1 Integrals

Variational methods are closely connected to integrals. So, before starting our variational adventures, let us recall some elements about integrals and their numerical evaluation.

The history of infinite sums is such as integrals may be brought to Zeno of Elea's paradox about the grain of millet. As described by Simplicius [FAI 98]:

> *Tell me, Protagoras, said he, does one grain of millet make a noise when it falls, or does the ten-thousandth part of a grain? On receiving the answer that it does not, he went on: Does a measure of millet grains make a noise when it falls, or not? He answered, it does make a noise. Well, said Zeno, does not the statement about the measure of millet apply to the one grain and the ten-thousandth part of a grain? He assented, and Zeno continued, Are not the statements as to the noise the same in regard to each? For as are the things that make a noise, so are the noises. Since this is the case, if the measure of millet makes a noise, the one grain and the ten-thousandth part of a grain make a noise.*

Here, Zeno considers the problem of the sum of small negligible components, what is the principle of the integration of infinitesimal contributions. In these ancient times, the concept of the limit was not known, so the notion of infinite sums and the evaluation of areas remained unsolved. In fact, many philosophers considered the question of limits, such as Antiphon the sophist, who argued that continuously doubling the number of sides of a polygon inscribed in a circle gives as result a polygon whose sides coincide with the circumference and having the same area as the circle [PEN 02]. Eudoxus of Cnidus proposed a generalization which led to the exhaustion *method*, which was extensively used for the evaluation of areas for about 2 thousand years. Archimedes applied this method to the evaluation of surfaces and volumes by considering sums of "lines" [BAU 09, BLO 11]. Archimedes' method was improved by Thabit ibn Qurra and inspired Bonaventura Cavalieri and Gilles de Roberval, namely the principle of *indivisibles*. Many researchers enriched the works of Cavalieri. For instance, Evangelista Torricelli introduced indivisibles having a thickness and Blaise Pascal considered triangular and pyramidal sums of indivisibles leading to the evaluation of double and triple integrals.

The differential and integral calculus introduced by Gottfried Leibniz and Isaac Newton opened up a new era. Agustin-Louis Cauchy introduced the concept of the definite integral and chose the notation proposed by Jean-Baptiste Joseph Fourier, which is used today. A complete formalization was proposed by Georg Riemann and led to the Riemann sums and integrals.

At this moment, the horizon seemed clear, but clouds appeared with the works of Karl Weierstrass and Richard Dedekind, which drew attention to fundamental inconsistencies in the theory. Georg Cantor, Camille Jordan and the proposition of a new set and measure theory furnished a new impulse. Emile Borel, René Baire and Henri Lebesgue formed the trinity that constructed a complete formalization of the integrals in the new framework and that gave the impression of taking away the difficulties raised by their predecessors.

But the adventure continues, since one of the most important foundations of the actual theory is the axiom of choice, which generates a paradox which has not yet been solved: the Banach-Tarski paradox, which establishes that a volume cannot be defined for a sphere of radius one. Research regarding this paradox will probably lead to new developments and evolutions in our understanding of these old concepts of areas and volumes, issues that have concerned humanity for millennia and whose complete solution eludes our sagacity every time we think we are near to their complete understanding.

1.1. Riemann integrals

The first formal theory concerning integrals was proposed by Riemann and was provoked by his interest in *Fourier series*. In 1807, a paper by Siméon-Denis Poisson [POI 08] mentioned that Joseph Fourier had proposed the representation of some functions by trigonometric series. The ideas of Fourier immediately aroused controversy. Much of the criticism was related to formal aspects. In his book *Théorie analytique de la chaleur* [FOU 22], Fourier made the following comment about trigonometric series:

> *One could doubt that there existed such a function, but this issue will be clarified later.*

In fact, as often in the history of science, the explanation furnished by Fourier was not complete and has raised fundamental questions, namely what are the functions that may be represented by a trigonometric series and the evaluation of integrals for the determination of the coefficients? Integrals were introduced by Isaac Newton and Gottfried Leibniz in the 17th Century, but their theory was the subject of discussion years before acceptation by the scientific community. About one century after, integrals were being taught in university courses, such as, for instance, those that Augustin-Louis Cauchy presented [CAU 23, CAU 29], but the formal theory waited for the works of Georg Friedrich Bernard Riemann about the questions raised by Fourier, namely his habilitation thesis [RIE 67]. In this thesis, Riemann introduced the fundamental elements for the definition of an integral. Basically, the evaluation of

$$I = \int_{\Omega} f(x)dx$$

requires a subdivision of Ω into a finite number of non-recovering subsets, i.e. a partition $\wp = \{\Omega_i : 1 \leq i \leq N\}$ of Ω:

$$\cup_{i=1}^{N} \Omega_i = \Omega \ , \quad \Omega_i \cap \Omega_j = \emptyset, if\ i \neq j \, .$$

For instance, when $x \in \mathbb{R}$ and $\Omega = (a, b) \subset \mathbb{R}$, we may consider a family of N subintervals $\Omega_i = (x_i, x_{i+1})$ $(i = 1, \dots, N)$ such that $a = x_1 < x_2 < \cdots < x_N < x_{N+1} = b$. For each subinterval Ω_i, the function f has a maximum $\overline{f_i}$ and a minimum $\underline{f_i}$. Thus, we may consider the Riemann sums:

$$\overline{R}(f, \wp) = \sum_{i=1}^{N} \overline{f_i}(x_{i+1} - x_i) \ , \underline{R}(f, \wp) = \sum_{i=1}^{N} \underline{f_i}(x_{i+1} - x_i) \ ,$$

which verify $\underline{R}(f, \wp) \leq \ \overline{R}(f, \wp)$ (see Figure 1.1).

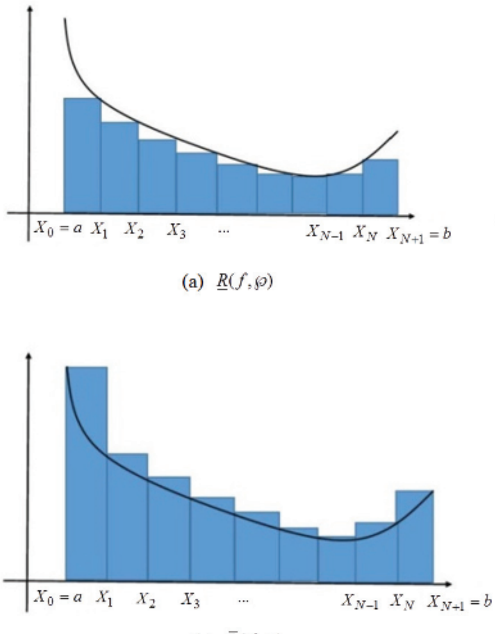


Figure 1.1. *Riemann sums: partitions of the horizontal axis*

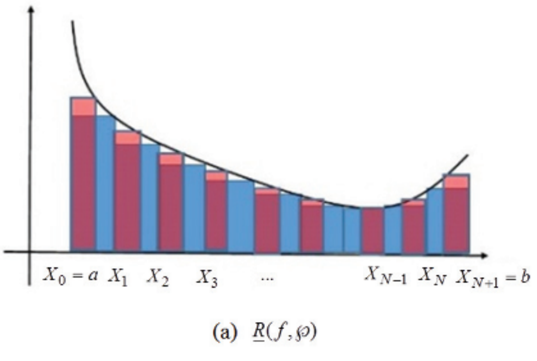
Adding supplementary points to the partition increases $\underline{R}$ and decreases $\overline{R}$ (Figure 1.1), so that, $\underline{R}$ has as cluster point an upper bound $\underline{I}$ and $\overline{R}$ has as cluster point a lower bound $\overline{I}$, $\underline{R}(f,\wp) \leq \underline{I} = \overline{I} \leq \overline{R}(f,\wp)$. When both these values coincide, we say that their common value is $I = \underline{I} = \overline{I}$.

A practical estimation of the value of I is:

$$I \approx \frac{1}{2}\left(\underline{R}(f,\wp) + \overline{R}(f,\wp)\right),$$

i.e.

$$I \approx \sum_{i=1}^{N}\left(\frac{\overline{f}_i+\underline{f}_i}{2}\right)(x_{i+1} - x_i)$$



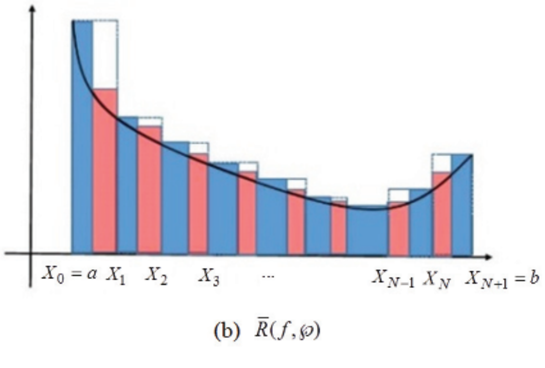


Figure 1.2. *Adding supplementary points increases $\underline{R}$ and decreases $\overline{R}$*

In practice, the values of $\overline{f_i}$ and $\underline{f_i}$ are not determined exactly, but are approximated by using the values $f_i = f(x_i)$ and $f_{i+1} = f(x_{i+1})$:

$$I \approx \sum_{i=1}^{N} \left(\frac{f_i + f_{i+1}}{2}\right)(x_{i+1} - x_i) \qquad [1.1]$$

This formula is known as the *trapezoidal rule*. A simple estimative is furnished by the *mean value approximation*:

$$I \approx \frac{b-a}{N} \sum_{i=1}^{N+1} f_i \ .$$

For $\Omega = (a_1, b_1) \times (a_2, b_2)$ or $\Omega = (a_1, b_1) \times (a_2, b_2) \times (a_3, b_3)$, we may consider families of intervals associated with each component and the Cartesian product of these families. Analogous Riemann sums $\underline{R}(f, \wp) \leq \overline{R}(f, \wp)$ may be defined in this case.

1.2. Lebesgue integrals

Riemann's theory was quickly challenged by the works of Richard Dedekind and Karl Weierstrass. Their work on the foundations of function theory led, on the one hand, to the development of set theory by Georg Cantor and, on the other hand, to measure theory initiated by Giuseppe Peano and Camille Jordan, then formalized by Emile Borel and René Baire. Henri Lebesgue adopted the point of view of measure theory in order to redefine integrals. The theory initiated by Lebesgue is not yet the end of integration theory, since measure theory produces sets that are not measurable (for instance, Vitali sets, presented in the pamphlet [VIT 05]) and, as a consequence, functions that are not measurable. These elements result straight from the axiom of choice (see, for instance, [SOU 10]) and lead to fundamental difficulties, from those presented Felix Hausdorff [HAU 14], and Stefan Banach and Alfred Tarski [BAN 24]. These difficulties have not been solved to date and promise interesting developments in the future.

The numerical evaluation of the Lebesgue integral

$$I = \int_{\Omega} f$$

requires more information than the evaluation of a Riemann integral; we need a partition $\wp = \{\Omega_i : 1 \le i \le N\}$ of Ω and a partition $\mathcal{F} = \{\mathcal{F}_i : 1 \le i \le M\}$ of the image $f(\Omega)$.

For instance, let us consider the situation where both Ω and $f(\Omega)$ are intervals. We define $\mathcal{F}$ by taking points $y_{min} = y_1 < y_2 < \cdots < y_N < y_{M+1} = y_{max}$, where $y_{min} = min\{f(x) : x \in \Omega\}$ and $y_{max} = max\{f(x) : x \in \Omega\}$. In practice, y_{min} and y_{max} may be approximations of these values satisfying where $y_{min} \le min\{f(x) : x \in \Omega\}$ and $y_{max} \ge max\{f(x) : x \in \Omega\}$.

Let us denote by ℓ the Lebesgue measure, given by:

$$\ell\big((\alpha,\beta)\big) = \beta - \alpha, \quad \ell\big((\alpha_1,\beta_1) \times (\alpha_2,\beta_2)\big) = (\beta_1 - \alpha_1)(\beta_2 - \alpha_2), \ldots$$

and

$$\ell_i = \ell\big(f^{-1}([y_i, y_{i+1}))\big) = \ell(\{x \in \Omega : y_i \le f(x) < y_{i+1}\})$$

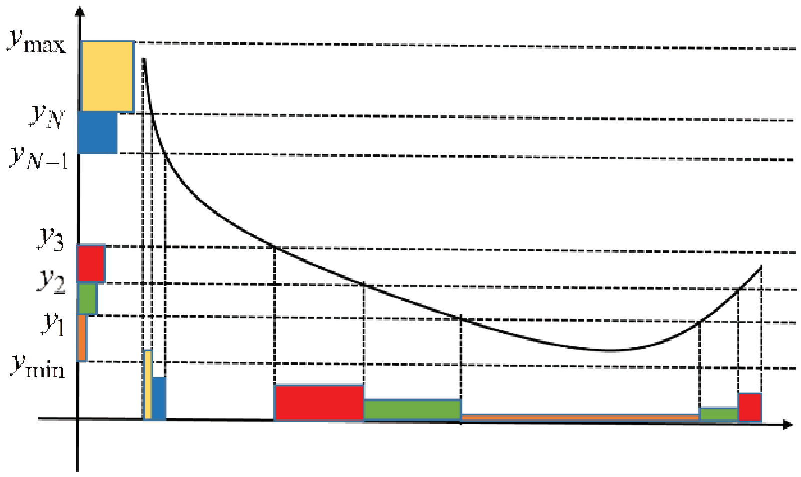


Figure 1.3. *Lebesgue's approach to integration: partition of the vertical axis*

We have:

$$I_{inf} = \sum_{i=1}^{M} y_i\, \ell_i \leq I \leq I_{isup} = \sum_{i=1}^{M} y_{i+1}\, \ell_i$$

Let us introduce:

$$\mu_{inf}(y) = \ell\left(S_{inf}(y)\right)\ ,\ S_{inf}(y) = \{x \in \Omega : f(x) < y\}\,.$$

Then

$$\ell_i = \mu_{inf}(y_{i+1}) - \mu_{inf}(y_i)$$

and

$$\begin{aligned} I_{inf} &= \sum_{i=1}^{M} y_i\, \ell_i = \sum_{i=1}^{M} y_i \left(\mu_{inf}(y_{i+1}) - \mu_{inf}(y_i)\right) \\ &= \sum_{i=2}^{M+1} y_{i-1}\, \mu_{inf}(y_i) - \sum_{i=1}^{M} y_i\, \mu_{inf}(y_i) \\ &= y_{M+1}\mu_{inf}(y_{M+1}) - y_1\mu_{inf}(y_1) - \sum_{i=2}^{M+1}(y_i - y_{i-1})\mu_{inf}(y_i)\ . \end{aligned}$$

Thus,

$$I_{inf} \longrightarrow\ y_{M+1}\mu_{inf}(y_{M+1}) - y_1\mu_{inf}(y_1) - \int_{y_1}^{y_{M+1}} \mu_{inf}(y)\, dy\,,$$

where the integral in this formula is a Riemann integral. So,

$$I \geq y_{M+1}\mu_{inf}(y_{M+1}) - y_1\mu_{inf}(y_1) - \int_{y_1}^{y_{M+1}} \mu_{inf}(y)\, dy\,. \quad [1.2]$$

Analogously,

$$I_{sup} = \sum_{i=1}^{M} y_{i+1}\, \ell_i = \sum_{i=1}^{M} y_{i+1}\left(\mu_{inf}(y_{i+1}) - \mu_{inf}(y_i)\right) = \sum_{i=2}^{M+1} y_i\, \mu_{inf}(y_i) - \sum_{i=1}^{M} y_{i+1}\, \mu_{inf}(y_i) = y_{M+1}\mu_{inf}(y_{M+1}) - y_1\mu_{inf}(y_1) - \sum_{i=1}^{M}(y_{i+1} - y_i)\mu_{inf}(y_i)\ .$$

Thus, we have

$$I_{sup} \longrightarrow\ y_{M+1}\mu_{inf}(y_{M+1}) - y_1\mu_{inf}(y_1) - \int_{y_1}^{y_{M+1}} \mu_{inf}(y)\, dy\,,$$

and

$$I \leq y_{M+1}\mu_{inf}(y_{M+1}) - y_1\mu_{inf}(y_1) - \int_{y_1}^{y_{M+1}} \mu_{inf}(y)\, dy\,. \quad [1.3]$$

Equations [1.2] and [1.3] show that:

$$I = y_{M+1}\mu_{\inf}(y_{M+1}) - y_1\mu_{\inf}(y_1) - \int_{y_1}^{y_{M+1}} \mu_{\inf}(y)\,dy. \qquad [1.4]$$

Notice that, if $y_{min} \le min\{f(x) : x \in \Omega\}$ and $y_{max} \ge max\{f(x) : x \in \Omega\}$, then $\mu_{inf}(y_1) = 0$, while $\mu_{inf}(y_{M+1}) = b - a$ and we have:

$$I = (b-a)y_{M+1} - \int_{y_1}^{y_{M+1}} \mu_{inf}(y)\,dy\,.$$

Let

$$\mu_{sup}(y) = \ell\left(S_{sup}(y)\right)\,,\ S_{sup}(y) = \{x \in \Omega : f(x) \ge y\}\,.$$

Then,

$$\ell_i = \mu_{sup}(y_i) - \mu_{sup}(y_{i+1})$$

and

$$I_{inf} = \sum_{i=1}^{M} y_i\,\ell_i = \sum_{i=1}^{M} y_i\left(\mu_{sup}(y_i) - \mu_{sup}(y_{i+1})\right) = \sum_{i=1}^{M} y_i\,\mu_{sup}(y_i) - \sum_{i=2}^{M+1} y_{i-1}\,\mu_{sup}(y_i) = y_1\mu_{sup}(y_1) - y_{M+1}\mu_{sup}(y_{M+1}) + \sum_{i=2}^{M+1}(y_i - y_{i-1})\mu_{sup}(y_i)\ .$$

Thus,

$$I_{inf} \longrightarrow\ y_1\mu_{sup}(y_1) - y_{M+1}\mu_{sup}(y_{M+1}) + \int_{y_1}^{y_{M+1}} \mu_{sup}(y)\,dy\,,$$

where the integral in this formula is a Riemann one. So,

$$I \ge y_1\mu_{sup}(y_1) - y_{M+1}\mu_{sup}(y_{M+1}) + \int_{y_1}^{y_{M+1}} \mu_{sup}(y)\,dy\,. \qquad [1.5]$$

Analogously,

$$I_{sup} = \sum_{i=1}^{M} y_{i+1}\,\ell_i = \sum_{i=1}^{M} y_{i+1}\left(\mu_{sup}(y_i) - \mu_{sup}(y_{i+1})\right),$$

so that

$$I_{sup} = y_1\mu_{sup}(y_1) - y_{M+1}\mu_{sup}(y_{M+1}) - \sum_{i=1}^{M}(y_{i+1} - y_i)\mu_{sup}(y_i) \ .$$

Thus, we have:

$$I_{sup} \longrightarrow \ y_1\mu_{sup}(y_1) - y_{M+1}\mu_{sup}(y_{M+1}) + \int_{y_1}^{y_{M+1}} \mu_{sup}(y)\, dy \ ,$$

and

$$I \leq y_1\mu_{sup}(y_1) - y_{M+1}\mu_{sup}(y_{M+1}) + \int_{y_1}^{y_{M+1}} \mu_{sup}(y)\, dy \ . \quad [1.6]$$

Equations [1.5] and [1.6] show that:

$$I = y_1\mu_{\text{sup}}(y_1) - y_{M+1}\mu_{\text{sup}}(y_{M+1}) + \int_{y_1}^{y_{M+1}} \mu_{\text{sup}}(y)\,dy. \quad [1.7]$$

Notice that, if $y_{min} \leq \min\{f(x): x \in \Omega\}$ and $y_{max} \geq \max\{f(x): x \in \Omega\}$, then $\mu_{sup}(y_1) = b - a$, while $\mu_{sup}(y_{M+1}) = 0$ and we have:

$$I = (b-a)y_1 + \int_{y_1}^{y_{M+1}} \mu_{sup}(y)\, dy \ .$$

1.3. Matlab® classes for a Riemann integral by trapezoidal integration

The one-dimensional (1D) trapezoidal rule is implemented in Matlab® in the intrinsic function `trapz` and may be extended to multidimensional situations. Assume that the structure `p` has fields `p.x, p.y, p.z` corresponding to the coordinates of the points x_i, y_j, z_k of the partitions of the intervals and `p.dim` corresponding to the dimension, while table `F` contains the values of f - $F_i = f(x_i)$, or $F_{ij} = f(x_i, y_j)$, or $F_{ijk} = f(x_i, y_j, z_k)$, according to the dimension of the integral. Let us summarize the data in a structure `data` such that `data.points = p` and `data.values = F`. Then we may use the class below (Program 1.1).

```
classdef riemann
    methods (Static, Access = private)
        function v = trapz2(x1,x2,F)
            sy = zeros(size(x1));
            for i = 1: length(sy)
                sy(i) = trapz(x2,F(i,:));
            end;
            v = trapz(x1,sy);
        end
        function v = trapz3(x1,x2,x3,F)
            sz = zeros(length(x1),length(x2));
            for i = 1: length(x1)
                for j = 1: length(x2)
                    sz(i,j) = trapz(x3,F(i,j,:));
                end;
            end;
            sy = zeros(size(x1));
            for i = 1: length(x1)
                sy(i) = trapz(x2,sz(i,:));
            end;
            v = trapz(x1,sy);
        end
    end
    methods(Static)
        function v = trpzd(data)
            p = data.points;
            F = data.values;
            n = p.dim;
            switch n
                case 1
                    v = trapz(p.x,F);
                case 2
                    v = riemann.trapz2(p.x,p.y,F);
                case 3
                    v = riemann.trapz3(p.x,p.y,p.
z,F);
```

```
                otherwise
                    v = [];
                    disp('error in arguments for
trpzd');
            end
        end
        function v = mean_value(data)
            p = data.points;
            F = data.values;
            n = p.dim;
            switch n
                case 1
                    v = mean(F)*p.measure;
                case 2
                    v = mean(mean(F))*p.measure;
                case 3
                    v =
mean(mean(mean(F)))*p.measure;
                otherwise
                    v = [];
                    disp('error in arguments for
mean value');
                end
        end
    end
end
```

Program 1.1. *A class for the evaluation of Riemann integrals*

This class contains only trapezoidal and mean value integration methods, but it can be enriched by the user with other methods of numerical integration.

EXAMPLE 1.1.– Let us evaluate:

$$I = \int_0^1 dx \int_0^2 (x^2 + y^2)\, dy$$

by using the points `x=0:0.01:1;` and `y=0:0.01:2;`. Assuming that `F(i,j)=x(i)^2+y(j)^2;`, the code:

```
p.x = x;
p.y = y;
p.dim  = 2;
p.measure = 2;
data.points = p;
data.values = F;
v1 = riemann.trpzd(data)
v2 = riemann.mean_value(data)
```

produces `v1 = 3.3334, v2 = 3.3433.` The exact value is $\frac{10}{3} \approx 3.3333$. ■

EXAMPLE 1.2.– Let us evaluate:

$$I = \int_0^1 dx \int_0^2 dy \int_0^3 (x^2 + y^2 + z^2)\, dz$$

by using the points `x=0:0.01:1;,y=0:0.01:2;, z=0:0.01:3.` Assuming that `F(i,j,k)=x(i)^2+y(j)^2+z(k)^2`, the code:

```
p.x = x;
p.y = y;
p.z = z;
p.dim  = 3;
p.measure = 6;
data.points = p;
data.values = F3;
v1 = riemann.trpzd(F3,p)
v2 = riemann.mean_value(F3,p)
```

produces `v1 = 28.0003, v2 = 28.0600.` The exact value is 28. ■

The creation of the tables `F` from a subprogram `f` evaluating a function f is made by the following class:

```
classdef spam
    properties
    end
    methods (Static, Access = private)
        function v = vspam1(x,f)
            v = zeros(size(x));
            for i = 1: length(x)
                v(i) = f(x(i));
            end;
        end
        function v = vspam2(x,y,f)
            v = zeros(length(x),length(y));
            for i = 1: length(x)
                for j = 1: length(y)
                    v(i,j) = f(x(i),y(j));
                end;
            end;
        end
        function v = vspam3(x,y,z,f)
            v = zeros(length(x),length(y),length(z));
            for i = 1: length(x)
                for j = 1: length(y)
                    for k = 1: length(z)
                        v(i,j,k) = f(x(i),y(j),z(k));
                    end;
                end;
            end;
        end
        function v = tspam1(x,f)
            v = zeros(size(x));
            for i = 1: length(x)
                v(i) = f(x(i));
            end;
        end
        function v = tspam2(x,y,f)
            v = zeros(size(x));
            for i = 1: size(x,1)
                for j = 1: size(x,2)
                    v(i,j) = f(x(i,j),y(i,j));
                end;
            end;
        end
        function v = tspam3(x,y,z,f)
            v = zeros(size(x));
            for i = 1: size(x,1)
                for j = 1: size(x,2)
                    for k = 1: size(x,3)
                        v(i,j,k) =
f(x(i,j,k),y(i,j,k),z(i,j,k));
                    end;
                end;
            end;
        end
    end
    methods (Static)
        function v = points(method,f,x,y,z)
            n = nargin() - 2;
            if isnumeric(f)
                ff = @(x) f;
            else
                ff = f;
            end;
            switch method
                case 'vector'
                    switch n
                        case 1
                            v = spam.vspam1(x,ff);
                        case 2
                            fr = @(x,y) ff([x, y]);
                            v = spam.vspam2(x,y,fr);
                        case 3
                            fr = @(x,y,z) ff([x, y, z]);
                            v = spam.vspam3(x,y,z,fr);
                    end;
                case 'table'
                    switch n
                        case 1
                            v = spam.tspam1(x,ff);
                        case 2
                            fr = @(x,y) ff([x, y]);
                            v = spam.tspam2(x,y,fr);
                        case 3
                            fr = @(x,y,z) ff([x, y, z]);
                             v = spam.tspam3(x,y,z,fr);
                    end;
                otherwise
                    v = [];
                    disp('invalid method for
spam.points');
            end
        end
        function v = partition(f,p)
            n = p.dim;
            switch n
                case 1
                    v = spam.points('vector',f,p.x);
                case 2
                   v = spam.points('vector',f,p.x,p.y);
                case 3
                    v = spam.points('vector',f,p.x,
p.y,p.z);
                otherwise
                    v = [];
            disp('invalid partition dim for
spam.partition');
            end;
        end
    end
end
```

Program 1.2. *A class for the creation of the tables F*

This class generates a table of values $F_i = f(x_i)$, $F_{ij} = F(x_i, y_j)$, $F_{ij} = F(x_{ij}, yi_j)$, $F_{ijk} = f(x_i, y_j, z_k)$, or $F_{ijk} = f(x_{ijk}, y_{ijj}, z_{ijk})$, according to the data furnished. Data is stored in the structure `p`: `p.dim` is the dimension, `p.x`, `p.y`, `p.z` are the coordinates. `Method` takes the value "`vector`" or "`array`" according to the dimensions of `p.x`, `p.y`, `p.z`. For instance, the code:

```
x = 0:0.1:1;
y = 0:0.1:2;
f = @(x) sum(x.^2);
p.x = x;
p.y = y;
p.dim  = 2;
[X,Y] = meshgrid(x,y);
xx = permute(X,[2 1]);
yy = permute(Y,[2 1]);
F1 = spam.partition(f,p);
F2 = spam.points('table',f,xx,yy);
v = sqrt(sum(sum((F1-F2).^2)));
```

produces as result `v=0`. Both the tables `F1` and `F2` coincide, but F1 is generated by using vectors `x` and `y`, while `F2` is generated using two-dimensional (2D) arrays. Adding the lines:

```
z = 0:0.1:3;
p.z = z;
p.dim = 3;
[X,Y,Z] = meshgrid(x,y,z);
xx =  permute(X,[2 1 3]);
yy  =  permute(Y,[2 1 3]);
zz =  permute(Z,[2 1 3]);
F1 = spam.partition(f,p);
F2 = spam.points('table',f,xx,yy,zz);
v3 = sqrt(sum(sum(sum((F1-F2).^2))));
```

produces as result `v3=0`: the resulting arrays are equal. Here, F1 is generated by using vectors `x`, `y` and `z`, while `F2` is generated using three-dimensional (3D) arrays.

Notice that the class `Riemann` requests data in the vector form. The extension to data defined in the array form will be useful for integration using the intrinsic functions of Matlab (see section 1.6).

1.4. Matlab® classes for Lebesgue's integral

Equations [1.4] and [1.7] provide a practical method for the evaluation of I: the Lebesgue integral is replaced by a Riemann one, which may be numerically evaluated by quadrature methods using the values $\mu_{inf}(y_1), \dots, \mu_{inf}(y_{M+1})$ (when using equation [1.4]) or $\mu_{sup}(y_1), \dots, \mu_{sup}(y_{M+1})$ (when using equation [1.7]). The construction of the values of μ_{inf} or μ_{sup} is performed by using the subdivision of Ω: for each i, the measure of the part of Ω_i belonging to $S_{inf}(y)$ (respectively, $S_{sup}(y)$) is estimated and added to $\mu_{inf}(y_i)$ (respectively, $\mu_{sup}(y_i)$). For instance, we may use the following class:

```
classdef lebesgue
    methods  (Static, Access = private)
        function v =  hexainds(i,j,k)
            v = [
                i j k
                i+1 j k
                i+1 j+1 k
                i j+1 k
                i j k+1
                i+1 j k+1
                i+1 j+1 k+1
                i j+1 k+1
                ];
        end
        function v = quadinds(i,j)
            v = [
                i j
                i+1 j
                i+1 j+1
                i j+1
                ];
        end
        function [m, flag] = inf1(y,x,yt)
            y2 = max(y);
            y1 = min(y);
            dx = x(2) - x(1);
            flag = 1;
            if yt < y1
                m = 0;
                flag = 0;
            elseif yt >= y2
                m = dx;
            else
                if y1 == -Inf
                    m = 0.5*dx;
                elseif y2 == Inf
                    m = 0.5*dx;
                else
                    slope = (y2- y1)/dx;
                    m = (yt - y1)/slope;
                end;
            end;
        end
        function [m, flag] = sup1(y,x,yt)
            y2 = max(y);
            y1 = min(y);
            dx = x(2) - x(1);
            flag = 1;
            if yt > y2
                m = 0;
                flag = 0;
            elseif yt <= y1
                m = dx;
            else
                if y1 == -Inf
                    m = 0.5*dx;
                elseif y2 == Inf
                    m = 0.5*dx;
                else
                    slope = (y2- y1)/dx;
                    m = (y2 - yt)/slope;
                end;
            end;
        end
        function [m, flag]  = m2(y,x,yt,m1)
            s = zeros(4,1);
            flg = zeros(4,1);
            [s(1),flg(1)] = m1(y(1:2),[0,1],yt);
            [s(2),flg(2)] = m1(y(2:3),[0,1],yt);
            [s(3),flg(3)] = m1(y(3:4),[0,1],yt);
            [s(4),flg(41)] = m1([y(1),y(4)],[0,1],yt);
            ta = x(2,:) - x(1,:);
            tb = x(4,:) - x(1,:);
            m = mean(s)*norm(ta)*norm(tb);
            flag = max(flg);
        end
        function [m,flag] = m3(y,x,yt,m1)
            s = zeros(6,1);
            flg = zeros(6,1);
            ind = [
                1 2 3 4
                2 3 7 6
                3 4 8 7
                1 4 8 5
                1 2 6 5
                5 6 7 8
                ];
            for i = 1: 6
[s(i),flg(i)] = lebesgue.m2(y(ind(i,:)),[0 0; 0 1; 1
1; 1 0],yt,m1);
            end;
            ta = x(2,:) - x(1,:);
            tb = x(3,:) - x(2,:);
            tc = x(5,:) - x(1,:);
            m = mean(s)*norm(ta)*norm(tb)*norm(tc);
            flag = max(flg);
        end
        function [m, flag] = inf_el(y,x,yt,dim)
            switch dim
                case 1
                    [m, flag] =  lebesgue.inf1(y,x,yt);
                case 2
                    [m, flag] = lebesgue.m2(y,x,yt,
 @lebesgue.inf1);
                case 3
                    [m, flag] = lebesgue.m3(y,x,yt,
 @lebesgue.inf1);
            end
        end
        function [m, flag] = sup_el(y,x,yt,dim)
            switch dim
                case 1
                    [m, flag] = lebesgue.sup1(y,x,yt);
                case 2
                    [m, flag] = lebesgue.m2(y,x,yt,
@lebesgue.sup1);
                case 3
                    [m, flag] = lebesgue.m3(y,x,yt,
@lebesgue.sup1);
            end
        end
        function mu = msre(mes,y,data,pas,hw)
            p = data.points;
            f = data.values;
            n = p.dim;
            ny = length(y);
            mu = zeros(ny,1);
            switch n
                case 1
                    x1 = p.x;
                    n1 = length(x1);
                    for i =  1:  n1 - 1
                        xe = [x1(i) , x1(i+1)];
                        ye = [f(i) , f(i+1)];
                        flag = 1;
                        if pas > 0
                            ind = 1;
                        else
                            ind = ny;
                        end;
  while flag > 0 && ((ind > 1 && pas < 0)  || (ind <
ny && pas > 0))
                            [m, flag] =
mes(ye,xe,y(ind),n);
                            mu(ind) = mu(ind) +  m;
                            ind = ind + pas;
                        end;
                        wx = i/n1;
         text = [sprintf(' %4.1f ',fix(1000*wx)/10),'
% generated'];
                        if pas > 0
                            waitbar(wx,hw,['\mu_{sup}
:',text]);
                        else
```

```
                            waitbar(wx,hw,['\mu_{inf}
:',text]);
                        end;
                    end;
                    close(hw)
                case 2
                    x1 = p.x;
                    x2 = p.y;
                    n1 = length(x1);
                    n2 = length(x2);
                    nt = n1*n2;
                    for i =  1:  n1 - 1
                        for j = 1: n2 - 1
                            ind = lebesgue.quadinds
(i,j);
                            xe = [ x1(ind(:,1))'
x2(ind(:,2))'];
                            ye = zeros(4,1);
                            for ii =  1:  4
                                ye(ii) = f(ind(ii,1),
ind(ii,2));
                            end;
                            flag = 1;
                            if pas > 0
                                ind = 1;
                            else
                                ind = ny;
                            end;
   while flag > 0 && ((ind > 1 && pas < 0) || (ind <
ny && pas > 0))
                                [m, flag] = mes(ye,xe,
y(ind),n);
                                mu(ind) = mu(ind) +
m;
                                ind = ind + pas;
                            end;
                            wx = (j + (i-1)*n2)/nt;
        text = [sprintf(' %4.1f ',fix(1000*wx)/10),' %
generated'];
                            if pas > 0
```

```
                                waitbar(wx,hw,
['\mu_{sup} :',text]);
                            else
                                waitbar(wx,hw,
['\mu_{inf} :',text]);
                            end;
                        end;
                    end;
                    close(hw)
                case 3
                    x1 = p.x;
                    x2 = p.y;
                    x3 = p.z;
                    n1 = length(x1);
                    n2 = length(x2);
                    n3 = length(x3);
                    nt = n1*n2*n3;
                    for i =  1:  n1 - 1
                        for j = 1: n2 - 1
                            for k = 1:n3 - 1
                                ind =
lebesgue.hexainds(i,j,k);
                xe = [ x1(ind(:,1))'  x2(ind(:,2))'
x3(ind(:,3))'];
                                ye = zeros(8,1);
                                for ii =  1:  8
                 ye(ii) = f(ind(ii,1), ind(ii,2),
ind(ii,3));
                                end;
                                flag = 1;
                                if pas > 0
                                    ind = 1;
                                else
                                    ind = ny;
                                end;
   while flag > 0 && ((ind > 1 && pas < 0) || (ind <
ny && pas > 0))
                                    [m, flag] =
mes(ye,xe,y(ind),n);
                                    mu(ind) = mu(ind)
+  m;
                                    ind = ind + pas;
                                end;
                wx = (k + (j-1)*n3 + (i-1)*n2*n3)/nt;
        text = [sprintf(' %4.1f ',fix(1000*wx)/10),' %
generated'];
                                if pas > 0
                                waitbar(wx,hw,
['\mu_{sup} :',text]);
                                else
                                waitbar(wx,hw,
['\mu_{inf} :',text]);
                                end;
                            end;
                        end;
                    end;
                    close(hw)
                otherwise
                    mu = [];
                    disp('error in args for measure');
                    close(hw)
            end
        end
        function mu = mesinf(y,data,hw)
            if nargin() < 3
 hw = waitbar(0,'0 % generated','Name','Lebesgue inf
measure');
            end;
            mu = lebesgue.msre(@lebesgue.inf_el,
            y,data,-1,hw);
        end
        function mu = messup(y,data,hw)
            if nargin() < 3
 hw = waitbar(0,'0 % generated','Name','Lebesgue sup
measure');
            end;
            mu = lebesgue.msre(@lebesgue.sup_el,
y,data,1,hw);
        end
        function v = intinf(y,data,interpmet,hw)
            if nargin() < 4
hw = waitbar(0,'0 % generated','Name','intinf:
Lebesgue inf meas');
```

```
            end
            mu1 = lebesgue.mesinf(y,data,hw);
            ff = @(t) interp1(y,mu1,t,interpmet);
     v = y(end)*mu1(end) - y(1)*mu1(1) -
integral(ff,y(1),y(end));
        end
        function v = intsup(y,data,interpmet,hw)
            if nargin() < 4
hw = waitbar(0,'0 % generated','Name','intsup:
Lebesgue sup meas');
            end
            mu2 = lebesgue.messup(y,data,hw);
            ff = @(t) interp1(y,mu2,t,interpmet);
   v = y(1)*mu2(1) - y(end)*mu2(end) +
integral(ff,y(1),y(end));
        end
    end
    methods (Static)
        function mu1 = measure_inf(y,data,interpmet)
            aux = lebesgue.mesinf(y,data);
            mu1 = @(t) interp1(y,aux,t,interpmet);
        end
        function mu2 = measure_sup(y,data,interpmet)
            aux = lebesgue.messup(y,data);
            mu2 = @(t) interp1(y,aux,t,interpmet);
        end
        function v = integrate(y,data, metod,
 interpmet)
            switch metod
                case 'inferior'
                    switch interpmet
                        case 'none'
                            mu1 = lebesgue.mesinf
 (y,data);
                            v = trapz(y,mu1);
                        otherwise
                            v =
lebesgue.intinf(y,data,interpmet);
                    end
                case 'superior'
                    switch interpmet
```

```
                                case 'none'
                                    mu2 = lebesgue.messup
(y,data);
                                    v = trapz(y,mu2);
                                otherwise
                                    v =
lebesgue.intsup(y,data,interpmet);
                            end
                        case 'mean'
                            switch interpmet
                                case 'none'
                                    mu1 =
lebesgue.mesinf(y,data);
                                    aux1 = trapz(y,mu1);
                                    mu2 =
lebesgue.messup(y,data);
                                    aux2 = trapz(y,mu2);
                                otherwise
hw = waitbar(0,'0 % generated','Name','intmean:
Lebesgue inf meas');
                                aux1 =
lebesgue.intinf(y,data,interpmet,hw);
hw = waitbar(0,'0 % generated','Name','intmean:
Lebesgue sup meas');
                                aux2 =
lebesgue.intsup(y,data,interpmet,hw);
                            end
                            v = 0.5*(aux1+aux2);
                        otherwise
                            v = [];
                            disp('error in args for
integral');
                    end
                end
            end
end
```

Program 1.3. *A class for the evaluation of Lebesgue integrals*

At a glance, we see that this class is more complex than the class Riemann. This is due to the fact that two partitions are requested and must be treated. Methods `inf_el` and `sup_el` determine the part of $(x_{j-1}, x_j), 1 \leq j \leq N,$ which belongs to $S_{inf}(y_i)$ and $S_{sup}(y_i)$, respectively. These subprograms return approximations of the Lebesgue measure of the part of (x_1, x_2) lying in the region of interest ($S_{inf}(y_t)$ for `mes_inf` or $S_{sup}(y_t)$ for `mes_sup`). The special cases where either $y_1 = -\infty$ or $y_2 = +\infty$ are treated by considering that a half of the interval (x_1, x_2) belongs to the region of interest – this choice is arbitrary and may be modified by the user. For the standard situation where both y_1 and y_2 are real numbers, a linear interpolation determines the part of the interval belonging to the region of interest – this choice may also be modified by the user. When using this class, the evaluation of the integral involves the choices, on the one hand, the choice of the evaluation points (vector `y`) and, on the other hand, the choice among the use of μ_{inf} (equation [1.4]), of μ_{sup} (equation [1.7]) or the arithmetic mean of these results. In addition, it is possible to reduce the time of computation by evaluating these measures in a limited number of points and using the interpolation function `interp1` for the evaluation of the integral. The functions μ_{inf}, μ_{sup} are evaluated, respectively, by the methods `measure inf` and `measure_sup`, both returning a function generated by interpolation of the evaluated values, using `interp1` and the interpolation method `interpmet`.

EXAMPLE 1.3.– Let us consider $\Omega = (-1,1)$ and $f(x) = x^2$. Then,

$$\mu_{inf}(y) = \begin{cases} 0, if\ y \leq 0 \\ 2\sqrt{y}, if\ 0 < y < 1 \\ 2, if\ y \geq 1 \end{cases} ; \quad \mu_{sup}(y) = \begin{cases} 2, if\ y \leq 0 \\ 2(1-\sqrt{y}), if\ 0 < y < 1 \\ 0, if\ y \geq 1 \end{cases}$$

For $N = 1024$, $M = 128$, $y_{\min} = -0.5$, $y_{\max} = 1.5$, $a = -1, b = 1$, the code:

```
hx = (b-a)/N;
x = a + hx*(0:N);
f = x.^2;
p.x = x;
p.dim = 1;
```

```
data.points = p;
data.values = f;
mu1 = lebesgue.measure_inf(y,data,'linear');
mu2 = lebesgue.measure_sup(y,data,'linear');
```

furnishes

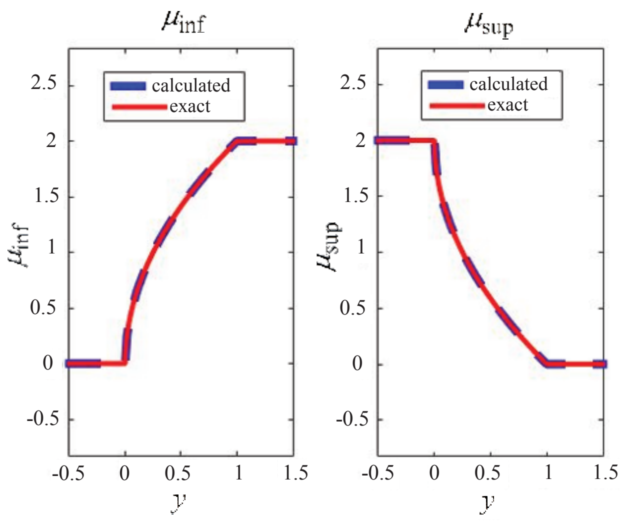


Figure 1.4. *Lebesgue measure obtained in example 1.3*

The code:

```
I1     =     lebesgue.integrate(y,data,'inferior',
'linear');
I2     =     lebesgue.integrate(y,data,'superior',
'linear');
I3 = lebesgue.integrate(y,data,'mean','linear')
```

furnishes `I1 = 0.66746`, `I2 = 0.66746` and `I3 = 0.66746`. The exact value is `2/3 ≈ 0.66667`. These results may be improved by refining the partitions, namely on the vertical axis. For instance, $N = 4096$, $M = 512$ lead to `I1 = 0.66677`, `I2 = 0.66677` and `I3 = 0.66677`. ■

When regular functions are considered – this is the case in example 1.3 – Lebesgue's approach is not the more efficient one. For instance, Riemann's approach furnishes more precise results. However, for functions having discontinuity points, Lebesgue's approach may be interesting (see examples below).

EXAMPLE 1.4.– Let us consider $\Omega = (0,2)$ and $f(x) = 1/\sqrt{x}$. Then,

$$\mu_{inf}(y) = \begin{cases} 0, if\ y \leq 1/\sqrt{2} \\ 2 - 1/y^2, otherwise \end{cases} \quad ; \quad \mu_{sup}(y) = \begin{cases} 2, if\ y \leq 1/\sqrt{2} \\ 1/y^2, otherwise \end{cases} .$$

Using $M = 256$, $y_{min} = 0$, $y_{max} = 100$, $N = 4096$, we obtain `I1 = 2.7905`, `I2 = 2.7906`, `I3 = 2.7906`, while the trapezoidal rule `trapz(x,f)` returns the value `Inf`. The exact value is `2*sqrt(2) ≈ 2.8284`. For $M = 1024$, N = 4096, we have `I1 = I2 = I3 = 2.7959` and `trapz (x,f)` returns the value `Inf`. The measures μ_{inf} and μ_{sup} are in Figure 1.5. ■

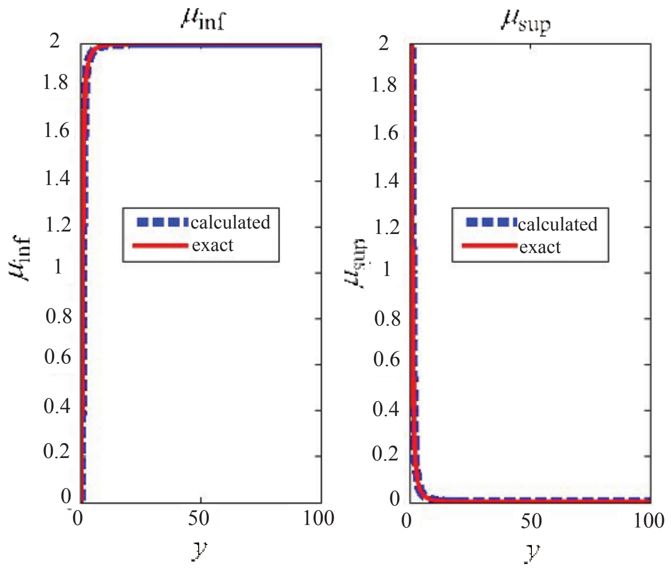


Figure 1.5. *Results for example 1.4*

EXAMPLE 1.5.– Let us consider $\Omega = (0,20)$ and $f(x) = log(|sin(\pi x)|)$. Then,

$$\mu_{inf}(y) = \begin{cases} (b-a), if\ y \geq 0 \\ \frac{2}{\pi}(b-a)\, sin^{-1}(e^y), otherwise \end{cases};$$

$$\mu_{sup}(y) = \begin{cases} 0, if\ y \geq 0 \\ (b-a)\left(1 - \frac{2}{\pi} sin^{-1}(e^y)\right), otherwise \end{cases}$$

In this case, when using N = 1024, $y_{min} = -2.5$, $y_{max} = 0.5$, M = 256, we obtain `I1 = I2 = I3 =-10.121`, while `trapz(x,f)` returns the value-`Inf`. For N = 4096, $y_{min} = -100$, $y_{max} = 0.5$, M = 1024, the values are `I1 = -14.3736, I2 = -14.3733, I3 = -14.3735` and `trapz(x,f)` returns the value-`Inf`. The measures μ_{inf} and μ_{sup} are in Figure 1.6. ■

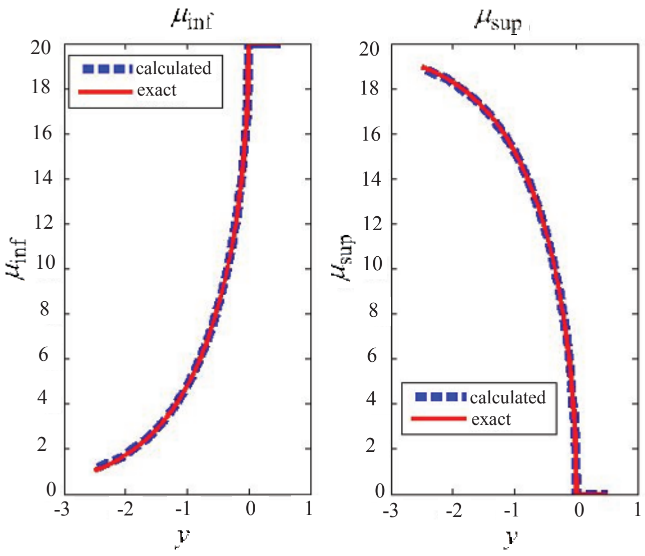


Figure 1.6. *Lebesgue measure in example 1.5*

EXAMPLE 1.6.– Let us consider $\Omega = (0,20)$ and $f(x) = sign(\sin(\pi x))$. Then,

$$\mu_{inf}(y) = \begin{cases} 0, if\ y \le -1 \\ 1, if\ -1 < y < 1 \\ 2, if\ y \ge 1 \end{cases} ; \quad \mu_{sup}(y) = \begin{cases} 2, if\ y \le -1 \\ 1, if\ -1 < y < 1 \\ 0, if\ y \ge 1 \end{cases} .$$

In this case, using N = 1024, $y_{min} = -1.5$, $y_{max} = 1.5$, $M = 256$, we obtain `I1= I2 = I3 =0.0097659` and `trapz(x,f)` returns the value `0.0098`. The exact value is `0`. The measures μ_{inf} and μ_{sup} are given in Figure 1.7. ■

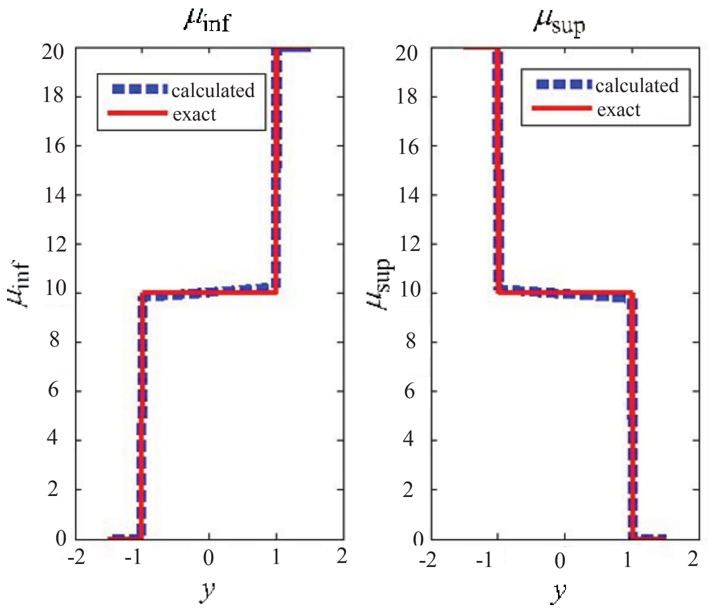


Figure 1.7. *Results for Lebesgue measure in example 1.6*

One of the interesting features of Lebesgue integrals is the fact that multidimensional integration reduces to 1D integration: indeed, only the evaluation of $\mu_{\text{inf}}, \mu_{\text{sup}}$ involves multidimensional calculations.

Once these quantities are obtained, equation [1.4] or equation [1.7] is used in order to evaluate the integral – only 1D integration is involved in this calculation. Examples are given below.

EXAMPLE 1.7.– Let us evaluate:

$$I = \int_0^1 dx \int_0^2 (x^2 + y^2)\, dy$$

by using the points `x=0:step:1;` and `y=0:step:2;` with `step = 0.01`. Assuming that `F(i,j)=x(i)^2+y(j)^2`, the code:

```
p.x = x;
p.y = y;
p.dim  = 2;
yleb = 0:0.1:5;
data.points = p;
data.values = F;
I1 = lebesgue.integrate(yleb,data,'inferior','linear');
I2 = lebesgue.integrate(yleb,data,'superior','linear');
I3 = lebesgue.integrate(yleb,data,'mean','linear')
```

produces `I1=I2=I3=3.3334.` The exact value is $\frac{10}{3} \approx 3.3333$. ■

EXAMPLE 1.8.– Let us evaluate:

$$I = \int_0^1 dx \int_0^2 dy \int_0^3 (x^2 + y^2 + z^2)\, dz$$

by using the points `x=0:step:1;,y=0:step:2;, z=0:step:3;,` with `step = 0.05`. Assuming that `F(i,j,k)=x(i)^2+y(j)^2+z(k)^2,` the code:

```
p.x = x;
p.y = y;
p.z = z;
p.dim  = 3;
yleb = 0:0.2:14;
data.points = p;
data.values = F;
```

```
I1 = lebesgue.integrate(yleb,data,'inferior','linear');
I2 = lebesgue.integrate(yleb,data,'superior','linear');
I3 = lebesgue.integrate(yleb,data,'mean','linear')
```

produces `I1=I2=I3=28.0094`.

Using as interpolation method `'cubic'` (or `'pchip'`, for newer versions of Matlab®) produces the result `28.0090`. The exact value is 28. ■

1.5. Matlab® classes for evaluation of the integrals when f is defined by a subprogram

Let us assume that the values of $f(x)$ are furnished by a subprogram `f.m` which receives as argument a vector `x` and returns a value `f` (x).

The internal functions for numerical integration are given in Table 1.1. Notice that `ay,by,az, bz` are function handles, i.e. anonymous functions or subprograms evaluating functions.

In order to use these intrinsic functions of Matlab®, we must transform `f`. Indeed, the intrinsic functions of Matlab® assume as arguments `(x,y)` or `(x,y,z)` and not a vector `x`. Moreover, they assume that the functions may receive multidimensional arrays of data and return a corresponding array of results. The adaptation is made by the subprograms of class `spam`. For instance, we may use the class given in program 1.4.

Intrinsic function	evaluates
`integral(f,a,b)` or `quadgk(f,a,b)`	$\int_a^b f(x)dx$
`integral2(f,ax,bx,ay,by)` or `quad2d(f,ax,bx,ay,by)`	$\int_{ax}^{bx} dx \int_{ay(x)}^{by(x)} f(x,y)dy$
`integral3(f,ax,bx,ay,by,az,bz)`	$\int_{ax}^{bx} dx \int_{ay(x)}^{by(x)} \int_{az(x,y)}^{bz(x,y)} f(x,y,z)dxdydz$

Table 1.1. *Intrinsic functions for the evaluation of integrals*

```
classdef subprogram_integration
    properties
        limits
        integrand
    end
    properties(Dependent)
        result
    end
    %
    methods
        function obj = subprogram_integration(f,xlim)
            if isnumeric(f)
                f = @(x) f;
            end;
            obj.integrand = f;
            obj.limits = xlim;
        end
        function v = get.result(obj)
            f = obj.integrand;
            xlim = obj.limits;
            a = xlim.lower;
            b = xlim.upper;
            n = xlim.dim;
            switch n
                case 1
                    fs = @(x) spam.points('table',
f,x);
                    v = integral(fs,a.x,b.x);
                case 2
                    fs = @(x,y) spam.points('table',
f,x,y);
                    if isnumeric(a.y)
                        a.y = @(x) a.y;
                    end;
                    ymin = @(x) spam.points('table',
a.y,x);
                    if isnumeric(b.y)
                        b.y = @(x) b.y;
                    end;
                    ymax = @(x) spam.points('table',
b.y,x);
```

```
                    v = integral2(fs,a.x,b.x,
ymin,ymax);
                    %   or quad2d(fs,a.x,b.x,
ymin,ymax);
                    %   or dblquad(fs,a.x,b.x,
a.y,b.y);
                case 3
                    fs = @(x,y,z) spam.points('table',
f,x,y,z);
                    if isnumeric(a.y)
a.y = @(x) a.y;
                    end;
                    ymin = @(x) spam.points('table',
a.y,x);
                    if isnumeric(b.y)
b.y = @(x) b.y;
                    end;
                    ymax = @(x) spam.points('table',
b.y,x);
                    if isnumeric(a.z)
                        a.z = @(x) a.z;
                    end;
                    zmin = @(x,y) spam.points('table',
a.z,x,y);
                    if isnumeric(b.z)
                        b.z = @(x) b.z;
                    end;
                    zmax = @(x,y)
spam.points('table',b.z,x,y);
                    v =
integral3(fs,a.x,b.x,ymin,ymax,zmin,zmax);
                    %  or
triplequad(fs,a.x,b.x,a.y,b.y,a.z,b.z);
            end;
        end
    end
end
```

Program 1.4. *A class for the integration of functions defined by subprograms*

In this program, `limits` (or `xlim`) is a data structure with the properties `dim`, `lower`, `upper`, which define dimension, lower and upper bounds for the integral, respectively.

EXAMPLE 1.9.– Let us consider the integration of:

$$f(x) = x_1 e^{-x_2} \quad on \ \Omega = \{x : 0 \le x_1 \le 1 \,, 0 \le x_2 \le x_1\}.$$

In this case,

```
xlim.lower.x = 0;
xlim.lower.y = 0;
xlim.upper.x = 1;
xlim.upper.y = @(x) x;
xlim.dim = 2;
f = @(x) x(1)*exp(-x(2));
sp = subprogram_integration(f,xlim);
v = sp.result
```

Program 1.5. *An example of evaluation*

This program produces `v = 0.2358`. The exact value is $\frac{2}{e} - \frac{1}{2} \approx 0.2358$. ■

EXAMPLE 1.10.– Let us consider the evaluation of the volume of a sphere. The code:

```
xlim.lower.x = -1;
xlim.lower.y = @(x) -sqrt(1- x(1)^2);
xlim.lower.z = @(x) -sqrt(1- x(1)^2  -x(2)^2);
xlim.upper.x = 1;
xlim.upper.y = @(x) sqrt(1- x(1)^2);
xlim.upper.z = @(x,y) sqrt(1- x(1)^2  - x(2)^2);
xlim.dim = 3;
f = 1;
sp = subprogram_integration(f,xlim);
v = sp.result
```

Program 1.6. *Evaluation of a 3D integral*

produces the result `v = 4.1888`. The exact result is $\frac{4\pi}{3} \approx 4.1888$. But the code:

```
xlim.lower.x = -1;
xlim.lower.y = -1;
xlim.lower.z = -1;
xlim.upper.x = 1;
xlim.upper.y = 1;
xlim.upper.z = 1;
xlim.dim = 3;
f = @(x) 1*(sum(x.^2)<1);
sp = subprogram_integration(f,xlim);
v = sp.result
```

Program 1.7. *Evaluation of a 3d integral with a discontinuous function*

produces an unsuccessful run: the result is `v = NaN`. ■

The intrinsic functions for older versions of Matlab® are given in Table 1.2. `triplequad` adapts to the same use as `integral3` by considering a box containing the region of integration and extending the integrand by zero outside the region of integration.

Notice that both `dblquad` *and* `triplequad` *assume that f accepts as input a vector* `x` *and scalars* `y` *and* `z` *and returns a vector of results, so that the preceding functions spam2 and spam3 have to be modified in order to satisfy these requirements.*

Intrinsic function	evaluates
`quad(f,a,b) orquadgk(f,a,b)`	$\int_a^b f(x)dx$
`quad2d(f,ax,bx,ay,by)`	$\int_{ax}^{bx} dx \int_{ay(x)}^{by(x)} f(x,y)dy$
`dblquad(f,ax,bx,ay,by)`	$\int_{ax}^{bx} dx \int_{ay}^{by} f(x,y)dy$
`triplequad(f,ax,bx,ay,by,az,bz)`	$\int_{a_x}^{b_x} dx \int_{a_y}^{b_y} \int_{a_z}^{b_z} f(x,y,z)dxdydz$

Table 1.2. *Intrinsic function for older versions of Matlab®0*

1.6. Matlab® classes for partitions including the evaluation of the integrals

Partitions may be assimilated to finite element meshes. For instance, in Matlab®, we may define classes having convenient properties and methods. For the situation where $\Omega = (a, b) \subset \mathbb{R}$, an example of such a class is given in program 1.8. Notice that the class includes a method for integration (it assumes finite values for all the elements of F).

```
classdef interval_partition
    properties
        x
    end
    properties(Constant)
        dim = 1
    end
    properties(Dependent)
        min
        max
        number_points
        number_intervals
        measure
    end
    methods
        function obj = interval_partition(points)
            obj.x = points;
        end
      function v =
integration(obj,method,F,np,metod,interpmetod)
            data.points = obj;
            data.values = F;
            switch method
                case 'riemann'
                    v = riemann.trpzd(data);
                case 'mean'
                    v = riemann.mean_value(data);
                case 'lebesgue'
                    m1 = min(F);
                    m2 = max(F);
                    pas = (m2 - m1)/np;
                    y = m1:pas:m2;
             v = lebesgue.integrate(y,data,metod,
interpmetod);
            end
        end
        function obj = set.x(obj,coordinates)
            xx = sort(coordinates);
            obj.x = xx;
        end
        function v = get.min(obj)
            v = min(obj.x);
        end
        function v = get.max(obj)
            v = max(obj.x);
        end
        function v = get.number_points(obj)
            v = length(obj.x);
        end
        function v = get.number_intervals(obj)
            v = length(obj.x)-1;
        end
        function v = get.measure(obj)
            v = max(obj.x) - min(obj.x);
        end
    end
end
```

Program 1.8. *Definition of a class for partition of intervals*

For instance, the code:

```
pas = 0.05;
x = 0:pas:1;
p = interval_partition(x);
```

creates the structure `p` of type `interval_partition`. Then,

```
f = @(x) exp(x);
F = spam.mapspam('vector',f,x);
```

```
v1 = p.integration('riemann',F)
v2 = p.integration('lebesgue',F,20,'inferior','pchip')
v3 = p.integration('mean',F)
```

evaluates the integral $I = \int_0^1 e^x dx = e - 1 \approx 1.7183$ using the trapezoidal rule or Lebesgue's method with 20 intervals on the vertical axis. For the points given, the result is `v1 = 1.7186, v2 = 1.7187, v3=1.7253`. It may be improved by considering a partition containing a larger number of points. For instance, if `pas = 0.01;` the result is `v1 = v2 = 1.7183, v3 = 1.7197.`

For $\Omega = (a_1, b_1) \times (a_2, b_2)$, we may define a class as follows:

```
classdef rectangle_partition
    properties
        x
        y
    end
    properties(Constant)
        dim = 2
    end
    properties(Dependent)
        number_points
        measure
    end
    methods(Access = private)
        function v = quadinds(obj,i,j)
            v = [
                i j
                i+1 j
                i+1 j+1
                i j+1
                ];
        end
        function [ind1, ind2] = tri24(obj,i,j)
            ind1 = [ i j; i+1 j; i j+1 ];
            ind2 = [ i+1 j; i+1 j+1; i j+1 ];
        end
```

```
function [ind1, ind2] = tri13(obj,i,j)
    ind1 = [ i j; i+1 j+1; i j+1 ];
    ind2 = [ i j; i+1 j; i+1 j+1 ];
end
function v = sumind(obj,F,ind)
    v = 0;
    for i = 1:size(ind,1)
        aux = F(ind(i,1), ind(i,2));
        v = v + aux;
    end;
end
function v = st24(obj,F)
    xx =  obj.x;
    yy = obj.y;
    s = 0;
    for i = 1: length(xx)-1
        hx = xx(i+1) - xx(i);
        for j = 1: length(yy)-1
            hy = yy(j+1) - yy(j);
            [ind1, ind2] = obj.tri24(i,j);
            surft = hx*hy/2;
            aux1 = obj.sumind(F,ind1);
            aux2 = obj.sumind(F,ind2);
            s = s + (aux1 + aux2)*surft;
        end
    end;
    v = s/3;
end
function v = st13(obj,F)
    xx =  obj.x;
    yy = obj.y;
    s = 0;
    for i = 1: length(xx)-1
        hx = xx(i+1) - xx(i);
        for j = 1: length(yy)-1
            hy = yy(j+1) - yy(j);
            surft = hx*hy/2;
            [ind1, ind2] = obj.tri13(i,j);
            aux1 = obj.sumind(F,ind1);
            aux2 = obj.sumind(F,ind2);
            s = s + (aux1 + aux2)*surft;
```

```
                end
            end;
            v = s/3;
        end
        function v = sq4(obj,F)
            xx = obj.x;
            yy = obj.y;
            s = 0;
            for i = 1: length(xx)-1
                hx = xx(i+1) - xx(i);
                for j = 1: length(yy)-1
                    hy = yy(j+1) - yy(j);
                    surfq = hx*hy;
                    ind = obj.quadinds(i,j);
                    saux = obj.sumind(F,ind);
                    aux = saux*surfq;
                    s = s + aux;
                end
            end;
            v = s/4;
        end
    end
    methods
        function obj = rectangle_partition(x1,x2)
            obj.x = x1;
            obj.y = x2;
        end
      function v =
integration(obj,method,F,np,metod,interpmetod)
            data.points = obj;
            data.values = F;
            switch method
                case 'trapezoid'
                    v = riemann.trpzd(data);
                case 'mean'
                    v = riemann.mean_value(data);
                case 'tri24'
                    v = obj.st24(F);
                case 'tri13'
                    v = obj.st13(F);
                case 'quad'
```

```
                        v = obj.sq4(F);
                    case 'lebesgue'
                        m1 = min(min(F));
                        m2 = max(max(F));
                        pas =  (m2 - m1)/np;
                        yy = m1:pas:m2;
                  v =
lebesgue.integrate(yy,data,metod,interpmetod);
                    otherwise
                        v = [];
                        disp('invalid method');
                end;
            end
            function v = get.number_points(obj)
                v.x =length(obj.x);
                v.y =length(obj.y);
            end
            function v = get.measure(obj)
                dx = max(obj.x) - min(obj.x);
                dy = max(obj.y) - min(obj.y);
                v = dx*dy;
            end
        end
end
```

Program 1.9. *A class for partitions of rectangles*

For instance, the code:

```
pas = 0.05;
x = 0:pas:1;
y = 0:pas:2;
pp = rectangular_partition(x,y);
```

creates the structure `pp` of type `rectangular_partition` and the code:

```
f = @(x) exp(x(1) - x(2));
F = spam.mapspam('vector',f,pp);
v1 = pp.integration('trapezoid',F);
v2 = pp.integration('tri13',F);
```

```
v3 = pp.integration('tri24',F);
v4 = pp.integration('quad',F);
v5= pp.integration('lebesgue',F,40,'inferior','pchip');
v6= pp.integration('lebesgue',F,40,'superior','pchip');
v7 = pp.integration('lebesgue',F,40,'mean','pchip');
```

produces `v1 = v4 = 1.4864, v2 = 1.4860, v3 = 1.4867, v5 = v6 = v7 = 1.4861`. The exact value is $(e-1)(1-e^{-2}) \approx 1.4857$. When using `pas = 0.01`, the result is `v1 = v2 = v3 = v4 = 1.4858, v5=v6=v7=1.4856.`

EXAMPLE 1.11.– Let us consider the integration of $f(x) = x_1 e^{-x_2}$ on $\Omega = (0,1) \times (0,2)$. The exact value of the integral is $\frac{1}{2}(e^2 - 1)$. Using `pas = 0.05`, the results are `v1=v2=v4=v5=3.1946, v2=0.8597, v6=3.1942`. For `pas=0.01, v1=v2=v4=v5=3.1946, v3=3.1945, v6=3.1942, v7=3.1944.` ■

EXAMPLE 1.12.– Let us consider the integration of

$$f(x) = \begin{cases} x_1 e^{-x_2}, if\ x_2 \leq x_1 \\ 0, otherwise\ . \end{cases}$$

on $\boldsymbol{\Omega} = (\mathbf{0},\mathbf{1}) \times (\mathbf{0},\mathbf{1})$. This situation corresponds to the integration of

$$f(x) = x_1 e^{-x_2} \quad on\ \ \Omega = \{x : 0 \leq x_1 \leq 1\ , 0 \leq x_2 \leq x_1\}.$$

and the exact value of the integral is $\frac{2}{e} - \frac{1}{2} \approx 0.2358$. Using `pas = 0.01` and `np=100` intervals on the vertical axis, the results are `v1=v2=v4=0.2344, v3=0.2345, v5 = 0.2363, v6 = 0.2344, v7 = 0.2354`. Due to discontinuity, finer partitions are requested for a good precision. ■

In 3D situations, $\Omega = (a_1, b_1) \times (a_2, b_2) \times (a_3, b_3)$, we may define a class containing methods for integration by trapezoidal rule or integration using a tetrahedral or hexahedral mesh as follows:

```
classdef cobble_partition
    properties
        x
        y
        z
    end
    properties(Constant)
        dim = 3
    end
    properties(Dependent)
        number_points
        measure
    end
    %
    methods(Static, Access = Private)
        function v =  hexainds(i,j,k)
            v = [
                i j k
                i+1 j k
                i+1 j+1 k
                i j+1 k
                i j k+1
                i+1 j k+1
                i+1 j+1 k+1
                i j+1 k+1
                ];
        end
        function v = tetra5()
            v = [
                1 2 4 5
                2 3 4 7
                2 7 5 6
                4 5 7 8
                2 4 5 7
                ];
        end
        function v = tetra6()
            v = [
                3 8 4 2
                1 8 2 4
                7 2 8 3
```

```
                 1  2  8  5
                 2  5  6  8
                 2  8  6  7
                 ];
        end
        function v = sumind(obj,F,ind)
            v = 0;
            for i = 1:size(ind,1)
                aux = F(ind(i,1), ind(i,2), ind(i,3));
                 v = v + aux;
            end;
        end
        function v = vol_tetra(obj,x,y,z,ind)
            xx = x(ind(:,1));
            yy = y(ind(:,2));
            zz = z(ind(:,3));
            v = abs(det([xx' yy' zz' ones(size
(xx'))]))/6;
        end
        function v = sumtetra(obj,x,y,z,F,
ind,tetras,volh)
            v = 0;
            sv = 0;
            for iaux = 1: size(tetras,1)
                iii = ind(tetras(iaux,:),:);
                saux = obj.sumind(F,iii);
                volt = obj.vol_tetra(x,y,z,iii);
                sv = sv + volt;
                v = v + saux*volt;
            end
            v = v*volh/sv;
        end
        function v = st5(obj,F)
            x = obj.x;
            y = obj.y;
            z = obj.z;
            tetras = obj.tetra5();
            s = 0;
            for i = 1: length(x)-1
                for j = 1: length(y)-1
```

```
                    for k = 1: length(z)-1
                        ind = obj.hexainds(i,j,k);
                        hx = x(i+1) - x(i);
                        hy = y(j+1) - y(j);
                        hz = z(k+1) - z(k);
                        volh = hx*hy*hz;
                      saux =
obj.sumtetra(x,y,z,F,ind,tetras,volh);
                        s = s + saux;
                    end;
                end
            end;
            v = 0.25*s;
        end
        function v = st6(obj,F)
            x = obj.x;
            y = obj.y;
            z = obj.z;
            tetras = obj.tetra6();
            s = 0;
            for i = 1: length(x)-1
                for j = 1: length(y)-1
                    for k = 1: length(z) - 1
                        ind = obj.hexainds(i,j,k);
                        hx = x(i+1) - x(i);
                        hy = y(j+1) - y(j);
                        hz = z(k+1) - z(k);
                        volh = hx*hy*hz;
                      saux =
obj.sumtetra(x,y,z,F,ind,tetras,volh);
                        s = s + saux;
                    end;
                end
            end;
            v = 0.25*s;
        end
        function v = sh8(obj,F)
            x = obj.x;
            y = obj.y;
            z = obj.z;
            s = 0;
            for i = 1: length(x)-1
                hx = x(i+1) - x(i);
                for j = 1: length(y)-1
                    hy = y(j+1) - y(j);
                    for k = 1: length(z) - 1
                        hz = z(k+1) - z(k);
                        volh = hx*hy*hz;
                        ind = obj.hexainds(i,j,k);
                        saux = obj.sumind(F,ind);
                        s = s + saux*volh;
                    end
                end;
            end;
            v = s/8;
        end
    end
    methods
        function obj = cobble_partition(x1,x2,x3)
            obj.x = x1;
            obj.y = x2;
            obj.z = x3;
        end
        function v =
integration(obj,method,F,np,metod,interpmetod)
            data.points = obj;
            data.values = F;
            switch method
                case 'trapezoid'
                    v = riemann.trpzd(data);
                case 'mean'
                    v = riemann.mean_value(data);
                case 'tetra5'
                    v = obj.st5(F);
                case 'tetra6'
                    v = obj.st6(F);
                case 'hexa'
                    v = obj.sh8(F);
                case 'lebesgue'
                    m1 = min(min(min(F)));
                    m2 = max(max(max(F)));
                    pas =  (m2 - m1)/np;
                    yy = m1:pas:m2;
                v =
lebesgue.integrate(yy,data,metod,interpmetod);
                otherwise
                    v = [];
                    disp('invalid method');
            end;
        end
        function v = get.number_points(obj)
            v.x =length(obj.x);
            v.y =length(obj.y);
            v.z =length(obj.z);
        end
        function v = get.measure(obj)
            dx = max(obj.x) - min(obj.x);
            dy = max(obj.y) - min(obj.y);
            dz = max(obj.z) - min(obj.z);
            v = dx*dy*dz;
        end
    end
end
```

Program 1.10. *A class for three-dimensional partitions*

For instance, the code:

```
pas = 0.05;
x = 0:pas:1;
y = 0:pas:2;
z = 0:pas:3;
ppp = cobble_partition(x,y,z);
```

creates the structure pp of type `cobble partition`. The code:

```
v1 = ppp.integration('trapezoid',F);
v2 = ppp.integration('tetra5',F);
v3 = ppp.integration('tetra6',F);
v4 = ppp.integration('hexa',F);
v5=ppp.integration('lebesgue',F,40,'inferior','pchip');
v6=ppp.integration('lebesgue',F,40,'superior','pchip');
```

```
v7=ppp.integration('lebesgue',F,40,'mean','pchip');
```

produces `v1 = v4 = 13.3791`, `v2 = 13.3789`, `v3 = 13.3763`, `v4 = 13.3791`, `v5 = 13.4117`, `v6 =13.4086`, `v7 = 13.4102`. The exact value is $9(e-1)(1-e^{-2}) \approx 13.3716$.

EXAMPLE 1.13.– Let us consider the evaluation of the 3D integral:

$$I = \iiint_{\|x\|<1} 1dx$$

We may consider $\Omega = (-1,1) \times (-1,1) \times (-1,1)$ and

$$f(x) = \begin{cases} 1, if\ \|x\| < 1, \\ 0,\ otherwise. \end{cases}$$

Then, we evaluate:

$$I = \iiint_{\Omega} f(x)dx\ .$$

Using `pas = 0.05`, the result obtained is `v1 = v2 = v3 = v4 = 4.1714`, `v5 = v6 = v7 = 4.1724`. The exact result is $\frac{4\pi}{3} \approx 4.1888$. ■

REMARK.– In some situations, the available data is not on a grid, but just on sparse points. This situation is considered in Chapter 3. ■