Copy No. _____

Defence Research and Development Canada
Recherche et développement pour la défense Canada

DEFENCE R&D DÉFENSE

C++ classes for representing curves and surfaces

Part IV: Distribution functions

David Hally

Defence R&D Canada – Atlantic

Technical Memorandum
DRDC Atlantic TM 2006-257
January 2007

Canada

This page intentionally left blank.

C++ classes for representing curves and surfaces

Part IV: Distribution functions

David Hally

Defence R&D Canada – Atlantic

Technical Memorandum

DRDC Atlantic TM-2006-257

January 2007

Principal Author

Original signed by David Hally

David Hally

Approved by

Original signed by R. Kuwahara

R. Kuwahara
Head/Signatures

Approved for release by

Original signed by K. Foster

K. Foster
Chair/Document Review Panel

© Her Majesty the Queen in Right of Canada as represented by the Minister of National Defence, 2007

© Sa Majesté la Reine (en droit du Canada), telle que représentée par le ministre de la Défense nationale, 2007

Abstract

This document describes C++ classes which implement distribution functions: strictly increasing differentiable functions which map $[0, 1]$ to $[0, 1]$ and are one-to-one and onto. A distribution function is useful whenever one wishes to redistribute values within a given range without changing their order; however, their primary use is in generating distributions of nodes in grids used for solving differential equations.

The distribution classes are based on the more general CurveLib library for representing multi-parameter differentiable functions. From the CurveLib classes they inherit arithmetic and composition operators that allow the distribution functions to be combined in complex ways.

Résumé

Le présent document décrit des classes C++ qui intègrent des fonctions de distribution, lesquelles sont des applications différentiables, strictement croissantes et bijectives de [0, 1] à [0, 1]. Ces fonctions sont utiles lorsque l'on veut redistribuer des valeurs dans un intervalle donné sans changer leur ordre. Toutefois, elles sont surtout employées pour produire des distributions de nœuds dans des réseaux utilisés pour la résolution d'équations différentielles.

Les classes de distributions sont fondées sur la bibliothèque plus générale CurveLib servant à représenter les fonctions différentiables multiparamétriques. Ces classes héritent des classes CurveLib, des opérateurs arithmétiques et de composition qui permettent de combiner de façon complexe les fonctions de distribution.

This page intentionally left blank.

Executive summary

C++ classes for representing curves and surfaces: Part IV: Distribution functions

David Hally; DRDC Atlantic TM-2006-257; Defence R&D Canada – Atlantic; January 2007.

Background: The flow around ships and propellers affects their performance in many ways. Defence R&D Canada – Atlantic uses Computational Fluid Dynamics (CFD) to calculate these flows so that the performance of the hull and propellers can be evaluated and improved. Before the flow can be calculated, computational grids must be created that conform to the surfaces of the ship hull or propeller. The current document describes a library of C++ classes which aid in the creation of the computational grid.

Principal results: A library of C++ classes for representing distribution functions has been developed. These classes can be used to distribute the nodes along the edges of a computational grid in a variety of ways. The classes are based on the more general CurveLib library of classes for representing multi-parameter differentiable functions (described in a companion report).

Significance: The library of C++ classes provides a useful tool that can be used when creating computational grids for use in CFD programs. However, their definition is quite general, so they can also be used in a wide variety of applications. In particular, they can be used whenever the one wishes to alter the parameterization of a differentiable function represented using the CurveLib library.

Sommaire

C++ classes for representing curves and surfaces: Part IV: Distribution functions

David Hally ; DRDC Atlantic TM-2006-257 ; R & D pour la défense Canada – Atlantique ; janvier 2007.

Contexte : L'écoulement de l'eau autour des navires et de leurs hélices influence leur comportement de différentes manières. R & D pour la défense Canada – Atlantique utilise la dynamique numérique des fluides pour calculer ces écoulements et ainsi évaluer et améliorer le comportement des carènes et des hélices. Avant de procéder au calcul de l'écoulement, on doit créer des réseaux de calcul qui épousent la surface de la carène ou de l'hélice. Le présent document présente une bibliothèque de classes C++ qui aideront à la création de ces réseaux.

Résultats : Nous avons élaboré une bibliothèque de classes C++ regroupant les fonctions de distribution. On pourra utiliser ces classes pour distribuer selon différentes modalités les nœuds le long des arêtes du réseau de calcul. Ces classes ont été produites à partir de la bibliothèque plus générale de classes C++ CurveLib, qui permet de représenter des fonctions multiparamétriques continûment différentiables (décrites dans un rapport connexe).

Importance : Cette bibliothèque de classes C++ est un outil précieux dont on pourra tirer parti pour créer les réseaux de calcul des logiciels de dynamique numérique des fluides. Puisque la définition de fonctions est très générale, on pourra les appliquer à de nombreux problèmes. En particulier, on pourra les utiliser pour modifier la paramétrisation d'une fonction différentiable représentée à l'aide de la bibliothèque CurveLib.

Table of contents

List of figures

1 Introduction

A distribution function is a strictly increasing differentiable function which maps $[0, 1]$ to $[0, 1]$ and is one-to-one and onto. A distribution is useful whenever one wishes to redistribute values within a given range without changing their order.

This document describes a library of C++ classes representing various types of distribution functions. The classes are based on the CurveLib library of classes for representing differentiable curves[1].

Although distributions are useful for many different applications, the one that has prompted the current work is the distribution of nodes along an edge in a computational grid. This application is discussed further in Section 2.

2 Node distribution functions

Distribution functions can be used to distribute the nodes of computational grids for solvers of differential equations. Suppose N nodes are to be distributed over the region $[a, b]$. If $f(x)$ is a distribution, then we define:

$$x_i = a + (b - a) f\left(\frac{i}{N-1}\right) \tag{1}$$

where i goes from 0 to $N - 1$. Because $f(x)$ is strictly increasing, $x_i < x_j$ if $i < j$, a property that is usually necessary for a well-defined computational grid.

2.1 Node spacing

The spacing between two nodes is

$$x_{i+1} - x_i = (b - a)\left[f\left(\frac{i+1}{N-1}\right) - f\left(\frac{i}{N-1}\right)\right] \approx \frac{b-a}{N-1} f'\left(\frac{i}{N-1}\right) \tag{2}$$

Therefore the derivative of the distribution is roughly proportional to the cell size. To obtain a node distribution having node spacing h_0 at $i = 0$ and h_1 at $i = N - 1$, we need a distribution such that

$$f'(0) = \frac{(N-1)h_0}{b-a}; \qquad f'(1) = \frac{(N-1)h_1}{b-a} \tag{3}$$

2.2 Growth rate

The ratio of the sizes of neighbouring cells, the growth rate, is given by:

$$r_i \equiv \frac{x_{i+1} - x_i}{x_i - x_{i-1}} = \frac{f\left(\frac{i+1}{N-1}\right) - f\left(\frac{i}{N-1}\right)}{f\left(\frac{i}{N-1}\right) - f\left(\frac{i-1}{N-1}\right)} \approx 1 + \frac{1}{(N-1)}\left(\frac{f''}{f'}\right)\left(\frac{i}{N-1}\right) \tag{4}$$

We will call f''/f' the growth rate function. If it is positive, the cell sizes get progressively larger; if it is negative, the cell sizes get progressively smaller.

To minimize errors and increase the stability of numerical solutions of differential equations, it is usually desirable to keep the growth rate as small as possible.

2.3 Reversed distributions

Let $g(x)$ be any distribution. Then

$$f(x) = 1 - g(1 - x) \tag{5}$$

is also a distribution. Notice that

$$f'(x) = g'(1 - x) \tag{6}$$

Therefore the spacing of the nodes generated by $f(x)$ is reversed relative to the spacing of the nodes generated by $g(x)$. In particular, the slopes at the end-points are reversed:

$$f'(0) = g'(1); \qquad f'(1) = g'(0) \tag{7}$$

A distribution for which the reverse is the same as the original distribution is said to be *invariant under reversal*.

Many of the distributions that will be discussed in the following sections are specified by assigning values of their end slopes, s_0 and s_1. It is usually very desirable that when the role of the end slopes is interchanged (i.e. s_0 is replaced by s_1 and vice versa), then the distribution is reversed. This implies that both ends of the distribution are treated in exactly the same way. Distributions with this property are said to have the *reversibility property*.

2.4 Inverse distributions

Let $f(x)$ be any distribution and let $f^{(-1)}(x)$ be its inverse: i.e.

$$f^{(-1)}\big(f(x)\big) = x \tag{8}$$

Then $f^{(-1)}(x)$ is also a distribution since $f^{(-1)}(0) = 0$, $f^{(-1)}(1) = 1$, and the inverse of a strictly increasing function is also strictly increasing. When $f(x)$ is used to distribute nodes according to Equation (1), then $f^{(-1)}$ provides a means of calculating the node number at a given value of x in $[a, b]$:

$$i = (N-1) f^{(-1)} \left(\frac{x-a}{b-a} \right) \tag{9}$$

Since

$$f^{(-1)'}(x) = \frac{1}{f'\big(f^{(-1)}(x)\big)} \tag{10}$$

the end slopes of the inverse distribution are the reciprocals of the end slopes of the original distribution:

$$f^{(-1)'}(0) = \frac{1}{f'(0)}; \qquad f^{(-1)'}(1) = \frac{1}{f'(1)} \tag{11}$$

If the growth rate function for the original distribution is $r(x)$, then the growth rate function of the inverse is:

$$r_{\text{inv}}(x) = -r\big(f^{(-1)}(x)\big) f^{(-1)'}(x) \tag{12}$$

2.5 The composition of distributions

Suppose $f(x)$ and $g(x)$ are distributions. Then $(f \circ g)(x) \equiv f\big(g(x)\big)$ is a distribution. The slopes at the end points of $(f \circ g)(x)$ are:

$$(f \circ g)'(0) = f'(0) g'(0); \qquad (f \circ g)'(1) = f'(1) g'(1) \tag{13}$$

Therefore, at the end points, the node spacing obtained from the composition of two distributions is proportional to the product of the node spacing obtained from each distribution separately.

The growth rate function of $f \circ g$ is

$$r_{f \circ g}(x) = r_f\big(g(x)\big) g'(x) + r_g(x) \tag{14}$$

where $r_f(x)$ denotes the growth rate of distribution $f(x)$.

The inverse of $f \circ g$ is

$$(f \circ g)^{(-1)} = g^{(-1)} \circ f^{(-1)} \tag{15}$$

3 C++ implementation of distributions

Distributions are implemented in C++ using the CurveLib library of classes for representing differentiable functions[1]. All classes used to implement distributions are included in the namespace `Distrib`.

The class `Distribution<F>` is a template base class for distributions. Its template argument, `F`, is the type of its parameter. All of the classes derived from `Distribution<F>` in namespace `Distrib` have a similar template argument. Usually `F` is either `float` or `double` but any type with the following characteristics can be used:

1. It is a model of a Comparable Scalar Object (see Reference 1, Annex A.3).
2. The elementary functions `exp(F)`, `log(F)`, `sqrt(F)`, `pow(F,F)`, `sin(F)`, `cos(F)`, `tan(F)`, `sinh(F)`, `cosh(F)`, `tanh(F)` and `atanh(F)` are defined. Each of these functions should return an `F`.
3. The inserter `operator<<(std::ostream&,const F&)` is defined and returns a `std::ostream&`.
4. `std::numeric_limits<F>` is defined.

If the template argument is omitted, it defaults to `double`: i.e. `Distribution<>` is equivalent to `Distribution<double>`. The same is true for all the classes derived from `Distribution<F>` described in this report.

The class `Distribution<F>` is derived from the base class `Curve<1U,F,F>` in namespace `CurveLib`: see Reference 1, Section 2. Instances of `Distribution<F>` are evaluated using standard operator syntax defined by `Curve<1U,F,F>`.

```
using namespace Distrib;
Distribution<double> f;
double x = 0.5;
double y = f(x); // Sets y to the value of f at x.
double dy = f(x,1); // Sets y to the 1st derivative of f at x.
double d2y = f(x,2); // Sets y to the 2nd derivative of f at x.
```

A `Distribution<F>` can be made by composing two instances of `Distribution<F>`:

```
using namespace Distrib;
GeometricDistribution<double> gd(0.1);
TanhDistribution<double> td(0.1,0.2));
Distribution<double> f = gd(td); // f(x) = dg(td(x));
```

A `Distribution<F>` may also be composed with a `Curve<1U,V,F>` to yield another curve of the same type. This operation is the normal way to distribute nodes along an edge of a block. Suppose that `edge` is an instance of `CurveLib::Curve<1U,Vec3,F>` where `Vec3` is the type of a three-dimensional vector. As its parameter varies between 0 and 1, `edge` traces a curve in space. Nodes can be defined by sampling values of `edge`:

```
int n = 11;
std::vector<Vec3> nodes;
for (int i = 0; i < n; ++i) {
  F x = i/F(n-1);
  Vec3 v = edge(x);
  nodes.push_back(v);
}
```

To redistribute the nodes, we can compose `edge` with a distribution. For example, if `f` is an instance of `Distribution<F>`, then the redistributed nodes can be defined by:

```
CurveLib::Curve<1U,Vec3,F> new_edge = edge(f);
std::vector<Vec3> nodes;
int n = 11;
for (int i = 0; i < n; ++i) {
  F x = i/F(n-1);
  Vec3 v = new_edge(x);
  nodes.push_back(v);
}
```

The class `Distribution<F>` and all classes derived from it have default constructors (i.e. constructors with no arguments). When the default constructor is invoked the normal behaviour is that the distribution remains undefined; attempting to evaluate it while it is undefined will result in an `Error` exception being thrown (see Reference 1, Annex F for a description of `Error` exceptions). When describing the member functions of a distribution class, we will usually omit the default constructor, the destructor and the assignment operator.

Figure 1 is an inheritance diagram for all the classes in the namespace `Distrib` that are derived from `Distribution<F>`. The inherited classes are described in the following sections.

`Distribution<F>` defines the following public member functions in addition to the default and copy constructors and the assignment operator:

`bool is_defined() const`
: Returns true if the distribution is defined; false otherwise.

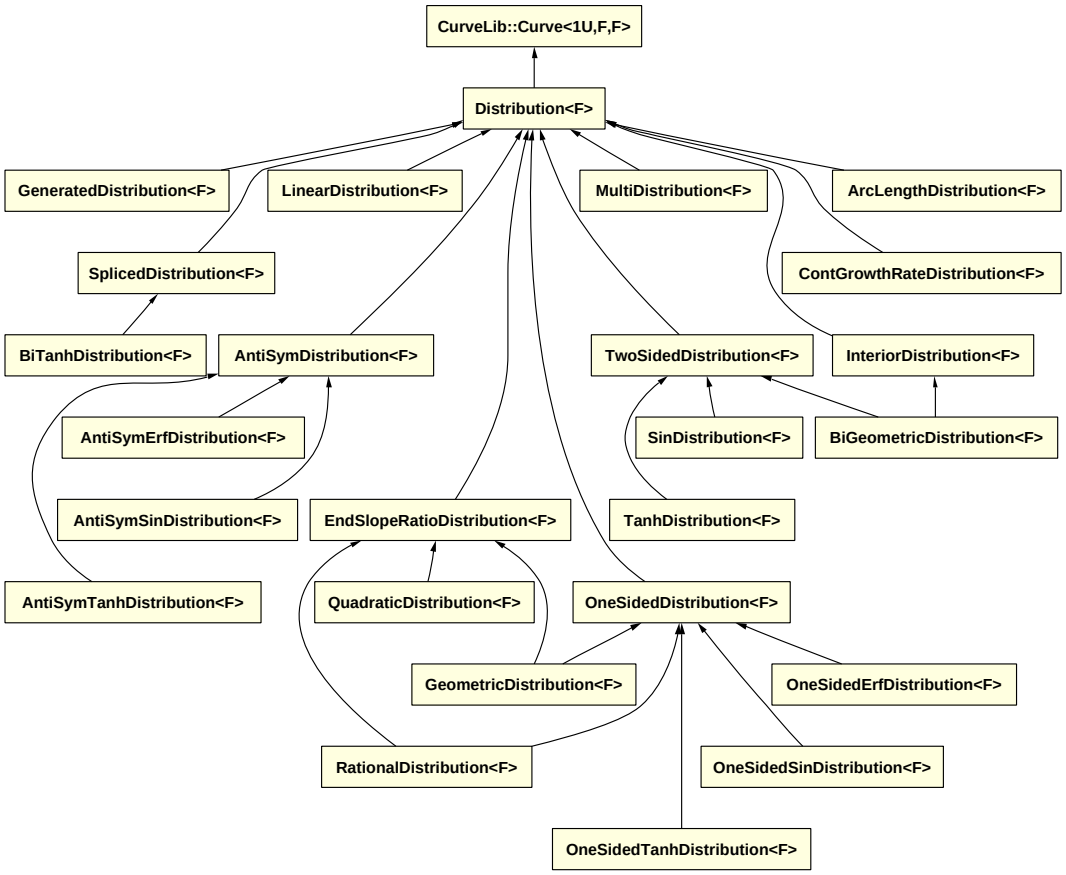

Figure 1: *Inheritance diagram for all classes derived from* `Distribution<F>`

`F operator()(F x) const`
: Returns the value of the curve at `x`. The value at 0 is guaranteed to be exactly 0 and the value at 1 is guaranteed to be exactly 1 (i.e. the values at the end-points are calculated with no round-off error.

`F operator()(F x, unsigned d) const`
: Returns the value of the $\texttt{d}^{\text{th}}$ derivative of the distribution at `x`.

`F operator()(const ParamType &p) const`
: Returns the value of the distribution at `p`. `ParamType` is the type of a one element vector defined by the base class `Curve<1U,F,F>` in namespace `CurveLib`. This function is equivalent to `operator()(p[0])`. It is defined because it is inherited from the base class, but is rarely used because it is so much more convenient to use the version whose argument is an `F`.

`F operator()(const ParamType &p, const DerivType &d) const`
: Returns the value of the differentiated curve at `p`. `DerivType` is the type of a one element vector of unsigned integers defined by the base class `Curve<1U,F,F>`. This function returns the same value as `operator()(p[0],d[0])`. It is defined because it is inherited from the base class, but is rarely used because it is so much more convenient to use the version whose arguments are of type `F` and `unsigned`.

`Distribution<F> operator()(const Distribution<F> &d) const`
: The composition operator. If the current distribution is $f(x)$, this function returns a distribution equivalent to $f \circ d$.

`Distribution<F> reverse() const`
: Returns the reverse distribution: i.e. if the distribution is $f(x)$, then a distribution equivalent to $1 - f(1 - x)$ is returned.

`Distribution<F> inverse() const`
: Returns the inverse distribution. The default implementation of this function uses `FInverseCurve<F>` from namespace `CurveLib` (see Reference 1, Section 11.3) to generate the inverse distribution. It will only be accurate to about 10^{-6}: i.e. if the current distribution is $f(x)$, then $|f\big(f^{(-1)}(x)\big) - x|$ is only guaranteed to be less than 10^{-6}, whereas it should be zero. Derived classes will often reimplement `inverse()` to return a more efficient or more accurate inverted distribution.

An important property of a `Distribution<F>` is that it is polymorphic, even though it has no virtual functions. For example, suppose a `TanhDistribution<>` is assigned to a `Distribution<>`:

```
Distribution<> d;
TanhDistribution<> td(0.1,0.2);
d = td;
```

When evaluated with the same arguments, `d` will return the same value as `td`. Moreover, both `d.inverse()` and `td.inverse()` return an inverse distribution that is generating using the known properties of the tanh distribution (see Section 11.3). Similarly with `d.reverse()` and `td.reverse()`.

4 Linear distribution

The simplest distribution is the linear distribution or identity function:

$$f(x) = x \tag{16}$$

Its growth rate is 1.0 and its growth rate function is $r(x) = 0$. Its reverse and inverse are both the same as the distribution itself.

The C++ class `LinearDistribution<F>`, derived from `Distribution<F>`, is used to represent a linear distribution. Its only constructor has no arguments. It defines no new member functions beyond those inherited from `Distribution<F>`.

5 Generating functions

Suppose that $g(x)$ is a strictly increasing differentiable function. Then

$$f(x) = \frac{g\big(a + x(b-a)\big) - g(a)}{g(b) - g(a)} \tag{17}$$

is a distribution for any a and b such that $[a, b]$ is in the domain of $g(x)$. The function $g(x)$ is called a generating function with domain $[a, b]$ for the distribution f. Clearly each generating function $g(x)$ generates a unique distribution $f(x)$ for each choice of a and b. However, any distribution can have many different generating functions. In particular, it is easily verified that any distribution is its own generating function with the choice $a = 0$ and $b = 1$.

Suppose that a sequence of nodes, y_i, is generated by sampling $g(x)$ at N points equally spaced between a and b:

$$y_i = g\big(a + x_i(b-a)\big); \qquad x_i = \frac{i}{N-1} \tag{18}$$

The same sequence of nodes is generated using $f(x)$ between $g(a)$ and $g(b)$:

$$y_i = g(a) + \big(g(b) - g(a)\big) f(x_i) \tag{19}$$

The slope of f at x is

$$f'(x) = \frac{(b-a)g'\big(a + x(b-a)\big)}{g(b) - g(a)} \tag{20}$$

Notice that in the limit that b approaches a the derivative approaches one everywhere and the distribution becomes the linear distribution.

The reverse of $f(x)$ is

$$1 - f(1-x) = \frac{g\big(b + x(a-b)\big) - g(b)}{g(a) - g(b)} \tag{21}$$

That is, the reverse of the distribution is obtained by swapping the roles of a and b; however, since $a < b$, we cannot say that the reverse is the distribution generated by $g(x)$ over the domain $[b, a]$. Nevertheless, if $g(x)$ is antisymmetric (i.e. $g(-x) = -g(x)$), then it is true that the reverse is the distribution generated by $g(x)$ over the domain $[-b, -a]$ and the distribution will have the reversibility property. For this reason all the generating functions considered in the following sections are antisymmetric.

The inverse of $f(x)$ is

$$f^{(-1)}(x) = \frac{g^{(-1)}\Big(g(a) + x\big(g(b) - g(a)\big)\Big) - a}{b - a} \tag{22}$$

which is the distribution generated by $g^{(-1)}(x)$ on the domain $[g(a), g(b)]$.

When evaluating $f(x)$, care must be taken when a and b are very close to one another since the numerator and denominator of Equation (17) will both be very close to zero resulting in a loss of precision. In this case we can use a Taylor expansion about $(a+b)/2$. Defining $x_m = (a+b)/2$ and $\Delta = b - a$ we get:

$$\begin{aligned} f(x) &= \frac{g\big(x_m + \Delta(x - \frac{1}{2})\big) - g(x_m - \frac{1}{2}\Delta)}{g(x_m + \frac{1}{2}\Delta) - g(x_m - \frac{1}{2}\Delta)} \\ &= x + \frac{g''(x_m)\Delta}{g'(x_m)} x(x-1) + O\left(\frac{\sqrt{3} g'''(x_m)\Delta^2}{216 g'(x_m)}\right) \end{aligned} \tag{23}$$

5.1 C++ implementation of generated distributions

The C++ class `GeneratedDistribution<F>`, derived from `Distribution<F>`, is used to represent a generated distribution. It has a single constructor besides the default constructor.

`GeneratedDistribution(const CurveLib::Curve<1U,F,F> &gcrv,`
`F a, F b)`

Makes a generated distribution from the strictly increasing function `gcrv` between the values `a` and `b`.

It also defines the following member function which can be used to define the distribution when the default constructor has been used:

`void define(const CurveLib::Curve<1U,F,F> &gcrv, F a, F b)`
: Defines the distribution to have generating function `gcrv` between the values `a` and `b`.

6 Antisymmetric distributions

An antisymmetric distribution is one which is antisymmetric about the point $(\frac{1}{2}, \frac{1}{2})$. The derivative of the distribution is symmetric about $x = \frac{1}{2}$ so that if the distribution is used to generate a series of nodes, the nodes will be symmetrically distributed about their mid-point.

Suppose $g(x)$ is a generating function with domain $[a, b]$ for the distribution $f(x)$. If $g(-x) = g(x)$ and if $a = -b$, then $f(x)$ is an antisymmetric distribution given by:

$$f(x) = \frac{1}{2}\left[1 + \frac{g\big((2x-1)b\big)}{g(b)}\right] \tag{24}$$

Antisymmetric distributions have the important property that the slopes at the end-points are the same:

$$s \equiv s_0 = s_1 = \frac{bg'(b)}{g(b)} \tag{25}$$

In fact, it is easily verified that an antisymmetric distribution is invariant under reversal. Moreover, since the end slopes are the same, it has the reversibility property.

The growth rate of an antisymmetric distribution is

$$r(x) = \frac{2bg''\big((2x-1)b\big)}{g'\big((2x-1)b\big)} \tag{26}$$

and

$$r(1) = -r(0) = \frac{2bg''(b)}{g'(b)} \tag{27}$$

For the generating functions that we will consider in the following sections, $g''(x)$ does not change sign when $x > 0$. This implies that when $s > 1$, the maximum slope for the distribution occurs at the end points and the minimum slope occurs at $x = \frac{1}{2}$; similarly if $s < 1$, the minimum slope occurs at the end points and the maximum slope occurs at $x = \frac{1}{2}$. The slope at $x = \frac{1}{2}$ is

$$f'\big(\tfrac{1}{2}\big) = \frac{bg'(0)}{g(b)} \tag{28}$$

Since $g(0) = 0$, the slope of the straight line between $(0, g(0))$ and $\big(x, g(x)\big)$ is $g(x)/x$. By the intermediate value theorem, this is equal to $g'(y)$ for some $y \in [0, x]$. Now, if $g''(x) > 0$ for all $x > 0$, then the maximum value of $g'(y)$ for $y \in [0, x]$ occurs when $y = x$. Therefore, if $g''(x) > 0$, then $g'(x) \geq g(x)/x$ and Equation (25) will only have a real solution when $s \geq 1$. Similarly, if $g''(x) < 0$ for all x, Equation (25) will only have a real solution when $s \leq 1$. However, if we admit imaginary values for b, the Equation (25) will have a solution for any real positive s. For the cases when b is imaginary we define $\gamma = -ib$ and $h(x) = -ig(ix)$. Notice that since g is an odd function of x, h is a real function. Now the distribution becomes:

$$f(x) = \frac{1}{2}\left[1 + \frac{h\big((2x-1)\gamma\big)}{h(\gamma)}\right] \tag{29}$$

and Equation (25) is

$$s \equiv s_0 = s_1 = \frac{\gamma h'(\gamma)}{h(\gamma)} \tag{30}$$

In other words, we can simply treat the distribution as being generated by $h(x)$ over $[-\gamma, \gamma]$ instead of $g(x)$ over $[-b, b]$.

6.1 Antisymmetric sin distribution

The antisymmetric sin distribution uses $\sin(x)$ as the generating function. Equation (25) then becomes

$$\frac{\tan(b)}{b} = \frac{1}{s} \tag{31}$$

If s is specified, this equation can be solved to determine b. However, s cannot exceed 1.0 since $\tan(x)/x \geq 1$ for all x. An efficient algorithm for solving Equation (31) is discussed in Annex A.2.

The growth rate function is

$$r(x) = -b\tan\big((2x-1)b\big) \tag{32}$$

If s is very small, $b \sim \pi/2$ and $r(0) \approx \pi^2/4s$. Therefore the antisymmetric sin distribution has a growth rate which can be large when s is very small.

The maximum slope for the antisymmetric sin distribution occurs when $x = \frac{1}{2}$:

$$s_{\max} = \frac{b}{\sin(b)} \tag{33}$$

It cannot exceed $\pi/2$, though it approaches that value as $s \to 0$ and $b \to \pi/2$.

In passing we note that when $b = \pi/2$, the sin distribution becomes

$$f(x) = \frac{1}{2}\big(1 - \cos(\pi x)\big) \tag{34}$$

which is normally referred to as a cos distribution. In fact, since the slopes at 0 and 1 are both zero, this is not strictly a distribution as defined for the purposes of this report.

When $s > 1$, we use $-i\sin(ix) = \sinh(x)$ as the generating function. In this case Equation (25) is

$$\frac{\tanh(b)}{b} = \frac{1}{s} \tag{35}$$

An efficient algorithm for solving Equation (35) is discussed in Annex A.3.

The growth rate function when $s > 1$ is

$$r(x) = 2b\tanh\big((2x-1)b\big) \tag{36}$$

If s is very large, $b \approx s$ and $r(0) \approx s$. Therefore the growth rate can be large if s is large.

The minimum slope for the sinh distribution occurs when $x = \frac{1}{2}$:

$$s_{\min} = \frac{b}{\sinh(b)} \tag{37}$$

When s is large, $s_{\min} \approx se^{-s}$. Because the minimum slope is exponentially small, the sinh distribution is usually a poor choice.

6.2 Antisymmetric tanh distribution

Consider the antisymmetric distribution with the generating function $g(x) = \tanh(x)$. Equation (25) then becomes

$$\frac{\sinh(2b)}{2b} = \frac{1}{s} \tag{38}$$

Since $\sinh(x)/x \geq 1$ for all x, this distribution can only be used when $s \leq 1$. An efficient algorithm for solving Equation (38) is discussed in Annex A.5.

The growth rate function is

$$r(x) = -4b\tanh\big((2x-1)b\big) \tag{39}$$

At the end points the growth function can be written:

$$r(0) = -r(1) = 4b\tanh(b) = 2\big(\sqrt{s^2 + 4b^2} - s\big) \tag{40}$$

If s is very small, $b \sim -\frac{1}{2}\ln(s)$ and $r(0) \sim -2\ln(s)$. Therefore the growth rate of the antisymmetric tanh distribution remains reasonably small even when s is very small. It is clearly to be preferred over the antisymmetric sin distribution when $s_1 < 1$.

The maximum slope for the tanh distribution is

$$s_{\max} = \frac{b}{\tanh(b)} = \tfrac{1}{2}\big(s + \sqrt{s^2 + 4b^2}\,\big) \tag{41}$$

If s is very small, $s_{\max} \sim -\frac{1}{2}\ln(s)$, so that the slope of the distribution is also reasonably well bounded.

When $s > 1$, we use $-i\tanh(ix) = \tan(x)$ as the generating function. In this case Equation (25) is

$$\frac{\sin(2b)}{2b} = \frac{1}{s} \tag{42}$$

An efficient algorithm for solving Equation (42) is discussed in Annex A.4.

The growth rate function when $s > 1$ is

$$r(x) = 4b\tan\big((2x-1)b\big) \tag{43}$$

At the end points the growth rate is

$$r(1) = -r(0) = 4b\tan(b) = \begin{cases} 2\big(s - \sqrt{s^2 - 4b^2}\,\big) & \text{if } s \le \pi/2 \\ 2\big(s + \sqrt{s^2 - 4b^2}\,\big) & \text{if } s \ge \pi/2 \end{cases} \tag{44}$$

If s is very large, $b \sim \pi/2$ and $r(1) \approx 4s$. Therefore the growth rate at the end points can be large when s is large.

The minimum slope when $s > 1$, occurring when $x = \frac{1}{2}$, is

$$s_{\min} = \frac{b}{\tan(b)} \tag{45}$$

When s is large, $s_{\min} \approx \pi^2/4s$.

6.3 Antisymmetric erf distribution

The antisymmetric erf distribution uses the error function, $\operatorname{erf}(x)$, as the generating function:

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}}\int_0^x e^{-x^2}\,dx \tag{46}$$

Equation (25) then becomes

$$\frac{\sqrt{\pi}e^{b^2}\operatorname{erf}(b)}{2b} = \frac{1}{s} \tag{47}$$

Since the left hand side is greater than 1 for all x, this distribution can only be used when $s \leq 1$. An efficient algorithm for solving this equation is discussed in Annex A.7.

The growth rate function is

$$r(x) = -4b^2(2x - 1) \tag{48}$$

Therefore the magnitude of the growth rate is strictly bounded by $4b^2$.

The maximum slope for the erf distribution is

$$s_{\max} = \frac{2b}{\sqrt{\pi}\,\mathrm{erf}(b)} \tag{49}$$

If s is very small, $s_{\max} \sim 2\sqrt{-\ln(s)/\pi}$, so that the slope of the distribution is also reasonably well bounded.

When $s > 1$, we could use

$$h(x) = -i\mathrm{erf}(ix) = \frac{2}{\sqrt{\pi}} \int_0^x e^{x^2}\, dx \tag{50}$$

as the generating function. This distribution has not been implemented in the C++ classes described here due to the lack of a suitable means of evaluating $h(x)$.

6.4 C++ implementation of antisymmetric distributions

`AntiSymDistribution<F>` is a base class for classes representing antisymmetric distributions. It is derived from the base class `Distribution<F>`. As it has only the default constructor, it cannot, of itself, be used to define an antisymmetric distribution.

`AntiSymDistribution<F>` defines the following member function:

`void set_end_deriv(F s)`
: Sets the derivatives at 0 and 1 to `s`.

`AntiSymDistribution<F>` also uses the fact that antisymmetric distributions are invariant under reversal to reimplement the member function `reverse()`. The distribution returned by `reverse()` is equivalent to the distribution itself; it will evaluate more efficiently than the reverse distribution returned by `Distribution<F>`.

The antisymmetric sin and sinh distributions are represented in C++ by the single class `AntiSymSinDistribution<F>` derived from `AntiSymDistribution<F>`. It has the following constructor:

`AntiSymSinDistribution(F s = 1)`
: Makes a antisymmetric distribution whose end slopes are `s`. If `s` is greater than 1, the generating function will be $\sinh(x)$; otherwise it will be $\sin(x)$.

All other member functions are inherited from the base classes.

The antisymmetric tanh and tan distributions are represented in C++ by the single class `AntiSymTanhDistribution<F>` derived from `AntiSymDistribution<F>`. It has the following constructor:

`AntiSymTanhDistribution(F s = 1)`
: Makes a antisymmetric distribution whose end slopes are `s`. If `s` is greater than 1, the generating function will be $\tan(x)$; otherwise it will be $\tanh(x)$.

All other member functions are inherited from the base classes.

The class `AntiSymErfDistribution<F>` represents an antisymmetric erf distribution. It is derived from the base class `AntiSymDistribution<F>` and has the following constructor:

`AntiSymErfDistribution(F s = 1)`
: Makes a antisymmetric erf distribution whose end slopes are `s`; `s` must not exceed 1.

All other member functions are inherited from the base classes.

7 End slope ratio distributions

An end slope ratio distribution is a distribution that can be uniquely defined given the ratio of the slopes at its end points, $\beta \equiv s_1/s_0$. As we shall see in Section 11, end slope ratio distributions can be used to convert antisymmetric distributions into distributions which can be specified by setting arbitrary slopes at each end point.

Suppose that $f(x)$ is an end slope distribution with end slope ratio β. Its inverse has end slopes $1/s_0$ and $1/s_1$ (see Section 2.4). Therefore the inverse distribution can also be considered to be an end slope distribution having end slope ratio $1/\beta$.

For end slope ratio distributions, having the reversibility property implies that the reverse of the distribution is the same as the distribution specified using end slope ratio $1/\beta$.

7.1 Quadratic distributions

A quadratic distribution is defined by

$$f(x) = (1 - \alpha)x + \alpha x^2 \tag{51}$$

for $\alpha \in (-1, 1)$. The slopes at the end points of a quadratic distribution are

$$f'(0) = 1 - \alpha; \qquad f'(1) = 1 + \alpha \tag{52}$$

They must be in the range $(0, 2)$. Because the range of end slopes is so limited, quadratic distributions are not very useful as one-sided distributions (distributions defined by setting the slope at one end: see Section 8). However, they can be used as end slope ratio distributions since the ratio of end slopes can have unlimited values:

$$\alpha = \frac{\beta - 1}{\beta + 1}; \qquad \beta = \frac{1 + \alpha}{1 - \alpha} \tag{53}$$

The growth rate function of a quadratic distribution is:

$$r(x) = \frac{2\alpha}{1 - \alpha + 2\alpha x} \tag{54}$$

If α is close to 1, the growth rate is very large near $x = 0$; conversely, if α is close to -1, the growth rate is very large near $x = 1$.

The reverse of a quadratic distribution is

$$f(x) = (1 + \alpha)x - \alpha x^2 \tag{55}$$

Therefore the reverse distribution is the quadratic distribution with α replaced with $-\alpha$ or by replacing β with $1/\beta$: the quadratic distribution has the reversibility property.

The inverse distribution is

$$f^{(-1)}(x) = \frac{\alpha - 1 + \sqrt{(1 - \alpha)^2 + 4\alpha x}}{2\alpha} \tag{56}$$

7.2 C++ implementation of end slope ratio distributions

`EndSlopeRatioDistribution<F>` is a base class representing end slope ratio distributions. It is derived from `Distribution<F>`. As it has only the default constructor, it cannot, of itself, be used to define an end slope ratio distribution.

`EndSlopeRatioDistribution<F>` defines the following member function:

`F get_slope_ratio() const`
: Returns the end slope ratio β.

`void set_slope_ratio(F beta)`
: Sets the end slope ratio to `beta`.

`bool has_reversibility_property() const`
: Returns true if the distribution is known to have the reversibility property. Note that it is possible that this function will return false even when the distribution does have the reversibility property.

When the distribution is known to have the reversibility property, `EndSlopeRatioDistribution<F>` also uses the fact that the reverse of the distribution has end slope ratio $1/\beta$ to reimplement the member function `reverse()`. The distribution returned by `reverse()` will evaluate more efficiently than the reverse distribution returned by `Distribution<F>`.

The class `QuadraticDistribution<F>` represents a quadratic distribution. It is derived from the base class `EndSlopeRatioDistribution<F>` and has the following constructor:

`QuadraticDistribution(F beta = 1)`
: Makes a quadratic distribution with end slope ratio `beta`.

All other member functions are inherited from the base classes.

Sections 9.1 and 10.1 describe other distribution classes derived from the class `EndSlopeRatioDistribution<F>`.

8 One-sided distributions

A one-sided distribution is a distribution for which the slope, s, can be set at one end point but not both. They are useful for generating node distributions in unbounded domains where the exact cell spacing far from boundaries is not important.

For a one-sided distribution, having the reversibility property implies that the reverse of the distribution is the same as the distribution obtained by specifying the slope at the other end point to be s.

Figure 2 plots four different one-sided distributions, their slopes and their growth rates. For each the slope at 0 is set to 0.1. The distributions are described in the following sections.

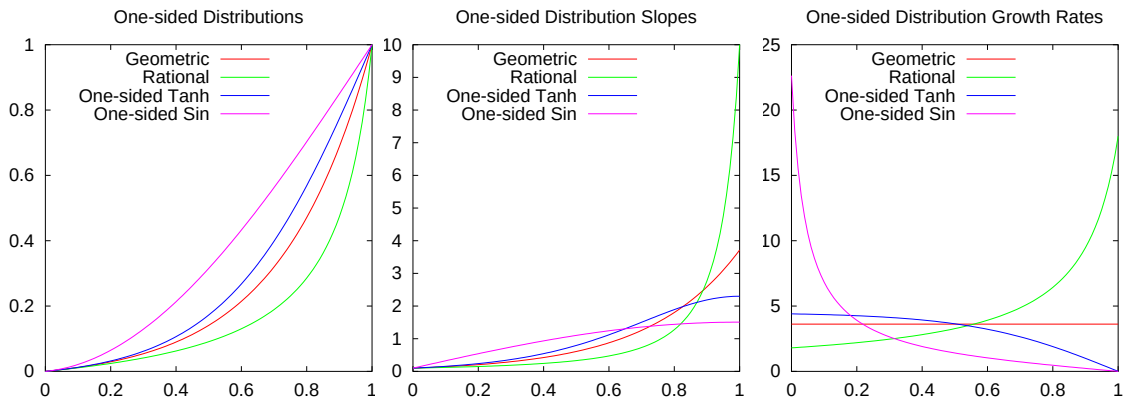

Figure 2: *Four one-sided distributions, their slopes and growth rates: the slope at 0 is 0.1.*

8.1 One-sided distributions from generating functions

A one-sided distribution with slope set at $x = 1$ can be generated from an antisymmetric generating function with domain $[0, b]$:

$$f(x) = \frac{g(bx)}{g(b)} \tag{57}$$

The slope at $x = 1$ is

$$s = \frac{bg'(b)}{g(b)} \tag{58}$$

which must be solved for b; this is the same as Equation (25). The growth rate is

$$r(x) = \frac{bg''(bx)}{g'(b)} \tag{59}$$

which is one half Equation (26) with $2x - 1$ replaced by x. Notice that $r(0) = 0$ since $g(x)$ is antisymmetric.

Suppose that $h(x)$ is the antisymmetric distribution which has $g(x)$ as its generating function over domain $[-b, b]$. Then

$$f(x) = 2h\left(\tfrac{1}{2}(x + 1)\right) - 1 \tag{60}$$

Therefore the properties of this distribution are similar to those of the antisymmetric distribution which uses the same generating function $g(x)$. In particular, when the specified slope is less than 1, the one-sided distribution derived from the tanh distribution is to be preferred over the one-sided sin distribution if the growth rate is to be kept small.

Similarly, we get a one-sided distribution with slope set at $x = 0$ by using an antisymmetric generating function with domain $[-b, 0]$:

$$f(x) = 1 - \frac{g\big(b(1-x)\big)}{g(b)} \tag{61}$$

As should have been expected, this is just the distribution of Equation (57) reversed: the distribution has the reversibility property. Therefore the equation to be solved for b is still Equation (58) except that s is now interpreted as the slope at $x = 1$. This distribution can be written in terms of the antisymmetric distribution $h(x)$ as

$$f(x) = 2h\big(\tfrac{1}{2}x\big) \tag{62}$$

8.2 C++ classes implementing one-sided distributions

The C++ class `OneSidedDistribution<F>` is a base class for one-sided distributions. It is derived from the base class `Distribution<F>` and has one constructor other than the default constructor:

`OneSidedDistribution(const AntiSymDistribution<F> &dist,`
`bool left, F s)`
: Makes a one-sided distribution from the antisymmetric distribution `dist`. If `left` is true, the derivative at 0 is `s`; otherwise the derivative at 1 is `s`.

It also has the following member functions:

`void set_deriv_at_start(F s)`
: Sets the slope at 0 to `s`.

`void set_deriv_at_end(F s)`
: Sets the slope at 1 to `s`.

`bool has_reversibility_property() const`
: Returns true if the distribution is known to have the reversibility property. Note that it is possible that this function will return false even when the distribution does have the reversibility property.

All other member functions are inherited from the base classes.

The class `OneSidedTanhDistribution<F>` represents a one-sided distribution whose generating function is tanh or tan. The former is appropriate when the specified end-slope is less than or equal to 1, the latter when the end-slope exceeds 1. `OneSidedTanhDistribution<F>` has the following constructor in addition to the default constructor:

`OneSidedTanhDistribution(bool left, F s)`
: Makes a one-sided distribution from the generating function $\tan(x)$ if `s` exceeds 1, otherwise from $\tanh(x)$. If `left` is true, the derivative at 0 is `s`; otherwise the derivative at 1 is `s`.

All other member functions are inherited from the base classes.

The class `OneSidedSinDistribution<F>` is similar but uses the generating function $\sin(x)$ if the end slope is less than or equal to 1 and $\sinh(x)$ if it exceeds 1. It has the following constructor in addition to the default constructor:

`OneSidedSinDistribution(bool left, F s)`
: Makes a one-sided distribution from the generating function $\sinh(x)$ if `s` exceeds 1, otherwise from $\sin(x)$. If `left` is true, the derivative at 0 is `s`; otherwise the derivative at 1 is `s`.

All other member functions are inherited from the base classes.

The class `OneSidedErfDistribution<F>` is similar but uses the generating function $\mathrm{erf}(x)$. It has the following constructor in addition to the default constructor:

`OneSidedErfDistribution(bool left, F s)`
: Makes a one-sided distribution from the generating function $\mathrm{erf}(x)$. If `left` is true, the derivative at 0 is `s`; otherwise the derivative at 1 is `s`. The value of `s` must not exceed 1.

All other member functions are inherited from the base classes.

Other distribution classes derived from `OneSidedDistribution<F>` are described in Sections 9.1 and 10.1.

9 Geometric distribution

A geometric distribution is given by

$$f(x) = \frac{\beta^x - 1}{\beta - 1} \tag{63}$$

for some $\beta > 0$. Its derivative is

$$f'(x) = \frac{\beta^x \ln(\beta)}{\beta - 1} \tag{64}$$

so that

$$f'(0) = \frac{\ln(\beta)}{\beta - 1} \equiv q(\beta); \qquad f'(1) = \frac{\beta \ln(\beta)}{\beta - 1} = q(1/\beta) \tag{65}$$

and

$$\beta = \frac{f'(1)}{f'(0)} \tag{66}$$

Therefore a geometric distribution is an end slope ratio distribution.

The slope at either end-point can be set provided that one can evaluate the inverse of the function $q(x)$. Suppose we wish to set the slope at 0 to s_0. Then it is sufficient to set

$$\beta = q^{(-1)}(s_0) \tag{67}$$

An efficient algorithm for evaluating $q^{(-1)}(x)$ is described in Annex A.6. Therefore a geometric distribution can also be considered a one-sided distribution.

The growth rate of a geometric distribution is constant; the growth rate function is also constant and is given by:

$$r(x) = \ln(\beta) \tag{68}$$

The geometric distribution has the smallest possible growth rate given the ratio of the two end slopes.

The reverse of a geometric distribution is

$$f(x) = \frac{(1/\beta)^x - 1}{1/\beta - 1} \tag{69}$$

Therefore the reverse distribution is the geometric distribution with β replaced with $1/\beta$: the geometric distribution has the reversibility property.

The inverse distribution is

$$f^{(-1)}(x) = \log_\beta\big(1 + x(\beta - 1)\big) \tag{70}$$

When β is very close to 1, there can be loss of accuracy if Equation (63) is evaluated directly. Instead, the following well-behaved approximation can be used:

$$\begin{aligned} f(x) \approx x \;&+\; \frac{x(x-1)}{2!}(\beta-1) \\ &+\; \frac{x(x-1)(x-2)}{3!}(\beta-1)^2 \\ &+\; \frac{x(x-1)(x-2)(x-3)}{4!}(\beta-1)^3 \\ &+\; \frac{x(x-1)(x-2)(x-3)(x-4)}{5!}(\beta-1)^4 \end{aligned} \tag{71}$$

9.1 C++ implementation of geometric distributions

The class `GeometricDistribution<F>` is used to represent geometric distributions. It is derived from the base classes `EndSlopeRatioDistribution<F>` and `OneSidedDistribution<F>` and has the following constructor:

`GeometricDistribution(F beta = 1)`
: Makes a geometric distribution having end slope ratio `beta`.

`GeometricDistribution<F>` also uses the fact that the reverse of the distribution has end slope ratio $1/\beta$ to reimplement the member function `reverse()`. The distribution returned by `reverse()` will evaluate more efficiently than the reverse distribution returned by `Distribution<F>`.

`GeometricDistribution<F>` also reimplements the member function `inverse()` so that it returns an explicit representation of the inverse curve. It will evaluate more efficiently and more accurately than the reverse distribution returned by the base class `Distribution<F>`.

All other member functions are inherited from the base classes.

`GeometricDistribution<F>` makes use of the class `LnxOxm1<F>` which evaluates $q(x) = \ln(x)/(x-1)$. When x is close to 1, it uses the expansion:

$$q(x) = 1 - \frac{x-1}{2} + \frac{(x-1)^2}{3} - \frac{(x-1)^3}{4} + \cdots \tag{72}$$

to avoid inaccuracies due to round-off.

10 Rational distributions

A rational distribution is defined by

$$f(x) = \frac{x}{\alpha + (1-\alpha)x} \tag{73}$$

for any positive α. Its derivative is

$$f'(x) = \frac{\alpha}{\big(\alpha + (1-\alpha)x\big)^2} \tag{74}$$

which is always positive since α is positive.

The slopes at the end points of a rational distribution are

$$f'(0) = \frac{1}{\alpha}; \qquad f'(1) = \alpha \tag{75}$$

Therefore a rational distribution can be treated as one-sided distribution. Moreover, since

$$\alpha = \sqrt{\frac{f'(1)}{f'(0)}} = \sqrt{\beta} \tag{76}$$

a rational distribution can also be treated as an end slope ratio distribution.

It is easily verified that the rational distribution has the reversibility property: its reverse is the rational distribution with α replaced with $1/\alpha$. The inverse of a rational distribution is the same as its reverse.

The growth rate function for a rational distribution is

$$r(x) = \frac{2(\alpha - 1)}{\alpha + (1 - \alpha)x} \tag{77}$$

If $\alpha < 1$, the growth rate decreases monotonically; if $\alpha > 1$ it increases monotonically.

Rational distributions are normally of little direct use because their growth rate becomes very large for both small and large α. However, as shown in Section 11.1, they can be composed with other distributions to generate well-behaved distributions whose end slopes can be set independently.

10.1 C++ implementation of rational distributions

The C++ class `RationalDistribution<F>` is used to represent rational distributions. Since a rational distribution is both an end slope distribution and a one-sided distribution, `RationalDistribution<F>` is derived from both `OneSidedDistribution<F>` and `EndSlopeRatioDistribution<F>`. It has the following constructor:

`RationalDistribution(F beta = 1)`
: Makes a rational distribution with end slope ratio `beta`.

It also reimplements the inherited functions `inverse()` and `reverse()` to provide more efficient alternatives than those provided by the base class `Distribution<F>`.

All other member functions are inherited from the base classes.

11 Two-sided distributions

A two-sided distribution is one for which the slopes at the end points can be set independently. They are useful for generating node distributions in bounded domains where the cell spacing at both ends of an edge is important.

Figure 3 plots three different two-sided distributions, their slopes and their growth

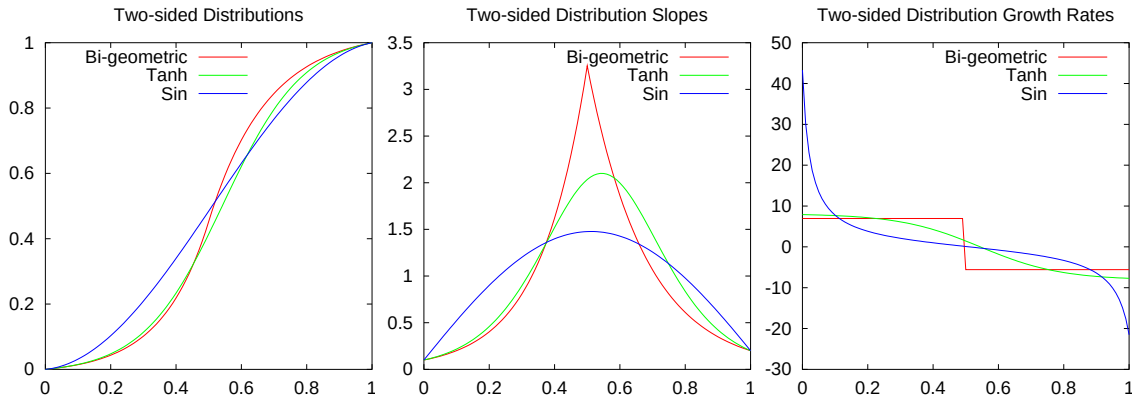

Figure 3: *Three two-sided distributions, their slopes and growth rates: the slope at 0 is 0.1 and the slope at 1 is 0.2.*

rates. For each the slope at 0 is set to 0.1 and the slope at 1 is set to 0.2. The distributions are described in the following sections.

11.1 Composition of end slope ratio and antisymmetric distributions

Let $a(x)$ be an antisymmetric distribution, $e(x)$ an end slope ratio distribution, and define a new distribution by $f = a \circ e$. Then

$$s_0 = f'(0) = a'(0)e'(0); \qquad s_1 = f'(1) = a'(1)e'(1) \tag{78}$$

Since $a'(0) = a'(1)$, we have

$$\frac{e'(1)}{e'(0)} = \frac{s_1}{s_0} \tag{79}$$

so that $e(x)$ is uniquely determined given s_0 and s_1. The antisymmetric distribution can then be uniquely determined since

$$a'(0) = a'(1) = \frac{s_0}{e'(0)} = \frac{s_1}{e'(1)} \tag{80}$$

Since the reverse of a is itself, the reverse of f is simply a composed with the reverse of e. The composed distribution will have the reversibility property if and only if e has the reversibility property.

Similar expressions also apply if f is defined as $e \circ a$.

11.2 Two-sided sin distribution

The two-sided sin distribution is a distribution generated from $\sin(x)$ over the domain $[a, b]$, where $-\pi/2 < a \leq b < \pi/2$. The values of a and b must be determined so that the end slopes are s_0 and s_1. From Equation (20) this means solving the following system of equations:

$$s_0 = \frac{(b-a)\cos(a)}{\sin(b) - \sin(a)} \tag{81}$$

$$s_1 = \frac{(b-a)\cos(b)}{\sin(b) - \sin(a)} \tag{82}$$

With the substitutions $a = x_m - \frac{1}{2}\Delta$ and $b = x_m + \frac{1}{2}\Delta$, this can be rewritten

$$s_0 = \frac{\frac{1}{2}\Delta}{\tan(\frac{1}{2}\Delta)} + \tfrac{1}{2}\Delta\tan(x_m) \tag{83}$$

$$s_1 = \frac{\frac{1}{2}\Delta}{\tan(\frac{1}{2}\Delta)} - \tfrac{1}{2}\Delta\tan(x_m) \tag{84}$$

Therefore

$$\frac{\tan(\frac{1}{2}\Delta)}{\frac{1}{2}\Delta} = \frac{2}{s_0 + s_1} \tag{85}$$

which can be solved for Δ using the methods described in Annex A.2. Once Δ is known, x_m is determined from

$$x_m = \arctan\left(\frac{s_0 - s_1}{\Delta}\right) \tag{86}$$

When Δ is close to zero, $x_m \approx \pm\pi/2$. Since this is a case in which the derivative of the generating function approaches zero, the distribution is not necessarily linear, even when $\Delta = 0$. Substituting Equation (86) into Equation (17) and expanding in Δ one obtains

$$f(x) = x + x(1-x)\left(\frac{s_0 - s_1}{2} - \frac{x(1-x)(s_0 - s_1)\Delta^2}{24} + \frac{(2x-1)\Delta^2}{12}\right) + O(\Delta^4) \tag{87}$$

When $\Delta = 0$ (i.e. when $s_0 + s_1 = 2$), the distribution reduces to a quadratic distribution with parameter $\alpha = 1 - s_0 = s_1 - 1$.

Notice that, since $\tan(x)/x > 1$ for all x, $s_0 + s_1$ must not exceed 2. However, if $s_0 + s_1$ does exceed 2, we can replace $\sin(x)$ by $\sinh(x)$. The expressions for Δ and x_m are then:

$$\frac{\tanh(\frac{1}{2}\Delta)}{\frac{1}{2}\Delta} = \frac{2}{s_0 + s_1} \tag{88}$$

$$x_m = \operatorname{arctanh}\left(\frac{s_1 - s_0}{\Delta}\right) \tag{89}$$

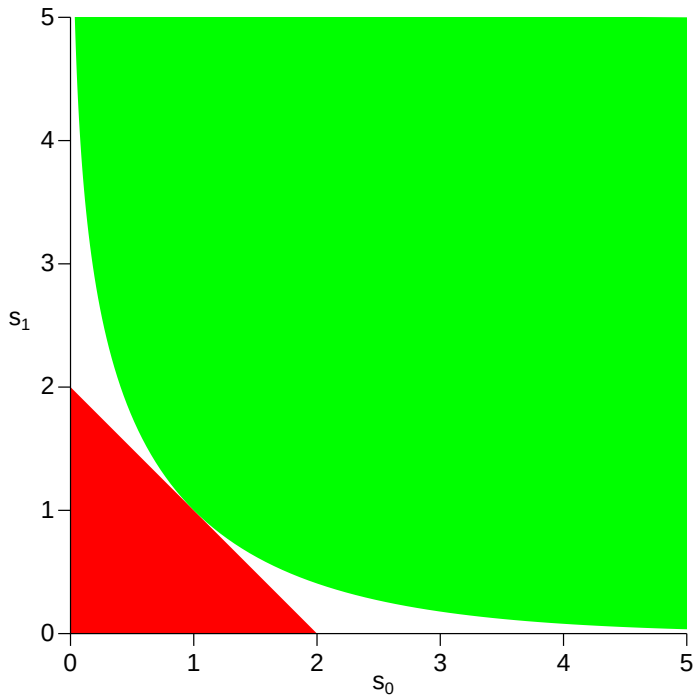

Figure 4: *The region of end slope values (shaded) for which the two-sided sin distribution is well-defined. For the red region sin is used as the generating function; for the green region sinh is used.*

There is now a requirement that $|s_1 - s_0| < \Delta$ otherwise x_m will not be well-defined. This restricts the domain of possible values of s_0 and s_1 for the sin distribution to the region shown in Figure 4. The fact that the sin distribution is not well-defined for all possible end slopes, as well as the fact that its growth rate is very high when the end-slopes are small, mean that the sin distribution is not very useful in practice.

11.3 The tanh distribution

The two-sided tanh distribution (usually simply called a tanh distribution) is a distribution generated from $\tanh(x)$ over the domain $[a, b]$. The values of a and b must be determined so that the end slopes are s_0 and s_1. This distribution is discussed in detail by Vinokur[2] and is the distribution of choice for many applications.

From Equation (20), the end slopes in terms of a and b are:

$$s_0 = \frac{(b-a)}{\cosh^2(a)\big(\tanh(b) - \tanh(a)\big)} = \frac{(b-a)\cosh(b)}{\cosh(a)\sinh(b-a)} \tag{90}$$

$$s_1 = \frac{(b-a)}{\cosh^2(b)\big(\tanh(b) - \tanh(a)\big)} = \frac{(b-a)\cosh(a)}{\cosh(b)\sinh(b-a)} \tag{91}$$

so that

$$\frac{\sinh(b-a)}{b-a} = \frac{1}{\sqrt{s_0 s_1}} \tag{92}$$

$$\frac{\cosh(a)}{\cosh(b)} = \sqrt{\frac{s_1}{s_0}} \tag{93}$$

With the substitutions $a = x_m - \frac{1}{2}\Delta$ and $b = x_m + \frac{1}{2}\Delta$, these can be rewritten

$$\frac{\sinh(\Delta)}{\Delta} = \frac{1}{\sqrt{s_0 s_1}} \tag{94}$$

$$\frac{1+\tanh(x_m)\tanh(\frac{1}{2}\Delta)}{1-\tanh(x_m)\tanh(\frac{1}{2}\Delta)} = \sqrt{\frac{s_1}{s_0}} \tag{95}$$

Equation (94) can be solved for Δ using the methods described in Annex A.5. Once Δ is known, Equation (95) can be solved for x_m:

$$x_m = \operatorname{arctanh}\left(\frac{\sqrt{\frac{s_1}{s_0}}-1}{\tanh(\frac{1}{2}\Delta)\left(\sqrt{\frac{s_1}{s_0}}+1\right)}\right) \tag{96}$$

Notice that, since $\sinh(x)/x > 1$ for all x, $s_0 s_1$ must not exceed 1. However, if $s_0 s_1$ does exceed 1, we can replace $\tanh(x)$ by $\tan(x)$. The expressions for Δ and x_m are then:

$$\frac{\sin(\Delta)}{\Delta} = \frac{1}{\sqrt{s_0 s_1}} \tag{97}$$

$$x_m = \arctan\left(\frac{\sqrt{\frac{s_1}{s_0}}-1}{\tan(\frac{1}{2}\Delta)\left(\sqrt{\frac{s_1}{s_0}}+1\right)}\right) \tag{98}$$

The two-sided tanh distribution can be written

$$f(x) = \frac{\tanh\big(x_m + (x-\frac{1}{2})\Delta\big) - \tanh(x_m - \frac{1}{2}\Delta)}{\tanh(x_m+\frac{1}{2}\Delta) - \tanh(x_m-\frac{1}{2}\Delta)} \tag{99}$$

Using the summation formula

$$\tanh(x+y) = \frac{\tanh(x)+\tanh(y)}{1-\tanh(x)\tanh(y)} \tag{100}$$

one gets

$$f(x) = \frac{\Big(\tanh\big((x-\frac{1}{2})\Delta\big) + \tanh\big(\frac{1}{2}\Delta\big)\Big)\Big(1-\tanh(x_m)\tanh\big(\frac{1}{2}\Delta\big)\Big)}{2\tanh\big(\frac{1}{2}\Delta\big)\Big(1-\tanh(x_m)\tanh\big((x-\frac{1}{2})\Delta\big)\Big)} \tag{101}$$

Now let $h(x)$ be the antisymmetric distribution generated by $\tanh(x)$ on $[-b, b]$:

$$h(x) = \frac{1}{2}\left[1 + \frac{\tanh\left((x - \frac{1}{2})\Delta\right)}{\tanh\left(\frac{1}{2}\Delta\right)}\right] \tag{102}$$

Then we can write

$$f(x) = \frac{h(x)}{\alpha + (1 - \alpha)h(x)} \tag{103}$$

with

$$\alpha = \frac{1 - \tanh(x_m)\tanh\left(\frac{1}{2}\Delta\right)}{1 + \tanh(x_m)\tanh\left(\frac{1}{2}\Delta\right)} \tag{104}$$

which shows that $f(x)$ is simply a rational distribution composed with an antisymmetric tanh distribution. Therefore the two-sided tanh distribution has the interesting property that it can be considered to be a generated distribution with domain $[a, b]$ given by Equations (94) and (96) or to be the composition of a rational distribution with an antisymmetric tanh distribution using the procedure described in Section 11.1.

11.4 C++ classes implementing two-sided distributions

The class `TwoSidedDistribution<F>` is a base class for two-sided distributions. It has one constructor other than the default:

```
TwoSidedDistribution(EndSlopeRatioDistribution<F> esrd,
                     AntiSymDistribution<F> asd,
                     bool esr_outer)
```

Makes a two-sided distribution by composing the end slope distribution `esrd` and the antisymmetric distribution `asd`. If `esr_outer` is true, `esrd` is the outer distribution and `asd` is the inner distribution: i.e. the distribution is `esrd(asd)`; otherwise, `asd` is the outer distribution: the distribution is `asd(esrd)`.

When this constructor is used, the member function `reverse()` will return the reverse of `esrd` composed with `asd`. This is more efficient to evaluate than the default reversed distribution.

Similarly, the member function `inverse()` will return the inverse of `esrd` composed with the inverse of `asd`. This will often be more efficient to evaluate than the default reversed distribution, but when neither `asd` nor `esrd` have efficient inverses defined, it will be less efficient. This should be taken into account when deciding whether to reimplement `inverse()` for derived classes.

`TwoSidedDistribution<F>` also has the following member function:

`void set_end_derivs(F s0, F s1)`
: Sets the derivative at 0 to `s0` and the derivative at 1 to `s1`.

`TwoSidedDistribution<F>` reimplements the member function `reverse()` so that, if the distribution is known to have the reversibility property, the returned distribution is generated by swapping the slopes at the end points. This distribution will be more efficient to evaluate than the default reversed distribution returned by `Distribution<F>`.

All other member functions are inherited from the base classes.

The class `TanhDistribution<F>` represents a tanh distribution. It is derived from the base class `TwoSidedDistribution<F>` and has the following constructor in addition to the default constructor:

`TanhDistribution(F slope0 = 1, F slope1 = 1)`
: Makes a tanh distribution with slope `slope0` at 0 and slope `slope1` at 1.

All other member functions are inherited from the base classes. It also inherits the efficient implementations of `reverse()` and `inverse()` defined in the base class `TwoSidedDistribution<F>`.

The class `SinDistribution<F>` represents a two-sided sin distribution. It is derived from the base class `TwoSidedDistribution<F>`.

`SinDistribution<F>` has the following constructor in addition to the default constructor:

`SinDistribution(F s0 = 1, F s1 = 1)`
: Makes a two-sided sin distribution with slope `slope0` at 0 and slope `slope1` at 1. The constructor is not guaranteed to be successful: see Figure 4 for the regions of allowed values of `s0` and `s1`. If the distribution cannot be constructed, an `Error` is thrown.

`SinDistribution<F>` reimplements the member function `inverse()` to return an explicit version of the inverse distribution which is more efficient and more accurate than the default inverse distribution returned by `Distribution<F>`.

All other member functions are inherited from the base classes. Since the two-sided distribution is known to have the reversibility property, it also inherits the efficient version of `reverse()` defined by the base class `TwoSidedDistribution<F>`.

Section 14.1 describes the class `BiGeometricDistribution<F>` which is also derived from `TwoSidedDistribution<F>`.

12 Interior distributions

In an interior distribution the slope is set at some point, x_m, in the interior of the range $[0, 1]$. We will denote the value of the distribution at x_m by y_m and its slope by s_m. The point (x_m, y_m) is called the cluster point.

In general it is not sufficient to be able to specify the slope at x_m. When generating a node distribution, this allows one to specify the spacing of the nodes at a particular node number, but not where that node will be on the edge. For the latter we need also to be able to specify y_m. Therefore an interior distribution requires that x_m, y_m and s_m can all be specified.

The simplest way to create an interior distribution is by splicing two one-sided distributions: see Section 13.1.

12.1 C++ implementation of interior distributions

The class `InteriorDistribution<F>` is a base class for interior distributions. It is derived from the base class `Distribution<F>` and has the following constructor besides the default.

```
InteriorDistribution(const OneSidedDistribution<F> g1,
                     const OneSidedDistribution<F> g2,
                     F xm, F ym, F sm)
```

Make the distribution by splicing `g1` and `g2` at `xm` with slope `sm` and value `ym`: see Section 13.1.

It also has the following member functions:

```
void define_cluster_point(F xm, F ym, F sm)
```

Sets the value and derivative of the distribution at `xm` to be `ym` and `sm` respectively. Both `xm` and `ym` must be in (0,1).

```
void get_cluster_point(F &xm, F &ym)
```

Returns the location and value of the distribution at the cluster point in `xm` and `ym`.

```
void set_slope(F s)
```

Sets the value of the derivative at the cluster point to `s`. Does not change the location of the cluster point or the value of the distribution there.

All other member functions are inherited from the base classes.

13 Spliced distributions

Any two distribution functions, $g_1(x)$ and $g_2(x)$, can be combined to generate a new distribution function by splicing the beginning of the second to the end of the first:

$$f(x) = \begin{cases} y_m g_1\left(\dfrac{x}{x_m}\right) & \text{for } 0 \le x \le x_m \\ y_m + (1-y_m) g_2\left(\dfrac{x-x_m}{1-x_m}\right) & \text{for } x_m \le x \le 1 \end{cases} \tag{105}$$

The point (x_m, y_m) is called the splice point.

To ensure that the node spacing is continuous where the two edges join, it is necessary that the slope of $f(x)$ is continuous at x_m. This yields a relation between the end-slopes of the two distributions:

$$y_m(1-x_m)g_1'(1) = (1-y_m)x_m g_2'(0) \tag{106}$$

If the slopes of g_1 and g_2 are known at the splice point, we can use this expression to specify x_m given y_m, or y_m given x_m:

$$y_m = \frac{x_m g_2'(0)}{(1-x_m)g_1'(1) + x_m g_2'(0)} \tag{107}$$

$$x_m = \frac{y_m g_1'(1)}{(1-y_m)g_2'(0) + y_m g_1'(1)} \tag{108}$$

The values of x_m and y_m obtained in this way will always be in the range $[0, 1]$, as required.

Suppose the slopes of $f(x)$ at 0, x_m and 1 are s_0, s_m and s_1 respectively. Then

$$s_0 = \frac{y_m}{x_m} g_1'(0) \tag{109}$$

$$s_m = \frac{y_m}{x_m} g_1'(1) = \left(\frac{1-y_m}{1-x_m}\right) g_2'(0) \tag{110}$$

$$s_1 = \frac{1-y_m}{1-x_m} g_2'(1) \tag{111}$$

and

$$\beta_1 \equiv \frac{g_1'(1)}{g_1'(0)} = \frac{s_m}{s_0}; \qquad \beta_2 \equiv \frac{g_2'(1)}{g_2'(0)} = \frac{s_1}{s_m} \tag{112}$$

so that

$$\beta \equiv \frac{s_1}{s_0} = \frac{g_1'(1)}{g_1'(0)} \frac{g_2'(1)}{g_2'(0)} = \beta_1 \beta_2 \tag{113}$$

Therefore the end slope ratio of the spliced distribution is the product of the end slope ratios of its constituents.

The reverse of a spliced distribution can be obtained by splicing the reverse of $g_2(x)$ with the reverse of $g_1(x)$ at $(1 - x_m, 1 - y_m)$.

The inverse of a spliced distribution is the inverse of $g_1(x)$ spliced with the inverse of $g_2(x)$ at (y_m, x_m).

The growth rate function for a spliced distribution is

$$r(x) = \begin{cases} \dfrac{1}{x_m} r_1\left(\dfrac{x}{x_m}\right) & \text{for } 0 \le x \le x_m \\ \dfrac{1}{1 - x_m} r_2\left(\dfrac{x - x_m}{1 - x_m}\right) & \text{for } x_m \le x \le 1 \end{cases} \tag{114}$$

where $r_1(x)$ and $r_2(x)$ are the growth rate functions of $g_1(x)$ and $g_2(x)$.

Suppose we wish to generate a distribution of nodes on the interval $[a, c]$ such that N_1 nodes are placed on $[a, b]$ and N_2 nodes are placed on $[b, c]$. The total number of nodes is $N = N_1 + N_2 - 1$ since one of the nodes is common to both edges. A distribution, $f(x)$, for the combined edge is given by Equation (105) with

$$x_m = \frac{N_1 - 1}{N_1 + N_2 - 2}; \quad y_m = \frac{b - a}{c - a} \tag{115}$$

Notice that for this common application it must be possible to set both x_m and y_m independently.

13.1 Interior distributions from spliced one-sided distributions

Two one-sided distributions can be spliced to form an interior distribution in which the slope, s_m, is specified at the splice point, (x_m, y_m). From Equation (110) it is sufficient to set the end slopes of the sub-distributions as follows:

$$g_1'(1) = \frac{x_m s_m}{y_m}; \qquad g_2'(0) = \frac{(1 - x_m) s_m}{1 - y_m} \tag{116}$$

which is sufficient to specify both g_1 and g_2.

13.2 Two-sided distributions from spliced one-sided distributions

Two one-sided distributions can be spliced to form a two-sided distribution. Suppose that the slopes of the spliced distribution at 0 and 1 are s_0 and s_1 respectively and

that x_m is known. Now choosing a value for y_m will fix the two sub-distributions as Equations (109) and (111) imply that their end-slopes must satisfy:

$$g_1'(0) = \frac{x_m s_0}{y_m}; \qquad g_2'(1) = \frac{(1-x_m)s_1}{1-y_m} \tag{117}$$

Equation (110) is now a constraint that can be solved for y_m. Clearly it would also have been possible to fix y_m and solve for x_m.

13.3 Spliced two-sided distributions

If the sub-distributions of a spliced distribution are both two-sided, then it is possible to set x_m, y_m, s_0, s_m and s_1 independently. The requirements for the end slopes of the sub-distributions are:

$$g_1'(0) = \frac{x_m s_0}{y_m} \tag{118}$$

$$g_1'(1) = \frac{x_m s_m}{y_m} \tag{119}$$

$$g_2'(0) = \frac{(1-x_m)s_m}{1-y_m} \tag{120}$$

$$g_2'(1) = \frac{(1-x_m)s_1}{1-y_m} \tag{121}$$

13.4 Multiple spliced distributions

It is easy to generalize a spliced distribution to many sub-distributions spliced together at more than one point. Let $x_0, \ldots, x_M$ and $y_0, \ldots, y_M$ be increasing sequences with $x_0 = y_0 = 0$ and $x_M = y_M = 1$ and let $g_1, \ldots, g_M$ be distributions. Then

$$f(x) = y_{m-1} + (y_m - y_{m-1}) g_m\left(\frac{x - x_{m-1}}{x_m - x_{m-1}}\right) \qquad \text{for } x_{m-1} \le x \le x_m \tag{122}$$

is also a distribution provided that

$$\frac{y_m - y_{m-1}}{x_m - x_{m-1}} g_m'(1) = \frac{y_{m+1} - y_m}{x_{m+1} - x_m} g_{m+1}'(0) \qquad \text{for all } m = 1, \ldots, M-1 \tag{123}$$

If each g_m is two-sided, then all the x_m, y_m and the slopes at each x_m, s_m, can be specified independently. The end slopes of the sub-distributions are:

$$g_m'(0) = \frac{s_m(x_m - x_{m-1})}{y_m - y_{m-1}} \tag{124}$$

$$g_m'(1) = \frac{s_{m+1}(x_m - x_{m-1})}{y_m - y_{m-1}} \tag{125}$$

13.5 C++ implementation of spliced distributions

The class `SplicedDistribution<F>` represents a spliced distribution. It is inherited from the base class `Distribution<F>` and has the following constructor in addition to the default:

```
SplicedDistribution(F xm, F ym,
                    const Distribution<F> &g1,
                    const Distribution<F> &g2)
```

Makes a spliced distribution using sub-distributions `g1` and `g2`. The location of the splice is `(xm,ym)`. It is up to the calling program to ensure that the distribution has continuity of slopes at the splice point.

It also has the following member function.

```
void get_splice_point(F &xm, F &ym)
```

Returns the location of the splice point in `xm` and `ym`.

`SplicedDistribution<F>` reimplements the member functions `reverse()` and `inverse()` to return distributions generated from the reverse or inverse of the sub-distributions. These distributions will often be more efficient and more accurate, and certainly will be no worse, than the default distributions returned by `Distribution<F>`.

All other member functions are inherited from the base classes.

The class `MultiDistribution<F>` represents a distribution generated by splicing several distributions together at a series of cluster points. It is derived from the base class `Distribution<F>` and has the following constructor in addition to the default.

```
MultiDistribution(const Spline::KnotSeq<F> &xvals,
                  const Spline::KnotSeq<F> &yvals,
                  const std::vector<F> &svals,
                  const OneSidedDistribution<F> &osd =
                    OneSidedTanhDistribution<F>(),
                  const TwoSidedDistribution<F> &tsd =
                    TanhDistribution<F>())
```

Makes a distribution by splicing distributions on the intervals defined by the strictly increasing sequence `xvals` (the first and last values of `xvals` must be 0 and 1 respectively). The values and slopes of the distribution at the splice points are given in `yvals` and `svals`. Each must be of the same length as `xvals`. A copy of `tsd` is made for each interior interval. Its end slopes are then adjusted to ensure that the slope of the overall distribution is correct. If the slopes at 0 and 1 are positive, a similar procedure is used on the first and last intervals. Otherwise, if the first slope is zero or negative, `osd` is copied then its end slope is adjusted so that the slope of the distribution is correct at

`xvals[1]`. A similar procedure is used on the last interval if the last slope is zero or negative.

The class `Spline::KnotSeq<F>` represents a vector of increasing values; it is described in Reference 3, Section 3.

```
MultiDistribution(const OneSidedDistribution<F> &osd,
                  const TwoSidedDistribution<F> &tsd)
```

Makes a multi-distribution which will use sub-distributions `osd` and `tsd`. The vectors of cluster points and their slopes must be defined by a call to `define()` before the distribution is evaluated.

It also has the following member functions.

```
void define(const Spline::KnotSeq<F> &xvals,
            const Spline::KnotSeq<F> &yvals,
            const std::vector<F> &svals,
            const OneSidedDistribution<F> &osd,
            const TwoSidedDistribution<F> &tsd)
```

Redefines the distribution. The arguments are similar to those of the first constructor above.

```
void define(const OneSidedDistribution<F> &osd,
            const TwoSidedDistribution<F> &tsd)
```

Changes the distributions to be used on each of the segments. The current cluster points and slopes will be retained.

```
void define(const Spline::KnotSeq<F> &xvals,
            const Spline::KnotSeq<F> &yvals,
            const std::vector<F> &svals)
```

Changes the cluster points and their slopes. The current distributions used on each of the segments will be retained; if none are defined, a `ProgError` will be thrown (see Reference 1, Annex F for a description of `ProgError`).

All other member functions are inherited from the base classes.

14 Bi-geometric distributions

A bi-geometric distribution is a distribution made by splicing two geometric distributions together as in Equation (105). Let β_1 be the ratio of end slopes for the geometric distribution $g_1(x)$; similarly β_2 is the ratio of end slopes for $g_2(x)$.

Then

$$g_1(x) = \frac{\beta_1^x - 1}{\beta_1 - 1}; \qquad g_2(x) = \frac{\beta_2^x - 1}{\beta_2 - 1}; \tag{126}$$

and the distribution is

$$f(x) = \begin{cases} y_m \dfrac{\beta_1^{\frac{x}{x_m}} - 1}{\beta_1 - 1} & \text{for } 0 \le x \le x_m \\ y_m + (1 - y_m)\dfrac{\beta_2^{\frac{x - x_m}{1 - x_m}} - 1}{\beta_2 - 1} & \text{for } x_m \le x \le 1 \end{cases} \tag{127}$$

From Equation (113) the requirement for continuity at x_m is

$$s_1 = s_0 \beta_1 \beta_2 \tag{128}$$

Since the two sub-distributions of a bi-geometric distribution are both one-sided distributions, a bi-geometric distribution can be made into an interior distribution whose cluster point is at the splice point: see Section 13.1.

We can also make a bi-geometric distribution two-sided by specifying the derivatives at the end points: see Section 13.2. This yields

$$s_0 = \frac{y_m}{x_m} \frac{\ln(\beta_1)}{\beta_1 - 1} \tag{129}$$

$$s_1 = \left(\frac{1 - y_m}{1 - x_m}\right) \frac{\beta_2 \ln(\beta_2)}{\beta_2 - 1} \tag{130}$$

With $q(x) \equiv \ln(x)/(x - 1)$ we can solve these equations for β_1 and β_2:

$$\beta_1 = q^{(-1)}\left(\frac{x_m s_0}{y_m}\right) \tag{131}$$

$$\beta_2 = \left[q^{(-1)}\left(\frac{(1 - x_m)s_1}{1 - y_m}\right)\right]^{-1} \tag{132}$$

Substituting into Equation (128) we obtain the following equation which, given x_m, s_0 and s_1, can be solved for y_m:

$$s_0\, q^{(-1)}\left(\frac{x_m s_0}{y_m}\right) = s_1\, q^{(-1)}\left(\frac{(1 - x_m)s_1}{1 - y_m}\right) \tag{133}$$

Annex A.6 describes an efficient algorithm for evaluating $q^{(-1)}(x)$.

Consider the bi-geometric distribution obtained by the following substitutions:

$$x_m \to 1 - x_m; \qquad y_m \to 1 - y_m; \qquad \beta_1 \to \frac{1}{\beta_2}; \qquad \beta_2 \to \frac{1}{\beta_1} \tag{134}$$

Then it is easily verified that Equations (128)–(130) are satisfied, that $s_0 \to s_1$ and $s_1 \to s_0$, and that the new distribution is the reverse of the old; therefore the bi-geometric distribution has the reversibility property.

A useful alternative for specifying a bi-geometric distribution is to set s_0, y_m and the growth rate function over $x \in [0, x_m]$ (a constant). Call the latter r. Then

$$r = \frac{\ln(\beta_1)}{x_m} \tag{135}$$

Substituting into Equation (129) we find

$$\beta_1 = 1 + \frac{y_m r}{s_0} \tag{136}$$

Since β_1 is now known, x_m can be determined from Equation (135):

$$x_m = \frac{\ln(\beta_1)}{r} \tag{137}$$

Using Equation (128) to eliminate s_1 from Equation (130) gives

$$\beta_2 = q^{(-1)}\left(\frac{s_0\beta_1(1 - x_m)}{1 - y_m}\right) \tag{138}$$

This option is particularly useful for generating node distributions for external boundary layer flows. The nodes can be given a fixed expansion rate until a fixed distance from the wall is reached. The distribution then changes to a geometric distribution which expands naturally to the outer wall.

14.1 C++ implementation of bi-geometric distributions

The class `BiGeometricDistribution<F>` is used to represent bi-geometric distributions. It is derived from the two base classes `InteriorDistribution<F>` and `TwoSidedDistribution<F>` and has the following member functions:

`BiGeometricDistribution(F xm, F slope0, F slope1)`
: Makes a distribution from two geometric distributions spliced at `xm`. The slopes at 0 and 1 are `slope0` and `slope1` respectively.

`void define(F xm, F slope0, F slope1)`
: Sets the splice point to `xm` and the values of the slopes at 0 and 1 to `slope0` and `slope1` respectively.

`void define_fixed_inner_ratio(F ym, F slope0, F ratio)`
: Sets the slope at the left end-point to `slope0` and the value of the distribution at the splice point to `ym`. The growth rate function between 0 and the splice point is constant; its value is `ratio`.

`void get_splice_point(F &xm, F &ym)`

Returns the splice point in `xm` and the value of the distribution at the splice point in `ym`.

All other member functions are inherited from the base classes.

15 Arclength distributions

An arclength distribution is used to convert the parameter of a curve in space, $\mathbf{c}(\xi)$, so that it approximates the fractional arclength along the curve. Define $a(\xi)$ to be the fractional arclength along $\mathbf{c}(\xi)$ for the parameter range $[\xi_o, \xi_1]$:

$$a(\xi) = \frac{\displaystyle\int_{\xi_0}^{\xi} \sqrt{\frac{\partial \mathbf{c}}{\partial \xi} \cdot \frac{\partial \mathbf{c}}{\partial \xi}}\, d\xi}{\displaystyle\int_{\xi_0}^{\xi_1} \sqrt{\frac{\partial \mathbf{c}}{\partial \xi} \cdot \frac{\partial \mathbf{c}}{\partial \xi}}\, d\xi} \tag{139}$$

The arclength distribution for $\mathbf{c}(\xi)$ for the parameter range $[\xi_o, \xi_1]$ is the inverse of $a(\xi)$.

$$f(\xi) = a^{(-1)}(\xi) \tag{140}$$

The curve

$$\mathbf{g}(x) = \mathbf{c}\big(\xi_0 + (\xi_1 - \xi_0) f(x)\big) \tag{141}$$

traces the same path in space as $\mathbf{c}$, but its parameter, x, represents the fractional arclength along the curve. Nodes generated along the curve using the arclength distribution will be equidistant in arclength.

For example, Figure 5 shows the parametric curve

$$\mathbf{c}(\xi) = \begin{pmatrix} \dfrac{(\xi+3)(13-\xi)}{60} \\ \dfrac{1}{1+\xi^2} \end{pmatrix} \tag{142}$$

for ξ in the range $[-3, 3]$. On the left are 21 points equally distributed in ξ. On the right are 21 points equally distributed in the arclength parameter x.

By composing a distribution with an arclength distribution we get a new distribution which can be used to distribute nodes with respect to fractional arclength rather than with respect to the curve parameter.

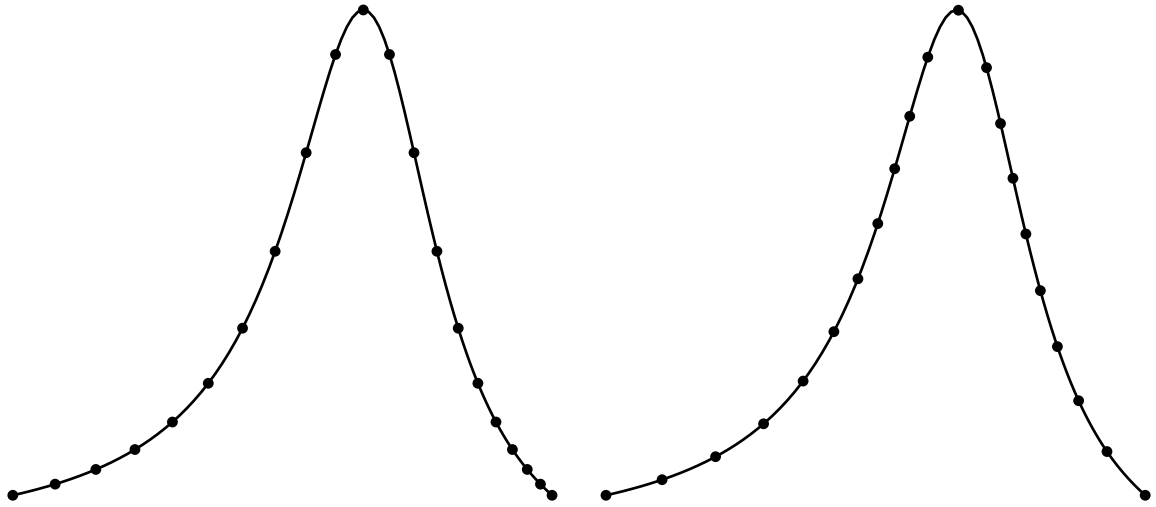

Figure 5: *Nodes on a curve equally spaced in the curve parameter (left) and after an arclength distribution has been applied.*

15.1 C++ implementation of arclength distributions

The C++ class `ArcLengthDistribution<V,F>` is used to represent arclength distributions. It is derived from the base class `Distribution<F>`. The template parameter `V` is the type of the point in space returned by the curve $\mathbf{c}(\xi)$ used to generate the distribution; it must be a model of an Absolute Object as well as an Arithmetic Object: see Reference 1, Annexes A.1 and A.5. As usual, the template parameter `F` is the type of the distribution parameter (usually `double` or `float`). The type of the curve $\mathbf{c}(\xi)$ is `CurveLib::Curve<1U,V,F>`: see Reference 1, Section 2.

To calculate the arclength distribution, the arclength along $\mathbf{c}(\xi)$ must be approximated. The curve is sampled at a number of points; the resulting nodes are joined with straight lines to approximate the arclength. The locations of the sampled points are stored in a `Spline::KnotSeq<F>` (see Reference 3, Section 3). The constructors allow several different methods for determining where the curve is sampled.

Once the fractional arclength has been approximated, a Hermite spline is constructed (see Reference 3, Section 8 for a description of Hermite splines) which uses the fractional arclengths as the knots and the locations of the sampled points as the values. This yields the inverse arclength curve that is required for the distribution.

`ArcLengthDistribution<V,F>` has the following constructors in addition to the default constructor:

```
ArcLengthDistribution(const CurveLib::Curve<1U,V,F> &c,
                      F pmin, F pmax,
                      const Spline::KnotSeq<F> &knots)
```

Makes an arclength distribution for curve `c` over the parameter range `[pmin,pmax]`. The knot sequence `knots` is used to sample the curve `c` to approximate its fractional arclength. The range of the knot sequence must be [0,1]. The sampled points will be `pmin+(pmax-pmin)*knots[i]` for `i` between 0 and `knots.size()-1`. The class `Spline::KnotSeq<F>` is described in Reference 3, Section 3.

```
ArcLengthDistribution(const CurveLib::Curve<1U,V,F> &c,
                      F pmin, F pmax, unsigned n)
```

Makes an arclength distribution for curve `c` over the parameter range `[pmin,pmax]`. To approximate its fractional arclength, the curve is sampled at `n` equally spaced parameters in the range `[pmin,pmax]`.

```
ArcLengthDistribution(const CurveLib::Curve<1U,V,F> &c,
                      F pmin, F pmax, unsigned nmin,
                      unsigned nmax, F acc)
```

Makes an arclength distribution for curve `c` over the parameter range `[pmin,pmax]`. To approximate its fractional arclength, the curve is first sampled at `nmin` equally spaced parameters in the range `[pmin,pmax]`. Extra parameter values are then added until the fractional arclength curve is accurate to `acc` or until the total number of knots is `nmax`. A segment of the curve between two points is 'accurate to `acc`' if adding a parameter value at the centre of the segment changes the arclength between the points by less than the factor `acc`.

16 Concluding remarks

This document has described a library of C++ classes which represent distribution functions of different types. The distribution classes are based on the more general CurveLib library for representing multi-parameter differentiable functions. From the CurveLib classes they inherit arithmetic and composition operators that allow the distribution functions to be combined in complex ways.

Although the distribution classes were designed for generating distributions of nodes in grids used for solving differential equations, they have wider applicability. A distribution function can be used whenever the parameterization of a function is to be altered without affecting its range. The arclength distribution provides a good example of this; it changes the parameterization of a one-parameter curve so that the parameter is proportional to the fractional arclength along the curve.

References

[1] Hally, D. (2006), C++ classes for representing curves and surfaces: Part I: Multi-parameter differentiable functions, (DRDC Atlantic TM 2006-254) Defence R&D Canada – Atlantic.

[2] Vinokur, M. (1980), On One-Dimensional Stretching Functions for Finite-Difference Calculations, (Contractor Report 3313) NASA.

[3] Hally, D. (2006), C++ classes for representing curves and surfaces: Part II: Splines, (DRDC Atlantic TM 2006-255) Defence R&D Canada – Atlantic.

This page intentionally left blank.

Annex A: Implementation of functions to calculate the end slopes of distributions

This annex describes efficient means for calculating the functions used when setting the end-slopes of different distributions. Six functions are described:

1. the inverse of $\sin(x)/x$;
2. the inverse of $\sinh(x)/x$;
3. the inverse of $\tan(x)/x$;
4. the inverse of $\tanh(x)/x$;
5. the inverse of $\ln(x)/(x-1)$; and
6. the inverse of $e^{x^2}\mathrm{erf}(x)/x$.

A.1 Newton-Raphson iterations

In the sections that follow, Newton-Raphson iterations are used to find the value, x, such that $f(x) = y$ for some continuously differentiable function $f(x)$ and value y. The Newton-Raphson method for determining x uses the fact that for x_n sufficient close to x, a better approximation to x is given by

$$x_{n+1} = x_n + \frac{y - f(x_n)}{f'(x_n)} \tag{A.1}$$

Let ϵ be the smallest number such that 1 and $1+\epsilon$ have the same representation as floating point numbers (ϵ is obtained from `std::numeric_limits<F>::epsilon()`, where `F` is the type of the floating point number). Also, let ϵ_f be the relative error in the computed value of $f(x_n)$: i.e. the error in $f(x_n)$ is $f(x_n)\epsilon_f$.

The iteration will fail when $|x_{n+1} - x_n| < \epsilon$ or if $|y - f(x_n)| < \epsilon_f$. In the functions that follow, it can usually be assumed that $\epsilon_f \approx \epsilon$. The iteration is considered converged when either $|x_{n+1} - x_n|$ or $|y - f(x_n)|$ is less than a small multiple of ϵ.

For x close to a point at which $f'(x) = 0$, the Newton-Raphson iteration is not well-behaved. In the neighbourhood of such points it is more accurate to use a direct representation of the inverse function. We will assume here that $f'(0) = 0$ and that $f(x)$ is symmetric. The series then has the form:

$$x = \sqrt{\frac{2|y - y_0|}{|f''(0)|}}\left(1 + a_n(y - y_0) + a_2(y - y_0)^2 + \cdots\right) \tag{A.2}$$

where $y_0 = f(0)$. If the series is truncated after n terms, the relative error can be estimated as $|a_n(y-y_0)^n|$. The series should be used if this error is smaller than the error when the Newton-Raphson iteration used:

$$\frac{f(x)\epsilon}{f'(x)} \approx \frac{y_0\epsilon}{xf''(0)} \approx \frac{y_0\epsilon}{\sqrt{2|(y-y_0)f''(0)|}} \tag{A.3}$$

Therefore, the series should be used if

$$|y-y_0| < \left(\frac{|y_0|\epsilon}{|a_n|\sqrt{2|f''(0)|}}\right)^{\frac{2}{2n+1}} \tag{A.4}$$

A.2 Evaluation of the inverse of tan(x)/x

To specify the slope at the end-points of distributions using $\sin(x)$ as a generating function, it is necessary to be able to solve $\tan(x)/x = y$ for x given a value of $y \geq 1$.

The following approximation for x is accurate to less than 0.036 for any $y > 1$:

$$x_0 = \begin{cases} \sqrt{3z}\left(1 - \frac{3z}{5} + \frac{93z^2}{175} - \frac{451^3}{875} + \frac{34893z^4}{67375}\right) & \text{if } y \in [1, 1.365) \\ \frac{\pi}{2}\left(1 - \frac{4}{\pi^2 y - 4}\right) & \text{if } y \geq 1.365 \end{cases} \tag{A.5}$$

where $z = y - 1$. If $y > 1 + (0.8\epsilon)^{2/11}$, a Newton-Raphson iteration is used to refine the value of x; for $y < 1 + (0.8\epsilon)^{2/11}$, the iteration will not improve the accuracy (see Annex A.1). The iteration is implemented as follows:

repeat {
$f_n = \dfrac{\tan(x_n)}{x_n}$
if $(|y - f_n| < \epsilon y)$ **break**
$x_{n+1} = x_n + \dfrac{x_n(y - f_n)}{1 + f_n(x_n^2 f_n - 1)}$
} **until** $(|x_{n+1} - x_n| < \epsilon x_0)$

More accurate approximations for the initial value of x have been tried, but they do not speed up the execution of the whole algorithm.

This algorithm is implemented in the C++ class `InvTanxOx<F>`, a specialization of `CurveLib::Curve<1U,F,F>` where `F` obeys the same restrictions as the template argument of a `Distribution<F>` (see Section 3). `InvTanxOx<F>` has only the default constructor and has no member functions other than those inherited from `CurveLib::Curve<1U,F,F>`. The accuracy parameter ϵ in the Newton-Raphson iteration is set to `5*std::numeric_limits<F>::epsilon()`.

A.3 Evaluation of the inverse of tanh(x)/x

To specify the slope at the end-points of distributions using $\sinh(x)$ as a generating function, it is necessary to be able to solve $\tanh(x)/x = y$ for x given a value of $y \in [0, 1]$. The following approximation for x is accurate to less than 0.17 for any $y \in [0, 1]$ and its relative error (the absolute error divided by the exact value) is less than 0.0003:

$$x = \begin{cases} \dfrac{1}{y} & \text{if } y \in [0, 0.3) \\ \dfrac{\tanh \frac{1}{y}}{y} & \text{if } y \in [0.3, 0.792) \\ \sqrt{3z}\left(1 + \dfrac{3z}{5} + \dfrac{93z^2}{175} + \dfrac{451^3}{875} + \dfrac{34893z^4}{67375}\right) & \text{if } y \geq 0.792 \end{cases} \tag{A.6}$$

with $z = 1 - y$. The approximation used when $y < 0.3$ is no more accurate that that used in the range [0.3,0.792], but it is faster to calculate. If $y > (0.8\epsilon)^{2/11}$, a Newton-Raphson iteration is used to refine the value of x; for $y < (0.8\epsilon)^{2/11}$, the iteration will not improve the accuracy (see Annex A.1). The iteration is implemented as follows:

repeat {
$f_n = \dfrac{\tanh(x_n)}{x_n}$
if $(|y - f_n| < \epsilon y)$ **break**
$x_{n+1} = x_n + \dfrac{x_n(y - f_n)}{1 - f_n(x_n^2 f_n + 1)}$
} **until** $(|x_{n+1} - x_n| < \epsilon x_0)$

More accurate approximations for the initial value of x have been tried, but they do not speed up the execution of the whole algorithm.

This algorithm is implemented in the C++ class `InvTanhxOx<F>`, a specialization of `CurveLib::Curve<1U,F,F>` where `F` obeys the same restrictions as the template argument of a `Distribution<F>` (see Section 3). `InvTanhxOx<F>` has only the default constructor and has no member functions other than those inherited from `CurveLib::Curve<1U,F,F>`. The accuracy parameter ϵ in the Newton-Raphson iteration is set to `5*std::numeric_limits<F>::epsilon()`.

A.4 Evaluation of the inverse of sin(x)/x

To specify the slope at the end-points of distributions using $\tan(x)$ as a generating function, it is necessary to be able to solve $\sin(x)/x = y$ for x given a value of $y \in [0, 1]$. The values returned are in the range $[0, \pi]$.

For y close to 1, we use the following series expansion to obtain the first approximation to x:

$$x_0 = \sqrt{6z}\left(1 + \frac{3z}{20} + \frac{321z^2}{5600} + \frac{3197z^3}{112000} + \frac{445617z^4}{27596800} + O(z^5)\right); \quad z = 1 - y \tag{A.7}$$

If $y > 1 - (87\epsilon)^{2/11}$, no further refinement with a Newton-Raphson iteration is necessary (see Annex A.1).

Vinokur[2] gives the following approximation for x which is accurate to less than 2.5×10^{-4} for any $y \in [0, 1]$:

$$x_0 = \begin{cases} \pi\big(1 - y + y^2 - (1 + \pi^2/6)y^3 + 6.794732y^4 \\ \quad - 13.205501y^5 + 11.726095y^6\big) & \text{if } y < 0.26938972 \\ \\ \sqrt{6z}\,(1 + 0.15z + 0.057321429z^2 + 0.048774238z^3 \\ \quad - 0.053337753z^4 + 0.075845134z^5) & \text{otherwise} \end{cases} \tag{A.8}$$

where $z = 1 - y$. This formula is more accurate than Equation (A.7) for $y < 0.6$, so it is used for $y \in [0, 0.6]$.

When $y < 1 - (87\epsilon)^{2/11}$, the value of x is refined using a Newton-Raphson iteration implemented as follows:

repeat {
$\quad f_n = \dfrac{\sin(x_n)}{x_n}$
$\quad$ **if** $(|y - f_n| < \epsilon y)$ **break**
$\quad x_{n+1} = x_n + \dfrac{x_n(y - f_n)}{\cos(x) - f_n}$
} **until** $(|x_{n+1} - x_n| < \epsilon x_0)$

This algorithm is implemented in the C++ class `InvSinxOx<F>`, a specialization of `CurveLib::Curve<1U,F,F>` where `F` obeys the same restrictions as the template argument of a `Distribution<F>` (see Section 3). `InvSinxOx<F>` has only the default constructor and has no member functions other than those inherited from `CurveLib::Curve<1U,F,F>`. The accuracy parameter ϵ in the Newton-Raphson iteration is set to `3*std::numeric_limits<F>::epsilon()`.

A.5 Evaluation of the inverse of sinh(x)/x

To specify the slope at the end-points of distributions using $\tanh(x)$ as a generating function, it is necessary to be able to solve $\sinh(x)/x = y$ for positive x given a value of $y \geq 1$.

For y close to 1, we use the following series expansion to obtain the first approximation to x:

$$x_0 = \sqrt{6z}\left(1 - \frac{3z}{20} + \frac{321z^2}{5600} - \frac{3197z^3}{112000} + \frac{445617z^4}{27596800} + O(z^5)\right); \quad z = y - 1 \tag{A.9}$$

If $y < 1 + (87\epsilon)^{2/11}$, no further refinement with a Newton-Raphson iteration is necessary (see Annex A.1).

Vinokur[2] gives the following approximation for x which is accurate to less than 5×10^{-4} for any $y \geq 1$:

$$x_0 = \begin{cases} \sqrt{6z}\,(1 - 0.15z + 0.057321429z^2 - 0.024907295z^3 \\ \qquad + 0.0077424461z^4 - 0.0010794123y^5) & \text{if } y < 2.7829681 \\ \\ v + \left(1 + \dfrac{1}{v}\right)\ln(2v) - 0.02041793 + 0.24902722w \\ \qquad + 1.9496443w^2 - 2.6294547w^3 + 8.56795911w^4 & \text{otherwise} \end{cases} \tag{A.10}$$

where $z = y - 1$, $v = \ln(y)$ and $w = 1/y - 0.028527431$. This formula is more accurate than Equation (A.9) for $y > 1.4$.

When $y > 1 + (87\epsilon)^{2/11}$, the value of x is refined using a Newton-Raphson iteration implemented as follows:

repeat {
$f_n = \dfrac{\sinh(x_n)}{x_n}$
if $(|y - f_n| < \epsilon y)$ **break**
$x_{n+1} = x_n + \dfrac{x_n(y - f_n)}{\cosh(x) - f_n}$
} **until** $(|x_{n+1} - x_n| < \epsilon x_0)$

This algorithm is implemented in the C++ class `InvSinhxOx<F>`, a specialization of `CurveLib::Curve<1U,F,F>` where `F` obeys the same restrictions as the template argument of a `Distribution<F>` (see Section 3). `InvSinhxOx<F>` has only the default constructor and has no member functions other than those inherited from `CurveLib::Curve<1U,F,F>`. The accuracy parameter ϵ in the Newton-Raphson iteration is set to `5*std::numeric_limits<F>::epsilon()`.

A.6 Evaluation of the inverse of ln(x)/(x-1)

To specify the slope at the end-point of a geometric distribution, it is necessary to be able to solve $\ln(x)/(x-1) = y$ for x given a value of y. The following approximation for x is used to initialize a Newton-Raphson iteration to determine x to arbitrary accuracy.

$$x = \begin{cases} \left(1 - \dfrac{\ln(y)}{y}\right)\Big(1.58081 - 0.0906588z - 1.70535z^2 + 1.80299z^3 \\ \qquad\qquad - 3.69287z^4 + 12.4842z^5\Big) & \text{if } y < 0.5 \\ 1 + 2(1-y) + \dfrac{8}{3}(1-y)^2 + \dfrac{28}{9}(1-y)^3 + \dfrac{464}{135}(1-y)^4 \\ \quad + \dfrac{1496}{405}(1-y)^5 + \dfrac{11072}{2835}(1-y)^6 & \text{if } 0.5 \le y < 1.3 \\ 0.287302 - 0.411807p + 0.355619p^2 - 0.252477p^3 \\ \quad + 0.164911p^4 - 0.103479p^5 & \text{if } 1.3 \le y < 2.35 \\ e^{-y}\left(1 + q + \dfrac{3}{2}q^2 + \dfrac{8}{3}q^3 + \dfrac{125}{24}q^4 + \dfrac{54}{5}q^5 + \dfrac{16807}{720}q^6\right) & \text{otherwise} \end{cases} \tag{A.11}$$

with $z = y - \frac{1}{4}$, $p = y - \frac{7}{4}$ and $q = ye^{-y}$. The iteration is implemented as follows:

repeat {
$f_n = \dfrac{\ln(x_n)}{x_n - 1}$
if $(|y - f_n| < \epsilon y)$ **break**
$x_{n+1} = x_n + \dfrac{x_n(x_n - 1)(y - f_n)}{1 - x_n f_n}$
} **until** $(|x_{n+1} - x_n| < \epsilon x_{n+1})$

This algorithm is implemented in the C++ class `InvLnxOxm1<F>`, a specialization of `CurveLib::Curve<1U,F,F>` where `F` obeys the same restrictions as the template argument of a `Distribution<F>` (see Section 3). `InvLnxOxm1<F>` has only the default constructor and has no member functions other than those inherited from `CurveLib::Curve<1U,F,F>`. The accuracy parameter ϵ in the Newton-Raphson iteration is set to `3*std::numeric_limits<F>::epsilon()`.

A.7 Evaluation of the inverse of e^{x^2}erf(x)/x

To specify the slope at the end-point of an erf distribution, it is necessary to be able to solve

$$\frac{\sqrt{\pi}e^{x^2}\mathrm{erf}(x)}{2x} = y \tag{A.12}$$

for x given a value of $y \geq 1$. The following approximation for x is used to initialize a Newton-Raphson iteration to determine x to arbitrary accuracy.

$$x = \begin{cases} \sqrt{\frac{3}{2} - \frac{9}{10}(y-1) + \frac{243}{350}(y-1)^2} & \text{if } 1 \leq y < 1.55 \\ 0.954407 + 0.372613(y-1.9) - 0.183625(y-1.9)^2 & \\ \quad +0.114576(y-1.9)^3 & \text{if } 1.55 \leq y < 3.06 \\ 1.3696 + 0.114073(y-4) - 0.020196(y-4)^2 & \\ \quad +0.00442091(y-4)^3 & \text{if } 3.06 \leq y < 7 \\ \sqrt{\ln\left(\frac{2yz}{\sqrt{\pi}}\right)}; \qquad z = \sqrt{\ln\left(\frac{2y}{\sqrt{\pi}}\right)} & y > 7 \end{cases} \tag{A.13}$$

The iteration is implemented as follows:

repeat {

$f_n = \dfrac{\sqrt{\pi}e^{x_n^2}\mathrm{erf}(x_n)}{2x_n}$

if $(|y - f_n| < \epsilon y)$ **break**

$x_{n+1} = x_n + \dfrac{(x_n - f_n)x_n}{1 - f_n + 2x_n^2 f_n}$

} **until** $(|x_{n+1} - x_n| < \epsilon x_{n+1})$

This code is implemented in the member function `set_end_deriv()` of the class `AntiSymErfDistRep<F>`. The value `3*std::numeric_limits<F>::epsilon()` is used for the accuracy parameter ϵ in the Newton-Raphson iteration.

This page intentionally left blank.

List of symbols

β	The ratio of the end slopes of a distribution: $\beta = s_1/s_0$.
ϵ	The smallest number such that 1 and $1 + \epsilon$ have the same representation as floating point numbers.
f, g, h	Distribution functions.
$q(x)$	$q(x) \equiv \ln(x)/(x-1)$
$q^{(-1)}(x)$	The inverse of $q(x)$.
s_0	The slope of a distribution at $x = 0$.
s_1	The slope of a distribution at $x = 1$.

Index

This page intentionally left blank.

Distribution list

DRDC Atlantic TM-2006-257

Internal distribution

1 Author

5 Library

Total internal copies: 6

External distribution

Department of National Defence

1 DRDKIM

2 DMSS 2

Others

2 Canadian Acquisitions Division
National Library of Canada
395 Wellington Street
Ottawa, Ontario
K1A ON4
Attn: Government Documents

1 Director-General
Institute for Marine Dynamics
National Research Council of Canada
P.O. Box 12093, Station A
St. John's, Newfoundland
A1B 3T5

1 Director-General
Institute for Aerospace Research
National Research Council of Canada
Building M-13A
Ottawa, Ontario
K1A OR6

1 Transport Development Centre
Transport Canada
6th Floor
800 Rene Levesque Blvd, West
Montreal, Que.
H3B 1X9
Attn: Marine R&D Coordinator

1 Canadian Coast Guard
Ship Safety Branch
Canada Building, 11th Floor
344 Slater Street
Ottawa, Ontario
K1A 0N7
Att: Chief, Design and Construction

MOUs

6 Canadian Project Officer ABCA-02-01 (C/SCI, DRDC Atlantic – 3 paper copies, 3 PDF files on CDROM)

Total external copies: 15

Total copies: 21

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when document is classified)

1. ORIGINATOR (The name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Centre sponsoring a contractor's report, or tasking agency, are entered in section 8.)

   Defence R&D Canada – Atlantic
   PO Box 1012, Dartmouth NS B2Y 3Z7, Canada

2. SECURITY CLASSIFICATION (Overall security classification of the document including special warning terms if applicable.)

   UNCLASSIFIED

3. TITLE (The complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S, C or U) in parentheses after the title.)

   C++ classes for representing curves and surfaces: Part IV: Distribution functions

4. AUTHORS (Last name, followed by initials – ranks, titles, etc. not to be used.)

   Hally, D.

5. DATE OF PUBLICATION (Month and year of publication of document.)

   January 2007

6a. NO. OF PAGES (Total containing information. Include Annexes, Appendices, etc.)

   68

6b. NO. OF REFS (Total cited in document.)

   3

7. DESCRIPTIVE NOTES (The category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

   Technical Memorandum

8. SPONSORING ACTIVITY (The name of the department project office or laboratory sponsoring the research and development – include address.)

   Defence R&D Canada – Atlantic
   PO Box 1012, Dartmouth NS B2Y 3Z7, Canada

9a. PROJECT NO. (The applicable research and development project number under which the document was written. Please specify whether project or grant.)

   11cj18

9b. GRANT OR CONTRACT NO. (If appropriate, the applicable number under which the document was written.)

10a. ORIGINATOR'S DOCUMENT NUMBER (The official document number by which the document is identified by the originating activity. This number must be unique to this document.)

   DRDC Atlantic TM-2006-257

10b. OTHER DOCUMENT NO(s). (Any other numbers which may be assigned this document either by the originator or by the sponsor.)

11. DOCUMENT AVAILABILITY (Any limitations on further dissemination of the document, other than those imposed by security classification.)

   (X) Unlimited distribution
   ( ) Defence departments and defence contractors; further distribution only as approved
   ( ) Defence departments and Canadian defence contractors; further distribution only as approved
   ( ) Government departments and agencies; further distribution only as approved
   ( ) Defence departments; further distribution only as approved
   ( ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (Any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in (11)) is possible, a wider announcement audience may be selected.)

13. ABSTRACT (A brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual.)

This document describes C++ classes which implement distribution functions: strictly increasing differentiable functions which map $[0, 1]$ to $[0, 1]$ and are one-to-one and onto. A distribution function is useful whenever one wishes to redistribute values within a given range without changing their order; however, their primary use is in generating distributions of nodes in grids used for solving differential equations.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (Technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus. e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

distribution functions
CurveLib
C++
computer programs

This page intentionally left blank.

Defence R&D Canada

Canada's leader in defence
and National Security
Science and Technology

R & D pour la défense Canada

Chef de file au Canada en matière
de science et de technologie pour
la défense et la sécurité nationale

www.drdc-rddc.gc.ca