Mathematics 152 — Spring 1999

These notes are brief. Experimenting with the calculator will fill them out.

Calculator data types

The calculator can deal with integers, real numbers, strings, vectors, and matrices. It cannot yet read real numbers in scientific notation, but it can read them as fractions in which there must be no spaces. Thus `-1/3` is a fraction but `-1 / 3` will be read differently. Vectors are arrays of real numbers in square brackets `[` and `]`, and matrices are arrays of vectors, which are interpreted as the rows of the matrix.

Calculator commands

These are grouped by the sort of thing they do.

1. **Basic arithmetic**

• **+** replaces the previous two items on the stack by their sum. Can add integers, real numbers, vectors, or matrices. Thus

```
7 6 +
```

calculates 7+6=13.

+ can also be used to build **strings**. A string is a phrase inside quotes. The sum of a string and any item tacks on a string representation of the item to the original string. Thus

```
"x = " 3 +
```

produces the string "x = 3". Using this feature is good for explaining in output exactly what displayed data means.

• **–** replaces the previous two items on the stack by their difference. Can subtract integers, real numbers, vectors, or matrices. Thus

```
7 6 -
```

calculates 7-6=1.

• * replaces the previous two items on the stack by their product. Can multiply integers or real numbers. Also calculates the dot product of two vectors; the scalar product of a vector and a scalar; the product of a matrix and a vector; and the product of two matrices. Thus

```
7 6 *
```

calculates 7*6=42.

• / replaces the previous two items on the stack by their quotient. Can divide integers or real numbers. Thus

```
14 2 /
```

calculates 14/2 = 7.

• **cross** replaces the previous two items by their cross product, if they are both three dimensional vectors.

2. **Mathematical functions**

• **exp** replaces the previous item x by e^x. Similarly for **cos**, **acos**, **sin**, **log** (which is the natural log).

• **atan2** has two arguments y and x **in that order**, and returns the angle coordinate of the point (x, y). (This odd and unfortunate choice of the order in which x and y are written conforms with that of most programming languages.)

• ˆ has two arguments x and y, and returns x^y. Here either $x > 0$ or y must be an integer.

• **pi** is a constant equal to $3.14159\ldots$

• **floor** replaces a number by the largest integer less than or equal to it. Thus 6.7 gets replaced by 6, while -6.7 gets replaced by -7.

• **sqrt** replaces the previous item by its square root, if it is a non-negative number.

3. **General**

• Integers inside parentheses `()` are indices into an array. Thus if `v` is a vector then `v(0)` is the 0-th element of `v`.

• **def** defines the previous item to be the item below it. The previous item must be a **variable name** such as `@x` or `@longVariableName`. A variable name is what you get by putting `@` before the variable itself. Thus `x` is a variable and `@x` is its name. (We have to distinguish between the variable and its name because the results of putting them in a program are very different. When the calculator comes across the variable, it attempts to make a substitution. This is similar to the difference between a variable and a pointer to the variable in some programming languages.) Thus

```
5 @x0 def
```

defines the variable `x0` to be 5. Subsequent occurrences of `x0` (with some exceptions to be explained some other time) will be replaced by 5.

If v is a vector then `4 @v(i) def` defines $v(i)$ to be 5.

• **dim** replaces a vector by its dimension.

• **fix** requires a non-negative integer on the stack. It sets the number of decimal figures displayed, and does not leave anything on the stack. Thus

```
5 fix
4.0 =
```

displays `4.00000`.

• **sci** is similar to `fix`, and produces output in scientific notation.

4. **Stack manipulations**

• **dup** makes an extra copy of the item at the top of the stack.

• **pop** just removes the item at the top of the stack.

• **exch** swaps the top two items on the stack.

5. **Matrix operations**

• **m i j rowswap** swaps rows i and j of the matrix m, leaving m on the stack.

• **m i j c rowsub** replaces $m(i)$ by $m(i) - c * m(j)$, leaving m on the stack.

• **rowscale** replaces $m(i)$ by $c * m(i)$, leaving m on the stack.

6. **Conditionals**

• **lt**, **le**, **gt**, **ge**, **eq** are tests on the previous two items, which should be numbers. The names stand for **less than**, **less than or equals to**, etc. The effect is to place either a **true** or a **false** on the stack.

• **ifelse** uses the top three items on the stack, which should be **true/false** and two procedures. If **true**, it executes the first procedure, while if **false** it executes the second. Either procedure can be empty.

• **not**, **&**, **|** are **boolean** operators. They combine true/false data. The notation for the last two ('and' and 'or') is close to that of other programming languages. Thus

```
5 0 gt
7 0 gt
&
```

returns true.

7. **Loops**

• **repeat** can be used to perform loops. It requires an integer and a **procedure** immediately preceding it. A procedure is a sequence of instructions inside brackets `{` and `}`. Thus

```
1000
10 { 1 - = } repeat
```

will output

```
999
998
997
996
995
994
993
992
991
990
```

• **break** will break out of an enclosing loop. This should be used together with conditionals in order to halt a `repeat` loop. Thus the following program will print out only the numbers 10, 9, 8, 7, 6.

```
10 @x def
10 {
x 5 eq { break } { x = x 1 - @x def } ifelse
} repeat
```

• **stop** halts the calculator temporarily, allowing you to view the stack, step through, or rerun the program.

• Any error will be signalled by displaying an error message. You should never ignore one of these messages. It is possible that it is caused by a bug in the program, in which case you should make a bug report.