Convergence Rates of Stochastic Global Optimisation Algorithms with Backtracking

A thesis presented in partial
fulfilment of the requirements
for the degree
of Doctor of Philosophy
in Statistics at
Massey University

D. L. J. Alexander
October 18, 2004

Abstract

A useful measure of quality of a global optimisation algorithm such as simulated annealing is the length of time it must be run to reach a global optimum within a certain accuracy. Such a performance measure assists in choosing and tuning algorithms. This thesis proposes an approach to obtaining such a measure through successive approximation of a generic stochastic global optimisation algorithm with a sequence of stochastic processes culminating in backtracking adaptive search.

The overall approach is to approximate the progress of an optimisation algorithm with that of a model process, backtracking adaptive search. The known convergence rate of the model then provides an estimator of the unknown convergence rate of the original algorithm. Parameters specifying this model are chosen based on observation of the optimisation algorithm.

The optimisation algorithm may first be approximated with a time-inhomogeneous Markovian process defined on the problem range. The distribution of the number of iterations to convergence for this averaged range process is shown to be identical with that of the original process. This process is itself approximated by a time-homogeneous Markov process in the range, the asymptotic averaged range process. This approximation is defined for all Markovian optimisation algorithms and a weak condition under which its convergence time closely matches that of the original algorithm is developed. The asymptotic averaged range process is of the same form as backtracking adaptive search, the final stage of approximation.

Backtracking adaptive search is an optimisation algorithm which generalises pure adaptive search and hesitant adaptive search. In this thesis the distribution of the number of iterations for which the algorithm runs in order to reach a sufficiently extreme objective function level is derived. Several examples of backtracking adaptive search on finite problems are also presented, including special cases that have received attention in the literature.

Computational results of the entire approximation framework are reported for several examples. The method can be applied to any optimisation algorithm to obtain an estimate of the time required to obtain solutions of a certain quality. Directions for further work in order to improve the accuracy of such estimates are also indicated.

Massey University

Wellington Campus
Private Box 756
Wellington
New Zealand
Telephone: 64 4 801 5799
Facsimile: 64 4 801 2692

This statement confirms that David Alexander has pursued the Doctoral course in accordance with the University's Doctoral regulations.

Supervisor signed: [illegible]

Date: 16 Dec 03

Te Kunenga ki Pūrehuroa

Inception to Infinity: Massey University's commitment to learning as a life-long journey

Massey University
COLLEGE OF SCIENCES

Institute of Information Sciences & Technology
Private Bag 11 222,
Palmerston North,
New Zealand
Telephone: 64 6 350 5799
Facsimile: 64 6 350 5723

Integrated Research and Teaching in the Fields of:

Statistics & Applied Statistics

Computer Science & Information Systems

Information & Electronic Engineering

Supervisory assistance has been received during the Doctoral research from Dr David Bulger of Massey University and Professor Graham Wood of Macquarie University, Australia. I have also collaborated with Professor Zelda Zabinsky of the University of Washington, United States of America and Dr Bill Baritompa of the University of Canterbury. In particular, Subsections 5.3.1, 5.3.2, 5.3.4 and 5.3.5 contain material on which I have collaborated but of which I am not the principal author. These results also appear in [9, 49]. Other results have also been published in [1, 48]. I have received financial support from a Bright Future Scholarship administered by the Foundation of Research, Science and Technology and the Marsden Fund administered by the Royal Society of New Zealand.

The thesis material has not been used for any other degree or diploma.

Doctoral Candidate signed: [illegible]

Date: 18 DECEMBER 2003

Massey University

Wellington Campus
Private Box 756
Wellington
New Zealand
Telephone: 64 4 801 5799
Facsimile: 64 4 801 2692

This statement verifies the following:

i. Reference to work other than that of the candidate has been appropriately acknowledged,

ii. Research practice, ethical and genetic technology policies have been complied with as appropriate, and

iii. The thesis does not exceed 100,000 words.

Doctoral Candidate signed: [illegible]

Supervisor signed: [illegible]

Date: 16 Dec 03

Te Kunenga ki Pūrehuroa

Inception to Infinity: Massey University's commitment to learning as a life-long journey

Contents

Chapter 1

Introduction

1.1 Preface

Stochastic global optimisation algorithms are widely used to solve optimisation problems that are currently beyond the reach of other solution methods. The theory, however, has long lagged behind the practice in this field; methods are often used without any rigorous justification beyond the growing empirical evidence that they work better than their alternatives in many situations. The aim in this thesis is to take some first steps down one possible path towards providing a theoretical basis for the use of these algorithms.

The issue at stake is one of processing time. Deterministic algorithms exist that are capable of solving any feasible finite optimisation problem (optimising an objective function over a domain of possible decision variable values)—by enumeration of all possible solutions, if necessary, or in many cases by more sophisticated approaches. All finite optimisation problems can thus be solved by a deterministic algorithm in finite time. Where such a deterministic method can be used in a reasonable time frame, it is always to be preferred, since there is a guarantee of finding the optimum. Stochastic global optimisation algorithms remain in use because there are many cases where no such fast deterministic algorithms are applicable. In these cases the deterministic methods are inadequate to the task; the time taken for these methods to converge to the global optimum is greater than the time practically available. (An algorithm "converges" when a sufficiently good solution to the problem is found.)

A note on the application of optimisation algorithms to infinite domains is necessary. In general optimisation on an infinite domain can be arbitrarily difficult; consider maximising the objective function

$$x = \begin{cases} 1, & x = x^* \\ 0, & \text{otherwise} \end{cases}$$

where x^* is unknown. No algorithm, deterministic or stochastic, could have a finite mean convergence time applied to this problem. When a condition such as the Lipschitz condition holds, however, it is possible for various methods to find a solution to a continuous problem that is optimal to within any specified accuracy. (In practice, the representation of any optimisation problem within a computer is discrete, due to finite machine precision, so that bounded domains are effectively finite in any case. It is possible to make a discrete approximation of any accuracy (within machine precision) to any continuous problem, simply by rounding values; again, a condition such as the Lipschitz condition is required for the error introduced by such an approximation to be bounded.)

The following section provides a broad discussion of stochastic global optimisation, describing some of the general ideas behind several algorithms and outlining some of the work in this field. Subsections 1.2.1 and 1.2.2 respectively describe practical and theoretical algorithms. Section 1.3 broadly summarises the approach taken in this thesis towards analysis of the convergence times of stochastic global optimisation algorithms. Section 1.4 then outlines the content of each of the chapters in the thesis.

1.2 Stochastic global optimisation

This section introduces the concepts involved in various stochastic global optimisation algorithms that have been studied. The present focus is on convergence times of these algorithms. The practical algorithms presented below are currently without any theoretical guarantee of the quality of solution that can be obtained in general within a reasonable time frame; on the other hand, the family of theoretical algorithms subsequently described has currently no practicable implementation.

In global optimisation, the function to be optimised may have in general many local optima but fewer global optima. A local optimum is a solution superior to any other solution in its immediate neighbourhood; global optima are solutions that are not bettered by any other solution in the entire problem domain. (Note that the term "solution" is applied to any domain point, not only to optimal solutions.) Successful optimisation methods are required to find not only local optima but also global optima. It is this characteristic of global optimisation that makes it difficult enough to demand its own toolbox of stochastic optimisation algorithms.

Any algorithm where the probability of transition between any feasible solutions in a finite number of iterations is bounded away from zero will visit a global optimum of a finite problem in finite time with probability one. (The required bound could be achieved by drawing solutions at random at regular (or irregular) intervals.) A useful algorithm is one that is likely to visit the optimum in a short amount of time. Expected convergence times of the practical algorithms mentioned here are difficult to obtain, however. Bounds on their value generally exceed practical time limits, so that they provide no useful guide for implementation of the algorithm.

1.2.1 Practical stochastic global optimisation algorithms

This subsection describes several practical approaches to stochastic global optimisation, with particular reference to the results concerning convergence rates of these algorithms.

Pure random search [8, 14, 15] is the simplest stochastic global optimisation algorithm. At each iteration a new solution is chosen according to some distribution on the domain. The expected number of iterations before entering a target area containing the global optima is simply the reciprocal of the weight placed by the search distribution on the target area.

This method can be seen as a base algorithm against which other algorithms should be compared. The aim of each other algorithm is to use information gained about the problem to select successive iterates in a more "intelligent" way than does pure random search, where the first iteration is as likely to attain any objective function level as any other.

Note that the algorithm will move away from global optima after they are sampled. The achievement of the algorithm after any period of time is therefore not necessarily the current solution, but the best solution found during the progress of the algorithm, which is stored independently of the current state of the algorithm. This characteristic applies to many of the algorithms considered in this section.

The **multistart** algorithm [7, 41] uses a deterministic local search method capable of finding local optima. This local search method is applied from multiple starting points, selected in some way from the domain such that there is a positive probability of sampling any domain point. If enough starting points are selected, one of them will eventually be "near" the global optimum. The local optimum found at this stage will then be the global optimum.

This method, in its simplest form, has the same convergence properties as pure random search, where an iteration consists of choosing a starting point and applying the local search method to it. The "target area" is now the set of all points in the domain from which the local search method will converge to a global optimum. The size of this target area is in general unknown. The time required by the algorithm is therefore difficult to predict.

The multistart method described above combines two basic ideas. Firstly, the use of the local search method can be viewed as effectively replacing the objective function value at each point in the domain by the objective function level obtained by a local search initiated from that point. The best solution found is then the local optimum yielding the best objective function value observed during the progress of the algorithm. Secondly, repeating the process from various initial solutions ensures that (as for pure random search) the global optimum will eventually be found. Either of these strategies may obviously be used in conjunction with any global optimisation algorithm. Results obtained by combining them with the Metropolis algorithm, described following, are reported in [32]. Computational results are reported to be good for smaller problems but increasingly poor for larger problems. The required convergence time remains difficult to predict.

The **Metropolis algorithm** [35] is based on an analogy with the energy level of a system of particles in a fluid. A result from statistical mechanics gives that the possible configurations of particles in such a system at temperature θ have a Boltzmann distribution with probability density function proportional to $e^{-E/(\kappa_B\theta)}$, where E is a variable denoting the energy level resulting from each possible configuration of particles in the system and κ_B is the Boltzmann constant.

The Metropolis algorithm commences by choosing an initial solution to the problem. A new candidate solution is then generated, based possibly on the current solution. This candidate solution is accepted as the new current solution for the algorithm whenever the change in objective function value from the current solution to the new candidate solution ΔE is non-positive (in a minimisation context) or with probability $e^{-\Delta E/(\kappa_B \theta)}$ otherwise, for some θ. It can be shown that the distribution of objective function values generated by this algorithm tends to the Boltzmann distribution.

If the "temperature" $\theta = 0$, no worsening of objective function values is allowed. In this case the algorithm is likely to reach only a local optimum, from which it is unable to escape. As the temperature tends to infinity, the probability of accepting worsening moves increases and the limiting probability of occupying a level close to the minimum decreases. The temperature level must therefore be a compromise between the two aims of allowing worsening moves so that the algorithm can quickly tend towards its limiting distribution and keeping θ low so that the limiting distribution places a high weighting on near-optimal states. Results bounding the time taken for the distribution to reach its limiting distribution in various situations are summarised in [13].

The following method attempts to progressively alter the temperature in such a way that the algorithm can always escape from local minima but the temperature also decreases towards zero, so that the probability of the objective function being within any fixed distance of the minimum tends to one over time.

Simulated annealing [29] is one of the most commonly used stochastic global optimisation algorithms. A motivation for this approach is the physical process of heating and cooling metals or glass in order to maximise their strength. When a liquid is rapidly cooled, the particles are forced to bond quickly but sub-optimally. Repeated reheating and gradual cooling allows particles to readjust and obtain a stronger crystalline structure at the freezing point. Thus the aim of obtaining a strong solid is best achieved by temporarily allowing the metal or glass to become more fluid on occasions, before eventually cooling again.

Analogously, in simulated annealing successive iterates are allowed to temporarily worsen in objective function value before again being required to improve. The algorithm can thus escape from local minima. While this idea could be implemented in many different ways, the simulated annealing algorithm bases its approach on the Metropolis algorithm with varying temperature.

The temperature governs the probability with which worsening iterates are accepted. By slowly letting the temperature tend to zero, the algorithm slowly causes the distribution of objective function levels at each iteration to tend to a Boltzmann distribution placing high weightings on global optima.

Provided that the way in which successive iterates are altered obeys certain conditions and the temperature is reduced in a certain way, [21] shows that simulated annealing converges to global optima with probability one. A similar result under different conditions is given in [5]. Analysis of optimal choices of temperature appears in [48]. However, algorithms satisfying these conditions are generally too slow for practical use. Moreover, a result in [26] indicates that simulated annealing algorithms obeying these conditions asymptotically perform worse than multistart, as measured by the probability of visiting a global optimum.

Different methods of reducing the temperature are used in practice. In this way good results can often be obtained within an acceptable time frame; however, the proof that final results are likely to be near the optimum no longer applies. The best scheme for reducing temperature is often hard to determine. An upper bound for the expected number of iterations before reaching a solution with objective function value within a certain accuracy of the globally optimal value is established, under various conditions, in [34]. Convergence rates of an idealisation of simulated annealing are studied in [42].

Threshold acceptance [17] is a similar algorithm, where worsening iterates are accepted with probability one provided the change in objective function value does not exceed some threshold value. In the same way that temperature is reduced in simulated annealing, the threshold in this method is gradually reduced to zero, until finally no worsening moves are accepted. Some convergence results are presented in [2]. These results, however, provide no practical guide of how to ensure the convergence of the algorithm to a certain level in a certain amount of time.

Another variant is the **Great Deluge** [16], in which worsening iterates are accepted with probability one provided the iterate is no worse than a certain level. This method is named for an analogy with a nonswimmer in a flood. The nonswimmer can move anywhere provided his path does not take him lower than the water line. Similarly, in a maximisation context the Great Deluge accepts any move provided the new objective function value is no lower than a certain level. This "water level" is gradually increased as the algorithm progresses, gradually encouraging iterates to take higher objective function values. However, the analogy makes clear the potential for this algorithm to become trapped in a local optimum.

Tabu search [20] uses another method of escaping local minima. The algorithm operates on discrete domains by the rule that successive iterates are always accepted unless they are on the current "tabu list". When the algorithm visits a solution, it places that solution on the tabu list, ensuring that the tabu rule forbids revisiting that solution. In this way the algorithm is forced to explore new regions of the solution space.

Several iterations after a solution has been placed on a tabu list it can be removed, allowing the algorithm to revisit it if the process eventually returns to that region. The tabu rule may also be broken if a tabu solution satisfies certain "aspiration criteria". This allows solutions to be visited more often than the normal tabu rule would allow, if it is thought to help the algorithm visit new and better solutions. A result in [22] shows that tabu search converges to global optima on finite domains. However, there is no guarantee of finding a solution of a given quality in less than the time required to directly enumerate objective function values for every feasible solution.

A different strategy is used by **evolutionary algorithms** [40]. This approach is based on the theory of natural selection. In its most simple form, the algorithm chooses the initial current solution in some way and generates new candidate solutions by making slight alterations to the current solution. This mimics genetic mutation. Whenever an alteration improves the objective function value, the candidate solution is accepted as the new current solution. This represents "natural selection" as proposed in the theory of evolution.

Formulated in this way, the algorithm has no way of escaping local optima. However, if the mutation step has a positive probability of producing any feasible solution, local optima can be escaped. A practical difficulty is that convergence is very slow.

An algorithm similar in structure is **Improving Hit and Run** [53]. This algorithm uses the Hit-and-Run algorithm of [46]. The Hit-and-Run algorithm generates a new point from a current point by first selecting a direction uniformly randomly and then choosing a point uniformly randomly from the intersection of the feasible region and a line in this direction passing through the current point. The improving hit and run global optimisation algorithm commences by choosing an initial solution in some way. At each subsequent iteration, a candidate solution is generated by the Hit-and-Run algorithm and accepted to replace the current point only if it has improved objective function value. Since this generator can sample any solution from any other with positive probability, the algorithm converges on bounded domains with probability one. A result in [55] proves that the algorithm converges in time polynomial in the problem dimension for a certain class of problems.

Genetic algorithms, introduced in [24], extend the idea behind evolutionary algorithms as described above to mimic the process of genetic recombination. Meiosis is the process whereby a parent cell produces sex cells containing only some of each chromosome in the cell. (When an egg is fertilised in sexual reproduction, the male and female half-cells recombine to form a complete cell containing genetic material from both the father and mother.)

Genetic algorithms commence by choosing a population of initial solutions in some way. Solutions are represented as strings of information, analogous to chromosomes. The strings can be binary or they can be divided into data units that are believed to represent distinct components of the solution. Pairs of solutions are then combined in a manner similar to meiosis and fertilisation. Two solutions are chosen and their string structure broken at some position. The first half of one is then combined with the second half of the other and vice versa. The resulting solutions form a new "generation" in the population, containing solutions that mix characteristics of two members of the previous generation.

Departures from the reproduction analogy arise in that both possible combinations of parent "gametes" can be used. Also, any number of different combinations of parent solutions can be used at each step.

The new generation may either replace the previous one or be combined with it in some way. In either case, the population after the meiosis step generally requires reduction. This is normally carried out by removing the solutions with worst objective function values. This step parallels a breeding strategy. The idea of mutation from evolutionary algorithms may also be applied.

There are many degrees of freedom in choosing an implementation of these algorithms. Convergence analysis is provided in [43], including a proof that some variants converge to global optima with probability one. Bounds on the time required for a genetic algorithm to sample all possible solutions to a problem are reported in [30]. These bounds are impractically great for any but the simplest problems, however.

Another population approach is **Controlled Random Search**, proposed in [38]. A random population is chosen initially. At each iteration, a new candidate point is selected in a certain way. When a candidate point has objective function value better than the worst among the current population members, the new candidate replaces this member in the population.

The method by which new candidate points are chosen is as follows. A subset of a certain size from the population is chosen at random and its geometric centroid calculated. A new point from the population m is then chosen at random and reflected in the centroid to calculate the new candidate point. Candidates generated in this way are called primary points. Each candidate point is compared with the current population members; if the candidate point is feasible and superior in objective function value to the current population member with worst objective function value then this current population member is removed from the population and the candidate point added. Candidate points are chosen in this way until the proportion of candidate points accepted into the population during the progress of the algorithm falls below a certain level. Then whenever a primary point is infeasible or has objective function value worse than the worst of the current population members, a new candidate point, known as a secondary point, is found as the midpoint of m and the centroid. After a secondary point is considered, the algorithm returns to take a new subset from the population and generate a new primary point as before. Figure 1.1 shows how primary and secondary points are generated.

Some convergence analysis for this algorithm applied on hyper-rectangular domains is provided in [23]. The rate of convergence, in cases where the algorithm converges, remains difficult to predict.

A final method, first applied to optimisation in [25], mimics the way in which the brain functions. Computer models of **neural networks** as they occur in the brain are constructed of interconnected "neurons". Neurons receive inputs from other neurons or external sources and send output signals to other neurons or external receivers based on a function of their inputs. Networks can be constructed in such a way that the network outputs of the system converge to a solution to an optimisation problem.

As introduced in [25], the neural network approach to optimisation can become trapped in a local optimum. Hybridisations of the method with other approaches such as simulated annealing and

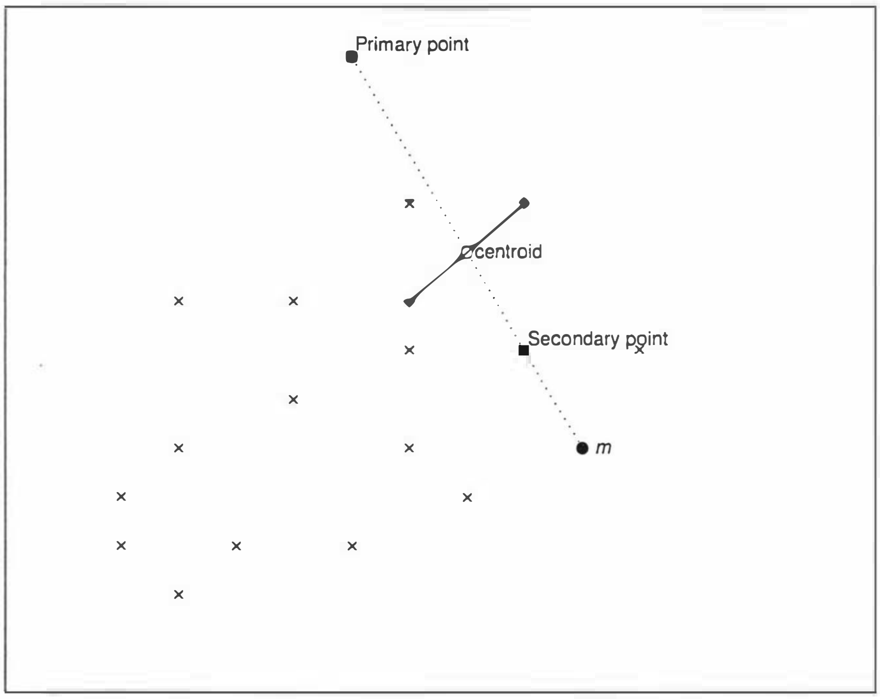

Figure 1.1: From a population of points denoted with crosses, a subset of two asterisked points is chosen. The centroid of these points is shown. A new point m is then chosen, leading to the generation of primary and secondary points as shown.

genetic algorithms have been developed to overcome this problem.

A review of results from this method is contained in [45]. A method of applying neural networks to any optimisation problem with objective function given by a quadratic function has been developed. On these problems neural networks are reported to perform as well as or sometimes better than simulated annealing.

1.2.2 Theoretical stochastic global optimisation algorithms

Distinct from these practical algorithms is a group of theoretical algorithms with known convergence properties but for which no known implementation takes an amount of time polynomial in the problem dimension. These methods are therefore no easier to implement than the deterministic algorithms they are designed to supersede. Their interest stems from the possibility of analytically obtaining expected convergence times, which is not possible for the practical algorithms mentioned above.

The first of these is **pure adaptive search** (PAS), introduced in [37]. In this algorithm, successive iterates are chosen according to a distribution on points in the domain with objective function values at least as good as that of the current iterate. Although this step currently cannot be implemented in polynomial time, a simple formula is derived in [54] to provide the expected convergence time of the algorithm if it could be carried out.

In [54] it is shown that the expected number of iterations before convergence for pure adaptive search increases only linearly with problem dimension for any problem satisfying the Lipschitz condition. If a polynomial time implementation of each iteration of pure adaptive search can be found, therefore, any optimisation problem satisfying the Lipschitz condition can be solved in polynomial time. Such an implementation for linear programming problems is developed in [18]. A polynomial time implementation of pure adaptive search on convex domains is given in [39]. A method of realising pure adaptive search for functions satisfying certain conditions, using quantum computation, is given in [10].

In fact a method of generating a new iterate from points in the domain with objective function values at least as good as a specified level (or reporting that no such points exist) could be used to solve the travelling salesperson recognition problem, so that this step of pure adaptive search is NP-hard [36]. A general implementation of pure adaptive search is therefore unlikely to be realisable within polynomial time on standard computers, unless P = NP.

A generalisation of pure adaptive search, introduced in [11], is **hesitant adaptive search**, where successive iterates are chosen in the same way as for pure adaptive search only with a certain probability. Otherwise the algorithm "hesitates" at the current range level.

The logical extension of these algorithms is **backtracking adaptive search** (BAS), introduced in [49]. This algorithm either betters, hesitates or backtracks at each iteration, according to probabilities that depend only on the current level. Bettering iterations are performed as in pure adaptive search. Worsening iterations are analogous: the new value is chosen according to a distribution on points in the domain with objective function values worse than that of the current iterate.

Many of the practical algorithms described above make use of backtracking in order to find improved solutions. In some cases backtracking is necessary to avoid becoming "stuck" at a local optimum. Consider the simple task of maximising $|x|$ over the domain $\{-1, 0, 1, 2\}$. An algorithm starting at x_0 that chooses each successive iterate from the intersection of the set $\{x_i - 1, x_i, x_i + 1\}$ and the domain is capable of solving this problem. However, if $x_0 = -1$, it is clear that a temporary reduction in objective function value will be required in order to reach the global optimum at 2. It is this characteristic of successful optimisation algorithms that backtracking adaptive search seeks to model.

These algorithms, particularly the last named, are discussed in more detail later in this thesis. A summary of past results, particularly relating to pure and hesitant adaptive search, is provided in [52]. The distribution of convergence times for backtracking adaptive search is established in Chapter 5. The following section outlines the method proposed in this thesis for linking these theoretical results with the practical stochastic global optimisation algorithms previously mentioned.

1.3 The approach

The general statistical approach to prediction is to observe a process, fit some kind of model to it, check goodness of fit of the model and, if satisfactory, calculate an estimate based on the assumption that the model holds. This general approach is applied in this thesis to predicting expected convergence rates for stochastic global optimisation algorithms.

The variable of interest in this case is the number of iterations before convergence for a stochastic global optimisation algorithm. As in any statistical context, there is uncertainty in the value of this variable; different runs of the same algorithm on the same problem may result in widely differing convergence times. The aim can only be to estimate an expected value of the convergence time, along with some idea of the variability that surrounds that estimate. This challenge is no different to that which faces any prediction in the face of uncertainty.

In order to make a prediction for the quantity of interest, some structure must be assumed of the underlying process. This structure takes the form of a model (whether parametric or not): it is assumed that the data is distributed as if it was generated by a process of a certain form, overlaid with a random component. If an appropriate model can be found, such that the observed data could reasonably be supposed to have been generated from the model, then this model will provide a suitable estimate for the unknown quantity. The assumption is made that this unknown quantity will also be distributed as though generated from this model.

Clearly, a required feature of the model is that it can be used to calculate the required quantity. In the present case, the required quantity is the expected number of iterations before convergence. The requirement that this quantity can be calculated is satisfied by the theoretical algorithms discussed in the previous section, most generally backtracking adaptive search. With the development of theory concerning the convergence times of these theoretical search algorithms, therefore, comes the possibility that they may form a suitable family of models for general stochastic optimisation processes.

As in any modelling, then, the parameters required for backtracking adaptive search are estimated based on observed data. The observed data is the progress of a stochastic global optimisation algorithm over several iterations. Methods for estimating backtracking adaptive search parameters from this data are detailed in subsequent chapters.

This discussion identifies two needs that must be satisfied in order to model convergence times of stochastic global optimisation algorithms. Firstly, the theory surrounding backtracking adaptive search must be developed for it to be used as a model. Secondly, a method by which backtracking adaptive search can approximate a general stochastic global optimisation algorithm is also required.

There are therefore two distinct arms to this thesis, relating respectively to analysis of convergence rates of the model family and to the development of a method for estimating its parameters based on observations of an optimisation algorithm. The structure of the thesis is outlined in the following section.

1.4 Thesis outline

A summary of the contents of each of the following chapters is now provided. In brief, there are two chapters outlining the method by which a general stochastic global optimisation algorithm is approximated by a backtracking adaptive search model, two chapters developing the analysis of backtracking adaptive search, a chapter illustrating computational results for some examples and a concluding chapter consolidating what has been achieved in this thesis and highlighting areas for continued research.

Chapter 2 introduces a framework of processes by which backtracking adaptive search approximates a stochastic global optimisation algorithm. The method is designed to operate in the completely general case where no assumption is made of the structure either of the algorithm or of the problem to which it is applied. All that is required is the sequence of observed iterations obtained by running the algorithm. This observed data is most easily modelled using a discrete partition of the range. A discrete backtracking adaptive search model can then be fitted to this discrete data.

However, the algorithm actually operates not in the range, but the problem domain. A model based on progress in the range therefore requires justification—is the movement of the algorithm in the range sufficient for use in predicting the convergence rate of an algorithm acting on the domain? This question is answered by means of a sequence of processes that approximate the optimisation algorithm in the domain: the range, averaged range and asymptotic averaged range processes. This framework of processes is defined and exemplified in Section 2.2. Each process in this sequence approximates the previous one, with the range process based directly on the algorithm in the domain. Thus the expected convergence time of a backtracking adaptive search approximation to the asymptotic averaged range process is indirectly an approximation of the convergence time of the original algorithm of interest.

Since the analysis is to be based on a discrete summary of the observed data, the theory developed also uses discrete techniques. In particular, much use is made of Markov chain theory. A finite Markov process uses a transition matrix that stores the probabilities of changing from one possible state of the process to another in one step of that process. The following is a simple Markov transition matrix.

$$\begin{array}{cc} & \text{Next state} \\ & \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \text{Current state} \begin{array}{c} 1 \\ 2 \\ 3 \end{array} & \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ \frac{1}{2} & \frac{1}{2} & 0 \end{bmatrix} \end{array}$$

This matrix implies that whenever the process is in state 1, it will remain in state 1 with certainty after one iteration. Similarly, if it is in state 2 it will transition to state 3 with certainty. However, if it is in state 3 then after one iteration the process is equally likely to be in either state 1 or state 2. This example is taken up in Section 2.2.

Chapter 3 fleshes out the theory behind the intermediary processes introduced in Chapter 2. Section 3.2 contains a proof that the number of iterations before convergence for the averaged range process defined in Chapter 2 has the same distribution as the number of iterations before convergence for the original algorithm. This stage of the approximation framework can thus be reached with no loss of information concerning convergence rates. Section 3.3 demonstrates the existence of the final intermediary process defined in Chapter 2, the asymptotic averaged range process. A subsection developing preliminary Markov chain theory precedes the existence proof, which is presented initially in the case where the domain transition matrix on suboptimal states is both irreducible and acyclic and then in the more general case where only irreducibility is assumed. The convergence behaviour of the asymptotic averaged range process is then examined in Section 3.4. The expected convergence time is different to the true value for this approximation; however, in many cases the error is very small.

Chapter 4 turns to the endpoint of the approximation process, backtracking adaptive search. This chapter provides a definition of backtracking adaptive search, a means of calculating its expected convergence time and a method of estimating the necessary parameters from an asymptotic averaged range process. In Section 4.2 a definition is provided for backtracking adaptive search on a finite range. This is the case necessary for use in the approximation framework; backtracking adaptive search on a continuous or mixed range is analysed in Chapter 5.

The expected convergence rate of backtracking adaptive search on a finite range is derived in Section 4.3. The following section then derives a more general result where a new parameter is specified for the distribution of the initial iteration. Several special cases of this backtracking adaptive search algorithm have been discussed in the literature. Convergence results for these and other examples are presented in Section 4.5. The method by which backtracking adaptive search approximates an asymptotic averaged range process is then treated in Section 4.6. This formally completes the method for obtaining a backtracking adaptive search model for any stochastic global optimisation algorithm.

In Chapter 5 the full distribution of the number of iterations to convergence for backtracking adaptive search is derived. This generalises the previous analysis of backtracking adaptive search in Chapter 4. Backtracking adaptive search is defined for a general range distribution in Section 5.2. In

Section 5.3 factorial moment generating functions are then derived for backtracking adaptive search in each possible case, where the range distribution is discrete, continuous or mixed. Factorial moment generating functions are used to derive the mean and variance of backtracking adaptive search in the case where the range distribution is finite (as it is in the application of the approximation framework of Chapter 2) and an expression for the mean in the case where the range distribution is continuous.

Computational results for the approximation framework are presented in Chapter 6. In this chapter the strategy proposed in Chapter 2 for estimating convergence rates of stochastic global optimisation algorithms is demonstrated. A few examples illustrate various aspects of the strategy and indicate how it can be implemented in practice. The intention here is not to showcase a completed piece of software, but to display the results of the theoretical investigation that has occupied the rest of this thesis.

In Section 6.2, two example global optimisation algorithms are used to illustrate each stage in the approximation framework. The first of these is small enough for each quantity used to be displayed explicitly; the second is a slightly larger and more challenging situation, demonstrating the approximation framework in a more realistic scenario.

Section 6.3 contains a further two examples. This section highlights some implementation considerations and suggests a way of applying the approximation framework in practice. The first example is again simpler, allowing the expected convergence time estimated by the approximation process to be compared to the true value obtained analytically. The second example shows the approximation process applied to a more challenging problem. The expected convergence time estimated in this case is compared with empirical results of applying the example algorithm several times and recording the convergence time for each run. A gauge as to the quality of convergence rate estimation afforded by the approximation framework is then available.

The final chapter summarises the work accomplished and highlights further areas for research arising out of this work.

Chapter 2

A framework of processes by which backtracking adaptive search approximates a stochastic global optimisation algorithm

2.1 Introduction

The distribution of the number of iterations before convergence is a very useful measure of performance for a stochastic global optimisation algorithm. Unfortunately, this distribution is not available for practicable algorithms; indeed, the distribution can only be found empirically. This analysis is available for backtracking adaptive search (and presented later in this thesis); however, backtracking adaptive search is not a practicable algorithm. Thus, on the one hand are algorithms having practical implementation but without any analysis of expected runtime; on the other, algorithms possessing this analysis but without any practical implementation. The challenge to be faced is in finding a way to meld these two desiderata together—to find a way of applying the kind of analysis available for backtracking adaptive search to practicable stochastic global optimisation algorithms.

The approach of this thesis is therefore to attempt to find a backtracking adaptive search approximation to a general stochastic global optimisation algorithm. The extent to which this approximation is accurate is then the extent to which convergence information concerning the backtracking adaptive search algorithm is applicable to the stochastic global optimisation algorithm it approximates.

There are a number of obstacles in the way of such an approximation: in general stochastic global optimisation algorithms can be non-Markovian time-inhomogeneous processes defined on the problem domain, while backtracking adaptive search (in the form to be presented later) is a Markovian time-homogeneous process, with a particular structure, defined on the problem range.

One possible approach to overcoming these obstacles would be to generalise the definition of backtracking adaptive search to a more encompassing family of processes, more easily able to describe the movement of a general stochastic global optimisation algorithm. Backtracking adaptive search is itself a generalisation of hesitant random search [11] and pure random search [54]; the generalisation is in the distribution of each successive iterate. In pure adaptive search, the next iterate is chosen from the improving set according to a distribution ρ on the range; in hesitant adaptive search, the next iterate can either be chosen from the improving set according to ρ or else remain at the current objective function level. Backtracking adaptive search again uses the distribution ρ but allows the next iterate to be taken from anywhere in the range, and is thus expected to provide a closer approximation to a general stochastic global optimisation algorithm, since in general algorithms may worsen at any given iteration as well as improve.

Still further generalisation is certainly possible; the structure imposed on backtracking adaptive search could be weakened by allowing variation in the range distribution ρ, or perhaps varying the way in which ρ is used. It may even be possible to define something like backtracking adaptive search directly on the domain. However, the point of specialising the form of the algorithm is that the distribution of the number of iterations before convergence can be evaluated easily; further complicating the process makes this distribution harder to obtain. At some point, accuracy of the approximating algorithm must be exchanged for susceptibility to analysis.

The nature of statistics is to summarise a complicated process with a simple model that can be analysed. This is the approach of this thesis, and the simple model chosen in what follows is the representation of backtracking adaptive search described later. This thesis marks a first step towards development of an adaptive search approximation to a general stochastic global optimisation algorithm; improvements in the model or the estimation of its parameters may well stem from future research.

Using backtracking adaptive search as the end-point of the approximation process, then, the challenge is to estimate its parameters from the domain based stochastic process (DP) of a general global optimisation algorithm. In particular, major difficulties lie in the structure of backtracking adaptive search; how can a process that is Markovian, time-homogeneous and defined in the range approximate an algorithm which in general obeys none of these conditions?

The approximation process is divided into several stages, employing three intermediate stochastic processes dealing with each of the three difficulties listed above. Firstly, the range process (RP) is defined as the image of the original domain process in the range. Secondly, a Markov process in the range, the averaged range process (ARP), is defined based closely on the range process. Thirdly, the asymptotic averaged range process (AARP), a time-homogeneous variant of the averaged range process, is constructed. Having now obtained a Markovian time-homogeneous process in the range, it is possible to choose parameters for backtracking adaptive search in order to reflect as closely as possible the characteristics of this process.

Note that it is assumed that any optimisation problem has a one dimensional objective function measuring the quality of each feasible solution. Even problems with multiple objective functions can be incorporated in this study, by combining each objective function in a weighted sum. Any automated multiobjective optimisation algorithm must have a means of determining whether one solution is better or worse than another; this implicitly defines a one dimensional ordering of all possible solutions. Various methods of defining this ordering have been proposed (see, for instance, [47]). The requirement of a one dimensional objective function is thus not restrictive. Furthermore, the fact that every problem possesses this simplifying measure recommends any analysis that is able to make use of this trait.

A final observation on the practical issue of implementing this approximation process: for a general stochastic global optimisation algorithm, even the structure of the original algorithm in the problem domain is unknown and must be estimated empirically. However, there is no reason that one of the processes later in the approximation framework could not also be described empirically. A practical implementation of the approximation process could begin by estimating the parameters of the asymptotic averaged range process and simply approximating immediately with backtracking adaptive search, skipping over the first two steps in the framework. The theoretical descriptions of the intermediate processes remain valid and useful in justifying the techniques used, even though these processes may never be defined in the implementation of the approximation process. Implementation details are discussed in Chapter 6.

In the next section a generic stochastic global optimisation algorithm is defined; all subsequent analysis centres on this algorithm. The sequence of approximations is then described, moving from this domain process through to a backtracking adaptive search process in several stages. Analytical results regarding each of the intermediate processes are reserved until Chapter 3; this chapter aims only to establish the framework of approximations that forms the major idea of the thesis. A final section summarises the import of this framework.

2.2 Approximating a stochastic global optimisation algorithm

The problem under consideration is the very general global optimisation problem

$$\text{minimise } f(x), \text{ subject to } x \in S$$

where S is a measurable space and $f : S \to \mathbb{R}$ is a measurable function. A general form for stochastic global optimisation algorithms is now introduced. The algorithm selects the first point according to a distribution δ_0 on the domain. Thereafter a local search measure for each $x \in S$, in general dependent on the iteration number n, is used to generate the next candidate point in the domain in a Markov process.

Generic stochastic global optimisation algorithm

Step 1 Generate X_0 in S according to δ_0. Set $Y_0 = f(X_0)$ and $n = 0$.

Step 2 Choose X_{n+1} according to a local search measure at X_n. Set $Y_{n+1} = f(X_{n+1})$.

Step 3 If a stopping criterion is met, stop. Otherwise, increment n and return to Step 2.

At Step 2 of the above algorithm, it is common (but of course not necessary) to first select a candidate point Z; if $f(Z) \leq Y_n$ then set $X_{n+1} = Z$, if $f(Z) > Y_n$ then set $X_{n+1} = Z$ with a certain probability (which may depend on X_n and Z) or otherwise set $X_{n+1} = X_n$.

To investigate the behaviour of this algorithm, a sequence of stochastic processes is constructed, each approximating the progress in the range of the generic stochastic global optimisation algorithm. As each successive stochastic process approximates the last, the accuracy of the approximation to the original stochastic algorithm will slowly deteriorate; however, the advantage of the procedure is that the distribution of the convergence rate of the final backtracking adaptive search approximation is known. The following stochastic processes are discussed:

Domain process: This is the sequence of domain points generated by the generic stochastic global optimisation algorithm.

Range process: This is the sequence of function values generated by the generic stochastic global optimisation algorithm; it is essentially the same as the domain process, but viewed in the range.

Averaged range process: This is a time-inhomogeneous Markov process in the range based on the range process. It is presented in Subsection 2.2.1.

Asymptotic averaged range process: This is obtained by homogenising the averaged range process. It is presented in Subsection 2.2.3.

Backtracking adaptive search: This is obtained by requiring the transition distributions in the range, conditioned on improvement or worsening, to be restrictions of a single range probability measure. It is presented and analysed for finite domains in Chapter 4 and for general domains in Chapter 5.

The first of these processes takes on values in the objective function's domain, whereas the others assume values in the range. This important distinction is illustrated in Figure 2.1.

The domain process (X_n) as defined above is Markovian. This is general enough to encompass simulated annealing [29]. Even with a further restriction that the algorithm should be time-homogenous,

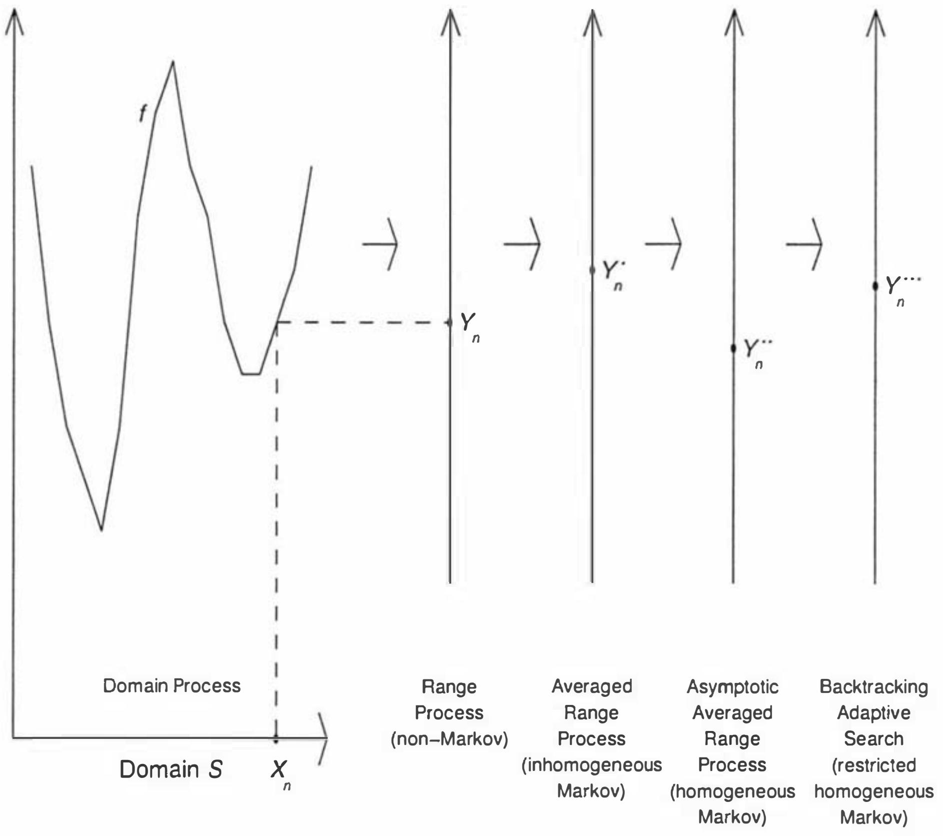
f
Y_n
$Y_n^{\cdot}$
$Y_n^{\cdot\cdot}$
$Y_n^{\cdot\cdot\cdot}$
Domain Process
Domain S
X_n
Range Process (non–Markov)
Averaged Range Process (inhomogeneous Markov)
Asymptotic Averaged Range Process (homogeneous Markov)
Backtracking Adaptive Search (restricted homogeneous Markov)

Figure 2.1: Steps in the approximation of the domain process with a backtracking adaptive search process.

the Metropolis algorithm [35], genetic algorithms [24] (with S then a product space, the number of factors being determined by the population size) and certain evolutionary algorithms [24] are all included within this definition.

The framework of approximations presented could be extended from the generic stochastic global optimisation algorithm defined above to any time-homogeneous Markov process in the domain. In fact, some time-inhomogeneous Markov processes could be approximated also; however, the analysis provided here is limited to time-homogeneous algorithms. The question of what processes may be approximated with backtracking adaptive search is treated in detail in Subsection 3.3.

Note that the range process (Y_n) is not in general Markovian, despite being the image under f of the Markov chain (X_n). Knowledge of the current domain state X_n provides complete information at any iteration concerning the distribution of the next iterate in the domain; however, knowledge of the current range level Y_n alone does not in general provide complete information concerning the distribution of the next iterate in the range.

For instance, suppose the objective function $f(x) = |x|$ is to be minimised over the set $\{-1, 0, 1, 2\}$, and successive iterates X_{n+1} are chosen randomly from those values of the set $\{x_n - 1, x_n, x_n + 1\}$ that fall in the domain. Knowledge that the current range level is 1 implies only that the domain location could be either -1 or 1 and is thus insufficient to determine the probability of visiting range level 0 at the next iteration; but if the previous iteration is known to be 2 then the current domain location must be 1 and the probability of visiting range level 0 at the next iteration may now be correctly determined. It is thus demonstrated that (Y_n) is not in general Markovian. Note also that if the history of the process is $(2, 1, 1, 1, \ldots, 1)$ then the entire history must be known in order to correctly determine the current transition probabilities. There is thus no Markov process of any order that accurately represents the range process.

In what follows, only first-order Markov approximations to the range process are considered; re-definition of the state space to include more than one iteration in each state would easily allow generalisation to higher order Markov approximations, perhaps providing worthwhile improvement in the accuracy of the approximation. This procedure is common in some global optimisation algorithms, such as Tabu Search [20].

Notation required in the following discussion is now described. A discrete domain S is assumed. Convergence is held (as in [54]) to occur when $Y_n \leq y$ for some real number y, chosen before algorithm commencement; domain locations with objective function value not greater than y are lumped into one absorbing state x_1 with arbitrary objective function level $y_1 \leq y$. (The analysis may be applied to study algorithm performance on test problems whose global minima are known; in this case y can be set within a certain tolerance of the global minimum. Even when the optimal solution itself is unknown, it may be possible in certain cases to find the value of the global minimum from the problem definition. Otherwise y must be chosen in some other way, perhaps with reference to the current best known objective function value.) It is assumed that all other domain locations are transient. The domain search algorithm may thus be considered as a Markov process with the following transition matrix, in block form:

$$P = \left[\begin{array}{c|c} 1 & \mathbf{0} \\ \hline \boldsymbol{r} & Q \end{array}\right].$$

The first row of this matrix, showing transitions from the absorbing state, has a one in the first position and zeros elsewhere, since the algorithm is held to terminate after first sampling x_1. (Only the iterations before attaining a state with objective function level not greater than y are of interest.) The column vector $\boldsymbol{r}$ gives the probability of moving directly to the absorbing state for each transient state. The remaining submatrix Q is substochastic (that is, the elements of Q are positive and the row sums are no greater than 1), and gives the transition probabilities between all of the transient states. (In practice P will not be known. Implementation issues are discussed in Chapter 6.)

Denote the transient domain states by $x_2, x_3, \ldots, x_l$ and the transient objective function levels by $y_2, y_3, \ldots, y_m$. Now let δ_n denote a row vector of length l comprising the probabilities of occupying each of the l domain states at the nth iteration. Then $\delta_{n+1} = \delta_n P$ for all $n \geq 0$.

It is useful to define a truncation transformation T_i, equal to the identity matrix of size i (I_i) with the first column removed. This matrix has the effect of removing the first component of a probability position vector, corresponding to the probability of being in the absorbing state. A vector of i ones is also denoted by $\mathbf{1}_i$.

A simple example is now introduced which will serve to illustrate the concepts introduced throughout this chapter.

Example Let $S = \{1, 2, 3\}$, $f(1) = 1$, $f(2) = 2$ and $f(3) = 2$. A search algorithm to find the minimum in this simple example is described by a Markov domain process with transition matrix

$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ \frac{1}{2} & \frac{1}{2} & 0 \end{bmatrix}, \text{ so that } \quad Q = \begin{bmatrix} 0 & 1 \\ \frac{1}{2} & 0 \end{bmatrix}.$$

Suppose the process is equally likely to begin in each of the domain states (so $\delta_0 = [\frac{1}{3}\ \frac{1}{3}\ \frac{1}{3}]$). Then standard Markov chain theory [27] provides that the expected number of iterations before convergence for this process is $\delta_0 T_3(I_2 - Q)^{-1}1_2 = 2\frac{1}{3}$.

The range process based on this domain algorithm has an absorbing state at the low level, 1. At the high level, 2, there are two possibilities at each iteration. If the domain process is at domain state 2 then (as is apparent from the transition matrix) it will move to domain state 3 with certainty and thus the range process will remain at the high level; if the domain process is at domain state 3 then RP is equally likely to remain at the high level or move to the low level.

The other two intermediate approximations of the range process are now presented in more detail. The final backtracking adaptive search approximation is treated in Chapter 4.

2.2.1 The averaged range process

In this stage RP is approximated with an inhomogeneous Markov process, ARP($Y_n^{\cdot}$). Here is an informal description of this process. At the nth iteration ARP is at level $Y_n^{\cdot}$; thus the corresponding state in the domain is some point in $f^{-1}(Y_n^{\cdot})$. The conditional probability of being at each of these domain states is determined by the normalised restriction of the domain distribution at the nth iteration to $f^{-1}(Y_n^{\cdot})$. Mappings of the transition distributions at each candidate domain state into the range (termed *local range distributions*) are then mixed according to their domain weightings. This produces an averaged range distribution at the nth iteration. Figure 2.2 illustrates the way in which ARP uses this convex combination of local range distributions of the search algorithm.

A formal definition of the averaged range process is now given. The initial distribution of ARP is identical to that of RP, being the image in the range of the initial distribution of the original domain process. At the nth iteration, suppose that ARP is in state y_i. Then the probability that ARP moves to state y_j at the next iteration is found by summing all the probabilities of distinct transitions in the domain from $f^{-1}(y_i)$ to $f^{-1}(y_j)$. That is, the ARP transition matrix at the nth iteration, R_n, is given by

$$\begin{aligned}
(R_n)_{ij} &= \mathrm{P}(Y_{n+1}^{\cdot} = y_j | Y_n^{\cdot} = y_i) \\
&= \sum_{x_p \in f^{-1}(y_j)} \sum_{x_k \in f^{-1}(y_i)} \mathrm{P}(X_n = x_k | Y_n = y_i) \mathrm{P}(X_{n+1} = x_p | X_n = x_k) \qquad (2.1) \\
&= \mathrm{P}(Y_{n+1} = y_j | Y_n = y_i) \qquad (2.2)
\end{aligned}$$

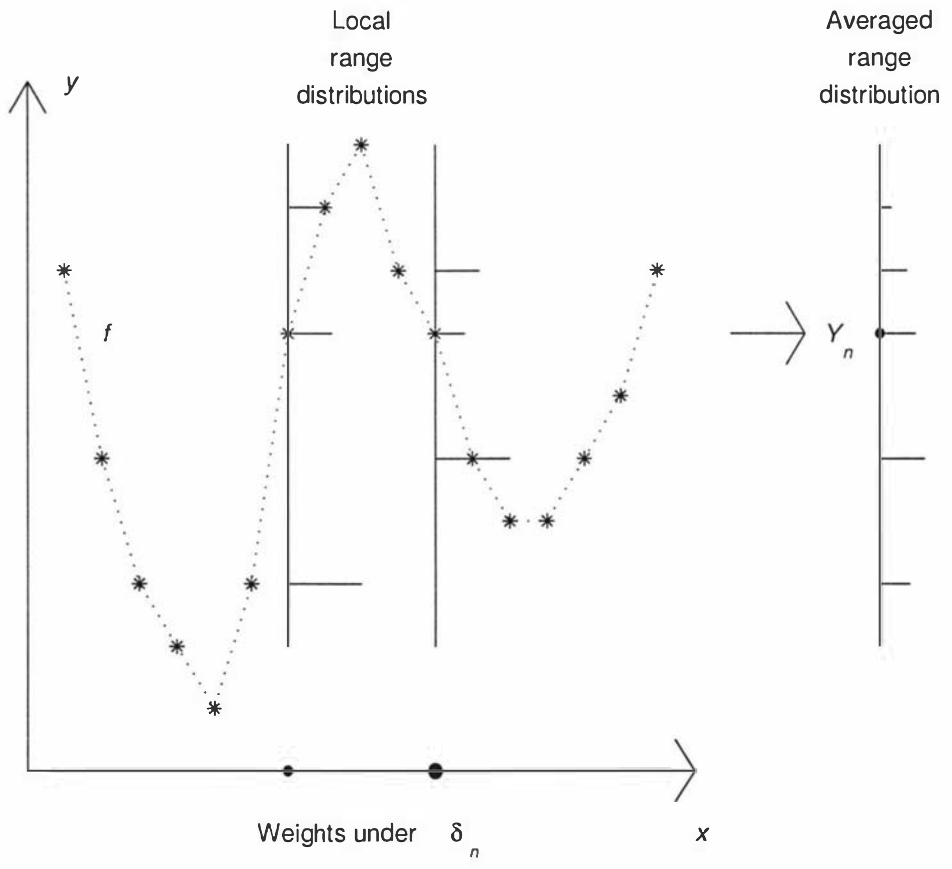

Figure 2.2: The Markovian approximation to the range process uses a mixture of local range distributions, with weights determined by the domain process. In this figure, the current range value may be due to either of two domain points, each of which gives rise to a probability distribution for the range value after one further iteration. These probability distributions are mixed as shown.

for $n \geq 0$ and $i, j \in \{1, 2, \ldots, m\}$ when $\mathrm{P}(Y_n = y_i) \neq 0$, and $(R_n)_{ij}$ is arbitrarily set to zero otherwise. (Alternative arbitrary assignments in the case when $\mathrm{P}(Y_n = y_i) = 0$ are possible, such as setting $(R_n)_{ij}$ to 1 if $i = j$ and 0 otherwise, or assigning equal values to $\mathrm{P}(X_n = x_k | Y_n = y_i)$ for all k such that $f(x_k) = y_i$ and proceeding according to Equation (2.1). The simpler definition is retained, however, as in [49]. This has the added benefit of making the need for a refined construction of the asymptotic averaged range process transition matrix as in Theorem 3.3.3 more apparent, as illustrated in the final example of Section 3.3.)

The construction of this transition matrix may be represented more compactly using the following definitions. Define domain weightings used by ARP, γ_n, based only on the current range level and iteration number, as

$$\gamma_n(i) = \mathrm{P}(X_n = x_i | Y_n = f(x_i)) = \frac{\mathrm{P}(X_n = x_i)}{\mathrm{P}(Y_n = f(x_i))} = \frac{\delta_n(i)}{\sum\limits_{\{j:f(x_j)=f(x_i)\}} \delta_n(j)} \tag{2.3}$$

for $i \in \{1, 2, \ldots, l\}$ where the denominator is positive. Otherwise, arbitrarily assign $\gamma_n(i) = 0$.

An $l \times m$ matrix M mapping from domain states to objective function states is now defined as

$$M_{ij} = \begin{cases} 1 & \text{if } f(x_i) = y_j, \\ 0 & \text{otherwise.} \end{cases}$$

Then

$$R_n \quad = \quad M^{\mathrm{T}}\mathrm{diag}(\gamma_n)PM \tag{2.4}$$

where diag(γ_n) is a diagonal matrix with entries of γ_n on the diagonal.

By way of justification for this expression, note that $M^{\mathrm{T}}_{ij} = 1$ whenever $f(x_j) = y_i$. Then, provided $P(Y_n = y_i) > 0$,

$$\begin{aligned} \left(M^{\mathrm{T}}\mathrm{diag}(\gamma_n)\right)_{ij} &= P(X_n = x_j | Y_n = y_i) \\ \left(M^{\mathrm{T}}\mathrm{diag}(\gamma_n)P\right)_{ij} &= P(X_{n+1} = x_j | Y_n = y_i) \\ \left(M^{\mathrm{T}}\mathrm{diag}(\gamma_n)PM\right)_{ij} &= P(Y_{n+1} = y_j | Y_n = y_i) \\ &= (R_n)_{ij} \end{aligned}$$

by Equation (2.2). If $P(Y_n = y_i) = 0$ then $\left(M^{\mathrm{T}}\mathrm{diag}(\gamma_n)\right)_{ij} = 0$ for all j, so $\left(M^{\mathrm{T}}\mathrm{diag}(\gamma_n)PM\right)_{ij} = 0$. In either case the definition of R_n in Equation (2.4) is identical with the definition provided earlier.

The sequence of processes studied in this thesis span a conceptual crevasse between, on one side, the generic stochastic global optimisation algorithm of interest, and on the other, BAS, for which full convergence analysis is available (and presented in Chapter 4). The value of this study is determined by the quality of the approximations; however, detailed derivations of the properties of each process are left to later chapters. To anticipate Theorem 3.2.1, though, it is in fact the case that the averaged range process and the range process (and therefore the domain process) share the same expected number of iterations before termination.

This fact is now illustrated in the simple example.

Example (continued) The objective function of the example problem implies

$$M = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}.$$

For this cyclic example, explicit expressions for P^n are available as

$$P^n = 2^{-\frac{n}{2}} \begin{bmatrix} 2^{\frac{n}{2}} & 0 & 0 \\ 2^{\frac{n}{2}} - 1 & 1 & 0 \\ 2^{\frac{n}{2}} - 1 & 0 & 1 \end{bmatrix} \text{ for } n \text{ even; } \quad P^n = 2^{-\frac{n+1}{2}} \begin{bmatrix} 2^{\frac{n+1}{2}} & 0 & 0 \\ 2^{\frac{n+1}{2}} - 2 & 0 & 2 \\ 2^{\frac{n+1}{2}} - 1 & 1 & 0 \end{bmatrix} \text{ for } n \text{ odd.}$$

Expressions for δ_n can be calculated as $\delta_0 P^n$. Thus when n is even $\delta_n = 2^{-\frac{n}{2}}[2^{\frac{n}{2}} - \frac{2}{3}\ \frac{1}{3}\ \frac{1}{3}]$ and when n is odd $\delta_n = 2^{-\frac{n+1}{2}}[2^{\frac{n+1}{2}} - 1\ \frac{1}{3}\ \frac{2}{3}]$. Then by Equation (2.3),

$$\begin{aligned} \gamma_n &= \begin{bmatrix} \frac{2^{\frac{n}{2}} - \frac{2}{3}}{2^{\frac{n}{2}} - \frac{2}{3}} & \frac{\frac{1}{3}}{\frac{1}{3} + \frac{1}{3}} & \frac{\frac{1}{3}}{\frac{1}{3} + \frac{1}{3}} \end{bmatrix} \\ &= \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{2} \end{bmatrix} \end{aligned}$$

for n even, and similarly

$$\gamma_n = \begin{bmatrix} 1 & \frac{1}{3} & \frac{2}{3} \end{bmatrix}$$

for n odd. All the ARP transition matrices can now be found explicitly by application of Equation (2.4). Thus

$$\begin{aligned}
R_n &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{2} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ \frac{1}{2} & \frac{1}{2} & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} 1 & 0 \\ \frac{1}{4} & \frac{3}{4} \end{bmatrix}
\end{aligned}$$

for n even, and similarly

$$R_n = \begin{bmatrix} 1 & 0 \\ \frac{1}{3} & \frac{2}{3} \end{bmatrix}$$

for n odd. Since there is only one absorbing domain state and corresponding range level, the first row and column of M are vectors of zeros with an initial one, and the first element of γ_n is a one. Also the first rows of transition matrices P and R_n are always vectors of zeros with an initial one. In this small example the transient portions of the range transition matrices R_n are single numbers, giving the probability of staying at the transient range level after one transition.

The expected value of the number of iterations before convergence for the averaged range process, N_a, can now be found as follows:

$$\begin{aligned}
\mathrm{E}(N_a) &= \sum_{n=1}^{\infty} \mathrm{P}(N_a \geq n) \\
&= \frac{2}{3} + \frac{2}{3} \cdot \frac{3}{4} + \frac{2}{3} \cdot \frac{3}{4} \cdot \frac{2}{3} + \frac{2}{3} \cdot \frac{3}{4} \cdot \frac{2}{3} \cdot \frac{3}{4} + \cdots \\
&= 2\frac{1}{3}
\end{aligned}$$

This is the same result as that stated earlier, from direct analysis of the process in the domain. Thus, in this example, ARP preserves the expected number of iterations to convergence of the domain process it approximates. Later this is shown to be true in general.

2.2.2 The asymptotic averaged range process

While ARP is a time-inhomogeneous Markov process, BAS is defined later as a time-homogeneous Markov process. To link the two, it is necessary to remove the iteration dependence of ARP. This is done by observing that the transition matrices R_n for ARP settle down in the limit to a constant matrix R, or, possibly, to oscillation between multiple constant matrices (in which case R is defined as the average of these limiting matrices). This is proven for all sequences of transition matrices R_n in Chapter 3.

The asymptotic averaged range process (AARP) is defined to be the time-homogeneous Markov process with initial distribution equal to that in both the range process and the averaged range process, and with transition matrix R given (in most cases) by

$$R = \lim_{N \to \infty} \frac{1}{N} \sum_{n=0}^{N-1} R_n,$$

as in [49]. This definition may be informally justified by averaging the first N transition matrices R_n and then obtaining R as the limit to which this average tends as N tends to infinity. Beyond a certain number of iterations the R_n will be cycling through the set of constant limiting matrices (with negligible discrepancies), so the average over all N matrices $R_0, R_1, \ldots, R_N$ will tend to the average of those limiting matrices. The exact definition requires more care to handle possible circumstances under which this definition leads to substochastic R (the details are provided in Chapter 3), but the definition above is correct whenever δ_0 gives positive weight to all x_i (so that the probability of commencing in any domain state is positive). This is a mild restriction that could be easily satisfied, for instance by commencing with pure random search with some positive probability. Except in special cases where the transition matrices R_n settle down in the limit to oscillation between multiple limit matrices, the expression simplifies to $R = \lim_{n \to \infty} R_n$.

Example (continued) The example illustrates the cyclic case. The limiting average transition matrix may be found as follows (since in this example δ_0 gives positive weight to all x_i):

$$\begin{aligned}
R = \lim_{N\to\infty} \frac{1}{2N} \sum_{n=0}^{2N-1} R_n &= \lim_{N\to\infty} \frac{1}{2N} \sum_{n=0}^{N-1} \left(\begin{bmatrix} 1 & 0 \\ \frac{1}{4} & \frac{3}{4} \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ \frac{1}{3} & \frac{2}{3} \end{bmatrix} \right) \\
&= \lim_{N\to\infty} \frac{1}{2N} N \begin{bmatrix} 2 & 0 \\ \frac{7}{12} & \frac{17}{12} \end{bmatrix} \\
&= \begin{bmatrix} 1 & 0 \\ \frac{7}{24} & \frac{17}{24} \end{bmatrix}
\end{aligned}$$

Applying standard Markov Chain theory to AARP using this limiting transition matrix gives an expected number of iterations to convergence of $\frac{2}{3}\left(1 - \frac{17}{24}\right)^{-1} = 2\frac{2}{7}$. This is close to the value for the domain process, $2\frac{1}{3}$. Some means of quantifying the error now introduced by the approximation to AARP is required; discussion of this problem is offered in Chapter 3.

The asymptotic averaged range process is of some interest in itself: it provides a greatly simplified representation of a global stochastic optimisation algorithm. It has the simple form of a time-homogeneous Markov process, and it operates in the range, which is one dimensional, instead of the domain, which is generally multidimensional. The number of possible range levels is potentially far less than the number of possible domain locations. If the expected number of iterations before convergence for the asymptotic averaged range process can be shown to be close to that of the original domain process, then obtaining this information from the asymptotic averaged range process will therefore be in general much simpler computationally.

2.2.3 Backtracking adaptive search

Since even the asymptotic averaged range process is still in general of a size that prohibits direct calculation of its convergence rate, another stage of approximation is required. The important result of the stages of approximation shown thus far is that a time-homogeneous Markov process in the range has been found to approximate a general stochastic global optimisation algorithm. Since this is the form of backtracking adaptive search, it is possible now to approximate the asymptotic averaged range process using backtracking adaptive search.

The simple example of this chapter is so small that the asymptotic averaged range process is already in the form of backtracking adaptive search; the number of iterations before convergence for the backtracking adaptive search approximation is thus identical with the number of iterations before convergence for the asymptotic averaged range process. A full discussion of the details of backtracking adaptive search, including a general method for approximating the asymptotic averaged range process with backtracking adaptive search, is left to Chapter 4.

In Chapter 4 a closed form expression for the expected number of iterations before convergence for the backtracking adaptive search algorithm is presented. This is the end-point of the approximation sequence described in this thesis. Convergence results for an approximation of a general stochastic global optimisation algorithm are thus available.

2.3 Summary

The framework described in this chapter provides a means for approximating the convergence rate of an arbitrary Markovian optimisation algorithm, by linking it to a tractable stochastic process via a chain of intermediate stochastic processes. Each process in the chain is derived from the previous one, and can be used to approximate its convergence behaviour.

The complete strategy for analysis enables predictions to be made of how long a particular stochastic global optimisation algorithm should be run to reach a set level. The effectiveness of the stochastic global optimisation algorithm on a particular problem is thus measured.

The following chapter fills in some of the theoretical detail concerning the averaged range process and the asymptotic averaged range process, and Chapter 4 then describes backtracking adaptive search and the method by which it approximates the asymptotic averaged range process. Computational results for the entire approximation framework are reported in Chapter 6.

Chapter 3

The range, averaged range and asymptotic averaged range processes

3.1 Introduction

This chapter extends and amplifies Chapter 2, analysing the averaged range process and the asymptotic averaged range process in more detail. It is shown formally that these processes are well defined, and the quality of their approximation of the domain process is examined. In particular, attention is focused on the number of iterations before convergence. The major aim of approximation is to find a practicable method of estimating the mean of this quantity. Such a method would be of considerable practical use.

The following section shows that the distribution of the number of iterations before convergence for the averaged range process is the same as for the domain process. Section 3.3 demonstrates the existence of a well defined asymptotic averaged range process approximation to any domain process. Section 3.4 then provides some discussion of the number of iterations to convergence for this process. Finally, a brief summary consolidates the results shown thus far in the framework. The last step in approximating the asymptotic averaged range process with a backtracking adaptive search process will be considered in Chapter 4.

3.2 The number of iterations before convergence for the averaged range process

The quality of each successive approximation is determined by the error introduced in estimating the number of iterations until convergence for the original domain process, based on the approximation. This section shows that the first stage of approximation in fact introduces no error in estimating the distribution of this quantity.

It should be borne in mind, however, that the expected convergence time of the averaged range process is still in general difficult to evaluate in practice; the reason for continuing the approximation process to backtracking adaptive search is that the expected convergence time for backtracking adaptive search can be calculated efficiently by computer. The result shown here is a theoretical justification of the first part of this approximation process.

Recall the definition of the averaged range process given in Subsection 2.2.1. The averaged range process $(Y_n^{\cdot})$ takes values on the range indexed as $y_1, y_2, \ldots, y_m$, where y_1 is the optimal level. The domain process (X_n) takes values on the domain $x_1, x_2, \ldots x_l$.

Where $(Y_n^{\cdot})$ goes at each iteration will in general depend on more than the last step. As the definition of ARP is based only on the current level and iteration number, therefore, it is distanced somewhat from RP. Despite this, the definition provided in Subsection 2.2.1 implies that, while the averaged range process and the range process can in general differ in joint distribution, they must be equal in marginal distribution at each iteration.

Theorem 3.2.1 *At any given iteration, the range process and the averaged range process have the same distribution.*

Proof The notation required for the averaged range process is first described.

As in Subsection 2.2.1, define $\gamma_n(i) = \mathrm{P}(X_n = x_i | Y_n = f(x_i))$ for $i \in \{1, 2, \ldots, l\}$ whenever

$P(Y_n = f(x_i)) > 0$; otherwise, arbitrarily assign $\gamma_n(i) = 0$. Also let

$$M_{ij} = \begin{cases} 1 & \text{if } f(x_i) = y_j, \\ 0 & \text{otherwise,} \end{cases}$$

so that the averaged range transition matrix at the nth iteration can be written as

$$R_n = M^{\mathrm{T}}\mathrm{diag}(\gamma_n)PM \tag{3.1}$$

where $\mathrm{diag}(\gamma_n)$ is a diagonal matrix with entries of γ_n on the diagonal.

The probabilities of the domain process occupying any state in the domain at the nth iteration are given by the vector δ_n where $n \geq 0$; define the averaged range process equivalent, π_n, to be a vector containing the probabilities of the averaged range process occupying any level in the range at the nth iteration for all $n \geq 0$. The definition of the range process implies that the probability distribution of range states in RP is the image of δ_n under f. The theorem may thus be stated as

$$\pi_n = \delta_n M.$$

The theorem may now be proved by induction. The initial averaged range distribution is given as the image in the range of the initial domain distribution, so that $\pi_0 = \delta_0 M$.

Now assume the result for some integer $k \geq 0$. Note that

$$\begin{aligned} \left[\delta_k M M^{\mathrm{T}}\mathrm{diag}(\gamma_k)\right](i) &= \mathrm{P}(Y_k = f(x_i))\mathrm{P}(X_k = x_i | Y_k = f(x_i)) \\ &= \mathrm{P}(X_k = x_i) \\ &= \delta_k(i) \end{aligned} \tag{3.2}$$

when $P(Y_k = f(x_i)) > 0$. If $P(Y_k = f(x_i)) = 0$, $[\pi_k M^{\mathrm{T}}\mathrm{diag}(\gamma_k)](i) = 0 = \delta_k(i)$ so Equation (3.2) still holds. Thus

$$\begin{aligned}
\pi_{k+1} &= \pi_k R_k && \text{by definition} \\
&= \delta_k M M^{\mathrm{T}} \mathrm{diag}(\gamma_k) P M && \text{by hypothesis and using Equation (3.1)} \\
&= \delta_k P M && \text{from Equation (3.2)} \\
&= \delta_{k+1} M.
\end{aligned}$$

Hence by induction $\pi_n = \delta_n M$ for all $n \geq 0$. ■

It is now possible to go further and prove the surprising result that the distribution of the number of iterations before convergence is in fact the same in both the domain process and the averaged range process—despite the fact that the two processes differ in joint distribution.

The following example illustrates the difference in joint distribution between a domain process and its averaged range process approximation. Suppose the function x^x is to be minimised over the domain $\{-1, 0, 1, 2\}$, and the algorithm applied in the domain at each iteration when $x > -1$ is simply to move to $x - 1$ with probability 0.5, or otherwise to remain at x. Clearly, at least three steps are required to find the optimum starting from $x = 2$.

The range values in this example are -1, 1 and 4 (using the convention that $0^0 = 1$); the averaged range process then assigns positive probabilities to the transitions from level 4 to level 1 and from level 1 either to level -1 or level 1. The values of these latter two transition probabilities depend on the relative likelihood of being at $x = 0$ or $x = 1$ at the current iteration. If the initial distribution places positive weight on all four domain states, though, it is clear that it is possible for the averaged range process approximation to find the optimum in only two steps starting from level 4, where $x = 2$.

The averaged range process can thus produce sample paths corresponding to impossible transitions in the domain. Nonetheless, the following corollary shows that the *lengths* of sample paths from algorithm commencement to convergence in each process are identically distributed.

Corollary 3.2.1 *The distribution of the number of iterations to convergence for the generic stochastic global optimisation algorithm in a finite domain is identical to the distributions for the corresponding range and averaged range processes.*

Proof The range process is defined as the image of the domain process under f. Consequently, the number of iterations to convergence for this process is stochastically equivalent to the number of iterations to convergence in the original domain algorithm. Reference to RP is by way of a stepping stone to the new Markov process in the range, ARP.

Let F_d be the cumulative distribution function of the number of iterations to convergence for the domain process, and F_a be the cumulative distribution function of the number of iterations to convergence for the averaged range process. Then if N_d is the number of iterations to convergence for the domain process,

$$\begin{aligned} F_d(n) &= P(N_d \leq n) \\ &= P(X_n = x_1) \\ &= \delta_n(1), \end{aligned}$$

the first component of the vector δ_n. Similarly $F_a(n) = \pi_n(1)$. Now since there is only one state at the optimal level, the probability of the domain process being in the optimal state is the same as the probability of the range process being in the optimal level. Theorem 3.2.1 then shows that this probability is the same as the probability of the averaged range process occupying the optimal level. Thus

$$\begin{aligned} F_d(n) &= \delta_n(1) \\ &= \pi_n(1) \\ &= F_a(n) \end{aligned}$$

so $F_d = F_a$ and the proof is complete. ■

Thus the first stage in the approximation is exactly equivalent to the original optimisation algorithm, in terms of the distribution of the number of iterations before convergence. This result corrects [49], where the number of iterations before convergence in the averaged range process is said to differ from the number of iterations before convergence in the domain process in moments higher than the first. The equality of means has been illustrated in the example of Subsection 2.2.1.

The ensuing section moves on to the next stage of approximation, showing that the asymptotic averaged range process provides a well defined approximation to any Markovian global optimisation algorithm.

3.3 Existence of the asymptotic averaged range process

This section demonstrates the existence of the asymptotic averaged range process approximation to any averaged range process. Then, since the averaged range process has been defined for all domain processes, the whole approximation framework is defined up to this stage. The proof is straightforward for most algorithms and problems, but care is required in making sure that the asymptotic averaged range process is well defined in each possible case.

To prove AARP exists is to prove that, as n tends to infinity, R_n tends to a constant matrix, or possibly to oscillation between a finite number of constant matrices. As defined in Equation (2.1) of Subsection 2.2.1, each entry in R_n is constructed as a sum of terms of the form

$$\frac{P(X_n = x_k)P(X_{n+1} = x_p | X_n = x_k)}{P(X_n \in f^{-1}(f(x_k)))} \tag{3.3}$$

over various values of x_p and x_k when $P(X_n \in f^{-1}(f(x_k))) > 0$. The conditional part of this is simply a one-step transition, determined from the transition matrix P, which is assumed constant.

It is possible to relax this assumption somewhat; the conditional part has a well defined limit as n tends to infinity when strong ergodicity obtains in the normalised transient state transition matrix [44, Definition 4.5], and the existence of a quasi-stationary vector implied by this condition means that in fact the whole expression has a limit (this is discussed in greater detail below). Since the main point is that in the limit a quasi-stationary vector should exist, the conditions under which the asymptotic averaged range process is well defined may be weakened even further; however, as stated earlier, only algorithms with constant transition matrix P are considered here.

Taking this conditional part as constant, then, it must still be shown that the remaining part of the expression in (3.3) has a limit. In Subsection 3.3.2 this is proved for primitive transient state transition matrices, and then in Subsection 3.3.3 a general theorem establishes that in fact for any domain process the expression in (3.3) tends to oscillation within a finite set of limiting values. The existence of an asymptotic averaged range process approximation to any time-homogeneous Markovian domain process can then be demonstrated.

Before these results can be proved, though, some knowledge of Markov chain theory is required. An introduction to the important points is presented in the following subsection.

3.3.1 Markov chains

The subsection following this one proves the existence of an asymptotic averaged range process approximation to any domain process where the transient state transition matrix is *primitive*. A matrix is primitive if and only if it is irreducible and acyclic. Define $p_{ij}^n = P(X_n = x_j | X_0 = x_i)$; then a matrix is irreducible if for each pair of states x_i and x_j there exists an n such that $p_{ij}^n > 0$. If the set $\{n : p_{ii}^n > 0\}$ has a greatest common denominator $d > 1$ for some i then the state is cyclic with period d; otherwise the state is acyclic.

All states in a set of states for which the transition matrix is irreducible have the same period $d > 0$ [44, Lemma 1.2] (if the "period" is 1 then the states are acyclic). States in this set are partitioned amongst exactly d non-empty cyclic subclasses (this follows from [44, Theorem 1.3]). The Markov chain moves around these subclasses in order, sampling one state from the current subclass at each iteration.

Note that a transition matrix for more than one state can never be irreducible if one of the states is absorbing, since the probability of a transition from an absorbing state to any other state in n steps is zero for all n. The transition matrix formed by exclusion of all absorbing states, however, may be irreducible. If this transient state transition matrix is also acyclic then it is primitive.

A Markov chain with primitive transition matrix P has a stationary distribution v such that $vP = v$. This stationary distribution is called the left Perron-Frobenius eigenvector of P [44]. Theorem 4.6 of [44], which is repeated in the lemma below, generalises this idea to consider the so-called quasi-stationary distribution of transient states when the transition matrix of transient states only is primitive. Note that I denotes the set of transient states (called inessential in the lemma below).

Lemma 3.3.1 *Let Q, the submatrix of P corresponding to transitions between the inessential states of the Markov chain corresponding to P, be primitive, and let there be a positive probability of $\{X_k\}$ beginning in some $i \in I$. Then for $j \in I$, as $k \to \infty$,*

$$P[X_k = j | X_k \in I] \to v_j^{(2)} \Big/ \sum_{j \in I} v_j^{(2)}$$

where $v^{(2)} = \{v_j^{(2)}\}$ is a positive vector independent of the initial distribution, and is, indeed, the left Perron-Frobenius eigenvector of Q.

Lemma 3.3.1 shows that Q has a quasi-stationary vector according to which the process will tend to distribute itself amongst the transient states before absorption. An extension of the argument applying also to cyclic irreducible transient state transition matrices will be given in Lemma 3.3.3.

The discussion provided so far has hinted at the use of eigenvalues and eigenvectors in analysing Markov chains. The maximum eigenvalue of any stochastic P is 1. If there is only one absorbing state then the corresponding left eigenvector is a row vector with all components equal to zero except for an initial one [44, Theorem 4.7]. (Eigenvalues and eigenvectors of P must satisfy $vP = \lambda v$; when $\lambda = 1$ this reduces to $vP = v$ and v must be the stationary distribution of P. Clearly, the stationary distribution of an absorbing Markov chain will place all the weight in the absorbing state.) If, in addition, the transient state transition matrix Q is primitive, then the second largest eigenvalue has corresponding to it a unique left eigenvector, the transient components of which form the Perron-Frobenius eigenvector of Q, called the quasi-stationary vector. The relative probabilities of being in each transient state, given that the process has not converged, tend over time to limiting values given by the quasi-stationary vector.

If Q is irreducible and cyclic with period d then the largest eigenvalue of Q^d is shown to have d associated eigenvectors in [44, Theorem 1.7].

The remainder of this subsection is spent developing Theorem 3.3.1, which characterises the powers of a matrix (stochastic or otherwise). A result from [12] is required before the statement and proof of this theorem.

Define the Jordan decomposition of any square matrix A as $A = XJX^{-1}$. Denote the distinct eigenvalues of A by $\lambda_1, \lambda_2, \lambda_3, \dots, \lambda_g$. Now let X_i denote the matrix formed by the columns of X associated with λ_i and $X_i^{\#}$ the matrix formed by the rows of X^{-1} associated with λ_i; then $P_i = X_i X_i^{\#}$ is the spectral projection associated with λ_i. The following lemma now describes the spectral decomposition of A.

Lemma 3.3.2 *Each matrix A possesses a spectral decomposition of the form*

$$A = \sum_{i=1}^{g} (\lambda_i P_i + D_i), \quad D_i^{l_i} = 0$$

for some finite l_i, where $D_i = X_i N_i X_i^{\#}$ and N_i is a matrix of the dimensions of $X_i^{\#} X_i$ with all elements equal to zero except for the superdiagonal elements, which are all equal to one except for a number one fewer than the number of linearly independent eigenvectors corresponding to λ_i, which are equal to zero and arranged so that no more than l_i ones on the superdiagonal are consecutive. The following relations are also satisfied:

$$P_i P_j = \delta_{ij} P_i, \quad D_i P_j = \delta_{ij} D_i, \quad D_i D_j = 0 \quad \text{if} \quad i \neq j$$

where δ_{ij} is the Kronecker delta.

This lemma abbreviates the definition provided in [12]. Sufficient detail is provided, however, to introduce the following theorem, which generalises Lemma 3.3.2.

Theorem 3.3.1 *Using the notation of Lemma 3.3.2,*

$$A^n = \sum_{i=1}^{g} \left(\lambda_i^n P_i + \sum_{k=1}^{\min(l_i - 1, n)} \left(\begin{pmatrix} n \\ k \end{pmatrix} \lambda_i^{n-k} D_i^k \right) \right)$$

for each matrix A, using the convention that a summation over an empty set is equal to zero.

Proof Lemma 3.3.2 provides an expansion for A; this theorem follows from the fact that all products in the expansion of A^n are zero excepting only those terms for which the subscript i of each multiplicand is identical.

The relation

$$\begin{aligned} P_i D_j &= X_i X_i^{\#} X_j N_j X_j^{\#} \\ &= \delta_{ij} X_i N_j X_j^{\#} \\ &= \delta_{ij} D_i \end{aligned} \tag{3.4}$$

is required in addition to those given in Lemma 3.3.2. This relation follows since the vectors making up X_i are columns of X and the vectors making up $X_i^{\#}$ are rows of X^{-1}. Now since $X^{-1}X$ is the identity matrix, it follows that $X_i^{\#} X_j = \delta_{ij} I$ where I is an identity matrix of the same size as N_j.

Now consider the non-zero terms of the expansion of A^n. Each of these is a product of n multiplicands, either of the form $\lambda_i P_i$ or D_i. Suppose the first multiplicand in a term is of the form $\lambda_i P_i$; then the next multiplicand must be either of the form $\lambda_j P_j$ or D_j. In either case, referring to Lemma 3.3.2 and Equation (3.4), the product is zero unless $i = j$. Alternatively, the first multiplicand in a term may be of the form D_i; but in this case also, its product with the next multiplicand is zero unless $i = j$. Thus all non-zero terms in the expansion of A^n are composed of n multiplicands sharing an identical subscript i.

If a term is composed solely of multiplicands of the form $\lambda_i P_i$ then, by Lemma 3.3.2, its value is $\lambda_i^n P_i$. Otherwise suppose there are k multiplicands of the form D_i and $n - k$ of the form $\lambda_i P_i$. Lemma 3.3.2 and Equation (3.4) combine to show that the value of this term is $\lambda_i^{n-k} D_i^k$. Since the order of multiplicands in each term is immaterial, this gives

$$A^n = \sum_{i=1}^{g} \left(\lambda_i^n P_i + \sum_{k=1}^{n} \left(\begin{pmatrix} n \\ k \end{pmatrix} \lambda_i^{n-k} D_i^k \right) \right).$$

The proof is completed by noting that $D^{l_i} = 0$, so that the summation of terms in D_i^k need not be taken further than $k = l_i - 1$, even when $n \geq l_i$. ■

These results are developed further in the following sections as they are needed.

3.3.2 Primitive transient state transition matrices

In this subsection the proof of the existence of an asymptotic averaged range process approximation to a domain process is presented in the case where the domain transient state transition matrix is primitive.

Averaged range process transition matrices are constructed by calculating the relative probability of being in any particular state, given that the domain process is known to be in a set of states with the same objective function level. Since there is only one absorbing state, the probability of being in that state given that the process is at the optimal level is exactly one. Moreover, the probability of being in any other state tends to zero over time, since all other states are transient. However, the conditional probability of being in any transient state given that the process is not absorbed tends to a limiting value. This is the result of Lemma 3.3.1. The theorem below now follows.

Theorem 3.3.2 *When the transient state transition matrix Q (introduced in Section 2.2) of a time-homogeneous Markovian domain process is primitive and $P(X_0 = x_1) < 1$, the averaged range process transition matrices R_n tend to a constant limit R as n tends to infinity.*

Proof In view of the discussion at the start of this section, the proof rests in showing that the expression $P(X_n = x_k)/P(Y_n = f(x_k))$ has a limit as n tends to infinity.

Note first that there is only one absorbing state, x_1; hence when $k = 1$, the value of the expression $P(X_n = x_k)/P(Y_n = f(x_k))$ is exactly one (and the limit thus certainly exists). Otherwise, if there is a positive probability that $X_0 \neq x_1$,

$$\begin{aligned}\frac{P(X_n = x_k)}{P(X_n \in f^{-1}(f(x_k)))} &= \frac{P(X_n = x_k)}{\sum\limits_{x_j \in f^{-1}(f(x_k))} P(X_n = x_j)} \\ &= \frac{P(X_n = x_k | X_n \in I)}{\sum\limits_{x_j \in f^{-1}(f(x_k))} P(X_n = x_j | X_n \in I)} \qquad (3.5)\end{aligned}$$

where I is the set of transient states, $\{x_2, x_3, \ldots, x_l\}$. The summation is always over a finite non-empty set, since the number of states at each level is finite (bounded above by l) and x_k is certainly an element of $f^{-1}(f(x_k))$. Since P is irreducible, $P(X_n = x_k)$ is positive for all sufficiently large n.

The probabilities in the top and bottom of the right hand side of Equation (3.5) are identical in form with that proven to have a positive limit as n tends to infinity in Lemma 3.3.1, applicable to any Markov chain where Q is primitive. The proof is thus complete for primitive Q. ■

As R_n tends to R, the difference between the averaged range process and the asymptotic averaged range process becomes very small. This suggests an explanation for the observed proximity of expected convergence times for the two processes, as illustrated in the example of Subsection 2.2.3. A more detailed consideration of this is left to Section 3.4.

An example of the asymptotic averaged range approximation to a domain process with primitive Q is now provided.

Example 1

Take an algorithm with domain transition matrix P as follows

$$\begin{array}{ccc} & & \text{Next state} \\ & & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \text{Current state} & \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \left[\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0.1 & 0.5 & 0.1 & 0.3 \\ 0.2 & 0.8 & 0 & 0 \\ 0.3 & 0.3 & 0.1 & 0.3 \end{array}\right] \end{array}$$

and initial vector $\delta_0 = [0.0816\ 0.1599\ 0.2615\ 0.497]$, randomly generated. The transient portion of P, denoted Q, is primitive. To see this, note that any transient state (states 2, 3 and 4) can reach any transient state in either 2 or 3 iterations.

Since P is stochastic with a single absorbing state, the largest eigenvalue is 1 and the corresponding left eigenvector is $[1\ 0\ 0\ 0]$. Eigenvalues and eigenvectors can be found from the following decomposition:

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -9.2426 & 5.2426 & 1 & 3 \\ -0.7574 & -3.2426 & 1 & 3 \\ -1 & -4 & 1 & 4 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0.8243 & 0 & 0 \\ 0 & 0 & 0.0243 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ -9.2426 & 5.2426 & 1 & 3 \\ -0.7574 & -3.2426 & 1 & 3 \\ -1 & -4 & 1 & 4 \end{bmatrix}^{-1}.$$

The relative weightings amongst the components of the second largest eigenvector of P corresponding to transient states are thus $\frac{1}{5.2426+1+3}[5.2426\ 1\ 3] = [0.5672\ 0.1082\ 0.3246]$, the Perron-Frobenius eigenvector of Q. This vector is the quasi-stationary vector of Q.

Let β_n list the probabilities of being in each transient state given that the process has not converged at iteration n. Thus in this case $\beta_n = \frac{1}{\delta_n(2)+\delta_n(3)+\delta_n(4)}[\delta_n(2)\ \delta_n(3)\ \delta_n(4)]$, where $\delta_n = \delta_0 P^n$. Then $\beta_0 = [0.1741\ 0.2847\ 0.5412]$, and β_n is expected to tend to the quasi-stationary vector over time [44, Theorem 4.6]. The process is now iterated four times, recording successive values of δ_n and β_n.

$$\delta_1 = [0.2990\ 0.4383\ 0.0657\ 0.1971], \quad \beta_1 = [0.6252\ 0.0937\ 0.2811]$$

$$\delta_2 = [0.4151\ 0.3308\ 0.0635\ 0.1906], \quad \beta_2 = [0.5655\ 0.1086\ 0.3258]$$

$$\delta_3 = [0.5180\ 0.2734\ 0.0521\ 0.1564], \quad \beta_3 = [0.5673\ 0.1082\ 0.3245]$$

$$\delta_4 = [0.6027\ 0.2253\ 0.0430\ 0.1289], \quad \beta_4 = [0.5672\ 0.1082\ 0.3246]$$

As expected, the weightings amongst all four states are tending towards the limiting distribution of P, $[1\ 0\ 0\ 0]$, and the weightings amongst the transient states are tending towards the Perron-Frobenius eigenvector of Q. After four iterations these weightings are already identical with those of the quasi-stationary vector, to four decimal places. (A further 48 iterations are required for the weightings amongst all four states to reach their limiting distribution.)

Hence the weightings among transient states, from which γ_n is calculated, quickly adopt their limiting values. Transition matrices for ARP, given by Equation (3.1) as $R_n = M^{\mathrm{T}}\mathrm{diag}(\gamma_n)PM$, are therefore equally quick to adopt limiting values. As discussed above, the difference between ARP and AARP thus tends to be very small. In particular, the expected convergence time for the algorithm in this example (equal to the expected convergence time for ARP) is 5.9 iterations; its AARP approximation (supposing for example that the second and third states share an objective function level) has an expected convergence time of 6.0 iterations.

3.3.3 Cyclic irreducible transient state transition matrices

This subsection completes the proof of the existence of an asymptotic averaged range process approximation to any Markovian domain process. The analysis is extended to the general case where the domain transient state transition matrix Q may be cyclic. Two examples are given for this cyclic case; in the second of these it is demonstrated that the definitions presented here provide a correction to those given in [49], which fail in certain cases.

Before the existence of the asymptotic averaged range process can be demonstrated, the following lemma is required. This lemma generalises the quasi-stationary vector result of Lemma 3.3.1 used in the primitive case.

Lemma 3.3.3 *Let the transient state transition matrix Q of a time-homogeneous Markovian domain process with a single absorbing state be irreducible, with each of $l-1$ transient states having period d. Then there are $d+1$ linearly independent eigenvectors (denoted $v_1, v_2, \dots, v_{d+1}$) corresponding to the largest two eigenvalues of P. There exists a unique decomposition of the initial probability vector δ_0,*

$$\delta_0 = a_1 v_1 + a_2 v_2 + a_3 v_3 + \ldots + a_{d+1} v_{d+1} + c,$$

such that (when $\delta_0(1) < 1$) the conditional probabilities of being in each transient state after a multiple of d iterations, given that the process has not converged, tend to limiting values given by the vector

$$v \quad = \quad \frac{(a_2 v_2 + a_3 v_3 + \ldots + a_{d+1} v_{d+1}) T_l}{\|(a_2 v_2 + a_3 v_3 + \ldots + a_{d+1} v_{d+1}) T_l\|_1}, \tag{3.6}$$

where $\|v\|_1$ is the L_1 norm of v and T_l is a truncation transformation equal to the identity matrix of size l with the first column removed.

Proof Arrange Q in canonical form [44] so that states in the same cyclic subclass are grouped together. The new matrix Q^d then represents the d-step transition matrix for the transient states. Whatever state the Markov chain is in at any stage, d steps later the Markov chain will again be in a state from the same cyclic subclass (unless it has reached the absorbing state), since Q is periodic with period d. Therefore Q^d is a zero matrix except for d submatrices on the diagonal, each corresponding to transitions between states within the same cyclic subclass of Q. These submatrices are primitive [44, Lemma 1.3].

As in the primitive case discussed earlier, each submatrix therefore has a quasi-stationary vector—that is, the relative weightings amongst the states within each cyclic subclass tend to a limit. For each of these quasi-stationary vectors, define a full state vector of l components by placing zeros in the components relating to states in the other cyclic subclasses. Then the relative weightings, according to which all transient states will tend to be distributed after successive application of Q^d, will be a weighted sum of these full state vectors. The weighted sum must take into account the probability of being in each of the cyclic subclasses after a multiple of d iterations, which depends on the initial probability distribution; within each cyclic subclass the transient states will tend to be distributed according to the quasi-stationary vector relating to that subclass.

The exact weightings are derived as follows. Since P is stochastic, the largest eigenvalue of P^d is $\lambda_1 = 1$; and (as in the primitive case) only the eigenvector v_1 placing all the weight in the absorbing state corresponds to this eigenvalue [44, Theorem 4.7]. There will be d eigenvectors of P^d corresponding to the second largest eigenvalue. These reflect the quasi-stationary vectors for each of the d cyclic subclasses; hence they are all real-valued. (Each eigenvector of P^d is the same as an eigenvector of Q^d with an extra entry relating to the absorbing state; the following analysis relates really to the eigenvectors of Q^d but is presented in terms of the full matrix P^d.)

It is necessary to refer to the Jordan form of P^d, defined as XJX^{-1} for some X and J [12]. Now let X_i denote the matrix formed by the columns of X associated with λ_i and $X_i^{\#}$ the matrix formed by the rows of X^{-1} associated with λ_i. Then $(P^d)^n$ can be expressed using Theorem 3.3.1 as

$$\sum_{i=1}^{g} \left(\lambda_i^n P_i + \sum_{k=1}^{\min(l_i-1,n)} \left(\begin{pmatrix} n \\ k \end{pmatrix} \lambda_i^{n-k} D_i^k \right) \right)$$

where the number of distinct eigenvalues of P^d is g, $P_i = X_i X_i^\#$, $D_i = X_i N_i X_i^\#$ for some N_i and $D_i^{l_i} = 0$.

The structure of J is detailed in [12]. The eigenvalues of P^d form the main diagonal of J. These can, by reordering indices in X and J, be taken to occur in the order $\lambda_1, \lambda_2, \ldots \lambda_g$. For every linearly independent eigenvector associated with λ_i there is a column in J made up entirely of zeros except for the diagonal element, which has the value λ_i; all other columns in J are made up of zeros except for the diagonal element, containing one of the eigenvalues, and the superdiagonal element, which has the value 1. A further condition ensures that the superdiagonal in the first column of J containing a particular eigenvalue λ_i is always 0.

This structure implies that the ith column of X is a right eigenvector associated with the eigenvalue in the ith column of J if and only if the superdiagonal in the ith column of J is zero, since $P^d X = XJ$ (and X is invertible, so each column contains nonzero elements). Moreover, this relation also implies that if the superdiagonal in the ith column of J (containing the eigenvalue λ_j) is 1 then the ith column of X, denoted by x^i, is a "generalised eigenvector" of P^d, satisfying $P^d x^i = \lambda_j x^i + x^{i-1}$, where x^{i-1} is itself either an eigenvector or a generalised eigenvector of P^d corresponding to λ_j. (Thus the Jordan basis [19] is composed solely of eigenvectors and these "generalised eigenvectors".) The further condition given above limits the number of columns with superdiagonals equal to 1 and ensures that there is at least one eigenvector associated with each eigenvalue. A similar argument can be used to show that rows of X^{-1} are left eigenvectors or generalised eigenvectors of P^d.

It is now shown that the number of linearly independent eigenvectors corresponding to λ_1 and λ_2, already known to be $d+1$, is equal to the number of times they appear in J. The first row of X^{-1}, denoted by $x^{1\#}$, satisfies $x^{1\#}P = x^{1\#}$ (since $\lambda_1 = 1$) and must therefore be the unique stationary distribution. Then if λ_1 is repeated in J there must be a generalised eigenvector $x^{2\#}$ satisfying $x^{2\#}P^d = x^{2\#} + x^{1\#}$; but this is impossible since $x^{2\#}$ can be scaled so that it is stochastic, whence the left hand side of this equation is stochastic but the right hand side is not (since $x^{1\#}$ is also stochastic). Thus λ_1 occurs only once in J. Also the eigenvectors associated with λ_2 are formed from Perron-Frobenius eigenvectors of each of the d cyclic subclasses of P^d, as discussed above; there are therefore no other eigenvectors or generalised eigenvectors associated with λ_2 in any of these subclasses [44]. Thus the columns of X associated with λ_1 and λ_2 are all eigenvectors, not merely generalised eigenvectors.

Now since X^{-1} is invertible and its rows are eigenvectors or generalised eigenvectors of P^d, there must exist a unique representation of δ_0 as a sum of eigenvectors and generalised eigenvectors:

$$\delta_0 = a_1 v_1 + a_2 v_2 + a_3 v_3 + \ldots + a_{d+1} v_{d+1} + c.$$

Multiplying by P^{nd}, remembering that the v_i are eigenvectors of P^d, and that d of them have the same eigenvalue, gives

$$\delta_{nd} = a_1 \lambda_1^n v_1 + a_2 \lambda_2^n v_2 + a_3 \lambda_2^n v_3 + \ldots + a_{d+1} \lambda_2^n v_{d+1} + c(P^d)^n. \tag{3.7}$$

The final term will tend to die out faster than the other terms as n tends to infinity; the proof of this again uses the Jordan form of P^d.

Since left eigenvectors and generalised eigenvectors are rows of X^{-1} and $X^{-1}X = I_l$, it follows that $x^{i\#} X_j = 0$ whenever $x^{i\#}$ is associated with an eigenvalue other than λ_j. Referring to Equation (3.7), c is a sum of eigenvectors and generalised eigenvectors associated with eigenvalues other than λ_1 and λ_2, so it is also true that $cX_j = 0$ for $j \in \{1, 2\}$. From this it follows that $cP_1 = 0$, $cP_2 = 0$, $cD_1 = 0$ and $cD_2 = 0$; therefore

$$c(P^d)^n = c \sum_{i=3}^{g} \left(\lambda_i^n \left(P_i + \sum_{k=1}^{\min(l_i-1,n)} \left(\begin{pmatrix} n \\ k \end{pmatrix} \lambda_i^{-k} D_i^k \right) \right) \right)$$

and $\lim_{n\to\infty} c(P^d)^n / \lambda_2^n = 0$ since $|\lambda_i| < \lambda_2$ for all $i > 2$.

Thus

$$\lim_{n\to\infty} \frac{\delta_{nd}}{\lambda_2^n} = a_1 \left(\frac{\lambda_1}{\lambda_2}\right)^n v_1 + a_2 v_2 + a_3 v_3 + \ldots + a_{d+1} v_{d+1}.$$

The first term will tend to overshadow the others, but this term affects only the weighting of probability in the absorbing state (due to the special form of v_1). The transient states will assume the weightings relative to each other described by the following d terms, once the remaining term dies out (which, as shown above, will happen geometrically quickly). These relative weightings, denoted v, are the L_1-normalisation of the transient component of

$$a_2 v_2 + a_3 v_3 + \ldots + a_{d+1} v_{d+1}$$

(that is, scale v so that it is a probability vector. Normalisation is possible since the algorithms that AARP approximates are finite). This completes the proof. ■

Note that the long term behaviour of the cyclic algorithm depends on the constants a_i, which are derived from the initial distribution; contrast the primitive case dealt with earlier, where there is no such dependency.

Thus after a multiple of d iterations, the transient states will tend to distribute themselves according to v. This is the cyclic generalisation of the quasi-stationary vector referred to in the primitive case. Since every power of Q can be expressed as Q^{kd+a} for some integer k, where $a \in \{0, 1, \ldots, d-1\}$, the vector of probabilities of being in each transient state given that the process has not converged will tend to cycle through the vectors $v, vQ/\|vQ\|_1, vQ^2/\|vQ^2\|_1 \ldots, vQ^{d-1}/\|vQ^{d-1}\|_1$.

This provides a general method of constructing R, the transition matrix amongst objective function levels in the range of the asymptotic averaged range process approximation to any time-homogeneous Markovian domain process.

Theorem 3.3.3 *A general equation to give the transition matrix of an asymptotic averaged range process approximation to any time-homogeneous Markovian domain process with irreducible Q and $P(X_0 \neq x_1) > 0$ is*

$$R \;=\; M^T diag(\gamma) P M \tag{3.8}$$

where γ is found directly from the limiting average weightings amongst transient states as

$$\gamma(i) = \lim_{k\to\infty} \frac{\sum_{a=0}^{d-1} \gamma_{kd+a}(i)}{\sum_{a=0}^{d-1} \sum_{\{j:f(x_j)=f(x_i)\}} \gamma_{kd+a}(j)}$$

for $i \in \{1, 2, 3, \ldots, l\}$.

Proof Since P is constant, the limit R_n as defined in Equation (3.1) will tend to R as defined in Equation (3.8) provided γ_n tends to γ as n tends to infinity.

Now since Lemma 3.3.3 shows that the process tends eventually to repeat a cycle of d steps, the limiting average of γ_{kd+a} is given simply by the average of the limits over one cycle. The average must be taken not simply over the d iterations in each cycle, but over the number of iterations in each cycle where $P(Y_n = f(x_i)) > 0$. This is given by the double summation in the denominator (note that $\sum_{a=0}^{d-1} \sum_{\{j:f(x_j)=f(x_i)\}} \gamma_{kd+a}(j) = \|\{a : P(Y_{kd+a} = f(x_i)) > 0\}\|$). ■

Note that Lemma 3.3.3 also provides a refined method of calculating $\lim_{k\to\infty} \gamma_{kd+a}$; that is, $\gamma_{kd+a}(1) = 1$ since there is only one absorbing state, and

$$\lim_{k\to\infty} \gamma_{kd+a}(i) = \begin{cases} \dfrac{[vQ^a](i)}{\sum_{\{j:f(x_j)=f(x_i)\}} [vQ^a](j)} & \text{where this is defined} \\ 0 & \text{otherwise,} \end{cases}$$

where $i \in \{2, 3, \ldots, l\}$. (Note that vQ^a need not be normalised in the preceding equation, since only the relative weightings amongst states are required.)

By this procedure R may be derived directly, with no call on infinite sums. (Long run characteristics of δ_n could still be used to find v, if this is more computationally efficient.)

Note that in general γ (and thus AARP) is dependent on the initial distribution. This is not the case for primitive Q. (Nor is it the case for cyclic processes where states at the same level are in the same cyclic subclasses. This is because $P(X_n = x_i | Y_n = f(x_i))$ depends in this case only on relative weightings within each cyclic subclass, there being no distribution of weight across subclasses since the conditioning event $Y_n = f(x_i)$ always fixes the cyclic subclass exactly when all states at the same level share a cyclic subclass.)

An example illustrates the ideas involved in constructing the asymptotic averaged range process approximation of a cyclic domain process.

Example 2

Take an algorithm with domain transition matrix

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.1 & 0 & 0.1 & 0.8 \\ 0.2 & 0.8 & 0 & 0 \\ 0.3 & 0.7 & 0 & 0 \end{bmatrix}$$

and the randomly generated initial probability distribution $\delta_0 = [0.0816\ 0.1599\ 0.2615\ 0.497]$ from Example 1. Evidently, this matrix is cyclic; transient states can return to themselves only after an even number of iterations. The cyclic subclasses are { state 2 } and { states 3 and 4 }.

Let

$$M = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

with the rows of M corresponding to states in the domain and the columns corresponding to objective function levels in the range. Where $M_{ij} = 1$ this implies that the ith state is at the jth objective function level. Thus state 4 is the only state at the third level.

As in the preceding discussion, focus is directed on the process with transition matrix

$$P^2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.36 & 0.64 & 0 & 0 \\ 0.28 & 0 & 0.08 & 0.64 \\ 0.37 & 0 & 0.07 & 0.56 \end{bmatrix},$$

which shows the separate subclasses amongst the transient states.

The limiting distribution of this process again places all the weight in the absorbing state. There are two left eigenvectors corresponding to the second largest eigenvalue, namely $[-1\ 1\ 0\ 0]$ and $[-9\ 0\ 1\ 8]$. The first of these is connected to the cyclic subclass corresponding to the second state in P. The second of these eigenvectors shows that the process will tend, on average, to spend eight times as long in the fourth state as in the third. These states constitute the second cyclic subclass. Projecting δ_0 onto the eigenvectors gives

$$\delta_0 \quad = \quad 0.083[1\ 0\ 0\ 0] + 0.160[-1\ 1\ 0\ 0] + 0.087[-9\ 0\ 1\ 8] + 0.025[1\ 0\ 7\ -8]$$

so the vector of limiting relative weightings of transient states under P^2 is the L_1-normalisation of $0.160[1\ 0\ 0] + 0.087[0\ 1\ 8]$ (since these are the transient components of the eigenvectors corresponding to the second largest eigenvalue of P^2), or $v = [0.1695\ 0.0923\ 0.7382]$. Every second iteration the process will tend to distribute itself among transient states according to this vector; on alternate iterations the limiting weightings are $vQ/\|vQ\|_1 = [0.7947\ 0.0228\ 0.1825]$.

The process is now iterated, recording successive values of δ_n and β_n.

$$\delta_0 = [0.0816\ 0.1599\ 0.2615\ 0.4970], \quad \beta_0 = [0.1741\ 0.2847\ 0.5412]$$

$$\delta_1 = [0.2990\ 0.5571\ 0.0160\ 0.1279], \quad \beta_1 = [0.7947\ 0.0228\ 0.1825]$$

$$\delta_2 = [0.3963\ 0.1023\ 0.0557\ 0.4457], \quad \beta_2 = [0.1695\ 0.0923\ 0.7382]$$

$$\delta_3 = [0.5514\ 0.3565\ 0.0102\ 0.0819], \quad \beta_3 = [0.7947\ 0.0228\ 0.1825]$$

As expected, the weightings amongst all four states are tending towards the limiting distribution of P, $[1\ 0\ 0\ 0]$. At every second iteration $\beta_n = v$, and at odd iterations $\beta_n = vQ/\|vQ\|_1$, as expected. (In this case the next largest eigenvalue of P^2 is 0 (as is obvious from the transient state transition matrix), so β_n adopts the limiting values after a single iteration. This does not happen in general.)

Using the limiting weightings v and $vQ/\|vQ\|_1$, found above,

$$\begin{aligned}\lim_{k\to\infty} \gamma_{kd} &= \left[\begin{array}{cccc} 1 & \frac{0.1695}{0.1695+0.0923} & \frac{0.0923}{0.1695+0.0923} & \frac{0.7382}{0.7382}\end{array}\right] \\ &= [1\ 0.6475\ 0.3525\ 1]\end{aligned}$$

and similarly

$$\lim_{k\to\infty} \gamma_{kd+1} = [1\ 0.9721\ 0.0279\ 1]\,.$$

Theorem 3.3.3 then gives

$$\begin{aligned}\gamma &= \left[\begin{array}{cccc} \frac{1+1}{1+1} & \frac{0.6475+0.9721}{0.6475+0.3525+0.9721+0.0279} & \frac{0.3525+0.0279}{0.6475+0.3525+0.9721+0.0279} & \frac{1+1}{1+1}\end{array}\right] \\ &= [1\ 0.8098\ 0.1902\ 1]\,,\end{aligned}$$

whence

$$\begin{aligned}R &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0.8098 & 0 & 0 \\ 0 & 0 & 0.1902 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.1 & 0 & 0.1 & 0.8 \\ 0.2 & 0.8 & 0 & 0 \\ 0.3 & 0.7 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0.1190 & 0.2331 & 0.6478 \\ 0.3 & 0.7 & 0 \end{bmatrix}.\end{aligned}$$

This transition matrix, together with the initial probability distribution in the range given by $\delta_0 M$, defines the asymptotic averaged range process approximation to the cyclic domain process.

In the limiting average derivation of R provided in [49], however, the asymptotic averaged range process may not be correctly defined. There the equation

$$\begin{aligned} R &= \lim_{N\to\infty} \frac{1}{N} \sum_{n=0}^{N-1} R_n \\ &= \lim_{N\to\infty} \frac{1}{N} \sum_{n=0}^{N-1} M^{\mathrm{T}} \mathrm{diag}(\gamma_n) P M \\ &= M^{\mathrm{T}} \mathrm{diag}(\lim_{N\to\infty} \frac{1}{N} \sum_{n=0}^{N-1} \gamma_n) P M \end{aligned}$$

is given. This formulation is appropriate for algorithms with primitive Q, since then Theorem 3.3.2 shows that $\lim_{n\to\infty} R_n = R$, but when Q is cyclic the transition matrix generated in this way may be substochastic.

The definition of γ_n shown in Subsection 2.2.1 provides for the case where $P(Y_n = f(x_k)) = 0$ by arbitrarily setting $\gamma_n(k)$ to 0. While these zero entries made no difference to the running of ARP (the marginal range distributions of ARP are identical with those of the range process, so that ARP never requires information concerning what to do from level $f(x_k)$ at iteration n unless $P(Y_n = f(x_k)) > 0$), it will make a difference to AARP. If in the limit $P(Y_n = f(x_k)) = 0$ once or more in each cycle of d iterations, then at least $1/d$ of the R_n will have a row of zeros. The formula for R will then include these rows, producing averaged rows that do not sum to 1; but R should be stochastic. The correct definition of R given in Theorem 3.3.3 forms γ_n by averaging only over the number of iterations in each cycle where $P(Y_n = f(x_i)) > 0$.

This situation is demonstrated in the following example.

Example 3

Consider the domain transition matrix

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.1 & 0 & 0.1 & 0.8 \\ 0.2 & 0.8 & 0 & 0 \\ 0.3 & 0.7 & 0 & 0 \end{bmatrix}$$

from Example 2 and initial probability distribution $\delta_0 = [0.3\ 0.7\ 0\ 0]$. Thus, with probability 0.3 the algorithm samples the optimum immediately; otherwise it begins in the second state, whence it will transition either to the optimum or to either the third or fourth state. From the initial distribution, it is clear that the process will never be in state 2 on odd iterations, or in states 3 or 4 on even iterations. As before,

$$M = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

so state 4 is the only state at the third level, and the process will never be at the third level at even iterations. Equation (2.3) therefore arbitrarily sets $\gamma_n(4)$ at even iterations to zero. This is the situation when rows of zeros appear in R_n. Thus

$$R_0 = \begin{bmatrix} 1 & 0 & 0 \\ 0.1 & 0.1 & 0.8 \\ 0 & 0 & 0 \end{bmatrix}, \quad R_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0.2 & 0.8 & 0 \\ 0.3 & 0.7 & 0 \end{bmatrix}$$

and each alternating R_n also has a zero row. (In fact, for this example $R_n = R_0$ at even iterations and $R_n = R_1$ at odd iterations.) Consequently, the limiting average R found using the formula for primitive Q given in [49] has a row that does not sum to 1:

$$R = \begin{bmatrix} 1 & 0 & 0 \\ 0.15 & 0.45 & 0.4 \\ 0.15 & 0.35 & 0 \end{bmatrix}$$

It would be sensible to ignore rows of zeros in R_n when taking the limiting average, since they are arbitrary. This is effectively the result of applying the revised definition of R given in Theorem 3.3.3, in the manner illustrated in Example 2 .

Application of Theorem 3.3.3 provides the proper AARP transition matrix,

$$R = \begin{bmatrix} 1 & 0 & 0 \\ 0.15 & 0.45 & 0.4 \\ 0.3 & 0.7 & 0 \end{bmatrix}.$$

Note that this is the matrix that would be obtained by averaging the R_n matrices ignoring zero rows. Each row of R is the limiting average of transition probabilities from that level *on the iterations at which the process has positive probability of transitioning from that level.* This is clearly the logical extension of AARP to cyclic processes.

Finally, it may be observed that the same algorithm used in Example 2 with a different initial vector produces a different asymptotic averaged range process approximation. This dependency on the initial probability distribution reflects the fact that the algorithm is periodic.

The construction of AARP described in [49] fails on this example because AARP requires a single matrix to direct what to do at each range level at each iteration, even though the domain process provides no data for some levels at some iterations. The construction of R in Theorem 3.3.3 defines the AARP approximation to this example by effectively interpolating local range distributions for states at each iteration even though the domain process has zero probability of reaching those states on each alternating iteration. A reasonable approach to overcoming the special challenges presented by cyclic examples is thus presented. This analysis extends the definition of AARP from that given in [49], to apply over all domain processes with irreducible transient state transition matrices.

An alternative solution to the problem illustrated above is to start the process with a positive probability of being in each domain state. There is therefore also a positive probability of being in the absorbing state at any subsequent iteration, and it can be shown by induction that the distribution at each iteration also assigns positive probability to each transient state. Suppose that the process has a positive probability of being in each transient state at iteration n. (This is true when $n = 0$ since the process starts with a positive probability of being in each domain state.) At iteration $n+1$ the process has probability $\sum_i P(X_n = x_i)p_{ij}$ of being in each other transient state j. This probability is positive provided Q is irreducible, using the above supposition. Thus by induction there is positive probability of being in each state at all iterations $n \in \{0, 1, 2, \ldots\}$, and thus $P(Y_n = f(x_k)) > 0$ for all n. In this case R_n will never have a zero row. Hence the limiting average definition of AARP given in [49] applies to domain processes where Q is irreducible and the initial probability distribution is positive.

The latter condition could be imposed by commencing with pure random search with some positive probability. (This condition is not required for primitive Q, or for the definition of R given in Theorem 3.3.3.) The irreducibility condition required for either definition of R means that the algorithm never completely writes off any part of the domain, which is a sensible assumption for good algorithms.

Using the general definition of the asymptotic averaged range process given in Theorem 3.3.3, a Markov homogeneous process in the range can thus be defined as an approximation of any time-homogeneous Markovian optimisation algorithm. The accuracy of this approximation in estimating the expected number of iterations before convergence will determine its usefulness; this accuracy is now considered in the following section.

3.4 Convergence time of AARP

As shown in Section 3.2, the averaged range process approximation to a general stochastic global optimisation algorithm preserves the distribution of the number of iterations to convergence. The asymptotic averaged range process does not; yet the expected number of iterations until convergence for this process seems empirically to be close to the expected number of iterations until convergence for the domain, range and averaged range processes. The example in Subsection 2.2.3 illustrates this tendency.

In this section it is shown that the error in expected convergence time introduced by the asymptotic averaged range process approximation is in general unbounded. However, it can be shown that the difference in convergence times of the asymptotic averaged range and domain processes is given by a summation where the terms tend to decay geometrically, in the case where the transient domain state transition matrix is primitive. In many cases, therefore, the difference in expected convergence times between the domain and asymptotic averaged range processes will be small.

Denote the convergence time of the asymptotic averaged range process by N_b, and the marginal distribution in the range at the nth iteration by τ_n. Now, employing a result similar to one in [4] (and recalling that T_m is equal to the identity matrix of size m with the first column removed),

$$\begin{aligned} \mathrm{E}(N_a) &= (\pi_0 + \pi_1 + \pi_2 + \cdots)T_m \mathbf{1}_{m-1} \\ &= 1 - \pi_0(1) + 1 - \pi_1(1) + 1 - \pi_2(1) + \cdots \end{aligned}$$

since π_n is stochastic. Similarly,

$$\mathrm{E}(N_b) = 1 - \tau_0(1) + 1 - \tau_1(1) + 1 - \tau_2(1) + \cdots$$

and the error introduced by approximating the averaged range process with the asymptotic averaged range process is given by

$$\mathrm{E}(N_b) - \mathrm{E}(N_a) = \pi_1(1) - \tau_1(1) + \pi_2(1) - \tau_2(1) + \cdots \tag{3.9}$$

(since the initial distributions for the averaged range and asymptotic averaged range processes are identical).

Equation (3.7) shows that all contributions to δ_{nd} except that of v_1 die out geometrically with rate $\lambda_2/\lambda_1 = \lambda_2$. The following theorem establishes that when $d = 1$, it is also true that all contributions to τ_n die out geometrically with the same rate (since the eigenvector corresponding to the largest eigenvalue of the asymptotic averaged range process transition matrix R places all the weight in the absorbing state and the second largest eigenvalue of R can in fact be shown to be λ_2). Thus the differences in Equation (3.9) tend to decay towards zero geometrically. If λ_2 is not close to one then the decay will be rapid and the total error in expected convergence time may be small.

Theorem 3.4.1 *When the transition matrix amongst transient domain states is primitive, the second largest eigenvalue of the domain process transition matrix P is the same as the second largest eigenvalue of the asymptotic averaged range process transition matrix R.*

Proof The proof begins with Equation (3.7). For any eigenvector v_i of P, it must be true that $v_iP = \lambda_i v_i$. Therefore

$$\begin{aligned} \lambda_i v_i \mathbf{1}_l &= v_i P \mathbf{1}_l \\ &= v_i \mathbf{1}_l \end{aligned}$$

since P is stochastic. Therefore $v_i\mathbf{1}_l = 0$ wherever $\lambda_i \neq 1$. This implies that, since v is defined in Lemma 3.3.3 as the last $l-1$ components of v_2 (where $d = 1$), and $\lambda_2 < 1$, the full l-vector v_2 must be the vector $[-1\ v]$ (since v is stochastic). It will now be shown that $[-1\ v]M$ is a stationary vector of R.

When $d = 1$, the definition of γ in Theorem 3.3.3 and the definition of $\lim\limits_{k\to\infty} \gamma_{kd+i}$ beneath it combine to show that $\gamma(1) = 1$ and

$$\gamma(i) = \frac{v(i)}{\sum\limits_{\{j: f(x_j) = f(x_i)\}} v(j)}$$

for $i \in \{2, 3, \ldots, l\}$. (Since P is irreducible, v must be positive so the denominator is always positive.) Comparing this with Equation (2.3), it can be seen that if $\delta_k = [0\ v]$ then $\gamma(i) = \gamma_k(i)$ for $i \in \{2, 3, \ldots, l\}$.

Equation (3.2) then gives $[\delta_k M M^{\mathrm{T}} \mathrm{diag}(\gamma)](i) = \delta_k(i)$ for $i \in \{2, 3, \ldots, l\}$. Moreover, $\gamma(1) = 1$ and all components of the first row and column of M are zero except $M_{11} = 1$; therefore it is also true that $[-1\ v]MM^{\mathrm{T}}\mathrm{diag}(\gamma) = [-1\ v]$. Thus, expanding R using Theorem 3.3.3,

$$\begin{aligned} [-1\ v]MR &= [-1\ v]MM^{\mathrm{T}}\mathrm{diag}(\gamma)PM \\ &= [-1\ v]PM \\ &= \lambda_2[-1\ v]M. \end{aligned}$$

It thus follows that $[-1\ v]M$ is an eigenvector of R corresponding to λ_2.

This vector is nonnegative, except for the first component. As the transient part of P is primitive, there is a transient state that can transition to itself in either n or $n+1$ iterations, for some n. Thus the transient part of R is also primitive. Since the stationary distribution of the submatrix of R pertaining to transient states is unique, therefore, it is given by the part of $[-1\ v]M$ corresponding to transient range levels; hence λ_2 is the largest eigenvalue of that submatrix. The second largest eigenvalue of R is thus λ_2; as in Equation (3.7), therefore, τ_{nd} tends to the limiting vector (which, since R is absorbing, places all the weight in the initial level) geometrically with rate λ_2. ■

Both $\delta_n(1)$ and $\tau_n(1)$ thus tend to one geometrically with rate λ_2. Initially the difference between them is biased towards zero by the contributions of other eigenvectors (since the initial error is exactly zero); as other eigenvectors die out the error will adopt a geometric decay with rate λ_2.

The proof applies only for domain processes where the transient state transition matrix is primitive. Otherwise P has d eigenvalues equal to the d complex roots of λ_2, so $[-1\ v]P \neq \lambda_2[-1\ v]$. It is possible to ensure that the transient state transition matrix is acyclic by altering the algorithm to conduct pure random search from some level with positive probability; with this slight alteration, the result of Theorem 3.4.1 is applicable.

In random matrices the difference between the largest and second largest eigenvalues may often be large enough that this rate of decay will be fast, and ARP will have only a few iterations before settling down to the limiting behaviour of AARP. In this case the convergence times of the two processes will be similar.

It is possible, however, to construct transition matrices with the difference between magnitudes of these eigenvalues set arbitrarily small. In this case ARP will differ significantly from AARP for a large number of iterations; the convergence times may then be very different. In principle, therefore, the error introduced in approximating the averaged range process with the asymptotic averaged range process is unbounded. In practice, the approximation tends to be quite close for many algorithms and problems. Experimental results are reported in Chapter 6.

As an example, take an algorithm with domain transition matrix

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.001 & 0.99 & 0.002 & 0.007 \\ 0.003 & 0.004 & 0.99 & 0.003 \\ 0.5 & 0.1 & 0.2 & 0.2 \end{bmatrix}$$

and the random initial probability distribution $\delta_0 = [0.0816\ 0.1599\ 0.2615\ 0.497]$ from Example 1.

The eigenvalues of this (acyclic) transition matrix are 1, 0.995, 0.987 and 0.198. The asymptotic averaged range process approximation can be constructed using the same mapping matrix M as in Example 2 . The error terms $\pi_n(1) - \tau_n(1)$ are recorded at each iteration and plotted in Figure 3.1.

The sum of the error at each iteration in Figure 3.1 gives the total error in expected convergence time introduced by the asymptotic averaged range process approximation to the domain process. In this case the total error is only 0.7 iterations; the expected number of iterations before convergence for the domain process is 121.4 and that of the asymptotic averaged range process is 120.7. Thus even though the behaviour of the averaged range process takes some time to settle down to that of the asymptotic averaged range process, the overall error is still small. If the domain process had its largest eigenvalues even closer to one, however, then the convergence would be slower and total error would increase.

Note that Figure 3.1 illustrates a reason why the error is small: the errors are initially biased towards zero. Indeed, as mentioned above, there is initially no error at all, since the asymptotic averaged range process and the averaged range process start with identical probability distributions.

Despite the worst case analysis revealing unbounded error in the approximation of expected conver-

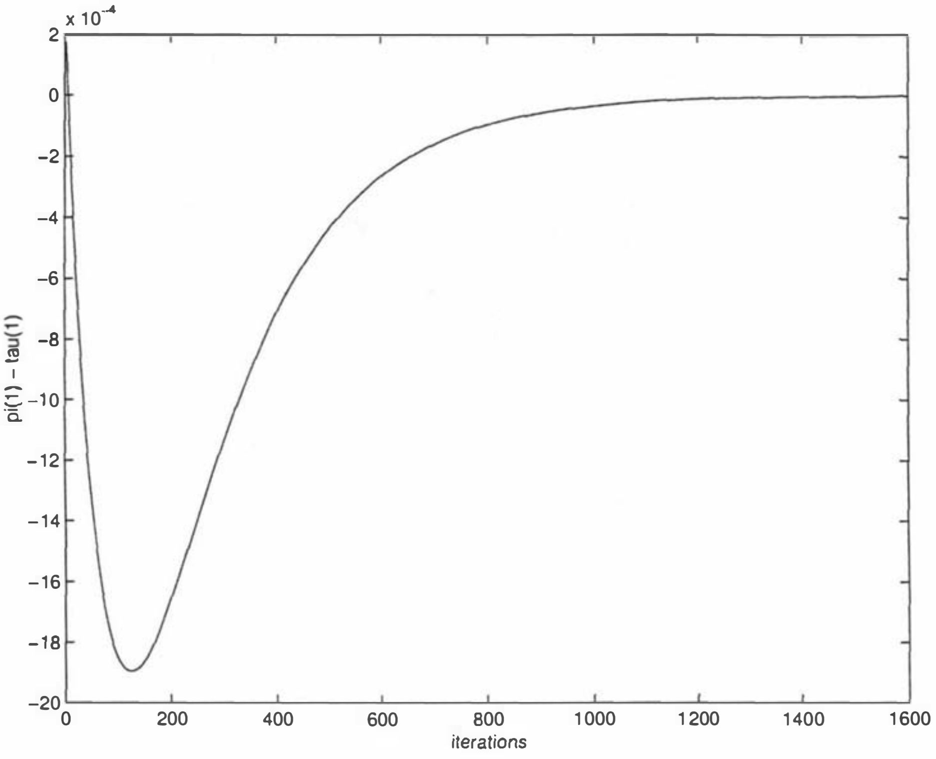

Figure 3.1: Time series plot of the error $\pi_n(1) - \tau_n(1)$ at each iteration in the example, illustrating geometric decay.

gence times using the asymptotic averaged range process, it can be anticipated that many algorithms applied to various problems will give rise to domain processes with eigenvalues far enough from one for the approximation to be very close. If it were possible to check this property of the algorithm, a small value of the second largest eigenvalue could provide an assurance that the averaged range process transition matrices converge quickly to their limit and thus that the error introduced by the asymptotic averaged range process is small. The process in the domain is thus very closely linked with the time-homogeneous Markov process in the range, AARP. If the backtracking adaptive search approximation to the asymptotic averaged range process is accurate, then it will be possible to find a good estimate of the expected number of iterations before convergence for the original algorithm in the domain.

3.5 Summary

This chapter fills out the theoretical background to the approximation framework outlined in Chapter 2. Firstly, the averaged range and asymptotic averaged range processes have been shown to be well-defined approximations to any time-homogeneous Markov domain process. The possibility of weakening this restriction on the domain process has been remarked.

Secondly, the number of iterations to convergence for the averaged range process has been shown to be identical in distribution to that of the domain process, and reasons have been advanced to suggest that the asymptotic averaged range process may have an expected number of iterations to convergence that very closely approximates the value for the domain and averaged range processes.

It remains therefore to complete the approximation process by finding a backtracking adaptive search approximation to the asymptotic averaged range process; this is the theme of the next chapter.

Chapter 4

Backtracking adaptive search

4.1 Introduction

Attention is now turned to the end-point of the approximation framework: backtracking adaptive search. This chapter provides a definition of backtracking adaptive search and, in particular, analysis is provided for this process as defined on a finite range. Then a method of approximating the asymptotic averaged range process with backtracking adaptive search is described.

In particular, the following section contains a description and definition of backtracking adaptive search. Section 4.3 then provides analysis of the number of iterations before convergence for backtracking adaptive search on a finite range, and Section 4.4 extends this analysis to allow an arbitrary initial distribution; this is of particular interest in the context of the framework of approximations. In both cases, a method is shown whereby the expected number of iterations before convergence for backtracking adaptive search can be found quickly via computer. Section 4.5 provides examples of backtracking adaptive search algorithms and shows how this analysis generalises several algorithms for which analysis has already been published. Finally, Section 4.6 details the method by which the asymptotic averaged range process is approximated by backtracking adaptive search. This completes the approximation framework; it is then possible to approximate the progress on the range of any algorithm with a backtracking adaptive search process and estimate the number of iterations required to reach a solution with a specified objective function level.

The entire moment generating function of the number of iterations before convergence for backtracking adaptive search on mixed domains is left to Chapter 5; this chapter seeks only to introduce enough theory to provide an end-point to the approximation framework.

4.2 Defining backtracking adaptive search

In this section backtracking adaptive search is described and the relevant notation is introduced. The basic concepts of backtracking adaptive search are discussed in the context of other similar theoretical optimisation algorithms.

The pure adaptive search algorithm has been defined and analysed in [54, 56]. Pure adaptive search is a theoretical algorithm for stochastic global optimisation in which successive iterates are generated from the improving region in the domain. This algorithm is an ideal, which is currently prohibitively difficult to implement in general. A recent encouraging advance towards implementing pure adaptive search is given in [39].

In general, algorithms will sometimes "hesitate" at the current point in the domain for a number of iterations before finding an improvement, or even "backtrack" by accepting new iterates with worsening objective function values. These generalisations are incorporated in hesitant adaptive search (HAS) and backtracking adaptive search (BAS), respectively. The full distribution of the number of iterations until convergence is given for HAS in [51].

In this chapter HAS is extended to BAS, and a closed form expression for the expected number of iterations before convergence for BAS on a finite domain is presented. This process is used in the expectation that BAS will provide a sufficiently flexible family of homogeneous Markov range processes for approximating the asymptotic averaged range distributions of stochastic global optimisation algorithms. In [31] a more general variant of BAS on finite domains is discussed, and bounds for expected search duration are presented. A closed form expression for the expected search duration of a special case studied in [31] is presented in Section 4.5.

The central idea of backtracking adaptive search is as follows. At each iteration, the objective function value either improves, remains at the current level, or worsens. It is assumed that the distribution of the next objective function value is then a normalised restriction of a single distribution on the range, constant with respect to time. This is acknowledged to be restrictive.

The definition of backtracking adaptive search now given applies only to problems with finitely many range levels. Only finite backtracking adaptive search is analysed in this chapter; analysis for the remaining cases is provided in Chapter 5.

The finite number of range levels in $f(S)$ are (without loss of generality) labelled $1, 2, \ldots, m$. Initially BAS samples Y_0 from this set according to the range probability measure $P(Y_0 = y) = \pi_y$, with masses assumed to be strictly positive for all $y \in \{1, 2, \ldots, m\}$. (For simplicity, the notation (Y_n) for backtracking adaptive search is used in this chapter and the following, instead of the notation $(Y_n^{\cdots})$ used in Figure 2.1 to distinguish backtracking adaptive search from the other processes in that figure.) Define $p_y = \sum_{i=1}^{y} \pi_i$. (Note that the symbol π_n is used in Chapter 3 to represent a vector of probabilities relating to the averaged range process; the use of π_y in this chapter is unrelated to the meaning applied there.) At each iteration thereafter, one of three things happens. With a known probability b_{Y_n}, the algorithm will make an improvement, sampling the next evaluation point Y_{n+1} according to the normalised restriction of π to the current improving set. With a second known probability w_{Y_n}, the algorithm will backtrack, sampling the next evaluation point Y_{n+1} according to the normalised restriction of π to the current worsening set. Otherwise, the algorithm will hesitate, remaining at the current evaluation point. The functions b and w depend only on the current level. Define $b_1 = w_m = 0$, and impose the condition $b_y + w_y > 0$ for all $y \in \{2, 3, \ldots, m\}$ to ensure that the algorithm can attain any objective function level in finite time. The algorithm is now presented formally.

Backtracking adaptive search

Step 1 Generate Y_0 in $f(S)$ according to π. Set $n = 0$.

Step 2 With probability b_{Y_n} choose the next iterate Y_{n+1} according to the normalised restriction of π to $\{1, 2, 3, \ldots, Y_n - 1\}$. With probability w_{Y_n} choose Y_{n+1} according to the normalised restriction of π to $\{Y_n + 1, Y_n + 2, Y_n + 3, \ldots, m\}$. Otherwise set $Y_{n+1} = Y_n$.

Step 3 If a stopping criterion is met, stop. Otherwise, increment n and return to Step 2.

Note that HAS is the special instance of BAS occurring when $w_y = 0$ for all $y \in f(S)$. In turn, pure adaptive search (PAS) is the special instance of HAS occurring when $b_y = 1$ for all $y \in f(S)$. Thus PAS must improve at every iterate.

The increasingly general families of homogeneous first order Markov chains provided by PAS, HAS and BAS can serve to approximate the asymptotic averaged range process of a stochastic optimisation algorithm.

4.3 Expected search duration

In this section a difference equation for the expected number of iterations of finite backtracking adaptive search before convergence is derived and then solved, as in [1].

Define $N(y) = \min\{n : Y_n \leq y\}$ for all $y \in \{1, 2, \ldots, m\}$. Since the initial objective function value is denoted Y_0, $N(y)$ is the number of iterations before reaching level y.

There is a well-known procedure for computing the expected search duration of a Markov process such as BAS, generally requiring matrix inversion (see for instance [27]). The theorem and corollary presented in this section take advantage of the structure of BAS, however, to allow this quantity to be evaluated without any call for inverting a matrix. Expected search durations can thus be found quickly by computer for problems of a size that would ordinarily prohibit such calculation.

Theorem 4.3.1 *The expected number of iterations of finite backtracking adaptive search before reaching a certain level y, $E[N(y)]$, satisfies*

i) For all m, $E[N(m)] = 0$.

ii) For $m > 1$, $E[N(m-1)] = \pi_m / b_m$.

iii) For $m > 2$,

$$E[N(y-1)] - E[N(y)] = \frac{\pi_y(w_y E[N(y-1)] + 1 - p_y)}{p_y(\pi_y w_y + (1-p_y)(b_y + w_y))}$$

for all $y \in \{2, 3, \ldots, m-1\}$.

Proof First the formulæ are established for $E[N(m)]$ and $E[N(m-1)]$. Evidently, $Y_0 \leq m$ so $E[N(m)] = 0$. If $m > 1$ then $Y_0 \leq m - 1$ with probability $1 - \pi_m$; in this case $N(m-1) = 0$. Otherwise, Y_0 must be m and the first value distinct from Y_0 must be less than m; in this case $N(m-1)$ is the length of the initial hesitation at Y_0. As $w_m = 0$, the probability of hesitating at each iteration is $1 - b_m$, whence $E[N(m-1)] = \pi_m / b_m$.

The proof of the difference equation makes use of the following equality, valid for $m > 2$ and $y \in \{2, 3, \ldots, m-1\}$:

$$\begin{aligned} E[N(y-1)] &= P(Y_0 < y)E[N(y-1)|Y_0 < y] \\ &\quad + P(Y_0 = y)E[N(y-1)|Y_0 = y] \\ &\quad + P(Y_0 > y)E[N(y-1)|Y_0 > y]. \end{aligned}$$

The expectation of $N(y-1)$ in the first term above is zero, since $Y_0 \leq y - 1$. Thus

$$\begin{aligned} E[N(y-1)] &= \pi_y E[N(y-1)|Y_0 = y] \\ &\quad + (1 - p_y)E[N(y-1)|Y_0 > y]. \end{aligned} \tag{4.1}$$

Define h as the first iteration number where Y_h differs from Y_0. In the case where $Y_0 = y$, conditioning $N(y-1)$ on whether Y_h is smaller or larger than y gives

$$\begin{aligned} E[N(y-1)] &= \pi_y P(Y_h < y)E[N(y-1)|Y_h < Y_0 = y] \\ &\quad + \pi_y P(Y_h > y)E[N(y-1)|Y_h > Y_0 = y] \\ &\quad + (1 - p_y)E[N(y-1)|Y_0 > y] \\ &= \pi_y E[h|Y_0 = y] \\ &\quad + \pi_y P(Y_h < y)E[N(y-1) - h|Y_h < Y_0 = y] \\ &\quad + \pi_y P(Y_h > y)E[N(y-1) - h|Y_h > Y_0 = y] \\ &\quad + (1 - p_y)E[N(y-1)|Y_0 > y]. \end{aligned}$$

The expectation of $N(y-1)-h$ in the second term above is zero, since in this case the sample path first falls below y at iteration h, whence $N(y-1)=h$. Thus

$$\begin{aligned} E[N(y-1)] &= \frac{\pi_y}{b_y+w_y} + \frac{\pi_y w_y E[N(y-1)|Y_0>y]}{b_y+w_y} \\ &\quad + (1-p_y)E[N(y-1)|Y_0>y], \end{aligned}$$

since the total probability of leaving y at any iteration is b_y+w_y. The expectation in the second term above has been simplified using the fact that the local range distribution of Y_h, given that $Y_h > Y_0 = y$, is a normalised restriction of the entire range distribution, and that the search algorithm is memoryless. Transition probabilities from level $Y_h > y$ are thus independent of any prior history. Rearranging the equation gives

$$E[N(y-1)|Y_0>y] = \frac{(b_y+w_y)E[N(y-1)]-\pi_y}{\pi_y w_y + (1-p_y)(b_y+w_y)}.$$

Combining this with (4.1) yields

$$E[N(y-1)|Y_0=y] = \frac{w_y E[N(y-1)]+1-p_y}{\pi_y w_y + (1-p_y)(b_y+w_y)}. \tag{4.2}$$

The difference equation can now be established in the following way, for $y \in \{2,3,\ldots,m-1\}$:

$$\begin{aligned} E[N(y-1)]-E[N(y)] &= E[N(y-1)-N(y)] \\ &= P(Y_{N(y)}=y)E[N(y-1)-N(y)|Y_{N(y)}=y] \\ &\quad + P(Y_{N(y)}<y)E[N(y-1)-N(y)|Y_{N(y)}<y]. \end{aligned}$$

The expectation of $N(y-1)-N(y)$ in the second term above is zero, since $Y_{N(y)} \le y-1$, whence $N(y-1)=N(y)$. Thus

$$\begin{aligned} E[N(y-1)]-E[N(y)] &= \frac{\pi_y E[N(y-1)-N(y)|Y_{N(y)}=y]}{p_y} \\ &= \frac{\pi_y E[N(y-1)|Y_0=y]}{p_y} \end{aligned}$$

again using the fact that the search algorithm is memoryless.

Substituting the expression for $E[N(y-1)|Y_0 = y]$ from (4.2) provides the desired result. ■

The following corollary can now be derived, using theory of difference equations.

Corollary 4.3.1 *The expected number of iterations of finite backtracking adaptive search before reaching level y, $E[N(y)]$, satisfies*

i) For all m, $E[N(m)] = 0$.

ii) For $m > 1$, $E[N(m-1)] = \pi_m/b_m$.

iii) For $m > 2$,

$$E[N(y)] = \left(\frac{\pi_m}{b_m} + \sum_{j=y+1}^{m-1}\left(\frac{\pi_j}{p_j b_j + p_{j-1}w_j}\prod_{i=j}^{m-1} V_i\right)\right)\prod_{i=y+1}^{m-1}\frac{1}{V_i}$$

where $V_i = \frac{(1-p_i)(p_i b_i + p_{i-1}w_i)}{p_i(\pi_i w_i + (1-p_i)(b_i+w_i))}$, for all $y \in \{1,2,\ldots,m-2\}$.

Proof The expectations of $N(m)$ and $N(m-1)$ are proved in the theorem. It is convenient to write the difference equation of Theorem 4.3.1 as follows,

$$E[N(y-1)] - A(y)E[N(y)] = B(y) \tag{4.3}$$

where $A(y) = p_y(\pi_y w_y + (1-p_y)(b_y+w_y))/((1-p_y)(p_y b_y + p_{y-1}w_y))$ and $B(y) = \pi_y/(p_y b_y + p_{y-1}w_y)$, for all $y \in \{2,3,\ldots,m-1\}$.

When $m = 3$, (4.3) can be used with $y = 2$ to show that $E[N(1)] = (\pi_3/b_3)A(2) + B(2)$, which agrees with the result stated in the corollary.

Assuming now that $m > 3$, divide both sides of (4.3) by $\prod_{i=y}^{m-1} A(i)$ (valid since $A(i) > 0$ for all $i \in \{2,3,\ldots,m-1\}$), to give

$$\frac{E[N(y-1)]}{\prod_{i=y}^{m-1} A(i)} - \frac{E[N(y)]}{\prod_{i=y+1}^{m-1} A(i)} = \frac{B(y)}{\prod_{i=y}^{m-1} A(i)}$$

for $y \in \{2, 3, \ldots, m-2\}$. The left hand side is now a first difference of $E[N(y)] / \prod_{i=y+1}^{m-1} A(i)$. The general solution for an equation of this form is a constant plus a summation in which the right hand side provides the term [33, p. 153]. Thus

$$\frac{E[N(y)]}{\prod_{i=y+1}^{m-1} A(i)} = C + \sum_{j=y+1}^{m-1} \frac{B(j)}{\prod_{i=j}^{m-1} A(i)}. \tag{4.4}$$

Observe that this solution satisfies Equation (4.3) for all $y \in \{1, 2, \ldots, m-2\}$.

It remains to find the constant C. Since $E[N(m-1)] = \pi_m / b_m$, (4.3) implies that $E[N(m-2)] = (\pi_m/b_m)A(m-1) + B(m-1)$. Setting $y = m-2$ in (4.4) gives $E[N(m-2)] = CA(m-1) + B(m-1)$, and since $A(m-1) > 0$ it follows that $C = \pi_m / b_m$. Substituting the expressions for $A(y)$, $B(y)$ and C back into (4.4) now yields the result of the corollary. ■

Thus the expected number of iterations before convergence for a backtracking adaptive search approximation to any domain process can be found without need for matrix inversion. This is illustrated in Section 4.5 via several examples, including special cases for which analysis has been published elsewhere. Section 4.6 then shows how the backtracking adaptive search approximation is formed.

First, though, a digression is made to generalise the result of this section.

4.4 Expected search duration for an algorithm with arbitrary initial distribution

This section produces a result completely analogous to the result of the preceding section, for a general backtracking adaptive search algorithm in which an initial distribution distinct from π is specified. The general stochastic global optimisation algorithm of Chapter 2 specifies a particular initial distribution; it is therefore sensible that the backtracking adaptive search approximation for this algorithm should be modified by incorporating this initial distribution. This section provides the necessary extension to the theory of backtracking adaptive search.

As before, the expression for the expected convergence time of this generalisation of backtracking adaptive search can be evaluated quickly by computer with no need for complicated procedures such as matrix inversion. This form of backtracking adaptive search is the one implemented in Chapter 6.

Backtracking adaptive search is defined exactly as before, except that the initial range level Y_0 is generated not according to the range distribution π but according to a new distribution defined as $P(Y_0 = y) = \sigma_y$. Define $s_y = \sum_{i=1}^{y} \sigma_i$. It is required that $\pi_1 > 0$ and that either σ_y or π_y must be positive for all $y \in \{2, 3, \ldots, m\}$. Define $y^* = \max\{y : \pi_y > 0\}$. Also $w_y = 0$ for all $y \geq y^*$; $b_1 = 0$ and $b_y + w_y > 0$ for all $y \in \{2, 3, \ldots, m\}$. Finally, the notation for expectations is now extended to specify to which form of backtracking adaptive search the expectation applies. Expectations denoted $E_\pi[\cdot]$ refer to ordinary finite backtracking adaptive search, while those denoted $E_\sigma[\cdot]$ apply to the generalisation with arbitrary initial distribution considered here.

Theorem 4.4.1 *The expected number of iterations of finite backtracking adaptive search with an arbitrary initial distribution before first reaching a certain level y, $E_\sigma[N(y)]$, satisfies*

i) For all m, $E_\sigma[N(m)] = 0$.

ii) For $m > 1$, $E_\sigma[N(y)] = \sum_{i=y+1}^{m} \sigma_i/b_i$ for all $y \in \{y^*, y^*+1, \ldots, m-1\}$ where $y^* < m$ and $E_\sigma[N(y^*-1)] = (1 - s_{y^*})\pi_{y^*}/b_{y^*} + \sum_{i=y^*}^{m} \sigma_i/b_i$ where $y^* > 1$.

iii) For $m > 2$,

$$E_\sigma[N(y-1)] - E_\sigma[N(y)] = \frac{(\sigma_y p_y + \pi_y(1 - s_y))(w_y E_\pi[N(y-1)] + 1 - p_y)}{p_y(\pi_y w_y + (1 - p_y)(b_y + w_y))}$$

where

$$E_\pi[N(y-1)] = \left(\frac{\pi_m}{b_m} + \sum_{j=y}^{m-1}\left(\frac{\pi_j}{p_j b_j + p_{j-1} w_j}\prod_{i=j}^{m-1} V_i\right)\right)\prod_{i=y}^{m-1}\frac{1}{V_i}$$

and $V_i = \frac{(1-p_i)(p_i b_i + p_{i-1} w_i)}{p_i(\pi_i w_i + (1-p_i)(b_i + w_i))}$, for all $y \in \{2, 3, \ldots, y^*-1\}$ where $y^* > 2$.

Proof As in the proof of Theorem 4.3.1, the formulæ for special cases are first established. Evidently, $Y_0 \leq m$ so $E[N(m)] = 0$.

If $m > 1$ then $Y_0 \leq y$ with probability s_y; in this case $N(y) = 0$. Otherwise, if $y \geq y^*$ then the first value distinct from Y_0, labelled Y_d, must be no greater than y; in this case $N(y)$ is the length of the initial hesitation at Y_0. Since $w_y = 0$ for $y \geq y^*$, the probability of hesitating at each iteration is $1 - b_y$, whence $E_\sigma[N(y)] = \sum_{i=y+1}^{m} \sigma_i/b_i$ where $y^* \leq y < m$.

Clearly, $E_\sigma[N(y^* - 1)]$ is positive only if $Y_0 \geq y^*$. Then since y^* is the only level greater than $y^* - 1$ for which $\pi_y > 0$, $E_\sigma[N(y^* - 1)]$ is the sum of any initial hesitation plus the hesitation at y^* if this level is visited. The probability of hesitating at each iteration is $1 - b_y$ for all $y \geq y^*$; thus when $y^* > 1$ and $s_{y^*-1} < 1$

$$\begin{aligned}
E_\sigma[N(y^* - 1)] &= P(Y_0 = y^*)E_\sigma[N(y^* - 1)|Y_0 = y^*] + \sum_{i=y^*+1}^{m} P(Y_0 = i)E_\sigma[N(y^* - 1)|Y_0 = i] \\
&= P(Y_0 = y^*)E_\sigma[d|Y_0 = y^*] \\
&\quad + \sum_{i=y^*+1}^{m} P(Y_0 = i)\left(E_\sigma[d|Y_0 = i] + E_\sigma[N(y^* - 1) - d|Y_0 = i]\right) \\
&= \sum_{i=y^*}^{m} P(Y_0 = i)E_\sigma[d|Y_0 = i] \\
&\quad + \sum_{i=y^*+1}^{m} P(Y_0 = i)\left(\, P(Y_d = y^*)(E_\sigma[N(y^* - 1) - d|Y_d = y^* < Y_0 = i]\right. \\
&\quad \left. + P(Y_d < y^*)(E_\sigma[N(y^* - 1) - d|Y_d < y^* < Y_0 = i]\,\right) \\
&= \sum_{i=y^*}^{m} \frac{\sigma_i}{b_i} + \frac{(1 - s_{y^*})\pi_{y^*}}{b_{y^*}}
\end{aligned}$$

since $N(y^* - 1) = d$ when $Y_d < y^* < Y_0$. The formula also correctly gives $E[N(y^* - 1)] = 0$ when $s_{y^*-1} = 1$.

The proof of the difference equation now proceeds using the following equality, valid for $m > 2$ and $y \in \{2, 3, \ldots, y^* - 1\}$ where $y^* > 2$:

$$\begin{aligned}
E_\sigma[N(y - 1)] - E_\sigma[N(y)] &= E_\sigma[N(y - 1) - N(y)] \\
&= P(Y_{N(y)} = y)E_\sigma[N(y - 1) - N(y)|Y_{N(y)} = y] \\
&\quad + P(Y_{N(y)} < y)E_\sigma[N(y - 1) - N(y)|Y_{N(y)} < y] \\
&= P(Y_{N(y)} = y)E_\sigma[N(y - 1) - N(y)|Y_{N(y)} = y]. \qquad (4.5)
\end{aligned}$$

Observe that $Y_{N(y)} = y$ either if $Y_0 = y$ or if $Y_0 > y$ and level y is reached in the process of the algorithm. This gives $P(Y_{N(y)} = y) = \sigma_y + (1 - s_y)\pi_y/p_y$.

Now since the algorithm is memoryless, and the different forms of backtracking adaptive search are identical conditional on any current level, $E_\sigma[N(y-1) - N(y)|Y_{N(y)} = y] = E_\pi[N(y-1)|Y_0 = y]$. Substituting into Equation (4.5) the probability derived above, the expression for $E_\pi[N(y-1)|Y_0 = y]$ from Equation (4.2) and then an expression for $E_\pi[N(y-1)]$ from Corollary 4.3.1 provides the desired result. ■

The usual form of backtracking adaptive search is a special case of the general form treated here; Theorem 4.3.1 can be obtained from Theorem 4.4.1 by letting $\pi = \sigma$.

In the same way as Corollary 4.3.1 was derived, the following corollary is now obtained.

Corollary 4.4.1 *The expected number of iterations of finite backtracking adaptive search with an arbitrary initial distribution before reaching level y, $E_\sigma[N(y)]$, satisfies*

i) For all m, $E_\sigma[N(m)] = 0$.

ii) For $m > 1$, $E_\sigma[N(y)] = \sum_{i=y+1}^{m} \sigma_i/b_i$ for all $y \in \{y^*, y^*+1, \ldots, m-1\}$ where $y^* < m$ and $E_\sigma[N(y^*-1)] = (1 - s_{y^*})\pi_{y^*}/b_{y^*} + \sum_{i=y^*}^{m} \sigma_i/b_i$ where $y^* > 1$.

iii) For $m > 2$,

$$E_\sigma[N(y)] = E_\sigma[N(y^*-1)] + \sum_{k=y+1}^{y^*-1} \frac{(\sigma_k p_k + \pi_k(1-s_k))(w_k E_\pi[N(k-1)] + 1 - p_k)}{p_k(\pi_k w_k + (1-p_k)(b_k + w_k))}$$

where

$$E_\pi[N(y-1)] = \left(\frac{\pi_m}{b_m} + \sum_{j=y}^{m-1}\left(\frac{\pi_j}{p_j b_j + p_{j-1} w_j}\prod_{i=j}^{m-1} V_i\right)\right)\prod_{i=y}^{m-1}\frac{1}{V_i}$$

and $V_i = \frac{(1-p_i)(p_i b_i + p_{i-1} w_i)}{p_i(\pi_i w_i + (1-p_i)(b_i + w_i))}$, for all $y \in \{1, 2, \ldots, y^*-2\}$ where $y^* > 2$.

Proof The expectations of the special cases are proved in the theorem. To solve the difference equation of the theorem, note that

$$E_\sigma[N(y)] = E_\sigma[N(y^* - 1)] + \sum_{k=y+1}^{y^*-1} \left(E_\sigma[N(k-1)] - E_\sigma[N(k)]\right),$$

into which the difference equation may be substituted to provide the solution. ∎

Analytic expressions for the expected number of iterations before convergence for a backtracking adaptive search approximation to any domain process have thus been found. The following section provides several examples. Finally, in Section 4.6 a method of forming a backtracking adaptive search approximation to an asymptotic averaged range process is presented.

4.5 Examples

The results of the preceding sections allow expected convergence times to be calculated for any algorithm that can be represented within the framework of finite BAS, with any initial distribution. This section illustrates the results using several instances of finite BAS.

4.5.1 A small example

Progress of the algorithm may be demonstrated on an artificial problem with a small range. (The exact nature of the domain in the artificial problem is irrelevant, since the definition of backtracking adaptive search makes use only of parameters defined in the range.) A uniform probability distribution with $m = 4$ levels is used, so $\pi_i = \frac{1}{4}$ for $i \in \{1, 2, 3, 4\}$. The vectors $(b_1, b_2, b_3, b_4) = (0, 0.1, 0.8, 0.9)$ and $(w_1, w_2, w_3, w_4) = (0, 0.4, 0.2, 0)$ are chosen to reflect a decreasing likelihood of improvement as the algorithm nears the optimum. Applying Corollary 4.3.1 with $y = 1$ now shows that the expected number of iterations before reaching range level 1 is $3\frac{1}{3}$.

Figure 4.1 shows the distribution of the number of iterations this algorithm takes to reach the absorbing state, using simulation. Corollary 4.3.1 provides the expected value of the number of iterations; the moment generating function of this distribution is derived in the following chapter.

Taking range level 1 as an absorbing level (representing convergence of the algorithm) gives rise to the following Markov range transition matrix. (Transition probabilities from the other levels are derived from the backtracking adaptive search parameters according to the definition of backtracking adaptive search in Section 4.2. For instance, the probability of improving from level 3 is $b(3) = 0.8$ and the normalised restriction of π to the first two levels is $\frac{1}{1/4+1/4}\begin{bmatrix}\frac{1}{4} & \frac{1}{4}\end{bmatrix} = \begin{bmatrix}\frac{1}{2} & \frac{1}{2}\end{bmatrix}$; thus the first two entries in the third row of the transition matrix are $0.8\begin{bmatrix}\frac{1}{2} & \frac{1}{2}\end{bmatrix} = [0.4\ 0.4]$.)

$$\text{Level}\quad \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.1 & 0.5 & 0.2 & 0.2 \\ 0.4 & 0.4 & 0 & 0.2 \\ 0.3 & 0.3 & 0.3 & 0.1 \end{bmatrix} \end{array}$$

Standard matrix theory [27] confirms that the expected time is $3\frac{1}{3}$ iterations. To obtain this result, the transient portion of the transition matrix is subtracted from the 3×3 identity matrix, and the difference inverted. Multiplying the transient portion of the initial distribution by the vector of row-sums of this inverse then gives the solution. However, forming the transition matrix and applying this method is prohibitively time-consuming for problems with a large number of range levels, due to the need for matrix inversion. This highlights the contribution made by Corollary 4.3.1.

4.5.2 A small example with randomly generated initial distribution

The same problem is now solved using a backtracking adaptive search algorithm with randomly generated initial distribution $\sigma = [0.2166\ 0.1130\ 0.2900\ 0.3804]$.

Corollary 4.4.1 now shows that the expected number of iterations before reaching range level 1 is 3.32. Figure 4.2 shows the distribution of the number of iterations this algorithm takes to reach the absorbing state, using simulation.

In exactly the same way as before, standard matrix theory [27] can confirm that the expected time is 3.32 iterations. Matrix inversion is again necessary using this method; however, the result can be obtained using Corollary 4.4.1 without this requirement. The corollary thus provides a means of calculating expected convergence times in problems too large for solution via matrix inversion.

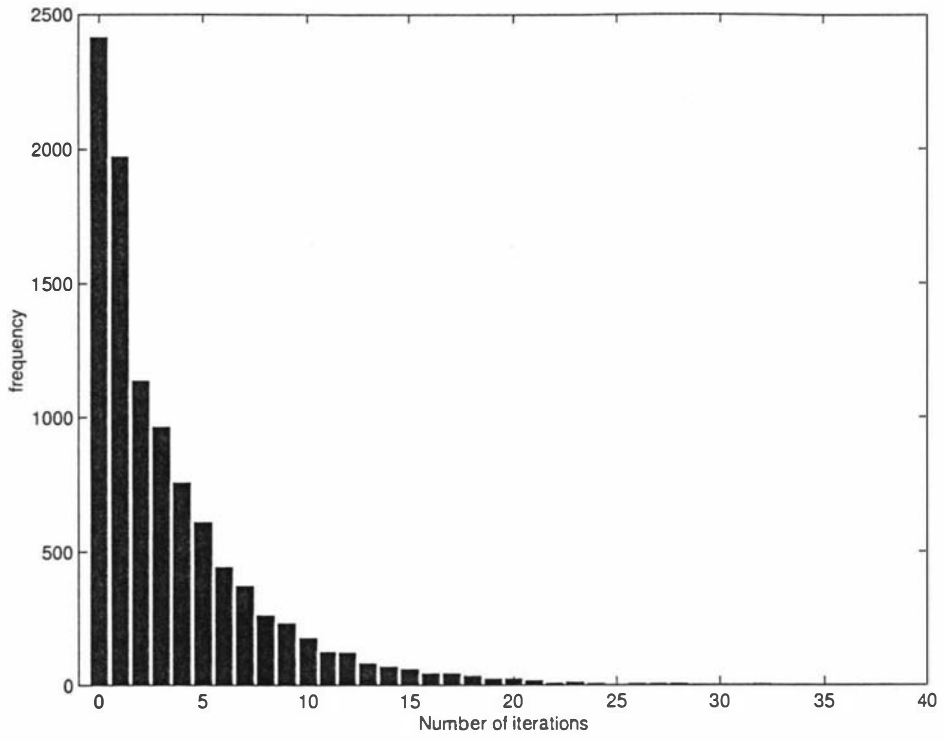

Figure 4.1: Number of iterations before convergence to the optimal state for 10000 runs of the algorithm given in Example 4.5.1.

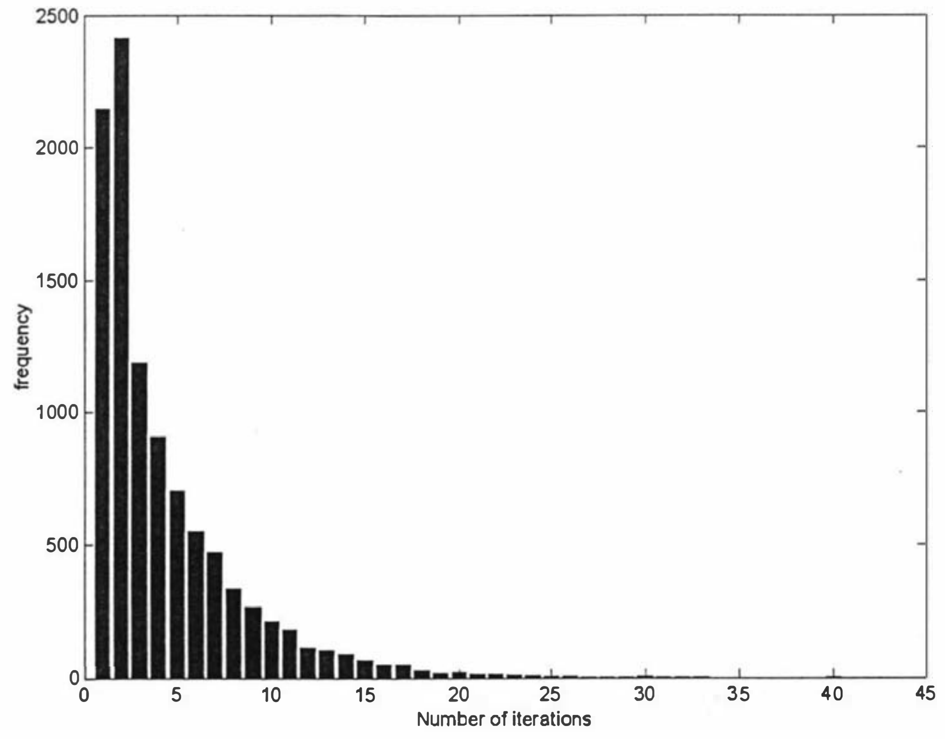

Figure 4.2: Number of iterations before convergence to the optimal state for 10000 runs of the algorithm given in Example 4.5.2.

4.5.3 Combination of pure adaptive search and pure random search

A particular realisation of BAS is analysed in [31]. This algorithm samples uniformly from points in the domain of equal or better objective function value with probability p, and otherwise performs pure random search. Worsening points are accepted with probability t; otherwise the algorithm hesitates. Bounds for the expected convergence time with constant p and t are presented in that paper; an expression for the exact value is now available. In this case, the appropriate formulæ are $b_y = pp_{y-1}/p_y + (1-p)p_{y-1}$ and $w_y = (1-p)(1-p_y)t$. Thus, from Corollary 4.3.1, $E[N(m)] = 0$, $E[N(m-1)] = \pi_m/p_{m-1}$ and

$$E[N(y)] = \left(\frac{\pi_m}{p_{m-1}} + \sum_{j=y+1}^{m-1}\left(\frac{\pi_j}{p_{j-1}(p+(1-p)(p_j+(1-p_j)t))}\prod_{i=j}^{m-1}V_i\right)\right)\prod_{i=y+1}^{m-1}\frac{1}{V_i}$$

where $V_i = \frac{p_{i-1}(p+(1-p)(p_i+(1-p_i)t))}{p_i(1-p)(1-p_{i-1})t+p_{i-1}(p+(1-p)p_i)}$, for $y \in \{1, 2, \ldots, m-2\}$ and $m > 2$. The result holds also if p and t are functions of y.

Figure 4.3 shows how expected convergence times vary with p and t in this algorithm. The probability mass function π_y is chosen to be uniform on $\{1, 2, \ldots, 20\}$. Unsurprisingly, since this algorithm will never benefit by backtracking, convergence time increases with decreasing p and increasing t. When $p = 0$ the algorithm has the convergence properties of pure random search; when $p = 1$ the algorithm is PAS. In neither of these cases does the probability of accepting a worsening point have any effect on convergence time: in PAS worsening points are never generated, while in pure random search the number of iterations remaining until reaching the optimum is the same whether worsening points are accepted or not. The results shown here verify the result of Corollary 4.3.1 using a previous investigation, reported in Figure 11 of [31].

4.5.4 Finite hesitant adaptive search

Corollary 4.4.1 provides as a special case the expected number of iterations before convergence for finite hesitant adaptive search with arbitrary initial distribution. The result for finite hesitant adaptive search is obtained by making the substitution $w_y = 0$ for all y.

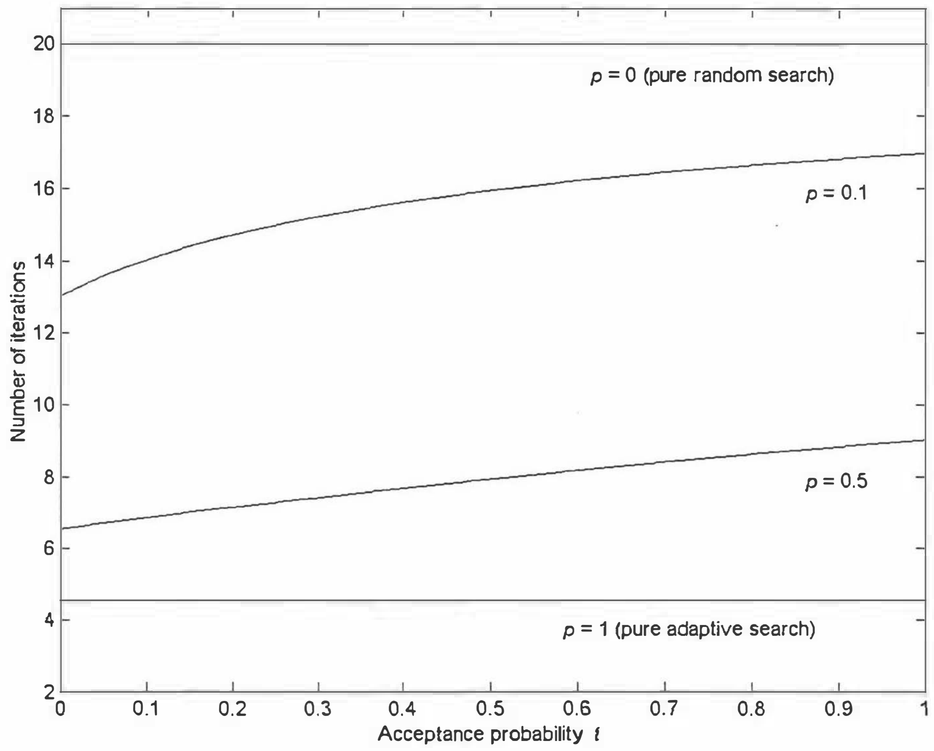

Figure 4.3: Number of iterations until convergence to the optimal state for the example given in Subsection 4.5.3 as t, the acceptance probability, ranges from 0 to 1 and for $p \in \{0, 0.1, 0.5, 1\}$.

Theorem 4.5.1 *The expected number of iterations of finite hesitant adaptive search with an arbitrary initial distribution before first reaching level y, $E_\sigma[N(y)]$, satisfies*

i) For all m, $E_\sigma[N(m)] = 0$.

ii) For $m > 1$, $E_\sigma[N(y)] = \sum_{i=y+1}^{m} \frac{\sigma_i p_i + \pi_i(1-s_i)}{p_i b_i}$ for all $y \in \{1, 2, \ldots, m-1\}$.

Proof The first case is inherited from Corollary 4.4.1. The second case in that corollary implies that for $m > 1$, $E_\sigma[N(y)] = \sum_{i=y+1}^{m} \sigma_i / b_i$ for all $y \in \{y^*, y^*+1, \ldots, m-1\}$ where $y^* < m$; Theorem 4.5.1 simplifies to this expression since $\pi_y = 0$ for $y > y^*$. Also,

$$\begin{aligned}
E_\sigma[N(y^*-1)] &= \frac{(1-s_{y^*})\pi_{y^*}}{b_{y^*}} + \sum_{i=y^*}^{m} \frac{\sigma_i}{b_i} \\
&= \frac{(1-s_{y^*})\pi_{y^*}}{b_{y^*}} + \frac{\sigma_{y^*}}{b_{y^*}} + E_\sigma[N(y^*)] \\
&= \frac{(1-s_{y^*})\pi_{y^*} + \sigma_{y^*} p_{y^*}}{p_{y^*} b_{y^*}} + \sum_{i=y^*+1}^{m} \frac{\sigma_i p_i + \pi_i(1-s_i)}{p_i b_i}
\end{aligned}$$

$$= \sum_{i=y^*}^{m} \frac{\sigma_i p_i + \pi_i(1-s_i)}{p_i b_i} \tag{4.6}$$

where $y^* > 1$; this also agrees with Theorem 4.5.1.

It remains therefore to prove that the third case of Corollary 4.4.1 is given by the expression of Theorem 4.5.1 when $w_y = 0$ for all y. This substitution leads, after simplification, to

$$E_\sigma[N(y)] = E_\sigma[N(y^*-1)] + \sum_{k=y+1}^{y^*-1} \frac{\sigma_k p_k + \pi_k(1-s_k)}{p_k b_k}$$

for all $y \in \{1, 2, \ldots, y^*-2\}$ where $y^* > 2$. Now substitution of the expression for $E_\sigma[N(y^*-1)]$ from Equation (4.6) produces the desired result. ■

When the initial distribution is π, this further simplifies to

$$E[N(y)] = \sum_{j=y+1}^{m} \frac{\pi_j}{p_j b_j}$$

for all $y \in \{1, 2, \ldots, m-1\}$ and $m > 1$, with $E[N(m)] = 0$. Note that the summation is over only transient range levels; this constitutes a minor correction to the result shown in [51].

4.5.5 Finite pure adaptive search

The equivalent results for PAS are now easily derived from the above.

Theorem 4.5.2 *The expected number of iterations of finite pure adaptive search with an arbitrary initial distribution before first reaching level y, $E_\sigma[N(y)]$, satisfies*

i) For all m, $E_\sigma[N(m)] = 0$.

ii) For $m > 1$, $E_\sigma[N(y)] = \sum_{j=y+1}^{m} \frac{\sigma_j p_j + \pi_j(1-s_j)}{p_j}$ for all $y \in \{1, 2, \ldots, m-1\}$.

Proof Substituting $b_y = 1$ for all y in Theorem 4.5.1 provides the result. ■

When the initial distribution is π, this gives

$$E[N(y)] = \sum_{j=y+1}^{m} \frac{\pi_j}{p_j}$$

for all $y \in \{1, 2, \ldots, m-1\}$ and $m > 1$, with $E[N(m)] = 0$. A special case of this formula is derived in [56]. There the search duration is defined to include the final iteration, and thus the solution is reported as $E[N(1)] = 1 + \sum_{i=2}^{m} \pi_i / p_i$ where $m > 1$.

4.5.6 Pure random search

Corollaries 4.3.1 and 4.4.1 also provide expected convergence times for pure random search with arbitrary initial distribution.

The expected number of iterations before convergence for pure random search is available from Corollary 4.3.1 by letting $b_y = p_{y-1}$ and $w_y = 1 - p_y$. Thus

$$\begin{aligned} E[N(y)] &= \left(\frac{\pi_m}{p_{m-1}} + \sum_{j=y+1}^{m-1} \left(\frac{\pi_j}{p_{j-1}} \prod_{i=j}^{m-1} \frac{p_{i-1}}{p_i} \right) \right) \prod_{i=y+1}^{m-1} \frac{p_i}{p_{i-1}} \\ &= \left(\frac{\pi_m}{p_{m-1}} + \sum_{j=y+1}^{m-1} \frac{\pi_j}{p_{m-1}} \right) \frac{p_{m-1}}{p_y} \\ &= \frac{1 - p_y}{p_y} \end{aligned} \tag{4.7}$$

for all $y \in \{1, 2, \ldots, m-2\}$ and $m > 2$. Note that this formula also correctly gives the values of $E[N(m-1)] = \pi_m / p_{m-1}$ and $E[N(m)] = 0$ for all m. This agrees with the result of direct calculation.

Substituting this result into Corollary 4.4.1 now provides the expected number of iterations before convergence of pure random search with arbitrary initial distribution σ. Note that $y^* = m$ for pure random search, so $E_\sigma[N(y^* - 1)] = \sigma_m / p_{m-1}$ where $m > 1$. Thus

$$\begin{aligned} E_\sigma[N(y)] &= \frac{\sigma_m}{p_{m-1}} + \sum_{k=y+1}^{m-1} \frac{\sigma_k p_k + \pi_k(1 - s_k)}{p_{k-1} p_k} \\ &= \sum_{k=y+1}^{m} \left(\frac{1 - s_{k-1}}{p_{k-1}} - \frac{1 - s_k}{p_k} \right) \end{aligned}$$

$$= \frac{1-s_y}{p_y}$$

for all $y \in \{1, 2, \ldots, m-2\}$ and $m > 2$. Note that this formula also correctly gives the values of $E[N(m-1)]$ and $E[N(m)] = 0$ for all m. Equation (4.7) can immediately be recovered by substituting $\sigma = \pi$. The same result can, of course, be calculated directly from Equation (4.7) as $\sum_{k=y+1}^{m} \sigma_k / p_y$.

4.6 Approximating the asymptotic averaged range process with backtracking adaptive search

This section details the method used to find a finite backtracking adaptive search approximation to an asymptotic averaged range process. The approach used is very simple and intended only as an illustration; a more complicated method is later shown to yield better results.

Finite backtracking adaptive search uses three parameters: b, w and π. (When the initial distribution is not π, a fourth parameter is used, the value of which is known.) Since the asymptotic averaged range process transition matrix has none of the restrictions imposed for finite backtracking adaptive search, any method of estimating these parameters inevitably introduces some error: the expected convergence times of the two processes will differ. The approach used in this thesis is one sensible approach among several that could be used. It is possible that another method may be proved better than the one described here; the present aim is only to initiate an approximation method in order to indicate the kind of results attainable. This approximation method stands to be improved by subsequent research.

Bettering probabilities at each level can be found directly from the asymptotic averaged range process transition matrix R as

$$b_i = \sum_{j=1}^{i-1} R_{ij} \tag{4.8}$$

for $i > 1$ and $b_1 = 0$. Similarly,

$$w_i = \sum_{j=i+1}^{m} R_{ij} \tag{4.9}$$

for $i < m$ and $w_m = 0$.

As illustrated in Subsection 4.5.1, there is a Markov range transition matrix corresponding to any particular values of b, w and π. This matrix has a special form, not in general shared by the transition matrix of the asymptotic averaged range process. It then remains to find an estimator for π given these values for b and w, so that the discrepancy between the asymptotic averaged range process transition matrix R and the transition matrix implied by backtracking adaptive search is made as small as possible.

An intuitive justification for the method chosen is first provided. A sensible estimate of the range distribution can be obtained from a weighted average of the bettering and worsening range distributions at each level in the asymptotic averaged range process. Distributions for levels more commonly visited are given a greater weighting and levels less frequently visited are given a smaller weighting. Note that this estimation method excludes hesitations in the range from consideration, since π is only used when the backtracking adaptive search algorithm either betters or worsens.

Using the results of Subsection 3.3.3, the limiting average probabilities of being in each transient domain state given that the process has not converged may be denoted by the vector

$$\bar{\beta} = \sum_{a=0}^{d-1} \frac{vQ^a}{d\|vQ^a\|_1}. \tag{4.10}$$

No transitions from the absorbing level are to be considered (since the algorithms involved make no use of these); thus the full l-vector of weightings for each state is $\bar{\beta}T_l^{\mathrm{T}}$ and the limiting average probabilities of being in each transient range level given that the process has not converged is $\bar{\beta}T_l^{\mathrm{T}}M$. Let

$$R_{ij}^{\mathrm{d}} = \begin{cases} R_{ij}, & i = j \\ 0, & \text{otherwise;} \end{cases}$$

then the estimate of π is

$$\frac{\bar{\beta}{T_l}^{\mathrm{T}} M(R - R^{\mathrm{d}})}{\|\bar{\beta}{T_l}^{\mathrm{T}} M(R - R^{\mathrm{d}})\|_1}. \tag{4.11}$$

The heuristic for estimating π described above is easy to evaluate for any asymptotic averaged range process. This section provides a simple method of estimating parameters b, w and π for backtracking adaptive search. Refinement of this approach is possible; however, the approach described in this section is sufficient to allow the full approximation process to be demonstrated.

4.7 Summary

This chapter provides the theory of backtracking adaptive search required for the approximation process described in Chapter 2. Any Markovian algorithm for which the domain process is known can now be approximated by a backtracking adaptive search algorithm in the range. The expected convergence time of the backtracking adaptive search algorithm is then an estimate of the expected convergence time of the original algorithm. Chapter 6 demonstrates this approximation process.

For the sake of completeness, however, the following chapter generalises the analysis in Section 4.3, where there is no parameter for the arbitrary initial distribution, to obtain the complete distribution of the number of iterations before convergence for backtracking adaptive search on mixed domains.

Chapter 5

The distribution of the number of iterations to convergence for backtracking adaptive search

5.1 Introduction

This chapter provides a full summary of results obtained concerning the distribution of the number of iterations before convergence for backtracking adaptive search. The factorial moment generating function of this distribution is derived for backtracking adaptive search in the general case where the range distribution has a mixture of continuous and discrete components.

The results of Section 4.3 could be taken as a special case of the material presented in this chapter; however, the later sections of that chapter are not dependent on the distributions derived in this chapter. Whereas Chapter 4 was concerned with defining and analysing backtracking adaptive search as an end-point to the approximation framework, this chapter provides a more complete analysis of the algorithm in a general context.

In Section 5.2 backtracking adaptive search is defined using a range distribution that may be continuous, discrete or a mixture of both. Section 5.3 presents the distribution results for each possible type of range distribution. In Subsection 5.3.1 a moment generating function is provided for the delays between improvements in the best level reached so far by the algorithm; this is used in Subsections 5.3.2, 5.3.3 and 5.3.4 to treat the discrete, finite and continuous cases respectively. Subsection 5.3.5 then provides a factorial moment generating function for backtracking adaptive search with a general range distribution. The final section presents a *summary* of the analysis of backtracking adaptive search.

5.2 A general definition of backtracking adaptive search

Since the results of this chapter apply to backtracking adaptive search where the range distribution may be continuous, the notation of finite backtracking adaptive search must be generalised.

The range is now allowed to be the real numbers. Define a range probability measure ρ, so that $P(Y_0 \in A) = \rho(A)$ for any measurable set of real numbers A. Finite backtracking adaptive search notation $\rho(\{t\}) = \pi_t$ is used when the set A is a singleton. The cumulative range distribution function is defined as $p_y = \rho(\{t : t \leq y\})$. The algorithm terminates on reaching any point with sufficiently low objective function level. The termination region is denoted by T. Note that for all levels y such that $p_y = 1$, the number of iterations before convergence is exactly zero. In what follows, analysis is restricted to the number of iterations before reaching levels y for which $p_y < 1$.

Again, the algorithm initially samples Y_0 from the range according to the probability measure ρ. At each iteration thereafter, one of three things happens. At the nth iteration, with probability b_{Y_n}, the algorithm improves, sampling the next evaluation point according to the normalised restriction of ρ to the current improving set. With probability w_{Y_n}, the algorithm backtracks, sampling the next evaluation point according to the normalised restriction of ρ to the current worsening set. Otherwise the algorithm hesitates, remaining at the current evaluation point. Functions b and w depend only on the current level.

Restrictions are required on b, w and ρ to ensure that the algorithm will sample from T in finite time. Firstly, if there exists a level t such that $\pi_t > 0$ and $p_t = 1$ then assume $w_t = 0$.

Secondly, it is assumed that there exist a positive real number $\epsilon < 1$ and a real number K so that the following conditions are satisfied:

1. $\rho(\{y : y \geq K\}) \geq \epsilon$ and $b_y \geq \epsilon$ for all $y \geq K$
2. $\rho(T) \geq \epsilon$
3. $b_y + w_y \geq \epsilon$ for all $y \notin T$

Then whenever a new level is chosen, that new level is either better or worse than the current level. If it is better, there is a probability at least $\epsilon > \epsilon^3$ of sampling from T; if it is worse, the probability of sampling a level no less than K is at least ϵ, the probability of improving at the next iteration is also at least ϵ, and finally the probability that this improvement is to an element of T is again at least ϵ. Thus the probability of the algorithm terminating at either the next level chosen or the one after that is at least ϵ^3. The number of *pairs* of levels required is therefore stochastically dominated by a geometric random variable with probability of success $\epsilon^3 > 0$. Hence the number of levels is almost surely finite.

The third condition ensures that the number of iterations at any level is finite. This and the previous conditions combine to ensure that the algorithm will almost surely sample from the termination region in finite time.

The algorithm is now formally presented in the more general context.

Backtracking adaptive search

Step 1 Generate Y_0 according to ρ. Set $n = 0$.

Step 2 With probability b_{Y_n} choose Y_{n+1} according to the normalised restriction of ρ to $(-\infty, Y_n)$. With probability w_{Y_n} choose Y_{n+1} according to the normalised restriction of ρ to (Y_n, ∞). Otherwise set $Y_{n+1} = Y_n$.

Step 3 If a stopping criterion is met, stop. Otherwise, increment n and return to Step 2.

This definition of the algorithm is suitable for continuous, discrete or mixed range distributions ρ. Analysis for each of these cases is now presented.

5.3 Factorial moment generating functions for distributions of hitting times

In this section distributions of the number of iterations before convergence are derived for backtracking adaptive search, using factorial moment generating functions. A brief definition of these functions is now given.

The factorial moment generating function of a random variable X is $M(z) = E[z^X]$, if this expectation exists in some interval $1-\epsilon < z < 1+\epsilon$ where ϵ is a positive real number [3]. Then $M^{(r)}(1) = E[X(X-1)(X-2)\dots(X-r+1)]$. This property can be used to derive moments of the distribution of X.

The key result required is the distribution of the number of iterations between improvements in the best level seen. Define a record as a level Y_k better than any preceding level, so $Y_k < Y_n$ for all $n < k$. A record Y_k is said to be current from iteration k until a new record is sampled. The number of records encountered by backtracking adaptive search before reaching a level no greater than some fixed level y is almost surely finite, owing to the conditions described in the previous section. Denote these records, ordered from largest to smallest, by $Y_{(1)}, Y_{(2)}, Y_{(3)}, \dots, Y_{(R)}$; then if $N_{(i)}$ denotes the delay at record $Y_{(i)}$, the total number of iterations before first reaching a level no greater than y is given by

$$N(y) = \sum_{i=1}^{R} N_{(i)}. \tag{5.1}$$

Note that the delay $N_{(i)}$ is the number of iterations for which $Y_{(i)}$ is current.

The distribution of these delays is established in Subsection 5.3.1. The following subsections then use this result to derive the distribution of the number of iterations before convergence of backtracking adaptive search in each of the cases where the range distribution ρ is countable, finite, continuous or finally mixed.

5.3.1 Delay distributions

The distribution of the delay at level $Y_{(i)}$ is that of $N(Y_{(i)}^-)$ given that $Y_0 = Y_{(i)}$, where $N(y^-) = \min\{n : Y_n < y\}$. This follows from the fact that backtracking adaptive search is Markovian. The distribution is provided in the following theorem.

Theorem 5.3.1 *Let $D(y,z) = E[z^{N(y^-)}|Y_0 = y]$ be the factorial moment generating function of the delay distribution at level y. Then*

$$D(y,z) = \frac{z\,(b_y p_y(1-p_y) + w_y(p_y - \pi_y)(M(y,z) - p_y))}{p_y(1-p_y)(1 - z(1 - b_y - w_y)) - z\pi_y w_y(M(y,z) - p_y)}$$

where $p_y < 1$ and $M(y,z) = E[z^{N(y)}]$ is the factorial moment generating function of $N(y)$.

Proof An expression is first required for $E[z^{N(y^-)}|Y_0 > y]$. Note that

$$\begin{aligned} M(y,z) &= E[z^{N(y)}] \\ &= P(Y_0 \le y)E[z^{N(y)}|Y_0 \le y] + P(Y_0 > y)E[z^{N(y)}|Y_0 > y] \\ &= p_y + (1-p_y)E[z^{N(y)}|Y_0 > y] \end{aligned} \tag{5.2}$$

since $N(y) = 0$ when $Y_0 \le 0$. Also

$$\begin{aligned} E[z^{N(y^-)-N(y)}|Y_0 > y] &= P(Y_{N(y)} < y)E[z^{N(y^-)-N(y)}|Y_0 > y > Y_{N(y)}] \\ &\quad + P(Y_{N(y)} = y)E[z^{N(y^-)-N(y)}|Y_0 > y = Y_{N(y)}] \\ &= \frac{p_y - \pi_y}{p_y} + \frac{\pi_y D(y,z)}{p_y} \end{aligned} \tag{5.3}$$

using the fact that the search algorithm is memoryless. Therefore, since the number of iterations before achieving a record is independent of the number of subsequent iterations before bettering it,

$$\begin{aligned} E[z^{N(y^-)}|Y_0 > y] &= E[z^{N(y)}|Y_0 > y]E[z^{N(y^-)-N(y)}|Y_0 > y] \\ &= \frac{M(y,z) - p_y}{1 - p_y}\left(\frac{p_y - \pi_y(1 - D(y,z))}{p_y}\right), \end{aligned} \tag{5.4}$$

substituting expressions for $E[z^{N(y)}|Y_0 > y]$ and $E[z^{N(y^-)-N(y)}|Y_0 > y]$ from Equations (5.2) and (5.3) respectively.

Now define h as the first iteration number where Y_h differs from Y_0. Then

$$\begin{aligned} E[z^h|Y_0 = y] &= \sum_{i=1}^{\infty} P(h=i)z^i \\ &= \sum_{i=1}^{\infty}(b_y + w_y)(1 - b_y - w_y)^{i-1}z^i \\ &= \frac{z(b_y + w_y)}{1 - z(1 - b_y - w_y)} \end{aligned} \tag{5.5}$$

where $-1 - \epsilon < z < 1 + \epsilon$. The proof now follows, since the length of the initial hesitation and the number of iterations before convergence thereafter are independent:

$$\begin{aligned} D(y,z) &= E[z^{N(y^-)}|Y_0 = y] \\ &= E[z^h|Y_0 = y]E[z^{N(y^-)-h}|Y_0 = y] \\ &= E[z^h|Y_0 = y]\Big(P(Y_h < y)E[z^{N(y^-)-h}|Y_h < Y_0 = y] \\ &\quad + P(Y_h > y)E[z^{N(y^-)-h}|Y_h > Y_0 = y]\Big) \\ &= E[z^h|Y_0 = y]\left(\frac{b_y}{b_y + w_y} + \frac{w_y E[z^{N(y^-)}|Y_0 > y]}{b_y + w_y}\right) \end{aligned}$$

using the fact that the algorithm is memoryless. The final result may be obtained by substituting the expressions for $E[z^h|Y_0 = y]$ and $E[z^{N(y^-)}|Y_0 > y]$ given in Equations (5.5) and (5.4) respectively. ■

The distribution of the delays at each record may now be used to find the overall factorial moment generating function for the number of iterations before convergence for backtracking adaptive search. In the following subsections these results are derived separately for discrete, continuous and general range distributions.

5.3.2 Discrete range distribution

In this subsection a factorial moment generating function is given for the number of iterations before convergence for backtracking adaptive search where the range distribution is discrete. The mean and variance of this distribution are derived in the following subsection, in the case where the range distribution is not only countable but finite.

Since the range distribution is discrete, there is a countable number of points of discontinuity. Let I_y be the set of points of discontinuity having a level greater than y.

Theorem 5.3.2 *For discrete backtracking adaptive search, the factorial moment generating function of $N(y)$ is*

$$M(y,z) = \prod_{t \in I_y} \frac{p_t - \pi_t(1 - D(t,z))}{p_t}$$

where $p_y < 1$ and $D(t,z) = E[z^{N(t^-)}|Y_0 = t]$ is the factorial moment generating function of the delay distribution at level t.

Proof Define N_t to be the number of iterations for which t is a record; if level t is never a record then $N_t = 0$. Then

$$\begin{aligned} M(y,z) &= E[z^{N(y)}] \\ &= E[z^{\sum_{t \in I_y} N_t}] \\ &= \prod_{t \in I_y} E[z^{N_t}] \qquad (5.6) \end{aligned}$$

since the numbers of iterations for which each level is a record are independent of each other. Now

$$\begin{aligned} E[z^{N_t}] &= P(Y_{N(t)} < t)E[z^{N_t}|Y_{N(t)} < t] + P(Y_{N(t)} = t)E[z^{N_t}|Y_{N(t)} = t] \\ &= \frac{p_t - \pi_t}{p_t} + \frac{\pi_t D(t,z)}{p_t} \qquad (5.7) \end{aligned}$$

since the search algorithm is memoryless. Substituting this into Equation (5.6) yields the result. ■

5.3.3 Finite range distribution

When there is a finite number of range levels y for which $\pi_y > 0$, the factorial moment generating function for $N(y)$ can be used to find the mean of the distribution, as derived in Section 4.3.

Without loss of generality, label the finite sequence of range levels (in increasing order) as $1, 2, \ldots, m$. Theorem 5.3.2 can then be stated as

$$M(y, z) = \prod_{i=y+1}^{m} \frac{p_i - \pi_i(1 - D(i, z))}{p_i}$$

The factorial moment generating function is now expressed as a difference equation, using Theorem 5.3.1:

$$M(y-1, z) = M(y, z)\left(A + \frac{zB + zM(y, z)C}{D + zE - zM(y, z)F}\right),$$

where $A = p_{y-1}/p_y$, $B = \pi_y p_y (b_y(1 - p_y) - w_y p_{y-1})$, $C = \pi_y p_{y-1} w_y$, $D = p_y^2(1 - p_y)$, $E = p_y^2(\pi_y w_y - (1 - p_y)(1 - b_y - w_y))$ and $F = \pi_y p_y w_y$, for $y \in \{2, 3, \ldots, m-1\}$. (Note that $M(y, 1) = 1$ for any level y, and hence the expression in parentheses above has the value 1 when $z = 1$.)

Differentiation of both sides with respect to z yields

$$\begin{aligned} M'(y-1, z) &= M'(y, z)\left(A + \frac{zB + zM(y, z)C}{D + zE - zM(y, z)F}\right) \\ &\quad + M(y, z)\left(\frac{B + M(y, z)C + zM'(y, z)C)}{D + zE - zM(y, z)F}\right. \\ &\quad \left. - \frac{(zB + zM(y, z)C)(E - M(y, z)F - zM'(y, z)F)}{(D + zE - zM(y, z)F)^2}\right) \end{aligned}$$

for $y \in \{2, 3, \ldots, m-1\}$. Setting $z = 1$ gives

$$E[N(y-1)] = \left(1 + \frac{C(D + E) + BF}{(D + E - F)^2}\right) E[N(y)] + \frac{(B + C)D}{(D + E - F)^2}.$$

Substituting the expressions for A,B,C,D,E and F and rearranging now yields the recursion for $E[N(y)]$ of Section 4.3. (The special cases are dealt with in that chapter also.)

The factorial moment generating function also allows calculation of higher moments.

Theorem 5.3.3 *For finite backtracking adaptive search, $E[(N(y))^2 - N(y)]$ satisfies*

i) For all m, $E[(N(m))^2 - N(m)] = 0$.

ii) For $m > 1$, $E[(N(m-1))^2 - N(m-1)] = 2\pi_m(1 - b_m)/b_m^2$.

iii) For $m > 2$, $E[(N(y-1))^2 - N(y-1)]$ is given by

$$\begin{aligned}
&\frac{p_y(\pi_y w_y + (1-p_y)(b_y+w_y))}{(1-p_y)(p_y b_y + p_{y-1}w_y)} E[(N(y))^2 - N(y)] \\
&+ \frac{2\pi_y p_y w_y(\pi_y w_y + (1-p_y)(b_y+w_y))}{(1-p_y)^2(p_y b_y + p_y - \pi_y w_y)^2}(E[N(y)])^2 \\
&+ \frac{2\pi_y p_y (b_y(1-p_y) + w_y(2-p_{y-1}))}{(1-p_y)(p_y b_y + p_{y-1}w_y)^2} E[N(y)] \\
&- \frac{2\pi_y(p_y(b_y-1) + p_{y-1}w_y)}{(p_y b_y + p_{y-1}w_y)^2}
\end{aligned}$$

for all $y \in \{2, 3, \ldots, m-1\}$.

Proof Since $Y_0 \le m$, $E[N(m)] = E[(N(m))^2] = 0$, and thus $E[(N(m))^2 - N(m)] = 0$. For $m > 1$, $N(m-1) > 0$ only if $Y_0 = m$; $N(m-1)$ is then the number of iterations until the algorithm leaves level m. Since the probability of leaving level m at any iteration is b_m, $E[N(m-1)] = \pi_m/b_m$ and $E[(N(m-1))^2] = \pi_m(2-b_m)/b_m^2$, whence $E[(N(m-1))^2 - N(m-1)] = 2\pi_m(1-b_m)/b_m^2$.

Differentiating both sides of the factorial moment generating function again with respect to z shows that $M''(y-1, z)$ is

$$\begin{aligned}
&M''(y,z)\left(A + \frac{zB + zM(y,z)C}{D + zE - zM(y,z)F}\right) \\
&+2M'(y,z)\left(\frac{B + M(y,z)C + zM'(y,z)C)}{D + zE - zM(y,z)F} - \frac{(zB + zM(y,z)C)(E - M(y,z)F - zM'(y,z)F)}{(D + zE - zM(y,z)F)^2}\right) \\
&+ M(y,z)\left(\frac{2M'(y,z)C + zM''(y,z)C}{D + zE - zM(y,z)F} + \frac{(zB + zM(y,z)C)(2M'(y,z)F + zM''(y,z)F)}{(D + zE - zM(y,z)F)^2}\right. \\
&- 2(E - M(y,z)F - zM'(y,z)F)\left(\frac{B + M(y,z)C + zM'(y,z)C}{(D + zE - zM(y,z)F)^2}\right. \\
&\left.\left.- \frac{(zB + zM(y,z)C)(E - M(y,z)F - zM'(y,z)F)}{(D + zE - zM(y,z)F)^3}\right)\right)
\end{aligned}$$

for $y \in \{2, 3, \ldots, m-1\}$, recalling the expressions for A,B,C,D,E and F given above. Setting $z = 1$

now shows that $E[(N(y-1))^2 - N(y-1)]$ is

$$\begin{aligned}\left(1+\frac{C(D+E)+BF}{(D+E-F)^2}\right) E[(N(y))^2-N(y)] \\ +\frac{2(C(D+E)+BF)(D+E)}{(D+E-F)^3}(E[N(y)])^2 \\ +\frac{2D(BF+(B+2C)(D+E))}{(D+E-F)^3}E[N(y)]-\frac{2(E-F)(B+C)D}{(D+E-F)^3}\end{aligned}$$

Substituting the expressions for A,B,C,D,E and F and rearranging provides the desired result. ■

In Section 4.3 a closed form expression for $E[N(y)]$ is derived, using theory of difference equations [33, p. 153]. The same theory now allows the above difference equation to be solved, treating $E[N(y)]$ as known. Thus the following corollary is derived.

Corollary 5.3.1 *The variance in the number of iterations of finite backtracking adaptive search before reaching level y,* $Var[N(y)]$*, satisfies*

i) For all m, $Var[N(m)] = 0$.

ii) For $m > 1$, $Var[N(m-1)] = \pi_m(2-b_m-\pi_m)/b_m^2$.

iii) For $m > 2$, $Var[N(y)]$ *is given by*

$$\left(\frac{2\pi_m(1-b_m)}{b_m^2}+\sum_{j=y+1}^{m-1}\left(B(j)\prod_{i=j}^{m-1}\frac{1}{A(i)}\right)\right)\prod_{i=y+1}^{m-1}A(i) \\ +E[N(y)](1-E[N(y)])$$

for all $y \in \{1,2,\ldots,m-2\}$*, where* $A(i) = \frac{p_i(\pi_i w_i+(1-p_i)(b_i+w_i))}{(1-p_i)(p_i b_i+p_{i-1}w_i)}$ *and*
$B(i) = \frac{2\pi_i p_i w_i(\pi_i w_i+(1-p_i)(b_i+w_i))}{(1-p_i)^2(p_i b_i+p_{i-1}w_i)^2}(E[N(i)])^2+\frac{2\pi_i p_i(b_i(1-p_i)+w_i(2-p_{i-1}))}{(1-p_i)(p_i b_i+p_{i-1}w_i)^2}E[N(i)]-\frac{2\pi_i(p_i(b_i-1)+p_{i-1}w_i)}{(p_i b_i+p_{i-1}w_i)^2}.$

Proof The difference equation of Theorem 5.3.3 can be written as follows,

$$E[(N(y-1))^2-N(y-1)] - A(y)E[(N(y))^2-N(y)] = B(y), \tag{5.8}$$

with $A(y)$ and $B(y)$ as defined above.

The variances of $N(m)$ and $N(m-1)$ are found as in the theorem, making use of the relationship $Var[N(y)] = E[(N(y))^2] - (E[N(y)])^2$. The remainder of the proof also lies in applying this relationship, using a closed form expression for $E[(N(y))^2 - N(y)]$ found by solving Equation (5.8). The method of solution is exactly that employed in Section 4.3. ■

This corollary and Corollary 4.3.1 now provide closed expressions for the mean and variance of the number of iterations before convergence for BAS with finite range distributions. The variances of the numbers of iterations before convergence for hesitant adaptive search and pure adaptive search with finite range distributions are available as special cases.

Setting $w_y = 0$ in Corollary 5.3.1 for all y and simplifying yields the variance of the number of iterations before convergence for hesitant adaptive search with a finite range distribution:

$$Var[N(y)] = \sum_{j=y+1}^{m} \frac{\pi_j(p_j(2-b_j)-\pi_j)}{b_j^2 p_j^2}$$

for all $y \in \{1,2,\ldots,m-1\}$ where $m > 1$, with $Var[N(m)] = 0$ for all m. This result is given in [51] (noting that the summation is to be taken over transient range levels only).

Further substituting $b_y = 1$ for all y gives the variance of the number of iterations before convergence for pure adaptive search with a finite range distribution:

$$Var[N(y)] = \sum_{j=y+1}^{m} \frac{\pi_j p_{j-1}}{p_j^2}$$

for all $y \in \{1,2,\ldots,m-1\}$ where $m > 1$, with $Var[N(m)] = 0$ for all m.

5.3.4 Continuous range distribution

In this subsection a factorial moment generating function is found for the number of iterations before convergence for backtracking adaptive search where the range distribution is continuous. The mean of this distribution is then derived.

The derivation of the factorial moment generating function rests on the following result concerning the distribution of record values for backtracking adaptive search.

Lemma 5.3.1 *Record values of backtracking adaptive search with a continuous range distribution form a Poisson process with mean value function $m(y) = -\ln p_y$.*

Proof Record values of pure adaptive search with a continuous range distribution are shown to form a Poisson process with mean value function $m(y) = -\ln p_y$ in [54, Theorem 4.2, Corollary 5.1]. (The function stated there includes the absorbing iteration and is thus $1 - \ln p_y$.) The proof then follows by observing that record values for pure adaptive search and backtracking adaptive search with identical continuous range distribution are identically distributed. ∎

The following theorem may now be derived.

Theorem 5.3.4 *The factorial moment generating function of $N(y)$ for backtracking adaptive search with a continuous range distribution is*

$$M(y, z) = e^{\int_{t=y}^{\infty} \frac{D(t,z)-1}{p_t} dp_t}$$

where $p_y < 1$ and $D(t, z) = E[z^{N(t^-)}|Y_0 = t]$ is the factorial moment generating function of the delay distribution at level t.

Proof Note first that $M(y, z)$ can be written as

$$E[z^{N(y)}] = E[z^{\sum_{i=1}^{R} N_{(i)}}],$$

using Equation (5.1). Conditioning on the number R and values $Y_{(1)}, Y_{(2)}, Y_{(3)}, \ldots, Y_{(R)}$ of records visited by backtracking adaptive search before convergence then gives

$$\begin{aligned} M(y, z) &= E_R[E_{Y_{(1)},Y_{(2)},Y_{(3)},\ldots,Y_{(R)}}[E[z^{\sum_{i=1}^{R} N_{(i)}}]]] \\ &= E_R[E_{Y_{(1)},Y_{(2)},Y_{(3)},\ldots,Y_{(R)}}[\prod_{i=1}^{R} E[z^{N_{(i)}}]]] \end{aligned}$$

since, given the values of all records, the delays at each record are independent. Substituting $D(y, z) =$

$E[z^{N(y)}]$ gives

$$M(y,z) = E_R[E_{Y_{(1)},Y_{(2)},Y_{(3)},\ldots,Y_{(R)}}[\prod_{i=1}^{R} D(Y_{(i)},z)]].$$

Lemma 5.3.1 shows that records occur as a Poisson process with mean $m(y)$; their values given the total number R are therefore independent identically distributed random variables in (y,∞) with common cumulative distribution function $F(t) = 1 - m(t)/m(y)$ [28, p22]. Let T denote this common random variable for record values. Then

$$\begin{aligned} M(y,z) &= E_R[\prod_{i=1}^{R} E_T[D(Y_{(i)},z)]] \\ &= E_R[(E_T[D(t,z)])^R] \\ &= e^{m(y)(E_T[D(t,z)]-1)} \end{aligned}$$

since R has a Poisson distribution with mean $m(y)$ and the expected delay at any level is finite.

Now since $F(t) = 1 - m(t)/m(y)$ and Lemma 5.3.1 gives $m(t) = -\ln p_t$, differentiation of $F(t)$ with respect to p_t leads to

$$\begin{aligned} \frac{dF(t)}{dp_t} &= \frac{d}{dp_t}(1 + \frac{\ln p_t}{m(y)}) \\ &= \frac{1}{p_t m(y)}, \end{aligned} \tag{5.9}$$

and the proof follows:

$$\begin{aligned} M(y,z) &= e^{E_T[m(y)(D(t,z)-1)]} \\ &= e^{\int_{t=y}^{\infty} m(y)(D(t,z)-1)dF(t)} \\ &= e^{\int_{t=y}^{\infty} \frac{D(t,z)-1}{p_t} dp_t} \end{aligned}$$

by Equation (5.9), as required. ∎

This expression can be solved for $M(y,z)$ when p_y is an absolutely continuous function, as shown in the following corollary.

Corollary 5.3.2 *The factorial moment generating function of $N(y)$ for backtracking adaptive search with an absolutely continuous range distribution is*

$$M(y,z) = \frac{e^{-\int_{t_0}^{y} P(t,z)dt}}{e^{-\int_{t_0}^{\infty} P(t,z)dt} + \int_y^\infty Q(u,z)e^{-\int_{t_0}^{u} P(t,z)dt}du}$$

where $p_y < 1$, $P(t,z) = \frac{p_t'(1-p_t)(z-1)-zw_t}{p_t(1-p_t)(1-z(1-b_t-w_t))}$, $Q(u,z) = \frac{-p_t'zw_t}{p_t(1-p_t)(1-z(1-b_t-w_t))}$, t_0 *is an arbitrary integration point and* p_t' *denotes* $\frac{dp_t}{dt}$.

Proof From the statement of Theorem 5.3.4, it follows that

$$\ln M(y,z) = \int_{t=y}^{\infty} \frac{D(t,z)-1}{p_t} dp_t, \tag{5.10}$$

whence, differentiating with respect to y,

$$\frac{1}{M(y,z)} \frac{\partial M(y,z)}{\partial y} = \frac{1-D(y,z)}{p_y}. \tag{5.11}$$

Since p_y is absolutely continuous, $\pi_y = 0$ for all y. Theorem 5.3.1 then gives

$$D(y,z) = \frac{z\,(b_y(1-p_y) + w_y(M(y,z)-p_y))}{(1-p_y)(1-z(1-b_y-w_y))};$$

substituting this expression into Equation (5.11) and rearranging produces

$$\frac{\partial M(y,z)}{\partial y} + P(y,z)M(y,z) = Q(y,z)(M(y,z))^2$$

with $P(y,z)$ and $Q(y,z)$ as defined above. This is an example of Bernoulli's equation, for which the solution is

$$\frac{1}{M(y,z)} e^{-\int_{t_0}^{y} P(t,z)dt} = -\int_{t_0}^{y} Q(u,z)e^{-\int_{t_0}^{u} P(t,z)dt}du + c(z) \tag{5.12}$$

where t_0 is an arbitrary integration point and $c(z)$ is a constant of integration [6, p367].

To find the value of $c(z)$, note that $\lim_{y\to\infty} N(y) = 0$, so that taking the limit as y tends to infinity of Equation (5.12) gives

$$e^{-\int_{t_0}^{\infty} P(t,z)dt} = -\int_{t_0}^{\infty} Q(u,z)e^{-\int_{t_0}^{u} P(t,z)dt} du + c(z).$$

Solving this equation and Equation (5.12) simultaneously provides the result. ■

Substituting $w_y = 0$ for all y in this equation now yields the factorial moment generating function for the number of iterations before convergence for hesitant adaptive search:

$$\begin{aligned} M(y,z) &= e^{-\int_y^{\infty} \frac{p_t'(z-1)}{p_t(1-z(1-b_t))}dt} \\ &= e^{-\int_{t=y}^{\infty} \frac{z-1}{p_t(1-z(1-b_t))}dp_t} \\ &= e^{-\int_{t=y}^{\infty} \frac{p_t m(y)(z-1)}{p_t(1-z(1-b_t))}dF(t)} \end{aligned}$$

by Equation (5.9). Analogously to that equation,

$$\frac{dF(t)}{d\pi_t} = \frac{1}{p_t m(y)}; \tag{5.13}$$

thus

$$M(y,z) = e^{-\int_{t=y}^{\infty} \frac{z-1}{p_t(1-z(1-b_t))}d\pi_t},$$

as presented in [51].

To conclude this subsection, the factorial moment generating function of Theorem 5.3.4 is used to find the expected number of iterations before convergence for backtracking adaptive search with a continuous range distribution, as derived by different means in [9].

Differentiating Equation (5.10) with respect now to z gives

$$\frac{1}{M(y,z)}\frac{\partial M(y,z)}{\partial z} = \int_{t=y}^{\infty} \frac{\partial}{\partial z}\frac{D(t,z)-1}{p_t}dp_t.$$

The expression for $D(t,z)$ can be substituted in from Theorem 5.3.1 and the differentiation carried out. Substituting $z = 1$ then yields

$$\begin{aligned} E[N(y)] &= \int_{t=y}^{\infty} \frac{1 - p_t + w_t E[N(t)]}{(1-p_t)(p_t(b_t + w_t) - \pi_t w_t)} dp_t \\ &= \int_{t=y}^{\infty} \frac{p_t m(y)(1 - p_t + w_t E[N(t)])}{(1-p_t)(p_t(b_t + w_t) - \pi_t w_t)} dF(t) \end{aligned}$$

by Equation (5.9). Equation (5.13) then gives

$$E[N(y)] = \int_{t=y}^{\infty} \frac{1 - p_t + w_t E[N(t)]}{(1-p_t)(p_t(b_t + w_t) - \pi_t w_t)} d\pi_t$$

as given in [9]. That paper continues to solve this equation for the case when p_y is absolutely continuous. (The same result could be obtained using the factorial moment generating function for backtracking adaptive search with an absolutely continuous range distribution, given in Corollary 5.3.2.)

5.3.5 Mixed range distribution

In the general case, the range distribution is a mixture of discrete and continuous components. Again let I_y be the set of points of discontinuity having a level greater than y. The complement of I_y in (y, ∞) is a countable number of open intervals; let U_i denote these intervals and C_y the set of all U_i. The distribution of the number of iterations before convergence for backtracking adaptive search is now derived.

Theorem 5.3.5 *The factorial moment generating function of the number of iterations before convergence for backtracking adaptive search is*

$$M(y,z) = \prod_{i \in I_y} \frac{p_i - \pi_i(1 - D(i,z))}{p_i} \prod_{U_i \in C_y} e^{\int_{t \in U_i} \frac{D(t,z)-1}{p_t} dp_t}$$

where $p_y < 1$ *and* $D(y,z) = E[z^{N(y^-)} | Y_0 = y]$ *is the factorial moment generating function of the delay distribution at level* y.

Proof If the number of iterations with the current record an element of U_i is denoted $N(U_i)$ then the total number of iterations before convergence is

$$N(y) = \sum_{i \in I_y} N_i + \sum_{U_i \in C_y} N(U_i)$$

where all $N(U_i)$ and N_i are independent of each other, since the number of iterations between reaching a level or set of levels and reaching a better level is unrelated to the progress of the algorithm before or after. Then

$$\begin{aligned} M(y,z) &= E[z^{N(y)}] \\ &= E[z^{\sum_{i \in I_y} N_i + \sum_{U_i \in C_y} N(U_i)}] \\ &= \prod_{i \in I_y} E[z^{N_i}] \prod_{U_i \in C_y} E[z^{N(U_i)}]. \end{aligned} \tag{5.14}$$

Now since the derivation of $E[z^{N_i}]$ in Equation (5.7) is valid also for a general range distribution, it remains only to determine an expression for $E[z^{N(U_i)}]$. This will be done analogously to the proof of Theorem 5.3.4.

Define a_i and b_i so that $U_i = (a_i, b_i)$ for each bounded interval U_i. Let R_i be the number of records that fall in U_i. The number of records reached before convergence to level y has a Poisson distribution with mean $-\ln p_y$; therefore the number of records falling in U_i has a Poisson distribution with mean $\ln p_{b_i^-} - \ln p_{a_i}$, where $p_{b_i^-}$ refers to $p_{b_i} - \pi_{b_i}$. The values of records falling in U_i given their number R_i are thus independent identically distributed random variables in (a_i, b_i) with common cumulative distribution function $F(t) = 1 - (\ln p_{b_i^-} - \ln p_t)/(\ln p_{b_i^-} - \ln p_{a_i})$. As in the proof of Theorem 5.3.4, then,

$$\begin{aligned} M(y,z) &= E_{R_i}[(E_T[D(t,z)])^{R_i}] \\ &= e^{(\ln p_{b_i^-} - \ln p_{a_i})(E_T[D(t,z)]-1)} \end{aligned}$$

since the expected delay at any level is finite.

Differentiating $F(t)$ with respect to p_t gives $dF(t)/dp_t = 1/(p_t(\ln p_{b_i^-} - \ln p_{a_i}))$; thus

$$\begin{aligned} M(y,z) &= e^{E_T[(\ln p_{b_i^-} - \ln p_{a_i})(D(t,z)-1)]} \\ &= e^{\int_{t \in U_i} (\ln p_{b_i^-} - \ln p_{a_i})(D(t,z)-1) dF(t)} \\ &= e^{\int_{t \in U_i} \frac{D(t,z)-1}{p_t} dp_t}. \end{aligned}$$

This expression and that in Equation (5.7) can now be substituted into Equation (5.14) to complete the proof. ■

This theorem provides the factorial moment generating function for the number of iterations before convergence for backtracking adaptive search with a general range distribution; the results in Subsections 5.3.2 and 5.3.4 are thus special cases of this result.

5.4 Summary

This chapter is a theoretical digression in the thesis, included for the state of completeness; a general theory for backtracking adaptive search is presented. Factorial moment generating functions for the number of iterations before convergence for backtracking adaptive search with discrete, continuous or mixed range distributions are derived, and solved for the finite and absolutely continuous cases. All results previously published on the number of iterations before convergence for hesitant adaptive search and pure adaptive search are special cases of the results of this chapter.

The following chapter now returns to the problem of approximating the expected number of iterations before convergence for a general stochastic global optimisation algorithm, taking up the theory of Chapter 4 and applying it to some examples.

Chapter 6

Computational Results

6.1 Introduction

In this chapter the prediction mechanism is implemented on some small examples. Each step in the framework can then be seen in practice. The usefulness of this method of modelling the movement of a search algorithm in the range is then illustrated, at least on certain examples.

Future work may be directed at improving the method by which some of the steps in the framework are implemented. Another important area for future work is to assess how well the initial behaviour of the range process reflects later behaviour. The implicit assumption made in estimating the convergence time of a process based only on a finite observation period is that the observed behaviour is characteristic of unobserved behaviour in the future and in regions of the domain that have not previously been observed. It may be possible to link the degree to which this assumption is justified with certain characteristics of the problem class and algorithm type applied. A third avenue for investigation is the extent to which the approach is affected by the "curse of dimensionality". The approach outlined is not directly dependent on the dimensionality of the domain, but it may be the case that the quality of the estimate of convergence time is affected by the number of variables in the problem. The aim in this chapter is simply to demonstrate the ideas on some examples. No statistical inference on the general effectiveness of the method is to be drawn from the few examples considered; rather, an illustration of the approach is provided.

All programs have been written using Matlab software. These are included as an appendix to the thesis. Electronic versions are available from the author on request.

The layout of Section 6.2 is similar to that of Section 2.2. Two disparate examples are introduced and the method for predicting expected convergence times is carried out on each. Details relating to the implementation are discussed at each stage. Section 6.3 then considers issues relating to actual implementation of the approximation method in practice. Two examples demonstrate aspects of the prediction method. The final section summarises the effectiveness of the approximation framework as implemented in this chapter. The potential for further development is also discussed.

6.2 Approximating two stochastic global optimisation algorithms

Two simple examples of stochastic global optimisation algorithms are analysed in this section, in order to illustrate the approximation algorithms described in Chapter 2. The first is a small example for which transition matrices in the domain and range can conveniently be written out in full; the second is a more practical algorithm on a more testing domain, demonstrating how the kind of analysis introduced in this thesis can be used to estimate the convergence time of a real algorithm applied to an unknown but complicated domain.

Note that the examples are presented purely to provide a numerical implementation of the theory that has been developed and exemplified earlier in the thesis. Their purpose is to demonstrate the degree of accuracy of the method as it has been presented. The successive stages in the approximation process as it is applied to each example are described in the following subsections, beginning with the domain process and continuing through range, averaged range and asymptotic averaged range processes to the final backtracking adaptive search approximation.

6.2.1 The domain process

The two examples are now introduced, defining a problem and an optimisation algorithm in the domain.

Example 1 A Markovian search algorithm is applied to a very simple problem with six points in the domain S, labelled $\{1, 2, 3, 4, 5, 6\}$. Suppose that the first two points have objective function value 1, the next two have objective function value 2 and points numbered 5 and 6 have objective function values 3 and 4 respectively. The aim is to minimise objective function value. Since there are two points with minimal objective function value, they may be combined; the search algorithm stops as soon as it samples either of them. The search algorithm on this set is then described by a Markov domain process with transition matrix P. For this example, we arbitrarily take P to equal

$$\begin{array}{ccc} & & \text{Next state} \\ & \begin{array}{c} \\ 1 \text{ or } 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} & \begin{array}{ccccc} 1 \text{ or } 2 & 3 & 4 & 5 & 6 \\ \end{array} \\ \text{Current state} & \begin{array}{c} 1 \text{ or } 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} & \left[\begin{array}{ccccc} 1 & 0 & 0 & 0 & 0 \\ 0.2 & 0.3 & 0.2 & 0.1 & 0.2 \\ 0.1 & 0.2 & 0.3 & 0.2 & 0.2 \\ 0 & 0.2 & 0.2 & 0.4 & 0.2 \\ 0.2 & 0.4 & 0.1 & 0.1 & 0.2 \end{array}\right]. \end{array}$$

If the first iterate is chosen randomly from a uniform distribution on the six possibilities (so $\delta_0 = [\frac{2}{6}\ \frac{1}{6}\ \frac{1}{6}\ \frac{1}{6}\ \frac{1}{6}]$), then standard Markov Chain theory [27] gives the expected number of iterations before convergence for the process as 5.049.

Example 2 A Markovian search algorithm is used to minimise an unknown function on the domain S consisting of $0 \leq x, y \leq \pi$. The search algorithm to be used in this example will only accept successive iterates if they are not worsening or with a constant probability if they are worsening. Successive candidate solutions are generated using the Hit-and-Run algorithm of [46] (discussed on p. 7).

In order to apply the full approximation framework, it is necessary to define the objective function and the domain transition matrix of the algorithm; in practice, of course, transition probabilities to the optimal state are unknown until the optimal state has been sampled, so an empirical approach may be necessary. As suggested in Chapter 2, the asymptotic averaged range process can more practically be estimated directly by observing progress of the search algorithm on the range, but in this section the aim is to illustrate each stage of the framework, comparing the expected convergence times of each approximation in the sequence. Section 6.3 discusses a more practical approach to the estimation of expected convergence times for black box algorithms and objective functions. (The term "black box algorithm" is applied here to any global optimisation algorithm that is applied to a problem in order to produce solutions, where in general there is no knowledge of how the algorithm produces these solutions. A unique feature of the prediction method described in this thesis is its applicability to a general Markovian algorithm with no requirement on the structure of the algorithm.)

The objective function used is $-2.5\sin(x)\sin(y) - \sin(5x)\sin(5y)$, as used in [53]; this can be shown analytically to have a global minimum at $(x, y) = (\pi/2, \pi/2)$. This objective function is chosen because it has several local minima, as shown in Figure 6.1. The algorithm is of course applied to the problem as though the objective function is unknown, so that algorithm results can be compared with the known global minimum.

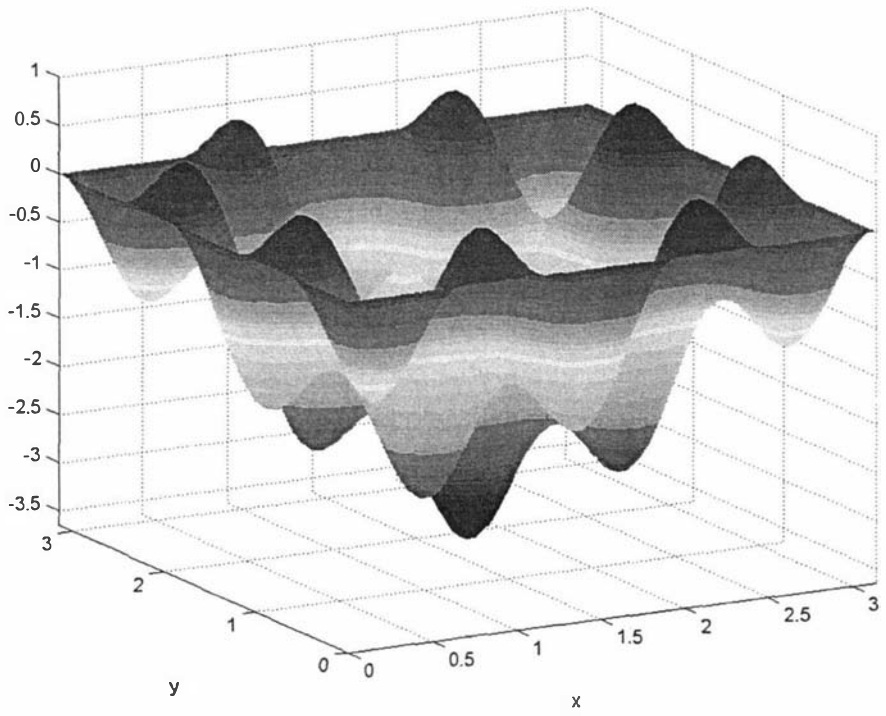

Figure 6.1: Objective function $-2.5\sin(x)\sin(y) - \sin(5x)\sin(5y)$ used in Example 2.

The search is conducted over a grid with mesh size $\pi/50$. The domain transition matrix P is thus a $51^2 \times 51^2$ matrix. The algorithm can now be implemented as a Markov chain. Each successive Hit-and-Run iterate is rounded to the nearest grid point. At each iteration, improving points are accepted with probability one and non-improving points are accepted with probability 0.1. The algorithm continues in this way until the global optimum is sampled. The number of iterations before sampling the optimum can thus be estimated. It is clear that the algorithm thus described makes no use of the analytic form of the objective function.

Suppose the initial distribution δ_0 is uniform on the domain. The domain transition matrix P can be constructed using the structure of the Hit-and-Run algorithm. (The matrix P used in this example is calculated only approximately, to simplify computation. Since the aim here is simply to construct a transition matrix, this discrepancy is immaterial.) Standard matrix theory then provides the expected number of iterations before convergence for this algorithm as 1199.53.

6.2.2 The range process

The range processes for each example are now identified.

Example 1 (continued) The range process is defined as the image of the domain process projected onto the range using the objective function. The number of iterations before convergence for the range process is identical with the number of iterations before convergence for the domain process. This range process is not Markovian, since its progress is viewed in the range but determined by the algorithm in the domain.

The range process is observed in the range, which in this example consists of four possible levels, though the algorithm is actually operating on a domain consisting of five independent states (treating the two optimal points as effectively one state of the process). Some data is thus lost by the simplification of considering only range levels. For instance, an observation of range level 2 could reflect a domain state of either 3 or 4. Observation of the range process alone is therefore insufficient to predict its future behaviour.

Example 2 (continued) The range process is again defined as the image of the domain process in the range. The distribution of the number of iterations before convergence for the range process is thus identical to that of the domain process.

Note again that the process cannot be predicted solely by observation in the range; it effectively "runs" in the problem domain. The range process is merely an observation of the domain process projected into a single dimension. Several domain points share objective function values; the number of range levels is only 326. As before, though, the range process is not Markovian on these range levels. (For instance, the probability of moving to the optimum from range level 0 is 0.00021 if the preceding range level was -0.8637, but 0.00022 if the preceding range level was -2.2613.)

6.2.3 The averaged range process

The range process for each of the two examples is now approximated by the averaged range process, a time-inhomogeneous Markov process in the range. The method used is that of Subsection 2.2.1.

Example 1 (continued) The objective function mapping matrix for this example is

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Equation (2.3) of Subsection 2.2.1 states that $\gamma_n(i) = \dfrac{\delta_n(i)}{\sum\limits_{\{j:f(x_j)=f(x_i)\}} \delta_n(j)}$, where $\delta_n = \delta_0 P^n$. Using the value of δ_0 provided above, $\delta_1 = [\frac{5}{12}\ \frac{11}{60}\ \frac{2}{15}\ \frac{2}{15}\ \frac{2}{15}]$ and $\delta_2 = [0.49\ 0.16\ 0.12\ 0.11\ 0.12]$. Since the second and third states in P share a level, $\gamma_0 = [1\ 0.5\ 0.5\ 1\ 1]$, $\gamma_1 = [1\ 0.579\ 0.421\ 1\ 1]$ and $\gamma_2 = [1\ 0.581\ 0.419\ 1\ 1]$.

Equation (3.1) then gives $R_n = M^{\mathrm{T}}\mathrm{diag}(\gamma_n)PM$, so

$$R_0 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.15 & 0.5 & 0.15 & 0.2 \\ 0 & 0.4 & 0.4 & 0.2 \\ 0.2 & 0.5 & 0.1 & 0.2 \end{bmatrix}, \quad R_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.1579 & 0.5 & 0.1421 & 0.2 \\ 0 & 0.4 & 0.4 & 0.2 \\ 0.2 & 0.5 & 0.1 & 0.2 \end{bmatrix}$$

$$\text{and} \quad R_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.1581 & 0.5 & 0.1419 & 0.2 \\ 0 & 0.4 & 0.4 & 0.2 \\ 0.2 & 0.5 & 0.1 & 0.2 \end{bmatrix}.$$

Corollary 3.2.1 shows that N_a, the number of iterations before convergence for the process defined in this way, has the same distribution as that of the original process. The mean of this distribution is thus 5.049 iterations, the same as the expectation determined on p. 106.

Example 2 (continued)

The objective function mapping matrix M can be constructed from the objective function f using the definition of Subsection 2.2.1. Averaged range process transition matrices are then available using Equations (2.3) and (3.1). Since there are 326 range levels, the averaged range process transition matrices are too large to write out here in full. The top left corner of R_0 (calculated using Matlab) is

$$R_0 = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots \\ 0.0062 & 0.3433 & 0.0022 & 0.0017 & \cdots \\ 0.0488 & 0.0210 & 0.4599 & 0.0253 & \cdots \\ 0.0031 & 0.0171 & 0.1014 & 0.3899 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \end{bmatrix};$$

This part of the matrix R_n remains identical to four decimal places for all n.

Again, Corollary 3.2.1 shows that the averaged range process now defined has the same expected number of iterations before convergence as the domain process, so $E[N_a] = 1199.53$.

6.2.4 The asymptotic averaged range process

A time-homogeneous Markov approximation to the averaged range process is now found for the two examples.

Example 1 (continued)

The limit of R_n as n tends to infinity can be found using Theorem 3.3.3 to be

$$R = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.1582 & 0.5 & 0.1418 & 0.2 \\ 0 & 0.4 & 0.4 & 0.2 \\ 0.2 & 0.5 & 0.1 & 0.2 \end{bmatrix}$$

This limit can also be found directly from the averaged range process transition matrices; R_n is identical with R to four decimal places when $n > 2$.

Since this example is small, the expected number of iterations before convergence for the asymptotic averaged range process approximation, $E[N_b]$, can be found directly using standard matrix theory [27] to be 5.024. This underestimates the true value by 0.025 iterations, or 0.5%. The relative error in this stage of the approximation process is usually small, as discussed in Section 3.4.

Example 2 (continued)

Theorem 3.3.3 again allows the limit of R_n as n tends to infinity, R, to be calculated for this example. It is also possible to obtain this result by calculating several values of R_n; for $n > 5$ these matrices are identical with R to four decimal places. The top left corner of R is the same to four decimal places as the top left corner of R_0 displayed in the previous subsection.

Clearly, R will be very large when the grid used is very fine. In the present case, though, it is possible to find the expected convergence time of the asymptotic averaged range process $E[N_b]$ directly by matrix inversion. The value, 1199.54, overestimates the true value by 0.005 iterations, or 0.0004%. The relative error is thus very small in this case; the Markov process in the range with $51^2 = 2601$ states has a convergence time almost exactly the same as that in the range with only 326 levels. This reduction in size and complexity of the problem could be used in some cases to provide analysis of the expected convergence time for a process with a domain transition matrix too large for direct analysis. If an accurate estimate of R can be obtained, then (in this example at least) an accurate estimate of the number of iterations before convergence for the domain process can be found. For many problems, though, even the asymptotic averaged range process transition matrix R will be too large for direct analysis; a method of approximating the convergence time for a Markov process in the range is thus required. The following subsection uses a backtracking adaptive search approximation for this purpose; this is just one of several methods that could be used.

6.2.5 Backtracking adaptive search

In the last stage of approximation, a backtracking adaptive search algorithm approximates the asymptotic averaged range process. The expected number of iterations before convergence for this approximation can then be found; this is an estimate of the expected number of iterations required by the original algorithm.

Example 1 (continued)

Vectors $b = [0\ 0.1582\ 0.4\ 0.8]$ and $w = [0\ 0.3418\ 0.2\ 0]$ can easily be found from R, using Equations (4.8) and (4.9). The limiting relative weightings of transient domain states are $\bar{\beta} = [0.32\ 0.23\ 0.22\ 0.23]$. This vector can be found either empirically or by Equation (4.10). Finally, Equation (4.11) gives $\pi = [0.23\ 0.34\ 0.17\ 0.26]$.

The expected number of iterations before convergence for the backtracking adaptive search approximation can be found using Corollary 4.4.1 to be 3.709. This underestimates the true value by 1.34 iterations, or 26.5%. The last stage in the approximation process is currently the least refined; the relative error in the final approximation is quite large. Improving the accuracy of the backtracking adaptive search approximation may lead to significant reduction in this error. However, the estimate is at least in the same order of magnitude as the true convergence time; some gauge of the likely convergence time is thus provided.

Example 2 (continued)

The same method can also be used to find parameters $b = [0\ 0.0062\ 0.0698\ 0.1217\ \cdots]$, $w =$

[0 0.6505 0.4703 0.4885 $\cdots$] and π = [0.0012 0.0037 0.0038 0.0034 $\cdots$] for the second example. (These are not written in full, for reasons of space.) The expected number of iterations before convergence for the backtracking adaptive search approximation can then be found, using Corollary 4.4.1. The value, 443.5 iterations, underestimates the true value by 756.0 iterations, or 63.0%.

The error in this final approximation is again large. There are two possible reasons for this. Firstly, the way in which a backtracking adaptive search approximation to the asymptotic averaged range process is found may need to be improved. Secondly, the backtracking adaptive search model itself may not be sufficiently flexible to represent adequately the structure of the asymptotic averaged range process. If this is true for some situations then there is no method of approximation that will reliably produce a backtracking adaptive search process with expected convergence time closer to the true value.

It is possible for the present examples to investigate alternative backtracking adaptive search approximations in order to test which of the two postulated reasons is the major cause of error. Improved backtracking adaptive search approximations are now presented for both examples, indicating that for these examples at least the error in the predicted mean convergence time is attributable largely to the method by which the backtracking adaptive search model is fitted to the asymptotic averaged range process and not to an intrinsic inadequacy in the backtracking adaptive search model.

Example 1 (continued)

Define b and w as before, based on the asymptotic averaged range process transition matrix R determined in Subsection 6.2.4. It is possible to find a vector π that maximises the likelihood of observing a backtracking adaptive search process over a finite number of iterations to give the empirical transition probabilities of R. This results in the vector π = [0.14 0.61 0.11 0.15]. The expected number of iterations before convergence for this new backtracking adaptive search approximation can be found using Corollary 4.4.1 to be 5.052. This overestimates the true value by only 0.003 iterations, or 0.06%.

Example 2 (continued)

Applying the same approach, let b and w take the values assigned previously, but adopt a new π = [0.0001 0.0003 0.0004 0.0004 $\cdots$]. The expected number of iterations before convergence for the new backtracking adaptive search approximation can now be found, using Corollary 4.4.1. The value, 1272.6 iterations, overestimates the true value by only 73.0 iterations, or 6.1%.

In both cases the estimates of expected convergence time are greatly improved. The maximum likelihood estimate of π used in both cases takes even longer to evaluate than does finding the expected convergence time of the asymptotic averaged range process directly via matrix inversion. But the whole aim of approximating with backtracking adaptive search is to remove any need for the amount of computational effort required to invert large matrices, so that this is not a practical method. However, it serves to illustrate that in these examples there is a backtracking adaptive search algorithm with expected convergence time that matches fairly closely the true value. This indicates that for these examples at least, the errors in predictions of expected convergence times are due not to insufficient flexibility of the backtracking adaptive search process but to the method used to estimate its parameter π. A refined method of approximating the asymptotic averaged range process with backtracking adaptive search could yield much more accurate results.

The choice of π in the backtracking adaptive search approximation is in itself an optimisation problem. The maximum likelihood approach used above was somewhat successful for these examples; however, this method is not practical when R is large. The simple method described in Chapter 4 is quick to implement but much less accurate in its results. The challenge is then to find an improved method, still capable of execution in reasonable computer time but yielding an improved estimate for π. Improvement of this step would allow the accuracy of the estimate of expected convergence time obtained using the simpler method of Chapter 4 to be significantly improved.

However, even this estimate gives some indication of convergence time. If estimating convergence times for general stochastic global optimisation algorithms is genuinely "hard", perhaps this estimate is as accurate as can be expected by any method. Certainly, errors of this magnitude are commonplace in estimating the runtimes of processes in other fields, for instance software engineering projects and internet downloads. Improving this last stage of approximation remains an avenue for future research.

This completes the analysis of the approximation framework as applied to these two examples. The asymptotic averaged range process approximations in both cases provide very accurate estimates of the number of iterations required before algorithm termination, while the backtracking adaptive search approximations are much less precise.

A further complication encountered in practice is that even the problem specification itself must be estimated. This issue is discussed in the following section.

6.3 Implementation considerations

In this section a method of implementing the approximation framework methodology on a true black box optimisation algorithm and objective function is suggested. The method proposed is designed to illustrate one possible application of the theoretical ideas of this thesis to some realistic problem scenarios.

The examples of the previous section used an explicit statement of the structure of the domain optimisation algorithm given by the transition matrix P. Averaged range process transition matrices R_n, the asymptotic averaged range process transition matrix R and backtracking adaptive search parameters b, w and π were calculated from one another in succession, using P, the objective function mapping matrix M and the initial distribution in the domain δ_0. In practice, only δ_0 can be taken as known; the others require estimation.

Several possibilities present themselves as candidates for this estimation procedure. These are now listed with a description of how each might be implemented.

1. **Domain process start.** One approach is to estimate P and M from a run of the algorithm, allowing the rest of the approximation framework to follow through as in Section 6.2. While M can easily be constructed from its definition in Subsection 2.2.1, the estimation of P requires the observation of a very large number of iterations. A sufficient number of transitions must be observed to estimate the distribution on the domain of transitions from each state or grouping of states in the domain, needed for rows of P. This approach is unlikely to prove computationally efficient unless the domain is very small.

2. **Averaged range process start.** An alternative is to estimate the averaged range process transition matrices R_n directly. This avoids any need to estimate P or M explicitly; Corollary 3.2.1 is invoked so that an unbiased estimator of the mean number of iterations before convergence

for the averaged range process is also an unbiased estimator of the number of iterations of the original process in the domain. To estimate transition matrices R_n for each integer n from zero up to some finite number n_0 requires repeated runs of $n_0 + 1$ iterations of the process, so that an empirical distribution of transitions from each range level at each iteration number no greater than n_0 can be built up. An estimate of the asymptotic averaged range process transition matrix R must then be made by extrapolation of the observed behaviour of the matrices R_n. For instance, R may be obtained by a weighted average of R_n that places more weight on later estimates. A further backtracking adaptive search approximation can then be found, as in the preceding section, allowing an estimate of the expected convergence time to be calculated.

3. **Asymptotic averaged range process start.** A third approach is to estimate the asymptotic averaged range process transition matrix R directly. (A method by which this can be accomplished is outlined below.) The backtracking adaptive search approximation to this process can then be used to estimate its expected convergence time. Section 3.4 discusses reasons to believe that the expected convergence time of the asymptotic averaged range process will often be close to that of the original domain process.

4. **Backtracking adaptive search start.** Finally, it is possible to estimate the parameters of backtracking adaptive search immediately, perhaps based on setting the ith component of π to the proportion of times that the ith level is visited during an observation period and setting the ith component of b and w to be the proportion of times the algorithm bettered or worsened respectively after visiting the ith level during the observation period. (However, this method of estimating π ignores the interaction with b and w and is thus biased; alternative methods may be found to give more accurate results.) This avoids all the approximation stages detailed in this thesis but also loses the theoretical properties of the processes and approximation techniques that have been developed. In the other methods described above, backtracking adaptive search approximates an asymptotic averaged range process, which is already a Markov process in the range. If the parameters for backtracking adaptive search are estimated in a way that avoids approximating the domain process with a Markov process in the range (namely the

asymptotic averaged range process) then the beginning and endpoint of the approximating step to backtracking adaptive search are much more disparate than in any of the other methods. The approximation might well therefore be that much less likely to be close.

A good approximation method would be one that takes advantage of the theoretical results concerning convergence times of the range, averaged range and asymptotic averaged range processes while avoiding prohibitively great computational requirements.

The approach illustrated here is the third of these possibilities: to observe range values of a stochastic global optimisation algorithm and to estimate the transition matrix of an asymptotic averaged range process from them. The discussion in Section 3.4 shows that the asymptotic averaged range process approximation often has an expected convergence time very close to that of the original algorithm. A single further stage of approximation to backtracking adaptive search is then required. The expected number of iterations before convergence can then be calculated for this approximation. The first three methods all make use of the asymptotic averaged range process and therefore share this advantage over the fourth method, which requires the less flexible backtracking adaptive search to approximate the observed behaviour of the optimisation algorithm directly. Of the first three methods, the third is the simplest and therefore likely to be preferable. Other methods may be considered if the accurate estimation of asymptotic averaged range process parameters directly from observation of the process proves difficult.

In this case the accuracy of the estimate of convergence time rests on the accuracy of the empirical estimation of asymptotic averaged range process parameters, on the accuracy of the expected convergence time of this asymptotic averaged range process and on the quality of its backtracking adaptive search approximation. The first of these is unknown. Probabilities of transitions between levels visited commonly may well be estimated accurately, but there will in general be levels from which only a few, if any, transitions are observed. The second of these has received comment in Section 3.4; for a wide range of problem instances the expected convergence time for an asymptotic averaged range process approximation is close to the expected convergence time for the original process. The third of these, approximating the asymptotic averaged range process with backtracking adaptive search, is the area where the approximation process becomes inaccurate. As exemplified in Section 6.2, an estimate for convergence time no more than ten times greater or less than the true value seems as much as can be expected from this step. (However, even an estimate no more accurate than this still provides at least some gauge of how long the process may be expected to run. Refining the methods of Section 4.6 may allow the accuracy of this approximation to be improved.)

The approach illustrated therefore involves running an algorithm on a problem for a large number of iterations. Transitions in the problem range are noted, so that the asymptotic averaged range process transition matrix R can be approximated empirically. In particular, order the observed range levels from least to greatest and label them as $1, 2, 3, \ldots, \hat{m}$. If W_{ij} is the number of observed transitions from the level labelled i to the level labelled j then the estimate of R is obtained by letting $R_{ij} = W_{ij}/\sum_{k=1}^{\hat{m}} W_{ik}$. (If the optimal level is visited then the corresponding row in R is altered to a row of zeros with an initial one, so that R is absorbing. Otherwise such a row must be added and the probability of transitioning to the optimal level at each iteration must be estimated in some way for each other level. Also if no transitions from the last level visited have been recorded then this level must be excluded from the estimate of R in order to ensure that the transient portion of R is irreducible.) It would be possible to ignore the initial iterations of the algorithm, allowing it to settle in before commencing observation; however, this technique is not employed in the analysis discussed in this section. The initial level is chosen uniformly on the domain (and new initial levels are chosen in the same way if the algorithm does happen to visit the optimal level during the observation period).

Note that in general the number of range levels visited by the algorithm will be less than the total number of range levels. (This will certainly be true if the range is continuous.) The labelling of levels in R will therefore be different in general to the labelling if R was generated from a true domain process transition matrix P. Provided the algorithm visits a level within the termination region, there is sufficient information for the expected number of iterations before sampling a value in that termination region to be estimated from R. However, if no levels within the termination region are sampled then the transition probabilities from each level to the termination region must be estimated in some other way. It must be stressed that this latter possibility is the normal situation for difficult optimisation problems; if a short observation period is sufficient for the algorithm to find the optimal solution then the need to estimate expected convergence times is not so great.

Predicting convergence times in this manner for a black box process before a single observation of convergence necessarily involves extrapolation. Clearly, no guarantee can be made for the accuracy of such an extrapolation without additional information about the process. Only if the algorithm has already been observed to converge can an estimate from data of the probability of absorption from any level be made and the expected convergence time calculated as in the previous section.

To illustrate the approach, the two subsections following exemplify the approximation process on two examples. In the second of these, a simple method of extrapolation is used to estimate probabilities of convergence to the absorbing level before it is sampled.

6.3.1 Example 2 revisited

The algorithm used in Example 2 of Section 6.2 is run for 1000000 iterations. There is no need in this case to discretise the algorithm, since the division of the range into discrete range levels can be made independently of the exact levels visited by the algorithm. For instance, the range may be divided into a finite number of intervals (perhaps, but not necessarily, of equal width). All transitions to or from each interval can then be combined in order to generate the discrete approximation to R. However, the following analysis again uses the discretised version of the algorithm, to provide a point of comparison with Section 6.2. The expected convergence time of the process is thus known to be 1199.5 iterations.

After 1000000 iterations, the algorithm had visited all 326 range levels, so the empirical R is a 326×326 matrix. The expected convergence time of the asymptotic averaged range process with this estimate of R is directly calculated to be 1177.9 iterations, providing a reasonable estimate of the true convergence time. The true value is underestimated by 21.6 iterations, or 1.8%.

In this case the number of levels is small enough for the expected convergence time of the asymptotic averaged range process approximation to be calculated directly. If the number of levels was greater, though, a further stage of approximation would be required. This further level of approximation is provided by backtracking adaptive search. (Or else a more coarse division of the range into levels could be used to reduce the dimension of R to a point where its expected convergence time could conveniently be discovered directly.) The backtracking adaptive search approximation in this case has an expected convergence time of 439.0 iterations, underestimating the true value by 63.4%. This error is comparable with that in the previous section; the comments on accuracy made there are again valid.

6.3.2 A further example

In this subsection the approximation process is applied to a three dimensional generalisation of the previous example. The number of iterations before convergence for this generalisation will be much greater; the probability of sampling the optimum within a few hundred observations is very small. After a number of observed iterations, therefore, the prediction method applies some simple extrapolation to estimate the probability of convergence required for π.

In this example the same algorithm is applied to the problem of minimising the three dimensional function $-2.5\sin(x)\sin(y)\sin(z) - \sin(5x)\sin(5y)\sin(5z)$ over the domain $0 \leq x, y, z \leq \pi$. The global minimum of this function can be shown analytically to be at $(x, y, z) = (\pi/2, \pi/2, \pi/2)$. The value at this point is -3.5. Since this problem is much harder than its two dimensional analogue, it is no longer possible to evaluate the expected number of iterations before convergence directly, as was the case in Section 6.2. However, it is still small enough that the expected convergence time can be estimated empirically by running the algorithm several times and taking the average of all convergence times. This average can then be compared with estimates obtained by the prediction method, providing a point of reference by which the effectiveness of the method can be measured.

It would be possible to increase the dimension of the problem further to make it harder and thus more like the problems for which this method is designed. However, in that case there would be no way of checking the accuracy of expected convergence times.

The algorithm of Example 2 in Section 6.2 is applied 1000 times to this problem, recording the number of iterations required to sample a value with objective function value less than -3.49. The average number required is 47000.

The prediction method is now applied to this example. The algorithm is run for 1000 iterations, from which data the expected convergence time is to be estimated. Note that the estimate itself is a random variable, dependent on the observed behaviour of the stochastic global optimisation algorithm in its first 1000 iterations. Different realisations of these iterations will therefore result in different estimates of the expected convergence time. Ideally the variance of these estimates would be small, but there is no reason why it should be so.

There is a slight possibility of sampling the optimum within these 1000 observations, in which case the method of Subsection 6.3.1 can be used. (The estimated convergence time in this case is likely to be an underestimate, since the algorithm was observed to converge within 1000 iterations.) Much more likely, however, is that the 1000 observations will not include any within the termination region. In this case the behaviour of the algorithm over the rest of the range must be used to predict its behaviour over the unsampled lower end.

An obvious estimation technique is simply to estimate π over the region of the range that has been sampled, using Equation (4.11) as before, and then to fit a curve to it. The value that this curve takes at the target range level, in this case -3.49, will determine an estimate for the value of π at this level. With the estimate of the first component of π obtained in this way, the method illustrated in previous sections may now be used to complete the approximation process.

The length of the fitted π vector is another parameter of the approximation process. The values observed by the algorithm must be divided into a number of distinct range levels. For the purposes of this example, the observed values will be divided into 100 range levels.

Figure 6.2 illustrates the procedure. After 1000 iterations of the algorithm, π is fitted to the observed range values by calculating an empirical estimate of the asymptotic averaged range process transition matrix R and applying the method of Section 4.6. A smooth curve is then fitted, in order to estimate the values of π outside the observed range values. A cubic has been chosen for this task; the smoothed curve is also plotted in the figure. (In order to ensure that predictions for π are never negative, the curve is fitted to the log of the data.) A prediction for π at level -3.49 is then available.

Other extrapolation methods could, of course, be used; a polynomial may be a bad choice if π must be extrapolated too far from the data. The cubic estimate appears reasonable in the example of Figure 6.2.

This procedure is now applied 10000 times to give an idea of the distribution of predictions available from this procedure. The average of the predictions is 132000000, which is 2800 times the average convergence time already found of 47000. However, the distribution of the 10000 estimates is highly

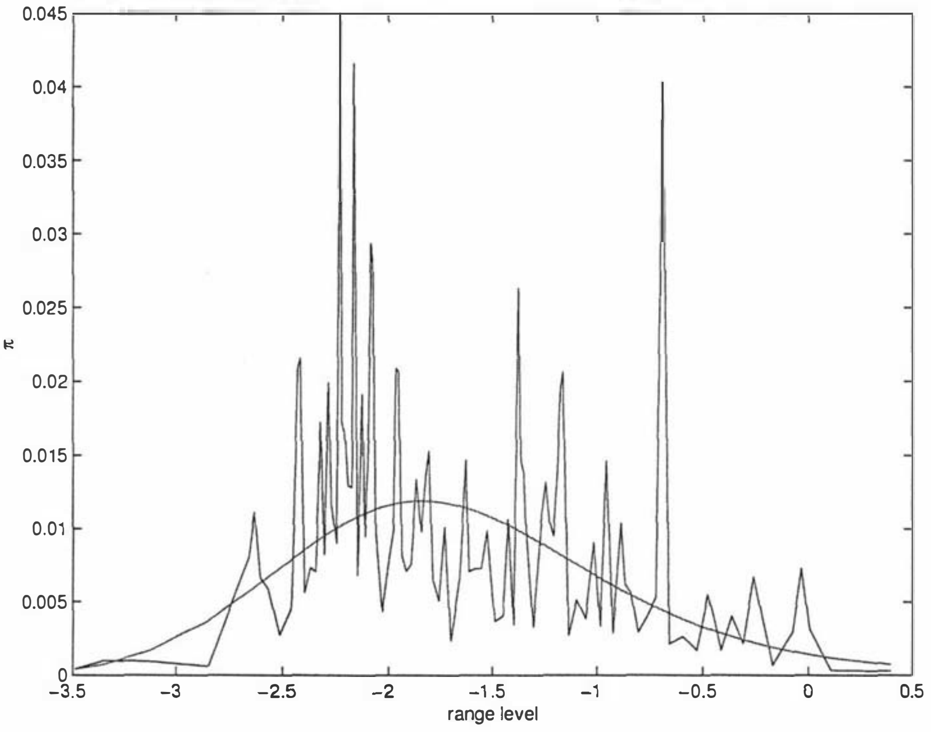

Figure 6.2: Fitting a curve to the estimate of π in the example of Subsection 6.3.2.

right skewed, as shown in Figure 6.3. Note that the plot is logarithmic, to counter the effect of the right skew so that the shape of the distribution can be seen.

Two outliers, with values over 10^{11}, are heavily influencing the average; almost all the data is much less than this average. Without these values, the mean comes down to 772000, only 16 times the average convergence time found previously. The median, by contrast, is only 5000, around a ninth of the empirical average convergence time.

This point estimate is perhaps not as much use as the histogram itself, which reveals the variation in estimates obtained from the prediction method. Estimates vary from as little as 100 up to a limit of around 400000, with 5% of values even higher. About half of these values fall within a factor of ten of the empirical average; however, the variation is such that little confidence could be placed in any particular estimate.

Estimates are highly sensitive to the first component of π, which is found based on extrapolation of the other values in π based on the observed behaviour of the algorithm in its first 1000 iterations. This behaviour is naturally highly variable. Less variation in estimates might be obtained by use of a

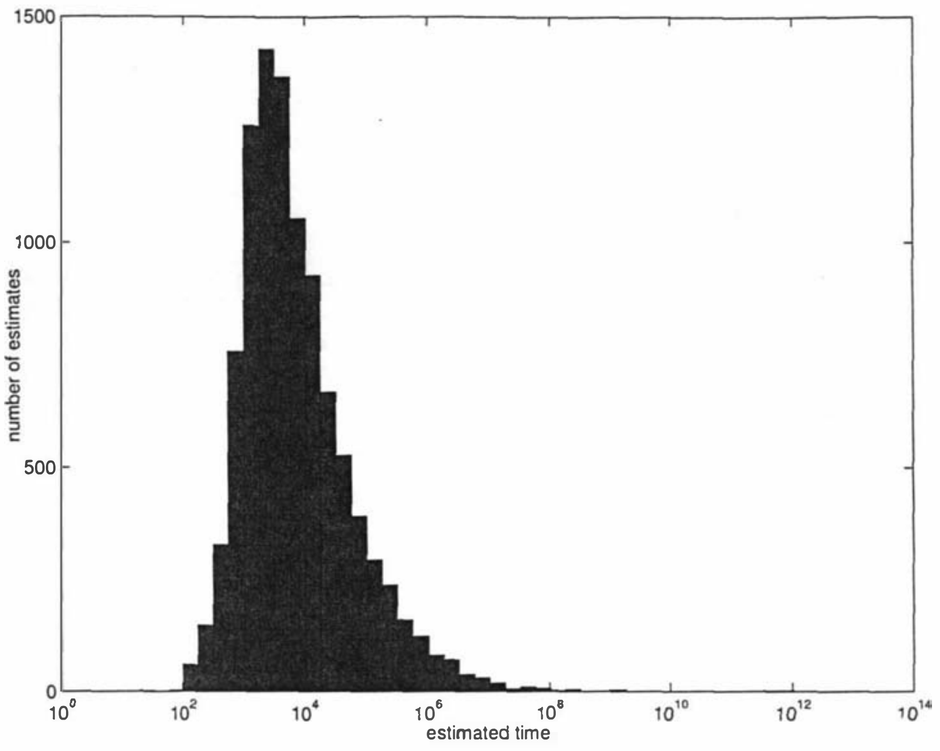

Figure 6.3: Distribution of estimates of expected convergence time for the example of Subsection 6.3.2.

longer observation period, or by a more robust method of calculating and extrapolating π.

While there remains scope for fine tuning the method, this example illustrates the approach to estimating convergence times of a stochastic global optimisation algorithm. The usual cautions necessary when applying extrapolation are applicable here: firstly, the implicit assumption has been made that the trend observed in π continues at lower range levels, but this is always unproven. It is even possible that there are no feasible solutions at the range level to which π is extrapolated. Secondly, the extrapolation leads to highly variable results.

This section illustrates an application of the approximation framework to a genuine black box process. The expected convergence time of the process is estimated from the asymptotic averaged range process and backtracking adaptive search approximations.

6.4 Summary

This chapter illustrates the approximation framework on several examples. Estimates of convergence times using the asymptotic averaged range process approximation for the examples are shown to be very accurate, though this accuracy is (naturally) not so great if the transition matrix R is only estimated empirically from an actual run of the algorithm.

The backtracking adaptive search approximation gives a much less accurate estimate of convergence time. The example of Subsection 6.3.2 shows that although the distribution of estimates places a large weighting on values close to the true expected convergence time, the variance in this version of the estimation procedure is too large for any great guarantee of accuracy to be made. Estimation of the convergence rate of a general optimisation algorithm by this technique as implemented here would thus be somewhat unreliable. Nonetheless, a method is now available for gaining an indication of the approximate runtime required by a stochastic global optimisation algorithm. In some cases where comparison is to be made between two algorithms sufficiently different from each other in their convergence behaviour, it may be possible to distinguish between them using this method.

If the range is discretised into a small enough number of intervals, the expected convergence time of R may be found directly via matrix inversion, avoiding the difficulty of approximating with backtracking adaptive search. A problem domain of high dimension will in general have a very large number of feasible solutions to consider; but since the range has only a single dimension, the number of possible range levels for the asymptotic averaged range process may be much less than the number of states for the original process in the domain.

Some avenues for future work are now highlighted. It may be possible to improve the accuracy of the backtracking adaptive search approximation to the asymptotic averaged range process; this is a significant contributor to the error in the estimates of expected convergence time for the examples examined in this section. Alternatively, a better means of approximating the expected convergence time of the asymptotic averaged range process may be found. Finding a suitable method for approximating the asymptotic averaged range process in situations where no transitions to the termination region have been sampled also remains an area for research.

The sequence of approximating processes developed and analysed in this thesis make a first step towards the estimation of the number of iterations before convergence for a general Markovian stochastic global optimisation algorithm. Subsequent research can develop these ideas further in order to improve the accuracy with which convergence times can be estimated; the examples in this chapter provide a taste of the kinds of results that may be obtained in this way.

Chapter 7

Conclusion

7.1 Summary of thesis

Two challenges are addressed in this thesis. The first of these is the analysis of backtracking adaptive search, the logical extension of pure adaptive search and hesitant adaptive search algorithms that have already received attention in optimisation literature. The second, more far-reaching goal is to provide a framework whereby the theoretical analysis now available for backtracking adaptive search can be used to estimate the convergence time of a general stochastic global optimisation algorithm.

The first of these aims is achieved in Chapters 4 and 5. A moment generating function for backtracking adaptive search on discrete, continuous or mixed domains has been presented. The mean and variance of convergence times for finite backtracking adaptive search are derived explicitly in Corollaries 4.3.1 and 5.3.1. The former of these has been published in [1]; the analogous result for a continuous range function π is given in [9].

In Section 4.4 a closed expression for the expected convergence time for finite backtracking adaptive search with arbitrary initial distribution is presented. Several special cases are illustrated in Section 4.5, including the derivation of results discussed in [31, 51, 56].

Chapter 5 contains a definition of backtracking adaptive search for a general range distribution. Moment generating functions for the number of iterations before convergence are then derived. The moment generating function approach also appears in [50].

The second of these aims seeks to apply backtracking adaptive search as a model for a general stochastic global optimisation algorithm. The analysis of the convergence rate of this algorithm, summarised above, is now complete. The challenge is then to model a general optimisation algorithm with backtracking adaptive search and use this model for the prediction of convergence times. Much further research is possible in this direction.

A framework of processes for approximating the convergence rate of an arbitrary Markovian stochastic global optimisation algorithm is presented in Chapter 2, with several theoretical results proved in Chapter 3. A paper outlining this framework and containing some of the analysis has also been published [49]. A chain of intermediate processes serve to break the approximation into several further stages. Each process in this chain is derived from the previous one, and can be used to approximate its convergence behaviour. The chain culminates in a tractable stochastic process, backtracking adaptive search. Using the results outlined above, convergence rates can be quickly obtained for the backtracking adaptive search model.

The range process is the image of the solutions sampled by an optimisation process in the domain, projected into the range using the objective function. Since the range process is not Markovian, the averaged range process is introduced as a time-inhomogeneous Markovian approximation to the range process. Theorem 3.2.1 shows that this process is identical in marginal distribution to the range process. Corollary 3.2.1 then shows that the number of iterations before convergence for both processes is also identical in distribution. An accurate estimate of the convergence time of the averaged range process would therefore also provide an accurate estimate of the convergence time of the original process.

Although the averaged range process is time-inhomogeneous, in many situations the transition matrices tend to a limit very quickly. The Markovian process using this limit transition matrix is called the asymptotic averaged range process. Theorem 3.3.3 provides a general definition of the asymptotic averaged range process transition matrix pertaining to any optimisation process. The expected convergence time of this process differs from that of the original process; however, under a weak assumption, Theorem 3.4.1 shows that the difference in expected convergence time between the asymptotic averaged range process and the original process it approximates is the sum of a series whose terms tend to decay geometrically. The required assumption can be ensured by imposing the weak condition that pure random search be conducted from some level with positive probability. For a range of examples discussed in Chapters 2, 3 and 6, the convergence rate of the asymptotic averaged range process is very similar to that of the optimisation process it approximates.

The asymptotic averaged range process and backtracking adaptive search are both time-homogeneous Markov processes defined on the range. The advantage of approximating the asymptotic averaged range process with backtracking adaptive search is that predictions can easily be made of the convergence rate of a backtracking adaptive search model. Similar predictions on general Markov processes require matrix inversion, which is in general very time-consuming. The special structure of backtracking adaptive search allows expected convergence times to be calculated using the analytical results of Chapters 4 and 5.

A method is proposed in Section 4.6 for approximating any asymptotic averaged range process with backtracking adaptive search. The complete framework thus provides a sequential procedure by which a backtracking adaptive search algorithm can be chosen as a model for any optimisation process. Prediction of convergence times for optimisation processes can then be made based on this model. Some examples of the complete procedure are given in Chapter 6.

Section 6.2 illustrates the entire framework under the assumption that the structure of the Markovian optimisation algorithm is known exactly. Although asymptotic averaged range process approximations give very good estimates of the expected convergence time, the simple method of obtaining backtracking adaptive search models suggested in Section 4.6 yields much less accurate estimates. An alternative method using maximum likelihood estimation provides much better results, but this method has impractical computational requirements.

Section 6.3 addresses the issue of estimating parameters for use in the model from a finite number of observations of the optimisation algorithm. Several possible approaches are listed. The preferred method is to estimate parameters for the asymptotic averaged range process directly from observation of the optimisation algorithm and then to proceed to the backtracking adaptive search model. The asymptotic averaged range process is much simpler than the original process in the domain, or the range or averaged range processes; yet Theorem 3.4.1 implies that its expected convergence time is in many cases very close to that of the original algorithm. The technique illustrated in this section is a practical method for estimating convergence time for any optimisation algorithm, based on the analysis developed in this thesis of the approximation framework and backtracking adaptive search model.

7.2 Further work

The method of Section 4.6 is very simple; improvement in this step remains an important area for development. Aside from the error inherent in the prediction of parameters from a finite number of observations of a process, this step is the major source of error in the approximation framework. A maximum likelihood technique was much more successful in the examples of Section 6.2, but this method is too complex to be practicable. Other methods may be able to give improved performance within an efficient time frame.

A second difficulty is encountered in Section 6.3, that the probabilities of transitions to optimal states must in general be estimated before such transitions have been observed. Evidently much uncertainty must inevitably surround such prediction. Future research may focus on restricted problem types for which such probabilities can be estimated with some accuracy.

One remaining challenge is thus to improve the method used to approximate the asymptotic averaged range process with backtracking adaptive search. Another major area to be addressed is the prediction of parameters for model processes based only on the observation of a finite number of iterations of an algorithm, before it converges. Both of these areas to be addressed represent significant difficulties in applying the methodology to estimate convergence rates of optimisation algorithms.

If these challenges can be answered, the prediction method put forward in this thesis stands to fill an important need. The complete strategy for analysis allows prediction of how long a particular stochastic global optimisation algorithm should be run to reach a set level. Quantitative measures of the effectiveness of stochastic global optimisation algorithms can then be made. A method for estimating the time required by an optimisation algorithm would be a valuable addition to the resources of optimisation practitioners. Information of this kind is also useful for tuning algorithms, for example through tailoring search region to landscape.

Bibliography

[1] Alexander, D. L. J., Bulger, D. W. and Wood, G. R. (2003), Expected search duration for finite backtracking adaptive search, *Journal of Algorithms* **47** 78–86.

[2] Althöfer, I. and Koschnick, K.-U. (1991), On the convergence of "threshold accepting", *Applied Mathematics and Optimization* **24** 183–195.

[3] Bain, L. J. and Engelhart, M. (1992), *Introduction to Probability and Mathematical Statistics*, PWS-KENT Publishing Company, Boston.

[4] Baritompa, W. P. and Steel, M. A. (1996), Bounds on absorption times of directionally biased random sequences, *Random Structures and Algorithms* **9** 279–293.

[5] Belisle, C. J. (1992), Convergence theorems for a class of simulated annealing algorithms on $\mathbb{R}^d$, *Journal of Applied Probability* **29** 885–895.

[6] Beyer, W. H. (1981), *CRC Standard Mathematical Tables, 26th Edition*, CRC Press, Inc., Boca Raton.

[7] Boender, C. G. E. and Rinnooy Kan, A. H. G. (1987), Bayesian stopping rules for multistart global optimization methods, *Mathematical Programming* **37** 59–80.

[8] Brooks, S. H. (1958), A discussion of random methods for seeking maxima, *Operations Research* **6** 244–251.

[9] Bulger, D. W., Alexander, D. L. J., Baritompa, W. P., Wood, G. R. and Zabinsky, Z. B. (2004), Expected hitting times for backtracking adaptive search, *Optimization* (to appear).

[10] Bulger, D. W., Baritompa, W. P. and Wood, G. R. (2003), Implementing pure adaptive search with Grover's quantum algorithm, *Journal of Optimization Theory and Applications* **116** 517–529.

[11] Bulger, D. W. and Wood, G. R. (1998), Hesitant adaptive search for global optimisation, *Mathematical Programming* **81** 89–102.

[12] Chatelin, F. (1993), *Eigenvalues of Matrices*, John Wiley & Sons Ltd, Chichester.

[13] Diaconis, P. and Saloff-Coste, L. (1998), What do we know about the Metropolis algorithm?, *Journal of Computer and System Sciences* **57** 20–36.

[14] Dixon, L. C. W. and Szegö, G. P., Eds. (1975), *Towards Global Optimization*, North-Holland, Amsterdam.

[15] Dixon, L. C. W. and Szegö, G. P., Eds. (1978), *Towards Global Optimization 2*, North-Holland, Amsterdam.

[16] Dueck, G. (1993), New optimization heuristics: the great deluge algorithm and the record-to-record travel, *Journal of Computational Physics* **104** 86–92.

[17] Dueck, G. and Scheuer, T. (1990), Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing, *Journal of Computational Physics* **90** 161–175.

[18] Gademann, A. J. R. M. (1993), *Linear Optimization in Random Polynomial Time*, PhD thesis, Department of Applied Mathematics, University of Twente, Enschede, The Netherlands.

[19] Gantmacher, F. R. (1959), *Matrix Theory*, Chelsea Publishing Company, New York.

[20] Glover, F. and Laguna, M. (1997), *Tabu Search*, Kluwer Academic Publishers, Boston.

[21] Hajek, B. (1988), Cooling schedules for optimal annealing, *Mathematics of Operations Research* **13** 311–329.

[22] Hanafi, S. (2000), On the convergence of tabu search, *Journal of Heuristics* **7** 47–58.

[23] Hendrix, E. M. T., Ortigosa, P. M. and García, I. (2000), On success rates for controlled random search, *Technical Note 00-01* Department of Mathematics, Wageningen University, Wageningen, The Netherlands.

[24] Holland, J. H. (1975), *Adaptation in natural and artificial systems*, The University of Michigan Press, Ann Arbor.

[25] Hopfield, J. J. and Tank, D. W. (1985), "Neural" computation of decisions in optimization processes, *Biological Cybernetics* **52** 141–152.

[26] Jacobson, S. H. and Yücesan, E. (2004), Global Optimization Performance Measures for Generalized Hill Climbing Algorithms, *Journal of Global Optimization* (to appear).

[27] Kemeny, J. G. and Snell, J. L. (1960), *Finite Markov Chains*, Van Nostrand, New York.

[28] Kingman, J. F. C. (1993), *Poisson Processes*, Oxford University Press, Oxford.

[29] Kirkpatrick, S., Gelatt, C. D. Jr. and Vecchi, M. P. (1983), Optimization by simulated annealing, *Science* **220** 671–680.

[30] Koehler, G. J. (1997), New directions in genetic algorithm theory, *Annals of Operations Research* **75** 49–68.

[31] Kristinsdottir, B. P., Zabinsky, Z. B. and Wood, G. R. (2002), Discrete backtracking adaptive search for global optimization, in *Stochastic and Global Optimization, Nonconvex Optimization and its Applications, Volume 59* (G. Dzemyda, V. Saltenis and A. Zilinskas, Eds.), Kluwer Academic Publishers.

[32] Leary, R. H. (2000), Global optimization on funneling landscapes, *Journal of Global Optimization* **18** 367–383.

[33] Levy, H. and Lessman, F. (1961), *Finite Difference Equations*, The Macmillan Company, New York.

[34] Locatelli, M. (2001), Convergence and first hitting time of simulated annealing algorithms for continuous global optimization, *Mathematical Methods of Operations Research* **54** 171–199.

[35] Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H. and Teller, E. (1953), Equations of state calculations by fast computing machines, *The Journal of Chemical Physics* **21** 1087–1092.

[36] Papadimitriou, C. H. and Steiglitz, K. (1982), *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Inc., New Jersey.

[37] Patel, N. R., Smith, R. L. and Zabinsky, Z. B. (1989), Pure adaptive search in Monte Carlo optimisation, *Mathematical Programming* **43** 317–328.

[38] Price, W. L. (1978), A controlled random search procedure for global optimization, in *Towards Global Optimization 2* (L. C. W. Dixon and G. P. Szegö, Eds.), North-Holland, Amsterdam.

[39] Reaume, D. J., Romeijn, H. E. and Smith, R. L. (2001), Implementing pure adaptive search for global optimization using Markov chain sampling, *Journal of Global Optimization* **20** 33–47.

[40] Rechenberg, I. (1973), *Evolution strategie: Optimierung technischer systeme nach prinzipien der biologischen evolution*, Frommann-Holzboog, Stuttgart.

[41] Rinnooy Kan, A. H. G. and Timmer, G. (1984), A stochastic approach to global optimization, in *Numerical Optimization* (P. Boggs, R. Byrd and R. B. Schnabel, Eds.), SIAM, Philadelphia.

[42] Romeijn, H. E. and Smith, R. L. (1994), Simulated annealing and adaptive search in global optimization, *Probability in the Engineering and Informational Sciences* **8** 571–590.

[43] Rudolph, G. (1994), Convergence analysis of canonical genetic algorithms, *IEEE Transactions on Neural Networks* **5** 96–101.

[44] Seneta, E. (1981), *Non-negative Matrices and Markov Chains*, Springer-Verlag, New York.

[45] Smith, K. A. (1999), Neural networks for combinatorial optimization: a review of more than a decade of research, *INFORMS Journal on Computing* **11** 15–34.

[46] Smith, R. L. (1984), Efficient Monte-Carlo procedures for generating points uniformly distributed over bounded regions, *Operations Research* **32** 1296–1308.

[47] Triantaphyllou, E. (2000), *Multi-Criteria Decision Making Methods: A Comparative Study*, Kluwer Academic Publishers, Dordrecht, The Netherlands.

[48] Trouvé, A. (1996), Cycle decompositions and simulated annealing, *SIAM J. Control and Optimization* **34** 966–986.

[49] Wood, G. R., Alexander, D. L. J. and Bulger, D. W. (2002), Approximation of the distribution of convergence times for stochastic global optimisation, *Journal of Global Optimization* **22** 277–284.

[50] Wood, G. R., Alexander, D. L. J. and Bulger, D. W. (2002), Backtracking adaptive search: the distribution of the number of iterations to convergence (Technical Report, IIST, Massey University).

[51] Wood, G. R., Zabinsky, Z. B. and Kristinsdottir, B. P. (2001), Hesitant adaptive search: the distribution of the number of iterations to convergence, *Mathematical Programming* **89** 479–486.

[52] Zabinsky, Z. B. (2003), *Stochastic Adaptive Search in Global Optimization*, Kluwer Academic Publishers, Boston.

[53] Zabinsky, Z. B., Graesser, D. L., Tuttle, M. E. and Kim, G. I. (1992), Global Optimization of Composite Laminates using Improving Hit and Run, in *Recent Advances in Global Optimization* (C. Floudas and P. Pardalos, Eds.), Princeton University Press.

[54] Zabinsky, Z. B. and Smith, R. L. (1992), Pure adaptive search in global optimization, *Mathematical Programming* **53** 323–338.

[55] Zabinsky, Z. B., Smith, R. L., McDonald, J. F., Romeijn, H. E. and Kaufman, D. E. (1993), Improving hit-and-run for global optimization, *Journal of Global Optimization* **3** 171–192.

[56] Zabinsky, Z. B., Wood, G. R., Steel, M. A. and Baritompa, W. P. (1995), Pure adaptive search for finite global optimization, *Mathematical Programming* **69** 443–448.

Appendix A

Programs

```
function [P, M, delta_0, t] = Chapter6(divisions)

%>Tailor made for the current situation; it would be better if it was more flexible. Set divisions = 51 for x values pi/50 apart.

%Checking

%IT'S GOT A SIMULATED ANNEALING PART

[x y] = Grid2(0, pi, pi/(divisions - 1)); %x is a column vector 1, 2, ..., divisions^2; y = f(x)

%probability(1,1) = 0; %probability of moving nowhere; this is updated for each state later
%for xincrement = 1:divisions
%   for yincrement = 0:xincrement - 1
%      probability(xincrement + 1, yincrement + 1) = ?; %probability of going along xincrement and up yincrement < xincrement
%   end
%   probability(xincrement + 1, xincrement + 1) = ?; %probability of going along xincrement and up xincrement
%end

ouh = 1;

number1 = 1; %row of P, corresponding at this stage to x = 0
for i1 = 1:divisions
   currentstate = [(i1 - 1) * pi/(divisions - 1) 0]; %the first one is (x,y) = (0,0)
   for j1 = 1:divisions
      number2 = 1; %destination column of P
      tic
      for i2 = 1:divisions
         nextstate = [(i2 - 1) * pi/(divisions - 1) 0];
         for j2 = 1:divisions
            if number1 ~= number2
               if y(number2) - y(number1) <= 0.00005
                  weight = 1;
               else
                  weight = 0.1;
               end

               transformedcurrentstate = currentstate;

               xincrement = nextstate(1) - currentstate(1);
               if xincrement < 0 %make all xincrements positive (this works based on symmetry)
                  transformedcurrentstate(1) = pi - currentstate(1);
                  xincrement = -1 * xincrement;
               end

               yincrement = nextstate(2) - currentstate(2);
               if yincrement < 0 %make all yincrements positive (this works based on symmetry)
```

```
                transformedcurrentstate(2) = pi - currentstate(2);
                yincrement = -1 * yincrement;
              end

              if yincrement > xincrement %make sure yincrement < xincrement (this works based on symmetry)
                temp = transformedcurrentstate(1);
                transformedcurrentstate(1) = transformedcurrentstate(2);
                transformedcurrentstate(2) = temp;
                temp = yincrement;
                yincrement = xincrement;
                xincrement = temp;
              end

%             P(number1, number2) = probability(xincrement + 1, yincrement + 1);

              if yincrement == 0
                d = pi * (1 + 1/(divisions - 1));
              else
                xintercept = -1 * pi/(2 * (divisions - 1));
                yintercept = transformedcurrentstate(2) + (transformedcurrentstate(1) - pi/(2 * (divisions - 1))) *
yincrement/xincrement;
                if yintercept < -1 * pi/(2 * (divisions - 1))
                  yintercept = -1 * pi/(2 * (divisions - 1));
                  xintercept = (-1 * pi/(2 * (divisions - 1)) - yintercept) * xincrement/yincrement;
                end

                xend = pi * (1 + 1/(2 * (divisions - 1)));
                yend = yintercept + pi * (1 + 1/(2 * (divisions - 1))) * yincrement/xincrement;
                if yend > pi * (1 + 1/(2 * (divisions - 1)))
                  yend = pi * (1 + 1/(2 * (divisions - 1)));
                  xend = (pi * (1 + 1/(2 * (divisions - 1))) - yintercept) * xincrement/yincrement;
                end

                d = sqrt((xend - xintercept)^2 + (yend - yintercept)^2);
              end
              r = sqrt(xincrement^2 + yincrement^2);
              P(number1, number2) = weight * (pi/(divisions - 1))^2 /(pi * r * d); %weight produces the simulated annealing part
            end

            nextstate(2) = nextstate(2) + pi/(divisions - 1);
            number2 = number2 + 1;
          end
        end
        P(number1, number1) = 1 - sum(P(number1, :)); %P(here, here) = 1 - P(here, anywhere else)
        if P(number1, number1) < 0
          P(number1, number1) = 0;
          P(number1, :) = P(number1, :)/sum(P(number1, :));
        end
        currentstate(2) = currentstate(2) + pi/(divisions - 1);
        number1 = number1 + 1;

        if mod(number1,100) == 0
          number1
        end

        t(ouh) = toc;
        ouh = ouh + 1;
    end
end


%P

delta_0 = ones(1, divisions^2)/divisions^2; %uniformly random

[M, P, delta_0] = FindM(x,y, P, delta_0); %P and delta_0 are now ordered and M has been created

P(1,1) = 1; %make it absorbing
for i = 2:divisions^2
  P(1,i) = 0;
end
```

```
function [x,y] = Grid2(minimum, maximum, fineness);

%>Produces a column vector x corresponding to the states of a grid over a Cartesian Plane starting at minimum and going up to
maximum
%in steps
%of
%fineness, and a corresponding vector y with values of f on the grid.

%It would be better to have this function independent of
%problem dimension, but I can't immediately see how to program that.

%Set minimum and maximum to be vectors containing values for each dimension?

%checked

%Calls f.m

number = 1;
numberOfStates = ceil((maximum - minimum)/fineness) + 1;
for i = 1:numberOfStates
   state = [minimum + (i - 1) * fineness minimum];
   for j = 1:numberOfStates
      x(number) = number;
      y(number) = f(state);
      state(2) = state(2) + fineness;
      number = number + 1;
   end
end

x = x';
y = y';
```

```
function [M, newP, newdelta_0] = FindM(x,y, P, delta_0);

%>Takes a column vector of distinct x values and a corresponding vector of y values, and outputs an ordered mapping matrix M.
%Transition matrix P and initial distribution delta_0 are also sorted to move states into the proper order.
%Objective function values are taken to be equal if they're within 0.00005 of each other.

%checked

%Should warn if x -> y is not a function
%Should check that states in x are distinct

order = sortrows([x y], 2);

totalStates = length(x);
j = 1;
oldlevel = order(1,2) - 1; %initialised to some value less than order(1,2)
for entry = 1:totalStates
   if order(entry, 2) - oldlevel > 0.00005
      oldlevel = order(entry, 2);
      for i = 1:totalStates
         if 0 <= order(i,2) - order(entry, 2)
            if order(i,2) - order(entry, 2) <= 0.00005
               M(i,j) = 1;
            else
               M(i,j) = 0;
            end
         end
      end
      j = j + 1;
   end
end

thismanylevels = j - 1;

for i = 1:totalStates
```

```
    tempP(i,:) = P(order(i,1),:);
    newdelta_0(i) = delta_0(order(i,1));
end

for i = 1:totalStates
    newP(:,i) = tempP(:,order(i,1));
end
```

function result = f(x)

```
%>This function is the one Eva uses (reference?). It takes a vector of coordinates and returns a (row) vector of function values.

%Checked

%result = abs(x(1) - x(2)); %a check function

for i = 1:length(x(:, 1))
    product = 1;
    product2 = 1;
    for j = 1:length(x(1,:))
        product = product * sin(x(i, j));
        product2 = product2 * sin(5 * x(i, j));
    end

    result(i) = -2.5 * product - product2;
end
```

function [R_n, P_n] = ARPmatrix(P, M, delta_0, n, P_nminusone)

```
%>Returns the ARP transition matrix after n iterations for a domain Markov transition matrix P with initial domain distribution
%delta_0
%and mapping matrix M. Matrix P should have a single absorbing state and all other states should be transient. States
corresponding
%to the same level should be congruent in P.

%Warning! Eventually the probability of being in a transient state is so small, it gets confused with 0. Then the weightings go
wrong
%and the answer is incorrect when n is too large.

%checked
%There should be a function which orders P and M so the user doesn't have to.
%It should be able to tell when probabilities of being in a state are so small that error is introduced, and stop if that happens.

P;
M;
delta_0;
n;

a = length(P);
m = length(M(1, :));

if P_nminusone == 0
   P_n = P^n;
else
   P_n = P * P_nminusone;
end
delta_n = delta_0 * P_n;

b = ones(1, a) * M; %b contains the number of domain levels with each range level
c = 1;
for i = 1:m
  total = sum(delta_n(c:c + b(i) - 1));
```

```
    for j = 1:b(i);
      if total > 0
        gamma_n(c) = delta_n(c) / total;
      else
        gamma_n(c) = 0;
      end
      c = c + 1;
    end
  end

R_n = M' * diag(gamma_n) * P * M;
```

function [expectedConvergenceTime, R, v, b, w, rho, AARPexpectedConvergenceTimeDelta, BASexpectedConvergenceTimeinit] = Approximateinit(P, M, delta_0)

```
%>Finds BAS with arbitrary initial distribution approximation to a Markov Chain with tranition matrix P, initial distribution
delta_0
%and mapping matrix M. Matrix P
%should have a single
%absorbing
%state and all other states should be transient. States corresponding
%to the same level should be congruent in P.

%There should be a function which orders P and M so the user doesn't have to.

%checked

%Calls Expected.m, ARPinf.m, AARPtoBAS.m and BAS.m and BASinit2.m

P;
delta_0;
M;

expectedConvergenceTime = Expected(P,delta_0)

%R = ARPinf(P,M,delta_0); %this is awful and the line below should be used.
[R v period] = AARPmatrix(P,M,delta_0);
[b w rho] = AARPtoBAS(R, P, M, v, period);
%rho
AARPexpectedConvergenceTimeDelta = Expected(R, delta_0 * M)
AARPexpectedConvergenceTimeRho = Expected(R,rho) %hopefully bigger than BASexpectedConvergenceTime, since then the
BAS approximation
%gives an upper bound to the AARP convergence time. To work this out takes as long as working
BASexpectedConvergenceTime out without
%using the results of the FBAS paper
BASexpectedConvergenceTime = BAS(b, w, rho, 0)
BASexpectedConvergenceTimeinit = BASinit2(b, w, rho, delta_0 * M, 0)
AndTheErrorIs = expectedConvergenceTime - BASexpectedConvergenceTimeinit
RelativeError = AndTheErrorIs/expectedConvergenceTime
```

function mean = Expected(P, delta_0)

```
%>Returns the expected number of iterations before absorption to the first state for a Markov chain with transition matrix P and
%initial distribution
%delta_0,
%including the absorbing step

%checked
%Should check all states except the first are transient

P;
delta_0;
```

```
a = length(P);
mean = delta_0(2:a) * inv(eye(a - 1) - P(2:a, 2:a)) * ones(a - 1, 1) + 1;
```

```
function [R, v , period] = AARPmatrix(P, M, delta_0);

%>Finds limiting weightings amongst transient states (after k period iterations) v and AARPmatrix R for an algorithm with
transition
%matrix P whose transient component has period period, objective function mapping matrix M and initial
%probability distribution delta_0, according to the method in AARP2.tex. P should have one absorbing state first and the rest
should
%be transient.

%checked

P;
ell = length(P);
k = length(M(1,:));

period = Periodfinder(P);

[W D] = eig(P^period.'); %D is eigenvalues of P^period. This failed for P = [1 0 0 0; 1 0 0 0; 0 0.1 0 0.9; 0 0 1 0]
W = W.'; %W is left eigenvectors of P^period
top = D(2,2);
rows = [];
for i = 2:ell
   if 1 > abs(D(i,i)) & abs(D(i,i)) > top
      top = abs(D(i,i));
      rows = [];
   end
   if D(i,i) == top
      rows = [rows i];
   end
end %period eigenvectors sharing second largest eigenvalue top occupy rows of W listed in rows

a = delta_0 * inv(W); %decomposition of delta_0 onto eigenvectors
v = zeros(1,ell - 1);
for i = 1:length(rows)
   v = v + a(rows(i)) * W(rows(i),2:ell);
end
v = real(v/sum(v));

%the next bit was coded twice
beta = zeros(period, ell);
for i = 0:period - 1
   beta(i+1, :) = [1 v * (P(2:ell, 2:ell))^i];
end %beta(i,:) is beta_{kd+a-1}, with a 1 in front

m = sum(M); %m(i) states have the ith level
n = 1;
betay = beta * M;
for i = 1:k %for each level
   for j = 1:m(i) %for each state at that level
      gamma(n) = 0;
      total = 0;
      for p = 1:period
         if betay(p,i) > 0
            gamma(n) = gamma(n) + beta(p,n)/betay(p,i);
            total = total + 1;
         end
      end %gamma{kd+a-1}(n) is defined for total/period iteration numbers
      gamma(n) = gamma(n)/total;
      n = n + 1;
   end
end


%I coded this twice, by mistake; if the above doesn't work, try the below:
```

```
%beta_kdplusa = limit;
%gamma = zeros(1,ell);
%gamma(1) = 1;
%total = zeros(1,ell); %total number of subclasses for which P(Y_n = f(x_i)) > 0
%for a = 0:period - 1
%   k = 2;
%   for i = 2:length(m) %for each level
%      weightAtLevel = [0 beta_kdplusa] * M(:,i);
%      if weightAtLevel > 0
%         for j = 1:m(i) %for each state at that level
%            gamma(k) = gamma(k) + beta_kdplusa(k - 1)/ weightAtLevel;
%            total(k) = total(k) + 1;
%            k = k + 1;
%         end
%      end
%   end
%   beta_kdplusa = beta_kdplusa * P(2:ell,2:ell);
%end

%for i = 2:ell
%   gamma(i) = gamma(i)/total(i);
%end

R = real(M' * diag(gamma) * P * M); %numerical error sometimes introduces a tiny fraction of imaginary number to entries of R
```

function period = Periodfinder(P);

```
%>Finds period of transient component of transition matrix P. P should have one absorbing state first and the rest should
%be transient.

%I'm changing its name to Periodfinder

%checked

P;
ell = length(P);

available = ones(1,ell); %not put into tree yet
groups = zeros(ell,ell); %groups(i,:) is a list of all levels of the tree into which state i fits
for i = 2:ell
   if i == 2
      current = 2;
      groups(2,1) = 1; %assign state 2 to group 1
      available(2) = 0; %state 2 is being used in tree
   else
      for j = 3:ell
         if groups(j,1) * available(j) > 0 %if state j is in tree but has no branches from it
            current = j;
            available(j) = 0; %state j is being used in tree
            break
         end
      end
   end

   for j = 2:ell
      if P(current,j) > 0 %if transition is possible then add branch from state current to state j in tree
         for k = 1:ell
            if groups(j, k) == 0
               groups(j, k) = groups(current, 1) + 1; %adding the branch
               break
            elseif groups(j, k) == groups(current, 1) + 1
               break %branch is already there
            end
         end
      end
   end
end
```

```
period = groups(2,2) - 1; %this is one possible number of steps in which state 2 can transition to itself
for i = 2:ell
   for j = 2:ell
      if groups(i,j) > 0
         period = gcd(period, abs(groups(i,j) - groups(i,1))); %greatest common factor of current period value and a possible
number
      else                                    %of steps in which state i transitions to itself
         break
      end
   end
end
```

function [b, w, rho] = AARPtoBAS(R, P, M, v, period);

```
%>Takes AARP transition matrix R (as output from ARPinf.m) and outputs BAS parameters using Findrho.m.

%checked

%Calls FindRho.m, BASmatrix.m

R;
m = length(R);
for i = 1:m
   w(i) = 1 - sum(R(i, 1:i));
   b(i) = 1 - R(i, i) - w(i);
end

%tic
rho = Simplerho(R, P, M, v, period);%MLErho2(R);%MLErho(R);%FindRho(R);%
%toc
%B = BASmatrix(rho, b, w);
```

function rho = Simplerho(R, P, M, v, period)

```
ell = length(P);
Q = P(2:ell, 2:ell);

betabar = zeros(1, ell - 1);
for a = 0:period - 1
   addon = v * Q^a;
   betabar = betabar + addon/(period * sum(addon));
end

R = R - diag(diag(R));
rho = [0 betabar] * M * R;
%sum(rho)
rho = rho/sum(rho);
```

function bestrho = MLErho2(W) %my attempt at programming MLErho, NOW with MultiStart!

```
m = size(W, 1);


if m == 3
   rho = W(3,:);
   rho(3) = (rho(1) + rho(2))/2;
   bestrho = rho/sum(rho);
   return
end


rho = sum(W(2:m,:));
for i = 2:m
   rhozeros(i) = rho(i) - W(i,i);
end

rhozeros(1) = rho(1);
rho = rhozeros/sum(rhozeros);
bestrho = rho;

%return
%rho = [0.3071    0.2468    0.0759    0.3702]



loglikelihood = L(W,rho) %not the actual loglikelihood, but a scaled version of it
maxLLL = loglikelihood;
iteration = 0;

MS = 1;
for ii = 1:MS


   num_steps = 0;
   while (num_steps == 0) | (sum(abs(delta)) > 0.000001)
      num_steps
  % Firstly determine 'd' & 'E', vector & array of first & second derivatives
  % of the log-likelihood with respect to the elements of rho:
      p = cumsum(rho);
      d = zeros(1, m); %unnecessary?
      E = zeros(m);

      for a = 1:m
         if a < m
            for i = a + 1:m
               c(i) = W(i,a)/rho(a) - sum(W(i,1:i - 1))/p(i - 1);
            end
            if a == 1 %this kind of conditional statement is because otherwise infinity - infinity type errors occur
               d(1) = sum(c(3:m));
            else
               d(a) = sum(c(a + 1:m));
            end
         end
         if a > 2
           for i = 2:a - 1
              f(i) = W(i,a)/rho(a) - sum(W(i,i + 1:m))/(1 - p(i));
           end
           if a == m
              d(m) = sum(f(2:m - 2));
           else
              d(a) = d(a) + sum(f(2:a - 1));
           end
        end

        if a < m
           for i = a + 1:m
              r(i) = sum(W(i,1:i - 1))/(p(i - 1))^2 - W(i,a)/(rho(a))^2;
           end
           if a == 1
              E(1,1) = sum(r(3:m));
           else
              E(a,a) = sum(r(a + 1:m));
           end
        end
        if a > 2
           for i = 2:a - 1
              q(i) = sum(W(i,i + 1:m))/(1 - p(i))^2 - W(i,a)/(rho(a))^2;
           end
           if a == m
```

```
            E(m,m) = sum(q(2:m - 2));
          else
            E(a,a) = E(a,a) + sum(q(2:a - 1));
          end
        end

        if a < m
          for b = a + 1:m
            if b < m
              for i = b + 1:m
                g(i) = sum(W(i,1:i - 1))/(p(i - 1))^2;
              end
              E(a,b) = sum(g(b + 1:m));
            end
            if a > 2
              for i = 2:a - 1
                h(i) = sum(W(i,i + 1:m))/(1 - p(i))^2;
              end
              E(a,b) = E(a,b) + sum(h(2:a - 1));
            end
            E(b,a) = E(a,b);
          end
        end
      end
      d;
      E;
      sumE = sum(sum(E));
      if sumE > 0
%          fprintf(1, 'oops! sum(sum(E)) > 0');
      end
 %    Einv = inv(E);



%        delta = ((d*Einv*ones(m))/(sum(sum(Einv))) - d)*Einv;
     Einvones = E\ones(m,1);
        delta = (sum((d*Einvones)/(sum(Einvones))) - d)/E;

   % Now increment rho & loop:
   rho = rho + delta;




     rho = real(rho); %the real part of rho - unnecessary, I think
     rho = rho + (abs(rho) - rho)/2; %make rho non-negative
     minrho = max(rho);
     for i = 1:m
       if 0 < rho(i) & rho(i) < minrho
         minrho = rho(i); %this was only a quick fix - when pi_m > 0 the other pi_i (except pi_1) are allowed to be 0.
       end
     end

     for i = 1:m
       if rho(i) == 0
         rho(i) = minrho; %this was only a quick fix - when pi_m > 0 the other pi_i (except pi_1) are allowed to be 0.
       end
     end

     spectrum = eig(E);

     rho = rho/sum(rho);
     oldLLL(num_steps + 1) = loglikelihood;
     loglikelihood = L(W,rho); %not the actual loglikelihood, but a scaled version of it

     num_steps = num_steps + 1;
     if num_steps == 340
       delta = 0;
     end
     if num_steps > 10999999999999 & oldLLL(1) > loglikelihood
       delta = 0;
```

```
        end
      end
      fprintf(1, 'The search used %d steps.\n', num_steps);
            rho;
            oldLLL;
 loglikelihood
 if loglikelihood < -1.1925
 %   fprintf(1, 'oops! different rho');
 end

      if loglikelihood >maxLLL
         fprintf(1, 'Hurrah\n');
         bestrho = rho;
         maxLLL = loglikelihood;
         iteration = ii;
      end

      rho = rand(1, m);
      for i = 1:m
         if rhozeros(i) == 0
            rho(i) = 0;
         end
      end
      rho = rho/sum(rho);
      loglikelihood =L(W, rho);
      oldLLL = [];
 end

 fprintf(1, 'Maximum likelihood of %f found at iteration %d for rho as below', maxLLL, iteration);
```

```
function expected = BAS(b, w, rho, vector);

%>Calculates expected number of iterations until convergence for BAS with parameters b, w and rho, including the absorbing step, using
%the FBAS corollary. It seemed useful at the time to allow calculation of the entire expected vector; set vector = 1 if you want it to
%do that. Anything calling this probably won't specify the vector parameter, since I only just did that part.

%A big problem is that it should take into account the initial vector, which is known. That would improve accuracy lots.

%checked

m = length(rho);

p_i(1) = rho(1);
for i = 2:m
   p_i(i) = p_i(i - 1) + rho(i);
end

for i = 2:m - 1
   a(i) = p_i(i) * (rho(i) * w(i) + (1 - p_i(i)) * (b(i) + w(i)));
   c(i) = p_i(i) * b(i) + p_i(i - 1) * w(i);
   if (1 - p_i(i)) * c(i) == 0
      (1 - p_i(i)) * c(i);
   end
   d(i) = a(i)/((1 - p_i(i)) * c(i));
   e(i) = 1/d(i);
end

for i = 2:m - 1
   f(i) = rho(i)/c(i) * prod(e(i:m - 1));
end

if m > 2
   if vector == 1
      top = m - 2;
   else
```

```
        top = 1;
    end
else
    top = 0;
end

for i = 1:top
    mu(i) = prod(d(i + 1:m - 1)) * (rho(m)/b(m) + sum(f(i +1:m - 1))) + 1; %I got a divide by zero problem here once
end

if m > 1
    mu(m - 1) = rho(m)/b(m) + 1;
end
mu(m) = 1;

if vector == 1
    expected = mu;
else
    expected = mu(1);
end
```

```
function expected = BASinit2(b, w, rho, pi_0, vector);

%>Calculates expected number of iterations until convergence for BAS with parameters b, w and rho and initial distribution
pi_0,
%including the absorbing step, using
%the FBAS with arbitrary initial distribution corollary. It seemed useful at the time to allow calculation of the entire expected
%vector; set vector = 1 if you want it to
%do that.

%checked

%calls BAS.m

pi_0;
rho;
b;
w;


m = length(rho);

p_i(1) = rho(1);
Phi_i(1) = pi_0(1);
for i = 2:m
    p_i(i) = p_i(i - 1) + rho(i);
    Phi_i(i) = Phi_i(i - 1) + pi_0(i);
end

mu(m) = 1;
for i = 2:m
    top = m + 2 - i; %top is greatest y such that E[N(y-1) - N(y)|Y_N(y) = y] is defined = y* - 1
    if rho(top) > 0 %then E[N(y-1) - N(y)|Y_N(y) = y] is defined, unless w(top) = 0 (which it inevitably will)
        if w(top) == 0 %in which case E[N(y-1) - N(y)|Y_N(y)=y] will be defined for y = top-1 and mu(top-1) can be calculated
directly
            mu(top - 1) = mu(top) + pi_0(top)/b(top) + (1 - Phi_i(top)) * rho(top)/b(top);
            top = top - 1;
        end
        break
    else %try another top and calculate mu(top) directly
        if top < m
            mu(top)= mu(top + 1) + pi_0(top + 1)/b(top + 1);
        end
    end
end

noInitExpected = BAS(b, w, rho, 1); %E[N(y-1)] + 1 where phi = pi
```

```
for i = 2:top
   j = top + 2 - i;
   mu(j - 1) = mu(j) + (pi_0(j) * p_i(j) + rho(j) * (1 - Phi_i(j))) * (w(j) * (noInitExpected(j - 1) - 1) + 1 - p_i(j))/(p_i(j) * (rho(j) *
w(j) + (1 - p_i(j)) * (b(j) + w(j))));
end
mu;
if vector == 1
   expected = mu;
else
   expected = mu(1);
end
```

```
function [R, v, b, w, rho, AARPexpectedConvergenceTimeDelta, BASexpectedConvergenceTimeinit] =
EstimateViaAARPHaR4(dimension, n, size, target)

%Different extrapolation method


for i = 1:dimension
   point(i) = rand(1);
end
point = pi * point;

value = f(point);
values = value;

for i = 1:n
  [point, value] = stepIHaR(point, value);
  values(i + 1) = value;
end


plot(values)


sortValues = sort(values);
distinct = sortValues(1);
for i = 2:n + 1
   if sortValues(i) > sortValues(i - 1)
      distinct = [distinct sortValues(i)];
   end
end
plot(distinct)

numberOfBetters = length(distinct);
for i = 1:length(distinct)
   if target < distinct(i)
      numberOfBetters = i - 1;
      break
   end
end

cutoff(1) = target;
if numberOfBetters > 0
   distinct = distinct(numberOfBetters + 1:length(distinct));
end

for i = 1:size - 2
   cutoff(i + 1) = distinct(floor(i * length(distinct)/(size - 1)) + 1);
%    cutoff(i + 1) = sortValues(1) + i * (sortValues(n + 1) - sortValues(1))/(size - 1);
end
%BUT WHAT IF IT DOES SAMPLE THE OPTIMUM? THEN THERE'S NO EXTRAPOLATION REQUIRED
%I've more or less allowed for this

R = zeros(size);
visited = 0;
for j = 1:size - 1
```

```
    if values(1) < cutoff(j)
       visited = 1;
       break
    end
 end
 if visited == 1
    to = j;
 else
    to = size;
 end

 for i = 2:n + 1
    from = to;
    visited = 0;
    for j = 1:size - 1
       if values(i) < cutoff(j)
          visited = 1;
          break
       end
    end
    if visited == 1
       to = j;
    else
       to = size;
    end
    R(from, to) = R(from, to) + 1;
 end

 go = 0;
          for j = 2:length(R)
             if sum(R(j,1:j-1)) + sum(R(j,j+1:size)) == 0;
                go = 1; %wait until there's enough data to estimate all of b and w
                break
             end
          end
 while go == 1
       [point, value] = stepIHaR(point, value);
   values(length(values) + 1) = value;
 sortValues = sort(values);
 if value < target
    numberOfBetters = numberOfBetters + 1;
 end

       from = to;
    visited = 0;
    for j = 1:size - 1
       if value < cutoff(j)
          visited = 1;
          break
       end
    end
    if visited == 1
       to = j;
    else
       to = size;
    end
    R(from, to) = R(from, to) + 1;
 n = n + 1;

    go = 0;
          for j = 2:length(R)
             if sum(R(j,1:j-1)) + sum(R(j,j+1:size)) == 0;
                go = 1; %wait until there's enough data to estimate all of b and w
                break
             end
          end

 end


 R(1,:) = [1 zeros(1, size - 1)];
 v = sum(R(:,2:size));
 v = v/sum(v);

 for i = 2:length(R)
    R(i,:) = R(i,:) / sum(R(i,:));
 end


 [b w rho] = AARPtoBAS(R, R, eye(size), v, Periodfinder(R)); %rho estimate no good here


 v2 = sum(R);

 fit = zeros(1,size);
 if numberOfBetters == 0
    x = (sortValues(1) + cutoff(2))/2;
    rho(2) = v2(2)/(cutoff(2) - sortValues(1));
    for i = 2:size - 2
       x(i) = (cutoff(i) + cutoff(i + 1))/2;
       rho(i + 1) = v2(i + 1)/(cutoff(i + 1) - cutoff(i));
    end
    x(size - 1) = (cutoff(size - 1) + sortValues(n + 1))/2;
    rho(size) = v2(size)/(sortValues(n + 1) - cutoff(size - 1));


    rho = rho/sum(rho);
    plot(x, rho(2:size))
    rho = [0 log(rho(2:size))];
    plot(x, rho(2:size))


    order = 3;
    p = polyfit(x, rho(2:size), order);
    for i = 1:size - 1
       for j = 0:order
          fit(i + 1) = fit(i + 1) + (x(i))^j * p(order + 1 - j);
       end
    end
    for j = 0:order
       fit(1) = fit(1) + target^j * p(order + 1 - j);
    end
    rho(1) = fit(1);
%rho(1) = rho(1)/10;
%   total = sum(rho);
%   fit = fit/total;
%   rho = rho/sum(rho);
    x = [target x];
    plot(x,[rho; fit]')


    rho = exp(rho);
fit = exp(fit);
   total = sum(rho);
   fit = fit/total;
   rho = rho/sum(rho);
plot(x,[rho; fit]')

AARPexpectedConvergenceTimeDelta = 0;
else
   AARPexpectedConvergenceTimeDelta = Expected(R, rho)
end

BASexpectedConvergenceTimeinit = BASinit2(b, w, rho, rho, 0);
%AndTheErrorIs = expectedConvergenceTime - BASexpectedConvergenceTimeinit
%RelativeError = AndTheErrorIs/expectedConvergenceTime
```

```
function [newpoint, newvalue] = StepIHaR(point, value)

%Simulates a transition from point according to Hit and Run and accepts it if it improves or with probability 0.1 if it doesn't. It's
%set up specifically for points in at least two dimensions with each coordinate between 0 and pi.

%x = rand(1) * pi;
%y = rand(1) * pi;

%checked

dimension = length(point);

angle = pi * rand(1);
direction(1) = sin(angle);
direction(dimension) = cos(angle);
for i = 2:dimension - 1
   previousAngle = angle;
   angle = pi * rand(1);
   direction(i) = direction(i - 1) * sin(angle)/tan(previousAngle);
   direction(dimension) = direction(dimension) * cos(angle);
end


%check = direction * direction';


while(1)
   distance = sqrt(dimension) * pi * (2 * rand(1) - 1);
   newpoint = point + distance * direction;
   if min(newpoint) >= 0
      if max(newpoint) <= pi
         break
      end
   end
end
newvalue = f(newpoint);
if newvalue > value
   randomnumber = rand(1);
   if randomnumber >= 0.1
      newpoint = point;
      newvalue = value;
   end
end
```