

elemapprox user manual

Title	elemapprox (Elementary functions approximation in ANSI C, Verilog HDL, and VHDL)
Author	Nikolaos Kavvadias (C) 2013, 2014, 2015, 2016, 2017
Contact	nikolaos.kavvadias@gmail.com
Website	http://www.nkavvadias.com
Release Date	03 July 2017
Version	1.4.1
Rev. history	
v1.4.1	2017-07-03 Add <code>cbrt</code> .
v1.4.0	2017-07-01 Add <code>verilogieee</code> (Verilog-2005) version. Add common directories for Verilog and C.
v1.3.3	2016-08-22 Add <code>fmin</code> , <code>fmax</code> for C, Verilog, and VHDL.
v1.3.2	2016-02-27 Fix ASCII rendering of function plots in the documentation.
v1.3.1	2015-12-25 Documentation updated with function plot examples.
v1.3.0	2014-10-12 Added <code>ansicstd</code> version which is an alternative C port that calls the standard C library mathematical functions as defined in <code>math.h</code> . Documentation updated.
v1.2.0	2014-10-10 Added <code>vhdlieee</code> version which is compatible to the IEEE <code>math_real</code> package. Documentation updated.
v1.1.0	2014-10-07 Added VHDL version for approximating and plotting the elementary functions. Numerous documentation fixes.
v1.0.1	2014-09-24 Minor documentation corrections.
v1.0.0	2014-09-16 Initial release. This is an expanded version built upon Mark G. Arnold's Verilog Transcendental Functions paper.

1. Introduction

`elemapprox` is an ANSI C code, Verilog HDL and VHDL collection of modules (Verilog HDL) and packages (VHDL) that provide the capability of evaluating and plotting transcendental functions by evaluating them in single precision. The original work supports ASCII plotting of a subset of the functions; this version provides a more complete list of functions in addition to bitmap plotting for the transcendental functions as PBM (monochrome) image files.

`elemapprox` has been conceived as an extension to Prof. Mark G. Arnold's work as published in HDLCON 2001. Most functions have been prefixed with the letter `k` in order to avoid function name clashes in both ANSI C and VHDL standard-based implementations. Currently, the plain VHDL version, `vhdl` uses unprefixed names (e.g., `acos` instead of `kacos`). The `vhdliieee` version is compatible to the `IEEE.math_real` package and features prefixed names.

1.1 Implementations

The following table summarizes the implemented versions in C, VHDL and Verilog HDL.

Implementation	Description
<code>ansic</code>	ANSI C version with custom approximations
<code>ansicstd</code>	ANSI C version using <code>math.h</code>
<code>verilog</code>	Verilog version with custom approximations
<code>verilogieeee</code>	Verilog 2005 version with the new arithmetic system functions
<code>vhdl</code>	VHDL version with custom approximations
<code>vhdliieee</code>	VHDL version using <code>IEEE.math_real</code>

1.2 Implemented functions

The transcendental functions supported include most elementary functions (hence the name `elemapprox`) and the list is as follows:

Function	Description
<code>kfabs(x)</code>	Floating-point absolute value (helper function).
<code>kfmax(x, y)</code>	Floating-point maximum value (helper function).
<code>kfmin(x, y)</code>	Floating-point minimum value (helper function).
<code>kfloor(x)</code>	Floating-point floor (helper function).
<code>kceil(x)</code>	Floating-point ceiling (helper function).
<code>kround(x)</code>	Floating-point round-to-nearest (helper function).
<code>ktrunc(x)</code>	Floating-point truncation (helper function).
<code>kfmod(x, y)</code>	Floating-point modulo.
<code>rootof2(x)</code>	Calculate root-of-2 (not in <code>vhdliieee</code> , <code>verilogieeee</code> and <code>ansicstd</code>).
<code>kacos(x)</code>	Arc cosine.

kacosh (x)	Inverse hyperbolic cosine.
kacot (x)	Arc cotangent.
kacoth (x)	Inverse hyperbolic cotangent.
kacsc (x)	Arc cosecant.
kacsch (x)	Inverse hyperbolic cosecant.
kasec (x)	Arc secant.
kasech (x)	Inverse hyperbolic secant.
kasin (x)	Arc sine.
kasinh (x)	Inverse hyperbolic sine.
katan (x)	Arc tangent.
katan2 (y, x)	Two-argument (x/y) arc tangent.
katanh (x)	Inverse hyperbolic tangent.
kcos (x)	Cosine.
kcosh (x)	Hyperbolic cosine.
kcot (x)	Cotangent.
kcoth (x)	Hyperbolic cotangent.
kcsc (x)	Cosecant.
kcsch (x)	Hyperbolic cosecant.
kexp (x)	Exponential.
khypot (x, y)	Hypotenuse.
klog (x)	Natural logarithm.
kpow (x, y)	Powering function.
ksec (x)	Secant (named secant in the plain VHDL port).
ksech (x)	Hyperbolic secant.
ksin (x)	Sine.
ksinh (x)	Hyperbolic sine.
ksqrt (x)	Square root.
kcbrt (x)	Cube root.
ktan (x)	Tangent.
ktanh (x)	Hyperbolic tangent.

The reference paper and the corresponding presentation are available from the web at the following links:

- https://web.archive.org/web/2010051500000*/http://www.cse.lehigh.edu/~caar/marnold/papers/sanjose_hdlcon.doc
- https://web.archive.org/web/2010041500000*/http://www.cse.lehigh.edu/~caar/marnold/presentations/freddy.ppt

Since the original links have been removed, the files are now only accessible from the Internet Archive.

2. File listing

The `elemapprox` C code implementation and Verilog HDL modules include the following files:

<code>/elemapprox</code>	Top-level directory
<code>AUTHORS</code>	List of authors.
<code>LICENSE</code>	License argeement (Modified BSD license).
<code>README.rst</code>	This file.
<code>README.html</code>	HTML version of README.
<code>README.pdf</code>	PDF version of README.
<code>rst2docs.sh</code>	Bash script for generating the HTML and PDF versions.
<code>VERSION</code>	Current version.
<code>/ansic</code>	ANSI C implementation (standalone)
<code>clean-math-ansic.sh</code>	Bash script for cleaning up the generated executables.
<code>elemapprox.c</code>	C code for the function approximations.
<code>Makefile</code>	GNU Makefile for building <code>testfunc.exe</code> .
<code>plot-ansic-ascii.sh</code>	Bash script for plotting the elementary functions as ASCII graphs using <code>testfunc.exe</code> .
<code>plot-ansic-pbm.sh</code>	Bash script for plotting the elementary functions as PBM images using <code>testfunc.exe</code> .
<code>test<func>.pbm</code>	Generated PBM image data for the function <func>.
<code>test<func>.txt</code>	Generated ASCII graph data for the function <func>.
<code>test<func>-ascii.txt</code>	Concatenation of the generated ASCII graph data for all supported functions.
<code>/ansicstd</code>	ANSI C implementation (based on <code>math.h</code>)
<code>clean-math-ansic.sh</code>	Bash script for cleaning up the generated executables.
<code>elemapprox.c</code>	C code for the function approximations.
<code>Makefile</code>	GNU Makefile for building <code>testfunc.exe</code> .
<code>plot-ansic-ascii.sh</code>	Bash script for plotting the elementary functions as ASCII graphs using <code>testfunc.exe</code> .
<code>plot-ansic-pbm.sh</code>	Bash script for plotting the elementary functions as PBM images using <code>testfunc.exe</code> .
<code>test<func>.pbm</code>	Generated PBM image data for the function <func>.
<code>test<func>.txt</code>	Generated ASCII graph data for the function <func>.
<code>test<func>-ascii.txt</code>	Concatenation of the generated ASCII graph data for all supported functions.
<code>/common</code>	Common files
<code>/common/ansic</code>	Common files for the C implementations
<code>elemapprox.h</code>	C header file for the above. Defines certain mathematical constants and declares function prototypes.
<code>funcplot.c</code>	Reference code for creating the plot data for the functions.
<code>funcplot.h</code>	C header file for the above.

<code>graph.c</code>	Collection of ASCII and PBM graphing functions.
<code>graph.h</code>	C header file for the above.
<code>/common/scripts</code>	Scripts for common use
<code>diff-ascii.sh</code>	Compare the results from ASCII plots between two different implementations (directories)
<code>pbm2png.sh</code>	Convert PBM to PNG images with ImageMagick's <code>convert</code> .
<code>/common/verilog</code>	Common files for the Verilog implementations
<code>funcplot.v</code>	Reference code for creating the plot data for the functions.
<code>graph.v</code>	Collection of ASCII and PBM graphing tasks.
<code>testfunc.v</code>	Application code for the elementary functions. Options include PBM or ASCII image generation and function selection.
<code>/common/vhdl</code>	Common files for the VHDL implementations
<code>graph.vhd</code>	VHDL package code with a collection of ASCII and PBM procedures.
<code>testfunc.vhd</code>	VHDL testbench code for the elementary functions. Options include PBM or ASCII image generation and function selection through a configuration file.
<code>/verilog</code>	Verilog HDL implementation
<code>clean-math-verilog.sh</code>	Bash script for cleaning up the generated interpreted intermediate code (for Icarus Verilog).
<code>constants.v</code>	Certain mathematical constants.
<code>elemapprox.v</code>	Verilog HDL code for the function approximations.
<code>plot-verilog-ascii.sh</code>	Bash script for plotting the elementary functions as ASCII graphs using <code>testfunc.v</code> . The script Icarus Verilog' VVP interpreter which is capable of parsing command-line options.
<code>plot-verilog-pbm.sh</code>	Bash script for plotting the elementary functions as PBM images using <code>testfunc.v</code> .
<code>test<func>.pbm</code>	Generated PBM image data for the function <code><func></code> .
<code>test<func>.txt</code>	Generated ASCII graph data for the function <code><func></code> .
<code>test<func>-ascii.txt</code>	Concatenation of the generated ASCII graph data for all supported functions.
<code>/verilogieee</code>	Verilog 2005 implementation
<code>clean-math-verilog.sh</code>	Bash script for cleaning up the generated interpreted intermediate code (for Icarus Verilog).
<code>elemapprox.v</code>	Verilog HDL code for the function approximations.
<code>plot-verilog-ascii.sh</code>	Bash script for plotting the elementary functions as ASCII graphs using <code>testfunc.v</code> . The script Icarus Verilog' VVP interpreter which is capable of parsing command-line options.

plot-verilog-pbm.sh	Bash script for plotting the elementary functions as PBM images using <code>testfunc.v</code> .
test<func>.pbm	Generated PBM image data for the function <func>.
test<func>.txt	Generated ASCII graph data for the function <func>.
test<func>-ascii.txt	Concatenation of the generated ASCII graph data for all supported functions.
/vhdl	VHDL implementation
elemapprox.do	Modelsim <code>.do</code> macro file for Modelsim simulation.
elemapprox.mk	GNU Makefile for running the testbench using GHDL.
elemapprox.vhd	VHDL package code for the function approximations and related mathematical constants.
funcplot.vhd	VHDL package code for creating the plot data for the elementary functions.
plot-ghdl-ascii.sh	Bash script for plotting the elementary functions as ASCII graphs using GHDL. The script generates a configuration file (<code>config.txt</code>) for controlling the simulation.
plot-ghdl-pbm.sh	Bash script for plotting the elementary functions as PBM images using GHDL. The script generates a configuration file (<code>config.txt</code>) for controlling the simulation.
plot-mti-ascii.sh	Bash script for plotting the elementary functions as ASCII graphs using Modelsim. The script generates a configuration file (<code>config.txt</code>) for controlling the simulation.
plot-mti-pbm.sh	Bash script for plotting the elementary functions as PBM images using Modelsim. The script generates a configuration file (<code>config.txt</code>) for controlling the simulation.
test<func>.pbm	Generated PBM image data for the function <func>.
test<func>.txt	Generated ASCII graph data for the function <func>.
test<func>-ascii.txt	Concatenation of the generated ASCII graph data for all supported functions.
/vhdlieee	VHDL implementation compatible to IEEE.math_real
elemapprox.do	Modelsim <code>.do</code> macro file for Modelsim simulation.
elemapprox.mk	GNU Makefile for running the testbench using GHDL.
elemapprox.vhd	VHDL package code for the function approximations and related mathematical constants.
funcplot.vhd	VHDL package code for creating the plot data for the elementary functions.

plot-ghdl-ascii.sh	Bash script for plotting the elementary functions as ASCII graphs using GHDL. The script generates a configuration file (<code>config.txt</code>) for controlling the simulation.
plot-ghdl-pbm.sh	Bash script for plotting the elementary functions as PBM images using GHDL. The script generates a configuration file (<code>config.txt</code>) for controlling the simulation.
plot-mti-ascii.sh	Bash script for plotting the elementary functions as ASCII graphs using Modelsim. The script generates a configuration file (<code>config.txt</code>) for controlling the simulation.
plot-mti-pbm.sh	Bash script for plotting the elementary functions as PBM images using Modelsim. The script generates a configuration file (<code>config.txt</code>) for controlling the simulation.
test<func>.pbm	Generated PBM image data for the function <func>.
test<func>.txt	Generated ASCII graph data for the function <func>.
test<func>-ascii.txt	Concatenation of the generated ASCII graph data for all supported functions.
/refs	Reference documentation
sanjose_hdlcon.doc	MS Word document for the manuscript: M. G. Arnold, C. Walter and F. Engineer, "Verilog Transcendental Functions for Numerical Testbenches," Proceedings of the Tenth International HDL conference, Santa Clara, California, March 1, 2001.
freddy.ppt	MS PowerPoint presentation of the above work.

3. Usage

Both the ANSI C and Verilog HDL versions can be used for generating graph data and depicting any of the supported transcendental functions via two similar scripts.

3.1 ANSI C

1. Run the following shell script from a Unix/Linux/Cygwin command line in order to generate an ASCII graph for each function.

```
$ cd ansic
```

or

```
$ cd ansicstd
```

followed by

```
$ ./plot-ansic-ascii.sh
```

All generated data are also concatenated to `testfunc-ascii.txt`.

2. Run the following shell script from a Unix/Linux/Cygwin command line in order to generate a PBM image for each function.

```
$ ./plot-ansic-pbm.sh
```

All generated data are saved in the form of PBM (monochrome bitmap) image files. Such files can be visualized using, e.g., the public domain `Imagine` viewer: <http://www.nyam.pe.kr/> on Windows or `eog` (Eye of Gnome) on Debian-based Linux distributions (for instance, Ubuntu).

3.2 Verilog HDL

1. Run the following shell script from a Unix/Linux/Cygwin command line in order to generate an ASCII graph for each function.

```
$ cd verilog
```

or

```
$ cd verilogeee
```

followed by

```
$ ./plot-verilog-ascii.sh
```

All generated data are also concatenated to `testfunc-ascii.txt`.

2. Run the following shell script from a Unix/Linux/Cygwin command line in order to generate a PBM image for each function.

```
$ ./plot-verilog-pbm.sh
```

All generated data are saved in the form of PBM (monochrome bitmap) image files.

3.3 VHDL

The VHDL version of `elemapprox` supports both GHDL (<http://ghdl.free.fr>) and Mentor Modelsim (<http://www.model.com>).

3.3.1 GHDL

1. Run the following shell script from a Unix/Linux/Cygwin command line in order to generate an ASCII graph for each function.

```
$ cd vhdl
```

or

```
$ cd vhdlieee
```

followed by

```
$ ./plot-ghdl-ascii.sh
```

All generated data are also concatenated to `testfunc-ascii.txt`.

2. Run the following shell script from a Unix/Linux/Cygwin command line in order to generate a PBM image for each function.

```
$ ./plot-ghdl-pbm.sh
```

All generated data are saved in the form of PBM (monochrome bitmap) image files.

3.3.2 Modelsim

1. Run the following shell script from a Unix/Linux/Cygwin command line in order to generate an ASCII graph for each function.

```
$ cd vhdl
```

or

```
$ cd vhdlieee
```

followed by

```
$ ./plot-mti-ascii.sh
```

All generated data are also concatenated to `testfunc-ascii.txt`.

2. Run the following shell script from a Unix/Linux/Cygwin command line in order to generate a PBM image for each function.

```
$ ./plot-mti-pbm.sh
```

All generated data are saved in the form of PBM (monochrome bitmap) image files.

4. Synthesis

The implementation code (either ANSI C, Verilog HDL or VHDL) for the transcendental functions has not been tested for high-level or RTL synthesis.

5. Prerequisites

- Standard UNIX-based tools (tested with gcc-4.6.2 on MinGW/x86) [optional if you use Modelsim].
 - make

- bash (shell)

For this reason, MinGW (<http://www.mingw.org>) or Cygwin (<http://sources.redhat.com/cygwin>) are suggested, since POSIX emulation environments of sufficient completeness.

- Icarus Verilog simulator (<http://iverilog.icarus.com/>). The Windows version can be downloaded from: <http://bleyer.org/icarus/>
 - GHDL simulator (<http://ghdl.free.fr>) for VHDL. Both Windows and Linux versions can be downloaded from this site. Updated GHDL releases are available (again for multiple OSes) from: <http://sourceforge.net/projects/ghdl-updates/>
 - Alternatively, a commercial simulator like Mentor Modelsim (<http://www.model.com>) can be used (however this has only been tested for the VHDL version of elemapprox).

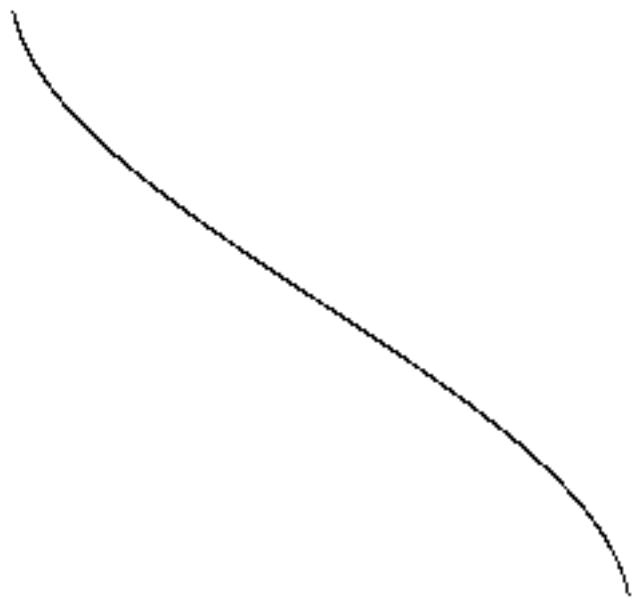
A. Function plot examples

This appendix provides sample function plot visualization as raster graphics (PNG) and ASCII for the elementary functions with provided implementations.

A.1. Arc cosine (acos)

ASCII rendering.

Bitmap rendering.

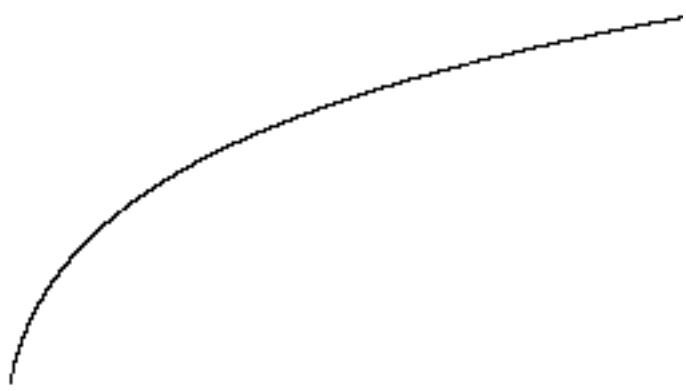


A.2. Inverse hyperbolic cosine (acosh)

ASCII rendering.

A sheet of dot-grid paper featuring a uniform grid of small, light-gray dots. A series of black asterisks (*) are placed along a diagonal line that slopes upwards from the bottom-left corner towards the top-right corner. The asterisks are evenly spaced and form a clear path across the grid.

Bitmap rendering.

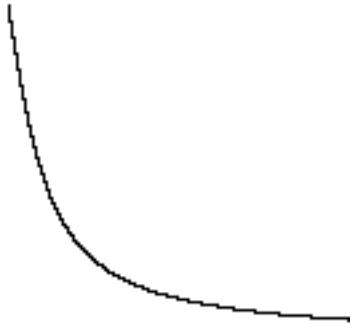


A.3. Arc cotangent (acot)

ASCII rendering.

This image shows a sheet of dot-grid paper. The background consists of a grid of small, evenly spaced dots. A decorative border is formed by a series of five-pointed stars of increasing size, starting from the bottom left and ending at the top right. The stars are composed of black lines and dots. The overall pattern is symmetrical and centered.

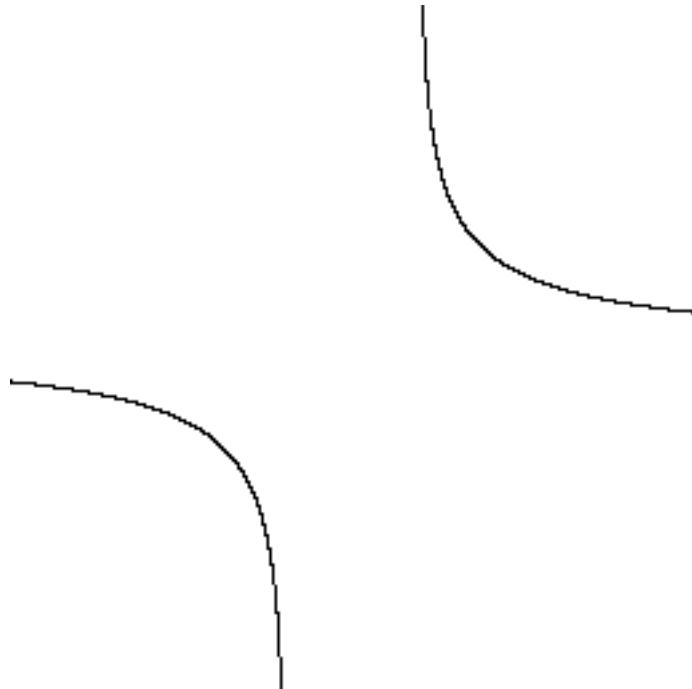
Bitmap rendering.



A.4. Inverse hyperbolic cotangent (acoth)

ASCII rendering.

Bitmap rendering.

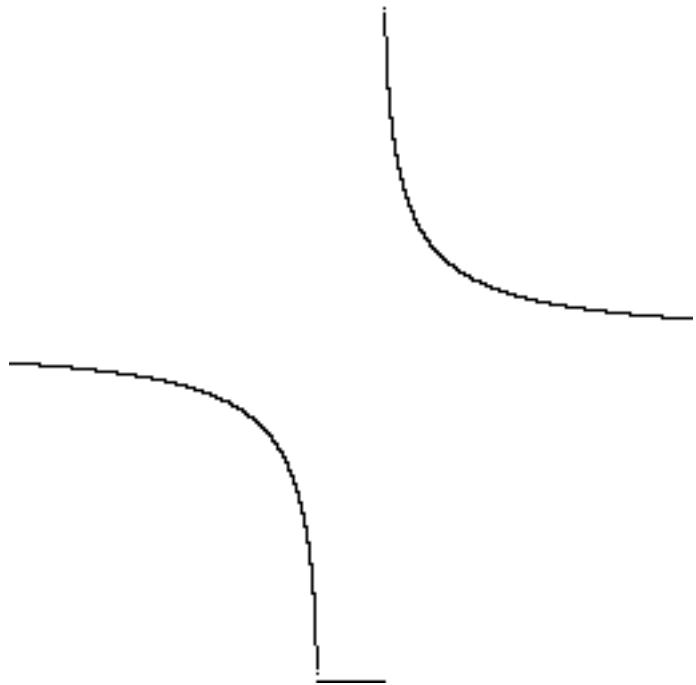


A.5. Arc cosecant (acsc)

ASCII rendering.

```
.....*.....  
.....*.....  
.....*.....  
.....*.....  
.....*.....  
.....**.....  
.....**.....  
.....***.....  
.....***.....  
.....****.....  
.....*****.....  
.....*****.....  
*****.....*****.....  
*****.....*****.....  
.....*****.....  
.....***.....  
.....**.....  
.....*.....  
.....*.....  
.....*.....  
.....*****.....
```

Bitmap rendering.

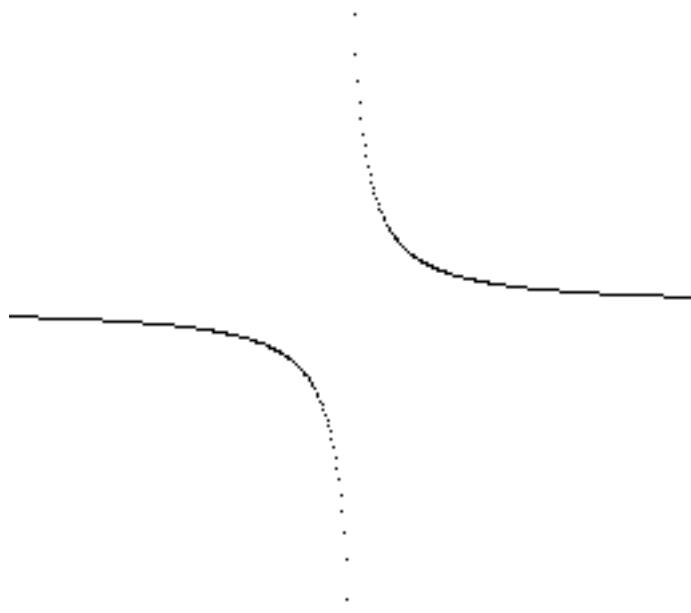


A.6. Inverse hyperbolic cosecant (acsch)

ASCII rendering.

```
.....*.....  
.....*.....  
.....*.....  
.....*.....  
.....*.....  
.....**.....  
.....**.....  
.....**.....  
.....***.....  
.....***.....  
.....*****.....  
.....*****.....  
*****.....*****.....  
*****.....*****.....  
.....*****.....  
.....****.....  
.....***.....  
.....*.....  
.....*.....  
.....*.....  
.....*.....  
.....*.....  
.....*.....
```

Bitmap rendering.

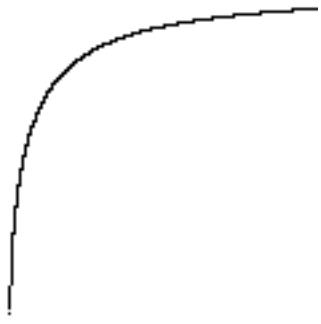


A.7. Arc secant (asec)

ASCII rendering.

The image shows a sheet of dot-grid paper. It features horizontal dotted lines spaced evenly down the page. Vertical grid lines intersect these dotted lines, creating a grid. In the upper right quadrant, there is a specific arrangement of characters: a row of hash symbols (#) at the top, followed by a row of asterisks (*), then a row of double asterisks (**), and finally a row of triple asterisks (***) below that. This pattern descends diagonally towards the bottom left of the page.

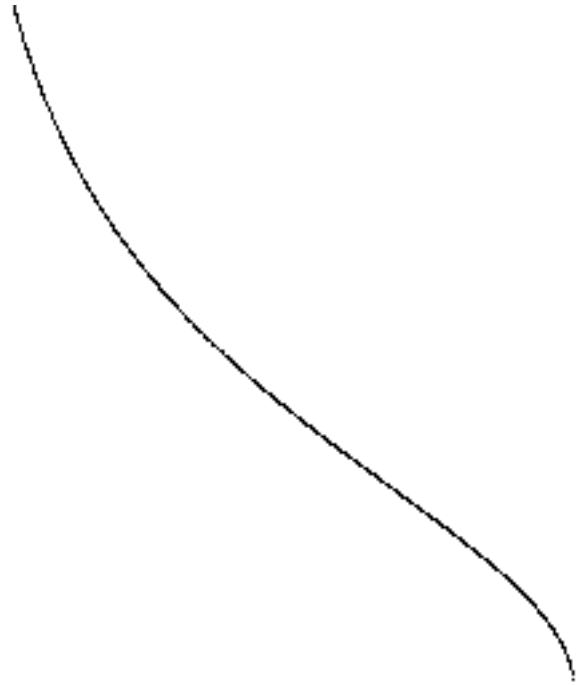
Bitmap rendering.



A.8. Inverse hyperbolic secant (asech)

ASCII rendering.

Bitmap rendering.

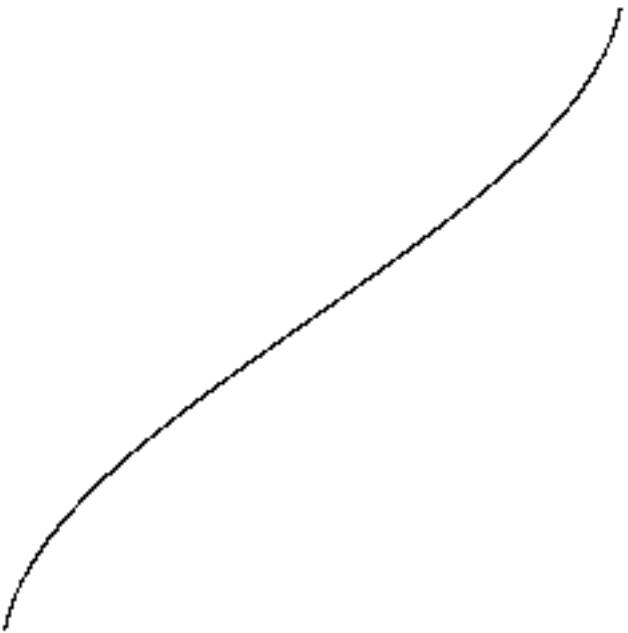


A.9. Arc sine (asin)

ASCII rendering.

A grid pattern where each cell contains a single black asterisk (*). The grid consists of approximately 20 columns and 15 rows of these asterisks.

Bitmap rendering.

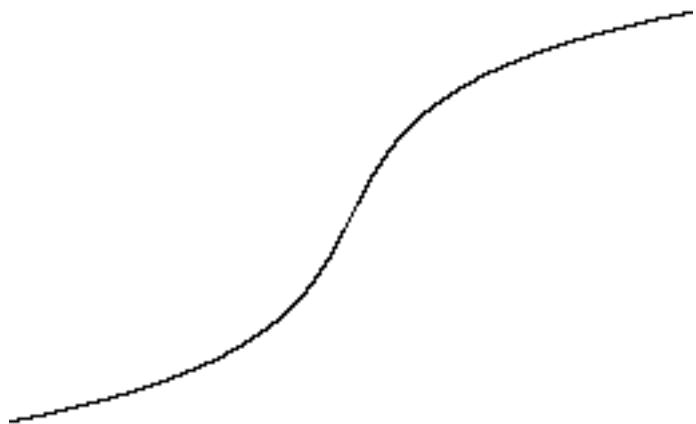


A.10. Inverse hyperbolic sine (asinh)

ASCII rendering.

The image shows a decorative border consisting of a repeating pattern of small black five-pointed stars arranged in a grid-like fashion. The stars are centered at the intersections of horizontal and vertical lines, creating a dense, dotted effect. The border is complete, with stars forming a continuous line around the entire perimeter.

Bitmap rendering.

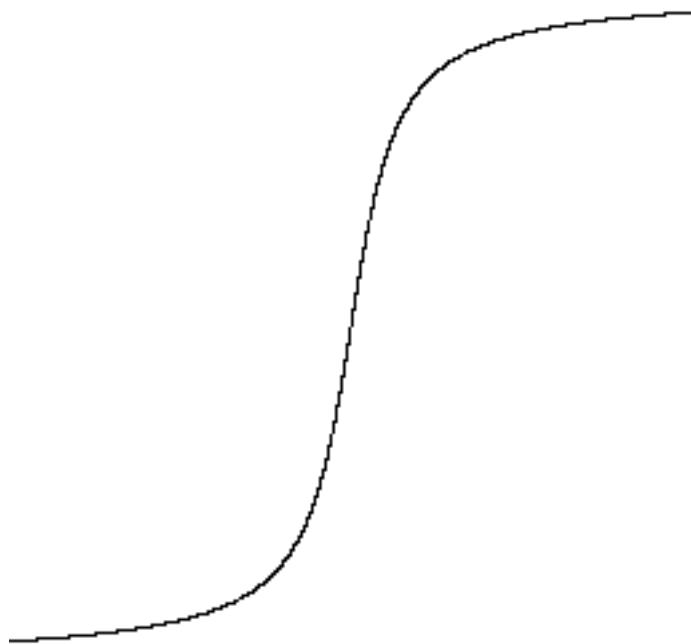


A.11. Arc tangent (atan)

ASCII rendering.

The image shows a decorative border composed of a repeating pattern of stars and dots. The pattern consists of a central four-pointed star surrounded by a circle of smaller stars, all set against a background of small dots. This pattern is repeated in a staggered fashion along the edges of the frame.

Bitmap rendering.

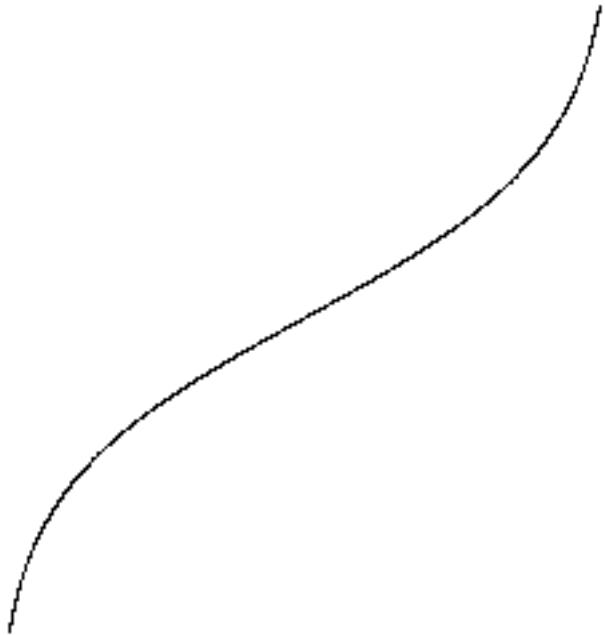


A.12. Inverse hyperbolic tangent (atanh)

ASCII rendering.

The image shows a collection of black asterisks (*) arranged in a grid-like pattern. The asterisks are placed at regular intervals, creating a visual effect where they form nested diamond shapes. The pattern is composed of several concentric layers of these diamonds, radiating from the center of the image. The asterisks are black and have a small white dot in the middle.

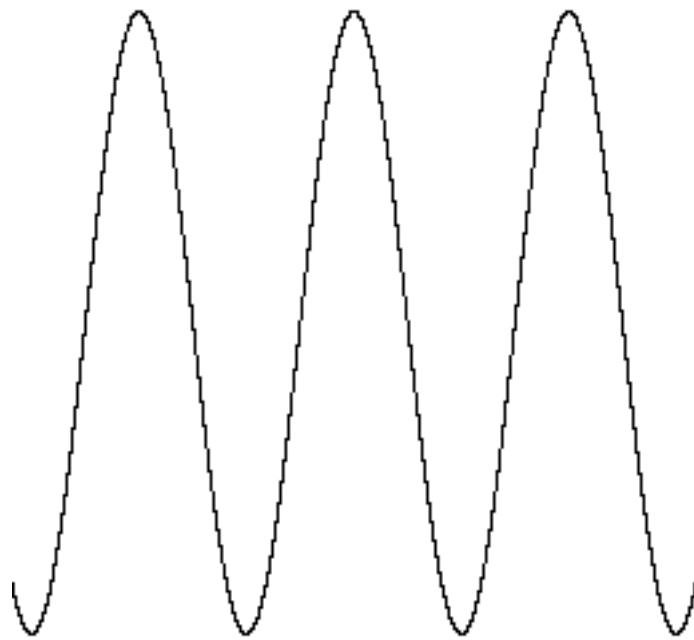
Bitmap rendering.



A.13. Cosine (cos)

ASCII rendering.

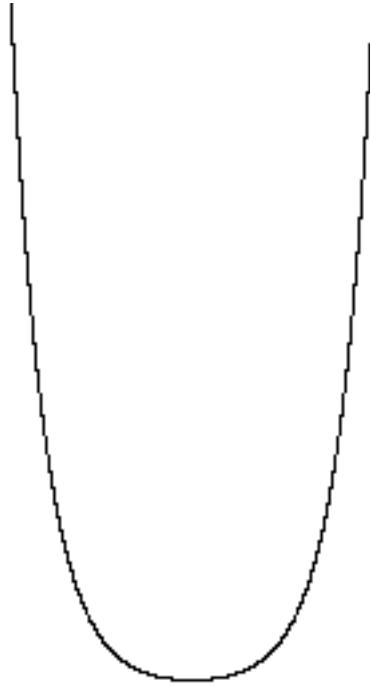
Bitmap rendering.



A.14. Hyperbolic cosine (\cosh)

ASCII rendering.

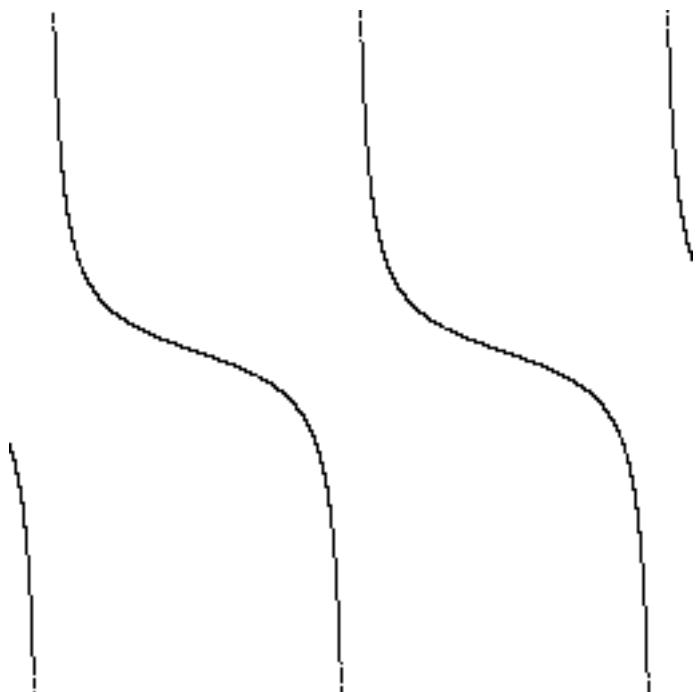
Bitmap rendering.



A.15. Cotangent (\cot)

ASCII rendering.

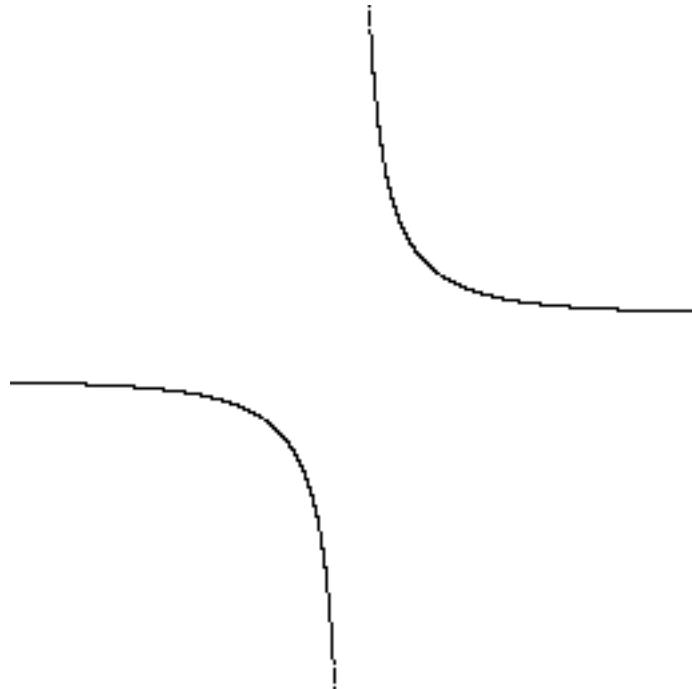
Bitmap rendering.



A.16. Hyperbolic cotangent (\coth)

ASCII rendering.

Bitmap rendering.

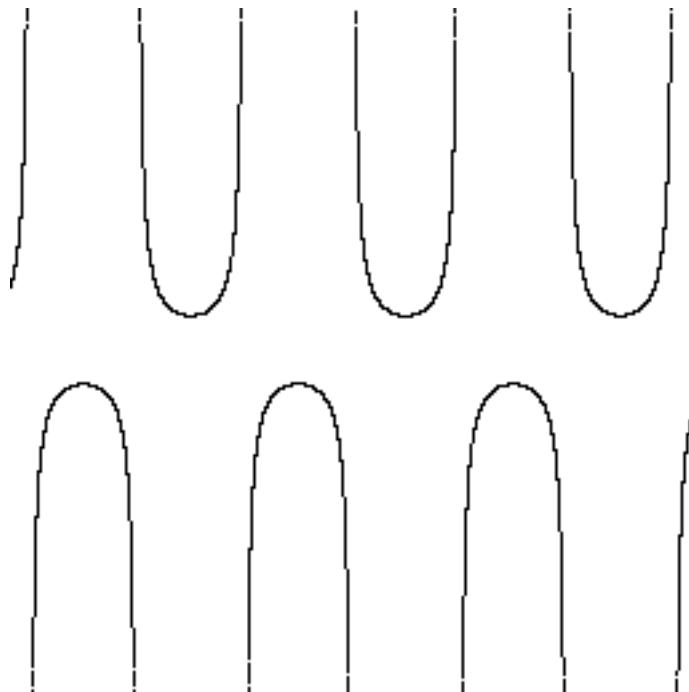


A.17. Cosecant (csc)

ASCII rendering.

```
.*.....*.....*.....*.....*.....*.....*.....*.....*.....*  
.*.....*.....*.....*.....*.....*.....*.....*.....*.....*  
.*.....*.....*.....*.....*.....*.....*.....**.....*.....*  
.*.....*.....*.....*.....*.....*.....*.....*.....*.....*  
.*.....*.....*.....*.....*.....*.....*.....*.....*.....*  
.*.....*.....*.....*.....*.....*.....*.....*.....*.....*  
*.....**.....*.....*.....*.....*.....*.....*.....*.....*  
.....*.....**.....*.....*.....**.....*.....**.....*.....*  
.....*****.....*****.....*****.....*****.....*****.....*  
.....*****.....*****.....*****.....*****.....*****.....*  
.....*.....*.....*.....*.....*.....*.....*.....*.....*.....*  
.....*.....*.....*.....*.....*.....*.....*.....*.....*.....*  
.....*.....*.....*.....*.....*.....*.....*.....*.....*.....*  
.....*.....*.....*.....*.....*.....*.....*.....*.....*.....*  
.....*.....*.....*.....*.....*.....*.....*.....*.....*.....*
```

Bitmap rendering.

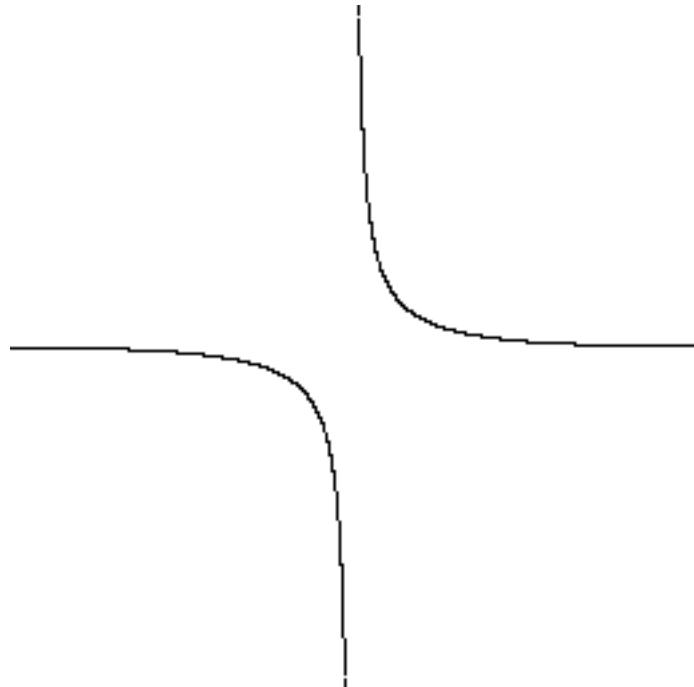


A.18. Hyperbolic cosecant (csch)

ASCII rendering.

```
.....*.....  
.....*.....  
.....*.....  
.....*.....  
.....*.....  
.....**.....  
.....**.....  
.....****.....  
*****.....*****  
*****.....*****  
.....***.....  
.....**.....  
.....*.....  
.....*.....  
.....*.....  
.....*.....  
.....*.....  
.....*.....
```

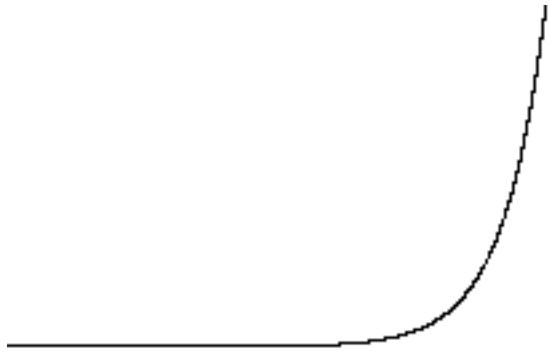
Bitmap rendering.



A.19. Exponential (exp)

ASCII rendering.

Bitmap rendering.

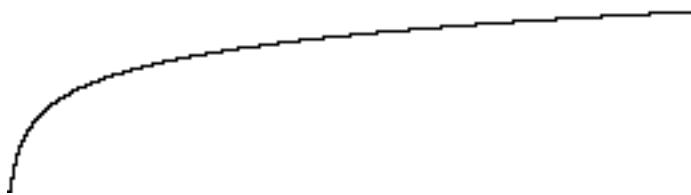


A.20. Natural logarithm (log)

ASCII rendering.

This image shows a full page of dot-grid paper. The grid consists of horizontal rows of small, evenly spaced dots. A single vertical line of dots runs along the left edge, creating a margin. The rest of the page is filled with the dot grid pattern.

Bitmap rendering.



A.21. Base-10 logarithm (log10)

ASCII rendering.

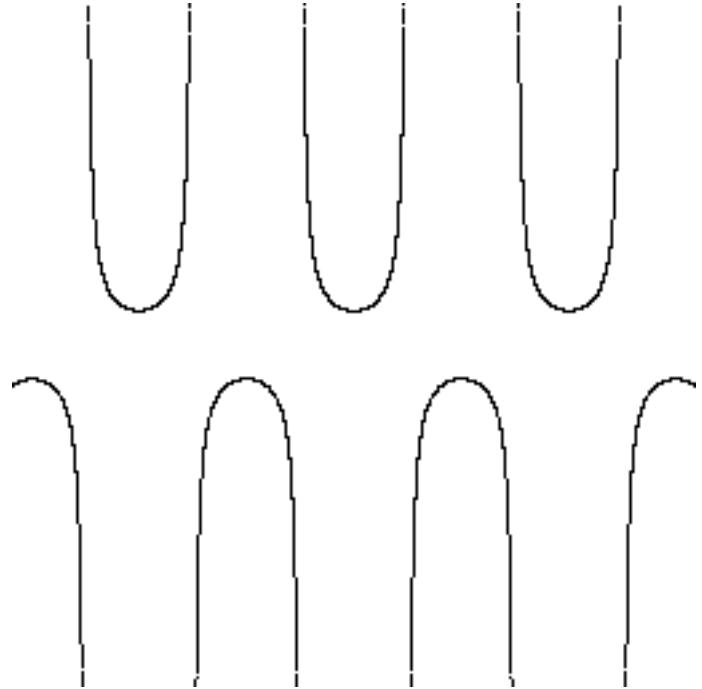
Bitmap rendering.



A.22. Secant (sec)

ASCII rendering.

Bitmap rendering.

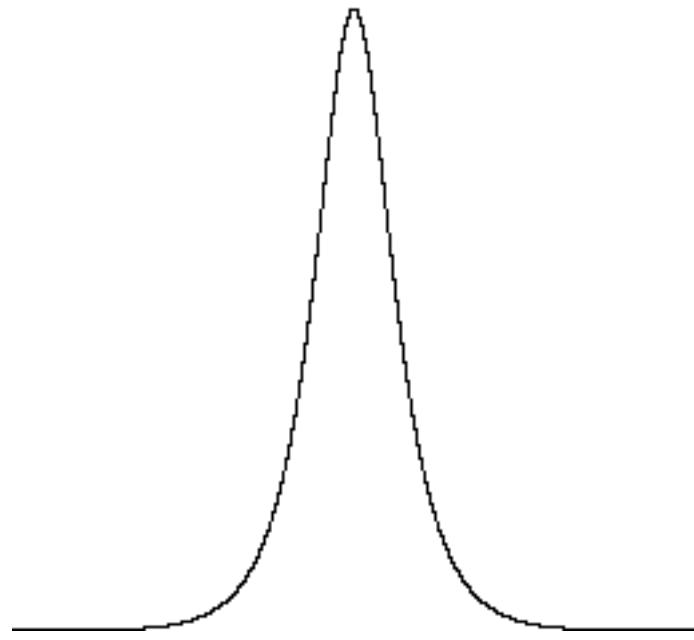


A.23. Hyperbolic secant (sech)

ASCII rendering.

The image shows a decorative border composed of a repeating pattern of stars and dots. The pattern consists of a central four-pointed star surrounded by a circle of smaller stars, all set against a background of small dots. This pattern is repeated in a staggered fashion along the edges of the frame.

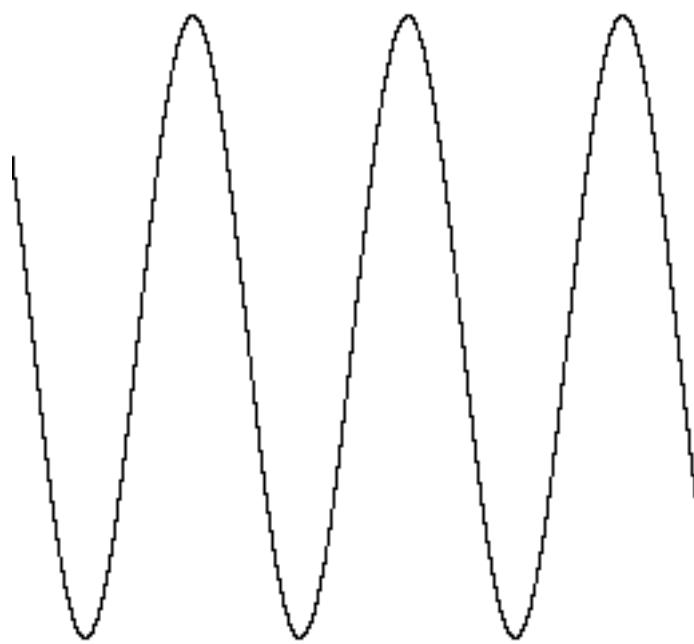
Bitmap rendering.



A.24. Sine (sin)

ASCII rendering.

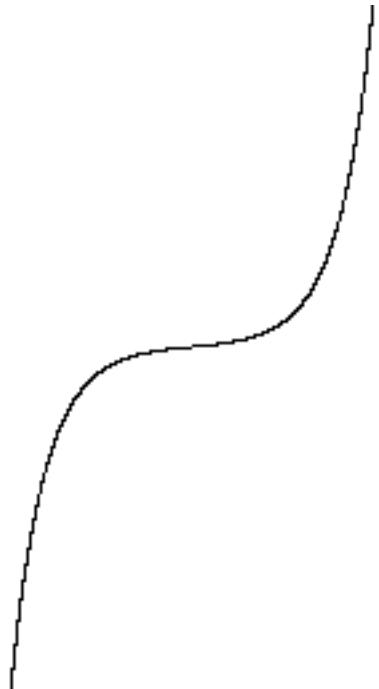
Bitmap rendering.



A.25. Hyperbolic sine (\sinh)

ASCII rendering.

Bitmap rendering.

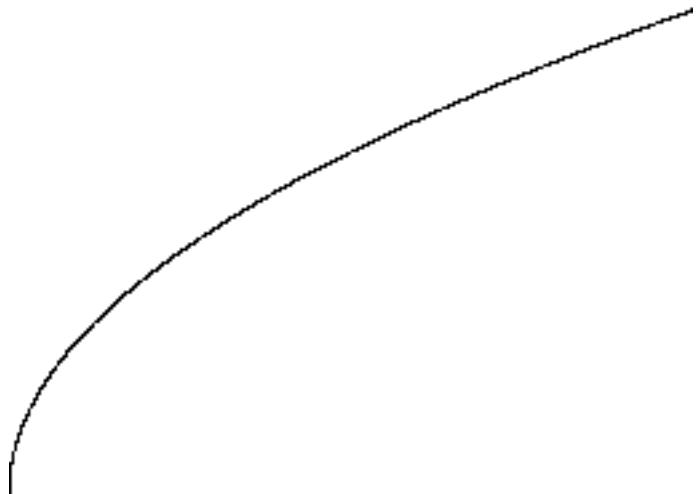


A.26. Square root (sqrt)

ASCII rendering.

A grid of 100 asterisks arranged in a 10x10 pattern. The asterisks are placed at regular intervals along the horizontal and vertical axes, creating a square pattern of stars.

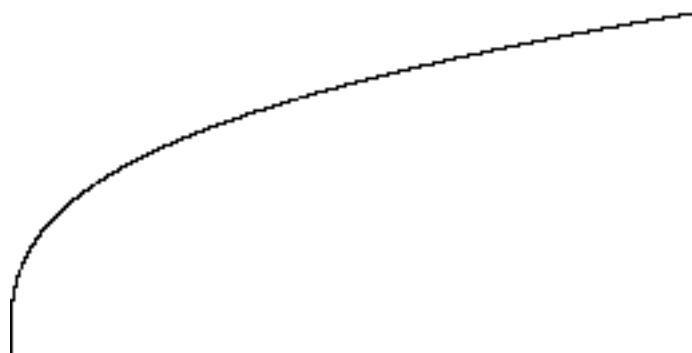
Bitmap rendering.



A.27. Cube root (cbrt)

ASCII rendering.

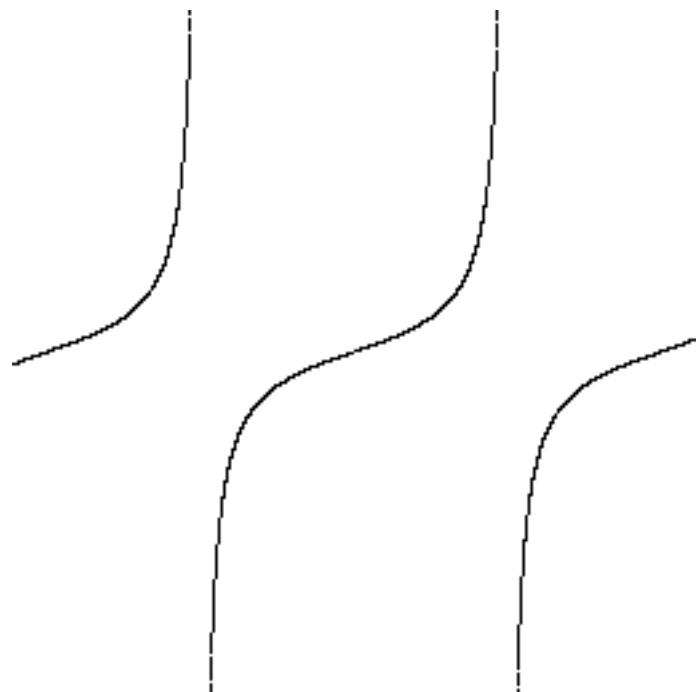
Bitmap rendering.



A.28. Tangent (tan)

ASCII rendering.

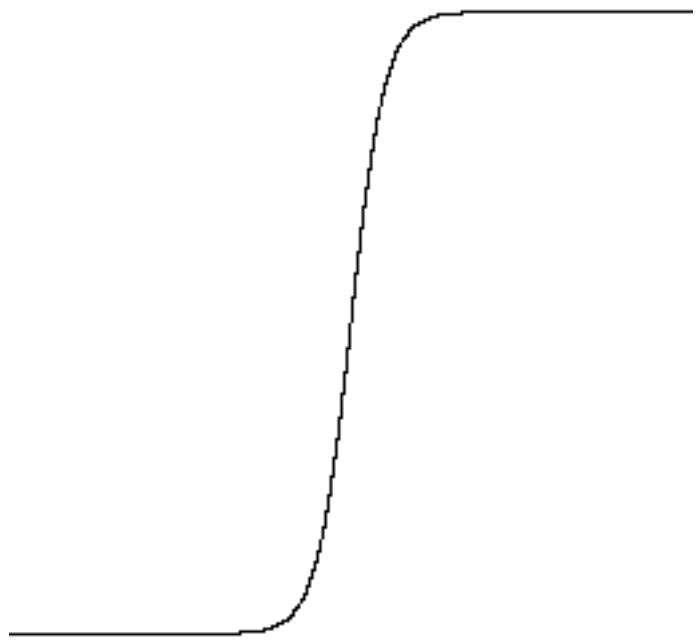
Bitmap rendering.



A.29. Hyperbolic tangent (\tanh)

ASCII rendering.

Bitmap rendering.



A.30. Floor (floor)

ASCII rendering.

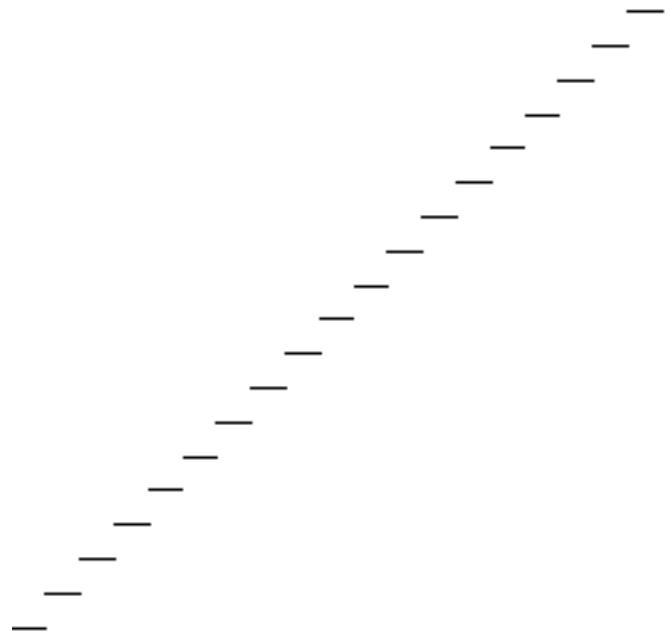
Bitmap rendering.

A.31. Ceiling (ceil)

ASCII rendering.

A sheet of dot-grid paper featuring horizontal dotted lines and vertical dashed lines that intersect to form a grid of columns. The grid is densely populated with four-pointed asterisk symbols (* * * *), which are arranged in a staggered pattern across the entire surface.

Bitmap rendering.



A.32. Round-to-nearest (round)

ASCII rendering.

Bitmap rendering.

A.33. Truncation (trunc)

ASCII rendering.

Bitmap rendering.

