PET Eng. Química – UFC

Minicurso De UserRPL

VIRTVS VNITA FORTIOR

PET · IEQ ENGENHARIA QUÍMICA · UFC

Prefácio

A motivação para a realização deste curso vem do fato de a HP50g ser uma ferramenta de cálculo bastante utilizada tanto para estudantes de engenharia como para engenheiros formados. Porém ainda um pouco mistificada. Este minicurso tem o intuito de mostrar como é simples a programação da HP de modo a que esta programação possa tornar-se mais uma linguagem que venha ao auxílio dos estudantes. Pese a isto o fato de que a calculadora programável é bastante compacta e rápida de manusear-se conferindo neste sentido vantagens em relação aos computadores: É mais fácil carregar uma calculadora que um notebook, por exemplo.

Aos estudantes lhes digo que a diferença principal entre a programação na HP e as outras linguagens de programação comuns (C, C++, Fortran, Pascal) é o fato de que nela usa-se predominantemente o RPN (Notação Polonesa Inversa) e isto será trabalhado ao longo do minicurso.

O uso desta calculadora apenas para operações aritméticas simples seguramente configura limitação, posto que as possibilidades de operações são quase tão imensas quanto se possa imaginar. A calculadora possui diversas ferramentas, porém o minicurso se restringirá à programação, dita UserRPL, que é a programação de alto nível da HP.

Fellipe Carvalho de Oliveira

Pet – Eng. Química UFC

Janeiro, 2012

Sumário

1. Introdução

Ao se programar a seqüência de comandos é colocada na memória RAM da calculadora onde será feito o processamento passo a passo. Para que se gere resultado coerente não deve haver erros de lógica tampouco de sintaxe. A lógica fica a cargo do programador. A retidão da sintaxe (de acordo com os padrões da linguagem) é verificada pela calculadora que compilará se esta estiver correta ou acusará erro senão for o caso. Nosso editor de código UserRPL nos auxilia com vários comandos pré-formados.

1.1 Entendendo um Programa: “Hello World”

Um programa em UserRPL é uma sequência de comandos, números ou outros objetos delimitados por « ». Faremos agora nosso primeiro programa, digite o seguinte código no editor e aperte F9, numa calculadora virtual aparecerá:

« "Hello World" »

A frase “Hello World” é colocada no nível 1 da pilha. Outro programa seria:

« 10 2 / »

O número 5 é colocado no nível 1 da pilha. Percebe-se desta maneira como trabalha o RPN.

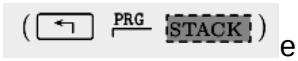
Primeiro os argumentos são colocados, logo então a função.

Os comentários são feitos através do símbolo @. Por exemplo:

« "Hello World" @ Este objeto é uma String ».

Observe que a segunda frase não aparece no visor da calculadora.

1.2 Manejo da Pilha e Funções para Números Reais

As funções para manejo da pilha estão no menu **STACK** 

e são elas:

Comando **DUP**

Este comando duplica o primeiro nível da pilha

Comando **SWAP**

Este comando troca o primeiro nível com o segundo.

Comando **DROP**

Apaga o primeiro nível da pilha

Comando **OVER**

Copia o nível 2 para o nível 1

Comando **ROT**

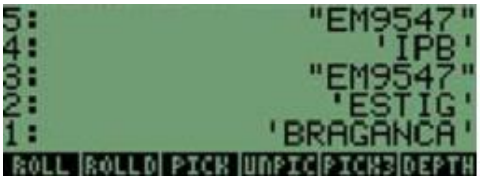
Roda em painel rolante os três primeiros níveis de baixo para cima.

Comando **UNROT**

Roda em sentido contrário do ROT

Comando **ROLL**

4 ROLL



Este comando tem de inserir o nível que quer trocar pelo primeiro, neste caso troquei o 4, (4 ROLL).

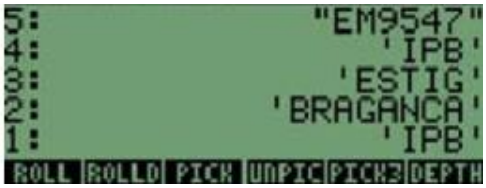
Comando **ROLLD**

3 ROLLD

Este comando passa o nível 1 para o nível indicado neste caso passamos o 1 para o 3 (3 ROLLD).

Comando **PICK**

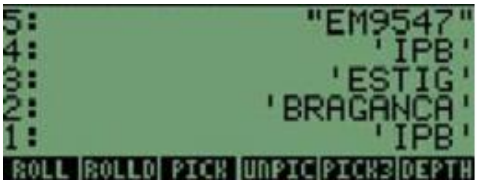
3 PICK

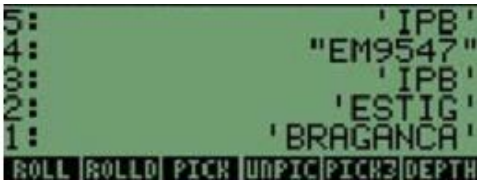


Este comando permite Copiar o nível desejado para o nível 1 neste caso copiamos o nível 3 (3 PICK).

Comando **UNPICK**

5 UNPICK





Este comando é similar ao anterior cota o nível 1 e cola no nível desejado, neste caso o nível 5 (5 UNPICK).

Comando **PICK3**

Este comando Copia para o nível 1 o comando que se encontra no nível 3

Comando **DEPTH**

Este comando dá-nos a quantidade de níveis que temos em na pilha, neste caso são três.

Comando **DUP2**

Este comando duplica os objetos que se encontrem nos dois primeiros níveis.

Comando **DUPN**

1 DUPN

Este comando é similar ao anterior, mas temos de fornecer o número de níveis que queremos duplicar.

Comando **DROP2**

Este comando apaga os dois primeiros níveis da pilha

Comando **DROPN**

3 DROPN

Este comando permite apagar um determinado número de níveis na pilha neste caso apagamos 3 (3 DROPN).

Comando **DUPDU**

Este comando triplica o objeto que esta no nível 1 da pilha.

Comando **NIP**

Este comando elimina o objeto que se encontra no nível 2.

Comando **NDUPN**

3 NDUPN

Este comando duplica n vezes o primeiro nível.

As funções para manejo da pilha estão no menu **REAL** (← MTH REAL) e são elas:

Comando **ABS**

Valor Absoluto

Comando **CEIL**

Menor inteiro maior ou igual ao argumento.

Comando **FLOOR**

Maior inteiro menor ou igual ao argumento.

Comando **FP**

Parte fracionária do argumento.

Comando **IP**

Parte inteira do argumento.

Comando **MANT**

Mantissa do argumento.

Comando **MAX**

Máximo valor entre os dois primeiros níveis da pilha.

Comando **MIN**

O menor valor entre dois primeiros níveis da pilha.

Comando **MOD**

Resto da divisão entre dois números.

Comando **RND**

Arredonda o número de acordo com o valor do argumento: n= 0 até 11.

Comando **SIGN**

Retorna +1 para argumentos positivos, −1 para argumentos negativos e 0 para argumentos nulos.

Comando **TRNC**

Trunca o número de acordo com o valor do argumento: n= 0 até 11.

1.3 Variáveis Globais e Locais

Variáveis são como arquivos em um disco rígido de computador. Uma variável pode armazenar um objeto (valores numéricos, expressões algébricas, listas, vetores, matrizes, programas, etc).

As variáveis são reconhecidas pelos seus nomes, que podem ser qualquer combinação de caracteres alfabéticos ou numéricos, iniciando com uma letra. Alguns caracteres não alfabéticos, tais como a seta (→) podem ser usados em um nome de variável, se combinados com um caractere alfabético. Assim, → A é um nome válido de variável, mas → não é. Exemplos válidos de nomes de variáveis são: A, B, a, b, α, β, A1, AB12, → A12, Vel, Z0, Z1, etc. Uma variável não pode ter o mesmo nome de uma função da calculadora. Você não pode ter uma variável SIN, por exemplo, já que existe um comando SIN na calculadora. Os nomes reservados das variáveis da calculadora são os seguintes: ALRMDAT, CST, EQ, EXPR, IERR, IOPAR, MAXR, MINR, PICT, PPAR, PRTPAR, VPAR, ZPAR, der−, e, i, n1, n2. . ., s1, s2, . . ., DAT, PAR, π , ∞.

Nota: Letras maiúsculas e minúsculas não são equivalentes (é Case Sensitive).

O comando → define nomes de variáveis locais (isto é, variáveis que somente são válidas dentro do programa em que foram definidas) − ao contrário das variáveis globais, que são definidas pelo comando **STO**.

Variáveis globais são todos os objetos que estão armazenados na sua calculadora e que serão criadas permanentemente (por assim dizer) manualmente ou através de programas. Seu uso dentro do programa representa uma desvantagem uma vez que trabalhar com essas variáveis é um pouco mais lento e para apagá-las é necessário executar o comando **PURGE**. Na maioria dos casos você não terá necessidade de usar variáveis globais já que as variáveis geralmente são usadas apenas temporalmente e aí entram as variáveis locais. Vejamos um exemplo do uso de variáveis globais.

```
«
6 'X' STO
8 'Y' STO
'X ^ 2 + Y ^ 2' EVAL
```

»

Ao executar o programa note a criação de duas novas variáveis no seu diretório atual. Se queremos que as variáveis não apareçam devemos fazer o seguinte:

```
«
6 'X' STO
8 'Y' STO
'X ^ 2 + Y ^ 2' EVAL
'X' PURGE
'Y' PURGE
»
```

Otimizando:

```
«
6 'X' STO
8 'Y' STO
'X ^ 2 + Y ^ 2' EVAL
{ X Y } PURGE
»
```

Variáveis locais são aquelas variáveis temporais do programa, ou seja, estão presentes apenas na execução de programas e serão as variáveis com as quais vamos trabalhar com freqüência.

Uma estrutura de variável local possui uma das seguintes organizações dentro de um programa:

“ → nome1 nome2. . . nomeN ‘objeto algébrico’ “

Ou,

“ → nome1 nome2. . . nomeN “ programa “ “

O comando (de atribuição) → remove n objetos da pilha e os armazena nas variáveis locais, cujos nomes são listados em seguida. Exemplo:

```
«
1 2 3 →□ a b c
«
a b c + +
»
»
```

Essas variáveis só podem ser utilizadas dentro deste subprograma e não são encontradas em outros ambientes ou subprogramas. No caso de variáveis locais não é necessária sua remoção uma vez que estas são eliminadas logo após o término do subprograma.

1.4 Operadores de Teste e Lógicos

Funções obtidas da seção **TEST** ():

Função de comparação	Descrição
SAME	Testa se dois objetos são iguais
==	Testa se dois objetos são iguais
≠	Testa se dois objetos são diferentes
<	Testa se o obj2 é maior que o obj1
>	Testa se o obj2 é menor que o obj1
≥	Testa se o obj2 é maior ou igual que o obj1
≤	Testa se o obj2 é menor ou igual que o obj1

Todas as funções acima retornam 1 para teste verdadeiro ou 0 para teste falso

Operadores lógicos ():

Função de comparação	Descrição
AND	Retorna 1 se os dois argumentos são verdadeiros (0 senão).
OR	Retorna 1 se os dois argumentos ou algum deles é verdadeiro (0 senão).
XOR	Retorna 1 se algum dos dois argumentos é verdadeiro. (0 se os dois são verdadeiros ou se os dois são falsos).
NOT	Retorna 1 se o argumento é 0 e vice-versa.

Exemplo

```
«
3 6 →□ a b
«
a b <
»
»
```

Exemplo

```
«
0 1 AND
»
```

1.5 Armazenando um Programa Na Calculadora

Digite qualquer um dos seguintes códigos no editor:

```
«
 → a b
 «
 '√(a+b)' EVAL
 »
»
```

```
«
 → a b
 «
 a b + 0.5 ^
 »
»
```

Perceba que eles dão o mesmo resultado. O da direita usa o RPN. .

Para guardar o programa em formato reconhecível para o editor apenas vá a ARQUIVO → SALVAR e então escolha um local para salvar o arquivo.

Vamos agora simular o armazenamento de um programa na calculadora.

Após digitar-se o programa no editor vá a EMULADOR -> ENVIAR AO EMULADOR. No emulador clique ENTER. Digite um nome para o programa e clique **ENTER**. Clique então em **STO**. Pronto o programa está armazenado na calculadora. No emulador visualize-o em 'seta para esquerda' → **APPS** → **OK**.

1.6 Executando um programa

Para executar o programa anterior através do editor digite os dois valores que irão ser armazenados nas variáveis locais 'a' e 'b' antes da seta e clique em F9. Observe o emulador.

Para executá-lo no emulador coloque os dois valores que serão armazenados pelas variáveis locais na pilha e clique em **VAR**. Procure o programa que você armazenou na tela de menus e clique nele.

1.7 Visualizando e Editando um Programa na Calculadora

Na calculadora clique em 'seta para esquerda' → **APPS** → **OK** procure o programa e então clique em **EDIT.** Faça as devidas alterações e clique **ENTER**.

1.8 Executando o DBUG

A calculadora nos oferece um recurso muito importante, principalmente para quem se inicia em programação, que é a possibilidade de se executar um programa passo-a-passo. Este recurso é poderoso e deve, amiúde, ser utilizado, por duas razões:

- Permite que se compreenda o funcionamento (e a lógica) de um programa;

- Facilita a correção de eventuais erros.

Este recurso nós o chamamos de **DBUG**. E fácil entender como um programa trabalha (funciona) se você executá-lo passo-a-passo, observando o efeito de cada etapa (comando). Isto facilita "depurar" seus próprios programas ou entender programas escritos por outros.

Este comando se encontra em 

.

Para utilizar o **DBUG** no editor após digitar-se o programa clique em F8 e vá ao emulador. Para avançar instrução por instrução vá clicando em SST. Se se quiser avançar de uma vez o programa digite **CONT** e aperte **ENTER** (ou então faça ON).

Para executá-lo na calculadora:

- Digite os valores requeridos na pilha e na ordem correta (se isto for necessário);
- Digite o nome do programa entre aspas simples e coloque-o na pilha (o programa pode ser acessado através do comando **VAR**).
- Vá a 

e vá avançando em SST.

Para avançar passo-a-passo no meio do programa:

1. Insira o comando **HALT** na linha de programa a partir da qual você deseja iniciar o avanço manual (**DBUG**);
2. Execute o programa normalmente. Ele pára quando o comando **HALT** é executado, o anúncio HALT é exibido (na tela);
3. Tome qualquer ação;

• Para ver o próximo passo do programa exibido na área de mensagens e então executá-lo, pressione o menu SST.;

• Para ver, mas não executar o próximo (ou os próximos dois passos), pressione **NXT.**

• Para continuar com a execução normal, pressione **CONT** (ON).

Quando você desejar que o programa seja executado normalmente (novamente) remova o comando **HALT** do programa. O comando **KILL** interrompe a depuração em qualquer instante.

2. Os objetos da Calculadora

O primeiro passo para iniciar-se na programação é a compreensão completa dos objetos que você pode trabalhar na calculadora já que a variedade e flexibilidade destes servirá para escolher corretamente o método de trabalho para se construir um programa em particular. Os objetos com os quais a calculadora trabalha são muito poderosos. As listas, por exemplo, têm uma grande variedade de usos. Vejamos, então, um pouco sobre os objetos os quais lida a calculadora, não vamos explicar todos, apenas os mais comumente utilizados na programação. Se você quer saber o tipo de objeto definido na calculadora você pode usar o comando **TYPE**. A lista completa está abaixo:

Tipo	Descrição	Exemplo
0	Número real	6.023E23
1	Número complexo	(2,5)
2	Texto String	“Orale”
3	Arrays (Vetor ou Matriz)	[1 3], [[2 3] [6 4]]
4	Arrays com números complexos	[(2,3) (1,2)]
5	Listas	{0 1 2}
6	Nomes	‘x’
7	Variável em uso	‘i’« FOR i 1+ NEXT»
8	Programa	« X 3 / »
9	Algébrico (Equação)	‘ X + 3 ’
10	Número Binário	#05B15h
11	Grob (gráficos)	Graphic 131x64
12	Objeto Etiquetado	Dados: 5354
13	Unidades	25_Km
14	Nome XLIB	XLIB 543 8
15	Diretório	DIR A5B2....END
16	Biblioteca	Library 1500: FIS2
17	Backup Objeto	Backup MYDIR
18	Função Interna	SIN
19	Comando Interno	CLEAR
20	Internal binary integer	<128d>
21	Número real estendido	1.23E2
22	Número complexo estendido	Complexo longo

24	Caractere	®
25	Code Object	Code
26	Dados de Biblioteca	Library Data
28	Números inteiros	25
30	Fontes	Ft8_25: Wilancha

2.1 Números Reais e Complexos

Os números reais (Regidos pelas funções para números reais Cap.1) podem ser, por exemplo:

2.5 5 -3.6

Os números complexos podem ser usados como coordenados cartesianas ou polares. Muitas das funções usam esses objetos como argumentos (especialmente comandos para gráficos).

Exemplo:

(5,8) ,este seria o complexo 5+8i, mas também pode ser a coordenada: X=5, Y=8.

2.2 String

Strings são bastante flexíveis e seu uso é dado geralmente para descrever as etapas dos programas, a flexibilidade destes nos permitem orientar o usuário através de entradas de dados.

Exemplo:

"Entre com o primeiro argumento"

O delimitador para este tipo de objeto são as aspas.

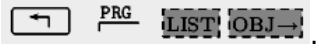
Para concatenar strings use o símbolo +. Para transformar um objeto em string use o comando →**STR**. Para verificá-los faça:

« 7 ‘a’ STO 70 ‘b’ STO

"Perdoarás não " a →STR + “vezes , porém ” b →STR + + " vezes 7 " +

»

Execute o programa acima no **DBUG** para ver seu funcionamento passo-a-passo.

Para desmontar uma string faça 

. Então na pilha aparecerá no nível 1 a seqüência de caracteres que havia na string. Se haviam palavras separadas por espaços na string ao desmontá-la surgirão strings diferentes para cada palavra.

2.3 Objeto Algébrico

Em geral vêem a ser as equações e são de uso regular dentro dos programas.

Exemplo:

'X+Y'

Para encontrar o valor de uma operação pode-se usar este objeto em vez das operações comuns da pilha. Vejamos:

```
«
 6 12 → x y
  «
  'x^2+y^2' EVAL
  »
»
```

A função **EVAL** é necessária neste caso. Esta função revela o conteúdo de qualquer variável podendo-lhe estar atribuído a esta variável qualquer objeto (matrizes, listas, complexos e etc.).

2.4 Programa

Os programas realizam seqüências de operações. Os programas também podem ser usados como funções dentro das equações, já que são capazes de recolher e devolver argumentos, desta forma a programação fica mais organizada, sendo esta prática bastante recomendável.

2.5 Listas

Uma lista é constituída de objetos (números, letras, gráfico, etc.) entre chaves e separados por um espaço (tecla **SPC**). Uma lista é o que, em matemática, comumente conhecemos por conjunto. Exemplo de lista:

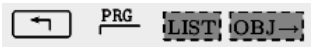
```
«
 { 1 5 a { b c} }
»
```

Este é um recurso muito importante para manipulação de objetos.

Para gerar uma lista a partir de objetos da pilha faça:

- Ponha os objetos na ordem a qual se quer que estejam na pilha. Esta ordem será a mesma destes objetos na lista.
- Digite o número de objetos e aperte **ENTER**.
- Vá a 

.

Para desmontar uma lista faça 

. Então na pilha aparecerá no nível 1 o número de objetos que havia na lista, logo então os objetos na ordem em que estavam na lista.

Para concatenar listas ou para agregar-lhes elementos use a função **+**. Para somar os elementos de duas listas use o comando **ADD**. Para multiplicar, subtrair e dividir elementos de duas listas use as seguintes funções ***, -, /.*** Para realizar potenciação use a função **^**.

Exemplo:

```
«
 { 1 5 6 44 } 555 +
 { 1 5 6 44 } { 5 6 8 7 } +
 { 1 5 6 44 } { 5 6 8 7 } ADD
 { 1 5 6 44 } { 5 6 8 7 } *
 { 1 5 6 44 } { 5 6 8 7 } -
 { 1 5 6 44 } { 5 6 8 7 } /
{ 1 5 6 44 } ^ 2
»
```

Execute-o no **DBUG** para ver seu funcionamento passo-a-passo.

2.6 Matrizes

A HP−50g possui grandes recursos para entrada e manipulação de matrizes. Muitas das operações descritas aqui também se aplicam a vetores (que são matrizes com apenas uma linha ou uma coluna). O nome matrizes também inclui objetos vetoriais.

Para criar-se vetores e matrizes no editor use os símbolos [ ], e [[ ]] respectivamente. Por exemplo:

```
«
[ [ 2   3 ]
[ 0  1 ]
[ 5  55 ]
]
»
```

Onde, no exemplo anterior, cada conjunto interno de colchetes é uma linha da matriz. Os elementos devem estar separados entre si por um espaço **SPC.**

No nosso editor apertando-se CRTL + F4 abre-se um editor de matrizes.

Para montar uma matriz a partir de uma seqüência de elementos:

1. Entre com os elementos na pilha na ordem de preenchimento por linhas.

2. Entre com uma lista contendo as dimensões da matriz desejada na forma: {linhas colunas}.

3. Pressione  . Para montar a matriz.

Exemplo:

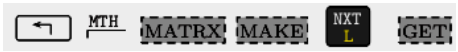
«
4 5 5 6 { 2 2 } →ARRY
»

Para extrair um elemento de uma posição específica:

1. Entre com a matriz na pilha.

2. Entre com um dos seguintes objetos:

- Uma lista contendo a linha e a coluna do elemento que você deseja extrair: {linha, coluna}.

3. Pressione  para extrair o elemento específico.

Exemplo:

«
4 5 5 6 { 2 2 } →ARRY
{ 1 1 } GET
»

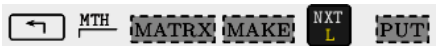
Para substituir um elemento de uma posição específica:

1. Entre com a matriz na pilha.

2. Entre com um dos seguintes objetos:

- Uma lista contendo a linha e a coluna do elemento que você deseja extrair: {linha coluna}.

3. Entre com o elemento substituto.

4. Pressione  para substituir o elemento específico na localização escolhida.

«
4 5 5 6 { 2 2 } →ARRY
{ 1 1 } 1.2323 PUT
»

Execute este último programa no **DBUG** para ver seu funcionamento.

Funções importantes para matrizes:

Símbolo	Exemplo
+ (Soma de matrizes)	« [[2 44] [3 5]] [[6 5] [78 9]] + »
- (Subtração de matrizes)	« [[2 44] [3 5]] [[6 5] [78 9]] - »
*(Multiplicação de matrizes)	« [[2 44] [3 5]] [[6 5] [78 9]] * »
INV (Obter matriz inversa)	«[[2 4] [3 5]] DUP INV »
TRN (Obter matriz transposta)	«[[2 4] [3 5]] TRN»
DET (Determinante)	«[[2 4] [3 5]] DET »
RDM (Redimensionar matriz)	« [[2 4] [3 5]] { 6 3 } RDM »

Execute os exemplos acima do **DBUG** para ver seu funcionamento. Observe que a lista que há antes da função **RDM** indica o novo número de linhas e colunas, respectivamente, da matriz redimensionada. Se há mais elementos que originalmente aos novos elementos lhes é atribuído o valor 0. Se há menos os elementos em excesso são eliminados.

Para utilizar um elemento da matriz numa expressão algébrica

1. Certifique-se de que a matriz está armazenada numa variável.

2. Crie a expressão algébrica e, no ponto onde o elemento de matriz será usado, digite o nome da matriz e pressione  .

3. Entre com os índices para o elemento:

- Para um vetor, digite um índice (número da posição do elemento).
- Para uma matriz digite dois índices separados por vírgula (número da linha, número coluna).
- A expressão resultante coloque entre aspas simples (Ex: ‘matriz(1,1)’) e ponha o **EVAL.**

Exemplo:

```
«
[ [ 2 4 ] [ 3 5 ] ] →  matriz
«
'matriz(1,1)'  EVAL  'matriz(2,2)'  EVAL +
»
»
```

Para desmontar uma matriz use **OBJ→**. No primeiro nível da pilha surge uma lista com o número de linhas e colunas da matriz, respectivamente. A partir do segundo nível vão sendo postos os elementos da matriz da esquerda para direita de cima para baixo. Execute o seguinte exemplo no **DBUG**:

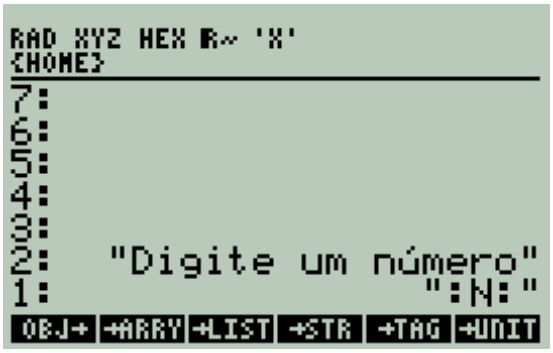
```
«  [ [ 2 4 ] [ 3 5 ] ]  OBJ→ »
```

3. Entrada de Dados

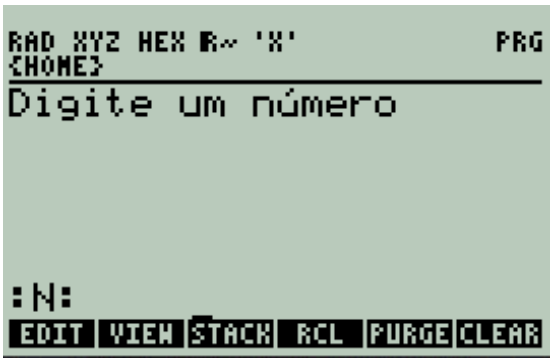
Um programa precisa de uma maneira ordenada para a entrada dos dados. A maneira mais natural de fazê-lo é através da pilha, porém a calculadora oferece maneiras mais apresentáveis. Todos os exemplos devem ser feitos no **DBUG** para melhor compreensão.

3.1 INPUT

Um dos comandos para se obter dados é o **INPUT** que nos apresenta uma tela com um título e um valor padrão em um editor similar à linha de comandos. Para realizar isto, o comando necessita de dois argumentos: uma string no nível 2 que será o título da tela e outra string no nível 1 que será o texto que aparecerá na linha de comandos.



Ao executar-se o **INPUT** obteríamos:



Logo, se se aceita a entrada, ou seja, se se pressiona **ENTER**, o **INPUT** retorna os dados introduzidos como uma string e que geralmente se converterão em objetos conhecidos com o comando **OBJ→**. Se se pressiona **CANCEL** cancela-se a execução do programa.

Exemplo:

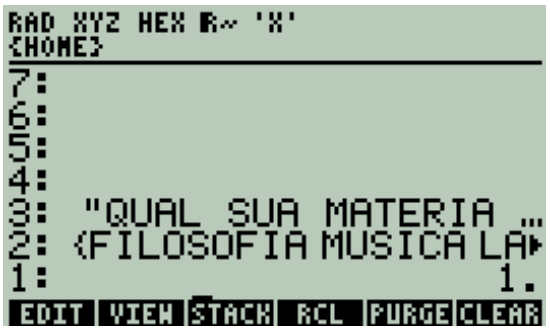
```
«
"Digite um número"
""
INPUT  OBJ→
»
```

No nosso editor o atalho para o **INPUT** é CTRL + F5

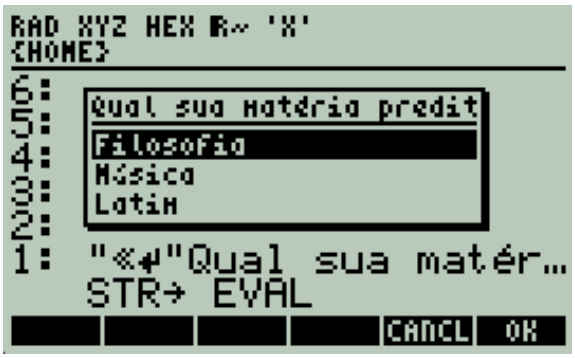
3.2 CHOOSE

CHOOSE se usa para escolher uma entre várias opções. Requer 3 argumentos: no nível 3 da pilha uma string que será o título da lista, no nível 2 uma lista com os elementos entre os quais escolher e no nível 1 o número correspondente a opção default, geralmente se coloca 1.

Este comando cria um menu de escolha no centro do display que permite a escolha usando o cursor (setas para cima e para baixo). Exemplo:



Aplicando-lhe **CHOOSE:**



A barra deve ser deslocada até o objeto desejado e em seguida pressionada a tecla **ENTER** ou **OK** para efetuarmos a escolha, ou **CANCEL** para cancelarmos. Se a função foi cancelada teremos como resposta o valor 0 na pilha. Se a função foi confirmada temos o valor 1 como resposta, e na segunda linha da pilha, teremos a opção (objeto) selecionada.

Exemplo:

«
"Qual sua matéria predita"
{
Filosofia
Música
Latim
}
1
CHOOSE
»

No nosso editor o atalho para o **CHOOSE** é CTRL + F6

3.3 INFORM

O **INFORM** permite-nos desenhar formulários para a introdução de dados mediante título, texto descritivo, de ajuda e de um tipo de objeto específico (facultativo). No nível 2 devolve os valores armazenados (que o usuário digitou) numa lista, e no nível 1 um valor que pode ser 1 se o formulário for desenhado corretamente e se se aperta **OK** ou **ENTER** ou 0 caso se cancele o formulário.

Sintaxe:

1- Nome que leva o formulário, este nome deve estar em string.

2- Elementos do formulário {{“nome” “ajuda” TIPO DE OBJETO(S)}}

3- Número de colunas que pode ser das seguintes formas: nº colunas, { nº colunas}, {nº colunas espaço}.

4- Valores Reset estes são os valores que restauram quando se pressiona a opção reset do menu do formulário {VALORES RESET}

5- Valores iniciais estes são valores de iniciais {VALORES INICIAIS}

6- **INFORM**

Aplicando-lhe **INFORM**

Exemplo:

```
«
"Formulário de nomes"
{
{ "nome 1" "primeiro nome" 2 }
{ "nome 2" "segundo nome"  }
{ "nome 3" "terceiro nome"  }
}
{ 1  0 }
{ Ataulfo Lago Mário }
{ Lira Alves Carlos }
INFORM
»
```

3.4 PROMPT

Este comando interrompe o programa e mostra uma mensagem na parte superior do display.

Sintaxe:

- String que contém a mensagem superior
- **PROMPT**
- Digite os valores de entrada separados por espaço **SPC**
- **ENTER**

Os valores digitados pelo usuário irão para a pilha cada um em um nível na mesma ordem em que foram digitados.

Logo após o **PROMPT** o programa é pausado automaticamente e o **HALT** é acionado. Para seguir a execução normal digite **CONT** e aperte **ENTER.**

3.5 DISP – FREEZE – HALT

O comando **DISP** mostra uma string em determinada fila da tela (enumeradas de 1 a 7, a tela é divida em 7 filas) ele precisa de dois argumentos: no nível 2 a string de mensagem e no nível 1 o número da fila do display onde será mostrada a imagem. Sozinho este comando só age enquanto dura a execução. Portanto usa-se outro comando para reter a tela do display chamado **FREEZE**. Este último comando retém a tela e precisa de um argumento para decidir que área da tela será congelada. A tabela a seguir mostra as possibilidades:

Argumento	Área Congelada
0	Toda a tela
1	A área de estado
2	A pilha
3	A área de estado e a pilha
4	Os menus
5	Os menus e a área de estado
6	A pilha e os menus

O comando **FREEZE** é interrompido assim que se aperta qualquer tecla, portanto até aqui não é feita entrada de dados, para tanto se utiliza o comando **HALT** que retorna à tela de comando para que se possa realizar operações: No nosso caso pôr dados na pilha. Para seguir a execução normal do programa use o **CONT**. Observe que o comando **HALT** por si só já seria suficiente para interromper a execução e permitir a entrada de dados (ou realizar quaisquer operações). Geralmente usa-se o comando **CLLCD** para limpar a área da pilha e a área de estado antes da execução do **DISP.**

Exemplo:

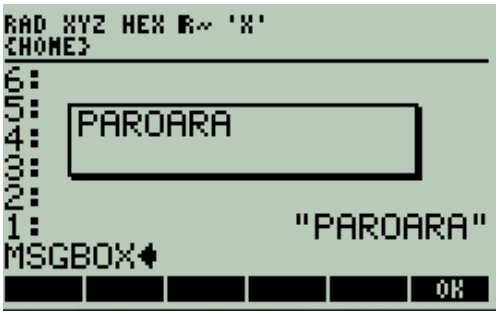
```
« CLLCD
"orale" 4 DISP 0 FREEZE HALT
 → a
« a 2 ^ » »
```

4. Apresentação de Dados

Comumente após a execução de um programa ou durante este, devemos apresentar alguns resultados. Igualmente ao caso passado a calculadora possui bons comandos para apresentar dados. Execute todos os exemplos no **DBUG.**

4.1 MSGBOX

MSGBOX mostra um quadro de mensagem e o único argumento que necessita é uma string, porém se queremos adicionar-lhe números ou outros objetos, simplesmente temos que converter-lhes em strings e somar-lhes.



Exemplo:

```
«
"Digite o raio do círculo"
""
INPUT  OBJ→
0  → r  a
«
r  2  ^ 'a' STO
"A ÁREA DO CÍRCULO É: "
 a  →STR  +
 MSGBOX
 »
»
```

4.2 TAG

A apresentação de dados se dá na pilha mediante a criação de um objeto etiquetado (criado pelo comando →**TAG**). O objeto do nível 2 da pilha se atrela ao do nível 1 .

Exemplo:

«

"Digite sua idade"

""

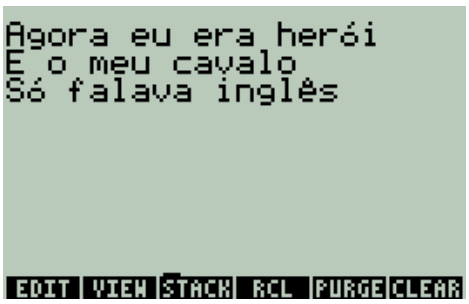
INPUT OBJ

"Sua idade é" →TAG

»

4.3 DISP – FREEZE

Dá-se de uma maneira análoga ao explicado no capítulo anterior. Agora a string de entrada será a mensagem que se quer expor.



Observe o exemplo abaixo no **DBUG**.

«

"Digite um primeiro número "

""

INPUT OBJ→

"Digite um segundo número"

""

INPUT OBJ→

CLLCD

* 'a' STO

```
"A Multiplicação " 1 DISP
"do primeiro pelo " 2 DISP
"segundo é:  " a →STR + 3 DISP
0 FREEZE
»
```

5. Trabalhando com Equações

O trabalho com equações tem desvantagens apenas com relação à velocidade, porém seu uso nos é mais familiar que as operações da pilha. A calculadora dispõe de muitas maneiras para criar, avaliar equações ou objetos algébricos. Observe o exemplo abaixo, tudo é muito intuitivo.

```
«
 1 →□ x
 «
  'F(X)=EXP(X)-2' DEFINE
  'F(x)' EVAL
  x F
 »
»
```

O comando **DEFINE** cria a função que trabalha de maneira similar a uma sub-rotina, que estudaremos mais adiante. Trata-se de uma passagem de parâmetros. Observe que após a execução do código a função F será armazenada na memória da calculadora, podendo ser acessada por outros programas. Para eliminá-la utilize **PURGE.**

Em geral se recomenda as operações de pilha, pode-se utilizar, por exemplo, **OVER, DROP, ROT, DUP** que são funções rápidas (ao menos para UserRPL) e os programas serão muito menores. Novamente, esta é só uma recomendação, sobretudo para aqueles que se interessam pela otimização.

Os programas podem ser utilizados como funções próprias da calculadora e dentro das equações ou expressões algébricas e devem neste caso devolver um ou mais valores para que sejam feitas novas operações. Por exemplo:

```
«
 « →□ a b
 «  a b + 2 /
 »
 » 'media' STO

 "Digite um primeiro número"
 ""
 INPUT OBJ→□
 "Digite um segundo número"
 ""
 INPUT OBJ→□
```

```
@ Vamos Agora aplicar a função Média
media
"A média aritmética dos dois números é" →TAG
‘media’ PURGE
»
```

Observe que a rotina que calcula a média foi guardada na variável global ‘media’ podendo ter sido chamada em outro instante. Na verdade o uso da função ‘media’ no exemplo anterior se enquadra no contexto da técnica das sub-rotinas que consiste na possibilidade de um programa ser acessado por outro. Uma situação em que é recomendável o uso de sub-rotina é quando temos um conjunto de instruções que é utilizado em diversas partes de um programa e para que não seja reescrito diversas vezes é colocado em um programa a parte onde o primeiro programa (podemos chamá-lo de principal) acessa o segundo (sub-rotina). Exemplo:

- Salve o seguinte o programa ‘media’ no emulador da calculadora;
- Digite e compile o seguinte código:

```
«
"Notas"
{
"1ª nota 1º bimestre: "
"2ª nota 1º bimestre: "
"1ª nota 2º bimestre: "
"2ª nota 2º bimestre: "
}
{ 1 0 }
{ }
{ }
INFORM
DROP OBJ→ DROP
→ a1 a2 b1 b2
    « a1 a2 media
      b1 b2 media
      media
      "A média semestral é" →TAG
    »  »
```

Na depuração de códigos para que as sub-rotinas sejam acessadas avance não mais em SST, porém em SST↓.

Porém antes de se realizar um programa que será utilizado como sub-rotina sempre será melhor verificar se há algum comando que realize a operação matemática que se deseja fazer, estes sempre são muito mais rápidos. Alguns serão vistos nos próximos tópicos:

5.1 Somatório

Argumento: Somatório no primeiro nível

Sintaxe:

'Σ(incógnita = valor inicial, valor final, equação ou objeto algébrico)' **EVAL.**

Exemplo:

« 'Σ(I=1,3,I+3)' EVAL »

5.2 Derivadas

Para derivada de função da incógnita X usa-se o comando DERVX

Argumento: Função de X no primeiro nível

Sintaxe:

'função de X' **DERVX**

Exemplo:

« 'LN(X)' DERVX »

Para derivada de funções de qualquer incógnita usa-se DERIV

Argumento: Função da incógnita no primeiro nível e a incógnita no segundo

Sintaxe:

'função de qualquer variável' 'variável' **DERIV**

Exemplo:

« 'L^12' 'L' DERIV »

Para divergente de um campo vetorial usa-se o DIV

Argumento: Vetor das componentes do campo vetorial no primeiro nível e vetor com as incógnitas no segundo

Sintaxe:

['1ª componente do campo vetorial' ' 2ª componente ' '3 ªcomponente Z '] ['variável 1' 'variável 2' ' variável 3'] **DIV**

Exemplo:

« ['X^2' 'Y^2' 'Z^2'] ['X' 'Y' 'Z'] DIV »

Para rotacional de um campo vetorial usa-se o CURL

Argumento: Vetor das componentes do campo vetorial no primeiro nível e vetor com as incógnitas no segundo

Sintaxe:

['1ª componente do campo vetorial' ' 2ª componente ' '3 ªcomponente Z '] ['variável 1' 'variável 2' ' variável 3'] **CURL**

Exemplo:

«

['Y*Z' 'X*Z' 'X*Y*Z'] ['X' 'Y' 'Z'] CURL

»

5.3 Integrais

Para integral indefinida de função da incógnita X usa-se o comando INTVX (é necessário que se esteja no modo exato e não no numérico).

Argumento: Função de X no primeiro nível

Sintaxe:

'função de X' INTVX

Exemplo:

«

'X^2' INTVX

»

Para integral de funções de qualquer incógnita usa-se INT (Se a calculadora estiver no modo exato a integral no ponto será mostrada, se estiver no modo numérico a integral será calculada).

Argumento: Função da incógnita no primeiro nível e a incógnita no segundo e no terceiro o ponto onde a antiderivada será avaliada, este último pode ser um número ou uma expressão

Sintaxe:

'função de qualquer variável' 'variável' 'ponto onde a integral será calculada' INT

Exemplo:

«

'2*m' 'm' '9' INT

»

Para integral definida usa-se o comando ∫.

Argumento: Nível 4 o limite inferior, no nível 3 o limite superior, no nível 2 a função da variável e no primeiro nível a variável.

Sintaxe: limite inferior limite superior 'função da variável escolhida' 'variável'

Exemplo:

«

2 5 'x^2-2' 'x' ∫

»

5.4 Equações Diferenciais

Para Resolução de Equações Diferenciais Ordinárias use o comando DESOLVE.

Argumento: no nível 2 a equação diferencial, no nível 1 a função desconhecida.

Sintaxe: 'Eq. diferencial' 'função desconhecida' DESOLVE

Exemplo: Resolve a seguinte equação diferencial ($\frac{d^2y}{dx^2} = x$)

«

«

'd1d1y(x)-x=0' 'y(x)' DESOLVE

»

Os cC0, cC1 ..... cCn que aparecem no resultado são as constantes de integração

6. Estruturas de Programação

Uma estrutura de programação (Decisão ou Repetição) permite a um usuário tomar uma decisão sobre como o programa deve ser executado, dependendo das condições dadas ou dos valores de argumentos em particular. Um uso cuidadoso e inteligente destas estruturas torna possível a criação de programas com extraordinária flexibilidade. Diríamos que a programação propriamente dita começa aqui com estruturas de programação, pois o que fizemos anteriormente foi a programação de fórmulas apenas. Estas estruturas que iremos estudar são comuns a várias linguagens de programação, como por exemplo, PASCAL, FORTRAN, C++, MATLAB, etc. Aqui mais do que nunca é necessário executar todos os exercícios no **DBUG**.

6.1 Estruturas de Repetição

Este tipo de estrutura executa determinada ação um número definido ou não de vezes.

6.1.1 START – NEXT, START – STEP

START... NEXT executa uma parte do programa um determinado número de vezes.

Sintaxe:

```
“…  início fim
START
cláusula do laço
NEXT
… ”
```

Exemplo: O programa abaixo faz uma lista de 10 cópias das letras “ABC”

```
«
1 10
START
 'ABC'
NEXT
10 →LIST
»
```

START toma dois números da pilha (início e fim) e os armazena como valor inicial e final para o contador do laço. Depois, executa a cláusula do laço. **NEXT** incrementa o contado em 1 e verifica se este valor é menor ou igual ao fim. Se for, executa novamente a cláusula do laço.

START... STEP funciona exatamente da mesma forma que o **START... NEXT**, exceto que permite especificar um incremento diferente de 1.

Sintaxe:

“… início fim
START
cláusula do laço
incremento
STEP
… ”

Exemplo: O programa seguinte toma um número x da pilha e calcula o quadrado deste número x/3 vezes.

```
«
9 →? x
«
 x 1
 START
 x SQ
 -3
 STEP
»
»
```

START toma dois números da pilha (início e fim) e os armazena como valor inicial e final para o contador do laço. Depois, executa a cláusula do laço. **STEP** toma o incremento da pilha e incrementa o contador com este valor. O incremento pode ser positivo ou negativo. Se for positivo, executa novamente a cláusula do laço quanto o contador é menor ou igual ao fim. Se o incremento é negativo, executa o laço quando o contador é maior ou igual ao fim.

6.1.2 FOR – NEXT, FOR – STEP

Um laço **FOR... NEXT** executa uma parte de um programa um número especificado de vezes, utilizando uma variável local como contador das iterações. Pode-se utilizar esta variável dentro do laço.

Sintaxe:

“... início fim
FOR contador
cláusula do laço
NEXT ...

”

Exemplo: O programa a seguir calcula o Fatorial de um número:

```
«
"Digite N"
""
INPUT OBJ
 → n
« 1 'produtorio' STO
  1 n
  FOR i
  produtorio i * 'produtorio' STO
  NEXT
  'produtorio' EVAL
»
»
```

FOR toma dois números da pilha (início e fim) e os armazena como valores inicial e final para o contador de iterações, depois cria uma variável local contador como contador de iterações. Depois, se executa a cláusula do laço. **NEXT** incrementa o contador em 1 e verifica se este valor é menor ou igual ao fim. Se for, executa novamente a cláusula do laço. Ao sair do laço o contador é apagado, ou seja, ele só existe dentro da cláusula do laço.

FOR... STEP funciona exatamente da mesma forma que **FOR... NEXT**, exceto que permite especificar um incremento diferente de 1 ao contador.

Sintaxe:

```
“... início fim
FOR contador
cláusula do laço
incremento
STEP ...
”
```

Exemplo: O programa seguinte calcula os quadrados dos inteiros ímpares de 1 a 21.

```
«
 1 21
 FOR j
 j SQ
```

```
 2
 STEP
»
```

FOR toma dois números da pilha (início e fim) e os armazena como valores inicial e final para o contador de iterações, depois cria uma variável local contador como contador de iterações. Depois, executa a cláusula do laço. **STEP** toma o incremento da pilha e incrementa o contador com este valor, e verifica se o valor do contador é menor ou igual ao fim. Se for, executa novamente a cláusula do laço. Ao sair do laço o contador é apagado, ou seja, ele só existe dentro da cláusula do laço.

6.1.3 DO – UNTIL – END

À diferença das outras duas estruturas de repetição estas próximas são indefinidas. O **DO... UNTIL... END** Executa repetidamente um laço enquanto a cláusula de teste retornar um valor falso. Como se executa primeiramente a cláusula do laço e depois a cláusula do teste executa-se ao menos uma vez o laço.

Sintaxe:

“... DO
cláusula do laço
UNTIL cláusula de teste
END ...”

Exemplo: O Exemplo abaixo testa se o número que se digita é igual a um valor da pilha

```
« 3 0 → r n
«
DO
"ADIVINHE O NUMERO"
        "" INPUT OBJ→ 'n' STO
        CASE n r >
THEN
         "MAIS BAIXO...!!!"
 END n r <
        THEN
"MAIS ALTO...!!!"
        END
"ACERTASTE!"
END
```

```
        MSGBOX
UNTIL r n ==
 END
»
»
```

DO começa a cláusula do laço. **UNTIL** finaliza a cláusula do laço e começa a cláusula de teste. A cláusula de teste deixa o resultado do teste na pilha. **END** extrai o resultado deste teste da pilha. Se o valor é zero, executa novamente a cláusula do laço; caso contrário, a execução do programa continua após o **END**.

6.1.4 WHILE – REPEAT – END

WHILE... REPEAT... END avalia repetidamente um teste e executa a cláusula do laço se o teste é verdadeiro. Como a cláusula do teste ocorre antes da cláusula do laço, nunca se executa um laço sem antes verificar se a cláusula do teste é verdadeira.

Sintaxe:

```
“... WHILE cláusula de teste
REPEAT
cláusula do laço
END ...””
```

Exemplo: O programa seguinte divide por dois todos os números pares de 1 até um valor especificado.

```
«
"Digite um número"
""
INPUT OBJ→ 'a' STO
1 'cont' STO
WHILE cont a ≤
REPEAT
IF cont 2 MOD 0 ==
THEN
cont 2 /
END
cont 1 + 'cont' STO
END
»
```

WHILE executa-se a cláusula do teste e devolve o resultado do teste para a pilha. **REPEAT** toma os valores da pilha. Se o valor é diferente de zero, continua a execução do laço, caso contrário, a execução do programa continua após o **END**.

6.2 Estruturas de Decisão

Este tipo de estrutura executa determinada ação se alguma condição é satisfeita.

6.2.1 IF – THEN – END

Usado para realizar determinadas ações se se cumpre uma condição**.**

Sintaxe:

```
“ . . . IF
Cláusula de teste
THEN
Cláusula verdadeira
END …
“
```

Exemplo:

```
«
"Digite um número"
""
INPUT OBJ→
 →  N
« IF N 2 MOD 0 ==
THEN
"O número é par. "
END
»
»
```

Neste programa o número posto pelo usuário foi avaliado quanto ao resto da divisão por 2 (função **MOD**). Se for 0 é porque o número é par.

6.2.2 IF – THEN – ELSE – END

Sintaxe:

```
“ . . . IF
Cláusula de teste
THEN
```

Cláusula verdadeira

ELSE

Cláusula falsa

END . . .

“

Esta estrutura executa uma seqüência de comandos se o teste resultar verdadeiro e outra, caso seja falso. Se a cláusula-de-teste for um objeto algébrico, ela é automaticamente convertida para um valor numérico. A palavra **IF** inicia a cláusula-de-teste, a qual deixa o resultado (0 ou 1) na pilha. **THEN** remove este resultado. Se o valor é diferente de 0, a cláusula-verdadeira é executada; caso contrário, a cláusula-falsa é executada. Após ter executado a cláusula apropriada, o programa prossegue com a instrução seguinte à **END**.

Exemplo:

(U.E. CE. - 80) Considere a sequência de números reais definidas por:

$$a_n = a_{n-1} \; se \; n \; é \; par \; ou$$

$$a_n = \frac{n+1}{2} \; se \; n \; é \; ímpar$$

Então o produto dos 6 primeiros termos é:

Resolução:

«

"Digite N"

""

INPUT OBJ→□

□ → N

« [0] { N } RDM 'A' STO

1 N

FOR n

IF n 2 MOD 0 ≠

THEN

'(n + 1)/2' EVAL 'A(n)' STO

ELSE

'A(n - 1)' EVAL 'A(n)' STO

END

NEXT

A

OBJ→□

DROP

```
1 N 1 -
FOR i
*
NEXT
"O Resultado é" →TAG
»
»
```

Observe que neste exemplo usou-se uma matriz para guardar valores. Se fôssemos usar as operações normais na pilha faríamos:

```
«
"Digite N
"
""
INPUT OBJ n
 «
1 n
FOR i
IF i 2 MOD 0 ==
THEN
DUP
ELSE
i 1 + 2 /
END
NEXT
1 n 1 -
FOR i
*
NEXT
"O resultado é" TAG
»
»
```

6.2.3 CASE – THEN – END – END

A estrutura **CASE... END** permite executar uma série de casos (testes). O primeiro teste que tem um resultado verdadeiro causa a execução da correspondente cláusula verdadeira, finalizando a estrutura **CASE... END**. Opcionalmente, pode-se incluir após o último teste uma cláusula de default que se executará se todos os testes forem falsos.

Sintaxe:

```
«
... CASE Cláusula de teste 1
THEN
Seqüências a se executar se a cláusula de teste 1 for verdadeira
END (Cláusula de teste 2)
THEN
Seqüências a se executar se a cláusula de teste 2 for verdadeira
END
```

Seqüências-padrão a se executar (se nenhuma das cláusulas anteriores for verdadeira), que são opcionais.

```
END
...
»
```

Exemplo:

Testar o tipo da variável. Não se esqueça de atribuir-lhe objeto.

```
«
CLLCD
→ y
«
CASE y TYPE 2 ==
THEN
"A variável é uma string " MSGBOX
END y TYPE 5 ==
THEN
"A variável é uma lista " MSGBOX
END y TYPE 8 ==
THEN
"A variável é um programa " MSGBOX
END
END
»
»
```

6.3 Exercícios

1). Faça um programa para sair com os N primeiros termos de uma P.A. (progressão aritmética), de dois modos distintos:

(a) em uma lista (b) em um vetor.

2). Faça um programa para calcular o produto dos N primeiros termos de uma P.A.

3). Faça um programa para calcular a soma dos N primeiros termos de uma P.G.

4). Faça um programa para calcular o seguinte somatório:

$$\sum_{i=1}^{N} i^2$$

5). Seja a seqüência:

1, 1, −1, −1, 1, 1, −1, −1, 1, 1, −1, −1, . . .

Cuja fórmula do termo geral e do produto são,

$$a_n = (-1)^{\frac{(n-1)(n-2)}{2}} \quad \text{e} \quad P_n = (-1)^{\frac{n(n-1)(n-2)}{6}}$$

Faça um programa para sair com um vetor contendo os N primeiros termos desta seqüência e mais ainda o produto destes N primeiros termos.

6). Faça um programa para calcular o somatório:

$$\sum_{i=1}^{n} i^m$$

Onde m e n são valores arbitrários que devem ser fornecidos ao programa.

7). Faça um programa para calcular para calcular a soma,

1·2 + 2·3 + 3·4 + ...+ n· (n + 1)

8). Faça um programa para calcular o produto interno canônico de dois vetores, assim:

[a1 a2 . . . an] · [b1 b2 . . . bn] = a1·b1 + a2·b2 + ... + an·bn

9). Faça um programa para sair com um vetor contendo os N primeiros termos de uma P.A., usando a fórmula de recorrência (definição) de uma P.A.:

$$\begin{cases} a_1 = a \\ a_n = a_{n-1} + r, \quad n \geq 2. \end{cases}$$

10). Faça um programa para sair com um vetor contendo os N primeiros termos de uma P.G., usando a fórmula de recorrência (definição) de uma P.G.:

$$\begin{cases} a_1 & = a \\ a_n & = a_{n-1} \cdot q, \quad n \geq 2. \end{cases}$$

11). Faça um programa para sair (em lista ou vetor) com os N primeiros termos das seguintes sequências:

i) $a1 = 4;\ an = (-1)^n \cdot a_{n-1}$.

ii) $a1 = -2;\ an = a_{n-1}{}^2$.

iii) (U.F.CE -81) Os termos da sucessão a1 , a2 , . . . , an, estão relacionados pela fórmula $a_{n+1} = 1 + 2 \cdot a_n$, onde n = 1, 2, 3, . . . Se a1 = 0, então a6 é:

a). 25 b). 27 c). 29 d). 31

12). (PUC-SP - 81) Na seqüência (a1 , a2 , . . .) tem-se:

$$a_1 = 1 \text{ e } a_{n+1} = \frac{2 + a_n^2}{2a_n}$$

Qual dos números abaixo está mais próximo de a3?

a) 1 b) 2 c)√2 d)√3 e)√5

Nota: Lembramos que nestes exercícios queremos que você faça um pouco mais do que o exercício pede. Queremos que você faça um programa para listar os n primeiros termos das seqüências dadas.

13). (U.F.BA. - 81) sejam as sequências

$$\begin{cases} a_1 & = 4 \\ a_{n+1} & = 1 + \frac{2}{a_n} \end{cases} \quad \text{e} \quad \begin{cases} b_1 & = 5 \\ b_{n+1} & = \frac{1}{1+b_n} \end{cases}$$

Se P = an·bn, tem-se:

a) P < 0 b) 0 ≤ P < 1 c) 1 ≤ P < 2 d) 2 ≤ P < 3 e) P ≥ 3

14). (U.F.PR. - 80) Seja f uma função tal que f(1) = 2 e f(x + 1) = f(x) − 1, para todo valor real de x. Então f(100) é igual a:

a) − 99 b) − 97 c) 96 d) 98 e) 100

15). (PUC - SP - 85) Na seqüência (a0 , a1 , a2 , . . .) onde $a_0 = 1\, e\, a_{n+1} = a_n + n$,para todo n ∈ N, a soma dos 7 primeiros termos é

a) 41 b) 42 c) 43 d) 63 e) 64

16). (PUC - SP - 70) Sendo f : R → R definida por f(x) = 2x + 3, então f(1) + f(2) + f(3) + . . . + f(25) é igual a:

a) 725 b) 753 c) 653 d) 1375 e) 400

17). (U. MACK. - 73) Sendo A = (aij) uma matriz quadrada de ordem 2 e aij = j – i 2, o determinante da matriz A é:

a) 0 b) 1 c) 2 d) 3 e) 4

18). Uma outra distância entre matrizes A = (aij) e B = (bij) é dada pela igualdade:

$$d(\mathbf{A}, \mathbf{B}) = \left[\sum_{i=1}^{m} \sum_{j=1}^{n} (a_{ij} - b_{ij})^2 \right]^{1/2}$$

onde A e B são matrizes retangulares de ordem m× n. Esta é conhecida como distância euclidiana. Programe a fórmula acima.

19). (UNESP - 84) Seja $Sn = \frac{1}{2^1} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^n}$, onde n é um número natural diferente de zero. O menor número n tal que Sn > 0,99 é:

a) 5 b) 6 c) 7 d) 8 e) 9

20). (F.C.M. STA. CASA - 82) Para que a soma dos termos da seqüência (−81, −77, −73, . . .) seja um número positivo, deve-se considerar no mínimo:

a) 35 termos b) 39 termos c) 41 termos d) 42 termos e) 43 termos

21). Determine N tal que:

$$\sum_{i=1}^{N} 2^i = 4094$$

.

22). Elabore um programa onde entramos com o primeiro termo e a razão de uma P.A., e ainda com um número L (maior ou igual ao primeiro termo), e o programa saia com n, a quantidade máxima de termos que podemos somar para que a soma não ultrapasse L.

23). Qual é o primeiro termo positivo da P.A. (−81, −77, −73, . . .)?

24). Encontre o valor de k de modo que:

$$\sum_{j=1}^{k} \sum_{i=1}^{30} (-2i + 3j) = 300$$

25). Quantos termos devem ser somados na seqüência ,

−1 3 − 1 3 – 1 3 − 1 3 − 1 3. . .

A partir do primeiro termo, para que a soma seja 13?

26). (PUC- SP - 76) Se A é uma matriz 3 por 2 definida pela lei:

$$a_{ij} = \begin{cases} 1, & \text{se } i = j; \\ i^2, & \text{se } i \neq j. \end{cases}$$

Então A se escreve:

a) $\begin{bmatrix} 1 & 4 & 9 \\ 1 & 1 & 9 \end{bmatrix}$ b) $\begin{bmatrix} 1 & 1 \\ 4 & 1 \\ 9 & 9 \end{bmatrix}$ c) $\begin{bmatrix} 1 & 1 \\ 1 & 4 \\ 9 & 9 \end{bmatrix}$ d) $\begin{bmatrix} 1 & 1 & 9 \\ 1 & 4 & 9 \end{bmatrix}$ e) $\begin{bmatrix} 1 & 1 \\ 4 & 1 \\ 6 & 6 \end{bmatrix}$

27). Mostre a matriz A cujos elementos estão dados pela relação,

$$a_{ij} = \begin{cases} (-1)^{j-1} \binom{j-1}{i-1}, & \text{se } i < j; \\ (-1)^{i-1}, & \text{se } i = j; \\ 0, & \text{se } i > j. \end{cases}$$

28). (F. SANTANA - 83) Dadas a matriz A quadrada de ordem 2, tal que:

$$a_{ij} = \begin{cases} 1 + i, & \text{se } i = j; \\ 0, & \text{se } i \neq j. \end{cases}$$

e B também quadrada de ordem 2 , tal que bij = 2i − 3j, então A + B é igual a:

a) $\begin{bmatrix} -1 & 4 \\ -1 & -2 \end{bmatrix}$ b) $\begin{bmatrix} 1 & -4 \\ -1 & -2 \end{bmatrix}$ c) $\begin{bmatrix} -1 & 4 \\ 1 & 2 \end{bmatrix}$ d) $\begin{bmatrix} 1 & -4 \\ 1 & 2 \end{bmatrix}$ e) $\begin{bmatrix} 1 & 4 \\ 1 & 2 \end{bmatrix}$

29). Fazer um programa para sair com a matriz identidade de ordem N:

30). Dada a matriz A quadrada de ordem 2 tal que:

$$a_{ij} = \begin{cases} \operatorname{sen} \frac{\pi}{2} j, & \text{se } i = j; \\ \cos \pi j, & \text{se } i \neq j. \end{cases}$$

Então A^2 é a matriz:

a) $\begin{bmatrix} -1 & -1 \\ 1 & 0 \end{bmatrix}$ b) $\begin{bmatrix} 0 & -1 \\ 1 & 1 \end{bmatrix}$ c) $\begin{bmatrix} -1 & 1 \\ 0 & 1 \end{bmatrix}$ d) $\begin{bmatrix} 0 & 1 \\ -1 & 1 \end{bmatrix}$ e) $\begin{bmatrix} 0 & 1 \\ -1 & -1 \end{bmatrix}$

31) A = (aij) é uma matriz de ordem 2 × 2 com aij = 2^{-i} se i = j e aij = 0 se i ≠ j. A inversa de A é:

a) $\begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{4} \end{bmatrix}$ b) $\begin{bmatrix} -\frac{1}{2} & 0 \\ 0 & -\frac{1}{4} \end{bmatrix}$ c) $\begin{bmatrix} 2 & 0 \\ 0 & 4 \end{bmatrix}$ d) $\begin{bmatrix} -2 & 0 \\ 0 & -4 \end{bmatrix}$ e) $\begin{bmatrix} 2 & 0 \\ 0 & 2^{\frac{1}{2}} \end{bmatrix}$

32). (FCM STA CASA - 81) Seja a matriz A = (aij), de ordem 3, tal que:

$$a_{ij} = \begin{cases} 1 & \text{se } i < j; \\ k & \text{se } i = j \text{ e } k \in \mathbb{R}; \\ -1 & \text{se } i > j. \end{cases}$$

Se o determinante de A é igual a zero, então k pertence ao conjunto:

a) k ∈ {−3, 1, 3 } b) k ∈ {−2, 1, 2 } c) k ∈ { 0, 1/3, 1/2 } d) k ∈ {−√3,√3 } e) k ∈ {−1/3, 1/3}

33) (U. MACK - 80) Seja a função f : R → R definida por:

$$f(x) = \begin{cases} |x| + 3 & \text{se } |x| \leq 2; \\ |x - 3| & \text{se } |x| > 2 \end{cases}$$

O valor de f(f(f(. . . f(0) . . .))),

a) é 0 b) pode ser 1 c) é 3 d) pode ser 3 e) é impossível de calcular.

34). (PUC CAMP. - 80) Considerando N = { 0, 1, 2, 3, . . .} e, ainda:

$$A = \{ x \in \mathbb{N} / \frac{24}{x} = n, \ x \in \mathbb{N} \}$$

$$B = \{ x \in \mathbb{N} / \ 3x + 4 < 2x + 9 \}$$

Podemos afirmar que,

a) A ∪ B tem 8 elementos b) A ∪ B = A c) A ∩ B = A d) A ∩ B possui 4 elementos e) n.d.a.

35). (ITA - 66) Quantos números inteiros existem, de 1000 a 10000, não divisíveis nem por cinco nem por sete?

36). (UFRN - 84) O número de múltiplos de sete entre 50 e 150 é:

a) 9 b) 12 c) 14 d) 16 e) 23

37). (CESCEA - 75) Quantos números ímpares há entre 14 e 192?

a) 88 b) 89 c) 87 d) 86 e) 90

38). (FGV - 81) A soma dos números naturais não superiores a 1000, não divisíveis por 7, é:

a) 429429 b) 500500 c) 500500/7 d) 999999/7 e) n.d.a.

39). (CESGRANRIO - 84) A soma dos números naturais menores que 100 e que divididos por 5 deixam resto 2 é:

a) 996 b) 976 c) 990 d) 991 e) 998

7. Alguns Atalhos para Métodos Matemáticos

Problemas envolvendo métodos números são algo constante na vida do estudante de engenharia. Portanto alguns atalhos para métodos numéricos são mostrados neste capitulo de forma a que não se precise fazer todo um programa para que se obtenha uma resposta rápida para determinado problema, o que otimiza muito o trabalho/estudo.

7.1 Raízes de Polinômios

Função: PROOT

Argumento: vetor no nível 1 da pilha com os coeficientes do polinômio (escrito na forma geral) em ordem decrescente de expoente (do maior para menor).

Retorno: vetor no nível 1 da pilha com as raízes do polinômio.

Sintaxe: [coef. de x^n coef. de x^{n-1} ... coef. de x coef. de x^0] PROOT

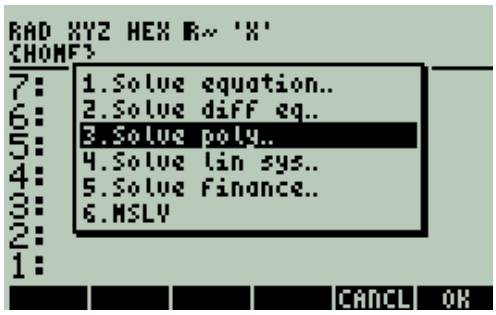
Exemplo: Retorna as raízes do polinômio $x^2 - 1 = 0$.

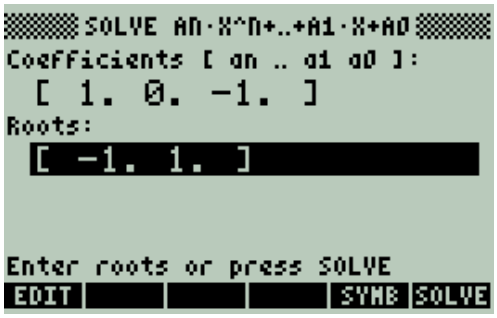
«

[1 0 -1] PROOT

»

Atalho na calculadora: *Seta para direita -> Botão 7 -> 3ª opção* (**Solve poly..**)





No primeiro campo se coloca um vetor com os coeficientes do polinômio na forma geral e no segundo através do **SOLVE** encontra-se as raízes

7.2 Resolução de Sistemas de Equações Lineares

Função: LINSOLVE

Argumento: vetor no nível 2 da pilha com as equações a serem resolvidas outro vetor no nível 1 da pilha com as variáveis.

Retorno: no nível 3 uma lista com os argumentos de entrada numa lista, no nível 2 uma lista de pontos pivô, no nível 1 a solução.

Sintaxe: [‘equação1’ ‘equação2’ ... ‘equaçãoN’] [‘x1’ ‘x2’ ... ‘xN’] **LINSOLVE**

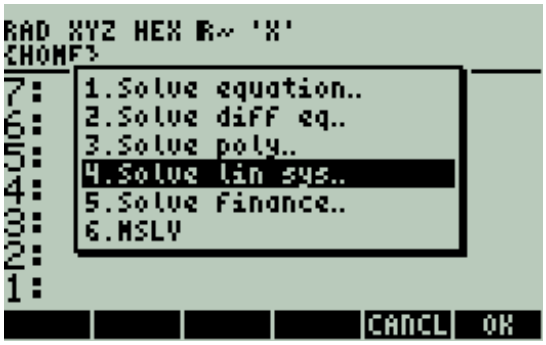
Exemplo: Resolve o sistema x+2y = 1 , x+y = 0.

«

['x+2*y-1=0' 'x+y=0'] ['x' 'y'] LINSOLVE

»

Atalho na calculadora: *Seta para direita -> Botão 7 -> 4ª opção* (**Solve linear system**)



No primeiro campo põe-se a matriz dos coeficientes no segundo a matriz dos termos independentes e no terceiro acha-se a solução do sistema apertando-se **SOLVE**.

7.3 Solução de Equação Não – Linear ou Transcendental.

Função: ROOT.

Argumento: equação que se quer resolver no nível 3, no nível 2 a variável e no nível 1 a aproximação inicial.

Retorno: a solução no nível 1.

Sintaxe: 'equação' 'variável' aproximação_inicial **ROOT**

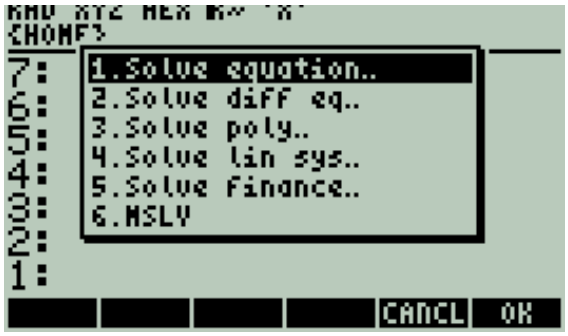
Exemplo:

«

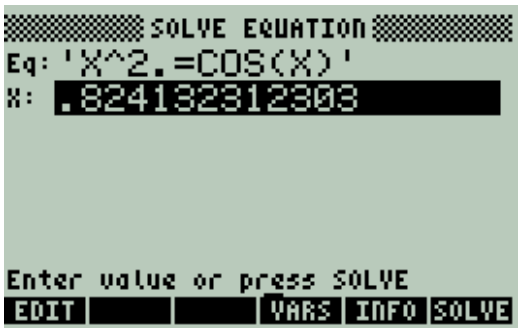
'EXP(x)-COS(x^2)=1' 'x' 1 ROOT

»

Atalho na calculadora*: Seta para direita -> Botão 7 -> 1ª opção* (**Solve equation**)



Digita-se a equação no primeiro campo entre aspas simples e logo então no campo da variável aperta-se **SOLVE.**



7.4 Resolução de Sistemas de Equações Não – Lineares ou Transcendentais

Função: MSLV

Argumento: vetor no nível 3 da pilha contendo as equações a serem resolvidas vetor no nível 2 da pilha contendo as variáveis e vetor no nível 1 da pilha contendo as aproximações iniciais.

Retorno: vetor no nível 1 com o conjunto solução.

Sintaxe: ['1ª equação' ... 'N-ésima equação'] ['1ª variável' ... 'N-ésima variável'] [aprox. inicial 1ª variável ... aprox. inicial da N-ésima variável] MSLV

Exemplo:.

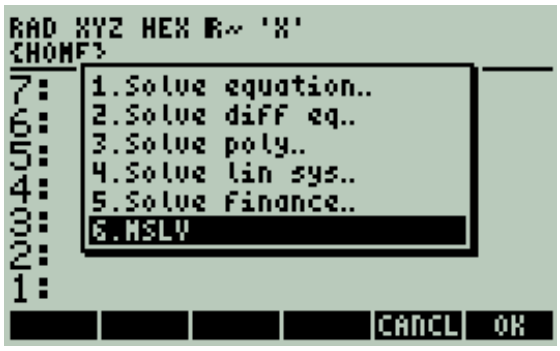
«

['(EXP(m)-569*n)=0' 'SIN(m)+COS(n)=0']

['m' 'n'] [1 1] MSLV

»

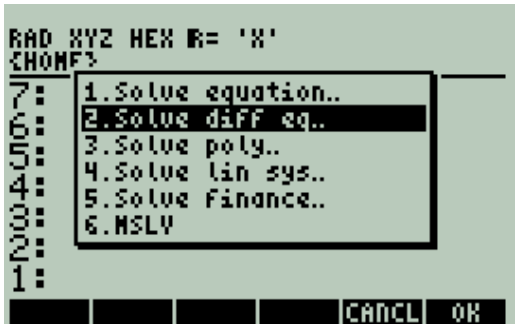
Atalho na calculadora*: Seta para direita -> Botão 7 -> 6ª opção* (**MSLV**)



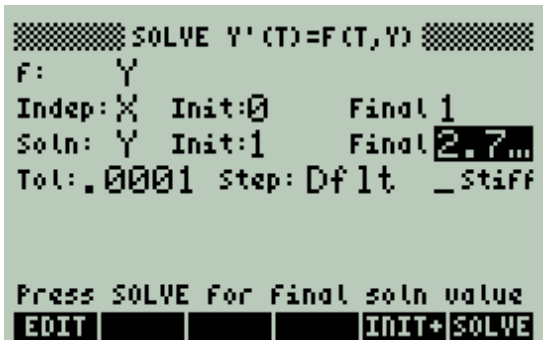
O **MSLV** é um comando da linguagem, para usá-lo na pilha faça o descrito acima.

7.5 Solução de Equações Diferenciais Ordinárias de Primeira Ordem (da forma y'(t)=f(t,y))

As funções da linguagem para resolução numérica de equações diferenciais todas usam a condição inicial x = 0 para y =0. Logo se você estiver pensando em realizar algum programa que precise resolver uma equação diferencial aconselho que você implemente algum método numérico, por exemplo, Diferenças Finitas, e coloque-o como sub-rotina do programa principal. Porém há um atalho fácil na calculadora para resolução de equações diferenciais ordinárias de primeira ordem. Para acessá-lo faça: *Seta para direita -> Botão 7 -> 2ª opção*



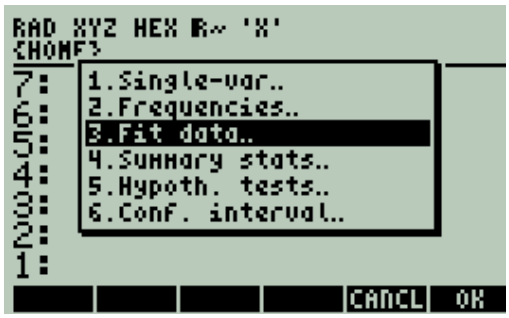
Após apertar-se **OK** vai-se para.



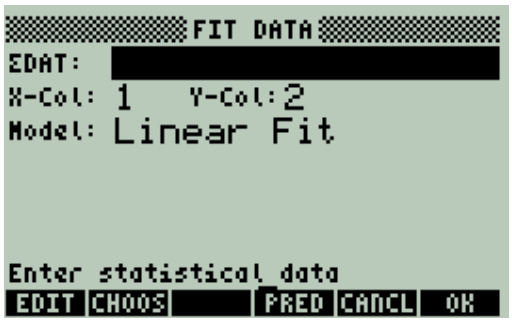
Onde se poderá digitar a função do lado direito da eq. Diferencial, as condições iniciais e o valor da variável independente e então calcular o valor da variável dependente da função integrada nesta situação.

7.6 Regressão

Digite*: Seta para direita -> Botão 5 -> 3ª opção* (**Fit data**)



Após apertar-se **OK**



Em **ΣDAT** digite na primeira coluna os valores da variável independente, na segunda os valores da variável dependente. Em **Model** escolha o tipo de regressão: Linear, Logarítmica, Exponencial, Potência de X ou a que apresente o melhor coeficiente de correlação r (em **Best fit**)' mais próximo de 1 ou de -1.

7.7 interpolação Polinomial

A regressão pode ser usada para interpolação. Porém há função específica para interpolação polinomial de um conjunto de pontos. A variável independente é a variável default.

Função: LAGRANGE

Argumento: matriz no nível 1 da pilha. Onde a primeira linha representa os valores de x e a segunda linha representa os respectivos valores de y.

Retorno: Função de interpolação.

Sintaxe: [[valores da variável independente] [respectivos valores da variável dependente]] **LAGRANGE**

Exemplo:

«

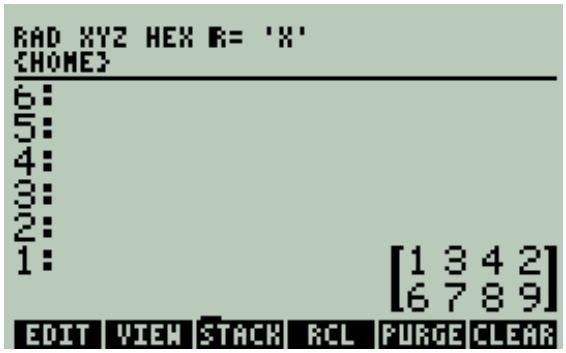
[[1 3 4 2]

[6 7 8 9]]

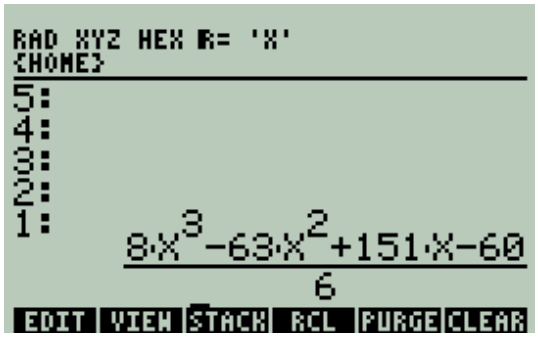
LAGRANGE

»

Atalho na calculadora: *Seta para esquerda -> Botão 1 -> 12ª opção* (**LAGRANGE**)



Então:



7.8 Exercício.

1). Para que fiquemos com um bom arsenal de ferramentas para métodos numéricos vamos programar a regra dos trapézios de integração numérica.

O algoritmo é:

- Entre com valores de x e y;
- Programe a fórmula da área de um trapézio para todos os pontos dois a dois $(Base\ maior + Base\ menor) \cdot \frac{altura}{2}$;
- Some as áreas.
- Salve este programa na sua calculadora.

8. Anexos 1

Aqui são mostradas funções para manipulação de strings, listas e matrizes.

8.1 Strings

As funções para modificação dos caracteres das Strings são:

SUB

5 11 SUB

Este comando devolve uma parte da string, temos que dar a posição: valores que são inicial e final.

REPL

3 “123” REPL

Substitui uma string de caracteres em outra mediante o valor da posição.

POS

“D” POS

Este comando indica a posição do caractere que se encontra na string, neste caso a letra D esta na posição 5.

SIZE

Este comando dá-nos a quantidade de caracteres que tem uma string.

NUM

Este comando devolve o numero do primeiro caractere de uma string, no exemplo acima o primeiro caractere é um E o numero do caractere E é 69.

CHR

Este comando é o inverso do anterior a partir do numero do caractere dá-nos o caractere.

HEAD

Este comando devolve o primeiro caractere de uma string.

TAIL

Este comando retira-nos o primeiro caractere de uma string.

8.2 Listas

As funções para manipulação de lista são:

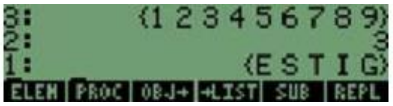
SUB

2 5 SUB

Este comando requer um valor inicial e outro final para selecionar os elementos que se desejam obter de uma lista.

REPL

3 ESTIG REPL



Este comando permite substituir elementos dentro de uma lista, este requer o valor da posição e o novo ou novos elementos.

GET

2 GET

Este comando permite obter um determinado elemento um a lista, este requer a posição.

GETI

2 GETI

Este comando é semelhante ao anterior, mas este permite obter a lista original, e o número da posição seguinte e o elemento, requer a posição do elemento.

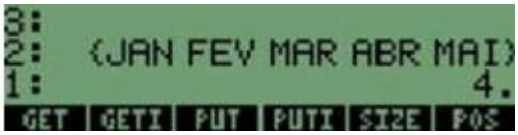
PUT

3 ABC PUT

Este comando trocar um elemento da lista por um novo, este precisa da posição e novo elemento.

PUTI

3 MAR PUTI



Este comando é similar ao anterior, a diferença é que este dá-nos a posição do elemento seguinte.

SIZE

Este elemento dá-nos o número de elementos que contem um a lista.

POS

E POS

Este comando indica a posição de um elemento na lista.

HEAD

Este comando devolve o primeiro elemento que está na lista

TAIL

Este comando devolve todos os elementos menos o primeiro.

DOLIST

Este comando executa um programa ou uma função a partir de listas.

Sintaxe:

1- Listas a utilizar

2- Quantidade de listas a utilizar

3- Programa ou função que se executara com os valores das listas.

4- DOLIST

(1)- Não se surpreenda os comandos seguintes vão ser explicados posteriormente

Exemplo:

« “Insira valores de A” { “{}” {1 2}V} IMPUT OBJ →

“Insira valores de B” { “{}” {1 2}V} IMPUT OBJ→

2 « 2 ^ SWAP 2 ^ + √ » DOLIST

»

DOSUBS

Este comando aplica um procedimento seqüencial com os elementos de uma lista.

Sintaxe:

1- Lista onde se encontram os valores a utilizar
2- Numero de elementos a utilizar em cada procedimento
3- Programa ou função que se efetua com esta sequência
4- DOSUBS

Exemplo:

Neste exemplo desejamos somar três números e elevá-los ao cubo.

« “INSIRA A LISTA” { “{}” {1 2}V} INPUT OBJ →

3 « + + 3 ^ »

DOSUBS »

Sintaxe 2 :

1- Lista onde se encontram os elementos
2- Programa ou função a efetuar-se em cada elemento
3- DOSUBS

Exemplo:

Neste exemplo vão-se efetuar as seguintes funções -> Ln(Abs(x*c os(x))) para n valores de X.

« “Insira valores” { “{}” {1 2}V} INPUT OBJ →

« → X « X X COS * ABS LN » »

DOSUBS »

NSUB

Este comando devolve a posição em que se esta efetuando o processo dentro do comando **DOSUBS**.

ENDSUB

Este comando devolve a quantidade de vezes que efetuou o programa por cada valor, e funciona dentro do comando **DOSUBS**.

STREAM

Este comando executa uma função com cada elemento que exista dentro de uma lista.

Sintaxe:

1- Lista onde se encontram os valores a utilizar por STREAM

2- Programa ou função a executar com os valores

3- STREAM

Exemplo:

Neste exemplo vão-se efetuar as seguintes funções -> Ln(Abs(x*cos(x))) para n valores de X.

« “Insira valores” { “{}” {1 2}V} INPUT OBJ →

« → X « X X COS * ABS LN » »

STREAM »

REVLIST

Este comando inverte a posição dos elementos de um a lista.

SORT

Este comando ordena os elementos de uma lista numericamente ou alfabeticamente.

SEQ

Este comando troca numa variável de uma função ou objeto valores dados um inicial e outro final, variando num incremento, devolve um a lista de valores obtidos.

Sintaxe:

1- Função ou objeto que se utiliza para trocar um a determinada variável.

2- Variável da função onde se deseja trocar os valores.

3- Valor inicia l a trocar na função.

4- Valor final a trocar na função.

5- Incremento dado para cada repetição.

6- SEQ

Exemplo:

Neste exemplo deseja-se substituir só os números pares de uma função, introduzindo a função valor inicial e valor final.

« “Insira a função” {“’” {1 2} ALG} INPUT OBJ→

“Insira a variável” { “” {1 1} ALG} IMPUT OBJ→

“Insira valor inicial e final” {“:Vi: :Vf:” {1 5}V} INPUT OBJ→ DTAG

SWAP DTAG DUPDUP 2 / IP 2 * = « 1 + » IFT SWAP 2 SEQ »

8.3 Matrizes

Para montar uma matriz por colunas a partir de uma série de vetores:

1. Introduza cada vetor na pilha, na mesma ordem que você deseja que apareçam na matriz.

2. Entre com o número de colunas para a matriz desejada.

3. Pressione 

para montar os vetores numa matriz.

Para criar uma matriz preenchida com uma constante dada:

1. Entre com um dos seguintes itens na pilha:
   - Uma lista contendo as dimensões da matriz desejada na forma: {linha coluna}.
   - Qualquer matriz, cujos elementos você não se importa que sejam alterados.

2. Introduza a constante com a qual você deseja preencher a matriz.

3. Pressione 

. Isto retorna uma matriz com as dimensões especificadas, preenchida com a constante fornecida.

Para criar uma matriz identidade:

1. Entre com um dos seguintes itens na pilha:
   - Um número real que representa o número de linhas e colunas de uma matriz identidade (quadrada).
   - Qualquer matriz quadrada, cujo conteúdo você não se importa que sejam alterados.

2. Pressione 

. Esta operação retorna uma matriz identidade com as dimensões dadas.

Para desmembrar uma matriz em vetores-linha:

1. Entre com a matriz na pilha.

2. Pressione [←] [MTH] MATRX ROW →ROW. A matriz é desmembrada em vetores (primeira linha até a última). O nível 1 da pilha contém um número real representando o número de linhas da matriz original.

Para desmembrar uma matriz em vetores-coluna:

1. Entre com a matriz na pilha.

2. Pressione [←] [MTH] MATRX COL →COL. A matriz é desmembrada em vetores (primeira coluna até a última). O nível 1 da pilha contém um número real representando o número de colunas da matriz original.

Inserindo linhas e colunas

Para inserir uma ou mais linhas novas numa matriz:

1. Entre com a matriz alvo (aquela que você deseja modificar) na pilha operacional.

2. Entre com o vetor, matriz ou elemento (quando o objeto alvo é um vetor) que você deseja inserir. Uma matriz inserida deve possuir o mesmo número de linhas e colunas que a matriz alvo.

3. Entre com o número da linha sobre a qual você deseja inserir a nova matriz. Todos os elementos localizados naquela linha da matriz alvo serão movidos para baixo, acomodando a inserção. Números de linhas começam de 1 e não de 0.

4. Pressione [←] [MTH] MATRX ROW ROW+ para inserir as novas linhas.

Para inserir uma ou mais colunas novas numa matriz:

1. Entre com a matriz alvo (aquela que você deseja modificar) na pilha operacional.

2. Entre com o vetor, matriz ou elemento (quando o objeto alvo é um vetor) que você deseja inserir. Uma matriz inserida deve possuir o mesmo número de linhas e colunas que a matriz alvo.

3. Entre com o número da coluna sobre a qual você deseja inserir a nova matriz. Todos os elementos localizados naquela coluna da matriz alvo serão movidos para a direita, acomodando a inserção. Números de colunas começam de 1 e não de 0.

4. Pressione [←] [MTH] MATRX COL COL+ para inserir as novas colunas.

Extraindo Linhas e Colunas

Para extrair uma linha específica de uma matriz:

1. Entre com a matriz na pilha operacional.

2. Entre com o número da linha (ou a posição do elemento, caso a matriz alvo seja um vetor) que você deseja extrair.

3. Pressione  . O vetor-linha (ou elemento) extraído é colocado no nível 1 e a matriz com a linha removida é colocada no nível 2.

Para extrair uma coluna específica de uma matriz:

1. Entre com a matriz na pilha operacional.

2. Entre com o número da coluna (ou a posição do elemento, caso a matriz alvo seja um vetor) que você deseja extrair.

4. Pressione  . O vetor-coluna (ou elemento) extraído é colocado no nível 1 e a matriz com a coluna removida é colocada no nível 2.

Invertendo Linhas e Colunas

Para inverter a posição de duas linhas numa matriz:

1. Entre com a matriz na pilha. Se a matriz for um vetor, ele será considerado um vetor-coluna.

2. Entre com os dois números das linhas que serão trocadas.

3. Pressione  . A matriz modificada será deixada no nível 1.

Para inverter a posição de duas colunas numa matriz:

1. Entre com a matriz na pilha. Se a matriz for um vetor, ele será considerado um vetor-linha.

2. Entre com os dois números das colunas que serão trocadas.

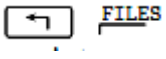
3. Pressione  . A matriz modificada será deixada no nível 1

9. Anexos 2

9.1 Salvar Programa Feitos no HPUserEdit 5 no computador.

- Escreva todo o programa no editor
- Clique em **Emulador -> Enviar ao emulador** (Certifique-se de que este programa irá para o nível 1 da pilha)
- No emulador clique em **Edit -> Save Object**
- Escolha um local no seu computador para guardá-lo, um nome para o programa. Este programa já pode ser enviado para um calculadora.

9.2 Transferências de Arquivos entre Calculadoras

Se o objeto (um programa, por exemplo) que você deseja transferir não se encontra no diretório corrente então você deve selecionar o diretório onde este objeto encontra-se; ou seja, pressione  para chegar à árvore de diretórios; isto é, a um visor similar ao seguinte:

Em seguida com a tecla ⊙ ou ⊙ selecione o diretório no qual o objeto, a ser transferido, encontra-se. Após selecionar o diretório pressione **CHDIR** para que este seja o diretório corrente. Talvez você queira também selecionar, na calculadora que irá receber os dados, o diretório que irá recebê-los. Então:

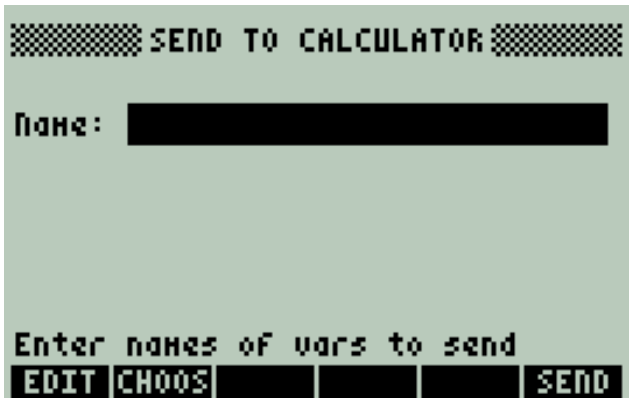
1. Alinhe as portas infravermelhas através das marcas △. As calculadoras não devem se distanciar mais de 5 cm.

Na figura acima a calculadora da esquerda irá transmitir e a da direita irá receber os dados.

A máquina que enviará a informação.

Pressione a seqüência de teclas: 

e então aperte **OK** em **Send to Calculator ..** e então você chegará na seguinte tela :

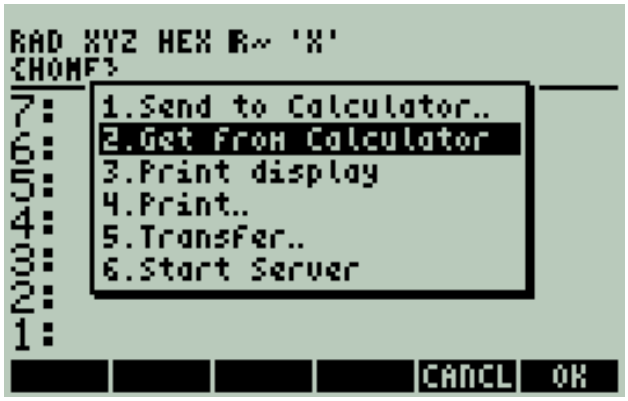


Pressionando **CHOOSE** a calculadora entrará no diretório corrente para que você possa selecionar o objeto a ser transmitido. Faça isto e pressione OK, a calculadora retornará com o visor acima já com o objeto pronto para ser enviado, o que pode ser feito pressionando-se **SEND** , mas antes você deve preparar a calculadora receptora.

A máquina que receberá a informação

Pressione a seqüência de teclas: 

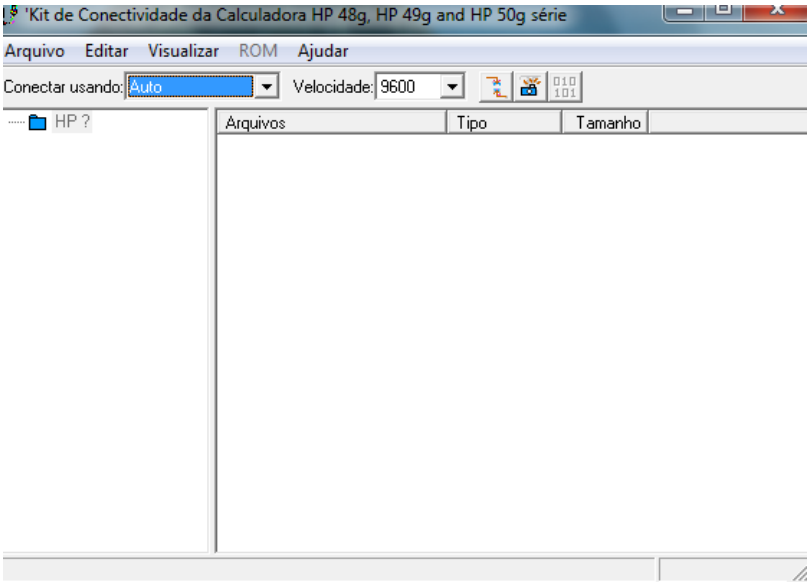
. Então aperte **OK** em **Get from Calculator.**



O ideal é que as duas calculadoras estejam mais ou menos sincronizadas.

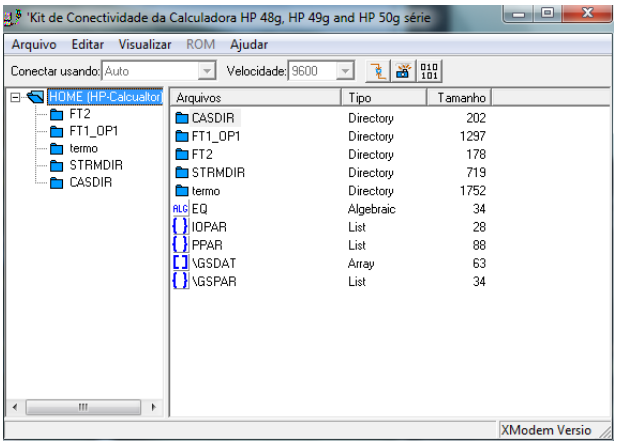
9.3 Transferência de Arquivos entre Calculadora e PC

Primeiramente você irá precisar instalar o programa **PC Connectivity Kit de Hewlett Packard (Conn4x_Portuguese)** em seu computador. Após feito isso abra o programa. Abrirá a seguinte tela:



Conecte sua calculadora via cabo USB ao computador. Após se certificar de que a calculadora está devidamente conectada ao computador clique em  . A tela da calculadora ficará da seguinte forma:

Logo então no programa clique em  . O computador e a calculadora se conectarão, então:



Apenas copie seus programas, bibliotecas no diretório desejado e então clique em  para desconectar os dispositivos. Uma caixa de seleção abrirá. Clique em cancelar.

9.4 Instalação e Desinstalação de Bibliotecas

As bibliotecas do UserRPL são basicamente um agregado de programas reunidos num diretório e instalados em alguma porta (0, 1 ou 2). Para acessar qualquer biblioteca instalada na calculadora a partir da pilha aperte *seta para direita -> Botão 2*. Escolha então a porta que contém a biblioteca desejada.

Para instalar a biblioteca na calculadora faça:

- Transfira a biblioteca para a calculadora e então a coloque no nível 1 da pilha.
- Digite o número da porta onde se quer instalar a biblioteca e coloque-o então no nível 1 da pilha (a biblioteca irá para o nível 2).
- Clique em **STO.**
- Resete a calculadora apertando **ON** e **F3** simultaneamente.

Para desinstalar uma biblioteca necessita-se de duas informações: o número da porta onde ela está instalada e seu número (acessado em *seta para direita -> Botão 2*). Para desinstalar uma biblioteca ponha o seguinte código no nível 1 da pilha:

:número da porta: número da biblioteca **PURGE**

Os dois pontos são importantes. Logo então clique em **ENTER.**

10. Referências Bibliográficas

Programando a HP 50g – Uma Introdução ao Fascinante Universo da Programação : http://www.pictronics.com.br/downloads/apostilas/programando-hp.pdf

Cálculo científico y técnico con HP49g/49g+/48gII/50g: http://www.epsem.upc.edu/~fpq/ale-hp/modulos/avanzado/user-rpl.pdf

Hp49G – Curso de Programação: http://pt.scribd.com/doc/20657504/programing-un-user-rpl-HP-50-G

Advanced User's Reference Manual, command reference and RPL guide – From *HP:* http://h10032.www1.hp.com/ctg/Manual/c00554621.pdf

HP 50g calculadora gráfica - guia do usuário http://www.dca.ufrn.br/~diogolr/hp50g/guia_usuario.pdf