# Extended QRGCD Algorithm

Kosaku Nagasaka and Takaaki Masui

Kobe University and Kobe-Takatsuka High School

09:30-10:00, September 10th

This work was supported in part by Japanese Ministry of Education, Culture, Sports, Science and Technology under Grant-in-Aid for Young Scientists, MEXT KAKENHI (22700011).

# What is “approximate GCD”?

# Approximate GCD

## GCD (Greatest Common Divisor) $\Leftarrow$ Exact GCD

$$f(x) = x^2 + 2x + 1,\ g(x) = x^2 - 1 \ \Rightarrow\ \mathrm{gcd}(f, g) = x + 1$$

The polynomial of maximum degree, which divides $f(x)$ and $g(x)$.

## Approximate GCD (Greatest Common Divisor)

$$f(x) = 0.999x^2 + 1.999x + 1.001,\ g(x) = 1.001x^2 - 0.999$$

$$\Rightarrow\ \mathrm{gcd}(f, g) = 1 \quad (\mathrm{coprime})$$

$$\begin{aligned}\Rightarrow\ \mathrm{agcd}(f, g) &= 1.00063x + 0.999375 \\ &= \mathrm{gcd}(f + (-0.0003343x^2 + 0.0003347x - 0.0003351), \\ &\qquad g + (0.0001666x^2 - 0.0001669x + 0.0001671))\end{aligned}$$

GCD with some consideration of a priori error (perturbation)
hence it's a GCD in the neighborhood of the given polynomials.

# Approximate GCD

## GCD (Greatest Common Divisor) $\Leftarrow$ Exact GCD

$$f(x) = x^2 + 2x + 1,\ g(x) = x^2 - 1 \ \Rightarrow \ \mathrm{gcd}(f, g) = x + 1$$

The polynor … gree, which divides $f(x)$ and $g(x)$.

$\Downarrow$ Adding Some Noise $\Downarrow$

## Approximate GCD (Greatest Common Divisor)

$$f(x) = 0.999x^2 + 1.999x + 1.001,\ g(x) = 1.001x^2 - 0.999$$

$$\Rightarrow \ \mathrm{gcd}(f, g) = 1 \quad \text{(coprime)}$$

$$\Rightarrow \ \mathrm{agcd}(f, g) = 1.00063x + 0.999375$$

$$= \mathrm{gcd}(f + (-0.0003343x^2 + 0.0003347x - 0.0003351),$$
$$g + (0.0001666x^2 - 0.0001669x + 0.0001671))$$

GCD with some consideration of a priori error (perturbation)
hence it's a GCD in the neighborhood of the given polynomials.

# Approximate GCD

## GCD (Greatest Common Divisor) $\Leftarrow$ Exact GCD

$$f(x) = x^2 + 2x + 1,\ g(x) = x^2 - 1 \ \Rightarrow\ \gcd(f,g) = x + 1$$

The polynor … gree, which divides $f(x)$ and $g(x)$.

Adding Some Noise

## Approximate GCD (Greatest Common Divisor)

$$f(x) = 0.999x^2 + 1.999x + 1.001,\ g(x) = 1.001x^2 - 0.999$$

$$\Rightarrow\ \gcd(f,g) = 1 \quad \text{(coprime)}$$

$$\Rightarrow\ \mathrm{agcd}(f,g) = 1.00063x + 0.999375$$

$$= \gcd(f + (-0.0003343x^2 + 0.0003347x - 0.0003351),\ g + (0.0001666x^2 - 0.0001669x + 0.0001671))$$

GCD with some consideration of a priori error (perturbation) hence it's a GCD in the neighborhood of the given polynomials.

# Approximate GCD

## GCD (Greatest Common Divisor) $\Leftarrow$ Exact GCD

$$f(x) = x^2 + 2x + 1,\ g(x) = x^2 - 1 \ \Rightarrow\ \gcd(f, g) = x + 1$$

The polynor … gree, which divides $f(x)$ and $g(x)$.

Adding Some Noise

## Approximate GCD (Greatest Common Divisor)

$$f(x) = 0.999x^2 + 1.999x + 1.001,\ g(x) = 1.001x^2 - 0.999$$

$$\Rightarrow\ \gcd(f, g) = 1 \quad \text{(coprime)}$$

$$\Rightarrow\ \mathrm{agcd}(f, g) = 1.00063x + 0.999375$$

$$= \gcd(f + (-0.0003343x^2 + 0.0003347x - 0.0003351),$$

$$g + (0.0001666x^2 - 0.0001669x + 0.0001671))$$

GCD with some consideration of a priori error (perturbation) hence it's a GCD in the neighborhood of the given polynomials.

# Problem Formulation of approximate GCD

Given: $f(x), g(x) \in \mathbb{R}[x]$, tolerance $\varepsilon \in \mathbb{R}_{\geq 0}$ ($\mathbb{R}[x]$ could be $\mathbb{C}[x]$)
Find: $d(x) \in \mathbb{R}[x]$ ($\Delta_f(x), \Delta_g(x), f_1(x), g_1(x) \in \mathbb{R}[x]$)

$$f(x) + \Delta_f(x) = f_1(x)d(x), \ g(x) + \Delta_g(x) = g_1(x)d(x)$$

$$\deg(\Delta_f) \leq \deg(f), \deg(\Delta_g) \leq \deg(g), \ \|\Delta_f\|_2 < \varepsilon\|f\|_2, \|\Delta_g\|_2 < \varepsilon\|g\|_2$$

$d(x)$: approximate GCD, $f_1(x)$ and $g_1(x)$: approximate cofactors,
$\Delta_f(x)$ and $\Delta_g(x)$: perturbations

Example: $f(x) = 0.999x^2 + 1.999x + 1.001, \ g(x) = 1.001x^2 - 0.999$

$d(x) = 1.00063x + 0.999375, \ \varepsilon = 0.0005796959,$
$f_1(x) = 0.998042x + 1.00129, \ g_1(x) = 1.00054x - 0.999458,$
$\Delta_f(x) = -0.000334269x^2 + 0.000334687x - 0.000335106,$
$\Delta_g(x) = 0.000166643x^2 - 0.000166851x + 0.00016706$

# Problem Formulation of approximate GCD

Given: $f(x), g(x) \in \mathbb{R}[x]$, tolerance $\varepsilon \in \mathbb{R}_{\geq 0}$ ($\mathbb{R}[x]$ could be $\mathbb{C}[x]$)
Find: $d(x) \in \mathbb{R}[x]$ ($\Delta_f(x), \Delta_g(x), f_1(x), g_1(x) \in \mathbb{R}[x]$)

$$f(x) + \Delta_f(x) = f_1(x)d(x),\ g(x) + \Delta_g(x) = g_1(x)d(x)$$

$$\deg(\Delta_f) \leq \deg(f), \deg(\Delta_g) \leq \deg(g),\ \|\Delta_f\|_2 < \varepsilon\|f\|_2, \|\Delta_g\|_2 < \varepsilon\|g\|_2$$

$d(x)$: approximate GCD, $f_1(x)$ and $g_1(x)$: approximate cofactors,
$\Delta_f(x)$ and $\Delta_g(x)$: perturbations

Example: $f(x) = 0.999x^2 + 1.999x + 1.001,\ g(x) = 1.001x^2 - 0.999$

$$d(x) = 1.00063x + 0.999375,\ \varepsilon = 0.0005796959,$$
$$f_1(x) = 0.998042x + 1.00129,\ g_1(x) = 1.00054x - 0.999458,$$
$$\Delta_f(x) = -0.000334269x^2 + 0.000334687x - 0.000335106,$$
$$\Delta_g(x) = 0.000166643x^2 - 0.000166851x + 0.00016706$$

# Known Methods (Revealing-Degree type, univariate polynomial)

**QRGCD (by R.M.Corless, S.M.Watt and L.Zhi, 2004) (widely used)**

- QR factoring of Sylvester matrix (PRS appear on rows of $R$).
- Do this twice (QR factoring detects roots inside the unit circle).

**UVGCD (by Z.Zeng, 2004 and 2011) (stable for multiple roots)**

- QR factoring of subresultant matrices (for the smallest singular value and the corresponding vector).
- Tentative approximate GCD by the least squares.
- Refine approximate GCD by the Gauss-Newton method.

**Fastgcd (by D.A.Bini and P.Boito, 2007 and 2010) (fast and stable)**

- Based on the LU factoring of Sylvester matrix (Toeplitz block).
- Transform it into Cauchy-like matrix and use a modified GKO (Gohberg-Kailath-Olshevsky) method for LU factoring.
- Refine approximate GCD by some refinement method.

# Known Methods (Revealing-Degree type, univariate polynomial)

**QRGCD (by R.M.Corless, S.M.Watt and L.Zhi, 2004) (widely used)**

- QR factoring of Sylvester matrix (PRS appear on rows of $R$).
- Do this twice (QR factoring detects roots inside the unit circle).

**UVGCD (by Z.Zeng, 2004 and 2011) (stable for multiple roots)**

- QR factoring of subresultant matrices (for the smallest singular value and the corresponding vector).
- Tentative approximate GCD by the least squares.
- Refine approximate GCD by the Gauss-Newton method.

**Fastgcd (by D.A.Bini and P.Boito, 2007 and 2010) (fast and stable)**

- Based on the LU factoring of Sylvester matrix (Toeplitz block).
- Transform it into Cauchy-like matrix and use a modified GKO (Gohberg-Kailath-Olshevsky) method for LU factoring.
- Refine approximate GCD by some refinement method.

# Known Methods (Revealing-Degree type, univariate polynomial)

**QRGCD (by R.M.Corless, S.M.Watt and L.Zhi, 2004) (widely used)**

- QR factoring of Sylvester matrix (PRS appear on rows of $R$).
- Do this twice (QR factoring detects roots inside the unit circle).

**UVGCD (by Z.Zeng, 2004 and 2011) (stable for multiple roots)**

- QR factoring of subresultant matrices (for the smallest singular value and the corresponding vector).
- Tentative approximate GCD by the least squares.
- Refine approximate GCD by the Gauss-Newton method.

**Fastgcd (by D.A.Bini and P.Boito, 2007 and 2010) (fast and stable)**

- Based on the LU factoring of Sylvester matrix (Toeplitz block).
- Transform it into Cauchy-like matrix and use a modified GKO (Gohberg-Kailath-Olshevsky) method for LU factoring.
- Refine approximate GCD by some refinement method.

# Known Methods (Revealing-Degree type, univariate polynomial)

**QRGCD (by R.M.Corless, S.M.Watt and L.Zhi, 2004) (widely used)**

- QR factoring of Sylvester matrix (PRS appear on rows of $R$).
- Do this twice (QR factoring detects roots inside the unit circle).

**UVGCD (by Z.Zeng, 2004 and 2011) (stable for multiple roots)**

- QR factoring of subresultant matrices (for the smallest singular value and the corresponding vector).
- Tentative approximate GCD by the least squares.
- Refine approximate GCD by the Gauss-Newton method.

**Fastgcd (by D.A.Bini and P.Boito, 2007 and 2010) (fast and stable)**

- Based on the LU factoring of Sylvester matrix (Toeplitz block).
- Transform it into Cauchy-like matrix and use a modified GKO (Gohberg-Kailath-Olshevsky) method for LU factoring.
- Refine approximate GCD by some refinement method.

# Brief framework of QRGCD.

# Brief framework of the QRGCD algorithm

Sylvester matrix (in this talk)

$$f(x) = f_m x^m + f_{m-1} x^{m-1} + \cdots + f_1 x + f_0,$$
$$g(x) = g_n x^n + g_{n-1} x^{n-1} + \cdots + g_1 x + g_0,$$

$$\mathrm{Syl}(f,g) = \begin{pmatrix} f_m & f_{m-1} & \cdots & f_1 & f_0 & & & & \\ & f_m & f_{m-1} & \cdots & f_1 & f_0 & & & \\ & & \ddots & \ddots & \ldots & \ddots & \ddots & & \\ & & & f_m & f_{m-1} & \cdots & f_1 & f_0 & \\ g_n & g_{n-1} & \cdots & g_1 & g_0 & & & & \\ & g_n & g_{n-1} & \cdots & g_1 & g_0 & & & \\ & & \ddots & \ddots & \ldots & \ddots & \ddots & & \\ & & & g_n & g_{n-1} & \cdots & g_1 & g_0 & \end{pmatrix}$$

$$\in \mathbb{C}^{(m+n)\times(m+n)}$$

# How the QRGCD algorithm works

$d(x) = 1$

$$\mathrm{Syl}(f, g) = Q \begin{pmatrix} r_{1,1} & r_{1,2} & \cdots & \cdots & \cdots & \cdots & r_{1,\ell} \\ & r_{2,2} & r_{2,3} & \cdots & \cdots & \cdots & r_{2,\ell} \\ & & \ddots & \ddots & \ddots & \ddots & \vdots \\ & & & r_{\ell-k,\ell-k} & r_{\ell-k,\ell-k+1} & \cdots & r_{\ell-k,\ell} \\ & & & & r_{\ell-(k-1),\ell-(k-1)} & \cdots & r_{\ell-(k-1),\ell} \\ & & & & & \ddots & \vdots \\ & & & & & & r_{\ell,\ell} \end{pmatrix}$$

$$\frac{\|R^{(k-1)}\|_2}{\|R^{(k)}\|_2} < 10\varepsilon \Longrightarrow d_1(x) := r_{\ell-k,\ell-k}x^k + \cdots + r_{\ell-k,\ell-1}x + r_{\ell-k,\ell}$$

$\ell = m + n$

# How the QRGCD algorithm works

$d(x) = 1$

$$
\mathrm{Syl}(f, g) = Q \begin{pmatrix}
r_{1,1} & r_{1,2} & \cdots & \cdots & \cdots & \cdots & r_{1,\ell} \\
 & r_{2,2} & r_{2,3} & \cdots & \cdots & \cdots & r_{2,\ell} \\
 & & \ddots & \ddots & \ddots & \ddots & \vdots \\
 & & & r_{\ell-k,\ell-k} & r_{\ell-k,\ell-k+1} & \cdots & r_{\ell-k,\ell} \\
 & & & & r_{\ell-(k-1),\ell-(k-1)} & \cdots & r_{\ell-(k-1),\ell} \\
 & & & \boxed{R^{(k)}} & & \ddots & \vdots \\
 & & & & \boxed{R^{(k-1)}} & & r_{\ell,\ell}
\end{pmatrix}
$$

$$
\frac{\|R^{(k-1)}\|_2}{\|R^{(k)}\|_2} < 10\varepsilon \Longrightarrow d_1(x) := r_{\ell-k,\ell-k}x^k + \cdots + r_{\ell-k,\ell-1}x + r_{\ell-k,\ell}
$$

$\ell = m + n$

# How the QRGCD algorithm works

$d(x) = d_1(x),\ f_1(x) \approx f(x) \div d_1(x),\ g_1(x) \approx g(x) \div d_1(x)$

$$\mathrm{Syl}(\mathrm{rev}(f_1), \mathrm{rev}(g_1)) =$$
$$Q \begin{pmatrix} r_{1,1} & r_{1,2} & \cdots & \cdots & \cdots & \cdots & r_{1,\tilde{\ell}} \\ & r_{2,2} & r_{2,3} & \cdots & \cdots & \cdots & r_{2,\tilde{\ell}} \\ & & \ddots & \ddots & \ddots & \ddots & \vdots \\ & & & r_{\tilde{\ell}-k,\tilde{\ell}-k} & r_{\tilde{\ell}-k,\tilde{\ell}-k+1} & \cdots & r_{\tilde{\ell}-k,\tilde{\ell}} \\ & & & & r_{\tilde{\ell}-(k-1),\tilde{\ell}-(k-1)} & \cdots & r_{\tilde{\ell}-(k-1),\tilde{\ell}} \\ & & & & & \ddots & \vdots \\ & & & & & & r_{\tilde{\ell},\tilde{\ell}} \end{pmatrix}$$

(the reversal of $h(x)$ is defined by $\mathrm{rev}(h) = h(x) \mapsto x^{\deg(h)} h(1/x)$)

$\tilde{\ell} = \deg(f_1) + \deg(g_1)$

# How the QRGCD algorithm works

$$d(x) = d_1(x),\ f_1(x) \approx f(x) \div d_1(x),\ g_1(x) \approx g(x) \div d_1(x)$$

$$\mathrm{Syl}(\mathrm{rev}(f_1), \mathrm{rev}(g_1)) =$$

$$Q \begin{pmatrix} r_{1,1} & r_{1,2} & \cdots & \cdots & \cdots & \cdots & r_{1,\tilde{\ell}} \\ & r_{2,2} & r_{2,3} & \cdots & \cdots & \cdots & r_{2,\tilde{\ell}} \\ & & \ddots & \ddots & \ddots & \ddots & \vdots \\ & & & r_{\tilde{\ell}-k,\tilde{\ell}-k} & r_{\tilde{\ell}-k,\tilde{\ell}-k+1} & \cdots & r_{\tilde{\ell}-k,\tilde{\ell}} \\ & & & & r_{\tilde{\ell}-(k-1),\tilde{\ell}-(k-1)} & \cdots & r_{\tilde{\ell}-(k-1),\tilde{\ell}} \\ & & & \boxed{R^{(k)}} & & \ddots & \vdots \\ & & & & \boxed{R^{(k-1)}} & & r_{\tilde{\ell},\tilde{\ell}} \end{pmatrix}$$

$$\frac{\|R^{(k-1)}\|_2}{\|R^{(k)}\|_2} < 10\varepsilon \implies d_2(x) := \mathrm{rev}(r_{\tilde{\ell}-k,\tilde{\ell}-k}x^k + \cdots + r_{\tilde{\ell}-k,\tilde{\ell}-1}x + r_{\tilde{\ell}-k,\tilde{\ell}})$$

$$\tilde{\ell} = \deg(f_1) + \deg(g_1)$$

# How the QRGCD algorithm works

$$d(x) = d_1(x)d_2(x),\ f_1(x) \approx f(x) \div d(x),\ g_1(x) \approx g(x) \div d(x)$$

$$\mathrm{Syl}(\mathrm{rev}(f_1), \mathrm{rev}(g_1)) =$$

$$Q \begin{pmatrix} r_{1,1} & r_{1,2} & \cdots & \cdots & \cdots & \cdots & r_{1,\tilde{\ell}} \\ & r_{2,2} & r_{2,3} & \cdots & \cdots & \cdots & r_{2,\tilde{\ell}} \\ & & \ddots & \ddots & \ddots & \ddots & \vdots \\ & & & r_{\tilde{\ell}-k,\tilde{\ell}-k} & r_{\tilde{\ell}-k,\tilde{\ell}-k+1} & \cdots & r_{\tilde{\ell}-k,\tilde{\ell}} \\ & & & & r_{\tilde{\ell}-(k-1),\tilde{\ell}-(k-1)} & \cdots & r_{\tilde{\ell}-(k-1),\tilde{\ell}} \\ & & & R^{(k)} & & \ddots & \vdots \\ & & & & R^{(k-1)} & & r_{\tilde{\ell},\tilde{\ell}} \end{pmatrix}$$

$$\frac{\|R^{(k-1)}\|_2}{\|R^{(k)}\|_2} < 10\varepsilon \Longrightarrow d_2(x) := \mathrm{rev}(r_{\tilde{\ell}-k,\tilde{\ell}-k}x^k + \cdots + r_{\tilde{\ell}-k,\tilde{\ell}-1}x + r_{\tilde{\ell}-k,\tilde{\ell}})$$

Done twice (Normal and Reversal sides)

$$\tilde{\ell} = \deg(f_1) + \deg(g_1)$$

# The QRGCD algorithm (briefly described)

1. Compute the QR decomposition of $\mathrm{Syl}(f,g)$: $\mathrm{Syl}(f,g) = QR$
2. For integer $k$ satisfying $\|R^{(k)}\|_2 > \varepsilon$ and $\|R^{(k-1)}\|_2 < \varepsilon$, do

   Case1: approximately coprime

   $$\|R^{(0)}\|_2 > \varepsilon$$

   Case2: absolute and relative gap found

   $$\frac{\|R^{(k-1)}\|_2}{\|R^{(k)}\|_2} < 10\varepsilon \Longrightarrow d(x) := \text{the last } k\text{-th row of } R$$

   Case3: relative gap found

   $$\exists k_1,\ \frac{\|R^{(k_1-1)}\|_2}{\|R^{(k_1)}\|_2} < 10\varepsilon \Longrightarrow d(x) := \text{last } k_1\text{-th row}$$

   Case4: no gap found but not coprime

   Otherwise (we will call the Split algorithm)
3. Do the above for the reversals of approximate cofactors. (the reversal of $h(x)$ is defined by $h(x) \mapsto x^{\deg(h)} h(1/x)$)

# The weak points of QRGCD

## How to find candidates of approximate GCD

- QRGCD seeks a row of $R$, which is absolutely close to $d(x)$.
- This allows the QRGCD algorithm to detect a polynomial whose structure is far from the nearest approximate GCD.

$\Longrightarrow$ It should be a relative closeness instead of absolute one.

## How to determine an approximate GCD among candidates

- QRGCD tries to detect a factor of maximum degree at once.
- This has a non-preferred effect which was not shown in QRGCD.
  - It may detect a fake common root inside the unit circle.
  - It may output a polynomial with $\|\Delta_f\| > \varepsilon \|f\|$, $\|\Delta_g\| > \varepsilon \|g\|$.

$\Longrightarrow$ It should be a factor with small perturbations regardless degrees.

# The weak points of QRGCD

## How to find candidates of approximate GCD

- QRGCD seeks a row of $R$, which is absolutely close to $d(x)$.
- This allows the QRGCD algorithm to detect a polynomial whose structure is far from the nearest approximate GCD.

$\Longrightarrow$ It should be a relative closeness instead of absolute one.

## How to determine an approximate GCD among candidates

- QRGCD tries to detect a factor of maximum degree at once.
- This has a non-preferred effect which was not shown in QRGCD.
  - It may detect a fake common root inside the unit circle.
  - It may output a polynomial with $\|\Delta_f\| > \varepsilon \|f\|$, $\|\Delta_g\| > \varepsilon \|g\|$.

$\Longrightarrow$ It should be a factor with small perturbations regardless degrees.

# The weak points of QRGCD

## How to find candidates of approximate GCD

- QRGCD seeks a row of $R$, which is absolutely close to $d(x)$.
- This allows the QRGCD algorithm to detect a polynomial whose structure is far from the nearest approximate GCD.

$\Longrightarrow$ It should be a relative closeness instead of absolute one.

## How to determine an approximate GCD among candidates

- QRGCD tries to detect a factor of maximum degree at once.
- This has a non-preferred effect which was not shown in QRGCD.
  - It may detect a fake common root inside the unit circle.
  - It may output a polynomial with $\|\Delta_f\| > \varepsilon \|f\|$, $\|\Delta_g\| > \varepsilon \|g\|$.

$\Longrightarrow$ It should be a factor with small perturbations regardless degrees.

# The weak points of QRGCD

## How to find candidates of approximate GCD

- QRGCD seeks a row of $R$, which is absolutely close to $d(x)$.
- This allows the QRGCD algorithm to detect a polynomial whose structure is far from the nearest approximate GCD.

$\Longrightarrow$ It should be a relative closeness instead of absolute one.

## How to determine an approximate GCD among candidates

- QRGCD tries to detect a factor of maximum degree at once.
- This has a non-preferred effect which was not shown in QRGCD.
  - It may detect a fake common root inside the unit circle.
  - It may output a polynomial with $\|\Delta_f\| > \varepsilon \|f\|$, $\|\Delta_g\| > \varepsilon \|g\|$.

$\Longrightarrow$ It should be a factor with small perturbations regardless degrees.

# The weak points of QRGCD

## How to find candidates of approximate GCD

- QRGCD seeks a row of $R$, which is absolutely close to $d(x)$.
- This allows the QRGCD algorithm to detect a polynomial whose structure is far from the nearest approximate GCD.

$\Longrightarrow$ It should be a relative closeness instead of absolute one.

## How to determine an approximate GCD among candidates

- QRGCD tries to detect a factor of maximum degree at once.
- This has a non-preferred effect which was not shown in QRGCD.
  - It may detect a fake common root inside the unit circle.
  - It may output a polynomial with $\|\Delta_f\| > \varepsilon \|f\|$, $\|\Delta_g\| > \varepsilon \|g\|$.

$\Longrightarrow$ It should be a factor with small perturbations regardless degrees.

# How improved in ExQRGCD?

# Important fact (1) used in our ExQRGCD

## Relative closeness of the rows of $R$ to the approximate GCD

Let $r(x)$ be a polynomial with coefficients $\vec{r}$ which is a row of $R$, and $d_r(x)$ be a factor of $d(x)$, whose roots are $\{\omega_1, \ldots, \omega_k\}$. Then, an upper bound of relative distance of $r(x)$ from $d_r(x)$ is given by

$$\frac{\| r(x) - d_r(x) \|_2}{\| r(x) \|_2} \leq \sqrt{k+1}\kappa_2(\Omega_*(d_r))\frac{\| \mathrm{Syl}(\Delta_f, \Delta_g) \|_2}{\| r(x) \|_2}$$

where $\Omega_*(d_r)$ is the matrix in $\mathbb{C}^{k\times(m+n)}$, whose $(i,j)$-element is $\omega_i^{m+n-j}$, and $\kappa_2(\Omega_*(d_r))$ denotes the condition number of $\Omega_*(d_r)$.

$\Longrightarrow$ ExQRGCD seeks a row $\vec{r_k}$ such that $\frac{\|R^{(k-1)}\|_2}{\|r_k\|_2}$ is small.

(instead of $\frac{\|R^{(k-1)}\|_2}{\|R^{(k)}\|_2}$ in QRGCD)

# Important fact (2) used in our ExQRGCD

**QR factoring may not detect common roots outside the unit circle**

There exists a pair of polynomials $f(x)$ and $g(x)$ such that the QR decomposition of $\mathrm{Syl}(f, g)$ cannot detect any outside-root factor of the approximate GCD. Moreover, we need to detect such factors from $\mathrm{rev}(f)$ and $\mathrm{rev}(g)$ or their cofactors several times and combine them.

Note that $\mathrm{rev}(h)$ is defined by $h(x) \mapsto x^{\deg(h)} h(1/x)$.

Please note that the same claim is also given in the original QRGCD. Our result is that we proved it more carefully by the different way (based on a property of Sylvester's single sum).

$\Longrightarrow$ ExQRGCD seeks a row $\vec{r_k}$ such that the resulting perturbations $\Delta_f$ and $\Delta_g$ are smallest among the candidates.

# How our ExQRGCD algorithm works

$d(x) = 1,\ f_1(x) = f(x),\ g_1(x) = g(x)$

$$\mathrm{Syl}(f_1, g_1) = Q \begin{pmatrix} r_{1,1} & r_{1,2} & \cdots & \cdots & \cdots & \cdots & r_{1,\ell} \\ & r_{2,2} & r_{2,3} & \cdots & \cdots & \cdots & r_{2,\ell} \\ & & \ddots & \ddots & \ddots & \ddots & \vdots \\ & & & r_{\ell-k,\ell-k} & r_{\ell-k,\ell-k+1} & \cdots & r_{\ell-k,\ell} \\ & & & & r_{\ell-(k-1),\ell-(k-1)} & \cdots & r_{\ell-(k-1),\ell} \\ & & & & & \ddots & \vdots \\ & & & & & & r_{\ell,\ell} \end{pmatrix}$$

$d_1(x) := r_k(x)$ s.t. the corresponding $\vec{r_k}$ meets the condition that $\frac{\|R^{(k-1)}\|_2}{\|r_k\|_2} \ll 1$ and $(\Delta_f,\ \Delta_g)$ w.r.t. $d_1(x)$ satisfies the tolerance.

$\ell = \deg(f_1) + \deg(g_1)$

# How our ExQRGCD algorithm works

$d(x) = 1,\ f_1(x) = f(x),\ g_1(x) = g(x)$

$$\mathrm{Syl}(f_1, g_1) = Q \begin{pmatrix} r_{1,1} & r_{1,2} & \cdots & \cdots & \cdots & \cdots & r_{1,\ell} \\ & r_{2,2} & r_{2,3} & \cdots & \cdots & \cdots & r_{2,\ell} \\ & & \ddots & \ddots & \ddots & \ddots & \vdots \\ & \boxed{\vec{r_k}} & & r_{\ell-k,\ell-k} & r_{\ell-k,\ell-k+1} & \cdots & r_{\ell-k,\ell} \\ & & & & r_{\ell-(k-1),\ell-(k-1)} & \cdots & r_{\ell-(k-1),\ell} \\ & & & & & \ddots & \vdots \\ & & & & \boxed{R^{(k-1)}} & & r_{\ell,\ell} \end{pmatrix}$$

$d_1(x) := r_k(x)$ s.t. the corresponding $\vec{r_k}$ meets the condition that $\frac{\|R^{(k-1)}\|_2}{\|r_k\|_2} \ll 1$ and $(\Delta_f,\ \Delta_g)$ w.r.t. $d_1(x)$ satisfies the tolerance.

$\ell = \deg(f_1) + \deg(g_1)$

# How our ExQRGCD algorithm works

$d(x) = d_1(x),\ f_2(x) = f(x) \div d(x),\ g_2(x) = g(x) \div d(x)$

$$\mathrm{Syl}(\mathrm{rev}(f_2), \mathrm{rev}(g_2)) = Q \begin{pmatrix} r_{1,1} & r_{1,2} & \cdots & \cdots & \cdots & \cdots & r_{1,\ell} \\ & r_{2,2} & r_{2,3} & \cdots & \cdots & \cdots & r_{2,\ell} \\ & & \ddots & \ddots & \ddots & \ddots & \vdots \\ & & & r_{\ell-k,\ell-k} & r_{\ell-k,\ell-k+1} & \cdots & r_{\ell-k,\ell} \\ & & & & r_{\ell-(k-1),\ell-(k-1)} & \cdots & r_{\ell-(k-1),\ell} \\ & & & & & \ddots & \vdots \\ & & & & & & r_{\ell,\ell} \end{pmatrix}$$

$d_2(x) := r_k(x)$ s.t. the corresponding $\vec{r_k}$ meets the condition that $\frac{\|R^{(k-1)}\|_2}{\|r_k\|_2} \ll 1$ and $(\Delta_f,\ \Delta_g)$ w.r.t. $d_1(x)d_2(x)$ satisfies the tolerance.

$\ell = \deg(f_2) + \deg(g_2)$

# How our ExQRGCD algorithm works

$$d(x) = d_1(x),\ f_2(x) = f(x) \div d(x),\ g_2(x) = g(x) \div d(x)$$

$$\mathrm{Syl}(\mathrm{rev}(f_2), \mathrm{rev}(g_2)) =$$

$$Q \begin{pmatrix} r_{1,1} & r_{1,2} & \cdots & \cdots & \cdots & \cdots & r_{1,\ell} \\ & r_{2,2} & r_{2,3} & \cdots & \cdots & \cdots & r_{2,\ell} \\ & & \ddots & \ddots & \ddots & \ddots & \vdots \\ & \boxed{\vec{r_k}} & & r_{\ell-k,\ell-k} & r_{\ell-k,\ell-k+1} & \cdots & r_{\ell-k,\ell} \\ & & & & r_{\ell-(k-1),\ell-(k-1)} & \cdots & r_{\ell-(k-1),\ell} \\ & & & & & \ddots & \vdots \\ & & & & \boxed{R^{(k-1)}} & & r_{\ell,\ell} \end{pmatrix}$$

$d_2(x) := r_k(x)$ s.t. the corresponding $\vec{r_k}$ meets the condition that $\frac{\|R^{(k-1)}\|_2}{\|r_k\|_2} \ll 1$ and $(\Delta_f,\ \Delta_g)$ w.r.t. $d_1(x)d_2(x)$ satisfies the tolerance.

$$\ell = \deg(f_2) + \deg(g_2)$$

# How our ExQRGCD algorithm works

$$d(x) = d_1(x)d_2(x),\ f_3(x) = f(x) \div d(x),\ g_3(x) = g(x) \div d(x)$$

$$\mathrm{Syl}(f_3, g_3) = Q \begin{pmatrix} r_{1,1} & r_{1,2} & \cdots & \cdots & \cdots & \cdots & r_{1,\ell} \\ & r_{2,2} & r_{2,3} & \cdots & \cdots & \cdots & r_{2,\ell} \\ & & \ddots & \ddots & \ddots & \ddots & \vdots \\ & & & r_{\ell-k,\ell-k} & r_{\ell-k,\ell-k+1} & \cdots & r_{\ell-k,\ell} \\ & & & & r_{\ell-(k-1),\ell-(k-1)} & \cdots & r_{\ell-(k-1),\ell} \\ & & & & & \ddots & \vdots \\ & & & & & & r_{\ell,\ell} \end{pmatrix}$$

$d_3(x) := r_k(x)$ s.t. the corresponding $\vec{r_k}$ meets the condition that $\frac{\|R^{(k-1)}\|_2}{\|r_k\|_2} \ll 1$ and $(\Delta_f,\ \Delta_g)$ w.r.t. $d_1 d_2 d_3$ satisfies the tolerance.

$$\ell = \deg(f_3) + \deg(g_3)$$

# How our ExQRGCD algorithm works

$$d(x) = d_1(x)d_2(x),\ f_3(x) = f(x) \div d(x),\ g_3(x) = g(x) \div d(x)$$

$$\mathrm{Syl}(f_3, g_3) =$$

$$Q \begin{pmatrix} r_{1,1} & r_{1,2} & \cdots & \cdots & \cdots & \cdots & r_{1,\ell} \\ & r_{2,2} & r_{2,3} & \cdots & \cdots & \cdots & r_{2,\ell} \\ & & \ddots & \ddots & \ddots & \ddots & \vdots \\ & \boxed{\vec{r_k}} & & r_{\ell-k,\ell-k} & r_{\ell-k,\ell-k+1} & \cdots & r_{\ell-k,\ell} \\ & & & & r_{\ell-(k-1),\ell-(k-1)} & \cdots & r_{\ell-(k-1),\ell} \\ & & & & & \ddots & \vdots \\ & & & & \boxed{R^{(k-1)}} & & r_{\ell,\ell} \end{pmatrix}$$

$d_3(x) := r_k(x)$ s.t. the corresponding $\vec{r_k}$ meets the condition that $\frac{\|R^{(k-1)}\|_2}{\|r_k\|_2} \ll 1$ and $(\Delta_f,\ \Delta_g)$ w.r.t. $d_1 d_2 d_3$ satisfies the tolerance.

$$\ell = \deg(f_3) + \deg(g_3)$$

# How our ExQRGCD algorithm works

$$d(x) = d_1(x)d_2(x)d_3(x),\ f_4(x) = f(x) \div d(x),\ g_4(x) = g(x) \div d(x)$$

$$\mathrm{Syl}(\mathrm{rev}(f_4), \mathrm{rev}(g_4)) =$$

$$Q \begin{pmatrix} r_{1,1} & r_{1,2} & \cdots & \cdots & \cdots & \cdots & r_{1,\ell} \\ & r_{2,2} & r_{2,3} & \cdots & \cdots & \cdots & r_{2,\ell} \\ & & \ddots & \ddots & \ddots & \ddots & \vdots \\ & & & r_{\ell-k,\ell-k} & r_{\ell-k,\ell-k+1} & \cdots & r_{\ell-k,\ell} \\ & & & & r_{\ell-(k-1),\ell-(k-1)} & \cdots & r_{\ell-(k-1),\ell} \\ & & & & & \ddots & \vdots \\ & & & & & & r_{\ell,\ell} \end{pmatrix}$$

There is no vector $\vec{r_k}$ satisfying the condition.

$\Longrightarrow$ Approximately coprime w.r.t. the ourside roots.

$$\ell = \deg(f_4) + \deg(g_4)$$

# How our ExQRGCD algorithm works

$d(x) = d_1(x)d_2(x)d_3(x),\ f_4(x) = f(x) \div d(x),\ g_4(x) = g(x) \div d(x)$

$$\mathrm{Syl}(\mathrm{rev}(f_4), \mathrm{rev}(g_4)) =$$

$$Q \begin{pmatrix} r_{1,1} & r_{1,2} & \cdots & \cdots & \cdots & \cdots & r_{1,\ell} \\ & r_{2,2} & r_{2,3} & \cdots & \cdots & \cdots & r_{2,\ell} \\ & & \ddots & \ddots & \ddots & \ddots & \vdots \\ & \boxed{\vec{r}_k} & & r_{\ell-k,\ell-k} & r_{\ell-k,\ell-k+1} & \cdots & r_{\ell-k,\ell} \\ & & & & r_{\ell-(k-1),\ell-(k-1)} & \cdots & r_{\ell-(k-1),\ell} \\ & & & & & \ddots & \vdots \\ & & & & \boxed{R^{(k-1)}} & & r_{\ell,\ell} \end{pmatrix}$$

There is no vector $\vec{r}_k$ satisfying the condition.

$\Longrightarrow$ Approximately coprime w.r.t. the ourside roots.

$\ell = \deg(f_4) + \deg(g_4)$

# How our ExQRGCD algorithm works

$$d(x) = d_1(x)d_2(x)d_3(x),\ f_4(x) = f(x) \div d(x),\ g_4(x) = g(x) \div d(x)$$

$$\mathrm{Syl}(f_4, g_4) = \\ Q \begin{pmatrix} r_{1,1} & r_{1,2} & \cdots & \cdots & \cdots & \cdots & r_{1,\ell} \\ & r_{2,2} & r_{2,3} & \cdots & \cdots & \cdots & r_{2,\ell} \\ & & \ddots & \ddots & \ddots & \ddots & \vdots \\ & & & r_{\ell-k,\ell-k} & r_{\ell-k,\ell-k+1} & \cdots & r_{\ell-k,\ell} \\ & & & & r_{\ell-(k-1),\ell-(k-1)} & \cdots & r_{\ell-(k-1),\ell} \\ & & & & & \ddots & \vdots \\ & & & & & & r_{\ell,\ell} \end{pmatrix}$$

There is no vector $\vec{r_k}$ satisfying the condition.
$\Longrightarrow$ Approximately coprime w.r.t. the ourside roots.

$$\ell = \deg(f_4) + \deg(g_4)$$

# How our ExQRGCD algorithm works

$$d(x) = d_1(x)d_2(x)d_3(x),\ f_4(x) = f(x) \div d(x),\ g_4(x) = g(x) \div d(x)$$

$$\mathrm{Syl}(f_4, g_4) =$$

$$Q \begin{pmatrix} r_{1,1} & r_{1,2} & \cdots & \cdots & \cdots & \cdots & r_{1,\ell} \\ & r_{2,2} & r_{2,3} & \cdots & \cdots & \cdots & r_{2,\ell} \\ & & \ddots & \ddots & \ddots & \ddots & \vdots \\ & \boxed{\vec{r}_k} & & r_{\ell-k,\ell-k} & r_{\ell-k,\ell-k+1} & \cdots & r_{\ell-k,\ell} \\ & & & & r_{\ell-(k-1),\ell-(k-1)} & \cdots & r_{\ell-(k-1),\ell} \\ & & & & & \ddots & \vdots \\ & & & & \boxed{R^{(k-1)}} & & r_{\ell,\ell} \end{pmatrix}$$

There is no vector $\vec{r}_k$ satisfying the condition.
$\Longrightarrow$ Approximately coprime w.r.t. the inside roots.

$$\ell = \deg(f_4) + \deg(g_4)$$

# How our ExQRGCD algorithm works

$$\underline{\underline{d(x) = d_1(x)d_2(x)d_3(x),\ f_1(x) = f(x) \div d(x),\ g_1(x) = g(x) \div d(x)}}$$

$$\mathrm{Syl}(f_4, g_4) =$$

$$Q \begin{pmatrix} r_{1,1} & r_{1,2} & \cdots & \cdots & \cdots & \cdots & r_{1,\ell} \\ & r_{2,2} & r_{2,3} & \cdots & \cdots & \cdots & r_{2,\ell} \\ & & \ddots & \ddots & \ddots & \ddots & \vdots \\ & \vec{r_k} & & r_{\ell-k,\ell-k} & r_{\ell-k,\ell-k+1} & \cdots & r_{\ell-k,\ell} \\ & & & & r_{\ell-(k-1),\ell-(k-1)} & \cdots & r_{\ell-(k-1),\ell} \\ & & & & & \ddots & \vdots \\ & & & & R^{(k-1)} & & r_{\ell,\ell} \end{pmatrix}$$

There is no vector $\vec{r_k}$ satisfying the condition.

$\Longrightarrow$ Approximately coprime w.r.t. the inside roots.

Done several times (Normal, Reversal, Normal, ...)

# Our ExQRGCD algorithm (briefly described)

1. Compute the QR decomposition of $\mathrm{Syl}(f, g)$: $\mathrm{Syl}(f, g) = QR$
2. Do until $\|R^{(k)}\|_2 > \varepsilon\sqrt{m+n}$

   Case1: approximately coprime

   $$\|R^{(0)}\|_2 > \varepsilon\sqrt{m+n}$$

   Case2: a factor of approximate GCD found

   $d(x) := r(x)$ having the smallest $\varepsilon_r = \frac{\|R^{(k-1)}\|_2}{\|r\|_2}$.

   Case3: no factor found

   The 3 smallest $\varepsilon_r$s found no factor $\Rightarrow$ **Split**
   Or goto Step 3 (if #loop $\geq$ the threshold)
   (The threshold is 3 in our implementation)
3. Do the above for the reversals of approximate cofactors.
   (Until approximately coprime is detected twice successively)

# Short Summary: ExQRGCD against QRGCD

## Advantage of ExQRGCD

- It seeks approximate GCD within the given tolerance.
- It seeks a factor such that the smallest perturbation among the candidates.
  $\Rightarrow$ The resulting degree may be larger than QRGCD in general.
- It uses the relative distance bound.
  $\Rightarrow$ It may not detect a factor having fake common roots.
  (even when the PRS has some outside roots instead of inside)

## Weak point of ExQRGCD

- Slower than QRGCD.
  (since ExQRGCD computes QR factoring several times)

# Short Summary: ExQRGCD against QRGCD

## Advantage of ExQRGCD

- It seeks approximate GCD within the given tolerance.
- It seeks a factor such that the smallest perturbation among the candidates.
  $\Rightarrow$ The resulting degree may be larger than QRGCD in general.
- It uses the relative distance bound.
  $\Rightarrow$ It may not detect a factor having fake common roots.
  (even when the PRS has some outside roots instead of inside)

## Weak point of ExQRGCD

- Slower than QRGCD.
  (since ExQRGCD computes QR factoring several times)

# Numerical Experiments

# Numerical Experiments against SNAP's QRGCD

**Random Polynomials (100 pairs of $(f, g)$ for each $i = 1, \ldots, 10$)**

$$f(x) = f_1(x)d(x),\ g(x) = g_1(x)d(x),\ d(x) = \sum_{j=0}^{5i} d_j x^j,$$
$$f_1(x) = \sum_{j=0}^{5i} f_{1,j} x^j,\ g_1(x) = \sum_{j=0}^{5i} g_{1,j} x^j$$

where $f_{1,j},\ g_{1,j},\ d_j \in [-99, 99] \subset \mathbb{Z}$ is randomly chosen, $f(x), g(x)$ are normalized ($\|f\|_2 = \|g\|_2 = 1$) and rounded with *Digits* := 10. We computed with tolerance $10^{-5}$. Moreover, we used Maple 16 with Digits := 16 on Linux (i7 3.30GHz and 64GB mem.).

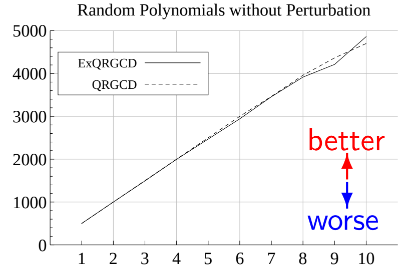
| Sum of detected degrees | The resulting perturbation |
|---|---|

- ExQRGCD is 1.59 times slower than QRGCD.
- QRGCD failed 11 times and didn't meet the tolerance 6 times.

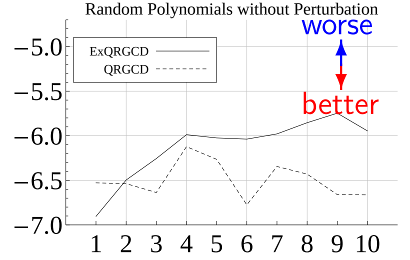
# Numerical Experiments against SNAP's QRGCD

**Random Polynomials (100 pairs of $(f, g)$ for each $i = 1, \ldots, 10$)**

$$f(x) = f_1(x)d(x),\ g(x) = g_1(x)d(x),\ d(x) = \sum_{j=0}^{5i} d_j x^j,$$
$$f_1(x) = \sum_{j=0}^{5i} f_{1,j} x^j,\ g_1(x) = \sum_{j=0}^{5i} g_{1,j} x^j$$




Sum of detected degrees




The resulting perturbation

- ExQRGCD is 1.59 times slower than QRGCD.
- QRGCD failed 11 times and didn't meet the tolerance 6 times.

# Numerical Experiments against SNAP's QRGCD

**Random Polynomials with Perturbations (100 pairs of $(f, g)$ for $i$)**

$f(x) = f_1(x)d(x)/\|f_1 d\|_2 + 10^{-8}\Delta_f(x)/\|\Delta_f\|_2,\ \Delta_f(x) = \sum_{j=0}^{10i} \Delta_{f\,j} x^j,$

$g(x) = g_1(x)d(x)/\|g_1 d\|_2 + 10^{-8}\Delta_g(x)/\|\Delta_g\|_2,\ \Delta_g(x) = \sum_{j=0}^{10i} \Delta_{g\,j} x^j$

where $\Delta_{f\,j},\ \Delta_{g\,j} \in [-99, 99] \subset \mathbb{Z}$ is randomly chosen,

$f_1(x)$, $g_1(x)$, $d(x)$ are the polynomials of the previous Example and rounded with *Digits* $:= 10$. We computed with tolerance $10^{-5}$.

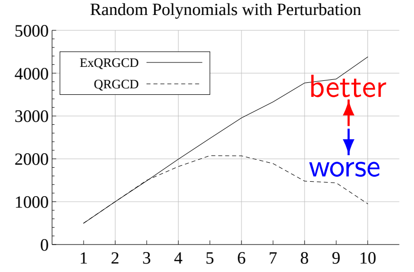
| Sum of detected degrees | The resulting perturbation |
|---|---|

- ExQRGCD is 1.99 times slower than QRGCD.
- QRGCD failed 315 times and didn't meet the tolerance 6 times.

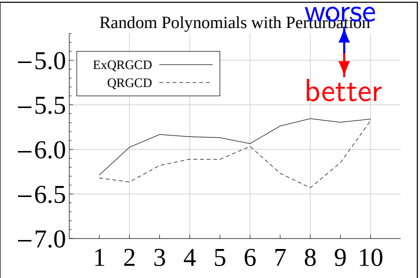
# Numerical Experiments against SNAP's QRGCD

## Random Polynomials with Perturbations (100 pairs of $(f, g)$ for $i$)

$$f(x) = f_1(x)d(x)/\|f_1 d\|_2 + 10^{-8}\Delta_f(x)/\|\Delta_f\|_2,\ \Delta_f(x) = \textstyle\sum_{j=0}^{10i} \Delta_{f_j} x^j,$$
$$g(x) = g_1(x)d(x)/\|g_1 d\|_2 + 10^{-8}\Delta_g(x)/\|\Delta_g\|_2,\ \Delta_g(x) = \textstyle\sum_{j=0}^{10i} \Delta_{g_j} x^j$$




Sum of detected degrees




The resulting perturbation

- ExQRGCD is 1.99 times slower than QRGCD.
- QRGCD failed 315 times and didn't meet the tolerance 6 times.

# Numerical Experiments against SNAP's QRGCD

**Fake Common Roots (100 pairs of $(f,g)$ for each $i=1,\dots,10$)**

$$f=d\cdot\prod_{j=1}^{2i}(x-\omega_{f,j})\prod_{j=1}^{2i}(x-\hat{\omega}_{f,j}),\ g=d\cdot\prod_{j=1}^{2i}(x-\omega_{g,j})\prod_{j=1}^{2i}(x-\hat{\omega}_{g,j}),$$

$$d=\prod_{j=1}^{3i}(x-\omega_{d,j})\prod_{j=1}^{3i}(x-\hat{\omega}_{d,j}),\ \omega_{\cdot,j}=O(10^{-2}),\ \hat{\omega}_{\cdot,j}=O(10^{2})$$

where $\omega_{\cdot,j},\hat{\omega}_{\cdot,j}$ are randomly chosen, $f(x),g(x)$ are normalized (i.e. $\|f(x)\|_2=\|g(x)\|_2=1$) and rounded with *Digits* $:=10$. We computed with tolerance $10^{-5}$. Note that the degree of approximate GCD $\geq 6i$.

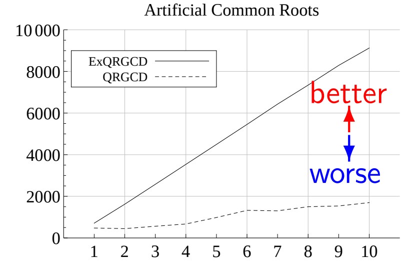
| Sum of detected degrees | The resulting perturbation |
|---|---|

- ExQRGCD is 39.8 times slower than QRGCD.
- QRGCD failed 624 times and didn't meet the tolerance 119 times.

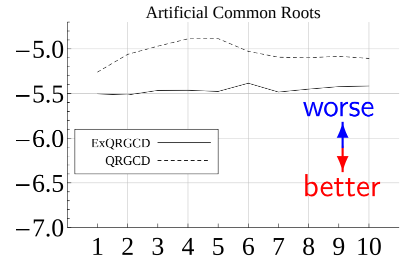
# Numerical Experiments against SNAP's QRGCD

**Fake Common Roots (100 pairs of $(f, g)$ for each $i = 1, \ldots, 10$)**

$f = d \cdot \prod_{j=1}^{2i}(x - \omega_{f,j}) \prod_{j=1}^{2i}(x - \hat{\omega}_{f,j}), g = d \cdot \prod_{j=1}^{2i}(x - \omega_{g,j}) \prod_{j=1}^{2i}(x - \hat{\omega}_{g,j}),$
$d = \prod_{j=1}^{3i}(x - \omega_{d,j}) \prod_{j=1}^{3i}(x - \hat{\omega}_{d,j}), \omega_{\cdot,j} = O(10^{-2}), \hat{\omega}_{\cdot,j} = O(10^{2})$




Sum of detected degrees




The resulting perturbation

- ExQRGCD is 39.8 times slower than QRGCD.
- QRGCD failed 624 times and didn't meet the tolerance 119 times.

# Numerical Experiments against Fastgcd/UVGCD

## Examples in Bini and Boito (2007 and 2010)

- Mignotte-like polynomials (Ex 8.2.1 and 8.2.2 in Boito (2007)),
- An ill-conditioned case (Ex 8.4.1 in Boito (2007)),
- Other examples in Boito (2007), for real univariate polynomials.

## The results

- For Example 8.2.1,
  ExQRGCD is not better than Fastgcd but same as UVGCD.
- For Example 8.2.2, ExQRGCD is almost better than others.
- For Example 8.4.1,
  ExQRGCD is not good though it detected the correct degree.
- For most of other examples,
  ExQRGCD is not better than Fastgcd and UVGCD.

# Numerical Experiments against Fastgcd/UVGCD

## Examples in Bini and Boito (2007 and 2010)

- Mignotte-like polynomials (Ex 8.2.1 and 8.2.2 in Boito (2007)),
- An ill-conditioned case (Ex 8.4.1 in Boito (2007)),
- Other examples in Boito (2007), for real univariate polynomials.

## The results

- For Example 8.2.1,
  ExQRGCD is not better than Fastgcd but same as UVGCD.
- For Example 8.2.2, ExQRGCD is almost better than others.
- For Example 8.4.1,
  ExQRGCD is not good though it detected the correct degree.
- For most of other examples,
  ExQRGCD is not better than Fastgcd and UVGCD.

# Summary

## The weak points of QRGCD and improvements in ExQRGCD

- Absolute closeness and once detection in QRGCD are the issue.
- Our contribution: relative closeness and conservative detection.

## Numerical Experiments

- ExQRGCD is much better than QRGCD in the following points:
  - Degree of detected approximate GCD: $\deg(d)$.
  - Size of resulting perturbations: $\|\Delta_f\|_2$ and $\|\Delta_g\|_2$.
- However, ExQRGCD still should be improved:
  - ExQRGCD is slower than QRGCD.
  - ExQRGCD is still not better than UVGCD and Fastgcd.
    - But there are polynomials for which ExQRGCD is better.

# Summary

## The weak points of QRGCD and improvements in ExQRGCD

- Absolute closeness and once detection in QRGCD are the issue.
- Our contribution: relative closeness and conservative detection.

## Numerical Experiments

- ExQRGCD is much better than QRGCD in the following points:
  - Degree of detected approximate GCD: $\deg(d)$.
  - Size of resulting perturbations: $\|\Delta_f\|_2$ and $\|\Delta_g\|_2$.
- However, ExQRGCD still should be improved:
  - ExQRGCD is slower than QRGCD.
  - ExQRGCD is still not better than UVGCD and Fastgcd.
    - But there are polynomials for which ExQRGCD is better.

# Summary

## The weak points of QRGCD and improvements in ExQRGCD

- Absolute closeness and once detection in QRGCD are the issue.
- Our contribution: relative closeness and conservative detection.

## Numerical Experiments

- ExQRGCD is much better than QRGCD in the following points:
  - Degree of detected approximate GCD: $\deg(d)$.
  - Size of resulting perturbations: $\|\Delta_f\|_2$ and $\|\Delta_g\|_2$.
- However, ExQRGCD still should be improved:
  - ExQRGCD is slower than QRGCD.
  - ExQRGCD is still not better than UVGCD and Fastgcd.
    - But there are polynomials for which ExQRGCD is better.

# Thanks for your attention!