Approximate Polynomial GCD: Structured Matrix Based Methods and Fast Algorithms

D. A. Bini\textsuperscript{1}  P. Boito\textsuperscript{2}

\textsuperscript{1}Università di Pisa

\textsuperscript{2}Institut de Mathématiques
Université Paul Sabatier, Toulouse
Outline

1. Introduction: Approximate GCD
2. Resultant Matrices
3. Displacement Structure and Fast LU
4. The Fastgcd Algorithm
5. Numerical Tests
Why approximate GCD?

### Problem

Given \( u(x), v(x) \in \mathbb{R}[x] \) or \( \mathbb{C}[x] \), compute \( g(x) = \text{GCD}(u, v) \).

- Applications: polynomial rootfinding, control theory, CAGD, image deblurring...
- But if the coefficients of \( u(x) \) and \( v(x) \) are affected by errors, the problem is ill-posed.
- We need a new definition of polynomial GCD [Schönhage '85].
Why approximate GCD?

**Problem**

Given \( u(x), v(x) \in \mathbb{R}[x] \) or \( \mathbb{C}[x] \), compute \( g(x) = \text{GCD}(u, v) \).

- Applications: polynomial rootfinding, control theory, CAGD, image deblurring...
- But if the coefficients of \( u(x) \) and \( v(x) \) are affected by errors, the problem is ill-posed.
- We need a new definition of polynomial GCD [Schönhage '85].
Why approximate GCD?

Problem

Given \( u(x), v(x) \in \mathbb{R}[x] \) or \( \mathbb{C}[x] \), compute \( g(x) = \text{GCD}(u, v) \).

- Applications: polynomial rootfinding, control theory, CAGD, image deblurring...
- But if the coefficients of \( u(x) \) and \( v(x) \) are affected by errors, the problem is ill-posed.
- We need a new definition of polynomial GCD [Schönhage '85].
Why approximate GCD?

Problem

Given $u(x), v(x) \in \mathbb{R}[x]$ or $\mathbb{C}[x]$, compute $g(x) = \gcd(u, v)$.

- Applications: polynomial rootfinding, control theory, CAGD, image deblurring...
- But if the coefficients of $u(x)$ and $v(x)$ are affected by errors, the problem is ill-posed.
- We need a new definition of polynomial GCD [Schönhage ‘85].
Definition: $\epsilon$-GCD

**Definition**

Given polynomials $u(x)$, $v(x)$ and a tolerance $\epsilon > 0$,

- $g(x)$ is an $\epsilon$-divisor of $u(x)$ and $v(x)$ if it is an exact divisor of $\hat{u}(x)$ and $\hat{v}(x)$, where:
  - $\hat{u}(x)$ and $\hat{v}(x)$ have the same degrees as $u(x)$ and $v(x)$;
  - $\|u(x) - \hat{u}(x)\|_2 < \epsilon$, $\|v(x) - \hat{v}(x)\|_2 < \epsilon$.

- $g(x)$ is an $\epsilon$-GCD of $u(x)$ and $v(x)$ if it is an $\epsilon$-divisor of maximum degree.

[Corless, Gianni, Trager, Watt ’95], [Emiris, Galligo, Lombardi ’97]

The degree of an $\epsilon$-GCD is uniquely defined, whereas its coefficients are not.
Definition: $\epsilon$-GCD

**Definition**

Given polynomials $u(x)$, $v(x)$ and a tolerance $\epsilon > 0$,

- $g(x)$ is an $\epsilon$-divisor of $u(x)$ and $v(x)$ if it is an exact divisor of $\hat{u}(x)$ and $\hat{v}(x)$, where:
  - $\hat{u}(x)$ and $\hat{v}(x)$ have the same degrees as $u(x)$ and $v(x)$;
  - $\|u(x) - \hat{u}(x)\|_2 < \epsilon$, $\|v(x) - \hat{v}(x)\|_2 < \epsilon$.

- $g(x)$ is an $\epsilon$-GCD of $u(x)$ and $v(x)$ if it is an $\epsilon$-divisor of maximum degree.

[Corless, Gianni, Trager, Watt ’95], [Emiris, Galligo, Lombardi ’97]
The degree of an $\epsilon$-GCD is uniquely defined, whereas its coefficients are not.
Definition: $\epsilon$-GCD

Given polynomials $u(x)$, $v(x)$ and a tolerance $\epsilon > 0$,

- $g(x)$ is an $\epsilon$-divisor of $u(x)$ and $v(x)$ if it is an exact divisor of $\hat{u}(x)$ and $\hat{v}(x)$, where:
  - $\hat{u}(x)$ and $\hat{v}(x)$ have the same degrees as $u(x)$ and $v(x)$;
  - $\|u(x) - \hat{u}(x)\|_2 < \epsilon$, $\|v(x) - \hat{v}(x)\|_2 < \epsilon$.

- $g(x)$ is an $\epsilon$-GCD of $u(x)$ and $v(x)$ if it is an $\epsilon$-divisor of maximum degree.

[Corless, Gianni, Trager, Watt ’95], [Emiris, Galligo, Lombardi ’97]

The degree of an $\epsilon$-GCD is uniquely defined, whereas its coefficients are not.
Definition: $\epsilon$-GCD

Given polynomials $u(x)$, $v(x)$ and a tolerance $\epsilon > 0$,

- $g(x)$ is an $\epsilon$-divisor of $u(x)$ and $v(x)$ if it is an exact divisor of $\hat{u}(x)$ and $\hat{v}(x)$, where:
  - $\hat{u}(x)$ and $\hat{v}(x)$ have the same degrees as $u(x)$ and $v(x)$;
  - $\|u(x) - \hat{u}(x)\|_2 < \epsilon$, $\|v(x) - \hat{v}(x)\|_2 < \epsilon$.

- $g(x)$ is an $\epsilon$-GCD of $u(x)$ and $v(x)$ if it is an $\epsilon$-divisor of maximum degree.

[Corless, Gianni, Trager, Watt '95], [Emiris, Galligo, Lombardi '97]

The degree of an $\epsilon$-GCD is uniquely defined, whereas its coefficients are not.
Computational approaches

For coefficient-based definitions ($\epsilon$-GCD, optimization approach):

- Euclidean algorithm ([Schönhage '85], [Noda, Sasaki '91], [Hribernig, Stetter '97], [Beckermann, Labahn '98], [Sasaki '07]);
- resultant/subresultant matrices ([Corless, Gianni, Trager, Watt '95], [Emiris, Galligo, Lombardi '97], [Rupprecht '99], [Zarowski '00], [Zhi '03], [Zeng '04], [Corless, Watt, Zhi '05]);
- optimization techniques ([Corless, Gianni, Trager, Watt '95], [Karmarkar, Lakshman '96], [Chin, Corless, Corliss '98], [Chu, Funderlic, Plemmons '03], [Kaltofen, Yang, Zhi '05], [Zhi et al.], [Markovsky, Van Huffel '06]).

For the root-based definition ($\delta$-GCD):

- graph-theoretical techniques ([Pan '96]).
Computational approaches

For coefficient-based definitions ($\varepsilon$-GCD, optimization approach):

- Euclidean algorithm ([Schonhage ’85], [Noda, Sasaki ’91], [Hribernig, Stetter ’97], [Beckermann, Labahn ’98], [Sasaki ’07]);
- resultant/subresultant matrices ([Corless, Gianni, Trager, Watt ’95], [Emiris, Galligo, Lombardi ’97], [Rupprecht ’99], [Zarowski ’00], [Zhi ’03], [Zeng ’04], [Corless, Watt, Zhi ’05]);
- optimization techniques ([Corless, Gianni, Trager, Watt ’95], [Karmarkar, Lakshman ’96], [Chin, Corless, Corliss ’98], [Chu, Funderlic, Plemmons ’03], [Kaltofen, Yang, Zhi ’05], [Zhi et al.], [Markovsky, Van Huffel ’06]).

For the root-based definition ($\delta$-GCD):

- graph-theoretical techniques ([Pan ’96]).
Computational approaches

For coefficient-based definitions ($\epsilon$-GCD, optimization approach):

- Euclidean algorithm ([Schönhage ’85], [Noda, Sasaki ’91], [Hribernig, Stetter ’97], [Beckermann, Labahn ’98], [Sasaki ’07]);
- resultant/subresultant matrices ([Corless, Gianni, Trager, Watt ’95], [Emiris, Galligo, Lombardi ’97], [Rupprecht ’99], [Zarowski ’00], [Zhi ’03], [Zeng ’04], [Corless, Watt, Zhi ’05]);
- optimization techniques ([Corless, Gianni, Trager, Watt ’95], [Karmarkar, Lakshman ’96], [Chin, Corless, Corliss ’98], [Chu, Funderlic, Plemmons ’03], [Kaltofen, Yang, Zhi ’05], [Zhi et al.], [Markovsky, Van Huffel ’06]).

For the root-based definition ($\delta$-GCD):

- graph-theoretical techniques ([Pan ’96]).
Computational approaches

For coefficient-based definitions ($\epsilon$-GCD, optimization approach):

- Euclidean algorithm ([Schönhage ’85], [Noda, Sasaki ’91], [Hribernig, Stetter ’97], [Beckermann, Labahn ’98], [Sasaki ’07]);
- resultant/subresultant matrices ([Corless, Gianni, Trager, Watt ’95], [Emiris, Galligo, Lombardi ’97], [Rupprecht ’99], [Zarowski ’00], [Zhi ’03], [Zeng ’04], [Corless, Watt, Zhi ’05]);
- optimization techniques ([Corless, Gianni, Trager, Watt ’95], [Karmarkar, Lakshman ’96], [Chin, Corless, Corliss ’98], [Chu, Funderlic, Plemmons ’03], [Kaltofen, Yang, Zhi ’05], [Zhi et al.], [Markovsky, Van Huffel ’06]).

For the root-based definition ($\delta$-GCD):

- graph-theoretical techniques ([Pan ’96]).
## Implementations

<table>
<thead>
<tr>
<th>name</th>
<th>author</th>
<th>algorithm</th>
<th>environment</th>
<th>availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>QuasiGCD</td>
<td>Beckermann, Labahn</td>
<td>EA with look-ahead</td>
<td>Maple</td>
<td>SNAP package</td>
</tr>
<tr>
<td></td>
<td>Corless et al.</td>
<td>QR of Sylvester</td>
<td>Maple</td>
<td>SNAP package</td>
</tr>
<tr>
<td></td>
<td>Zeng</td>
<td>SVD, subresultants</td>
<td>Matlab/Maple</td>
<td>upon request</td>
</tr>
<tr>
<td>UVGCD</td>
<td>Zeng</td>
<td>structured low rank approx.</td>
<td>Maple</td>
<td>website</td>
</tr>
<tr>
<td>STLN</td>
<td>Kaltofen et al.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Plus other implementations which are not yet fully developed or easily available (e.g., work by Zarowski, Zhi, Markovsky, Allan&Winkler...).
Our approach

We wish to use:

- **Matrices**: many properties of polynomials can be translated into the language of linear algebra;

- **Structure**: matrices related to polynomial properties often display some kind of structure (e.g., displacement structure).

We develop a fast algorithm that exploits displacement structure.
Sylvester matrix

Let $u(x) = \sum_{i=0}^{n} u_i x^i$, \quad $v(x) = \sum_{i=0}^{m} v_i x^i$.

Define

$$\text{Sylv}(u, v) = \begin{pmatrix}
  u_n & u_{n-1} & \cdots & \cdots & u_0 & 0 \\
  \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\
  0 & u_n & u_{n-1} & \cdots & \cdots & u_0 \\
  v_m & v_{m-1} & \cdots & v_0 \\
  \vdots & \vdots & \ddots & \ddots & \ddots & \ddots \\
  0 & 0 & \cdots & v_m & v_{m-1} & \cdots & v_0
\end{pmatrix}.$$ 

$\text{Sylv}(u, v)$ is a square matrix of order $n + m$. 
Let $u(x) = \sum_{i=0}^{n} u_i x^i$, $v(x) = \sum_{i=0}^{m} v_i x^i$, $n \geq m$.

Then the rational function

$$b(x, y) = \frac{u(x)v(y) - u(y)v(x)}{x - y} = \sum_{i,j} b_{ij} x^i y^j$$

is a bivariate polynomial.

The Bézout matrix is defined by $(\text{Bez}(u, v))_{ij} = b_{ij}$.

Bez($u, v$) is square and symmetric, of order $n$. 

Bézout matrix
Let \( u(x) = \sum_{i=0}^{n} u_i x^i \), \( v(x) = \sum_{i=0}^{m} v_i x^i \), \( n \geq m \).

Then the rational function

\[
b(x, y) = \frac{u(x)v(y) - u(y)v(x)}{x - y} = \sum_{i,j} b_{ij} x^i y^j
\]

is a bivariate polynomial.

The Bézout matrix is defined by \( (\text{Bez}(u, v))_{ij} = b_{ij} \).

\( \text{Bez}(u, v) \) is square and symmetric, of order \( n \).
Let $u(x) = \sum_{i=0}^{n} u_{i} x^{i}$, $v(x) = \sum_{i=0}^{m} v_{i} x^{i}$, $n \geq m$. Then the rational function

$$b(x, y) = \frac{u(x)v(y) - u(y)v(x)}{x - y} = \sum_{i,j} b_{ij} x^{i} y^{j}$$

is a bivariate polynomial.

The Bézout matrix is defined by $(\text{Bez}(u, v))_{ij} = b_{ij}$.

$\text{Bez}(u, v)$ is square and symmetric, of order $n$. 
Matrix rank and GCD degree

Let $M = \text{Sylv}(u, v)$ or $\text{Bez}(u, v)$. Then

**Theorem**

$$\dim \ker(M) = \deg \gcd(u, v).$$

- In the exact case

  $$\deg \gcd(u, v) \leftrightarrow \text{rank of } M.$$

- In the approximate case

  $$\deg \epsilon\text{-GCD}(u, v) \leftrightarrow \text{approximate rank of } M.$$
Matrix rank and GCD degree

Let $M = \text{Sylv}(u, v)$ or $\text{Bez}(u, v)$. Then

**Theorem**

$$\dim \ker(M) = \deg \text{GCD}(u, v).$$

- In the exact case

$$\deg \text{GCD}(u, v) \leftrightarrow \text{rank of } M.$$

- In the approximate case

$$\deg \epsilon\text{-GCD}(u, v) \leftrightarrow \text{approximate rank of } M.$$
Matrix rank and GCD degree

Let $M = \text{Sylv}(u, v)$ or $\text{Bez}(u, v)$. Then

**Theorem**

$$\dim \ker(M) = \deg \text{GCD}(u, v).$$

- In the exact case

$$\deg \text{GCD}(u, v) \leftrightarrow \text{rank of } M.$$

- In the approximate case

$$\deg \varepsilon\text{-GCD}(u, v) \leftrightarrow \text{approximate rank of } M.$$
Outline of an algorithm

General idea for computing an $\epsilon$-GCD:

1. estimate deg $\epsilon$-GCD via the approximate rank of Sylv$(u, v)$ or Bez$(u, v)$;
2. compute the coefficients (solve a linear system defined by a submatrix of Sylv$(u, v)$ or Bez$(u, v)$);
3. iterative refinement (Newton).

The computational cost is usually $O(n^3)$.

Our goal: do this in a stable way and with $O(n^2)$ computational cost (exploit displacement structure).
General idea for computing an $\epsilon$-GCD:

1. estimate deg $\epsilon$-GCD via the approximate rank of $\text{Sylv}(u, v)$ or $\text{Bez}(u, v)$;
2. compute the coefficients (solve a linear system defined by a submatrix of $\text{Sylv}(u, v)$ or $\text{Bez}(u, v)$);
3. iterative refinement (Newton).

The computational cost is usually $O(n^3)$.

Our goal: do this in a stable way and with $O(n^2)$ computational cost (exploit displacement structure).
Toeplitz-like matrices

$T \in \mathbb{C}^{n \times n}$ is Toeplitz-like with displacement rank $r$ if

$$\nabla_{\{1,-1\}}(T) = Z_1 \cdot T - T \cdot Z_{-1} = G \cdot B,$$

where $G \in \mathbb{C}^{n \times r}$, $B \in \mathbb{C}^{r \times n}$ are the generators, and

$$Z_\phi = \begin{pmatrix}
0 & \ldots & \ldots & 0 & \phi \\
1 & 0 & \ldots & \ldots & 0 \\
0 & 1 & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
0 & \ldots & 0 & 1 & 0
\end{pmatrix}.$$

Sylv($u$, $v$) and Bez($u$, $v$) are Toeplitz-like with $r = 2$. 
Toeplitz-like matrices

\( T \in \mathbb{C}^{n \times n} \) is Toeplitz-like with displacement rank \( r \) if

\[
\nabla_{\{1,-1\}}(T) = Z_1 \cdot T - T \cdot Z_{-1} = G \cdot B,
\]

where \( G \in \mathbb{C}^{n \times r} \), \( B \in \mathbb{C}^{r \times n} \) are the generators, and

\[
Z_\phi = \begin{pmatrix}
0 & \cdots & \cdots & 0 & \phi \\
1 & 0 & \cdots & \cdots & 0 \\
0 & 1 & \ddots & \ddots & \ddots \\
\vdots & \ddots & \ddots & \ddots & \ddots \\
0 & \cdots & 0 & 1 & 0
\end{pmatrix}.
\]

\( \text{Sylv}(u, v) \) and \( \text{Bez}(u, v) \) are Toeplitz-like with \( r = 2 \).
Cauchy-like matrices

- $C \in \mathbb{C}^{n \times n}$ is Cauchy-like with displacement rank $r$ if

$$\nabla\{F,A\}(C) = F \cdot C + C \cdot A^* = G \cdot B,$$

where $G \in \mathbb{C}^{n \times r}$, $B \in \mathbb{C}^{r \times n}$ are the generators and
$F = \text{diag}(f_0, f_1, \ldots, f_{n-1})$, $A = \text{diag}(a_0, a_1, \ldots, a_{n-1})$ are diagonal matrices with disjoint spectra.

- We have

$$C = \left[ \frac{g_i b_j^*}{f_i - \overline{a}_j} \right]_{i,j=0}^{n-1}.$$
Cauchy-like matrices

- $C \in \mathbb{C}^{n \times n}$ is Cauchy-like with displacement rank $r$ if

$$\nabla_{\{F,A\}}(C) = F \cdot C + C \cdot A^* = G \cdot B,$$

where $G \in \mathbb{C}^{n \times r}$, $B \in \mathbb{C}^{r \times n}$ are the generators and $F = \text{diag}(f_0, f_1, \ldots, f_{n-1})$, $A = \text{diag}(a_0, a_1, \ldots, a_{n-1})$ are diagonal matrices with disjoint spectra.

- We have

$$C = \left[ \frac{g_i b_j^*}{f_i - \bar{a}_j} \right]_{i,j=0}^{n-1}.$$
Fast LU with partial pivoting (GKO)

- Let $C \in \mathbb{C}^{n \times n}$ be Cauchy-like. The GKO algorithm ([Gohberg, Kailath, Olshevsky '95]) allows to compute $C = \Pi LU$ (Gaussian elimination with partial pivoting) with a computational cost of $O(n^2)$.

- Let $T \in \mathbb{C}^{n \times n}$ be Toeplitz-like. Then $T$ can be transformed into Cauchy-like in a fast and stable way ([Heinig '94], [Pan '90]).

Therefore we have a fast algorithm for the LU factorization of resultant matrices.
Fast LU with partial pivoting (GKO)

Let $C \in \mathbb{C}^{n \times n}$ be Cauchy-like. The GKO algorithm ([Gohberg, Kailath, Olshevsky ’95]) allows to compute

$$C = \Pi LU$$

(Gaussian elimination with partial pivoting) with a computational cost of $O(n^2)$.  

Let $T \in \mathbb{C}^{n \times n}$ be Toeplitz-like. Then $T$ can be transformed into Cauchy-like in a fast and stable way ([Heinig ’94], [Pan ’90]).  

Therefore we have a fast algorithm for the LU factorization of resultant matrices.
Fast LU with partial pivoting (GKO)

Let \( C \in \mathbb{C}^{n \times n} \) be Cauchy-like. The GKO algorithm ([Gohberg, Kailath, Olshevsky '95]) allows to compute

\[
C = \Pi LU
\]

(Gaussian elimination with partial pivoting) with a computational cost of \( \mathcal{O}(n^2) \).

Let \( T \in \mathbb{C}^{n \times n} \) be Toeplitz-like. Then \( T \) can be transformed into Cauchy-like in a fast and stable way ([Heinig '94], [Pan '90]).

Therefore we have a fast algorithm for the LU factorization of resultant matrices.
GKO is often unstable when applied to resultant matrices, because of generator growth. A different pivoting technique is needed:

> at each step, re-orthogonalize the first generator and apply an appropriate pivoting to the second one ([Gu '98]).

The transformation from Toeplitz-like to Cauchy-like must be rewritten for rectangular matrices.
GKO and Approximate Rank
Computation of the degree

An estimate on the $\epsilon$-GCD degree may be obtained:

- via bisection, or
- using a heuristic criterion: an upper bound on the degree is given by $n - k_\epsilon$, where

$$k_\epsilon = \max\{k : a_k < \epsilon \sqrt{n + m}\}$$

and $a_k$ is the $k$-th pivot in the factorization of the Cauchy-like matrix obtained from Sylv$(u, v)$. 
Sylvester matrix and GCD coefficients

\[ g(x) = \text{GCD}(u, v) \iff \begin{cases} 
  u(x) = p(x)g(x) \\
  v(x) = q(x)g(x) 
\end{cases} \iff u(x)q(x) = v(x)p(x). \]

In matrix form:

\[
S \begin{bmatrix} q \\ -p \end{bmatrix} = 0, \quad S = \begin{pmatrix}
  u_0 & 0 & v_0 & 0 \\
  \vdots & \ddots & \vdots & \ddots \\
  u_n & u_0 & v_m & v_0 \\
  0 & u_n & 0 & v_m 
\end{pmatrix}
\]

and \( g(x) = u(x)/p(x) = v(x)/q(x). \) Polynomial division is performed via FFT.
Sylvester matrix and GCD coefficients

\[ g(x) = \text{GCD}(u, v) \iff \begin{cases} u(x) = p(x)g(x) \\ v(x) = q(x)g(x) \end{cases} \iff u(x)q(x) = v(x)p(x). \]

In matrix form:

\[
S \begin{bmatrix} q \\ -p \end{bmatrix} = 0, \quad S = \begin{pmatrix}
    u_0 & 0 & v_0 & 0 \\
    \vdots & \ddots & \vdots & \vdots \\
    u_n & 0 & v_m & v_0 \\
    0 & u_n & 0 & v_m
\end{pmatrix}
\]

and \( g(x) = u(x)/p(x) = v(x)/q(x). \) Polynomial division is performed via FFT.
Generalization of GKO to rectangular matrices

Let $T \in \mathbb{C}^{n \times m}$, $n \neq m$, be Toeplitz-like with displacement rank $r$. The reduction of $T$ to Cauchy-like must be rewritten using the following property:

$$
\nabla_\theta(T) = Z_1 \cdot T - T \cdot Z_\theta
$$

has rank $r$ for every $\theta \in \mathbb{C}$ such that $|\theta| = 1$.

Given the $\epsilon$-GCD degree, we can compute a set of coefficients in a cheap ($O(n^2)$) and stable way.
Iterative refinement (I)

The system

\[
\begin{align*}
    u(x) &= g(x)p(x) \\
    v(x) &= g(x)q(x)
\end{align*}
\]

can be written as

\[
F(z) = A(z) - w = 0,
\]

where

\[
z = \begin{bmatrix} g \\ p \\ q \end{bmatrix}, \quad w = \begin{bmatrix} u \\ v \end{bmatrix}, \quad A(z) = \begin{bmatrix} C(p)g \\ C(q)g \end{bmatrix} = \begin{bmatrix} C(g)p \\ C(g)q \end{bmatrix}
\]

and \(C(p), C(q), C(g)\) are convolution matrices.
Iterative refinement (II)

Newton’s method:

- we want to minimize $\| F(z) \|_2 = \| A(z) - w \|_2$;
- general Newton step:

$$z_{j+1} = z_j - J(z_j)^\dagger F(z_j),$$

- at each step, solve the linear least squares problem

$$J(z_j - z_{j+1}) = F(z_j),$$

where $J$ is the Jacobian matrix associated with $F(z)$;
- in practical implementation, we add an equation to $F(z)$ so that the new Jacobian $J_{\text{new}}$ is not rank-deficient:

$$J_{\text{new}} = \begin{pmatrix} g^T, & J, & -p^T, & -q^T \end{pmatrix}.$$
Fast iterative refinement

Recall that at each step of Newton’s method we have to solve the linear least squares problem

\[ J(z_j - z_{j+1}) = F(z_j), \]

with \( J \) the Jacobian matrix associated with \( F(z) \).

- \( J \) is Toeplitz-like.
- The modified GKO method can be used to compute a sequence \( \{z_j\} \) that converges at the same speed as the classical Newton’s method.
Recall that at each step of Newton’s method we have to solve the linear least squares problem

\[ J(z_j - z_{j+1}) = F(z_j), \]

with \( J \) the Jacobian matrix associated with \( F(z) \).

- \( J \) is Toeplitz-like.
- The modified GKO method can be used to compute a sequence \( \{z_j\} \) that converges at the same speed as the classical Newton’s method.
Iterative refinement (III)

Stopping criteria:
1. $\|F(z)\|_2 < \text{fixed threshold}$,
2. # iterations > fixed maximum,
3. $\|F(z)\|_2$ stops decreasing.

Is (3) a good criterion? (it is difficult to predict global behaviour for Newton...) We introduce a line search: solve

$$z_{j+1} = z_j - \alpha_j J(z_j)^\dagger F(z_j),$$

where $\alpha_j$ minimizes $\|F(z_{j+1})\|_2$, but do this only when necessary, otherwise convergence slows down.
Iterative refinement (III)

Stopping criteria:

1. \( \| F(z) \|_2 < \) fixed threshold,
2. \# iterations > fixed maximum,
3. \( \| F(z) \|_2 \) stops decreasing.

Is (3) a good criterion? (it is difficult to predict global behaviour for Newton...) We introduce a line search: solve

\[
\begin{aligned}
z_{j+1} &= z_j - \alpha_j J(z_j) F(z_j), \\
\end{aligned}
\]

where \( \alpha_j \) minimizes \( \| F(z_{j+1}) \|_2 \), but do this only when necessary, otherwise convergence slows down.
Iterative refinement (III)

Stopping criteria:
1. $\|F(z)\|_2 < \text{fixed threshold}$,
2. $\# \text{iterations} > \text{fixed maximum}$,
3. $\|F(z)\|_2$ stops decreasing.

Is (3) a good criterion? (it is difficult to predict global behaviour for Newton...) We introduce a line search: solve

$$z_{j+1} = z_j - \alpha_j J(z_j)^\dagger F(z_j),$$

where $\alpha_j$ minimizes $\|F(z_{j+1})\|_2$, but do this only when necessary, otherwise convergence slows down.
The Fastgcd algorithm

Input: polynomials \( u(x) \) and \( v(x) \), a tolerance \( \epsilon \).
Output: an \( \epsilon \)-GCD, cofactors, residual (backward error).

1. Find an estimate \( k \) for deg \( \epsilon \)-GCD.
2. Compute the coefficients of an approximate divisor \( g(x) \) of degree \( k \), along with cofactors.
3. Perform iterative refinement.
4. Is \( g(x) \) an \( \epsilon \)-divisor?
5. If it is, set \( k=k+1 \) and compute a new approximate divisor, until an \( \epsilon \)-divisor of maximum degree is found.
6. If it isn’t, set \( k=k-1 \) and compute a new approximate divisor, until an \( \epsilon \)-divisor is found.
Numerical tests

We implemented the algorithm Fastgcd in Matlab and tested it on several examples. Good stability properties and quadratic cost are confirmed by the experiments.

Some significant results are shown here, along with comparisons with other methods for approximate GCD (UVGCD by Zeng and QRGCD by Corless et al.)
High GCD Degree

- \( v(x) = \sum_{j=0}^{3} x^j, \ w(x) = \sum_{j=0}^{4} (-x)^j; \)
- \( u_n(x) \) has degree \( n \) and random integer coefficients in \([-5, 5];\)
- \( p_n = u_n v, \ q_n = u_n w. \)

<table>
<thead>
<tr>
<th>( n )</th>
<th>Fastgcd</th>
<th>res</th>
<th>cwe</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td></td>
<td>2.97 ( \times 10^{-16} )</td>
<td>5.04 ( \times 10^{-16} )</td>
</tr>
<tr>
<td>100</td>
<td></td>
<td>2.91 ( \times 10^{-16} )</td>
<td>1.41 ( \times 10^{-15} )</td>
</tr>
<tr>
<td>200</td>
<td></td>
<td>5.08 ( \times 10^{-16} )</td>
<td>7.29 ( \times 10^{-15} )</td>
</tr>
<tr>
<td>500</td>
<td></td>
<td>4.04 ( \times 10^{-16} )</td>
<td>3.12 ( \times 10^{-15} )</td>
</tr>
<tr>
<td>1000</td>
<td></td>
<td>3.98 ( \times 10^{-16} )</td>
<td>3.28 ( \times 10^{-15} )</td>
</tr>
</tbody>
</table>
Tolerance-sensitive degree (I)

Let

\[ u(x) = \prod_{j=1}^{10} (x - x_j), \]
\[ v(x) = \prod_{j=1}^{10} (x - x_j + 10^{-j}), \]

with \( x_j = (-1)^j (j/2) \).

The roots of \( u(x) \) and \( v(x) \) have decreasing distances equal to 0.1, 0.01, etc.
### Tolerance-sensitive degree (II)

<table>
<thead>
<tr>
<th>$\epsilon$</th>
<th>Fastgcd</th>
<th>UVGCD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>deg</td>
<td>res</td>
</tr>
<tr>
<td>$10^{-2}$</td>
<td>9</td>
<td>0.0045</td>
</tr>
<tr>
<td>$10^{-3}$</td>
<td>8</td>
<td>$2.63 \times 10^{-4}$</td>
</tr>
<tr>
<td>$10^{-4}$</td>
<td>7</td>
<td>$9.73 \times 10^{-6}$</td>
</tr>
<tr>
<td>$10^{-6}$</td>
<td>6</td>
<td>$2.78 \times 10^{-7}$</td>
</tr>
<tr>
<td>$10^{-7}$</td>
<td>5</td>
<td>$8.59 \times 10^{-9}$</td>
</tr>
</tbody>
</table>

(*)Here UVGCD outputs the same result as above due to a different definition of residual.
Let \( k \in \mathbb{N} \), \( u(x) = (x^3 + 3x - 1)(x - 1)^k \) and \( v(x) = u'(x) \). This table shows the residuals (backward errors) and the coefficient-wise errors on the computed GCD w.r.t. the exact GCD, given by Fastgcd.

<table>
<thead>
<tr>
<th>( k )</th>
<th>( \text{res} )</th>
<th>( \text{err} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>( 1.49 \times 10^{-16} )</td>
<td>( 5.18 \times 10^{-13} )</td>
</tr>
<tr>
<td>25</td>
<td>( 2.63 \times 10^{-16} )</td>
<td>( 9.31 \times 10^{-11} )</td>
</tr>
<tr>
<td>35</td>
<td>( 1.96 \times 10^{-16} )</td>
<td>( 1.53 \times 10^{-8} )</td>
</tr>
<tr>
<td>45</td>
<td>( 1.56 \times 10^{-16} )</td>
<td>( 6.61 \times 10^{-6} )</td>
</tr>
</tbody>
</table>

Here QRGCD does not detect a GCD of correct degree for \( k \geq 25 \).
Small leading coefficient

For a given (small) parameter $\alpha \in \mathbb{R}$, let

\[
g(x) = \alpha x^3 + 2x^2 - x + 5, \\
p(x) = x^4 + 7x^2 - x + 1, \quad q(x) = x^3 - x^2 + 4x - 2 \\
u(x) = g(x)p(x), \quad v(x) = g(x)q(x).\]

We applied Fastgcd and QRGCD to this example, with $\alpha$ ranging between $10^{-5}$ and $10^{-15}$.

- For $\alpha < 10^{-5}$, QRGCD fails to recognize the correct GCD degree and outputs a GCD of degree 2.
- Fastgcd always outputs a correct GCD, with a residual of about $10^{-16}$. 
Let $k \in \mathbb{N}$, $n_1 = 25k$, $n_2 = 15k$, $n_3 = 10k$. Define $u_k(x) = g(x)p_k(x)$ and $v_k(x) = g(x)q_k(x)$, where

\[
p_k(x) = (x^{n_1} - 1)(x^{n_2} - 2)(x^{n_3} - 3),
\]
\[
q_k(x) = (x^{n_1} + 1)(x^{n_2} + 5)(x^{n_3} + i),
\]
\[
g(x) = x^4 + 10x^3 + x - 1.
\]
Computational cost (II)

\[ y = 2.06^x - 9.27 \]
Computational cost (III) - Comparison with UVGCD

![Graph comparing computational cost of Fastgcd and UVGCD vs degree]

- **Fastgcd**
- **UVGCD**

The graph illustrates the comparison of computational cost between Fastgcd and UVGCD as the degree increases. The cost is measured in time units on the y-axis.
New efficient algorithms for $\epsilon$-GCD computation have been designed, implemented and tested. In particular, the Fastgcd algorithm has a low computational cost, while retaining good stability properties.

Ideas for further work:
- further exploitation of structure in resultant matrices (e.g., in QR factorization),
- generalization to many polynomials, or to multivariate polynomials.
Definition 2: Optimization approach

**Definition**

*Given polynomials* \(u(x), v(x)\) *and a positive integer* \(k < m, n\), *a polynomial* \(g(x)\) *is an approximate GCD of* \(u(x)\) *and* \(v(x)\) *if*

- \(g(x)\) *has degree* \(k\),
- \(g(x)\) *is an exact divisor of perturbed polynomials* \(\hat{u}(x)\) *and* \(\hat{v}(x)\),
- *the perturbation norm* \(\eta = \|u - \hat{u}\|_2^2 + \|v - \hat{v}\|_2^2\) *is minimized over all the triples* \((\hat{u}, \hat{v}, g)\).

[Corless, Gianni, Trager, Watt ’95], [Karmarkar, Lakshman’97], [Kaltofen, Yang, Zhi ’05]
Definition 2: Optimization approach

Definition

Given polynomials $u(x)$, $v(x)$ and a positive integer $k < m, n$, a polynomial $g(x)$ is an approximate GCD of $u(x)$ and $v(x)$ if

- $g(x)$ has degree $k$,
- $g(x)$ is an exact divisor of perturbed polynomials $\hat{u}(x)$ and $\hat{v}(x)$,
- the perturbation norm $\eta = \|u - \hat{u}\|_2^2 + \|v - \hat{v}\|_2^2$ is minimized over all the triples $(\hat{u}, \hat{v}, g)$.

[Corless, Gianni, Trager, Watt '95], [Karmarkar, Lakshman'97], [Kaltofen, Yang, Zhi '05]
Definition 2: Optimization approach

**Definition**

Given polynomials $u(x)$, $v(x)$ and a positive integer $k < m, n$, a polynomial $g(x)$ is an approximate GCD of $u(x)$ and $v(x)$ if

- $g(x)$ has degree $k$,
- $g(x)$ is an exact divisor of perturbed polynomials $\hat{u}(x)$ and $\hat{v}(x)$,
- the perturbation norm $\eta = \| u - \hat{u} \|_2^2 + \| v - \hat{v} \|_2^2$ is minimized over all the triples $(\hat{u}, \hat{v}, g)$.

[Corless, Gianni, Trager, Watt ’95], [Karmarkar, Lakshman’97], [Kaltofen, Yang, Zhi ’05]
Definition 2: Optimization approach

**Definition**

Given polynomials \( u(x) \), \( v(x) \) and a positive integer \( k < m, n \), a polynomial \( g(x) \) is an approximate GCD of \( u(x) \) and \( v(x) \) if

- \( g(x) \) has degree \( k \),
- \( g(x) \) is an exact divisor of perturbed polynomials \( \hat{u}(x) \) and \( \hat{v}(x) \),
- the perturbation norm \( \eta = \| u - \hat{u} \|_2^2 + \| v - \hat{v} \|_2^2 \) is minimized over all the triples \((\hat{u}, \hat{v}, g)\).

[Corless, Gianni, Trager, Watt '95], [Karmarkar, Lakshman’97], [Kaltofen, Yang, Zhi ’05]
Definition 3: $\delta$-GCD

Let

$$u(x) = u_n \prod_{j=1}^{n} (x - \alpha_j), \quad v(x) = v_m \prod_{j=1}^{m} (x - \beta_j), \quad u_n v_m \neq 0.$$  

Define the $\delta$-neighborhoods of $u(x)$ and $v(x)$ as

$$\mathcal{N}_\delta(u) = \left\{ \hat{u}(x) = u_n \prod_{j=1}^{n} (x - \hat{\alpha}_j) : |\alpha_j - \hat{\alpha}_j| \leq \delta, \; j = 1, \ldots, n \right\},$$

$$\mathcal{N}_\delta(v) = \left\{ \hat{v}(x) = v_m \prod_{j=1}^{m} (x - \hat{\beta}_j) : |\beta_j - \hat{\beta}_j| \leq \delta, \; j = 1, \ldots, m \right\}.$$  

Definition

- A $\delta$-divisor of $u(x)$ and $v(x)$ is a monic polynomial $g(x)$ that divides exactly some pair $\hat{u} \in \mathcal{N}_\delta(u), \; \hat{v} \in \mathcal{N}_\delta(v)$.
- A $\delta$-GCD of $u(x)$ and $v(x)$ is a $\delta$-divisor of maximum degree.
Definition 3: $\delta$-GCD

Let

$$u(x) = u_n \prod_{j=1}^{n} (x - \alpha_j), \quad v(x) = v_m \prod_{j=1}^{m} (x - \beta_j), \quad u_n v_m \neq 0.$$ 

Define the $\delta$-neighborhoods of $u(x)$ and $v(x)$ as

$$N_\delta(u) = \{ \hat{u}(x) = u_n \prod_{j=1}^{n} (x - \hat{\alpha}_j) : |\alpha_j - \hat{\alpha}_j| \leq \delta, \ j = 1, \ldots, n \},$$

$$N_\delta(v) = \{ \hat{v}(x) = v_m \prod_{j=1}^{m} (x - \hat{\beta}_j) : |\beta_j - \hat{\beta}_j| \leq \delta, \ j = 1, \ldots, m \}.$$ 

Definition

- A $\delta$-divisor of $u(x)$ and $v(x)$ is a monic polynomial $g(x)$ that divides exactly some pair $\hat{u} \in N_\delta(u)$, $\hat{v} \in N_\delta(v)$.

- A $\delta$-GCD of $u(x)$ and $v(x)$ is a $\delta$-divisor of maximum degree.

[Pan ’98]
Definition 3: $\delta$-GCD

Let

\[ u(x) = u_n \prod_{j=1}^{n} (x - \alpha_j), \quad v(x) = v_m \prod_{j=1}^{m} (x - \beta_j), \quad u_nv_m \neq 0. \]

Define the $\delta$-neighborhoods of $u(x)$ and $v(x)$ as

\[ \mathcal{N}_{\delta}(u) = \left\{ \hat{u}(x) = u_n \prod_{j=1}^{n} (x - \hat{\alpha}_j) : |\alpha_j - \hat{\alpha}_j| \leq \delta, \; j = 1, \ldots, n \right\}, \]

\[ \mathcal{N}_{\delta}(v) = \left\{ \hat{v}(x) = v_m \prod_{j=1}^{m} (x - \hat{\beta}_j) : |\beta_j - \hat{\beta}_j| \leq \delta, \; j = 1, \ldots, m \right\}. \]

Definition

- A $\delta$-divisor of $u(x)$ and $v(x)$ is a monic polynomial $g(x)$ that divides exactly some pair $\hat{u} \in \mathcal{N}_{\delta}(u)$, $\hat{v} \in \mathcal{N}_{\delta}(v)$.

- A $\delta$-GCD of $u(x)$ and $v(x)$ is a $\delta$-divisor of maximum degree.

[Pan '98] back
Definition 3: \( \delta \)-GCD

Let
\[
    u(x) = u_n \prod_{j=1}^{n} (x - \alpha_j), \quad v(x) = v_m \prod_{j=1}^{m} (x - \beta_j), \quad u_n v_m \neq 0.
\]

Define the \( \delta \)-neighborhoods of \( u(x) \) and \( v(x) \) as
\[
    \mathcal{N}_\delta(u) = \left\{ \hat{u}(x) = u_n \prod_{j=1}^{n} (x - \hat{\alpha}_j) : |\alpha_j - \hat{\alpha}_j| \leq \delta, \; j = 1, \ldots, n \right\},
\]
\[
    \mathcal{N}_\delta(v) = \left\{ \hat{v}(x) = v_m \prod_{j=1}^{m} (x - \hat{\beta}_j) : |\beta_j - \hat{\beta}_j| \leq \delta, \; j = 1, \ldots, m \right\}.
\]

**Definition**

- A \( \delta \)-**divisor** of \( u(x) \) and \( v(x) \) is a monic polynomial \( g(x) \) that divides exactly some pair \( \hat{u} \in \mathcal{N}_\delta(u) \), \( \hat{v} \in \mathcal{N}_\delta(v) \).

- A \( \delta \)-**GCD** of \( u(x) \) and \( v(x) \) is a \( \delta \)-divisor of maximum degree.

[Pan '98]
Let $C \in \mathbb{C}^{n \times n}$ be Cauchy-like. Gaussian elimination with partial pivoting can be applied to $C$ with a computational cost of $O(n^2)$ ([GKO95]). Indeed:

- a step of Gaussian elimination is equivalent to computing a Schur complement,
- Schur complementation preserves displacement structure,
- row permutation on $C$ preserves Cauchy-like structure,
- therefore one may work using generators rather than the whole matrix.
Let $C \in \mathbb{C}^{n \times n}$ be Cauchy-like. Gaussian elimination with partial pivoting can be applied to $C$ with a computational cost of $O(n^2)$ ([GKO95]). Indeed:

- a step of Gaussian elimination is equivalent to computing a Schur complement,
- Schur complementation preserves displacement structure,
- row permutation on $C$ preserves Cauchy-like structure,
- therefore one may work using generators rather than the whole matrix.
Theorem

Let $T \in \mathbb{C}^{n \times n}$ be Toeplitz-like with generators $G, B$. Then

$$C = \mathcal{F} T D_0^{-1} \mathcal{F}^*$$

is Cauchy-like, i.e.,

$$\nabla_{D_1, D_{-1}}(C) = D_1 C - C D_{-1} = \hat{G} \hat{B},$$

where $\mathcal{F}$ is the Fourier matrix, $D_0, D_1, D_{-1}$ are diagonal matrices and

$$\hat{G} = \mathcal{F} G, \quad \hat{B}^* = \mathcal{F} D_0 B^*.$$
Theorem

Let $A \in \mathbb{C}^{m \times n}$. Then there exist

- unitary matrices $U \in \mathbb{C}^{m \times m}$ and $V \in \mathbb{C}^{n \times n}$
- an $m \times n$ real diagonal matrix $\Sigma = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_p)$, with $p = \min\{m, n\}$, such that

$$A = U\Sigma V^H,$$

where

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p \geq 0.$$

- The $\sigma_i$’s are the singular values.
- The columns $u_1, \ldots, u_m$ of $U$ are the left singular vectors.
- The columns $v_1, \ldots, v_m$ of $V$ are the right singular vectors.
If $\sigma_1 \geq \cdots \geq \sigma_r > \sigma_{r+1} = \cdots = 0$ then

- $\text{rank } (A) = r$
- $\ker (A) = \text{span } \{v_{r+1}, \ldots, v_n\}$
- $\text{range } (A) = \text{span } \{u_1, \ldots, u_r\}$
- $A = \sum_{i=1}^{r} \sigma_i u_i v_i^H$

**Theorem**

If $k < r = \text{rank } (A)$ and $A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^H$, then

$$\min_{\text{rank}(B)=k} \|A - B\|_2 = \|A - A_k\|_2 = \sigma_{k+1}.$$
D. A. Bini, P. Boito
A fast algorithm for approximate polynomial gcd based on structured matrix computations.

D. A. Bini, P. Boito
Structured matrix based methods for polynomial $\epsilon$-GCD: analysis and comparisons.
*Proc. ISSAC ’07.*

B. Beckermann, G. Labahn
When are two numerical polynomials relatively prime?
B. Beckermann, G. Labahn
A fast and numerically stable Euclidean-like algorithm for detecting relatively prime numerical polynomials

R. M. Corless, P. M. Gianni, B. M. Trager, S. M. Watt
The Singular Value Decomposition for Approximate Polynomial Systems
Proc. ISSAC ’95.

R. M. Corless, S. M. Watt, L. Zhi
QR Factoring to Compute the GCD of Univariate Approximate Polynomials
I. Z. Emiris, A. Galligo, H. Lombardi
Certified approximate univariate GCDs.

I. Gohberg, T. Kailath, V. Olshevsky
Fast Gaussian elimination with partial pivoting for matrices with displacement structure.

M. Gu
Stable and Efficient Algorithms for Structured Systems of Linear Equations.
G. Heinig
Inversion of generalized Cauchy matrices and other classes of structured matrices.

N. K. Karmarkar, Y. N. Lakshman
Approximate polynomials greatest common divisors and nearest singular polynomials.
*Proc. ISSAC 1996*.

V. Y. Pan
Approximate polynomial GCDs, Padé approximation, polynomial zeros and bipartite graphs.
V. Y. Pan
On Computations with dense structured matrices.

A. Schönhage
Quasi-GCD Computations.

Z. Zeng
The Approximate GCD Of Inexact Polynomials, *to appear*. 