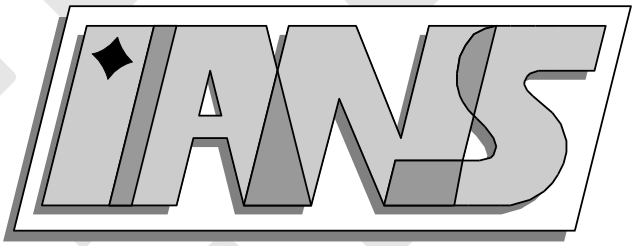
Universität Stuttgart



RANDEXPR: A Random Symbolic Expression Generator

Andreas Klimke

Berichte aus dem Institut für Angewandte Analysis und Numerische Simulation

Preprint 2003/004

Universität Stuttgart

RANDEXPR: A Random Symbolic Expression Generator

Andreas Klimke

Berichte aus dem Institut für Angewandte Analysis und Numerische Simulation

Preprint 2003/004

Institut für Angewandte Analysis und Numerische Simulation (IANS)
Fakultät Mathematik und Physik
Fachbereich Mathematik
Pfaffenwaldring 57
D-70 569 Stuttgart

E-Mail: ians-preprints@mathematik.uni-stuttgart.de
WWW: http://preprints.ians.uni-stuttgart.de

ISSN **1611-4176**

Abstract

The development of new numerical algorithms in fields such as optimization, sensitivity analysis, interval and fuzzy arithmetics, etc. usually requires intensive testing with a large set of functions to verify correctness and robustness of the code. To facilitate this task, we suggest to use an automatic tool to generate long, multi-variate expressions that would be difficult and cumbersome to code by hand. `randexpr` provides a facility with a customizable, easy-to-use interface to generate nested symbolic expressions of various types in MATLAB[1].

Keywords: random expressions, MATLAB

1 Introduction

For testing of our algorithms in fuzzy arithmetics, we needed a facility to generate random symbolic expressions in MATLAB. Hand-coding the expressions became too time consuming, we also feared that by using hand-coded expressions we might unwillingly bias testing results, or miss unlikely combinations of operators and/or elementary functions. Unfortunately, we could not find a suitable software to perform the automatic generation of random expressions (apart from random polynomials, see [1, 2]), meaning we had to develop our own routines. The following sections describe the basic use and advanced features of our implementation.

2 Basic use of `randexpr`

`randexpr` lets you generate random symbolic expressions from a predefined set of operators and functions. By default, `randexpr` will generate functions using the following set of operators and elementary functions:

1. Elementary arithmetic operators `+,-,*,/` and the power operator `^`,
2. trigonometric functions: `sin`, `cos`, `tan`, `cot`, `sec`, `csc`, `asin`, `acos`, `atan`, `acot`, `asec`, `acsc`,
3. exponential and logarithmic functions: `exp`, `log`, `log2`, `log10`, `pow2`, and
4. the square root `sqrt` and the absolute function `abs`.

The required input arguments of `randexpr` are the maximum number `nv` of different variables to be used in the expression and the length `len` of the expression, i.e. the number of variable occurrences. Syntax: `randexpr(nv,len)`. Here are a few examples:

Example 1. We would like to generate a random expression of a single variable (x), which should occur twice in each expression. Subsequent calls of `randexpr(1,2)` may produce the following results:

[1] MATLAB is a registered trademark of The Mathworks, Inc.

```
>> randexpr(1,2)
ans =
tan(x)-x

>> randexpr(1,2)
ans =
sqrt(9/x)*x

>> randexpr(1,2)
ans =
pow2(2^7/x)/6/x

>> randexpr(1,2)
ans =
x^9*4*x
```

Example 2. To generate a function of several variables, we would use, for example, `randexpr(3,5)`. The expressions below contain 5 variable occurrences each, however, note that the expression may not necessarily contain each of the three variables `x1`, `x2`, `x3`, such as the second expression.

```
>> randexpr(3,5)
ans =
tan(x1/(sqrt(9^3*log2(8-sin(x2)))-3/x3)^5*5^5)*exp(9-x3)/x1

>> randexpr(3,5)
ans =
(log2(1/x3)*x1-1*log2(x3))^3+x3-x1
```

It is important to note that the generator does not perform any verification whether the function is valid, e.g. check for a division by zero. Furthermore, no simplification of the expression is performed, e.g. it is possible to obtain expressions such as `x - x`.

2.1 Optional arguments

With four additional arguments, one can modify the basic behavior of the generator. The parameters may be arbitrarily combined. The extended syntax of `randexpr` is

`randexpr(nv, len, constants, constantsRange, constantsType, nesting)`.

Description of the optional parameters:

- **Constants.** The `constants` parameter can be used to increase or decrease the number of constants that the expression will contain. `constants` must be a real number within the interval $[0, 1]$. A value of 0 (zero) will suppress the occurrence of any constant, while a value of 1 will result in a high number of constants. The default value for `constants` is 0.5.

- **ConstantsRange.** The `constantsRange` parameter lets you specify a closed interval $[a, b]$ of real numbers from which the generator will create constants. You may specify any interval of the real axis, however, do not use intervals such as `[-inf,inf]` or `[eps,inf]`, since in this case rescaling of a number obtained by MATLAB's `rand` function (which produces floating point numbers in the closed interval `[2^(-53), 1-2^(-53)]`) to the desired range will not work properly. The default range is `[1,9]`.
- **ConstantsType.** The `constantsType` parameter specifies the type of constants generated. Possible values are `'double'` and `'integer'` (default).
- **Nesting.** You may alter the likelihood of nesting during expression generation with the `nesting` parameter. To suppress any nesting of the expression, use a `nesting` value of 0 (zero). To obtain an expression with a very high degree of nesting, use a value of 1. The default value is 0.5.

We provide a few examples below to illustrate the optional arguments.

Example 1. `randexpr(1,5,[],[-1,1],'double')` will generate a function of 1 variable x with 5 occurrences and some real constants in the interval $[-1, 1]$:

```
>> randexpr(1,5,[],[-1,1],'double')
ans =
log(tan(x)*x+0.74142-x^(-0.22903)*x*0.76942)+log(0.55061*x)
```

Example 2. `randexpr(2,6,0,[],[],0)` will generate a function of up to 2 variables x_1, x_2 with a total number of 6 variable occurrences, without any constants and without nesting.

```
>> randexpr(2,6,0,[],[],0)
ans =
x1+x2^2+x1+log2(x1)-abs(x2)*x1
```

3 Advanced use of `randexpr`

All options, operator, and function definitions of the `randexpr` routines are contained in a structure array. One can easily retrieve the default options structure and manipulate it. Thus, maximum flexibility is guaranteed to allow the generation of functions matching quite specific criteria.

3.1 Retrieving the options structure

To obtain the default options structure of `randexpr`, simply call the function with the string `'getoptions'` as the third parameter.

```
>> options = randexpr([],[],'getoptions')

options =

             group: {1x10 cell}
```

```
 probabilities: [0.7000 0 0 0.1000 0.1000 0 0.1000 0 0 0]
       nesting: 0.5000
     constants: 0.5000
constantsRange: [1 9]
 constantsType: 'integer'
     switchOff: {'\'  'sign'}
variablePowers: 'off'
```

You may now alter any of the parameters. To call `randexpr` with the modified options set, use the syntax `randexpr(nv, len, options)`, i.e. pass the altered options structure to the function as the third argument.

3.2 Description of the options structure member fields

The following paragraphs describe each of the member fields of the options structure.

3.2.1 group.

This field contains a cell array of operator and function groups to be used by the generator. The default options structure contains 10 such groups (see Table 1). This is the only member field that is actually a required field.

Each function or operator group must contain three fields:

1. `vals` is an array of strings containing the actual operators or functions of the group.
2. `nargs` specifies the number of function or operator arguments, i.e. 2 for binary operators, such as `*`, or 1 for elementary unary functions, such as `sin`.
3. The third required field, `type`, specifies the type of the group, either `'operator'` or `'function'`.

An additional field named `probabilities`, which is a vector of real numbers, assigns a unique probability value to each operator or function of the group. This field is optional.

Below, we show the default group definitions #1 and #5.

```
>> options.group{1}
ans =
          vals: {'+'  '-'  '*'  '/'  '\'  '^'}
         nargs: 2
          type: 'operator'
 probabilities: [0.1900 0.1900 0.1900 0.1900 0.1900 0.0500]

>> options.group{5}
ans =
          vals: {'exp'  'log'  'log2'  'log10'  'pow2'}
         nargs: 1
          type: 'function'
 probabilities: [0.2000 0.2000 0.2000 0.2000 0.2000]
```

As can be seen above, group #1 contains operators. Therefore, the value of the `type` field is `'operator'`. Since the operators are binary, i.e. they require a left- and a right-side argument, the `nargs` field value is 2. The probabilities are about evenly distributed, however, the power operator is set to a lower probability, which will reduce its occurrence in comparison to the other operators of the group to about one-fourth.

Table 1: Predefined groups of functions and operators

#	*type*	*nargs*	*contents*	*classification*
1	operator	2	`+ - * / \ ^`	binary arithmetic
2	operator	2	`== ~= < > <= >= & |`	binary relational / logical
3	operator	1	`+ - ~`	unary arithmetic / logical
4	function	1	`sin cos tan cot sec csc asin acos atan acot asec acsc`	trigonometric
5	function	1	`exp log log2 log10 pow2`	logarithmic / exponential
6	function	1	`sinh cosh tanh coth sech csch asinh acosh atanh acoth asech acsch`	hyperbolic
7	function	1	`sqrt abs sign`	other unary
8	function	1	`round floor ceil fix nextpow2`	unary rounding related
9	function	2	`rem mod min max`	rounding / quantitative
10	function	2	`bitand bitor bitxor`	bitwise

3.2.2 probabilities (optional).

To adjust the number of appearances of an operator or function from a specific group with respect to the other groups, the generator uses a column vector of probabilities. Each group is assigned a unique probability that determines its relative occurrence probability. The size of the vector must correspond to the number of operator groups. Therefore, for the default operator groups (10), the probability vector must contain 10 entries. The default value of the vector is $(.7, 0, 0, .1, .1, 0, .1, 0, 0, 0)$, i.e. we give the basic operator group #1 a higher weight than the other groups. Some of the function groups are actually deactivated by default, which is achieved by setting their probability to zero. If the `probabilities` field is omitted, all function/operator groups receive the same probability of occurrence.

3.2.3 nesting (optional), constants (optional), constantsRange (optional), constantsType (optional).

See description in Section 2.1.

3.2.4 switchOff (optional).

The `switchOff` field contains a cell array of strings containing operators and functions that are included in a group, but should not occur in the generated expression. Thus, you may use the `switchOff` field to temporarily deactivate specific operators or functions without explicitly removing them from its group. By default, the left division operator \ and the `sign` function are switched off. As an example, consider switching off the power operator (`^`) as well:

```
>> options = randexpr([],[],'getoptions');
>> options.switchOff
ans =
    '\'    'sign'
>> options.switchOff = {options.switchOff{:}, '^'};
>> options.switchOff
ans =
    '\'    'sign'    '^'
```

3.2.5 variablePowers (optional).

This switch may hold the values `'on'` or `'off'` (default). If `variablePowers` is set to `'on'`, the generator may select variables or even nested expressions as the exponent when a power (`^`) operator is encountered. Otherwise, only real or integer numbers, depending on the current setting of `constantsType`, are used as power arguments. If the `variablePowers` member field is omitted, it is assumed as turned `'off'`.

3.3 Defining a new options structure

In the following, we describe how to define a new valid set of options from scratch that can be passed to `randexpr`. As an example, we use a limited set of binary operators to generate random polynomial expressions. The first step is to define the operator groups. Since all operators are binary, we can put all of them in the same group. Furthermore, we specify the number of arguments (2 for binary operators) and the group type.

```
>> options.group{1}.vals = {'+','-','^'};
>> options.group{1}.nargs = 2;
>> options.group{1}.type = 'operator';
```

Now, we turn off nesting, and provide a constant range. By choosing the interval $[1, 4]$, we limit the maximum degree of the obtained polynomial to 4.

```
>> options.nesting = 0;
>> options.constantsRange = [1 4];
```

Let us view some results:

```
>> s = randexpr(1,10,options)
s =
x+2+x^2-2-x^2-4-x+2^2+x^2+x-2+x+2-x+2^4+x-x

>> s = randexpr(1,5,options)
s =
x^3+2^4+x+x^1+1+x+x
```

You could easily add more function or operator groups. Please refer to the forthcoming Section 3.4.

3.4 Modifying an existing options structure

You can of course modify an existing options structure, such as the default options of `randexpr`. To do that, use the standard MATLAB commands for structure manipulation, such as assignment of values `S = setfield(S,'field',V)` (or `S.field = V`), or removal of fields `S = rmfield(S,'field')`. Below are some common modifications you might wish to perform.

Removing a group. To remove one or more groups from the options structure, two steps are necessary. First, remove the cell fields containing the desired groups from the `group` cell array. E.g. to remove group 4 from the default options structure, use

```
>> options = randexpr([],[],'getoptions');
>> options.group = options.group([1:3,5:10]);
```

This statement will create a new `group` cell array within the options structure containing 9 entries. Second, the probabilities field must be adjusted accordingly to contain 9 probability values for the 9 groups. Use

```
>> options = rmfield(options,'probabilities');
```

to remove a previous definition of group probabilities, or define new ones for each group, e.g.

```
>> options.probabilities = [0.5 0 0 0.1 0.1 0.1 0.1 0.1 0];
```

The probability vector should, of course, contain 9 entries and add up to 1.

Modifying a group. You may edit each group separately to add new operators and functions, or to change the probability of each operator individually. To view the properties of a specific group, e.g. group #1, use

```
>> options.group{1}
ans =
             vals: {'+'  '-'  '*'  '/'  '\'  '^'}
            nargs: 2
             type: 'operator'
    probabilities: [0.1900 0.1900 0.1900 0.1900 0.1900 0.0500]
```

You may add new a operator, for example, by assigning a new cell array of strings to the `vals` field:

```
>> options.group{1}.vals = {options.group{1}.vals{:}, '\%'};
>> options.group{1}
ans =
             vals: {'+'  '-'  '*'  '/'  '\'  '^'  '%'}
            nargs: 2
             type: 'operator'
    probabilities: [0.1900 0.1900 0.1900 0.1900 0.1900 0.0500]
```

The probability vector of the group must be adjusted accordingly, of course.

Deactivating a group. An alternative to explicitly removing a group is to just deactivate it by setting its probability to 0 (zero). Use the `options.probabilities` field to do this.

4 Efficient use of `randexpr` in MATLAB

Expressions generated with `randexpr` can be easily combined with standard MATLAB commands for further processing. You may wish to `vectorize` the function to allow fast processing of the generated function for multiple function values:

```
>> s = randexpr(3,5)
s =
x3+(tan(tan(8/x1)*x2)/9+x2)+8/x3

>> vectorize(s)
ans =
x3+(tan(tan(8./x1).*x2)./9+x2)+8./x3
```

Or, create an `inline` function from the expression string:

```
>> f = inline(s)
f =
     Inline function:
     f(x1,x2,x3) = x3+(tan(tan(8/x1)*x2)/9+x2)+8/x3
```

If your MATLAB installation includes the Symbolic Math Toolbox, you may also generate a symbolic expression with the `sym` command to perform further processing, such as simplifying the expression, pretty-printing the expression, generating a LaTeX-format string, or generating a C or Fortran code representation of the symbolic expression:

```
>> s = randexpr(1,4)
s =
x/x/4*(sqrt(8*asec(x))+x)
>> f = sym(s);
>> simplify(f)
```

```
ans =
1/2*2^(1/2)*asec(x)^(1/2)+1/4*x
>> pretty(f);
                                 1/2         1/2
                          1/2 2    asec(x)     + 1/4 x
>> latex(f)
ans =
1/2\,\sqrt {2}\sqrt {{\it arcsec}(x)}+1/4\,x
>> ccode(f)
ans =
      t0 = sqrt(2.0)*sqrt(acos(1/x))/2.0+x/4.0;
```

5 Obtaining `randexpr`

The software consists of a the single `.m` file `randexpr.m`. No additional MATLAB packages are required. It is available for download at

`http://matlabdb.mathematik.uni-stuttgart.de/search_result.jsp?Search=Randexpr&SearchCategory=Title`

References

[1] Waterloo Maple, Inc. randpoly - random polynomial generator. *Documentation of Maple*, `http://www.maplesoft.com`, last visited March 2003.

[2] F. J. Wright. RANDPOLY: A Random Polynomial Generator. *Documentation of Reduce Computer Algebra System*, 1994, `http://www.zib.de/Symbolik/reduce/`, last visited March 2003.

Andreas Klimke

Institute of Applied Analysis and Numerical Simulation
University of Stuttgart
Pfaffenwaldring 57
70550 Stuttgart, Germany

E-Mail: `klimke@ians.uni-stuttgart.de`

Erschienene Preprints ab Nummer 2003/001

Komplette Liste: `http://preprints.ians.uni-stuttgart.de`

2003/001 *Lamichhane, B. P., Wohlmuth, B. I.*: Mortar Finite Elements for Interface Problems.

2003/002 *Dryja, M., Gantner, A., Widlund, O. B., Wohlmuth, B. I.*: Multilevel Additive Schwarz Preconditioner For Nonconforming Mortar Finite Element Methods.

2003/003 *Klimke, A., Hanss, M.*: On the Reliability of the Influence Measure in the Transformation Method of Fuzzy Arithmetic.

2003/004 *Klimke, A.*: RANDEXPR: A Random Symbolic Expression Generator.