Approximation and Integration

Felix Kubler[1]

[1]DBF, University of Zurich and Swiss Finance Institute

January 25, 2017

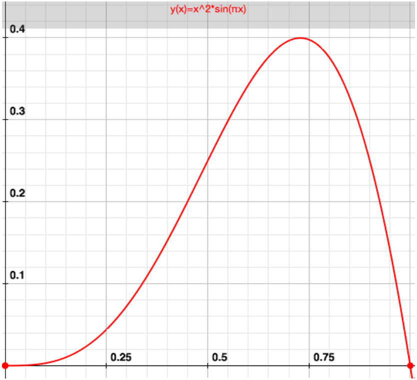
Numerical Integration

- In economic problems with uncertainty, we typically assume that agents solve complicated integrals
- Computational economists need learn methods for numerical integration:
  - Monte Carlo
  - Quadrature and Cubature

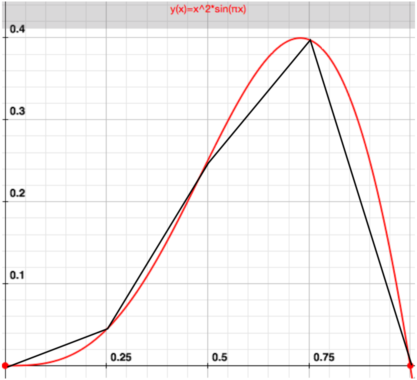
Function approximation

- As it turns out one way to integrate complicated functions is to approximate them by simple functions whose integral we know!
- Later this week, we learn about other reasons why we want to approximate functions: Dynamic programming, projection methods etc...
- Will mostly focus on the one-dimensional case today and discuss higher dimensional approximation later
- How did you approximate functions in kindergarten?

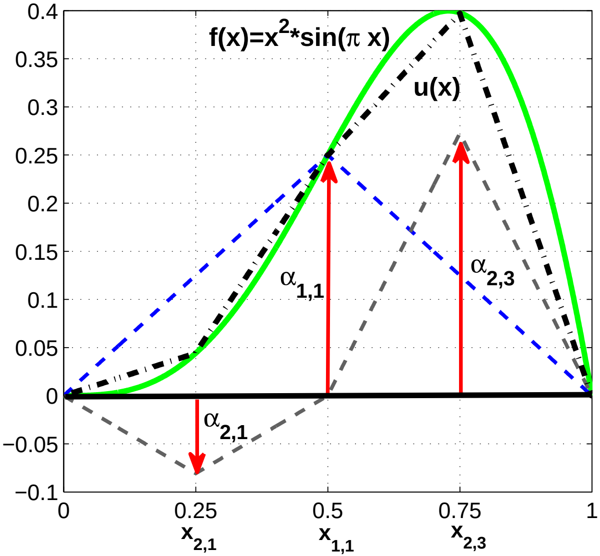
Piecewise linear interpolation



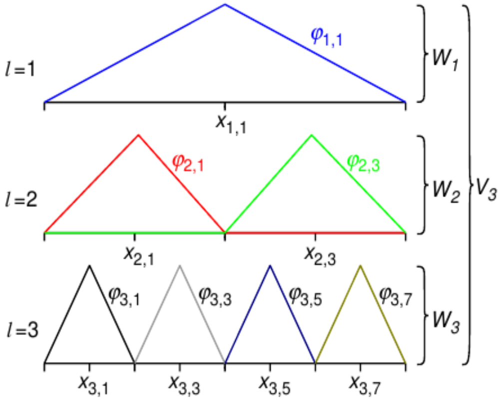
Piecewise linear interpolation



Piecewise linear interpolation



Piecewise linear interpolation



Piecewise linear interpolation

- In one dimension, we take as basic function on $[-1, 1]$

$$\phi(x) = \max(0, 1 - |x|)$$

and twist them to generate a family of basis functions on $[0, 1]$

$$\phi_{l,i}(x) = \phi(2^l x - i), i = 1, ..., 2^l - 1, \ i \text{ odd}$$

- Define

$$I_l = \{i \in \mathbb{N} : 1 \leq i \leq 2^l - 1, i \text{ odd}\}$$

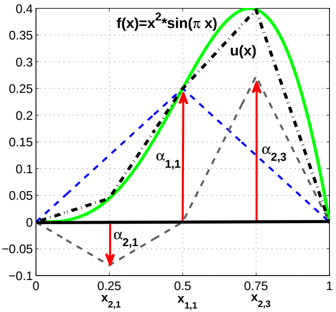
and

$$W_l = \text{span}\{\phi_{l,i}, i \in I_l\}$$

The space of piecewise linear functions is then

$$V_n = \bigoplus_{l \leq n} W_l$$

Piecewise linear interpolation - coefficients



- $f(x) \simeq u(x) = \sum_{k=1}^{l} \sum_{i \in I_k} \alpha_{k,i} \phi_{k,i}(x)$
- The coefficients, $\alpha_{k,i}$ are *hierarchical surpluses*. They correct the interplant of level $l-1$ at $x_{l,i}$ to the actual value of $f(x_{l,i})$
- Become small as approximation becomes better

Piecewise linear approximation

- Given the basis functions $\phi_{k,i}(x)$, we chose the $\alpha_{k,i}$ so that the approximating function matches the true function at a predetermined set of points
- Beyond Kindergarten, one can think of other good approximations, e.g. some norm on function space
- Which norm to take? We'll briefly talk about 2-norm, 1-norm and sup-norm.

Uniform approximation

- Piecewise linear approximation seems a bit odd if the true function is sufficiently smooth.
- Even if the function is non-linear, one can approximate it arbitrarily well by polynomials. Weierstrass Theorem: Given any continuous function $f : [a, b] \to \mathbb{R}$ and any $\epsilon > 0$ there exists a polynomial $p(x)$ such that
$$\max_{x \in [a,b]} |f(x) - p(x)| < \epsilon$$
- How do we find this polynomial?
- In general that is tough, but can do it on a finite set of points - then one can also consider least squares, or least first power approximation.

Polynomial interpolation 1

- We want to approximate a smooth function $f : [-1, 1] \to \mathbb{R}$ by a polynomial of degree d.
- In order to do so, we will interpolate n of its function values, i.e. given some points $x_i, f(x_i)$, we try to find a polynomial that matches the function values at the points x_i.
- Polynomial is uniquely pinned down by $d + 1$ distinct points
- Alternatively we could take more than $d + 1$ points and do least square – for now we want to consider interpolation

Polynomial interpolation 2

- Interpolate n points by a univariate polynomial of degree $n-1$,

$$p_{n-1}(x) = \sum_{j=0}^{n-1} \theta_j x^j$$

- To find the unknown n coefficients $(\theta_0, \ldots, \theta_{n-1})$, we could use the n equations

$$y_i = \sum_{j=0}^{n-1} \theta_j (x_i)^j \text{ for } i = 1, \ldots, n$$

- Unfortunately, the 'Vandermonde' matrix

$$V = \begin{pmatrix} 1 & x_1 & x_1^2 & \ldots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \ldots & x_2^{n-1} \\ \vdots & \vdots & \ddots & \vdots & \\ 1 & x_n & x_n^2 & \ldots & x_n^{n-1} \end{pmatrix}$$

is typically extremely badly conditioned

Polynomial interpolation 3

- Better to use Lagrange polynomials.
- Let the i'th Lagrange polynomial be defined by

$$l_i(x) = \prod_{j=1, j \neq i}^{n} \frac{x - x_j}{x_i - x_j}.$$

- Note that $l_i(x_j) = 1$ if and only if $i = j$ and it is zero otherwise.
- Therefore we can simply set

$$g(x) = \sum_{i=1}^{n} g(x_i) l_i(x).$$

- Simplest way to get an interpolation polynomial, but evaluation of l_i could be costly..

Orthogonal polynomials

- A good choice of polynomials are orthogonal under some inner product, in fact we have

$$\int_{-\infty}^{\infty} e^{-x^2} I_m(x) I_n(x) = 0 \text{ if } m \neq n$$

- Define Legendre polynomials as $P_0(x) = 1$, $P_1(x) = x$ and

$$P_{n+1}(x) = \frac{1}{n+1}\left((2n+1)xPn(x) - nP_{n-1}(x)\right)$$

  We have

$$\int_{-1}^{1} P_m(x) P_n(x) dx = \frac{2}{2n+1} \delta_{mn}$$

- Define Chebychev terms as follows. Let $T_0(x) = 1$, $T_1(x) = x$, and recursively $T_{i+1}(x) = 2xT_i(x) - T_{i-1}(x), \quad i = 1, ..., n$. We have

$$\int_{-1}^{1} (1 - x^2)^{-1/2} T_m(x) T_n(x) dx = 0 \text{ if } m \neq n.$$

Polynomial Interpolation 4

- Define Chebychev terms as follows. Let $T_0(x) = 1$, $T_1(x) = x$, and recursively

$$T_{i+1}(x) = 2xT_i(x) - T_{i-1}(x), \quad i = 1, ..., n.$$

- Then set

$$g(x) = \sum_{i=0}^{n-1} \xi_i T_i(x)$$

with

$$\xi_i = \frac{2}{d_i(n-1)} \sum_{j=0}^{n-1} \frac{1}{d_j} T_i(x_j) g(x_j)$$

with $d_0 = f_{n-1} = 2$ and $d_i = 1$ otherwise.

- MATLAB: *chebyshevT*(n, x)

Polynomial interpolation 5

- Can write the approximating polynomials in many different, efficient ways.
- Much more important issue which polynomials to use is at which points to interpolate the function one wants to approximate.
- While it is true that one can approximate continuous functions arbitrarily well by polynomials, it is not true that one can do so by interpolating equi-spaced points.

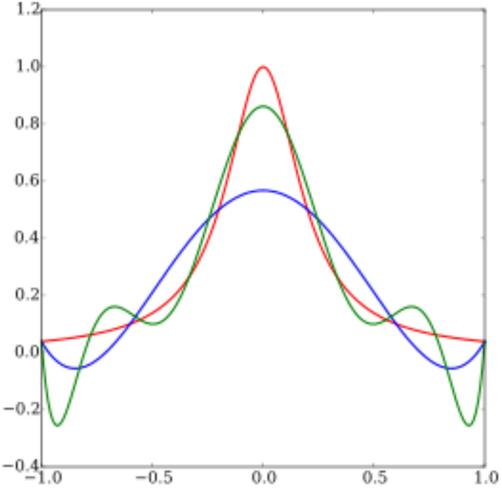
Example (Runge)

Consider the function

$$f(x) = \frac{1}{1 + 25x^2}$$

on the interval $[-1, 1]$.

Runge



Polynomial interpolation 6

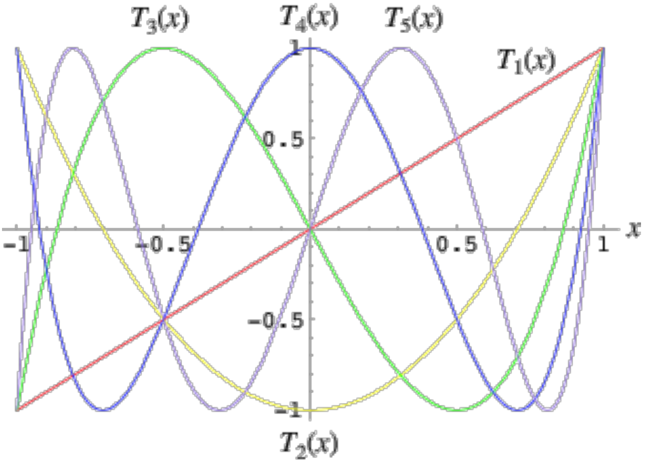
- It turns out that there are two good choices of points for which the problem does not occur.
- The first one is to use the zeros of Chebychev polynomials. These are given by

$$z_k = -\cos\left(\frac{2k-1}{2m}\pi\right), \quad k = 1, ..., m.$$

- An alternative, which is as good, is to use extrema of Chebychev polynomials. These are given by

$$z_k = -\cos\left(\frac{\pi(k-1)}{m-1}\right), \quad k = 1, ..., m, \quad m > 1.$$

Chebychev points



Some math to impress grandma

- Given a continuous function $f : [a, b] \to \mathbb{R}$ and N points $z_1, \ldots, z_N \in [a, b]$, there is a unique interpolating polynomial of degree $N - 1$, $g(x)$. There is also the best uniform approximation of f in the space of polynomials of degree N, let's call this $h(x)$.
- Under the sup-norm we have

$$\|f - g\| \leq (1 + \Lambda)\|f - h\|,$$

where Λ is the Lebesgue constant

$$\Lambda = \max_{x \in [a,b]} \sum_{j=1}^{N+1} |l_j(x)|$$

- In principle, one can compute the optimal points by minimizing the Lebesgue constant. In practice it turns out that Chebychev points have a Lebesgue constant close to the optimal one.

Limits to polynomial interpolation

- Consider the continuous function

$$f(x) = \sqrt{|x|}, x \in [-1, 1]$$

- Using Chebychev zeros, in order to approximate this function so that the sup-norm between approximation and function is less than 10^{-3} one needs 1.1 million points !!!!
- $\rightarrow$ Splines

Splines 1

- A function $s : [a, b] \to \mathbb{R}$ is a spline of order n if it is C^{n-2} on $[a, b]$ and there are nodes $a = x_0 < x_1 < ..., x_n = b$ such that $s(x)$ is polynomial of degree $n - 1$ in each subinterval $[x_i, x_{i+1}]$
- Obviously, we can use splines to interpolate points $(x_i, y_i)_{i=0}^n$.
- The easiest scheme is piece-wise linear. This can be a good way to approximate difficult functions – will get back to this soon.

Splines 2

- Want to focus on cubic splines (i.e. splines of order 4) that consist of cubic polynomials $p_1, ..., p_n$ such that

$$p_i(x_i) = y_i, \quad p_i(x_{i+1}) = y_{i+1}$$

  and for all $i = 1, ..., n-1$

$$p_i'(x_i) = p_{i-1}'(x_i), p_i''(x_i) = p_{i-1}''(x_i)$$

- Since each cubic spline has 4 unknown coefficients, we have $4n$ unknowns and $2n + 2n - 2$ equations. Hm – too many unknowns.
- We need to impose some more conditions. Natural splines impose $s''(x_0) = s''(x_n) = 0$ to pin down the two remaining unknown coefficients We can also impose $s'(x_0) = y_0', s'(x_n) = y_n'$ – these are called Hermite splines.

Splines 3: B-splines

- Can represent piecewise polynomial functions as the weighted sum of basis (B-) splines.
- Let (t_j) be a non-decreasing (knot)-sequence. The j'th B-spline of order k for (t_j) is denoted by $B_{j,k,t}$ and they can be defined recursively by

$$B_{i,k}(x) = \frac{x - x_i}{x_{i+k} - x_i} B_{i,k-1}(x) + \frac{x_{i+k+1} - x}{x_{i+k+1} - x_{i+1}} B_{i+1,k-1}(x)$$

with

$$B_{i,0}(x) = \begin{cases} 0, & x < x_i \\ 1, & x_i \leq x \leq x_{i-1} \\ 0, & x \geq x_i \end{cases}$$

- With this we can write

$$\hat{f}(x) = \sum_{i=1}^{n} \alpha_i B_{i,k}(x)$$

where the coefficients α_i can be obtained from solving a linear system of equations.

In MATLAB

- $c = polyfit(x, y, n)$ does least-squares polynomial approximation of degree n
- $y = polyval(c, x)$ evaluates the interpolant at the new points.
- Piecewise polynomial: $y1 = interp1(x, y, xn,' method')$, whereh method is one of 'linear', 'spline', 'cubic'
- Piecewise polynomial can be evaluated with *ppval*

Higher dimensions..

- Can we use polynomials to approximate 10-dimensional functions? What about 100 dimensions?
- While interval is the clear domain in one dimension, in several dimensions could have different geometric structures.
- Typically one focuses on the cube $[-1, 1]^d$ but this obviously can create problems.
- Almost nothing is known about points with low Lebesgue constant in higher dimensions...

Smolyak's method

- In the 1960's Smolyak developed a method for high dimensions that does not use Lebesgue points but it still pretty good
- A little tedious to explain and it will become clearer next week, but here are the basics...
- For details, look up
  Barthelmann, V.„ E. Novak and K. Ritter, "High dimensional polynomial interpolation on sparse grids", 2000, Advances of Computational Mathematics 12, 273-288, or
  Krueger, D. and F. Kubler, "Computing equilibrium in OLG models with stochastic production ", 2004, JEDC 28, 1411-1436.

Smolyak's method

- Sequence of 1-D interpolation points $\chi^i \subset [-1, 1]$, i=1,2,...
- Define $m_1 = 1$ and $m_i = 2^{i-1} + 1$, $i > 1$ to be the total number of elements of set χ^i
- Choose $\chi^1 = \{0\}$ and for $i > 1$, $\chi^i = \{x_1^i, ..., x_{m_i}^i\} \subset [-1, 1]$ as the set of the extrema of the Chebychev polynomials

$$x_j^i = -cos\frac{\pi(j-1)}{m_i - 1} \quad j = 1, ..., m_i$$

- So $\chi^1 = \{0\}$, $\chi^2_\Delta = \{-1, 1\}$, $\chi^3_\Delta = \{\cos(\frac{3\pi}{4}), -\cos(\frac{3\pi}{4})\}$ and

$$\chi^4_\Delta = \{-\cos(\frac{\pi}{8}), -\cos(\frac{3\pi}{8}), \cos(\frac{\pi}{8}), \cos(\frac{3\pi}{8})\}$$

Smolyak's method

- For a given level, l define a d-dimensional grid as

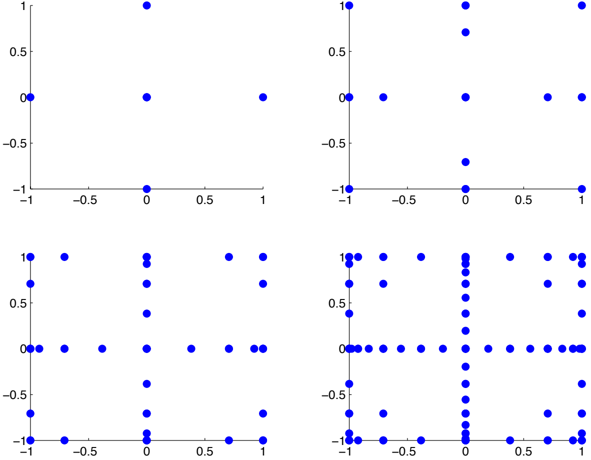
$$\mathcal{H}(l,d) = \bigcup_{|\mathbf{i}|_1 \leq d+l} (\chi^{i_1} \times \ldots \times \chi^{i_d}),$$

- The Smolyak construction of an interpolating polynomial is then

$$\mathcal{A}(l,d) = \sum_{|\mathbf{i}|_1 \leq d+l} (-1)^{d+l-|\mathbf{i}|} \begin{pmatrix} d-1 \\ d+l-|\mathbf{i}|_1 \end{pmatrix} \alpha_{i_1,\ldots,i_d} (\mathcal{U}^{i_1} \otimes \ldots \otimes \mathcal{U}^{i_d})$$

$\mathcal{U}$ are interpolating polynomials (e.g. Chebychev or Lagrange)

Smolyak's grid



Remarks

- $\mathcal{A}(2,d)$ reproduces the polynomials x_j^4, x_j^3, x_j^2, x_j, 1, $x_j^2 x_k^2$, $x_j^2 x_k$, $x_j x_k$.
  $\mathcal{A}(k,d)$ is exact for polynomials up to degree k
- The number of points in $\mathcal{H}(l,d)$ is given by
  $l = 1 : 1 + 2d$
  $l = 2 : 1 + 4d + 4\frac{d(d-1)}{2}$
  $l = 3 : 1 + 8d + 16\frac{d(d-1)}{2} + 8\frac{d(d-1)(d-2)}{6}$
- Number of points grows exponentially in l
- Method only really useful for $l = 2$ or $l = 3$.
- For large d (e.g. 50) need a lot of smoothness for a good approximation

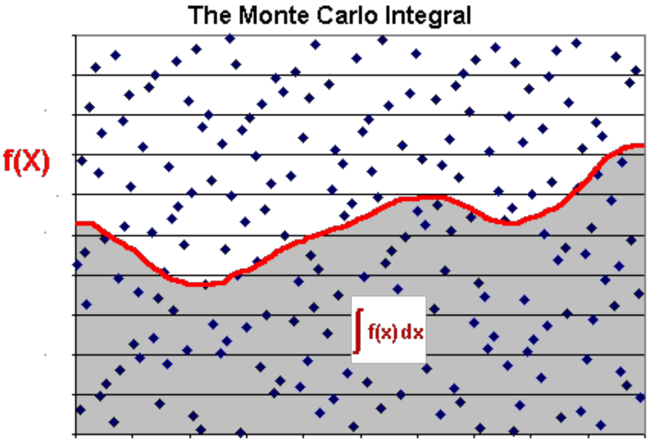
More Remarks

- Maliar and Maliar have matlab code on their website: http://stanford.edu/ maliars/Files/Codes.html

Numerical Integration

- Want to solve
$$\int_{[-1,1]^d} f(x)dx$$
or
$$\int_{\mathbb{R}^d} f(x)\exp(-x^T x)dx$$
- More general $\int_\Omega f(x)dx$
- Choice of best method depends on properties of f and on d
- Simplest method: Monte Carlo (use for $d > 100$, non-smooth f). For $50 < d < 100$ often quasi-Monte-Carlo is preferable
- For many economic problems, quadrature (or cubature) works better

Monte Carlo



Monte Carlo

- Simple observation: For sufficiently large M and M randomly chosen points in Ω,

$$\frac{\int_{\Omega} f(x)dx}{\int_{\Omega} dx} \simeq \frac{1}{M} \sum_{i=1}^{M} f(x_i)$$

- Advantages: Simple, embarrassingly parallel, works for arbitrary functions and domains
- Disadvantages: What are random numbers? Most applications use pseudo-random numbers that are completely deterministic. Quantum random number generators are used commercially, e.g. *www.idquantique.com*.
- Quasi-Monte-Carlo seems to solve the problem: Uses deterministic points that have good properties.
- But once we choose the points, might as well choose according to the application

Numerical Integration –Quadrature

Let's focus on the one-dimensional case and develop a very simple and efficient method for continuous (probably smooth) functions. First we consider the bounded domain case. Remember from high-school that

$$\int_a^b f(x)\,dx = \frac{b-a}{2}\int_{-1}^{1} f\left(\frac{b-a}{2}x + \frac{a+b}{2}\right)\,dx.$$

Numerical Integration – Gaussian Quadrature

- Suppose want to solve

$$\int_{[-1,1]} f(x)dx$$

If $f(.)$ can be approximated well by a polynomial of degree d, it suffices to take f at $d+1$ points and then compute the integral of the polynomial (which we can do exactly)

$$\int_{[-1,1]} f(x)dx \simeq \sum_{i=1}^{n} w_i f(x_i)$$

- Want to find good points $(x_i)_{i=1}^{n}$ and then need to work out the w_i...

Numerical Integration – Gauss-Legendre

- Gauss-Legendre is call Legendre because it takes the (x_i) to be the roots of the Legendre polynomials.

$$w_i = \frac{2}{(1 - x_i^2) P_n'(x_i)^2}$$

- Recall that they are orthogonal under the L^2 norm...

Quadrature more general

- More generally, we might want to integrate $\int_\Omega f(x)g(x)dx$ for some weighting function $g(x)$. Need to find polynomials that are orthogonal under these weights and on the specified domain
- To solve $\int_{-\infty}^{\infty} f(x)\exp(-x^2)$ use Gauss-Hermite
- In one dimension, this is also pretty simple and well understood
- Look up 'integrate' in matlab

Numerical Integration – higher dimensions

- Polynomial interpolation in higher dimensions is not so clearcut.
- What are good points in weights in $\mathbb{R}^d$?
- One possibility is to use the Smolyak points (with appropriate weights that can be looked up).
- For $d < 30$ dimension monomial rules work extremely well and they are much easier to implement

Monomials rules 1

- Stroud (1971) is the standard reference. See also http://nines.cs.kuleuven.be/research/ecf/ for recent developments and Judd's book for some simple formulas.
- Want to approximate integral by sum. Let $\Omega \subset \mathbb{R}^M$ and want

$$\int_{\Omega} g(x) f(x) dx \simeq \sum_{i=1}^{N} w_i f(x_i)$$

- Define remainder

$$R[f] = \int_{\Omega} g(x) f(x) dx - \sum_{i=1}^{N} w_i f(x_i)$$

If $R[f] = 0$ when f is an arbitrary linear combinations of monomials with degree less than or equal to some d then we say the cubature formula has degree n

Monomials rules 2

- Want to have cubature formulas with relatively few points. The minimal number of points obviously depends on the dimension, on the degree but also on Ω. Often no known (just lower and upper bounds).
- The N points and the weights are a solution of a system of polynomial equations

$$\sum_{j=1}^{N} w_i f_l(x^i) = \int_\Omega f_k(x)dx$$

where the monomials f_k form a monomial basis.
Tough problem

Numerical Integration – higher dimensions

A good rule to use for negative exponential weighting function is the following monomial rule:

$$\int_{\mathbb{R}^d} f(x) e^{-\sum_{i=1}^d x_i^2} dx \approx Af(0) + B \sum_{i=1}^{d} \left(f(re^i) + f(-re^i) \right) + D \sum_{i=1}^{d-1} \sum_{j=i+1}^{d}$$
$$\left(f(se^i + se^j) + f(se^i - se^j) + f(-se^i + se^j) + f(-se^i - se^j) \right),$$

with

$$r = \sqrt{1 + d/2}, s = \sqrt{1/2 + d/4}, A = \frac{2\pi^{d/2}}{d+2}, B = \frac{(4-d)\pi^{d/2}}{2(d+2)^2}$$

$$D = \frac{\pi^{d/2}}{(d+2)^2}.$$