Block CG Algorithms Revisited: Theory and Numerical Reproduction

Siqing Cai

Charles University

November 26, 2025

Table of Contents

Introduction & Motivation

The Problem

- Goal: Solve $Ax = b$ for multiple b's.

Introduction & Motivation

The Problem

- Goal: Solve $Ax = b$ for multiple b's.
- Block form: $Ax = b$, where $b, x \in \mathbb{R}^{n \times m}$

Introduction & Motivation

The Problem

- Goal: Solve $Ax = b$ for multiple b's.
- Block form: $Ax = b$, where $b, x \in \mathbb{R}^{n \times m}$
- A is Symmetric Positive Definite (SPD)

Core Challenge

- Rank Deficiency: Vectors in a block become linearly dependent.

Introduction & Motivation

The Problem

- Goal: Solve $Ax = b$ for multiple b's.
- Block form: $Ax = b$, where $b, x \in \mathbb{R}^{n \times m}$
- A is Symmetric Positive Definite (SPD)

Core Challenge

- Rank Deficiency: Vectors in a block become linearly dependent.
- Leads to ill-conditioned coefficients.

Introduction & Motivation

The Problem

- Goal: Solve $Ax = b$ for multiple b's.
- Block form: $Ax = b$, where $b, x \in \mathbb{R}^{n \times m}$
- A is Symmetric Positive Definite (SPD)

Core Challenge

- Rank Deficiency: Vectors in a block become linearly dependent.
- Leads to ill-conditioned coefficients.
- Result: Slow convergence, loss of accuracy.

Goal

- **Goal 1 (Theory):** Clarify BCG vs. Block Lanczos relationship.

Introduction & Motivation

The Problem

- Goal: Solve $Ax = b$ for multiple b's.
- Block form: $Ax = b$, where $b, x \in \mathbb{R}^{n \times m}$
- A is Symmetric Positive Definite (SPD)

Core Challenge

- Rank Deficiency: Vectors in a block become linearly dependent.
- Leads to ill-conditioned coefficients.
- Result: Slow convergence, loss of accuracy.

Goal

- **Goal 1 (Theory):** Clarify BCG vs. Block Lanczos relationship.
- **Goal 2 (Practice):** Find a robust BCG variant for finite precision (i.e., handle rank deficiency).

The "Classic" Algorithm: O'Leary's BCG (OL-BCG)

Algorithm 4 Core Recursions

- Solution: $x_k = x_{k-1} + p_{k-1}\gamma_{k-1}$
- Residual: $r_k = r_{k-1} - Ap_{k-1}\gamma_{k-1}$
- Direction: $p_k = (r_k + p_{k-1}\delta_k)\phi_k$

The "Classic" Algorithm: O'Leary's BCG (OL-BCG)

Algorithm 4 Core Recursions

- Solution: $x_k = x_{k-1} + p_{k-1}\gamma_{k-1}$
- Residual: $r_k = r_{k-1} - Ap_{k-1}\gamma_{k-1}$
- Direction: $p_k = (r_k + p_{k-1}\delta_k)\phi_k$

Block Coefficients (HS-BCG: $\phi_k = I$)

- Step size $\gamma_{k-1} \propto (p_{k-1}^T A p_{k-1})^{-1}$
- Direction $\delta_k \propto (r_{k-1}^T r_{k-1})^{-1}$

The "Classic" Algorithm: O'Leary's BCG (OL-BCG)

Algorithm 4 Core Recursions

- Solution: $x_k = x_{k-1} + p_{k-1}\gamma_{k-1}$
- Residual: $r_k = r_{k-1} - Ap_{k-1}\gamma_{k-1}$
- Direction: $p_k = (r_k + p_{k-1}\delta_k)\phi_k$

Block Coefficients (HS-BCG: $\phi_k = I$)

- Step size $\gamma_{k-1} \propto (p_{k-1}^T A p_{k-1})^{-1}$
- Direction $\delta_k \propto (r_{k-1}^T r_{k-1})^{-1}$

Source of Instability

- What if p_{k-1} or r_{k-1} are rank-deficient?
- The inverses $(p_{k-1}^T A p_{k-1})^{-1}$ and $(r_{k-1}^T r_{k-1})^{-1}$ become singular.
- $\rightarrow$ Algorithm fails.

Goal 1: The "Apples to Oranges" Problem

Block Lanczos (Alg 3)

- Produces orthonormal blocks V_k.
- $V_k^T V_k = I$
- Defines a symmetric block-tridiagonal matrix T_k.
- $AV_k = V_k T_k + ...$

Goal 1: The "Apples to Oranges" Problem

Block Lanczos (Alg 3)

- Produces orthonormal blocks V_k.
- $V_k^T V_k = I$
- Defines a symmetric block-tridiagonal matrix T_k.
- $AV_k = V_k T_k + ...$

Block CG (Alg 4)

- Produces residual blocks R_k.
- R_k blocks are *not* orthogonal.
- Defines a non-symmetric matrix $\hat{T}_k$ (Lemma 1).
- $AR_k = R_k \hat{T}_k + ...$

Goal 1: The "Apples to Oranges" Problem

Block Lanczos (Alg 3)

- Produces orthonormal blocks V_k.
- $V_k^T V_k = I$
- Defines a symmetric block-tridiagonal matrix T_k.
- $AV_k = V_k T_k + \dots$

Block CG (Alg 4)

- Produces residual blocks R_k.
- R_k blocks are *not* orthogonal.
- Defines a non-symmetric matrix $\hat{T}_k$ (Lemma 1).
- $AR_k = R_k \hat{T}_k + \dots$

The Question

How to relate the non-orthogonal R_k and $\hat{T}_k$ from BCG back to the "pure" orthogonal V_k and T_k from Lanczos?

Goal 1: The Bridge (Lemma 2, Thm 1)

Step 1: Normalize the Residuals

- We *define* new, normalized blocks $\tilde{V}_k$ from the BCG residual blocks R_k.

Goal 1: The Bridge (Lemma 2, Thm 1)

Step 1: Normalize the Residuals

- We *define* new, normalized blocks $\tilde{V}_k$ from the BCG residual blocks R_k.
- $\tilde{v}_j = (-1)^{j-1} r_{j-1} \rho_{j-1}^{-1}$ (where ρ_j are normalization factors)

Step 2: Find Their Recurrence (Lemma 2)

- These *normalized* blocks $\tilde{V}_k$ *do* satisfy a symmetric recurrence:

Goal 1: The Bridge (Lemma 2, Thm 1)

Step 1: Normalize the Residuals

- We *define* new, normalized blocks $\tilde{V}_k$ from the BCG residual blocks R_k.
- $\tilde{v}_j = (-1)^{j-1} r_{j-1} \rho_{j-1}^{-1}$ (where ρ_j are normalization factors)

Step 2: Find Their Recurrence (Lemma 2)

- These *normalized* blocks $\tilde{V}_k$ *do* satisfy a symmetric recurrence:
- $A\tilde{V}_k = \tilde{V}_k \tilde{T}_k + \tilde{v}_{k+1} \tilde{\beta}_{k+1} e_k^T$

Goal 1: The Bridge (Lemma 2, Thm 1)

Step 1: Normalize the Residuals

- We *define* new, normalized blocks $\tilde{V}_k$ from the BCG residual blocks R_k.
- $\tilde{v}_j = (-1)^{j-1} r_{j-1} \rho_{j-1}^{-1}$ (where ρ_j are normalization factors)

Step 2: Find Their Recurrence (Lemma 2)

- These *normalized* blocks $\tilde{V}_k$ *do* satisfy a symmetric recurrence:
- $A\tilde{V}_k = \tilde{V}_k \tilde{T}_k + \tilde{v}_{k+1} \tilde{\beta}_{k+1} e_k^T$
- This $\tilde{T}_k$ is symmetric, just like the Lanczos matrix!

Step 3: Connect the Matrices (Theorem 1)

- This new $\tilde{T}_k$ (from BCG) is not T_k (from Lanczos), but they are "unitarily similar".

Goal 1: The Bridge (Lemma 2, Thm 1)

Step 1: Normalize the Residuals

- We *define* new, normalized blocks $\tilde{V}_k$ from the BCG residual blocks R_k.
- $\tilde{v}_j = (-1)^{j-1} r_{j-1} \rho_{j-1}^{-1}$ (where ρ_j are normalization factors)

Step 2: Find Their Recurrence (Lemma 2)

- These *normalized* blocks $\tilde{V}_k$ *do* satisfy a symmetric recurrence:
- $A\tilde{V}_k = \tilde{V}_k \tilde{T}_k + \tilde{v}_{k+1} \tilde{\beta}_{k+1} e_k^T$
- This $\tilde{T}_k$ is symmetric, just like the Lanczos matrix!

Step 3: Connect the Matrices (Theorem 1)

- This new $\tilde{T}_k$ (from BCG) is not T_k (from Lanczos), but they are "unitarily similar".
- $T_k = U_k^T \tilde{T}_k U_k$

Goal 1: The Bridge (Lemma 2, Thm 1)

Step 1: Normalize the Residuals

- We *define* new, normalized blocks $\tilde{V}_k$ from the BCG residual blocks R_k.
- $\tilde{v}_j = (-1)^{j-1} r_{j-1} \rho_{j-1}^{-1}$ (where ρ_j are normalization factors)

Step 2: Find Their Recurrence (Lemma 2)

- These *normalized* blocks $\tilde{V}_k$ *do* satisfy a symmetric recurrence:
- $A\tilde{V}_k = \tilde{V}_k \tilde{T}_k + \tilde{v}_{k+1} \tilde{\beta}_{k+1} e_k^T$
- This $\tilde{T}_k$ is symmetric, just like the Lanczos matrix!

Step 3: Connect the Matrices (Theorem 1)

- This new $\tilde{T}_k$ (from BCG) is not T_k (from Lanczos), but they are "unitarily similar".
- $T_k = U_k^T \tilde{T}_k U_k$

Goal 1: The "How-To" (Theorem 2)

Making Similarity into Equality

- **Question:** Can we force $U_k = I$ so that $\tilde{T}_k = T_k$?

Goal 1: The "How-To" (Theorem 2)

Making Similarity into Equality

- **Question:** Can we force $U_k = I$ so that $\tilde{T}_k = T_k$?
- **Answer (Theorem 2):** Yes, by carefully choosing the normalization factors ρ_j at each step.

Goal 1: The "How-To" (Theorem 2)

Making Similarity into Equality

- **Question:** Can we force $U_k = I$ so that $\tilde{T}_k = T_k$?
- **Answer (Theorem 2):** Yes, by carefully choosing the normalization factors ρ_j at each step.
- This involves a QR factorization: $[\theta_j, \beta_{j+1}] = \mathrm{qr}(\sigma_j \tau_j)$

Goal 1: The "How-To" (Theorem 2)

Making Similarity into Equality

- **Question:** Can we force $U_k = I$ so that $\tilde{T}_k = T_k$?
- **Answer (Theorem 2):** Yes, by carefully choosing the normalization factors ρ_j at each step.
- This involves a QR factorization: $[\theta_j, \beta_{j+1}] = \mathrm{qr}(\sigma_j \tau_j)$

The Result (Achieves Goal 1)

- We can now compute the *exact* Lanczos coefficients (α_j, β_j) from *inside* the Block CG algorithm.

Goal 1: The "How-To" (Theorem 2)

Making Similarity into Equality

- **Question:** Can we force $U_k = I$ so that $\tilde{T}_k = T_k$?
- **Answer (Theorem 2):** Yes, by carefully choosing the normalization factors ρ_j at each step.
- This involves a QR factorization: $[\theta_j, \beta_{j+1}] = \mathrm{qr}(\sigma_j \tau_j)$

The Result (Achieves Goal 1)

- We can now compute the *exact* Lanczos coefficients (α_j, β_j) from *inside* the Block CG algorithm.
- This is the "one-to-one correspondence" the abstract mentions.

Goal 1: The "How-To" (Theorem 2)

Making Similarity into Equality

- **Question:** Can we force $U_k = I$ so that $\tilde{T}_k = T_k$?
- **Answer (Theorem 2):** Yes, by carefully choosing the normalization factors ρ_j at each step.
- This involves a QR factorization: $[\theta_j, \beta_{j+1}] = \mathrm{qr}(\sigma_j \tau_j)$

The Result (Achieves Goal 1)

- We can now compute the *exact* Lanczos coefficients (α_j, β_j) from *inside* the Block CG algorithm.
- This is the "one-to-one correspondence" the abstract mentions.

The Practical Pivot...

- All of this beautiful theory (Sec 4) depends on the **full-rank assumption**.

Goal 1: The "How-To" (Theorem 2)

Making Similarity into Equality

- **Question:** Can we force $U_k = I$ so that $\tilde{T}_k = T_k$?
- **Answer (Theorem 2):** Yes, by carefully choosing the normalization factors ρ_j at each step.
- This involves a QR factorization: $[\theta_j, \beta_{j+1}] = \mathrm{qr}(\sigma_j \tau_j)$

The Result (Achieves Goal 1)

- We can now compute the *exact* Lanczos coefficients (α_j, β_j) from *inside* the Block CG algorithm.
- This is the "one-to-one correspondence" the abstract mentions.

The Practical Pivot...

- All of this beautiful theory (Sec 4) depends on the **full-rank assumption**.
- ...But in finite precision, this assumption breaks down.

The Core Problem: How to Handle Rank Deficiency?

What Happens in Practice?

- The r_k or p_k blocks become (nearly) singular.

The Core Problem: How to Handle Rank Deficiency?

What Happens in Practice?

- The r_k or p_k blocks become (nearly) singular.
- We cannot compute the coefficients (inverses fail).

The Core Problem: How to Handle Rank Deficiency?

What Happens in Practice?

- The r_k or p_k blocks become (nearly) singular.
- We cannot compute the coefficients (inverses fail).
- Algorithm stagnates or fails.

Two Competing Strategies

- **Strategy 1: Deflation (Remove)**
  - Find and remove the linearly dependent vectors.

The Core Problem: How to Handle Rank Deficiency?

What Happens in Practice?

- The r_k or p_k blocks become (nearly) singular.
- We cannot compute the coefficients (inverses fail).
- Algorithm stagnates or fails.

Two Competing Strategies

- **Strategy 1: Deflation (Remove)**
    - Find and remove the linearly dependent vectors.
    - $\rightarrow$ Block size m shrinks.

The Core Problem: How to Handle Rank Deficiency?

What Happens in Practice?

- The r_k or p_k blocks become (nearly) singular.
- We cannot compute the coefficients (inverses fail).
- Algorithm stagnates or fails.

Two Competing Strategies

- **Strategy 1: Deflation (Remove)**
  - Find and remove the linearly dependent vectors.
  - $\rightarrow$ Block size m shrinks.
- **Strategy 2: Regularization (Replace)**

The Core Problem: How to Handle Rank Deficiency?

What Happens in Practice?

- The r_k or p_k blocks become (nearly) singular.
- We cannot compute the coefficients (inverses fail).
- Algorithm stagnates or fails.

Two Competing Strategies

- **Strategy 1: Deflation (Remove)**
  - Find and remove the linearly dependent vectors.
  - $\rightarrow$ Block size m shrinks.
- **Strategy 2: Regularization (Replace)**
  - Replace the "bad" basis with a "good" one (Dubrulle).

The Core Problem: How to Handle Rank Deficiency?

What Happens in Practice?

- The r_k or p_k blocks become (nearly) singular.
- We cannot compute the coefficients (inverses fail).
- Algorithm stagnates or fails.

Two Competing Strategies

- **Strategy 1: Deflation (Remove)**
  - Find and remove the linearly dependent vectors.
  - $\rightarrow$ Block size m shrinks.
- **Strategy 2: Regularization (Replace)**
  - Replace the "bad" basis with a "good" one (Dubrulle).
  - $\rightarrow$ Block size m is maintained.

Dubrulle's Idea: Regularization

Concept

- Don't deflate (remove vectors).

Dubrulle's Idea: Regularization

Concept

- Don't deflate (remove vectors).
- Maintain the full block size m.

Dubrulle's Idea: Regularization

Concept

- Don't deflate (remove vectors).
- Maintain the full block size m.
- Ensure blocks are always well-conditioned.

Dubrulle's Idea: Regularization

Concept

- Don't deflate (remove vectors).
- Maintain the full block size m.
- Ensure blocks are always well-conditioned.
- Mechanism: Change of basis via factorization.

The Tool: Householder QR

- For any (potentially singular) block v:

$$[w, \sigma] = \mathrm{qr}(v, \text{"econ"})$$

Dubrulle's Idea: Regularization

Concept

- Don't deflate (remove vectors).
- Maintain the full block size m.
- Ensure blocks are always well-conditioned.
- Mechanism: Change of basis via factorization.

The Tool: Householder QR

- For any (potentially singular) block v:

$$[w, \sigma] = \mathrm{qr}(v, \text{"econ"})$$

- Property 1: w is *always* column-orthonormal.

Dubrulle's Idea: Regularization

Concept

- Don't deflate (remove vectors).
- Maintain the full block size m.
- Ensure blocks are always well-conditioned.
- Mechanism: Change of basis via factorization.

The Tool: Householder QR

- For any (potentially singular) block v:

$$[w, \sigma] = \mathrm{qr}(v, \text{"econ"})$$

- Property 1: w is *always* column-orthonormal.
- Property 2: $\text{colspan}(w) \supseteq \text{colspan}(v)$.

Dubrulle's Idea: Regularization

Concept

- Don't deflate (remove vectors).
- Maintain the full block size m.
- Ensure blocks are always well-conditioned.
- Mechanism: Change of basis via factorization.

The Tool: Householder QR

- For any (potentially singular) block v:

$$[w, \sigma] = \mathrm{qr}(v, \text{"econ"})$$

- Property 1: w is *always* column-orthonormal.
- Property 2: $\mathrm{colspan}(w) \supseteq \mathrm{colspan}(v)$.
- Key: Use w to continue, avoid inverting σ.

Solution 1A: Dubrulle-R (DR-BCG)

Strategy (Algorithm 5)

- **Regularize the Residual block r_k.**

Solution 1A: Dubrulle-R (DR-BCG)

Strategy (Algorithm 5)

- **Regularize the Residual block r_k.**
- Derivation:

Solution 1A: Dubrulle-R (DR-BCG)

Strategy (Algorithm 5)

- **Regularize the Residual block r_k.**
- Derivation:
  1. QR of residual: $r_k = w_k \sigma_k$.

Solution 1A: Dubrulle-R (DR-BCG)

Strategy (Algorithm 5)

- **Regularize the Residual block r_k.**
- Derivation:
  1. QR of residual: $r_k = w_k \sigma_k$.
  2. Use w_k (orthonormal) in the recursions.

Solution 1A: Dubrulle-R (DR-BCG)

Strategy (Algorithm 5)

- **Regularize the Residual block r_k.**
- Derivation:
  1. QR of residual: $r_k = w_k \sigma_k$.
  2. Use w_k (orthonormal) in the recursions.
  3. Smart choice of ϕ_k.

Solution 1A: DR-BCG (Formulas)

The "Antidote": Regularize the Residual

- We start by factoring the (potentially bad) residual: $r_k = w_k \sigma_k$

Solution 1A: DR-BCG (Formulas)

The "Antidote": Regularize the Residual

- We start by factoring the (potentially bad) residual: $r_k = w_k \sigma_k$
- We use the orthonormal w_k to build our *new* search direction s_k.

Solution 1A: DR-BCG (Formulas)

The "Antidote": Regularize the Residual

- We start by factoring the (potentially bad) residual: $r_k = w_k \sigma_k$
- We use the orthonormal w_k to build our *new* search direction s_k.
- All recursions are now based on w_k and s_k.

Solution 1A: DR-BCG (Formulas)

The "Antidote": Regularize the Residual

- We start by factoring the (potentially bad) residual: $r_k = w_k \sigma_k$
- We use the orthonormal w_k to build our *new* search direction s_k.
- All recursions are now based on w_k and s_k.

Algorithm 5 Key Formula

$$\xi_{k-1} = (s_{k-1}^T A s_{k-1})^{-1}$$

Solution 1A: DR-BCG (Formulas)

The "Antidote": Regularize the Residual

- We start by factoring the (potentially bad) residual: $r_k = w_k \sigma_k$
- We use the orthonormal w_k to build our *new* search direction s_k.
- All recursions are now based on w_k and s_k.

Algorithm 5 Key Formula

$$\xi_{k-1} = (s_{k-1}^T A s_{k-1})^{-1}$$

Numerical Benefit

The only inverse, ξ_{k-1}, is now computed from s_{k-1}, which is built from the well-conditioned w_{k-1}. This inverse is **stable**!

Solution 1B: Dubrulle-P (DP-BCG)

Strategy (Algorithm 6)

- **Regularize the Direction block p_k.**

Solution 1B: Dubrulle-P (DP-BCG)

Strategy (Algorithm 6)

- **Regularize the Direction block p_k.**
- Derivation:

Solution 1B: Dubrulle-P (DP-BCG)

Strategy (Algorithm 6)

- **Regularize the Direction block p_k.**
- Derivation:
  1. Use alternative formulas for γ, δ.

Solution 1B: Dubrulle-P (DP-BCG)

Strategy (Algorithm 6)

- **Regularize the Direction block p_k.**
- Derivation:
  1. Use alternative formulas for γ, δ.
  2. Regularize the new direction candidate via QR.

Solution 1B: Dubrulle-P (DP-BCG)

Strategy (Algorithm 6)

- **Regularize the Direction block p_k.**
- Derivation:
  1. Use alternative formulas for γ, δ.
  2. Regularize the new direction candidate via QR.
  3. $[p_k, \psi_k] = \mathrm{qr}(r_k + p_{k-1}\delta_k)$

Solution 1B: DP-BCG (The "Antidote")

The "Antidote": Regularize the Direction

- We calculate the "classic" direction update $r_k + p_{k-1}\delta_k$.

Solution 1B: DP-BCG (The "Antidote")

The "Antidote": Regularize the Direction

- We calculate the "classic" direction update $r_k + p_{k-1}\delta_k$.
- Then we *force it* to be orthonormal using QR:

$$[p_k, \psi_k] = \mathrm{qr}(r_k + p_{k-1}\delta_k)$$

Solution 1B: DP-BCG (Formulas & Benefit)

Supporting Formulas (Alg. 6)

$$\begin{aligned} \gamma_{k-1} &= (p_{k-1}^T A p_{k-1})^{-1} p_{k-1}^T r_{k-1} \\ \delta_k &= -(p_{k-1}^T A p_{k-1})^{-1} p_{k-1}^T A r_k \end{aligned}$$

Solution 1B: DP-BCG (Formulas & Benefit)

Supporting Formulas (Alg. 6)

$$\gamma_{k-1} = (p_{k-1}^T A p_{k-1})^{-1} p_{k-1}^T r_{k-1}$$
$$\delta_k = -(p_{k-1}^T A p_{k-1})^{-1} p_{k-1}^T A r_k$$

Numerical Benefit

- p_k is *always* column-orthonormal by construction.

Solution 1B: DP-BCG (Formulas & Benefit)

Supporting Formulas (Alg. 6)

$$\gamma_{k-1} = (p_{k-1}^T A p_{k-1})^{-1} p_{k-1}^T r_{k-1}$$
$$\delta_k = -(p_{k-1}^T A p_{k-1})^{-1} p_{k-1}^T A r_k$$

Numerical Benefit

- p_k is *always* column-orthonormal by construction.
- This ensures that the inverse $(p_k^T A p_k)^{-1}$ for the *next* step $(k+1)$ will be well-conditioned.

Solution 2: Breakdown-Free (BF-BCG)

Strategy

- **Deflate the Direction block.**

Solution 2: Breakdown-Free (BF-BCG)

Strategy

- **Deflate the Direction block.**
- Uses same alternative formulas as DP-BCG.

Solution 2: Breakdown-Free (BF-BCG)

Strategy

- **Deflate the Direction block.**
- Uses same alternative formulas as DP-BCG.
- Instead of full QR, computes orthonormal basis.

Crucial Difference: Regularize vs. Deflate

- **DP-BCG (Regularize):**
  - p_k is always $n \times m$.

Solution 2: Breakdown-Free (BF-BCG)

Strategy

- **Deflate the Direction block.**
- Uses same alternative formulas as DP-BCG.
- Instead of full QR, computes orthonormal basis.

Crucial Difference: Regularize vs. Deflate

- **DP-BCG (Regularize):**
  - p_k is always $n \times m$.
  - Maintains block size.
- **BF-BCG (Deflate):**

Solution 2: Breakdown-Free (BF-BCG)

Strategy

- **Deflate the Direction block.**
- Uses same alternative formulas as DP-BCG.
- Instead of full QR, computes orthonormal basis.

Crucial Difference: Regularize vs. Deflate

- **DP-BCG (Regularize):**
  - p_k is always $n \times m$.
  - Maintains block size.
- **BF-BCG (Deflate):**
  - p_k becomes $n \times m_k$, where $m_k \leq m$.

Solution 2: Breakdown-Free (BF-BCG)

Strategy

- **Deflate the Direction block.**
- Uses same alternative formulas as DP-BCG.
- Instead of full QR, computes orthonormal basis.

Crucial Difference: Regularize vs. Deflate

- **DP-BCG (Regularize):**
  - p_k is always $n \times m$.
  - Maintains block size.
- **BF-BCG (Deflate):**
  - p_k becomes $n \times m_k$, where $m_k \leq m$.
  - Block size shrinks if rank deficient.

Practical Need: Preconditioning

Goal

- Accelerate convergence.

Practical Need: Preconditioning

Goal

- Accelerate convergence.
- Use a preconditioner $M \approx A$.

Practical Need: Preconditioning

Goal

- Accelerate convergence.
- Use a preconditioner $M \approx A$.
- Implicitly solve $M^{-1}Ax = M^{-1}b$.

Two Preconditioned Variants

- P-DR-BCG (Algorithm 7)

Practical Need: Preconditioning

Goal

- Accelerate convergence.
- Use a preconditioner $M \approx A$.
- Implicitly solve $M^{-1}Ax = M^{-1}b$.

Two Preconditioned Variants

- P-DR-BCG (Algorithm 7)
- P-DP-BCG (Algorithm 8)

Preconditioning: The Details

P-DP-BCG (Alg. 8)

- Simpler

P-DP-BCG (Alg. 8)

- Simpler
- Only requires M^{-1} action.

Preconditioning: The Details

P-DP-BCG (Alg. 8)

- Simpler
- Only requires M^{-1} action.
- Key step: $z_k = M^{-1} r_k$.

P-DR-BCG (Alg. 7)

- More complex

Preconditioning: The Details

P-DP-BCG (Alg. 8)

- Simpler
- Only requires M^{-1} action.
- Key step: $z_k = M^{-1} r_k$.

P-DR-BCG (Alg. 7)

- More complex
- Requires split $M = LL^T$.

Preconditioning: The Details

P-DP-BCG (Alg. 8)

- Simpler
- Only requires M^{-1} action.
- Key step: $z_k = M^{-1} r_k$.

P-DR-BCG (Alg. 7)

- More complex
- Requires split $M = LL^T$.
- Needs L^{-1} and L^{-T} actions.

Experimental Setup

Software & Matrices

- MatLab R2023a
- `bcsstk03`: $n = 112$, ill-conditioned (no precon).
- `s3dkt3m2`: $n = 90449$, very ill-conditioned (with precon).

Experimental Setup

Software & Matrices

- MatLab R2023a
- `bcsstk03`: $n = 112$, ill-conditioned (no precon).
- `s3dkt3m2`: $n = 90449$, very ill-conditioned (with precon).

Parameters

- Right-hand sides: Random `rand(n,m)`.
- Block sizes m: 1, 2, 4, 6, 16, 64.
- Preconditioner: Incomplete Cholesky (`ichol`).

Results 1: HS vs. DR vs. DP

Reproduction of Figures 1 & 2

[Space for Plots]

Results 1: HS vs. DR vs. DP

Reproduction of Figures 1 & 2

[Space for Plots]

Observations

- HS-BCG: Performance degrades. Stagnates as m increases.
- DR-BCG & DP-BCG: Remain stable.

Results 1: HS vs. DR vs. DP

Reproduction of Figures 1 & 2

[Space for Plots]

Observations

- HS-BCG: Performance degrades. Stagnates as m increases.
- DR-BCG & DP-BCG: Remain stable.
- **DR-BCG: Consistently the winner.**
  - Faster convergence.
  - Better max accuracy.

Results 2: DP vs. BF-BCG

Reproduction of Figure 3

[Space for Plots]

Results 2: DP vs. BF-BCG

Reproduction of Figure 3

[Space for Plots]

Observations

- BF-BCG (Deflation): Slower convergence.
- DP-BCG (Regularization): Clearly superior.
- **Conclusion**: Regularization (Dubrulle) is practically better than Deflation (BF-BCG) in finite precision.

Conclusion

Summary of Findings

- O'Leary/HS-BCG is numerically fragile for $m > 1$.

Conclusion

Summary of Findings

- O'Leary/HS-BCG is numerically fragile for $m > 1$.
- Dubrulle's regularization is an effective, stable remedy.

Conclusion

Summary of Findings

- O'Leary/HS-BCG is numerically fragile for $m > 1$.
- Dubrulle's regularization is an effective, stable remedy.
- Regularization (DR, DP) Deflation (BF) in practice.

Conclusion

Summary of Findings

- O'Leary/HS-BCG is numerically fragile for $m > 1$.
- Dubrulle's regularization is an effective, stable remedy.
- Regularization (DR, DP) Deflation (BF) in practice.
- **DR-BCG shows the best overall performance.**

Practical Recommendation

For solving block linear systems, the preconditioned DR-BCG variant (Algorithm 7) is the most robust and efficient choice.