=        16768
R18=         4744
R19=          977
R20=          124
R21=           15
M:[21x1]
```

Ahora mandamos este vector a “qd”:

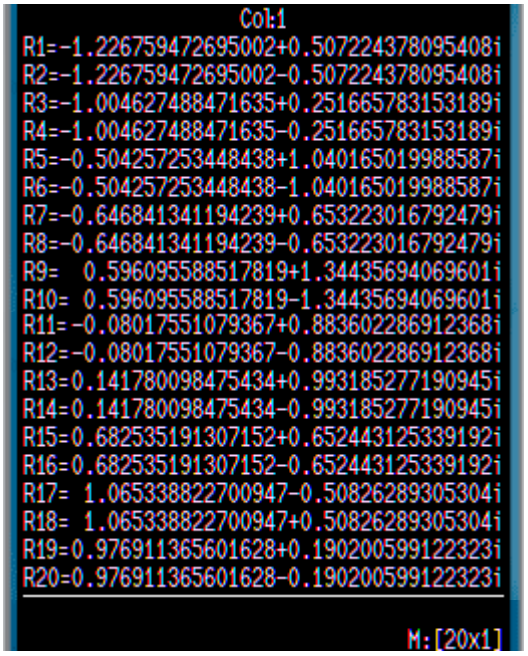
```
                    Col:1
R1=-2.226776228746196+0.507226267245201i
R2=-2.226776228746196-0.507226267245201i
R3=-2.003317480311001+0.261652408301505i
R4=-2.003317480311001-0.261652408301505i
R5= -1.50188692770918+1.032669627640181i
R6= -1.50188692770918-1.032669627640181i
R7=-1.650502218182531+0.652186885750208i
R8=-1.650502218182531-0.652186885750208i
R9=-0.396880214081675+1.361230000037716i
R10=-0.39688021408167-1.361230000037716i
R11=-1.09335889019107+0.909856568673609i
R12=-1.09335889019107-0.909856568673609i
R13=-0.85206342038807+0.987342043859338i
R14=-0.85206342038807-0.987342043859338i
R15=-0.31746480868299+0.652443125338185i
R16=-0.31746480868299-0.652443125338185i
R17=0.065338822691087-0.508262893047916i
R18=0.065338822691087+0.508262893047916i
R19=-0.02308863439837+0.190200599122323i
R20=-0.02308863439837-0.190200599122323i
M:[20x1]
```

Como se puede ver, en este caso todas las soluciones son complejas. Dado que en este caso se ha hecho el cambio de variable, para obtener las soluciones del polinomio original debemos sumar un vector unidad con 20 elementos a este resultado:

```
                    Col:1
R1=-1.226776228746196+0.507226267245201i
```



Empleando estas soluciones como valores iniciales del método de Newton obtenemos las soluciones más precisas:



Las mismas que pueden ser comparadas con las obtenidas con Mathematica:

```
x -> -1.2267594726950024 - 0.5072243780954078*I
x -> -1.2267594726950024 + 0.5072243780954078*I
x -> -1.0046274884716349 - 0.2516657831531895*I
x -> -1.0046274884716349 + 0.2516657831531895*I
x -> -0.6468413411942394 - 0.653223016792479*I
x -> -0.6468413411942394 + 0.653223016792479*I
x -> -0.5042572534484376 - 1.0401650199885866*I
x -> -0.5042572534484376 + 1.0401650199885866*I
x -> -0.0801755107936667 - 0.8836022869123685*I
x -> -0.0801755107936667 + 0.8836022869123685*I
x -> 0.14178009847543452 - 0.9931852771909448*I
x -> 0.14178009847543452 + 0.9931852771909448*I
```

```
x -> 0.5960955885178191 - 1.3443569406960103*I
x -> 0.5960955885178191 + 1.3443569406960103*I
x -> 0.6825351913071521 - 0.6524431253391921*I
x -> 0.6825351913071521 + 0.6524431253391921*I
x -> 0.9769113656016282 - 0.19020059912232326*I
x -> 0.9769113656016282 + 0.19020059912232326*I
x -> 1.065338822700947 - 0.5082628930530398*I
x -> 1.065338822700947 + 0.5082628930530398*I
```

Que una vez más son prácticamente los mismos resultados.

7.3.3. Ejercicios

11. Encuentre las soluciones reales e imaginarias del polinomio que se presenta al pie de la pregunta. Obtenga los valores iniciales con QD y las soluciones más exactas con Newton:

$$P_4(x) = x^4 - 16x^3 + 72x^2 - 96x + 24 = 0$$

12. Encuentre las soluciones reales e imaginarias del polinomio que se presenta al pie de la pregunta. Obtenga los valores iniciales con QD y las soluciones más exactas con Newton:

$$P_9(x) = 5x^9 - 8x^7 + 4x^6 - 2x^5 + x^4 - 6x^3 + 3x^2 - 2x - 80 = 0$$

13. Encuentre las soluciones reales e imaginarias del polinomio que se presenta al pie de la pregunta. Obtenga los valores iniciales con QD y las soluciones más exactas con Newton:

$$P_{10}(x) = 12x^{10} - 8x^9 + 6x^7 - 3x^3 + 5x^2 - 3x - 10 = 0$$

14. Encuentre las soluciones reales e imaginarias del polinomio que se presenta al pie de la pregunta. Obtenga los valores iniciales con QD y las soluciones más exactas con Newton:

$$P_6(x) = x^6 + x^5 - 8x^4 + 231x^3 + 21x^2 - 265x + 150 = 0$$

15. Encuentre las soluciones reales e imaginarias del polinomio que se presenta al pie de la pregunta. Obtenga los valores iniciales con QD y las soluciones más exactas con Newton:

$$P_8(x) = x^8 + x^7 - 9x^6 + 13x^5 + 21x^4 - 125x^3 + 77x^2 + 111x - 90 = 0$$

8. ECUACIONES POLINOMIALES - 3

En este tema continuamos el estudio de los métodos que nos permiten resolver ecuaciones polinomiales y en esta ocasión completamos dicho estudio con uno de los métodos más empleados para resolver ecuaciones de este tipo: el método de Bairstow.

Puesto que en dicho método se emplea a la división sintética doble, estudiaremos primero dicho proceso.

8.1. División Sintética Doble

Al igual que la división sintética simple (entre un monomio) es útil para encontrar una de las raíces o soluciones de una ecuación polinomial, la división sintética doble (entre un binomio) es útil para encontrar dos de las raíces o soluciones de una ecuación polinomial.

Para recordar las operaciones que se realizan cuando se lleva a cabo una división sintética doble, dividamos el polinomio:

$$P_4(x) = x^4 - 1.1x^3 + 2.3x^2 + 0.5x + 3.3 = 0 \qquad (8.1)$$

Entre x^2+x+1. Para lo cual debemos llevar a cabo las siguientes operaciones:

	1	-1.1	2.3	0.5	3.3
$r=-1$		-1.0	2.1	-3.4	0.8
$s=-1$			-1.0	2.1	-3.4
	1	-2.1	3.4	-0.8	0.7

Si denominamos *"b"* a los coeficientes resultantes de la primera división, entonces el residuo de esta división es: $b_n(x-r)+b_{n+1}$ (donde "n" es el grado del polinomio). Con los resultados del ejemplo el residuo es: -0.8*(x+1)+0.7 = -0.8*x-0.1.

Para una segunda división sintética, empleando los resultados obtenidos en la primera división (sin tomar en cuenta el último elemento), realizamos las operaciones:

	1	-2.1	3.4	-0.8
$r=-1$		-1.0	3.1	-5.5
$s=-1$			-1.0	3.1
	1	-3.1	5.5	-3.2

Ahora, si denominamos "c" a estos resultados, dichos resultados son las derivadas parciales: $c_{i-1}=\partial bi/\partial r$ y $c_{i-2}=\partial bi/\partial s$.

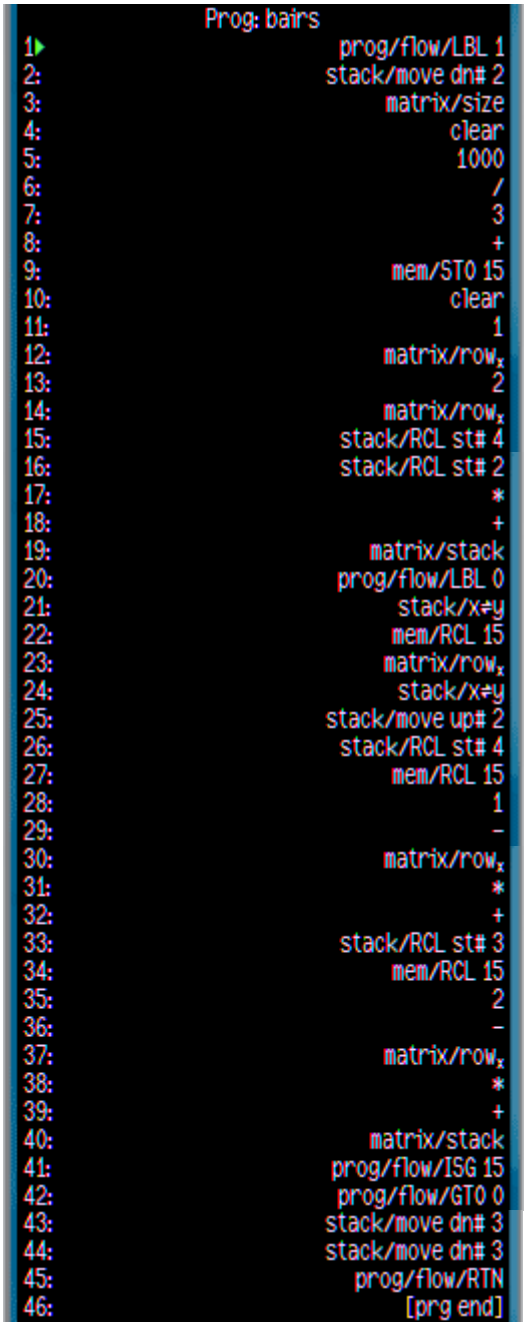
Analizando el procedimiento seguido en las dos anteriores divisiones, denominando "a" a los coeficientes del polinomio original y "b" a los coeficientes resultantes, podemos deducir que la ecuación general para la división de un polinomio de grado "n" entre el binomio $x^2+rx+x=0$ es:

$$b_i = a_i + b_{i-1}r + b_{i-2}s \quad \begin{cases} b_1 = a_1 \\ b_2 = a_2 + c_1 r \\ i = 3 \rightarrow n+1 \\ resíduo = b_n(x-r) + b_{n+1} \end{cases} \qquad (8.2)$$

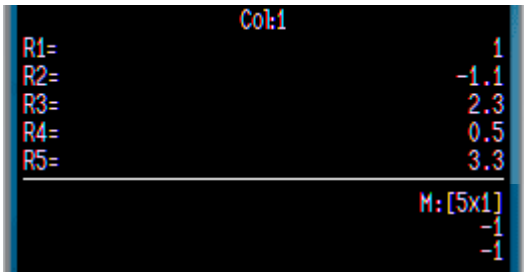
8.1.1. Programa

Dado que la división sintética doble será empleada exclusivamente como parte del método de Bairstow, el programa se denominará “bairs” y el mismo requiere que estén en la pila el vector de los coeficientes y los divisores “r” y “s”.

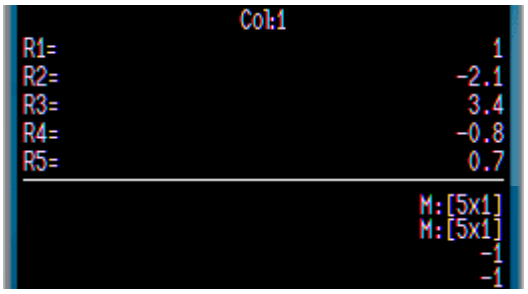
Por lo anteriormente mencionado el programa ha sido escrito directamente en forma de subrutina (entre la etiqueta LBL 1 y RTN). En el mismo se emplea la memoria 15 como el contador del ciclo. Y a finalizar el proceso deja en la pila (por conveniencia) la matriz original, la matriz resultante de la división sintética y los dos divisores:



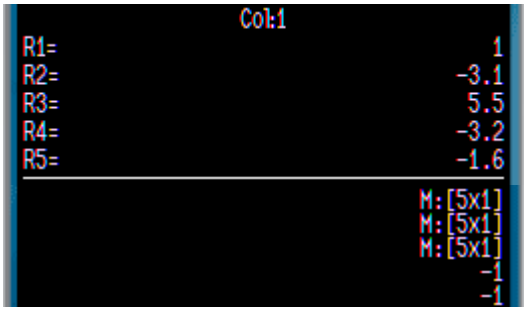
Mandando a este programa el vector de coeficientes del polinomio (8.1) y los divisores -1, -1 (“r” y “s”):



Obtenemos:



Que son los mismos resultados obtenidos manualmente. Ahora llevando a cabo una segunda división sintética obtenemos:



Que también concuerda con los resultados calculados manualmente (excepto que en este caso se calcula también el valor del elemento c_{n+1}). Puesto que ambos resultados son correctos, podemos suponer que el programa ha sido correctamente elaborado. Observe que en la pila quedan los coeficientes del polinomio original, los coeficientes de la primera división, los coeficientes de la segunda división y los divisores.

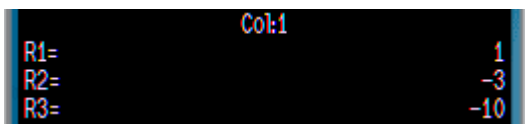
Si se necesita ejecutar esta subrutina, una vez que se haya añadido el resto del código correspondiente al método de Bairstow, lo que se puede hacer es escribir la instrucción “GTO” (GTO 1) al principio del programa, de manera que salte directamente a esta subrutina.

8.1.2. Ejemplos

1. Divida el polinomio que se presenta al pie de la pregunta entre x^2+x+1, luego divida el polinomio resultante (sin su último elemento) entre x^2+x+1.

$$P_5(x) = x^5 - 3x^4 - 10x^3 + 10x^2 + 44x + 48 = 0$$

Mandamos a “bairs” los siguientes datos:



```
R4=                    10
R5=                    44
R6=                    48
-----------------------------
                  M:[6x1]
                      -1
                      -1
```

Con lo que obtenemos:

```
                  Col:1
R1=                     1
R2=                    -4
R3=                    -7
R4=                    21
R5=                    30
R6=                    -3
-----------------------------
                  M:[6x1]
                  M:[6x1]
                      -1
                      -1
```

Para la segunda división, simplemente volvemos a hacer correr el programa “bairs”, con lo que obtenemos:

```
                  Col:1
R1=                     1
R2=                    -5
R3=                    -3
R4=                    29
R5=                     4
R6=                   -36
-----------------------------
                  M:[6x1]
                  M:[6x1]
                  M:[6x1]
                      -1
                      -1
```

Y como ya se hizo notar quedan en la pila los coeficientes del polinomio, los resultados de la primera división, los resultados de la segunda división y los divisores.

2. Divida el polinomio que se presenta al pie de la pregunta entre x^2+3x-4, luego divida el polinomio resultante entre x^2+3x-4.

$$P_6(x) = x^6 + x^5 - 8x^4 + 231x^3 + 21x^2 - 265x + 150 = 0$$

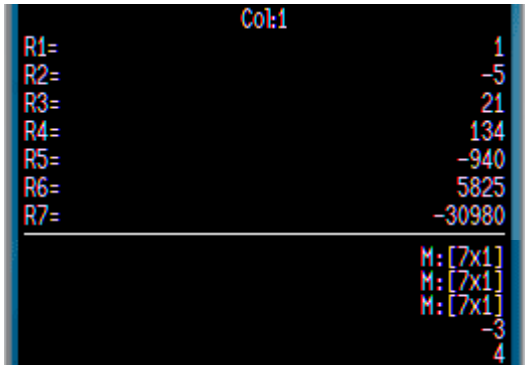
Mandamos los siguientes datos a “bairs”:

```
                  Col:1
R1=                     1
R2=                     1
R3=                    -8
R4=                   231
R5=                    21
R6=                  -265
R7=                   150
-----------------------------
                  M:[7x1]
                      -3
                       4
```

Con lo que se obtiene:

```
                  Col:1
R1=                     1
R2=                    -2
R3=                     2
```

Que son los resultados de la primera división sintética. Para la segunda, simplemente volvemos a hacer correr el programa “bairs”:



8.1.3. Ejercicios

1. Divida el polinomio que se presenta al pie de la pregunta entre x^2-5x+2, luego divida el polinomio resultante entre x^2+3x-4.

$$P_8(x) = x^8 + x^7 - 9x^6 + 13x^5 + 21x^4 - 125x^3 + 77x^2 + 111x - 90 = 0$$

2. Divida el polinomio que se presenta al pie de la pregunta entre x^2+2x-3, luego divida el polinomio resultante entre x^2+3x-4.

$$P_{10}(x) = 12x^{10} - 8x^9 + 6x^7 - 3x^3 + 5x^2 - 3x - 10 = 0$$

8.2. Método de Bairstow

El método de Bairstow nos permite calcular 2 de las raíces de una ecuación polinomial. Para ello se divide el polinomio entre una ecuación de segundo grado y se van cambiando los coeficientes de esta ecuación hasta que el residuo de la división se hace cero, entonces se calculan dos raíces del polinomio con la ecuación de segundo grado que hace cero el residuo.

El procedimiento puede ser repetido con el polinomio residual (cuyo grado es “n-2”) encontrándose así otras dos soluciones y continuar de esa manera hasta encontrar todas las soluciones del polinomio. No obstante y al igual que ocurre con el método de Newton - Raphson, este procedimiento tiene el inconveniente de ir propagando el error de los cálculos previos, por esta razón, generalmente se emplea el polinomio original, y se van encontrando las raíces (por pares) empleando valores iniciales cercanos a las soluciones buscadas. Los valores iniciales normalmente se encuentran con otros métodos, como el método QD estudiado en el tema anterior.

Otra de las desventajas de este método radica en que no siempre converge hacia las soluciones y ello depende en gran medida de los valores iniciales asumidos.

El fundamento del método es el siguiente: al dividir el polinomio general, entre la ecuación de segundo grado: x^2+rx+s, resulta:

$$\frac{P_n(x)}{x^2+r+s} = Q_{n-2}(x)(x^2+rx+s) + resíduo = 0 \qquad (8.3)$$

Entonces, el método de *Bairstow* busca los valores de *"r"* y *"s"* que hacen el residuo cero (o casi cero). Cuando esto sucede la ecuación (3) queda igualada a cero y se cumple que:

$$\begin{aligned} Q_{n-2}(x) &= 0 \\ x^2+rx+s &= 0 \end{aligned} \qquad (8.4)$$

Entonces dos de las raíces (soluciones) pueden ser calculadas resolviendo la ecuación de segundo grado. Posteriormente es posible encontrar otras dos raíces repitiendo el procedimiento con el polinomio Q_{n-2} y continuar de esa manera hasta determinar todas las raíces del polinomio original. Aunque como ya se dijo este procedimiento tiene el inconveniente de ir propagando el error cometido en los cálculos previos.

La división del polinomio $P_n(x)$ entre la ecuación de segundo grado es simplemente la división sintética doble. Como ya se dijo, el objetivo del método de *Bairstow* es hacer cero el residuo: $b_n(x-r)+b_{n+1}$. Para ello, se expanden en series de Taylor los coeficientes b_n y b_{n+1} como funciones de "r" y "s":

$$\begin{aligned} b_n(r+\Delta r, s+\Delta s) &= b_n(r,s) + \frac{\partial b_n(r,s)}{\partial r}\Delta r + \frac{\partial b_n(r,s)}{\partial s}\Delta s + \frac{1}{2!}\left(\frac{\partial b_n(r,s)}{\partial r}\Delta r + \frac{\partial b_n(r,s)}{\partial s}\Delta s\right)^2 + ...\infty = 0 \\ b_{n+1}(r+\Delta r, s+\Delta s) &= b_{n+1}(r,s) + \frac{\partial b_{n+1}(r,s)}{\partial r}\Delta r + \frac{\partial b_{n+1}(r,s)}{\partial s}\Delta s + \frac{1}{2!}\left(\frac{\partial b_{n+1}(r,s)}{\partial r}\Delta r + \frac{\partial b_{n+1}(r,s)}{\partial s}\Delta s\right)^2 + ...\infty = 0 \end{aligned} \qquad (8.5)$$

Donde *"Δr"* y *"Δs"* son los valores que deben restarse a *"r"* y *"s"* para que tanto *"b_n"* como *"b_{n+1}"* se igualen a cero. Por supuesto, no es posible en la práctica resolver este sistema (que es más complicado aún que el problema original), por eso sólo se toman los términos de primer orden con lo que el sistema se simplifica a:

$$\begin{aligned} b_n + \frac{\partial b_n}{\partial r}\Delta r + \frac{\partial b_n}{\partial s}\Delta s &= 0 \\ b_{n+1} + \frac{\partial b_{n+1}}{\partial r}\Delta r + \frac{\partial b_{n+1}}{\partial s}\Delta s &= 0 \end{aligned} \qquad (8.6)$$

Entonces si se calculan las derivadas de la ecuación (8.6), la misma es simplemente un sistema de dos ecuaciones lineales con 2 incógnitas.

Las derivadas parciales pueden ser calculadas realizando una segunda división sintética del polinomio residual (sin incluir el último coeficiente) entre la ecuación cuadrática x^2+rx+s. Si denominamos *"c"* a los coeficientes resultantes de esta segunda división, entonces las derivadas parciales son:

$$\begin{aligned} \frac{\partial b_i}{\partial r} &= c_{i-1} \\ \frac{\partial b_i}{\partial s} &= c_{i-2} \end{aligned} \qquad (8.7)$$

Reemplazando estas relaciones la ecuación (8.6) resulta:

$$\begin{aligned} c_{n-1} * \Delta r + c_{n-2} * \Delta s &= -b_n \\ c_n * \Delta r + c_{n-1} * \Delta s &= -b_{n+1} \end{aligned} \qquad (8.8)$$

Que puede ser resuelta aplicando la regla de Cramer:

$$\Delta r = \begin{vmatrix} -b_n & c_{n-2} \\ -b_{n+1} & c_{n-1} \\ \hline c_{n-1} & c_{n-2} \\ c_n & c_{n-1} \end{vmatrix} = \frac{b_{n+1} * c_{n-2} - b_n * c_{n-1}}{c_{n-1}^2 - c_n * c_{n-2}}$$
$$\Delta s = \frac{-b_n - c_{n-1} * \Delta r}{c_{n-2}} \qquad (8.9)$$

Sin embargo, puesto que estos valores han sido calculados despreciando los términos de segundo orden y superiores, son solamente valores aproximados, que al ser sumados a "*r*" y "*s*" no igualan "b_n" y "b_{n+1}" a cero, aunque si los aproximan a este valor.

Por lo tanto, el cálculo de los valores de "*r*" y "*s*" que hacen "b_n" y "b_{n+1}" cero, es iterativo: comenzando con valores iniciales asumidos se calcula por división sintética, los coeficientes *"b"* y se verifica si *"b_n"* y *"b_{n+1}"* son cero (o casi cero), de ser así el proceso termina habiéndose encontrado los valores de "r" y "s" que hacen cero el residuo, caso contrario se calculan los coeficientes *"c"*, mediante otra división sintética, y con los mismos los valores de *"Δr"* y *"Δs"*. Si estos valores son cercanos a cero el proceso concluye, caso contrario se calculan nuevos valores de "r" y "s": $r=r+\Delta r$, $s=s+\Delta s$ y se repite el proceso.

8.2.1. Programa

Antes de comenzar el programa propiamente es necesario convertir los valores aproximados asumidos en los correspondientes valores de "r" y "s". Para ello realizamos la siguiente multiplicación:

$$(x-x_1)(x-x_2)=0$$
$$x^2 - x_2x - x_1x + x_1x_2 = 0 \qquad (8.10)$$
$$x^2 - (x_1+x_2)x + x_1x_2 = 0$$

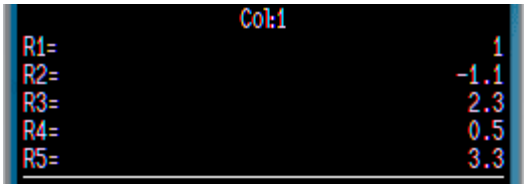
Donde "x_1" y "x_2" son los valores (o soluciones) asumidas. Comparando esta ecuación con la ecuación de segundo grado empleada en el método de Bairstow: $x^2-rx-s=0$, vemos que se cumplen las siguientes relaciones:

$$r = x_1 + x_2$$
$$s = -x_1x_2 \qquad (8.11)$$

Para elaborar el programa que automatiza el proceso descrito previamente trabajaremos con la ecuación (8.1) que recordemos es la siguiente:

$$P_4(x) = x^4 - 1.1x^3 + 2.3x^2 + 0.5x + 3.3 = 0$$

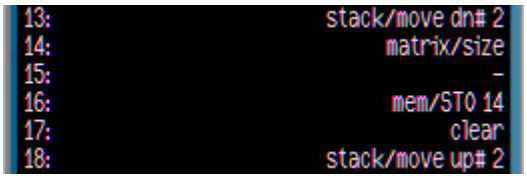
Siendo los valores asumidos x_1=-0.5+0.9I y x_2=-0.5-0.9I. Por lo tanto los valores que deberán estar en la pila al momento de comenzar a programar el método son:



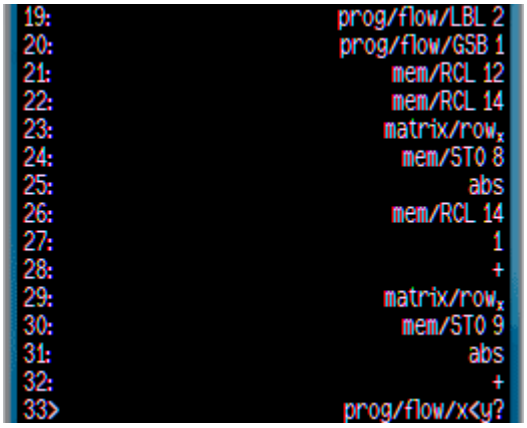
Al igual que en el método de Newton, la precisión y el límite de iteraciones no cambian frecuentemente, por lo que los mismos se establecen en el programa y se guardan en las posiciones de memoria 12 y 13 respectivamente. La parte del programa donde se realizan estas asignaciones y se calculan los valores iniciales de "r", "s" en base a los valores asumidos es la siguiente, la misma que debe ser añadida, al programa "bairs", en el cual se implementó en el acápite anterior la división sintética doble (se recalca que no debe ser borrado ese código, sino añadir el siguiente antes del mismo).

Colocando la instrucción "STOP" (mode/prog/flow/STOP) después de este paso y haciendo correr el programa hasta esta parte se obtiene:

Que son los valores iniciales de "r" y "s". Ahora se determina el grado de la matriz y se guarda en la memoria 14:



A partir de este punto comienza el proceso iterativo, por lo que en esta parte se coloca una etiqueta y se programa el método QD, es decir se programan las ecuaciones (8.9), donde las llamadas a GSB 1 son las llamadas a la división sintética doble que ya está programada:



```
34:                    prog/flow/GTO 3
35:                             clear
36:                   stack/move up# 3
37:                    prog/flow/GSB 1
38:                   stack/move dn# 4
39:                         mem/RCL 9
40:                        mem/RCL 14
41:                                 2
42:                                 -
43:                        matrix/row_x
44:                        mem/STO 11
45:                                 *
46:                         mem/RCL 8
47:                        mem/RCL 14
48:                                 1
49:                                 -
50:                        matrix/row_x
51:                        mem/STO 10
52:                                 *
53:                                 -
54:                        mem/RCL 10
55:                                 x²
56:                        mem/RCL 14
57:                        matrix/row_x
58:                        mem/RCL 11
59:                                 *
60:                                 -
61:                                 /
62:                         mem/STO 9
63:                               abs
64:                         mem/RCL 8
65:                               +/-
66:                        mem/RCL 10
67:                         mem/RCL 9
68:                                 *
69:                                 -
70:                        mem/RCL 11
71:                                 /
72:                         mem/STO 8
73:                               abs
74:                                 +
75:                   stack/move dn# 4
76:                             clear
77:                    prog/flow/x<y?
78:                    prog/flow/GTO 3
79:                             clear
80:                             clear
81:                         mem/RCL 8
82:                                 +
83:                   stack/move up# 2
84:                         mem/RCL 9
85:                                 +
86:                   stack/move up# 2
87:                             clear
88:                   prog/flow/DSE 13
89:                    prog/flow/GTO 2
90:                                 0
91:                   stack/move up# 2
92:                                 0
93:                                 0
94:                                 /
95:                   stack/move up# 4
```

```
96:                    0
97:                    0
98:          prog/flow/LBL 3
99:                clear
100:               clear
101:                 +/-
102:          stack/x≠y
103:                 +/-
104:          stack/x≠y
105:    stack/move dn# 2
106:               clear
107:    stack/move dn# 2
108:               clear
```

Si después de este punto se coloca un “STOP”:

```
109:          prog/flow/STOP
```

Y se hace correr el programa con los datos iniciales, se obtiene:

```
                     0.9
                     1.1
```

Que son los valores finales de “r” y “s” (con el error establecido de $1x10^{-12}$). Ahora con estos valores y la ecuación cuadrática: $x^2-rx-s=0$, calculamos dos de las soluciones del polinomio. Para ello volvemos a copiar el programa “cuad” a partir de este punto, sin embargo, dado que el primer coeficiente siempre es 1 (es decir “a” siempre tiene un valor igual a 1), podemos simplificar un poco más el cálculo de la ecuación cuadrática como sigue (borrando por supuesto el “STOP”):

```
109:    stack/RCL st# 1
110:                 x²
111:                 4
112:    stack/RCL st# 2
113:                 *
114:                 -
115:                √x
116:    stack/RCL st# 2
117:      prog/util/sgn
118:                 *
119:    stack/move dn# 2
120:                 +
121:                -2
122:                 /
123:              ENTER
124:    stack/move up# 2
125:                 /
126:          stack/x≠y
127:       prog/flow/RTN
```

Con ello se tendría concluido el método de Bairstow, no olvidando que a continuación debe encontrarse la división sintética doble, programada en el acápite anterior, con ello el programa tiene en total 173 pasos.

Ahora haciendo correr el programa con los datos iniciales se obtiene:

```
          -0.45+0.9473647660748208i
          -0.45-0.9473647660748208i
```

8.2.2. Ejemplos

3. Encuentre las soluciones reales e imaginarias del polinomio que se presenta al pie de la pregunta. Obtenga los valores iniciales con QD y las soluciones más exactas con Bairstow:

$$x^8 - 7x^7 + 11x^6 + 41x^5 - 183x^4 + 231x^3 - 21x^2 - 265x - 150 = 0$$

Para obtener los valores iniciales, mandando a "qd" la matriz (y guardamos la misma en memoria, por ejemplo la memoria 5, pues será necesaria luego para calcular las soluciones más precisas con Bairstow):

```
                Col:1
R1=                 1
R2=                -7
R3=                11
R4=                41
R5=              -183
R6=               231
R7=               -21
R8=              -265
R9=              -150
                M:[9x1]
```

Con lo que obtenemos:

```
                Col:1
R1=                 3.382352737341295
R2= 2.579653937492198-1.703085851092344i
R3= 2.579653937492198+1.703085851092344i
R4=                -2.894155858029523
R5= 1.199121973240708-1.639261227675233i
R6= 1.199121973240708+1.639261227675233i
R7=-0.5228743503887914+0.30423654736033i
R8=-0.5228743503887914-0.30423654736033i
                M:[8x1]
```

Ahora empleamos estas soluciones como valores iniciales del método de Bairstow, teniendo el cuidado de mandar siempre las soluciones conjugadas como pares conjugados. Así, primero mandamos como valores iniciales la primera y 4 soluciones reales:

```
                Col:1
R1=                 1
R2=                -7
R3=                11
R4=                41
R5=              -183
R6=               231
R7=               -21
R8=              -265
R9=              -150
                M:[9x1]
  3.382352737341295
 -2.894155858029523
```

Con lo que obtenemos:

```
                M:[8x1]
 -3.024270576725285
  3.382353975978588
```

Que son las mismas soluciones obtenidas con el método de Newton Raphson en el tema anterior.

Para las dos primeras soluciones imaginarias mandamos a “qd”:

```
                Col:1
R1=                                1
R2=                               -7
R3=                               11
R4=                               41
R5=                             -183
R6=                              231
R7=                              -21
R8=                             -265
R9=                             -150
                                M:[9x1]
    2.579653937492198-1.703085851092344i
    2.579653937492198+1.703085851092344i
```

Con lo que obtenemos

```
                                M:[8x1]
    2.644710677521432-1.649034226910478i
    2.644710677521432+1.649034226910478i
```

Que son las mismas soluciones calculadas con el método de Newton. Procediendo de manera similar obtenemos las otras soluciones:

```
                Col:1
R1=                  -3.024270576725285
R2=                   3.382353975978588
R3= 2.644710677521432-1.649034226910478i
R4= 2.644710677521432+1.649034226910478i
R5= 1.199121973240708-1.639261227675233i
R6= 1.199121973240708+1.639261227675233i
R7=-0.5228743503887914+0.304236547360331
R8=-0.5228743503887914-0.304236547360331
```

Que son las mismas soluciones obtenidas con Newton, sólo que en este caso se han reordenado un poco a fin de encontrar siempre pares conjugados en el caso de las soluciones complejas.

4. Encuentre las soluciones reales e imaginarias del polinomio que se presenta al pie de la pregunta. Obtenga los valores iniciales con QD y las soluciones más exactas con Bairstow:

$$x^8 - 60x^7 + 1554x^6 - 22680x^5 + 203889x^4 - 1155420x^3 + 4028156x^2 - 7893840x + 6652800 = 0$$

Este caso es muy similar al anterior, básicamente sólo cambian los coeficientes del polinomio. Para obtener los valores iniciales, mandamos la siguiente matriz a “qd”:

```
                Col:1
R1=                                1
R2=                              -60
R3=                             1554
R4=                            -22680
R5=                            203889
R6=                          -1155420
R7=                           4028156
R8=                          -7893840
R9=                           6652800
                                M:[9x1]
```

Con lo que obtenemos las soluciones aproximadas:

```
              Col:1
R1=  11.00000244253071
R2=  9.999998527465296
R3=  8.999999228889507
R4=  7.999999822560593
R5=  6.999999979554382
R6=  5.999999999012767
R7=  4.999999999986768
R8=  3.999999999999979

                M:[8x1]
```

Ahora, tomando pares de valores como valores iniciales, calculamos las soluciones más exactas con el método de Bairstow, así para las dos primeras soluciones mandamos a “bairs”:

```
              Col:1
R1=                1
R2=              -60
R3=             1554
R4=           -22680
R5=           203889
R6=         -1155420
R7=          4028156
R8=         -7893840
R9=          6652800
                M:[9x1]
  11.00000244253071
  9.999998527465296
```

Con lo que se obtiene:

```
                M:[8x1]
  9.999999999999915
  11.00000000000001
```

Que básicamente son las mismas soluciones obtenidas con Newton. Procediendo de similar manera con los otros valores calculamos las restantes soluciones:

```
              Col:1
R1=  9.999999999999915
R2=  11.00000000000001
R3=  8.000000000000172
R4=  8.999999999999838
R5=  6.000000000000053
R6=  6.999999999999842
R7=  3.999999999999999
R8=  5.000000000000009
```

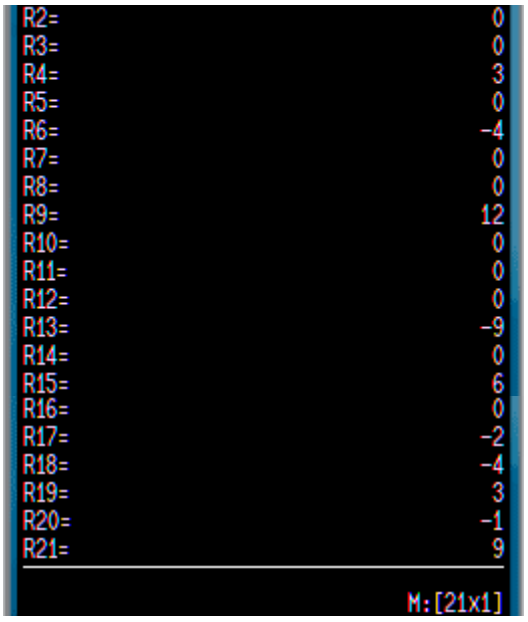
Que esencialmente son las mismas soluciones obtenidas con Newton, sólo que un tanto reordenadas porque el método de la raíz cuadrada siempre devuelve las soluciones ordenadas ascendentemente.

5. Encuentre las soluciones reales e imaginarias del polinomio que se presenta al pie de la pregunta. Obtenga los valores iniciales con QD y las soluciones más exactas con Bairstow:

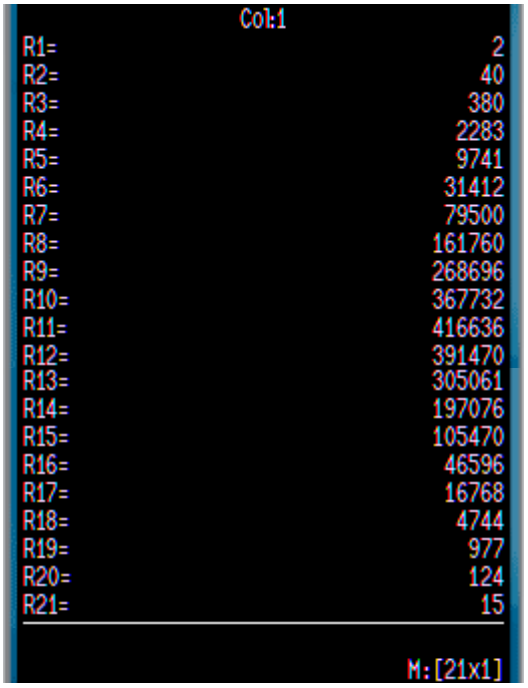
$$2x^{20}+3x^{17}-4x^{15}+12x^{12}-9x^{8}+6x^{6}-2x^{4}-4x^{3}+3x^{2}-x+9=0$$

Como este polinomio tiene coeficientes cero, mandamos primero el vector de coeficientes del polinomio “cvar”:

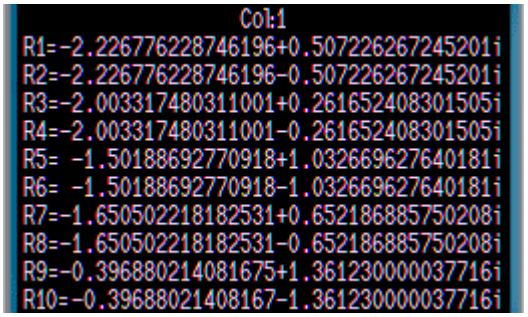
```
              Col:1
R1▶                2
```

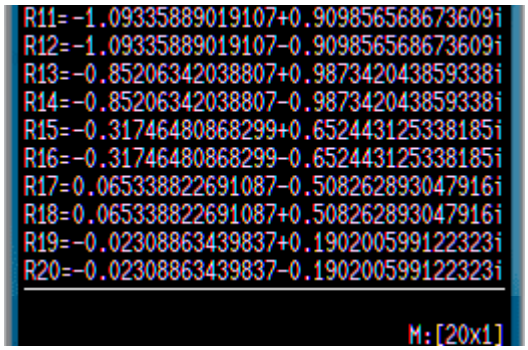


Con lo que obtenmos:

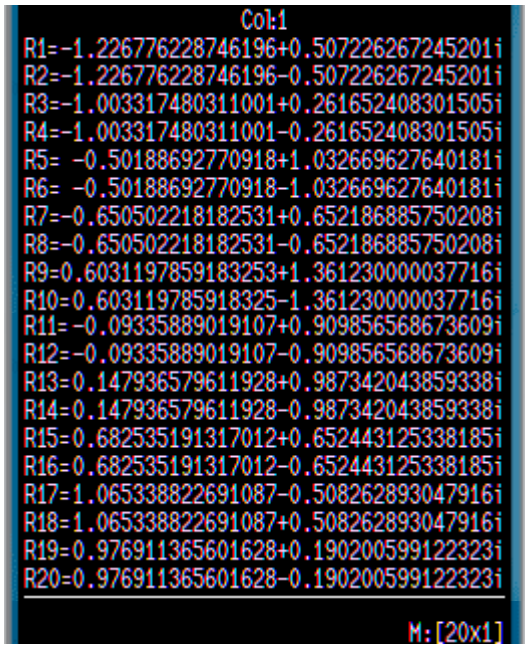


Ahora mandamos este vector a “qd”:

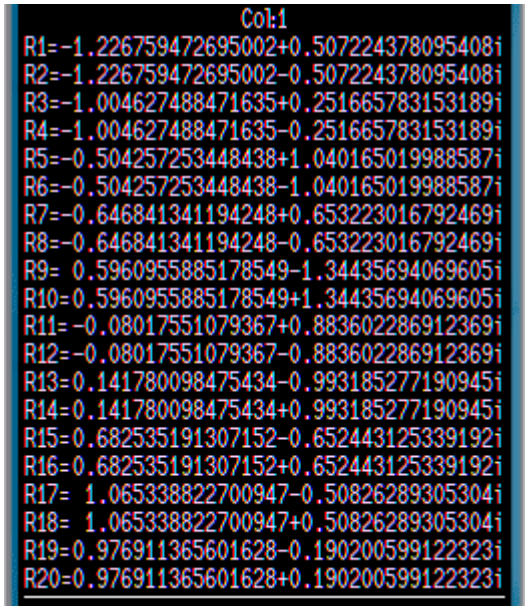




Pero para obtener las soluciones del polinomio original debemos sumar el vector unidad (con 20 elementos) a este resultado, obteniendo así:



Empleando estas soluciones como valores iniciales del método de Bairstow, obtenemos:



Que una vez más son esencialmente las mismas soluciones encontradas con Newton en el tema anterior.

En todos estos ejemplos se han resuelto ecuaciones de grado par, cuando se presenta una ecuación de grado impar no todas las soluciones pueden ser calculadas con Bairstow, por lo que por lo menos una de ellas (una de las soluciones reales) debe ser calculada con el método de Newton Raphson.

8.2.3. Ejercicios

3. Encuentre las soluciones reales e imaginarias del polinomio que se presenta al pie de la pregunta. Obtenga los valores iniciales con QD y las soluciones más exactas con Bairstow:

$$P_4(x) = x^4 - 16x^3 + 72x^2 - 96x + 24 = 0$$

4. Encuentre las soluciones reales e imaginarias del polinomio que se presenta al pie de la pregunta. Obtenga los valores iniciales con QD y las soluciones más exactas con Bairstow y una con Newton:

$$P_9(x) = 5x^9 - 8x^7 + 4x^6 - 2x^5 + x^4 - 6x^3 + 3x^2 - 2x - 80 = 0$$

5. Encuentre las soluciones reales e imaginarias del polinomio que se presenta al pie de la pregunta. Obtenga los valores iniciales con QD y las soluciones más exactas con Bairstow:

$$P_{10}(x) = 12x^{10} - 8x^9 + 6x^7 - 3x^3 + 5x^2 - 3x - 10 = 0$$

6. Encuentre las soluciones reales e imaginarias del polinomio que se presenta al pie de la pregunta. Obtenga los valores iniciales con QD y las soluciones más exactas con Bairstow:

$$P_6(x) = x^6 + x^5 - 8x^4 + 231x^3 + 21x^2 - 265x + 150 = 0$$

7. Encuentre las soluciones reales e imaginarias del polinomio que se presenta al pie de la pregunta. Obtenga los valores iniciales con QD y las soluciones más exactas con Bairstow:

$$P_8(x) = x^8 + x^7 - 9x^6 + 13x^5 + 21x^4 - 125x^3 + 77x^2 + 111x - 90 = 0$$

9. SISTEMAS DE ECUACIONES LINEALES - 1

Con frecuencia en el campo de la ingeniería es necesario resolver sistemas de ecuaciones lineales que tienen entre 2 y cientos de miles de ecuaciones lineales.

Los métodos que permiten resolver ecuaciones lineales pueden clasificarse en dos: a) *No iterativos o directos*, que permiten resolver sistemas de ecuaciones que cuentan con algunos cientos de ecuaciones lineales (no por las limitaciones de los métodos, sino por los errores de redondeo) y b) *Los métodos iterativos*, que permiten resolver sistemas con hasta cientos de miles de ecuaciones lineales, aunque como normalmente sucede con este tipo de métodos, no siempre logran la convergencia.

En este tema estudiaremos algunos de los métodos pertenecientes a la primera categoría.

9.1. Solución de ecuaciones lineales con Calc-Java

Calc-Java no cuenta con una instrucción específica para resolver sistemas de ecuaciones lineales, sin embargo permite calcular la inversa de una matriz y realizar otras operaciones como la multiplicación de matrices, con las cuales es posible resolver un sistema de ecuaciones lineales.

Si "A" es la matriz de los coeficientes, "B" la matriz (columna) de las constantes, y "x" el vector (columna) de soluciones, el sistema de ecuaciones lineales puede representarse matricialmente de la siguiente forma:

$$A*x = B \qquad (9.1)$$

Multiplicando ambos miembros de esta ecuación por la matriz inversa (A^{-1}), se obtiene.

$$A^{-1}*A*x = A^{-1}*B$$

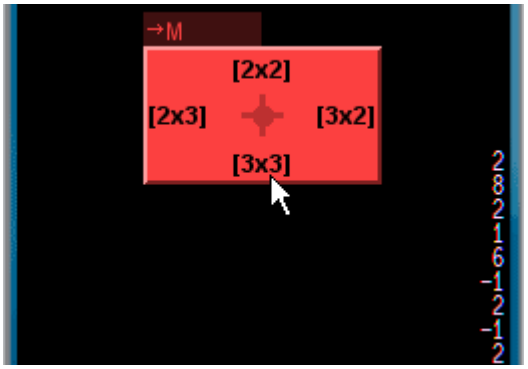
$$x = A^{-1}*B \qquad (9.2)$$

Por lo tanto es posible encontrar las soluciones del sistema (el vector "x") multiplicando la matriz inversa de los coeficientes por la matriz (o vector columna) de las constantes.

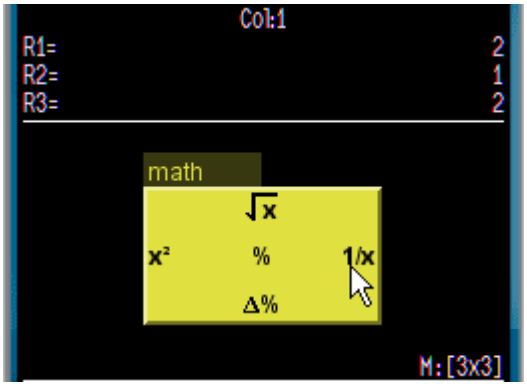
Por ejemplo, para resolver el siguiente sistema de ecuaciones lineales:

$$\begin{aligned} 2x_1 + 8x_2 + 2x_3 &= 14 \\ x_1 + 6x_2 - x_3 &= 13 \\ 2x_1 - x_2 + 2x_3 &= 5 \end{aligned} \qquad (9.3)$$

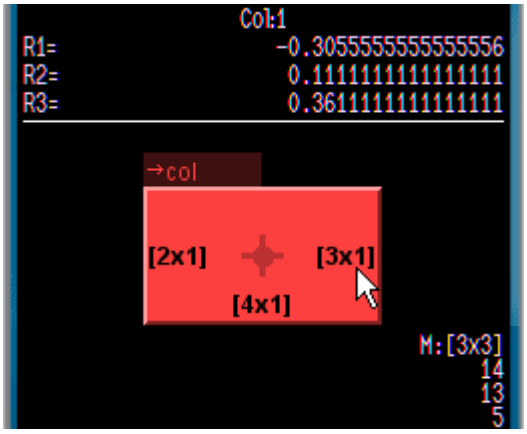
Se crea la matriz de los coeficientes (y se guarda la misma en una posición de memoria):



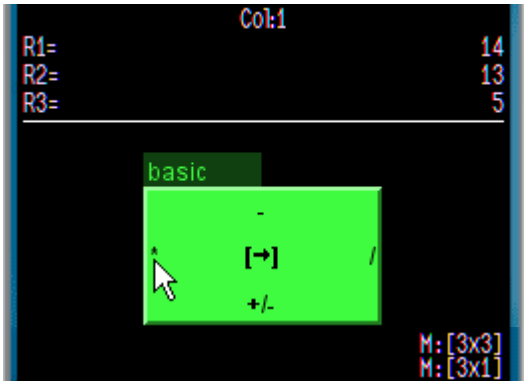
Se obtiene la matriz inversa:



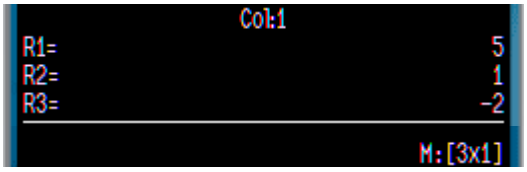
Se crea el vector de las constantes (y se la guarda en otra posición de memoria):



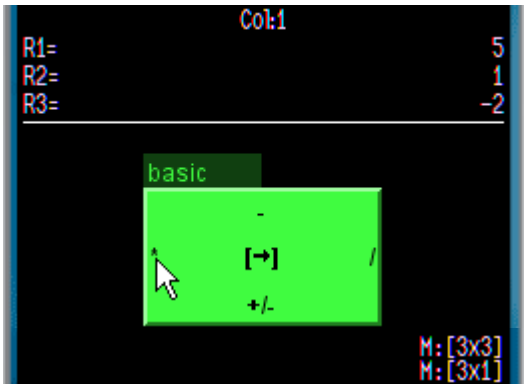
Entonces se multiplican las matrices:

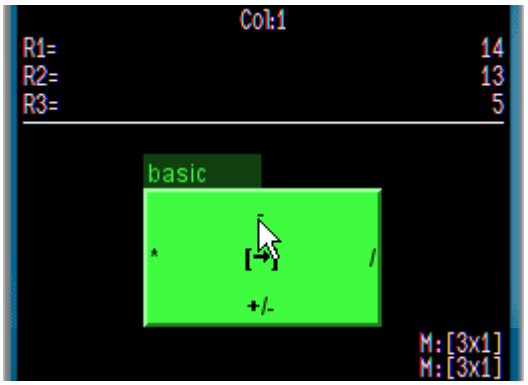


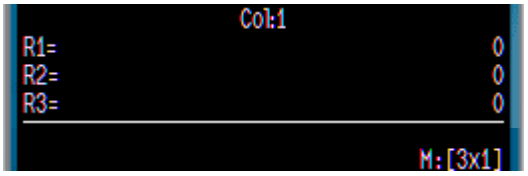
Con lo que se obtienen las soluciones:



Soluciones que pueden ser corroboradas multiplicando la matriz de los coeficientes por esta matriz (la matriz resultante) y restando la matriz de las constantes. Que deberá devolver un vector con ceros pues al reemplazar las soluciones en la matriz, si los resultados son correctos, se deben obtener valores iguales a las constantes:





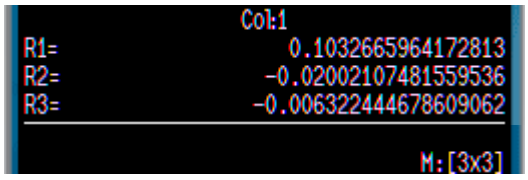


9.1.1. Ejemplos

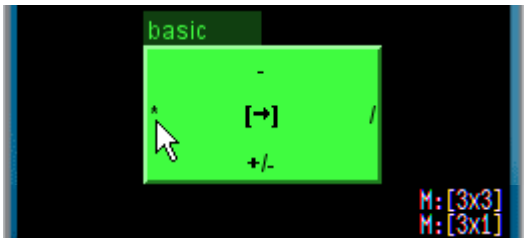
1. Resuelva el siguiente sistema de ecuaciones lineales:

$$\begin{aligned} 10x_1 + x_2 + 2x_3 &= 44 \\ 2x_1 + 10x_2 + x_3 &= 51 \\ x_1 + 2x_2 + 10x_3 &= 61 \end{aligned}$$

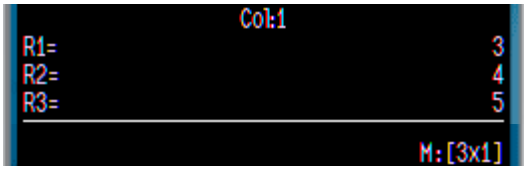
Calculamos la inversa de la matriz de los coeficientes:



Multiplicamos esta matriz por el vector de las constantes:



Con lo que obtenemos las tres soluciones del sistema:



2. Resuelva el siguiente sistema de ecuaciones lineales:

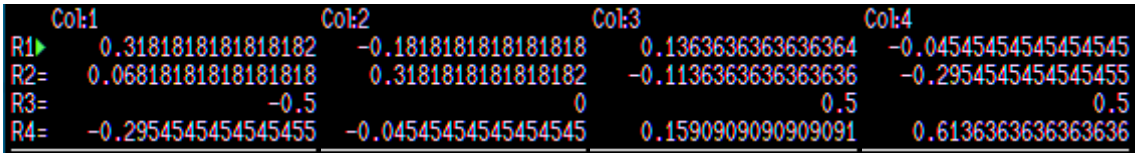
$$\begin{aligned} 3x_1 + 2x_2 - x_3 + 2x_4 &= 0 \\ x_1 + 4x_2 + 2x_4 &= 0 \\ 2x_1 + x_2 + 2x_3 - x_4 &= 1 \\ x_1 + x_2 - x_3 + 3x_4 &= 0 \end{aligned}$$

$$\begin{aligned} 3y_1 + 2y_2 - y_3 + 2y_4 &= -2 \\ y_1 + 4y_2 + 2y_4 &= 2 \\ 2y_1 + y_2 + 2y_3 - y_4 &= 3 \\ y_1 + y_2 - y_3 + 3y_4 &= 4 \end{aligned}$$

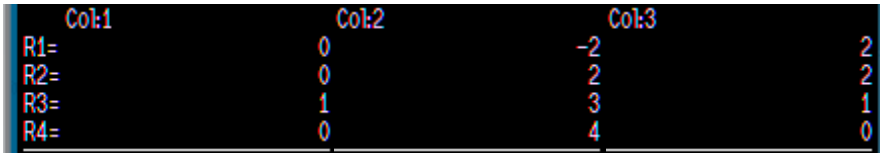
$$\begin{aligned} 3z_1 + 2z_2 - z_3 + 2z_4 &= 2 \\ z_1 + 4z_2 + 2z_4 &= 2 \\ 2z_1 + z_2 + 2z_3 - z_4 &= 1 \\ z_1 + z_2 - z_3 + 3z_4 &= 0 \end{aligned}$$

Como estos tres sistemas tienen los mismos coeficientes pueden ser resueltos al mismo tiempo multiplicando la inversa de la matriz de los coeficientes por la matriz de las constantes.

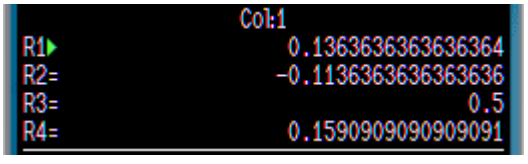
Calculamos primero la matriz inversa de los coeficientes:



Creamos la matriz de las constantes:



Entonces multiplicamos ambas matrices, con lo que obtenemos los resultados, siendo la primera columna los resultados del primer sistema, la segunda del segundo y la tercera del tercer sistema:



9.1.2. Ejercicios

1. Encuentre las soluciones del siguiente sistema de ecuaciones lineales calculando la inversa de la matriz de los coeficientes.

$$\begin{aligned} 3x_1 - x_2 + 2x_3 &= 12 \\ x_1 + 2x_2 + 3x_3 &= 11 \\ 2x_1 - 2x_2 - x_3 &= 2 \end{aligned}$$

2. Encuentre las soluciones del siguiente sistema de ecuaciones lineales calculando la inversa de la matriz de los coeficientes.

$$\begin{aligned} 2x_1 - 2x_2 + 5x_3 &= 13 \\ 2x_1 + 3x_2 + 4x_3 &= 20 \\ 3x_1 - x_2 + 3x_3 &= 10 \end{aligned}$$

3. Encuentre las soluciones del siguiente sistema de ecuaciones lineales calculando la inversa de la matriz de los coeficientes.

$$\begin{aligned} x_1 + x_2 + x_3 + x_4 &= -1 \\ 4x_1 + 5x_2 + 6x_3 + 7x_4 &= 0 \\ 6x_1 + 10x_2 + 15x_3 + 21x_4 &= 0 \\ 12x_1 + 30x_2 + 60x_3 + 105x_4 &= 0 \end{aligned}$$

4. Encuentre las soluciones del siguiente sistema de ecuaciones lineales calculando la inversa de la matriz de los coeficientes.

$$\begin{aligned} 30x_1 + x_2 + 2x_3 + 6x_4 + 5x_5 - 3x_6 &= 12 \\ 2x_1 + 25x_2 + x_3 - x_4 + 2x_5 - 5x_6 &= 24 \\ x_1 + 2x_2 + 33x_3 + 5x_4 - x_5 + 2x_6 &= 17 \\ 3x_1 + 5x_2 + 7x_3 - 45x_4 + 4x_5 + x_6 &= 37 \\ 4x_1 + 3x_2 + 2x_3 + 3x_4 + 40x_5 - 4x_6 &= 23 \\ 5x_1 + 6x_2 - 3x_3 - 4x_4 + 3x_5 + 36x_6 &= 24 \end{aligned}$$

5. Encuentre las soluciones de los siguientes sistemas de ecuaciones lineales calculando la inversa de la matriz de los coeficientes.

$$\begin{aligned} 30x_1 + 1x_2 + 2x_3 + 6x_4 + 5x_5 - 3x_6 &= 12 \\ x_1 + 2x_2 + 33x_3 + 5x_4 - x_5 + 2x_6 &= 7 \\ 3x_1 + 5x_2 + 7x_3 - 45x_4 + 4x_5 + x_6 &= 7 \\ 4x_1 + 3x_2 + 2x_3 + 3x_4 + 40x_5 - 4x_6 &= 3 \\ 5x_1 + 6x_2 - 3x_3 - 4x_4 + 3x_5 + 36x_6 &= 4 \\ 2x_1 + 25x_2 + x_3 - x_4 + 2x_5 - 5x_6 &= 4 \end{aligned}$$

$$\begin{aligned} 30y_1 + 1y_2 + 2y_3 + 6y_4 + 5y_5 - 3y_6 &= 14 \\ y_1 + 2y_2 + 33y_3 + 5y_4 - y_5 + 2y_6 &= 1 \\ 3y_1 + 5y_2 + 7y_3 - 45y_4 + 4y_5 + y_6 &= 3 \\ 4y_1 + 3y_2 + 2y_3 + 3y_4 + 40y_5 - 4y_6 &= 2 \\ 5y_1 + 6y_2 - 3y_3 - 4y_4 + 3y_5 + 36y_6 &= 2 \\ 2y_1 + 25y_2 + y_3 - y_4 + 2y_5 - 5y_6 &= 7 \end{aligned}$$

$$\begin{aligned}
30z_1 + 1z_2 + 2z_3 + 6z_4 + 5z_5 - 3z_6 &= 4\\
z_1 + 2z_2 + 33z_3 + 5z_4 - z_5 + 2z_6 &= 27\\
3z_1 + 5z_2 + 7z_3 - 45z_4 + 4z_5 + z_6 &= 27\\
4z_1 + 3z_2 + 2z_3 + 3z_4 + 40z_5 - 4z_6 &= 13\\
5z_1 + 6z_2 - 3z_3 - 4z_4 + 3z_5 + 36z_6 &= 14\\
2z_1 + 25z_2 + z_3 - z_4 + 2z_5 - 5z_6 &= 14
\end{aligned}$$

9.2. Método de eliminación de Gauss - Jordán

El método de eliminación de Gauss – Jordán, que como se sabe por los estudios de álgebra, es una modificación del método de Gauss, encuentra las soluciones del sistema reduciendo la matriz de los coeficientes a una matriz unidad. Así si aplicamos el método de eliminación de Gauss-Jordán al siguiente sistema de ecuaciones:

$$\begin{aligned}
a_{1,1}x_1 + a_{1,2}x_2 + a_{1,3}x_3 + a_{1,4}x_4 &= c_1\\
a_{2,1}x_1 + a_{2,2}x_2 + a_{2,3}x_3 + a_{2,4}x_4 &= c_2\\
a_{3,1}x_1 + a_{3,2}x_2 + a_{3,3}x_3 + a_{3,4}x_4 &= c_3\\
a_{4,1}x_1 + a_{4,2}x_2 + a_{4,3}x_3 + a_{4,4}x_4 &= c_4
\end{aligned} \qquad (9.4)$$

Al final del proceso se transforma en:

$$\begin{aligned}
x_1 + 0 + 0 + 0 &= c'_1\\
0 + x_2 + 0 + 0 &= c'_2\\
0 + 0 + x_3 + 0 &= c'_3\\
0 + 0 + 0 + x_4 &= c'_4
\end{aligned} \qquad (9.5)$$

Donde las soluciones del sistema son los valores del vector c’. En forma matricial, el proceso puede ser representado de la siguiente forma:

$$\begin{vmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5}\\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5}\\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5}\\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} & a_{4,5} \end{vmatrix} \xrightarrow{GAUSS-JORDAN} \begin{vmatrix} 1 & 0 & 0 & 0 & a'_{1,5}\\ 0 & 1 & 0 & 0 & a'_{2,5}\\ 0 & 0 & 1 & 0 & a'_{3,5}\\ 0 & 0 & 0 & 1 & a'_{4,5} \end{vmatrix} \qquad (9.6)$$

En este caso, las soluciones del sistema se encuentran en la última columna de la matriz aumentada “a” (la columna 5).

Para convertir la matriz aumentada en una matriz unidad, se deben efectuar reducciones de filas (donde se convierten en unos los elementos de la diagonal principal) y reducciones de columnas (donde se convierten en ceros los elementos que se encuentran por debajo de la diagonal principal). Estas operaciones se deben efectuar primero para la fila y columna 1, luego para la fila y columna 2, luego para la fila y columna 3 y así sucesivamente hasta llegar a la última fila del sistema.

Puesto que los valores de la matriz sólo se emplean una vez en los cálculos, la matriz resultante puede ser almacenada en la matriz original, ahorrando así memoria.

En las siguientes ecuaciones llamaremos *“p” (pivote)* al contador que determina la fila y columna que se está reduciendo, *“m”* al número de ecuaciones del sistema y *“n”* al número de columnas de la matriz aumentada.

Para reducir el elemento a_{pp} a 1, dividimos los elementos de la fila p entre a_{pp}, es decir:

$$a_p = \frac{a_p}{a_{p,p}} \qquad (9.7)$$

Donde "a_p" es la fila "p" de la matriz "a".

Para reducir a cero los elementos que se encuentra en la columna "p" por encima y por debajo del elemento "a_{pp}" (que con la anterior operación ya se ha convertido en 1), restamos a cada una de las filas los elementos de la fila "p" multiplicada por el valor del elemento "$a_{i,p}$", siendo "i" el número de fila que se está reduciendo, es decir:

$$a_i = a_i - a_p * a_{i,p} \quad \{i = 1 \rightarrow m,\ i \neq p \qquad (9.8)$$

Donde "a_i" es la fila "i" de la matriz "a". Donde "p" varía desde 1 hasta "m".

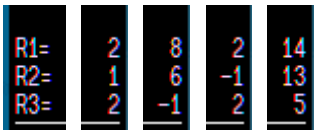
9.2.1. Programa

Primero programaremos las ecuaciones (9.7) y (9.8), que constituyen el método en sí y más tarde añadiremos código a este programa para tomar en cuenta algunos casos como la división entre cero y los errores de redondeo.

Emplearemos como sistema de prueba el (9.3):

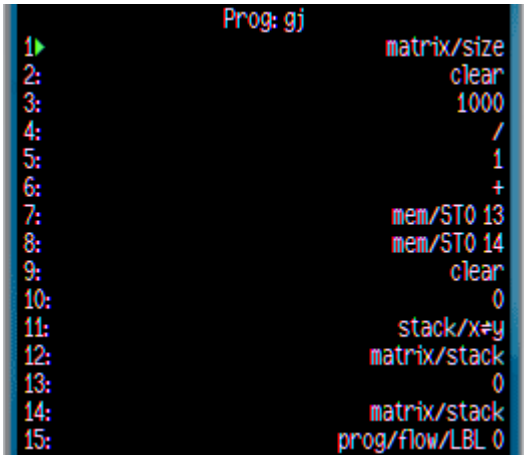
$$\begin{aligned} 2x_1 &+ 8x_2 &+ 2x_3 &= 14 \\ x_1 &+ 6x_2 &- x_3 &= 13 \\ 2x_1 &- x_2 &+ 2x_3 &= 5 \end{aligned} \qquad (9.3)$$

Pero en esta ocasión creamos la matriz aumentada, es decir la matriz de los coeficientes más la columna de las constantes (una matriz de 3x4):



Básicamente lo que se hace es aplicar las ecuaciones mencionadas, pero, para emplear el mismo procedimiento en todos los casos se añade una fila de ceros al principio y al final de la matriz, pues de otra forma no sería posible hacer las operaciones con la primera y última filas.

Como son dos ciclos, en el programa se emplean dos contadores, uno en la memoria 14 para el pivote y otro en la memoria 15 para las filas. El programa elaborado es el siguiente:



```
16:                    mem/RCL 14
17:                  matrix/split
18:                             1
19:                  matrix/split
20:                    stack/x⇄y
21:                    mem/RCL 14
22:                  matrix/col_x
23:                             /
24:                    mem/STO 12
25:                    stack/x⇄y
26:                  matrix/stack
27:                  matrix/stack
28:                    mem/RCL 13
29:                    mem/STO 15
30>                           clear
31:                prog/flow/LBL 1
32:                    mem/RCL 15
33:                    mem/RCL 14
34:                prog/flow/x=y?
35:                prog/flow/GTO 2
36:                           clear
37:                  matrix/split
38:                             1
39:                  matrix/split
40:                    stack/x⇄y
41:                    mem/RCL 14
42:                  matrix/col_x
43:                    mem/RCL 12
44:                             *
45>                             -
46:                    stack/x⇄y
47:                  matrix/stack
48:                  matrix/stack
49:                             0
50:                             0
51:                prog/flow/LBL 2
52:                           clear
53:                           clear
54:                prog/flow/ISG 15
55:                prog/flow/GTO 1
56:                prog/flow/ISG 14
57:                prog/flow/GTO 0
58:                    mem/RCL 14
59:                  matrix/split
60:                           clear
61:                             1
62:                  matrix/split
63:                    stack/x⇄y
64:                           clear
65:                       [prg end]
```

Haciendo correr el programa con la matriz (9.3) se obtiene:

```
R1=   1   0   0   5
R2=   0   1   0   1
R3=  -0  -0   1  -2
```

Y como se puede observar, el método convierte la matriz de los coeficientes en una matriz identidad, quedando los resultados en la columna de las constantes. Por lo tanto las soluciones del sistema de ecuaciones (9.3) son x_1=5, x_2=1 y x_3=-2 (que son los mismos resultados encontrados con la matriz inversa).

9.3. Pivotaje

Al aplicar el método de eliminación de *Gauss* en ocasiones puede ocurrir que el elemento de la diagonal principal (el elemento pivote) sea cero, en cuyo caso se produciría un error por división entre cero. En esos casos se debe realizar un intercambio de filas y/o columnas para evitar que el elemento pivote sea cero.

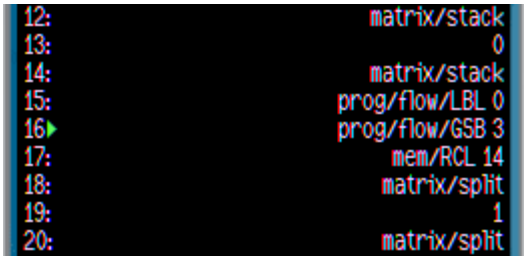
Por otra parte, mientras mayor sea el valor absoluto del elemento pivote, menores son los errores de redondeo, por lo que no sólo se debe cuidar que el elemento pivote no sea cero, sino también que tenga el mayor valor absoluto posible.

Al proceso de búsqueda (en las filas y columnas no reducidas) del elemento con el mayor valor absoluto y el posterior intercambio de filas y columnas para que dicho valor quede en la posición pivote se conoce como *pivotaje total*. Si la búsqueda del mayor valor absoluto se realiza sólo en la columna pivote, el proceso se conoce como *pivotaje parcial*. En la práctica, para que el método de Gauss-Jordan sea de utilidad real, es necesario realizar por lo menos un *pivotaje parcial*.

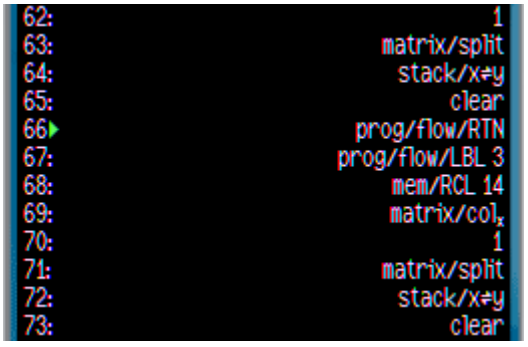
El algoritmo para realizar el *pivotaje parcial* es el siguiente: en la columna pivote, se ubica la fila donde se encuentra el elemento con el mayor valor absoluto, buscando sólo desde la fila pivote hasta la última fila. Una vez ubicado el elemento, se intercambia la fila pivote con la fila donde se encuentra el mayor valor absoluto. Si el mayor valor absoluto es 0, se trata de un sistema homogéneo en cuyo caso se genera un error, pues el sistema no tiene una única solución.

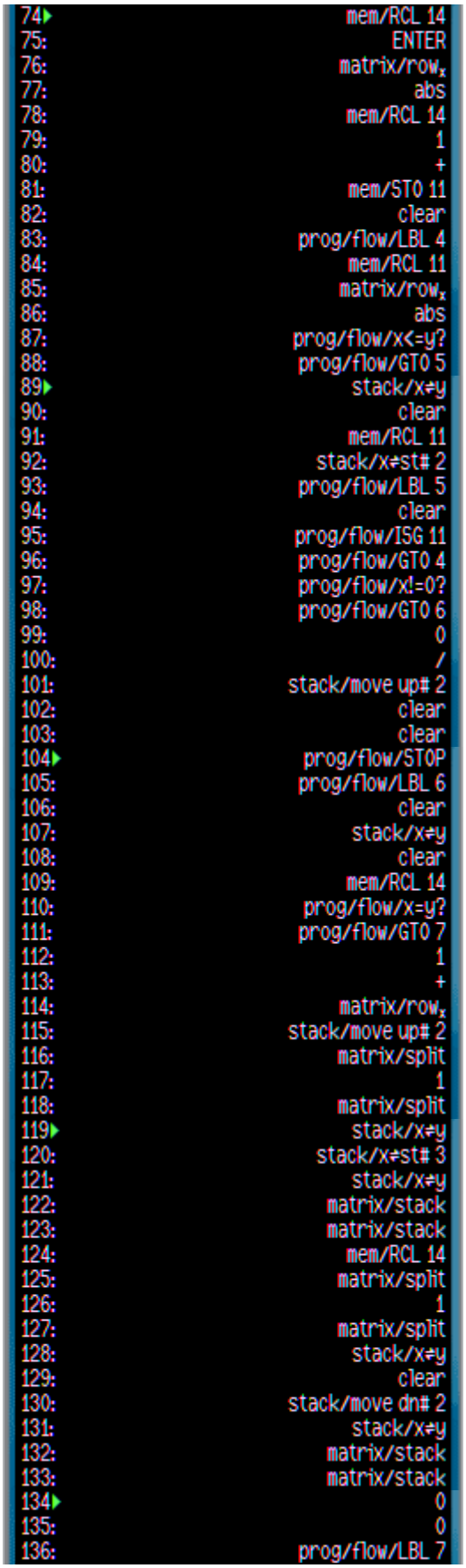
9.3.1. Programa

La parte del programa que realiza el *pivotaje parcial*, se añade al programa anterior como una subrutina, a la cual se llama antes de comenzar el proceso de reducción de filas y columnas, es decir después de la etiqueta 0:



Y la subrutina que realiza el pivotaje parcial, añadida al final del método de Gauss-Jordán (donde se vuelve a mostrar la parte final del programa de Gauss-Jordán para ubicar más fácilmente el código añadido), es la siguiente:




74 mem/RCL 14
75: ENTER
76: matrix/row$_x$
77: abs
78: mem/RCL 14
79: 1
80: +
81: mem/STO 11
82: clear
83: prog/flow/LBL 4
84: mem/RCL 11
85: matrix/row$_x$
86: abs
87: prog/flow/x<=y?
88: prog/flow/GTO 5
89 stack/x⇄y
90: clear
91: mem/RCL 11
92: stack/x⇄st# 2
93: prog/flow/LBL 5
94: clear
95: prog/flow/ISG 11
96: prog/flow/GTO 4
97: prog/flow/x!=0?
98: prog/flow/GTO 6
99: 0
100: /
101: stack/move up# 2
102: clear
103: clear
104 prog/flow/STOP
105: prog/flow/LBL 6
106: clear
107: stack/x⇄y
108: clear
109: mem/RCL 14
110: prog/flow/x=y?
111: prog/flow/GTO 7
112: 1
113: +
114: matrix/row$_x$
115: stack/move up# 2
116: matrix/split
117: 1
118: matrix/split
119 stack/x⇄y
120: stack/x⇄st# 3
121: stack/x⇄y
122: matrix/stack
123: matrix/stack
124: mem/RCL 14
125: matrix/split
126: 1
127: matrix/split
128: stack/x⇄y
129: clear
130: stack/move dn# 2
131: stack/x⇄y
132: matrix/stack
133: matrix/stack
134 0
135: 0
136: prog/flow/LBL 7

Haciendo correr el programa con el sistema de ecuaciones del acápite anterior se obtiene, como era de esperar, los mismos resultados, pero como veremos en los siguientes ejemplos, no ocurre lo mismo con otras matrices, donde si no se realiza el *pivotaje parcial*, no es posible encontrar las soluciones.

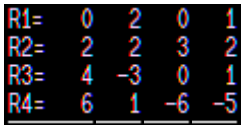
9.3.2. Ejemplos

3. Encuentre las soluciones del siguiente sistema de ecuaciones lineales empleando el método de Gauss-Jordán con pivotaje parcial.

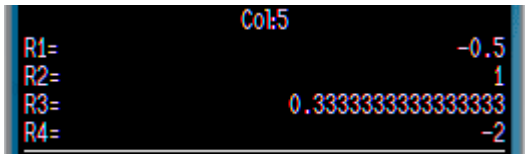
$$\begin{aligned} 2x_2 + x_4 &= 0 \\ 2x_1 + 2x_2 + 3x_3 + 2x_4 &= -2 \\ 4x_1 - 3x_2 + x_4 &= -7 \\ 6x_1 + x_2 - 6x_3 - 5x_4 &= 6 \end{aligned}$$

Este es uno de los casos que no puede ser resuelto si no se lleva a cabo el pivotaje parcial, pues al ser el primer elemento de la primera fila cero, se produciría una división entre cero.

Primero creamos la matriz aumentada:



Y mandamos la misma a “gj” con lo que se obtienen las soluciones en la columna 5:



Por lo tanto las soluciones del sistema son: x_1=-0.5; x_2=1; x_3=0.3333333333333333; x_4=-2.

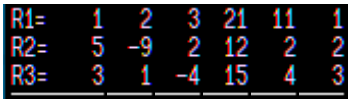
4. Encuentre las soluciones de los siguientes sistemas de ecuaciones lineales empleando el método de Gauss-Jordán con pivotaje parcial.

$$\begin{aligned} x + 2y + 3z &= 21 \\ 5x - 9y + 2z &= 12 \\ 3x + y - 4z &= 15 \end{aligned}$$

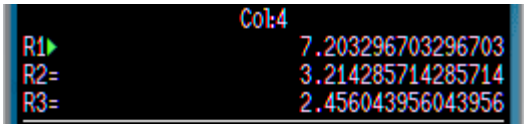
$$\begin{aligned} u + 2v + 3w &= 11 \\ 5u - 9v + 2w &= 2 \\ 3u + v - 4w &= 4 \end{aligned}$$

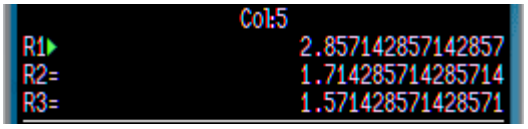
$$\begin{aligned} r + 2s + 3t &= 1 \\ 5r - 9s + 2t &= 2 \\ 3r + s - 4t &= 3 \end{aligned}$$

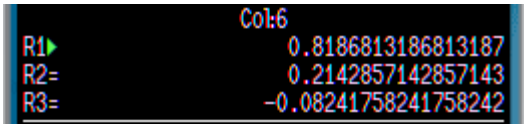
Como los tres sistemas tienen los mismos coeficientes, pueden ser resueltos al mismo tiempo, para ello, creamos la matriz aumentada con tres columnas adicionales: una para cada sistema:



Y mandamos la misma al método de Gauss-Jordán con pivotaje parcial, con lo que obtenemos las tres soluciones de los tres sistemas en las tres últimas columnas:







Por lo tanto, las soluciones son: x = 7.203296703296703; y = 3.214285714285714; z = 2.456043956043956; u = 2.857142857142857; v = 1.714285714285714; w = 1.571428571428571; r = 0.8186813186813187; s = 0.2142857142857143; t = -0.08241758241758242.

9.3.3. Ejercicios

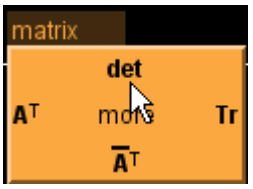
6. Resuelva el ejercicio 1 empleando el método de Gauss-Jordán con pivotaje parcial.
7. Resuelva el ejercicio 2 empleando el método de Gauss-Jordán con pivotaje parcial.
8. Resuelva el ejercicio 3 empleando el método de Gauss-Jordán con pivotaje parcial.
9. Resuelva el ejercicio 4 empleando el método de Gauss-Jordán con pivotaje parcial.
10. Resuelva el ejercicio 5 empleando el método de Gauss-Jordán con pivotaje parcial.

10. SISTEMAS DE ECUACIONES LINEALES - 2

10.1. Solución de ecuaciones lineales empleando Determinantes

Como se sabe del estudio del álgebra de matrices, los determinantes permiten encontrar las soluciones de sistemas de ecuaciones lineales, siendo el método más conocido el de Kramer, sin embargo, dicho método resulta práctico sólo para resolver dos sistemas de ecuaciones lineales, pues para tres o más ecuaciones el número de operaciones que requiere es excesivo, inclusive para una computadora.

Existen sin embargo otros métodos (como el de eliminación de Gauss) que permiten calcular de forma más eficiente el determinante. Calc-Java cuenta con una función para el cálculo de determinantes: “det”, que se encuentra en el menú math\matrix\math\:



Podemos emplear dicha función para calcular las soluciones de un sistema de ecuaciones, recordando que las soluciones se obtienen una a una, dividiendo el determinante de la matriz de los coeficientes con una de sus columnas reemplazada por la columna de las constantes (la columna correspondiente a la solución que se quiere encontrar) entre el determinante de los coeficientes.

Así por ejemplo, si tenemos un sistema con cuatro ecuaciones lineales:

$$\begin{aligned} a_{1,1}x_1 + a_{1,2}x_2 + a_{1,3}x_3 + a_{1,4}x_4 &= c_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + a_{2,3}x_3 + a_{2,4}x_4 &= c_2 \\ a_{3,1}x_1 + a_{3,2}x_2 + a_{3,3}x_3 + a_{3,4}x_4 &= c_3 \\ a_{4,1}x_1 + a_{4,2}x_2 + a_{4,3}x_3 + a_{4,4}x_4 &= c_4 \end{aligned}$$

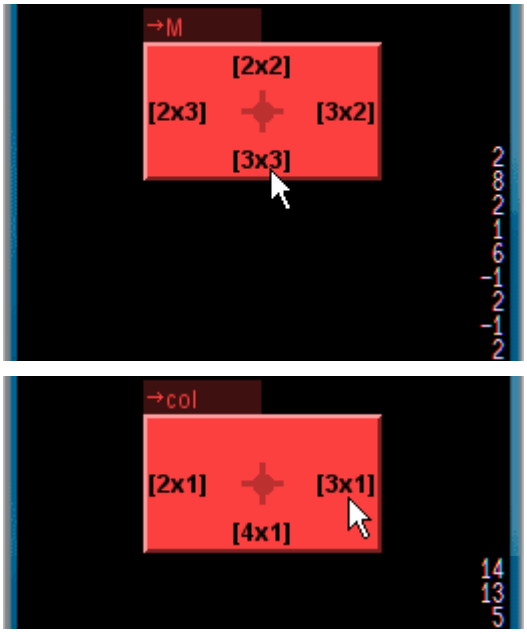
Y queremos encontrar la segunda solución del sistema, debemos realizar la siguiente operación:

$$x_2 = \frac{\begin{vmatrix} a_{1,1} & c_2 & a_{1,3} & a_{1,4} \\ a_{2,1} & c_2 & a_{2,3} & a_{2,4} \\ a_{3,1} & c_3 & a_{3,3} & a_{3,4} \\ a_{4,1} & c_4 & a_{3,4} & a_{4,4} \end{vmatrix}}{\begin{vmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{2,3} & a_{3,3} & a_{3,4} \\ a_{4,1} & a_{2,4} & a_{3,4} & a_{4,4} \end{vmatrix}}$$

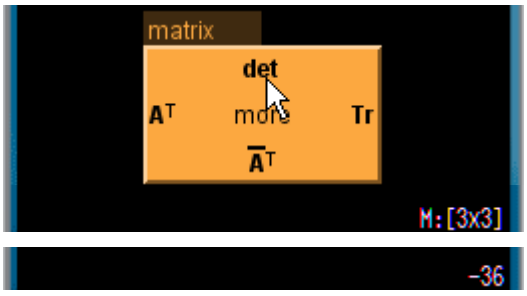
Para comprender mejor el método se encontrarán las soluciones del siguiente sistema de ecuaciones lineales:

$$\begin{aligned} 2x_1 + 8x_2 + 2x_3 &= 14 \\ x_1 + 6x_2 - x_3 &= 13 \\ 2x_1 - x_2 + 2x_3 &= 5 \end{aligned} \qquad (10.1)$$

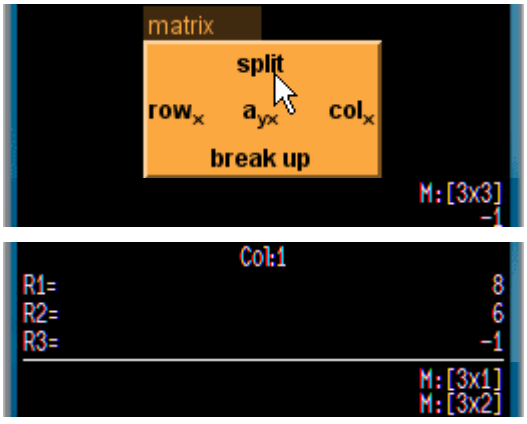
Primero creamos dos matrices, para la matriz de los coeficientes y la matriz de las constantes y las guardamos en dos posiciones de memoria, por ejemplo las posiciones 0 y 1, para no tener que trabajar con la pila.



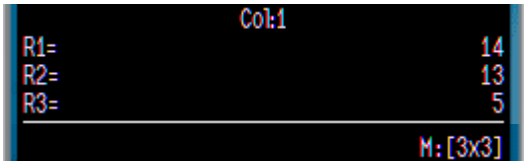
Ahora encontramos el determinante de la matriz de los coeficientes y guardamos el resultado en otra posición de memoria, por ejemplo la posición 2 (si no se quiere trabajar con la pila).



Ahora, para calcular la primera solución debemos reemplazar la primera columna de la matriz de los coeficientes con la columna de la matriz de las constantes. Para ello dividimos la matriz en la primera columna (para dividir una matriz con “split” por las columnas en lugar de las filas se emplean números negativos):

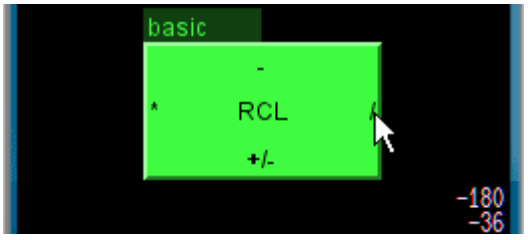


Ahora borramos la primera columna y la reemplazamos con la columna de las constantes (stack/x⇔y; clear; special/mem/RCL/<0-3>/1 ; stack/x⇔y; math/matrix/combine/concat):



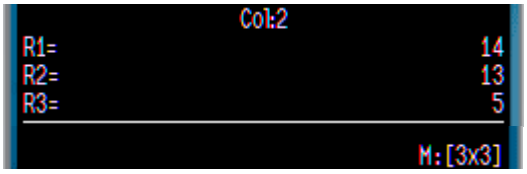
Ahora calculamos el determinante de esta matriz:

Y dividimos este resultado entre el determinante de los coeficientes, con lo que obtenemos la primera solución:



Procedemos de manera similar con la segunda solución, solo que ahora reemplazamos la segunda columna con la columna de las constantes:

```
special/mem/RCL/<0-3>/0
-1
math/matrix/parts/split
-1
math/matrix/parts/split
special/stack/x⇔y
clear
special/mem/RCL/<0-3>/1
special/stack/x⇔y
math/matrix/combine/concat
math/matrix/combine/concat
```



Cuyo determinante es:

Por lo tanto la segunda solución es (special/mem/RCL/<0-3>/2; basic//):

Procedemos de manera similar para encontrar la tercera solución, sólo que ahora reemplazamos la tercera columna de la matriz de los coeficientes por la columna de las constantes:

```
special/mem/RCL/<0-3>/0
-2
```

```
math/matrix/parts/split
clear
special/mem/RCL/<0-3>/1
math/matrix/combine/concat
```

Cuyo determinante es:

Por lo tanto la tercera solución es (mem/special/RCL/<0-3>/2; basic//):

En consecuencia las tres soluciones del sistema son x_1=5; x_2=1 x_3=-2.

Como se puede apreciar, aún cuando el cálculo de la determinante ya está implementado en Calc-Java, el proceso de reemplazar cada una de las columnas de la matriz de los coeficientes por la columna de las constantes es un tanto moroso, por lo que puede resultar conveniente automatizar el proceso mediante un programa.

10.1.1. Ejercicios

1. Encuentre las soluciones del siguiente sistema de ecuaciones lineales calculando determinantes.

$$\begin{aligned} 3x_1 - x_2 + 2x_3 &= 12 \\ x_1 + 2x_2 + 3x_3 &= 11 \\ 2x_1 - 2x_2 - x_3 &= 2 \end{aligned}$$

2. Encuentre las soluciones del siguiente sistema de ecuaciones lineales calculando determinantes.

$$\begin{aligned} 2x_1 - 2x_2 + 5x_3 &= 13 \\ 2x_1 + 3x_2 + 4x_3 &= 20 \\ 3x_1 - x_2 + 3x_3 &= 10 \end{aligned}$$

3. Encuentre las soluciones del siguiente sistema de ecuaciones lineales calculando determinantes.

$$\begin{aligned} x_1 + x_2 + x_3 + x_4 &= -1 \\ 4x_1 + 5x_2 + 6x_3 + 7x_4 &= 0 \\ 6x_1 + 10x_2 + 15x_3 + 21x_4 &= 0 \\ 12x_1 + 30x_2 + 60x_3 + 105x_4 &= 0 \end{aligned}$$

4. Encuentre las soluciones del siguiente sistema de ecuaciones lineales calculando determinantes.

$$\begin{aligned} x_1 + 2x_2 + 33x_3 + 5x_4 - x_5 + 2x_6 &= 17 \\ 2x_1 + 25x_2 + x_3 - x_4 + 2x_5 - 5x_6 &= 24 \\ 5x_1 + 6x_2 - 3x_3 - 4x_4 + 3x_5 + 36x_6 &= 24 \\ 3x_1 + 5x_2 + 7x_3 - 45x_4 + 4x_5 + x_6 &= 37 \\ 30x_1 + x_2 + 2x_3 + 6x_4 + 5x_5 - 3x_6 &= 12 \\ 4x_1 + 3x_2 + 2x_3 + 3x_4 + 40x_5 - 4x_6 &= 23 \end{aligned}$$

10.2. Método Gauss - Seidel

Cuando el número de ecuaciones en un sistema es elevado (con cientos o miles de ecuaciones) los métodos de eliminación no devuelven los resultados correctos y ello sucede no porque los métodos fallen, sino porque el error se acumula debido a la propagación, llegando a ser tan grande que los resultados obtenidos resultan ser muy diferentes de los resultados correctos.

La propagación del error se da porque los resultados intermedios se emplean para calcular otros y estos a su vez para calcular otros y así sucesivamente hasta llegar a los resultados finales y dado que en todos estos cálculos se producen errores de redondeo, dichos errores se propagan y se acumulan siendo cada vez mayores.

En estos casos se debe recurrir a los métodos iterativos, en los cuales partiendo de valores iniciales asumidos se calculan en iteraciones sucesivas nuevos valores hasta que las igualdades del sistema se cumplen con cierto margen de error. De esta manera se evita la propagación del error y el error permitido se fija de antemano.

Uno de dichos métodos es el método de Gauss – Seidel. En este método se asumen valores iniciales para todas las incógnitas, usualmente dichos valores son inicialmente ceros. Empleando los valores asumidos se calcula con la primera ecuación un nuevo valor para la primera incógnita. Empleando el valor calculado y los valores asumidos se calcula con la segunda ecuación un nuevo valor para la segunda incógnita. Empleando los valores calculados y los valores asumidos, se calcula con la tercera ecuación un nuevo valor para la tercera incógnita y así sucesivamente hasta calcular nuevos valores para todas las incógnitas.

Entonces, se verifica si con los valores calculados se cumplen las igualdades del sistema (es decir si las ecuaciones se igualan a cero), de ser así el proceso concluye y se tienen ya las soluciones del sistema (los valores calculados), caso contrario los valores calculados se convierten en valores asumidos y se repite el proceso.

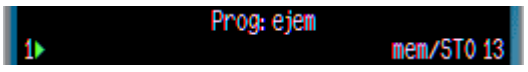
Para comprender mejor el procedimiento descrito encontremos las soluciones aproximadas del siguiente sistema de ecuaciones lineales:

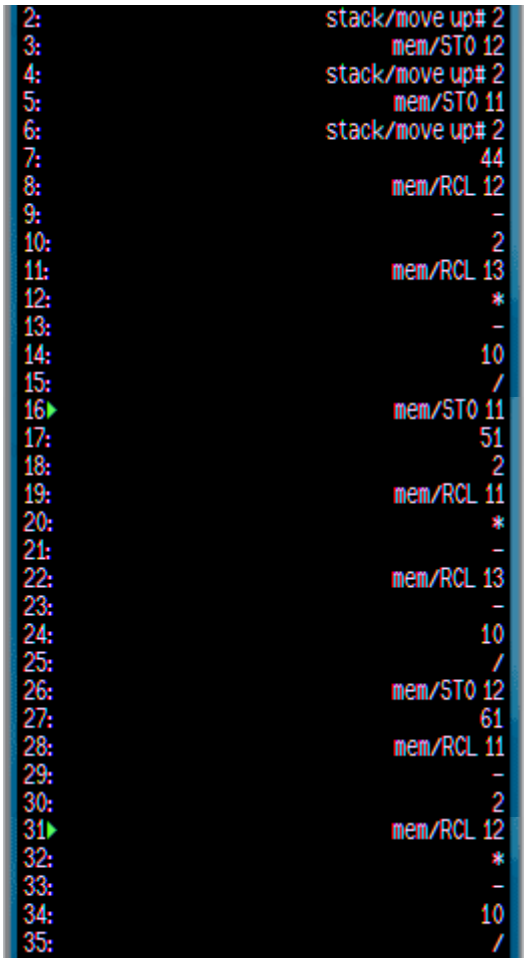
$$\begin{aligned} 10x_1 + x_2 + 2x_3 &= 44 \\ 2x_1 + 10x_2 + x_3 &= 51 \\ x_1 + 2x_2 + 10x_3 &= 61 \end{aligned} \qquad (10.2)$$

Para ese fin debemos despejar una incógnita de cada una de las ecuaciones, es decir:

$$\begin{aligned} x_1^* &= \frac{44 - x_2 - 2x_3}{10} \\ x_2^* &= \frac{51 - 2x_1^* - x_3}{10} \\ x_3^* &= \frac{61 - x_1^* - 2x_2^*}{10} \end{aligned} \qquad (10.3)$$

Donde las variables con “*” hacen referencia a los nuevos valores calculados, mientras que las variables sin “*” son los valores asumidos (como se dijo, normalmente ceros). Ahora para evitarnos el trabajo de realizar estos cálculos manualmente, los programamos:

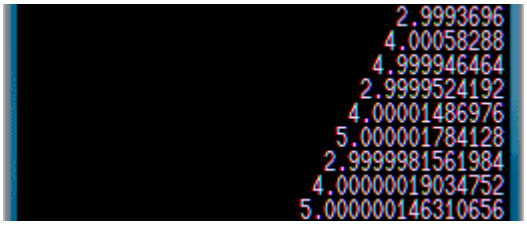




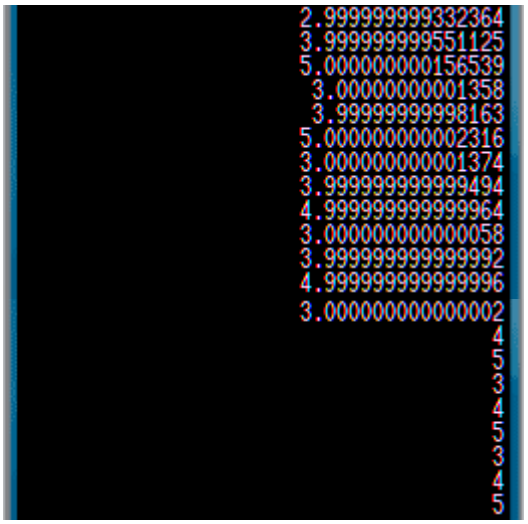
Tomando como valores iniciales ceros y haciendo correr el programa se obtiene:

Donde, como era de esperar, los valores iniciales asumidos (ceros) y los calculados son muy diferentes. Volviendo a hacer correr el programa con estos nuevos valores se obtiene:

Que todavía son diferentes a los valores asumidos, pero donde ya se puede observar una cierta similitud. Haciendo correr el programa cuatro veces más, se obtienen los siguientes valores:



Como se puede observar con cada iteración los valores calculados son más cercanos. Haciendo correr el programa 7 veces más (hasta que los valores calculados son iguales a los asumidos) se obtiene:



Que son las soluciones del sistema de ecuaciones: x_1=3; x_2=4; x_3=5.

10.2.1. Programa

La ecuación general, que puede ser deducida fácilmente en base al ejemplo desarrollado en el anterior acápite es:

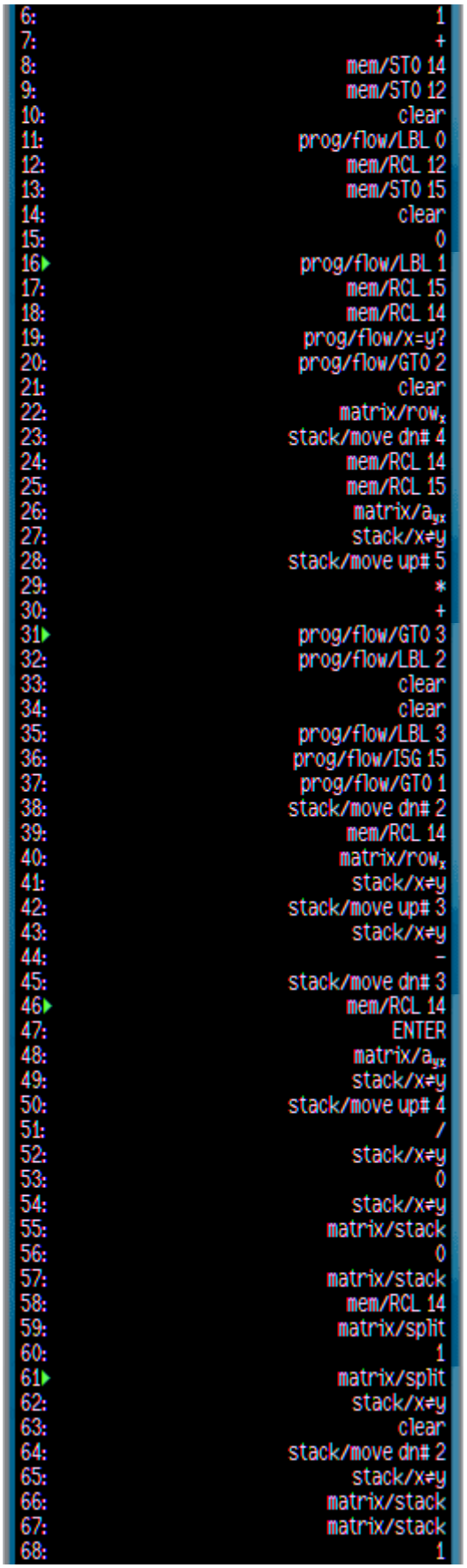
$$y_i = \frac{a_{i,n} - \sum_{j=1}^{i-1} a_{ij} * y_j - \sum_{j=i+1}^{m} a_{ij} * x_j}{a_{ii}} \quad \{i = 1 \rightarrow m \qquad (10.4)$$

Donde *"a"* son los coeficientes de la matriz aumentada, *"x"* el vector con los valores asumidos, *"y"* el vector con los valores calculados, *"m"* es el número de ecuaciones en el sistema (filas de la matriz aumentada) y *"n"* el número de columnas (m+1).

Si inicialmente los valores asumidos se asignan al vector *"y"*, entonces los nuevos valores pueden ser calculados empleando únicamente el vector *"y"*, si se procede de esa manera la ecuación (10.4) toma la forma:

$$x_i = \frac{a_{i,n} - \sum_{j=1}^{m} a_{ij} * x_j \quad \{j \neq i}{a_{ii}} \quad \{i = 1 \rightarrow m \qquad (10.5)$$

Entonces para programar el método de Gauss-Seidel, básicamente lo que se debe hacer es programar la ecuación 10.5. El código de dicho programa es el siguiente:


6: 1
7: +
8: mem/STO 14
9: mem/STO 12
10: clear
11: prog/flow/LBL 0
12: mem/RCL 12
13: mem/STO 15
14: clear
15: 0
16: prog/flow/LBL 1
17: mem/RCL 15
18: mem/RCL 14
19: prog/flow/x=y?
20: prog/flow/GTO 2
21: clear
22: matrix/row$_x$
23: stack/move dn# 4
24: mem/RCL 14
25: mem/RCL 15
26: matrix/a$_{gx}$
27: stack/x≠y
28: stack/move up# 5
29: *
30: +
31: prog/flow/GTO 3
32: prog/flow/LBL 2
33: clear
34: clear
35: prog/flow/LBL 3
36: prog/flow/ISG 15
37: prog/flow/GTO 1
38: stack/move dn# 2
39: mem/RCL 14
40: matrix/row$_x$
41: stack/x≠y
42: stack/move up# 3
43: stack/x≠y
44: -
45: stack/move dn# 3
46: mem/RCL 14
47: ENTER
48: matrix/a$_{gx}$
49: stack/x≠y
50: stack/move up# 4
51: /
52: stack/x≠y
53: 0
54: stack/x≠y
55: matrix/stack
56: 0
57: matrix/stack
58: mem/RCL 14
59: matrix/split
60: 1
61: matrix/split
62: stack/x≠y
63: clear
64: stack/move dn# 2
65: stack/x≠y
66: matrix/stack
67: matrix/stack
68: 1

```
69:                    matrix/split
70:                      stack/x≠y
71:                           clear
72:                     mem/RCL 13
73:                    matrix/split
74:                           clear
75:                  prog/flow/ISG 14
76▸                  prog/flow/GTO 0
77:                  stack/RCL st# 2
78:                  stack/RCL st# 1
79:                               *
80:                  stack/RCL st# 2
81:                               -
82:                     mem/RCL 12
83:                     mem/STO 15
84:                           clear
85:                  prog/flow/LBL 4
86:                     mem/RCL 15
87:                    matrix/row$_x$
88:                           abs
89:     0.0000000000000001
90:                    prog/flow/x>y?
91▸                  prog/flow/GTO 5
92:                           clear
93:                           clear
94:                     mem/RCL 12
95:                     mem/STO 14
96:                           clear
97:                           clear
98:                  prog/flow/GTO 0
99:                  prog/flow/LBL 5
100:                          clear
101:                          clear
102:                 prog/flow/ISG 15
103:                 prog/flow/GTO 4
104:                          clear
105:                stack/move up# 2
106▸                          clear
107:                          clear
108:                      [prg end]
```

Donde como se puede ver se ha trabajado con una exactitud de 16 dígitos. Esta exactitud puede resultar demasiado alta para la mayoría de los casos prácticos, por lo que la misma puede ser disminuida a unos 9 dígitos.

Para probar el programa hacemos correr el mismo con los datos del anterior acápite, para ello mandamos como datos la matriz de los coeficientes, el vector de las constantes y el vector con los valores iniciales asumidos (ceros para este ejemplo):

```
                Col:1
R1=                                0
R2=                                0
R3=                                0
──────────────────────────────────────
                           M:[3x3]
                           M:[3x1]
                           M:[3x1]
```

```
                Col:1
R1=                                3
R2=                                4
R3=                                5
──────────────────────────────────────
                           M:[3x1]
```

Y como se puede observar, se obtienen los resultados correctos.

Un detalle que debe tomarse muy en cuenta cuando se emplea el método de **Gauss-Seidel**, es que el mismo **converge cuando el sistema tiene una diagonal dominante**, es decir cuando todos los elementos de la diagonal principal son mayores (en valor absoluto) a la suma de los otros elementos en la misma fila (tal como ocurre con la matriz del ejemplo). Si el sistema no tiene una matriz dominante la convergencia no está asegurada, es decir que es probable que el método no encuentre las soluciones y se quede iterando indefinidamente (razón por la cual podría ser de utilidad incluir un contador en el programa y terminar el mismo cuando el contador alcance un límite especificado).

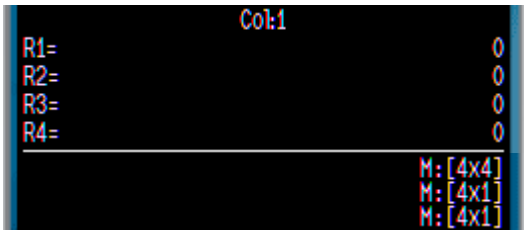
En caso de que el sistema tenga una matriz dominante, pero esté en desorden, es necesario ordenar primero el sistema (intercambiar filas), pues de lo contrario el método no convergería y no se encontrarían los resultados.

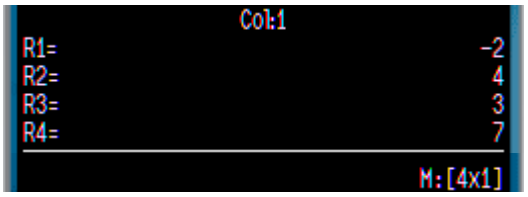
10.2.2. Ejemplos

1. Encuentre las soluciones del siguiente sistema de ecuaciones lineales empleando el método de Gauss - Seidel.

$$\begin{aligned} 12x_1 + 3x_2 - x_3 + 2x_4 &= -1 \\ 2x_1 + 13x_2 + x_3 - 3x_4 &= 30 \\ x_1 + 2x_2 - 14x_3 + 4x_4 &= -8 \\ x_1 + x_2 + x_3 + 10x_4 &= 75 \end{aligned}$$

En este caso el sistema es dominante y está ordenado, por lo que la convergencia está asegurada, siendo suficiente mandar los datos al programa de Gauss - Seidel (gs):





Por lo tanto las soluciones del sistema son: x_1=-2; x_2=4; x_3=3; x_4=-7.

2. Encuentre las soluciones de los siguientes sistemas de ecuaciones lineales empleando el método Gauss – Seidel:

$2x_1+3x_2+15x_3+x_9-2x_{10}=32$
$12x_1+2x_2+x_3+x_4-x_8-x_9-2x_{10}=-50$
$x_1+10x_2-x_5-x_7+2x_8+2x_9=21$
$x_1+2x_2+16x_4+x_5+x_6+x_7-x_8-2x_9=80$
$x_3+x_4+14x_5+x_8+x_9+x_{10}=131$
$-2x_1+x_5+16x_6=155$
$-3x_1+2x_2+2x_3+18x_7-2x_{10}=136$
$x_1+x_2+x_3-x_4-2x_5-3x_6+2x_7+20x_8+3x_9+2x_{10}=104$
$x_6+x_7+x_8+11x_9=131$

$x_4+x_5+x_6+2x_7+2x_8+2x_9+19x_{10}=274$

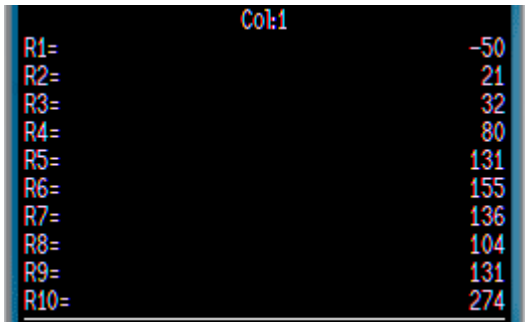
En este caso el sistema tiene una diagonal principal pero está desordenado, por lo que es necesario primero ordenar las ecuaciones de manera que los coeficientes formen una diagonal principal:

$12x_1+2x_2+x_3+x_4-x_8-x_9-2x_{10}=-50$
$x_1+10x_2-x_5-x_7+2x_8+2x_9=21$
$2x_1+3x_2+15x_3+x_9-2x_{10}=32$
$x_1+2x_2+16x_4+x_5+x_6+x_7-x_8-2x_9=80$
$x_3+x_4+14x_5+x_8+x_9+x_{10}=131$
$-2x_1+x_5+16x_6=155$
$-3x_1+2x_2+2x_3+18x_7-2x_{10}=136$
$x_1+x_2+x_3-x_4-2x_5-3x_6+2x_7+20x_8+3x_9+2x_{10}=104$
$x_6+x_7+x_8+11x_9=131$
$x_4+x_5+x_6+2x_7+2x_8+2x_9+19x_{10}=274$

Por lo tanto, la matriz de los coeficientes que debe mandarse al método de Gauss – Seidel es:

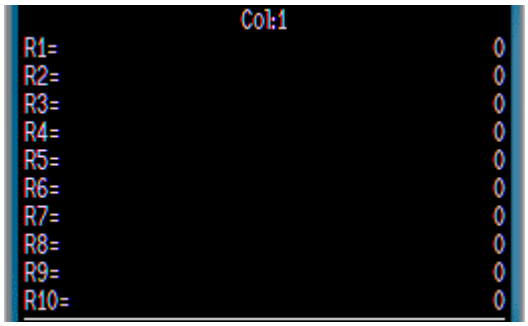
12	2	1	1	0	0	0	-1	-1	-2
1	10	0	0	-1	0	-1	2	2	0
2	3	15	0	0	0	0	0	1	-2
1	2	0	16	1	1	1	-1	-2	0
0	0	1	1	14	0	0	1	1	1
-2	0	0	0	1	16	0	0	0	0
-3	2	2	0	0	0	18	0	0	-2
1	1	1	-1	-2	-3	2	20	3	2
0	0	0	0	0	1	1	1	11	0
0	0	0	1	1	1	2	2	2	19

Que la mostramos en esa forma debido a que Calc-Java sólo permite mostrar una columna a la vez. El vector de las constantes es:

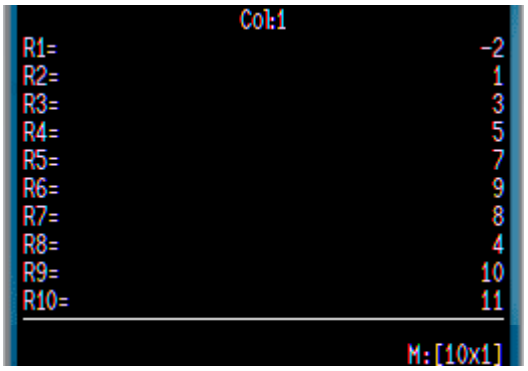


Y los valores iniciales (ceros) son:

10; 1; math/matrix/create/new:



Haciendo correr el programa “gs” con estos datos se obtienen las soluciones:



Que son las 10 soluciones del sistema de ecuaciones.

10.2.3. Ejercicios

5. Encuentre las soluciones del siguiente sistema de ecuaciones empleando el método de Gauss- Seidel.

$x_1+2x_2+33x_3+5x_4-x_5+2x_6=17$
$2x_1+25x_2+x_3-x_4+2x_5-5x_6=24$
$5x_1+6x_2-3x_3-4x_4+3x_5+36x_6=24$
$3x_1+5x_2+7x_3-45x_4+4x_5+x_6=37$
$30x_1+x_2+2x_3+6x_4+5x_5-3x_6=12$
$4x_1+3x_2+2x_3+3x_4+40x_5-4x_6=23$

6. Encuentre las soluciones del siguiente sistema de ecuaciones empleando el método de Gauss – Seidel.

$x_2+20x_3+x_4+x_5+x_6=-66$
$15x_1+x_2+x_3-x_5-2x_6=-103$
$3x_1+13\ x_2+x_5-2x_6=29$
$x_1+x_3+2x_4+2x_5+19x_6=144$
$-3x_1-18x_4-x_6=-44$
$x_3+x_4-14x_5=27$

7. Encuentre las soluciones del siguiente sistema de ecuaciones empleando el método de Gauss – Seidel.

$-2x_1+17x_3+x_4+x_5+x_6-x_7=113$
$21x_1+x_2+x_3-x_4-x_6-2x_7=-53$
$-x_1-2x_2+x_4-19x_5+x_6+2x_7=-81$
$-x_1-16x_2+2x_5+x_7=-63$
$x_3-3x_4+2x_5+22x_6+3x_7=13$
$x_4+2x_5+2x_6-25x_7=-154$
$x_3+14x_4+x_5=-101$

8. Encuentre las soluciones del siguiente sistema de ecuaciones empleando el método de Gauss – Seidel.

$-3x_1+19x_2-2x_7-x_8=-131$
$31x_1+2x_2+x_3+x_4-x_7-x_8=96$
$x_1+2x_2-18x_3+x_4+x_5+2x_6+3x_8=-132$
$2x_1+x_4-20x_5+2x_6+3x_8=71$
$3x_1+2x_2+2x_3+x_5+16x_6-2x_8=$
$x_3+24x_4+x_7+x_8=99$
$3x_4+x_5+4x_6-26x_7=202$
$x_1+x_3+5x_4+2x_5+3x_6+2x_7-23x_8=19$

9. Encuentre las soluciones del siguiente sistema de ecuaciones empleando el método de Gauss – Seidel.

$x_1+19x_2+x_5-x_7-3x_8+2x_9=-28$
$x_1-2x_2+21x_3+x_4-5x_5+x_6-4x_7-2x_8=71$
$15x_1+x_2+2x_3+x_4-x_8-x_9=97$
$3x_2+x_4-17x_5+4x_8=-111$
$2x_1+2x_2-19x_6+4x_8-x_9=-158$
$x_2+x_3+24x_4-3x_7+x_8-2x_9=-232$
$6x_5+x_6+3x_7-23x_8=87$
$x_1-x_3+x_4+x_5-x_6+2x_8+11x_9=53$
$x_1+3x_2+x_3+5x_4-x_5+2x_6+22x_7+3x_8+2x_9=-4$

11. SISTEMAS DE ECUACIONES NO LINEALES

Para la resolución de sistemas de ecuaciones no lineales, no se cuentan con métodos analíticos, por lo tanto dichos sistemas sólo pueden ser resueltos recurriendo a los métodos numéricos.

Como sucede con prácticamente todos los métodos numéricos, en estos métodos se requieren valores iniciales asumidos (valores de prueba) para comenzar el proceso. En general, a medida que incrementa el número de ecuaciones del sistema, incrementa también el tiempo requerido para lograr convergencia, por lo que es importante contar con valores iniciales razonablemente cercanos a las soluciones.

El objetivo del presente tema es que al concluir el mismo estén capacitados para encontrar las soluciones de sistemas de ecuaciones no lineales empleando los métodos estudiados en los temas 3, 4 y 5, así como el método de la Gradiente.

11.1. Empleando métodos para ecuaciones no lineales con una incógnita

Como ya se vio en los temas 3, 4 y 5, cuando un sistema de ecuaciones no lineales puede ser reordenado de manera que dependa de una sola variable, es posible encontrar las soluciones con los métodos estudiados en dichos temas.

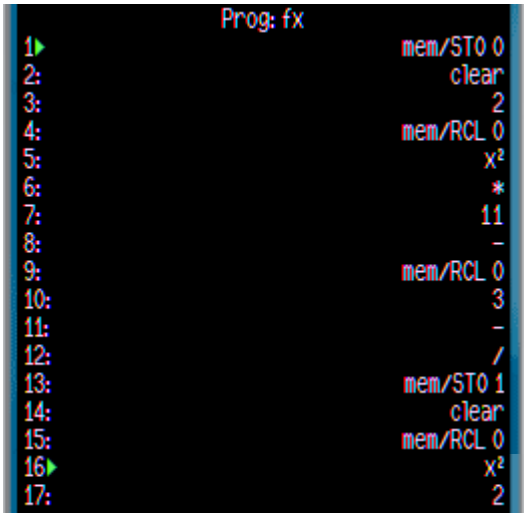
Por ejemplo el siguiente sistema de dos ecuaciones lineales con dos incógnitas:

$$x^2 + 2y^2 = 22$$
$$-2x^2 + xy - 3y = -11$$

Puede ser reordenado como función de una sola incógnita:

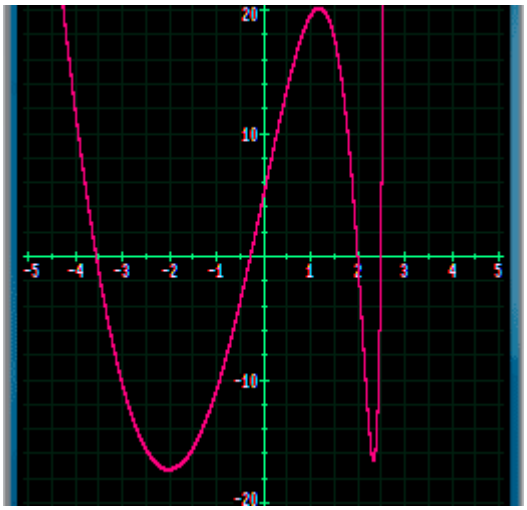
$$f(x) = x^2 + 2y^2 - 22 = 0$$
$$y = \frac{2x^2 - 11}{x - 3}$$

Y una vez reordenado puede ser resuelto con cualquiera de los métodos empleados en los temas mencionados. Por ejemplo para encontrar las soluciones con “solve” de Calc-Java, creamos primero la función:



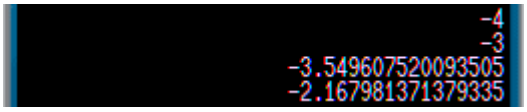
Donde como se ve se puede observar, se emplea la memoria 0 para guardar el valor de “x” y la memoria 1 para el valor de “y”.

Es posible inclusive graficar la función para ver el lugar de las soluciones. Así por ejemplo graficando la función entre -5, 5, -20 y 20, se obtiene:



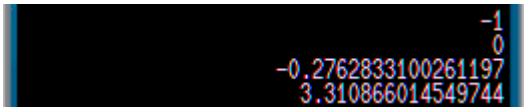
En este caso existen cuatro soluciones: una cerca a -3.5, otra cerca a -0.5, otra cerca a 2 y otra cerca a 2.5. En la mayoría de los casos prácticos se tiene una idea de la magnitud que debe tener el resultado, por lo que en general sólo se busca una de las posibles soluciones. Así por ejemplo, cuando el resultado corresponde a una magnitud física como peso, distancia, volumen, etc., se sabe que dicho resultado no puede ser negativo, por lo que se descartan las soluciones negativas.

En este caso, para practicar, encontraremos las cuatro posibles soluciones, primero encontramos las soluciones cerca a -3.5:



Donde el primer resultado es el valor de “x” y el segundo, obtenido recuperando la memoria 1, es el valor de “y”.

Las soluciones cerca a -0.5 son:



Las soluciones cerca a 2 son:

Y las soluciones cerca de 2.5 son:

```
                                2.2
                                2.6
 2.492557496786291
-2.809551309837076
```

Para resolver con otro método, como el de Newton-Raphson, añadimos, como de costumbre la función al final del programa “nr”:

```
68▸                prog/flow/LBL 0
69:                     mem/STO 0
70:                         clear
71:                             2
72:                     mem/RCL 0
73:                            x²
74:                             *
75:                            11
76:                             -
77:                     mem/RCL 0
78:                             3
79:                             -
80:                             /
81:                     mem/STO 1
82:                         clear
83▸                     mem/RCL 0
84:                            x²
85:                             2
86:                     mem/RCL 1
87:                            x²
88:                             *
89:                             +
90:                            22
91:                             -
92:                prog/flow/RTN
93:                    [prg end]
```

Trabajando con una precisión de 15 dígitos y un límite de 50 iteraciones y un valor inicial asumido igual a -3.5:

```
1▸ 0.000000000000001
2:                50.00001
3:                    -3.5
```

Se obtiene:

```
-3.549607520093505
-2.167981371379335
```

Donde el primer valor es “x” y el segundo el de “y” (recuperado de la memoria 1).

Para el segundo se tiene:

```
1: 0.000000000000001
2:                50.00001
3:                    -0.5
```

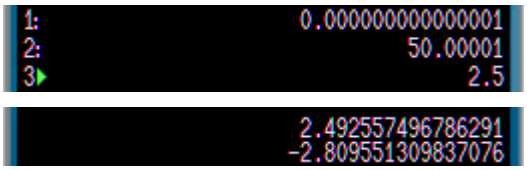
```
-0.2762833100261197
 3.310866014549744
```

Para el tercero:

```
1▸ 0.000000000000001
2:                50.00001
3:                     1.8
```

```
2
3
```

Y para el cuarto:



De manera similar se procede con el método de la Secante y como se recordará, para el método de Sustitución Directa es necesario colocar la ecuación en la forma x=g(x).

11.1.1. Ejemplos

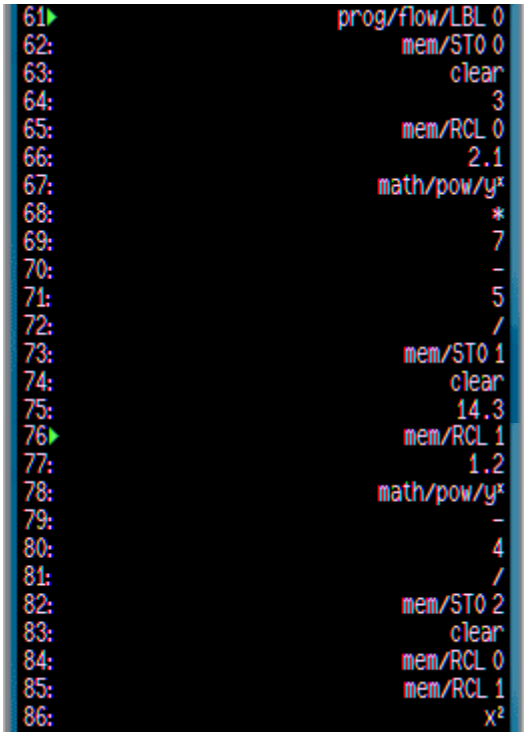
1. Encuentre las soluciones del siguiente sistema de ecuaciones no lineales empleando el método de la secante.

$$3x^{2.1} - 5y = 7.0$$
$$y^{1.2} + 4z = 14.3$$
$$x + y^2 + z^2 = 14.0$$

Primero se coloca el sistema de ecuaciones en función de una sola variable:

$$f(x) = x + y^2 + z^2 - 14.0 = 0$$
$$y = \frac{3x^{2.1} - 7.0}{5}$$
$$z = \frac{14.3 - y^{1.2}}{4}$$

Ahora se programa esta función al final del método de la secante, guardando los valores de “x”, “y” y “z” en las memorias 0, 1 y 2 respectivamente:

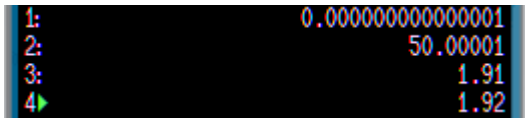


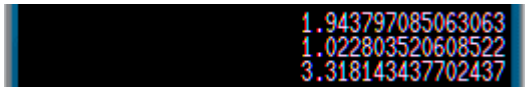
Trabajando con una precisión de 15 dígitos, un límite de 50 iteraciones y valores iniciales iguales a 1.7 y 1.8:

Se obtiene:

Donde el primer valor es “x”, el segundo “y” (recuperado de la memoria 1) y el tercero es “z” (recuperado de la memoria 2).

La segunda solución, cerca de 1.95 (que puede ser vista graficando la función), se calcula con:





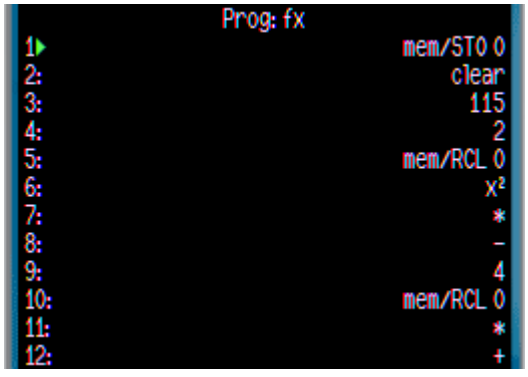
2. Encuentre las soluciones del siguiente sistema de ecuaciones no lineales empleando el método de Newton – Raphson:

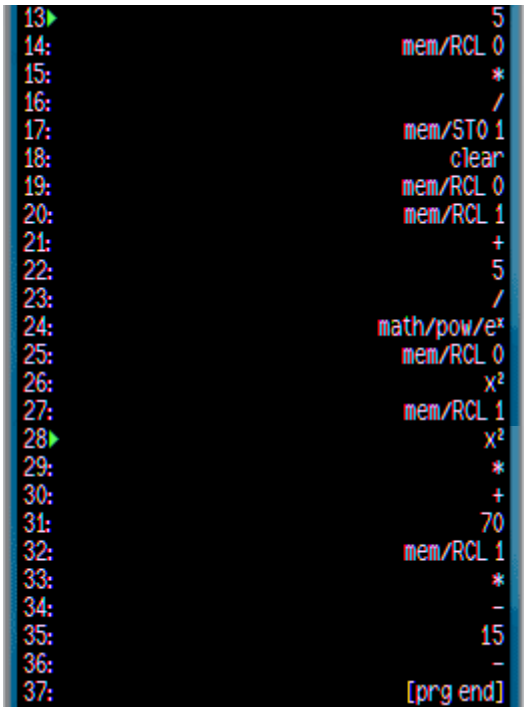
$$2x^2 + 5xy - 4x = 115$$
$$e^{\frac{x+y}{5}} + x^2y^2 - 70y = 15$$

Primero se coloca la ecuación en la forma “f(x)=0”

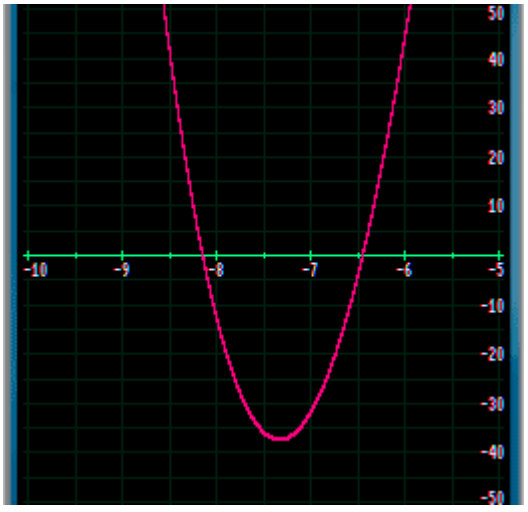
$$f(x) = e^{\frac{x+y}{5}} + x^2y^2 - 70y - 15 = 0$$
$$y = \frac{115 - 2x^2 + 4x}{5x}$$

Dado que en este caso no se tiene idea del lugar de las soluciones, es conveniente graficar primero la función, para lo cual debe ser primero programada:



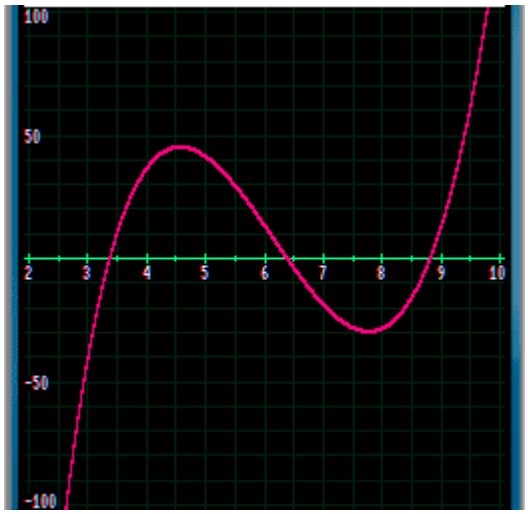


Graficando esta función entre algunos límites, se encuentra que existen varias soluciones y también algunas discontinuidades. Por ejemplo, entre -10 y -5 se tiene dos soluciones:



Entre -2 y 1 existe una discontinuidad, es decir un sector donde no es posible obtener los valores de la función.

Entre 2 y 10 se tienen otras tres soluciones:



Como en este ejemplo y en los anteriores, cuando se trabaja con sistemas de ecuaciones no lineales, lo más frecuente es que se tenga no una sino múltiples soluciones, razón por la cual es importante contar con valores iniciales próximos a la solución que se quiere encontrar.

Para calcular soluciones más exactas con el método de Newton – Raphson, primero, como de costumbre, se programa la función al final del método:

```
68▸ prog/flow/LBL 0
69:      mem/STO 0
70:          clear
71:            115
72:              2
73:      mem/RCL 0
74:             x²
75:              *
76:              -
77:              4
78:      mem/RCL 0
79:              *
80:              +
81:              5
82:      mem/RCL 0
83▸              *
84:              /
85:      mem/STO 1
86:          clear
87:      mem/RCL 0
88:      mem/RCL 1
89:              +
90:              5
91:              /
92:     math/pow/eˣ
93:      mem/RCL 0
94:             x²
95:      mem/RCL 1
96:             x²
97:              *
98▸              +
99:             70
100:     mem/RCL 1
101:             *
102:             -
103:            15
```

```
104:                                -
105:                  prog/flow/RTN
106:                      [prg end]
```

Empleando valores iniciales obtenidos de las anteriores gráficas, podemos entonces encontrar todas las soluciones (con 15 dígitos de precisión). Para la primera solución tenemos:

```
1:                  0.000000000000001
2:                           50.00001
3:                               -8.5

                    -8.14595647078032
                    1.234895858116432
```

Donde como de costumbre, la primera solución es “x” y la segunda (recuperada de la memoria 1) es “y”.

Para la segunda:

```
1:                  0.000000000000001
2:                           50.00001
3:                               -6.5

                    -6.446490218620618
                    -0.189236829379074
```

Para la tercera:

```
1:                  0.000000000000001
2:                           50.00001
3:                                3.5

                    3.370456456874712
                    6.27582711394879
```

Para la cuarta:

```
1:                  0.000000000000001
2:                           50.00001
3:                                6.5

                    6.387131765563222
                    1.846143822797428
```

Para la quinta:

```
1:                  0.000000000000001
2:                           50.00001
3:                                  9

                    8.816091929784202
                    -0.1175649188751936
```

11.1.2. Ejercicios

1. Encuentre las soluciones del siguiente sistema de ecuaciones no lineales con “solve” de Calc – Java:

$$15y + 20x^2 - x^3 = 1500$$
$$9z - 1.5x^2 + e^{\frac{x}{2}} = 300$$
$$y^2 - z^3 - 5x = 166.81$$

2. Encuentre las soluciones del siguiente sistema de ecuaciones no lineales con el método de Newton – Raphson:

$$x^{1/2} + z^2 = 35$$
$$y + e^{z/2} - \frac{8}{z} = 30$$
$$w + z^2 - 2z = 31$$
$$x + w + y = 39$$

3. Encuentre las soluciones del siguiente sistema de ecuaciones no lineales con el método de la secante:

$$x^{2.1} + 3yz - \frac{5xy^{1.3}}{z^{5.6}} = 70$$
$$x^{2.2} + x^{0.3} - 3y = 25$$
$$3xy - x^{1.5} + 3y^{1.3} - 8z = 8$$

11.2. Método de la Gradiente

Cuando el sistema de ecuaciones no lineales no puede ser colocado en función de una sola variable, se debe recurrir a métodos propios para resolver sistemas de ecuaciones lineales. Uno de dichos métodos es el método de la gradiente, que básicamente es el método de Newton–Raphson, pero aplicado a sistemas de ecuaciones no lineales en lugar de una sola ecuación.

En este método se expanden las ecuaciones que conforman el sistema empleando series de Taylor, por ejemplo, si en el sistema se tienen 3 ecuaciones no lineales:

$$\begin{aligned} f_1(x_1, x_2, x_3) &= 0 \\ f_2(x_1, x_2, x_3) &= 0 \\ f_3(x_1, x_2, x_3) &= 0 \end{aligned} \qquad (11.1)$$

Siendo x_1, x_2 y x_3, los valores asumidos para las tres variables del sistema. Expandiendo las tres ecuaciones en series de Taylor (todas las funciones se evalúan en x_1, x_2 y x_3) obtenemos:

$$\begin{aligned} f_1(x_1 + \Delta x_1, x_2 + \Delta x_2, x_3 + \Delta x_3) &= f_1 + \frac{\partial f_1}{\partial x_1}\Delta x_1 + \frac{\partial f_1}{\partial x_2}\Delta x_2 + \frac{\partial f_1}{\partial x_3}\Delta x_3 + ...\infty = 0 \\ f_2(x_1 + \Delta x_1, x_2 + \Delta x_2, x_3 + \Delta x_3) &= f_2 + \frac{\partial f_2}{\partial x_1}\Delta x_1 + \frac{\partial f_2}{\partial x_2}\Delta x_2 + \frac{\partial f_2}{\partial x_3}\Delta x_3 + ...\infty = 0 \\ f_3(x_1 + \Delta x_1, x_2 + \Delta x_2, x_3 + \Delta x_3) &= f_3 + \frac{\partial f_3}{\partial x_1}\Delta x_1 + \frac{\partial f_3}{\partial x_2}\Delta x_2 + \frac{\partial f_3}{\partial x_3}\Delta x_3 + ...\infty = 0 \end{aligned} \qquad (11.2)$$

Donde Δx_1, Δx_2 y Δx_3 son los valores que deberían añadirse a los valores asumidos (x_1, x_2 y x_3) para que las funciones (f_1, f_2 y f_3) se hagan cero. En otras palabras si fuera posible calcular los valores de Δx_1, Δx_2 y Δx_3, las soluciones del sistema serían $y_1 = x_1 + \Delta x_1$, $y_2 = x_2 + \Delta x_2$ y $y_3 = x_3 + \Delta x_3$.

Por supuesto al tratarse de series infinitas, no es posible en la práctica calcular dichos valores, sin embargo, si es posible obtener una aproximación de los mismos tomando en cuenta sólo los términos de primero orden (que es lo que hace el método de Newton-Raphson), es decir, truncando las series de Taylor en sus primeros 4 términos:

$$\begin{aligned}
\frac{\partial f_1}{\partial x_1}\Delta x_1 + \frac{\partial f_1}{\partial x_2}\Delta x_2 + \frac{\partial f_1}{\partial x_3}\Delta x_3 &= -f_1 \\
\frac{\partial f_2}{\partial x_1}\Delta x_1 + \frac{\partial f_2}{\partial x_2}\Delta x_2 + \frac{\partial f_2}{\partial x_3}\Delta x_3 &= -f_2 \\
\frac{\partial f_3}{\partial x_1}\Delta x_1 + \frac{\partial f_3}{\partial x_2}\Delta x_2 + \frac{\partial f_3}{\partial x_3}\Delta x_3 &= -f_3
\end{aligned} \qquad (11.3)$$

Como se puede observar, procediendo de esa manera, se forma un sistema de 3 ecuaciones lineales con 3 incógnitas (asumiendo por supuesto que sea posible calcular las derivadas parciales). Se debe recordar no obstante que estos valores, al resultar de la serie truncada, son sólo aproximados y que en consecuencia no permiten calcular realmente las soluciones, aunque si permiten acercarse a las mismas. Por lo tanto $y_1 = x_1+\Delta x_1$, $y_2 = x_2+\Delta x_2$ y $y_3 = x_3+\Delta x_3$, son valores más cercanos a la solución que los valores asumidos (x_1, x_2 y x_3), pero no son todavía las soluciones del sistema.

Por lo tanto, el proceso para encontrar las soluciones del sistema de ecuaciones se convierte en un proceso iterativo: comenzando con los valores asumidos x_1 *a* x_n, se calculan nuevos valores y_1 *a* y_n resolviendo el sistema de ecuaciones lineales (previo cálculo de las derivadas parciales), entonces se comparan los valores calculados con los asumidos y si son aproximadamente iguales el proceso concluye siendo las soluciones los valores desde y_1 *a* y_n, caso contrario los valores calculados (y_1 *a* y_n) se convierten en los nuevos valores asumidos (x_1 *a* x_n) y el proceso se repite.

Además de comparar los valores asumidos con los calculados (es decir la precisión), es posible también comparar el valor de las funciones con cero (es decir la exactitud).

Cuando se alcanza ya sea la precisión o la exactitud especificadas, el proceso concluye, siendo las soluciones los valores desde y_1 *a* y_n (es decir los últimos valores calculados) caso contrario los valores calculados (y_1 *a* y_n) se convierten en los nuevos valores asumidos (x_1 *a* x_n) y el proceso se repite.

Si bien en algunos casos donde las funciones son sencillas, es posible calcular las derivadas analíticas, en la mayoría de los casos esa alternativa no es práctica, por lo que en general, dichas derivadas deben ser calculadas numéricamente. Para este fin se puede emplear, como se hizo en el método de Newton-Raphson, la fórmula de diferencia central de segundo orden, adaptada para el cálculo de las derivadas parciales:

$$\frac{\partial f_i}{\partial x_j} = \frac{f_i(x_1, x_2, ..., x_{j+h}, ...x_n) - f_i(x_1, x_2, ..., x_{j-h}, ...x_n)}{2h}; \quad h = x_j * 10^{-6} \qquad (4)$$

Como ya se explicó en el método de Newton – Raphson, el valor de "h" que se propone en la ecuación (x_j*10^{-6}), es por lo general lo suficientemente pequeño como para permitir un cálculo aproximado de la derivada y lo suficientemente grande como para no generar errores apreciables de redondeo.

El sistema de ecuaciones lineales que se forma en el método, puede ser resuelto empleando los métodos estudiados en los temas 9 y 10.

Para comprender mejor el método, se encontrarán soluciones aproximadas del siguiente sistema de ecuaciones, empleando el método de la gradiente:

$$x^2 + 2y^2 = 22$$
$$-2x^2 + xy - 3y = -11$$

Para estar de acuerdo con la simbología empleada en las ecuaciones del método, se coloca reescriben estas ecuaciones en función de las variables x_1 y x_2 y se las iguala a cero:

$$f_1 = {x_1}^2 + 2{x_2}^2 - 22 = 0$$
$$f_2 = -2{x_1}^2 + x_1 x_2 - 3x_2 + 11 = 0$$

Ahora se asumen dos valores iniciales, en este caso serán: $x_1=1.5$ y $x_2=2$. Con estos valores se calcula el valor de las dos ecuaciones del sistema y se les cambia el signo:

$$-f_1 = -f_1(x_1, x_2) = -f_1(1.5, 2) = 11.75$$
$$-f_2 = -f_2(x_1, x_2) = -f_2(1.5, 2) = -3.5$$

Como estos valores no son ceros, o cercanos a cero, se sabe que no son las soluciones, por lo que se procede a calcular las derivadas parciales que conforman el sistema de ecuaciones lineales (2 en el ejemplo):

$$h = x_1 * 10^{-6} = 1.5 * 10^{-6} = 0.0000015$$

$$\frac{\partial f_1}{\partial x_1} = \frac{f_1(x_{1+h}, x_2) - f_1(x_{1-h}, x_2)}{2h} = \frac{f_1(1.5000015, 2) - f_1(1.49985, 2)}{2 * 0.0000015} =$$
$$\frac{-11.74999549999775 - 11.75000449999775}{0.000003} = 3.000000000000109$$
$$\frac{\partial f_2}{\partial x_1} = \frac{f_2(x_{1+h}, x_2) - f_2(x_{1-h}, x_2)}{2h} = \frac{f_2(1.5000015, 2) - f_2(1.49985, 2)}{2 * 0.0000015} =$$
$$\frac{3.4999939999955 - 3.500005999995}{0.000003} = -3.999999999999953$$

$$h = x_2 * 10^{-6} = 2 * 10^{-6} = 0.000002$$

$$\frac{\partial f_1}{\partial x_2} = \frac{f_1(x_1, x_{2+h}) - f_1(x_1, x_{2-h})}{2h} = \frac{f_1(1.5, 2.00002) - f_1(1.5, 1.9998)}{2 * 0.000002} =$$
$$\frac{-11.749983999992 + 11.750015999992}{0.000004} = 7.999999999999761$$
$$\frac{\partial f_2}{\partial x_2} = \frac{f_2(x_1, x_{2+h}) - f_2(x_1, x_{2-h})}{2h} = \frac{f_2(1.5, 2.00002) - f_2(1.5, 1.9998)}{2 * 0.000002} =$$
$$\frac{3.499997 - 3.500003}{0.000004} = -1.49999999999982$$

Con estas derivadas y el valor de las funciones, se forma el sistema de ecuaciones lineales:

$$\frac{\partial f_1}{\partial x_1}\Delta x_1 + \frac{\partial f_1}{\partial x_2}\Delta x_2 = 3.000000000000109\Delta x_1 + 7.999999999999761\Delta x_2 = 11.75 = -f_1$$
$$\frac{\partial f_2}{\partial x_1}\Delta x_1 + \frac{\partial f_2}{\partial x_2}\Delta x_2 = -3.999999999999953\Delta x_1 - 1.49999999999982\Delta x_2 = -3.5 = -f_2$$

El cual al ser resuelto (con la matriz inversa por ejemplo) devuelve los valores que deben ser añadidos a los valores asumidos para acercar los mismos a las soluciones correctas:

$$\Delta x_1 = 0.3772727272727242$$
$$\Delta x_2 = 1.327272727272763$$

Por lo tanto los nuevos valores de las incógnitas son:

$$y_1 = x_1 + \Delta x_1 = 1.5 + 0.3772727272727242 = 1.877272727272724$$
$$y_2 = x_2 + \Delta x_2 = 2 + 1.327272727272763 = 3.327272727272763$$

Ahora el proceso se repite con estos valores como valores como valores asumidos. Reemplazando los mismos en las ecuaciones del sistema se obtiene:

$$-f_1 = -f_1(x_1, x_2) = -3.665640495868232$$
$$-f_2 = -f_2(x_1, x_2) = -0.2160743801652619$$

Que todavía están lejos de cero, por lo que el proceso debe continuar. Los valores de las derivadas son:

$$h = x_1 * 10^{-6} = 0.000001877272727272724$$

$$\frac{\partial f_1}{\partial x_1} = \frac{f_1(x_{1+h}, x_2) - f_1(x_{1-h}, x_2)}{2h} = 3.754545454545337$$
$$\frac{\partial f_2}{\partial x_1} = \frac{f_2(x_{1+h}, x_2) - f_2(x_{1-h}, x_2)}{2h} = -4.181818181818361$$

$$h = x_2 * 10^{-6} = 2 * 10^{-6} = 0.000003327272727272763$$

$$\frac{\partial f_1}{\partial x_2} = \frac{f_1(x_1, x_{2+h}) - f_1(x_1, x_{2-h})}{2h} = 13.30909090909069$$
$$\frac{\partial f_2}{\partial x_2} = \frac{f_2(x_1, x_{2+h}) - f_2(x_1, x_{2-h})}{2h} = -1.122727272727289$$

Resolviendo el sistema de ecuaciones lineales y añadiendo los resultados del mismo a las variables asumidas, los nuevos valores de las incógnitas son:

$$y_1 = x_1 + \Delta x_1 = 2.013181556745032$$
$$y_2 = x_2 + \Delta x_2 = 3.013508518859119$$

Que se convierten en los nuevos valores de prueba.

Con estos nuevos valores de prueba los valores de las ecuaciones son:

$$-f_1 = -f_1(x_1, x_2) = 0.2153671668914142$$
$$-f_2 = -f_2(x_1, x_2) = -0.07958574615284283$$

Que están más cercanos a cero que los anteriores, pero que todavía son valores muy grandes, por lo que el proceso debe continuar. Las nuevas derivadas son:

$$h = x_1 * 10^{-6} = 0.000002013181556745032$$

$$\frac{\partial f_1}{\partial x_1} = \frac{f_1(x_{1+h}, x_2) - f_1(x_{1-h}, x_2)}{2h} = 4.026363113490076$$
$$\frac{\partial f_2}{\partial x_1} = \frac{f_2(x_{1+h}, x_2) - f_2(x_{1-h}, x_2)}{2h} = -5.039217708120465$$

$$h = x_2 * 10^{-6} = 2 * 10^{-6} = 0.000003013508518859119$$

$$\frac{\partial f_1}{\partial x_2} = \frac{f_1(x_1, x_{2+h}) - f_1(x_1, x_{2-h})}{2h} = 12.05403407543596$$
$$\frac{\partial f_2}{\partial x_2} = \frac{f_2(x_1, x_{2+h}) - f_2(x_1, x_{2-h})}{2h} = -0.986818443255216$$

Resolviendo el sistema de ecuaciones lineales y añadiendo los resultados del mismo a las variables asumidas, los nuevos valores de las incógnitas son:

$$y_1 = x_1 + \Delta x_1 = 2.000026613992285$$
$$y_2 = x_2 + \Delta x_2 = 3.000035801790688$$

Que se convierten en los nuevos valores de prueba.

Con estos nuevos valores de prueba los valores de las ecuaciones son:

$$-f_1 = -f_1(x_1, x_2) = -0.0005360807292332723$$
$$-f_2 = -f_2(x_1, x_2) = 0.0001688722158931164$$

Que están más cercanos a cero que los anteriores, pero recién tienen una exactitud de 3 dígitos, por lo que el proceso debe continuar. Las nuevas derivadas son:

$$h = x_1 * 10^{-6} = 0.000002000026613992285$$

$$\frac{\partial f_1}{\partial x_1} = \frac{f_1(x_{1+h}, x_2) - f_1(x_{1-h}, x_2)}{2h} = 4.000053227984464$$
$$\frac{\partial f_2}{\partial x_1} = \frac{f_2(x_{1+h}, x_2) - f_2(x_{1-h}, x_2)}{2h} = -5.00007065417758$$

$$h = x_2 * 10^{-6} = 2 * 10^{-6} = 0.000003000035801790688$$

$$\frac{\partial f_1}{\partial x_2} = \frac{f_1(x_1, x_{2+h}) - f_1(x_1, x_{2-h})}{2h} = 12.00014320716307$$
$$\frac{\partial f_2}{\partial x_2} = \frac{f_2(x_1, x_{2+h}) - f_2(x_1, x_{2-h})}{2h} = -0.9999733860076268$$

Resolviendo el sistema de ecuaciones lineales y añadiendo los resultados del mismo a las variables asumidas, los nuevos valores de las incógnitas son:

$$y_1 = x_1 + \Delta x_1 = 2.000000000040957$$
$$y_2 = x_2 + \Delta x_2 = 3.000000000258998$$

Que se convierten en los nuevos valores de prueba.

Con estos nuevos valores de prueba los valores de las ecuaciones son:

$$-f_1 = -f_1(x_1, x_2) = 0.000000003271801756604686$$
$$-f_2 = -f_2(x_1, x_2) = -0.000000000463784562837465 9$$

Que ya tiene cerca de 9 dígitos de precisión, por lo que el proceso puede concluir en este punto, sin embargo, repitiendo el mismo una vez más se obtiene:

$$h = x_1 * 10^{-6} = 0.000002000000000040957$$

$$\frac{\partial f_1}{\partial x_1} = \frac{f_1(x_{1+h}, x_2) - f_1(x_{1-h}, x_2)}{2h} = 4.00000000008168$$
$$\frac{\partial f_2}{\partial x_1} = \frac{f_2(x_{1+h}, x_2) - f_2(x_{1-h}, x_2)}{2h} = -4.999999999904775$$

$$h = x_2 * 10^{-6} = 2 * 10^{-6} = 0.00000300000000258998$$

$$\frac{\partial f_1}{\partial x_2} = \frac{f_1(x_1, x_{2+h}) - f_1(x_1, x_{2-h})}{2h} = 12.00000000103571$$
$$\frac{\partial f_2}{\partial x_2} = \frac{f_2(x_1, x_{2+h}) - f_2(x_1, x_{2-h})}{2h} = -0.9999999999591921$$

Resolviendo el sistema de ecuaciones lineales y añadiendo los resultados del mismo a las variables asumidas, los nuevos valores de las incógnitas son:

$$y_1 = x_1 + \Delta x_1 = 2$$
$$y_2 = x_2 + \Delta x_2 = 3$$

Que se convierten en los nuevos valores de prueba.

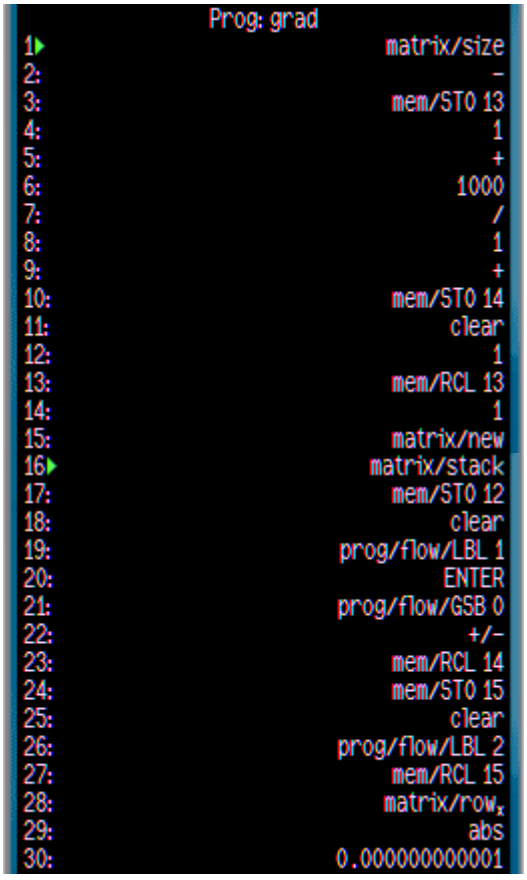
Reemplazando estos nuevos valores en las ecuaciones se obtiene:

$$-f_1 = -f_1(x_1, x_2) = 0$$
$$-f_2 = -f_2(x_1, x_2) = 0$$

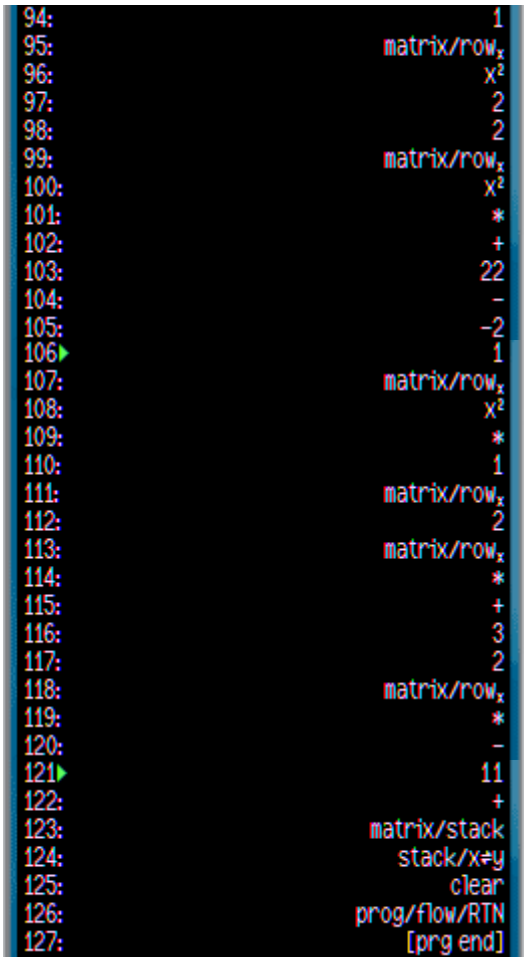
Lo que nos indica que dichos valores constituyen las soluciones exactas, por lo tanto las soluciones del sistema son: x_1=2 y x_3=3.

11.2.1. Programa

El programa para el método de la gradiente, donde se siguen los pasos ejemplificados en el anterior acápite es el siguiente:

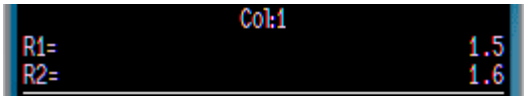


```
31▸                 prog/flow/x<y?
32:               prog/flow/GTO 3
33:                        clear
34:                        clear
35:               prog/flow/ISG 15
36:               prog/flow/GTO 2
37:                        clear
38:                 prog/flow/RTN
39:               prog/flow/LBL 3
40:                        clear
41:                        clear
42:                   stack/x≠y
43:                           0
44:                   stack/x≠y
45:                  mem/RCL 14
46▸                  mem/STO 15
47:                        clear
48:               prog/flow/LBL 4
49:                  mem/RCL 15
50:                 matrix/row$_x$
51:                    0.000001
52:                           *
53:                   stack/x≠y
54:                  mem/RCL 12
55:               stack/RCL st# 2
56:                           *
57:               stack/RCL st# 1
58:               stack/RCL st# 1
59:                           +
60:               prog/flow/GSB 0
61▸               stack/RCL st# 2
62:              stack/move dn# 2
63:                           -
64:               prog/flow/GSB 0
65:                           -
66:                           2
67:              stack/move dn# 3
68:                           *
69:                           /
70:              stack/move dn# 2
71:                   stack/x≠y
72:                 matrix/concat
73:                   stack/x≠y
74:                  mem/RCL 12
75:                  mem/RCL 13
76▸                 matrix/split
77:                   stack/x≠y
78:                 matrix/stack
79:                  mem/STO 12
80:                        clear
81:               prog/flow/ISG 15
82:               prog/flow/GTO 4
83:              stack/move up# 2
84:                          -1
85:                 matrix/split
86:                   stack/x≠y
87:                        clear
88:                          1/x
89:                   stack/x≠y
90:                           *
91▸                           +
92:               prog/flow/GTO 1
93:               prog/flow/LBL 0
```



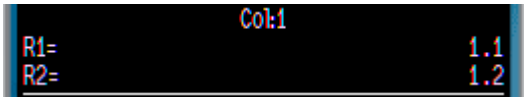
El método propiamente está programado hasta la línea 92. A partir de la línea 93 (etiqueta 0) comienza la subrutina donde se escriben las ecuaciones correspondientes al sistema (igualadas a cero).

En esta subrutina se reciben los valores de las variables en un vector y se calculan los resultados de cada una de las ecuaciones, apilando los resultados en un vector (con stack). Finalmente, antes de devolver el vector resultante se elimina el vector recibido (el vector con los valores de las variables). El programa recibe como datos el vector con los valores iniciales asumidos, así por ejemplo, haciendo correr el programa con los siguientes valores iniciales:

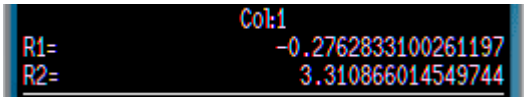


Se obtienen las soluciones:

Que son las mismas soluciones del ejemplo manual. Por supuesto si se emplean otros valores iniciales es posible que el método converja a otras soluciones. Por ejemplo, haciendo correr el programa con los valores:



Se obtiene:



Que son otra de las soluciones, de las cuatro existentes entre x=-5 y x=5 (mostradas gráficamente y calculadas en el anterior acápite).

En el programa se trabaja con una exactitud de 12 dígitos, que por lo general es suficiente para la mayoría de los cálculos ingenieriles, sin embargo, dicha exactitud puede ser cambiada ya sea para incrementarla o para reducirla modificando el valor de la línea 30.

11.2.2. Ejemplo

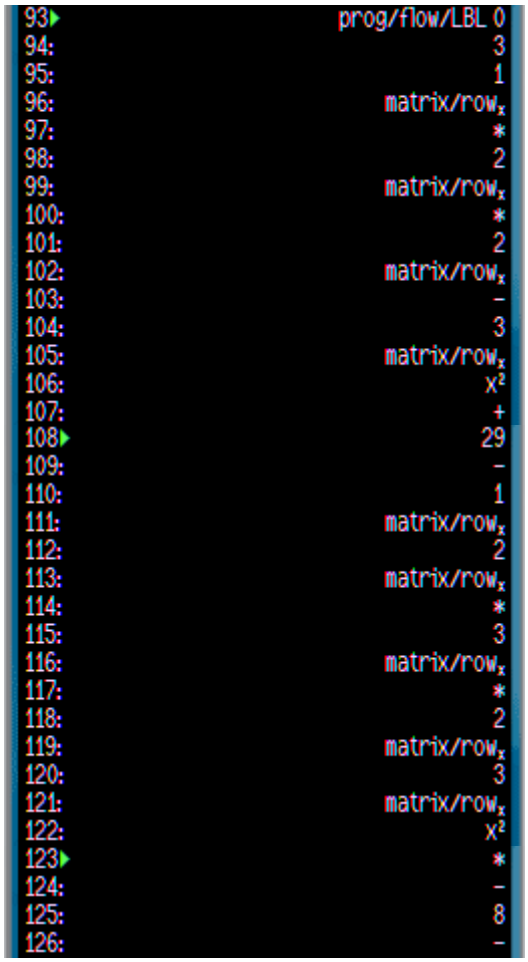
3. Encuentre las soluciones del siguiente sistema de ecuaciones lineales empleando el método de Gauss - Seidel.

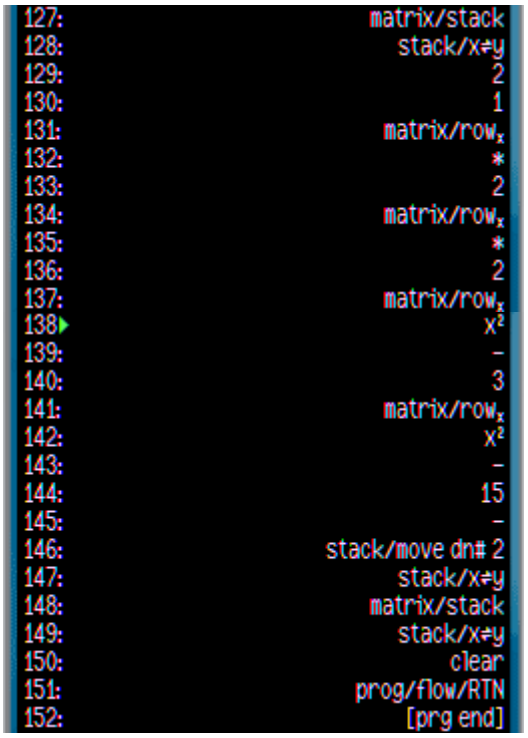
$$3xy - y + z^2 = 29$$

$$xyz - yz^2 = 8$$

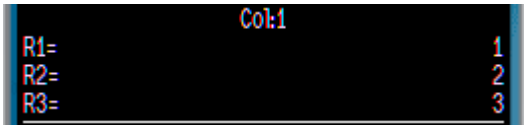
$$2xy - y^2 - z^2 = 15$$

El programa en sí constituye un primer ejemplo para el método de la gradiente. Como un segundo ejemplo se resolverá el sistema de tres ecuaciones no lineales presentado, para ello programamos las tres ecuaciones a partir de la línea 94 del programa “grad”:

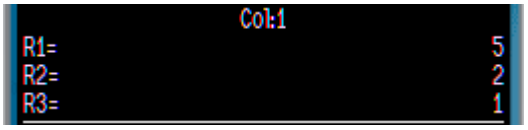




Ahora haciendo correr el programa con los siguientes valores iniciales:



Se obtiene:



Por lo tanto las soluciones del sistema de ecuaciones no lineales son: x=5; y=2; z=1.

11.2.3. Ejercicios

4. Encuentre las soluciones del siguiente sistema de ecuaciones no lineales empleando el método de la gradiente (valores iniciales: x=-1, y=-2).

$$4 - x^2 + y^2 = 0$$
$$1 - e^x - y = 8$$

5. Encuentre las soluciones del siguiente sistema de ecuaciones no lineales empleando el método de la gradiente (valores iniciales: x=1.1, y=1.2).

$$e^x - y = 0$$
$$xy - e^x = 0$$

6. Encuentre las soluciones del siguiente sistema de ecuaciones no lineales empleando el método de la gradiente (valores iniciales: x=1, y=1.2, z=1.6).

$$x^2 + y^2 + z^2 = 17$$
$$xyz = 12$$
$$x + y - z^2 = -4$$

7. Encuentre las soluciones del siguiente sistema de ecuaciones no lineales empleando el método de la gradiente (valores iniciales: x=1, y=1.1, z=1.2).

$$xyz - x^2 + y^2 = 1.34$$
$$xy - z^2 = 0.09$$
$$e^x - e^y + z = 0.41$$

8. Encuentre las soluciones del siguiente sistema de ecuaciones no lineales empleando el método de la gradiente (valores iniciales: $x1=_1$, $x_2=1$, $x_3=1$, $x_4=x$, $x_5=1$, $x_6=1$, $x_7=1$, $x_8=1$, $x_9=1$, $x_{10}=1$ y $x_{11}=10$).

$$x_1 + x_4 = 3$$
$$2x_1 + x_2 + x_4 + x_7 + x_8 + x_9 + 2x_{10} = 10 + r$$
$$x_2 + 2x_5 + x_6 + x_7 = 8$$
$$2x_3 + x_5 = 4r$$
$$x_1 x_5 = a_1 x_2 x_4$$
$$x_6 x_2^{1/2} = a_2 (x_2 x_4 x_{11})^{1/2}$$
$$x_7 x_4^{1/2} = a_3 (x_1 x_4 x_{11})^{1/2}$$
$$x_8 x_4 = a_4 x_2 x_{11}$$
$$x_9 x_4 = a5x1 (x_3 x_{11})^{1/2}$$
$$x_{10} x_4^{\,2} = a_6 x_4^{\,2} x_{11}$$
$$x_{11} = x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10}$$
$$a_1 = 0.193; \quad a_2 = 0.002597; \quad a_3 = 0.003448;$$
$$a_4 = 0.00001799; \quad a_5 = 0.002155; \quad a_6 = 0.00004836;$$
$$r = 4.056734$$