Accelerated First-Order Methods for Exascale Simulation and Learning

Robert Freund, Cuong Nguyen, and Jaime Peraire

MIT

AFOSR, August 2018

Outline

- Accelerated Residual Descent Method (ARDM)
- Accelerated Algorithms for Compressed Simulation via Deep Learning
- Generalized Stochastic Frank-Wolfe Algorithm for Loss Minimization
- Thoughts on Directions for the Program

Accelerated Residual Descent Method (ARDM)

Accelerated Residual Descent Method (ARDM)

Accelerated methods for exascale simulation

- Numerical simulation in fluid dynamics, solid mechanics, electromagnetism, etc. leads to large nonlinear systems of equations

$$\boldsymbol{f}(\boldsymbol{u}) = \boldsymbol{0} \quad \text{with } \boldsymbol{u} \in \mathbb{R}^{10^{6-10}} \tag{1}$$

- The matrix $\partial \boldsymbol{f} / \partial \boldsymbol{u}$ exceeds the memory available in large supercomputers, therefore matrix-free methods are needed and used
- State-of-the-art massively-parallel matrix-free method for (1) is (preconditioned) forward Euler, which is analogous to gradient descent in the optimization setting
- We extend the ideas of accelerated methods for optimization to devise accelerated methods for nonlinear systems like (1)

Difference between optimization and simulation

	Matrix $\partial \boldsymbol{f} / \partial \boldsymbol{u}$	Eigenvalues of $\partial \boldsymbol{f} / \partial \boldsymbol{u}$
Optimization	Symmetric	Real
Simulation	Non-symmetric	Complex

We use linear stability theory to devise accelerated methods that are better suited to simulation (i.e., to systems with complex eigenvalues)

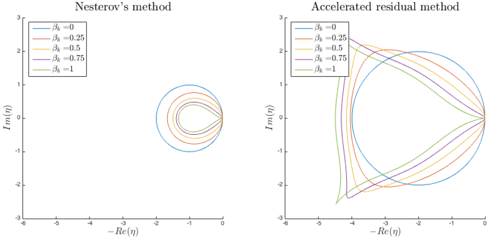


Figure: Stability regions of Nesterov's method and our accelerated residual method. β_k denotes the acceleration parameter. $\eta := \lambda\, \alpha_k$.

Accelerated Residual Descent Method

- Accelerated Nesterov's method

$$\boldsymbol{v}_k = \boldsymbol{u}_k + \beta_k(\boldsymbol{u}_k - \boldsymbol{u}_{k-1}), \tag{2a}$$

$$\boldsymbol{u}_{k+1} = \boldsymbol{v}_k - \alpha_k \boldsymbol{f}(\boldsymbol{v}_k). \tag{2b}$$

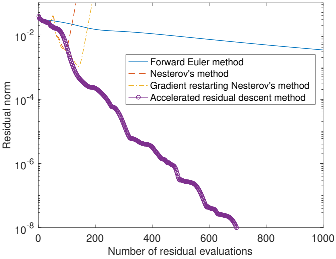
- Our method: Accelerated Residual Descent Method (ARDM)

$$\boldsymbol{v}_k = \boldsymbol{u}_k + \beta_k(\boldsymbol{u}_k - \boldsymbol{u}_{k-1}) - \alpha_k(1+\beta_k)\boldsymbol{f}(\boldsymbol{u}_k), \tag{3a}$$

$$\boldsymbol{u}_{k+1} = \boldsymbol{v}_k - \alpha_k \boldsymbol{f}(\boldsymbol{v}_k). \tag{3b}$$

Extra-term in red makes the method better suited (more robust) for simulation

Numerical results: Burgers equations



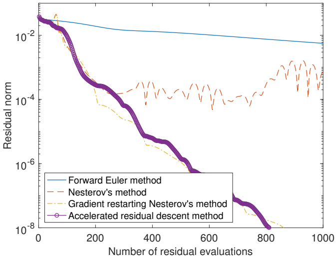


Figure: Left: No preconditioner Right: Block-Jacobi preconditioner

- Accelerated methods (Nesterov, ARDM) converge much faster than non-accelerated method (forward Euler)
- ARDM is more robust than Nesterov's method

Accelerated Algorithms for Compressed Simulation via Deep Learning

Accelerated Algorithms for
Compressed Simulation via Deep Learning

Compressed Simulation

The numerical discretization of many problems of interest produces dynamical systesm of the form

$$\boldsymbol{x}_t = \boldsymbol{f}(\boldsymbol{x}_{t-1}), \qquad \boldsymbol{x}_t \in R^n, \quad t \in [1, \ldots, T] \tag{4}$$

$\boldsymbol{f}()$: nonlinear map

Koopman Theory

Nonlinear discrete-time system can be mapped to a linear discrete-time system of the form

$$g(\boldsymbol{x}_t) = \mathcal{K} g(\boldsymbol{x}_{t-1})$$

$\boldsymbol{g}()$: observable function of the state vector $\boldsymbol{x}$
$\mathcal{K}$: infinite-dimensional linear operator

The Koopman Invariant Subpace

If there exists a finite number of observable functions $\{g_1, \ldots, g_m\}$ that span $\mathcal{G}$ such that

$$\mathcal{K}g \in \mathcal{G}, \qquad \text{for any} \qquad g \in \mathcal{G}$$

then, $\mathcal{G}$ is an invariant subpsace and $\mathcal{K}$ becomes *finite-dimensional*
In this case, instead of solving (4) we can solve the linear problem

$$\boldsymbol{y}_t = \boldsymbol{K}\boldsymbol{y}_{t-1}, \qquad \boldsymbol{y}_t = \boldsymbol{g}(\boldsymbol{x}_t) \tag{5}$$

where $\boldsymbol{g} = [g_1, \ldots, g_m]^T$ and $\boldsymbol{K} = \boldsymbol{Y}\boldsymbol{X}^*$ with
$\boldsymbol{X} = [\boldsymbol{g}(\boldsymbol{x}_0), \ldots, \boldsymbol{g}(\boldsymbol{x}_{T-1})]$, $\boldsymbol{Y} = [\boldsymbol{g}(\boldsymbol{x}_1), \ldots, \boldsymbol{g}(\boldsymbol{x}_T)]$,
and recover

$$\boldsymbol{x}_t = \boldsymbol{g}^{-1}(\boldsymbol{y}_t)$$

Dimensionality Reduction

- Koopman theory allows for model reduction of nonlinear system (4) of dimension n to linear system (5) of dimension m.
- When $m \ll n$ we can have dimensionality reduction by several orders of magnitude.
- Challenges:
  - The construction of $\boldsymbol{K}$ requires the computation of $\{\boldsymbol{x}_0, \ldots, \boldsymbol{x}_T\}$, i.e., we need to perform the full simulation (4)
  - The second challenge is to construct/recover the vector-valued observable $\boldsymbol{g}$ and its inverse $\boldsymbol{g}^{-1}$.

Learning Koopman Invariant Subspaces

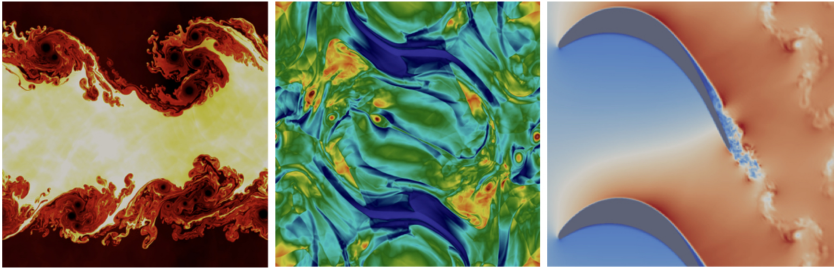
- Learn $(\boldsymbol{K}, \boldsymbol{g}, \boldsymbol{g}^{-1})$ from data $\{\boldsymbol{x}_0, \ldots, \boldsymbol{x}_p\}$ with $p \ll T$
- Training sets $\boldsymbol{X}_0 = [\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{p-1}]$ and $\boldsymbol{X}_1 = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_p]$
- Encoder Neural Net generates $\boldsymbol{X}_0 = [\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{p-1}]$ and $\boldsymbol{X}_1 = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_p]$
- Advance simulation $\hat{\boldsymbol{y}}_t = (\hat{\boldsymbol{Y}}\hat{\boldsymbol{X}}^*)\,\hat{\boldsymbol{y}}_{t-1}$
- Decoder Neural Net applies mapping $\boldsymbol{g}^{-1}$ to obtain $\{\hat{\boldsymbol{x}}_0, \ldots, \hat{\boldsymbol{x}}_p\}$
- Neural Nets are trained by solving

$$\min_{\boldsymbol{g}} L(\boldsymbol{g}; (\boldsymbol{x}_0, \ldots, \boldsymbol{x}_p)) := \|\boldsymbol{X}_0 - \hat{\boldsymbol{X}}_0(\boldsymbol{g})\|_F^2 + \|\boldsymbol{X}_1 - \hat{\boldsymbol{X}}_1(\boldsymbol{g})\|_F^2$$

where $\hat{\boldsymbol{X}}_0 = [\hat{\boldsymbol{x}}_0, \ldots, \hat{\boldsymbol{x}}_{p-1}]$ and $\hat{\boldsymbol{X}}_1 = [\hat{\boldsymbol{x}}_1, \ldots, \hat{\boldsymbol{x}}_p]$

Compressed Simulation

Plan on applying compressed simulation ideas to problems of practical interest.



High-fidelity simulations of Kelvin-Helmholtz instability (left), Orszag-Tang MHD flow (center), and turbine cascade flow (right). These simulations are performed using a high-order finite element code (DIGASO) developed by the PIs.

Generalized Stochastic Frank-Wolfe (GSFW) for Loss Minimization

Generalized Stochastic Frank-Wolfe (GSFW) for Loss Minimization

Problem of Interest

The problem of interest is the following general loss minimization problem:

$$\begin{array}{llll} P: & L_n^* = & \min_\beta & L_n(\beta) := \frac{1}{n}\sum_{j=1}^n l_j(x_j^T\beta) \\ & & \text{s.t.} & \beta \in Q \end{array}$$

$l_j(\cdot)$ is a univariate loss function that is strictly convex, differentiable, and γ-smooth:

$$|\dot{l}_j(a) - \dot{l}_j(b)| \le \gamma|a-b| \ \text{ for all } a, b \in \mathbb{R}$$

$Q \subset \mathbb{R}^p$ is a closed and bounded convex set

Define the separable loss function $L_n(s) := \frac{1}{n}\sum_{j=1}^n l_j(s_j)$

Denote $X := [x_1^T; x_2^T; \ldots; x_n^T]$. Then $s := X\beta$ and $L_n(s) = L_n(X\beta)$

Examples in Statistical and Machine Learning

- LASSO

$$\min_\beta \quad \frac{1}{2n}\sum_{j=1}^n (y_j - x_j^T\beta)^2$$

$$s.t. \quad \|\beta\|_1 \leq \delta \ ,$$

where $l_j(\cdot) = \frac{1}{2}(y_j - \cdot)^2$ and $Q = \{\beta : \|\beta\|_1 \leq \delta\}$

- Sparse Logistic Regression

$$\min_\beta \quad \frac{1}{n}\sum_{j=1}^n \ln(1+\exp(-y_j x_j^T\beta))$$

$$s.t. \quad \|\beta\|_1 \leq \delta \ ,$$

where $l_j(\cdot) = \ln(1+\exp(-y_j\cdot))$ and $Q = \{\beta : \|\beta\|_1 \leq \delta\}$

- Matrix Completion

$$\min_{\beta\in\mathbb{R}^{n\times p}} \quad \frac{1}{2|\Omega|}\sum_{(i,j)\in\Omega}(M_{i,j} - \beta_{i,j})^2$$

$$s.t. \quad \|\beta\|_* \leq \delta \ ,$$

where $l_{(i,j)}(\cdot) = \frac{1}{2}(\cdot - M_{i,j})^2$ and $Q = \{\beta : \|\beta\|_* \leq \delta\}$

- others ...

Frank-Wolfe method

$$\begin{array}{llll} \text{P:} & L_n^* = & \min_\beta & L_n(\beta) := \frac{1}{n}\sum_{j=1}^n l_j(x_j^T\beta) \\ & & \text{s.t.} & \beta \in Q \end{array}$$

Each iteration of Frank-Wolfe involves two main computations:

$$\text{compute } \nabla L_n(\beta^k)$$

and

$$\text{solve:} \quad \tilde{\beta}^k \leftarrow \arg\min_{\beta \in Q}\{\nabla L_n(\beta^k)^T\beta\}$$

The Frank-Wolfe method is an attractive first-order method to solve P:

- $O(1/\varepsilon)$ iteration bound for ε-optimal solution
- method produces "structured iterates": β^k is at most k-sparse (or rank-k) after k iterations

Stochastic Frank-Wolfe method

$$\begin{array}{llll} \text{P:} & L_n^* = & \min_\beta & L_n(\beta) := \frac{1}{n}\sum_{j=1}^n l_j(x_j^T\beta) \\ & & \text{s.t.} & \beta \in Q \end{array}$$

- gradient is $\nabla L_n(\beta^k) = \frac{1}{n}\sum_{j=1}^n \dot{l}_j(x_j^T\beta^k)x_j$
- when $n \gg 0$ it is too expensive to update $x_j^T\beta^k$ for all $j = 1, \ldots, n$ at iteration k
- Choose $j \in \mathcal{U}[1, \ldots, n]$
- $\tilde{g}^k = \dot{l}_j(x_j^T\beta^k)x_j$
- $\tilde{g}^k$ is an <u>unbiased</u> randomized estimate of $\nabla L_n(\beta^k)$
- using $\tilde{g}^k$ instead of $\nabla L_n(\beta^k)$ leads to Stochastic Frank-Wolfe methods

Comparison of Stochastic Frank-Wolfe Methods

To achieve an ε-optimal solution:

Algorithm and Reference	Number of Exact Gradient Calls	Number of Stochastic Gradient Calls	Number of Linear Optimization Oracle Calls
FW, Frank and Wolfe	$O(\frac{1}{\varepsilon})$	0	$O(\frac{1}{\varepsilon})$
SFW, Hazan and Luo	0	$O(\frac{1}{\varepsilon^3})$	$O(\frac{1}{\varepsilon})$
Online-FW, Hazan and Kale	0	$O(\frac{1}{\varepsilon^4})$	$O(\frac{1}{\varepsilon^4})$
SCGS, Lan and Zhou	0	$O(\frac{1}{\varepsilon^2})$	$O(\frac{1}{\varepsilon})$
SVRFW, Hazan and Luo	$O(\ln \frac{1}{\varepsilon})$	$O(\frac{1}{\varepsilon^2})$	$O(\frac{1}{\varepsilon})$
STORC, Hazan and Luo	$O(\ln \frac{1}{\varepsilon})$	$O(\frac{1}{\varepsilon^{1.5}})$	$O(\frac{1}{\varepsilon})$
SCGM, Mokhtari et al.	0	$O(\frac{1}{\varepsilon^3})$	$O(\frac{1}{\varepsilon^3})$
GSFW, this work	1	$O(\frac{1}{\varepsilon})$	$O(\frac{1}{\varepsilon})$

GSFW uses a biased estimate $\check{g}^k$ of $\nabla L_n(\beta^k)$

Convergence Guarantee of GSFW

Define $M := \max_{\beta \in Q} \max_{j=1,\ldots,n}\{|x_j^T \beta|\}$

Theorem: Convergence Guarantee of GSFW

For the GSFW algorithm it holds for all $k \geq 0$ that

$$\mathbb{E}\left[L_n(\bar{\beta}^k) - L_n^*\right] \;\leq\; \frac{8n\gamma M^2}{(4n+k)} + \frac{2n(2n-1)\gamma M^2}{(4n+k)(k+1)}$$

where

$$\bar{\beta}^k \;=\; \tfrac{2}{(4n+k)(k+1)} \sum_{i=0}^{k} (2n+i)\tilde{\beta}^i \;.$$

Some Thoughts on Directions for the Program

Some Thoughts on
Directions for the Program

Some Thoughts on Directions for the Program

- Optimization in the intertwined contexts of:
  - "big data"
  - data science
  - machine learning (including deep learning)
- Optimization for simulation problems in engineering design
- Machine learning (and deep learning) applied to engineering design

Thoughts on Directions for the Program, continued

- Theory: develop rigorous theoretical analyses of deep neural networks to shed light on why/when DNNs work and not work
  - DNNs do not work for all applications, it is thus important to genuinely understand their scope/limitations
- Algorithms: develop new AI/ML algorithms that are more rigorous/robust than SGD methods
  - new algorithms do not need to be more efficient than SGD
  - develop theory and performance bounds for algorithms in context of AI/ML applications

Thoughts on Directions for the Program, continued

- Applications: DNNs work well for images, textures, and languages. How about engineering design?
  - can DNNs be used to optimize engineering design or to augment optimization in engineering design?
- Data pre-processing: convolutional neural nets work well with images. For engineering applications, traditional convolutional layers may not work.
  - develop new types of pre-processing layers in DNNs for engineering design
- Big/small Data: data is often very expensive to collect in engineering design problems
  - develop new types of DNNs in this context that are intended to work well with small data

Backup for Generalized Stochastic Frank-Wolfe for Loss Minimization

Backup for

Generalized Stochastic Frank-Wolfe
for Loss Minimization

Problem of Interest

The problem of interest is the following general loss minimization problem:

$$\begin{array}{llll} \mathrm{P:} & P^* = & \min_\beta & P(\beta) := \frac{1}{n}\sum_{j=1}^n l_j(x_j^T\beta) \\ & & \text{s.t.} & \beta \in R \end{array}$$

$l_j(\cdot)$ is a univariate loss function that is strictly convex and γ-smooth:

$$|\dot{l}_j(a) - \dot{l}_j(b)| \leq \gamma|a-b| \ \text{ for all } a, b \in \mathbb{R}$$

R is a closed and bounded convex set, and $0 \in R$

Define the separable loss function $L(s) := \sum_{j=1}^n l_j(s_j)$

Denote $X := [x_1^T; x_2^T; \ldots; x_n^T]$. Then $s := X\beta$ and $L(s) = L(X\beta)$

Examples in Statistical and Machine Learning

- LASSO

$$\begin{aligned} \min_\beta \quad & \tfrac{1}{2n} \textstyle\sum_{j=1}^n (y_j - x_j^T \beta)^2 \\ s.t. \quad & \|\beta\|_1 \leq \delta \ , \end{aligned}$$

  where $l_j(\cdot) = \frac{1}{2}(y_j - \cdot)^2$ and $R = \{\beta : \|\beta\|_1 \leq \delta\}$

- Sparse Logistic Regression

$$\begin{aligned} \min_\beta \quad & \tfrac{1}{n} \textstyle\sum_{j=1}^n \ln(1 + \exp(-y_j x_j^T \beta)) \\ s.t. \quad & \|\beta\|_1 \leq \delta \ , \end{aligned}$$

  where $l_j(\cdot) = \ln(1 + \exp(-y_j \cdot))$ and $R = \{\beta : \|\beta\|_1 \leq \delta\}$

- Matrix Completion

$$\begin{aligned} \min_{\beta \in \mathbb{R}^{n \times p}} \quad & \tfrac{1}{2|\Omega|} \textstyle\sum_{(i,j) \in \Omega} (M_{i,j} - \beta_{i,j})^2 \\ s.t. \quad & \|\beta\|_* \leq \delta \ , \end{aligned}$$

  where $l_{(i,j)}(\cdot) = \frac{1}{2}(\cdot - M_{i,j})^2$ and $R = \{\beta : \|\beta\|_* \leq \delta\}$

- others ...

Frank-Wolfe and Stochastic Frank-Wolfe updates

The Frank-Wolfe method update is:

Frank-Wolfe method update

$$\text{Compute} \quad \tilde{\beta}^i \in \arg\min_{\beta \in P} \left\{ \nabla P(\beta^i)^T \beta \right\}$$

$$\beta^{i+1} \leftarrow (1-\alpha_i)\beta^i + \alpha_i \tilde{\beta}^i$$

In the traditional stochastic setting, we can only compute an (unbiased?) estimator $\tilde{g}$ of the gradient $\nabla P(\beta^i)$, and the update is

Stochastic Frank-Wolfe method update

$$\text{Compute} \quad \tilde{\beta}^i \in \arg\min_{\beta \in Q} \left\{ (\tilde{g}^i)^T \beta \right\}$$

$$\beta^{i+1} \leftarrow (1-\alpha_i)\beta^i + \alpha_i \tilde{\beta}^i$$

Stochastic Frank-Wolfe unbiased estimate of gradient

$$
\text{P:} \quad P^* = \quad \min_\beta \quad P(\beta) := \frac{1}{n}\sum_{j=1}^{n} l_j(x_j^T\beta)
$$
$$
\text{s.t.} \qquad \beta \in R
$$

- gradient is $\nabla P(\beta) = \frac{1}{n}\sum_{j=1}^{n} \dot{l}_j(x_j^T\beta)x_j$
- When n is large it is too expensive to update $x_j^T\beta$ for all $j = 1, \ldots, n$ at each iteration
- Choose $j \in \mathcal{U}[1, \ldots, n]$
- $\tilde{g} = \dot{l}_j(x_j^T\beta)x_j$
- $\tilde{g}$ is an unbiased estimate of the gradient

Stochastic Frank-Wolfe Methods

Comparison of computational complexity of recent stochastic Frank-Wolfe methods to achieve an absolute ε-optimal solution

Algorithm and Reference	Number of Exact Gradient Calls	Number of Stochastic Gradient Calls	Number of Linear Optimization Oracle Calls
FW, Frank and Wolfe	$O(\frac{1}{\varepsilon})$	0	$O(\frac{1}{\varepsilon})$
SFW, Hazan and Luo	0	$O(\frac{1}{\varepsilon^3})$	$O(\frac{1}{\varepsilon})$
Online-FW, Hazan and Kale	0	$O(\frac{1}{\varepsilon^4})$	$O(\frac{1}{\varepsilon^4})$
SCGS, Lan and Zhou	0	$O(\frac{1}{\varepsilon^2})$	$O(\frac{1}{\varepsilon})$
SVRFW, Hazan and Luo	$O(\ln\frac{1}{\varepsilon})$	$O(\frac{1}{\varepsilon^2})$	$O(\frac{1}{\varepsilon})$
STORC, Hazan and Luo	$O(\ln\frac{1}{\varepsilon})$	$O(\frac{1}{\varepsilon^{1.5}})$	$O(\frac{1}{\varepsilon})$
SCGM, Mokhtari et al.	0	$O(\frac{1}{\varepsilon^3})$	$O(\frac{1}{\varepsilon^3})$
GSFW, this work	1	$O(\frac{1}{\varepsilon})$	$O(\frac{1}{\varepsilon})$

"Substitute" Gradient

$$\text{P:} \quad P^* = \quad \min_\beta \quad P(\beta) := \frac{1}{n}\sum_{j=1}^n l_j(x_j^T\beta)$$

$$\text{s.t.} \qquad \beta \in R$$

- gradient is $\nabla P(\beta) = \frac{1}{n}\sum_{j=1}^n \dot{l}_j(x_j^T\beta)x_j$
- When n is large it is too expensive to update $x_j^T\beta$ for all $j = 1, \ldots, n$ in each iteration
- "Substitute" gradient d is computed by $d = \frac{1}{n}\sum_{j=1}^n \dot{l}_j(s_j)x_j$
- We will only update one s_j in each iteration
- d may not be an unbiased estimator of the gradient

Generalized Stochastic Frank-Wolfe Method with Substitute Gradient

Generalized Stochastic Frank-Wolfe with Substitute Gradient (GSFW)

Initialize with $\bar{\beta}^{-1} = 0$, $s^0 = 0$, and substitute gradient $d^0 = \frac{1}{n} X^T \nabla L(s^0)$, with step-size sequences $\{\alpha_i\} \in (0, 1]$, $\{\eta_i\} \in (0, 1]$.

For iterations $i = 0, 1, \ldots,$ do:

Solve l.o.o. subproblem: Compute $\tilde{\beta}^i \in \arg\min_{\beta \in R} \left\{ \left(d^i\right)^T \beta \right\}$

Choose random index: Choose $j_i \in \mathcal{U}[1, \ldots, n]$

Update s value: $s_{j_i}^{i+1} \leftarrow (1 - \eta_i) s_{j_i}^i + \eta_i (x_{j_i}^T \tilde{\beta}^i)$, and $s_j^{i+1} \leftarrow s_j^i$ for $j \neq j_i$

Update substitute gradient:

$$d^{i+1} = \frac{1}{n} X^T \nabla L(s^{i+1}) = d^i + \frac{1}{n} \left(\dot{l}_{j_i}(s_{j_i}^{i+1}) - \dot{l}_{j_i}(s_{j_i}^i) \right) x_{j_i}$$

Update primal variable: $\bar{\beta}^i \leftarrow (1 - \alpha_i) \bar{\beta}^{i-1} + \alpha_i \tilde{\beta}^i$.

(Optional Accounting:) $w^{i+1} \leftarrow \nabla L(s^{i+1})$

Generalized Stochastic Frank-Wolfe Method with Substitute Gradient

Generalized Stochastic Frank-Wolfe with Substitute Gradient (GSFW)

Initialize with $\bar{\beta}^{-1} = 0$, $s^0 = 0$, and substitute gradient $d^0 = \frac{1}{n} X^T \nabla L(s^0)$, with step-size sequences $\{\alpha_i\} \in (0,1]$, $\{\eta_i\} \in (0,1]$.

For iterations $i = 0, 1, \ldots,$ do:

Solve l.o.o. subproblem: Compute $\tilde{\beta}^i \in \arg\min_{\beta \in R} \left\{ \left(d^i\right)^T \beta \right\}$

Choose random index: Choose $j_i \in \mathcal{U}[1, \ldots, n]$

Update s value: $s_{j_i}^{i+1} \leftarrow (1-\eta_i) s_{j_i}^i + \eta_i (x_{j_i}^T \tilde{\beta}^i)$, and $s_j^{i+1} \leftarrow s_j^i$ for $j \neq j_i$

Update substitute gradient:

$$d^{i+1} = \frac{1}{n} X^T \nabla L(s^{i+1}) = d^i + \frac{1}{n} \left(\dot{l}_{j_i}(s_{j_i}^{i+1}) - \dot{l}_{j_i}(s_{j_i}^i) \right) x_{j_i}$$

Update primal variable: $\bar{\beta}^i \leftarrow (1-\alpha_i)\bar{\beta}^{i-1} + \alpha_i \tilde{\beta}^i$.

(Optional Accounting:) $w^{i+1} \leftarrow \nabla L(s^{i+1})$

Generalized Stochastic Frank-Wolfe Method with Substitute Gradient

Generalized Stochastic Frank-Wolfe with Substitute Gradient (GSFW)

Initialize with $\bar{\beta}^{-1} = 0$, $s^0 = 0$, and substitute gradient $d^0 = \frac{1}{n} X^T \nabla L(s^0)$, with step-size sequences $\{\alpha_i\} \in (0,1]$, $\{\eta_i\} \in (0,1]$.

For iterations $i = 0, 1, \ldots,$ do:

Solve l.o.o. subproblem: Compute $\tilde{\beta}^i \in \arg\min_{\beta \in R} \left\{ \left(d^i\right)^T \beta \right\}$

Choose random index: Choose $j_i \in \mathcal{U}[1, \ldots, n]$

Update s value: $s_{j_i}^{i+1} \leftarrow (1-\eta_i) s_{j_i}^i + \eta_i (x_{j_i}^T \tilde{\beta}^i)$, and $s_j^{i+1} \leftarrow s_j^i$ for $j \neq j_i$

Update substitute gradient:

$$d^{i+1} = \frac{1}{n} X^T \nabla L(s^{i+1}) = d^i + \frac{1}{n} \left(\dot{l}_{j_i}(s_{j_i}^{i+1}) - \dot{l}_{j_i}(s_{j_i}^i) \right) x_{j_i}$$

Update primal variable: $\bar{\beta}^i \leftarrow (1-\alpha_i) \bar{\beta}^{i-1} + \alpha_i \tilde{\beta}^i$.

(Optional Accounting:) $w^{i+1} \leftarrow \nabla L(s^{i+1})$

Generalized Stochastic Frank-Wolfe Method with Substitute Gradient

Generalized Stochastic Frank-Wolfe with Substitute Gradient (GSFW)

Initialize with $\bar{\beta}^{-1} = 0$, $s^0 = 0$, and substitute gradient $d^0 = \frac{1}{n} X^T \nabla L(s^0)$, with step-size sequences $\{\alpha_i\} \in (0,1]$, $\{\eta_i\} \in (0,1]$.

For iterations $i = 0, 1, \ldots,$ do:

Solve l.o.o. subproblem: Compute $\tilde{\beta}^i \in \arg\min_{\beta \in R} \left\{ \left(d^i\right)^T \beta \right\}$

Choose random index: Choose $j_i \in \mathcal{U}[1, \ldots, n]$

Update s value: $s_{j_i}^{i+1} \leftarrow (1-\eta_i) s_{j_i}^i + \eta_i (x_{j_i}^T \tilde{\beta}^i)$, and $s_j^{i+1} \leftarrow s_j^i$ for $j \neq j_i$

Update substitute gradient:

$$d^{i+1} = \frac{1}{n} X^T \nabla L(s^{i+1}) = d^i + \frac{1}{n} \left(\dot{l}_{j_i}(s_{j_i}^{i+1}) - \dot{l}_{j_i}(s_{j_i}^i) \right) x_{j_i}$$

Update primal variable: $\bar{\beta}^i \leftarrow (1-\alpha_i) \bar{\beta}^{i-1} + \alpha_i \tilde{\beta}^i$.

(Optional Accounting:) $w^{i+1} \leftarrow \nabla L(s^{i+1})$

Generalized Stochastic Frank-Wolfe Method with Substitute Gradient

Generalized Stochastic Frank-Wolfe with Substitute Gradient (GSFW)

Initialize with $\bar{\beta}^{-1} = 0$, $s^0 = 0$, and substitute gradient $d^0 = \frac{1}{n} X^T \nabla L(s^0)$, with step-size sequences $\{\alpha_i\} \in (0,1]$, $\{\eta_i\} \in (0,1]$.

For iterations $i = 0, 1, \ldots,$ do:

Solve l.o.o. subproblem: Compute $\tilde{\beta}^i \in \arg\min_{\beta \in R} \left\{ \left(d^i\right)^T \beta \right\}$

Choose random index: Choose $j_i \in \mathcal{U}[1, \ldots, n]$

Update s value: $s_{j_i}^{i+1} \leftarrow (1-\eta_i) s_{j_i}^i + \eta_i (x_{j_i}^T \tilde{\beta}^i)$, and $s_j^{i+1} \leftarrow s_j^i$ for $j \neq j_i$

Update substitute gradient:

$$d^{i+1} = \frac{1}{n} X^T \nabla L(s^{i+1}) = d^i + \frac{1}{n} \left(\dot{l}_{j_i}(s_{j_i}^{i+1}) - \dot{l}_{j_i}(s_{j_i}^i) \right) x_{j_i}$$

Update primal variable: $\bar{\beta}^i \leftarrow (1-\alpha_i) \bar{\beta}^{i-1} + \alpha_i \tilde{\beta}^i$.

(Optional Accounting:) $w^{i+1} \leftarrow \nabla L(s^{i+1})$

Generalized Stochastic Frank-Wolfe Method with Substitute Gradient

Generalized Stochastic Frank-Wolfe with Substitute Gradient (GSFW)

Initialize with $\bar{\beta}^{-1} = 0$, $s^0 = 0$, and substitute gradient $d^0 = \frac{1}{n} X^T \nabla L(s^0)$, with step-size sequences $\{\alpha_i\} \in (0,1]$, $\{\eta_i\} \in (0,1]$.

For iterations $i = 0, 1, \ldots,$ do:

Solve l.o.o. subproblem: Compute $\tilde{\beta}^i \in \arg\min_{\beta \in R} \left\{ \left(d^i\right)^T \beta \right\}$

Choose random index: Choose $j_i \in \mathcal{U}[1, \ldots, n]$

Update s value: $s_{j_i}^{i+1} \leftarrow (1-\eta_i) s_{j_i}^i + \eta_i (x_{j_i}^T \tilde{\beta}^i)$, and $s_j^{i+1} \leftarrow s_j^i$ for $j \neq j_i$

Update substitute gradient:

$$d^{i+1} = \frac{1}{n} X^T \nabla L(s^{i+1}) = d^i + \frac{1}{n} \left(\dot{l}_{j_i}(s_{j_i}^{i+1}) - \dot{l}_{j_i}(s_{j_i}^i) \right) x_{j_i}$$

Update primal variable: $\bar{\beta}^i \leftarrow (1-\alpha_i) \bar{\beta}^{i-1} + \alpha_i \tilde{\beta}^i$

(Optional Accounting:) $w^{i+1} \leftarrow \nabla L(s^{i+1})$

Generalized Stochastic Frank-Wolfe Method with Substitute Gradient

Generalized Stochastic Frank-Wolfe with Substitute Gradient (GSFW)

Initialize with $\bar{\beta}^{-1} = 0$, $s^0 = 0$, and substitute gradient $d^0 = \frac{1}{n} X^T \nabla L(s^0)$, with step-size sequences $\{\alpha_i\} \in (0,1]$, $\{\eta_i\} \in (0,1]$.

For iterations $i = 0, 1, \ldots,$ do:

Solve l.o.o. subproblem: Compute $\tilde{\beta}^i \in \arg\min_{\beta \in R} \left\{ \left(d^i\right)^T \beta \right\}$

Choose random index: Choose $j_i \in \mathcal{U}[1, \ldots, n]$

Update s value: $s_{j_i}^{i+1} \leftarrow (1-\eta_i) s_{j_i}^i + \eta_i (x_{j_i}^T \tilde{\beta}^i)$, and $s_j^{i+1} \leftarrow s_j^i$ for $j \neq j_i$

Update substitute gradient:

$$d^{i+1} = \frac{1}{n} X^T \nabla L(s^{i+1}) = d^i + \frac{1}{n} \left(\dot{l}_{j_i}(s_{j_i}^{i+1}) - \dot{l}_{j_i}(s_{j_i}^i) \right) x_{j_i}$$

Update primal variable: $\bar{\beta}^i \leftarrow (1-\alpha_i)\bar{\beta}^{i-1} + \alpha_i \tilde{\beta}^i$

(Optional Accounting:) $w^{i+1} \leftarrow \nabla L(s^{i+1})$

Convergence Guarantee of GSFW

Define $M := \max_{\beta \in R} \max_{j=1,\ldots,n} \{|x_j^T \beta|\}$

Theorem: Convergence Guarantee of GSFW

Consider GSFW with step-size sequences $\alpha_i = \frac{2(2n+i)}{(i+1)(4n+i)}$ and $\eta_i = \frac{2n}{2n+i+1}$ for $i = 0, 1, \ldots$. It holds for all $k \geq 0$ that

$$\mathbb{E}\left[P(\bar{\beta}^k) - P^*\right] \leq \frac{8n\gamma M^2}{(4n+k)} + \frac{2n(2n-1)\gamma M^2}{(4n+k)(k+1)} .$$

Convergence Guarantee of GSFW

Define $M := \max_{\beta \in R} \max_{j=1,\ldots,n}\{|x_j^T \beta|\}$

Theorem: Convergence Guarantee of GSFW

Consider GSFW with step-size sequences $\alpha_i = \frac{2(2n+i)}{(i+1)(4n+i)}$ and $\eta_i = \frac{2n}{2n+i+1}$ for $i = 0, 1, \ldots$. It holds for all $k \geq 0$ that

$$\mathbb{E}\left[P(\bar{\beta}^k) - P^*\right] \leq \frac{8n\gamma M^2}{(4n+k)} + \frac{2n(2n-1)\gamma M^2}{(4n+k)(k+1)} .$$