The Stata Journal (2007)
7, Number 4, pp. 542–555

A generic function evaluator implemented in Mata

Henrik Støvring
Research Unit for General Practice
University of Southern Denmark
Odense, Denmark
hstovring@health.sdu.dk

Abstract.

Background: When implementing new statistical procedures, there is often a need for simple—and yet computationally efficient—ways of numerically evaluating composite distribution functions. If the statistical procedure must support calculations for censored and noncensored cases, those calculations should be carried out using efficient computational implementations of both definite and indefinite integrals (e.g., calculation of tail areas of distribution functions).

Method: We developed a generic function evaluator such that users may specify a function using reverse Polish notation. As its argument the function evaluator takes a matrix of pointers and then applies the rows of this matrix to its internally defined stack of pointers. Accordingly, each row of the argument matrix defines a single operation such as evaluating a function on the current element of the stack, applying an algebraic operation to the two top elements of the stack, or manipulating the stack itself. Defining new composite distribution functions from other (atomic) distribution functions then corresponds to joining two or more function-defining matrices vertically. This approach can further be used to obtain integrals of any defined function. As an example we show how the density and distribution function for the minimum of two Weibull distributed random variables can be numerically evaluated and integrated.

Results: The procedure provides a flexible and extensible framework for implementing numerical evaluation of general, composite distributions. The procedure is numerically relatively efficient, although not optimal.

Keywords: pr0034, rpnfcn(), RPN, Mata

1 Introduction

In applied statistics there is often a need for numerically evaluating probability distributions that are composite in the sense that they arise from two or more random variables, each with their own particular distribution function.

One example is a two-component mixture with the following generic form for its cumulative distribution function (cdf)

$$F(t; p, \theta_1, \theta_2) = p\, F_1(t; \theta_1) + (1 - p) F_2(t; \theta_2)$$

where F_1 and F_2 are two cdfs, p is a probability, and θ_1 and θ_2 are parameter vectors.

A second example is the probability density function (pdf) of $Y = \min(X, Z)$, where X and Z are two independent, nonnegative random variables with pdfs f_X and f_Z, respectively, and associated survivor functions S_X and S_Z. It is well known (see, for example, Degroot 1986, 160) that here the pdf for Y is given by (with obvious notation)

$$f_Y(t; \theta_1, \theta_2) = f_X(t; \theta_1) S_Z(t; \theta_2) + S_X(t; \theta_1) f_Z(t; \theta_2) \tag{1}$$

and survivor function given by

$$S_Y(t; \theta_1, \theta_2) = S_X(t; \theta_1) S_Z(t; \theta_2) \tag{2}$$

A third example where the problem becomes more compounded is the so-called forward recurrence distribution given by the following generic form

$$f_R(t; \theta) = \frac{S_T(t; \theta)}{\mu_T(\theta)} \tag{3}$$

where $\mu_T(\theta)$ is the mean of T, a nonnegative random variable with survivor function S_T (Støvring and Vach 2005). Because T is nonnegative, its mean can be written as

$$\mu = \int_0^\infty S_T(s; \theta)\, ds$$

Often no analytical expressions for such integrals of S_T exist, and so numerical evaluation must be applied to obtain this mean, as well as the cdf of R, which is given by

$$F_R(t; \theta) = \mu_T^{-1}(\theta) \int_0^t S_T(s; \theta) ds$$

One straightforward method of implementing these functions as ordinary Mata functions would be to code a new function that first invokes the subfunctions it is made of, and then combines results of these function calls appropriately. This nesting approach implies that we would need to pass the vector of all parameter values, (p, θ_1, θ_2), say, to the new function F, which then must parse them and pass them on to the relevant functions, F_1 and F_2. Although this is possible, it becomes tedious when F_1 or F_2 are themselves composite functions or if we want to allow for different functional shapes of F_1 and F_2. Different shapes effectively require writing a new definition of F for each shape with dedicated and appropriate parsing of relevant parameters. Further, when

we need to integrate the new composite function, it is difficult to implement a generic integration function, which can easily accommodate the different structures of the integrand and the varying number of arguments needed. Again, although possible, so far no generic routines exist for solving the problem generally, as far as the author is aware.

To overcome this we suggest and implement a new function evaluator based on reverse Polish notation (RPN). The motivation for using RPN is that it avoids nesting of functions and instead naturally specifies computations sequentially. This has the advantage of allowing function definitions to be framed as matrices of pointers, where each row defines one computational step in the RPN algorithm. These definition matrices can be straightforwardly stacked to construct new composite functions, thus facilitating implementation of complicated pdfs and cdfs. Finally, the strategy allows a general implementation of integration as this can be obtained from sandwiching any integrand between two general, special purpose functions defining the points of evaluation and summarizing the integrand at the evaluated points.

The paper is organized as follows: section 2.1 introduces the principle of RPN before section 2.2 describes how it is implemented in Mata. In sections 2.3 and 2.4, I show how the procedure allows for general numerical integration, both for definite and indefinite integrals. To show the potential of the procedure, section 2.5 gives an example on evaluation of a forward recurrence distribution arising from a random variable, which itself is the minimum of two Weibull distributed random variables. Finally, I discuss the primary advantages and limitations of the suggested procedure in section 3.

2 Methods

2.1 The principle of RPN

Conceptually, RPN is based on first defining two operands and then applying a binary operator (addition, multiplication, etc.) to obtain the result. Although the notation for this principle is less intuitive than the standard infix notation, its main advantage is obliviating the need for parentheses and equation signs; i.e., it essentially specifies computations in one (long) serial sequence instead of the intuitively appealing nesting of the usual notation with parentheses. Anyone with experience in using an RPN calculator has recognized that RPN involves fewer button pushes and allows one to compute almost straight from left to right of any numerical expression, although it needs a slightly different organization of computations than a traditional calculator.

The key step in implementing RPN is to setup an indexed stack with elements containing operands and results. Then one must define functions for manipulating stack elements. These functions roughly fall into three categories: stack manipulation, binary operators, and single element functions. Consider for example the steps involved in the computation of the exponential density given by

$$f(t;\lambda) = \lambda \exp[-\lambda t]$$

With RPN, the computation would be given by the algorithm presented in figure 1.

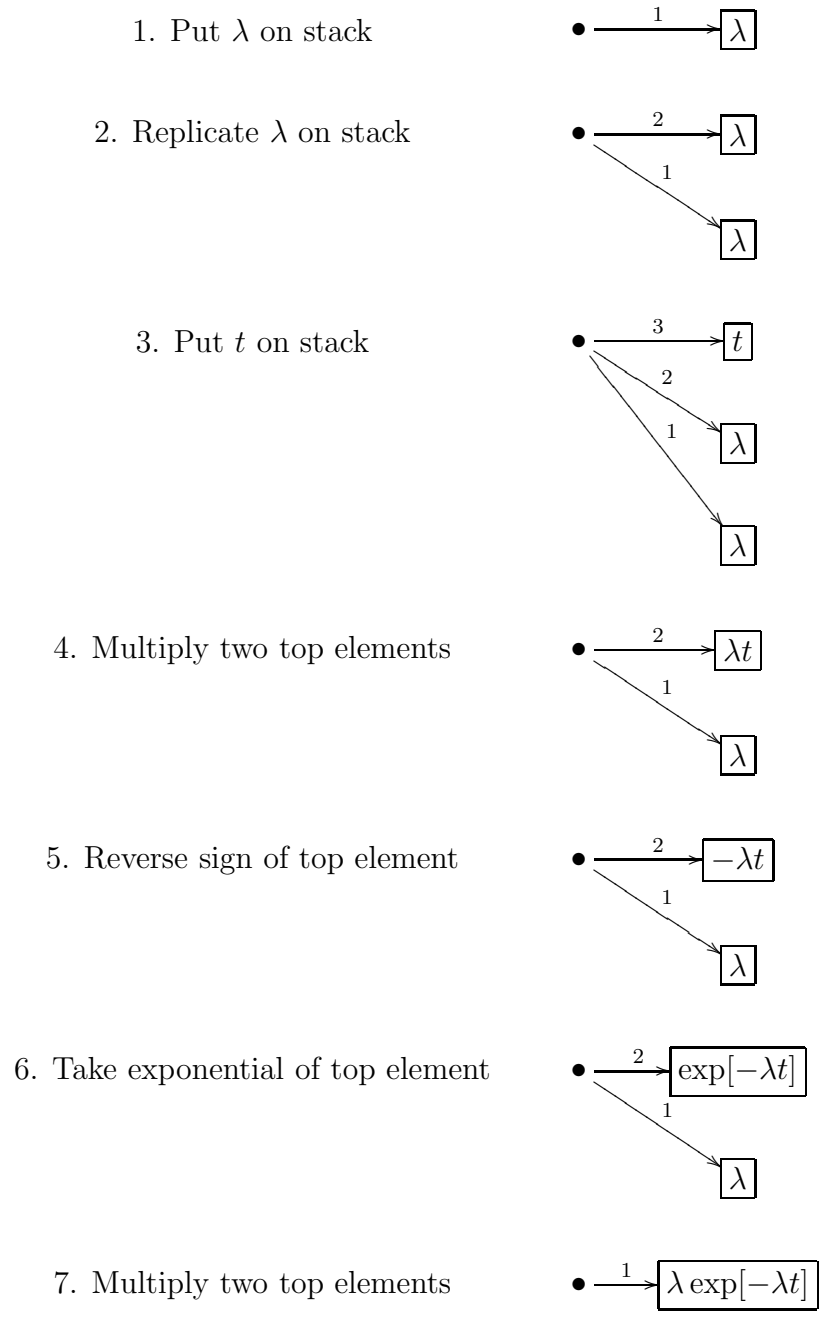

Figure 1: The steps involved in computing the exponential pdf using RPN. The • with arrows denotes the stack, which is nothing more than an indexed set of pointers, 1, 2, ..., here denoted by arrows, and the referenced boxes contain the elements of the stack after the indicated action. Operations on elements of the stack place the result of the operation on the stack.

Steps 1–3 are examples of stack manipulation actions: steps 1 and 3 put arguments λ and t, respectively, into the stack, whereas step 2 copies the current top element to a new top element of the stack. Steps 4 and 7 apply the binary operator of multiplication to the two top elements of the stack. Steps 5 and 6 apply a single element function to the top element of the stack and returns the result in the same place.

2.2 Implementing RPN for matrices in Mata

The implementation of an RPN function evaluator in Mata consists of two main ingredients. The first is a function, `rpnfcn()`, which sets up the stack and sequentially applies the individual steps of any algorithm defined in a matrix of pointers in which the rows each define one step in the algorithm. Internally, `rpnfcn()` relies heavily on a helper function, `anyeqpt()`, for determining the type of action to take in each step. The code of `rpnfcn()` and `anyeqpt()` are included in the `rpn` package. The second main ingredient consists of various stack functions of the three types described above. A short description of the basic implemented stack functions is given in table 1. The implemented pdfs and cdfs are only examples and more should be implemented whenever need arises.

The actual parameterizations chosen for the densities are described by the following expressions for the associated survivor functions:

$$\begin{aligned} \text{Exponential:} \qquad & S(t;\theta=\beta) = \exp[-e^{\beta}t] \\ \text{Weibull:} \qquad & S\{t;\theta=(\beta,\alpha)\} = \exp\left[-(e^{\beta}t)^{\exp(\alpha)}\right] \end{aligned} \tag{4}$$

where Φ is the standard normal cdf. The parameterization is chosen so as to avoid restricting parameters to be positive, which is useful in avoiding boundary problems in estimation procedures.

Table 1: Implemented stack functions. Argument types are internal elements of the stack (*Int*) or external arguments (*Ext*) supplied by a pointer in the controlling algorithm matrix.

Type	Name	i	Arguments *Int*	Arguments *Ext*	Description
Manipulation	`tostack`	$+1$		x	Puts x onto stack
	`enter`	$+1$	e_i		Copies e_i to new top element e_{i+1}
	`swapst`	0	e_i, e_{i-1}		Swap content of e_i and e_{i-1}
	`rotst`	0	$e_1, e_2, \ldots, e_i$		Rotate stack such that e_1 is new top element and all other elements are pushed down one step
Binary operators[a]	`add`	-1	e_i, e_{i-1}		Add e_i and e_{i-1}
	`subtract`	-1	e_i, e_{i-1}		Subtract e_i from e_{i-1}
	`product`	-1	e_i, e_{i-1}		Multiply e_i and e_{i-1}
	`divide`	-1	e_i, e_{i-1}		Divide e_{i-1} with e_i
Single element functions	`fexp`	0	e_i	θ	Exponential pdf
	`Sexp`	0	e_i	θ	Exponential survivor function
	`fwei`	0	e_i	θ	Weibull pdf
	`Swei`	0	e_i	θ	Weibull survivor function
Integration	`MCatnode`	0	e_i	(a, b)	Setup nodes for stratified, antithetic Monte Carlo integration
	`MCatwt`	0	e_i	(a, b)	Setup weights for stratified, antithetic Monte Carlo integration
	`tlMCatnode`	0	e_i	(a)	Setup nodes for stratified, antithetic Monte Carlo indefinite integration
	`tlMCatwt`	0	e_i	(a)	Setup weights for stratified, antithetic Monte Carlo indefinite integration
	`intres`	-1	e_i, e_{i-1}		Row sum of e_{i-1} multiplied elementwise with e_i

[a] All binary operators act elementwise using the Mata colon operators `:+`, `:-`, `:*`, and `:/`, respectively.

Example on basic use

To illustrate how to apply the `rpnfcn()` function, consider computation of the density for the minimum of two Weibull distributed random variables, i.e., the density given in (1) with the Weibull parameterization of (4). The algorithm matrix for this density takes the following form in Mata syntax:

```
. mata
------------------------------------------------- mata (type end to exit) -----
: x = (NULL)
: theta1 = (NULL)
: theta2 = (NULL)
: fweimin = (&tostack(),  &x \
>                 &enter(),    NULL \
>                 &enter(),    NULL \
>                 &fwei(),     &theta1 \
>                 &swapst(),   NULL \
>                 &Swei(),     &theta2 \
>                 &product(),  NULL \
>                 &rotst(),    NULL \
>                 &enter(),    NULL \
>                 &Swei(),     &theta1 \
>                 &swapst(),   NULL \
>                 &fwei(),     &theta2 \
>                 &product(),  NULL \
>                 &add(),      NULL
>                 )
: end
------------------------------------------------------------------------------
```

We defined `x`, `theta1`, and `theta2` to be empty matrices as they must exist before defining the algorithm matrix. Their contents can, however, be set after defining the algorithm matrix. The following Mata code fills the matrices and evaluates the function defined in `fweimin`:

```
. mata
------------------------------------------------- mata (type end to exit) -----
: x = (0, 2, 8 \
>      .7, 1, 6)
: theta1 = (-2, 2)
: theta2 = (0, .3)
: rpnfcn(fweimin)
                  1             2             3
  +-------------------------------------------+
1 |               0   .1344860648   4.74373e-08 |
2 |    .6423218484   .4965861523   .0000298698 |
  +-------------------------------------------+
: end
------------------------------------------------------------------------------
```

2.3 Definite integration with RPN

Suppose that we want to evaluate

$$H(a,b) = \int_a^b h(s)ds$$

Conceptually, numerical integration can be divided into three steps: first, define the points at which to evaluate h, the so-called nodes; second, apply h to these nodes; and third, sum the weighted results of this evaluation, where the weights depend on the choice of nodes. For an example of how to integrate a function with stratified, antithetic Monte Carlo integration, see Ripley (1987, 131–132). Formally, the computation to be done is given by

$$\int_a^b f(s)ds \approx \frac{b-a}{2M} \sum_{j=0}^{M-1} \Big[f\left\{a + \delta(j+u_j)\right\} + f\left\{a + \delta(j+1-u_j)\right\} \Big]$$

where $\delta = (b-a)/M$ and all u_j are uniformly distributed and mutually independent. The nodes here are thus

$$a + \delta(j+u_j) \text{ and } a + \delta(j+1-u_j)$$

all with weights $(b-a)/(2M)$.

Following this, three functions have been implemented to setup a matrix containing the nodes (`MCatnode`), setup a matrix containing the weights (`MCatwt`), and compute the integration result by summing the weighted evaluation results (`intres`), respectively. All three functions are generic in the sense that they are completely independent of the evaluator to be used: `MCatnode` replaces the matrix in the current top element of the stack with a matrix of the relevant nodes. The function to be integrated is then applied to the nodes leaving the result in the top element of the stack, and finally, the weights are computed and the summation function is applied, yielding the result matrix. For algorithm matrix notation, this means sandwiching the rows for the function to be integrated between rows concerning `MCatnode` initially, and—at the bottom—`MCatwt` and `intres`. For this to work, the function to be evaluated must only rely on the single matrix of nodes present in the stack after execution of `MCatnode`.

Example on definite integration

Consider integration of the density for the minimum of two Weibull distributed random variables, `fweimin` defined above. Let the matrix `ab` be an $n \times 2$ matrix, where each row contains integration limits, such that the first column is the lower limit and the second column is the upper limit. Assume that `u` is an $n \times m$ matrix containing uniformly distributed random numbers. For example, the matrices could be given by

```
. set seed 9876
. mata
------------------------------------------------- mata (type end to exit) -----
: ab = (0, 4 \
>       0, 2 \
>       2, 4 \
>       4, 10)
: u = uniform(rows(ab), 30)
: end
--------------------------------------------------------------------------------
```

The algorithm matrix for the integration can now be defined, and the integration performed as follows:

```
. mata
------------------------------------------------- mata (type end to exit) -----
: intfweimin = (&tostack(), &u \
>               &MCatnode(), &ab ) \
>               fweimin[2..rows(fweimin), .] \
>              (&tostack(), &u \
>               &MCatwt(), &ab \
>               &intres(), NULL)
: rpnfcn(intfweimin)
                 1
  +--------------+
1 |  .9972108522 |
2 |   .921117737 |
3 |  .0766576168 |
4 |  .0014903807 |
  +--------------+
:
: end
--------------------------------------------------------------------------------
```

2.4 Indefinite integration with RPN

Suppose that we want to evaluate

$$S(a) = \int_a^{\infty} h(s)ds$$

One way to do this numerically is to apply the transformation $s \mapsto e^{-s}$ so that the integral becomes

$$S(a) = \int_0^{\exp(-a)} \frac{h(-\log u)}{u} du$$

Because the integral is now definite, it can be evaluated as above, except that the weights should now take into account the term u^{-1}. For stratified, antithetic sampling the nodes become

$$a - \log\left(\frac{j+u_j}{M}\right) \quad \text{and} \quad a - \log\left(\frac{j+1-u_j}{M}\right)$$

for $j = 0, 1, \ldots, M-1$. The corresponding weights are

$$\frac{1}{2(j+u_j)} \quad \text{and} \quad \frac{1}{2(j+1-u_j)}$$

Example on indefinite integration

Consider tail integration of the density for the minimum of two Weibull distributed random variables, `fweimin` defined above. Let the matrix `a` be an $n \times 1$ vector, where each element is a lower integration limit. Assume that `u` is an $n \times m$ matrix containing uniformly distributed random numbers. For example, the matrices could be given by

```
. set seed 4757
. mata
------------------------------------------------- mata (type end to exit) -----
: a = (0\
>      2\
>      4)
: u = uniform(rows(a), 20)
: end
--------------------------------------------------------------------------------
```

The algorithm matrix for the integration can now be defined, and the integration performed as follows:

```
. mata
------------------------------------------------- mata (type end to exit) -----
: Sweimin = (&tostack(), &u \
>            &tlMCatnode(), &a ) \
>           fweimin[2..rows(fweimin), .] \
>           (&tostack(), &u \
>            &tlMCatwt(), &a \
>            &intres(), NULL)
: rpnfcn(Sweimin)
                  1
  +-------------------+
1 |   .9986485593     |
2 |   .0781537442     |
3 |   .0014923006     |
  +-------------------+
:
: end
--------------------------------------------------------------------------------
```

Here it is also possible to evaluate the integral explicitly by using the formula in (2), and so we can validate the results above:

```
. mata
------------------------------------------------- mata (type end to exit) -----
: Sweimin_exact = (&tostack(), &a \
>                  &enter(), NULL \
>                  &Swei(), &theta1 \
>                  &swapst(), NULL \
>                  &Swei(), &theta2 \
>                  &product(), NULL)
: rpnfcn(Sweimin_exact)
                 1
  +-------------------+
1 |                 1 |
2 |    .0781648043   |
3 |    .0014922392   |
  +-------------------+
: end
--------------------------------------------------------------------------------
```

Although the numerical evaluation based on stratified, antithetic sampling leads to results close to their true values, tail integration of a density for a nonnegative random variable all the way from zero should be considered an extreme case, where precision is low. In practical applications, such integrals are of course best replaced by their (known) theoretical value of 1. For slightly larger values of a, the integral is best replaced by a definite integral from a to some large constant b (larger than all elements of a) plus an indefinite one from b. The indefinite integral from b only needs to be computed once and hence allows using many strata (M) for improving precision. One simple choice for b is

$$b = \max(a_i | i = 1, 2, \ldots, n)$$

where a_i's are the elements of a.

2.5 Example on forward recurrence density for minimum of two Weibull distributed random variables

To show some of the potential of the implemented RPN evaluator, let us consider the distribution arising from transforming the minimum of two Weibull distributed random variables into a forward recurrence time. Based on the expressions in (2) and (3) the density for the new random variable R is given by

$$f_R(r) = \frac{S_X(r)S_Z(r)}{\int_0^\infty S_X(s)S_Z(s)\,ds}$$

where S_X and S_Z are the survivor functions of the two original Weibull distributed random variables. This density can be implemented and evaluated using the RPN procedure as follows:

```
. mata
------------------------------------------------- mata (type end to exit) -----
: a0 = J(1, 1, 0)
: u0 = uniform(rows(a0), 20)
: frfweimin_exact = (&tostack(), &x ) \
>                   Sweimin_exact[2..rows(Sweimin_exact), .] \
>                   (&tostack(), &u0 \
>                    &tlMCatnode(), &a0 ) \
>                   Sweimin_exact[2..rows(Sweimin_exact), .] \
>                   (&tostack(), &u0 \
>                    &tlMCatwt(), &a0 \
>                    &intres(), NULL \
>                    &divide(), NULL)
: x
       1    2    3
    +-------------+
  1 |   0    2    8 |
  2 |  .7    1    6 |
    +-------------+
: rpnfcn(frfweimin_exact)
                1            2            3
    +-------------------------------------+
  1 |  1.090430679   .0852333006   1.16102e-08 |
  2 |  .5878359215   .4011468755   .0000116703 |
    +-------------------------------------+
: end
------------------------------------------------------------------------------
```

The corresponding survivor function can be implemented and evaluated as follows:

```
. mata
------------------------------------------------- mata (type end to exit) -----
: a0 = J(3, 1, 0)
: u0 = uniform(rows(a0), 10)
:
: frSweimin_exact = (&tostack(), &u \
>                    &tlMCatnode(), &a ) \
>                   Sweimin_exact[2..rows(Sweimin_exact), .] \
>                   (&tostack(), &u \
>                    &tlMCatwt(), &a \
>                    &intres(), NULL \
>                    &tostack(), &u0 \
>                    &tlMCatnode(), &a0 ) \
>                   Sweimin_exact[2..rows(Sweimin_exact), .] \
>                   (&tostack(), &u0 \
>                    &tlMCatwt(), &a0 \
>                    &intres(), NULL \
>                    &divide(), NULL)
: a
       1
    +---+
  1 |  0 |
  2 |  2 |
  3 |  4 |
    +---+
```

```
: rpnfcn(frSweimin_exact)
                  1
  ┌──────────────────┐
1 │   .9985048706    │
2 │   .0459127621    │
3 │   .0007040949    │
  └──────────────────┘

: end
```

3 Discussion

In this paper we have developed and described a general procedure for numerical evaluation in Mata of complex density and distribution functions and for general integration of such functions. The procedure uses RPN to serialize computations, where individual steps in the computation are specified in rows of a controlling algorithm matrix. The approach allows building complex expressions by simply stacking relevant rows of algorithm matrices, and we have shown how to implement general integration based on stratified, antithetic sampling both for definite and indefinite integrals. The algorithm is numerically relatively efficient as it encourages not copying argument matrices more than needed in any given algorithm, and internally, the result matrix replaces the input matrix whenever possible, helping to minimize the strain on memory.

From one perspective, the procedure does not add anything new: what can be calculated with RPN can also be calculated with the ordinary `infix` notation already present in Mata. The more interesting question is, however, how easy it is to implement a complex function in a systematic way. From that perspective, the main virtue of the RPN implementation is its use of algorithm matrices. These matrices make the function definition—together with its required arguments—explicit, and it straightforwardly allows the function to be reused in even more complex functions without subparsing of arguments. It is possible that this could have also been obtained by clever exploitation of Mata's facility for defining structures, but the author is not aware of existing implementations of this type.

The first limitation of the described RPN procedure is its reliance on specific, named arguments, which all must exist at the time of defining the algorithm matrix. This makes algorithm matrices less universal, although this can in part be overcome with appropriate naming conventions for arguments. The primary objective of the RPN procedure—at least as envisaged by its author—is, however, to be used internally in other user-developed packages, and so the naming of arguments is less problematic.

Second, it is not clear how double integrals are to be handled with the proposed RPN procedure. Although it is often possible to avoid this, situations do arise where double integrals are inevitable. If, for example, no direct analytical expression had existed for the survivor function for the minimum of two Weibull distributed random variables, a double integral would have arisen when considering the survivor function for the associated forward recurrence distribution evaluated in section 2.5.

A further inconvenience of the procedure is that algorithm matrices quickly become complex and hard to read, which hinders debugging. To some degree this is offset by the ability to build complex functions from simpler algorithm matrices, which can be debugged individually, but in some settings the debugging may still prove problematic. Accordingly, the RPN procedure should not be considered a panache but rather its advantages and shortcomings should be carefully weighed against each other when considering how to implement a specific numerical evaluation.

With the present implementation of the RPN procedure, only the most basic stack functions and densities are supplied. For the RPN procedure to become more useful more subfunctions are needed, particularly implementation of other probability distributions and other integration rules.

4 References

Degroot, M. 1986. *Probability and Statistics*. 2nd ed. Reading, MA: Addison–Wesley.

Ripley, B. D. 1987. *Stochastic Simulation*. New York: Wiley.

Støvring, H., and W. Vach. 2005. Estimation of prevalence and incidence based on occurrence of health-related events. *Statistics in Medicine* 24: 3139–3154.

About the author

Henrik Støvring is a senior researcher at the Research Unit for General Practice, University of Southern Denmark, Odense, Denmark.