NumCpp

Version 1.0

Sandesh Kalantre, Anuj Shetty, Sanjana Sabu

User Manual

Contents

1 Installing the Program **2**
1.1 *nix system 2
1.2 Windows 2

2 Using the Program **2**
2.1 command_line arguments 2
2.2 define 2
2.3 Routines 3
2.4 Help 3
2.5 Read 4
2.6 Write 4
2.7 evaluate 5
2.8 Solve simultaneous equations 5
2.9 FFT 5
2.10 showrpn 5

3 Appendix **6**
3.1 List of Routines 6

1 Installing the Program

1.1 *nix system

Make sure the libraries GNU-MPFR[1] and GNU-GMP[2] are installed on the system.If not download the repective archives from the respective sites and install them as per the instructions given.

1. Extract the archive.
2. Open a terminal inside the archive and run `make` command.
3. The program will be compiled and a `NumCpp` file will be created.
4. Run the program from the terminal as `./NumCpp`.

1.2 Windows

1. Extract the archive
2. Navigate to the Windows folder inside the archive
3. Run the `.exe` file inside the folder.

2 Using the Program

2.1 command_line arguments

- `-p unsigned_long`
  Set the precision to the the given argumemt.Due to efficiency reasons the value of the parameter can range only from 10 to 100000.
- `-r unsigned_long`
  Set the print precision to the argument.This parameter decides the number of digits printed after the decimal point.

After running the program, a prompt will be displayed as >>>.

2.2 define

The keyword *define* will used to define a function or a variable or an ndArray.

Syntax:

```
define variable_name/function_name/array_name = expression;
```

[1]GNU MPFR `http://www.mpfr.org/`
[2]GNU GMP`http://www.gmplib.org`

Examples:

```
define myvariable = 42
define myfunc(x) = x*sin(x)
define myarray[4] = [1,2,3,4]
define myarray[3,3] = [[1,2,3],[4,5,6],[7,6,4]]
```

The syntax of mathematical expressions while defining functions is same as that used in mathematics.Predefined functions in the program as well as user defined functions can be used in the definition of a function.However defining a function recursively will result in an `Expression Error`.

Note: Every expression or a definition ends with a semicolon.Though it is okay to omit them and the expression will parse as well.

Rules for a variable/function name

- Variable or function name can consist of letters, numbers or an underscore.
- The name can not start with a number.
- The name can not be name of a keyword such as define, help,etc.

Representation of decimal number

- The numbers internally are stored as mpfr_*t* floating type numbers with default precison of 64 bytes.
- All numbers begin with a digit. For example, .123 is not an acceptable number.
- Scientic notation is allowed though only **capital E** can be used for it.Example: $1.23E4$ is valid but $1.23e4$ is not.

Scientific output For output in scientific notation surround the expression with keyword `sci`.

Note The variable _ and the array `_[]` is used to store the result of the last expression evaluated.

2.3 Routines

The various methods/routines implemented in the library can be implemented as

Syntax:

```
routine_name(parameters)
```

For an exhaustive list of routines available in the library, use `help(routines)`
For help on a particular routine use `help(routines_name)`

2.4 Help

Help for each routine can be found using the help command.

Syntax:

```
help(routine_name)
```

The list of functions stored in the program can be seen by using

Syntax:

```
help(functions)
```

For a complete list,see `functions.txt` file in the help folder in the archive. The list of constants stored in the program can be seen by using

Syntax:

```
help(constants)
```

For a complete list,see `constants.txt` file in the help folder in the archive.

2.5 Read

The `read` command is used to read data from files into arrays.The arrays themselves must have been defined beforehand.The data must be stored in the same format as arrays are declared.All spaces and newlines in the file being read will be ignored.

If the file does not exist,the program throws a File Error.

If the dimensions of array in file and the array data is being stored are not same,the progam will read the data though it will be incorectly formatted and it won't be possible to process it further.*Always check array dimension while reading from a file.*

Syntax:

```
read(array_name[],file_name)
```

2.6 Write

The `write` command is used to write data to files from arrays.The arrays themselves must have been defined beforehand.The data will be stored in the same format as arrays are declared.

Syntax:

```
write(array_name[],file_name)
```

The read function can take an optional argument for writing $2d$ arrays to csv files

Syntax:

```
write(array_name[],file_name,csv)
```

2.7 evaluate

The evaluate keyword is used to evaluate a single variable function on an entire ndArray.

Syntax:

```
evaluate(array[],function())
```

2.8 Solve simultaneous equations

Simulataneous equations can be solved using the `solve` command.

Syntax:

```
solve(A[],B[])
```

where $A[]$ is the matrix of coefficients and $B[]$ is the right hand side of the equations.
If the dimensions of the arrays do not match,then Dimension Error is thrown.

2.9 FFT

The fft command can be used to find the fast Fourier transform of an array of complex numbers.An array of complex number is represented as a $2D$ array with the second dimension as 2.The size of the input array must be a power of 2 otherwise Dimension Error will be thrown.

Syntax:

```
fft(A[])
```

For the inverse transform,use the optional argument inv.

Syntax:

```
fft(A[],inv)
```

2.10 showrpn

The showrpn command can be used to see the reversed Polish notation of the given expression.

Syntax:

```
showrpn(expression)
```

3 Appendix

3.1 List of Routines

- `differentiate(x_value,function_name()).`
- `partial.diff2d(x_value,y_value,axis,function_name()).`
- `partial.diff3d(x_value,y_value,z_value,axis,function_name()).`
- `integrate(left_limit, right_limit, function_name()).`
- `integrate.rm(left_limit, right_limit, function_name()).`
- `integrate.rm_n(left_limit, right_limit, num_divisions, function_name()).`
- `integrate.rt(left_limit, right_limit, function_name()).`
- `integrate.mc(left_limit, right_limit, function_name()).`
- `integrate2d.rect(bot_limit, top_limit, l_limit(), r_limit(), function_name()).`
- `integrate2d.type1(l_limit, r_limit, bot_limit(), top_limit(), function_name()).`
- `integrate2d.type2(bot_limit, top_limit, l_limit(), r_limit(), function_name()).`
- `integrate2d.line(left_limit, right_limit, x(), y(), function_name()).`
- `integrate3d.cub(l_limit, r_limit, bot_limit, top_limit, front_limit, back_limit, function_name()).`
- `integrate3d.line(left_limit, right_limit, x(), y(), z(), function_name())`
- `integrate3d.surf(l_limit,r_limit,bot_limit,top_right_limit,X(),Y(),Z(),f())`
- `solve.n(guess, function_name)`
- `solve.b(left_end, right_end, function_name())`