ITERATIVE PROJECTION METHODS FOR SPARSE LINEAR SYSTEMS AND EIGENPROBLEMS
CHAPTER 7 : SHORT RECURRENCES

Heinrich Voss

voss@tu-harburg.de

Hamburg University of Technology
Institute of Numerical Simulation

TUHH
Technische Universität Hamburg-Harburg

Short recurrence

In this Section we consider methods which determine approximations $x^k \in x^0 + \mathcal{K}_k(r^0, A)$ from short recurrences.

By the Theorem of Faber & Manteuffel these methods can not minimize the residuum. Hence, measured by the number of iteration steps necessary to reduce the residual norm by a given factor they must be outperformed by GMRES.

However, the arithmetic work necessary to obtain a given accuracy can be much smaller than for GMRES.

BiCG method: Lanczos (1952), Fletcher (1976)

1: $r^0 = b - Ax^0$
2: Choose $\tilde{r}^0$ with $\alpha_0 = (\tilde{r}^0)^T r^0 \neq 0$
3: $d^1 = r^0$; $\tilde{d}^1 = \tilde{r}^0$
4: **for** $k = 1, 2, \ldots$ until convergence **do**
5: $\quad s^k = Ad^k$
6: $\quad \gamma_k = (\tilde{d}^k)^T s^k$
7: $\quad \tau_k = \alpha_{k-1} / \gamma^k$
8: $\quad x^k = x^{k-1} + \tau_k d^k$
9: $\quad r^k = r^{k-1} - \tau_k s^k$
10: $\quad \tilde{r}^k = \tilde{r}^{k-1} - \tau_k A^T \tilde{d}^k$
11: $\quad \beta_k = 1 / \alpha_{k-1}$
12: $\quad \alpha_k = (\tilde{r}^k)^T r^k; \quad \beta_k = \alpha_k \beta_k$
13: $\quad d^{k+1} = r^k + \beta_k d^k$
14: $\quad \tilde{d}^{k+1} = \tilde{r}^k + \beta_k \tilde{d}^k$
15: **end for**

Cost fo BiCG

2 matrix-vector products
2 scalar products
5 _axpy

Storage requirements: 6 vectors

If A is symmetric and positive definite, and if $\tilde{r}^0 = r^0$, then $\tilde{r}^k = r^k$ and $\tilde{d}^k = d^k$ for every k, and the BiCG method coincides with the CG method.

Whereas for SPD matrices (in exact arithmetic) the method can stop only if the solution $x^k = x^*$ of the linear system $Ax = b$ is reached, in the general case it can break down with $(\tilde{r}^k)^T r^k = 0$ or $(\tilde{d}^k)^T A d^k = 0$.

BiCG ct.

The orthogonality of the residuals and the conjugacy of the search directions in the CG method have to be replaced by the following properties of the BiCG method:

4 vector sequences r^k, $\tilde{r}^k$, d^k and $\tilde{d}^k$ are determined such that r^k and $\tilde{r}^k$ are biorthogonal, i.e.

$$(\tilde{r}^i)^T r^j = 0 \quad \text{for } i \neq j$$

and such that d^k und $\tilde{d}^k$ are biconjugate, i.e.

$$(\tilde{d}^i)^T A d^j = 0 \quad \text{for } i \neq j.$$

Theorem 7.1

The vectors r^j, $\tilde{r}^j$, $j = 0, \ldots, k$ and d^j, $\tilde{d}^j$, $j = 1, \ldots, k+1$, produced by the biconjugate gradient method are such that

$$(\tilde{r}^i)^T r^j = 0 \quad \text{for } i \neq j \quad \text{(biorthogonality property)} \qquad (*)$$
$$(\tilde{d}^i)^T A d^j = 0 \quad \text{for } i \neq j \quad \text{(biconjugacy property)} \qquad (**).$$

Proof:

We prove by induction that (*) and (**) hold for $0 \leq i, j \leq k$ and $1 \leq i, j \leq k+1$, respectively.

Proof ct.

For $k = 0$ nothing has to be shown. Assume that the statement is true for some $k \geq 0$. Then it follows for $j = 0, \ldots, k$ (assuming $\beta_0 = 0$ if necessary)

$$\begin{aligned} (\tilde{r}^j)^T r^{k+1} &= (\tilde{r}^j)^T r^k - \tau_{k+1} (\tilde{r}^j)^T A d^{k+1} \\ &= (\tilde{r}^j)^T r^k - \tau_{k+1} (\tilde{d}^{j+1} - \beta_j \tilde{d}^j)^T A d^{k+1}. \end{aligned}$$

Hence, $(\tilde{r}^j)^T r^{k+1} = 0$ for $j < k$ by assumption and for $j = k$ by the definition of τ_{k+1}. In a similar way we get $(r^j)^T \tilde{r}^{k+1} = 0$ for $j = 0, \ldots, k$, and therefore, (*) is proved for k replaced by $k + 1$.

Proof ct.

For $j = 1, \ldots, k+1$

$$\begin{aligned}(d^{k+2})^T A^T \tilde{d}^j &= (r^{k+1})^T A^T \tilde{d}^j + \beta_{k+1} (d^{k+1})^T A^T \tilde{d}^j \\ &= (r^{k+1})^T (\tilde{r}^{j-1} - \tilde{r}^j)/\tau_j + \beta_{k+1} (d^{k+1})^T A^T \tilde{d}^j.\end{aligned}$$

For $j \le k$ we obtain $(d^{k+2})^T A^T \tilde{d}^j = 0$ by assumption.

For $j = k+1$ one gets from the definition of γ_{k+1}

$$\begin{aligned}(d^{k+2})^T A^T \tilde{d}^{k+1} &= (r^{k+1})^T (\tilde{r}^k - \tilde{r}^{k+1})/\tau_{k+1} + \beta_{k+1}\gamma_{k+1} \\ &= -\alpha_{k+1}/\tau_{k+1} + \beta_{k+1}\gamma_{k+1} \\ &= -\alpha_{k+1}\gamma_{k+1}/\alpha_k + \beta_{k+1}\gamma_{k+1},\end{aligned}$$

and the definition of β_{k+1} yields $(d^{k+2})^T A^T \tilde{d}^{k+1} = 0$. Likewise $(\tilde{d}^{k+2})^T A^T d^j = 0$ can be established for $j = 1, \ldots, k+1$, and therefore, (**) holds for k replaced by $k+1$. This completes the proof. □

BiCG ct.

A direct consequence of the biorthogonality of r^j and $\tilde{r}^j$ is that the BiCG method stops after at most n Steps.

From $d^{k+1} = r^k + \beta_k d^k$ it follows by induction

$$d^{k+1} = r^k + \sum_{j=0}^{k-1} \delta_j r^j.$$

Hence, d^{k+1} and $\tilde{r}^j$ are orthogonal for $j > k$, and correspondingly $\tilde{d}^{k+1}$ and r^j are orthogonal for $j > k$.

BiCG ct.

As for the CG method it follows by induction

$$\text{span}\{d^1, \ldots, d^k\} = \text{span}\{r^0, Ar^0, \ldots, A^{k-1}r^0\}.$$

Hence, the iterates are contained in the shifted Krylov space $x^0 + \mathcal{K}_k(r^0, A)$.

Similarly, it holds that

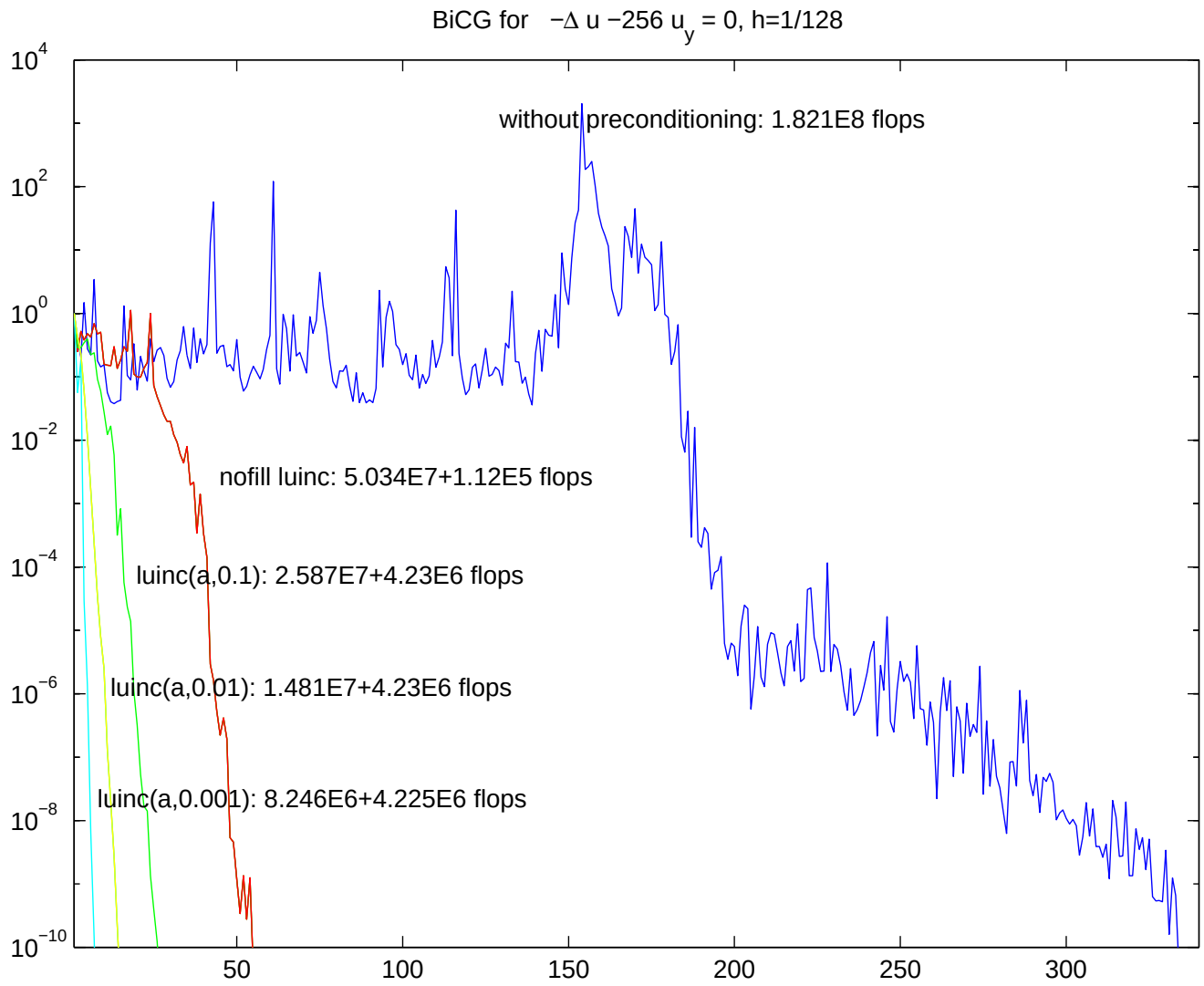
$$\text{span}\{\tilde{r}^0, \ldots, \tilde{r}^{k-1}\} = \text{span}\{\tilde{r}^0, A^T\tilde{r}^0, \ldots, (A^T)^{k-1}\tilde{r}^0\}.$$

Hence, the biorthogonality $(\tilde{r}^j)^T r^k = 0$, $j = 0, \ldots, k-1$ is equivalent to

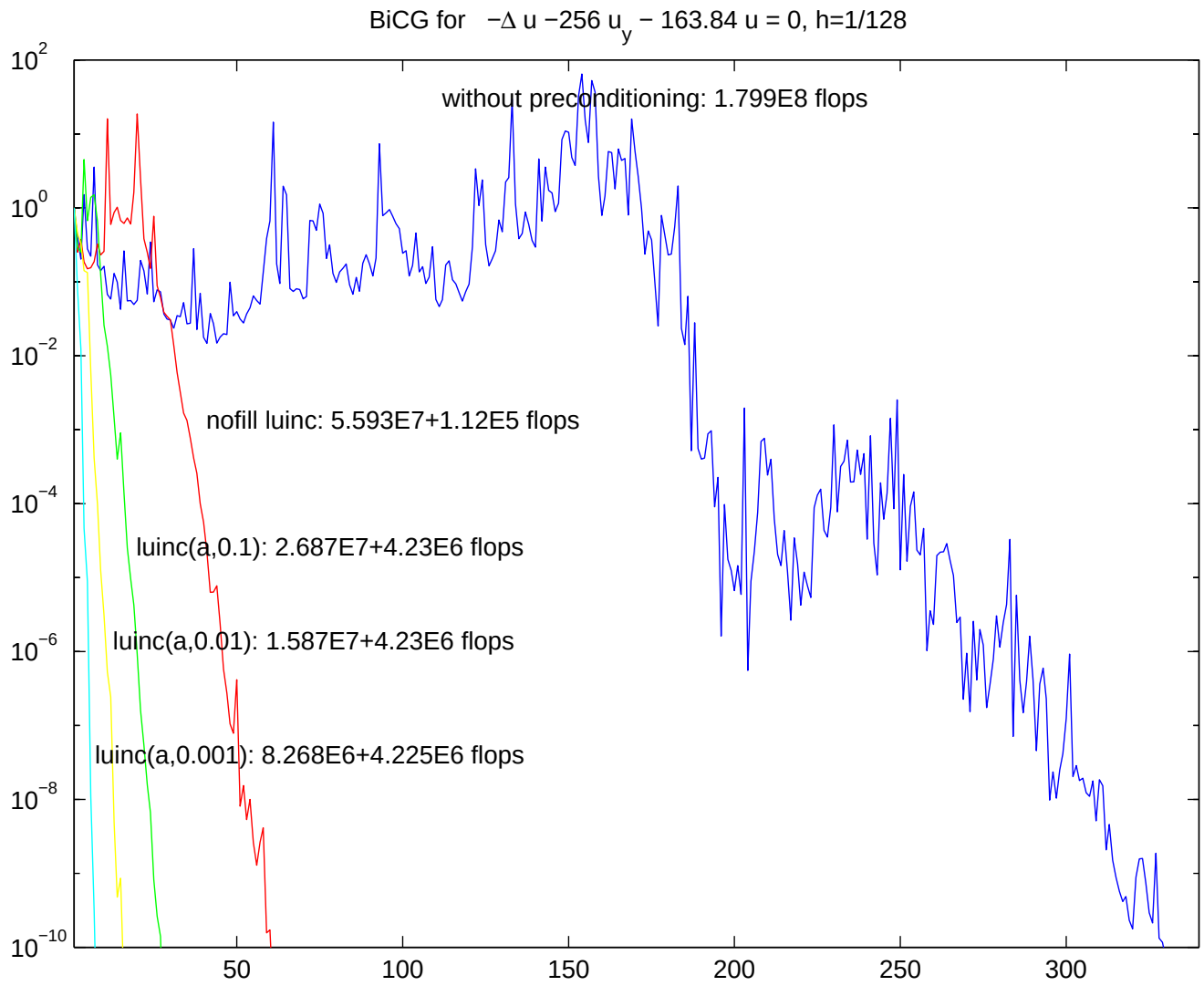
$$((A^T)^j\tilde{r}^0)^T(b - Ax^k) = 0,\ j = 0, \ldots, k-1.$$

Therefore, the iterate x^k of the BiCG method is the Petrov-Galerkin approximation with respect to the ansatz space $V_k := \mathcal{K}_k(r^0, A)$ and the test space $W_k = \mathcal{K}_k(\tilde{r}^0, A^T)$.

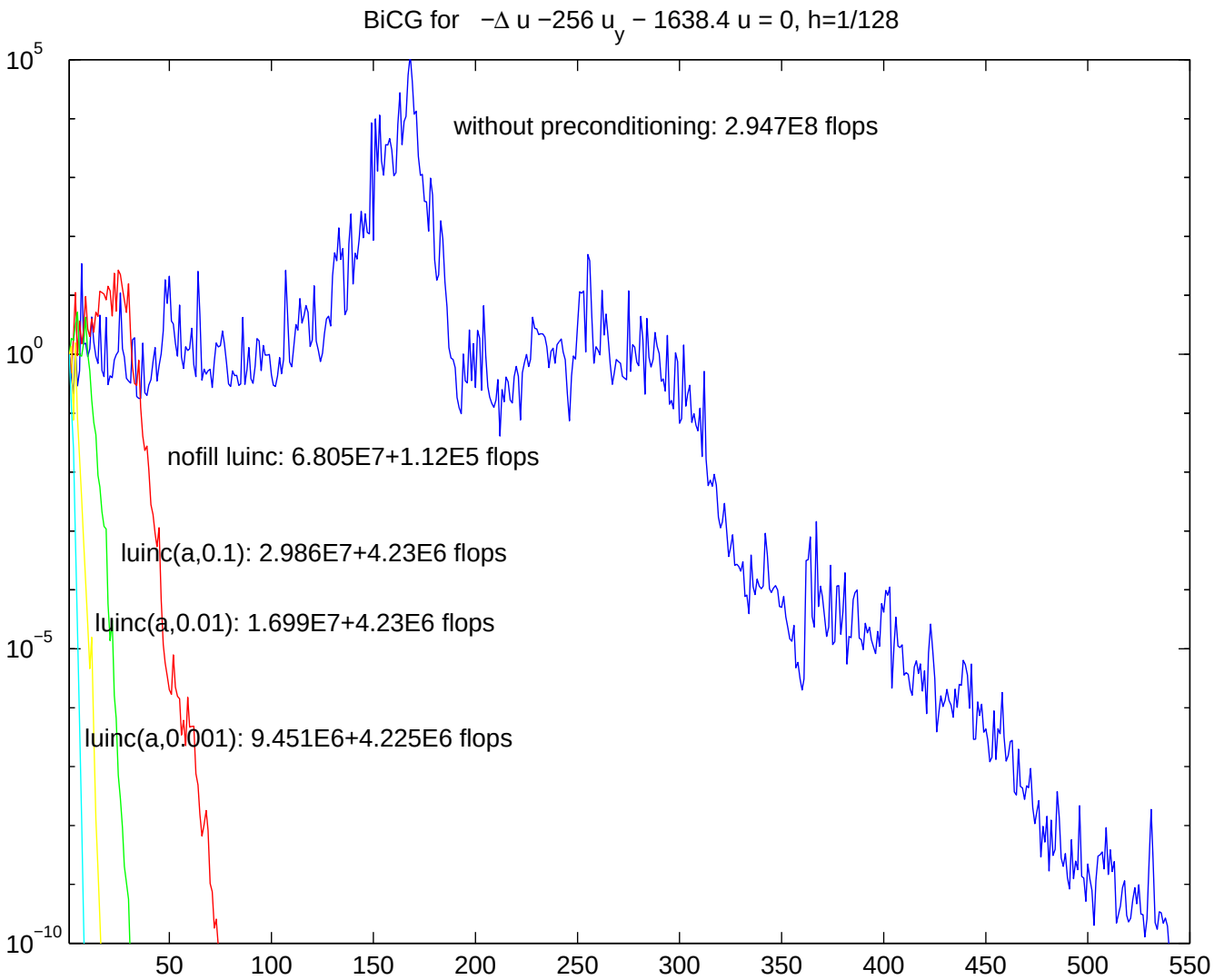
Example


BiCG for −Δ u −256 u_y = 0, h=1/128
10^4
10^2
10^0
10^−2
10^−4
10^−6
10^−8
10^−10
50
100
150
200
250
300
without preconditioning: 1.821E8 flops
nofill luinc: 5.034E7+1.12E5 flops
luinc(a,0.1): 2.587E7+4.23E6 flops
luinc(a,0.01): 1.481E7+4.23E6 flops
luinc(a,0.001): 8.246E6+4.225E6 flops

Example


BiCG for −Δ u −256 u_y − 163.84 u = 0, h=1/128
without preconditioning: 1.799E8 flops
nofill luinc: 5.593E7+1.12E5 flops
luinc(a,0.1): 2.687E7+4.23E6 flops
luinc(a,0.01): 1.587E7+4.23E6 flops
luinc(a,0.001): 8.268E6+4.225E6 flops
10^2
10^0
10^−2
10^−4
10^−6
10^−8
10^−10
50
100
150
200
250
300

Example


BiCG for −Δ u −256 u_y − 1638.4 u = 0, h=1/128
10^5
10^0
10^−5
10^−10
50
100
150
200
250
300
350
400
450
500
550
without preconditioning: 2.947E8 flops
nofill luinc: 6.805E7+1.12E5 flops
luinc(a,0.1): 2.986E7+4.23E6 flops
luinc(a,0.01): 1.699E7+4.23E6 flops
luinc(a,0.001): 9.451E6+4.225E6 flops

CGS method

One severe drawback of the BiCG method is that in every step one multiplication with A^T is needed, which is not at hand in many cases (in FEM modeling Ax is accumulated from its element proportions, and $A^T x$ is not available) or which can not be realized easily (on parallel computers with distributed memory, e.g.).

The matrix A^T in the BiCG method appears only in the update formulas for $\tilde{r}^k$ and $\tilde{d}^k$, and these vectors are only used in the computation of $\alpha_k = (\tilde{r}^k)^T r^k$ and $\gamma_k = (\tilde{d}^k)^T A d^k$.

Sonneveld (1989) proposed a different way to compute these numbers, and he presented the CGS method (Conjugate Gradient Squared) which is a transposition free variant of the BiCG method, i.e. which avoids multiplications by A^T.

In this method the operator of the BiCG method that reduces the initial residual is applied twice. This may yield a method which is twice as fast as BiCG, but on the other hand it may amplify huge residuals of BiCG twice as large.

CG and orthogonal polynomials

We first consider a different interpretation of the CG method.

Let A be symmetric and positive definite.

From

$$\text{span}\{d^1, \ldots, d^k\} = \text{span}\{r^0, Ar^0, \ldots, A^{k-1}r^0\}$$

it follows that there exists a polynomial $\psi_k \in \Pi_{k-1}$ with

$$d^k = \psi_{k-1}(A)r^0,$$

and $r^k = d^{k+1} - \beta_k d^k$ can be written as

$$r^k = \phi_k(A)r^0, \ \phi_k \in \Pi_k.$$

Orthogonal polynomials

From

$$\begin{aligned} d^{k+1} &= r^k + \beta_k d^k = \phi_k(A)r^0 + \beta_k \psi_{k-1}(A)r^0 \\ r^{k+1} &= r^k - \tau_{k+1} A d^{k+1} = \phi_k(A)r^0 - \tau_{k+1} A \psi_k(A) r^0 \end{aligned}$$

one obtains the recurrences

$$\begin{aligned} \phi_{k+1}(t) &= \phi_k(t) - \tau_{k+1} t \psi_k(t) \\ \psi_k(t) &= \phi_k(t) + \beta_k \psi_{k-1}(t). \end{aligned}$$

The parameters β_k and τ_k can be determined from the polynomials ϕ_k and ψ_k and the bilinear form

$$\langle \phi, \psi \rangle := (\phi(A)r^0)^T \psi(A) r^0.$$

Orthogonal polynomials ct.

$$\langle \phi, \psi \rangle := (\phi(A)r^0)^T \psi(A)r^0$$

is almost a scalar product. It is bilinear and symmetric and nonnegative. Only the positive definiteness is missing, since a polynomial ϕ might exist such that $\phi(A)r^0 = 0$.

Additionally it holds that

$$\langle \phi\psi, \chi \rangle = \langle \phi, \psi\chi \rangle \quad \text{for every polynomial } \phi, \psi, \chi.$$

Rewritten with polynomials the CG algorithm obtains the following form:

Orthogonal polynomials with CG

1: $\phi_0 \equiv 1; \psi_0 \equiv 1$
2: $\alpha_0 = 1$
3: **for** $k = 1, 2, \ldots$ **do**
4: $\quad \gamma_{k-1} = \langle \psi_{k-1}, \iota\psi_{k-1} \rangle$
5: $\quad \tau_k = \alpha_{k-1} / \gamma_{k-1}$
6: $\quad \phi_k = \phi_{k-1} - \tau_k \iota\psi_{k-1}$
7: $\quad \beta_k = 1/\alpha_{k-1}$
8: $\quad \alpha_k = \langle \phi_k, \phi_k \rangle$
9: $\quad \beta_k = \alpha_k \beta_k$
10: $\quad \psi_k = \phi_k + \beta_k \psi_{k-1}$
11: **end for**

where the polynomial ι is defined by $\iota(t) := t$.

CGS method

The orthogonality of the residuals and the conjugacy of the search directions in the CG method yield

$$\langle \phi_j, \phi_k \rangle = \alpha_j \delta_{jk}, \quad \langle \psi_j, \iota\psi_k \rangle = \gamma_k \delta_{jk}$$

for the constructed polynomials ϕ_j and ψ_j.

These equations do not depend on the scalar product $\langle \cdot, \cdot \rangle$ under consideration, but they hold for every symmetric bilinear form $[\,\cdot\,,\,\cdot\,]$ with

$$[\phi\psi, \chi] = [\phi, \psi\chi] \quad \text{for every polynomial } \phi, \psi, \chi.$$

Hence, the CG method can be considered as a method for determining orthogonal polynomials with respect to a given bilinear form $[\cdot, \cdot]$.

CGS ct.

Correspondingly the BiCG method can be considered as a method for determining polynomials $\phi_k, \tilde{\phi}_k \in \Pi_k$ such that $r^k = \phi_k(A)r^0$ and $\tilde{r}^k = \tilde{\phi}_k(A^T)\tilde{r}^0$, and the algorithm indicates that for every k it holds $\phi_k = \tilde{\phi}_k$.

Therefore,

$$r^k = \phi_k(A)r^0, \quad \tilde{r}^k = \phi_k(A^T)\tilde{r}^0.$$

Comparing BiCG with the CG Method it follows that this polynomial is the residual polynomial of the CG method.

Likewise,

$$d^k = \psi_{k-1}(A)r^0, \quad \tilde{d}^k = \psi_{k-1}(A^T)\tilde{r}^0,$$

where ψ_k is obtained from the CG algorithm.

CGS ct.

The parameters α_k and γ_k can be determined from

$$\alpha_k = (\tilde{r}^k)^T r^k = (\phi_k(A^T)\tilde{r}^0)^T \phi_k(A) r^0 = (\tilde{r}^0)^T (\phi_k(A))^2 r^0$$

and

$$\gamma_k = (\tilde{d}^k)^T A d^k = (\psi_{k-1}(A^T)\tilde{r}^0)^T A \psi_{k-1}(A) r^0 = \tilde{r}^0 A (\psi_{k-1}(A))^2 r^0,$$

i.e. without using A^T.

With

$$[\phi, \psi] := (\phi(A^T)\tilde{r}^0)^T \psi(A) r^0$$

α_k and γ_k can be calculated from

$$\alpha_k = [1, \phi_k^2], \quad \gamma_k = [1, \iota\psi_{k-1}^2].$$

CGS ct.

From

$$\phi_k = \phi_{k-1} - \tau_k \psi_{k-1},\ \psi_k = \phi_k + \beta_k \psi_{k-1}$$

we get for

$$\Phi_k := \phi_k^2,\ \Psi_k := \psi_k^2,\ \Xi_k := \phi_k \psi_{k-1},\ \Theta_k := \phi_k \psi_k = \Phi_k + \beta_k \Xi_k :$$

$$\begin{aligned}
\Xi_k &= (\phi_{k-1} - \tau_k \iota \psi_{k-1})\psi_{k-1} = \Theta_{k-1} - \tau_k \iota \Psi_{k-1} \\
\Phi_k &= \Phi_{k-1}^2 - 2\tau_k \iota \Theta_{k-1} + \tau_k^2 \iota^2 \Psi_{k-1} = \Phi_{k-1} - \tau_k \iota (\Theta_{k-1} + \Xi_k) \\
\Theta_k &= \Phi_k + \beta_k \Xi_k \\
\Psi_k &= \Phi_k + 2\beta_k \Xi_k + \beta_k^2 \Psi_{k-1} = \Theta_k + \beta_k (\Xi_k + \beta_k \Psi_{k-1}).
\end{aligned}$$

Hence, the following algorithm allows to compute Φ_k, Ψ_k, Ξ_k and Θ_k, and thus α_k and γ_k, without using A^T.

CGS for orthogonal polynomials

$\Phi_0 \equiv 1; \Psi_0 \equiv 1; \Theta_0 \equiv 1;$
$\alpha_0 = 1$
for $k = 1, 2, \ldots$ **do**
$\gamma_k = [1, \iota\Psi_{k-1}]$
$\tau_k = \alpha_{k-1}/\gamma_k$
$\Xi_k = \Theta_{k-1} - \tau_k \iota \Psi_{k-1}$
$\Phi_k = \Phi_{k-1} - \tau_k \iota(\Theta_{k-1} + \Xi_k)$
$\beta_k = 1/\alpha_{k-1}$
$\alpha_k = [1, \Phi_k]$
$\beta_k = \alpha_k \beta_k$
$\Theta_k = \Phi_k + \beta_k \Xi_k$
$\Psi_k = \Theta_k + \beta_k(\Xi_k + \beta_k \Psi_{k-1})$
end for

CGS method

Substituting A into the polynomials and defining

$$\begin{aligned} r^k &= \phi_k^2(A)r^0 = \Phi_k(A)r^0 \\ d^k &= \psi_k^2(A)r^0 = \Psi_{k-1}(A)r^0 \\ u^k &= \phi_k(A)\psi_k(A)r^0 = \Theta_k(A)r^0 \\ q^k &= \phi_k(A)\psi_{k-1}(A)r^0 = \Xi_k(A)r^0 \end{aligned}$$

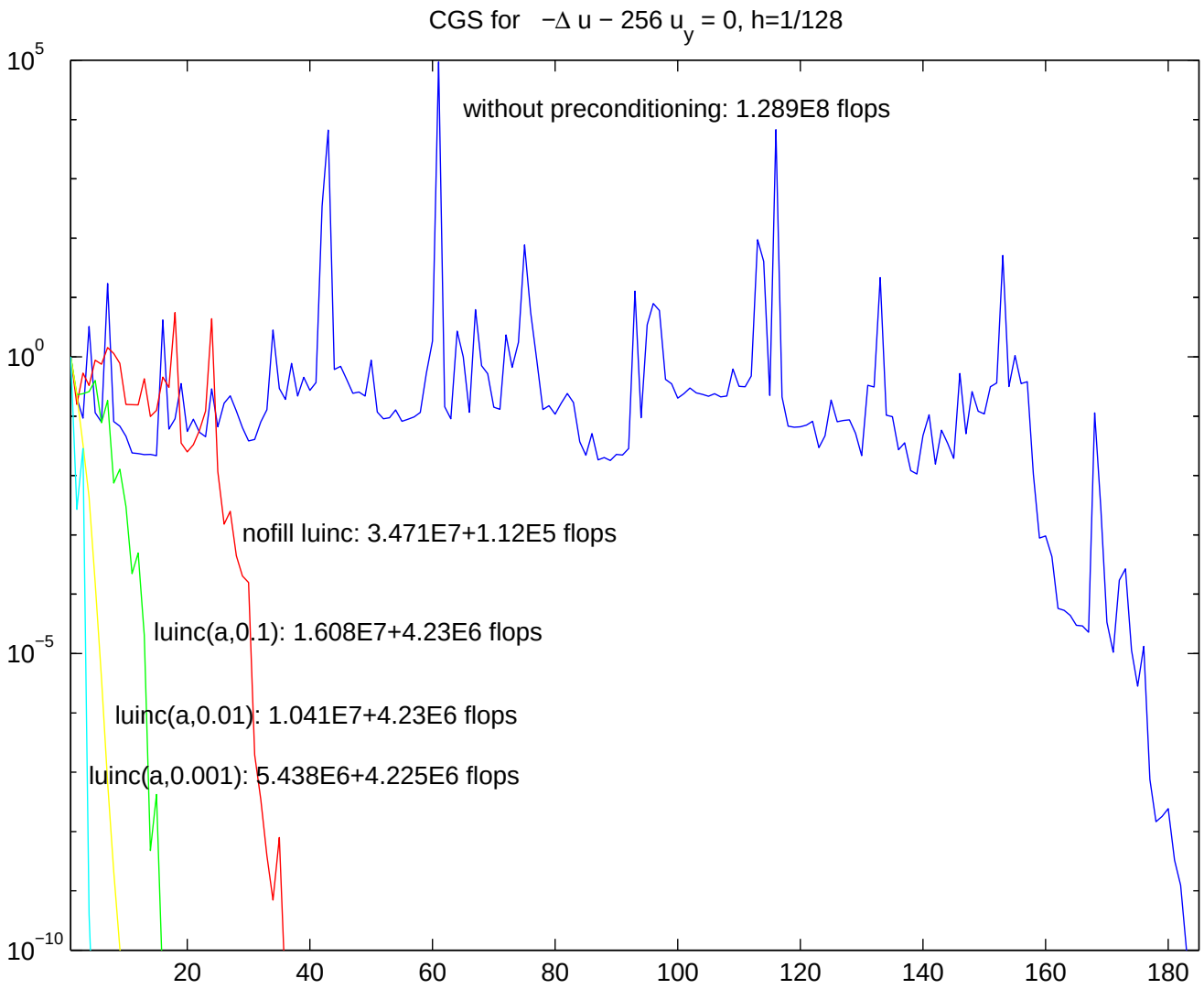
we get:

CGS Sonneveld 1989

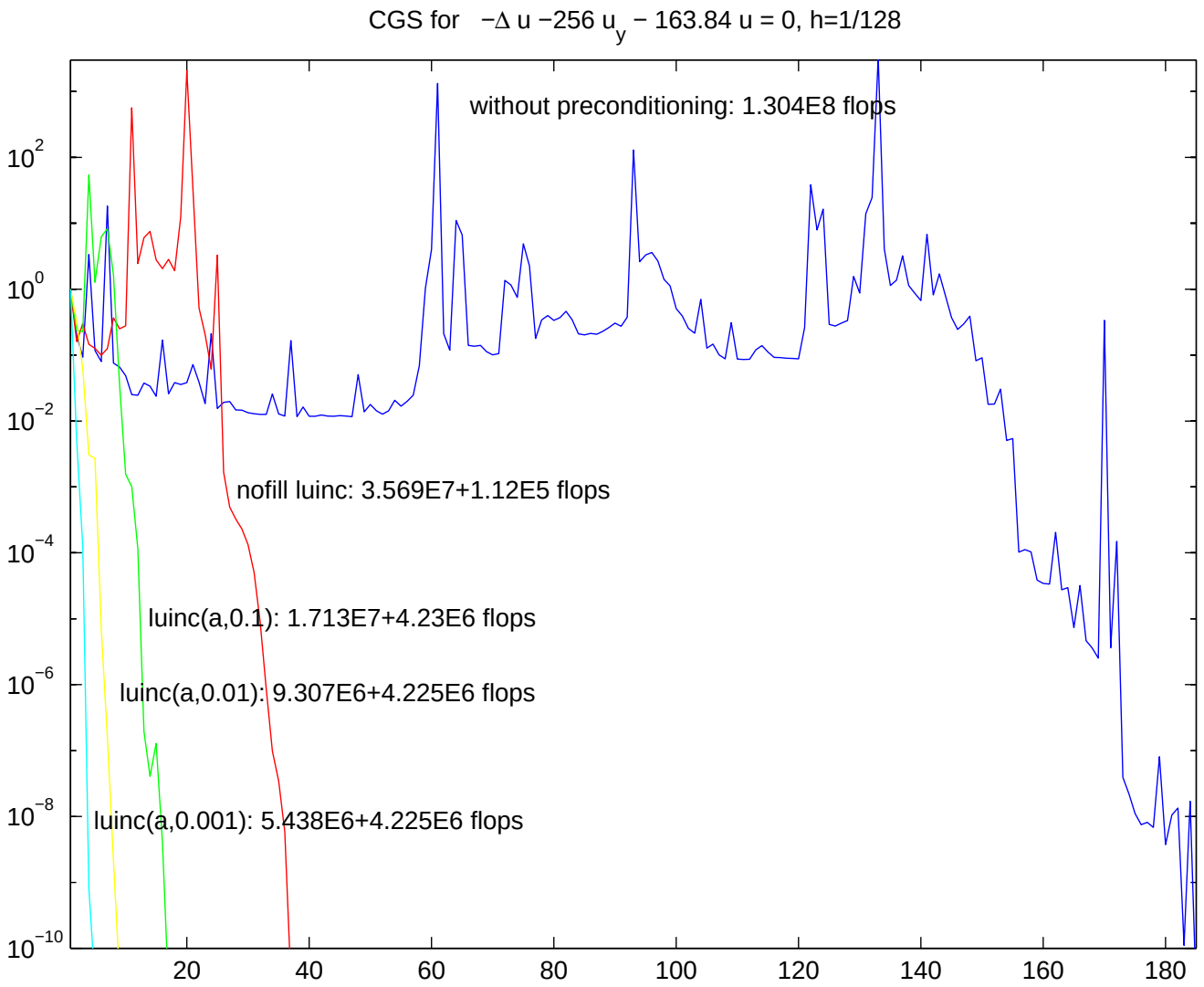
$r^0 = b - Ax^0$; $d^1 = r^0$; $u^0 = r^0$
Choose $\tilde{r}^0$ with $\alpha_0 = (\tilde{r}^0)^T r^0 \neq 0$
for $k = 1, 2, \ldots$ until convergence **do**
  $s^k = Ad^k$
  $\gamma_k = (\tilde{r}^0)^T s^k$
  $\tau_k = \alpha_{k-1}/\gamma_k$
  $q^k = u^{k-1} - \tau_k s^k$
  $w^k = u^{k-1} + q^k$
  $x^k = x^{k-1} + \tau_k w^k$
  $r^k = r^{k-1} - \tau_k A w^k$
  $\beta_k = 1/\alpha_{k-1}$; $\alpha_k = (\tilde{r}^0)^T r^k$; $\beta_k = \alpha_k \beta_k$
  $u^k = r^k + \beta_k q^k$
  $d^{k+1} = u^k + \beta_k (q^k + \beta_k d^k)$
end for

Cost of CGS: 2 matrix-vector products (not with A^T)
2 scalar products
7 _axpy
Storage requirements: 8 vectors

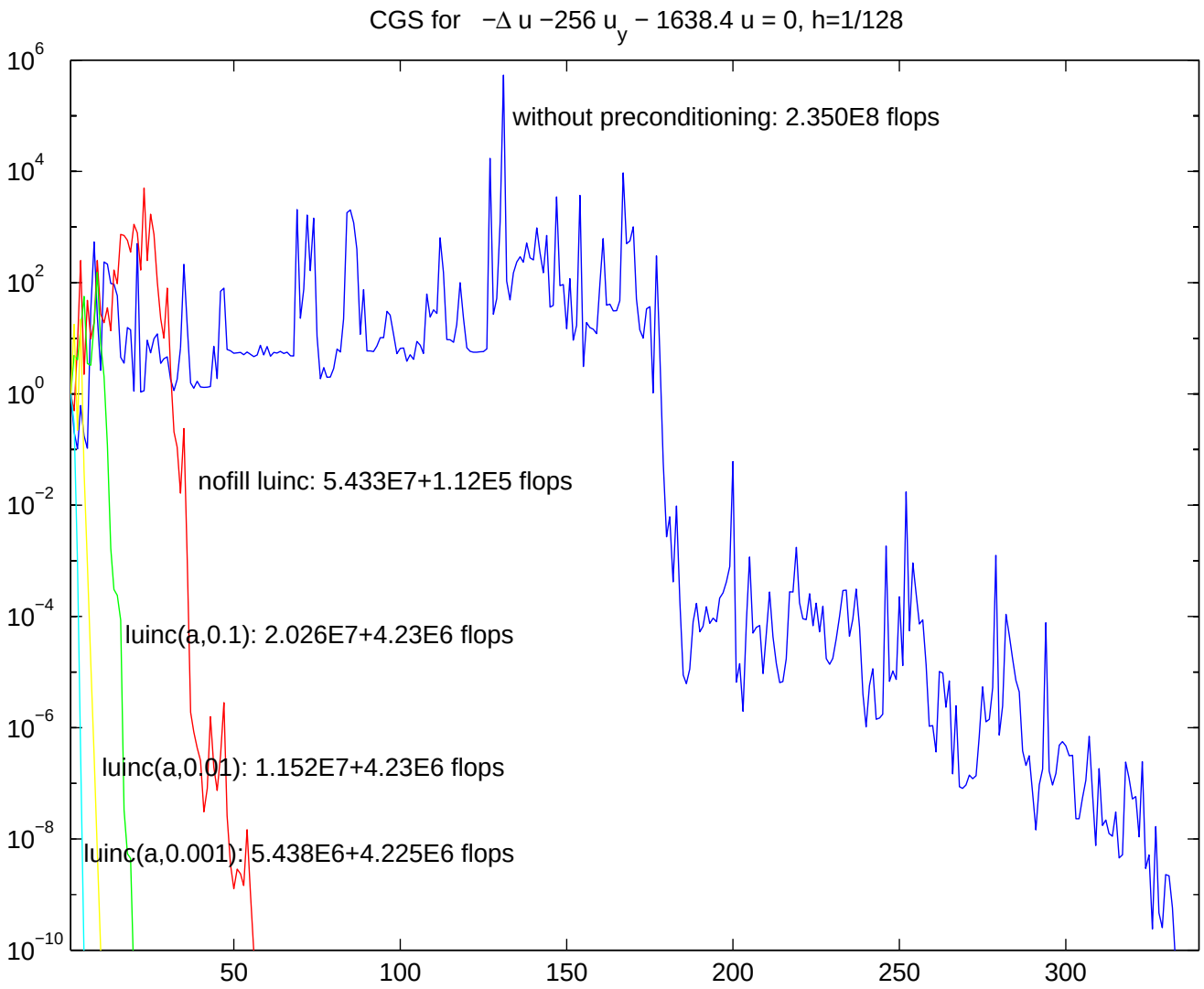
Example


CGS for −Δ u − 256 u_y = 0, h=1/128
without preconditioning: 1.289E8 flops
nofill luinc: 3.471E7+1.12E5 flops
luinc(a,0.1): 1.608E7+4.23E6 flops
luinc(a,0.01): 1.041E7+4.23E6 flops
luinc(a,0.001): 5.438E6+4.225E6 flops
10^5
10^0
10^−5
10^−10
20
40
60
80
100
120
140
160
180

Example


CGS for −Δ u −256 u_y − 163.84 u = 0, h=1/128
without preconditioning: 1.304E8 flops
nofill luinc: 3.569E7+1.12E5 flops
luinc(a,0.1): 1.713E7+4.23E6 flops
luinc(a,0.01): 9.307E6+4.225E6 flops
luinc(a,0.001): 5.438E6+4.225E6 flops
10^2
10^0
10^-2
10^-4
10^-6
10^-8
10^-10
20
40
60
80
100
120
140
160
180

Example

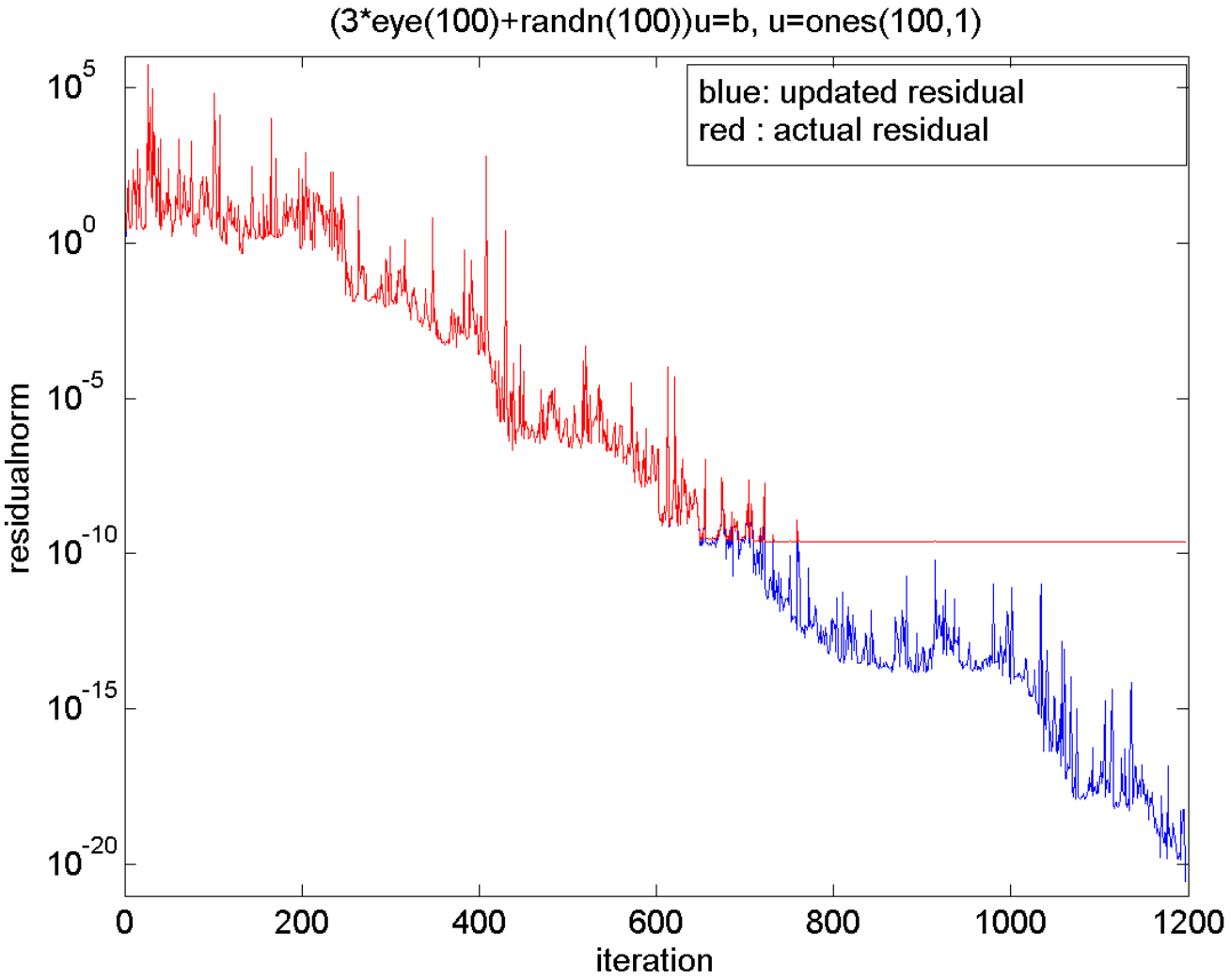

CGS for −Δ u −256 u_y − 1638.4 u = 0, h=1/128
without preconditioning: 2.350E8 flops
nofill luinc: 5.433E7+1.12E5 flops
luinc(a,0.1): 2.026E7+4.23E6 flops
luinc(a,0.01): 1.152E7+4.23E6 flops
luinc(a,0.001): 5.438E6+4.225E6 flops
10^6
10^4
10^2
10^0
10^−2
10^−4
10^−6
10^−8
10^−10
50
100
150
200
250
300

Residual update

Unfortunately, the true and the updated residual often drift apart during the iteration process, and this leads to misleading impressions of the actual error. When the method produces a small updated residual r^k, then the true residual $b - Ax^k$ may not be small at all.

It may also be the case that we have done to many iteration steps, since the actual errors tend to stagnate in many cases where the updated residuals still further decrease beyond a certain value.

Example

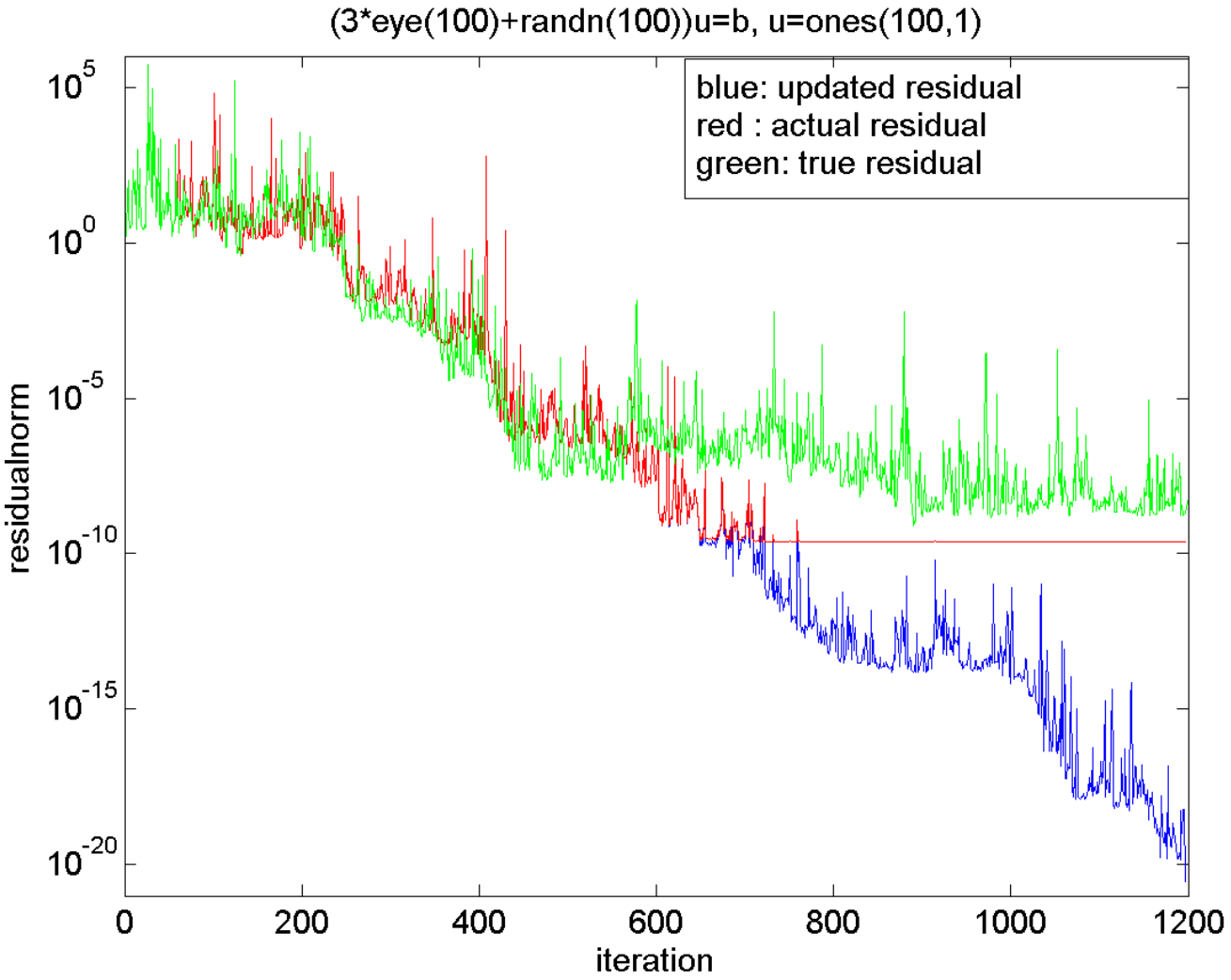

(3*eye(100)+randn(100))u=b, u=ones(100,1)
blue: updated residual
red : actual residual
residualnorm
iteration
10^5
10^0
10^-5
10^-10
10^-15
10^-20
0
200
400
600
800
1000
1200

Residual update ct.

The vector Aw^k is only used to update the residuum r^k. At the same expense one can determine the residuum $r^k = b - Ax^k$ directly in CGS. (For other methods the computation of the true residual would require an additional matrix-vector product)

Van der Vorst (1992) reports that in many examples it has been observed that the direct calculation of the residuum has a negative effect on the iteration process, i.e. it slows down the convergence of CGS.

Example


(3*eye(100)+randn(100))u=b, u=ones(100,1)
blue: updated residual
red : actual residual
green: true residual
residualnorm
iteration
10^5
10^0
10^-5
10^-10
10^-15
10^-20
0
200
400
600
800
1000
1200

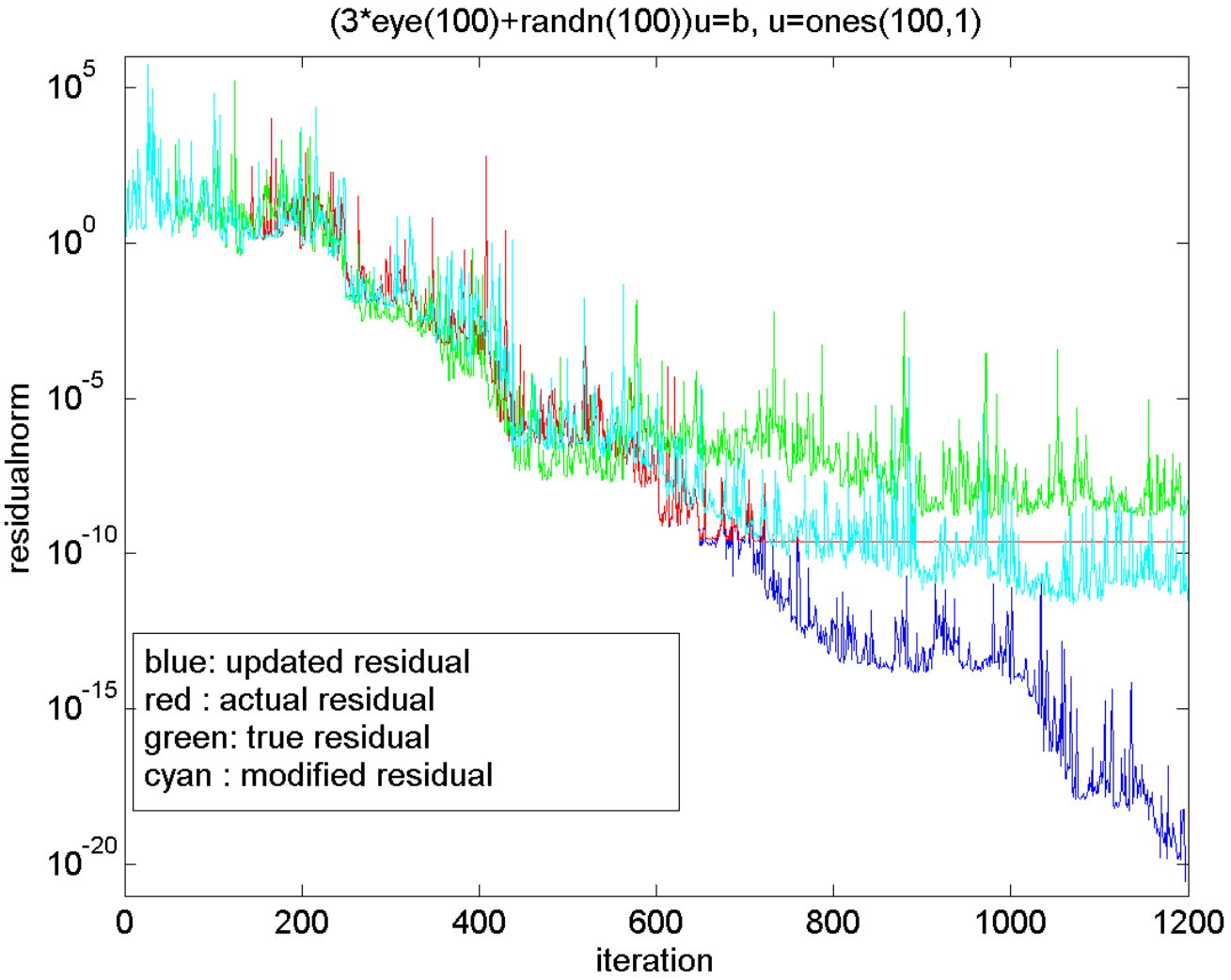
Residual update ct.

Among other modified strategies for determining the residual Sleijpen & van der Vorst (1996) suggested to replace the updated residual only by the true one, if the updated residual becomes smaller than the previous residual norms.

The CGS algorithm is modified in the following way: Two new variables are initialized as $xx = 0$ and $r_{\min} = norm(r^0)$.
Statement 9 is replaced by $xx = xx + \tau_k w^k$, and the following if-clause is inserted after statement 10:

if $\|r^k\| < r_{\min}$ **then**
  $x = x + xx$;
  $xx = 0$
  $r^k = b - Ax$
  $r_{\min} = \|r^k\|$
end if

Example


(3*eye(100)+randn(100))u=b, u=ones(100,1)
residualnorm
iteration
blue: updated residual
red : actual residual
green: true residual
cyan : modified residual

Observation

The BiCG method often shows a very erratic convergence behavior with wildly oscillating residual norms $\|r^k_{BiCG}\|_2 = \|\phi_k(A)r^0\|_2$, and for $\|r^k_{CGS}\|_2 = \|(\phi_k(A))^2 r^0\|_2$ this behavior is even worse.

The polynomials ϕ_k are characterized by $\phi_k(0) = 1$ and

$$(\phi_j(A^T)\tilde{r}^0)^T \phi_k(A)r^0 = 0,\ j = 0, \ldots, k-1,$$

i.e. by the fact that $\phi_k(A)r^0$ is orthogonal to the Krylov space $\mathcal{K}_k(\tilde{r}^0, A^T)$.

Equivalent is the requirement that $\phi_k(A)r^0$ is orthogonal to $\tilde{\phi}_j(A^T)\tilde{r}^0$ for suitably chosen polynomials $\tilde{\phi}_j$ of degree j for $j = 0, \ldots, k-1$.

BiCGStab

Van der Vorst (1992) proposed the choice

$$\tilde{\phi}_k(t) = \prod_{j=1}^{k}(1-\omega_j t) = (1-\omega_k t)\tilde{\phi}_{k-1}(t),$$

where ω_k is determined such that the residual norm $\|r^k\|_2$ is minimal.

The algorithm BiCGStab results which shows a much smoother convergence behavior as BiCG or CGS.

The method is deduced in a similar way as CGS.

BiCGStab ct.

For

$$\Phi_j := \tilde{\phi}_j \phi_j, \ \Psi_j := \tilde{\phi}_j \psi_j$$

it holds

$$\begin{aligned}
\Phi_k &= \tilde{\phi}_k \phi_k = (1 - \omega_k \iota) \tilde{\phi}_{k-1} (\phi_{k-1} - \tau_k \iota \psi_{k-1}) \\
&= \Phi_{k-1} - \tau_k \iota \Psi_{k-1} - \omega_k \iota (\Phi_{k-1} - \tau_k \iota \Psi_{k-1}), \\
\Psi_k &= \tilde{\phi}_k (\phi_k + \beta_k \psi_{k-1}) = \tilde{\phi}_k \phi_k + \beta_k (1 - \omega_k \iota) \tilde{\phi}_{k-1} \psi_{k-1} \\
&= \Phi_k + \beta_k \Psi_{k-1} - \beta_k \omega_k \iota \Psi_{k-1}
\end{aligned}$$

To execute these recurrences, we have to represent β_k and τ_k of the BiCG method as scalar products of the polynomials Φ_j and Ψ_j.

Let

$$\tilde{\alpha}_k := \langle 1, \Phi_k \rangle = \langle \tilde{\phi}_k, \phi_k \rangle.$$

BiCGStab ct.

From

$$\tilde{\phi}_k = (-1)^k \Big(\prod_{j=1}^{k} \omega_j \Big) \iota^k + \hat{\phi}_{k-1}, \quad \hat{\phi}_{k-1} \in \Pi_{k-1},$$

and the orthogonality of ϕ_k and Π_{k-1} we get

$$\tilde{\alpha}_k = (-1)^k \prod_{j=1}^{k} \omega_j \langle \iota^k, \phi_k \rangle,$$

and from

$$\phi_k = (-1)^k \prod_{j=1}^{k} \tau_j \iota^k + \bar{\phi}_{k-1}, \quad \bar{\phi}_{k-1} \in \Pi_{k-1},$$

it follows

$$\alpha_k = \langle \phi_k, \phi_k \rangle = (-1)^k \prod_{j=1}^{k} \tau_j \langle \iota^k, \phi_k \rangle = \prod_{j=1}^{k} \frac{\tau_j}{\omega_j} \tilde{\alpha}_k,$$

BiCGStab ct.

$$\alpha_k = \prod_{j=1}^{k} \frac{\tau_j}{\omega_j} \tilde{\alpha}_k \quad \Longrightarrow \quad \beta_k = \frac{\alpha_k}{\alpha_{k-1}} = \frac{\tilde{\alpha}_k}{\tilde{\alpha}_{k-1}} \cdot \frac{\tau_k}{\omega_k}.$$

Similarly from $\langle \chi, \iota\psi_{k-1} \rangle = 0$ for every $\chi \in \Pi_{k-2}$ one gets

$$\tilde{\gamma}_k := \langle 1, \iota\Psi_{k-1} \rangle = \langle \tilde{\phi}_{k-1}, \iota\psi_{k-1} \rangle = (-1)^{k-1} \prod_{j=1}^{k-1} \omega_j \langle \iota^k, \iota\psi_{k-1} \rangle$$

$$\gamma_k = \langle \psi_{k-1}, \iota\psi_{k-1} \rangle = (-1)^{k-1} \prod_{j=1}^{k-1} \tau_j \langle \iota^{k-1}, \iota\psi_{k-1} \rangle = \prod_{j=1}^{k-1} \frac{\tau_j}{\omega_j} \tilde{\gamma}_j,$$

and therefore

$$\tau_k = \frac{\alpha_{k-1}}{\gamma_k} = \frac{\tilde{\alpha}_{k-1}}{\tilde{\gamma}_k}.$$

BiCGStab ct.

Finally

$$\begin{aligned} r^k &= \Phi_k(A)r^0 \\ &= (I-\omega_k A)\tilde{\phi}_{k-1}(A)(\phi_{k-1}(A)-\tau_k A\psi_{k-1}(A))r^0 \\ &= (I-\omega_k A)(\Phi_{k-1}(A)-\tau_k A\Psi_{k-1}(A))r^0 \\ &=: (I-\omega_k A)u^k \end{aligned}$$

yields that $\|r^k\|_2$ is minimal if and only if

$$\omega_k = \frac{(u^k)^T A u^k}{\|Au^k\|_2^2}.$$

BiCGStab: Van der Vorst 1992

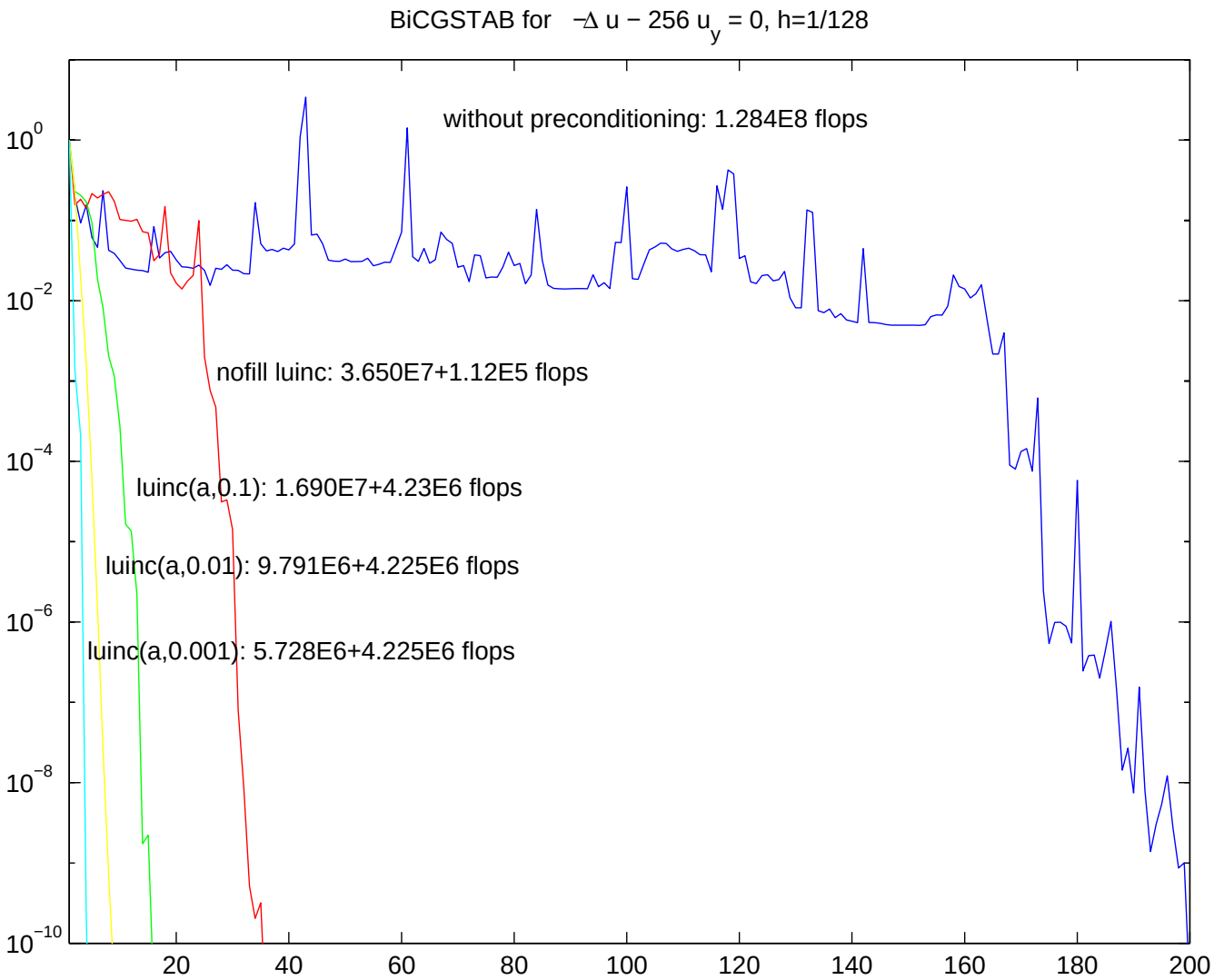
$r^0 = b - Ax^0$; $d^1 = r^0$
Choose $\tilde{r}^0$ with $\tilde{\alpha}_0 = (\tilde{r}^0)^T r^0 \neq 0$
for $k = 1, 2, \ldots$ until convergence **do**
    $s^k = Ad^k$
    $\tilde{\gamma}_k = (\tilde{r}^0)^T s^k$
    $\tau_k = \tilde{\alpha}_{k-1} / \tilde{\gamma}_k$
    $u^k = r^{k-1} - \tau_k s^k$
    $t^k = Au^k$
    $\omega_k = (t^k)^T u^k / \|t^k\|_2^2$
    $x^k = x^{k-1} + \tau_k d^k + \omega_k u^k$
    $r^k = u^k - \omega_k t^k$
    $\beta_k = 1/\tilde{\alpha}_{k-1}$;
    $\tilde{\alpha}_k = (\tilde{r}^0)^T r^k$;
    $\beta_k = \tilde{\alpha}_k \beta_k \tau_k / \omega_k$
    $d^{k+1} = r^k + \beta_k (d^k - \omega_k s^k)$;
end for

Cost of BiCGStab

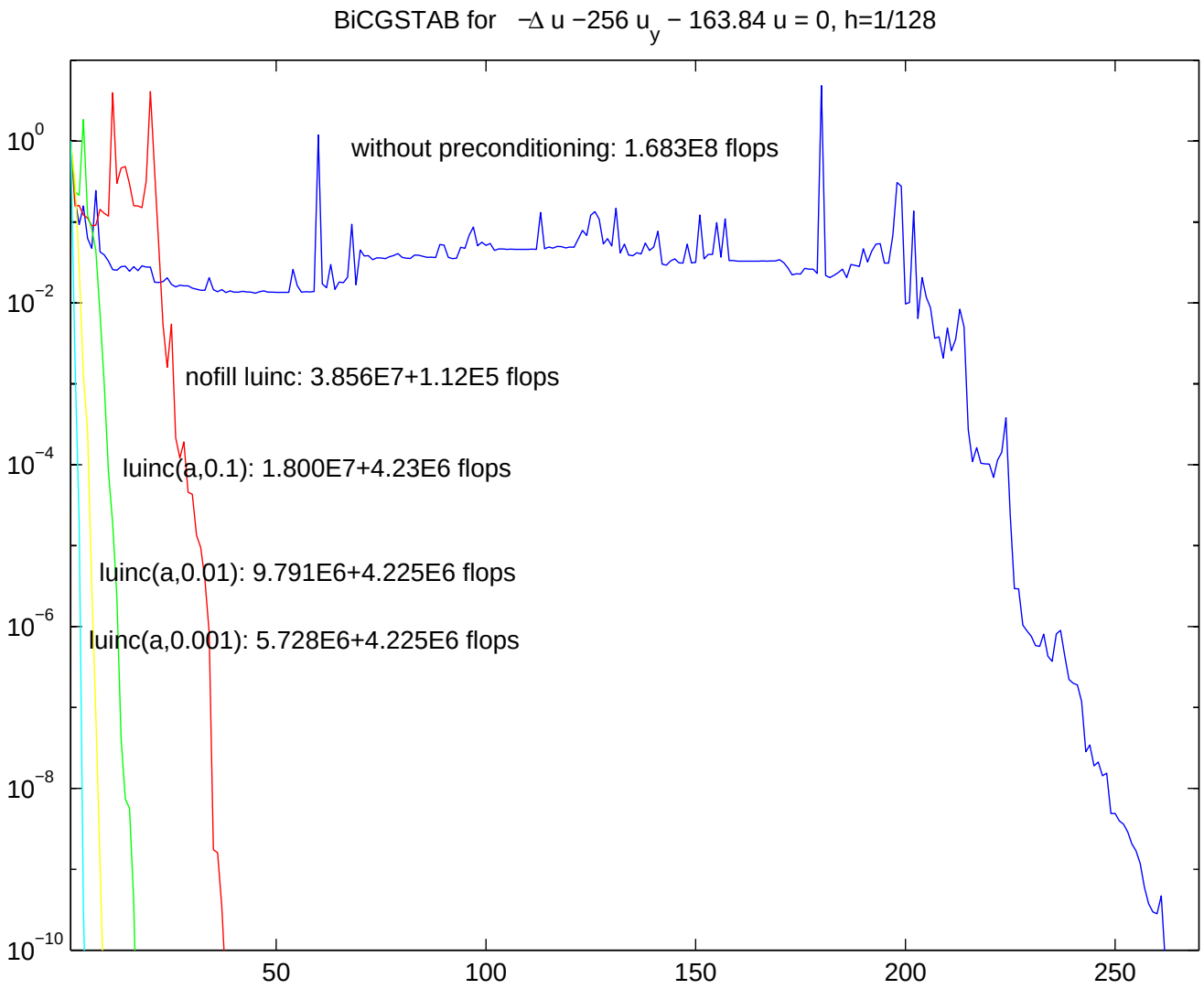
2 matrix-vector products (not with A^T)
4 scalar products
6 _axpy

Storage requirements: 8 vectors

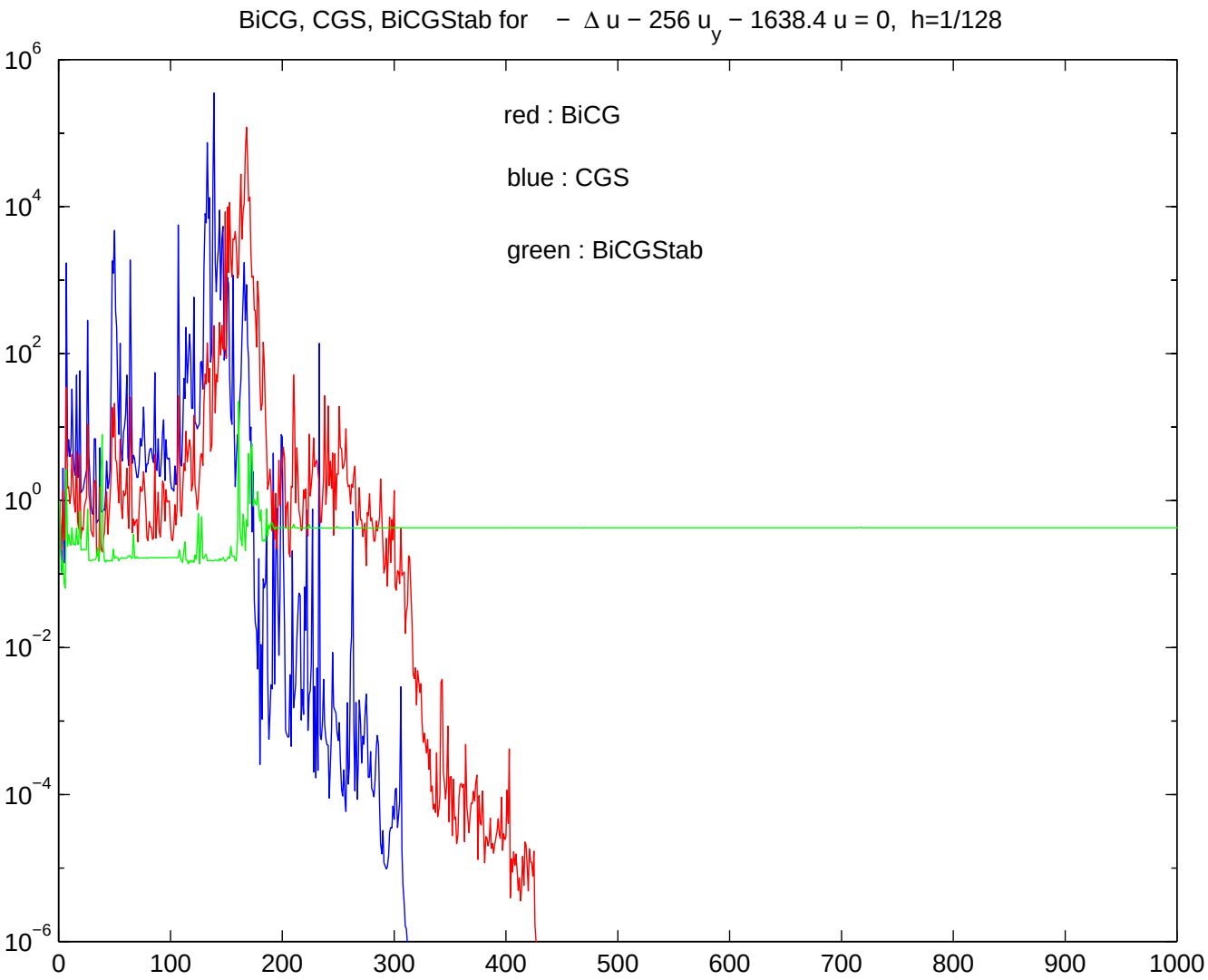
Example


BiCGSTAB for $-\Delta u - 256\, u_y = 0$, h=1/128
without preconditioning: 1.284E8 flops
nofill luinc: 3.650E7+1.12E5 flops
luinc(a,0.1): 1.690E7+4.23E6 flops
luinc(a,0.01): 9.791E6+4.225E6 flops
luinc(a,0.001): 5.728E6+4.225E6 flops
10^{0}
10^{-2}
10^{-4}
10^{-6}
10^{-8}
10^{-10}
20
40
60
80
100
120
140
160
180
200

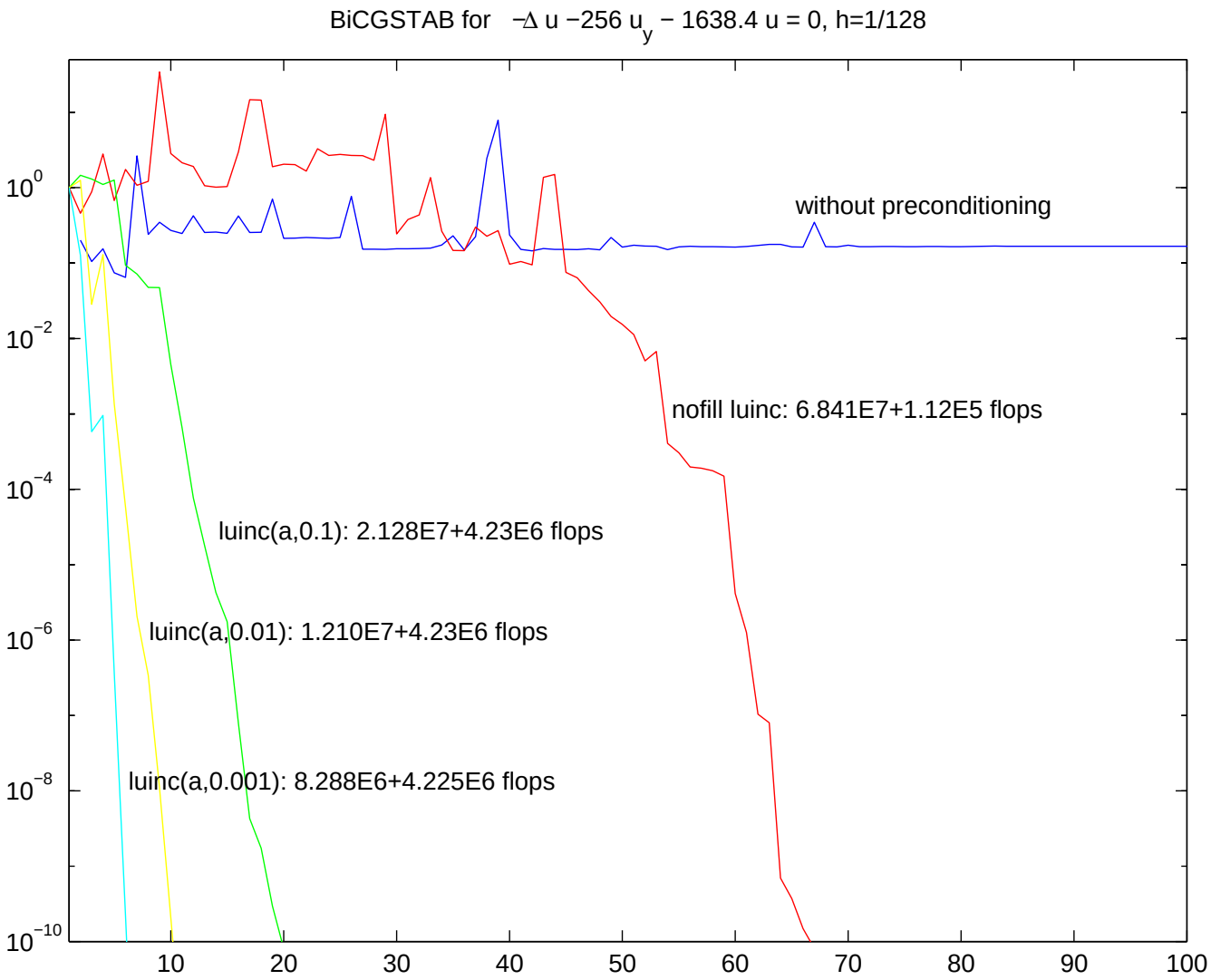
Example


BiCGSTAB for $-\Delta u - 256\, u_y - 163.84\, u = 0$, h=1/128
without preconditioning: 1.683E8 flops
nofill luinc: 3.856E7+1.12E5 flops
luinc(a,0.1): 1.800E7+4.23E6 flops
luinc(a,0.01): 9.791E6+4.225E6 flops
luinc(a,0.001): 5.728E6+4.225E6 flops
10^{0}
10^{-2}
10^{-4}
10^{-6}
10^{-8}
10^{-10}
50
100
150
200
250

Example


BiCG, CGS, BiCGStab for − Δ u − 256 u_y − 1638.4 u = 0, h=1/128
red : BiCG
blue : CGS
green : BiCGStab
10^6
10^4
10^2
10^0
10^−2
10^−4
10^−6
0
100
200
300
400
500
600
700
800
900
1000

Example


BiCGSTAB for −Δ u −256 u_y − 1638.4 u = 0, h=1/128
10^0
10^−2
10^−4
10^−6
10^−8
10^−10
10
20
30
40
50
60
70
80
90
100
without preconditioning
nofill luinc: 6.841E7+1.12E5 flops
luinc(a,0.1): 2.128E7+4.23E6 flops
luinc(a,0.01): 1.210E7+4.23E6 flops
luinc(a,0.001): 8.288E6+4.225E6 flops

Stagnation

BiCGStab steps can be viewed as a combination of a BiCG step and a GMRES(1) step. As soon as the GMRES(1) part (nearly) stagnates this behavior can often not be remedied by the following BiCG step.

Another inconvenient aspect of BiCGStab is the fact that the polynomials ψ_k by construction only have real eigenvalues, whereas optimal reduction polynomials for matrices with nonreal eigenvalues often have nonreal roots.

Gutknecht (1993) proposed BiCGStab2 where in odd-numbered iteration steps the polynomial ψ is expanded by a linear factor, but in the following even-numbered step this linear factor is discarded, and the polynomial ψ from the preceding even-numbered step is expanded by a quadratic factor. This approach improved BiCGStab in examples with nonreal eigenvalues.

A possible weakness in Gutknecht's approach may be that in odd-numbered steps the same problem may occur as in BiCGStab, and since the following even-numbered step relies on this result, this may lead to poor convergence.

Stagnation ct.

Sleijpen & Fokkema (1993) suggested the following (even simpler) approach called BiCGStab(2) which avoids the intermediate BiCGStab-type steps.

Having applied two BiCG steps, the polynomial ψ is expanded directly by a quadratic factor without constructing the linear factors.

The method leads only in the even-numbered steps to significant residuals, whereas it does not generate useful approximations in the odd-numbered steps.

Stagnation ct.

More generally the polynomial ψ can be constructed as a product of polynomials of degree $\ell \geq 2$ resulting in the BiCGStab(ℓ) method which works as follows:

In the k-th step, $k = m\ell + \ell$, $m = 0, 1, \ldots$ the polynomial ψ_k is chosen as $\psi_k = \chi_m \psi_{k-\ell}$, where $\chi_m \in \Pi_\ell$ is determined such that $\chi_m(0) = 1$ and

$$\|r^k\|_2 = \|\chi(A)\psi_{k-\ell}(A)\phi_k(A)r^0\|_2, \ \chi \in \Pi_\ell, \ \chi(0) = 1,$$

is minimal. The minimum norm solution is obtained through solving the associated normal equations (i.e. solving a linear system of dimension ℓ).

The vectors $r^{m\ell+i}$, $i = 1, \ldots, \ell - 1$, are constructed by (transpose free) BiCG steps, where $\psi_{m\ell+i}$ is chosen as $\psi_{m\ell+i} = \iota^i \psi_{m\ell}$.

A BiCGStab(ℓ) implementation in MATLAB and FORTRAN is available from

http://www.math.uu.nl/people/sleijpen

BiCGStab(2): Sleijpen, Fokkema 1992

$r^0 = b - Ax^0$; Choose $\tilde{r}^0$ with $\tilde{\alpha}_0 = (\tilde{r}^0)^T r^0 \neq 0$
$\rho_0 = 1$; $u = 0$; $\alpha = 0$; $\omega_2 = 1$;
for $k = 0, 2, 4, \ldots$ until convergence **do**
    $\rho_0 = -\omega_2 \rho_0$;
    $\rho_1 = (\tilde{r}^0)^T r^i$; $\beta = \alpha \rho_1 / \rho_0$; $\rho_0 = \rho_1$;
    $u = r^i - \beta u$; $v = Au$;
    $\gamma = v^T \tilde{r}^0$; $\alpha = \rho / \gamma$;
    $r = r^i - \alpha v$; $s = Ar$; $x = x^i + \alpha u$;
    $\rho_1 = (\tilde{r}^0)^T s$; $\beta = \alpha \rho_1 / \rho_0$; $\rho_0 = \rho_1$;
    $v = s - \beta v$; $w = Av$;
    $\gamma = w^T \tilde{r}^0$; $\alpha = \rho / \gamma$;
    $u = r - \beta u$; $r = r - \alpha v$; $s = s - \alpha w$; $t = As$;
    $\omega_1 = r^T s$; $\mu = s^T s$; $\nu = s^T t$; $\tau = t^T t$; $\omega_2 = r^T t$; $\tau = \tau - \nu^2 / \mu$;
    $\omega_2 = (\omega_2 - \nu \omega_1 / \mu) / \tau$; $\omega_1 = (\omega_1 - \nu \omega_2) / \mu$;
    $x^{i+2} = x + \omega_1 r + \omega_2 s \alpha u$;
    $r^{i+2} = r - \omega_1 s - \omega_2 t$;
    if x^{i+2} accurate enough **then** quit
    $u = u - \omega_1 v - \omega_2 w$;
end for

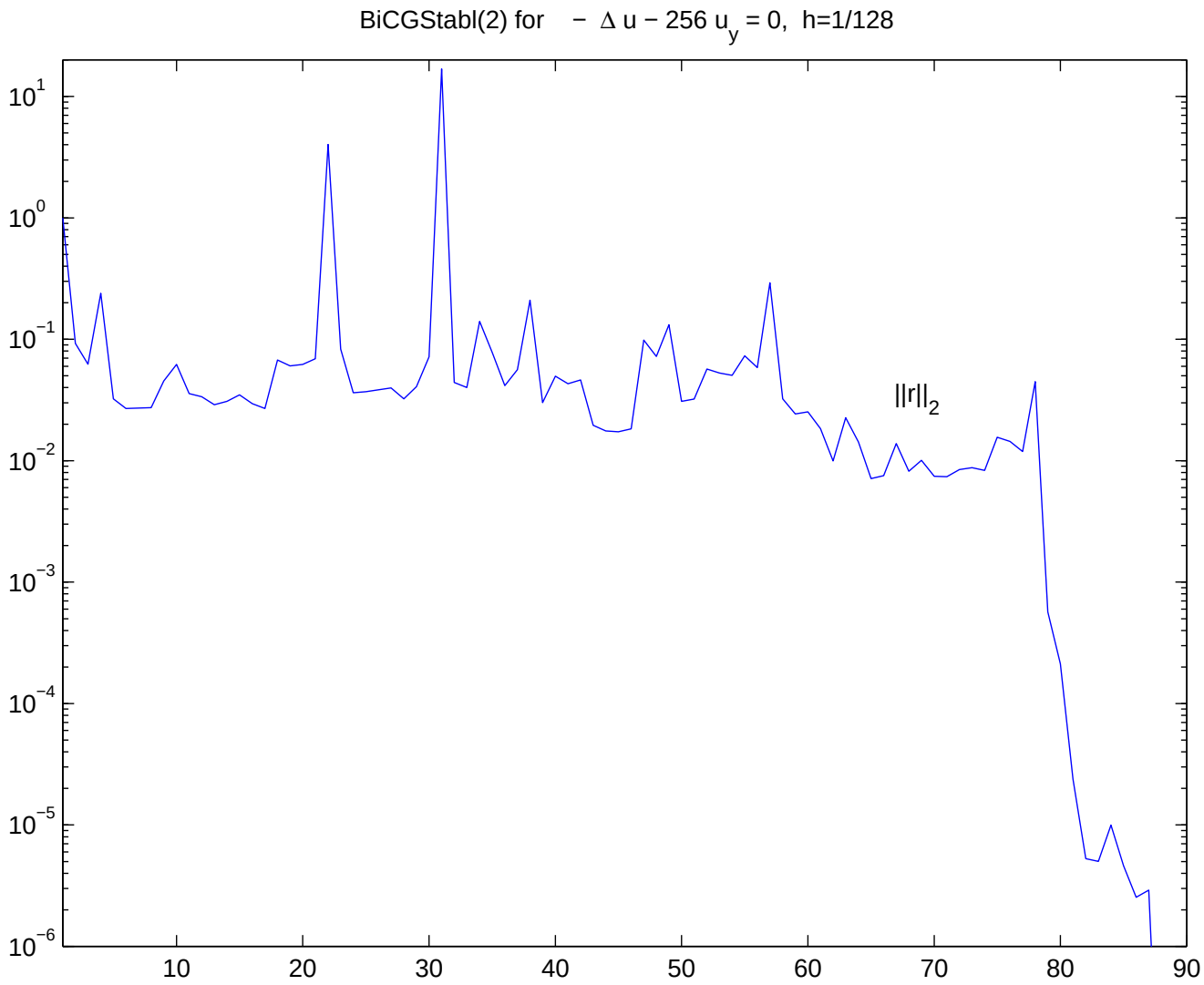
Comments on BiCGStab(2)

The even BiCG step is performed in steps 5-8, the odd BiCG step in steps 9-12, and steps 13-16 correspond to the GMRES(2) part.

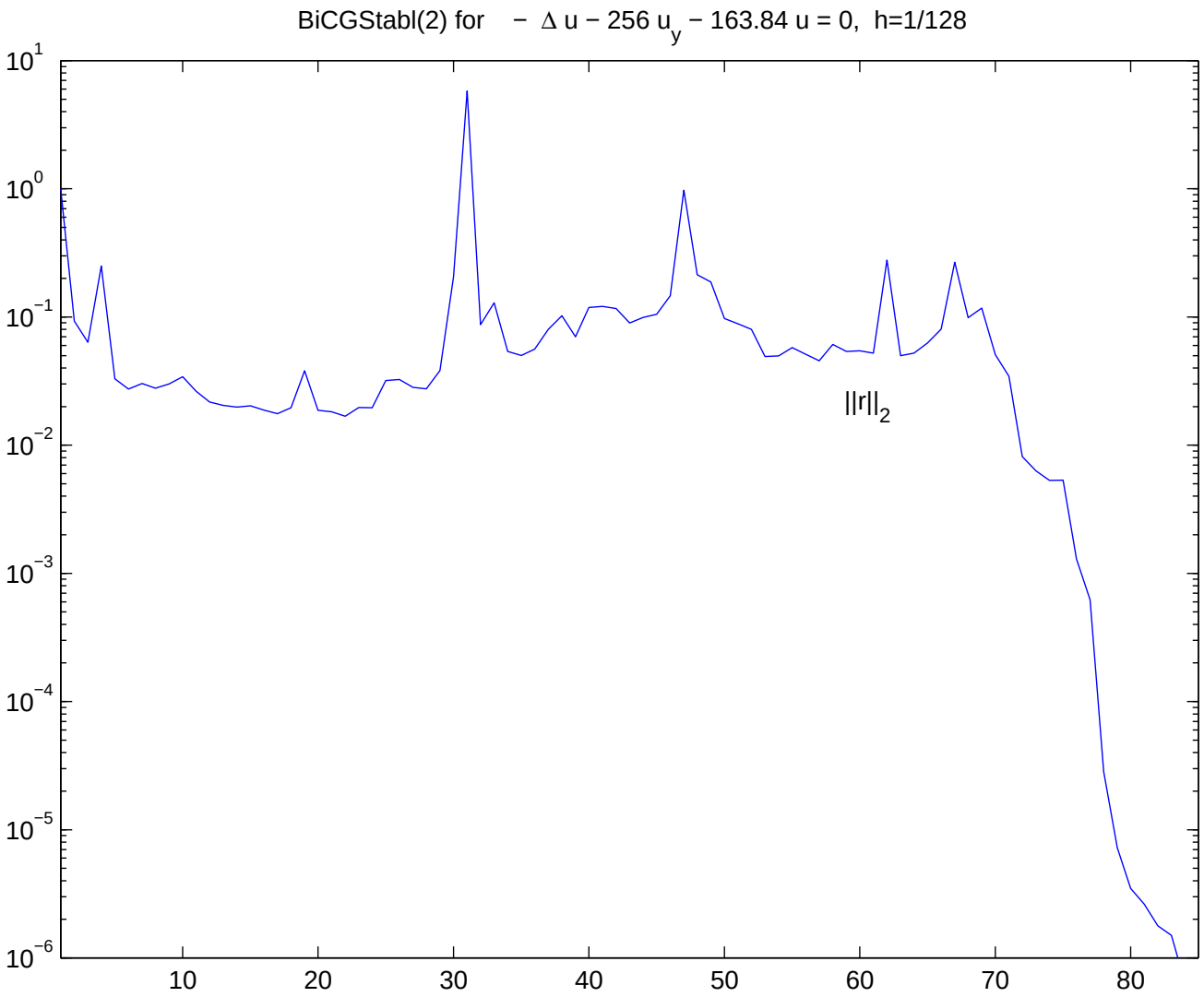
The BiCGStab(2) algorithm requires 14 _axpy, 9 dot products, and 4 matrix-vector products per full cycle, and is only a little more expensive than two steps of BiCGStab, which requires 12 _axpy, 8 dot products, and 4 matrix-vector multiplications.

A further variant of BiCG called GPBi-CG (generalized product BiCG) proposed by Zhang (1997) generates the ψ polynomials by a three term recurrence relation. For a special choice of a parameter it reduces to BiCGStab(ℓ).

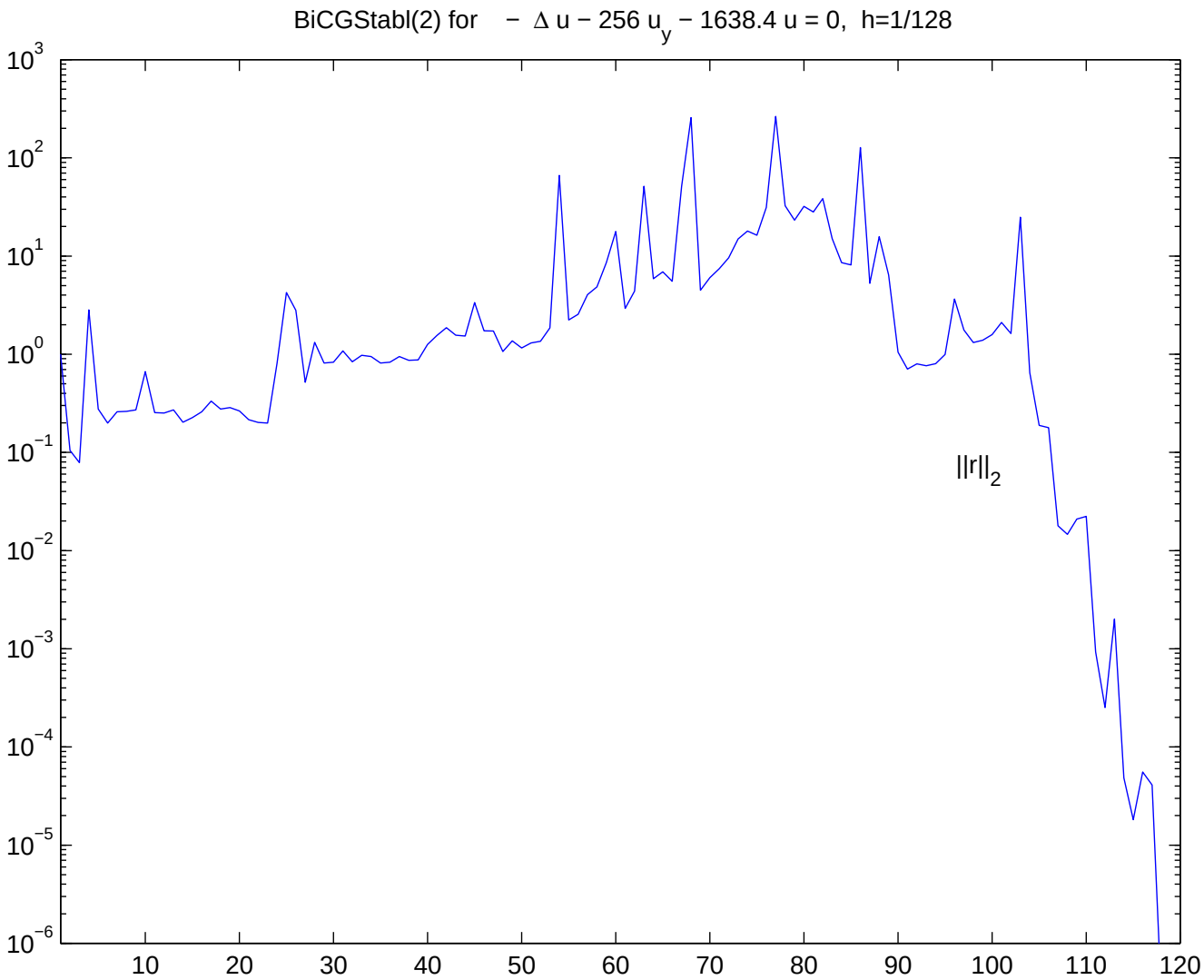
Example


BiCGStabl(2) for − Δ u − 256 u_y = 0, h=1/128
10^1
10^0
10^−1
10^−2
10^−3
10^−4
10^−5
10^−6
10
20
30
40
50
60
70
80
90
||r||_2

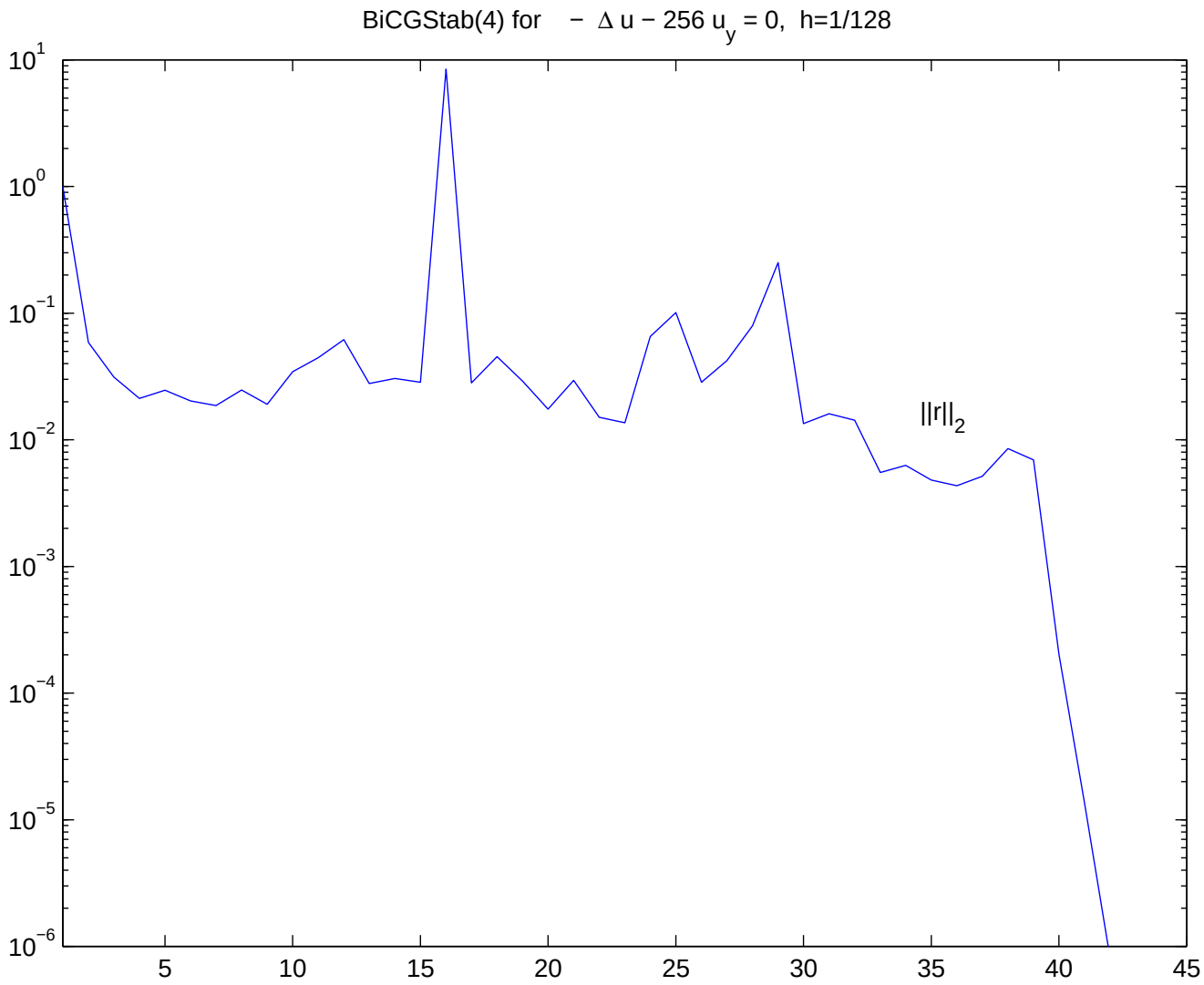
Example


BiCGStabl(2) for − Δ u − 256 u_y − 163.84 u = 0, h=1/128
10^1
10^0
10^−1
10^−2
10^−3
10^−4
10^−5
10^−6
10
20
30
40
50
60
70
80
||r||_2

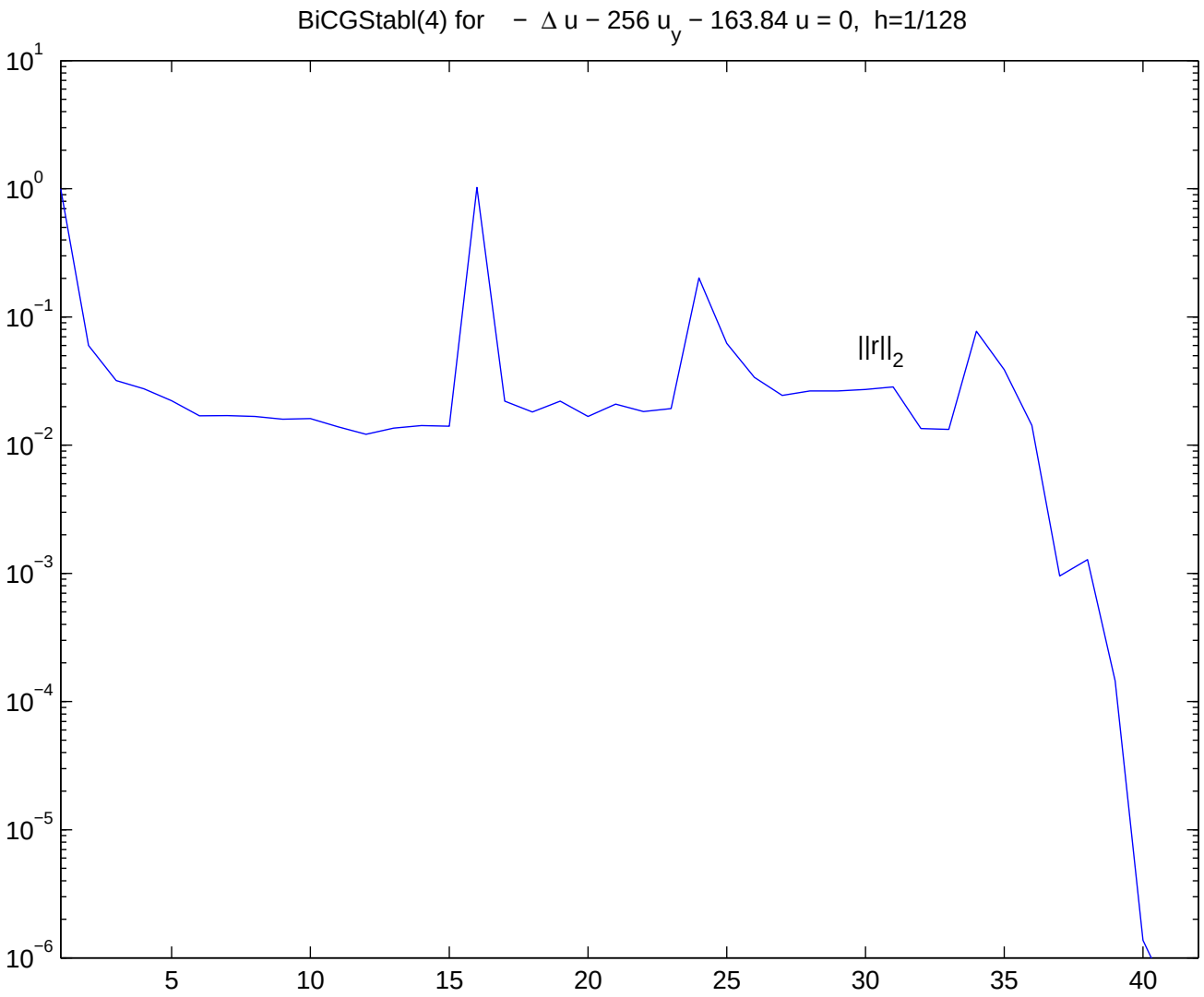
Example


BiCGStabl(2) for − Δ u − 256 u_y − 1638.4 u = 0, h=1/128
10^3
10^2
10^1
10^0
10^−1
10^−2
10^−3
10^−4
10^−5
10^−6
10
20
30
40
50
60
70
80
90
100
110
120
||r||_2

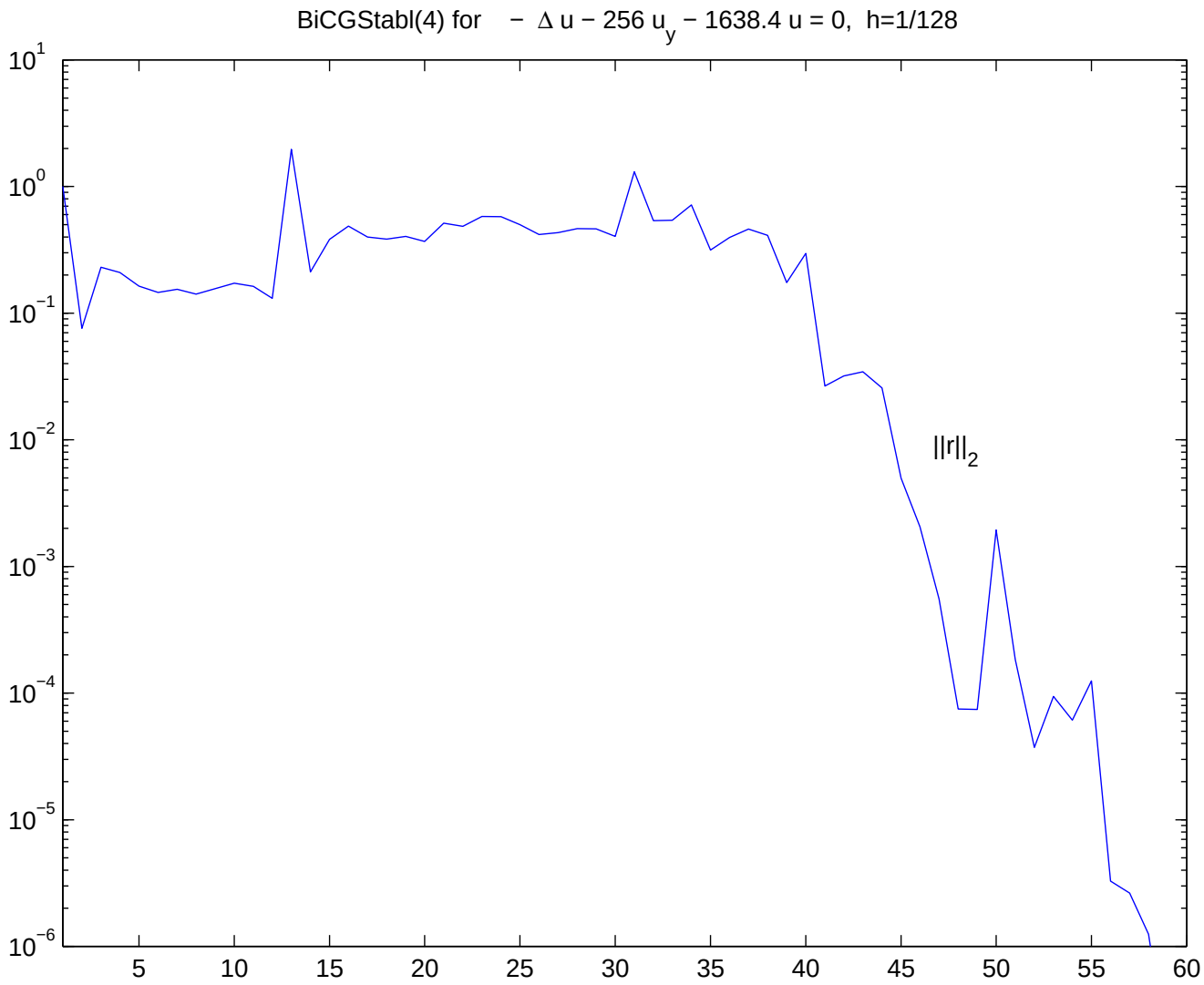
Example


BiCGStab(4) for − Δ u − 256 u_y = 0, h=1/128
10^1
10^0
10^−1
10^−2
10^−3
10^−4
10^−5
10^−6
5
10
15
20
25
30
35
40
45
||r||_2

Example


BiCGStabl(4) for − Δ u − 256 u_y − 163.84 u = 0, h=1/128
10^1
10^0
10^−1
10^−2
10^−3
10^−4
10^−5
10^−6
5
10
15
20
25
30
35
40
||r||_2

Example


BiCGStabl(4) for − Δ u − 256 u_y − 1638.4 u = 0, h=1/128
10^1
10^0
10^−1
10^−2
10^−3
10^−4
10^−5
10^−6
5
10
15
20
25
30
35
40
45
50
55
60
||r||_2