AMS-SIAM Special Session on Symbolic-Numeric Computation and Applications

A Comparison of Heuristics for Solving Problems in Approximate Polynomial Algebra

John May

(Joint work with Mark Giesbrecht)

University of Waterloo

Waterloo, Ontario, Canada

Approximate Polynomial Problems

For a set of polynomials $F = \{f_1, f_2, \ldots, f_n\}$ and a property of a set of polynomials $\mathfrak{P}$:

e.g. $F = \{f\}$ and $\mathfrak{P} = f$ is factorizable.

e.g. $F = \{g, h\}$ and $\mathfrak{P} = g$ and h have a non-trivial GCD

Approximate Polynomial Problems

For a set of polynomials $F = \{f_1, f_2, \ldots, f_n\}$ and a property of a set of polynomials $\mathfrak{P}$:

e.g. $F = \{f\}$ and $\mathfrak{P} = f$ is factorizable.

e.g. $F = \{g, h\}$ and $\mathfrak{P} = g$ and h have a non-trivial GCD

Optimization Problems:

Full optimization (hard): Given F compute the nearest (in the 2-norm) set $\tilde{F} = \{\tilde{f}_1, \tilde{f}_2, \ldots \tilde{f}_n\}$ to F so that $\deg \tilde{f}_i \leq \deg f_i$ and $\tilde{F}$ has property $\mathfrak{P}$

"Soft" approximate problem: Given F, find a $\tilde{F}$ so that $\tilde{F}$ has property $\mathfrak{P}$, $\sum_i \| f_i - \tilde{f}_i \|_2^2$ is relatively small and $\deg \tilde{f}_i \leq \deg f_i$

Linearization of Polynomial Problems

Many polynomial problems can be reformulated as computing the nullspace of a structured matrix

Linearization of Polynomial Problems

Many polynomial problems can be reformulated as computing the nullspace of a structured matrix

This may not be efficient for exact computation, but lends itself well to approximate computation

Linearization of Polynomial Problems

Many polynomial problems can be reformulated as computing the nullspace of a structured matrix

This may not be efficient for exact computation, but lends itself well to approximate computation

For example, compute $\gcd(f,g)$ becomes: find the "smallest" non-zero vector in the nullspace of $\mathrm{Syl}(f,g)$

Linearization of Polynomial Problems

Many polynomial problems can be reformulated as computing the nullspace of a structured matrix

This may not be efficient for exact computation, but lends itself well to approximate computation

For example, compute $\gcd(f,g)$ becomes: find the "smallest" non-zero vector in the nullspace of $\mathrm{Syl}(f,g)$

Approximate algebra version of GCD: find the nearest non-relatively prime pair $\tilde{f},\tilde{g}$ to f,g and compute $\gcd(\tilde{f},\tilde{g})$

Linearization of Polynomial Problems

Many polynomial problems can be reformulated as computing the nullspace of a structured matrix

This may not be efficient for exact computation, but lends itself well to approximate computation

For example, compute $\gcd(f,g)$ becomes: find the "smallest" non-zero vector in the nullspace of $\mathrm{Syl}(f,g)$

Approximate algebra version of GCD: find the nearest non-relatively prime pair $\tilde{f},\tilde{g}$ to f,g and compute $\gcd(\tilde{f},\tilde{g})$

Reformulated:

Find the rank deficient Sylvester matrix S which is nearest to $\mathrm{Syl}(f,g)$. In the two norm: find the *Structured Total Least Squares* solution to the system $\mathrm{Syl}(f,g)\cdot x = 0$

Total Least Squares

Least Squares: find Δb with smallest norm possible so that $Ax = (b + \Delta b)$ is consistent.

Easy: find the projection of b onto the column space of A

Total Least Squares

Least Squares: find Δb with smallest norm possible so that $Ax = (b + \Delta b)$ is consistent.
Easy: find the projection of b onto the column space of A

Total Least Squares (TLS): find Δb and ΔA each with smallest norm possible so that $(A + \Delta A)\, x = (b + \Delta b)$ is consistent.
Harder: form the augmented matrix $M = [A \,|\, b]$ and compute the nearest singular matrix to M using the singular value decomposition

Total Least Squares

Least Squares: find Δb with smallest norm possible so that $Ax = (b + \Delta b)$ is consistent.
Easy: find the projection of b onto the column space of A

Total Least Squares (TLS): find Δb and ΔA each with smallest norm possible so that $(A + \Delta A)\,x = (b + \Delta b)$ is consistent.
Harder: form the augmented matrix $M = [A \,|\, b]$ and compute the nearest singular matrix to M using the singular value decomposition

Structured Total Least Squares (STLS): restrict $[\Delta A \,|\, \Delta b] \in \mathfrak{S} \subset \mathbb{C}^{r \times c}$ where $\mathfrak{S}$ is a set of structured matrices (for algebraic problems $\mathfrak{S}$ is often a linear subspace of $\mathbb{C}^{r \times c}$)
Very Hard in general: polynomial time solutions exist for certain special cases only

Strategies for STLS

Ignore Structure

Try to solve STLS problems arising in approximate algebra by treating them as TLS problems then improve the answer using an iterative scheme
Leads to “soft” optimization algorithms and lower bounds

Strategies for STLS

Ignore Structure

Try to solve STLS problems arising in approximate algebra by treating them as TLS problems then improve the answer using an iterative scheme
Leads to “soft” optimization algorithms and lower bounds

Iterative Heuristics that Account for Structure

Our method: Riemannian SVD

RiSVD is easy to implement but slow

RiSVD works with linear structures $\mathfrak{S} = \text{span}\{B_i\}$ and norm $\|A\|^2 = \sum |a_i|^2$ where $A = \sum a_i B_i$

Strategies for STLS

Ignore Structure

Try to solve STLS problems arising in approximate algebra by treating them as TLS problems then improve the answer using an iterative scheme
Leads to “soft” optimization algorithms and lower bounds

Iterative Heuristics that Account for Structure

Our method: Riemannian SVD

RiSVD is easy to implement but slow

RiSVD works with linear structures $\mathfrak{S} = \text{span}\{B_i\}$ and norm $\|A\|^2 = \sum |a_i|^2$ where $A = \sum a_i B_i$

Another method: STLN

Harder to implement, but faster. Optimizes a different norm (matrix 2-norm).

The Riemannian SVD

Let $A = \sum_i c_i B_i$ be the input matrix. Find the triplet (u, τ, v) corresponding to the smallest τ such that

$$Av = D_v u \tau \quad u^T D_v u = 1$$

$$A^T u = D_u v \tau \quad v^t D_u v = 1$$

$$v^T v = 1$$

The Riemannian SVD

Let $A = \sum_i c_i B_i$ be the input matrix. Find the triplet (u, τ, v) corresponding to the smallest τ such that

$$Av = D_v u \tau \quad u^T D_v u = 1$$

$$A^T u = D_u v \tau \quad v^t D_u v = 1$$

$$v^T v = 1$$

- D_u and D_v have a quadratic dependence on u, v respectively
- v (the null vector) and $\hat{c}_i = c_i - u^T T_i v \tau$ form a solution to the STLS problem
- In order to determine (u, τ, v), we hold D_u, D_v fixed, and perform an inverse iteration to determine updated vectors u, v
- Each iteration is solved by putting the constraints of the RiSVD into a large linear system, and solving for u, v
- D_u, D_v are then updated, and another iteration is performed

Linearizable Polynomial Algebra Problems

- Univariate and Multivariate Polynomial GCD
  via the Sylvester type matrices

Linearizable Polynomial Algebra Problems

- Univariate and Multivariate Polynomial GCD
  via the Sylvester type matrices
- Specified Degree GCD and Polynomial Division
  via Sylvester type matrices

Linearizable Polynomial Algebra Problems

- Univariate and Multivariate Polynomial GCD
  via the Sylvester type matrices
- Specified Degree GCD and Polynomial Division
  via Sylvester type matrices
- Multivariate Polynomial Factorizations
  via the Ruppert matrix

Linearizable Polynomial Algebra Problems

- Univariate and Multivariate Polynomial GCD
  via the Sylvester type matrices
- Specified Degree GCD and Polynomial Division
  via Sylvester type matrices
- Multivariate Polynomial Factorizations
  via the Ruppert matrix
- Univariate Polynomial Decomposition
  via reduction to multivariate factorization

Linearizable Polynomial Algebra Problems

- Univariate and Multivariate Polynomial GCD
  via the Sylvester type matrices
- Specified Degree GCD and Polynomial Division
  via Sylvester type matrices
- Multivariate Polynomial Factorizations
  via the Ruppert matrix
- Univariate Polynomial Decomposition
  via reduction to multivariate factorization
- Ore Polynomial GCRD
  via a generalization of the Sylvester matrix

Univariate Decomposition

Given $f \in \mathbb{C}[x]$, f is decomposable if $\exists\, g,h \in \mathbb{C}[x]$, $\deg h > 1$ so that $f = g \circ h$

For a fixed h, computing g is clearly a linear problem

Due to a theorem of Fried (1969), except for few special cases, f decomposes exactly when $\Phi(f) = (f(x) - f(y))/(x-y)$ factors over $\mathbb{C}[x,y]$

f decomposes if and only if $\mathrm{Rup}(\Phi(f))$ has a non-trivial nullspace

Decompostion factors of f can be recovered from the factors of $\Phi(f)$ as in Barton and Zippel

Univariate GCD Experiments

500 examples such that: $6 \leq \deg g = \deg f \leq 11$, before noise $\deg \gcd(f,g) = 1$. Coeffs of f and g are floating point numbers in $[-10..10]$ with added noise of size $\approx 10^{-2}$.

The output from three methods used at the starting point for a local optimization routine.

Method	Average CPU time	Avg. bw. err	# of Failures
SNAP	0.211s	0.851994725594685	177
SVD GCD	0.099s	0.237547300725886	48
RiSVD GCD	3.135s	0.077418578991535 1	2

Comparing with an early implementation of STLN, we found that STLN and RiSVD achieve nearly the same backward error in most cases with RiSVD slightly more accurate, and STLN several times faster.

Decomposition Experiments

			CGJW			AppFac			RiSVD		
deg g	deg h	ε	Error	Best	Abs	Error	Best	Abs	Error	Best	Abs
2	2	10^{-4}	1.53e-5	0%	100%	1.59e-5	0%	98%	1.53e-5	0%	98%
2	3	10^{-4}	1.15e-4	3%	100%	9.70e-3	0%	45%	1.26e-4	0%	97%
3	2	10^{-4}	1.90e-5	0%	96%	1.90e-5	0%	100%	1.90e-5	0%	100%
4	2	10^{-4}	1.85e-4	5%	97%	6.21e-2	2%	28%	2.20e-4	0%	88%
2	4	10^{-4}	2.99e-5	0%	98%	4.17e-5	0%	83%	2.99e-5	2%	98%
2	2	10^{-1}	8.74e-3	2%	95%	9.18e-3	2%	83%	9.52e-3	3%	87%
2	3	10^{-1}	5.24e-2	5%	70%	5.37e-2	2%	43%	5.08e-2	12%	80%
3	2	10^{-1}	1.54e-2	0%	85%	1.47e-2	8%	95%	1.54e-2	2%	87%
2	4	10^{-1}	1.21e-1	5%	50%	1.61e-1	8%*	25%	1.21e-1	7%	53%
4	2	10^{-1}	2.50e-2	2%	43%	2.28e-2	18%*	30%	2.50e-2	3%	40%
2	2	1	2.40e-2	0%	95%	2.40e-2	0%	93%	2.40e-2	0%	95%
2	3	1	1.83e-1	2%	75%	1.74e-1	7%*	70%	1.79e-1	7%	75%
3	2	1	1.62e-1	0%	77%	1.51e-1	7%*	77%	1.44e-1	8%	87%
2	4	1	3.78e-1	2%	50%	3.45e-1	30%*	28%	3.70e-1	8%	55%
4	2	1	2.87e-1	8%	58%	2.63e-1	22%*	40%	2.89e-1	5%	60%

Ongoing and Future Work

- All RiSVD iterations stay in the coefficient field. What if the optimum is complex?
  One possibility: Replace structured matrix with a block structured matrix. Numbers replaced with 2×2 blocks representing complex multiplication. This can find complex optimums.

Ongoing and Future Work

- All RiSVD iterations stay in the coefficient field. What if the optimum is complex?
  One possibility: Replace structured matrix with a block structured matrix. Numbers replaced with 2×2 blocks representing complex multiplication. This can find complex optimums.

- RiSVD for GCD often out performs the TLS strategy. How about for other problems?
  Main difficulties: 1. reconstruction of the solution of the polynomial problem from the STLS solution.
  2. RiSVD is less effective for non-square structures! It seems especially bad on the Ruppert (factorization) structure. One imperfect fix: replace $A \rightarrow ADA^T$, and $B_i \rightarrow B_i DA^T$ for randomly chosen diagonal D.

Ongoing and Future Work

- All RiSVD iterations stay in the coefficient field. What if the optimum is complex?
  One possibility: Replace structured matrix with a block structured matrix. Numbers replaced with 2×2 blocks representing complex multiplication. This can find complex optimums.

- RiSVD for GCD often out performs the TLS strategy. How about for other problems?
  Main difficulties: 1. reconstruction of the solution of the polynomial problem from the STLS solution.
  2. RiSVD is less effective for non-square structures! It seems especially bad on the Ruppert (factorization) structure. One imperfect fix: replace $A \rightarrow ADA^T$, and $B_i \rightarrow B_i DA^T$ for randomly chosen diagonal D.

- Efficiency. The matrices can become very large and a general RiSVD implementation can be very slow.

  Can we use black box representations of the matrices B_i?

  Can RiSVD be tuned to run quickly on specific problems?

Fin