Simulating Population Genetics on the XT5

E. A. Duenez-Guzman, A. D. Vose, M. D. Vose, S. Gavrilets
The University of Tennessee, Knoxville

ABSTRACT: We describe our experience developing custom C code for simulating evolution and speciation dynamics using Kraken, the Cray XT5 system at the National Institute for Computational Sciences. The problem's underlying quadratic complexity was problematic, and the numerical instabilities we faced would either compromise or else severely complicate large-population simulations. We present lessons learned from the computational challenges encountered, and describe how we have dealt with them within the constraints presented by hardware.

KEYWORDS: simulating population genetics, numerical integration, residue calculus

1 Introduction

Simulating natural selection, genetic drift, mutation, and gene flow can be computationally intensive. The model developed by Sergey Gavrilets in collaboration with Jonathan Losos (Harvard University) to investigate adaptation and speciation in Anolis lizards has underlying quadratic complexity with respect to both deme size (all-pairs interactions are important to within-deme mating and viability) and spacial extent (area – hence the number of demes – varies quadratically with radius).

Trading deme size for more demes within a given area is an appealing way to tame the complexity, if one has access to computational resources enabling the processing of demes in parallel by mapping each deme to a processor.

This strategy is not a panacea, however. Between-deme migration of genetic material comes at the price of interprocess communication. Unless ecological properties are uniform from one deme to the next, there is also the price of increased model complexity. Moreover, the dynamics of selection, genetic drift, and gene flow are influenced by deme size and the number of demes. Nevertheless, mapping demes to processors on hardware like the Cray XT5 system at the National Institute for Computational Sciences makes simulations feasible on a scale which otherwise would be impossible.

From an implementation perspective, a natural unit of computation in the Anolis model is the evaluation of a double integral

$$I(\alpha, \beta, \gamma, \delta, \xi)$$

It turns out that the various probabilities required to simulate the discrete-time, explicit-genetics, individual-based, stochastic model can be obtained from the integral, where various rational functions of additive phenotypic characters of individuals and model parameters determine the arguments $\alpha, \beta, \gamma, \delta, \xi$.

2 Canned Routines

Integration subroutines provided by computation packages and scientific libraries are widely used, but it is a mistake to suppose they can be trusted. An incorrect answer may be returned (in some cases accompanied by a small error estimate), misleading the caller. For example, we naively believed Maple would give a reasonable answer using 16 digits of decimal precision (default precision is 10).

```
> Digits := 16;
> I_(0.91381,0.095649,0.57591,4.1584,1.4782);

      0.00097...
```

Thinking that result too small, we tried repeatedly but doubled each time the number of digits:

```
> Digits := 32;
> I_(0.91381,0.095649,0.57591,4.1584,1.4782);

      0.00687...

> Digits := 64;
> I_(0.91381,0.095649,0.57591,4.1584,1.4782);

      0.0219...

> Digits := 128;
> I_(0.91381,0.095649,0.57591,4.1584,1.4782);

      0.0322...
```

The relative error of the last answer – compared with the true answer – is over 6%, and Maple is excruciatingly slow.[1] Routines from NAG and the Gnu Scientific Library are comparable and relatively fast, taking 0.036 seconds to do the outer integral when provided with an explicit solution for the inner integral. But they return 0.00013... with estimated relative error below $2.0e-08$. That is unacceptable; we need the *correct* answer 0.034... and we need it produced *thousands* of times faster!

By interchanging the order of integration and hand-coding 128-point Gauss-Laguerre quadrature, we achieved results faster, which *seemed* accurate. Alas, our application was later observed to generate integrals for which that method yielded answers having more than 160% relative error. To make matters worse, it had become apparent that significant additional speedup was essential.

3 Complex Variables

Cauchy's integral theorem is a well-known tool for integration. A lesser-known method described by Boas and Schoenfeld [1] dispenses with having to choose suitable contours by focusing instead on residues of functions related to the integrand defined over the Riemann sphere. They discuss integrals of three types, the second of which is covered by the following theorem.

Theorem 1. *Let F be holomorphic in the extended plane except for a finite number of singularities, let F be holomorphic on (a,b) except for simple poles, and let F be holomorphic at a and at b. Then*

$$P.V. \int_a^b F(t)\,dt \;=\; -(R+r)$$

where R is the sum of the residues of

$$F(z)\log\{(z-a)/(z-b)\}$$

for z in the extended plane but not on $[a,b]$, and r is the sum of the residues of

$$F(z)\log\{(z-a)/(b-z)\}$$

for z on (a,b).

Our integration problem can be reduced to evaluating the form

$$\int_u^v \exp(-(\frac{t-a}{b})^2 + (t^{-1}-d)^2)\,\mathrm{erfc}(t^{-1}-d)\,\frac{dt}{t}$$

Difficulty using Theorem 1 arises from the integrand's essential singularities at 0 and ∞. Because determining residues at essential singularities for general parameter values a, b, d, u, v is problematic, we first approximate the integrand by functions which are meromorphic on the Riemann sphere (i.e., rational [3]).

This approach introduces other difficulties. Suitably approximating the integrand involves a trade-off between either the number of poles or complexity of residues and the number of sub intervals involved; a rational approximation suitable over all of (u,v) – one sub interval – would have high degree – either many poles or computationally expensive residue(s).

Fortunately, exp satisfies a functional equation enabling approximation with a single rational function (at the potential price of many sub intervals of (u,v)):

$$\begin{aligned} \exp(-f(t)) &= \exp(s)\exp(-s-f(t)) \\ &\approx \exp(s)\,\mathrm{rational}_0(f(t)+s) \quad (1) \end{aligned}$$

If $\mathrm{rational}_0(t)$ suitably approximates $\exp(-t)$ over $[x,y]$, the approximation above is valid for $t \in [u',v']$ provided f maps $[u',v']$ into $[x-s, y-s]$. By first subdividing (u,v) into sub intervals over which f is monotonic, we may assume f is invertible. Hence

$$\{u',v'\} = f^{-1}(\{x-s, y-s\}) \qquad (2)$$

and s parametrizes a sub range $t \in [u',v'] \subset [u,v]$ of integration. Both subdividing (u,v) into sub intervals over which f is monotonic and determining $[u',v']$ via (2) are done at run-time.

[1]For the example above, we discovered later that Maple returns a reasonable answer using default precision if specialized syntax is used. Unfortunately, that syntax only shifts problems elsewhere; unacceptable results are returned when using that syntax with other parameter values.

Approximations for $g(x) = \exp(x^2)\mathrm{erfc}(x)$ were based upon the value of $w = t^{-1} - d$:

$$\begin{array}{rll} 2\exp(w^2) & \text{if} & w \le -3 \\ \mathrm{rational}_1(w) & \text{if} & -3 \le w \le 0 \\ \mathrm{rational}_2(w) & \text{if} & 0 \le w \le 10 \\ \mathrm{rational}_3(w) & \text{if} & 10 \le w \end{array}$$

Except for $\mathrm{rational}_3$ which is the first two terms from the series expansion of g at ∞, we used Maple's implementation of the Remez algorithm (provided by the numapprox package) to obtain rational approximations (see [2] for a survey of widely used methods for generating rational or polynomial approximations to continuous functions). Our approximations are holomorphic in the sub intervals over which they are used, hence there is no contribution to the integral from r as described in Theorem 1. As functions of t, each of our rational approximations has a numerator with degree not exceeding that of its denominator. Therefore

$$-t^{-2}\,\mathrm{rational}(t^{-1})/t^{-1} \log \frac{1 - ta}{1 - tb}$$

is holomorphic at $t = 0$, hence there is no contribution from poles at ∞ to R as described in Theorem 1.

The most complicated residues arise from sub intervals involving both $\mathrm{rational}_0$ and $\mathrm{rational}_3$; residues are needed at $t = 1/d$ for

$$\mathrm{rational}_3(t^{-1} - d)/t \, \frac{\xi}{((t-a)/b)^2 + s - \zeta} \log \frac{t - u'}{t - v'}$$

Simpler residues arise from sub intervals involving $\mathrm{rational}_0$ and $\mathrm{rational}_k$ for $k \in \{1, 2\}$; residues are needed at $t = a \pm b\sqrt{\zeta - s}$ for

$$\mathrm{rational}_k(t^{-1} - d)/t \, \frac{\xi}{((t-a)/b)^2 + s - \zeta} \log \frac{t - u'}{t - v'}$$

and at $t = 1/(d + \zeta)$ for

$$\mathrm{rational}_0(((t-a)/b)^2 + s) \, \frac{\xi}{t^{-1} - d - \zeta} \, \frac{1}{t} \log \frac{t - u'}{t - v'}$$

These residues are obtained easily via the following lemmas (Lemma 3 is from [1]). The first lemma allows one to focus on the factor containing the singularity, the second deals with forms having singularities – such as $\xi/(z + s - \zeta)$ – that have been complicated by function composition – such as $z = ((t-a)/b)^2$.

Lemma 2. *Let ψ have a simple pole at ζ with residue ξ. If ϕ is holomorphic at ζ, then the residue of $\psi(z)\phi(z)$ at $z = \zeta$ is $\xi\phi(\zeta)$*

Lemma 3. *Let ζ be a point of the Riemann sphere where either $\phi'(\zeta) \ne 0$ or else ϕ has a simple pole. Let $\omega = \phi(\zeta)$ and let ψ either be holomorphic at ω or have an isolated singularity there. If φ is a local inverse of ϕ in a neighborhood of ω, then the residue of $\psi(\phi(z))$ at $z = \zeta$ is equal to the residue of $\psi(z)\varphi'(z)$ at $z = \omega$.*

The residues arising from sub intervals involving $\mathrm{rational}_0$ alone are those of

$$\frac{\xi}{((t-a)/b)^2 - (t^{-1} - d)^2 + s - \zeta} \, \frac{1}{t} \log \frac{t - u'}{t - v'}$$

and are easily obtained using Lemmas 2 and 3. The product of the first two factors in the expression above is equal to

$$\frac{b^2\xi}{t^4 - 2at^3 + (a^2 - b^2(\zeta + d^2 - s))t^2 + 2b^2dt - b^2}\, t$$

Focusing on the first factor (via Lemma 2), we take ψ and ϕ of Lemma 3 to be

$$\begin{aligned} \psi(z) &= \frac{b^2\xi}{z - b^2} \\ \phi(z) &= z^4 - 2az^3 + (a^2 - b^2(\zeta + d^2 - s))z^2 + 2b^2dz \end{aligned}$$

The ζ of Lemma 3 are roots of $\phi(z) = b^2$, the $\varphi'(\omega)$ of Lemma 3 is $1/\phi'(\zeta)$ (which follows from the chain rule, since φ is a local inverse to ϕ), and ω is b^2.

In summary: *residues are precomputed for parameter-free rational approximations to the integrand, parameters are incorporated by way of function composition, and their influence on residues is computed at run-time (via Lemma 1 and Lemma 2).*

If A approximates a positive integrand F, then

$$\begin{aligned} \left|1 - \frac{\int A}{\int F}\right| &= \left|\frac{\int F\,\{1 - A/F\}}{\int F}\right| \\ &\le \frac{\int F\,|1 - A/F|}{\int F} \\ &\le \|1 - A/F\|_\infty \end{aligned}$$

To achieve an integration tolerance of 1% relative error, we conservatively used rational approximations having less than 0.1% relative error. Underpinning assumptions of our calculation (like $\phi'(\zeta) \ne 0$ in Lemma 3 for instance) are checked at run-time. Typical time to evaluate $I(\alpha, \beta, \gamma, \delta, \xi)$ by way of residues is 0.000048 seconds (on a single 2.5 GHz AMD K10 core).

4 Lazy Evaluation

Having achieved acceptably accurate integration, subsequent optimism was based on the expectation that the carrying capacity of a deme acting together with selective pressure would constrain speciation and limit the number of phenotypes. Since the number of distinct integrals was quadratic in the number of distinct phenotypes, integrals were computed as needed and saved for later use to avoid recomputation.

We implemented a multi-level least-recently-used caching scheme using threaded splay trees. The first level stored integrals associated with a phenotype and created a perfect hash h_p for each phenotype p in the cache. The second level had keys of the form $\langle h_p, h_{p'} \rangle$ and stored integrals related to the interaction of p with p'.

This worked well enough to enable simulations not previously possible. However, as the number of simulated generations increased, so too did the number of phenotypes, and it became apparent that the limited amount of memory per node did not permit a sufficiently large second-level cache (we began development on Kraken before the upgrade; it previously had 1GB of memory per core, and not all of that was dedicated to the process running on the core). Cache thrashing was destroying performance for long runs, and eliminating the cache was not an option; computing integrals without reusing them was too slow.

To increase second-level cache size, we implemented it as a distributed cache. Whereas that scheme in some sense eliminated the recomputation of integrals as a bottleneck, it exacerbated what problems we faced with communication. Early versions of our code – computing integrals on demand without caching – were computation-bound, but could keep a parallel machine 90% busy. Later versions of our code – using a mulit-level distributed cache – ran faster and extended the range of simulations, but were communication-bound, and utilization was typically less than 10%.

5 Equivalence

We employed the following equivalence principle to recast parts of our simulation as a computation dealing with equivalence classes. If $f : A \to B$ is a nonempty function, the set

$$\mathcal{C} \;=\; \{f^{-1}(b) \,|\, b \in B\}$$

is a partition of A, and the corresponding relation

$$\equiv \;=\; \{(a,b) \,|\, f(a) = f(b)\}$$

is an equivalence relation on A. The most expensive part of our simulation (measured either in time or in cache space) was the computation of

$$\sum_{I'} \int \Xi(I|u,v)\, \Xi(I'|u,v)\, d\lambda(u,v) \;=\; \sum_{I'} \mathrm{Ne}(I,I')$$

where I and I' are genotypes. Defining equivalence between genotypes by

$$I \equiv J \iff \Xi(I|u,v) = \Xi(J|u,v)$$

makes Ne well-defined on equivalence classes c, c' by

$$\mathrm{Ne}(c,c') = \mathrm{Ne}(I,I') \quad \text{where} \quad I \in c,\; I' \in c'$$

Let C be the set of equivalence classes, let $I \in c \in \mathcal{C}$, and let [*expression*] denote 1 if *expression* is true, and 0 otherwise. It follows that

$$\begin{aligned}
\sum_{I'} \mathrm{Ne}(I,I') &= \sum_{I'} \mathrm{Ne}(I,I') \sum_{c' \in \mathcal{C}} [I' \in c'] \\
&= \sum_{c' \in \mathcal{C}} \sum_{I'} \mathrm{Ne}(I,I') [I' \in c'] \\
&= \sum_{c' \in \mathcal{C}} \sum_{I'} \mathrm{Ne}(c,c') [I' \in c'] \\
&= \sum_{c' \in \mathcal{C}} \mathrm{Ne}(c,c') \sum_{I'} [I' \in c'] \\
&= \sum_{c' \in \mathcal{C}} \mathrm{Ne}(c,c') \;\; |\, c' \,|
\end{aligned}$$

where $|\; c' \;|$ denotes the size of class c'. The advantage is that the number of nonzero terms (i.e., integrations) in the last sum above is significantly smaller than the number of terms in the first sum above.

We used the equivalence principle in previous implementations, based on the observation that Ne is well-defined on phenotypes (hence complexity was quadratic in phenotypes). Phenotypic equivalence is a *refinement* of the genotypic equivalence defined above, however, and it was a mistake to have not used the coarsest equivalence compatible with the computation.

We rewrote our simulation so as to take full advantage of the equivalence principle wherever possible (using the coarsest compatible equivalence relation). This reduced memory requirements and also resulted in significant speedup.

6 Precomputation

Although reimplementation (as described in the previous section) significantly improved performance (but utilization remained low), large-scale simulations were infeasible. Moreover, it was not sufficient to perform a single run for given parameter settings; many runs were required to obtain average behavior because of the stochastic nature of the computation. Thus however long a single simulation might be, obtaining usable results took many times longer.

Further progress could be made if both the efficiency and the effective size of the integral cache could be increased by:

- Reducing the run time and run space *overhead* for caching integrals.
- Reducing cache *size* by representing integrals as floats instead of doubles.
- Combining the memory devoted to processors on a node for caching integrals, and sharing it among the processors on that node.
- Eliminating recomputation. Computing integrals at run time for insertion into an empty cache is wasteful; each of the many runs required to obtain average behavior repeats that effort.

This was achieved by precomputing the values of integral-based functions of equivalence classes, and memory mapping the read-only file of results. That file is seen by the processes as a shared read-only cache. Cache access is (from the application's point of view) a simple array look-up. We essentially co-opted the operating system's i/o and memory system to manage caching for us. Moreover, run time previously spent in recomputing integrals for cache insertion was thereby eliminated and cache-thrashing became a non-issue.

To get a rough idea of the workload for a small-sized simulation, consider a 32×32 patch of demes (1024 demes total), with an average of $4,150$ children per deme per generation for a $100,000$ generation epoch. A *naive* implementation – computing integrals on demand (no caching and no equivalence class optimizations) – would need to plow through approximately $5,344,509,440,000,000$ integrations, nearly all of which are recompuation. Under optimistic assumptions – 90% utilization and no down time – Kraken's $66,048$ compute cores would take over 1.8 *months* to finish a single small-sized simulation. Performing ten runs to estimate average behavior would be difficult – over 1.5 years – and if the implementation used canned integration routines, the results could be meaningless.

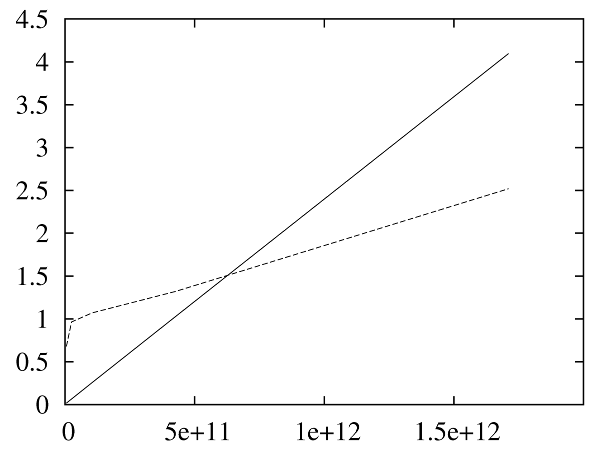
In contrast, completing ten small-sized runs in parallel using $10,250$ cores ($1,025$ cores for each run) takes under 1.4 hours using our optimized implementation, including precomputation time. Moreover, we have some degree of confidence in the results.

The key question that has yet to be addressed relates to the complexity of the genome. As the complexity of the genome increases, the number of equivalence classes increases, and both simulation time and the number of precomputed integrals are quadratic in the number of classes. Because the memory per node is severely limited (16GB maximum), a compute node will run out of memory pages with which to efficiently map the file of integrals, and thrashing will set in as the complexity of the genome increases.

The number of distinct equivalence classes in the Anolis model is $(2b + 1)^4$ where b is the bit-complexity of each gene. Time given above for a small-sized simulation is with $b = 4$. It is doubtful that simulations for $b > 8$ would be feasible without restructuring the implementation to leverage the aggregate memory distributed across many nodes by implementing a distributed-memory memory mapping scheme.

7 Scaling

The following graph gives some indication of how our simulation scales (the bit-complexity of each gene was $b = 4$). The line emanating from the origin plots the number of compute cores in units of $1,000$ (y-axis) against the number of individuals simulated (x-axis). The broken-line plots the completion time in hours (y-axis) against the number of individuals simulated (x-axis).



The model parameter that varied to generate the graph was the number of demes (which is one less than the number of compute cores). The number of generations was fixed at 100,000 because that is the epoch of interest (the completion time is linear in generations – e.g., two epocs takes twice as long).

Linearly approximating the graph data yields: time to complete an epoch as a function of the number d of demes (d = compute nodes minus one)

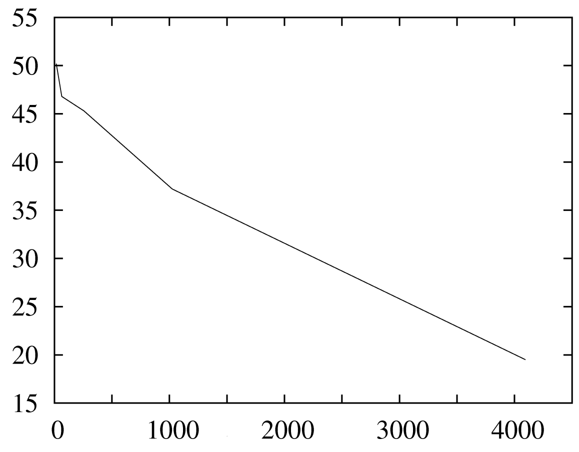
$$.9181 + 0.000391 * d$$

and number of individuals simulated per epoch as a function of d

$$418629838 * d - 3749585620$$

By extrapolation (which is probably over optimistic) using all of Kraken's 66,048 cores to simulate an epoch would process over 27.6 trillion individuals in under 26.8 hours. This represents the evolution of 66,047 demes at an approximate rate of 1 second per generation, where each deme yields 4,179 children per generation and the bit-complexity of a gene is 4.

We used the CrayPat performance analysis infrastructure to estimate cpu-utilization. It decreases as the number of demes increases. The following graph plots percent cpu-utilization (y-axis) against the number of demes (x-axis).

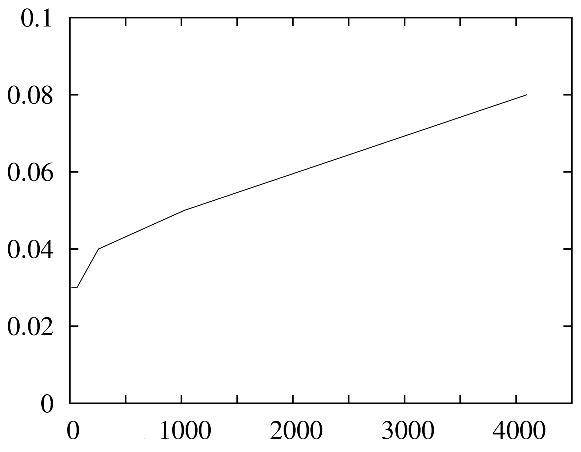


8 Tuning

In a single run, over 2MB of genetic material per deme per epoch are funneled to a single output node to be logged to disk. That corresponds to genes having low bit-complexity ($b = 4$), and communication scales linearly with bit-complexity.

Logging takes place every 1,000 generations (hence 100 times per epoch), and is comprised of serialized MPI transactions from *each* deme to the output node and subsequent writes from the output node to disk (the current implementation interleaves the MPI transactions with disk writes). Consequently, the simulation essentially waits until logging completes before resuming computation. We believe these MPI transactions and disk writes are the major factor responsible for the decreasing cpu-utilization displayed in the previous graph.

A different view of diminishing cpu-utilization is provided by the following graph, which plots average time per generation (y-axis) against the number of demes (x-axis).



A MPI_THREAD_FUNNELED implementation of logging has the potential to dramatically improve performance. The MPI transactions can be eliminated by buffering output at the deme level – rather than sending it to an output node – and a thread (at the deme level) can asynchronously write buffers to disk. That would have the additional advantage of parallel disk writes.

Rather than an extrapolated 1 second per generation corresponding to 66,047 demes, something more like the 0.04 seconds per generation corresponding to 256 demes in the graph above should be achievable. In that case – assuming 0.05 seconds per generation – the optimistic extrapolated time of 26.8 hours reported in the last section to simulate 66,047 demes for one epoch drops to 1.34 hours.

We have a good understanding of our simulation and its dynamics (through theory, experimentation, and embedded instrumentation) but have no explanation for the approximately linear increase in time per generation if it is not related to the logging described above (which the MPI_THREAD_FUNNELED approach should help alleviate).

9 Time warp

The Anolis model is spacial (demes correspond to vertices in a two dimensional grid) and requires between-deme migration of genetic material between nearest-neighbors at every generation. Interprocess communication must complete before computing the next generation.

We anticipate additional speedup – beyond that discussed above – could be achieved by implementing asynchronous migration so as to allow compute threads to proceed independent of whether migrants had arrived. This would produce a "time warp" in the sense that migrants from the *past* could eventually show up in the current generation much later than is credible (imagine injecting the genetic material of extinct organisms into an evolutionary system).

The potential utility of an asynchronous implementation relates to its speed. The parameter space of the Anolis model is too large to be adequately explored, but increased utilization of compute resources potentially enabled by an asynchronous implementation might allow for a more systematic investigation.

Whereas asynchronous runs do not yield usable results – we are not interested in how extinct genotypes would hypothetically influence the current direction of evolution – they nevertheless could help to identify interesting areas within the parameter space to be revisited with the intended *synchronous* model.

10 Acknowledgements

This work was supported by National Institutes of Health grant "New approaches to the modeling of speciation" (GM56693).

References

[1] R.P.Boas, L.Schoenfeld, Indefinite Integration by Residues, *SIAM Review*, v 8 n 2 (1966), 173-183.

[2] W.J.Cody, A Survey of Practical Rational and Polynomial Approximation of Functions, *SIAM Review*, v 12, n 3 (1970), 400-423.

[3] R.Nevalinna, V.Paatero, *Introduction to Complex Analysis*, AMS Chelsea Publishing (2007).

About the Authors

Dr. Sergey Gavrilets is a Distinguished Professor of Ecology & Evolutionary Biology and Mathematics, and Associate Director for Scientific Activities at the National Institute for Mathematical and Biological Synthesis. Edgar A. Duenez-Guzman is a research associate in the department of Electrical Engineering and Computer Science. Aaron D. Vose is a research assistant in the Gavrilets Lab. Michael D. Vose is a member of the department of Electrical Engineering and Computer Science.

email: gavrila@tiem.utk.edu, duenez@eecs.utk.edu, avose@eecs.utk.edu, vose@eecs.utk.edu