Polynomial Approximation—A New Computational Technique in Dynamic Programming: Allocation Processes

By Richard Bellman, Robert Kalaba, and Bella Kotkin

1. Introduction. The problem of maximizing the function

$$F(x_1, x_2, \cdots, x_N) = g_1(x_1) + g_2(x_2) + \cdots + g_N(x_N) \tag{1.1}$$

over the domain

$$x_1 + x_2 + \cdots + x_N = x, \quad x_i \geqq 0 \tag{1.2}$$

can be reduced via familiar dynamic programming techniques (see [1]) to that of determining the sequence of functions $\{f_n(x)\}$ generated by means of the recurrence relation

$$\begin{aligned} f_1(x) &= g_1(x), \\ f_{n+1}(x) &= \max_{0 \leqq y \leqq x} [g_{n+1}(y) + f_n(x - y)]. \end{aligned} \tag{1.3}$$

The problem can thus be solved numerically in a very simple fashion, regardless of the complexity of the functions $g_i(x)$. A number of important allocation processes can be resolved in this way. If we consider cases in which two distinct types of resources must be allocated, we face the problem of maximizing the function

$$\begin{aligned} \tilde{F}(x_1, x_2, \cdots, x_N ; y_1, y_2, \cdots, y_N) \\ = g_1(x_1, y_1) + g_2(x_2, y_2) + \cdots + g_N(x_N, y_N) \end{aligned} \tag{1.4}$$

over the domain

$$\begin{aligned} x_1 + x_2 + \cdots + x_N &= x, \quad x_i \geqq 0, \\ y_1 + y_2 + \cdots + y_N &= y, \quad y_i \geqq 0. \end{aligned} \tag{1.5}$$

Theoretically, there is no difficulty in applying the same techniques.

The maximization problem can be reduced to the determination of the sequence $\{f_n(x, y)\}$ generated by means of the recurrence relation

$$f_{n+1}(x, y) = \max_{\substack{0 \leqq w \leqq x \\ 0 \leqq r \leqq y}} [g_{n+1}(w, r) + f_n(x - w, y - r)]. \tag{1.6}$$

In principle, this equation can be solved computationally using the same technique that applies so well to (1.3). In practice (see [1] for a discussion), questions of time and accuracy arise. There are a number of ways of circumventing these difficulties, among which the Lagrange multiplier plays a significant role.

In this series of papers, we wish to present a number of applications of a new, simple and quite powerful method, that of *polynomial approximation*. We shall begin with a discussion of the allocation process posed in the foregoing paragraphs and continue, in subsequent papers, with a treatment of realistic trajectory and

Received March 2, 1962.

guidance processes. In a separate series of papers we shall apply this fundamental attack upon dimensionality to the solution of a number of the equations of mathematical physics.

We would like to express our appreciation to Oliver Gross for the analytic solution of some test problems we used to check the accuracy of our techniques, and for his general interest in the program.

2. Polynomial Approximation. In the systematic study of dynamic programming as a computational algorithm given in [1], a function $f(x)$ is considered to be a table of values at an appropriate set of grid points:

(2.1)

x	$f(x)$
0	f_0
Δ	f_1
2Δ	f_2
$\vdots$	$\vdots$
$K\Delta$	f_K.

In this way, the function $f(x)$ is stored in the computer. For functions of one variable with $K = 100$ or 1000, this is a reasonably convenient way to proceed. For functions of two variables, this procedure becomes a bit inconvenient since $(K + 1)$ values for x combined with $(K + 1)$ values for y yields a total set of $(K + 1)^2$ values. Consequently, when we encounter functions of three or more variables, we must balance accuracy against time and the limited storage of contemporary computers in our choice of K.

The storage of functional values by means of a table of values is ideally suited to the treatment of problems involving functions of quite arbitrary form. It is, on the other hand, quite wasteful and inefficient if we are dealing with functions possessing a definite structure, which is to say, situations in which there is a high correlation between the values of $f(r\Delta)$ and $f(s\Delta)$. Since functions of this type occur in many important applications, and throughout mathematical physics, it is worthwhile to develop methods which take advantage of the "smoothness" of the function.

One such method is *polynomial approximation*, or to be precise, generalized polynomial approximation. We represent the function in the form

$$f(x) \cong \sum_{k=1}^{M} a_k \varphi_k(x), \tag{2.2}$$

where the $\varphi_k(x)$ are known elementary functions such as x^k, $\sin kx$, $P_k(x)$ (the Legendre polynomial of degree k) or $T_k(x)$ (the Chebyshev polynomial of degree k), and then store the function for all values of interest by means of the set of M coefficients $[a_1, a_2, \cdots, a_M]$.

It is important to point out that (2.2) is particularly useful in automatically furnishing the interpolation values frequently needed in dynamic programming calculations. If one uses a table of values of the form shown in (2.1), interpolation is frequently a source of difficulty.

To determine the coefficients a_k it is convenient to make the $\varphi_k(x)$ an orthonormal set. Then

$$a_k = \int_0^1 f(x)\varphi_k(x)\, dx, \tag{2.3}$$

assuming for the purposes of convenience that $0 \leqq x \leqq 1$. We can, of course, use a Chebyshev fit instead, and we will explore this in subsequent papers. A priori, we would suspect that it would be more efficient to use a mean-square fit (implied by (2.3)), and take M to be larger, than to go to the trouble of determining the a_k to minimize the function

$$\max_{0 \leqq x \leqq 1} |f(x) - \sum_{k=1}^{M} a_k \varphi_k(x)|, \tag{2.4}$$

which for a fixed M may be expected to yield a more accurate approximation. Alternatively, one could use an approximation by polygonal functions [1].

To evaluate the a_k without requiring a knowledge of too many values of $f(x)$, we use a quadrature technique

$$a_k \cong \sum_{i=1}^{R} w_i f(x_i) \varphi_k(\bar{x}_i), \tag{2.5}$$

where the weights w_i and the quadrature points $\bar{x}_i$ are chosen so that the equation

$$\int_0^1 g(x)\, dx = \sum_{i=1}^{R} w_i g(\bar{x}_i) \tag{2.6}$$

is exact for polynomials in x of degree $(2R - 1)$ or less. This requirement narrows us down to the Gauss quadrature technique [2]. If we use generalized polynomials (expressions such as (2.6)), we will obtain different weights and quadrature points.

We may then store the function $f(x)$ for $0 \leqq x \leqq 1$ by storing the coefficients a_k or the particular values $f(\bar{x}_i)$ which enable us to compute the a_k.

3. Application to Dynamic Programming. Consider now the application of these ideas to the computational solution of the functional equations of dynamic programming. Suppose that we wish to compute the sequence $\{f_n(x)\}$ determined by

$$f_N(x) = \max_{0 \leqq y \leqq x} [g_N(y) + f_{N-1}(x - y)], \tag{3.1}$$

$N \geqq 2$, given that $f_1(x) = g_1(x)$. To avoid the tabulation of each of the functions $f_N(x)$ at the x-grid $[0, \Delta, \cdots, K\Delta]$, where K may be a large number, we approximate to each function $f_N(x)$ in the manner indicated above. Starting with $f_1(x)$, we store the values $f_1(x_i)$, $i = 1, 2, \cdots, R$, needed to evaluate $f_1(x - y)$ in the formula determining $f_2(x)$,

$$f_2(x) = \max_{0 \leqq y \leqq x} [g_2(x_2) + f_1(x - y)]. \tag{3.2}$$

We then determine successively $f_2(x_1), f_2(x_2), \cdots, f_2(x_R)$, a set of values which stores the function $f_2(x)$. This process is then repeated.

Although the calculation of $f_{n-1}(x - y)$ using (2.2) is far more time-consuming than taking a value from storage, we expect to gain time over-all because of the fact that we are required to calculate only a few values, $f_n(x_i)$, $i = 1, 2, \cdots, R$, at each stage.

4. The Legendre Polynomials. Since in allocation processes we have a range $[0, x_0]$ which stays constant as the process continues from stage to stage, we can normalize and consider the basic interval to be $[0, 1]$. Since the interval is finite

and we want, at the moment, a polynomial approximation, we shall employ Legendre polynomials.

Let $P_k(x)$ denote the standard Legendre polynomial, defined over $[-1, 1]$, and let $\varphi_k(x)$ be defined by the relation

$$\varphi_k(x) = (2k + 1)^{1/2} P_k(2x - 1). \tag{4.1}$$

The sequence $\{\varphi_k(x)\}$ is then orthonormal over $[0, 1]$, i.e.,

$$\int_0^1 \varphi_k(x)\varphi_\ell(x)\, dx = \delta_{k\ell}\,. \tag{4.2}$$

From the standard recurrence relations for $P_k(x)$, we obtain the relation

$$\begin{aligned} &\varphi_1(x) = 1, \quad \varphi_2(x) = 3^{1/2}(2x - 1), \\ &\varphi_{i+2}(x) = \frac{(2i + 3)^{1/2}}{(i + 1)}\left[(2i + 1)^{1/2}(2x - 1)\varphi_{i+1}(x) - \frac{i}{(2i - 1)^{1/2}}\varphi_i(x)\right]. \end{aligned} \tag{4.3}$$

This relation makes the evaluation of the sequence of values of $\varphi_i(x)$ for any x a relatively simple matter.

5. Examples. In order to experiment with this new approximating procedure, we devised a FORTRAN program for the IBM-7090 whereby at each stage, after obtaining the new coefficients $a_k^{(i)}$, we computed $f_i(x)$ from (2.2) for a succession of as many values of x as desired, and printed the result after each computation. We used two modes of output, either a list of numerical values, $[x, f_i(x)]$, or a graphical plot (done directly by the 7090) of x versus $f_i(x)$.

We experimented with several types of functions $g_i(x)$ for which the results could be derived from analytic considerations.* Using the known analytic results as a checkpoint, we varied the parameters R, M, and the grid size H, in order to determine the degree of accuracy we might expect in general. We found remarkably good agreement to 2 or more significant figures with a relatively low order of approximation, namely $R = 5$, $M = 6$. This is encouraging from the point of view of extending the method in the experimental investigations of higher-dimensional allocation processes where, as pointed out above, time and storage aspects become significant. We also incorporated in our program restraints of the form $0 \leqq a_i \leqq x_i \leqq b_i \leqq x$, since constraints of this nature often occur in applications.

a. Time Estimates. Following are some estimates of the execution time required on the 7090:

1. Ten seconds for 4 stages, $R = 5$, $M = 6$ and a grid of 0.05 both for the search and output listing.

2. Three minutes for a total of 4 cases of 10 stages each; $R = 3$, $M = 4$; $R = 5$, $M = 6$; $R = 7$, $M = 8$; $R = 10$, $M = 11$. The grid in the search for the maximum is 0.01 and the output listing is given for every x along the grid.

3. Three minutes for the total of 4 cases mentioned above, 10 stages per case and a grid of 0.01 for the search. The output is a graph where the independent variable x is listed at intervals of 0.025.

---

* Oliver Gross was of considerable assistance to us in this respect.

b. Numerical Results. 1. $g_i(x) = i(x)^{1/2}$. Using the Schwarz inequality, we readily obtain the values

$$f_N(x) = \left(\frac{N}{6}(N+1)(2N+1)x\right)^{1/2}.$$

For $R = 10$, $M = 11$, $H = 0.01$ we found poor agreement at the origin. This is to be expected because of the infinite slope at $x = 0$. Agreement between exact and computed values was good as soon as x moved away from the singular value 0, as can be seen from the following table:

Function	Exact	Computed
$f_1(0)$	0.0	0.064
$f_1(2)$	0.448	0.447
$f_1(1)$	1.00	1.00
$f_{10}(0)$	0.0	3.13
$f_{10}(1)$	19.6	19.6

2. $g_i(x) = i(x + 0.1)^{1/2}$. We avoided the previous difficulty at $x = 0$ (see the computed value of $f_{10}(0)$ above), but found the function still rather sensitive near the origin. As N (the number of stages) increased, the agreement at $x = 0$ decreased.

3. $g_i(x) = i(x + 1)^{1/2}$. The Schwarz inequality yields the upper bound

$$f_N(x) \leqq \left(\frac{N}{6}(N+1)(2N+1)(x+N)\right)^{1/2}.$$

However, since the x_i are subject to the restraint $0 \leqq x_i \leqq x$, we do not necessarily achieve the upper bound. Some exact values based on an analysis by O. A. Gross are listed as check points, $R = 10$, $M = 11$, $H = 0.01$.

Function	Upper Bound	Exact Value	Computed Value
$f_1(0)$	1.00	1.00	1.00
$f_1(1)$	1.41	1.41	1.41
$f_3(0)$	6.48	—	6.00
$f_3(.5)$	7.00	6.67	6.67
$f_7(.35)$	32.1	29.1	29.1
$f_{10}(1)$	—	59.3	59.3

4. $g_i(x) = e^{-5/(1+10x)}$, $R = 10$, $M = 11$, $H = .01$.

Function	Exact	Computed
$f_2(.2)$	0.196	0.197
$f_2(.7)$	0.659	0.659
$f_3(.2)$	0.203	0.204
$f_3(.9)$	0.860	0.862
$f_5(.2)$	0.218	0.217
$f_{10}(1)$	1.001	1.001

6. Two-Dimensional Approximation. The problem of maximizing the function

$$\sum_{i=1}^{N} g_i(x_i, y_i) \tag{6.1}$$

over the domain

$$\begin{aligned} &\sum_{i=1}^{N} x_i = x, \quad 0 \leqq a_i \leqq x_i \leqq b_i, \quad x_i \leqq x, \\ &\sum_{i=1}^{N} y_i = y, \quad 0 \leqq c_i \leqq y_i \leqq d_i, \quad y_i \leqq y, \end{aligned} \tag{6.2}$$

can be readily handled by the methods described in Sections 2 and 3.

The dynamic programming recurrence relation is:

$$f_N(x, y) = \max_R [g_N(x_N, y_N) + f_{N-1}(x - x_N, y - y_N)],$$

where R is determined by $a_N \leqq x_N \leqq \min(x, b_N)$, $c_N \leqq y_N \leqq \min(y, d_N)$. Let $\bar{x}_i$, $i = 1, \cdots, R$, be the roots of the normalized Legendre polynomial $\phi_R(x)$, and let $\{\bar{y}_j\}$ be a duplicate set of these quadrature points. Each function $f_N(x, y)$ is expressed approximately by the relation

$$f_N(x, y) = \sum_{r=1}^{M} \sum_{s=1}^{M} a_{r,s}^{(N)} \phi_r(x) \phi_s(y), \tag{6.3}$$

where the coefficients, using the quadrature method, are given by

$$a_{k,s}^{(N)} = \sum_{i=1}^{R} \sum_{j=1}^{R} w_i w_j f_N(\bar{x}_i, \bar{y}_j) \phi_k(\bar{x}_i, \bar{y}_j). \tag{6.4}$$

As in the one-dimensional case, we start with the known values $f_1(x, y) = g_1(x_1, y_1)$, $x_1 = x$, $y_1 = y$. In stage n we store the values $f_n(\bar{x}_i, \bar{y}_j)$, for $i, j = 1, 2, \cdots, M$, then compute and store the values $a_{r,s}^{(n)}$ in the storage allotted to the previous stage. The latter coefficients are utilized in the computation of $f_n(x - x_{n+1}, y - y_{n+1})$ to obtain the values of $f_{n+1}(x_i, y_j)$ in the next stage.

7. Examples.

a. Time Estimate.

The execution time required on the 7090 was 2 minutes for 4 stages, with $R = 5$, $M = 6$, a grid of $H = 0.05$ in the two-dimensional search, and an output listing of 3 test values of $f(x, y)$ in each stage.

b. Numerical Results.

1. $g_i(x, y) = (2i - 1)^{1/2}(xy)^{1/4}$. Using the Hölder inequality, we readily obtain the exact values $f_n(x, y) = n(xy)^{1/4}$, $x_n(x, y) = (2n - 1)x/n^2$, $y_n(x, y) = (2n - 1)y/n^2$. Our results for $R = 5$, $M = 6$, and a grid of 0.05 in the search are:

Function	Exact	Computed
$f_2(.5, .5)$	1.40	1.40
$x_2(.5, .5)$	0.375	0.35 (to the nearest 0.05)
$f_4(1, 1)$	4.00	3.91

Here again the agreement was poor at the origin, presumably because of the singular behavior of $(xy)^{1/4}$ at $x = 0$, $y = 0$. To confirm this hypothesis, we considered the next case.

2. $g_i(x, y) = (x + iy)/(1 + x + iy)$. This case, as well as the theoretical values, was suggested by O. A. Gross, and he determined the exact values.

Function	Exact	Computed
$f_2(1, 1)$	1.17	1.17
$f_2(1, 0)$	0.667	0.647

8. Discussion. As can be seen, the agreement in general is quite satisfactory. We can obtain reasonably accurate values of the maximum return and of the optimal allocation policy using small amounts of machine time.

Combining these techniques with the method of the Lagrange multiplier, we can expect to solve three- and four-dimensional resource allocation problems. Extending the method to cover the approximation of functions of 3, 4, 5 and 6 variables, we can treat Hitchcock-Koopmans allocation processes of quite high dimension.

Finally, if we combine these techniques—polynomial approximations and Lagrange multipliers—with that of successive approximations, there should be very few allocation processes which still resist our efforts.

The Rand Corporation
Santa Monica, California

1. R. BELLMAN & S. DREYFUS, *Applied Dynamic Programming*, Princeton University Press, Princeton, N.J., 1962.