Checking Properties of Polynomials ⋆

(Extended Abstract)

Bruno Codenotti,[1] Funda Ergün,[2] Peter Gemmell,[3] and S Ravi Kumar[2]

[1] IMC-CNR, Via S. Maria 46, 56126-Pisa, Italy. (codenotti@imc.pi.cnr.it)
[2] Cornell University, Ithaca, NY 14853. ({ergun, ravi}@cs.cornell.edu)
[3] Sandia National Labs, Albuquerque, NM 87185. (psgemme@cs.sandia.gov)

Abstract. In this paper we show how to construct efficient checkers for programs that supposedly compute *properties* of polynomials. The properties we consider are roots, norms, and other analytic/algebraic functions of polynomials. In our model, both the program Π and the polynomial p are available to the checker each as a black box. We show how to check programs that compute a specific root (e.g., the largest) or a subset of roots of the given polynomial.
The checkers, in addition to never computing the root(s) themselves, strive to minimize both the running time (preferably $o(\deg^2 p)$) and the number of black box evaluations of p (preferably $o(\deg p)$). We obtain deterministic checkers when a separation bound between the roots is known and probabilistic checkers when the roots can be arbitrarily close. We then extend the checkers to handle the situations when the program Π returns an approximation to the root and when the evaluation of the polynomial p is approximate. Our results translate into efficient checkers for matrix spectra computations both in the exact and approximate settings, operating in the library model of [BLR93]. Next we show that the usual characterization of norms using the triangle inequality is not suited for self-testing in the exact case, but surprisingly, could be used in the approximate case.
Our results are complementary to most of the existing results on testing polynomials. The testers in the latter have the goal of determining whether a program computes a polynomial of given degree, whereas we are interested in checking the properties of a *given* polynomial.

1 Introduction

The paradigm of program checking and its extensions, self-testing, and self-correcting, have received considerable attention (e.g., [Blu88, BK89, BLR93, Lip91, GLR$^+$91, RS96, ABC$^+$93, GGR96, EKR96].) The results in this field

⋆ This work was done while the first, second, and fourth authors were visiting Sandia National Labs. The second and fourth authors are also supported by the NSF Career Award CCR-9624552, the Alfred P. Sloan Research Award, and the NSF grant DMI-91157199.

have practical value as tools for efficient verification of the correctness of programs. Furthermore, they have been applied to develop efficient probabilistically checkable proofs [ALM+92].

In this paper we investigate the problem of checking and testing (both in the exact and approximate cases) programs that compute *properties* (i.e., functions or relations) of polynomials. The properties we consider include the set of all roots, the largest root, the smallest root, norms, multiplication, differentiation, resultants, etc. Our checkers for root-finding problems only assume an oracle access to the polynomial p. Note that this is a weaker requirement than the availability of an explicit representation of p. This model lets us view the checkers for matrix spectra computations in the library setting of [BLR93]. In this framework, checkers call already tested programs in the library, counting each call as a unit time call. Such calls naturally correspond to the evaluation of the polynomial in our model. Consequently, it is imperative that the number of evaluations of p be minimized.

Our approach is complementary to previous work on checking and testing polynomials. The main difference is the following. Most of the existing results are concerned with checking/testing programs purportedly *evaluating* polynomials. In this paper we are interested in checking programs that take a polynomial as an input and compute its properties.

Our Results. We describe efficient checkers for programs that compute one, few, all, or specific roots (e.g., the largest) of a polynomial p. We address at length the checking of programs computing the largest root. For this problem, we construct some checkers that run in time $o(\deg^2 p)$ and make only $o(\deg p)$ calls to p (thus ruling out an explicit interpolation) using powerful tools from analysis. This translates into more efficient checkers than ones offered by several other methods that use explicit interpolation. We obtain deterministic checkers when a separation bound between the roots is known and probabilistic checkers when the roots can be arbitrarily close (Section 3). We also consider the situations where (i) the program $\Pi(p)$ is computing an approximation to the root(s) of polynomial p and (ii) the oracle returns an approximate evaluation of the polynomial p (Section 4). In these cases, we provide checkers for some of the problems.

Next we consider programs that claim to compute some (unspecified) norm on the domain (Section 5). There are several norms for polynomials (see [Z93]); the goal is to test whether there exists a norm that agrees with the program on most inputs. We show that the standard characterization of norms (using triangle inequality) cannot be used to construct exact testers. I.e., there are extremely "bad" programs (those that do not agree with any one norm for any non-trivial fraction of the inputs) that still pass the test. The same test, however, can be used to verify that the program *approximates* some norm for a non-trivial fraction of the domain. Our result, which applies to norms defined on any domain, is intriguing because most of the current techniques for testing use an exact characterization to build an exact tester and an approximate characterization (where the equalities are relaxed to approximations, see [ABC+93, EKR96] for further exposition) to build an approximate tester. The exact characterization for norms is too lenient to lead to an exact tester, however, surprisingly, is strong enough for an approximate one (even without resorting to an approximate characterization). Additionally, this is the first instance where an unbounded inequality (i.e., an inequality of the form $|h(\cdot)| > 0$, where h is an expression) has been addressed in testing.

The nature of these properties entails the use of techniques from several disciplines (like numerical and complex analysis, geometry, and in particular, geometry of polynomials) that are new to checking.

Applications. We show how to check programs that perform matrix spectra computations, which are fundamental in scientific computing (Section 6). We exploit the fact that the eigenvalues of a matrix are the roots of its characteristic polynomial. The characteristic polynomial is evaluated using a library program for the determinant that has been tested, for instance using the exact checker of [Kan90] or the approximate checker of [ABC$^+$93]. Several vital parameters in control theory (e.g., stability of a system) are related to the location of the roots of certain polynomials. Programs that compute these parameters are very common in practice [BCL82]; our checkers could be used to check such programs. Another application of property testing of polynomials is in verifying parts of computational algebra systems. We have taken an initial step in this direction but many interesting questions remain.

Previous and Related Work. The problem of testing root-finding programs is considered as early as 1975 in [JT75]. Here, the authors lay down some concrete requirements for an efficient testing of such programs. The setting proposed, however, is very different from ours and is mostly heuristic and informal.

A number of papers deal with testing whether a program is computing a low-degree polynomial in the exact [GLR$^+$91, AS92, GLR$^+$91, RS96] and approximate [EKR96] settings. Testing certain polynomial functions like polynomial multiplication and FFT is investigated in [BLR93, Erg95]. Checkers for several linear algebra computations like matrix rank, determinant, matrix multiplication are given in [Fre79, BK89, Kan90, BLR93]. Approximate testers for several linear algebra computations can be found in [ABC$^+$93]. Testing graph properties is considered in [GGR96].

2 Preliminaries

Our Model. We consider properties f of polynomials p. In this context, we assume that properties are relations such as those binding p to one or more of its roots. For shorthand, we sometimes use "$f(p)$" to denote *one* of the values to which f binds p.

Although checkers are defined for properties and are otherwise independent of the programs that they check, we sometimes refer to a checker for a program Π. Implicit in these references is that the checker is for the property f that Π purports to compute, i.e. that the checker verifies that $\Pi(x) \in f(x)$ for the input x in question.

Definition 1. Let Π be a program that purports to compute a property f. Let $\beta > 0$ be a security parameter. Then, a $(q(n), t(n); \epsilon_1, \epsilon_2)$-*checker* for f is a (probabilistic) oracle program $T^{\Pi,p}$ that has oracle access to both Π and p such that it

1. makes $O(q(n))$ oracle accesses to p (i.e., it evaluates p at $O(q(n))$ points)
2. runs in time $O(t(n))$, counting oracle calls as one unit of time
3. if $\exists y \in f(p) : |\Pi(p) - y| < \epsilon_1$, outputs "PASS" with probability $\geq 1 - \beta$
4. if $\forall y \in f(p) : |\Pi(p) - y| > \epsilon_2$, outputs "FAIL" with probability $\geq 1 - \beta$.

To simplify notation, we adopt the following conventions: (i) if $q(n) = t(n)$, we omit one of them, (ii) if $\epsilon_1 = \epsilon_2$, we omit one of them, and (iii) if $\epsilon_1 = \epsilon_2 = 0$, we omit both from the checker's parameters.

Note that the above model is more general than the standard checking model in that p is available as an oracle rather than in an explicit form. (It is often unrealistic or less efficient to assume that an explicit representation of p is available to T.) We will see that this model (i) captures the library setting of [BLR93] and helps us build efficient checkers, (ii) is useful in our applications to checking matrix spectra computations, and (iii) elegantly extends our checkers to the approximate setting.

Variations of the Model. Our model permits the following variations and their combinations: (i) The program purports to return an approximation to f. In this case, the program is denoted by $\tilde{\Pi}$. (ii) Each oracle call to evaluate p returns an approximation. In this case, the oracle is denoted by $\tilde{p}$. and (iii) p is "close" to a polynomial (as in the PCP setting). We will address the first and second variations. They make the problem more appealing since in practice we are seldom guaranteed an exact answer to any numerical question. In this paper, we will call a checker for the second scenario an *approximate checker*.

Self-Testing, Self-Correcting, Checking, and Libraries. Self-testing ensures that Π equals the target function f (from a function family F) on most inputs. A self-tester usually has two stages [BLR93]: (i) testing if Π is a member of F (the *property test*) and (ii) testing if Π is *the* specific member, i.e., f (the *equality test*). Self-correction involves taking a Π that is correct on most inputs and converting it into a program that is correct on all inputs. A self-tester together with a self-corrector gives a result-checker. In the library setting, a collection of previously checked programs is used to build checkers for new functions. For details see [BLR93].

Mathematical Notation. We consider polynomials over a field $\mathcal{F}$. Let $\mathbb{R}$ denote the real numbers and $\mathbb{C}$ denote the complex numbers.

Let $\mathcal{F}_n[x]$ denote the ring of polynomials of degree $\leq n$ with coefficients from $\mathcal{F}$. Let $p(\cdot)$ be a degree n polynomial (i.e., $p \in \mathcal{F}_n[x]$). Assuming p factors completely in $\mathcal{F}$, let the roots of p be $|\lambda_1| \geq \cdots \geq |\lambda_n|$. When $\mathcal{F} = \mathbb{R}$, we call p a *real polynomial* and if all the roots of p are real, we call p a *real-root polynomial*.

For any $a \in \mathbb{C}$, let $\overline{a} \in \mathbb{C}$ denote its complex conjugate. For any curve (line segment, interval) $\mathcal{C}$, let $|\mathcal{C}|$ denote its length and $\operatorname{int} \mathcal{C}$ its interior. For $x, y \in \mathbb{R}^2$, let $\overline{xy}$ denote the line segment between x and y. A *convex* curve in $\mathbb{R}^2$ is called a *contour* if it encloses the origin. A curve $\mathcal{C} : [0, 2\pi) \to \mathbb{R}^2$ is called *star-shaped* if it is an injective closed curve.

Let $p(x) = \prod_{i=1}^{n}(x - \lambda_i) = \sum_{i=0}^{n} a_i x^i, a_n = 1$. Then, it easily follows that $p'(x) = \sum_{i=1}^{n} \prod_{j \neq i}(x - \lambda_j)$.

$\mathbf{g}(\mathbf{x})$: We will use $\mathbf{g}(\mathbf{x})$ to denote $p'(x)/p(x) = \sum_{i=1}^{n} 1/(x - \lambda_i) = d \ln |p(x)|/dx$.

$\lambda_{\inf}, \lambda_{\max}$: Cauchy's inequality [BCL82] gives bounds on the roots of p as $\lambda_{\inf} = |a_n|/(|a_n| + \max_{i=0}^{n-1}\{|a_i|\}) < |\lambda_{\min}| \leq |\lambda_{\max}| < 1 + \max_{i=1}^{n}\{|a_i|\}/|a_0| = \lambda_{\sup}$.

δ: A separation bound between the roots of p is given by [BCL82] as $\delta = \min_{\lambda_i \neq \lambda_j} |\lambda_i - \lambda_j| > \sqrt{3} n^{-(n+1)/2} \|p\|^{1-n} \sqrt{\mathrm{disc}(p)}$, where the discriminant $\mathrm{disc}(p) = |\prod_{i \neq j}(\lambda_i - \lambda_j)| = |\operatorname{res}(p, p')|$ and $\|p\|^2 = \sum_{i=0}^{n} |a_i|^2$ [Z93]. Here, resultant $\operatorname{res}(p, q) = \prod_{i=1}^{\deg p} q(\lambda_i)$ where λ_i is a root of p. Some of our checkers assume that a lower bound on δ is known.

Problem Definitions. Let Π be a program that purports to compute one or more roots of p and let $\{\mu_i\}$ be the value(s) computed by Π. Let $\{\lambda_i\}$ be the actual root(s) that Π should have output. (Thus for instance, $\Pi_{\max}$, which purports to compute $\lambda_{\max}$, outputs $\mu_{\max}$ to be the largest root.) Given a polynomial p of degree n, let:

- $\mathcal{R}_1(p)$ be a relation mapping p to any one of its roots. We refer to programs that purport to compute a value $\mathcal{R}_1(p)$ as Π_1.
- $\mathcal{R}_r(p)$ be a relation mapping p to any r of its roots. We refer to programs that purport to compute a set $\mathcal{R}_r(p)$ as Π_r.
- $\mathcal{R}_{\langle k \rangle}(p)$ be the kth largest root of p in absolute value (i.e., λ_k). $\Pi_{\langle k \rangle}$ refers to a program that purports to compute $\mathcal{R}_{\langle k \rangle}$.
- $\mathcal{R}_{\max} = \mathcal{R}_{\langle 1 \rangle}, \mathcal{R}_{\min} = \mathcal{R}_{\langle n \rangle}$ and $\Pi_{\max}$, $\Pi_{\min}$ refer to programs that supposedly compute $\mathcal{R}_{\max}, \mathcal{R}_{\min}$.

In general, we use a tilde to denote programs that purport to return approximations to the corresponding exact relation (e.g., $\tilde{\Pi}_{\max}$).

3 Checking Roots: Exact Setting

Checkers for $\mathcal{R}_1, \mathcal{R}_r, \mathcal{R}_n, \mathcal{R}_{\max}$.

Theorem 2. *Let $|\mathcal{F}| \geq n + \Omega(n)$. There is:*

1. *a (1)-checker for $\mathcal{R}_1(p)$,*
2. *a $(1, n)$-checker for $\mathcal{R}_n(p)$, and*
3. *a $(\min\{r, n-r\}, \max\{r, n-r\})$-checker for $\mathcal{R}_r(p)$.*

In the exact setting, given $\mu_{\max}$, it is trivial to verify that it is *a root of* p. It is non-trivial to verify the maximality claim. Theorem 3 below states a checker for $\Pi_{\max}(p)$. In the next section, we will show more efficient checkers (that avoid explicit interpolation) for $\Pi_{\max}(p)$.

Theorem 3. *$\forall\epsilon > 0$, there is an $(n, n^2; \epsilon)$-checker for $\mathcal{R}_{\min}(p)$. If p is a real-root polynomial, $\forall\epsilon > 0$, there is an $(n, n^2; \epsilon)$-checker for $\mathcal{R}_{\max}(p)$.*

A checker for $\Pi_{\max}$ is constructed from a checker for $\Pi_{\min}$ in an obvious manner by observing that $1/\lambda_i$ are the roots of $x^n p(1/x)$. Note that the checkers given by Theorem 3 can also be used to check $\Pi_{\langle k \rangle}$.

Improved Checkers for $\mathcal{R}_{\max}$: δ known. For the rest of this section, we will take either $\mathcal{F} = \mathbb{C}$ or $\mathcal{F} = \mathbb{R}$. We use the following theorems from complex analysis (see [Con78]). Let $n(\mathcal{C}; z)$ be the number of times $\mathcal{C}$ "winds" around the point $z \in \mathbb{C}$.

Theorem 4 Cauchy's Residue Theorem. *Let G be an open subset of the plane and $f : G \to \mathbb{C}$ an analytic function. If $\mathcal{C}$ is a closed rectifiable (finite length) curve in G such that $n(\mathcal{C}; z) = 0\ \forall z \in \mathbb{C} \backslash G$, then for $\lambda \in G \backslash \mathcal{C}$, $2\pi i f(\lambda) n(\mathcal{C}; \lambda) = \int_{\mathcal{C}} f(z)/(z - \lambda) dz$.*

Corollary 5. *Let G be an open subset of the plane and f be an analytic function on G with zeros $\lambda_1, \ldots, \lambda_n$ (repeated according to multiplicity). If $\mathcal{C}$ is a closed rectifiable curve in G which does not pass through any point λ_k, and $n(\mathcal{C}; z) = 0, \forall z \in \mathbb{C} \backslash G$, then $\int_{\mathcal{C}} f'(z)/f(z)\, dz = 2\pi i \sum_{i=1}^{n} n(\mathcal{C}; \lambda_i)$ counts (with multiplicities) the number of roots of $f(z)$ within $\mathcal{C}$.*

Theorem 6. *There is a $((|\mu_{\max}|/\delta)^{3/2}; \delta/2)$-checker for $\mathcal{R}_{\max}(p)$.*

Proof. If $\mathcal{C}$ is a circle, then by Corollary 5, $\int_{\mathcal{C}} p'(z)/p(z) dz$ computes $2\pi i$ times the number of zeros of p that are within $\mathcal{C}$ (noting that $\mathcal{C}$ winds once around each root). So, our goal is to check that $\int_{\mathcal{C}} p'(z)/p(z) dz = 2\pi n$, where $\mathcal{C}(t) = (|\mu_{\max}| + \delta/2)e^{it}$. Recall that $g(z) = p'(z)/p(z)$. We compute an approximation S to $\int_{\mathcal{C}} g(z) dz$, which must satisfy $|S - 2\pi n| < \pi$. If we use trapezoidal rule, we have $\left|\int_{\mathcal{C}} g(z) dz - \sum_{i=1}^{N} \alpha_i g(z_i)\right| \leq (|\mathcal{C}|^3/N^2) \max_{z \in \mathcal{C}} |g''(z)|$, where α_i's are constants and N is the number of points of evaluation.

Since we can only approximate $p'(z)$, we actually end up computing $\sum_{i=1}^{N} \alpha_i (g(z_i) + \epsilon_i)$. Therefore, we can evaluate the overall error as

$$\left|\int_{\mathcal{C}} g(z) dz - \sum_{i=1}^{N} \alpha_i (g(z_i) + \epsilon_i)\right| \leq \frac{|\mathcal{C}|^3}{N^2} \max_{z} |g''(z)| + c_1 \epsilon |\mu_{\max}|$$

where c_1 is a small constant and $\epsilon = \max_i |\epsilon_i|$. Our goal is to find conditions under which $(|\mathcal{C}|^3/N^2) \max_z |g''(z)| + c_1 \epsilon |\mu_{\max}| < \pi$, such that rounding always gives the correct value. We first find a bound on ϵ for which $c_1 \epsilon |\mu_{\max}| < \pi/2$. If p' is approximated by finite differences, then $\epsilon = \max_\zeta |(p(\zeta + \Delta) - p(\zeta))/(\Delta p(\zeta)) - g(\zeta)| \leq \Delta p''_{\max}(\zeta')/p(\zeta)$, for some $\zeta' \in (\zeta, \zeta + \Delta)$. An upper bound on Δ is dictated by these conditions. Now, we have $p''_{\max}(\zeta')/p(\zeta) = \sum_{i \neq j} (\zeta - \lambda_i)^{-1} (\zeta - \lambda_j)^{-1} \prod_{k \neq i,j} (\zeta' - \lambda_k)/(\zeta - \lambda_k) \leq e^{n\Delta/\delta}(n^2/\delta^2)$. Thus, Δ must satisfy $\Delta \leq (c_2 \delta^2)/(n^2 |\mu_{\max}|)$, where c_2 is a small constant.

The other error term $(|\mathcal{C}|^3/N^2) \max_z |g''(z)|$, can now be upper bounded as $(|\mathcal{C}|^3/N^2) \max_z |g''(z)| \leq (c_3 |\mu_{\max}|^3)/(N^2 \delta^3)$, from which the number of evaluation points $N = O(|\mu_{\max}|/\delta)^{3/2}$.

For real-root polynomials, the number of oracle calls to p can be reduced by stronger bounds on $\max_z |g''(z)|$. The proof is omitted.

Corollary 7. *If p is a real-root polynomial, then there exists a $(\sqrt{n}|\mu_{\max}| + (1/\delta)^{1/2+o(1)}; \delta/2)$-checker for $\mathcal{R}_{\max}(p)$.*

If $\delta = O(1/n)$, then the above corollary yields asymptotically better checkers than those given by Theorems 3 and 6. We also give a different checker (proof omitted) that can be extended to work in the approximate setting.

Theorem 8. *There is a $((\lambda_{\sup}^2 - |\mu_{\max}|^2)/(\delta^{5/2}\epsilon); \delta/2)$-checker for $\mathcal{R}_{\max}(p)$ where $\epsilon \leq 1/|p''(x)|$, $\forall x$.*

Improved Checkers for $\mathcal{R}_{\max}$: δ unknown. We obtain the following checkers for the case when δ is not known. The proofs of the theorems are omitted.

Theorem 9. *Let p be a real-root polynomial. $\forall \epsilon, \beta > 0$, there is a $(\sqrt{n}\log^{3/2}(n/\epsilon)\log(1/\beta); \epsilon/2, \epsilon)$-checker for $\mathcal{R}_{\max}(p)$ that is correct with probability $\geq 1 - \beta$.*

The above theorem can be extended to the case when the roots are complex. The checker is still attractive in terms of its running time, but has more evaluations of p.

Corollary 10. *$\forall \epsilon, \beta > 0$, there is an $(n^{3/2}/\epsilon \log(1/\beta); \epsilon/2, \epsilon)$-checker for $\mathcal{R}_{\max}(p)$ that is correct with probability $\geq 1 - \beta$.*

The checkers in this section can be extended to check $\Pi_{\langle k \rangle}$.

4 Checking Roots: Approximate Setting

So far, we have been using the assumption that the programs being checked should return the exact root(s) and the oracle returns the exact value. As we stated in the description of our model, we have two variants – $\tilde{\Pi}$ and $\tilde{p}$. The former turns out to be easier than the latter.

Case I: $\tilde{\Pi}$. Suppose $\tilde{\Pi}_1(p)$ returns an ϵ-approximation (i.e., it claims $|\mu_i - \lambda_i| \leq \epsilon < \delta$). When p is real-root, with two oracle calls to p we can check if there is a sign-change in $[\mu_i - \epsilon, \mu_i + \epsilon]$. For $\tilde{\Pi}_n(p)$ (resp. $\tilde{\Pi}_r(p)$), we can extend the above checker with $2n$ (resp. $2r$) calls to p. Since we do not have a nice analog of Rolle's theorem in complex analysis, the problem becomes harder when p is not real-root.

All our checkers for $\mathcal{R}_{\max}$ in Section 3 can be extended to this approximate setting. This can be done as follows: (i) first we check if $\mu_{\max}$ is an approximate root and (ii) then we check if it is indeed the maximum root. The former is accomplished by checking if there is a root inside a small circle around $\mu_{\max}$ (see previous paragraph) and the latter is accomplished by selecting two curves separated by ϵ and then performing the numerical integration twice. Thus, we obtain the following theorem:

Theorem 11. *$\forall \epsilon, \beta > 0$, there is (i) a $((|\mu_{\max}|/\delta)^{3/2}; \delta/2)$-checker for $\tilde{\mathcal{R}}_{\max}(p)$ and (ii) an $(n^{3/2}/\epsilon \log(1/\beta); \epsilon/2, \epsilon)$-checker for $\tilde{\mathcal{R}}_{\max}(p)$ that is correct with probability $\geq 1-\beta$. If p is a real-root polynomial, $\forall \epsilon, \beta > 0$, there is a (i) $(\sqrt{n}|\mu_{\max}| + (1/\delta)^{1/2+\epsilon}; \delta/2)$-checker for $\tilde{\mathcal{R}}_{\max}(p)$, (ii) $(\sqrt{n}\log^{3/2}(n/\epsilon)\log(1/\beta); \epsilon/2, \epsilon)$-checker for $\tilde{\mathcal{R}}_{\max}(p)$ that is correct with probability $\geq 1-\beta$.*

Case II: $\tilde{p}$. Theorem 8 can be extended to the case when we have only $\tilde{p}$ (i.e., the evaluation of p is approximate). The proof is omitted.

Corollary 12. *There is a $((\lambda_{\sup}^2 - |\mu_{\max}|^2)/(\delta^{5/2}\epsilon); \delta/2)$-approximate checker for $\tilde{\mathcal{R}}_1(\tilde{p})$.*

The checkers in Section 3 are not directly usable in this case because of the instability of numerical derivative computation in the presence of errors.

5 Testing Norms

A function $f : \mathcal{V}(\mathcal{F}) \to \mathbb{R}^+$, where $\mathcal{V}$ is a vector space over $\mathcal{F}$, is called a *norm* if it satisfies: (i) $f(\mathbf{x}) = 0 \iff \mathbf{x} = \mathbf{0}$, (ii) $\forall \mathbf{x} \in \mathcal{V}, k \in \mathcal{F}, f(k\mathbf{x}) = kf(\mathbf{x})$ (*scalability*), and (iii) $\forall \mathbf{x}, \mathbf{y} \in \mathcal{V}, f(\mathbf{x}+\mathbf{y}) \leq f(\mathbf{x}) + f(\mathbf{y})$ (*triangle inequality*).

In this section we investigate the problem of checking whether the function computed by a program Π_{norm} is close to a norm (i.e., there is a norm that agrees with Π_{norm} on most inputs). In the specific case of vector p-norms on $\mathbb{R}^n$, which are of the form $|\mathbf{x}|_p = (\sum_{i=1}^n x_i^p)^{1/p}$, the problem reduces to the well-studied problem of multivariate degree-testing [AS92, RS96]. In fact, matrix spectral norms can be checked using our techniques in Section 3 and Section 4. In the more general case of checking whether the function is close to *any* norm, we show that the properties characterizing a norm are *not usable* for exact self-testing. This result is already interesting in that our tests are almost exactly the same as the standard linearity test except for an inequality in the second test. This, however, makes a big difference in the validity of the test, which leads us to believe that inequalities in general do not lead to (exact) self-testers. In a striking contrast, we show that these properties characterizing norms can lead to approximate self-testers. The following discussion is for $\mathbb{R}^2$ and can be extended to $\mathbb{R}^n$.

Exact Testing. To check scalability of Π_{norm}, note that along a vector $\mathbf{x}$, scalability defines the same set of functions as linearity. Checking $\Pi_{\text{norm}}(a\mathbf{x}) + \Pi_{\text{norm}}(b\mathbf{x}) = \Pi_{\text{norm}}((a+b)\mathbf{x})$ for $\mathbf{x}, |\mathbf{x}| = 1$ will determine if Π_{norm} is scalable along $\mathbf{x}$ (this is the linearity test of [BLR93]). By performing this test at many $\mathbf{x}$, we can ensure that Π_{norm} is scalable for many $\mathbf{x}$. Therefore, for the rest of this discussion, we can assume that Π_{norm} is scalable. $\forall i \in \mathbb{R}$, define the "concentric" contours $\mathcal{C}_i = \{\mathbf{x} \mid \Pi_{\text{norm}}(\mathbf{x}) = i\}$. We first show that checking the triangle inequality is equivalent to checking the convexity of $\mathcal{C}_i$ in $\mathbb{R}^2$ for any $i \in \mathbb{R}$.

Lemma 13. *Let f be a scalable function, i.e., $f(k\mathbf{x}) = kf(\mathbf{x})$. Then, $\exists \mathbf{a}, \mathbf{b} \in \mathcal{V}$, such that $f(\mathbf{a}+\mathbf{b}) > f(\mathbf{a}) + f(\mathbf{b}) \iff \forall i$, the i-th contour $\mathcal{C}_i$ is not convex (the non-convexity occurs along $\mathbf{a}+\mathbf{b}$.)*

We show in Theorem 14 that random sampling of condition 3 does not work: there are extremely "bad" programs that pass it.

Theorem 14. *$\forall 0 \leq \delta \leq 1$, there exists a scalable Π_{norm} that is at least δ away from the nearest convex function g, i.e., $\Pr_x[\Pi_{\text{norm}}(x) \neq g(x)] \geq \delta$, but Π_{norm} passes the test for condition 3 with arbitrarily high probability.*

Approximate Testing. In contrast to exact testing, we show that the properties characterizing norms can be used to test if a program approximately computes a norm at a non-trivial fraction of the inputs.

For a given star-shaped $\mathcal{C}$, let the diameter be $\operatorname{diam} \mathcal{C} = \sup_{t_2,t_2}\{|\mathcal{C}(t_1) - \mathcal{C}(t_2)|\}$. Given two curves $\mathcal{C}_1, \mathcal{C}_2$, let the distance between them be $|\mathcal{C}_1 - \mathcal{C}_2| = \sup_t\{|\mathcal{C}_1(t) - \mathcal{C}_2(t)|\}$. For two contours $\mathcal{C}_1, \mathcal{C}_2$ and for any other star-shaped $\mathcal{C}$, let the deviation measure be $\operatorname{dev}_{\mathcal{C}_1,\mathcal{C}_2}(\mathcal{C}) = \Pr_t[\mathcal{C}(t) \geq \mathcal{C}_1(t), \mathcal{C}_2(t) \text{ or } \mathcal{C}(t) \leq \mathcal{C}_1(t), \mathcal{C}_2(t)]$. This measures the fraction of $\mathcal{C}$ not lying between $\mathcal{C}_1$ and $\mathcal{C}_2$. For a star-shaped $\mathcal{C}$, let $\Delta = \Delta(\mathcal{C}) = \Pr_{s,t,z \in \overline{\mathcal{C}(s)\mathcal{C}(t)}}[z \notin \mathcal{C} \cup \operatorname{int} \mathcal{C}]$. In other words, Δ is the probability that, if we pick random $s, t \in \mathcal{C}$ and a random point z on the line joining them, then z lies outside $\mathcal{C}$. Testing condition 3 on random $\mathbf{x}, \mathbf{y}$, we can estimate Δ corresponding to the contour defined by Π_{norm} (assuming it is star-shaped, which is easy to check).

Theorem 15. *Given $\rho > 0$, $\exists \epsilon = \epsilon(\rho) < 1, \gamma = \gamma(\epsilon) > 0$ such that for any star-shaped $\mathcal{C}$ with $\operatorname{diam} \mathcal{C} \leq 1$, if $\Delta(\mathcal{C}) \leq \gamma$ then there is a contour $\overline{\mathcal{C}}$ such that $\Pr_t[|\overline{\mathcal{C}}(t) - \mathcal{C}(t)| > \rho] \leq \epsilon$.*

6 Some Applications: Matrix Computations

In this section, we show applications of our checkers for polynomial roots to matrix spectra computations. Let the eigenvalues of $\mathbf{A} \in \mathcal{F}^{n \times n}$ be $\Lambda(\mathbf{a}) = \{\lambda_i \mid 1 \leq i \leq n\}$ with $|\lambda_{\max} = \lambda_1| \geq \cdots \geq |\lambda_n|$. It is easy to find an upper bound $\lambda_{\sup}$ on $\lambda_{\max}$ (e.g., set $\lambda_{\sup} = \|\mathbf{A}\|_\infty$). We denote by δ a separation bound between the eigenvalues of $\mathbf{A}$. Let Det be a correct program available in the library for computing the determinant of a matrix. Det corresponds to the oracle p in our model.

Eigenvalues in the Exact Setting. All the checkers in Section 3 and Section 4 translate to checkers for eigenvalues. We now illustrate more efficient checkers for some special cases which are of interest in practice:

Lemma 16. *Let $\mathbf{A} \in \mathbb{R}^{n \times n}$ with $\mathbf{A} = \mathbf{A}^T$. There is an (n)-checker for program computing the largest eigenvalue of $\mathbf{A}$. If $\mathbf{A}$ is tridiagonal, there is an (n)-checker for a program computing the k-th largest eigenvalue of $\mathbf{A}$.*

These checkers can be used to check programs designed for computing the second largest eigenvalue of a regular graph (or the largest eigenvalue of its Laplacian), which is related to the expansion of the graph. Another natural application of these checkers is to check programs that decide whether a matrix is positive definite.

Eigenvalues in the Approximate Setting. All of our approximate checkers for roots can be used in this case. We consider an interesting special case of this problem. Let $\langle \lambda, \mathbf{x} \rangle$ be an exact eigenvalue–eigenvector pair of $\mathbf{A}$ with $\|x\| = 1$. Let $\tilde{\mathcal{R}}_1$ be a relation that binds matrix $\mathbf{A}$ to pairs $\langle \mu, \tilde{\mathbf{x}} \rangle$ with $|\mu - \lambda| < \epsilon_1$ and $\|\tilde{\mathbf{x}} - \mathbf{x}\| < \epsilon_2$. Let $\tilde{\Pi}_1$ be a program that, on input $\mathbf{A}$, purports to compute a $\langle \mu, \tilde{\mathbf{x}} \rangle \in \tilde{\mathcal{R}}_1(\mathbf{A})$. The most natural way of checking $\tilde{\mathcal{R}}_1$ would be by checking that $\|\mathbf{A}\tilde{\mathbf{x}} - \mu\tilde{\mathbf{x}}\| < \epsilon$, for a certain threshold ϵ, and passing $\tilde{\Pi}_1$ if the above inequality is satisfied. Unfortunately, from perturbation theory [GV89] we have that the value of ϵ above can be small, but $|\mu - \lambda|$ be as large as $\epsilon / |\mathbf{y}^H \mathbf{x}|$, where $\mathbf{y}^H$ is a unit length left eigenvector of $\mathbf{A}$ ($\mathbf{y}^H \mathbf{A} = \lambda \mathbf{y}^H$), assuming λ to be a simple eigenvalue. Thus, we might need to set ϵ to a very small value if we want to make sure that $|\mu - \lambda| < \epsilon_1$ for a reasonably small value ϵ_1. Note however that for normal matrices , $|\mathbf{y}^H \mathbf{x}| = 1$ so that $\|\mathbf{A}\mu - \tilde{\mathbf{z}}\mu\| < \epsilon \Rightarrow |\lambda - \mu| < \epsilon$. Thus, we have the following lemma:

Lemma 17. *$\tilde{\mathcal{R}}_1(\mathbf{A})$ can be approximately checked when $\mathbf{A}$ is normal.*

In general, if we do not make assumptions on the problem condition, the approximate checker may yield very poor bounds. This is because the determinant of a matrix can be very close to zero (e.g., $1/2^n$) despite all eigenvalues being well-separated from zero (e.g., $\lambda_i = 1/2$).

Singular Values. Suppose MULT is a correct library program for matrix multiplication. If Π_{sing} is a program that purports to compute the singular values $\sigma_1, \ldots, \sigma_n$ of $\mathbf{A}$, construct a checker for Π_{sing} as follows: (i) check if $\sigma_i \geq 0, 1 \leq i \leq n$, (ii) compute the matrix $\mathbf{A}^T\mathbf{A} \in \mathcal{F}^{n \times n}$ using MULT, and (iii) use the checkers for eigenvalues to verify if $\{\sigma_1^2, \ldots, \sigma_n^2\} = \Lambda(\mathbf{A}^T\mathbf{A})$. The correctness of this construction is from the definition of singular values (see [GV89]).

7 Further Work

All of our checkers are assumed to perform exact arithmetic. This assumption is not always true in practice. It will be interesting to design checkers when the checker's numerical errors are critical. Many issues are still unresolved in the case of $\tilde{p}$. Are there efficient checkers for programs that compute Gröbner bases, programs that solve Diophantine problems and lattice problems? Such checkers would find numerous applications in computational algebra systems. Can we get efficient checkers for sparse-matrix computations?

Acknowledgements

We thank Richard Allen (Sandia), Bruce Hendrickson (Sandia), Mahan Mitra (Berkeley), Ronitt Rubinfeld (Cornell), Nick Trefethen (Cornell), Divakar Vishwanath (Cornell), and Richard Zippel (Cornell) for interesting discussions and suggestions.

References

[ABC+93] S. Ar, M. Blum, B. Codenotti, and P. Gemmell. Checking approximate computations over the reals. *Proc. 25th STOC*, pp. 786–795, 1993.

[ALM+92] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and hardness of approximation problems. *Proc. 33rd FOCS*, pp. 14–23, 1992.

[AS92] S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of NP. *Proc. 33rd FOCS*, pp. 2–13, 1992.

[Blu88] M. Blum. Designing programs to check their work. TR 88-009, ICSI, 1988.

[BK89] M. Blum and S. Kannan. Program correctness checking ... and the design of programs that check their work. *Proc. 21st STOC*, pp. 86–97, 1989.

[BLR93] M. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. *JCSS*, 47(3):549–595, 1993.

[BCL82] B. Buchberger, G.E. Collins, and R. Loos. *Computer Algebra – Symbolic and Algebraic Computation.* Springer–Verlag, 1982.

[Con78] J.B. Conway. *Functions of One Complex Variable.* Springer–Verlag, 1978.

[Erg95] F. Ergün. Testing multivariate linear functions: Overcoming the generator bottleneck. *Proc. 27th STOC*, pp. 407–416, 1995.

[Fre79] R. Freivalds. Fast probabilistic algorithms. *Proc. 8th MFCS*, LNCS 74, pp. 57–69, 1979.

[EKR96] F. Ergün, S. Ravi Kumar, and R. Rubinfeld. Approximate checking of polynomials and functional equations. *Proc. 37th FOCS*, To appear.

[GLR+91] P. Gemmell, R. Lipton, R. Rubinfeld, M. Sudan, and A. Wigderson. Self-testing/correcting for polynomials and for approximate functions. *Proc. 23rd STOC*, pp. 32–42, 1991.

[GGR96] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *Proc. 37th FOCS*, pp. 339–348, 1996.

[GV89] G.H. Golub and C. Van Loan. *Matrix Computations.* Johns Hopkins U. Press, 1989.

[JT75] M.A. Jenkins and J.F. Traub. Principles for testing polynomial zero-finding programs. *ACM Trans. on Mathematical Software,* 1:26–34, 1975.

[Kan90] S. Kannan. *Program Result Checking with Applications.* PhD thesis, U. of California at Berkeley, 1990.

[Lip91] R. Lipton. New directions in testing. *Proc. DIMACS Workshop on Distributed Computing and Cryptography*, pp. 191–202, 1991.

[Mar66] M. Marden. *Geometry of Polynomials.* AMS, 1966.

[PTV92] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C.* Cambridge U. Press, 1992.

[RS96] R. Rubinfeld and M. Sudan. Robust characterizations of polynomials and their applications to program testing. *SICOMP*, 25(2):252–271, 1996.

[Wil65] J. H. Wilkinson. *The Algebraic Eigenvalue Problem.* Clarendon Press, 1966.

[Z93] R.E. Zippel. *Effective Polynomial Computation.* Kluwer Academic Press, 1993.